Advanced Data Management (CSCI 490/680)

Data Wrangling

Dr. David Koop

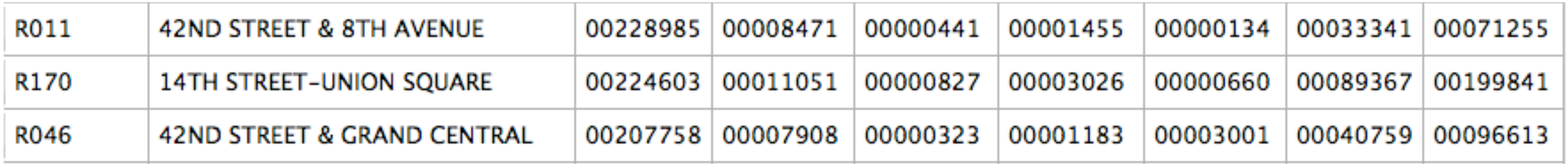
NumPy

- Fast **vectorized** array operations for data munging and cleaning, subsetting and filtering, transformation, and any other kinds of computations
- Common array algorithms like sorting, unique, and set operations
- Efficient descriptive statistics and aggregating/summarizing data
- Data alignment and relational data manipulations for merging and joining together heterogeneous data sets
- Expressing conditional logic as array expressions instead of loops with `if-elif-else` branches
- Group-wise data manipulations (aggregation, transformation, function application).

[W. McKinney, Python for Data Analysis]

Data

- What is this data?



R011	42ND STREET & 8TH AVENUE	00228985	00008471	00000441	00001455	00000134	00033341	00071255
R170	14TH STREET-UNION SQUARE	00224603	00011051	00000827	00003026	00000660	00089367	00199841
R046	42ND STREET & GRAND CENTRAL	00207758	00007908	00000323	00001183	00003001	00040759	00096613

- Semantics: real-world meaning of the data
- Type: structural or mathematical interpretation
- Both often require metadata
    - Sometimes we can infer some of this information
    - Line between data and metadata isn't always clear

Semantics

- The meaning of the data
- Example: 94023, 90210, 02747, 60115

Semantics

- The meaning of the data
- Example: 94023, 90210, 02747, 60115
  - Attendance at college football games?

Semantics

- The meaning of the data
- Example: 94023, 90210, 02747, 60115
  - Attendance at college football games?
  - Salaries?

Semantics

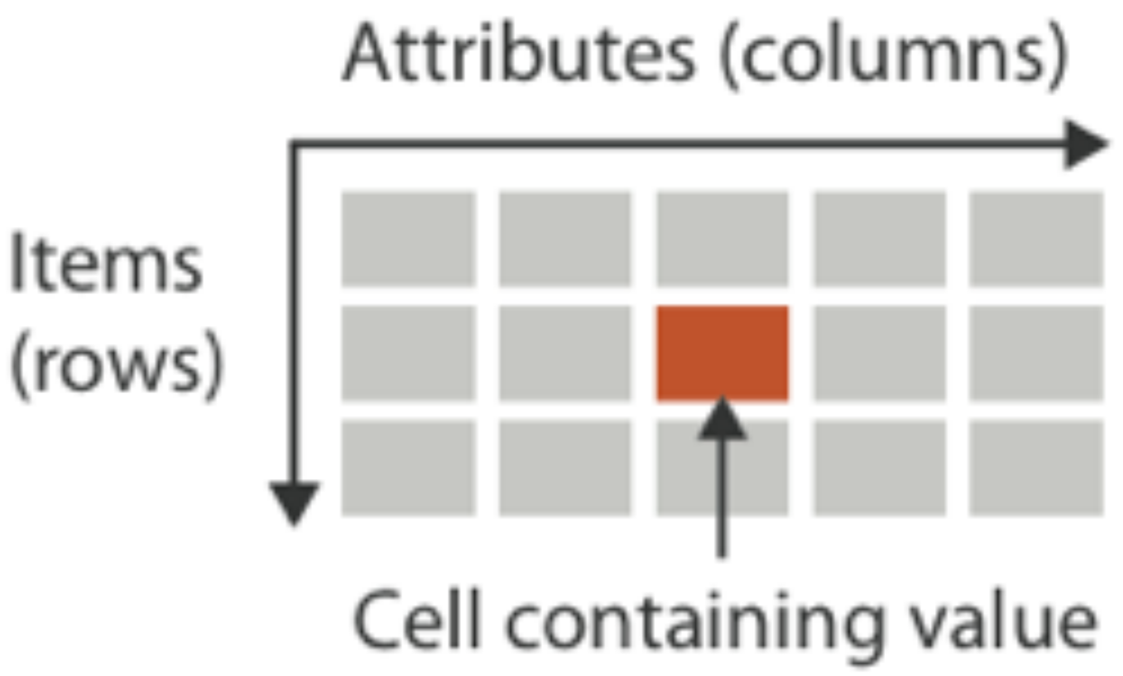
- The meaning of the data
- Example: 94023, 90210, 02747, 60115
  - Attendance at college football games?
  - Salaries?
  - Zip codes?
- Cannot always infer based on what the data looks like
- Often require semantics to better understand data, column names help
- May also include rules about data: a zip code is part of an address that uniquely identifies a residence
- Useful for asking good questions about the data

