# Data exploration with Pandas

## CS424: Visualization & Visual Analytics

Fabio Miranda

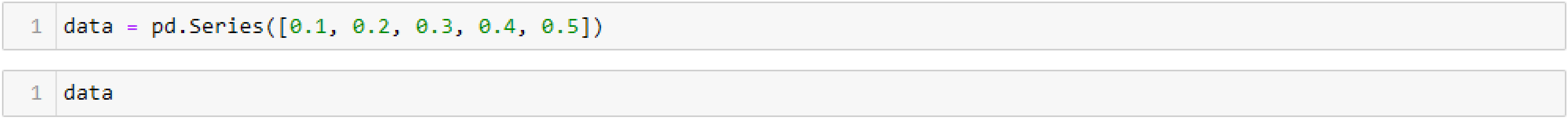
https://fmiranda.me

UIC COMPUTER SCIENCE

# Pandas

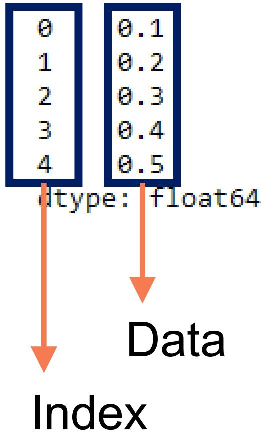
- Powerful Python package for manipulating tables.
- Built on top of numpy.
- Save time by abstracting lower-level code for manipulating, extracting, and deriving data tables.
- Easy & quick visualization with matplotlib.
- Main data structures: **Series** and **DataFrame**

# Simple series



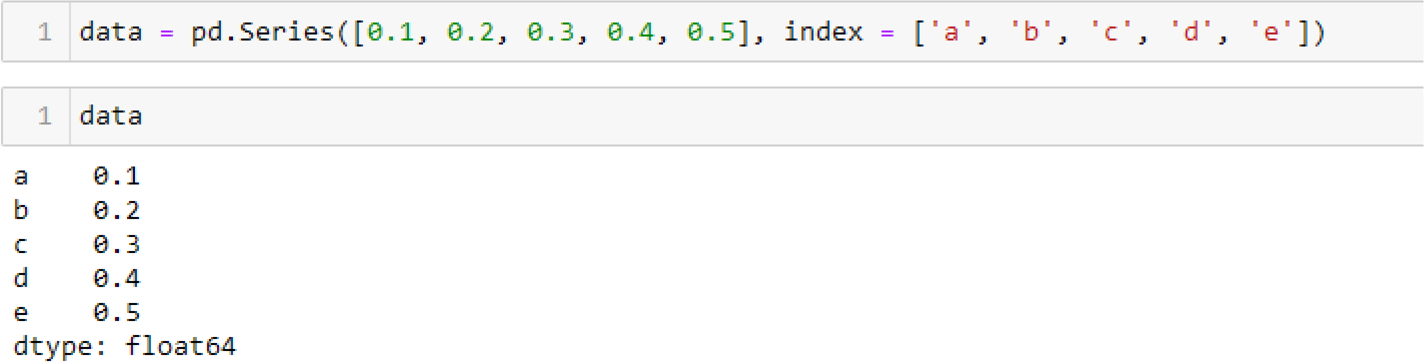
```
data = pd.Series([0.1, 0.2, 0.3, 0.4, 0.5])
```

```
data
```



```
0    0.1
1    0.2
2    0.3
3    0.4
4    0.5
dtype: float64
```

Explicit index



```
data = pd.Series([0.1, 0.2, 0.3, 0.4, 0.5], index = ['a', 'b', 'c', 'd', 'e'])
```

```
data
```

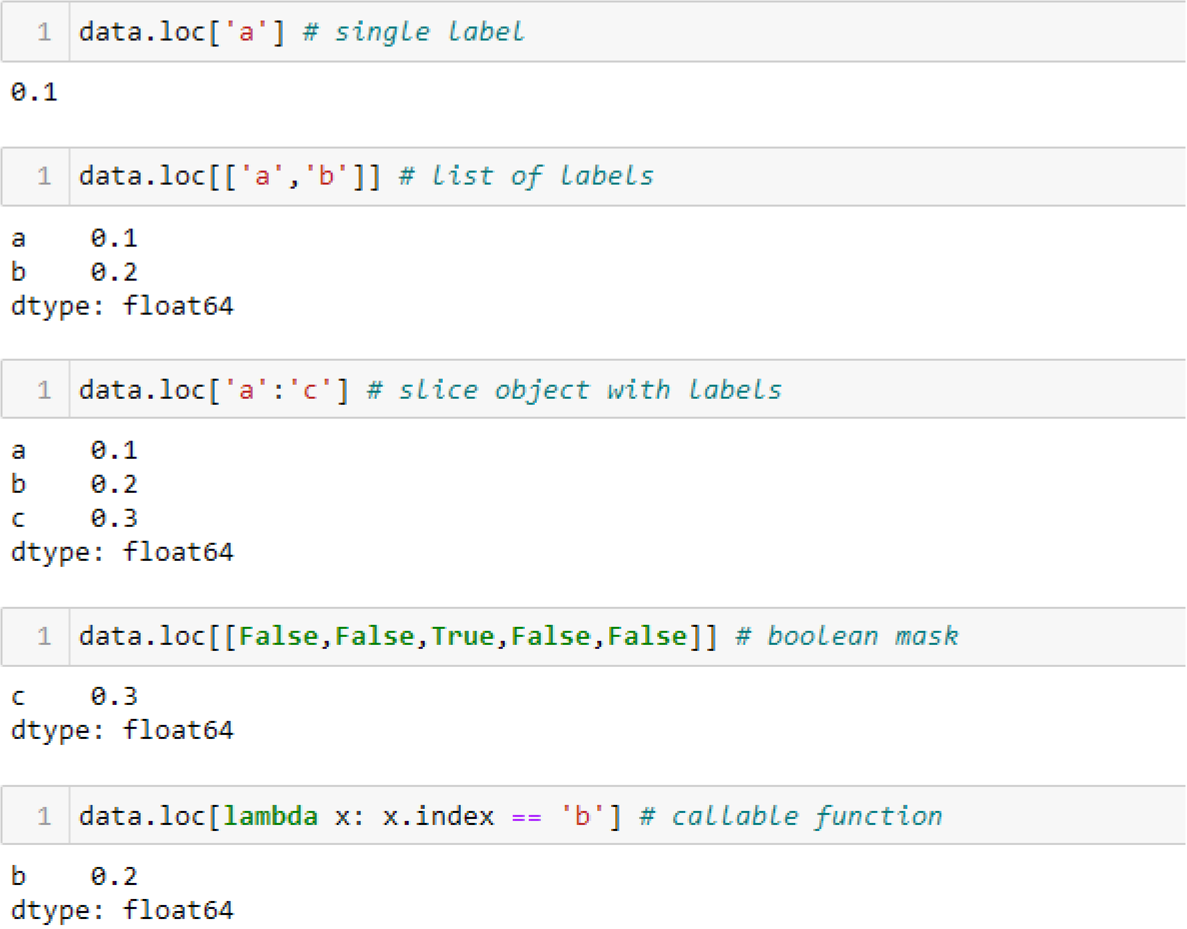
```
a    0.1
b    0.2
c    0.3
d    0.4
e    0.5
dtype: float64
```

# Indexing & accessing data

- .loc label based:
  - A single label
  - A list of array of labels
  - A slice object with labels
  - A boolean array
  - A callable function with one argument that returns valid output for indexing (one of the above).
- .iloc integer position based:
  - An integer
  - A list of array of integers
  - A slice object with integers
  - A boolean array
  - A callable function with one argument that returns valid output for indexing (one of the above)

# Indexing & accessing data

## Selection using label



```
data.loc['a'] # single label
```

```
0.1
```

```
data.loc[['a','b']] # list of labels
```

```
a    0.1
b    0.2
dtype: float64
```

```
data.loc['a':'c'] # slice object with labels
```

```
a    0.1
b    0.2
c    0.3
dtype: float64
```

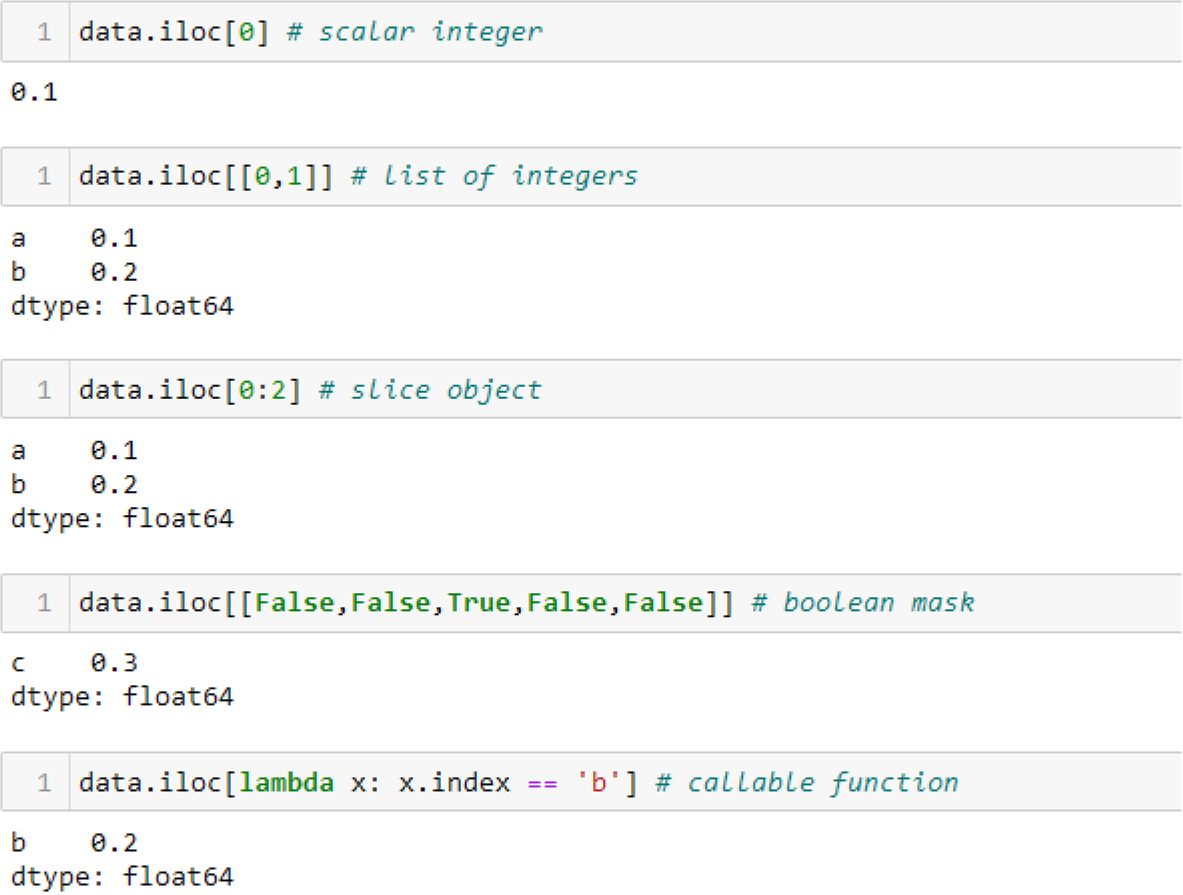
```
data.loc[[False,False,True,False,False]] # boolean mask
```

```
c    0.3
dtype: float64
```

```
data.loc[lambda x: x.index == 'b'] # callable function
```

```
b    0.2
dtype: float64
```

