**RECORD WORK**

**PRACTICAL – 1**

1. **Use matplotlib and plot an inline in jupyter.**

**Aim :** To plot the graphs using matplotlib.

**Matplotlib :** A matplotlib is a python library that helps in visualizing and analyzing the data and helps in better understanding of the data with the help of graphical, pictorial visualizations that can be simulated using the matplolib library. Matplotlib is a comprehensive library for static, animated and interactive visualizations.

**Matplotlib inline :**  Now, let’s talk about the %matplotlib function. It sets up the matplot lib to work interactively. It lets you activate the matplolib interactively. It lets you activate the matplotlib interactive support anywhere in an **IPython** session (like in jupyter notebook).

When you enable the ‘inline’ matplotlib backend, the output of the plotting commands written will be displayed inline within the frontends like jupyter notebook. It means, the plot/graph will be displayed directly below the cell(where the plotting commands are written) and the resulting plot/graph will also be included (stored) in your notebook document.

#Scatter plots are used to observe relationships between variables and use dots to represent the relationship between them.The scatter() method in the matplotlib library is used to draw a scatter plot.

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

tips=pd.read\_csv("C:/Users/dell/Desktop/tips.csv")

#scatter plot with day against tip

plt.scatter(tips['day'], tips['tip'])

#adding title to the plot

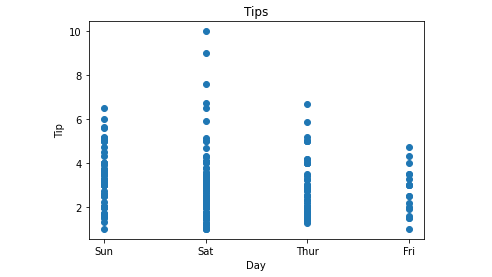
plt.title("Tips")

#adding names to the x and y axises

plt.xlabel("Day")

plt.ylabel("Tip")

plt.show()



**PRACTICAL – 2**

**Implement Commands of Python Language basics**

Python is **an interpreted, interactive, object-oriented programming language**. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. It supports multiple programming paradigms beyond object-oriented programming, such as procedural and functional programming.

Some of the basic commands in python are:

* **Comments:**

To write comments in python we use hash symbol (#).

**In[] : #** Sum of two numbers

a=5

b=4

c = a + b

print(c)

**Out[] :** 9

* **Creating variables:**

Variables in python must start with a lower case letter. Python variable is areserved memory location to store values.

Ex:

**1.In []:** x = 5 # Value assigning to a variable. Single assignment

x

**Out []:** 5

**2. In []:** a=b=1

**Out []:** 1

* **Identifiers:**
* Identifiers are names given to different entities such as constants, variables, structures, functions. An Identifier can be of any length.
* Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore \_.
* An identifier cannot start with a digit Keywords cannot be used as identifiers.

**Valid Identifiers:**

1.var1

2. \_var1

3.\_1\_var

4. var\_1

* Special characters should not be used in identifiers.
* **Keywords:**

The keywords are some predefined and reserved words in python that have special meanings. Keywords are used to define the syntax of the coding. The keyword cannot be used as an identifier, function, and variable name.

**Keywords in python are**:

| and | as | [assert](https://www.geeksforgeeks.org/python-assert-keyword/) | [break](https://www.geeksforgeeks.org/python-break-statement/) |
| --- | --- | --- | --- |
| [class](https://www.geeksforgeeks.org/python-classes-and-objects/) | [continue](https://www.geeksforgeeks.org/python-continue-statement/) | def | [del](https://www.geeksforgeeks.org/python-del-to-delete-objects/) |
| elif | else | [except](https://www.geeksforgeeks.org/python-try-except/) | False |
| [finally](https://www.geeksforgeeks.org/finally-keyword-in-python/) | [for](https://www.geeksforgeeks.org/python-for-loops/) | from | [global](https://www.geeksforgeeks.org/global-keyword-in-python/) |
| if | [import](https://www.geeksforgeeks.org/import-module-python/) | in | is |
| [lambda](https://www.geeksforgeeks.org/python-lambda/) | None | nonlocal | not |
| or | [pass](https://www.geeksforgeeks.org/python-pass-statement/) | raise | [return](https://www.geeksforgeeks.org/python-return-statement/) |
| True | [try](https://www.geeksforgeeks.org/python-try-except/) | [while](https://www.geeksforgeeks.org/python-while-loop/) | [with](https://www.geeksforgeeks.org/with-statement-in-python/) |
| yield |  |  |  |

* **Literals:**

These are notations for representing a fixed value. They can also be defined as raw value or data given in variables or constants.

Types of literals:

1. Numeric Literals:

They are immutable and there are three types of numeric literal :

**In []:** x **=** 88 # Integer literal

y **=** 24.3 # Float literal

z = 6+2i # Complex literal

print(x, y ,z)

**Out []:** 88 24.3 6+2i

1. String literals:

A string literal can be created by writing a text(a group of Characters ) surrounded by the single(”), double(“”), or triple quotes. By using triple quotes we can write multi-line strings or display in the desired way.

**Ex:**

**In []:** s **=** 'Hello'

# in double quotes

d **=** " Hello everyone "

 # multi-line String

k **=** '''Hello

            everyone'''

print(s)

print(d)

print(k)

**Out []:**

Hello

Hello everyone

Hello

Everyone

C) Boolean Literals:

There are only two Boolean literals in Python. They are **true**and **false**.

**In []:** a **=** (1 **==** True)

b **=** (0 **==** False)

**Out []:** a is True

B is False

**PRACTICAL – 3**

**Create Tuples, Lists and illustrate slicing conventions.**

**List:** Lists are used to store multiple items in a single variable. Lists are created using square brackets. List items are ordered, changeable, and allow duplicate values. List items are indexed, the first item has index [0], the second item has index [1] etc.

# creation a list

my\_list = [20,50,"alex",60]

# some of the list operations

my\_list.append(55)

my\_list

[20, 50, 'alex', 60, 55]

my\_list.extend(["manu",76])

my\_list

[20, 50, 'alex', 60, 55, 'manu', 76]

my\_list.insert(2, "love")

my\_list

[20, 50, 'love', 'alex', 60, 55, 'manu', 76]

my\_list.pop(4)

60

my\_list

[20, 50, 'love', 'alex', 55, 'manu', 76]

my\_list.remove(50)

my\_list

[20, 'love', 'alex', 55, 'manu', 76]

## extract one element

my\_list[4]

'manu'

## slicing the list - extraction of elements from the list

## Slice operation is performed on Lists with the use of a colon(:) [start index : end index] - if it is in sequence

my\_list[:]

[20, 'love', 'alex', 55, 'manu', 76]

my\_list[2:]

['alex', 55, 'manu', 76]

my\_list[1:5]

['love', 'alex', 55, 'manu']

## if we are using the index values from left to right it is from 0 if vice-versa its from -1

## we can also use negative indexing in lists

my\_list[-3:]

[55, 'manu', 76]

my\_list[-5]

'love'

**Tuples:** Tuples are used to store multiple items in a single variable. A tuple is a collection which is ordered and **unchangeable**. Tuples are written with round brackets.

## tuples

my\_tuple = ("Alex", "Sowji", 100, 95, 76)

my\_tuple

('Alex', 'Sowji', 100, 95, 76)

## slicing the tuple

my\_tuple[1:]

('Sowji', 100, 95, 76)

my\_tuple[:4]

('Alex', 'Sowji', 100, 95)

my\_tuple[4]

76

my\_tuple[-5]

'Alex'

**PRACTICAL – 4**

**Create built-in Sequence functions**

Python has a handful of useful sequence functions.

* **sorted () :**

The sorted function returns a new sorted list from the elements of any sequence.

**Ex:**

1. **In []:** sorted ([58,96,2,0,42,14])

**Out []:** [0,2,14,42,58,96]

1. **In []:** sorted (‘foot wear’)

**Out []:** [‘’, ‘a’, ‘e’, ‘f’, ‘o’, ‘o’, ‘r’, ‘t’, ‘w’]

The sorted function accepts the same arguments as the sort method lists.

* **zip ():**

Zip function pairs up the elements of a number of lists, tuples or other sequences to create a list of tuples:

**Ex:**

1.**In []:** s1 = [‘happy’, ‘see’, ‘foot’]

**In []:** s2 = [‘end’, ‘you’, ‘ball’]

**In []:** x = list (zip (s1, s2))

**Out []:** [(‘happy’, ‘end’), (‘see’, ‘you’), (‘foot’, ‘ball’)]

Zip can be applied in a clever way to “unzip” the sequence. Another way to think about this is converting a list of rows into a list of columns.

2.**In []:** d = [(‘Kajal’,‘Agarwal’), (‘Pooja’ , ‘Hegde’),(‘Aishwarya’ , ‘Roy’)]

**In []:** first\_ names, last\_ names = zip(\*d)

**In []:** first \_ names

**Out []:** (‘Kajal’, ‘Pooja’, ‘Aishwarya’)

**In []:** last\_ names

**Out []:** (‘Agarwal’, ‘Hegde’, ‘Roy’)

* **reversed ():**

Reversed iterates over the elements of a sequence in reverse order.

**Ex:**

1.**In []:** list (reversed (range (5)))

**Out []:** [4,3,2,1,0]

2.**In []:** tuple (reversed (range (6)))

**Out []:** (5,4,3,2,1,0)

* **enumerate ():**

Python has inbuilt function, enumerate, returns a sequence of (i, values) tuples:

The enumerate function adds a counter as the key of the enumerate object.

**Syntax:**

Enumerate (*iterable*, *start*)

Ex:

1. **In []:** x = ('Orange', 'papaya', 'cherry')

y = enumerate(x,0)

print(list(y))

1. **In []:** some \_list = ['foo', 'bar', 'bit']

mapping = {}

for i, v in enumerate (some \_list):

mapping[v] = i

mapping

**Out []:** {'foo': 0, 'bar': 1, 'bit': 2}

**PRACTICAL – 5**

**Clean the elements and transform them by using List, Set , DictionaryComprehensions**

In Python, use list[] methods clear(), pop(), and remove(), to remove items(elements) from a list.

# Using clear () method:

#clearing list:

In [1]: GEEK = [39, 43, 54, 65]

Print(‘GEEK before clear:’ , GEEK)

GEEK.clear()

Print(‘GEEK after clear:’ ,GEEK)

Out [1]:

GEEK before clear: [39,43,54,65]

GEEK after clear: []

This is how we clean the elements in the list

Now about transforming the elements using List, Set and Dictionary Comprehensions .

Firstly transforming through Lists: List comprehensions are one of the most-loved Python language features. They allow you to concisely form a new list by filtering the elements of a collection, transforming the elements passing the filter in one concise expression. They take the basic form:

[expr for val in collection if condition]

It can be done in for-in loop

This is equivalent to the following loop:

result = []

for val in collection:

if condition:

result.append(expr)

The filter condition can be omitted, leaving only the expression. For example, given a list of strings, we could filter out strings with length 2 or less and also convert them to uppercase like this:

In [2]: strings = [‘a’, ‘as’, ‘bat’, ‘car’, ‘dove’, ‘python’]

In [2]: [x.upper() for x in strings if len(x) > 2]

Out [2]: [‘BAT’, ‘CAR’, ‘DOVE’, ‘PYTHON’ ]

Set and dict comprehensions are a natural extension, producing sets and dicts in an idiomatically similar way instead of lists. A dict comprehension looks like this: dict\_comp = {key-expr: value-expr for value in collection if condition}

A set comprehension, looks like the equivalent list comprehension except with curly braces instead of square brackets:

Set\_comp = {expr for value in collection if condition}

Like list comprehensions, set and dict comprehensions are mostly conveniences, but they similarly can make code both easier to write and read. Consider the list of strings from before. Suppose we wanted a set containing just the lengths of the strings contained in the collection; we could easily compute this using a set comprehension:

In [3]: unique\_lengths = {len(x) for x in strings}

Unique\_lengths

Out [3]: {1, 2, 3, 4, 6}

As a simple dict comprehension example, we could create a lookup map of these strings to their locations in the list:

In [4]: loc\_mapping = {val : index for index, val in enumerate(strings)}

In [4]: loc\_mapping Out [4]: {‘a’: 0, ‘as’: 1, ‘bat’: 2, ‘car’: 3, ‘dove’: 4, ‘python’: 5}

**PRACTICAL – 6**

**Create a functional pattern to modify the strings at a high level.**

### **1. Stripping Whitespace**

Stripping whitespace is an elementary string processing requirement. You can strip leading whitespace with the lstrip() method (left), trailing whitespace with rstrip() (right), and both leading and trailing with strip().