Data Terminology

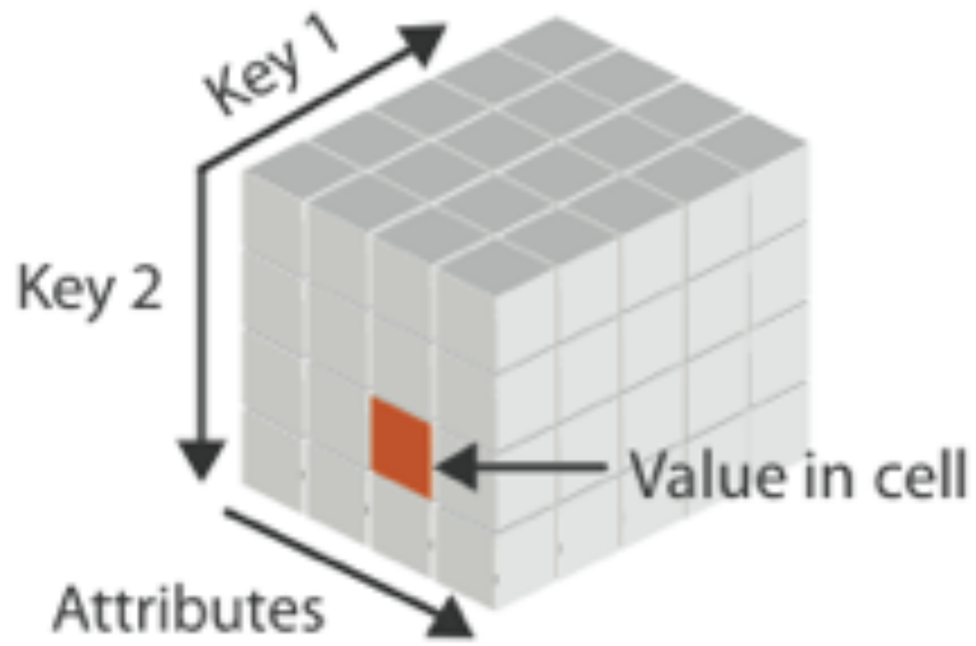
- Items
  - An **item** is an individual discrete entity
  - e.g., a row in a table
- Attributes
  - An **attribute** is some specific property that can be measured, observed, or logged
  - a.k.a. variable, (data) dimension
  - e.g., a column in a table

