

---

# Manual

## Applications of Python using NumPy and Pandas

---



Sri Ch Rami Naidu  
Smt. P Saroja  
Smt. P Neelima  
Sri KDV Pavan Kumar  
Sri. LV Srinivas

This lab manual is intended to aid the undergraduate second-year computer science and engineering students for their course APPLICATIONS OF PYTHON USING NUMPY AND PANDAS Lab [B20CS2207].

### About the author

**Ch Rami Naidu:** He is currently pursuing (PhD) in PUDUCHERRY TECHNOLOGICAL UNIVERSITY, M.Tech From JNTUK and B.Tech From JNTUK. Presently he is working as an assistant professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India. He has a prominent experience in Python Programming and DBMS.

**P Saroja:** She is currently pursuing (PhD) in ANNAMALIA University, M.Tech From Andhra University and B.Tech From Andhra University. Presently She is working as an assistant professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India.

**P Neelima:** She is currently pursuing (PhD) in BPUT, M.Tech From Andhra University and B.Tech From Andhra University. Presently She is working as an assistant professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India.

**KDV Pavan Kumar:** He is currently pursuing (PhD) in ANNAMALIA University, M.Tech From Andhra University and B.Tech From Andhra University. Presently He is working as an assistant professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India.

**L.V Srinivas:** He is currently pursuing (PhD) in Andhra University, M.Tech From Andhra University and B.Tech From Andhra University. Presently He is working as an assistant professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India.

For private circulation among 2/4 B.Tech.(CSE) students.

## Preface

Python is increasingly being used as a scientific language. Matrix and vector manipulations are extremely important for scientific computations. Both NumPy and Pandas have emerged to be essential libraries for any scientific computation, including machine learning, in python due to their intuitive syntax and high-performance matrix computation capabilities. NumPy stands for 'Numerical Python' or 'Numeric Python'. It is an open source module of Python which provides fast mathematical computation on arrays and matrices. Similar to NumPy, Pandas is one of the most widely used python libraries in data science. It provides high-performance, easy to use structures and data analysis tools. Unlike NumPy library which provides objects for multi-dimensional arrays, Pandas provides in-memory 2d table object called Data frame. It is like a spreadsheet with column names and row labels.

NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation. NumPy can be imported into the notebook using

Python is a high-level, interpreted, reflective, dynamically typed, open source, multi-paradigm and general-purpose programming language. It is quite powerful and easy. It offers no special tools or features that let you do things that you cannot do with other languages, but its elegant design and combination of certain features make Python a pleasure to use.

Although all care has been taken to check for mistakes in this laboratory manual, yet it is impossible to claim perfection especially as this is the first edition. Any such errors and suggestions for improvement can be brought to our notice and are highly welcome.

Ch Rami Naidu  
P Saroja  
P Neelima  
KDV Pavan Kumar  
LV Srinivas

Evaluation of Practical subjects: For practical subjects there shall be continuous evaluation during the semester for 15 internal marks and 35 end examination marks. The internal 15 marks shall be awarded as follows: day to day work - 5 marks, Record-5 marks and the remaining 5 marks to be awarded by conducting an internal laboratory test. The end examination shall be conducted by the teacher concerned and external examiner appointed by Dean Evaluation/ Controller of examinations

Evaluation Scheme	
Examination	Marks
Exercise Programs	5
Record	5
Internal Exam	5
External Exam	35

# Applications of Python using NumPy and Pandas

## MODULE - 1: NUMPY

### Course Objectives

1. The objective of this lab is to acquire programming skills in Python package NumPy and perform mathematical and statistical operation

### Course Outcomes

At the end of the course Students will be able

<b>CO1</b>	Explain how data is collected, managed and stored for processing
<b>CO2</b>	Understand the workings of various numerical techniques, different descriptive measures of Statistics, correlation and regression to solve the engineering problems
<b>CO3</b>	Understand how to apply some linear algebra operations to n-dimensional arrays
<b>CO4</b>	Use NumPy perform common data wrangling and computational tasks in Python.

## List of Experiments

1. NumPy Installation using different scientific python distributions (Anaconda, Python(x,y), WinPython, Pyzo)
2. NumPy Basics (np.array, np.arange, np.linspace, np.zeros, np.ones, np.random.random, np.empty)
3. Arrays ( array.shape, len(array), array.ndim, array.dtype, array.astype(type), type(array))
4. Array Manipulation (np.append, np.insert, np.resize, np.delete, np.concatenate, np.vstack, np.hstack)
5. Mathematical Operations (np.add, np.subtract, np.divide, np.multiply, np.sqrt, np.sin, np.cos, np.log, np.dot, np.roots) , Statistical Operations( np.mean, np.median, np.std, array.corrcoef( ) )
6. NumPy data types
7. NumPy ndarray
8. NumPy String Operations
9. NumPy Financial functions
10. NumPy Functional Programming

## Additional Experiments

1. Write a NumPy program to create a 3x3 matrix with values ranging from 2 to 10.  
Expected Output:  

```
[[ 2 3 4]
 [ 5 6 7]
 [ 8 9 10]]
```
2. Write a NumPy program to reverse an array (the first element becomes the last).  
Original array:  

```
[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]
```

  
Reverse array:  

```
[37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12]
```
3. Write a NumPy program to convert Centigrade degrees into Fahrenheit degrees. Centigrade values are stored in a NumPy array.  
Sample Array [0, 12, 45.21, 34, 99.91]  

```
[-17.78, -11.11, 7.34, 1.11, 37.73, 0. ]
```
4. Expected Output:  
Values in Fahrenheit degrees:  

```
[ 0. 12. 45.21 34. 99.91 32. ]
```

  
Values in Centigrade degrees:  

```
[-17.78 -11.11 7.34 1.11 37.73 0. ]
```

  
Values in Centigrade degrees:  

```
[-17.78 -11.11 7.34 1.11 37.73 0. ]
```

  
Values in Fahrenheit degrees:  

```
[-0. 12. 45.21 34. 99.91 32. ]
```
5. If you only had Rs 200/month to pay towards the loan, how long would it take to pay-off a loan of Rs 10,000 at 8% annual interest?
6. What is the monthly payment needed to pay off a \$100,000 loan in 12 years at an annual interest rate of 8.5%?

## MODULE - 2: PANDAS

### Course Objectives:

1. To Understand the Fundamentals of The Pandas Library in Python
2. How It Is Used to Handle Data and Also Develop Basic Skills in Data Analysis and Visualization

### Course Outcomes:

At the end of the course Students will be able

CO1	Use Pandas to create and manipulate data structures like Series and Data Frames.
CO2	Work with arrays, queries, and data frames
CO3	Query Data Frame structures for cleaning and processing and manipulating files
CO4	Understand best practices for creating basic charts



## List of Experiments

### 1. Perform the following:

1. Pandas Installation
2. Creating Data Frames

### 2. A) Pandas Data Series:

1. Write a Pandas program to create and display a one-dimensional array-like object containing an array of data using Panda's module.
2. Write a Pandas program to convert a Panda module Series to Python list and it's type
3. Write a Pandas program to add, subtract, multiple and divide two Pandas Series.
4. Write a Pandas program to convert a NumPy array to a Pandas series.

Sample Series:

NumPy array:

```
[10 20 30 40 50]
```

Converted Pandas series:

```
0 10
```

```
1 20
```

```
2 30
```

```
3 40
```

```
4 50
```

dtype: int64

### 3. B) Pandas Data Frames:

Consider Sample Python dictionary data and list labels:

```
exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael',  
                    'Matthew', 'Laura', 'Kevin', 'Jonas'],
```

```
'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
```

```
'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
```

```
'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}
```

```
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

1. Write a Pandas program to create and display a Data Frame from a specified dictionary data which has the index labels.
2. Write a Pandas program to change the name 'James' to 'Suresh' in name column of the DataFrame.

3. Write a Pandas program to insert a new column in existing DataFrame.
4. Write a Pandas program to get list from DataFrame column headers.
5. Write a Pandas program to get list from DataFrame column headers.

#### 4. C) Pandas Index:

1. Write a Pandas program to display the default index and set a column as an Index in a given dataframe.
2. Write a Pandas program to create an index label by using 64-bit integers, using floating-point numbers in a given dataframe.

#### 5. D) Pandas String and Regular Expressions:

1. Write a Pandas program to convert all the string values to upper, lower cases in a given pandas series. Also find the length of the string values.
2. Write a Pandas program to remove whitespaces, left sided whitespaces and right sided whitespaces of the string values of a given pandas series.
3. Write a Pandas program to count of occurrence of a specified substring in a DataFrame column.
4. Write a Pandas program to swap the cases of a specified character column in a given DataFrame.

#### 6. E) Pandas Joining and merging DataFrame:

1. Write a Pandas program to join the two given dataframes along rows and assign all data.
2. Write a Pandas program to append a list of dictionaries or series to a existing DataFrame and display the combined data.
3. Write a Pandas program to join the two dataframes with matching records from both sides where available.

#### 7. F) Pandas Time Series:

1. Write a Pandas program to create
  - a) Datetimeobject for Jan 15 2012.
  - b) Specific date and time of 9:20 pm.
  - c) Local date and time.
  - d) A date without time.
  - e) Current date.
  - f) Time from a datetime.
  - g) Current local time.

2. Write a Pandas program to create a date from a given year, month, day and another date from a given string formats.
3. Write a Pandas program to create a time-series with two index labels and random values. Also print the type of the index.

#### 8. G) Pandas Grouping Aggregate:

Consider dataset:

school	class	name	date_Of_Birth	age	height	weight	address
S1	s001	V Alberto Franco	15/05/2002	12	173	35	street1
S2	s002	V Gino Mcneill	17/05/2002	12	192	32	street2
S3	s003	VI Ryan Parkes	16/02/1999	13	186	33	street3
S4	s001	VI Eesha Hinton	25/09/1998	13	167	30	street1
S5	s002	V Gino Mcneill	11/05/2002	14	151	31	street2
S6	s004	VI David Parkes	15/09/1997	12	159	32	street4

1. Write a Pandas program to split the following dataframe into groups based on school code. Also check the type of GroupBy object.
2. Write a Pandas program to split the following dataframe by school code and get mean, min, and max value of age for each school.

#### 9. H) Pandas Styling:


1. Create a dataframe of ten rows, four columns with random values. Write a Pandas program to highlight the negative numbers red and positive numbers black.
2. Create a dataframe of ten rows, four columns with random values. Write a Pandas program to highlight the maximum value in each column.
3. Create a dataframe of ten rows, four columns with random values. Write a Pandas program to highlight dataframe's specific columns.

#### 10. I) Excel:

1. Write a Pandas program to import excel data into a Pandas dataframe.
2. Write a Pandas program to find the sum, mean, max, min value of a column of file.

#### 11. J) Plotting:

1. Write a Pandas program to create a horizontal stacked bar plot of opening, closing stock

- 
- prices of any stock dataset between two specific dates.
2. Write a Pandas program to create a histograms plot of opening, closing, high, low stock prices of stock dataset between two specific dates.
  3. Write a Pandas program to create a stacked histograms plot of opening, closing, high, low stock prices of stock dataset between two specific dates with more bins.

**12. K) Pandas SQL Query:**

1. Write a Pandas program to display all the records of a student file.
2. Write a Pandas program to select distinct department id from employee's file.

## Additional Experiments

1. Convert to ordered categorical type with custom ordering
2. Sort values inplace
3. Write a Pandas program to append rows to an existing DataFrame and display the combined data.

### Test Data:

tudent\_data1

	student_id	name	marks
0	S1	name_1	200
1	S2	name_2	210
2	S3	name_3	190
3	S4	name_4	222
4	S5	name_5	199

New Row(s)

student_id	S6
name	name_6
marks	205

dtype: object

4. Create a dataframe of ten rows, four columns with random values. Write a Pandas program to highlight the negative numbers red and positive numbers black.

### Expected Output:

	A	B	C	D	E
0	1	1.32921	-0.770033	-0.31628	-0.99081
1	2	-1.07082	-1.43871	0.564417	0.295722
2	3	-1.6264	0.219565	0.678805	1.88927
3	4	0.961538	0.104011	-0.481165	0.850229
4	5	1.45342	1.05774	0.165562	0.515018
5	6	-1.33694	0.562861	1.39285	-0.063328
6	7	0.121668	1.2076	-0.00204021	1.6278
7	8	0.354493	1.03753	-0.385684	0.519818
8	9	1.68658	-1.32596	1.42898	-2.08935
9	10	-0.12982	0.631523	-0.586538	0.29072

5. Write a Pandas program to get the data types of the given excel data (create your own excel sheet ) fields.

## Session #1

### Learning Objective

NumPy Installation using different scientific python distributions (Anaconda, Python(x,y), WinPython, Pyzo)

### Learning Context

To install pandas or numpy on Ubuntu 20.04 through apt or pip or conda. **Pandas** is a fast, efficient, modular and easy-to-use open-source framework for data analysis and manipulation. It's designed on top of the Python programming language and thus Pandas is pythonic.

### Materials & Resources

Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, O'Reilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

### Installing via Python Package index pip

To install NumPy, we strongly recommend using a scientific Python distribution.

```
python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose
```

### Ubuntu

```
sudo apt-get install python-numpy python-scipy python-matplotlib ipython ipython-notebook python-pandas
```

---

## Session #2

---

### Learning Objective

NumPy Basics (np.array, np.arange, np.linspace, np.zeros, np.ones, np.random.random, np.empty)

### Learning Context

NumPy stands for numeric python which is a python package for the computation and processing of the multidimensional and single dimensional array elements. It is the fundamental package for scientific computing with Python. It is open-source software. **Travis Oliphant** created NumPy package in 2005 by injecting the features of the ancestor module Numeric into another module Numarray.

### np.array

NumPy is used to work with arrays. The array object in NumPy is called ndarray. We can create a NumPy ndarray object by using the array() function.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5]) print(arr)
print(type(arr))
```

O/P: [1 2 3 4 5]

<class 'numpy.ndarray'>

type(): This built-in Python function tells us the type of the object passed to it. Like in above code it shows that arr is numpy.ndarray type.



## Access Array Elements

Array indexing is the same as accessing an array element. You can access an array element by referring to its index number. The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

### Example

Get third and fourth elements from the following array and add them. import numpy as np

```
arr = np.array([1, 2, 3, 4]) print(arr[2] + arr[3])
```

## Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element. Think of 2-D arrays like a table with rows and columns, where the row represents the dimension and the index represents the column.

### Example

Access the element on the 2nd row, 5th column:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('5th element on 2nd row: ', arr[1, 4])
```

## Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

### Example

Access the third element of the second array of the first array:

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

## Negative Indexing

Negative Indexing is used in Python to begin slicing from the end of the string i.e. the last. Slicing in Python gets a sub-string from a string. The slicing range is set as parameters

Example

Print the last element from the 2nd dim:

i.e. start, stop, and step.

**Use negative indexing to access an array from the end.**

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('Last element from 2nd dim: ', arr[1, -1])
```

## Slicing arrays

Slicing in python means taking elements from one given index to another given index. We pass slice instead of index like this: [start:end].

We can also define the step, like this: [start:end:step]. If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1

Example

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7]) print(arr[1:5])
```

### Negative Slicing

Use the minus operator to refer to an index from the end:

Example

Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7]) print(arr[-3:-1])
np.arange()
```

NumPy arange() is one of the array creation routines based on numerical ranges. It creates an instance of ndarray with evenly spaced values and returns the reference to it.

`numpy.arange([start, ]stop, [step, ], dtype=None) -> numpy.ndarray`

1. start is the number (integer or decimal) that defines the first value in the array.
2. stop is the number that defines the end of the array and isn't included in the array.
3. step is the number that defines the spacing (difference) between each two consecutive values in the array and defaults to 1.
4. dtype is the type of the elements of the output array and defaults to None.
5. step can't be zero. Otherwise, you'll get a `ZeroDivisionError`. You can't move away anywhere from start if the increment or decrement is 0.

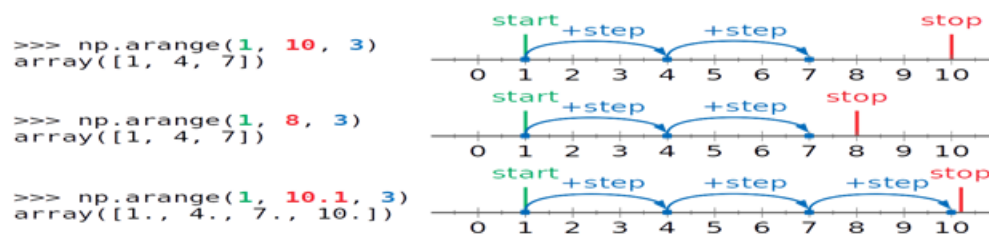
```
import numpy as np
np.arange(start=1, stop=10, step=3)
```

o/p: `array([1, 4, 7])`

In this example, start is 1. Therefore, the first element of the obtained array is 1. step is 3,

which is why your second value is  $1+3$ , that is 4, while the third value in the array is  $4+3$ , which equals 7.