**Input:**

s = ' This is a sentence with whitespace. \n'

print('Strip leading whitespace: {}'.format(s.lstrip()))

print('Strip trailing whitespace: {}'.format(s.rstrip()))

print('Strip all whitespace: {}'.format(s.strip()))

**Output:**

Strip leading whitespace: This is a sentence with whitespace.

Strip trailing whitespace: This is a sentence with whitespace.

Strip all whitespace: This is a sentence with whitespace.

Interested in stripping characters other than whitespace? The same methods are helpful, and are used by passing in the character(s) you want stripped.

**Input:**

s = 'This is a sentence with unwanted characters.AAAAAAAA'

print('Strip unwanted characters: {}'.format(s.rstrip('A')))

**Output:**

Strip unwanted characters: This is a sentence with unwanted characters.

Don't forget to check out the string format() documentation if necessary.

### **2. Splitting Strings**

Splitting strings into lists of smaller substrings is often useful and easily accomplished in Python with the split() method.

**Input:**

s = 'KDnuggets is a fantastic resource'

print(s.split())

**Output:**

['KDnuggets', 'is', 'a', 'fantastic', 'resource']

By default, split() splits on whitespace, but other character(s) sequences can be passed in as well.

**Input:**

s = 'these,words,are,separated,by,comma'

print('\',\' separated split -> {}'.format(s.split(',')))

s = 'abacbdebfgbhhgbabddba'

print('\'b\' separated split -> {}'.format(s.split('b')))

**Output:**

',' separated split -> ['these', 'words', 'are', 'separated', 'by', 'comma']

'b' separated split -> ['a', 'ac', 'de', 'fg', 'hhg', 'a', 'dd', 'a']

### **3. Joining List Elements Into a String**

Need the opposite of the above operation? You can join list element strings into a single string in Python using the join() method.

**Input:**

s = ['KDnuggets', 'is', 'a', 'fantastic', 'resource']

print(' '.join(s))

**Output:**

KDnuggets is a fantastic resource

### **4. Reversing a String**

Python does not have a built-in string reverse method. However, given that strings can be [sliced](https://docs.python.org/3/reference/expressions.html?highlight=slice#slicings) like lists, reversing one can be done in the same succinct fashion that a list's elements can be reversed.

**Input:**

s = 'KDnuggets'

print('The reverse of KDnuggets is {}'.format(s[::-1]))

**Output:**

The reverse of KDnuggets is: steggunDK

### **5. Converting Uppercase and Lowercase**

Converting between cases can be done with the upper(), lower(), and swapcase() methods.

**Input:**

s = 'KDnuggets'

print('\'KDnuggets\' as uppercase: {}'.format(s.upper()))

print('\'KDnuggets\' as lowercase: {}'.format(s.lower()))

print('\'KDnuggets\' as swapped case: {}'.format(s.swapcase()))

**Output:**

'KDnuggets' as uppercase: KDNUGGETS

'KDnuggets' as lowercase: kdnuggets

'KDnuggets' as swapped case: kdNUGGETS

### **6. Checking for String Membership**

The easiest way to check for string membership in Python is using the in operator. The syntax is very natural language-like.

**Input:**

s1 = 'perpendicular'

s2 = 'pen'

s3 = 'pep'

print('\'pen\' in \'perpendicular\' -> {}'.format(s2 in s1))

print('\'pep\' in \'perpendicular\' -> {}'.format(s3 in s1))

**Output:**

'pen' in 'perpendicular' -> True

'pep' in 'perpendicular' -> False

If you are more interested in finding the location of a substring within a string (as opposed to simply checking whether or not the substring is contained), the find() string method can be more helpful.

**Input:**

s = 'Does this string contain a substring?'

print('\'string\' location -> {}'.format(s.find('string')))

print('\'spring\' location -> {}'.format(s.find('spring')))

**Output:**

'string' location -> 10

'spring' location -> -1

find() returns the index of the first character of the first occurrence of the substring by default, and returns -1 if the substring is not found. Check the documentation for available tweaks to this default behavior.

### **7. Replacing Substrings**

What if you want to *replace* substrings, instead of just find them? The Python replace() string method will take care of that.

**Input:**

s1 = 'The theory of data science is of the utmost importance.'

s2 = 'practice'

print('The new sentence: {}'.format(s1.replace('theory', s2)))

**Output:**

The new sentence: The practice of data science is of the utmost importance.

An optional count argument can specify the maximum number of successive replacements to make if the same substring occurs multiple times.

### **8. Combining the Output of Multiple Lists**

Have multiple lists of strings you want to combine together in some element-wise fashion? No problem with the [zip()](https://docs.python.org/3/library/functions.html#zip) function.

**Input:**

countries = ['USA', 'Canada', 'UK', 'Australia']

cities = ['Washington', 'Ottawa', 'London', 'Canberra']

for x, y in zip(countries, cities):

print('The capital of {} is {}.'.format(x, y))

**Output:**

The capital of USA is Washington.

The capital of Canada is Ottawa.

The capital of UK is London.

The capital of Australia is Canberra.

### **9. Checking for Anagrams**

Want to check if a pair of strings are anagrams of one another? Algorithmically, all we need to do is count the occurrences of each letter for each string and check if these counts are equal. This is straightforward using the Counter class of the collections module.

**Input :**

from collections import Counter

def is\_anagram(s1, s2):

return Counter(s1) == Counter(s2)

s1 = 'listen'

s2 = 'silent'

s3 = 'runner'

s4 = 'neuron'

print('\'listen\' is an anagram of \'silent\' -> {}'.format(is\_anagram(s1, s2)))

print('\'runner\' is an anagram of \'neuron\' -> {}'.format(is\_anagram(s3, s4)))

**Output :**

'listen' an anagram of 'silent' -> True

'runner' an anagram of 'neuron' -> False

### **10. Checking for Palindromes**

How about if you want to check whether a given word is a palindrome? Algorithmically, we need to create a reverse of the word and then use the == operator to check if these 2 strings (the original and the reverse) are equal.

**Input**

def is\_palindrome(s):

reverse = s[::-1]

if (s == reverse):

return True

return False

s1 = 'racecar'

s2 = 'hippopotamus'

print('\'racecar\' a palindrome -> {}'.format(is\_palindrome(s1)))

print('\'hippopotamus\' a palindrome -> {}'.format(is\_palindrome(s2)))

**Output**

'racecar' is a palindrome -> True

'hippopotamus' is a palindrome -> False

**PRACTICAL – 7**

**Write a python program to cast a string to a floating-point number but fails with value error on improper inputs using errors and exception handling.**

*->Handling python errors or exceptions gracefully is an important part of building robust programs.*

*->Python's float function is capable of casting a string to a floating-point number, but fails with valueError on improper inputs.*

**In [1]: float ('1.2345')**

**Out [1]: 1.2345**

**In [2]: float('something')**

**Out [2]: ValueError** Traceback **ValueError**: could not convert string to float: 'something'

*Suppose we wanted a version of float that fails gracefully, returning the input argument. We can do this by writing a function that encloses the call to float in a try/except block:*

***def attempt\_float(x):***

***try:***

***return float(x)***

***except:***

**return x**

*The code in the except part of the block will only be executed if float(x) raises an exception:*

**In [3]: attempt\_float('1.2345')**

**Out [3]: 1.2345**

**In [4]: attempt\_float('something')**

**Out [4]: 'something'**

*You might notice that float can raise exceptions other than ValueError:*

**In [5]: float((1,2))**

**Out [5]: TypeError** Traceback **TypeError**: float() argument must be a string or a number, not 'tuple'  
  
*you might want to only suppress ValueError, since a TypeError (the input was not a string or numeric value) might indicate a legitimate bug in your program. To do that, write the exception type after except:*

**def attempt\_float(x):**

**try:**

**return float(x)**

**except ValueError:**

**return x**

*We have then:*

**In [6]: attempt\_float((1,2))**

**Out [6]: TypeError**

**TypeError**: float() argument must be a string or a number, not 'tuple'.

*You can catch multiple exception types by writing a tuple of exception types instead.*

**def attempt\_float(x):**

**try:**

**return float(x)**

**except (TypeError,ValueError):**

**return x**

*In some cases, you may not want to suppress an exception, but you want some code to be executed regardless of whether the code in the try block succeeds or not.*

**PRACTICAL – 8**

**AIM: Create an n array object and use operations on it.**

➢ Numpy is a Python package which means ‘Numerical Python’. It is the library for logical computing, which contains a powerful n-dimensional array object.

➢ NumPy Array: Numpy array is a powerful N-dimensional array object which is in the form of rows and columns. We can initialize NumPy arrays from nested Python lists and access it elements.

Creating nd arrays:

1d array:

In [ 1 ] : d1 = [6, 7.5, 8, 0, 1]

In [ 2 ] : arr1 = np.array ( d1 )

In [ 3 ] : arr1

Out [ 3 ] : array ( [6. , 7.5, 8. , 0. , 1. ] )

Multi dimensional array:

In [ 4 ] : d2 = [ [1, 2, 3, 4] , [5, 6, 7, 8] ]

In [ 5 ] : arr2 = np.array ( d2 )

In [ 6 ] : arr2

Out [ 6 ] : array ( [1, 2, 3, 4], [5, 6, 7, 8] ] )

Operations on arrays:

In [ 7 ] : arr2.ndim

Out [ 7 ] : 2

In [ 8 ] : arr2.shape

Out [ 8 ] : (2,4)

In [ 9 ] : arr1.dtype

Out [ 9 ] : dtype (‘Float64’)

In [ 10 ] : np.zeros ( 3 )

Out [ 11 ] : array ( [0., 0., 0.] )

In [ 12 ] : np.zeros ( (3,3) )

Out [12]: Array ( [ [0., 0., 0. ], [0., 0., 0. ], [0., 0., 0. ] ] )

In [13] : np.empty ( (2, 3, 2) )

Out [13] : array ( [ [ [0., 0. ], [0., 0. ], [0., 0. ] ], [ [0., 0. ], [0., 0. ], [0., 0. ] ] ] )

In [ 14 ] : np.arange ( 10 )

Out [ 15 ] : array ( [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ] )

**PRACTICAL – 9**

**Use arithmetic operations on Numpy Arrays**

Create a NumPy ndarray Object

NumPy is used to work with arrays.

We can create a NumPy array by using the array() function.

Example:

Input: import numpy as np

X = np.array([12,34,56,86,34])

print(X)

Output: [12,34,56,86,34]

Arithmetic Operations on numpy array Addition “+” symbol is used to add two arrays.

Example:

Input: import numpy as np

arr1 = np.array([10, 11, 12, 13, 14, 15])

arr2 = np.array([20, 21, 22, 23, 24, 25])

newarr =arr1 + arr2

print(newarr)

Output: [30 32 34 36 3840]

Subtraction “-“ symbol is used to subtract one array from another.

Example:

Input: import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])

arr2 = np.array([20, 21, 22, 23, 24, 25])

newarr =arr1 - arr2

print(newarr)

Output: [-10 -1 8 17 26 35]

Multiplication “\*” symbol is used to multiply two arrays.

Example:

Input: import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])

arr2 = np.array([20, 21, 22, 23, 24, 25])

newarr = arr1 \* arr2

print(newarr)

Output: [200 420 660 920 1200 1500]

Division “/” symbol is used to divide two arrays. It will generate an warning message when any element is divided by zero but not an error.

Example:

Input: import numpy as np

arr1 = np.array([10, 20, 30, 40, 50, 60])

arr2 = np.array([3, 5, 10, 8, 2, 33])

newarr = arr1 / arr2

print(newarr)

Output: [3.3333 4. 3. 5. 25. 1.81818182]

**PRACTICAL - 10**

**Using Numpy arrays perform Indexing and Slicing Boolean Indexing, FancyIndexing operations.**

**Array :**  Array is a container which can hold a fix number of items and these items should be of the same type.

**Array Indexing :** Array indexing is the same as accessing an array element. You can access an array element by referring to its index number.

**Indexing for 1-D array**

import numpy as np  
arr = np.array([1, 2, 3, 4])  
print(arr[1]) # here we got second element from array

**output** : 2

**Indexing for 2-D array**

import numpy as np  
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])  
print('2nd element on 1st row: ', arr[0, 1])

**output: 2**

**Indexing for 3-D 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])

**output: 6**

**Negative Indexing :**

Use negative indexing to access an array from the end with -1,-2………

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])

**output** :10

## **Slicing arrays :**

Slicing means taking elements from one given index to another given index.

