PANDAS (PANel DAta)

high-level data manipulation tool used for analysing data

very easy to import and export data using Pandas library

Pandas has three important data structures, namely – Series, DataFrame and Panel to make the process of

analysing data organised, effective and efficient.

1. A Numpy array requires homogeneous data, while a Pandas DataFrame can have different data types (float, int, string, datetime, etc.).
2. Pandas have a simpler interface for operations like file loading, plotting, selection, joining, GROUP BY, which come very handy in data-processing applications.
3. Pandas DataFrames (with column names) make it very easy to keep track of data.
4. Pandas is used when data is in Tabular Format, whereas Numpy is used for numeric array based data manipulation.

A data structure is a collection of data values and operations that can be applied to that data. It enables efficient storage, retrieval and modification to the data.

In [ ]:

```
#A Series is a one-dimensional array containing a sequence of values of any data type (
#string, etc) which by default have numeric data labels starting from zero.

#The data label associated with a particular value is called its index.
```

In [ ]:

```
import pandas as pd
import numpy as np
s=pd.Series(dtype=object)
print(s)
```

In [ ]:

```
#Creation of Series from Scalar Values

import pandas as pd
```

```
s=pd.Series([10,20,30])
print(s)
print(s.values)
print(s.index)
```

In [ ]:

```
s=pd.Series(["rama","sita","ravana","hanuma"],index=[4,1,2,3])
print(s)
```

In [ ]:

```
s=pd.Series(["rama","sita","ravana","hanuma"],index=['a','z','f','k'])
print(s)
```

In [ ]:

```
#Creation of Series from NumPy Arrays
import numpy as np
import pandas as pd
a=np.array([10,20,30])
s=pd.Series(a)
print(s)
```

In [ ]:

```
import numpy as np
import pandas as pd
a=np.array([10,20,30])
s=pd.Series(a,index=('xx','yy','zz','kk'))
print(s)
```

In [ ]:

```
dict = {'India': 'NewDelhi', 'UK':'London', 'Japan': 'Tokyo'}
s=pd.Series(dict)
print(s)
```

In [ ]:

```
# Accessing Elements of a Series
#Indexing
s=pd.Series([11,22,55,33,44,88,66,77])
print(s[4],s[1])

s=pd.Series([10,20,30,40,50],index=('j','g','r','p','l'))
print(s['l'],s['g'])
print(s[['j','p']])
print(s[3])
```

In [ ]:

```
#Slicing
#extract a part of a series
#When we use positional indices for slicing, the value at the endindex position is excl

scapitals = pd.Series(['NewDelhi', 'WashingtonDC', 'London',
'Paris'], index=['India', 'USA', 'UK', 'France'])
print(scapitals[1:3])
print(scapitals['USA':'France'])
print(scapitals[-3:-1])
```

In [ ]:

```
#If labelled indexes are used for slicing, then value at the end index label is also in
scapitals = pd.Series(['NewDelhi', 'WashingtonDC', 'London',
'Paris'], index=['India', 'USA', 'UK', 'France'])
print(scapitals['USA':'France'])
```

In [ ]:

```
#slicing to modify the values
import numpy as np
s = pd.Series(np.arange(10,16,1),index = ['a', 'b', 'c', 'd', 'e', 'f'])
print(s)
s['c':'e']=99
print(s)
```

In [ ]:

```
#Attributes of Series
#name         --  assigns a name to the Series
#index.name   --   assigns a name to the index of the series
#values prints a list of the values in the series
#size prints the number of values in the Series object
#empty prints True if the series is empty, and False otherwise

a=np.array([10,20,30])
s=pd.Series(a)
s.name='Data Analysis'
print(s,'\n\n')

s.index.name=' naturals'
print(s)

print(s.values)
print(s.index)
print(s.size)
print(s.empty)
```

In [ ]:

```
#Methods of Series
a=np.array([10,20,30,40,50,60,70])
s=pd.Series(a)
print(s.head())
print(s.head(3))

print(s.count())
print(s.tail())
```

In [ ]:

```
#Mathematical Operations on Series
#Addition - two ways
s1= pd.Series([1,2,3,4,5], index = ['a', 'b', 'c', 'd', 'e'])
s2 = pd.Series([10,20,-10,-50,100],index = ['z', 'y', 'a', 'c', 'e'])

# Addition of two Series
print(s1,s2)
print(s1+s2)
```

In [ ]:

```
#second method is applied when we do not want to have NaN values in the output.
# use the series method add() and a parameter fill_value to replace missing value with
```

```
s1= pd.Series([1,2,3,4,5], index = ['a', 'b', 'c', 'd', 'e'])
s2 = pd.Series([10,20,-10,-50,100],index = ['z', 'y', 'a', 'c', 'e'])
print(s1.add(s2,fill_value=0))
```

In [ ]:

```
#Subtraction of two Series
#two ways
#replace the missing values with 1000 before subtracting seriesB from seriesA using exp
#subtraction method sub().
print(s1.sub(s2,fill_value=1000))

#Multiplication of two Series
#Two ways
print(s1*s2)
#replace the missing values with 0 before multiplication of seriesB with seriesA using
#multiplication method mul().
print(s1.mul(s2,fill_value=0))

#Division of two Series
print(s1/s2)
print(s1.div(s2,fill_value=1))
```

In [ ]:

```
#String methods
s = pd.Series(["A", "B",
    "C", "Aaba", "Baca", np.nan, "CABA", "dog", "cat"], dtype="string")
print(s.str.lower())
print(s.str.upper())
print(s.str.len())

idx=pd.Index(["  jack","jill   ","jesse   ","   frank"])
print(idx)
print(idx.str.rstrip())
print(idx.str.lstrip())
print("\n\n")
idx=pd.Index(["  jack  ","  jill   ","jesse","  frank"])
print(idx)
print(idx.str.strip())
```

In [ ]:

```
#Splitting
s = pd.Series(["a_b_c", "c_d_e", np.nan, "f_g_h"], dtype="string")
print(s.str.split("_"))

#Elements in the split lists can be accessed using get or [] notation:
print(s.str.split("_").str.get(1))
```

In [ ]:

```
#Concatenating a single Series into a string
s = pd.Series(["a", "b", "c", "d"], dtype="string")
print(s.str.cat(sep=","))
print(s.str.cat())

s1 = pd.Series(["a", "b", np.nan, "d"], dtype="string")
print(s1.str.cat(sep=",", na_rep="-"))
```

In [ ]:

```
#Concatenating a Series and something list-like into a Series
print(s.str.cat(["A", "B", "C", "D"]))
```

In [ ]:

```
import pandas as pd
s=pd.Series([10,20,30])
print(s)
print(s.ndim)
print(s.size)
print(s.array)
print(s.nbytes)
print(s.shape)
print(s.dtype)
print(s.memory_usage)
print(s.name)
print(s.flags)
```

In [ ]:

```
print(s.item())
print(s)
print(s.pop(1))
print(s)
```

In [ ]:

In [ ]: