GISpark Documentation

发布 ***0.1***

openthings

4月 27, 2016

Contents

Contents:

CHAPTER 1

1、运行环境

1.1 1.1 Docker和Mesos

1.2 1.2 Python和Jupyter

通过功能强大的Notebook进行远程数据分析。

- Notebook
- ReadTheDocs

1.3 1.3 Spark分布式计算环境

CHAPTER 2

2、开放数据

2.1 Hello World

2.2 Markdown test

- Markdown test

2.3 《Data Master》

2.3.1 基础篇—Python快速入门

List，使用[a,b,c,...]方式声明。

列表是基础的数据结构。

```
In [9]: alist = [0,1,2,3,4]
        print("总计: ",len(alist))
        print("元素:", alist)
```

```
总计:  5
元素: [0, 1, 2, 3, 4]
```

字符串的列表。

```
In [20]: colours = ["red","green","blue"]
         for colour in colours:
             print(colour)
```

```
red
green
blue
```

列表的传统方式遍历。

```
In [24]: for i in range(0,len(alist)):
             print(alist[i])
```

```
0
1
2
3
4
```

列表的递归方式遍历。

```
In [25]: for i in alist:
             print(i)
```

```
0
1
2
3
4
```

可以直接调用一个列表。

```
In [26]: for obj in [0,1,2,3,4]:
             print(obj)
```

```
0
1
2
3
4
```

创建一个矩阵。

```
In [63]: olist = [[11,12,13],[21,22,23],[31,32,33]]
         for row in olist:
             print(row)
```

```
[11, 12, 13]
[21, 22, 23]
[31, 32, 33]
```

生成一个数据序列。在做数值检验时很有用。

```
In [43]: for obj in range(5):
             print(obj)
```

```
0
1
2
3
4
```

生成一个数据序列：range(开始值，结束值，步长)

```
In [27]: for obj in range(5,10,2):
             print(obj)
```

```
5
7
9
```

String as a list of char. 字符串是字符数组。

```
In [7]: name='BeginMan'
        for obj in range(len(name)):
            print('(%d)' %obj,name[obj])
```

```
(0) B
(1) e
(2) g
(3) i
(4) n
(5) M
(6) a
(7) n
```

2.3.2 Dictionary，词典，｛key:value,...｝

词典数据就是一系列k:v值对的集合。

```
In [30]: dict = {'name':'BeginMan','job':'pythoner','age':22}

         print("Dict Length: ",len(dict))
         print(dict)
```

```
Dict Length:  3
'age': 22, 'job': 'pythoner', 'name': 'BeginMan'
```

注意：上面的词典数据的输出与json表示是完全一致的，后面在json会专门介绍。

词典的遍历：

```
In [71]: dict["name"]
```

```
Out[71]: 'BeginMan'
```

```
In [41]: print("Key","\t Value")
         print("=================")
         for key in dict:
             print(key,"\t",dict[i])
```

```
Key      Value
=================
age      BeginMan
job      BeginMan
name     BeginMan
```

dict的每一个item(obj)是一个二元组（下面介绍元组）。

```
In [31]: for obj in dict.items():
             print(obj)
```

```
('age', 22)
('job', 'pythoner')
('name', 'BeginMan')
```

```
In [46]: for k,v in dict.items():
             print(k,v)
```

```
age 22
job pythoner
name BeginMan
```

```
In [72]: import json

         j = json.dumps(dict)
         print(repr(j))
```

```
'"age": 22, "job": "pythoner", "name": "BeginMan"'
```

```
In [53]: d = json.loads('{"age": 22, "job": "pythoner", "name": "BeginMan"}')
         print("Type of d: ", type(d))
         print(d)
```

```
Type of d:  <class 'dict'>
'name': 'BeginMan', 'job': 'pythoner', 'age': 22
```