## Selection using integer position



```
data.iloc[0] # scalar integer
```

```
0.1
```

```
data.iloc[[0,1]] # list of integers
```

```
a    0.1
b    0.2
dtype: float64
```

```
data.iloc[0:2] # slice object
```

```
a    0.1
b    0.2
dtype: float64
```

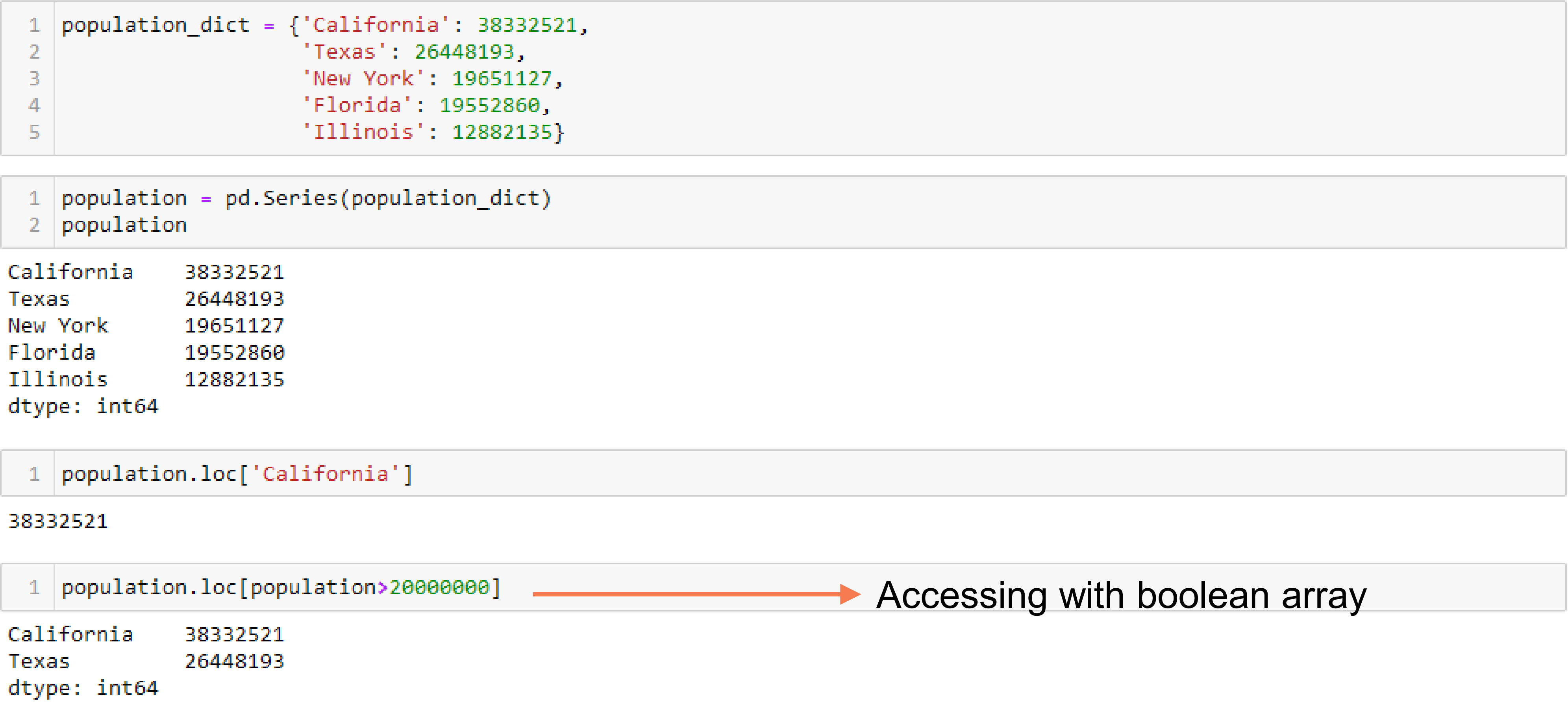
```
data.iloc[[False,False,True,False,False]] # boolean mask
```

```
c    0.3
dtype: float64
```

```
data.iloc[lambda x: x.index == 'b'] # callable function
```

```
b    0.2
dtype: float64
```

# Dictionary as a series



```python
population_dict = {'California': 38332521,
                   'Texas': 26448193,
                   'New York': 19651127,
                   'Florida': 19552860,
                   'Illinois': 12882135}
```

```python
population = pd.Series(population_dict)
population
```

```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

```python
population.loc['California']
```

```
38332521
```

```python
population.loc[population>20000000]
```

Accessing with boolean array

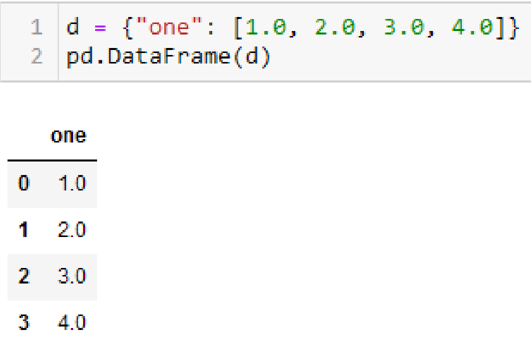
```
California    38332521
Texas         26448193
dtype: int64
```

# DataFrame object

- DataFrame is a 2-dimensional labeled data structure with columns of (potentially) different types.
  - Just like a spreadsheet or SQL table, or dict of Series objects.
- DataFrame can be created with:
  - Dict of 1D arrays, lists, dicts, or Series
  - 2D numpy array
  - Series
  - Another DataFrame

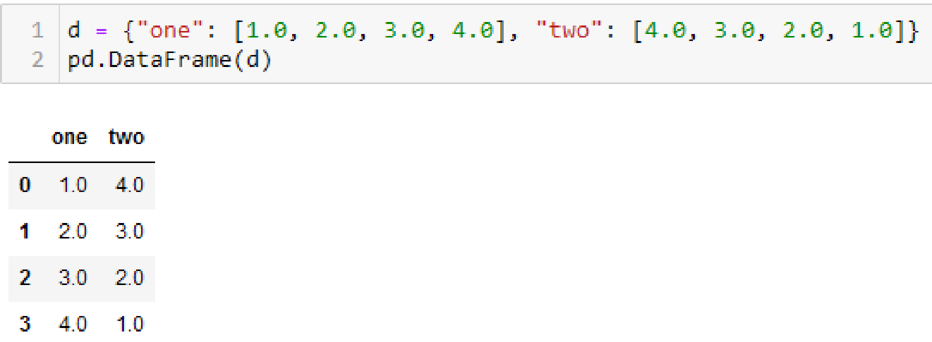
# Constructing a DataFrame

- From a dictionary or list of dictionaries:



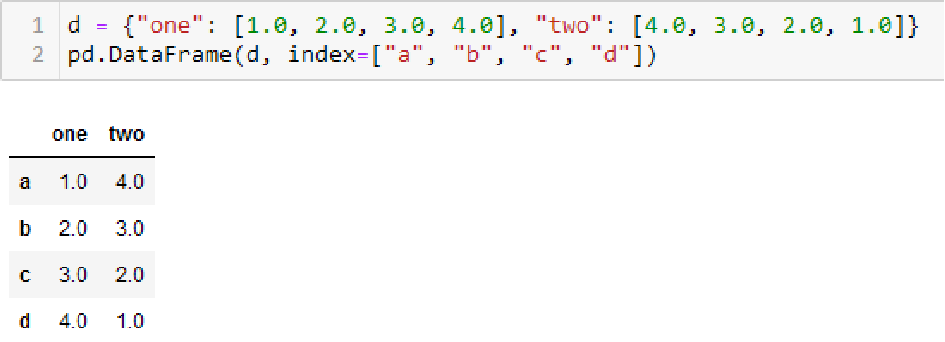
```python
d = {"one": [1.0, 2.0, 3.0, 4.0]}
pd.DataFrame(d)
```

| | one |
|---|---|
| **0** | 1.0 |
| **1** | 2.0 |
| **2** | 3.0 |
| **3** | 4.0 |



```python
d = {"one": [1.0, 2.0, 3.0, 4.0], "two": [4.0, 3.0, 2.0, 1.0]}
pd.DataFrame(d)
```

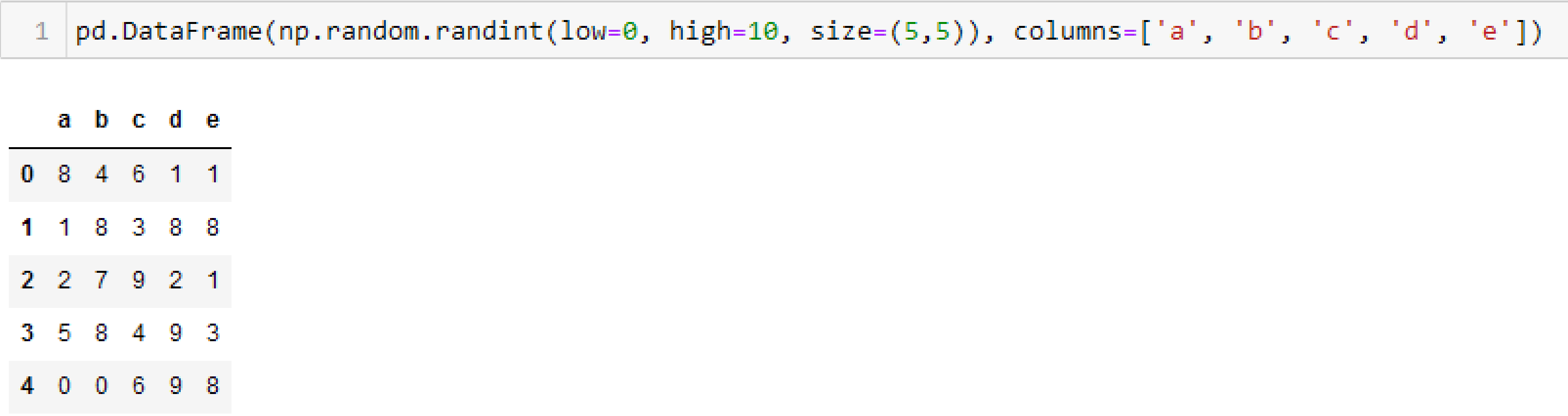
| | one | two |
|---|---|---|
| **0** | 1.0 | 4.0 |
| **1** | 2.0 | 3.0 |
| **2** | 3.0 | 2.0 |
| **3** | 4.0 | 1.0 |



