PYTHON-PANDAS

--------------------------

pandas is open source python library

It provides high-performance,easy to use data structure and data analysis tools for python programming language

python with pandas is used in wide range of fields including academic and commercial domains including finance,economics,statistics,analytics etc...

Introduction to data structure:

-------------------------------

Pandas deals with the following three data structures

Series

DataFrame

Panel

Series:

---------

1D homogeneous array

size immutable

Data Frames

-----------

DataFrame is widely used and one of the most important data structures. Panel is used much less.

2D heterogeneous array

size mutable

Panel

-----------

Panel is a three-dimensional data structure with heterogeneous data.

It is hard to represent the panel in graphical representation.

But a panel can be illustrated as a container of DataFrame.

3D array,size mutable

SERIES:

-------

syntax:

-------

pandas.Series( data, index, dtype, copy)

data-data takes various forms like ndarray, list, constants

index-ndex values must be unique and hashable, same length as data. Default np.arrange(n) if no index is passed.

dtype-dtype is for data type

copy-Copy data

Series can be created using various input like:

\*Array

\*Dict

\*Scalar or constant

create an empty series:

-----------------------

import pandas as pd

s = pd.Series()

print(s)

o/p:

-----

Warning (from warnings module):

File "C:/Users/ELCOT/Desktop/pandass.py", line 2

s = pd.Series()

DeprecationWarning: The default dtype for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this warning.

Series([], dtype: float64)

creating a series from ndarray:

-------------------------------

import pandas as pd

import numpy as np

data = np.array(['a','b','c','d'])

s = pd.Series(data)

print(s)

o/p:

----

0 a

1 b

2 c

3 d

dtype: object

import pandas as pd

import numpy as np

data = np.array(['a','b','c','d'])

s = pd.Series(data,index=[100,101,102,103])

print(s)

output:

--------

100 a

101 b

102 c

103 d

dtype: object

creating a series from dict:

----------------------------

import pandas as pd

import numpy as np

data = {'a' : 0., 'b' : 1., 'c' : 2.}

s = pd.Series(data)

print(s)

output:

--------

a 0.0

b 1.0

c 2.0

dtype: float64

Dictionary keys are used to construct index.

--------------------------------------------

import pandas as pd

data = {'a' : 0., 'b' : 1., 'c' : 2.}

s = pd.Series(data,index=['b','c','d','a'])

print(s)

output:

---------

b 1.0

c 2.0

d NaN

a 0.0

dtype: float64

Create a Series from Scalar

------------------------------

import pandas as pd

s = pd.Series(5, index=[0, 1, 2, 3])

print(s)

output:

---------

0 5

1 5

2 5

3 5

dtype: int64

retrive the element using index:

---------------------------------

import pandas as pd

s=pd.Series([1,2,3,4],index=['a','b','c','d']

print(s[0])

output:

------

1

print(s['a'])

output:

--------

1

Retrieve multiple elements using a list of index label values.

---------------------------------------------------------------

import pandas as pd

s=pd.Series([1,2,3,4],index=['a','b','c','d'])

print(s[['a','b','c']])

output:

-------

a 1

b 2

c 3

dtype: int64

DATA FRAMES:

--------------

syntax:

-------

pandas.DataFrame(data,index,columns,dtype,copy)

Data Frames can be created using

------------------------------------

\*Lists

\*Dicts

\*Series

\*Numpy ndarrays

\*Another DataFrame

Creating Empty DataFrame:

------------------------

import pandas as pd

s=pd.DataFrame()

print(s)

output:

-------

Empty DataFrame

Columns: []

Index: []

creating dataframe from list:

-----------------------------

import pandas as pd

data=[1,2,3,4,5]

s=pd.DataFrame(data)

print(s)

output:

--------

0

0 1

1 2

2 3

3 4

4 5

import pandas as pd

data=[['Narmadha',20,'IT'],['Shankar',20,'IT'],['Sathya',17,'BCA']]

s=pd.DataFrame(data,columns=['Name','Age','Dept'])

print(s)

output:

----------

Name Age Dept

0 Narmadha 20 IT

1 Shankar 20 IT

2 Sathya 17 BCA

import pandas as pd

data=[['Narmadha',20,'IT'],['Shankar',20,'IT'],['Sathya',17,'BCA']]

s=pd.DataFrame(data,columns=['Name','Age','Dept'],dtype=float)

print(s)

output:

----------

Name Age Dept

0 Narmadha 20.0 IT

1 Shankar 20.0 IT

2 Sathya 17.0 BCA

creating dataframe from dict of lists:

--------------------------------------

import pandas as pd

data={'Name':['Narmadha','Shankar','Sathya'],'Age':[20,20,17]}

s=pd.DataFrame(data)

print(s)

output:

---------

Name Age

0 Narmadha 20

1 Shankar 20

2 Sathya 17

import pandas as pd

data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

df1 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b'])

df2 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b1'])

print(df1)

print(df2)

output:

---------

#df1

a b

first 1 2

second 5 10

#df2

a b1

first 1 NaN

second 5 NaN

creating dataframe from dict of series:

----------------------------------------

Dictionary of Series can be passed to form a DataFrame. The resultant index is the union of all the series indexes passed.

