PANDAS

Pandas is an open-source library for data manipulation in Python. It provides data structures for efficiently storing and manipulating large and complex datasets. The datastructures are

* Series
* Data Frame.

A Series is a one-dimensional labeled array that can hold any data type such as integers, floats, strings, and even Python objects.

S=pd.Series([1,1,2,2,3,3,3,4,4,4,4]) //The data is given as a list

Data ={‘NewYork’:8623000,’LosAngeles’:4000000,’Chicago’:2700000,’Houston’:2300000}

Population=pd.Series(data) //The data is given as a dictionary

In a nutshell, Series is a pandas object with a single column only.

A DataFrame is a two-dimensional labeled data structure that can hold data of different types. It is like a table in a relational database or a spreadsheet in Excel.

Df= pd.DataFrame() is the function to be used to create a dataframe. Now is the interesting part, the data given inside the parenthesis can be

* A dictionary
* A comma separated file(csv)
* An excel file

\*\* One important caution when we type the above code, we are likely to get an error, hence it is very essential to import the pandas library before hand.

Import pandas as pd //Here we are giving an alias name pd henceforth we will be using alias name in our entire code.

Let’s see the creation of a dataframe using a dictionary:

import pandas as pd //importing the pandas library & creating an alias pd

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],

'Age': [25, 30, 35, 40],

'Salary': [50000, 60000, 70000, 80000]}

df = pd.DataFrame(data) //creating a dataframe using the dictionary

Now let’s see the creation of a dataframe using csv,excel:

With a prerequisite that the data is already available with commas in a notepad or an excel all we have to do is read the file from the location.

Df =Pd.read\_csv(“Please enter the location of the csv file”)

Df=Pd.read\_xlsx(“Please enter the location of the excel file”)

Then,convert it into dataframe

Df1= pd.DataFrame(Df) //don’t forget to include the pandas library

Simple operations on a series:

lst =['A','B','C']

lst2 = ['a','b','c']

Print(pd.Series(lst,lst2))

Output:

a A //so the first parameter is taken as data and the second parameter is taken as index

b B

c C

dtype: object

Print(pd**.**Series(data **=** lst,index**=**lst2))

Output:

a A // we had directly specified the data and index

b B

c C

dtype: object

d **=** {'a':'A','b':'B','c':'C'}

Print(pd**.**Series(d))

Output:

a A //In a dictionary key is taken as index, values are taken as the data

b B

c C

fam **=** ['raja','rajeshwari','dhivya','dinesh']

s1 **=** pd**.**Series(fam,index **=**[1,2,3,4])

S1

Output:

1 raja  
2 rajeshwari  
3 dhivya  
4 dinesh

g **=** ['ragul','sound','dinesh','surya']

s2 **=** pd**.**Series(g)

s2  
Output:

0 ragul  
1 sound  
2 dinesh  
3 surya  
dtype: object

g **=** pd**.**Series([1,2,3,4],index **=** ['ragul','sound','dinesh','surya'])

G

Output:

ragul 1  
sound 2  
dinesh 3  
surya 4  
dtype: int64

g1 **=** pd**.**Series([1,2,3,4],index **=** ['ragul','sound','dinesh','surya'])

g1  
Output:

ragul 1  
sound 2  
dinesh 3  
surya 4  
dtype: int64

g2 **=** pd**.**Series([1,2,4,5],index **=** ['ragul','vicky','dinesh','surya'])  
g2  
Output:

ragul 1  
vicky 2  
dinesh 4  
surya 5  
dtype: int64

gf1**+**gf2

Output:

dinesh 7.0 // This is the most interesting when you add two series the result is the common columns alone are added remaining gets a NAN value.

ragul 2.0  
sound NaN  
surya 9.0  
vicky NaN  
dtype: float64

In a nutshell, the series function can have syntax as:

Pd.Series(lst or lst=[1,2,3,4] or [1,2,3,4],lst or index= lst1 or index=[1,2,3,4])

NEED OF PANDAS:

* Provides a wide range of functions and methods for data manipulation such as filtering, sorting, grouping, merging, reshaping, and aggregating data. These operations can be performed on individual columns or rows, or on the entire DataFrame.
* Ability to handle missing or incomplete data. It provides various functions for detecting, filling, and removing missing values from the data.
* Ability to handle time-series data. It provides a range of functions for working with time-series data such as resampling, shifting, and rolling windows.
* provides tools for data visualization, which makes it easy to generate high-quality plots and charts from data stored in a DataFrame or a Series.
* user-friendly interface and extensive documentation make it an ideal choice for data analysts and scientists working with large and complex datasets.

WHY NOT SQL:

* MySQL requires a significant amount of programming to perform data manipulation tasks and may not be as user-friendly as Python, especially for non-technical users.