```python
d = {"one": [1.0, 2.0, 3.0, 4.0], "two": [4.0, 3.0, 2.0, 1.0]}
pd.DataFrame(d, index=["a", "b", "c", "d"])
```

| | one | two |
|---|---|---|
| **a** | 1.0 | 4.0 |
| **b** | 2.0 | 3.0 |
| **c** | 3.0 | 2.0 |
| **d** | 4.0 | 1.0 |

- From numpy ndarray:

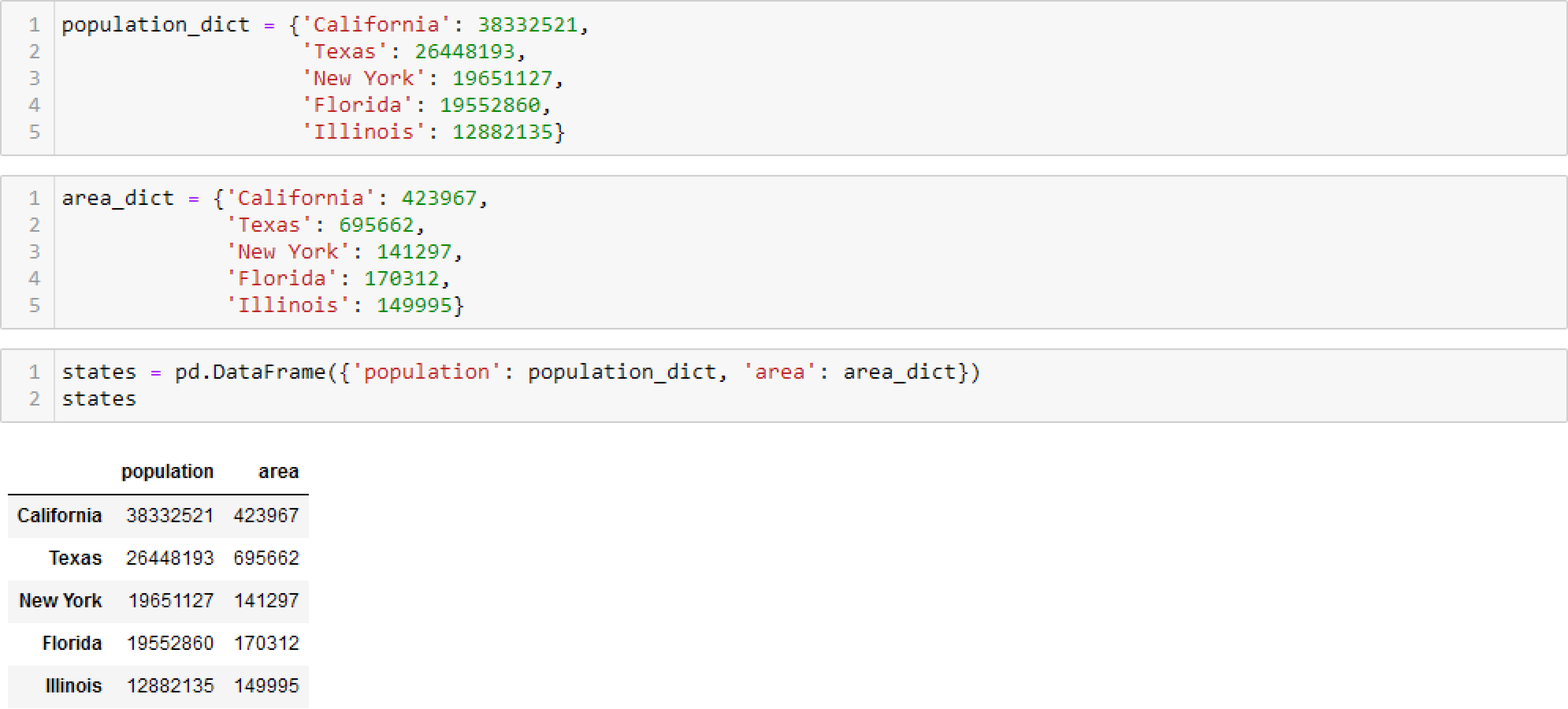


```python
pd.DataFrame(np.random.randint(low=0, high=10, size=(5,5)), columns=['a', 'b', 'c', 'd', 'e'])
```

| | a | b | c | d | e |
|---|---|---|---|---|---|
| **0** | 8 | 4 | 6 | 1 | 1 |
| **1** | 1 | 8 | 3 | 8 | 8 |
| **2** | 2 | 7 | 9 | 2 | 1 |
| **3** | 5 | 8 | 4 | 9 | 3 |
| **4** | 0 | 0 | 6 | 9 | 8 |

# Constructing a DataFrame

- From dictionaries or Series



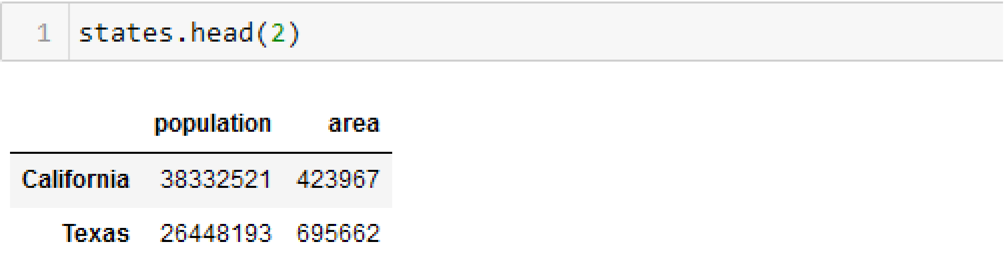
```python
population_dict = {'California': 38332521,
                   'Texas': 26448193,
                   'New York': 19651127,
                   'Florida': 19552860,
                   'Illinois': 12882135}
```

```python
area_dict = {'California': 423967,
             'Texas': 695662,
             'New York': 141297,
             'Florida': 170312,
             'Illinois': 149995}
```

```python
states = pd.DataFrame({'population': population_dict, 'area': area_dict})
states
```

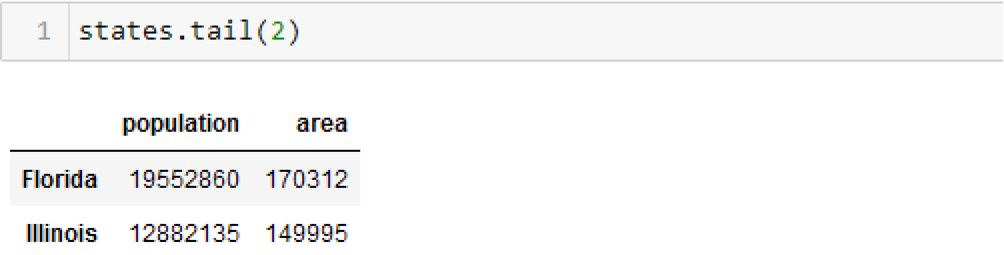
| | population | area |
|---|---|---|
| **California** | 38332521 | 423967 |
| **Texas** | 26448193 | 695662 |
| **New York** | 19651127 | 141297 |
| **Florida** | 19552860 | 170312 |
| **Illinois** | 12882135 | 149995 |

# Viewing data & statistics



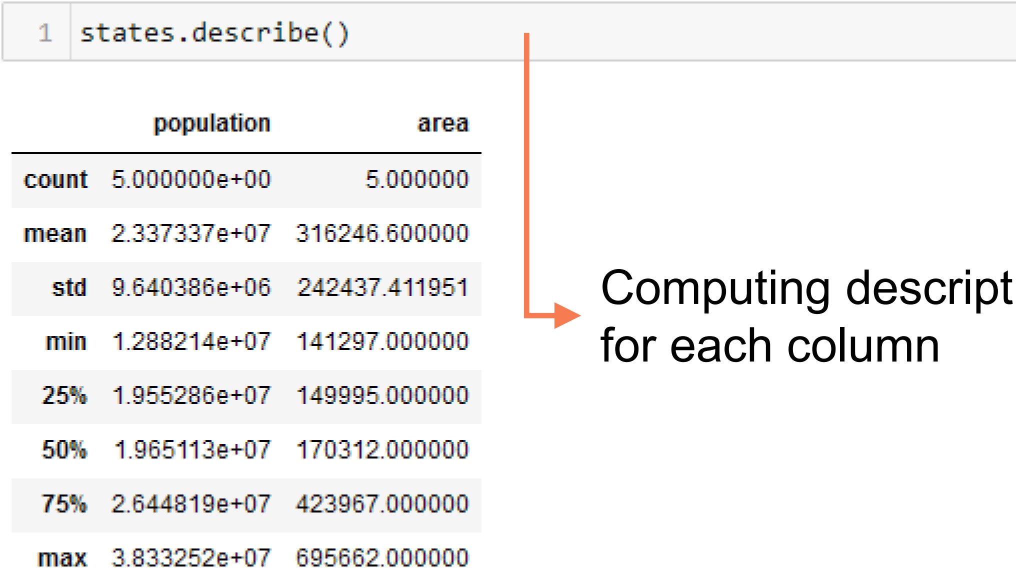
```python
states.head(2)
```

|  | population | area |
|---|---|---|
| California | 38332521 | 423967 |
| Texas | 26448193 | 695662 |



```python
states.tail(2)
```

|  | population | area |
|---|---|---|
| Florida | 19552860 | 170312 |
| Illinois | 12882135 | 149995 |

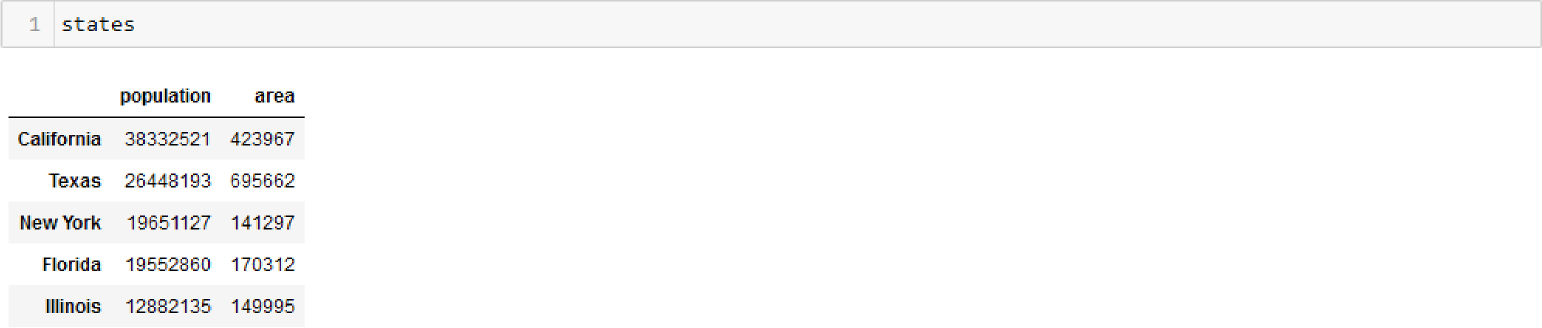


