**PRIYANSH SALIAN**

**2003148**

**C31**

## Experiment No: 12

**Aim :** Pandas in Python

**Question 1-**

Write a pandas program to:

i) add, subtract, multiple and divide two pandas series ii) compare the elements ofthe two Pandas Series. iii) convert a dictionary to a Pandas series. iv) convert a NumPy array to a Pandas series.

**Function Used**:

1. **Pandas:** Pandas is a Python package that provides fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal. pandas is well suited for many different kinds of data:

> Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet

> Ordered and unordered (not necessarily fixed-frequency) time series data.

>Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels

>Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

1. **Pandas Series:** Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index.

Pandas Series is nothing but a column in an excel sheet. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

1. **Operations on Series:** Basic arithmetic operations like addition, subtraction, multiplication, and division on two Pandas Series can be performed.

Perform the required arithmetic operation using the respective arithmetic operator between the two Series

![image.png](attachment:image.png)

Result can be assigned the to another Series.

Similarly Relation Operators can be used to compare two Series. The result is obtained as a new series with boolean values by element to element comparison

![image-2.png](attachment:image-2.png)

![image-3.png](attachment:image-3.png)

1. **Numpy:** NumPy is a Python library used for working with arrays.It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

**1)**

**Code**:

import pandas as pd

s1=pd.Series({'one':6,'two':2,'three':3},index=['one','two','four']) s2=pd.Series({'six':4,'two':9,'one':7}) print(f'Series s1 :\n',s1)

print(f'\nSeries s2 :\n',s2)

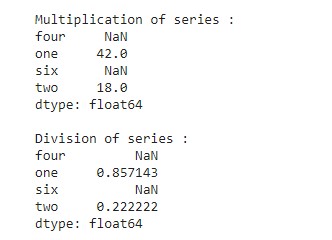
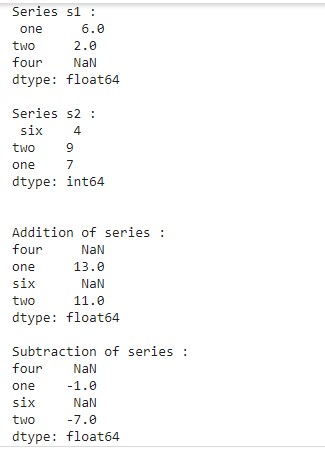
#Arithmetic operations on series

s3=s1+s2 s4=s1-s2 s5=s1\*s2

s6=s1/s2print(f'\n\nAddition of series :\n{s3}') print(f'\nSubtraction of series :\n{s4}') print(f'\nMultiplication of series :\n{s5}')

print(f'\nDivision of series :\n{s6}')

**Output:**



**2)**

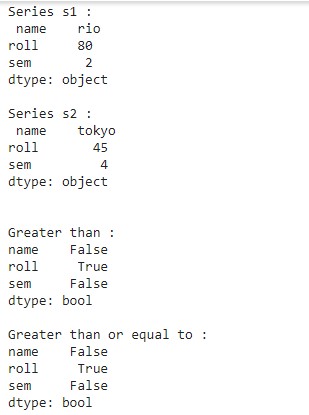
**Code:**

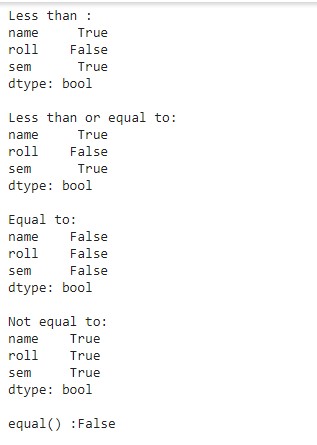
s1=pd.Series({'name':'rio','roll':80,'sem':'2'}) s2=pd.Series({'name':'tokyo','roll':45,'sem':'4'}) print(f'Series s1 :\n',s1) print(f'\nSeries s2 :\n',s2) #Comparision operators s3=s1>s2

s4=s1>=s2 s5=s1<s2 s6=s1<=s2 s7=s1==s2 s8=s1!=s2 s9=s1.equals(s2) s10=s1.compare(s2)

print(f'\n\nGreater than :\n{s3}') print(f'\nGreater than or equal to :\n{s4}') print(f'\nLess than :\n{s5}') print(f'\nLess than or equal to:\n{s6}') print(f'\nEqual to:\n{s7}') print(f'\nNot equal to:\n{s8}') print(f'\nequal() :{s9}') print(f'\ncompare():\n{s10}')