Tables

Flat



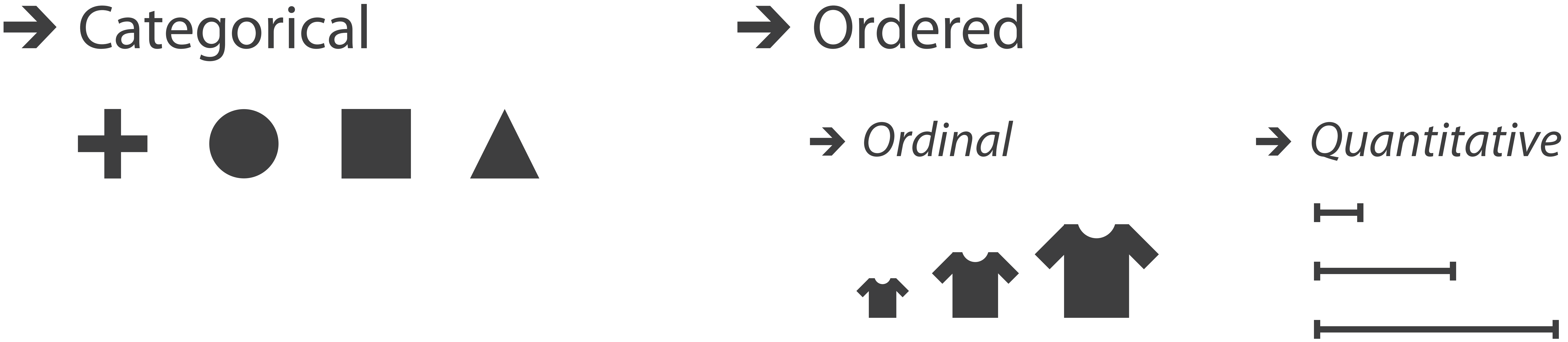
Multidimensional



- Data organized by rows & columns
  - row ~ item (usually)
  - column ~ attribute
  - label ~ attribute name
- Key: identifies each item (row)
  - Usually **unique**
  - Allows **join** of data from 2+ tables
  - Compound key: key split among multiple columns, e.g. (state, year) for population
- Multidimensional:
  - Split compound key

[Munzner (ill. Maguire), 2014]

Attribute Types



➔ Categorical

➔ Ordered

- ➔ *Ordinal*
- ➔ *Quantitative*

[Munzner (ill. Maguire), 2014]

Assignment 1

- Due today at 11:59pm
- Using Python for data analysis on Info Wanted ads
- Provided a1.ipynb file (right-click and download)
- Use basic python for now to demonstrate language knowledge
  - No pandas (for now)
- Use Anaconda or hosted Python environment
- Turn .ipynb file in via Blackboard
- Notes:
  - Bug in URL (https instead of http),
  - Bug in Problem 1 solution

Assignment 2

- Coming soon
- Similar to Assignment 1, now with pandas

Reading

- Wednesday
- Discussing paper:
  - "Wrangler: Interactive Visual Specification of Data Transformation Scripts"
  - Kandel et al.
  - http://vis.stanford.edu/files/wrangler.pdf
- Read
- Come prepared with questions, thoughts
  - Compare with how things work in pandas

pandas

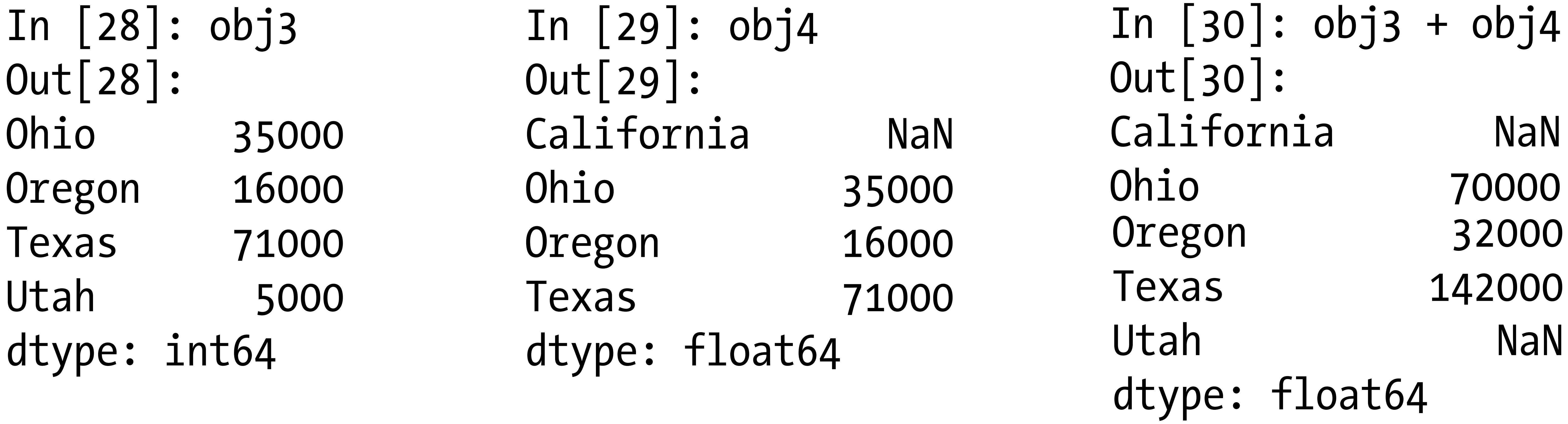
- Contains high-level data structures and manipulation tools designed to make data analysis fast and easy in Python
- Built on top of NumPy
- Requirements:
  - Data structures with labeled axes (aligning data)
  - Time series data
  - Arithmetic operations that include metadata (labels)
  - Handle missing data
  - Merge and relational operations