`np.arange(1, 10.1, 3)`



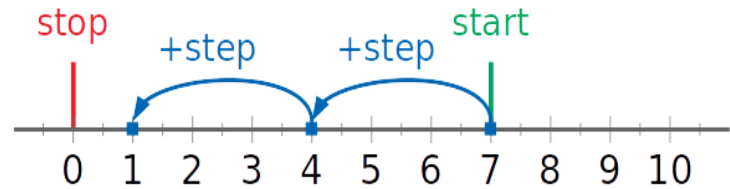
## Providing Negative Arguments

If you provide negative values for start or both start and stop, and have a positive step, then `arange()` will work the same way as with all positive arguments:

```
>>> np.arange(-5, -1)
array([-5, -4, -3, -2])
>>> np.arange(-8, -2, 2)
array([-8, -6, -4])
>>> np.arange(-5, 6, 4)
array([-5, -1, 3])
```

```
>>> np.arange(5, 1, -1)
array([5, 4, 3, 2])
>>> np.arange(7, 0, -3)
array([7, 4, 1])
```

```
>>> np.arange(7, 0, -3)
array([1, 4, 7])
```



### Counting Backwards

Sometimes you'll want an array with the values decrementing from left to right. In such cases, you can use `arange()` with a negative value for `step`, and with a `start` greater than `stop`:

### Getting Empty Arrays

There are several edge cases where you can obtain empty NumPy arrays with `arange()`. These are regular instances of `numpy.ndarray` without any elements.

If you provide equal values for `start` and `stop`, then you'll get an empty array:

```
>>> np.arange(2, 2)
array([], dtype=int64)
np.linspace
```

```
numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)[
source]
```

Return evenly spaced numbers over a specified interval.

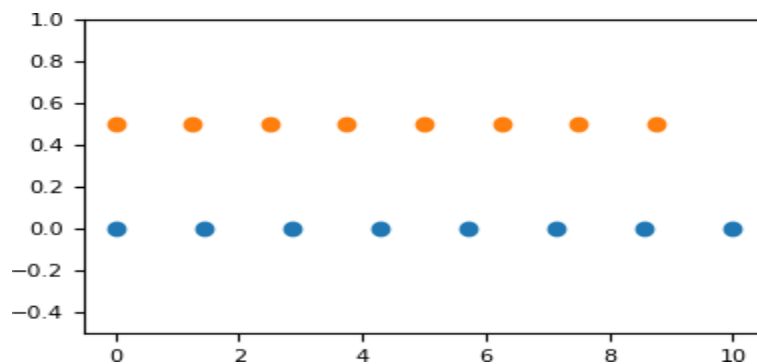
Returns `num` evenly spaced samples, calculated over the interval `[start, stop]`. The endpoint of the interval can optionally be excluded.

```
np.linspace(2.0, 3.0, num=5)
array([2. , 2.25, 2.5 , 2.75, 3. ])
>>> np.linspace(2.0, 3.0, num=5, endpoint=False)
array([2. , 2.2, 2.4, 2.6, 2.8])
>>> np.linspace(2.0, 3.0, num=5, retstep=True)
(array([2. , 2.25, 2.5 , 2.75, 3. ]), 0.25)
```

```

import matplotlib.pyplot as plt
>>> N = 8
>>> y = np.zeros(N)
>>> x1 = np.linspace(0, 10, N, endpoint=True)
>>> x2 = np.linspace(0, 10, N, endpoint=False)
>>> plt.plot(x1, y, 'o')
[<matplotlib.lines.Line2D object at 0x...>]
>>> plt.plot(x2, y + 0.5, 'o')
[<matplotlib.lines.Line2D object at 0x...>]
>>> plt.ylim([-0.5, 1])
(-0.5, 1)
>>> plt.show()

```



`numpy.zeros()`

The `numpy.zeros()` function returns a new array of given shape and type, with zeros.

Syntax:

`numpy.zeros(shape, dtype = None, order = 'C')`

Parameters :

`shape` : integer or sequence of integers

`order` : `C_contiguous` or `F_contiguous`

`C`-contiguous order in memory (last index varies the fastest)

`C` order means that operating row-wise on the array will be slightly quicker FORTRAN-contiguous order in memory (first index varies the fastest).

`F` order means that column-wise operations will be faster.

`dtype` : [optional, float (by default)] Data type of returned array.

Returns :

ndarray of zeros having given shape, order and datatype.

```
import numpy as np
b = np.zeros(2, dtype = int) print("Matrix b : \n", b)
a = np.zeros([2, 2], dtype = int) print("\nMatrix a : \n", a)
c = np.zeros([3, 3]) print("\nMatrix c : \n", c)
O/P
```

```
Matrix b : [0 0]
Matrix a : [[0 0] [0 0]]
Matrix c : [[ 0. 0. 0.] [ 0. 0. 0.] [ 0. 0. 0.]]
```

Code 2 : Manipulating data types

```
import numpy as np
# manipulation with data-types
b = np.zeros((2,), dtype=[('x', 'float'), ('y', 'int')]) print(b)
```

Output :

```
[(0.0, 0) (0.0, 0)]
```

numpy.ones()

The numpy.ones() function returns a new array of given shape and type, with ones.

Syntax: numpy.ones(shape, dtype = None, order = 'C')

```
import numpy as np
b = np.ones(2, dtype = int) print("Matrix b : \n", b)
a = np.ones([2, 2], dtype = int) print("\nMatrix a : \n", a)
c = np.ones([3, 3]) print("\nMatrix c : \n", c)
```

Output :

```
Matrix b : [1 1]
Matrix a : [[1 1] [1 1]]
Matrix c : [[ 1. 1. 1.] [ 1. 1. 1.] [ 1. 1. 1.]]
```

numpy.random.random random.random(size=None)

Return random floats in the half-open interval [0.0, 1.0). Alias for random\_sample to ease forward-porting to the new random API.

```
>>> np.random.random_sample()
```

```
0.47108547995356098
```

Return random floats in the half-open interval [0.0, 1.0).

```
>>> type(np.random.random_sample())
<type 'float'>
>>> np.random.random_sample((5,))
array([ 0.30220482, 0.86820401, 0.1654503 , 0.11659149, 0.54323428])
```

Three-by-two array of random numbers from [-5, 0):

```
>>>
>>> 5 * np.random.random_sample((3, 2)) - 5
array([[ -3.99149989, -0.52338984],
       [ -2.99091858, -0.79479508],
       [ -1.23204345, -1.75224494]])
```

### **numpy.empty().**

The numpy module of Python provides a function called `numpy.empty()`. This function is used to create an array without initializing the entries of given shape and type.

Just like `numpy.zeros()`, the `numpy.empty()` function doesn't set the array values to zero, and it is quite faster than the `numpy.zeros()`. This function requires the user to set all the values in the array manually and should be used with caution.

#### **Syntax**

```
numpy.empty(shape, dtype=float, order='C')
```

shape: int or tuple of ints

This parameter defines the shape of the empty array, such as (3, 2) or (3, 3).

dtype: data-type(optional)

This parameter defines the data type, which is desired for the output array.

order: {'C', 'F'}(optional)

This parameter defines the order in which the multi-dimensional array is going to be stored either in row-major or column-major. By default, the order parameter is set to 'C'

#### **Returns:**

This function returns the array of uninitialized data that have the shape, dtype, and order defined in the function.

#### **Example 1:**

```
import numpy as np
x = np.empty([3, 2])
```

x

#### **Output:**

```
array([[ 7.56544226e-316,  2.07617768e-316],
       [ 2.02322570e-316,  1.93432036e-316],
       [ 1.93431918e-316,  1.93431799e-316]])
```

In the above code

We have imported numpy with alias name np.

We have declared the variable 'x' and assigned the returned value of the np.empty() function.

We have passed the shape in the function.

Lastly, we tried to print the value of 'x' and the difference between elements.

Example 2:

```
import numpy as np
```

```
x = np.empty([3, 3], dtype=float)
```

```
array([[ 2.94197848e+120, -2.70534020e+252, -4.25371363e+003],
       [ 1.44429964e-088,  3.12897830e-053,  1.11313317e+253],
       [-2.28920735e+294, -5.11507284e+039,  0.00000000e+000]])
```

Example 3:

```
import numpy as np
```

```
x = np.empty([3, 3], dtype=float, order='C')
```

```
x
```

```
array([[ 2.94197848e+120, -2.70534020e+252, -4.25371363e+003],
       [ 1.44429964e-088,  3.12897830e-053,  1.11313317e+253],
       [-2.28920735e+294, -5.11507284e+039,  0.00000000e+000]])
```

Output:

In the above code, We have imported numpy with alias name np. We have declared the variable 'x' and assigned the returned value of the np.empty() function. We have passed the shape, data-type, and order in the function.

Lastly, we tried to print the value of 'x' and the difference between elements. In the output, it shows an array of uninitialized values of defined shape, data type, and order.



## Materials & Resources

### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, O'Reilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

---

## Session #3

---

### Learning Objective

Arrays (array.shape, len(array), array.ndim, array.dtype, array.astype(type), type(array))

### Learning Context

A data type is a way to specify the type of data that will be stored in an array. Python lists are a substitute for arrays, but they fail to deliver the performance required while computing large sets of numerical data. Unlike lists, numpy arrays are of fixed size, and changing the size of an array will lead to the creation of a new array while the original array will be deleted.

The dtype attribute of the NumPy array object returns the array's data type

Numpy arrays are faster, more efficient, and require less syntax than standard Python sequences.

### Materials & Resources

#### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

### Shape of an Array

The shape of an array is the number of elements in each dimension.

### Get the Shape of an Array

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

Print the shape of a 2-D array:

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

Create an array with 5 dimensions using ndmin using a vector with values 1,2,3,4 and verify that last dimension has value 4:

```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5) print(arr)
print('shape of array :', arr.shape)
```

### What does the shape tuple represent?

Integers at every index tells about the number of elements the corresponding dimension has.

In the example above at index-4 we have value 4, so we can say that 5th ( 4 + 1 th) dimension has 4 elements.

### Reshaping arrays

The word "reshape" simply indicates changing the shape and that is what this function is used for

- The reshape() function in the NumPy library is mainly used to change the shape of the array without changing its original data.
- Thus reshape() function helps in providing new shape to an array, which can be useful based on your usecase.
- np. reshape(x, m, n): This creates , X matrices with M rows and N columns
- np. reshape(2, 3, 2): This creates , 2 matrices with 3 rows and 2 columns

**Syntax of reshape():** The syntax required to use this function is as follows:

```
numpy.reshape(a, newshape, order='C')
```

```

import numpy as np
a = np.arange(12)
print("The Original array : \n", a)
# shaping the array with 2 rows and 4 columns
a1= np.arange(12).reshape(2, 6)
print("\n The reshaped array with 2 rows and 6 columns : \n",a1)
# shaping the array with 4 rows and 2 columns
a2 = np.arange(12).reshape(6,2)
print("\n The reshaped array with 6 rows and 2 columns : \n", a2)

```

## OUTPUT

The Original array :

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

The reshaped array with 2 rows and 6 columns:

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
```

The reshaped array with 6 rows and 2 columns :

```
[[ 0  1]
 [ 2  3]
 [ 4  5]
 [ 6  7]
 [ 8  9]
 [10 11]]
```

### **# Construction of a 3D array**

```

import numpy as np
a3 = np.arange(12).reshape(2, 3, 2)
a = np.arange(12)
print("The Original array : \n", a)
# shaping the array with 2 rows and 4 columns
a1= np.arange(12).reshape(2, 6)
print("\n The reshaped array with 2 rows and 6 columns : \n", a1)
# shaping the array with 4 rows and 2 columns
a2 = np.arange(12).reshape(6,2)
print("\n The reshaped array with 6 rows and 2 columns : \n", a2)

```

```
# Construction of a 3D array
a3 = np.arange(12).reshape(2, 3, 2)
print("\nAfter reshaping the original array to 3D : \n", a3)
```

#### OUTPUT

The Original array :

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

The reshaped array with 2 rows and 6 columns :

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
```

The reshaped array with 6 rows and 2 columns :

```
[[ 0  1]
 [ 2  3]
 [ 4  5]
 [ 6  7]
 [ 8  9]
 [10 11]]
```

After reshaping the original array to 3D :

```
[[[ 0  1]
   [ 2  3]
   [ 4  5]]
```

```
[[ 6  7]
 [ 8  9]
 [10 11]]]
```

#### **Reshape From 1-D to 2-D**

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

#### OUTPUT

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

**Convert the following 1-D array with 12 elements into a 3-D array.**

The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(2, 3, 2)
print(newarr)
array.ndim
```

OUTPUT

```
[[[ 1  2]
   [ 3  4]
   [ 5  6]]

  [[ 7  8]
   [ 9 10]
  [11 12]]]
```

3

NumPy Arrays provides the `ndim` attribute that returns an integer that tells us how many dimensions the array have.

Example : Check how many dimensions the arrays have:

```
import numpy as np
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

OUTPUT

```
0
1
2
3
```

## Data Types in Python

By default Python have these data types:

- strings - used to represent text data, the text is given under quote marks. e.g. "ABCD"
- integer - used to represent integer numbers. e.g. -1, -2, -3
- float - used to represent real numbers. e.g. 1.2, 42.42
- boolean - used to represent True or False.
- complex - used to represent complex numbers. e.g.  $1.0 + 2.0j$ ,  $1.5 + 2.5j$
- Data Types in NumPy
- NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.
- Below is a list of all data types in NumPy and the characters used to represent them.
  - i - integer
  - b - boolean
  - u - unsigned integer
  - f - float
  - c - complex float
  - m - timedelta
  - M - datetime
  - O - object
  - S - string
  - U - unicode string
  - V - fixed chunk of memory for other type ( void )

## Checking the Data Type of an Array

Note: The NumPy array object has a property called dtype that returns the data type of the array.

Example

**# Get the data type of an array object:**

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

**# Get the data type of an array containing strings:**

```
import numpy as np
arr = np.array(['apple', 'banana', 'cherry'])
print(arr.dtype)
```

### Example

Creating Arrays With a Defined Data Type

We use the array() function to create arrays, this function can take an optional argument: dtype that allows us to define the expected data type of the array elements:

```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='S')
print(arr)
print(arr.dtype)
```

OUTPUT

```
b'1' b'2' b'3' b'4']
|S1
```

For i, u, f, S and U we can define size as well.

Example

**Create an array with data type 4 bytes integer:**

```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='i4')
print(arr)
print(arr.dtype)
```

What if a Value Can Not Be Converted?

If a type is given in which elements can't be casted then NumPy will raise a ValueError.

**ValueError:** In Python ValueError is raised when the type of passed argument to a function is unexpected/incorrect.

A non-integer string like 'a' can not be converted to integer (will raise an error):

```
import numpy as np
arr = np.array(['a', '2', '3'], dtype='i')
```

OUTPUT:



Traceback (most recent call last):  
File "./prog.py", line 3, in  
ValueError: invalid literal for int() with base 10: 'a'  
astype()

### **Converting Data Type on Existing Arrays**

The best way to change the data type of an existing array, is to make a copy of the array with the `astype()` method.

The `astype()` function creates a copy of the array, and allows you to specify the data type as a parameter. The data type can be specified using a string, like 'f' for float, 'i' for integer etc. or you can use the data type directly like float for float and int for integer.

#### **Change data type from float to integer by using 'i' as parameter value:**

```
import numpy as np
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype('i')
print(newarr)
print(newarr.dtype)
output
[1 2 3]
int32
```

#### **Change data type from float to integer by using int as parameter value:**

```
import numpy as np
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype(int)
print(newarr)
print(newarr.dtype)
output
[1 2 3]
int64
```

### Change data type from integer to boolean:

```
import numpy as np
arr = np.array([1, 0, 3])
newarr = arr.astype(bool)
print(newarr)
print(newarr.dtype)
O/P:
[TRUE,FALSE,TRUE]
```

### Type(array) versus dtype(array), len(array)

Python len() method enables us to find the total number of elements in the array. That is, it returns the count of the elements in the array.

Syntax:

```
len(array)
import numpy as np
arr = np.array([1.1, 2.6, 3.7, 4.7, 5.5])
x = arr.copy()
arr[0] = 42
a1=np.array(['divyagdgd'],dtype='S')
a2=np.array(['divyagdgd'])
print(type(arr))
print(arr.dtype)
print(type(a1))
print(a1.dtype)
print(type(a2))
print(a2.dtype)
print(len(arr))
```

O/P:

```
<class 'numpy.ndarray'> float64
<class 'numpy.ndarray'>
|S10
<class 'numpy.ndarray'>
<U10 5
```

## Session #4

### Learning Objective

Array Manipulation (`np.resize` , `np.append`, `np.insert`, `np.delete`, `np.concatenate`, `np.vstack`, `np.hstack`)

### Learning Context

#### Array Manipulations

Several routines are available in NumPy package for manipulation of elements in ndarray object.