2.3.3 tuple，(obj1,obj2,...)，元组

一个元组可包含多种类型的对象，不可修改。

```
In [59]: tup = 'abc',1,2,'x',True
```

```
In [60]: len(tup),tup
```

```
Out[60]: (5, ('abc', 1, 2, 'x', True))
```

```
In [56]: x,y =1,2
```

```
In [10]: x,y
```

```
Out[10]: (1, 2)
```

一个字典、元组构成的复合列表对象。

```
In [67]: ao = [{"k1":"key","k2":2},(3,"element")]
         ao
```

```
Out[67]: [{'k1': 'key', 'k2': 2}, (3, 'element')]
```

访问这个符合对象的值。

len(ao),ao[0]["k1"],ao[1][0],ao[1][1] 从上面可以看出，python的数据结构是非常灵活的，是数据探索和分析、处理的利器。

2.4 Pandas_QuickStart

Origin from http://pandas.pydata.org/pandas-docs/stable/
by openthings@163.com, 2016-04.

2.4.1 6.1 Object Creation

Creating a Series by passing a list of values, letting pandas create a default integer index:

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

        s = pd.Series([1,3,5,np.nan,6,8])
        s
```

```
Out[1]: 0    1.0
        1    3.0
        2    5.0
        3    NaN
        4    6.0
        5    8.0
        dtype: float64
```

Creating a DataFrame by passing a numpy array, with a datetime index and labeled columns:

```
In [2]: dates = pd.date_range('20130101', periods=6)
        dates
```

```
Out[2]: DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
                       '2013-01-05', '2013-01-06'],
                      dtype='datetime64[ns]', freq='D')
```

```
In [7]: df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))

        df
```

Creating a DataFrame by passing a dict of objects that can be converted to series-like.

```
In [8]: df2 = pd.DataFrame({ 'A' : 1.,
        'B' : pd.Timestamp('20130102'),
        'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
        'D' : np.array([3] * 4,dtype='int32'),
        'E' : pd.Categorical(["test","train","test","train"]),
        'F' : 'foo' })

        df2
```

```
In [9]: df2.dtypes
```

```
Out[9]: A           float64
        B    datetime64[ns]
        C           float32
        D             int32
        E          category
        F            object
        dtype: object
```

If you're using IPython, tab completion for column names (as well as public attributes) is automatically enabled. Here's a subset of the attributes that will be completed:

```
In [13]: df2.<TAB>
```

```
In [11]: df2.
```

```
Out[11]: 0    1.0
         1    1.0
         2    1.0
         3    1.0
         Name: A, dtype: float64
```

As you can see, the columns A, B, C, and D are automatically tab completed. E is there as well; the rest of the attributes have been truncated for brevity.

2.4.2 6.2 Viewing Data

```
In [14]: df.head()
```

```
In [15]: df.tail(3)
```

```
In [16]: df.index
```

```
Out[16]: DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
                        '2013-01-05', '2013-01-06'],
                       dtype='datetime64[ns]', freq='D')
```

```
In [17]: df.values
```

```
Out[17]: array([[-1.33401275, -0.34829657,  0.38865407, -0.22596701],
                [-0.13997444, -1.34778853,  0.81707707,  0.19247685],
                [-1.0827386 , -0.5441047 , -1.42388302, -1.24736743],
                [ 0.03478847, -0.67722051,  0.12044917,  0.7943414 ],
                [ 0.42854678, -0.61015602, -0.95089113, -0.0580473 ],
                [ 0.12563068, -0.11665286, -0.54457518, -1.57878468]])
```

```
In [18]: df.describe()
```

```
In [19]: df.T
```

```
In [20]: df.sort_index(axis=1, ascending=False)
```

```
In [21]: df.sort_values(by='B')
```

2.4.3 6.3 Selection

Getting

```
In [22]: df['A']
```

```
Out[22]: 2013-01-01   -1.334013
         2013-01-02   -0.139974
         2013-01-03   -1.082739
         2013-01-04    0.034788
         2013-01-05    0.428547
         2013-01-06    0.125631
         Freq: D, Name: A, dtype: float64
```