```python
states.describe()
```

|  | population | area |
|---|---|---|
| count | 5.000000e+00 | 5.000000 |
| mean | 2.337337e+07 | 316246.600000 |
| std | 9.640386e+06 | 242437.411951 |
| min | 1.288214e+07 | 141297.000000 |
| 25% | 1.955286e+07 | 149995.000000 |
| 50% | 1.965113e+07 | 170312.000000 |
| 75% | 2.644819e+07 | 423967.000000 |
| max | 3.833252e+07 | 695662.000000 |

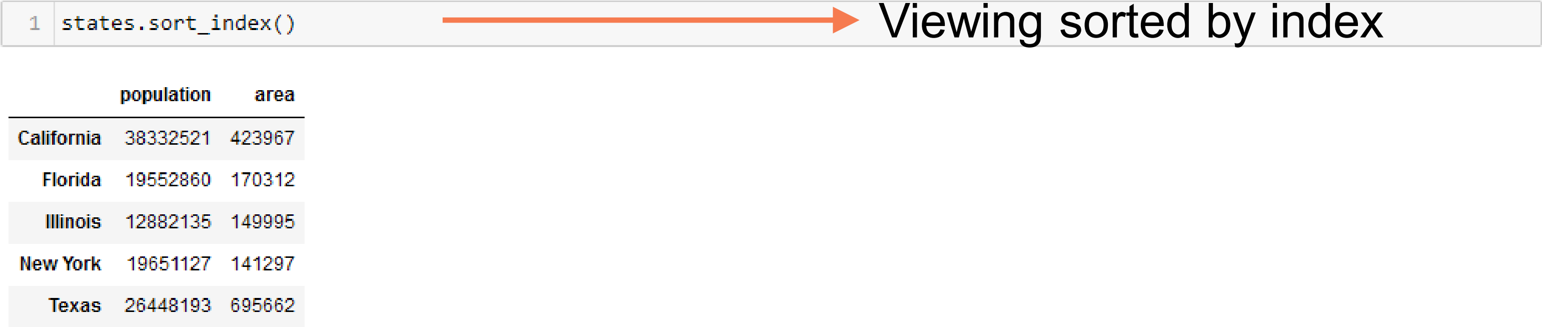
Computing descriptive stats for each column

# Viewing sorted DataFrame



```
states
```

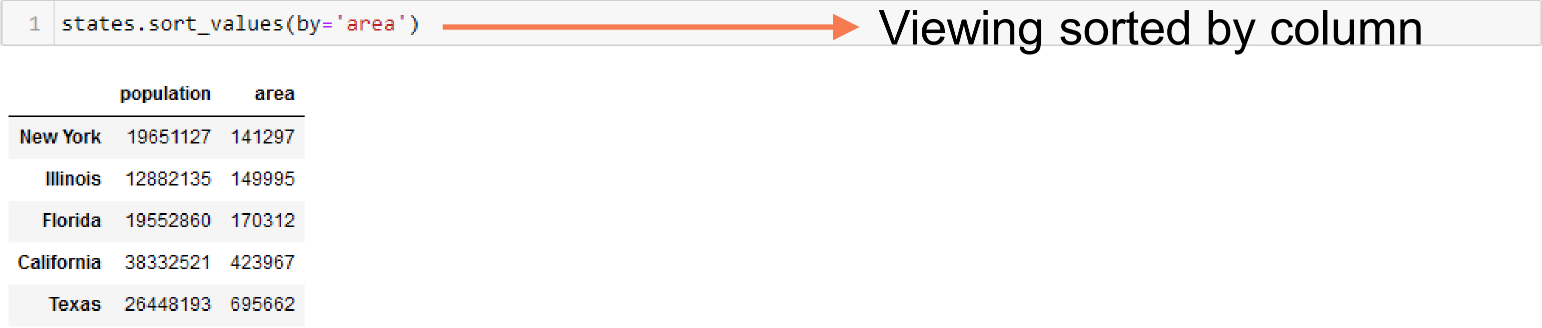
| | population | area |
|---|---|---|
| California | 38332521 | 423967 |
| Texas | 26448193 | 695662 |
| New York | 19651127 | 141297 |
| Florida | 19552860 | 170312 |
| Illinois | 12882135 | 149995 |



```
states.sort_index()
```

Viewing sorted by index

| | population | area |
|---|---|---|
| California | 38332521 | 423967 |
| Florida | 19552860 | 170312 |
| Illinois | 12882135 | 149995 |
| New York | 19651127 | 141297 |
| Texas | 26448193 | 695662 |



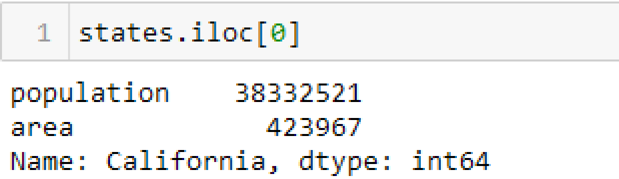
```
states.sort_values(by='area')
```

Viewing sorted by column

| | population | area |
|---|---|---|
| New York | 19651127 | 141297 |
| Illinois | 12882135 | 149995 |
| Florida | 19552860 | 170312 |
| California | 38332521 | 423967 |
| Texas | 26448193 | 695662 |

# Selecting & filtering data

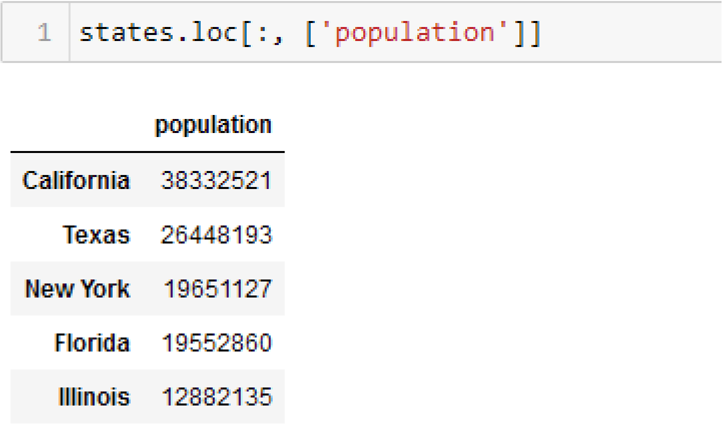
- Selection using integer position



```python
states.iloc[0]
```

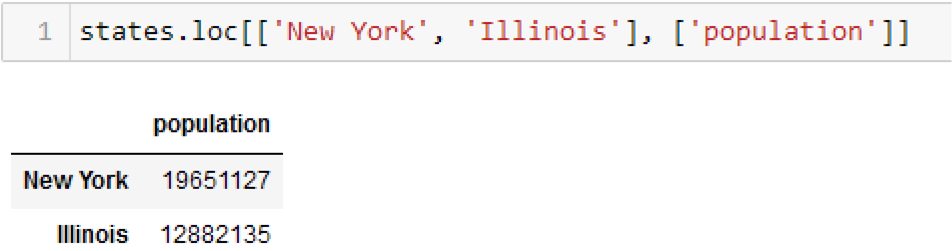
```
population    38332521
area            423967
Name: California, dtype: int64
```

- Multi-axis selection by label



```python
states.loc[:, ['population']]
```

| | population |
|---|---|
| California | 38332521 |
| Texas | 26448193 |
| New York | 19651127 |
| Florida | 19552860 |
| Illinois | 12882135 |

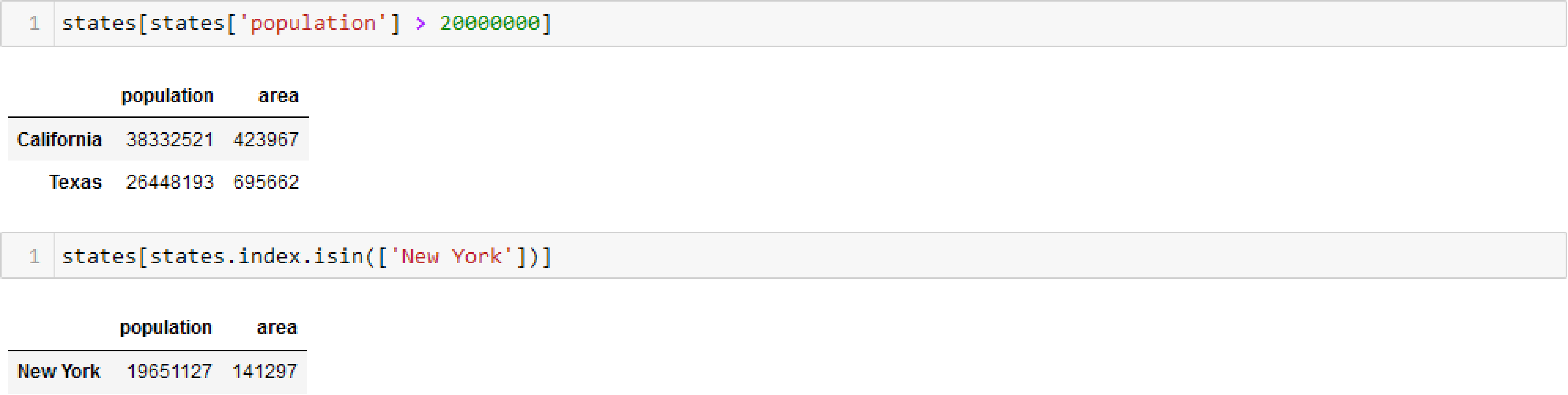


```python
states.loc[['New York', 'Illinois'], ['population']]
```

| | population |
|---|---|
| New York | 19651127 |
| Illinois | 12882135 |

# Selecting & filtering data

- Boolean indexing



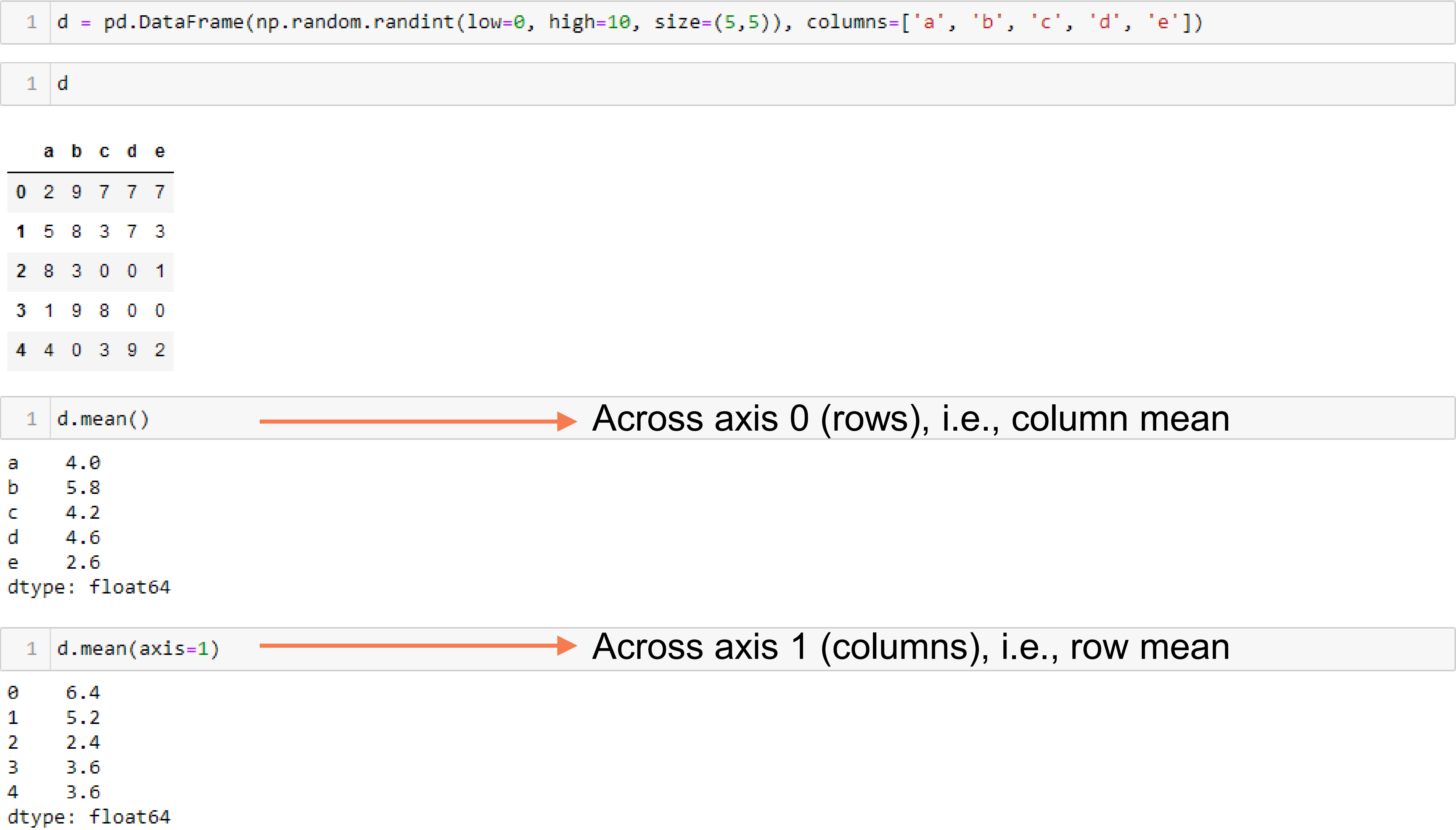
```python
states[states['population'] > 20000000]
```

|  | population | area |
|---|---|---|
| California | 38332521 | 423967 |
| Texas | 26448193 | 695662 |

```python
states[states.index.isin(['New York'])]
```

|  | population | area |
|---|---|---|
| New York | 19651127 | 141297 |

# Operations



```python
d = pd.DataFrame(np.random.randint(low=0, high=10, size=(5,5)), columns=['a', 'b', 'c', 'd', 'e'])
```

```python
d
```

|   | a | b | c | d | e |
|---|---|---|---|---|---|
| **0** | 2 | 9 | 7 | 7 | 7 |
| **1** | 5 | 8 | 3 | 7 | 3 |
| **2** | 8 | 3 | 0 | 0 | 1 |
| **3** | 1 | 9 | 8 | 0 | 0 |
| **4** | 4 | 0 | 3 | 9 | 2 |

```python
d.mean()
```

→ Across axis 0 (rows), i.e., column mean

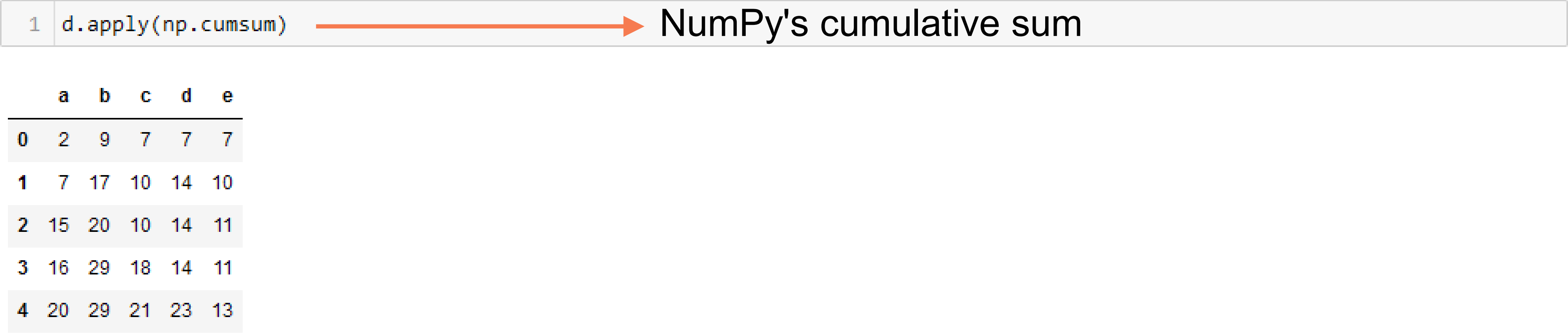
```
a    4.0
b    5.8
c    4.2
d    4.6
e    2.6
dtype: float64
```

```python
d.mean(axis=1)
```

→ Across axis 1 (columns), i.e., row mean

```
0    6.4
1    5.2
2    2.4
3    3.6
4    3.6
dtype: float64
```

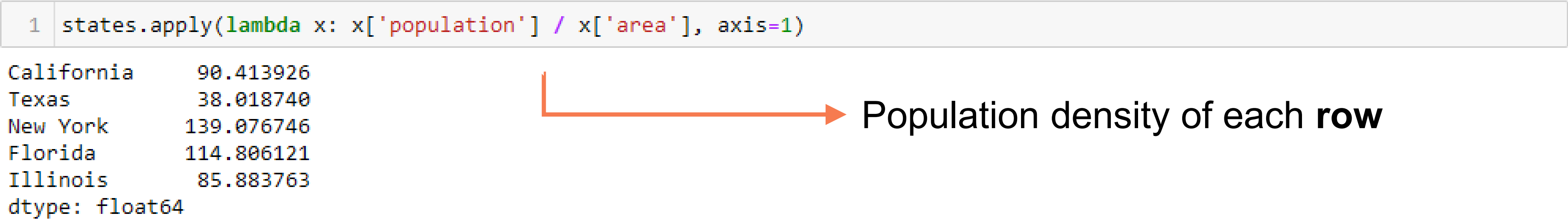
# Operations



```
d.apply(np.cumsum)
```

NumPy's cumulative sum

|   | a | b | c | d | e |
|---|---|---|---|---|---|
| **0** | 2 | 9 | 7 | 7 | 7 |
| **1** | 7 | 17 | 10 | 14 | 10 |
| **2** | 15 | 20 | 10 | 14 | 11 |
| **3** | 16 | 29 | 18 | 14 | 11 |
| **4** | 20 | 29 | 21 | 23 | 13 |

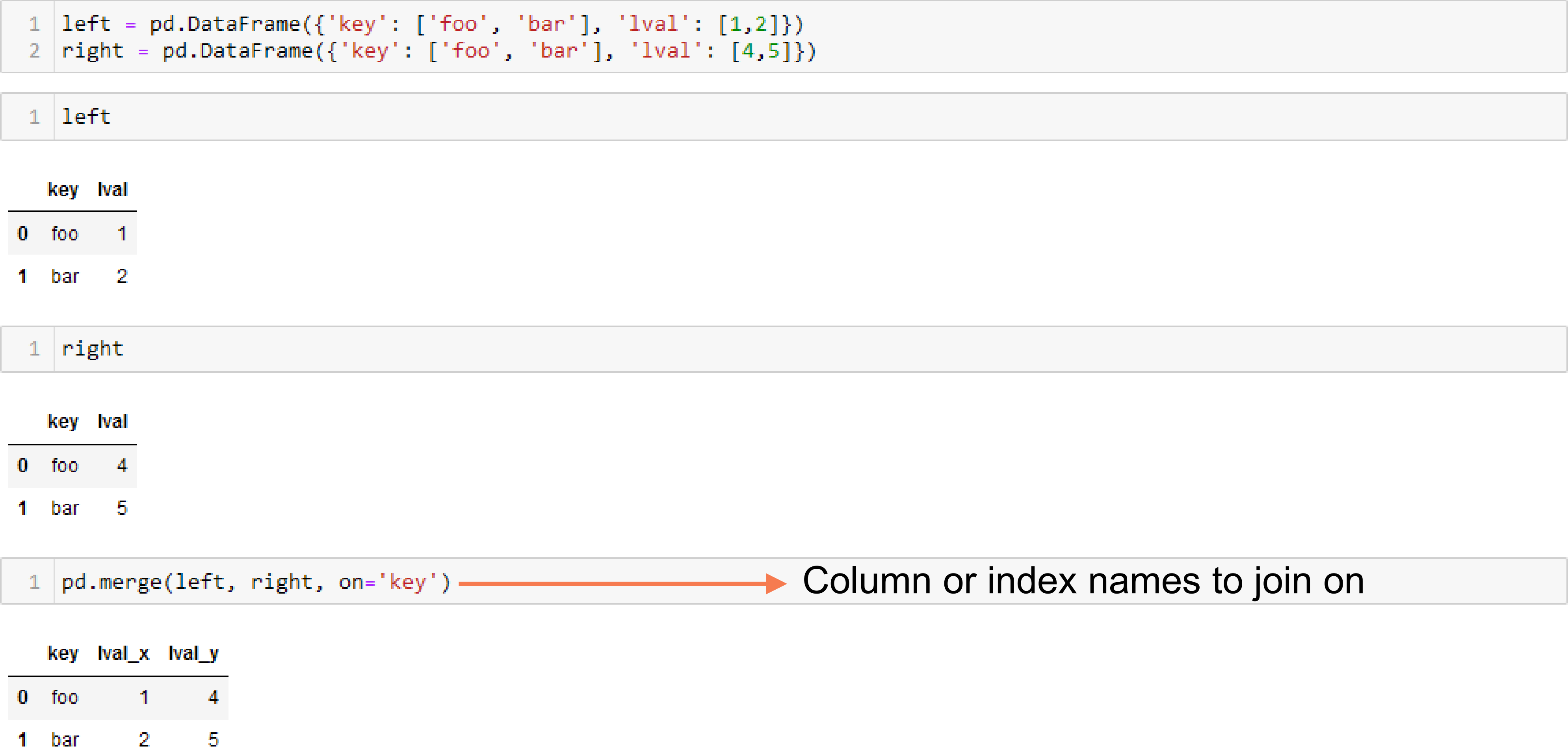


```
states.apply(lambda x: x['population'] / x['area'], axis=1)
```

```
California     90.413926
Texas          38.018740
New York      139.076746
Florida       114.806121
Illinois       85.883763
dtype: float64
```

Population density of each **row**

# Merging tables



```python
left = pd.DataFrame({'key': ['foo', 'bar'], 'lval': [1,2]})
right = pd.DataFrame({'key': ['foo', 'bar'], 'lval': [4,5]})
```

```python
left
```

|   | key | lval |
|---|---|---|
| 0 | foo | 1 |
| 1 | bar | 2 |

```python
right
```

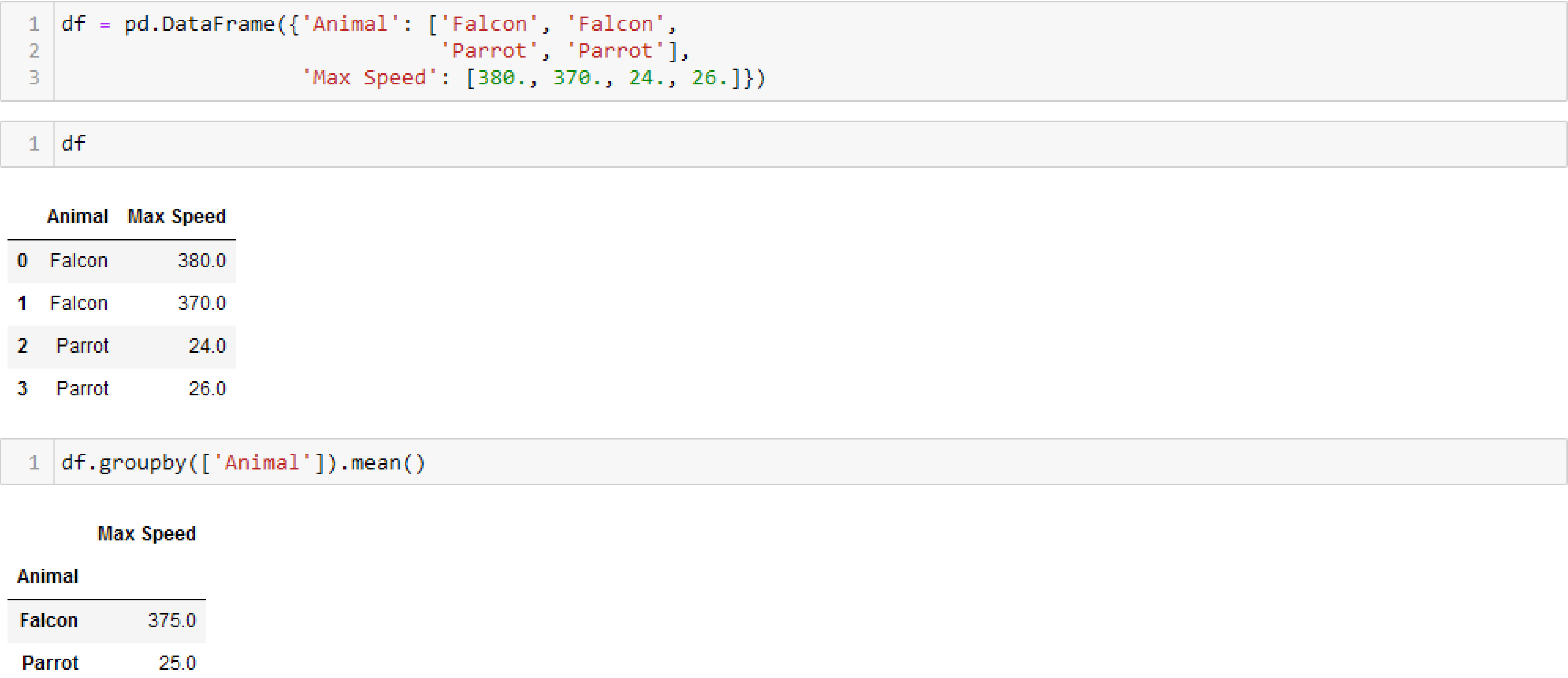
|   | key | lval |
|---|---|---|
| 0 | foo | 4 |
| 1 | bar | 5 |

```python
pd.merge(left, right, on='key')
```

Column or index names to join on

|   | key | lval_x | lval_y |
|---|---|---|---|
| 0 | foo | 1 | 4 |
| 1 | bar | 2 | 5 |

# Grouping



```python
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
                              'Parrot', 'Parrot'],
                   'Max Speed': [380., 370., 24., 26.]})
```

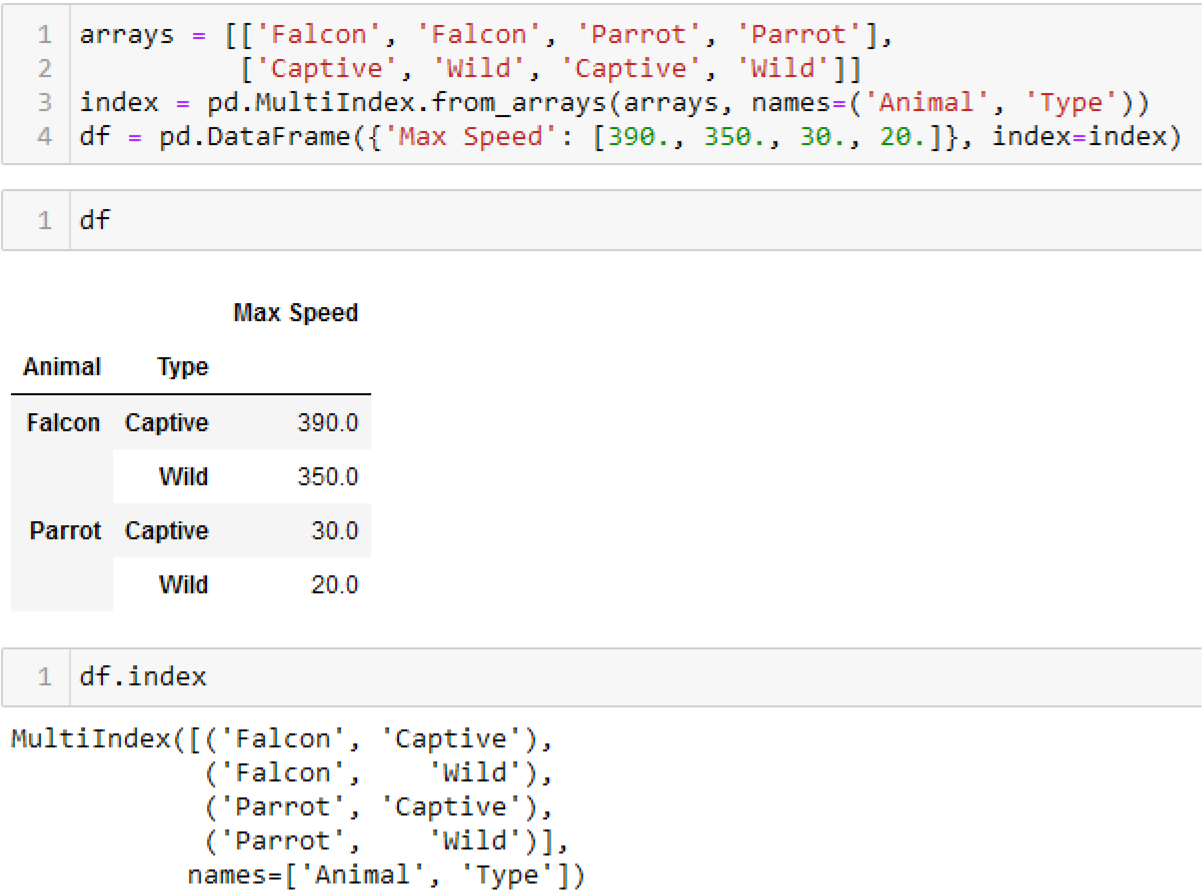
```python
df
```

|   | Animal | Max Speed |
|---|---|---|
| **0** | Falcon | 380.0 |
| **1** | Falcon | 370.0 |
| **2** | Parrot | 24.0 |
| **3** | Parrot | 26.0 |

```python
df.groupby(['Animal']).mean()
```

|   | Max Speed |
|---|---|
| **Animal** |   |
| **Falcon** | 375.0 |
| **Parrot** | 25.0 |

# Grouping



```python
arrays = [['Falcon', 'Falcon', 'Parrot', 'Parrot'],
          ['Captive', 'Wild', 'Captive', 'Wild']]
index = pd.MultiIndex.from_arrays(arrays, names=('Animal', 'Type'))
df = pd.DataFrame({'Max Speed': [390., 350., 30., 20.]}, index=index)
```

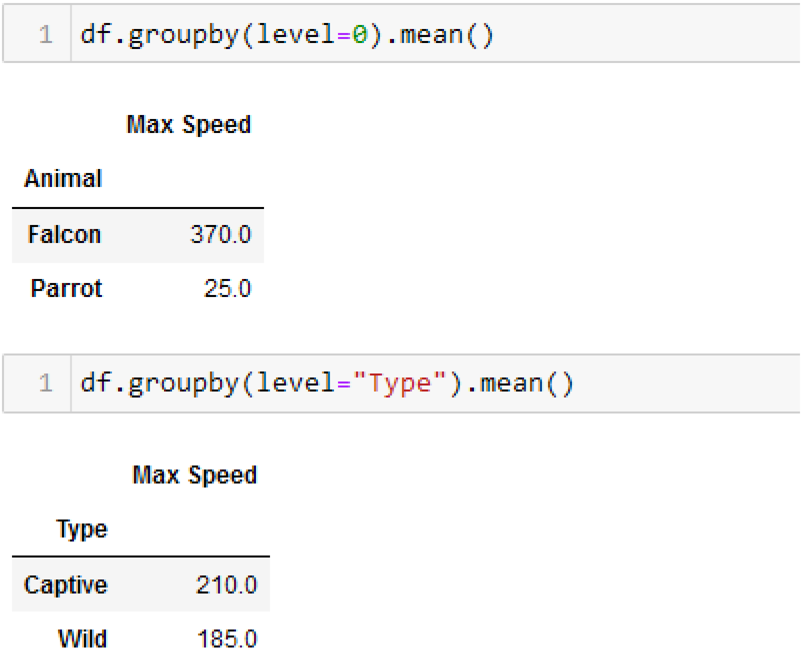
```python
df
```

| | | Max Speed |
|---|---|---|
| Animal | Type | |
| Falcon | Captive | 390.0 |
| | Wild | 350.0 |
| Parrot | Captive | 30.0 |
| | Wild | 20.0 |

```python
df.index
```

```
MultiIndex([('Falcon', 'Captive'),
            ('Falcon',    'Wild'),
            ('Parrot', 'Captive'),
            ('Parrot',    'Wild')],
           names=['Animal', 'Type'])
```

Grouping by index:



```python
df.groupby(level=0).mean()
```

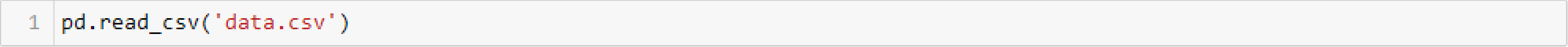
| | Max Speed |
|---|---|
| Animal | |
| Falcon | 370.0 |
| Parrot | 25.0 |

```python
df.groupby(level="Type").mean()
```

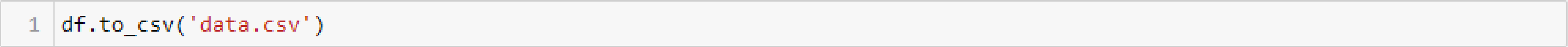
| | Max Speed |
|---|---|
| Type | |
| Captive | 210.0 |
| Wild | 185.0 |

# Importing & exporting data

- Reading and writing a CSV file:

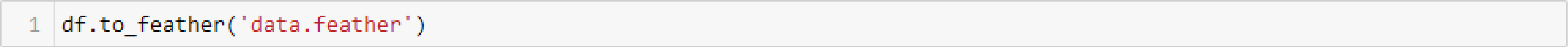