**SLICING FOR 1-D ARRAY :**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])

**output** : [2 3 4 5]

> print(arr[4:])

output:[5 6 7]

> print(arr[:4])

output:[1 2 3 4]

> print(arr[-3:-1])

output:[5 6]

**Slicing 2-D Arrays**

**> import**numpy as np  
 arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])  
 print(arr[1, 1:4])

output: [7 8 9]

> print(arr[0:2, 1:4])

output:[[2 3 4]

[7 8 9]]

**Numpy Boolean Indexing:**

**import** **numpy** **as** **np**

A = np.array([4, 7, 3, 4, 2, 8])

**print**(A == 4)

### **OUTPUT: [ True False False True False False]**

**print**(A < 5)

### **OUTPUT: [ True False True True True False]**

It tells only the Boolean statements

B = np.array([[42,56,89,65],

[99,88,42,12],

[55,42,17,18]])

**print**(B>=42)

### **OUTPUT:[[ True True True True]**

### **[ True True True False]**

### **[ True True False False]]**

**FANCY INDEXING OPERATIONS :**

Fancy indexing  means passing an array of indices to access multiple array elements at once.

fancy indexing always allows you to select enter rows or columns out of order to show this.

Input: s=np.zeros((2,2))

s

output: array([[0., 0.,],

[0., 0.,]])

> s[[2,4,6,8]]

Output: array([[0., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 0., 0.],

[0., 0., 0., 0., 0., 0.]])

> f **=** s.shape[0] *# here 0 indicates no.of columns*

f

output:10

*# set up array*

Input: s[[2,4]]

Output: array([[2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],

[4., 4., 4., 4., 4., 4., 4., 4., 4., 4.]])

*# allows any order*

Input: s[[5,2,1]]

Output: array([[5., 5., 5., 5., 5., 5., 5., 5., 5., 5.],

[2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],

[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])

**PRACTICAL - 11**

**Aim:- create an image plot from a two-dimensional array of function values.** Procedure:-

BAR PLOT - A bar plot shows categorical data as rectangular bars with the height of bars proportional to the value they represent.

In[1] : # importing libraries

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

# Define a data set

data= {'apples' : 20, 'Mangoes' : 15, 'lemon' : 30, 'oranges' : 10}

names = list(data.keys())

values= list(data.values())

fig = plt.figure(figsize=(10,5))

plt.barh(names,values,color="orange")

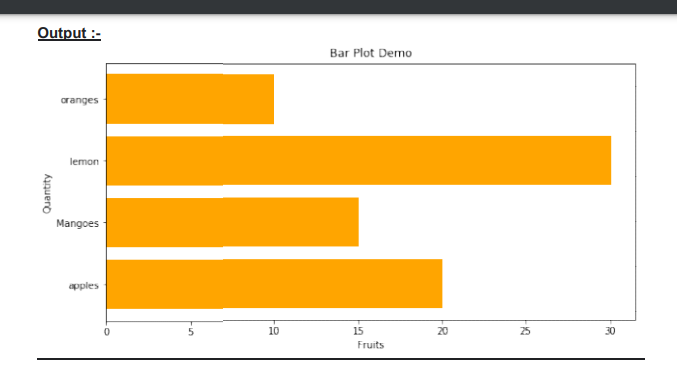
plt.title("Bar Plot Demo")

plt.xlabel("Fruits")

plt.ylabel("Quantity")

plt.show()

Output :-



BOX PLOT :-

Boxplots are set is.

In[2] :

import matplotlib.pyplot as plt

%matplotlib inline

#Data Prep

total = [20,4,1,30,20,10,20,70,30,10]

order = [10,3,1,15,17,2,30,44,2,1]

discount = [30,10,20,5,10,20,50,60,20,45]

data = list([total, order, discount])

#plotting the data

plt.boxplot(data, showmeans)

plt.title("Box Plot")

plt.grid(True)

plt.show()

Boxplots are a measure of how well distributed the data in a data.

In[2] :

import matplotlib.pyplot as plt

%matplotlib inline

#Data Prep

total = [20,4,1,30,20,10,20,70,30,10]

order = [10,3,1,15,17,2,30,44,2,1]

discount = [30,10,20,5,10,20,50,60,20,45]

data = list([total, order, discount])

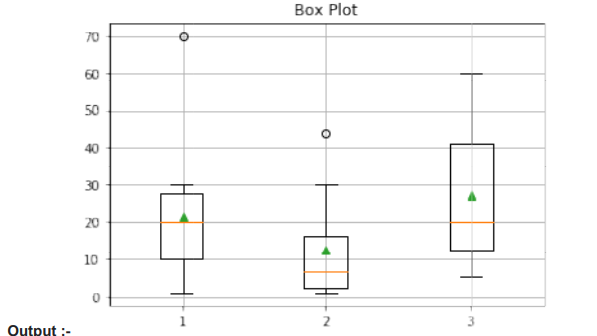
#plotting the data

plt.boxplot(data, showmeans)

plt.title("Box Plot")

plt.grid(True)

plt.show()



HISTOGRAM :- A histogram is basically some groups.

In[8] : import matplotlib.pyplot as plt

%matplotlib inline

numbers = [10,12,13,45,67,23,45,89,12,45,90,32]

plt.hist(numbers,bins=[0,20,40,60,80,100],edgecolor= plt.title("Histogram Plot Demo") plt.xlabel("Rane of values")

plt.ylabel("Frequencies")

plt.grid(True) Axis") Axis")

plt.legend(['line1'],loc="best")

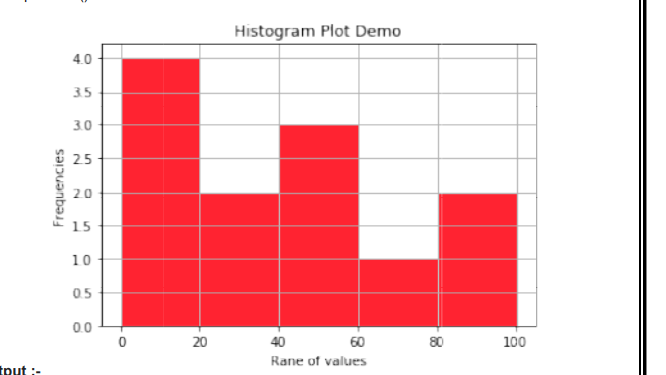
A histogram is basically used to represent data provided in a form of

import matplotlib.pyplot as plt

numbers = [10,12,13,45,67,23,45,89,12,45,90,32]

plt.hist(numbers,bins=[0, 20,40,60,80,100], edgecolor='#FFFFFF', color='#FF2331')

plt.show()



**PRACTICAL – 12**

**Basic array statistical methods**

**Aim: T**o implement some basic array statistical methods and sorting with sort method

Some basic array statistical methods are:

* sum
* mean
* std
* var
* min
* max
* argmin
* argmax
* cumsum
* cumprod

Input: import numpy as np

a=np.array([[5,6,1],[2,5,7],[3,6,5]])

a

output: array([[5, 6, 1],

[2, 5, 7],

[3, 6, 5]])

**1.sum( ):**

Input: np.sum(a)

output:40

**2.mean( ):**

Input: np.mean(a)

output:4.444444444444444445

**3.std( ):**

Input: np.std(a)

output:892154040658489

**4.var( ):**

Input: np.var(a)

output:3.580246913580247

**5.min( ):**

Input: np.min(a)

output:1

**6.max( ):**

Input: np.max(a)

output:7

**7.argmin( ):**returns the index of minimum value in array

Input: np.argmin(a)

output:2

**8.argmax( ):** returns the index of maximum value in array

Input: np.argmax(a)

output:5

**9.cumsum( ):** returns the cumulative sum of all the elements

Input: np.cumsum(a)

output: array([ 5, 11, 12, 14, 19, 26, 29, 35, 40], dtype=int32)

**10.cumprod( ):** returns the cumulative product of all the elements

Input: np.argmax(a)

output: array([ 5, 30, 30, 60, 300, 2100,

6300, 37800,189000], dtype=int32)

**Sorting**

**sort( ):** This method sorts the elements of the array in ascending or descending order

**1-D array:**

Input: import numpy as np

b=np.array([2,5,6,3])

b

output: array([2, 5, 6, 3])

Input: np.sort(b) #ascending order

output: array([2, 3, 5, 6])

Input: np.sort(b)[::-1] #descending order

output: array([6, 5, 3, 2])

**2-D array:**

Input: import numpy as np

a=np.array([[5,6,1],[2,5,7],[3,6,5]])

a

output: array([[5, 6, 1],

[2, 5, 7],

[3, 6, 5]])

Input: np.sort(a,axis=0) #sorts columns in ascending order

output: array([[2, 5, 1],

[3, 6, 5],

[5, 6, 7]])

Input: np.sort(a,axis=1) #sorts rows wise in ascending order

output: array([[1, 5, 6],

[2, 5, 7],

[3, 5, 6]])

**PRACTICAL - 13**

**13.** To implement numpy.random functions are used to generate random data in Python.

They are some ways:-

• randint()

• choice()

• shuffle()

• uniform()

• random()

• rand()

• permutation()

➢ np.random.rand()

Input:

import numpy as np

x= np.random.rand(3,2)

#getting a random float from 3 to 2

print(x)

Output: [[0.62131296 0.81520914] [0.79070212 0.55770464] [0.74670143 0.29963054]]

➢ np.random.randit()

Input:

np.random.randint(5, size=(2,4))

#returns a random numbers between the given range

Output: array([[0, 3, 1, 4], [1, 1, 3, 4]])

➢ np.random.choice()

Input: np.random.choice(5,2)

#generate a random sample from a given 1-D array

Output: array([3, 4])

Input: a=['Car','Bike','Cycle','Train'] b=np.random.choice(a)

#generate a random sample from a given list print('She has a '+b)

Output: She has a Bike

➢ np.random.shuffle()

Input:

arr=np.arange(9).reshape((3,3))

# modify as sequence in-place by shuffling its contents

np.random.shuffle(arr)

arr

Output: array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])

➢ np.random.uniform()

Input: s=np.random.uniform(-1,0,10)

# any value within the given interval is equally likely to be drawn by uniform s Output : array([-0.38279886, -0.74777372, -0.24510703, -0.88106394, -0.39874888, -0.12382905, -0.70945686, -0.12948132, -0.60052525, -0.64751423]) Input: import numpy as np

from numpy import random

x=np.random.uniform(size=(2,3))

x

Output: array([[0.82380789, 0.51628835, 0.98263177], [0.54818994, 0.15069844, 0.75708986]]

np.random.permutation(4)

Output: array([2, 3, 1, 0])

Input: np.random.seed(7)

print(random.random())

# Seed function is used to save the state of a random function.

Output: 0.07630828937395717

➢ np.random.permutation()

Input: import numpy as np

from numpy import random

arr =np.arange(16).reshape((4,4))

# Randomly permute a sequence, or return a permuted range. np.random.permutation(arr)

Output: array([[12, 13, 14, 15], [ 0, 1, 2, 3], [ 4, 5, 6, 7], [ 8, 9, 10, 11]])

➢ np.random.random()

Input: import numpy as np

x= np.random.random(5)

# it is randomly selected

print(x)

Output: [0.93120602 0.02489923 0.60054892 0.9501295 0.23030288]

**PRACTICAL - 14**

1. Plot the values of first 100 values on the values obtained from random walks.

**Aim :** To plot first 100 values on the values obtained from random walks.

**Random walk :** A random walk is a mathematical object, known as a stochastic or random process, that describes a path that consist of a succession of random steps on some mathematical space such as the integers.

An elementary example of a random walk is the random walk on the integer number line, which starts at 0 and at each step moves +1 or -1 with equal probability.A random walk is a discrete fractal , but a weiner process trajectory is a true fractal, and there is a connection between the two.

**Plotting the values :**

#To plot the random numbers obtained from random walks.

#importing numpy library.

import numpy as np

#importing matplotlib library.

#importing matplolib.pyplot.

import matplotlib.pyplot as plt

#matplolib.pyplot is a collection function that makes matplotlib work like MATLAB.

import random

#we use random.rand for random values.

rand = np.random.randn(100)

rand

#To obtain the first 100 values from random walk

rand\_plot= random.sample(range(1,300),100)

plt.plot(rand\_plot,color = 'r',alpha = 0.6)

plt.title("Random\_Number\_Plotting")

plt.show()

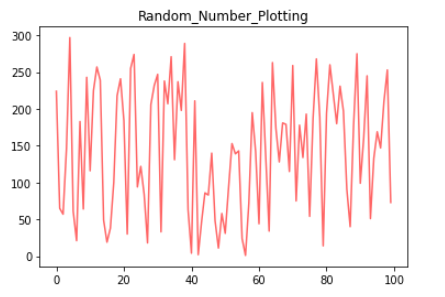
rand\_gen = random.sample(range(1,300),100)

plt.plot(rand\_gen,color = 'r',alpha = 0.6)

plt.title("Random\_number\_plotting")

plt.show()

# The required final output.



**PRACTICAL - 14**

**Plot the values of first 100 values on the values obtained from random walks.**

**Aim :** To plot first 100 values on the values obtained from random walks.

**Random walk :** A random walk is a mathematical object, known as a stochastic or random process, that describes a path that consist of a succession of random steps on some mathematical space such as the integers.

An elementary example of a random walk is the random walk on the integer number line, which starts at 0 and at each step moves +1 or -1 with equal probability.A random walk is a discrete fractal , but a weiner process trajectory is a true fractal, and there is a connection between the two.

**Plotting the values :**

#To plot the random numbers obtained from random walks.

#importing numpy library.

import numpy as np

#importing matplotlib library.

#importing matplolib.pyplot.

import matplotlib.pyplot as plt

#matplolib.pyplot is a collection function that makes matplotlib work like MATLAB.

import random

#we use random.rand for random values.

rand = np.random.randn(100)

rand

#To obtain the first 100 values from random walk

rand\_plot= random.sample(range(1,300),100)

plt.plot(rand\_plot,color = 'r',alpha = 0.6)

plt.title("Random\_Number\_Plotting")

plt.show()

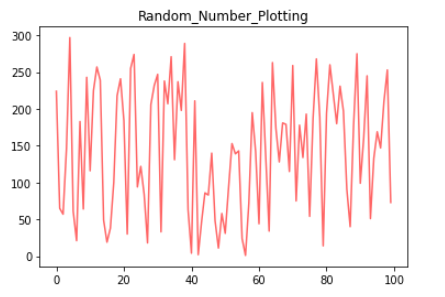
rand\_gen = random.sample(range(1,300),100)

plt.plot(rand\_gen,color = 'r',alpha = 0.6)

plt.title("Random\_number\_plotting")

plt.show()

# The required final output.



**Practical – 15**

**Create a data frame using pandas and retrieve the rows and columns in it by performing some indexing options and transpose it.**

**AIM:** To create a DATAFRAME and retrieving rows and columns from it by using some indexing options and transposing it.

**DATA FRAME:** A Data Frame in Pandas is a 2-dimensional data structure, like a 2dimensional array, or a table with rows and columns. Pandas Data Frame consists of three principal components the data, rows and columns.

**CREATING A DATAFRAME:**

# importing pandas library

import pandas as pd

# creating the data

data = {'Name': ['John', 'David', 'Jessi', 'Mary'],'Group': ['BSc','MScs','Mpcs','Bca'],'Subject': [ 'Data Science','Statistics','Maths','Computers'],'Marks': [91,89,89,95]}

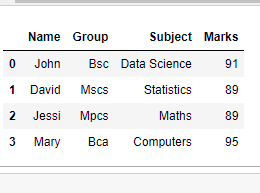
data

# converting data into DATAFRAME

df=pd.DataFrame(data)

df

**OUTPUT:**



**RETRIEVING ROWS AND COLUMNS USING INDEXING OPTIONS:**

We can use different ways to retrieve data from data frame by using indexing options. Indexing in pandas means simply selecting particular rows and columns of data from a Data Frame. Indexing can also be known as **Subset Selection**.

The [.loc](https://www.geeksforgeeks.org/python-pandas-extracting-rows-using-loc/) and [.iloc](https://www.geeksforgeeks.org/python-extracting-rows-using-pandas-iloc/) indexers also use the indexing operator to make selections.

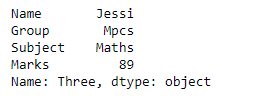
# indexing a data frame using indexing operator []

# Retrieving rows by loc method

row=df.loc["Three"]

row

**OUTPUT:**

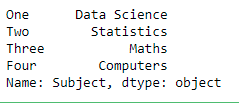


#Retrivieing columns by indexing operator

column= df["Subject"]

column

**OUTPUT:**

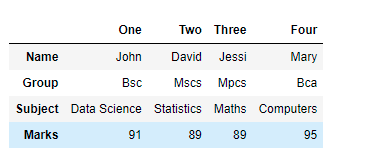


**######## TRANSPOSE DATAFRAME ##############**

df\_transpose = df.T

df\_transpose

**OUTPUT:**



**RESULTS:**

* Data Frame is created.
* Rows and columns are retrieved using indexing options.
* Data Frame is transposed.

**PRACTICAL - 16**

**Implement the methods of descriptive and summary statistics.**

Descriptive or summary statistics in python-pandas, can be obtained by using describe function – describe(). Describe function gives the mean, std and IQR values. We need to add a variable named include=’all’ to get the summary statistics or descriptive statistics of both numeric and character columns.

|  |
| --- |

|  |  |
| --- | --- |

## creation of dataframe

**In[1] :**Import pandas as pd

Import numpy as np

d={‘Name’:pd.Series([‘Alisa’,’Bobby’,’Rocky’]),’Age’:pd.Series([25,26,25]),’Rating’:pd.Series([4.23,3.24,3.98])}

**In[2] :**df=pd.DataFrame(d)

print df

**Output:**

|  | Name | Age | Rating |
| --- | --- | --- | --- |
| 0 | Alisa | 25 | 4.23 |
| 1 | Bobby | 26 | 3.24 |
| 2 | Rocky | 25 | 3.98 |

**In[3] :**print(df.sum())

Name AlisaBobbyRocky

Age 76

Rating 11.45

dtype:object

**In[4] :**print(df.sum(1))

**Out[4]:** 0 29.23

1 29.24

2 28.98

dtype:float 64

**In[5] :**print(df.mean())

**Out[5]:**Age 25.333333

Rating 3.816667

dtype:float 64

**In[6] :**print(df.std())

**Out[6]:**Age 0.577350

Rating 0.514814

dtype:float64

**In[7] :**print(df.count())

**Out[7]:**Name 3

Age 3

Rating 3

dtype:int64

**In[8] :**print(df.min())

**Out[8]:** Name Alisa

Age 25

Rating 3.24

dtype:object

**In[9] :**print(df.max())

**Out[9]:**Name Rocky

Age 26

Rating 4.23

dtype:object

**In[10] :**print(df.median())

**Out[10]:** Age 25.00

Rating 3.98

dtype:float64

**In[11] :**print(df.mode())

**Out[11] :** Name Age Rating

0 Alisa 25.0 3.24

1 Bobby NaN 3.98

2 Rocky NaN 4.23

**In[12] :**print(df.describe())

Age Rating

Count 3.000000 3.000000

Mean 25.333333 3.816667

Std 0.577350 0.514814

Min 25.000000 3.240000

25% 25.000000 3.610000

50% 25.500000 3.980000

75% 25.500000 4.105000

Max 26.000000 4.230000

**In[13] :**df.sort\_values(by=[‘Rating’])

**Out[13]:** Name Age Rating

1 Bobby 26 3.24

2 Rocky 25 3.98

0 Alisa 25 4.23

**PRACTICAL – 17**

**File Formats: Reading and Writing Data in Text Format**

All the powerful data structures like the Series and the DataFrames would avail to nothing, if the Pandas module wouldn't provide powerful functionalities for reading in and writing out data. It is not only a matter of having functions for interacting with files. To be useful to data scientists it also needs functions which support the most important data formats like

* Delimiter-separated files, like e.g. csv
* Microsoft Excel files
* HTML
* HXML
* JSON

Pandas offers two ways to read in CSV or DSV files to be precise

* DataFrame.from\_csv
* read\_csv

There is no big difference between those two functions, e.g. they have different default values in some cases and read\_csv has more parameters. We will focus on read\_csv, because DataFrame.from\_csv is kept inside Pandas for reasons of backwards compatibility

import pandas as pd

exchange\_rates = pd.read\_csv("/data1/dollar\_euro.txt", sep="\t")

print(exchange\_rates)

Pandas is a very powerful and popular framework for data analysis and manipulation. One of the most striking features of Pandas is its ability to read and write various types of files including CSV and Excel. You can effectively and easily manipulate CSV files in Pandas using functions like read\_csv() and to\_csv()

**File Formats: Reading and Writing Data in Binary Format**

A binary data file is nothing more than a large array of bytes that encodes a series of data elements such as integers, floats, or character arrays. While there are many formats for the binary encoding, one common format consists of a series of individual ‘records’ stored back-to-back one after another.

Generally, there will be multiple record types in the file, all of which share a common header format. For example, binary data from a car’s computer might have one record type for driver controls such as the brake pedal and steering wheel positions, and another type to record engine statistics such as fuel consumption and temperature.

**PRACTICAL -18**

**Implement the data Cleaning and Filtering methods**

Input : df.dropna(inplace=True)

print(df.isnull().sum())

output : A 0

B 0

C 0

Input : df

Output : A B C

0 45 56 19.0  
 1 30.0 NaN 25.0

2 NaN NaN 39.0

3 60.0 83.0 NaN

Input : df[“A”]=df[“A”].replace(np.NaN, df[“A”].mean())

print(df[“A”])

output : 0 45.0

1. 30.0
2. 45.0
3. 60.0

Name:A,dtype:float64

Input : import statistics

df[‘A’]=df[‘A’].replace(np.NaN,

statistics.mode(df[‘A’]))

print(df[‘A’])

output : 0 45.0

Name:A,dtype:float64

Input : df[‘C’]=df[‘C’].fillna(“0”)

df.isnull().sum()

output : A 0

B 0

C 0

dtype :int64

input: from sklearn impute SimpleImputer

import numpy as np

imputer = SimpleImputer(missing\_values=np.NaN,

strategy=”mean”)

imputer=imputer.fit(df[[“A”]])

df[“A”]=imputer.transform(df[[“A”]])

df

output: A B C

0 45.0 56.0 19.0

Input : f=lambda x:x\*2

df[‘A’]=df[‘A’].apply(f)

df

output : A B C

0 180.0 56.0 19.0

Here,we filled the missing value in ‘A’.By following

the above process we can fill for all columns and rows.

**PRACTICAL-19**

**Transform the data using function or mapping**

**map( ) :**

* The map( ) function executes a specified function for each item in an iterable. The item is

sent to the function as parameter.

* Python’s map( ) is a built-in function that allows you to process and transform all the items in iterable without using an explicit for loop, this technique commonly known as mapping.

**Syntax** :

* map (function, iterable)

| Parameter | Description |
| --- | --- |
| function | The function to execute for each item. |
| iterable | A sequence, collection or an iterator object.you can send as many iterables as you like , just make sure the function has one parameter for each iterable. |

* **map( ) with list[] :**
* We use [for loop](https://www.simplilearn.com/tutorials/python-tutorial/python-for-loop) and the map functions separately, to understand how the map() function works.

**Example: Using For Loop**

In[]: num = [3, 5, 7, 11, 13]

mul = []

for n in num:

     mul.append(n \*\* 2)

print (mul)

Output: [ 9, 25, 49, 121, 169]

**Example : Using the Python map() Function**

In[] : def mul(i):

      return i \* i

num = (3, 5, 7, 11, 13)

result = map(mul, num)

print(result)

# making the map object readable

mul\_output = list(result)

print(mul\_output)

Output: <map object at 0x7fc5a85e1ca0>

[ 9, 25, 49, 121, 169 ]

As you can see, the map() function iterates through the iterable, just like the for loop. Once the iteration is complete, it returns the map object. You can then convert the map object to a list and print it.

* **map () with Tuple:**

### this code, you will take a tuple containing some string values. You will then define a function to convert the strings to uppercase. Lastly, you must use the map in Python to use the tuple and the function, to convert the string values to uppercase.

### In[]: def letter(s):

return s.upper()

tuple\_exm = ('this' ,'is' ,'map' ,'in' ,'python')

upd\_tup = map(letter, tuple\_exm)

print(upd\_tup)

print(tuple(upd\_tup))

output: <map object at 0x7f835d87bca0>

[‘THIS’ , ‘IS’ , ‘MAP’ , ‘IN’ , ‘PYTHON’ ]

* **Using map( ) with Len( ) :**

In[] : data= [“Data science” , ”python” , “map”]

X= list(map(len,data))

print(x)

Output: [12, 6, 3]

In the above code , we use python len( ) function along with map( ) to find the length of the some words

**PRACTICAL - 20**

**Rearrange the data using the unstack() method of hierarchical Indexing.**

**Hierarchical indexing:** hierarchical indexing is nothing but working with more than 2 dimensional data using series and DataFrame concepts. It is also called multi - indexing.