```
In [23]: df[0:3]
```

```
In [24]: df['20130102':'20130104']
```

6.3.2 Selection by Label

For getting a cross section using a label

```
In [25]: df.loc[dates[0]]
```

```
Out[25]: A   -1.334013
         B   -0.348297
         C    0.388654
         D   -0.225967
         Name: 2013-01-01 00:00:00, dtype: float64
```

Selecting on a multi-axis by label

```
In [26]: df.loc[:,['A','B']]
```

Showing label slicing, both endpoints are included

```
In [27]: df.loc['20130102':'20130104',['A','B']]
```

Reduction in the dimensions of the returned object

```
In [30]: df.loc['20130102',['A','B']]
```

```
Out[30]: A   -0.139974
         B   -1.347789
         Name: 2013-01-02 00:00:00, dtype: float64
```

For getting a scalar value

```
In [31]: df.loc[dates[0],'A']
```

```
Out[31]: -1.3340127475498547
```

For getting fast access to a scalar (equiv to the prior method)

```
In [32]: df.at[dates[0],'A']
```

```
Out[32]: -1.3340127475498547
```

6.3.3 Selection by Position

See more in Selection by Position Select via the position of the passed integers

```
In [33]: df.iloc[3]
```

```
Out[33]: A    0.034788
         B   -0.677221
         C    0.120449
         D    0.794341
         Name: 2013-01-04 00:00:00, dtype: float64
```

By integer slices, acting similar to numpy/python

```
In [34]: df.iloc[3:5,0:2]
```

By lists of integer position locations, similar to the numpy/python style

```
In [35]: df.iloc[[1,2,4],[0,2]]
```

For slicing rows explicitly

```
In [36]: df.iloc[1:3,:]
```

For slicing columns explicitly

```
In [37]: df.iloc[:,1:3]
```

For getting a value explicitly

```
In [39]: df.iloc[1,1]
```

```
Out[39]: -1.3477885295869219
```

For getting fast access to a scalar (equiv to the prior method)

```
In [40]: df.iat[1,1]
```

```
Out[40]: -1.3477885295869219
```

6.3.4 Boolean Indexing

Using a single column's values to select data.

```
In [41]: df[df.A > 0]
```

A where operation for getting.

```
In [42]: df[df > 0]
```

Using the isin() method for filtering:

```
In [43]: df2 = df.copy()
```

添加一列。

```
In [44]: df2['E'] = ['one', 'one','two','three','four','three']
```

```
In [45]: df2
```

```
In [46]: df2[df2['E'].isin(['two','four'])]
```

6.3.5 Setting

Setting a new column automatically aligns the data by the indexes

```
In [48]: s1 = pd.Series([1,2,3,4,5,6], index=pd.date_range('20130102', periods=6))
         s1
```

```
Out[48]: 2013-01-02    1
         2013-01-03    2
         2013-01-04    3
         2013-01-05    4
         2013-01-06    5
         2013-01-07    6
         Freq: D, dtype: int64
```

Setting values by position

```
In [49]: df.iat[0,1] = 0
```

Setting by assigning with a numpy array

```
In [50]: df.loc[:,'D'] = np.array([5] * len(df))
```

The result of the prior setting operations

```
In [51]: df
```

A where operation with setting.

```
In [52]: df2 = df.copy()
```

```
In [53]: df2[df2 > 0] = -df2
```

```
In [54]: df2
```

```
In [ ]:
```

- Datasource-OpenStreetMap-OSM/TM/SRTM

CHAPTER 3

3、入门教程

GDAL的Geometry使用:

http://nbviewer.jupyter.org/github/supergis/git_notebook/blob/master/gdal/gdal-geometry.ipynb

CHAPTER 4

高级数据分析

CHAPTER 5

Indices and tables

- genindex
- modindex
- search