1)

import pandas as pd

df={'one':pd.Series([1,2,3,4],index=['a','b','c','d']),'two':pd.Series([5,6,7,8

,9],index=['a','b','c','d','e'])}

s=pd.DataFrame(df)

print(s)

output:

--------

one two

a 1.0 5

b 2.0 6

c 3.0 7

d 4.0 8

e NaN 9

2)

import pandas as pd

df={'one':pd.Series([1,2,3,4],index=['a','b','c','d']),'two':pd.Series([5,6,7,8

,9],index=['a','b','c','d','e']),'three':pd.Series([10,2,3,6],index=['a','b',

'c','d']),'four':pd.Series([11,12,13,16],index=['a','b',

'c','d'])}

s=pd.DataFrame(df)

print(s)

output:

---------

one two three four

a 1.0 5 10.0 11.0

b 2.0 6 2.0 12.0

c 3.0 7 3.0 13.0

d 4.0 8 6.0 16.0

e NaN 9 NaN NaN

3)

import pandas as pd

df={'one':pd.Series([1,2,3,4],index=['a','b','c','d']),'two':pd.Series([5,6,7,8],index=['a','b','c','d']),'three':pd.Series([10,2,3,6],index=['a','b',

'c','d']),'four':pd.Series([11,12,13,16],index=['a','b',

'c','d'])}

s=pd.DataFrame(df)

print(s)

output:

--------

one two three four

a 1 5 10 11

b 2 6 2 12

c 3 7 3 13

d 4 8 6 16

Column Addition:

-----------------

import pandas as pd

data1={'Name':['Narmadha','Narmadha','Sathya'],'Age':[20,20,17],'Salary':[30000,

30000,50000]}

s['Dept']=['Sales','Marketing','Developing']

print(s)

output:

----------

Name Age Salary Dept

0 Narmadha 20 30000 Sales

1 Narmadha 20 30000 Marketing

2 Sathya 17 50000 Developing

Delete column from dataframe:

--------------------------------

import pandas as pd

data1={'Name':['Narmadha','shankar','Sathya'],'Age':[20,20,17],'Salary':[30000,

35000,50000]}

s=pd.DataFrame(data1)

s['Dept']=['Sales','Marketing','Developing']

del s['Dept']

s.pop('Age')

print(s)

output:

-------

Name Salary

0 Narmadha 30000

1 shankar 35000

2 Sathya 50000

Row addition:

--------------

import pandas as pd

data1={'Name':['Narmadha','shankar','Sathya'],'Age':[20,20,17],'Salary':[30000,

35000,50000]}

data2={'Name':['vidya','samja','aruna'],'Age':[21,22,27],'Salary':[20000,

25000,56000]}

s=pd.DataFrame(data1)

s1=pd.DataFrame(data2)

s=s.append(s1)

print(s)

output:

--------

Name Age Salary

0 Narmadha 20 30000

1 shankar 20 35000

2 Sathya 17 50000

0 vidya 21 20000

1 samja 22 25000

2 aruna 27 56000

Delete row from dataframe:

--------------------------

import pandas as pd

data1={'Name':['Narmadha','shankar','Sathya'],'Age':[20,20,17],'Salary':[30000,

35000,50000]}

data2={'Name':['vidya','samja','aruna'],'Age':[21,22,27],'Salary':[20000,

25000,56000]}

s=pd.DataFrame(data1)

s1=pd.DataFrame(data2)

s=s.append(s1)

s=s.drop(0)

print(s)

output:

---------

Name Age Salary

1 shankar 20 35000

2 Sathya 17 50000

1 samja 22 25000

2 aruna 27 56000

comparing all the records are same in two table using python pandas:

--------------------------------------------------------------------

import pandas as pd

import numpy as np

import pandas as pd

data1={'Name':['Narmadha','Shankar','Sathya'],'Age':[20,20,17]}

data2={'Name':['Narmadha','Shankar','Sathya'],'Age':[20,20,17]}

s=pd.DataFrame(data1)

s1=pd.DataFrame(data2)

if(s.equals(s1)):

print("True")

else:

print("False")

output:

--------

True

remove duplicates:

--------------------

import pandas as pd

data1={'Name':['Narmadha','Narmadha','Sathya'],'Age':[20,20,17],'Salary':[30000,

30000,50000]}

s=pd.DataFrame(data1)

n=s.drop\_duplicates()

print(n)

output:

--------

Name Age Salary

0 Narmadha 20 30000

2 Sathya 17 50000

Descriptive stastics

------------------------

sum()-Returns the sum of the values for the requested axis. By default, axis is index (axis=0).

import pandas as pd

data1={'Name':['Narmadha','shankar','Sathya'],'Age':[20,20,17],'Salary':[30000,

35000,50000]}

data2={'Name':['vidya','samja','aruna'],'Age':[21,22,27],'Salary':[20000,

25000,56000]}

s=pd.DataFrame(data1)

s1=pd.DataFrame(data2)

print(s.sum())

output:

------

Name NarmadhashankarSathya

Age 57

Salary 115000

dtype: object

when axis=1

------------

import pandas as pd

data1={'Name':['Narmadha','shankar','Sathya'],'Age':[20,20,17],'Salary':[30000,

35000,50000]}

data2={'Name':['vidya','samja','aruna'],'Age':[21,22,27],'Salary':[20000,

25000,56000]}

s=pd.DataFrame(data1)

s1=pd.DataFrame(data2)

print(s.sum(1))

output:

-------

0 30020

1 35020

2 50017

dtype: int64

mean()-returns the average value

import pandas as pd

data1={'Name':['Narmadha','shankar','Sathya'],'Age':[20,20,17],'Salary':[30000,

35000,50000]}

data2={'Name':['vidya','samja','aruna'],'Age':[21,22,27],'Salary':[20000,

25000,56000]}

s=pd.DataFrame(data1)

s1=pd.DataFrame(data2)

print(s.mean())

output:

---------

Age 19.000000

Salary 38333.333333

dtype: float64

prod()-product of value

s.prod()

output:

-------

Age 6800

Salary 52500000000000

dtype: int64

median()-Median of Values

mode()-mode of values

std()-standard deviation

abs()-absolute value

min()-minimum value

max()-maximum value

cumsum()-cumulative sum

cumprod()-cumulative product

BASIC FUNCTIONALITIES:

----------------------

random.randn()

---------------

The numpy.random.randn() function creates an array of specified shape and fills it with random values as per standard normal distribution.

create a series with random numbers:

import pandas as pd

import numpy as np

s = pd.Series(np.random.randn(4))

print(s)

output:

------

0 -0.793656

1 0.016717

2 0.085518

3 0.409168

dtype: float64

axes:

-------

import pandas as pd

import numpy as np

s = pd.Series(np.random.randn(4))

print(s.axes)

output:

-------

[RangeIndex(start=0, stop=4, step=1)]

to check the series is empty or not:

------------------------------------

import pandas as pd

import numpy as np

s = pd.Series(np.random.randn(4))

print(s.empty)

output:

---------

False

ndim-is to check dimension

--------------------------

import pandas as pd

import numpy as np

s = pd.DataFrame(np.random.randn(4,4))

print(s.ndim)

output:

-------

2

size

-------

import pandas as pd

import numpy as np

s = pd.DataFrame(np.random.randn(4,4))

print(s)

print(s.size)

output:

----------

0 1 2 3

0 0.419306 -1.145582 0.114435 -0.312969

1 0.174148 -0.568488 -1.347267 -0.402977

2 0.791812 0.573500 -0.354626 -0.547734

3 -0.410054 -0.603162 0.671832 2.558583

16

s.values

-------

output:

--------

[[ 0.13007024 0.5468105 -0.61112045 -2.10121585]

[ 1.76286462 -2.22310166 1.64893815 -0.8588524 ]

[-0.74950921 -0.88665008 -0.22146796 -2.03098955]

[ 0.16411089 0.8107648 -0.94237778 -0.50683384]]

s.head(n)-returns first n rows:

-------------------------------

import pandas as pd

import numpy as np

s = pd.Series(np.random.randn(4))

print ("The original series is:")

print(s)

print ("The first two rows of the data series:")

print(s.head(2))

output:

-------

The original series is:

0 0.080646

1 -0.729294

2 0.101370

3 -0.890155

dtype: float64

The first two rows of the data series:

0 0.080646

1 -0.729294

dtype: float64

s.tail(n)-returns last n rows:

-------------------------------

s.tail(2)

The original series is:

0 -0.098233

1 1.517126

2 0.573892

3 0.102237

dtype: float64

The last two rows of the data series:

2 0.573892

3 0.102237

dtype: float64

Basic functinality of dataframes:

------------------------------------

T (Transpose)

-------------------

Returns the transpose of the DataFrame. The rows and columns will interchange.

import pandas as pd

data1={'Name':['Narmadha','Narmadha','Sathya'],'Age':[20,20,17],'Salary':[30000,

30000,50000]}

s=pd.DataFrame(data1)

print("Original dataframe")

print(s)

print("After Transpose")

print(s.T)

output:

-------

Original dataframe

Name Age Salary

0 Narmadha 20 30000

1 Narmadha 20 30000

2 Sathya 17 50000

After Transpose

0 1 2

Name Narmadha Narmadha Sathya

Age 20 20 17

Salary 30000 30000 50000

s.axes

------

[RangeIndex(start=0, stop=3, step=1), Index(['Name', 'Age', 'Salary'], dtype='object')]

dtypes

--------

Returns the data type of each column.

output:

-------

Name object

Age int64

Salary int64

dtype: object

ndim-Returns the number of dimensions of the object. By definition, DataFrame is a 2D object.

shape-s.shape/(o/p):(3, 3)

size-returns no of element in dataframe

s.size=9

values-Returns the actual data in the DataFrame as an NDarray.

head()-returns first n rows dataframe,The default number of elements to display is five, but you may pass a custom number.

tail()-returns last n rows in dataframe

Function Application:

----------------------

To apply your own or another library’s functions to Pandas objects, you should be aware of the three important methods

Table wise Function Application: pipe()

Row or Column Wise Function Application: apply()

Element wise Function Application: applymap()

pipe():

------

Custom operations can be performed by passing the function and the appropriate number of parameters as pipe arguments.

Thus, operation is performed on the whole DataFrame.

for eg: multiply a value 2 to all the elements in the DataFrame. Then,add a value 3 to all dataframe

import pandas as pd

import numpy as np

def add(ele1,ele2,ele3):

return (ele1\*ele2)+ele3

data=[[12,20,13],[15,20,16],[21,17,22]]

s=pd.DataFrame(data,columns=['col1','col2','col3'])

print(s)

print(s.pipe(add,2,3))

output:

--------

col1 col2 col3

0 12 20 13

1 15 20 16

2 21 17 22

col1 col2 col3

0 27 43 29

1 33 43 35

2 45 37 47

apply()

---------

taking mean for values column wise

import pandas as pd

import numpy as np

data=[[12,20,13],[15,20,16],[21,17,22]]

s=pd.DataFrame(data,columns=['col1','col2','col3'])

print(s.apply(np.mean))

output:

-------

col1 16.0

col2 19.0

col3 17.0

dtype: float64

row wise by default axis value is 0

if give axis=1 it takes rowwise axis=0 takes columnwise

import pandas as pd

import numpy as np

data=[[12,20,13],[15,20,16],[21,17,22]]

s=pd.DataFrame(data,columns=['col1','col2','col3'])

print(s.apply(np.mean,axis=1))

output:

--------

0 15.0

1 17.0

2 20.0

dtype: float64

map()-is used for series object

------------------------------

import pandas as pd

import numpy as np

data=[[12,20,13],[15,20,16],[21,17,22]]

s=pd.DataFrame(data,columns=['col1','col2','col3'])

print(s['col1'].map(lambda x:x\*100))------>map function is used only for col1

output:

--------

0 1200

1 1500

2 2100

applymap()- used for dataframes

--------------------------------

import pandas as pd

import numpy as np

data=[[12,20,13],[15,20,16],[21,17,22]]

s=pd.DataFrame(data,columns=['col1','col2','col3'])

print(s.applymap(lambda x:x\*100))

output:

--------

col1 col2 col3

0 1200 2000 1300

1 1500 2000 1600

2 2100 1700 2200

reindexing

----------

import pandas as pd

import numpy as np

index = ['Narmadha', 'Shankar', 'Sathya', 'Shivani', 'Vino']

df = pd.DataFrame({'DOB': ['15-06-2000','14-01-2000','03-03-2003','09-09-1998','13-06-1995'],

'Age': [20,20,17,22,24]},

index=index)

new\_index=['Narmadha','Revathy','Sathya','Uvasri','Sandhiya']

print(df)

print(df.reindex(new\_index))

output:

----------

DOB Age

Narmadha 15-06-2000 20

Shankar 14-01-2000 20

Sathya 03-03-2003 17

Shivani 09-09-1998 22

Vino 13-06-1995 24

DOB Age

Narmadha 15-06-2000 20.0

Revathy NaN NaN

Sathya 03-03-2003 17.0

Uvasri NaN NaN

Sandhiya NaN NaN

fill\_value()