```
pd.read_csv('data.csv')
```



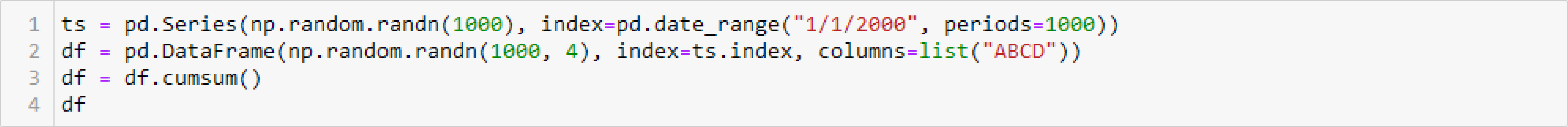
```
df.to_csv('data.csv')
```

- DataFrame to binary Feather format:

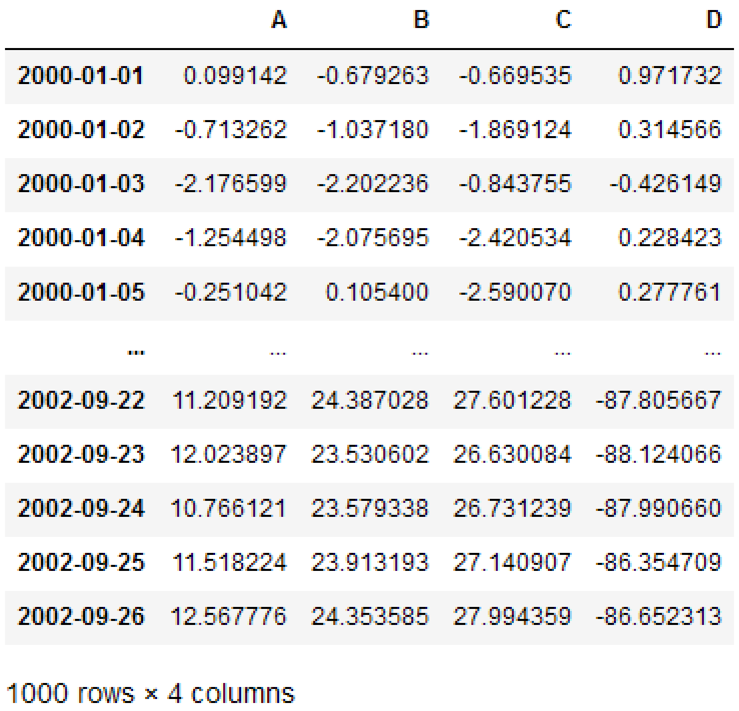


```
df.to_feather('data.feather')
```

# Basic plotting with matplotlib



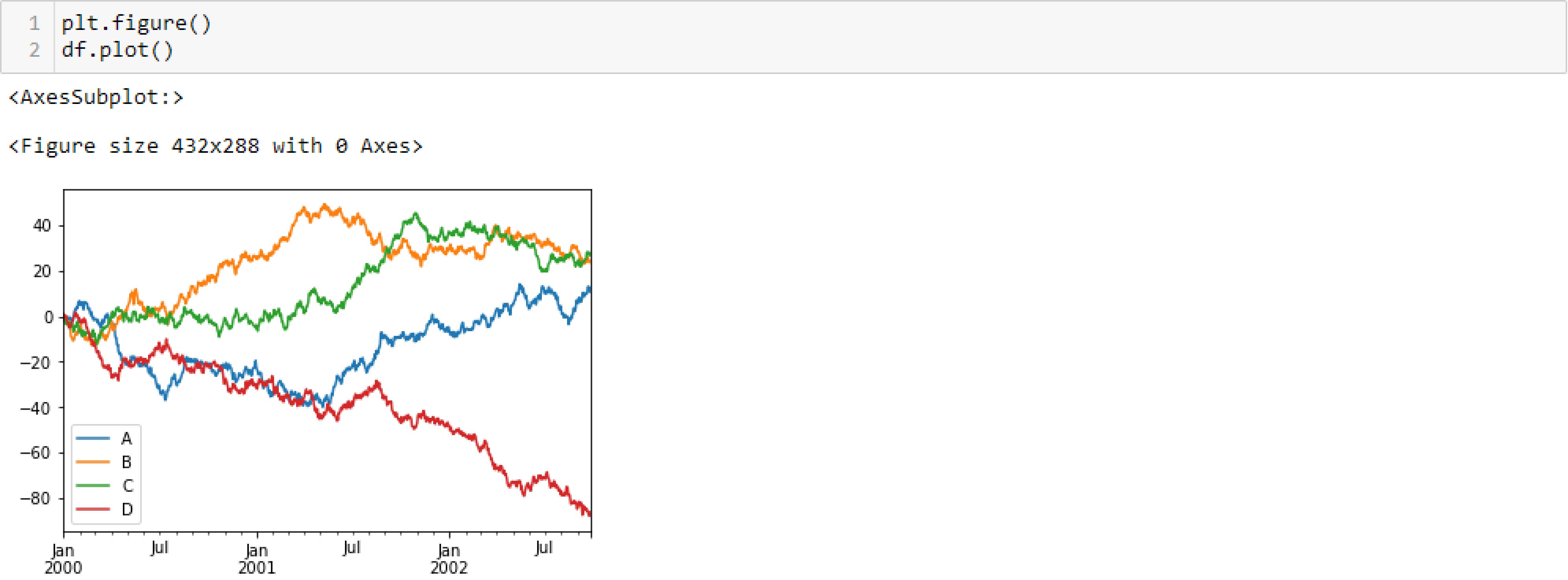
```
ts = pd.Series(np.random.randn(1000), index=pd.date_range("1/1/2000", periods=1000))
df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, columns=list("ABCD"))
df = df.cumsum()
df
```



| | A | B | C | D |
|---|---|---|---|---|
| **2000-01-01** | 0.099142 | -0.679263 | -0.669535 | 0.971732 |
| **2000-01-02** | -0.713262 | -1.037180 | -1.869124 | 0.314566 |
| **2000-01-03** | -2.176599 | -2.202236 | -0.843755 | -0.426149 |
| **2000-01-04** | -1.254498 | -2.075695 | -2.420534 | 0.228423 |
| **2000-01-05** | -0.251042 | 0.105400 | -2.590070 | 0.277761 |
| **...** | ... | ... | ... | ... |
| **2002-09-22** | 11.209192 | 24.387028 | 27.601228 | -87.805667 |
| **2002-09-23** | 12.023897 | 23.530602 | 26.630084 | -88.124066 |
| **2002-09-24** | 10.766121 | 23.579338 | 26.731239 | -87.990660 |
| **2002-09-25** | 11.518224 | 23.913193 | 27.140907 | -86.354709 |
| **2002-09-26** | 12.567776 | 24.353585 | 27.994359 | -86.652313 |

1000 rows × 4 columns

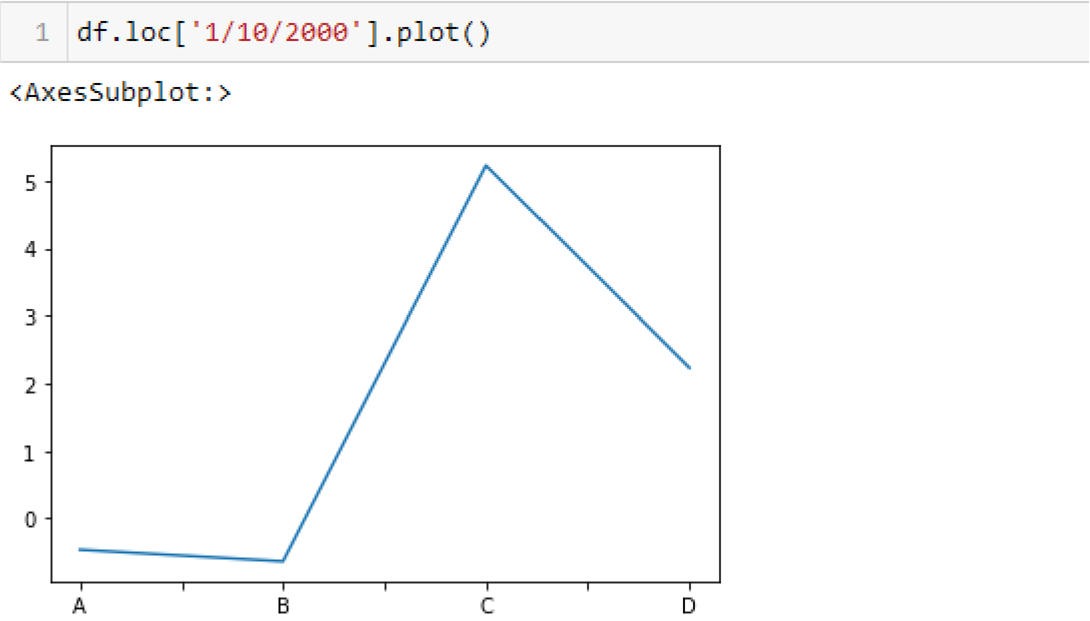
# Basic plotting with matplotlib



```
plt.figure()
df.plot()
```

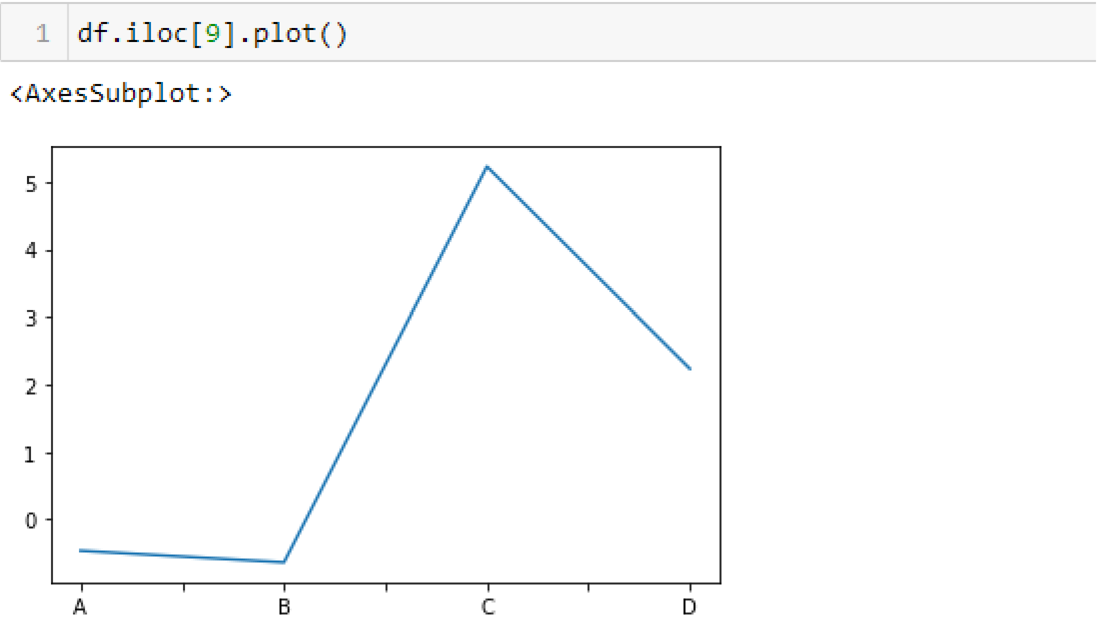
```
<AxesSubplot:>

