DATA INDEXING AND SELECTION

Data Indexing and Selection

A Series object acts in many ways like a one dimensional NumPy array, and in many ways like a standard Python dictionary. It will help us to understand the patterns of data indexing and selection in these arrays.

•Series as dictionary

•Series as one-dimensional array

•Indexers: loc, iloc, and ix

Series as dictionary

Like a dictionary, the Series object provides a mapping from a collection of keys to a collection of values.

data = pd.Series([0.25, 0.5, 0.75, 1.0],

index=['a', 'b', 'c', 'd']) data

a 0.25

b 0.50

c 0.75

d 1.00

dtype: float64 data['b']

0.5

Examine the keys/indices and values

We can also use dictionary-like Python expressions and methods to examine the keys/indices and values

i. 'a' in data

True

ii. data.keys()

Index(['a', 'b', 'c', 'd'], dtype='object')

iii. list(data.items())

[('a', 0.25), ('b', 0.5), ('c', 0.75), ('d', 1.0)]

Modifying series object

Series objects can even be modified with a dictionary-like syntax. Just as you can extend a dictionary by assigning to a new key, you can extend a Series by assigning to a new index value.

data['e'] = 1.25 data

a 0.25

b 0.50

c 0.75

d 1.00

e 1.25

dtype: float64

Series as one-dimensional array

A Series builds on this dictionary-like interface and provides array-style item selection via the same basic mechanisms as NumPy arrays—that is, slices, masking, and fancy indexing.

Slicing by explicit index

data['a':'c']

a 0.25

b 0.50

c 0.75

dtype: float64

Slicing by implicit integer index

data[0:2]

a 0.25

b 0.50

dtype: float64

Masking

data[(data > 0.3) & (data < 0.8)]

b 0.50

c 0.75

dtype: float64

Fancy indexing

data[['a', 'e']]

a 0.25

e 1.25

dtype: float64

Indexers: loc, iloc, and ix

Pandas provides some special indexer attributes that explicitly expose certain indexing schemes. These are not functional methods, but attributes that expose a particular slicing interface to the data in the Series.

data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5]) data

1 a

3 b

5 c

dtype: object

loc - the loc attribute allows indexing and slicing that always references the explicit index.

data.loc[1] 'a'

data.loc[1:3] 1 a

3 b

dtype: object

iloc - The iloc attribute allows indexing and slicing that always references the implicit Python-style index.

data.iloc[1] 'b'

data.iloc[1:3] 3 b

5 c

dtype: object

ix- ix is a hybrid of the two, and for Series objects is equivalent to standard [ ]-based indexing.

Data Selection in DataFrame

•DataFrame as a dictionary

•DataFrame as two-dimensional array

•Additional indexing conventions

DataFrame as a dictionary

The first analogy we will consider is the DataFrame as a dictionary of related Series objects.

The individual Series that make up the columns of the DataFrame can be accessed via dictionary-style indexing of the column name.

Dictionary-style indexing of the column name. result=pd.DataFrame({'DS':sub1,'FDS':sub2}) result*‘DS’+

	DS
sai	90
ram	85
kasim	92
tamil	89

Attribute-style access with column names that are strings

result.DS

	DS
sai	90
ram	85
kasim	92
tamil	89

Comparing attribute style and dictionary style accesses

result.DS is result*‘DS’+

True

Modify the object

Like with the Series objects this dictionary-style syntax can also be used to modify the object, in this case to add a new column:

result*‘TOTAL’+=result*‘DS’++result*‘FDS’+ result

	DS	FDS	TOTAL
sai	90	91	181

ram 85 95 180

kasim 92 89 181

tamil 89 90 179

DataFrame as two-dimensional array

•Transpose

We can transpose the full DataFrame to swap rows and columns.

result.T

DS sai

90 ram

85 kasim

92 tamil

89

FDS 91 95 89 90

TOTAL 181 180 181 179

Pandas again uses the loc, iloc, and ix indexers mentioned earlier. Using the iloc indexer, we can index the underlying array as if it is a simple NumPy array (using the implicit Python-style index), but the DataFrame index and column labels are maintained in the result

- loc

result.loc[: 'ram', : 'FDS']

DS FDS

sai 90 91

ram 85 95

- iloc

result.iloc[:2, :2]

DS FDS

sai 90 91

ram 85 95

- ix

result.ix[:2, :'FDS']

	DS	FDS
sai	90	91
ram	85	95

Masking and Fancy indexing

In the loc indexer we can combine masking and fancy indexing as in the following:

result.loc[result.total>180,['DS', 'FDS']]

	DS	FDS
sai	90	91
kasim	92	89

Modifying values

Indexing conventions may also be used to set or modify values; this is done in the standard way that you might be accustomed to from working with NumPy.

result.iloc[1,1] =70

		DS	FDS	TOTAL
sai		90	91	181
ram		85	70	180
kasim	92	89	181	
tamil	89	90	179	

Additional indexing conventions Slicing row wise

result['sai':'kasim']