------------

print(df.reindex(new\_index,fill\_value='Missing'))

output:

---------

DOB Age

Narmadha 15-06-2000 20

Revathy Missing Missing

Sathya 03-03-2003 17

Uvasri Missing Missing

Sandhiya Missing Missing

ITERATION:

--------------

DataFrame-Iterating a DataFrame gives column names

import pandas as pd

data1={'Name':['Narmadha','shankar','Sathya'],'Age':[20,20,17],'Salary':[30000,

35000,50000]}

s=pd.DataFrame(data1)

for i in s:

print(i)

output:

--------

Name

Age

Salary

To iterate over the rows of the DataFrame, we can use the following functions

iteritems() − to iterate over the (key,value) pairs

iterrows() − iterate over the rows as (index,series) pairs

itertuples() − iterate over the rows as namedtuples

iteritems():

------------

Iterates over each column as key, value pair with label as key and column value as a Series object.

import pandas as pd

data1={'Name':['Narmadha','shankar','Sathya'],'Age':[20,20,17],'Salary':[30000,

35000,50000]}

s=pd.DataFrame(data1)

for key,value in s.iteritems():

print(key,value)

output:

-------

Name

0 Narmadha

1 shankar

2 Sathya

Name: Name, dtype: object

Age

0 20

1 20

2 17

Name: Age, dtype: int64

Salary

0 30000

1 35000

2 50000

Name: Salary, dtype: int64

iterrows():

-----------

import pandas as pd

data1={'Name':['Narmadha','shankar','Sathya'],'Age':[20,20,17],'Salary':[30000,

35000,50000]}

s=pd.DataFrame(data1)

for i,j in s.iterrows():

print(i,j)

output:

--------

0

Name Narmadha

Age 20

Salary 30000

Name: 0, dtype: object

1

Name shankar

Age 20

Salary 35000

Name: 1, dtype: object

2

Name Sathya

Age 17

Salary 50000

Name: 2, dtype: object

itertuples():

--------------

import pandas as pd

data1={'Name':['Narmadha','shankar','Sathya'],'Age':[20,20,17],'Salary':[30000,

35000,50000]}

s=pd.DataFrame(data1)

for i in s.itertuples():

print(i)

output:

-------

Pandas(Index=0, Name='Narmadha', Age=20, Salary=30000)

Pandas(Index=1, Name='shankar', Age=20, Salary=35000)

Pandas(Index=2, Name='Sathya', Age=17, Salary=50000)

SORTING:

----------

There are two kinds of sorting available in Pandas. They are −

By label

By Actual Value

By label-Using the sort\_index() method, by passing the axis arguments and the order of sorting, DataFrame can be sorted.

By default, sorting is done on row labels in ascending order.

by index:

-----------

import pandas as pd

df = pd.DataFrame([1, 2, 3, 4, 5], index=[100, 29, 234, 1, 150],

columns=['A'])

print(df)

print(df.sort\_index())

output:

---------

A

100 1

29 2

234 3

1 4

150 5

after sorting:

-----------

A

1 4

29 2

100 1

150 5

234 3

descending:

-----------

print(df.sort\_index(ascending=False))

output:

---------

A

234 3

150 5

100 1

29 2

1 4

sorting by values:

-------------------

sort the value by col1

import pandas as pd

df = pd.DataFrame({

'col1': ['A', 'A', 'B','E','D', 'C'],

'col2': [2, 1, 9, 8, 7, 4],

'col3': [0, 1, 9, 4, 2, 3]

})

print(df)

print(df.sort\_values(by=['col1']))

output:

----------

col1 col2 col3

0 A 2 0

1 A 1 1

2 B 9 9

3 E 8 4

4 D 7 2

5 C 4 3

col1 col2 col3

0 A 2 0

1 A 1 1

2 B 9 9

5 C 4 3

4 D 7 2

3 E 8 4

sort by multiple column

-----------------------

print(df.sort\_values(by=['col1', 'col2']))

output:

---------

col1 col2 col3

1 A 1 1

0 A 2 0

2 B 9 9

5 C 4 3

4 D 7 2

3 E 8 4

descending order:

----------------

print(df.sort\_values(by=['col1'], ascending=False))

output:

-------

col1 col2 col3

3 E 8 4

4 D 7 2

5 C 4 3

2 B 9 9

0 A 2 0

1 A 1 1

Sorting with a key function

------------------------------

import pandas as pd

df = pd.DataFrame({

'col1': ['A', 'A', 'e','f','D', 'C'],

'col2': [2, 1, 9, 8, 7, 4],

'col3': [0, 1, 9, 4, 2, 3]})

print(df)

print(df.sort\_values(by='col1', key=lambda col: col.str.lower()))

output:

--------

col1 col2 col3

0 A 2 0

1 A 1 1

2 e 9 9

3 f 8 4

4 D 7 2

5 C 4 3

after sorting:

col1 col2 col3

0 A 2 0

1 A 1 1

5 C 4 3

4 D 7 2

2 e 9 9

3 f 8 4

Putting NAs first

-----------------

df.sort\_values(by='col1', ascending=False, na\_position='first')

WORKING WITH TEXTDATA:

------------------------

Pandas provides a set of string functions which make it easy to operate on string data. Most importantly, these functions ignore (or exclude) missing/NaN values.

lower():

------

import pandas as pd

import numpy as np

s = pd.Series(['Tom', 'William Rick', 'John', 'Alber@t', np.nan, '1234','SteveSmith'])

print(s.str.lower())

output:

------

0 tom

1 william rick

2 john

3 alber@t

4 NaN

5 1234

6 stevesmith

dtype: object

upper()-convert string to uppercase

-----------------------------------

print(s.str.upper())

output:

-------

0 TOM

1 WILLIAM RICK

2 JOHN

3 ALBER@T

4 NaN

5 1234

6 STEVESMITH

dtype: object

len()

-------

print(s.str.len())

output:

--------

print length of all strings

strip()-remove the whitespace

split(' ')-Splits each string with the given pattern.

cat(sep=' ')-Concatenates the series/index elements with given separator.

eg:

---

print(s.str.cat(sep='\_'))

output:

--------

Tom\_William Rick\_John\_Alber@t\_1234\_SteveSmith

get\_dummies()

contains(pattern)-Returns a Boolean value True for each element if the substring contains in the element, else False.

replace(a,b)-replace the a with b value

repeat()-Repeats each element with specified number of times.

eg:

---

print(s.str.repeat(2))

output:

--------

0 TomTom

1 William RickWilliam Rick

2 JohnJohn

3 Alber@tAlber@t

4 NaN

5 12341234

6 SteveSmithSteveSmith

count()-Returns count of appearance of pattern in each element.

startswith(pattern)-Returns true if the element in the Series/Index starts with the pattern.

eg:print(s.str. startswith ('T'))

endswith(pattern)-Returns true if the element in the Series/Index ends with the pattern.

find()-Returns the first position of the first occurrence of the pattern.

findall(pattern)-Returns a list of all occurrence of the pattern.

swapcase()-Swaps the case lower/upper.

islower()-check all the character in string is lower case or not.returns boolen value

isupper()-check all the character in string is upper case or not.returns boolen value

isnumeric()-check all the character in string is numeric or not.returns boolen value

OPTION AND CUSTOMIZATION:

---------------------------------------------

Pandas have some default factors which restrict the analysis of data,it is important to know the various methods to change the default pandas values

Types of pandas option and customization:

* get\_option()
* set\_option()
* reset\_option()
* describe\_option()
* option\_context()

pandas.get\_option():

---------------------------

Which gives details about default values in pandas.Using “display.max\_rows” and

“Display.max\_columns” as parameter we get a maximum number of rows and columns that can display by default.

import pandas as pd

print(pd.get\_option("display.max\_rows"))

Output:

60

Columns:

import pandas as pd

print(pd.get\_option("display.max\_columns"))

Output:

20

pandas.set\_option():

----------------------------

The set\_option() function allow us to change a default value to something of our choice.

display.max\_rows:

------------------------

import pandas as pd

pd.set\_option("display.max\_rows",80)

print(pd.get\_option("display.max\_rows"))

Output:

80

display.max\_columns:

------------------------------

import pandas as pd

pd.set\_option("display.max\_columns",30)

print(pd.get\_option("display.max\_columns"))

Output:

30

reset\_option

-----------------

import pandas as pd

pd.reset\_option("display.max\_rows")

print(pd.get\_option("display.max\_rows"))

Output:

60

describe\_option()

------------------------

prints the description of the argument.

option\_context()

---------------------

option\_context context manager is used to set the option in with statement temporarily.

Option values are restored automatically when you exit the with block

display.expand\_frame\_repr

------------------------------------

Displays DataFrames to Stretch Pages

display.max\_colwidth

----------------------------

Displays maximum column width

display.precision

----------------------

Displays precision for decimal numbers

AGGREGATIONS:

-------------------------

import pandas as pd

df = pd.DataFrame([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]],

columns=['A', 'B', 'C'])

print(df)

print(df.agg(['sum', 'min']))

Output:

A B C

0 1 2 3

1 4 5 6

2 7 8 9

A B C

sum 12 15 18

min 1 2 3

Different aggregations per column

----------------------------------------------

import pandas as pd

df = pd.DataFrame([[1, 2,3],

[4, 5,6]],

columns=['A', 'B','C'])

print(df.agg({'A' : ['sum', 'min'], 'B' : ['min', 'max']}))

Output:

A B

max NaN 5.0

min 1.0 2.0

sum 5.0 NaN

Aggregate over column:

----------------------------------

import pandas as pd

df = pd.DataFrame([[1, 2,3],

[4, 5,6]],

columns=['A', 'B','C'])

print(df)

print(“after aggregation”)

print(df.agg("mean", axis="columns"))

Output:

A B C

0 1 2 3

1 4 5 6

after aggregation

0 2.0

1 5.0

dtype: float64

MISSING VALUES:

------------------------------

Using reindexing, we have created a DataFrame with missing values. In the output, NaN means Not a Number

import pandas as pd

data=[[1,2,3],[4,5,6],[7,8,9]]

df=pd.DataFrame(data,index=['a','c','e'],columns=['col1','col2','col3'])

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

print(df)

Output:

col1 col2 col3

a 1.0 2.0 3.0

b NaN NaN NaN

c 4.0 5.0 6.0

d NaN NaN NaN

e 7.0 8.0 9.0

f NaN NaN NaN

To make detecting missing values easier (and across different array dtypes), Pandas provides the isnull() and notnull() functions,

which are also methods on Series and DataFrame objects

print(df['col1'].isnull())

Output:

a False

b True

c False

d True

e False

f True

Notnull

print(df[‘col1’].notnull())

Output:

a True

b False

c True

d False

e True

f False

Calculations with Missing Data

When summing data, NA will be treated as Zero

If the data are all NA, then the result will be NA

print(df[‘col1’].sum())

Output:

12.0

Replacing NAN with 0

print(df.fillna(0))

Output:

col1 col2 col3

a 1.0 2.0 3.0

b 0.0 0.0 0.0

c 4.0 5.0 6.0

d 0.0 0.0 0.0

e 7.0 8.0 9.0

f 0.0 0.0 0.0

pad/fill:fill method forward

------------------------------------

print(df.fillna(method=’pad’))

Output:

col1 col2 col3

a 1.0 2.0 3.0

b 1.0 2.0 3.0

c 4.0 5.0 6.0

d 4.0 5.0 6.0

e 7.0 8.0 9.0

f 7.0 8.0 9.0

bfill/backfill -fill method backward

---------------------------------------------

print(df.fillna(method=’bfill’))

Output:

col1 col2 col3

a 1.0 2.0 3.0

b 4.0 5.0 6.0

c 4.0 5.0 6.0

d 7.0 8.0 9.0

e 7.0 8.0 9.0

f NaN NaN NaN

Drop Missing values:

---------------------------

print(df.dropna())

Output:

col1 col2 col3

a 1.0 2.0 3.0

c 4.0 5.0 6.0

e 7.0 8.0 9.0

print(df.dropna(axis=1))

Output:

Empty DataFrame

Columns: []

Index: [a, b, c, d, e, f]

Replace:

--------------

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

GROUPBY

-----------------------

groupby involves splitting the object,Applying a function,combining the result

In many situations,we split the data into sets and we apply some functionality on each subset.In apply functionality we can perform the following operations

Aggregation

Transformation

Filtration

import pandas as pd

ipl\_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',

'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],

'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],

'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],

'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}

df = pd.DataFrame(ipl\_data)

print(df)

Output:

Points Rank Team Year

0 876 1 Riders 2014

1 789 2 Riders 2015

2 863 2 Devils 2014

3 673 3 Devils 2015

4 741 3 Kings 2014

5 812 4 kings 2015

6 756 1 Kings 2016

7 788 1 Kings 2017

8 694 2 Riders 2016

9 701 4 Royals 2014

10 804 1 Royals 2015

11 690 2 Riders 2017

Applying groupby in dataframe:

print(df.groupby(‘Team’).groups)

Output:

{'Devils': [2, 3], 'Kings': [4, 6, 7], 'Riders': [0, 1, 8, 11], 'Royals': [9, 10], 'kings': [5]}

Groupby with multiple columns:

print(df.groupby([‘Team’,’Year’]).groups)

Output:

{('Devils', 2014): [2], ('Devils', 2015): [3], ('Kings', 2014): [4], ('Kings', 2016): [6], ('Kings', 2017): [7], ('Riders', 2014): [0], ('Riders', 2015): [1], ('Riders', 2016): [8], ('Riders', 2017): [11], ('Royals', 2014): [9], ('Royals', 2015): [10], ('kings', 2015): [5]}