**Output:**





**3)**

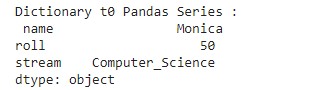
**Code:**

#Converting dictionary to pandas series

details=dict(name='Monica', roll=55, stream='Computer\_Science') s1=pd.Series(details)

print(f'Dictionary t0 Pandas Series :\n',s1)

**Output:**



**4)**

**Code:**

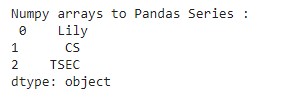
import numpy as np

#Converting numpy arrays to pandas series nd\_arr=np.array(['Lily',"CS",'TSEC'])

s1=pd.Series(nd\_arr)

print(f'Numpy arrays to Pandas Series :\n',s1)

**Output:**



**Question2**-

Write a program to read csv file in a dataframe, replace missing values with anyvalue, drop the row if all values are missing or contain null values.

**Function Used:**

1. **Pandas DataFrame:** Pandas DataFrame is two-dimensional sizemutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.

1. **CSV file:** A comma-separated values (CSV) file is a plaintext file with a .csv extension that holds tabular data. This is one of the most popular file formats for storing large amounts of data. Each row of the CSV file represents a single table row. The values in the same row are by default separated with commas, but you could change the separator to a semicolon, tab, space, or some other character.

1. **read\_csv():** Pandas read\_csv() function imports a CSV file to DataFrame format. header: this allows you to specify which row will be used as column names for your dataframe. Expected an int value or a list of int values.

Default value is header=0 , which means the first row of the CSV file will be treated as column names.

1. **head():**The head() function is used to get the first n rows.

This function returns the first n rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

1. **isnull():** isnull() function detect missing values in the given series object. It return a boolean same-sized object indicating if the values are NA. Missing values gets mapped to True and non-missing value gets mapped to False .

1. **fillna():** DataFrame.fillna() method fills(replaces) NA or NaN values in the DataFrame with the specified values. fillna() method can be used to fill NaN values in the whole DataFrame, or specific columns, or modify inplace, or limit on the number of fillings, or choose an axis along which filling has to take place etc.

1. **dropna():** Pandas dropna() method allows the user to analyze and drop Rows/Columns with Null values in different ways. Parameters: axis: axis takes int or string value for rows/columns. Input can be 0 or 1 for Integer and 'index' or 'columns' for String.

**Code:**

import pandas as pd import numpy as np #read csv file

df=pd.read\_csv('flights\_data.csv') df=df.head(50)

print("\033[1m Dataframe of flights\_data.csv file:::: \033[0m") print(df)

#finding all the values with NAN

print('\n\n\033[1m Boolean Dataframe of movies.csv file having values

NAN :::: \033[0m')

print(df.isnull())

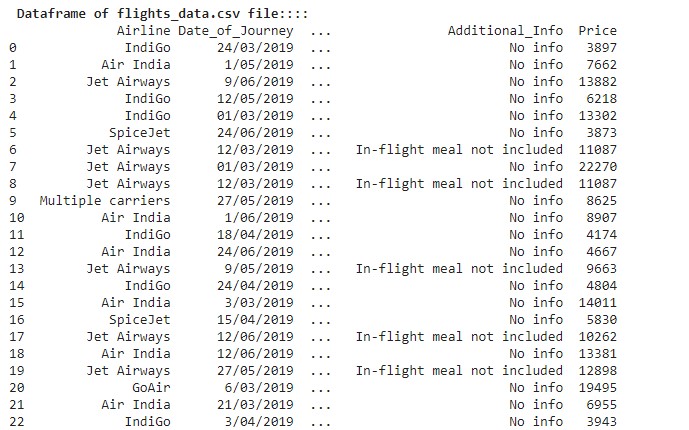
#filling all NAN values with 999999

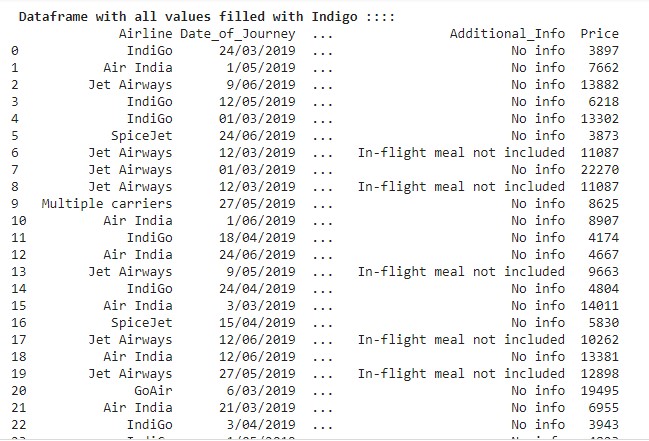
print('\033[1m Dataframe with all values filled with Indigo ::::\033[0m') print(df.fillna('Indigo'))

#Droping all the rows with NAN values

print('\n\n \033[1m Dataframe neglecting all rows with NAN :::: \033[0m') print(df.dropna())

**Output:**





**Question 3-**

Write a program to demonstrate merging of Frames: i) on the basis of id

ii) using how

**Function Used:**

**1. Merge DataFrames:** Pandas DataFrame merge() function is used to merge two DataFrame objects with a database-style join operation.  **merge() arguments**-

Pandas provides a single function, merge, as the entry point for all standard database join operations between DataFrame objects − pd.merge(left, right, how='inner', on=None, left\_on=None, right\_on=None, left\_index=False, right\_index=False, sort=True) Here, we have used the following parameters −

**left** − A DataFrame object.

**right** − Another DataFrame object.

**on** − Columns (names) to join on. Must be found in both the left and right DataFrame

**objects.left\_on** − Columns from the left DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame.

**right\_on** − Columns from the right DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame.

**left\_index** − If True, use the index (row labels) from the left DataFrame as its join key(s). In case of a DataFrame with a MultiIndex

(hierarchical), the number of levels must match the number of join keys from the right DataFrame.

**right\_index** − Same usage as left\_index for the right DataFrame.

**how** − One of 'left', 'right', 'outer', 'inner'. Defaults to inner. Each method has been described below.

**sort** − Sort the result DataFrame by the join keys in lexicographical order. Defaults to True, setting to False will improve the performance substantially in many cases.

**1)**

**Code:**

import pandas as pd df\_left=pd.DataFrame({

'Id':[79,78,77,76,75],

'Name':['Isha','Aanchal','Nishita','Laveena','Muskan'],

'Subject':['Python','Java','App\_Dev','AOA','Web\_Dev']

})

#df\_left=df\_left.set\_index("Id") df\_right=pd.DataFrame({

'Id':[79,78,77,76,75],

'Name':['Dash','Lily','Noah','Sara','Joe'], 'Subject':['Gamer','Python','App\_Dev','AOA','PM']

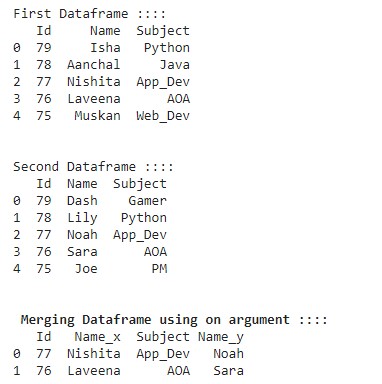
})

#df\_right=df\_right.set\_index('Id') print(f'First Dataframe :::: \n{df\_left}') print(f'\n\nSecond Dataframe :::: \n{df\_right}')

#using on=''

print(f'\n\n\033[1m Merging Dataframe using on argument ::::\033[0m') print(pd.merge(df\_left,df\_right,on=['Id','Subject']))

**Output:**



**2)**

**Code:**

#using how='outer'

print(f'\033[1m Merging Dataframe using how="outer" argument ::::\033[0m') print(pd.merge(df\_left,df\_right,on='Subject',how='outer'))

#using how='inner'

print(f'\n\n\033[1m Merging Dataframe using how="inner" argument ::::\033[0 m')

print(pd.merge(df\_left,df\_right,on='Subject',how='inner'))

#using how='left'

print(f'\n\n\033[1m Merging Dataframe using how="left" argument :::\033[0m') print(pd.merge(df\_left,df\_right,on='Subject',how='left'))

#using how='right'

print(f'\n\n\033[1m Merging Dataframe using how="right" argument ::::\033[0m'

)

print(pd.merge(df\_left,df\_right,on='Subject',how='right'))

**Output:**