**Note -** though we have panel data structure for multi dimensions, hierarchical indexing is a familiar concept.

**## Creating data - ie. DataFrame**

In[1] data = [("Employed",100000,200000,300000,"Male"), ("Unemployed",20000,100000,120000,"Male"), ("Employed",50000,100000,150000,"Female"), ("Unemployed",10000,50000,60000,"Female")]

In[2] pop = pd.DataFrame(data, columns = ["Job status", "Literates", "Illiterates", "Total", "Sex"])

pop

**Output:**

| **Job status** | **Literates** | **Illiterates** | **Total** | **Sex** |
| --- | --- | --- | --- | --- |
| **0** | Employed | 100000 | 200000 | 300000 | Male |
| **1** | Unemployed | 20000 | 100000 | 120000 | Male |
| **2** | Employed | 50000 | 100000 | 150000 | Female |
| **3** | Unemployed | 10000 | 50000 | 60000 | Female |

## Setting row index with two columns.

In[3] df = pop.set\_index(["Job status","Sex"])

In[4] df

**Output:**

**Literates Illiterates Total**

**Job status Sex**

**Employed Male** 100000 200000 300000

**Unemployed Male**  20000 100000 120000

**Employed Female** 50000 100000 150000

**Unemployed Female** 10000 50000 60000

**Unstack():** It is a function to change the data of the column index of one of the layers into a row index.

In[5] df.unstack()

**Output:**

**Literates Illiterates Total**

**Sex Female Male Female Male Female Male**

**Job status**

**Employed**  50000 100000 100000 200000 150000 300000

**Unemployed** 10000 20000 50000 100000 60000 120000

**PRACTICAL - 21**

**21.Implement the methods that summarize the statistics by levels.**

Descriptive or summary statistics in python – pandas, can be obtained by using describe function – describe(). Describe Function gives the mean, std and IQR values.

* Generally describe() function excludes the character columns and gives summary statistics of numeric columns
* We need to add a variable named include=’all’ to get the summary statistics or descriptive statistics of both numeric and character columns.

# creation of DataFrame

import pandas as pd

import numpy as np

#Create a Dictionary of series

d = {'Name':pd.Series(['Alisa','Bobby','Cathrine','Madonna','Rocky','Sebastian','Jaqluine',

'Rahul','David','Andrew','Ajay','Teresa']),

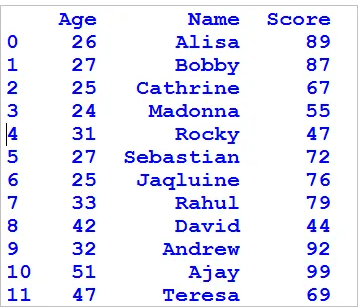
'Age':pd.Series([26,27,25,24,31,27,25,33,42,32,51,47]),

'Score':pd.Series([89,87,67,55,47,72,76,79,44,92,99,69])}

#Create a DataFrame

df = pd.DataFrame(d)

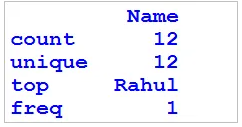
print df



# summary statistics of character column

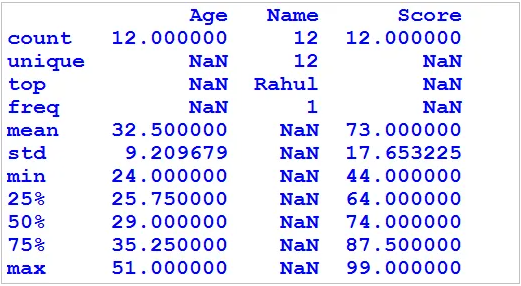
print df.describe(include=['object'])

* describe() Function with an argument named include along with value object i.e include=’object’ gives the summary statistics of the character columns.



# summary statistics of character column

print df.describe(include='all')



**PRACTICAL - 22**

**22. Use different Join types with arguments and merge data with keys and multiple keys.**

The pandas module contains various features to perform various operations on join two dataframes. There are mainly five types of Joins in Pandas:

> Inner Join

> Left Outer Join

> Right Outer Join

> Full Outer Join or simply Outer Join

> Index Join

## To perform a join operation we should create 2 data frames first.

In[1]: import pandas as pd

## creation of DataFrame

In[2]: wheat\_2020 = [("Punjab",125.84),

("Madhya Pradesh",113.38),

("Haryana",70.65),

("Uttar Pradesh",20.39),

("Rajasthan",10.63),

("Uttarakhand",0.31),

("Gujarat",0.21),

("Chandigarh",0.12)]

pulses\_2020 = [("Punjab",29.20),

("Rajasthan",4497.13),

("Uttar Pradesh",2447.32),

("Uttarakhand",57.79), ("Sikkim",5.04), ("Tamil Nadu",605.41),

("Telangana",549.18), ("Tripura",18.67)]

In[3]: wheat\_2020 = pd.DataFrame(wheat\_2020,

columns = ["States", "Production(In Tons)"])

In[4]: pulses\_2020 = pd.DataFrame(pulses\_2020,

columns = ["States","Production(In Tons)"])

In[5]: wheat\_2020

| Out[5]: **States** | | **Production(In Tons)** | |
| --- | --- | --- | --- |
| **0** | Punjab | | 125.84 |
| **1** | Madhya Pradesh | | 113.38 |
| **2** | Haryana | | 70.65 |
| **3** | Uttar Pradesh | | 20.39 |
| **4** | Rajasthan | | 10.63 |
| **5** | Uttarakhand | | 0.31 |
| **6** | Gujarat | | 0.21 |
| **7** | Chandigarh | | 0.12 |

In[6]: pulses\_2020

| Out[6]: **States** | | **Production(In Tons)** | |
| --- | --- | --- | --- |
| **0** | Punjab | | 29.20 |
| **1** | Rajasthan | | 4497.13 |
| **States** | | **Production(In Tons)** | |
| **2** | Uttar Pradesh | | 2447.32 |
| **3** | Uttarakhand | | 57.79 |
| **4** | Sikkim | | 5.04 |
| **5** | Tamil Nadu | | 605.41 |
| **6** | Telangana | | 549.18 |
| **7** | Tripura | | 18.67 |
|  |  | |  |

## Inner Join: Inner join is the most common type of join you’ll be working with. It returns a dataframe with only those rows that have common characteristics. This is similar to the intersection of two sets.

In[7]: df\_inner = pd.merge(wheat\_2020, pulses\_2020,

on='States', how='inner')

df\_inner

out[7]:

| **States** | | **Production(In Tons)\_x** | | **Production(In Tons)\_y** | |
| --- | --- | --- | --- | --- | --- |
| **0** | Punjab | | 125.84 | | 29.20 |
| **1** | Uttar Pradesh | | 20.39 | | 2447.32 |
| **2** | Rajasthan | | 10.63 | | 4497.13 |
| **3** | Uttarakhand | | 0.31 | | 57.79 |

## Left Outer Join: With a left outer join, all the records from the first dataframe will be displayed, irrespective of whether the keys in the first dataframe can be found in the second dataframe. Whereas, for the second dataframe, only the records with the keys in the second dataframe that can be found in the first dataframe will be displayed.

In[8]: df\_left = pd.merge(wheat\_2020, pulses\_2020,

on='States', how='left')

df\_left

out[8]:

| **States** | | **Production(In Tons)\_x** | | **Production(In Tons)\_y** | |
| --- | --- | --- | --- | --- | --- |
| **0** | Punjab | | 125.84 | | 29.20 |
| **1** | Madhya Pradesh | | 113.38 | | NaN |
| **2** | Haryana | | 70.65 | | NaN |
| **3** | Uttar Pradesh | | 20.39 | | 2447.32 |
| **4** | Rajasthan | | 10.63 | | 4497.13 |
| **5** | Uttarakhand | | 0.31 | | 57.79 |
| **6** | Gujarat | | 0.21 | | NaN |
| **7** | Chandigarh | | 0.12 | | NaN |

## Right Outer Join: For a right join, all the records from the second dataframe will be displayed. However, only the records with the keys in the first dataframe that can be found in the second dataframe will be displayed.

In[9]: df\_right = pd.merge(wheat\_2020, pulses\_2020,

on='States', how='right')

df\_right

out[9]:

| **States** | | **Production(In Tons)\_x** | | **Production(In Tons)\_y** | |
| --- | --- | --- | --- | --- | --- |
| **0** | Punjab | | 125.84 | | 29.20 |
| **1** | Rajasthan | | 10.63 | | 4497.13 |
| **2** | Uttar Pradesh | | 20.39 | | 2447.32 |
| **3** | Uttarakhand | | 0.31 | | 57.79 |
| **4** | Sikkim | | NaN | | 5.04 |
| **5** | Tamil Nadu | | NaN | | 605.41 |
| **6** | Telengana | | NaN | | 549.18 |
| **7** | Tripura | | NaN | | 18.67 |

## Full Outer Join: A full outer join returns all the rows from the left dataframe, all the rows from the right dataframe, and matches up rows where possible, with NaNs elsewhere. But if the dataframe is complete, then we get the same output.

In[10]: df\_outer = pd.merge(wheat\_2020, pulses\_2020,

on='States', how='outer')

df\_outer

out[10]:

| **States** | | **Production(In Tons)\_x** | | **Production(In Tons)\_y** | |
| --- | --- | --- | --- | --- | --- |
| **0** | Punjab | | 125.84 | | 29.20 |
| **1** | Madhya Pradesh | | 113.38 | | NaN |
| **2** | Haryana | | 70.65 | | NaN |
| **3** | Uttar Pradesh | | 20.39 | | 2447.32 |
| **4** | Rajasthan | | 10.63 | | 4497.13 |
| **5** | Uttarakhand | | 0.31 | | 57.79 |
| **6** | Gujarat | | 0.21 | | NaN |
| **7** | Chandigarh | | 0.12 | | NaN |
| **8** | Sikkim | | NaN | | 5.04 |
| **9** | Tamil Nadu | | NaN | | 605.41 |
| **10** | Telengana | | NaN | | 549.18 |
| **11** | Tripura | | NaN | | 18.67 |

In[11]: df\_index = pd.merge(wheat\_2020, pulses\_2020,

right\_index = True, left\_index = True)

df\_index

out[11]:

| **States\_x** | | **Production(In Tons)\_x** | | **States\_y** | | **Production(In Tons)\_y** | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Punjab | | 125.84 | | Punjab | | 29.20 |
| **1** | Madhya Pradesh | | 113.38 | | Rajasthan | | 4497.13 |
| **2** | Haryana | | 70.65 | | Uttar Pradesh | | 2447.32 |
| **3** | Uttar Pradesh | | 20.39 | | Uttarakhand | | 57.79 |
| **4** | Rajasthan | | 10.63 | | Sikkim | | 5.04 |
| **5** | Uttarakhand | | 0.31 | | Tamil Nadu | | 605.41 |
| **6** | Gujarat | | 0.21 | | Telengana | | 549.18 |
| **7** | Chandigarh | | 0.12 | | Tripura | | 18.67 |

In[12]: lis = [6, 4, 4, 3, 2, 2, 1, 2]

In[13]: wheat\_2020["Ranks"] = lis

In[14]: wheat\_2020

Out[14]:

| **States** | | **Production(In Tons)** | | **Ranks** | |
| --- | --- | --- | --- | --- | --- |
| **0** | Punjab | | 125.84 | | 6 |
| **1** | Madhya Pradesh | | 113.38 | | 4 |
| **2** | Haryana | | 70.65 | | 4 |
| **3** | Uttar Pradesh | | 20.39 | | 3 |
| **4** | Rajasthan | | 10.63 | | 2 |
| **5** | Uttarakhand | | 0.31 | | 2 |
| **6** | Gujarat | | 0.21 | | 1 |
| **7** | Chandigarh | | 0.12 | | 2 |

In[15]: lis2 = [6, 2, 1, 6, 5, 3, 2, 5]

In[16]: pulses\_2020["Ranks"] = lis2

In[17]: pulses\_2020

Out[17]:

| **States** | | **Production(In Tons)** | | **Ranks** | |
| --- | --- | --- | --- | --- | --- |
| **0** | Punjab | | 29.20 | | 6 |
| **1** | Rajasthan | | 4497.13 | | 2 |
| **2** | Uttar Pradesh | | 2447.32 | | 1 |
| **3** | Uttarakhand | | 57.79 | | 6 |
| **4** | Sikkim | | 5.04 | | 5 |
| **5** | Tamil Nadu | | 605.41 | | 3 |
| **6** | Telengana | | 549.18 | | 2 |
| **7** | Tripura | | 18.67 | | 5 |

**LAB MANUAL**

**NUMPY**

1 create an empty NumPy array

Ans.

import numpy as np

emptyarray = np.empty((3, 4), dtype=int)

print("Empty Array")

print(emptyarray)

2. create a full NumPy array

Ans.

import numpy as np

fullarray = np.full([3, 3], 55, dtype=int)

print("\n Full Array")

print(fullarray)

3. Check whether a Numpy array contains a specified row

Ans.

import numpy as np

array = np.array([[1,2,3,4,5],

[6, 7, 8, 9, 10],

[11, 12, 13, 14, 15],

[16, 17, 18, 19, 20]

])

print(array)

# check for some lists

print([1, 2, 3, 4, 5] in array.tolist())

print([16, 17, 20, 19, 18] in array.tolist())