Iterating through groups:

grouped=df.groupby(‘Year’)

for i,j in grouped:

print(i)

print(j)

Output:

2014

Team Rank Year Points

0 Riders 1 2014 876

2 Devils 2 2014 863

4 Kings 3 2014 741

9 Royals 4 2014 701

2015

Team Rank Year Points

1 Riders 2 2015 789

3 Devils 3 2015 673

5 kings 4 2015 812

10 Royals 1 2015 804

2016

Team Rank Year Points

6 Kings 1 2016 756

8 Riders 2 2016 694

2017

Team Rank Year Points

7 Kings 1 2017 788

11 Riders 2 2017 690

get\_group():

Through this method we can select single group

grouped=df.groupby(‘Year’)

print(grouped.get\_group(‘2017’))

Output:

Team Rank Year Points

7 Kings 1 2017 788

11 Riders 2 2017 690

Aggregation:

grouped=df.groupby('Team')

print(grouped.agg('size'))

Output:

Team

Devils 2

Kings 3

Riders 4

Royals 2

kings 1

dtype: int64

Multiple aggregate function:

import pandas as pd

import numpy as np

ipl\_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',

'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],

'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],

'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],

'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}

df = pd.DataFrame(ipl\_data)

grouped=df.groupby('Team')

print(grouped['Points'].agg([np.sum, np.mean, np.std]))

Output:

sum mean std

Team

Devils 1536 768.000000 134.350288

Kings 2285 761.666667 24.006943

Riders 3049 762.250000 88.567771

Royals 1505 752.500000 72.831998

kings 812 812.000000 NaN

Filtration:

grouped=df.groupby('Team')

print(df.groupby('Team').filter(lambda x: len(x) >= 3))

Output:

Team Rank Year Points

0 Riders 1 2014 876

1 Riders 2 2015 789

4 Kings 3 2014 741

6 Kings 1 2016 756

7 Kings 1 2017 788

8 Riders 2 2016 694

11 Riders 2 2017 690

MERGING/JOINING:

Join operation is like join in sql

left in pandas is equal to left outer join in sql

right=right outer join

outer=full outer join

inner =inner join

Two tables to join:

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(left)

print(right)

Output:

id Name subject\_id

0 1 Alex sub1

1 2 Amy sub2

2 3 Allen sub4

3 4 Alice sub6

4 5 Ayoung sub5

id Name subject\_id

0 1 Billy sub2

1 2 Brian sub4

2 3 Bran sub3

3 4 Bryce sub6

4 5 Betty sub5

join:

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

Output:

id Name\_x subject\_id\_x Name\_y subject\_id\_y

0 1 Alex sub1 Billy sub2

1 2 Amy sub2 Brian sub4

2 3 Allen sub4 Bran sub3

3 4 Alice sub6 Bryce sub6

4 5 Ayoung sub5 Betty sub5

Merge two dataframes on multiple keys:

print(pd.merge(left,right,on=['id','subject\_id']))

Output:

id Name\_x subject\_id Name\_y

0 4 Alice sub6 Bryce

1 5 Ayoung sub5 Betty

By using how argument we use sql joins:

Left join:

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

Output:

id\_x Name\_x subject\_id id\_y Name\_y

0 1 Alex sub1 NaN NaN

1 2 Amy sub2 1.0 Billy

2 3 Allen sub4 2.0 Brian

3 4 Alice sub6 4.0 Bryce

4 5 Ayoung sub5 5.0 Betty

Right join:

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

id\_x Name\_x subject\_id id\_y Name\_y

0 2.0 Amy sub2 1 Billy

1 3.0 Allen sub4 2 Brian

2 NaN NaN sub3 3 Bran

3 4.0 Alice sub6 4 Bryce

4 5.0 Ayoung sub5 5 Betty

Outer join:

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

Output:

id\_x Name\_x subject\_id id\_y Name\_y

0 1.0 Alex sub1 NaN NaN

1 2.0 Amy sub2 1.0 Billy

2 3.0 Allen sub4 2.0 Brian

3 4.0 Alice sub6 4.0 Bryce

4 5.0 Ayoung sub5 5.0 Betty

5 NaN NaN sub3 3.0 Bran

Inner join:

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

Output:

id\_x Name\_x subject\_id id\_y Name\_y

0 2 Amy sub2 1 Billy

1 3 Allen sub4 2 Brian

2 4 Alice sub6 4 Bryce

3 5 Ayoung sub5 5 Betty