Getting Started with pandas

Part 4

Essential Functionality

Part 2

Indexing, Selection, and Filtering

• Series indexing (obj [...]) works analogously to NumPy array indexing, except you can use the Series's index values instead of only integers.

```
In [85]: obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
         obj
Out[85]: a
              2.0
         dtype: float64
In [86]: obj['b']
Out[86]: 1.0
In [87]: obj[1]
Out[87]: 1.0
In [88]: obj[2:4]
Out[88]: c
         dtype: float64
```

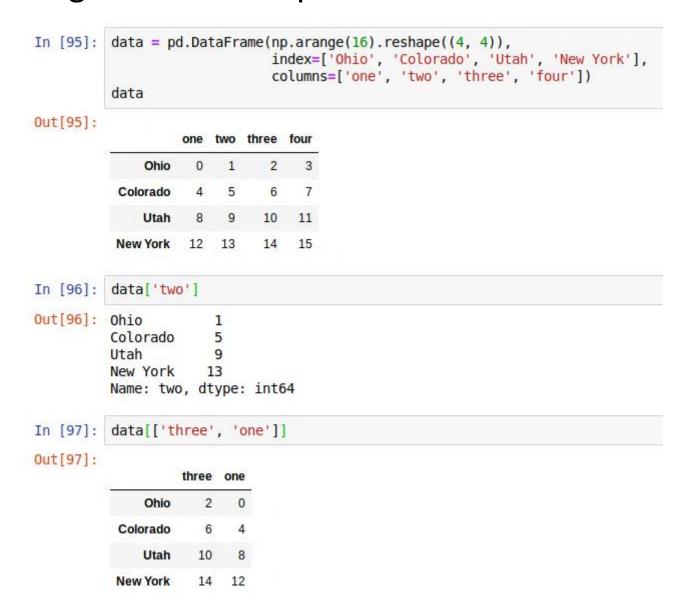
• Slicing with labels behaves differently than normal Python slicing in that the endpoint is inclusive:

• Setting using these methods modifies the corresponding section of the Series:

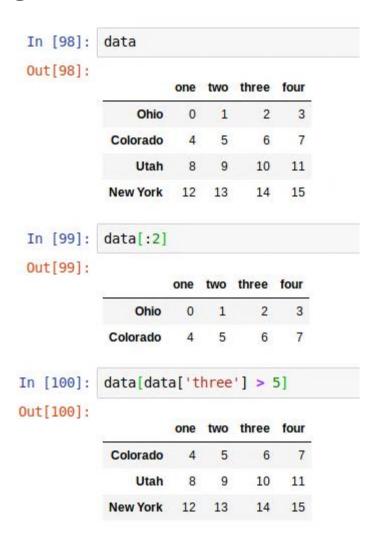
```
In [93]: obj
Out[93]: a    0.0
    b    1.0
    c    2.0
    d    3.0
    dtype: float64

In [94]: obj['b':'c'] = 5
    obj
Out[94]: a    0.0
    b    5.0
    c    5.0
    d    3.0
    dtype: float64
```

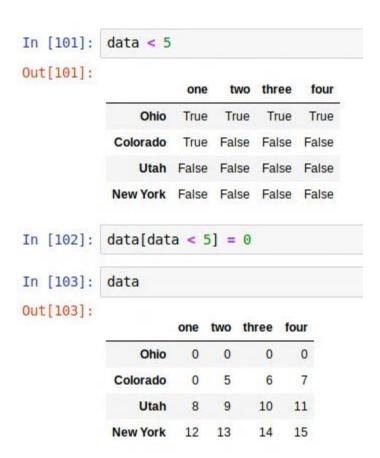
 Indexing into a DataFrame is for retrieving one or more columns either with a single value or sequence:



- Indexing like this has a few special cases.
- First, slicing or selecting data with a boolean array:



 Another use case is in indexing with a boolean DataFrame, such as one produced by a scalar comparison:



Selection with loc and iloc

- For DataFrame label-indexing on the rows, I introduce the special indexing operators loc and iloc.
- They enable you to select a subset of the rows and columns from a DataFrame with NumPy-like notation using either axis labels (loc) or integers (iloc).

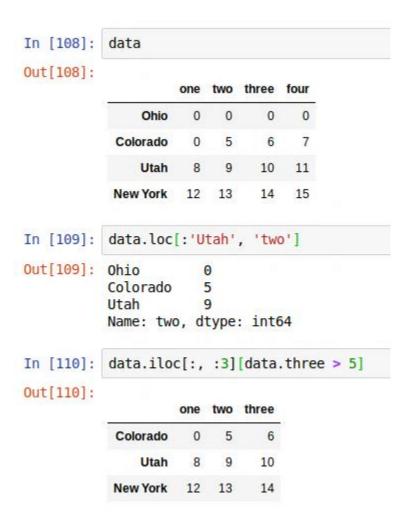
 As a preliminary example, let's select a single row and multiple columns by label:

```
In [104]: data.loc['Colorado', ['two', 'three']]
Out[104]: two    5
    three    6
    Name: Colorado, dtype: int64
```

• We'll then perform some similar selections with integers using iloc:

```
In [105]: data.iloc[2, [3, 0, 1]]
Out[105]:
                   11
           one
                    8
          Name: Utah, dtype: int64
In [106]: data.iloc[2]
Out[106]: one
           two
           three
                    10
                    11
           four
          Name: Utah, dtype: int64
In [107]:
          data.iloc[[1, 2], [3, 0, 1]]
Out[107]:
                   four one two
           Colorado
              Utah
                    11
                         8
```

 Both indexing functions work with slices in addition to single labels or lists of labels:



Integer Indexes

• Working with pandas objects indexed by integers is something that often trips up new users due to some differences with indexing semantics on built-in Python data structures like lists and tuples.