Series

- A one-dimensional array (with a type) with an **index**
- Index defaults to numbers but can also be text (like a dictionary)
- Allows easier reference to specific items
- `obj = pd.Series([7,14,-2,1])`
- Basically two arrays: `obj.values` and `obj.index`
- Can specify the index explicitly and use strings
-
  ```
  obj2 = pd.Series([4, 7, -5, 3],
                   index=['d', 'b', 'a', 'c'])
  ```
- Kind of like fixed-length, ordered dictionary + can create from a dictionary
-
  ```
  obj3 = pd.Series({'Ohio': 35000, 'Texas': 71000,
                    'Oregon': 16000, 'Utah': 5000})
  ```

Series

- Indexing: `s[1]` or `s['Oregon']`
- Can check for missing data: `pd.isnull(s)` or `pd.notnull(s)`
- Both index and values can have an associated name:
  - `s.name = 'population'; s.index.name = 'state'`
- Addition and NumPy ops work as expected and preserve the index-value link
- These operations **align**:



```
In [28]: obj3
Out[28]:
Ohio      35000
Oregon    16000
Texas     71000
Utah       5000
dtype: int64
```

```
In [29]: obj4
Out[29]:
California      NaN
Ohio          35000
Oregon        16000
Texas         71000
dtype: float64
```

```
In [30]: obj3 + obj4
Out[30]:
California      NaN
Ohio          70000
Oregon        32000
Texas        142000
Utah            NaN
dtype: float64
```

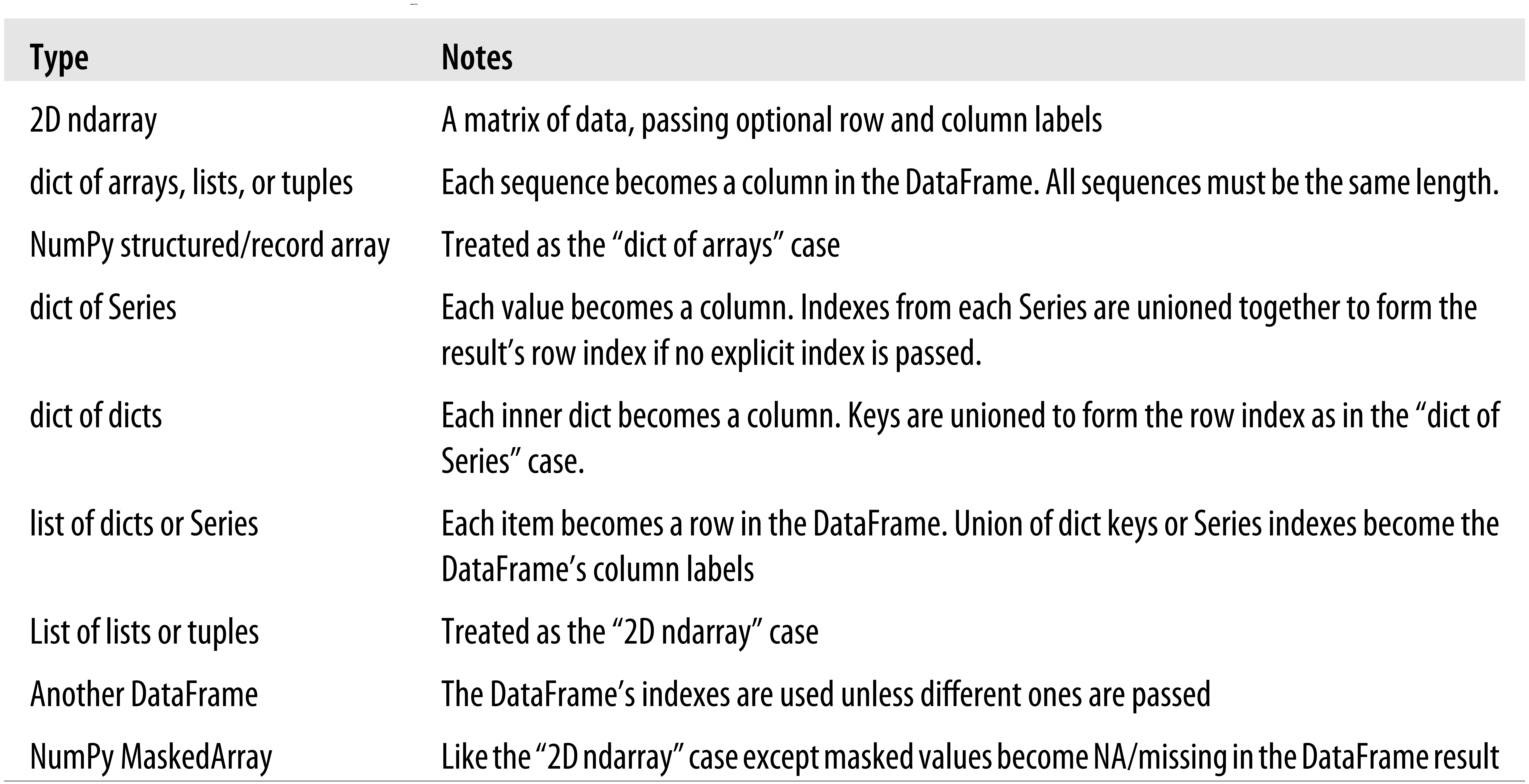
[W. McKinney, Python for Data Analysis]