<Figure size 432x288 with 0 Axes>
```

# Basic plotting with matplotlib



```
df.loc['1/10/2000'].plot()
```

```
<AxesSubplot:>
```

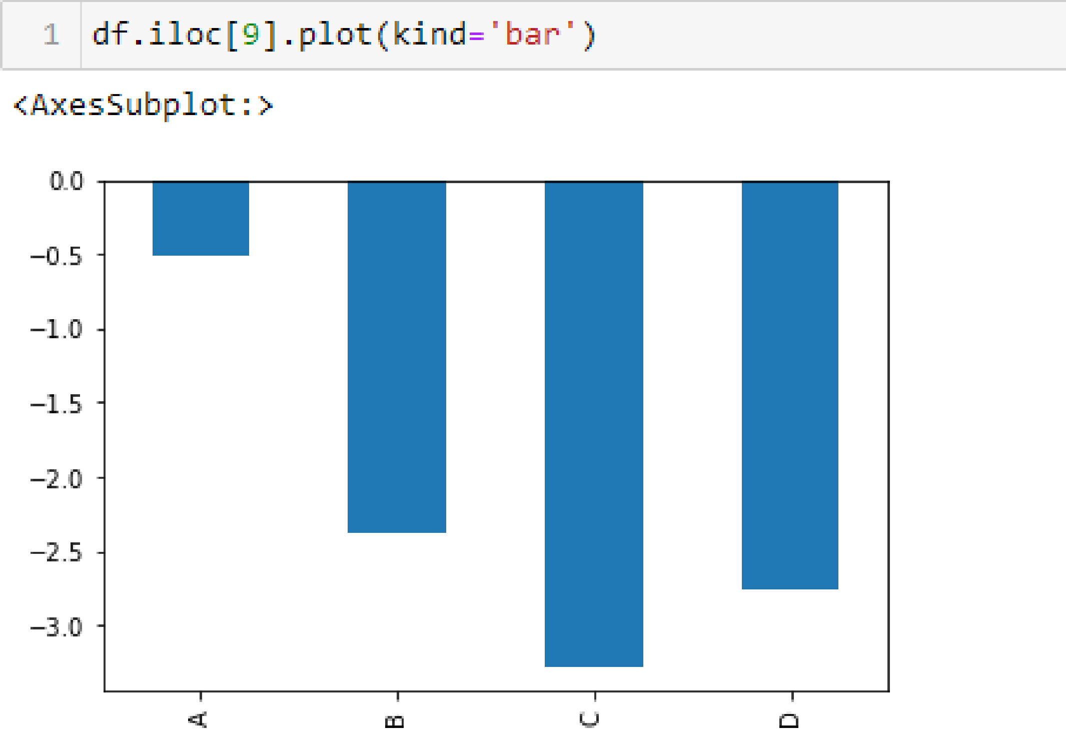


```
df.iloc[9].plot()
```

```
<AxesSubplot:>
```

# Basic plotting with matplotlib

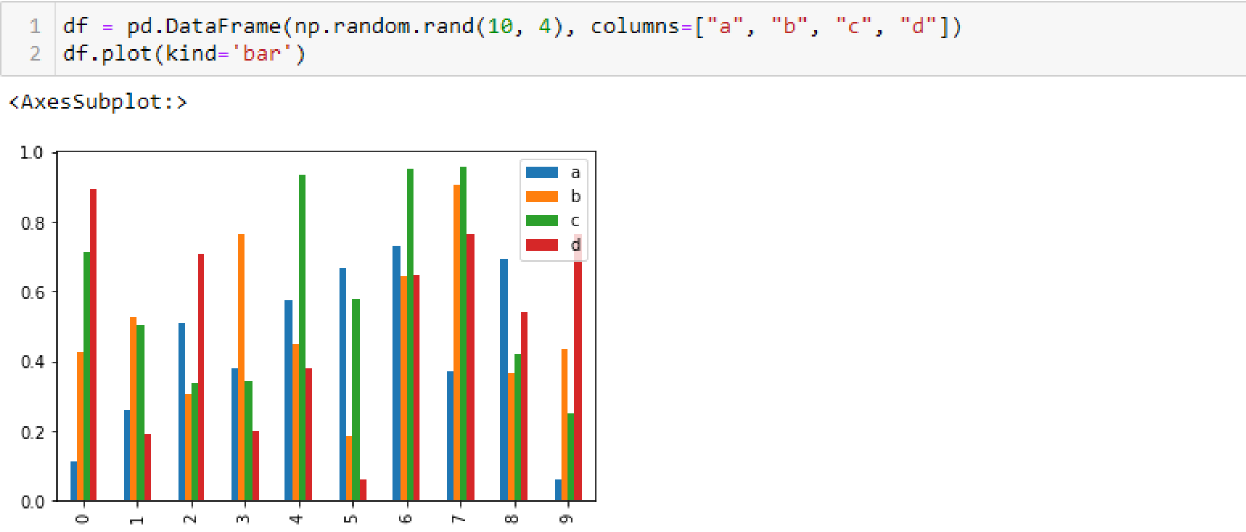
- *bar* for bar plots
- *hist* for histogram
- *box* for boxplot
- *kde* for density plots
- *area* for area plots
- *scatter* for scatter plots
- …



```
df.iloc[9].plot(kind='bar')
```

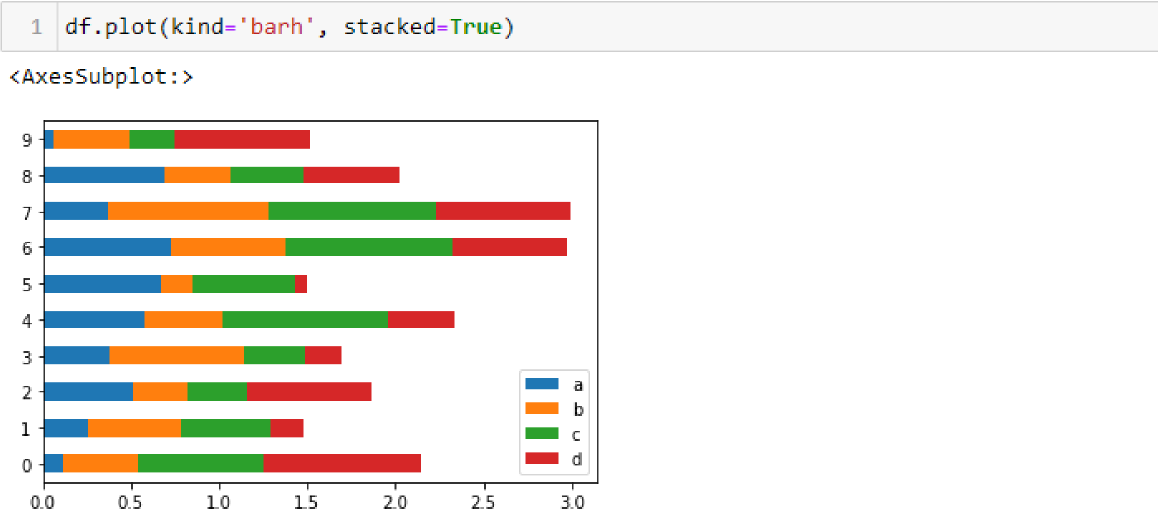
```
<AxesSubplot:>
```

# Basic plotting with matplotlib



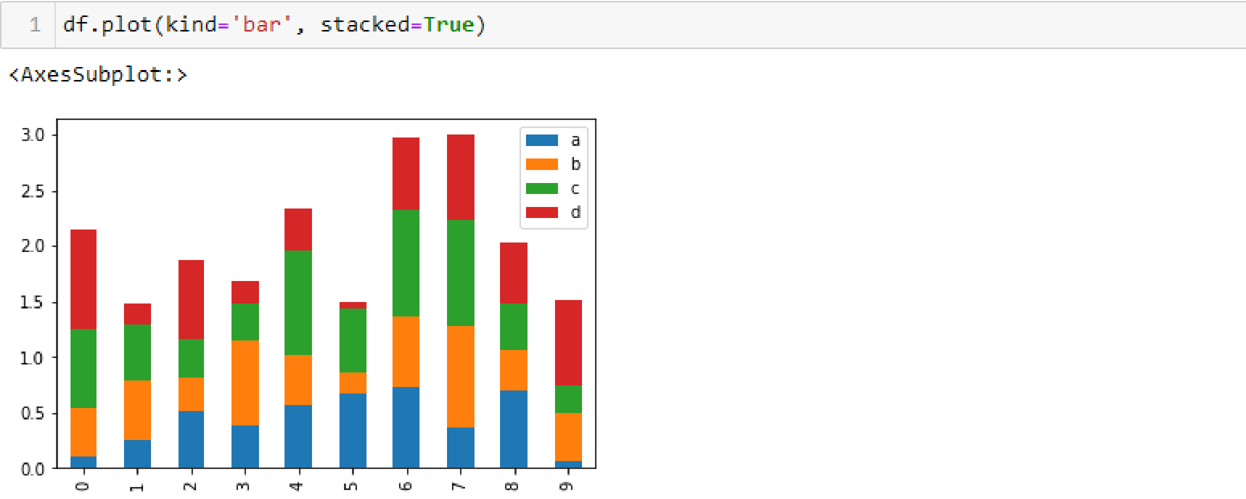
```
df = pd.DataFrame(np.random.rand(10, 4), columns=["a", "b", "c", "d"])
df.plot(kind='bar')
```

```
<AxesSubplot:>
```



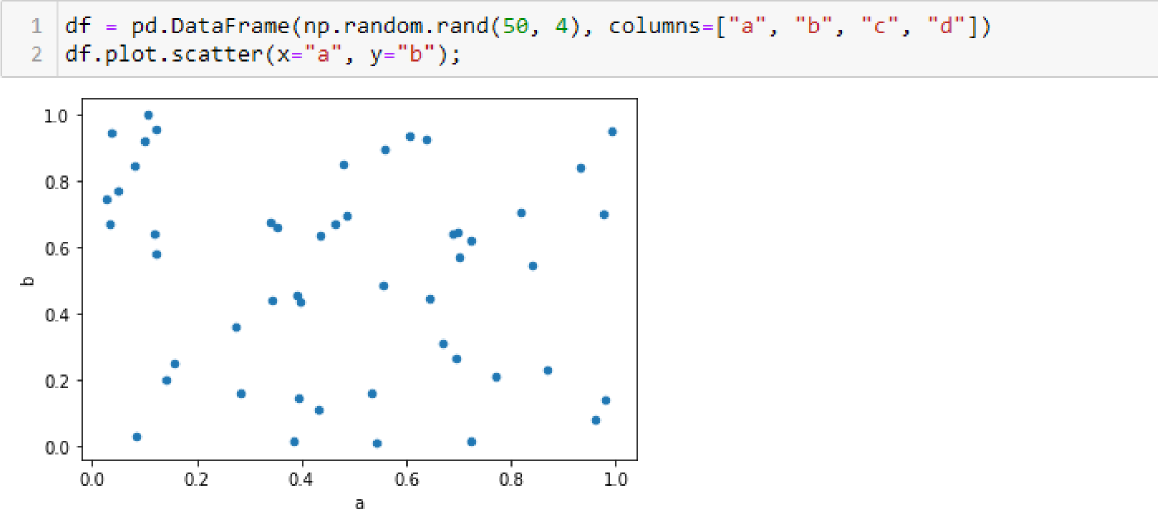
```
df.plot(kind='barh', stacked=True)
```

```
<AxesSubplot:>
```



```
df.plot(kind='bar', stacked=True)
```

```
<AxesSubplot:>
```



```
df = pd.DataFrame(np.random.rand(50, 4), columns=["a", "b", "c", "d"])
df.plot.scatter(x="a", y="b");
```