#### Adding / Removing Elements

##### **Resize( )**

This function returns a new array with the specified size. If the new size is greater than the original, the repeated copies of entries in the original are contained. The function takes the following parameters. `numpy.resize(arr, shape)`

##### **append( )**

This function adds values at the end of an input array. The append operation is not inplace, a new array is allocated. Also the dimensions of the input arrays must match otherwise `ValueError` will be generated. The function takes the following parameters.

`numpy.append(arr, values, axis)`

**Values:** To be appended to arr. It must be of the same shape as of arr (excluding axis of appending)

**Axis:** The axis along which append operation is to be done. If not given, both parameters are flattened

##### **insert( )**

This function inserts values in the input array along the given axis and before the given index. If the type of values is converted to be inserted, it is different from the input array. Insertion is not done in place and the function returns a new array. Also, if the axis is not mentioned, the input array is flattened.

The insert() function takes the following parameters -

`numpy.insert(arr, obj, values, axis)`

obj- The index before which insertion is to be made

values- The array of values to be inserted

axis- The axis along which to insert. If not given, the input array is flattened

### **delete()**

This function returns a new array with the specified subarray deleted from the input array. As in case of insert() function, if the axis parameter is not used, the input array is flattened. The function takes the following parameters -

`Numpy.delete(arr, obj, axis)`

Obj- Can be a slice, an integer or array of integers, indicating the subarray to be deleted from the input array

Axis- The axis along which to delete the given subarray. If not given, arr is flattened

### **unique( )**

This function returns an array of unique elements in the input array. The function can be able to return a tuple of array of unique vales and an array of associated indices. Nature of the indices depend upon the type of return parameter in the function call.

`numpy.unique(arr, return_index, return_inverse, return_counts)`

### **split()**

This function divides the array into subarrays along a specified axis. The function takes three parameters.

`numpy.split(ary, indices_or_sections, axis)`

ary -Input array to be split

indices\_or\_sections-Can be an integer, indicating the number of equal sized subarrays to be created from the input array. If this parameter is a 1-D array, the entries indicate the points at which a new subarray is to be created.

Axis- Default is 0

The `numpy.hsplit` is a special case of `split()` function where axis is 1 indicating a horizontal split regardless of the dimension of the input array.

`numpy.vsplit` is a special case of `split()` function where axis is 1 indicating a vertical split regardless of the dimension of the input array.

## Joining Arrays

### Concatenate

Concatenation refers to joining. This function is used to join two or more arrays of the same shape along a specified axis. The function takes the following parameters.

`numpy.concatenate((a1, a2, ...), axis)`

a1,a2...-Sequence of arrays of the same type

axis- Axis along which arrays have to be joined. Default is 0

### stack()

This function joins the sequence of arrays along a new axis.

`numpy.stack(arrays, axis)`

arrays: Sequence of arrays of the same shape

axis: Axis in the resultant array along which the input arrays are stacked

### hstack

Variants of `numpy.stack` function to stack so as to make a single array horizontally.

### vstack

Variants of `numpy.stack` function to stack so as to make a single array vertically.

## Materials & Resources

### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

### Adding / Removing Elements

<pre>import numpy as np a = np.array([[1,2,3],[4,5,6]]) print( 'First array:' ) print(a) print('\n') print( 'The shape of first array:' ) print( a.shape ) print( '\n' ) b = np.resize(a, (3,2)) print(' Second array:' ) print(b) print('\n')</pre>	<pre>print(' The shape of second array:' ) print(b.shape) print('\n' ) # Observe that first row of a is repeated in b since size is bigger Print('Resize the second array:' ) b = np.resize(a,(3,3)) print(b)</pre>
--	---

The above program will produce the following output -

First array:  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

The shape of first array: (2, 3)

Second array: $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$

The shape of second array: (3, 2)

Resize the second array:  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix}$

#### **np.append( )**

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print('First array:')
print(a)
print('\n')
print('Append elements to array:')
```

```

print(np.append(a, [7,8,9]) )
print('\n')
print(' Append elements along axis 0:')
print(np.append(a, [[7,8,9]],axis = 0))
print('\n')
print(' Append elements along axis 1:')
print(np.append(a, [[5,5,5],[7,8,9]],axis = 1))

```

Output:

```

First array: [[1 2 3]      [4 5 6]]
Append elements to array: [1 2 3 4 5 6 7 8 9]
Append elements along axis 0: [[1 2 3] [4 5 6] [7 8 9]]
Append elements along axis 1: [[1 2 3 5 5 5] [4 5 6 7 8 9]]

```

### **np.insert( )**

```

import numpy as np
a = np.array([[1,2],[3,4],[5,6]])
print('First array:')
print(a)
print( '\n' )
print ( 'Axis parameter not passed. The input array is flattened before insertion.')
print (np.insert(a,3,[11,12]))
print( '\n' )
print('Axis parameter passed. The values array is broadcast to match input
array.')
print ( 'Broadcast along axis 0:')
print (np.insert(a,2,[11],axis = 0) )
print( '\n' )
print('Broadcast along axis 1:' )
print (np.insert(a,2,11,axis = 1))

```

Output:

```

First array: [[1 2]  [3 4]  [5 6]]
Axis parameter not passed. The input array is flattened before insertion.
[ 1 2 3 11 12 4 5 6]

```

Axis parameter passed. The values array is broadcast to match input array.

Broadcast along axis 0:

```
[[ 1 2] [ 3 4] [11 11] [ 5 6]]
```

Broadcast along axis 1: 

```
[[ 1 2 11] [ 3 4 11] [ 5 6 11]]
```

### **np.delete()**

```
import numpy as np
a = np.arange(12).reshape(3,4)
print('First array:')
print(a)
print( '\n' )
print(' Array flattened before delete operation as axis not used: ' )
print( np.delete(a,5) )
print( '\n' )
print( 'Column 2 deleted:' )
print(np.delete(a,1,axis = 1))
print( '\n' )
print( 'Column 2 deleted:' )
print(np.delete(a,1,axis = 0))
print( '\n' )
print( 'A slice containing alternate values from array deleted:' )
a = np.array([1,2,3,4,5,6,7,8,9,10])
print( np.delete(a, np.s_[::2]))
```

Output :

```
First array:[[ 0  1  2  3] [ 4  5  6  7] [ 8  9 10 11]]
Array flattened before delete operation as axis not used:[ 0 1 2 3 4 6 7 8 9 10 11]
Column 2 deleted:[[ 0  2  3] [ 4  6  7] [ 8 10 11]]
Column 2 deleted:[[ 0 1 2 3] [ 8 9 10 11]]
A slice containing alternate values from array deleted:[ 2 4 6 8 10]
```

### **numpy.unique( )**

```
import numpy as np
a = np.array([5,2,6,2,7,5,6,8,2,9])
```



```

print('First array:')
print(a)
print( '\n' )
print ('Unique values of first array:' ) u = np.unique(a)
print (u)
print( '\n' )
print ('Unique array and Indices array:' )
u,indices = np.unique(a, return_index = True)
print (indices)
print( '\n' )

```

Output:

```

First array: [5 2 6 2 7 5 6 8 2 9]
Unique values of first array: [2 5 6 7 8 9]
Unique array and Indices array: [1 0 2 4 7 9]

```

## Splitting Arrays

### numpy.split()

```

import numpy as np
a = np.arange(9)
print('First array:')
print(a)
print( '\n' )
print ('Split the array in 3 equal-sized subarrays:' )
b = np.split(a,3)
print(b)
print( '\n' )
print( 'Split the array at positions indicated in 1-D array:' )
b = np.split(a,[4,7])
print(b)

```

Output:

```

First array: [0 1 2 3 4 5 6 7 8]
Split the array in 3 equal-sized subarrays:
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8])]

```

Split the array at positions indicated in 1-D array:  
[array([0, 1, 2, 3]), array([4, 5, 6]), array([7, 8])]

### **numpy.hsplit**

```
import numpy as np
a = np.arange(16).reshape(4,4)
print('First array:')
print(a)
print( '\n' )
print('Horizontal splitting:' )
b = np.hsplit(a,2)
print(b)
print( '\n' )
print('VERTICAL splitting:' )
c = np.vsplit(a,2)
print(c)
print( '\n' )
```

Output:

```
First array: [[ 0  1  2  3] [ 4  5  6  7] [ 8  9 10 11] [12 13 14 15]]
Horizontal splitting: [array([[ 0,  1], [ 4,  5], [ 8,  9], [12, 13]]),
array([[ 2,  3], [ 6,  7], [10, 11], [14, 15]])]
VERTICAL splitting:
[array([[0, 1, 2, 3], [4, 5, 6, 7]]), array([[ 8,  9, 10, 11], [12, 13, 14, 15]])]
```

## **Joining Arrays**

### **Concatenate**

```
import numpy as np
a = np.array([[1,2],[3,4]])
print('First array:')
print (a)
print ( '\n' )
b = np.array([[5,6],[7,8]])
print ( 'Second array:' )
```

```

print (b)
print ('\n' )
# both the arrays are of same dimensions
print ('Joining the two arrays along axis 0:' )
print (np.concatenate((a,b)) )
print ('\n' )
print ('Joining the two arrays along axis 1:' )
print (np.concatenate((a,b),axis = 1))

```

#### Output

```

First array:[[1 2][3 4]]
Second array:[[5 6]      [7 8]]
Joining the two arrays along axis 0:
    [[1  2]
     [3  4]
     [5  6]
     [7  8]]
Joining the two arrays along axis 1: [[1 2 5 6] [3 4 7 8]]

```

#### **Numpy.stack()**

```

import numpy as np
a = np.array([[1,2],[3,4]])
print('First array:')
print (a)
print ('\n' )
b = np.array([[5,6],[7,8]])
print ('Second array:' )
print (b)
print ('\n' )
print ('Stack the two arrays along axis 0:' )
print (np.stack((a,b),0) )
print ('\n' )
print ('Stack the two arrays along axis 1:' )
print (np.stack((a,b),1))

```

## Output

First array:

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

Second array:

```
[[5 6]
 [7 8]]
```

Stack the two arrays along axis 0:

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

```
 [[5 6]
  [7 8]]]
```

Stack the two arrays along axis 1:

```
[[[1 2]
  [5 6]]
```

```
 [[3 4]
  [7 8]]]
```

## **numpy.hstack**

```
import numpy as np
a = np.array([[1,2],[3,4]])
print('First array:')
print(a)
print('\n')
b = np.array([[5,6],[7,8]])
print('Second array:')
print(b)
print('\n')
print('Horizontal stacking:')
c = np.hstack((a,b))
print(c) print('\n')
```

## Output

```
First array: [[1 2]
[3 4]]
Second array: [[5 6]
[7 8]]
Horizontal stacking:
[[1 2 5 6]
[3 4 7 8]]
```

## numpy.vstack

```
import numpy as np
a = np.array([[1,2],[3,4]])
print('First array:')
print(a)
print('\n')
b = np.array([[5,6],[7,8]])
print('Second array:')
print(b)
print('\n')
print('Horizontal stacking:')
c = np.vstack((a,b))
print(c)
print('\n')
```

## Output

```
First array:
[[1 2]
[3 4]]
Second array:
[[5 6]
[7 8]]
Horizontal stacking:
[[1 2]
[3 4]
[5 6]
[7 8]]
```

---

## Session #5

---

### Learning Objective

Mathematical Operations (np.add, np.subtract, np.divide, np.multiply, np.sqrt, np.sin, np.cos, np.log, np.dot, np.roots) , Statistical Operations( np.mean, np.median, np.std, array.corrcoef( ) )

### Learning Context

#### **NumPy - Arithmetic Operations**

Input arrays for performing arithmetic operations such as add(), subtract(), multiply(), and divide() must be either of the same shape or should conform to array broadcasting rules.

#### **# Python program explaining numpy.add() function when inputs are scalar**

```
import numpy as np
in_num1 = 10
in_num2 = 15
print ("1st Input number : ", in_num1)
print ("2nd Input number : ", in_num2)
out_num = np.add(in_num1, in_num2)
```

#### **#Python program explaining arithmetic operations when inputs are not scalar**

```
import numpy as np
a=np.arange(9, dtype=float).reshape(3,3)
print(a)
b=np.array([10,10,10])
print(b)
print(a+b)
print(np.add(a,b))
print(a-b)
```

```
print(np.subtract(a,b))
print(a*b)
print(np.multiply(a,b))
print(a/b)
print(np.divide(a,b))
```

### Dot Product

This function returns the dot product of two arrays. For 2-D vectors, it is the equivalent to matrix multiplication. For 1-D arrays, it is the inner product of the vectors. For N-dimensional arrays, it is a sum product over the last axis of a and the second-last axis of b.

#### # Python program explaining log() function

```
import numpy as np
print(np.log(2**8))
print(np.log(4**4))
a=np.array([[1,2],[3,4]])
b=np.array([[11,12],[13,14]])
print(np.dot(a,b))
x=10
y=20
print(np.dot(x,y))
```

Output

```
5.545177444479562
5.545177444479562
[[37 40]
 [85 92]]
200
```

Note that the dot product is calculated as -

```
[[1*11+2*13, 1*12+2*14],[3*11+4*13, 3*12+4*14]]
```

**numpy.roots():** To find roots for the given values.

Syntax: `numpy.roots(p)`

Parameter

It takes the coefficients of an given polynomial.

Return Value : The function will return the roots of the polynomial.

Example :

Let us consider an equation:  $x^2 + 5x + 6$

The coefficients are 1, 5 and 6.

```
#numpy.roots(p)
```

```
import numpy as np
```

```
p=[1,5,6]
```

```
roots=np.roots(p)
```

```
print(roots)
```

Output : [-3. -2.]

### **Roots of three-degree polynomial**

To find the roots of the three-degree polynomial we need to factorize the given polynomial equation

first so that we get a linear and quadratic equation. Then, we can easily determine the zeros of the

three-degree polynomial. Let us understand with the help of an example.

Example:  $2x^3 - x^2 - 7x + 2$

Divide the given polynomial by  $x - 2$  since it is one of the factors.

$$2x^3 - x^2 - 7x + 2 = (x - 2)(2x^2 + 3x - 1)$$

Now we can get the roots of the above polynomial since we have got one linear equation and one

quadratic equation for which we know the formula.

Now let us consider the following polynomial for a cubic equation:

$$x^3 - 6x^2 + 11x - 6$$

The coefficients are 1, -6, 11 and -6.

```
#Third degree polynomial numpy.roots(p)
```

```
import numpy as np
```

```
p=[1,-6,11,-6]
```





```
roots=np.roots(p)
```

```
print(roots)
```

```
Output: [3. 2. 1.]
```

## Trigonometric Functions

NumPy has standard trigonometric functions which return trigonometric ratios for a given angle in radians.

```
#Trigonometric functions Given angle in radians)
```

```
import numpy as np
```

```
a=np.array([0,30,45,60,90])
```

```
print("sine values")
```

```
print(np.sin(a*np.pi/180))
```

```
print("Tan values")
```

```
print(np.tan(a*np.pi/180))
```

```
print("Cosine values")
```

```
print(np.cos(a*np.pi/180))
```

Output

```
sine values
```

```
[0.      0.5      0.70710678 0.8660254  1.      ]
```

```
Tan values
```

```
[0.00000000e+00 5.77350269e-01 1.00000000e+00 1.73205081e+00  
1.63312394e+16]
```

```
Cosine values
```

```
[1.00000000e+00 8.66025404e-01 7.07106781e-01 5.00000000e-01  
6.12323400e-17]
```

## NumPy - Statistical Functions

**numpy.median()** : Median is defined as the value separating the higher half of a data sample from the lower half.

Example

```
#numpy.median(--Separates upper half from lower half )
```

```
import numpy as np
```

```
a=np.array([[30,65,70],[80,95,10],[50,90,60]])
```

```
print(a)
print(np.median(a))
print(np.median(a,axis=0))
print(np.median(a,axis=1))
```

Output

```
[[30 65 70]
 [80 95 10]
 [50 90 60]]
65.0
[50. 90. 60.]
[65. 80. 60.]
```

**numpy.mean()** : Arithmetic mean is the sum of elements along an axis divided by the number of elements. The `numpy.mean()` function returns the arithmetic mean of elements in the array. If the axis is mentioned, it is calculated along it.