Data Frame

- A dictionary of Series (labels for each series)
- A spreadsheet with column headers
- Has an index shared with each series
- Allows easy reference to any cell

-
  ```
  df = DataFrame({'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada'],
                  'year': [2000, 2001, 2002, 2001],
                  'pop': [1.5, 1.7, 3.6, 2.4]})
  ```
- Index is automatically assigned just as with a series but can be passed in as well via index kwarg
- Can reassign column names by passing columns kwarg

DataFrame Constructor Inputs



Type	Notes
2D ndarray	A matrix of data, passing optional row and column labels
dict of arrays, lists, or tuples	Each sequence becomes a column in the DataFrame. All sequences must be the same length.
NumPy structured/record array	Treated as the “dict of arrays” case
dict of Series	Each value becomes a column. Indexes from each Series are unioned together to form the result’s row index if no explicit index is passed.
dict of dicts	Each inner dict becomes a column. Keys are unioned to form the row index as in the “dict of Series” case.
list of dicts or Series	Each item becomes a row in the DataFrame. Union of dict keys or Series indexes become the DataFrame’s column labels
List of lists or tuples	Treated as the “2D ndarray” case
Another DataFrame	The DataFrame’s indexes are used unless different ones are passed
NumPy MaskedArray	Like the “2D ndarray” case except masked values become NA/missing in the DataFrame result

[W. McKinney, Python for Data Analysis]