• For example, you might not expect the following code to generate an error:

```
In [111]: ser = pd.Series(np.arange(3.))
In [112]: ser
Out[112]: 0
               0.0
               1.0
               2.0
          dtype: float64
In [113]: ser[-1]
                                                    Traceback (most recent call last)
          KevError
          <ipython-input-113-44969a759c20> in <module>
          ----> 1 ser[-1]
          ~/anaconda3/lib/python3.7/site-packages/pandas/core/series.py in getitem (self, key)
                          key = com.apply if callable(key, self)
              866
              867
          --> 868
                              result = self.index.get value(self, key)
              869
                              if not is scalar(result):
              870
          ~/anaconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py in get value(self, series, key)
             4373
                          try:
             4374
                              return self. engine.get value(s, k,
          -> 4375
                                                            tz=getattr(series.dtype, 'tz', None))
                          except KeyError as el:
             4376
                              if len(self) > 0 and (self.holds integer() or self.is boolean()):
             4377
          pandas/ libs/index.pyx in pandas. libs.index.IndexEngine.get value()
          pandas/ libs/index.pyx in pandas. libs.index.IndexEngine.get value()
          pandas/ libs/index.pyx in pandas. libs.index.IndexEngine.get loc()
          pandas/ libs/hashtable class helper.pxi in pandas. libs.hashtable.Int64HashTable.get item()
```

- In this case, pandas could "fall back" on integer indexing, but it's difficult to do this in general without introducing subtle bugs.
- Here we have an index containing 0, 1, 2, but inferring what the user wants (label-based indexing or position-based) is difficult:

• On the other hand, with a non-integer index, there is no potential for ambiguity:

```
In [115]: ser2 = pd.Series(np.arange(3.), index=['a', 'b', 'c'])
In [116]: ser2
Out[116]: a    0.0
    b    1.0
    c    2.0
    dtype: float64

In [117]: ser2[-1]
Out[117]: 2.0
```

- To keep things consistent, if you have an axis index containing integers, data selection will always be label-oriented.
- For more precise handling, use loc (for labels) or iloc (for integers):

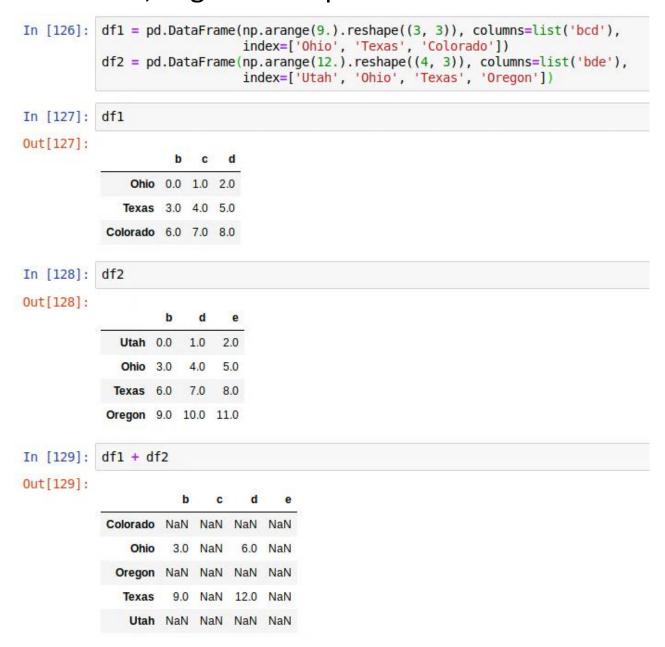
```
In [118]: ser
Out[118]: 0
               1.0
               2.0
          dtype: float64
In [119]: ser[:1]
Out[119]:
               0.0
          dtype: float64
In [120]: ser.loc[:1]
Out[120]: 0
               0.0
          dtype: float64
In [121]: ser.iloc[:1]
Out[121]:
               0.0
          dtype: float64
```

Arithmetic and Data Alignment

- An important pandas feature for some applications is the behavior of arithmetic between objects with different indexes.
- When you are adding together objects, if any index pairs are not the same, the respective index in the result will be the union of the index pairs.
- For users with database experience, this is similar to an automatic outer join on the index labels.

```
In [122]: s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])
          s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],
                         index=['a', 'c', 'e', 'f', 'g'])
In [123]: s1
Out[123]: a
              7.3
              -2.5
          C
              3.4
          d
             1.5
          dtype: float64
In [124]: s2
Out[124]: a
              -2.1
              3.6
             -1.5
              4.0
               3.1
          dtype: float64
In [125]: s1 + s2
Out[125]: a
               5.2
              1.1
              NaN
               0.0
              NaN
               NaN
          dtype: float64
```

• In the case of DataFrame, alignment is performed on both the rows and the columns:



• If you add DataFrame objects with no column or row labels in common, the result will contain all nulls:

```
In [130]: df1 = pd.DataFrame({'A': [1, 2]})
          df2 = pd.DataFrame(\{'B': [3, 4]\})
In [131]: df1
Out[131]:
           1 2
In [132]: df2
Out[132]:
          df1 + df2
In [133]:
Out[133]:
           0 NaN NaN
           1 NaN NaN
```