4. Write a NumPy program to convert a list of numeric value into a one-dimensional NumPy array.

Ans.

import numpy as np

l = [12.23, 13.32, 100, 36.32]

print("Original List:",l)

a = np.array(l)

print("One-dimensional NumPy array: ",a)

5. Write a NumPy program to create an array with values ranging from 12 to 38.

Ans.

import numpy as np

x = np.arange(12, 38)

print(x)

6. Write a NumPy program to reverse an array (first element becomes last).

Ans.

import numpy as np

import numpy as np

x = np.arange(12, 38)

print("Original array:")

print(x)

print("Reverse array:")

x = x[::-1]

print(x)

7. Write a NumPy program to convert a list and tuple into arrays.

Ans.

import numpy as np

my\_list = [1, 2, 3, 4, 5, 6, 7, 8]

print("List to array: ")

print(np.asarray(my\_list))

my\_tuple = ([8, 4, 6], [1, 2, 3])

print("Tuple to array: ")

print(np.asarray(my\_tuple))

8. Write a NumPy program to create an empty and a full array.

Ans.

import numpy as np

# Create an empty array

x = np.empty((3,4))

print(x)

# Create a full array

y = np.full((3,3),6)

print(y)

9. Write a NumPy program to test whether each element of a 1-D array is also present in a second array.

Ans.

import numpy as np

array1 = np.array([0, 10, 20, 40, 60])

print("Array1: ",array1)

array2 = [0, 40]

print("Array2: ",array2)

print("Compare each element of array1 and array2")

print(np.in1d(array1, array2))

10. Write a NumPy program to get the unique elements of an array.

Ans.

import numpy as np

x = np.array([10, 10, 20, 20, 30, 30])

print("Original array:")

print(x)

print("Unique elements of the above array:")

print(np.unique(x))

x = np.array([[1, 1], [2, 3]])

print("Original array:")

print(x)

print("Unique elements of the above array:")

print(np.unique(x))

11. Write a NumPy program to change the dimension of an array.

Ans.

import numpy as np

x = np.array([1, 2, 3, 4, 5, 6])

print("6 rows and 0 columns")

print(x.shape)

y = np.array([[1, 2, 3],[4, 5, 6],[7,8,9]])

print("(3, 3) -> 3 rows and 3 columns ")

print(y)

x = np.array([1,2,3,4,5,6,7,8,9])

print("Change array shape to (3, 3) -> 3 rows and 3 columns ")

x.shape = (3, 3)

print(x)

12. Write a NumPy program to create a new shape to an array without changing its data.

Ans.

import numpy as np

x = np.array([1, 2, 3, 4, 5, 6])

y = np.reshape(x,(3,2))

print("Reshape 3x2:")

print(y)

z = np.reshape(x,(2,3))

print("Reshape 2x3:")

print(z)

13. Write a NumPy program to create a 1-D array going from 0 to 50 and an array from 10 to 50.

Ans.

import numpy as np

x = np.arange(50)

print("Array from 0 to 50:")

print(x)

x = np.arange(10, 50)

print("Array from 10 to 50:")

print(x)

14. Write a NumPy program to collapse a 3-D array into one dimension array.

Ans.

import numpy as np

x = np.eye(3)

print("3-D array:")

print(x)

f = np.ravel(x, order='F')

print("One dimension array:")

print(f)

15. Write a NumPy program to concatenate two 2-dimensional arrays.

Ans.

import numpy as np

a = np.array([[0, 1, 3], [5, 7, 9]])

b = np.array([[0, 2, 4], [6, 8, 10]])

c = np.concatenate((a, b), 1)

print(c)

16. Write a NumPy program to create an array of (3, 4) shape, multiply every element value by 3 and display the new array.

Ans.

import numpy as np

x= np.arange(12).reshape(3, 4)

print("Original array elements:")

print(x)

for a in np.nditer(x, op\_flags=['readwrite']):

a[...] = 3 \* a

print("New array elements:")

print(x)

17. Write a NumPy program to create a record array from a (flat) list of arrays.

Ans.

import numpy as np

a1=np.array([1,2,3,4])

a2=np.array(['Red','Green','White','Orange'])

a3=np.array([12.20,15,20,40])

result= np.core.records.fromarrays([a1, a2, a3],names='a,b,c')

print(result[0])

print(result[1])

print(result[2])

**PANDAS**

1. Write the code in python to create a dataframe from a given list.

L1 = [“Anil”, “Ruby”, “Raman”, “Suman”]

L2 = [35, 56, 48, 85]

Ans.

import pandas as pd

L1 = ["Anil", "Ruby", "Raman", "Suman"]

L2 = [35, 56, 48, 85]

DF = pd.DataFrame([L1, L2])

print(DF)

1. Write a program to create dataframe “DF” from “data.csv”

Ans.

import pandas as pd

DF = pd.read\_csv("data.csv")

print(df)

1. Creating a Pandas dataframe using list of tuples

Ans.

import pandas as pd

data = [('Peter', 18, 7),

('Riff', 15, 6),

('John', 17, 8),

('Michel', 18, 7),

('Sheli', 17, 5) ]

df = pd.DataFrame(data, columns =['Name', 'Age', 'Score'])

print(df)

1. Creating a dataframe from Pandas series

Ans.

import pandas as pd

import matplotlib.pyplot as plt

author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']

auth\_series = pd.Series(author)

print(auth\_series)

1. Reindexing the Rows using pandas

Ans.

# import numpy and pandas module

import pandas as pd

import numpy as np

column=['a','b','c','d','e']

index=['A','B','C','D','E']

df1 = pd.DataFrame(np.random.rand(5,5),

columns=column, index=index)

print(df1)

print('\n\nDataframe after reindexing rows: \n',

df1.reindex(['B', 'D', 'A', 'C', 'E']))

1. Changing the column name using df.columns

Ans.

import pandas as pd

df=pd.DataFrame({"Name":['Tom','Nick','John','Peter'],

"Age":[15,26,17,28]})

df.columns =['Col\_1', 'Col\_2']

1. Changing the row index using df.index attribute.

Ans.

import pandas as pd

df=pd.DataFrame({"Name":['Tom','Nick','John','Peter'],

"Age":[15,26,17,28]})

df.index = ['Row\_1', 'Row\_2', 'Row\_3', 'Row\_4']

df

1. iterate over rows in Pandas Dataframe

Ans.

import pandas as pd

input\_df = [{'name':'Sujeet', 'age':10},

{'name':'Sameer', 'age':11},

{'name':'Sumit', 'age':12}]

df = pd.DataFrame(input\_df)

for index, row in df.iterrows():

print(row['name'], row['age'])

1. Consider 2 dataframes and append df2 at the end of df1

Ans.

import pandas as pd

df1 = df = pd.DataFrame({"a":[1, 2, 3, 4],

"b":[5, 6, 7, 8]})

df2 = pd.DataFrame({"a":[1, 2, 3],

"b":[5, 6, 7]})

df1.append(df2)

1. Sort rows in pandas DataFrame

Ans.

import pandas as pd

data = {'name': ['Simon', 'Marsh', 'Gaurav', 'Alex', 'Selena'],

'Maths': [8, 5, 6, 9, 7],

'Science': [7, 9, 5, 4, 7],

'English': [7, 4, 7, 6, 8]}

df = pd.DataFrame(data)

a = df.sort\_values(by ='Science', ascending = 0)

print(a)

1. Selecting all the rows from the given dataframe in which ‘Percentage’ is greater than 80 using the basic method.

Ans.

import pandas as pd

record = {

'Name': ['Ankit', 'Amit', 'Aishwarya', 'Priyanka', 'Priya', 'Shaurya' ],

'Age': [21, 19, 20, 18, 17, 21],

'Stream': ['Math', 'Commerce', 'Science', 'Math', 'Math', 'Science'],

'Percentage': [88, 92, 95, 70, 65, 78] }

dataframe = pd.DataFrame(record, columns = ['Name', 'Age', 'Stream', 'Percentage'])

rslt\_df = dataframe[dataframe['Percentage'] > 80]

print(rslt\_df)

1. Load a CSV file into a Pandas DataFrame:

Ans.

import pandas as pd

df = pd.read\_csv('data.csv')

print(df.to\_string())

1. Create a DataFrame from dict of ndarrays:

Ans.

import pandas as pd

info = {'ID' :[101, 102, 103],'Department' :['B.Sc','B.Tech','M.Tech',]}

info = pd.DataFrame(info)

print (info)

**DATA CLEANING**

1. Check for Missing Values

Ans-

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',

'h'],columns=['one', 'two', 'three'])

df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

print df['one'].isnull()

2. Replace NaN with a Scalar Value

Ans-

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(3, 3), index=['a', 'c', 'e'],columns=['one',

'two', 'three'])

df = df.reindex(['a', 'b', 'c'])

print df

print ("NaN replaced with '0':")

print df.fillna(0)

3. Drop Missing Values

Ans-

import pandas as pd

import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',

'h'],columns=['one', 'two', 'three'])

df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

print df.dropna()

4. Replace Missing (or) Generic Values

Ans-

import pandas as pd

import numpy as np

df = pd.DataFrame({'one':[10,20,30,40,50,2000],

'two':[1000,0,30,40,50,60]})

print df.replace({1000:10,2000:60})

5. Handling Missing Data in Pandas

Ans-

import pandas as pd

import numpy as np

df = pd.DataFrame.from\_dict({

'Name': ['Nik', 'Kate', 'Evan', 'Kyra', np.NaN],

'Age': [33, 32, 40, 57, np.NaN],

'Location': ['Toronto', 'London', 'New York', np.NaN, np.NaN]

})

print(df.isnull().sum())

6.Dropping Missing Data in a Pandas DataFrame

Ans-

import pandas as pd

import numpy as np

df = pd.DataFrame.from\_dict({

'Name': ['Nik', 'Kate', 'Evan', 'Kyra', np.NaN],

'Age': [33, 32, 40, 57, np.NaN],

'Location': ['Toronto', 'London', 'New York', np.NaN, np.NaN]

})

df.dropna( axis=0, how='any', thresh=None, subset=None, inplace=False )

print(df)

7.Filling Missing Data in a Pandas DataFrame

Ans-

import pandas as pd

import numpy as np

df = pd.DataFrame.from\_dict({

'Name': ['Nik', 'Kate', 'Evan', 'Kyra', np.NaN],

'Age': [33, 32, 40, 57, np.NaN],

'Location': ['Toronto', 'London', 'New York', np.NaN, np.NaN]

})

df = df.fillna(0)

print(df)

8. Filling NA values in Columns with Different Values

Ans-

import pandas as pd

import numpy as np

df = pd.DataFrame.from\_dict({

'Name': ['Nik', 'Kate', 'Evan', 'Kyra', np.NaN],

'Age': [33, 32, 40, 57, np.NaN],

'Location': ['Toronto', 'London', 'New York', np.NaN, np.NaN]

})

df = df.fillna({'Name': 'Someone', 'Age': 25, 'Location': 'USA'})

print(df)

9. Identifying Duplicate Records in a Pandas DataFrame

Ans-

import pandas as pd

import numpy as np

df = pd.DataFrame.from\_dict({

'Name': ['Nik', 'Kate', 'Evan', 'Kyra', np.NaN],

'Age': [33, 32, 40, 57, np.NaN],

'Location': ['Toronto', 'London', 'New York', np.NaN, np.NaN]

})

print(df.duplicated())

10. Removing Duplicate Data in a Pandas DataFrame

Ans-

import pandas as pd

import numpy as np

df = pd.DataFrame.from\_dict({

'Name': ['Nik', 'Kate', 'Evan', 'Kyra', np.NaN],

'Age': [33, 32, 40, 57, np.NaN],

'Location': ['Toronto', 'London', 'New York', np.NaN, np.NaN]

})

df = df.sort\_values(by='Date Modified', ascending=False)

df = df.drop\_duplicates(subset=['Name', 'Age'], keep='first')

print(df)

11. Cleaning Strings in Pandas

Ans-

import pandas as pd

df = pd.DataFrame.from\_dict({

'Name': ['Tranter, Melvyn', 'Lana, Courtney', 'Abel, Shakti', 'Vasu, Imogene', 'Aravind, Shelly'],

'Region': ['Region A', 'Region A', 'Region B', 'Region C', 'Region D'],

'Location': ['TORONTO', 'LONDON', 'New york', 'ATLANTA', 'toronto'],

'Favorite Color': [' green ', 'red', ' yellow', 'blue', 'purple ']

})

df['Favorite Color'] = df['Favorite Color'].str.strip()

print(df)

12. Splitting Strings into Columns in Pandas

Ans-

import pandas as pd

df = pd.DataFrame.from\_dict({

'Name': ['Tranter, Melvyn', 'Lana, Courtney', 'Abel, Shakti', 'Vasu, Imogene', 'Aravind, Shelly'],