```
[0. 0.5 0.70710678 0.8660254 1. ]
```

Example

```
#numpy.mean(--Sum of elements along axis divided by number of elements)
import numpy as np
a=np.array([[1,2,3],[3,4,5],[4,5,6]])
print(a)
print(np.mean(a))
print(np.mean(a,axis=0))
print(np.mean(a,axis=1))
```

Output

```
[[1 2 3]
 [3 4 5]
 [4 5 6]]
3.6666666666666665
[2.66666667 3.66666667 4.66666667]
[2. 4. 5.]
```

**numpy.average()** : Weighted average is an average resulting from the multiplication of each component by a factor reflecting its importance. The `numpy.average()` function computes the weighted average of elements in an array according to their respective weight given in another array. The function can have an axis parameter. If the axis is not specified, the array is flattened.

Considering an array `[1,2,3,4]` and corresponding weights `[4,3,2,1]`, the weighted average is calculated by adding the product of the corresponding elements and dividing the sum by the sum of weights.

Weighted average =  $(1*4+2*3+3*2+4*1)/(4+3+2+1)$

Example

```
#numpy.average() --Weighted average is average resulting from the
#multiplication of each component by a factor reflecting its importance.
import numpy as np
a=np.array([1,2,3,4])
print(a)
print(np.average(a))
wt=np.array([4,3,2,1])
print(np.average(a,weights=wt))
wavg=np.average([1,2,3,4],weights=[4,3,2,1],returned=True)
print(wavg)
wavg1=np.average([1,2,3,4],weights=[4,3,2,1],axis=0,returned=True)
print(wavg1)
```

Output

```
[1 2 3 4]
2.5
2.0
(2.0, 10.0)
(2.0, 10.0)
```

In a multi-dimensional array, the axis for computation can be specified.

Example

```
#Specifying axis for average in a multidimensional array
import numpy as np
a=np.arange(6).reshape(3,2)
print(a)
wt1=np.array([3,5])
print(np.average(a,axis=1,weights=wt1))
wt2=np.array([1,3,5])
print(np.average(a,axis=0,weights=wt2))
```

Output

```
[[0 1]
 [2 3]
 [4 5]]
[0.625 2.625 4.625]
[2.88888889 3.88888889]
```

**Standard Deviation** : Standard deviation is the square root of the average of squared deviations from mean. The formula for standard deviation is as follows -

```
std = sqrt(mean(abs(x - x.mean())**2))
```

If the array is [1, 2, 3, 4], then its mean is 2.5. Hence the squared deviations are [2.25, 0.25, 0.25, 2.25] and the square root of its mean divided by 4, i.e.,  $\sqrt{5/4}$  is 1.1180339887498949.

Example

```
#Standard Deviation-Square root of average of squared deviations from mean
#std=sqrt(mean(abs(x-x.mean())**2))
import numpy as np
print(np.std([1,2,3,4]))
```

Output

```
1.118033988749895
```

**Variance** : Variance is the average of squared deviations, i.e.,  $\text{mean}(\text{abs}(x - x.\text{mean()})**2)$ . In other words, the standard deviation is the square root of variance.

Example

```
#Variance is average of squared deviations from mean
#std=sqrt(mean(abs(x-x.mean())**2))
import numpy as np
print(np.var([1,2,3,4]))
```

Output

1.25

**Pearson Correlation Coefficient:** It is most frequently used correlation metrics in machine learning or statistics.

**Pearson correlation = covariance(x,y)/std(x) X std(y)**

Covariance is measure of variation between x and y variable. std(x) is standard deviation of variable x and std(y) is standard deviation of variable y.

```
#Import Libraries
import pandas as pd
import seaborn as sns
# Get the dataset from seaborn library
tips = sns.load_dataset('tips')
# Get pearson correlation coefficient
tip.corr(method='pearson')
# Get spearman correlation coefficient
tip.corr(method='spearman')
```

## Materials & Resources

### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Wilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Session #6

### Learning Objective

Data Types in NumPy

### Learning Context

#### Data Types in Python

By default Python have these data types:

- **strings** - used to represent text data, the text is given under quote marks. e.g. "ABCD"
- **integer** - used to represent integer numbers. e.g. -1, -2, -3
- **float** - used to represent real numbers. e.g. 1.2, 42.42
- **boolean** - used to represent True or False.
- **complex** - used to represent complex numbers. e.g.  $1.0 + 2.0j$ ,  $1.5 + 2.5j$

NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent them.

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type ( void )

## Materials & Resources

### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Wilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

### Checking the Data Type of an Array

Note: The NumPy array object has a property called `dtype` that returns the data type of the array.

#### Example

Get the data type of an array object:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

Get the data type of an array containing strings:

```
import numpy as np
arr = np.array(['apple', 'banana', 'cherry'])
print(arr.dtype)
```

### Creating Arrays With a Defined Data Type

We use the `array()` function to create arrays, this function can take an optional argument: `dtype` that allows us to define the expected data type of the array elements:

#### Example

Create an array with data type string:

```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='S')
print(arr)
print(arr.dtype)
```



For i, u, f, S and U we can define size as well

Example

Create an array with data type 4 bytes integer:

```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='i4')
print(arr)
print(arr.dtype)
```

Output:

```
[1 2 3 4]
int32
```

### What if a Value Can Not Be Converted?

If a type is given in which elements can't be casted then NumPy will raise a ValueError

A non-integer string like 'a' cannot be converted to integer (will raise an error):

<pre>import numpy as np arr = np.array(['a', '2', '3'], dtype='i')</pre>	<p>Output:</p> <p>Traceback (most recent call last):</p> <p>File "./prog.py", line 3, in</p> <p>ValueError: invalid literal for int() with base 10: 'a' astype()</p>
--	--

### Converting Data Type on Existing Arrays

The best way to change the data type of an existing array, is to make a copy of the array with the astype() method.

The astype() function creates a copy of the array, and allows you to specify the data type as a Parameter

The data type can be specified using a string, like 'f' for float, 'i' for integer etc. or you can use the data type directly like float for float and int for integer.

**Change data type from float to integer by using 'i' as parameter value:**

<pre>import numpy as np arr = np.array([1.1, 2.1, 3.1]) newarr = arr.astype('i') print(newarr) print(newarr.dtype)</pre>	<p>Output</p> <pre>[1 2 3] int32</pre>
--	--

**Change data type from float to integer by using int as parameter value:**

<pre>import numpy as np arr = np.array([1.1, 2.1, 3.1]) newarr = arr.astype(int) print(newarr) print(newarr.dtype)</pre>	<pre>Output [1 2 3] Int32</pre>
--	---------------------------------

**Change data type from integer to Boolean:**

<pre>import numpy as np arr = np.array([1, 0, 3]) newarr = arr.astype(bool) print(newarr) print(newarr.dtype)</pre>	<pre>output: [ True False  True] bool</pre>
---	---

**Example**

<pre>import numpy as np arr = np.array([1.1, 2.6, 3.7, 4.7, 5.5]) x = arr.copy() arr[0] = 42 a1=np.array(['divyagdgfd'],dtype='S') a2=np.array(['divyagdgfd']) print(type(arr)) print(arr.dtype) print(type(a1)) print(a1.dtype) print(type(a2)) print(a2.dtype) print(len(arr))</pre>	<pre>output: &lt;class 'numpy.ndarray'&gt; float64 &lt;class 'numpy.ndarray'&gt;  S10 &lt;class 'numpy.ndarray'&gt; &lt;U10 5</pre>
--	---

---

## Session #7

---

### Learning Objective

NumPy ndarray

### Learning Context

ndarray is one of the most important classes in the NumPy python library. It is basically a multidimensional or n-dimensional array of fixed size with homogeneous elements (i.e., the data type of all the elements in the array is the same).

In Numpy, the number of dimensions of the array is given by Rank. Thus, in the above example, the ranks of the array of 1D, 2D, and 3D arrays are 1, 2 and 3 respectively.

```
np.ndarray(shape, dtype=float, buffer=None, offset=0, strides=None,
           order=None)
```

### Materials & Resources

Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

### Working with Ndarray

An array can be created using the following functions:

- `np.ndarray(shape, type)`: Creates an array of the given shape with random numbers.
- `np.array(array_object)`: Creates an array of the given shape from the list or tuple.
- `np.zeros(shape)`: Creates an array of the given shape with all zeros.
- `np.ones(shape)`: Creates an array of the given shape with all ones.
- `np.full(shape,array_object, dtype)`: Creates an array of the given shape with complex numbers.
- `np.arange(range)`: Creates an array with the specified range.
- `shapetuple of ints` - Shape of created array.
- `dtype-data-type, optional`- Any object that can be interpreted as a numpy data type.
- `bufferobject exposing buffer interface, optional`- Used to fill the array with data.
- `offsetint, optional`- Offset of array data in buffer.
- `stridestuple of ints, optional`- Strides of data in memory.
- `order{'C', 'F'}, optional`- Row-major (C-style) or column-major (Fortran-style) order.

There are two modes of creating an array using `new` :

1. If `buffer` is `None`, then only `shape`, `dtype`, and `order` are used.
2. If `buffer` is an object exposing the buffer interface, then all keywords are interpreted.

### Example

<pre>import numpy as np arr=np.ndarray(shape=(2,2),dtype=float,order='F',buffer = None) arr=([[1.56, 3.43],[3.6987, 2.5323]]) print(arr)</pre>	<pre>output: [[1.56, 3.43],  [3.6987,  2.5323]]</pre>
--	---

### `array.ndim`

NumPy Arrays provides the `ndim` attribute that returns an integer that tells us how many dimensions the array has.

### Example

Check how many dimensions the arrays have:

<pre>import numpy as np a = np.array(42) b = np.array([1, 2, 3, 4, 5])</pre>	<pre>Output: 0 1</pre>
--	------------------------

<pre>c = np.array([[1, 2, 3], [4, 5, 6]])</pre>	2
<pre>d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])</pre>	3
<pre>print(a.ndim)</pre>	
<pre>print(b.ndim)</pre>	
<pre>print(c.ndim)</pre>	
<pre>print(d.ndim)</pre>	

## Session #8

### Learning Objective

NumPy String Operations

.

### Learning Context

The following functions are used to perform vectorized string operations for arrays of dtype `numpy.string_` or `numpy.unicode_`. They are based on the standard string functions in Python's built-in library.

Function	Description
<code>add()</code>	Returns element-wise string concatenation for two arrays of <code>str</code> or <code>Unicode</code>
<code>multiply()</code>	Returns the string with multiple concatenation, element-wise
<code>center()</code>	Returns a copy of the given string with elements centered in a string of specified length
<code>capitalize()</code>	Returns a copy of the string with only the first character capitalized
<code>title()</code>	Returns the element-wise title cased version of the string or unicode
<code>lower()</code>	Returns an array with the elements converted to lowercase
<code>upper()</code>	Returns an array with the elements converted to uppercase
<code>split()</code>	Returns a list of the words in the string, using separate or delimiter
<code>splitlines()</code>	Returns a list of the lines in the element, breaking at the line boundaries
<code>strip()</code>	Returns a copy with the leading and trailing characters removed
<code>join()</code>	Returns a string which is the concatenation of the strings in the sequence
<code>replace()</code>	Returns a copy of the string with all occurrences of substring replaced by the new string

## Materials & Resources

### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

**Add()** - This function performs elementwise string concatenation.

<pre>#String Operations import numpy as np print("Concatenate two strings:") print(np.char.add(['hello'], [' xyz'])) print('\n') print('Concatenation example:') print(np.char.add(['hello', 'hi'], [' abc', ' xyz']))</pre>	<pre>output Concatenate two strings: ['hello xyz'] Concatenation example: ['hello abc' 'hi xyz']</pre>
--	--

**Multiply()** - This function performs multiple concatenations.

<pre>import numpy as np print np.char.multiply('RamiNaidu ',3)</pre>	<pre>outpu RamiNaidu  RamiNaidu  RamiNaidu</pre>
--	--

**Center()** - This function returns an array of the required width so that the input string is centered and padded on the left and right with fillchar.

<pre>import numpy as np print(np.char.center('Rami Naidu', 20, fillchar = '*'))</pre>	<pre>output *****Rami Naidu*****</pre>
---	--

**Capitalize()** - This function returns the copy of the string with the first letter capitalized.

<pre>import numpy as np print(np.char.capitalize('welcome to srkrec'))</pre>	<pre>output Welcome to srkrec</pre>
--	-------------------------------------

**Title()** - This function returns a title cased version of the input string with the first letter of each word capitalized.

<pre>import numpy as np print(np.char.title('hello how are you?'))</pre>	<pre>output Hello How Are You?</pre>
--	--------------------------------------

**Lower()** - This function returns an array with elements converted to lowercase. It calls `str.lower` for each element.

<pre>import numpy as np print(np.char.lower(['HELLO', 'WORLD'])) print(np.char.lower('HELLO'))</pre>	<pre>output ['hello' 'world'] hello</pre>
--	---

**Upper()** - This function calls `str.upper` function on each element in an array to return the uppercase array elements.

<pre>import numpy as np print(np.char.upper('hello')) print(np.char.upper(['hello', 'world']))</pre>	<pre>output HELLO ['HELLO' 'WORLD']</pre>
--	---

**Splitlines()** - This function returns a list of elements in the array, breaking at line boundaries.

<pre>import numpy as np print(np.char.splitlines('hello\nhow are you?')) print(np.char.splitlines('hello\rhow are you?'))</pre>	<pre>output ['hello', 'how are you?'] ['hello', 'how are you?']</pre>
---	---

'\n', '\r', '\r\n' can be used as line boundaries.

**Strip()** - This function returns a copy of array with elements stripped of the specified characters leading and/or trailing in it.

<pre>import numpy as np print(np.char.strip('Rami Naidu', 'i')) print(np.char.strip(['Rami', 'Naidu', 'Python'], 'i'))</pre>	<pre>output Rami Naidu ['Ram' 'Naidu' 'Python']</pre>
--	---



**Join()** - This method returns a string in which the individual characters are joined by separator character specified.

import numpy as np print(np.char.join(':', 'dmy')) print(np.char.join([':', '-'], ['dmy', 'ymd']))	output d:m:y ['d:m:y' 'y-m-d']
--	--------------------------------------

**Replace()** - This function returns a new copy of the input string in which all occurrences of the sequence of characters is replaced by another given sequence.

import numpy as np print(np.char.replace('He is a good boy', 'is', 'was'))	output He was a good boy
---	-----------------------------

## Session #9

### Learning Objective

NumPy Financial functions

### Learning Context

The numpy-financial Python package is a collection of elementary financial functions. These functions were copied to this package from version 1.17 of NumPy. The source code for this package is available at <https://github.com/numpy/numpy-financial>. The importable name of the package is `numpy_financial`. The recommended alias is `npf`.

#### Example

What is the future value after 10 years of saving \$200 now, with an additional monthly savings of \$200. Assume the interest rate is 6% (annually) compounded monthly?

