- BY POLU SREEKANTH REDDY

Polu Sreekanth Reddy
SreeData

PANDAS SERIES

Pandas series

- BY POLU SREEKANTH REDDY

Pandas

Pandas is a data manipulation package in Python for tabular data. That is, data in the form of rows and columns, also known as DataFrames. Intuitively, you can think of a DataFrame as an Excel sheet.

Key Features of Pandas

1. It has a DataFrame object that is quick and effective, with both standard and custom indexing.
2. Utilized for reshaping and turning of the informational indexes.
3. For aggregations and transformations, group by data.
4. It is used to align the data and integrate the data that is missing.
5. Provide Time Series functionality.
6. Process a variety of data sets in various formats, such as matrix data, heterogeneous tabular data, and time series.
7. Manage the data sets' multiple operations, including subsetting, slicing, filtering, groupBy, reordering, and reshaping.
8. It incorporates with different libraries like SciPy, and scikit-learn.
9. Performs quickly, and the Cython can be used to accelerate it even further.

Benefits of Pandas

The following are the advantages of pandas overusing other languages:

Representation of Data: Through its DataFrame and Series, it presents the data in a manner that is appropriate for data analysis.

- BY POLU SREEKANTH REDDY

Clear code: Pandas' clear API lets you concentrate on the most important part of the code. In this way, it gives clear and brief code to the client.

Installation of Pandas

If you have Python and PIP already installed on a system, then installation of Pandas is very easy.

Install it using this command:

C:\Users\Your Name>pip install pandas

Import Pandas

Once Pandas is installed, import it in your applications by adding the import keyword:

import pandas

Pandas as pd

Pandas is usually imported under the pd alias.

alias: In Python alias are an alternate name for referring to the same thing.

Create an alias with the as keyword while importing:

import pandas as pd

Pandas Data Structures

Pandas generally provide two data structures for manipulating data, They are:

- **Series**
- **DataFrame**

- BY POLU SREEKANTH REDDY

1.Pandas Series

The Pandas Series can be defined as a one-dimensional array that is capable of storing various data types. We can easily convert the list, tuple, and dictionary into series using "series' method. The row labels of series are called the index. A Series cannot contain multiple columns.

Creating a Series:

1. Create an empty Series

```
import pandas as pd
s = pd.Series(dtype='float64')
print(s)
```

```
Series([], dtype: float64)
```

2. Create a Series using arrays.

```
import pandas as pd
values = [10,20,30,40,50]
pd.Series(values)
```

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

3. Create series from ndarray

```
import numpy as np
data = np.array(['x','y','z'])
s = pd.Series(data)
s
```

```
0    x
1    y
2    z
dtype: object
```

- BY POLU SREEKANTH REDDY

4. Create a series from a dict

```
data = {1:'Sreekanth',2:'Ram'}
s = pd.Series(data)
s
```

```
1    Sreekanth
2          Ram
dtype: object
```

5. Create a Series using Scalar

```
x = pd.Series(4, index=[0, 1, 2, 3])
print(x)
```

```
0    4
1    4
2    4
3    4
dtype: int64
```

Pandas series get index

```
data = np.array(['x','y','z'])
s = pd.Series(data)
s[1]
```

```
'y'
```

Retrieving Index array and data array of a series object

```
x=pd.Series(data=[2,4,6,8])
y=pd.Series(data=[11.2,18.6,22.5], index=['a','b','c'])
print(x.index)
print(x.values)
print(y.index)
print(y.values)
```

```
RangeIndex(start=0, stop=4, step=1)
[2 4 6 8]
Index(['a', 'b', 'c'], dtype='object')
[11.2 18.6 22.5]
```

Retrieving Types (dtype) and Size of Type (itemsize)

```
import numpy as np
import pandas as pd
a=pd.Series(data=[1,2,3,4])
```

- BY POLU SREEKANTH REDDY

```
b=pd.Series(data=[4.9,8.2,5.6],index=['x','y','z'])
print(a.dtype)
print(a.size)
print(b.dtype)
print(b.size)
```

```
int64
4
float64
3
```

Retrieving Shape

```
a=pd.Series(data=[1,2,3,4])
b=pd.Series(data=[4.9,8.2,5.6],index=['x','y','z'])
print(a.shape)
print(b.shape)
```

```
(4,)
(3,)
```

Retrieving Dimension, Size and Number of bytes

```
a=pd.Series(data=[1,2,3,4])
b=pd.Series(data=[4.9,8.2,5.6],
index=['x','y','z'])
print(a.ndim, b.ndim)
print(a.size, b.size)
print(a.nbytes, b.nbytes)
```

```
1 1
4 3
32 24
```

Checking Emptiness and Presence of NaNs

```
a=pd.Series(data=[1,2,3,np.NaN])
b=pd.Series(data=[4.9,8.2,5.6],index=['x','y','z'])
c=pd.Series(dtype='float64')
print(a.empty,b.empty,c.empty)
print(a.hasnans,b.hasnans,c.hasnans)
print(len(a),len(b))
print(a.count( ),b.count( ))
```

- BY POLU SREEKANTH REDDY

Output :

```
False False True
True False False
4 3
3 3
```

Pandas Series.map()

The main task of map() is used to map the values from two series that have a common column. To map the two Series, the last column of the first Series should be the same as the index column of the second series, and the values should be unique.

Syntax:

Series.map(arg, na_action=None)

- *arg:*
  function, dict, or Series. It refers to the mapping correspondence.
- *na_action:*
  {None, 'ignore'}, Default value None. If ignore, it returns null values, without passing it to the mapping correspondence.
- *Return type:*
  Pandas Series with same as index as caller

1. we will take a series and then apply the map function.

```
import pandas as pd
s = pd.Series(['cat', 'cow', 'dog'])
print("Series:\n", s)
print("Mapping: ")
s.map({'cat': 'kitten', 'cow': 'calf'})
```

```
Series:
 0    cat
1    cow
2    dog
dtype: object
Mapping:

0    kitten
1      calf
2       NaN
dtype: object
```

- BY POLU SREEKANTH REDDY

2. We can also directly pass a function to the map.

```
import pandas as pd
s = pd.Series(['lily', 'rose','lotus'])
s.map('This is a {}'.format)
```

```
0     This is a lily
1     This is a rose
2    This is a lotus
dtype: object
```

Pandas Series.std()

The Pandas std() is defined as a function for calculating the standard deviation of the given set of numbers, DataFrame, column, and rows.

```
import pandas as pd
# calculate standard deviation
import numpy as np
print(np.std([4,7,2,1,6,3]))
print(np.std([6,9,15,2,-17,15,4]))
```

```
2.1147629234082532
10.077252622027656
```

Pandas Series.to_frame()

Series is defined as a type of list that can hold an integer, string, double values, etc. It returns an object in the form of a list that has an index starting from 0 to n where n represents the length of values in Series.

```
s = pd.Series(["a", "b", "c"],
name="vals")
s.to_frame()
```

```
  vals
0    a
1    b
2    c
```

- BY POLU SREEKANTH REDDY

Pandas Series.unique()

While working with the DataFrame in Pandas, you need to find the unique elements present in the column. For doing this, we have to use the unique() method to extract the

unique values from the columns. The Pandas library in Python can easily help us to find unique data.

```
import numpy as np
import pandas as pd
pd.Series([2, 4, 3, 3]).unique()
```

```
array([2, 4, 3], dtype=int64)
```

```
import pandas as pd
pd.unique(pd.Series([2, 1, 3, 3]))
pd.unique(pd.Series([pd.Timestamp('20160101'),
pd.Timestamp('20160101')]))
```

```
array(['2016-01-01T00:00:00.000000000'], dtype='datetime64[ns]')
```

```
import pandas as pd
import numpy as np
pd.unique(pd.Index([pd.Timestamp('20160101', tz='US/Eastern'),
pd.Timestamp('20160101', tz='US/Eastern')]))
```

```
DatetimeIndex(['2016-01-01 00:00:00-05:00'], dtype='datetime64[ns,

US/Eastern]', freq=None)
```

Pandas Series.value_counts()

The value_counts() function returns a Series that contain counts of unique values. It returns an object that will be in descending order so that its first element will be the most frequently-occurred element.

By default, it excludes NA values.

- BY POLU SREEKANTH REDDY

```
import pandas as pd

# Creating the Series
sr = pd.Series(['New York', 'Chicago', 'Toronto', 'Lisbon', 'Rio', 'Chicago',
'Lisbon'])

# Print the series
print(sr)
```

```
0    New York
1     Chicago
2     Toronto
3      Lisbon
4         Rio
5     Chicago
6      Lisbon
dtype: object
```

```
sr.value_counts()
```

```
Chicago     2
Lisbon      2
New York    1
Toronto     1
Rio         1
dtype: int64
```