'Region': ['Region A', 'Region A', 'Region B', 'Region C', 'Region D'],

'Location': ['TORONTO', 'LONDON', 'New york', 'ATLANTA', 'toronto'],

'Favorite Color': [' green ', 'red', ' yellow', 'blue', 'purple ']

})

print(df['Name'].str.split(','))

13. Replacing Text in Strings in Pandas

Ans-

import pandas as pd

df = pd.DataFrame.from\_dict({

'Name': ['Tranter, Melvyn', 'Lana, Courtney', 'Abel, Shakti', 'Vasu, Imogene', 'Aravind, Shelly'],

'Region': ['Region A', 'Region A', 'Region B', 'Region C', 'Region D'],

'Location': ['TORONTO', 'LONDON', 'New york', 'ATLANTA', 'toronto'],

'Favorite Color': [' green ', 'red', ' yellow', 'blue', 'purple ']

})

df['Region'] = df['Region'].str.replace('Region ', '')

print(df)

14. Calculate the percentage of missing records in each column.

Ans-

import pandas as pd

df = pd.DataFrame.from\_dict({

'Name': ['Tranter, Melvyn', 'Lana, Courtney', 'Abel, Shakti', 'Vasu, Imogene', 'Aravind, Shelly'],

'Region': ['Region A', 'Region A', 'Region B', 'Region C', 'Region D'],

'Location': ['TORONTO', 'LONDON', 'New york', 'ATLANTA', 'toronto'],

'Favorite Color': [' green ', 'red', ' yellow', 'blue', 'purple ']

})

print(df.isnull().sum() / len(df))

15. Drop any duplicate records based only on the Name column, keeping the last record.

Ans-

import pandas as pd

df = pd.DataFrame.from\_dict({

'Name': ['Tranter, Melvyn', 'Lana, Courtney', 'Abel, Shakti', 'Vasu, Imogene', 'Aravind, Shelly'],

'Region': ['Region A', 'Region A', 'Region B', 'Region C', 'Region D'],

'Location': ['TORONTO', 'LONDON', 'New york', 'ATLANTA', 'toronto'],

'Favorite Color': [' green ', 'red', ' yellow', 'blue', 'purple ']

})

df = df.drop\_duplicates(subset='Name', keep='last')

**DATA MANIPULATION**

1. Selecting Pandas DataFrame rows using logical operators

Ans-

import pandas as pd

data = {'name':['Anthony', 'Maria'], 'age':[30, 28]}

df = pd.DataFrame(data)

# Selecting rows where age is over 20

df[df.age > 20]

# Selecting rows where name is not John

df[df.name != "John"]

# Selecting rows where age is less than 10

# OR greater than 70

df[(df.age < 10) | (df.age > 70)]

2. Pandas apply() function

Ans-

import pandas as pd

data = {'name':['Anthony', 'Maria'], 'age':[30, 28]}

df = pd.DataFrame(data)

def double(x):

return 2\*x

# Apply this function to double every value in a specified column

df.column1 = df.column1.apply(double)

# Lambda functions can also be supplied to `apply()`

df.column2 = df.column2.apply(lambda x : 3\*x)

# Applying to a row requires it to be called on the entire DataFrame

df['newColumn'] = df.apply(lambda row:

row['column1'] \* 1.5 + row['column2'],

axis=1

)

3. Pandas DataFrames adding columns

Ans-

import pandas as pd

data = {'name':['Anthony', 'Maria'], 'age':[30, 28]}

df = pd.DataFrame(data)

# Specifying each value in the new column:

df['newColumn'] = [1, 2, 3, 4]

# Setting each row in the new column to the same value:

df['newColumn'] = 1

# Creating a new column by doing a

# calculation on an existing column:

df['newColumn'] = df['oldColumn'] \* 5

4. Crosstab in pandas

Ans-

import pandas

import numpy

# creating some data

a = numpy.array(["foo", "foo", "foo", "foo",

"bar", "bar", "bar", "bar",

"foo", "foo", "foo"],

dtype=object)

b = numpy.array(["one", "one", "one", "two",

"one", "one", "one", "two",

"two", "two", "one"],

dtype=object)

c = numpy.array(["dull", "dull", "shiny",

"dull", "dull", "shiny",

"shiny", "dull", "shiny",

"shiny", "shiny"],

dtype=object)

# form the cross tab

pandas.crosstab(a, [b, c], rownames=['a'], colnames=['b', 'c'])

5. Merge DataFrames in pandas.

Ans-

import pandas as pd

left = pd.DataFrame({

'id':[1,2,3,4,5],

'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],

'subject\_id':['sub1','sub2','sub4','sub6','sub5']})

right = pd.DataFrame(

{'id':[1,2,3,4,5],

'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],

'subject\_id':['sub2','sub4','sub3','sub6','sub5']})

print(pd.merge(left,right,on='id'))

6. Outer Join, Inner Join, Left Join, Right Join in pandas.

Ans-

import pandas as pd

left = pd.DataFrame({

'id':[1,2,3,4,5],

'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],

'subject\_id':['sub1','sub2','sub4','sub6','sub5']})

right = pd.DataFrame(

{'id':[1,2,3,4,5],

'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],

'subject\_id':['sub2','sub4','sub3','sub6','sub5']})

print(pd.merge(left, right, on='subject\_id', how='left'))

print(pd.merge(left, right, on='subject\_id', how='right'))

print(pd.merge(left, right, on='subject\_id', how='outer'))

print(pd.merge(left, right, on='subject\_id', how='inner'))

7. Sorting DataFrames using pandas

Ans-

import pandas as pd

left = pd.DataFrame({

'id':[1,2,3,4,5],

'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],

'subject\_id':['sub7','sub8','sub4','sub6','sub5']})

print(left.sort\_values(by='subject\_id'))

8. Merging two tables

Ans-

marks = pd.DataFrame([100, 98, 91], index=['Student 1','Student 2','Student 3'],columns=['Subject 1'])

marks

marks\_2 = pd.DataFrame([92, 93, 99], index=['Student 1', 'Student 2', 'Student 3'],columns=['Subject 2'])

marks\_2

data\_merged = marks.merge(right=marks\_2, how='inner', left\_index=True, right\_index=True, sort=False)

print(data\_merged)

9. Finding spread of data across 2 categorical columns:

Ans-

import pandas as pd

left = pd.DataFrame({

'iD':[1,2,3,None,None],

'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],

'subject\_id':['sub7','sub8','sub4','sub6','sub5']})

pd.crosstab(left["Name"],left["subject\_id"],margins=True)

10. Locating specific rows

Ans-

import pandas as pd

left = pd.DataFrame({

'iD':[1,2,3,None,None],

'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],

'subject\_id':['sub7','sub8','sub4','sub6','sub5']})

# Use `iloc[]` to select row `0`

print(left.iloc[\_])

# Use `loc[]` to select column `'A'`

print(left.loc[:,'\_'])

**SEABORN**

1. Install Seaborn:

C:\Users\Your Name>pip install seaborn

1. Import Matplotlib:

import matplotlib.pyplot as plt

1. Import Seaborn:

import seaborn as sns

1. Plotting a Distplot:

import matplotlib.pyplot as plt  
import seaborn as sns  
  
sns.distplot([0, 1, 2, 3, 4, 5])  
  
plt.show()

1. Plotting a Distplot Without the Histogram

import matplotlib.pyplot as plt  
import seaborn as sns  
  
sns.distplot([0, 1, 2, 3, 4, 5], hist=False)  
  
plt.show()

Python Seaborn Plotting Functions

1. Barplot:

Res = sns.barplot(mtcars[‘cy1’],mtcars[‘carb’])

plot.show()

1. Countplot:

Sns.countplot(x=’cy1’, data=mtcars, pallete=”set1”)

1. Distribution Plot:

Sns.distplot(mtcars.mpg, bins = 10, color = ‘g’)

1. Heatmap:

Sns.heatmap(mtcars.corr(), cbar=True, linewidth = 0.5)

1. Load Data To Construct Seaborn Plots

import pandas

import matplotlib

import scipy

import seaborn as sns

print(sns.get\_dataset\_names())

Output:

Output: ['anagrams', 'anscombe', 'attention', 'brain\_networks', 'car\_crashes', 'diamonds', 'dots', 'exercise', 'flights', 'fmri', 'gammas', 'geyser', 'iris', 'mpg', 'penguins', 'planets', 'tips', 'titanic']

1. Whitegrid

from matplotlib import pyplot as plt

import seaborn as sns

plt.scatter(df.speeding,df.alcohol)

sns.set\_style("whitegrid")

plt.show()

1. remove the top and right axis spines using the despine() function.

from matplotlib import pyplot as plt

import seaborn as sns

plt.scatter(df.speeding,df.alcohol)

sns.set\_style("ticks")

sns.despine()

plt.show()

1. **Styling and Themes in Seaborn**

from matplotlib import pyplot as plt

import seaborn as sns

plt.scatter(df.speeding,df.alcohol)

plt.show()

1. style this plot using the set() function

from matplotlib import pyplot as plt

import seaborn as sns

plt.scatter(df.speeding,df.alcohol)

sns.set()

plt.show()

1. **Seaborn Color Palette**

sns.palplot(sns.color\_palette("deep", 10))

sns.palplot(sns.color\_palette("PiYG", 10))

sns.palplot(sns.color\_palette("GnBu", 10))

**VIVA QUESTIONS**

1. What is Python? List some popular applications of Python in the world of technology?  
   Python is a general-purpose coding language that is used across many web development and information technology jobs to complete a variety of programming tasks.Python is often used as a support language for software developers, for build control and management, testing, and in many other ways.
2. Which library would you prefer for plotting in Python language: Seaborn or Matplotlib?  
   Seaborn and Matplotlib are two of Python's most powerful visualization libraries. Seaborn uses fewer syntax and has stunning default themes and Matplotlib is more easily customizable through accessing the classes.
3. What is the main difference between a Pandas series and a single-column DataFrame in Python?  
   Series is a type of list which can take integer values, string values, double value and more. Series can only contain single list with index, whereas dataframes can be made of more than one series
4. Which method in pandas.tools.plotting is used to create scatter plot matrix?  
   The scatter plot matrix can be created by using DataFrame. plot. scatter() method.
5. Why you should use NumPy arrays instead of nested Python lists?  
   The NumPy arrays takes significantly less amount of memory as compared to python lists. It also provides a mechanism of specifying the data types of the contents, which allows further optimisation of the code.
6. Differentiate between List and Tuple?  
    A list doesn’t need to be always homogeneous- making it one of the most powerful tools used in Python. It exists as a type of container in the data structures in Python. We use it for storing multiple data and information at the very same time.   
   A tuple refers to a collection of various Python objects that stay separated by commas. A tuple is comparatively much faster than a list because it is static in nature.
7. Is Python a compiled language or an interpreted language?  
   Python is an interpreted language, which means the source code of a Python program is converted into bytecode that is then executed by the Python virtual machine.
8. What is the difference between Set and Dictionary?  
   A set is an unordered collection. A dictionary is an unordered collection of data that stores data in key-value pairs.
9. What is List Comprehension?   
   List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.
10. What is a lambda function?  
    Python Lambda function is known as the anonymous function that is defined without a name. Python allows us to not declare the function in the standard manner, i.e., by using the def keyword.
11. What is difference between / and // in Python?  
    Normal Division : Divides the value on the left by the one on the right. Notice that division results in a floating-point value.  
    divide=10/3 #Normal Division   
    print(divide)   
    OUTPUT : 3.333333333333333   
    Floor Division : Divides and returns the integer value of the quotient. It neglects the digits after the decimal.  
    divide=10//3 #Floor Division  
    print(divide)   
    OUTPUT : 3
12. What is Polymorphism in Python?  
    The word polymorphism means having many forms. In programming, polymorphism means the same function name (but different signatures) being used for different types.
13. How is Exceptional handling done in Python?  
    Try and except statements are used to catch and handle exceptions in Python. Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause.
14. Which python library is used for Machine Learning?  
    Scikit-learn is the most popular Python machine learning library for creating machine learning algorithms. It was created on top of two Python libraries – NumPy and SciPy. Scikit-learn is a Python library that provides a standard interface for supervised and unsupervised learning techniques.
15. Why python is used for Data analysis?  
    Python works well on every stage of data analysis. It is the Python libraries that were designed for data science that are so helpful. Data mining, data processing, and modeling along with data visualization are the 3 most popular ways of how Python is being used for data analysis.