```
import numpy_financial as npf
a=npf.fv(0.06/12, 10*12, -200, -200)
print(a)
```

output

```
33139.748708098065
```

```
import numpy_financial as npf
b=npf.pv(0.06/12, 10*12, -200, 33139.748708098065)
print(b)
```

output

```
-200.000000000000185
```

Name	Description	Syntax
fv()	Compute the future value.	<code>numpy.fv(rate, nper, pmt, pv, when='end')</code> [source]
pv()	Compute the present value.	<code>numpy.pv(rate, nper, pmt, fv=0.0,when='end')</code>

npv()	Returns the NPV (Net Present Value) of a cash flow series.	numpy.npv(rate, values)
pmt()	Compute the payment against loan principal plus interest.	numpy.pmt(rate, nper, pv, fv=0, when='end')
ppmt()	Compute the payment against loan principal.	numpy.ppmt(rate, per, nper, pv, fv=0.0, when = 'end')
ipmt	Compute the interest portion of a payment.	numpy.ipmt(rate, per, nper, pv, fv=0.0, when = 'end')
irr	The (IRR) function return the Internal Rate of Return.	numpy.irr(values)
mirr	Modified internal rate of return.	numpy.mirr(values, finance_rate, reinvest_rate)
nper()	Compute the number of periodic payments.	numpy.nper(rate, pmt, pv, fv=0, when='end')
rate()	Compute the rate of interest per period.	numpy.rate(nper, pmt, pv, fv, when='end', guess=0.1, tol=1e-06, maxiter=100)

## Materials & Resources

### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Additson Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus VilhelmPersson and Luiz Felipe Martins, Mastering Python Data Analysis, PacktPublishing Ltd

## Exercise

**NumPy.fv** - Compute the future value.

Example:

By convention, the negative sign represents cash flow out (i.e. money not available today). Thus, saving \$200 a month at 6% annual interest leads to \$33,139.75 available to spend in 10 years.

If any input is array\_like, returns an array of equal shape. Let's compare different interest rates from the example above.

<pre>import numpy_financial as npf import numpy as np x = np.array((0.04, 0.06, 0.07))/12 a=npf.fv(x, 10*12, -200, -200) print(a) b=npf.pv(x, 10*12, -200, 33139.75) print(b)</pre>	<pre>output [29748.12748158      33139.7487081  35018.89376205] [-2474.98535444 -200.00071007   735.05492211]</pre>
---	---

**NumPy. npv()** - The npv() function is used to get the NPV (Net Present Value) of a cash flow series.

Syntax:

`numpy.npv(rate, values)`

values - the values of the time series of cash flows.

The (fixed) time interval between cash flow "events" must be the same as that for which rate is given (i.e., if rate is per year, then precisely a year is understood to elapse between each cash flow event). By convention, investments or "deposits" are negative, income or "withdrawals" are positive; values must begin with the initial investment, thus values[0] will typically be negative.

array\_like, shape(M, )

<pre>import numpy_financial as np a=np.npv(0.281,[ -200, 39, 59, 55,20]) print(a) print(np.npv(0.281,[-100, 39, 55, 20])</pre>	<pre>output -100.00847859163845 -26.52366357393724</pre>
--	--

NPV value with rate 0.5 is: -85.92592592592592 NPV value with rate 0.3 is: -81.83887118798361 NPV value with rate 1 is: -91.25

<pre>import numpy_financial as np values = [-100,10, 10,10] rate1= 0.50 rate2= 0.30 rate3= 1 print("NPV value with rate ", rate1, " is: ",np.npv(rate1, values)) print("NPV value with rate ",rate2," is:",np.npv(rate2 , values)) print("NPV value with rate ",rate3,"is:",np.npv(rate3, values))</pre>	<pre>output NPV value with rate 0.5 is: -85.92592592592592 NPV value with rate 0.3 is: -81.83887118798361 NPV value with rate 1 is: -91.25</pre>
--	--

**NumPy.pmt()** - The pmt() function is used to compute the payment against loan principal plus interest.

Syntax:

`numpy.pmt(rate, nper, pv, fv=0, when='end')`

<pre>import numpy_financial as npf print(npf.pmt(0.085/12,12*12, 100000)) print( npf.ppmt(0.085/12,1, 12*12, 100000))</pre>	<pre>output -1110.0555643145096 -401.72223098117627</pre>
---	---

<pre>import numpy as np y = np.array([[2,4,6], [1,3,5], [9,7,8]]) print(np.apply_along_axis(sorted, 1, y)) print(np.apply_along_axis(sorted, 0, y))</pre>	<pre>output [[2 4 6]  [1 3 5]  [7 8 9]] [[1 3 5]  [2 4 6]  [9 7 8]]</pre>
---	---

A DataFrame object has two axes: "axis 0" and "axis 1". "axis 0" represents rows and "axis 1" represents columns. Now it's clear that Series and DataFrame share the same direction for "axis 0" - it goes along rows direction.

```
import numpy as np
y = np.array([[2,4,6], [1,3,5], [9,7,8]])
print(np.apply_along_axis(np.diag, -1, y))
```

output

```
[[[2  0  0]
  [0  4  0]
  [0  0  6]]

 [[1  0  0]
  [0  3  0]
  [0  0  5]]

 [[9  0  0]
  [0  7  0]
  [0  0  8]]]
```

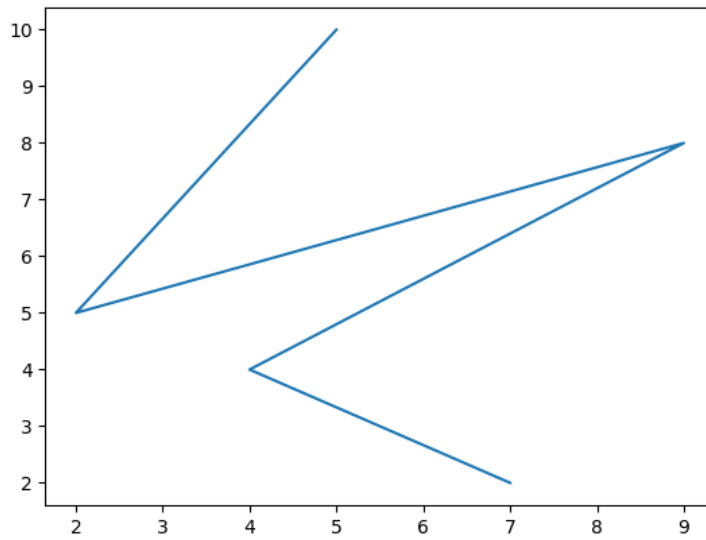
```
import numpy as np
x = np.arange(24).reshape(3,2,4)
print(np.apply_over_axes(np.sum, x, [0,0]))
```

output

```
[[[24 27 30 33]
  [36 39 42 45]]]
```

```
# importing matplotlib module
from matplotlib import pyplot as plt # x-axis values
x = [5, 2, 9, 4, 7]
# Y-axis values
y = [10, 5, 8, 4, 2]
# Function to plot
plt.plot(x, y)
# function to show the plot
plt.show()
```

output



```
import numpy as np
a=np.ndarray((4,2),
buffer=np.arange(9),dtype=int)
print(a)
G=np.ndarray((3,2),
buffer=np.array([1,2,3,4,5,6]),dtype=int)
print ("Array with buffer specified:\n", G)
```

output

```
[[0 1]
 [2 3]
 [4 5]
 [6 7]]
Array with buffer specified:
[[1 2]
 [3 4]
 [5 6]]
```

## Session #10

### Learning Objective

NumPy Functional Programming

### Learning Context

Functional programming is a programming paradigm in which the primary method of computation is evaluation of functions.

A pure function is a function whose output value follows solely from its input values, without any observable side effects. In functional programming, a program consists entirely of evaluation of pure functions. Computation proceeds by nested or composed function calls, without changes to state or mutable data.

The functional paradigm is popular because it offers several advantages over other programming paradigms. Functional code is:

- High level: You're describing the result you want rather than explicitly specifying the steps required to get there. Single statements tend to be concise but pack a lot of punch.
- Transparent: The behavior of a pure function depends only on its inputs and outputs, without intermediary values. That eliminates the possibility of side effects, which facilitates debugging.
- Parallelizable: Routines that don't cause side effects can more easily run in parallel with one another.

To support functional programming, it's useful if a function in a given programming language has two abilities:

1. To take another function as an argument
2. To return another function to its caller

**`apply_along_axis(func1d, axis, arr, *args, ...)`** - Apply a function to 1-D slices along the given axis.

**`apply_over_axes(func, a, axes)`** - Apply a function repeatedly over multiple axes.

**`vectorize(pyfunc[, otypes, doc, excluded, ...])`** - Generalized function class.



## Materials & Resources

### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Wilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

### Example\_1

<pre>def my_func(a):     """Average first and last element of a 1-D array"""     return (a[0] + a[-1]) * 0.5 b = np.array([[1,2,3], [4,5,6], [7,8,9]]) np.apply_along_axis(my_func, 0, b)  np.apply_along_axis(my_func, 1, b)</pre>	<p>O/P : array([4., 5., 6.])</p> <p>O/P : array([2., 5., 8.])</p>
---	---

### Example\_2

<pre>import numpy as np y = np.array([[2,4,6], [1,3,5], [9,7,8]]) print(y) print('\n') print(np.apply_along_axis(sorted, 1, y)) print('\n') print(np.apply_along_axis(sorted, 0, y))</pre>	<p>OUTPUT</p> <p>[[2 4 6] [1 3 5] [9 7 8]]</p> <p>[[2 4 6] [1 3 5] [7 8 9]]</p> <p>[[1 3 5] [2 4 6] [9 7 8]]</p>
--	--

## MODULE 2: PANDAS

### Introduction To Pandas:

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python.

Additionally, it has the broader goal of becoming the most powerful and flexible open-source data analysis/manipulation tool available in any language. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, Series (1-dimensional) and Data Frame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, Data Frame provides everything that R's data. Frame provides and much more. pandas are built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

### Installing Pandas:

After you launch the command prompt, the next step in the process is to type in the required command to initialize pip installation.

Enter the command “pip install pandas” on the terminal. This should launch the pip installer. The required files will be downloaded, and Pandas will be ready to run on your computer.

## Session #1

### Learning Objective

Perform the following:

- 1) Pandas Installation
- 2) Creating Data Frames

### Learning Context

**Pandas** is a fast, efficient, modular and easy-to-use open-source framework for data analysis and manipulation. It's designed on top of the Python programming language and thus Pandas is pythonic.

In order to check if pandas are already installed, use **pip list** command to get the list of the package installed.

If you don't have Pandas installed then this command doesn't list it.

You can also use **pip3 list** command if you have pip3 installed.

if you already have Pandas installed but it is an old version, you can upgrade Pandas to the latest or to a specific version using **python -m pip install --upgrade pip**.

### Materials & Resources

Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

### 1. Install pandas on Linux

Install pandas using apt-get - **sudo apt-get install python3-pandas**

### 2. Creating Data Frames

Pandas Data Frame is a 2-dimensional labeled data structure like any table with rows and columns. The size and values of the data frame are mutable., can be modified. It is the most commonly used panda object.

To create a dataframe, we need to import pandas. Dataframe can be created using dataframe() function. The dataframe() takes one or two parameters. The first one is the data which is to be filled in the dataframe table. The data can be in form of list of lists or dictionary of lists. In case of list of lists data, the second parameter is the columns name.

Pandas Data Frame can be created in multiple ways.

#### Create an empty data frame

We can create a basic empty Dataframe. The dataframe constructor needs to be called to create the DataFrame.

#### Example

<pre># import pandas as pd import pandas as pd # Calling DataFrame constructor df = pd.DataFrame() print(df)</pre>	<pre>output Empty DataFrame Columns: [] Index: []</pre>
--	---

#### Create a data frame using List

We can create dataframe using a single list or list of lists.

<pre># importing pandas library import pandas as pd # string values in the list lst = ['Python', 'JAVA', 'C++', 'C', 'DBMS', 'DS', 'WT'] # Calling DataFrame constructor on list dframe = pd.DataFrame(lst) print(dframe)</pre>	<pre>Output 0 0 Python 1 JAVA 2 C++ 3 C 4 DBMS 5 DS 6 WT</pre>
---	--

### Create Dataframe from dict of ndarray/lists

The dict of ndarray/lists can be used to create a dataframe, all the ndarray must be of the same length. The index will be a range(n) by default; where n denotes the array length.

#### Example

<pre>import pandas as pd data= {'Name' : ['RamiNaidu', 'Bhaksar', 'Manikanta'], 'Age': [31, 25, 23]} df = pd.DataFrame(data) print(df)</pre>	<div>Output</div> <table><tr><th></th><th>Name</th><th>Age</th></tr><tr><td>0</td><td>RamiNaidu</td><td>31</td></tr><tr><td>1</td><td>Bhaksar</td><td>25</td></tr><tr><td>2</td><td>Manikanta</td><td>23</td></tr></table>		Name	Age	0	RamiNaidu	31	1	Bhaksar	25	2	Manikanta	23
	Name	Age											
0	RamiNaidu	31											
1	Bhaksar	25											
2	Manikanta	23											

### Create a indexes Dataframe using arrays

Python code demonstrate creating pandas DataFrame with indexed by DataFrame using arrays.

#### Example

<pre>import pandas as pd data={'Name':['Raminaidu','Bhaskar','Manikanta'], 'marks':[99, 98, 95]} df=pd.DataFrame(data,index=['rank1','rank2', 'rank3']) print(df)</pre>	<table><tr><th colspan="3">Output</th></tr><tr><th></th><th>Name</th><th>marks</th></tr><tr><td>rank1</td><td>Rami naidu</td><td>99</td></tr><tr><td>rank2</td><td>Bhaskar</td><td>98</td></tr><tr><td>rank3</td><td>Manikanta</td><td>95</td></tr></table>	Output				Name	marks	rank1	Rami naidu	99	rank2	Bhaskar	98	rank3	Manikanta	95
Output																
	Name	marks														
rank1	Rami naidu	99														
rank2	Bhaskar	98														
rank3	Manikanta	95														

### Create Dataframe from list of dicts

We can pass the lists of dictionaries as input data to create the Pandas dataframe. The column names are taken as keys by default.

#### Examples

```
import pandas as pd
data = [{'A': 10, 'B': 20, 'C':30}, {'x':100, 'y': 200, 'z': 300}]
df = pd.DataFrame(data)
print(df)
```

---

Output

	A	B	C	x	y	z
0	10.0	20.0	30.0	NaN	NaN	NaN
1	NaN	NaN	NaN	100.0	200.0	300.0

### Create Dataframe using the zip() function

The zip() function is used to merge the two lists.

#### Example

<pre>import pandas as pd Name = ['Rami Naidu', 'Bhaskar', 'Manikanta'] Marks = [95, 63, 54] list_tuples = list(zip(Name, Marks)) print(list_tuples) dframe = pd.DataFrame(list_tuples, columns = ['Name', 'Marks']) print(dframe)</pre>	<p>Output</p> <pre>[('Rami Naidu', 95), ('Bhaskar', 63), ('Manikanta', 54)]</pre> <table><tr><th></th><th>Name</th><th>Marks</th></tr><tr><td>0</td><td>Rami Naidu</td><td>95</td></tr><tr><td>1</td><td>Bhaskar</td><td>63</td></tr><tr><td>2</td><td>Manikanta</td><td>54</td></tr></table>		Name	Marks	0	Rami Naidu	95	1	Bhaskar	63	2	Manikanta	54
	Name	Marks											
0	Rami Naidu	95											
1	Bhaskar	63											
2	Manikanta	54											

---

## Session #2

---

### Learning Objective

#### To implement pandas' data series

1. Write a Pandas program to create and display a one-dimensional array-like object containing an array of data using Pandas module ?
2. Write a Pandas program to convert a Panda module Series to Python list and it's type.
3. Write a Pandas program to add, subtract, multiple and divide two Pandas Series
4. Write a Pandas program to convert a NumPy array to a Pandas series.

Sample Series:

NumPy array: [10 20 30 40 50]

Converted Pandas series:

0 10

1 20

2 30

3 40

4 50

dtype: int64.

### Learning Context

A Pandas Series is like a column in a table. It is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.).

We can easily convert the list, tuple, and dictionary into series using "**series**" method.

The row labels of series are called the index. A Series cannot contain multiple columns.

It has the following parameter:

- data: It can be any list, dictionary, or scalar value.
- index: The value of the index should be unique and hashable. It must be of the same length as data. If we do not pass any index, default np.arange(n) will be used.
- dtype: It refers to the data type of series.
- copy: It is used for copying the data.

### Creating a Series:

We can create a Series in two ways:

1. Create an empty Series
2. Create a Series using inputs.

### Create an Empty Series:

We can easily create an empty series in Pandas which means it will not have any value.

The syntax that is used for creating an Empty Series:

```
<series object> = pandas.Series()
```

### Example

```
import pandas as pd
x = pd.Series()
print(x)
```

Output

```
Series([], dtype: float64)
```

### Creating a Series using inputs:

We can create Series by using various inputs:

- Array
- Dict
- Scalar value

### Creating Series from Array:

Before creating a Series, firstly, we have to import the numpy module and then use array() function in the program. If the data is ndarray, then the passed index must be of the same length.

If we do not pass an index, then by default index of range(n) is being passed where n defines the length of an array, i.e., [0,1,2,...,range(len(array))-1].

### Example

import pandas as pd	0	P
import numpy as np	1	a
info = np.array(['P','a','n','d','a','s'])	2	n
a = pd.Series(info)	3	d
print(a)	4	a
	5	s
	dtype: object	



### Create a Series from dict

We can also create a Series from dict. If the dictionary object is being passed as an input and the index is not specified, then the dictionary keys are taken in a sorted order to construct the index.

If index is passed, then values correspond to a particular label in the index will be extracted from the dictionary.

#### Example

import pandas as pd import numpy as np info = {'x' : 0., 'y' : 1., 'z' : 2.} a = pd.Series(info) print (a)	output x    0.0 y    1.0 z    2.0 dtype: float64
--	--

### Create a Series using Scalar:

If we take the scalar values, then the index must be provided. The scalar value will be repeated for matching the length of the index.

#### Example

#import pandas library import pandas as pd import numpy as np x = pd.Series(4, index=[0, 1, 2, 3]) print (x)	output 0    4 1    4 2    4 3    4 dtype: int64
--	--

## Materials & Resources

#### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

1. Write a Pandas program to create and display a one-dimensional array-like object containing an array of data using Pandas module ?

<pre>import pandas as pd ds = pd.Series([2, 4, 6, 8, 10]) print(ds)</pre>	<b>output</b> 0    2 1    4 2    6 3    8 4   10 dtype: int64
---	---

2. Write a Pandas program to convert a Panda module Series to Python list and it's type.

<pre>import pandas as pd ds = pd.Series([2, 4, 6, 8, 10]) print("Pandas Series and type") print(ds) print(type(ds)) print("Convert Pandas Series to Python list") print(ds.tolist()) print(type(ds.tolist()))</pre>	<b>output</b> Pandas Series and type 0    2 1    4 2    6 3    8 4   10 dtype: int64 <class 'pandas.core.series.Series'> Convert Pandas Series to Python list [2, 4, 6, 8, 10] <class 'list'>
---	---

3. Write a Pandas program to add, subtract, multiple and divide two Pandas Series.

<pre>import pandas as pd ds1 = pd.Series([2, 4, 6, 8, 10]) ds2 = pd.Series([1, 3, 5, 7, 9]) ds = ds1 + ds2 print("Add two Series:") print(ds) print("Subtract two Series:") ds = ds1 - ds2</pre>	<b>output</b> Add two Series: 0    3 1    7 2   11 3   15 4   19 dtype: int64
--	--

<pre> print(ds) print("Multiply two Series:") ds = ds1 * ds2 print(ds) print("Divide Series1 by Series2:") ds = ds1 / ds2 print(ds) </pre>	<pre> Subtract two Series: 0    1 1    1 2    1 3    1 4    1 dtype: int64 Multiply two Series: 0     2 1    12 2    30 3    56 4    90 dtype: int64 Divide Series1 by Series2: 0    2.000000 1    1.333333 2    1.200000 3    1.142857 4    1.111111 dtype: float64 </pre>
--	---

#### 4. Write a Pandas program to convert a NumPy array to a Pandas series.

Sample Series:

NumPy array:

[10 20 30 40 50]

Converted Pandas series:

0 10

1 20

2 30

3 40

4 50

dtype: int64

<pre> import numpy as np import pandas as pd </pre>	<pre> <b>output</b> NumPy array: </pre>
---	---

<pre>np_array = np.array([10, 20, 30, 40, 50]) print("NumPy array:") print(np_array) new_series = pd.Series(np_array) print("Converted Pandas series:") print(new_series)</pre>	<pre>[10 20 30 40 50] Converted Pandas series: 0    10 1    20 2    30 3    40 4    50 dtype: int32</pre>
---	---

## Session #3

### Learning Objective

To implement Pandas Data Frames:

Consider Sample Python dictionary data and list labels:

```
exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael',  
'Matthew', 'Laura', 'Kevin', 'Jonas'],  
'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],  
'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],  
'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}  
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

1. Write a Pandas program to create and display a Data Frame from a specified dictionary data which has the index labels.
2. Write a Pandas program to change the name 'James' to 'Suresh' in name column of the Data Frame.
3. Write a Pandas program to insert a new column in existing Data Frame.
4. Write a Pandas program to get list from Data Frame column headers.
5. Write a Pandas program to get list from Data Frame column headers.

### Learning Context

A Pandas Series is like a column in a table. It is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.).

We can easily convert the list, tuple, and dictionary into series using "**series**" method.

The row labels of series are called the index. A Series cannot contain multiple columns.

It has the following parameter:

- data: It can be any list, dictionary, or scalar value.
- index: The value of the index should be unique and hashable. It must be of the same length as data. If we do not pass any index, default np.arange(n) will be used.
- dtype: It refers to the data type of series.
- copy: It is used for copying the data.

#### Creating a Series:

We can create a Series in two ways:

3. Create an empty Series

4. Create a Series using inputs.

## Materials & Resources

### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

1. Write a Pandas program to create and display a DataFrame from a specified dictionary data which has the index labels.

```
import pandas as pd
import numpy as np
exam_data = {'name': ['Anastasia',
'Dima', 'Katherine', 'James', 'Emily',
'Michael', 'Matthew', 'Laura', 'Kevin',
'Jonas'], 'score': [12.5, 9, 16.5, np.nan, 9,
20, 14.5, np.nan, 8, 19], 'attempts': [1, 3, 2,
3, 2, 3, 1, 1, 2, 1], 'qualify': ['yes', 'no',
'yes', 'no', 'no', 'yes', 'yes', 'no', 'no',
'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j']
df = pd.DataFrame(exam_data ,
index=labels)
print(df)
```

### Output

	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
d	James	NaN	3	no
e	Emily	9.0	2	no
f	Michael	20.0	3	yes
g	Matthew	14.5	1	yes
h	Laura	NaN	1	no
i	Kevin	8.0	2	no
j	Jonas	19.0	1	yes

2. Write a Pandas program to change the name 'James' to 'Suresh' in name column of the DataFrame.

```
import pandas as pd
import numpy as np
exam_data = {'name': ['Anastasia',
'Dima', 'Katherine', 'James', 'Emily',
'Michael', 'Matthew', 'Laura', 'Kevin',
'Jonas'], 'score': [12.5, 9, 16.5, np.nan,
9, 20, 14.5, np.nan, 8, 19], 'attempts':
[1, 3, 2, 3, 2, 3, 1, 1, 2, 1], 'qualify':
['yes', 'no', 'yes', 'no', 'no', 'yes',
'yes', 'no', 'no', 'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g',
'h', 'i', 'j']
df = pd.DataFrame(exam_data ,
index=labels)
print("Original rows:")
print(df)
print("\nChange the name 'James' to
'Suresh':")
df['name'] =
df['name'].replace('James', 'Suresh')
print(df)
```

**Output**

Original rows:

	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
d	James	NaN	3	no
e	Emily	9.0	2	no
f	Michael	20.0	3	yes
g	Matthew	14.5	1	yes
h	Laura	NaN	1	no
i	Kevin	8.0	2	no
j	Jonas	19.0	1	yes

Change the name 'James' to 'Suresh':

	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
d	Suresh	NaN	3	no
e	Emily	9.0	2	no
f	Michael	20.0	3	yes
g	Matthew	14.5	1	yes
h	Laura	NaN	1	no
i	Kevin	8.0	2	no
j	Jonas	19.0	1	yes

3. Write a Pandas program to insert a new column in existing DataFrame ?

<pre>import pandas as pd import numpy as np exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'], 'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19], 'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1], 'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}</pre>	<div>Output</div> <div>Original rows:</div> <table><thead><tr><th></th><th>name</th><th>score</th><th>attempts</th><th>qualify</th></tr></thead><tbody><tr><td>a</td><td>Anastasia</td><td>12.5</td><td>1</td><td>yes</td></tr><tr><td>b</td><td>Dima</td><td>9.0</td><td>3</td><td>no</td></tr><tr><td>c</td><td>Katherine</td><td>16.5</td><td>2</td><td>yes</td></tr><tr><td>d</td><td>James</td><td>NaN</td><td>3</td><td>no</td></tr><tr><td>e</td><td>Emily</td><td>9.0</td><td>2</td><td>no</td></tr><tr><td>f</td><td>Michael</td><td>20.0</td><td>3</td><td>yes</td></tr><tr><td>g</td><td>Matthew</td><td>14.5</td><td>1</td><td>yes</td></tr><tr><td>h</td><td>Laura</td><td>NaN</td><td>1</td><td>no</td></tr><tr><td>i</td><td>Kevin</td><td>8.0</td><td>2</td><td>no</td></tr><tr><td>j</td><td>Jonas</td><td>19.0</td><td>1</td><td>yes</td></tr></tbody></table> <div>New DataFrame after inserting the 'color' column</div> <table><thead><tr><th></th><th>name</th><th>score</th><th>attempts</th><th>qualify</th><th>color</th></tr></thead></table>		name	score	attempts	qualify	a	Anastasia	12.5	1	yes	b	Dima	9.0	3	no	c	Katherine	16.5	2	yes	d	James	NaN	3	no	e	Emily	9.0	2	no	f	Michael	20.0	3	yes	g	Matthew	14.5	1	yes	h	Laura	NaN	1	no	i	Kevin	8.0	2	no	j	Jonas	19.0	1	yes		name	score	attempts	qualify	color
	name	score	attempts	qualify																																																										
a	Anastasia	12.5	1	yes																																																										
b	Dima	9.0	3	no																																																										
c	Katherine	16.5	2	yes																																																										
d	James	NaN	3	no																																																										
e	Emily	9.0	2	no																																																										
f	Michael	20.0	3	yes																																																										
g	Matthew	14.5	1	yes																																																										
h	Laura	NaN	1	no																																																										
i	Kevin	8.0	2	no																																																										
j	Jonas	19.0	1	yes																																																										
	name	score	attempts	qualify	color																																																									

<pre> labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'] df = pd.DataFrame(exam_data , index=labels) print("Original rows:") print(df) color = ['Red','Blue','Orange','Red','Wh ite','White','Blue','Green','Gree n','Red'] df['color'] = color print("\nNew DataFrame after inserting the 'color' column") print(df) </pre>	<pre> a  Anastasia  12.5      1    yes    Red b      Dima    9.0      3     no    Blue c  Katherine  16.5      2    yes  Orange d      James   NaN      3     no    Red e      Emily    9.0      2     no   White f   Michael   20.0      3    yes   White g   Matthew   14.5      1    yes    Blue h      Laura   NaN      1     no   Green i      Kevin    8.0      2     no   Green j      Jonas   19.0      1    yes    Red </pre>
---	--

4. Write a Pandas program to get list from DataFrame column headers.

<pre> import pandas as pd import numpy as np exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'], 'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19], 'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1], 'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']} labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'] df = pd.DataFrame(exam_data , index=labels) print(list(df.columns.values)) print(list(df.index.values)) </pre>	<pre> Output ['name', 'score', 'attempts', 'qualify'] ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'] </pre>
---	---



## Session #4

### Learning Objective

To implement Pandas Index.

1. Write a Pandas program to display the default index and set a column as an Index in a given data frame.
2. Write a Pandas program to create an index label by using 64-bit integers, using floating-point numbers in a given data frame.

### Learning Context

Indexing in pandas means simply selecting particular rows and columns of data from a Data Frame. Indexing could mean selecting all the rows and some of the columns, some of the rows and all of the columns, or some of each of the rows and columns. Indexing can also be known as **Subset Selection**.

#### **pandas.Index**

```
class pandas.Index(data=None, dtype=None, copy=False, name=None, tupleize_cols=True)  
[source]
```

Immutable sequence used for indexing and alignment. The basic object storing axis labels for all pandas objects.

#### **Parameters:**

Data : array-like (1-dimensional)

Dtype : NumPy dtype (default: object)

If dtype is None, we find the dtype that best fits the data. If an actual dtype is provided, we coerce to that dtype if it's safe. Otherwise, an error will be raised.

Copy : bool - Make a copy of input ndarray.

Name : object - Name to be stored in the index.

tupleize\_cols : bool (default: True)

When True, attempt to create a MultiIndex if possible.

## Materials & Resources

### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd

Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

1. Write a Pandas program to display the default index and set a column as an Index in a given dataframe.

```
import pandas as pd
df = pd.DataFrame({'school_code': ['s001', 's002', 's003', 's001', 's002', 's004'],
                  'class': ['V', 'V', 'VI', 'VI', 'V', 'VI'], 'name': ['Alberto Franco', 'Gino
Mcneill', 'Ryan Parkes', 'Eesha Hinton', 'Gino Mcneill', 'David
Parkes'], 'date_Of_Birth': ['15/05/2002', '17/05/2002', '16/02/1999',
'25/09/1998', '11/05/2002', '15/09/1997'], 'weight': [35, 32, 33, 30, 31, 32],
                  'address': ['street1', 'street2', 'street3', 'street1', 'street2', 'street4'],
                  't_id': ['t1', 't2', 't3', 't4', 't5', 't6']})
print("Default Index:")
print(df.head(10))
print("\nschool_code as new Index:")
df1 = df.set_index('school_code')
print(df1)
print("\nt_id as new Index:")
df2 = df.set_index('t_id')
print(df2)
```

2. Write a Pandas program to create an index labels by using 64-bit integers, using floating-point numbers in a given dataframe.

Test Data:

0	s001	V	Alberto Franco	15/05/2002	35	street1	t1
1	s002	V	Gino Mcneill	17/05/2002	32	street2	t2
2	s003	VI	Ryan Parkes	16/02/1999	33	street3	t3
3	s001	VI	Eesha Hinton	25/09/1998	30	street1	t4
4	s002	V	Gino Mcneill	11/05/2002	31	street2	t5
5	s004	VI	David Parkes	15/09/1997	32	street4	t6

```
import pandas as pd
print("Create an Int64Index:")
df_i64 = pd.DataFrame({'school_code':
['s001','s002','s003','s001','s002','s004'], 'class': ['V', 'V', 'VI', 'VI', 'V',
'VI'], 'name': ['Alberto Franco', 'Gino Mcneill', 'Ryan Parkes', 'Eesha Hinton',
'Gino Mcneill', 'David Parkes'], 'date_Of_Birth':
['15/05/2002', '17/05/2002', '16/02/1999',
'25/09/1998', '11/05/2002', '15/09/1997'], 'weight': [35, 32, 33, 30, 31,
32], 'address': ['street1', 'street2', 'street3', 'street1', 'street2', 'street4']},
index=[1, 2, 3, 4, 5, 6])
print(df_i64)
print("\nView the Index:")
print(df_i64.index)
print("\nFloating-point labels using Float64Index:")
df_f64 = pd.DataFrame({'school_code':
['s001','s002','s003','s001','s002','s004'], 'class': ['V', 'V', 'VI', 'VI', 'V',
'VI'], 'name': ['Alberto Franco', 'Gino Mcneill', 'Ryan Parkes', 'Eesha Hinton',
'Gino Mcneill', 'David Parkes'], 'date_Of_Birth ': ['15/05/2002', '17/05/2002',
'16/02/1999', '25/09/1998', '11/05/2002', '15/09/1997'], 'weight': [35, 32, 33,
30, 31, 32], 'address': ['street1', 'street2', 'street3', 'street1', 'street2',
'street4']}, index=[.1, .2, .3, .4, .5, .6])
print(df_f64)
print("\nView the Index:")
print(df_f64.index)
```

## Session #5

### Learning Objective

To implement Pandas String and Regular Expressions:

### Learning Context

#### String Methods in pandas

The Pandas library is very useful for the manipulation of strings as it provides us with various handy string methods. It saves time and makes our program efficient.

	Method	Description
1.	upper()	Converts a string into uppercase
2.	lower()	Converts a string into lowercase
3.	isupper()	Checks whether the character is uppercase or not
4.	islower()	Checks whether the character is lowercase or not
5.	len()	Identifies the length of the string.
6.	startswith()	Returns true if the element starts with the pattern
7.	split()	Splits the string at a particular index or character
8.	find()	Returns the index at where the given string is found
9.	strip()	Strips whitespaces from each string from both sides.
10.	replace()	Replaces a part of the string with another one.

#### Regular expressions in pandas

There are several pandas methods which accept the regex in pandas to find the pattern in a String within a Series or Dataframe object. These methods works on the same line as Python's re module. It's really helpful if you want to find the names starting with a particular character or search for a pattern within a dataframe column or extract the dates from the text.

Here are the pandas' functions that accept regular expression

Methods	Description
count()	Count occurrences of pattern in each string of the Series/Index
replace()	Replace the search string or pattern with the given value
contains()	Test if pattern or regex is contained within a string of a Series or Index. Calls re.search() and returns a boolean
extract()	Extract capture groups in the regex pat as columns in a DataFrame and returns the captured groups
findall()	Find all occurrences of pattern or regular expression in the

	Series/Index. Equivalent to applying re.findall() on all elements
match()	Determine if each string matches a regular expression. Calls re.match() and returns a boolean
split()	Equivalent to str.split() and Accepts String or regular expression to split on
rsplit()	Equivalent to str.rsplit() and Splits the string in the Series/Index from the end

## Materials & Resources

### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd

Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

1. Write a Pandas program to convert all the string values to upper, lower cases in a given pandas series. Also find the length of the string values ?

```
import pandas as pd
import numpy as np
s = pd.Series(['R', 'A', 'M', 'Aaba', 'Baca', np.nan, 'CABA', None, 'bird', 'horse', 'dog'])
print("Original series:")
print(s)
print("\nConvert all string values of the said Series to upper case:")
print(s.str.upper())
print("\nConvert all string values of the said Series to lower case:")
print(s.str.lower())
print("\nLength of the string values of the said Series:")
print(s.str.len())
```

2. Write a Pandas program to remove whitespaces, left sided whitespaces and right sided whitespaces of the string values of a given pandas series.



```
import pandas as pd
color1 = pd.Index([' Green', 'Black ', ' Red ', 'White', ' Pink '])
print("Original series:")
print(color1)
print("\nRemove whitespace")
print(color1.str.strip())
print("\nRemove left sided whitespace")
print(color1.str.lstrip())
print("\nRemove Right sided whitespace")
print(color1.str.rstrip())
```

3. Write a Pandas program to count of occurrence of a specified substring in a DataFrame column?

```
import pandas as pd
df = pd.DataFrame({'name_code': ['c001', 'c002', 'c022', 'c2002', 'c2222'],
                  'date_of_birth': ['12/05/2002', '16/02/1999', '25/09/1998', '12/02/2022', '15/09/1997'],
                  'age': [18.5, 21.2, 22.5, 22, 23]})
print("Original DataFrame:")
print(df)
print("\nCount occurrence of 2 in date_of_birth column:")
df['count'] = list(map(lambda x: x.count("2"), df['name_code']))
print(df)
```

4. Write a Pandas program to swap the cases of a specified character column in a given DataFrame.

```
import pandas as pd
df = pd.DataFrame({'company_code': ['Abcd', 'EFGF', 'zefsalf', 'sdfslew', 'zekfsdf'],
                  'date_of_sale': ['12/05/2002', '16/02/1999', '25/09/1998', '12/02/2022', '15/09/1997'],
                  'sale_amount': [12348.5, 233331.2, 22.5, 2566552.0, 23.0]})
print("Original DataFrame:")
print(df)
print("\nSwapp cases in comapny_code:")
```



```
df['swapped_company_code'] = list(map(lambda x: x.swapcase(), df['company_code']))  
print(df)
```

## Session #6

### Learning Objective

Pandas Joining and merging Data Frame

### Learning Context

Pandas provides various facilities for easily combining together Series or DataFrame with various kinds of set logic for the indexes and relational algebra functionality in the case of join / merge-type operations.