NUMPY

1. What is Numpy?  
   Ans: NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. … A powerful N-dimensional array object. Sophisticated (broadcasting) functions.
2. Why NumPy is used in Python?  
   Ans: NumPy is a package in Python used for Scientific Computing. NumPy package is used to perform different operations. The ndarray (NumPy Array) is a multidimensional array used to store values of same datatype. These arrays are indexed just like Sequences, starts with zero.
3. What does NumPy mean in Python?
4. Ans: NumPy (pronounced /ˈnʌmpaɪ/ (NUM-py) or sometimes /ˈnʌmpi/ (NUM-pee)) is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
5. Where is NumPy used?  
   Ans: NumPy is an open source numerical Python library. NumPy contains a multi-dimentional array and matrix data structures. It can be utilised to perform a number of mathematical operations on arrays such as trigonometric, statistical and algebraic routines. NumPy is an extension of Numeric and Numarray.
6. How to import numpy in python?

Ans: import numpy as np

1. What Is The Difference Between Numpy And Scipy?  
   NumPy stands for Numerical Python while SciPy stands for Scientific Python. Both NumPy and SciPy are modules of Python, and they are used for various operations of the data. Coming to NumPy first, it is used for efficient operation on homogeneous data that are stored in arrays. In other words, it is used in the manipulation of numerical data.
2. List the advantages NumPy Arrays have over (nested) Python lists  
   Size - Numpy data structures take up less space  
   Performance - they have a need for speed and are faster than lists  
   Functionality - SciPy and NumPy have optimized functions such as linear algebra operations built in.
3. What is difference between NumPy and Pandas?  
   The Pandas module mainly works with the tabular data, whereas the NumPy module works with the numerical data.

* The Pandas provides some sets of powerful tools like DataFrame and Series that mainly used for analyzing the data, whereas in NumPy module offers a powerful object called Array.
* The performance of NumPy is better than the NumPy for 50K rows or less.
* The performance of Pandas is better than the NumPy for 500K rows or more. Between 50K to 500K rows, performance depends on the kind of operation.
* NumPy library provides objects for multi-dimensional arrays, whereas Pandas is capable of offering an in-memory 2d table object called DataFrame.
* NumPy consumes less memory as compared to Pandas.
* Indexing of the Series objects is quite slow as compared to NumPy arrays.

PANDAS

1. Mention the different types of Data Structures in Pandas?  
   Pandas provide two data structures, which are supported by the pandas library, Series, and DataFrames. Both of these data structures are built on top of the NumPy.  
   A Series is a one-dimensional data structure in pandas, whereas the DataFrame is the two-dimensional data structure in pandas.
2. Define Series in Pandas?  
   A Series is defined as a one-dimensional array that is capable of storing various data types. The row labels of series are called the index. By using a 'series' method, we can easily convert the list, tuple, and dictionary into series. A Series cannot contain multiple columns.
3. Define DataFrame in Pandas?  
   A DataFrame is a widely used data structure of pandas and works with a two-dimensional array with labeled axes (rows and columns) DataFrame is defined as a standard way to store data and has two different indexes, i.e., row index and column index. It consists of the following properties:

* The columns can be heterogeneous types like int and bool.
* It can be seen as a dictionary of Series structure where both the rows and columns are indexed. It is denoted as "columns" in the case of columns and "index" in case of rows.

1. What are the significant features of the pandas Library?   
   The key features of the panda's library are as follows:

* Memory Efficient
* Data Alignment
* Reshaping
* Merge and join
* Time Series

1. What is the name of Pandas library tools used to create a scatter plot matrix?

Scatter\_matrix

1. Explain Categorical data in Pandas?

A Categorical data is defined as a Pandas data type that corresponds to a categorical variable in statistics. A categorical variable is generally used to take a limited and usually fixed number of possible values. Examples: gender, country affiliation, blood type, social class, observation time, or rating via Likert scales. All values of categorical data are either in categories or np.nan.

1. How to iterate over a Pandas DataFrame?  
   You can iterate over the rows of the DataFrame by using for loop in combination with an iterrows() call on the DataFrame.

**DATA CLEANING**

### **What is the difference between data mining and data analysis?**

| Data Mining | Data Analysis |
| --- | --- |
| It refers to the process of identifying patterns in a pre-built database. | It is used to order and organize raw data in a meaningful manner. |
| Data mining is done on clean and well-documented data. | Data analysis involves cleaning the data hence it is not presented in a well-documented format. |
| The outcomes are not easy to interpret. | The outcomes are easy to interpret. |
| It is mostly used for Machine Learning where used to recognize the patterns with the help of algorithms. | It is used to gather insights from raw data, which has to be cleaned and organized before performing the analysis. |

1. What is the Difference between Data Profiling and Data Mining?  
   Data Profiling: It refers to the process of analyzing individual attributes of data. It primarily focuses on providing valuable information on data attributes such as data type, frequency, length, occurrence of null values.  
   Data Mining: It refers to the analysis of data with respect to finding relations that have not been discovered earlier. It mainly focuses on the detection of unusual records, dependencies and cluster analysis.
2. What is the Process of Data Analysis?Data analysis is the process of collecting, cleansing, interpreting, transforming, and modeling data to gather insights and generate reports to gain business profits. Refer to the image below to know the various steps involved in the process.  
   Collect Data: The data is collected from various sources and stored to be cleaned and prepared. In this step, all the missing values and outliers are removed.  
   Analyse Data: Once the data is ready, the next step is to analyze the data. A model is run repeatedly for improvements. Then, the model is validated to check whether it meets the business requirements.  
   Create Reports: Finally, the model is implemented, and then reports thus generated are passed onto the stakeholders.
3. What is Data Wrangling or Data Cleansing/Cleaning?  
   Data Cleansing is the process of identifying and removing errors to enhance the quality of data. We must check for the following things and correct where needed:

Are all variables as expected (variables names & variable types).

Are there some variables that are unexpected?

Are the data types and length across variables correct?

For known variables, is the data type as expected (For example if age is in date format something is  suspicious)

Have labels been provided and are sensible?

If anything suspicious we can further investigate it and correct it accordingly.What are Some of the Challenges You Have Faced during Data Analysis?

### Poor quality of data, with lots of missing and erroneous values

### Lack of understanding of the data, variables, and availability data dictionary

### Unrealistic timelines and expectation from the business stakeholders

### Challenge in blending/ integrating the data from multiple sources, particular when there no consistent parameters and conventions

### Wrong selection of tools and data architecture to achieve analytics goals in a timely manner

### What is VLOOKUP? VLOOKUP stands for ‘Vertical Lookup’. It is a function that makes Excel search for a certain value in a column (or the ‘table array’), in order to return a value from a different column in the same row.

### **What is a Pivot Table, and What are the Different Sections of a Pivot Table?** A Pivot Table is used to summarise, sort, reorganize, group, count, total or average data stored in a table. It allows us to transform columns into rows and rows into columns. It allows grouping by any field (column) and using advanced calculations on them.

### A Pivot table is made up of four different sections:

### Values Area: Values are reported in this area

### Rows Area: The headings which are present on the left of the values.

### Column Area: The headings at the top of the values area make the columns area.

Filter Area: This is an optional filter used to drill down in the data set.

1. What is Conditional Formatting? How can it be used?  
   A conditional format changes the appearance of cells based on conditions that you specify. If the conditions are true, the cell range is formatted; if the conditions are false, the cell range is not formatted.
2. What is the Difference Between Mean, Median, and Mode?   
   Mean (or average) is the numerical value of the center of distribution and used when the data is concentrated)  
   Median (also known as the 50th percentile) is the middle observation in a data set. Median is calculated by sorting the data, followed by the selection of the middle value. The median of a data set has an odd number of  observations is the observation number [N + 1] / 2.  
   For data sets having an even number of observations, the median is midway between N / 2 and [N / 2] + 1. N is the number of observations.  
   A mode is a value that appears most frequently in a data set. A data set may have single or multiple modes, referred to as unimodal, bimodal, or trimodal, depending on the number of modes.

### **What is the Difference Between Covariance and Correlation?** Covariance measures the variance of the variable with itself, and correlation measures the strength and direction of a linear relationship between two or more variables. A correlation between two variables doesn’t imply that the change in one variable is the cause of the change in the other variable’s values.

1. How do you identify missing values?  
   The function used to identify the missing value is through .isnull()  
   The code below gives the total number of missing data points in the data frame  
   missing\_values\_count = sf\_permits.isnull().sum()

**SEABORN AND MATPLOT**

1. What is Seaborn?  
   Seaborn is an open source, Python data visualisation library built on matplotlib that is tightly integrated with pandas data structures. The core component of Seaborn is visualisation, which aids in data exploration and understanding.Data can be represented as plots, which are simple to study, explore, and interpret.
2. What is Matplotlib?  
   Matplotlib is a plotting library for Python with NumPy, the Python numerical mathematics extension. It offers an object-oriented API for embedding plots into applications utilising GUI toolkits such as Tkinter, wxPython, Qt, or GTK.
3. Does Seaborn need Matplotlib?  
   The only library we need to import is Seaborn. Seaborn's plots are drawn using matplotlib behind the scenes. Seaborn is a library that uses Matplotlib underneath to plot graphs.  
   Seaborn helps in resolving the two major issues faced by Matplotlib;   
   the problems are:  
   Default Matplotlib parameters  
   Working with data frames
4. What is CMAP in Seaborn?  
   Sequential Palette : one color only

With the cmap option of the heatmap() method in seaborn, you can adjust the colours of your heatmap. The vmax and vmin parameters in the function can also be used to establish maximum and minimum values for the colour bar on a seaborn heatmap.

1. What is the Seaborn function for colouring plots?  
   color\_palette() is a Seaborn function that can be used to give colours to plots and give them additional artistic appeal.
2. What is Histograms in Seaborn?  
   Histograms show the distribution of data by constructing bins throughout the data's range and then drawing bars to show how many observations fall into each bin.
3. How do you plot a histogram in Seaborn?  
   We can plot a histogram in Seaborn by using histplot() to plot a histogram with a density plot.
4. How to change the legend font size of FacetGrid plot in Seaborn?  
   We can access the legend from the FacetGrid in which sns.displot will return with FacetGrid.legend.
5. How do I make all of the lines in seaborn.lineplot black?  
   We can set the line colour to black using the line graph as given in the official reference as an example. When there is only one hue, though, it is impossible to distinguish the data. For visualisation, it may be preferable to utilise a single colour tone.
6. How to color the data points by a category like to assign colors to the 'regional indicators'?  
   The simplest solution is to choose the columns and use.melt to reshape them into a long dataframe.  
   Then use both together sns.lmplot and sns.regplot.  
   Hue can be used to define colours based on region, however this results in a separate regression line for each data point, rather than one for all data points, and as such the regression line is not shown for.lmplot, but is plotted separately for each axis with.regplot.

seaborn is a high-level API for matplotlib.

DATA MANIPULATIONS

1. How are missing values imputed using Pandas  
   ‘fillna()’ does it in one go. It is used for updating missing values with the overall mean/mode/median of the column.
2. What is a Crosstab in pandas?  
   The pandas crosstab function builds a cross-tabulation table that can show the frequency with which certain groups of data appear.
3. How do you merge two DataFrames in pandas?  
   Dataframes in Pandas can be merged using pandas. merge() method. Returns : A DataFrame of the two merged objects. The concat() function can be used to concatenate two Dataframes by adding the rows of one to the other.
4. How do you sort data frames in pandas?  
   In order to sort the data frame in pandas, function sort\_values() is used. Pandas sort\_values() can sort the data frame in Ascending or Descending order.
5. What is boolean indexing in Python pandas?  
   In boolean indexing, we will select subsets of data based on the actual values of the data in the DataFrame and not on their row/column labels or integer locations. In boolean indexing, we use a boolean vector to filter the data.
6. What is Apply function in pandas?  
   Pandas. apply allow the users to pass a function and apply it on every single value of the Pandas series. It comes as a huge improvement for the pandas library as this function helps to segregate data according to the conditions required due to which it is efficiently used in data science and machine learning.
7. What is pivot table in pandas?  
   Pivot table in pandas is an excellent tool to summarize one or more numeric variable based on two other categorical variables. Pivot tables in pandas are popularly seen in MS Excel files. In python, Pivot tables of pandas dataframes can be created using the command: pandas. pivot\_table .
8. What is multiple indexing pandas?  
   The MultiIndex object is the hierarchical analogue of the standard Index object which typically stores the axis labels in pandas objects. You can think of MultiIndex as an array of tuples where each tuple is unique. A MultiIndex can be created from a list of arrays (using MultiIndex).
9. How does pandas encode categorical data?  
   A technique called "label encoding", which allows you to convert each value in a column to a number. Numerical labels are always between 0 and n\_categories-1. You can do label encoding via attributes . cat.
10. How do I iterate over rows in a Pandas DataFrame?  
    DataFrame. iterrows() method is used to iterate over DataFrame rows as (index, Series) pairs. Note that this method does not preserve the dtypes across rows due to the fact that this method will convert each row into a Series .