In Dataframe `df.merge()`, `df.join()`, and `df.concat()` methods help in joining, merging and concating different dataframe.

In addition, pandas also provide utilities to compare two Series or DataFrame and summarize their differences.

**The `concat()`** function (in the main pandas namespace) does all of the heavy lifting of performing concatenation operations along an axis while performing optional set logic (union or intersection) of the indexes (if any) on the other axes.

**The `merge()`** method updates the content of two DataFrame by merging them together, using the specified method(s).

Use the parameters to control which values to keep and which to replace.

While `merge()` is a **module function**, `.join()` is an instance method that lives on your DataFrame. This enables you to specify only one DataFrame, which will join the DataFrame you call `.join()` on.



## Materials & Resources

### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd

Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

1. Write a Pandas program to join the two given data frames along rows and assign all data?

```
import pandas as pd
student_data1 = pd.DataFrame({'student_id': ['S1', 'S2', 'S3', 'S4', 'S5'], 'name':
['Danniella Fenton', 'Ryder Storey', 'Bryce Jensen', 'Ed Bernal', 'Kwame Morin'],
'marks': [200, 210, 190, 222, 199]})
student_data2 = pd.DataFrame({'student_id': ['S4', 'S5', 'S6', 'S7', 'S8'], 'name':
['Scarlette Fisher', 'Carla Williamson', 'Dante Morse', 'Kaiser William', 'Madeeha
Preston'], 'marks': [201, 200, 198, 219, 201]})
print("Original DataFrames:")
print(student_data1)
print("-----")
print(student_data2)
print("\nJoin the said two data frames along rows:")
result_data = pd.concat([student_data1, student_data2])
print(result_data)
```

2. Write a Pandas program to append a list of dictionaries or series to an existing Data Frame and display the combined data?

```
import pandas as pd
student_data1 = pd.DataFrame({'student_id': ['S1', 'S2', 'S3', 'S4', 'S5'], 'name':
['Danniella Fenton', 'Ryder Storey', 'Bryce Jensen', 'Ed Bernal', 'Kwame Morin'],
'marks': [200, 210, 190, 222, 199]})
s6 = pd.Series(['S6', 'Scarlette Fisher', 205], index=['student_id', 'name', 'marks'])
dicts = [{'student_id': 'S6', 'name': 'Scarlette Fisher', 'marks': 203},
        {'student_id': 'S7', 'name': 'Bryce Jensen', 'marks': 207}]
print("Original DataFrames:")
print(student_data1)
print("\nDictionary:")
print(s6)
combined_data = student_data1.append(dicts, ignore_index=True, sort=False)
print("\nCombined Data:")
print(combined_data)
```

3. Write a Pandas program to join the two dataframes with matching records from both sides where available.

```
import pandas as pd
student_data1 = pd.DataFrame({'student_id': ['S1', 'S2', 'S3', 'S4', 'S5'],
'name': ['Danniella Fenton', 'Ryder Storey', 'Bryce Jensen', 'Ed Bernal', 'Kwame
Morin'], 'marks': [200, 210, 190, 222, 199]})
student_data2 = pd.DataFrame({'student_id': ['S4', 'S5', 'S6', 'S7', 'S8'], 'name':
['ScarletteFisher', 'Carla Williamson', 'Dante Morse', 'Kaiser William', 'Madeeha
Preston'], 'marks': [201, 200, 198, 219, 201]})
print("Original DataFrames:")
print(student_data1)
print(student_data2)
merged_data = pd.merge(student_data1, student_data2, on='student_id', how='outer')
print("Merged data (outer join):")
print(merged_data)
```

## Session #7

### Learning Objective

To implement Pandas Time Series

### Learning Context

Pandas contains extensive capabilities and features for working with time series data for all domains. Using the NumPy datetime64 and timedelta64 dtypes, pandas has consolidated a large number of features from other Python libraries like scikits.timeseries as well as created a tremendous amount of new functionality for manipulating time series data.

Pandas captures 4 general time related concepts:

1. **Date times:** A specific date and time with timezone support. Similar to datetime.datetime from the standard library.
2. **Time deltas:** An absolute time duration. Similar to datetime.timedelta from the standard library.
3. **Time spans:** A span of time defined by a point in time and its associated frequency.
4. **Date offsets:** A relative time duration that respects calendar arithmetic. Similar to dateutil.relativedelta.relativedelta from the dateutil package.

### Materials & Resources

Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd

Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

1. Write a Pandas program to create
  - a) Datetime object for Jan 15 2012.
  - b) Specific date and time of 9:20 pm.
  - c) Local date and time.
  - d) A date without time.
  - e) Current date.
  - f) Time from a datetime.
  - g) Current local time

```
import datetime
from datetime import datetime
print("Datetime object for Jan 11 2012:")
print(datetime(2012, 1, 11))
print("\nSpecific date and time of 9:20 pm")
print(datetime(2011, 1, 11, 21, 20))
print("\nLocal date and time:")
print(datetime.now())
print("\nA date without time: ")
print(datetime.date(datetime(2012, 5, 22)))
print("\nCurrent date:")
print(datetime.now().date())
print("\nTime from a datetime:")
print(datetime.time(datetime(2012, 12, 15, 18, 12)))
print("\nCurrent local time:")
print(datetime.now().time())
```

2. Write a Pandas program to create a date from a given year, month, day and another date from a given string formats?

```
from datetime import datetime
from dateutil import parser
date1 = datetime(year=2020, month=12, day=25)
print("Date from a given year, month, day:")
print(date1)
date2 = parser.parse("1st of January, 2021")
print("\nDate from a given string formats:")
print(date2)
```

3. Write a Pandas program to create a time-series with two index labels and random values. Also print the type of the index.

```
import pandas as pd
import numpy as np
import datetime
from datetime import datetime, date
dates = [datetime(2011, 9, 1), datetime(2011, 9, 2)]
print("Time-series with two index labels:")
time_series = pd.Series(np.random.randn(2), dates)
print(time_series)
print("\nType of the index:")
print(type(time_series.index))
```

---

## Session #8

---

### Learning Objective

To implement Pandas Grouping and Aggregate

### Learning Context

Grouping and aggregating will help to achieve data analysis easily using various functions. These methods will help us to the group and summarize our data and make complex analysis comparatively easy.

Pandas' Groupby operation is a powerful and versatile function in Python. It allows you to split your data into separate groups to perform computations for better analysis.

An essential piece of analysis of large data is efficient summarization: computing aggregations like `sum()`, `mean()`, `median()`, `min()`, and `max()`, in which a single number gives insight into the nature of a potentially large dataset.

### Materials & Resources

Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

Consider dataset:

school	class	name	date_Of_Birth	age	height	weight	address
S1	s001	V	Alberto Franco	15/05/2002	12	173	35 street1
S2	s002	V	Gino Mcneill	17/05/2002	12	192	32 street2
S3	s003	VI	Ryan Parkes	16/02/1999	13	186	33 street3
S4	s001	VI	Eesha Hinton	25/09/1998	13	167	30 street1
S5	s002	V	Gino Mcneill	11/05/2002	14	151	31 street2
S6	s004	VI	David Parkes	15/09/1997	12	159	32 street4

1. Write a Pandas program to split the following dataframe into groups based on school code. Also check the type of GroupBy object.

```
import pandas as pd
pd.set_option('display.max_rows', None)
#pd.set_option('display.max_columns', None)
student_data = pd.DataFrame({'school_code':
['s001','s002','s003','s001','s002','s004'],
    'class': ['V', 'V', 'VI', 'VI', 'V', 'VI'],
    'name': ['Alberto Franco','Gino Mcneill','Ryan Parkes', 'Eesha Hinton', 'Gino Mcneill',
'David Parkes'],
    'date_Of_Birth ':
['15/05/2002','17/05/2002','16/02/1999','25/09/1998','11/05/2002','15/09/1997'],
    'age': [12, 12, 13, 13, 14, 12],
    'height': [173, 192, 186, 167, 151, 159],
    'weight': [35, 32, 33, 30, 31, 32],
    'address': ['street1', 'street2', 'street3', 'street1', 'street2', 'street4']},
    index=['S1', 'S2', 'S3', 'S4', 'S5', 'S6'])
print("Original DataFrame:")
print(student_data)
print('\nSplit the said data on school_code wise:')
result = student_data.groupby(['school_code'])
for name,group in result:
    print("\nGroup:")
    print(name)
    print(group)
print("\nType of the object:")
print(type(result))
```

2. Write a Pandas program to split the following dataframe by school code and get mean, min, and max value of age for each school.

```
import pandas as pd
pd.set_option('display.max_rows', None)
#pd.set_option('display.max_columns', None)
student_data = pd.DataFrame({'school_code':
['s001','s002','s003','s001','s002','s004'],'class': ['V', 'V', 'VI', 'VI', 'V', 'VI'],
    'name': ['Alberto Franco','Gino Mcneill','Ryan Parkes', 'Eesha Hinton', 'Gino Mcneill',
'David Parkes'], 'date_Of_Birth ': ['15/05/2002', '17/05/2002', '16/02/1999',
'25/09/1998', '11/05/2002', '15/09/1997'],
    'age': [12, 12, 13, 13, 14, 12], 'height': [173, 192, 186, 167, 151, 159],
    'weight': [35, 32, 33, 30, 31, 32], 'address': ['street1', 'street2', 'street3',
'street1', 'street2', 'street4']}, index=['S1', 'S2', 'S3', 'S4', 'S5', 'S6'])
print("Original DataFrame:")
print(student_data)
print('\nMean, min, and max value of age for each value of the school:')
grouped_single = student_data.groupby('school_code').agg({'age': ['mean', 'min',
'max']})
print(grouped_single)
```



## Session #9

### Learning Objective

Pandas Styling

### Learning Context

A pandas dataframe is a tabular structure comprising rows and columns. One prevalent environment for data-related tasks is Jupyter notebooks, which are web-based, platform-independent integrated development environments (IDEs). In Jupyter notebooks, the pandas style of the dataframe is achieved through the use of HTML tags and CSS for rendering.

#### Styling the DataFrame

To leverage all the pandas styling properties for the dataframe, employ the pandas styling accessor (assuming the dataframe is stored in the variable "df"):

**df.style**

This accessor helps in the modification of the styler object (df.style), which controls the display of the dataframe on the web. Let's look at some of the methods to style the dataframe.

#### Highlight Min-Max values

The dataframes can take a large number of values but when it is of a smaller size, then it makes sense to print out all the values of the dataframe.

**For highlighting maximum values:** Chain ".highlight\_max()" function to the styler object. Additionally, you can also specify the axis for which you want to highlight the values. (axis=1: Rows, axis=0: Columns - default).

**df.style.highlight\_max()**

**For highlighting minimum values:** Chain ".highlight\_min()" function to the styler object. Here also, you can specify the axis at which these values will be highlighted.

**df.style.highlight\_min()**

#### Highlight Null values

Every dataset has some or the other null/missing values. These values should be either removed or handled in such a way that it doesn't introduce any biasness. To highlight such values, you can chain the ".highlight\_null()" function to the styler object.

## Create Heatmap within dataframe

Heatmaps are used to represent values with the color shades. The higher is the color shade, the larger is the value present. These color shades represent the intensity of values as compared to other values. To plot such a mapping in the dataframe itself, there is no direct function but the "styler.background\_gradient()" workaround does the work.

`df.style.background_gradient()`

	A	B	C	D	E
0	1.000000	0.106884	nan	0.956563	0.068411
1	2.000000	1.068514	0.997183	-0.931548	0.730430
2	3.000000	-0.171214	-1.288691	1.061436	-0.040501
3	4.000000	-0.903721	-1.554133	nan	0.200526
4	5.000000	-0.747329	1.068081	-0.277024	0.086557
5	6.000000	-0.253221	-1.212041	0.277273	0.552219
6	7.000000	-0.467743	-1.427493	0.885805	2.360634
7	8.000000	-1.522006	-0.215945	0.190327	0.722256
8	9.000000	-0.870716	0.101749	0.555358	0.962261
9	10.000000	1.433499	-0.701818	-0.856952	-0.858158

## Table Properties

As mentioned earlier also, the dataframe presented in the Jupyter notebooks is a table rendered using HTML and CSS. The table properties can be controlled using the "set\_properties" method. This method is used to set one or more data-independent properties.

This means that the modifications are done purely based on visual appearance and no significance as such. This method takes in the properties to be set as a dictionary.

Example: Making table borders green with text color as purple.

```
df.style.set_properties(**{'border': '1.3px solid green',  
                           'color': 'magenta'})
```

	A	B	C	D	E
0	1.000000	0.106884	nan	0.956563	0.068411
1	2.000000	1.068514	0.997183	-0.931548	0.730430
2	3.000000	-0.171214	-1.288691	1.061436	-0.040501
3	4.000000	-0.903721	-1.554133	nan	0.200526
4	5.000000	-0.747329	1.068081	-0.277024	0.086557
5	6.000000	-0.253221	-1.212041	0.277273	0.552219
6	7.000000	-0.467743	-1.427493	0.885805	2.360634
7	8.000000	-1.522006	-0.215945	0.190327	0.722256
8	9.000000	-0.870716	0.101749	0.555358	0.962261
9	10.000000	1.433499	-0.701818	-0.856952	-0.858158

### Create Bar charts

Just as the heatmap, the bar charts can also be plotted within the dataframe itself. The bars are plotted in each cell depending upon the axis selected. By default, the axis=0 and the plot color are also fixed by pandas but it is configurable. To plot these bars, you simply need to chain the ".bar()" function to the styler object.

**df.style.bar()**

## Materials & Resources

### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

## Exercise

1. Create a dataframe of ten rows, four columns with random values. Write a Pandas program to highlight the negative numbers red and positive numbers black.

```
import pandas as pd
import numpy as np
np.random.seed(24)
df = pd.DataFrame({'A': np.linspace(1, 10, 10)})
df = pd.concat([df, pd.DataFrame(np.random.randn(10, 4),
                                columns=list('BCDE'))], axis=1)
print("Original array:")
print(df)
def color_negative_red(val):
    color = 'red' if val < 0 else 'black'
    return 'color: %s' % color
print("\nNegative numbers red and positive numbers black:")
df.style.applymap(color_negative_red)
```

Original array:

	A	B	C	D	E
0	1.0	1.329212	-0.770033	-0.316280	-0.990810
1	2.0	-1.070816	-1.438713	0.564417	0.295722
2	3.0	-1.626404	0.219565	0.678805	1.889273
3	4.0	0.961538	0.104011	-0.481165	0.850229
4	5.0	1.453425	1.057737	0.165562	0.515018
5	6.0	-1.336936	0.562861	1.392855	-0.063328
6	7.0	0.121668	1.207603	-0.002040	1.627796
7	8.0	0.354493	1.037528	-0.385684	0.519818
8	9.0	1.686583	-1.325963	1.428984	-2.089354
9	10.0	-0.129820	0.631523	-0.586538	0.290720

Negative numbers red and positive numbers black:

	A	B	C	D	E
0	1.000000	1.329212	-0.770033	-0.316280	-0.990810
1	2.000000	-1.070816	-1.438713	0.564417	0.295722
2	3.000000	-1.626404	0.219565	0.678805	1.889273
3	4.000000	0.961538	0.104011	-0.481165	0.850229
4	5.000000	1.453425	1.057737	0.165562	0.515018
5	6.000000	-1.336936	0.562861	1.392855	-0.063328
6	7.000000	0.121668	1.207603	-0.002040	1.627796

7	8.000000	0.354493	1.037528	-0.385684	0.519818
8	9.000000	1.686583	-1.325963	1.428984	-2.089354
9	10.000000	-0.129820	0.631523	-0.586538	0.290720

2. Create a dataframe of ten rows, four columns with random values. Write a Pandas program to highlight the maximum value in each column.

```
import pandas as pd
import numpy as np
np.random.seed(24)
df = pd.DataFrame({'A': np.linspace(1, 10, 10)})
df = pd.concat([df, pd.DataFrame(np.random.randn(10, 4),
columns=list('BCDE'))],axis=1)
df.iloc[0, 2] = np.nan
df.iloc[3, 3] = np.nan
df.iloc[4, 1] = np.nan
df.iloc[9, 4] = np.nan
print("Original array:")
print(df)
def highlight_max(s):
    #highlight the maximum in a Series green.
    is_max = s == s.max()
    return ['background-color: green' if v else '' for v in is_max]
print("\nHighlight the maximum value in each column:")
df.style.apply(highlight_max,subset=pd.IndexSlice[:, ['B', 'C', 'D', 'E']])
```

Original array:

	A	B	C	D	E
0	1.0	1.329212	NaN	-0.316280	-0.990810
1	2.0	-1.070816	-1.438713	0.564417	0.295722
2	3.0	-1.626404	0.219565	0.678805	1.889273
3	4.0	0.961538	0.104011	NaN	0.850229
4	5.0	NaN	1.057737	0.165562	0.515018
5	6.0	-1.336936	0.562861	1.392855	-0.063328
6	7.0	0.121668	1.207603	-0.002040	1.627796
7	8.0	0.354493	1.037528	-0.385684	0.519818
8	9.0	1.686583	-1.325963	1.428984	-2.089354

9 10.0 -0.129820 0.631523 -0.586538 NaN

Highlight the maximum value in each column:

Out[28]:

	A	B	C	D	E
0	1.000000	1.329212	nan	-0.316280	-0.990810
1	2.000000	-1.070816	-1.438713	0.564417	0.295722
2	3.000000	-1.626404	0.219565	0.678805	1.889273
3	4.000000	0.961538	0.104011	nan	0.850229
4	5.000000	nan	1.057737	0.165562	0.515018
5	6.000000	-1.336936	0.562861	1.392855	-0.063328
6	7.000000	0.121668	1.207603	-0.002040	1.627796
7	8.000000	0.354493	1.037528	-0.385684	0.519818
8	9.000000	1.686583	-1.325963	1.428984	-2.089354
9	10.000000	-0.129820	0.631523	-0.586538	nan

3. Create a dataframe of ten rows, four columns with random values. Write a Pandas program to highlight dataframe's specific columns

```
import pandas as pd
import numpy as np
np.random.seed(24)
df = pd.DataFrame({'A': np.linspace(1, 10, 10)})
df = pd.concat([df, pd.DataFrame(np.random.randn(10, 4), columns=list('BCDE'))],
               axis=1)
df.iloc[0, 2] = np.nan
df.iloc[3, 3] = np.nan
df.iloc[4, 1] = np.nan
df.iloc[9, 4] = np.nan
print("Original array:")
print(df)
def highlight_cols(s):
    color = 'grey'
```

```
return 'background-color: %s' % color
print("\nHighlight specific columns:")
df.style.applymap(highlight_cols, subset=pd.IndexSlice[:, ['B', 'C']])
```

Original array:

	A	B	C	D	E
0	1.0	1.329212	NaN	-0.316280	-0.990810
1	2.0	-1.070816	-1.438713	0.564417	0.295722
2	3.0	-1.626404	0.219565	0.678805	1.889273
3	4.0	0.961538	0.104011	NaN	0.850229
4	5.0	NaN	1.057737	0.165562	0.515018
5	6.0	-1.336936	0.562861	1.392855	-0.063328
6	7.0	0.121668	1.207603	-0.002040	1.627796
7	8.0	0.354493	1.037528	-0.385684	0.519818
8	9.0	1.686583	-1.325963	1.428984	-2.089354
9	10.0	-0.129820	0.631523	-0.586538	NaN

Highlight specific columns: **Out[29]:**

	A	B	C	D	E
0	1.000000	1.329212	nan	-0.316280	-0.990810
1	2.000000	-1.070816	-1.438713	0.564417	0.295722
2	3.000000	-1.626404	0.219565	0.678805	1.889273
3	4.000000	0.961538	0.104011	nan	0.850229
4	5.000000	nan	1.057737	0.165562	0.515018
5	6.000000	-1.336936	0.562861	1.392855	-0.063328
6	7.000000	0.121668	1.207603	-0.002040	1.627796
7	8.000000	0.354493	1.037528	-0.385684	0.519818
8	9.000000	1.686583	-1.325963	1.428984	-2.089354
9	10.000000	-0.129820	0.631523	-0.586538	nan

## Session #10

### Learning Objective

1. Write a Pandas program to import excel data into a Pandas dataframe.
2. Write a Pandas program to find the sum, mean, max, min value of a column of file.

### Materials & Resources

#### Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, O'Reilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

### Exercise

1. Write a Pandas program to import excel data into a Pandas dataframe.

```
import pandas as pd
import numpy as np
df = pd.read_excel('E:\coalpublic2013.xlsx')
df.tail(n=10)
```

#### Excel Data :

Year	MSHA ID	Mine_Name	Production	Labor_Hours
2013	103381	Tacoa Highwall Miner	56,004	22,392
2013	103404	Reid School Mine	28,807	8,447
2013	100759	North River #1 Underground Min	14,40,115	4,74,784
2013	103246	Bear Creek	87,587	29,193



2013	103451	Knight Mine	1,47,499	46,393
2013	103433	Crane Central Mine	69,339	47,195
2013	100329	Concord Mine	0	1,44,002
2013	100851	Oak Grove Mine	22,69,014	10,01,809
2013	102901	Shoal Creek Mine	0	12,396
2013	102901	Shoal Creek Mine	14,53,024	12,37,415
2013	103180	Sloan Mountain Mine	3,27,780	1,96,963
2013	103182	Fishtrap	1,75,058	87,314
2013	103285	Narley Mine	1,54,861	90,584
2013	103332	Powhatan Mine	1,40,521	61,394
2013	103375	Johnson Mine	580	1,900
2013	103419	Maxine-Pratt Mine	1,25,824	1,07,469
2013	103432	Skelton Creek	8,252	220
2013	103437	Black Warrior Mine No	11,45,924	70,926
2013	102976	Piney Woods Preparation Plant	0	14,828
2013	102976	Piney Woods Preparation Plant	0	23,193
2013	103380	Calera	0	12,621
2013	103380	Calera	0	1,402
2013	103422	Clark No 1 Mine	1,22,727	1,40,250
2013	103467	Helena Surface Mine	59,664	30,539
2013	101247	No 4 Mine	26,22,528	15,51,141
2013	101401	No 7 Mine	54,05,412	24,64,719
2013	103172	Searles Mine No.2, 3, 4, 5, 6	2,58,078	1,19,542
2013	103179	Fleetwood Mine No 1	75,937	63,745
2013	103303	Shannon Mine	3,17,491	1,64,388
2013	103323	Deerlick Mine	1,33,452	46,381
2013	103364	Brc Alabama No. 7 Llc	0	14,324
2013	103436	Swann's Crossing	1,37,511	77,190

2013	100347	Choctaw Mine	5,37,429	2,15,295
2013	101362	Manchester Mine	2,19,457	1,16,914
2013	102996	Jap Creek Mine	3,75,715	1,64,093
2013	103155	Corinth Prep Plant	0	27,996
2013	103155	Corinth Prep Plant	0	51,994
2013	103195	Mccollum/Sparks Branch Mine	71,910	17,411
2013	103342	Reese's Branch Mine	2,63,888	1,15,123
2013	103370	Crescent Valley Mine	2,860	621
2013	103372	Cane Creek Mine	66,258	32,401
2013	103376	Town Creek	2,99,167	1,76,499
2013	103389	Carbon Hill Mine	76,241	84,966
2013	103410	Coal Valley Mine	4,07,841	1,58,591
2013	103423	Dutton Hill Mine	37,275	9,162
2013	1519322	Ghm #25	25,054	3,108
2013	103321	Poplar Springs	1,89,370	76,366
2013	103358	Old Union	2,84,563	1,61,805
2013	5000030	Usibelli	16,31,584	2,86,079
2013	201195	Kayenta Mine	76,02,722	10,15,333

2. Write a Pandas program to find the sum, mean, max, min value of a column of file.

```
import pandas as pd
import numpy as np
df = pd.read_excel('E:\coalpublic2013.xlsx')
print("Sum: ",df["Production"].sum())
print("Mean: ",df["Production"].mean())
print("Maximum: ",df["Production"].max())
print("Minimum: ",df["Production"].min())
```

## Session #11

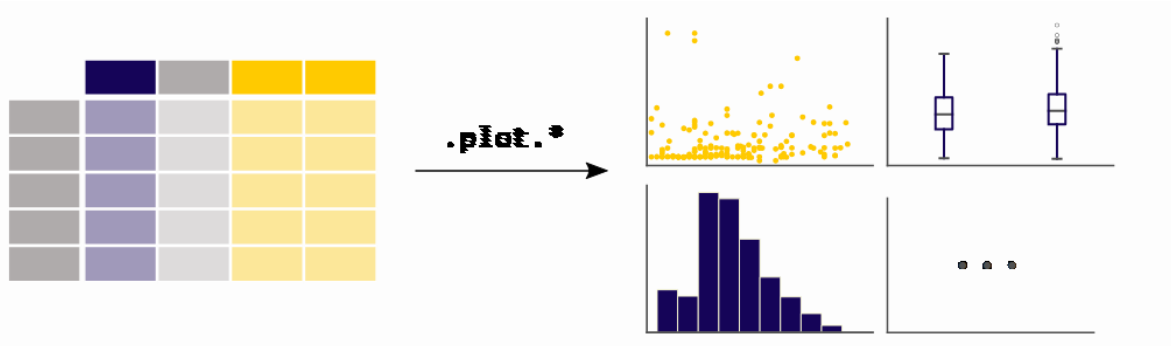
### Learning Objective

Plotting.

### Learning Context

Pandas uses the `plot()` method to create diagrams. We can use `Pyplot`, a submodule of the `Matplotlib` library to visualize the diagram on the screen.

Python has many popular plotting libraries that make visualization easy. Some of them are **Matplotlib**, **Seaborn**, and **Python Plotly**. It has great integration with `Matplotlib`. We can plot a `Dataframe` using **the `plot()` method**. But we need a `Dataframe` to plot. We can create a `Dataframe` by just passing a dictionary to the **`DataFrame()` method** of the `Pandas` library.



#### Scatter Plot

To get the scatterplot of a dataframe all we have to do is to just call the `plot()` method by specifying some parameters.

```
kind='scatter',x= 'some_column',y='some_colum',color='somecolor'
```

#### Histogram

Use the `kind` argument to specify that you want a histogram:

```
kind = 'hist'
```

A histogram needs only one column.

A histogram shows us the frequency of each interval, e.g. how many workouts lasted between 50 and 60 minutes?

#### Bar Plot

Similarly, we have to specify some parameters for `plot()` method to get the bar plot.

```
kind='bar',x= 'some_column',y='some_colum',color='somecolor'
```

## Line Plot

The line plot of a single column is not always useful, to get more insights we have to plot multiple columns on the same graph. To do so we have to reuse the axes.

```
kind='line',x= 'some_column',y='some_colum',color='somecolor',ax='someaxes'
```

## Box Plot

Box plot is majorly used to identify outliers, we can information like median, maximum, minimum, quartiles and so on.

## Materials & Resources

Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1stEdition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language,Second Edition, Additson Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus VilhelmPersson and Luiz Felipe Martins, Mastering Python Data Analysis, PacktPublishing Ltd

## Exercise

1. Write a Pandas program to create a horizontal stacked bar plot of opening, closing stock prices of any stock dataset between two specific dates.

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("alphabet_stock_data.csv")
start_date = pd.to_datetime('2020-4-1')
end_date = pd.to_datetime('2020-4-30')
df['Date'] = pd.to_datetime(df['Date'])
new_df = (df['Date']>= start_date) & (df['Date']<= end_date)
df1 = df.loc[new_df]
df2 = df1[['Date', 'Open', 'Close']]
df3 = df2.set_index('Date')
```

```
plt.figure(figsize=(20,20))
df3.plot.barh(stacked=True)
plt.suptitle('Opening/Closing stock prices Alphabet Inc.,\n01-04-2020 to 30-04-2020', fontsize=12, color='black')
plt.show()
```

### [Download alphabet\\_stock\\_data.csv](#)

2. Write a Pandas program to create a histograms plot of opening, closing, high, low stock prices of stock dataset between two specific dates?

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("alphabet_stock_data.csv")
start_date = pd.to_datetime('2020-4-1')
end_date = pd.to_datetime('2020-9-30')
df['Date'] = pd.to_datetime(df['Date'])
new_df = (df['Date'] >= start_date) & (df['Date'] <= end_date)
df1 = df.loc[new_df]
df2 = df1[['Open', 'Close', 'High', 'Low']]
#df3 = df2.set_index('Date')
plt.figure(figsize=(25,25))
df2.plot.hist(alpha=0.5)
plt.suptitle('Opening/Closing/High/Low stock prices of Alphabet Inc.,\n From 01-04-2020 to 30-09-2020', fontsize=12, color='blue')
plt.show()
```

3. Write a Pandas program to create a stacked histograms plot of opening, closing, high, low stock prices of stock dataset between two specific dates with more bins.

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("alphabet_stock_data.csv")
start_date = pd.to_datetime('2020-4-1')
end_date = pd.to_datetime('2020-9-30')
df['Date'] = pd.to_datetime(df['Date'])
new_df = (df['Date'] >= start_date) & (df['Date'] <= end_date)
df1 = df.loc[new_df]
df2 = df1[['Open', 'Close', 'High', 'Low']]
```

```
plt.figure(figsize=(30,30))
df2.hist();
plt.suptitle('Opening/Closing/High/Low stock prices of Alphabet Inc., From 01-04-2020
to 30-09-2020', fontsize=12, color='black')
plt.show()
```

## Session #12

### Learning Objective

Pandas SQL Query

### Materials & Resources

Reference Books:

1. Wesley J Chen, Core Python Programming 2nd Edition, Kindle Edition
2. Jake VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data 1st Edition, Kindle Edition, Oreilly Publisher
3. Mark Summerfield, Programming in Python 3-A Complete Introduction to the Python Language, Second Edition, Addison Wesley
4. Phuong Vo.T.H, Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Ivan Idris, Python Data Analysis, Packt Publishing Ltd
6. Magnus Vilhelm Persson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd

### Exercise

1. Write a Pandas program to display all the records of a student file.

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa


```
import pandas as pd
employees = pd.read_csv(r"EMPLOYEES.csv")
departments = pd.read_csv(r"DEPARTMENTS.csv")
job_history = pd.read_csv(r"JOB_HISTORY.csv")
jobs = pd.read_csv(r"JOBS.csv")
countries = pd.read_csv(r"COUNTRIES.csv")
regions = pd.read_csv(r"REGIONS.csv")
locations = pd.read_csv(r"LOCATIONS.csv")
print("All the records from regions file:")
print(regions)
```

All the records from regions file:		
REGION_ID	REGION_NAME	
0	1	Europe
1	2	Americas
2	3	Asia
3	4	Middle East and Africa

2. Write a Pandas program to select distinct department id from employee's file.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	121	1500
50	Shipping	103	1400
60	IT	204	2700
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury	0	1700
130	Corporate Tax	0	1700
140	Control and Credit	0	1700
150	Shareholder Services	0	1700
160	Benefits	0	1700
170	Manufacturing	0	1700
180	Construction	0	1700
190	Contracting	0	1700
200	Operations	0	1700
210	IT Support	0	1700
220	NOC	0	1700
230	IT Helpdesk	0	1700
240	Government Sales	0	1700
250	Retail Sales	0	1700
260	Recruiting	0	1700
270	Payroll	0	1700





```
import pandas as pd
employees = pd.read_csv(r"EMPLOYEES.csv")
departments = pd.read_csv(r"DEPARTMENTS.csv")
job_history = pd.read_csv(r"JOB_HISTORY.csv")
jobs = pd.read_csv(r"JOBS.csv")
countries = pd.read_csv(r"COUNTRIES.csv")
regions = pd.read_csv(r"REGIONS.csv")
locations = pd.read_csv(r"LOCATIONS.csv")
print("Distinct department_id:")
print(employees.department_id.unique())
```

**Distinct department\_id:**

```
[ 90.  60. 100.  30.  50.  80. nan  10.  20.  40.  70. 110.]
```