DataFrame Access and Manipulation

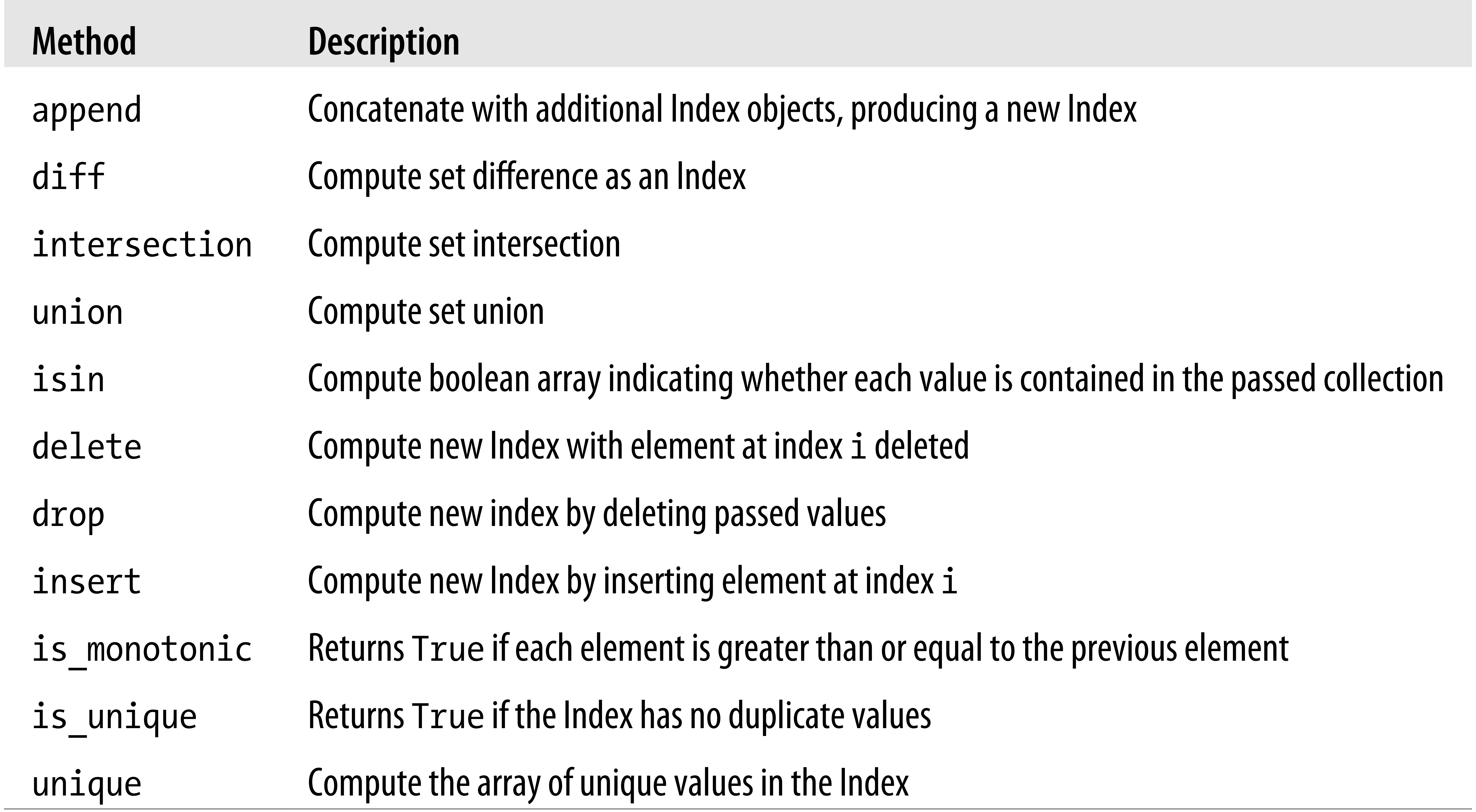
- `df.values → 2D NumPy array`
- Accessing a column:
  - `df["<column>"]`
  - `df.<column>`
  - Both return Series
  - Dot syntax only works when the column is a valid identifier
- Assigning to a column:
  - `df["<column>"] = <scalar> # all cells set to same value`
  - `df["<column>"] = <array>  # values set in order`

  -
    ```
    df["<column>"] = <series> # values set according to match
                              # between df and series indexes
    ```

DataFrame Index

- Similar to index for Series
- Immutable
- Can be shared with multiple structures (DataFrames or Series)
- `in` operator works with: `'Ohio' in df.index`

Index methods and properties



Method	Description
`append`	Concatenate with additional Index objects, producing a new Index
`diff`	Compute set difference as an Index
`intersection`	Compute set intersection
`union`	Compute set union
`isin`	Compute boolean array indicating whether each value is contained in the passed collection
`delete`	Compute new Index with element at index `i` deleted
`drop`	Compute new index by deleting passed values
`insert`	Compute new Index by inserting element at index `i`
`is_monotonic`	Returns `True` if each element is greater than or equal to the previous element
`is_unique`	Returns `True` if the Index has no duplicate values
`unique`	Compute the array of unique values in the Index

[W. McKinney, Python for Data Analysis]

Reindexing

- `reindex` creates a new object with the data conformed to new index
- `obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])`
- Missing values: handle with kwargs
  - `fill_value`: fill any missing value with a specific value
  - `method='ffill'`: fill values forward
  - `method='bfill'`: fill values backward
- Data Frames:
  - reindex rows as with series
  - reindex columns using columns kwarg

Dropping entries

- Can drop one or more entries
- Series:
  - `new_obj = obj.drop('c')`
  - `new_obj = obj.drop(['d', 'c'])`
- Data Frames:
  - `axis` keyword defines which axis to drop (default 0)
  - `axis=0` → rows, `axis=1` → columns
  - `axis = 'columns'`

Indexing

- Same as with NumPy arrays but can use Series's index labels
- Slicing with labels: NumPy is **exclusive**, Pandas is **inclusive**!

  -
    ```
    s = Series(np.arange(4))
    s[0:2] # gives two values like numpy
    ```

  -
    ```
    s = Series(np.arange(4), index=['a', 'b', 'c', 'd'])
    s['a':'c'] # gives three values, not two!
    ```
- Obtaining data subsets
  - `[]`: get columns by label
  - `loc`: get rows/cols by label
  - `iloc`: get rows/cols by position (integer index)

- For single cells (scalars), also have `at` and `iat`

Indexing

- `s = Series(np.arange(4.), index=[4,3,2,1])`
- `s[3]`
- `s.loc[3]`
- `s.iloc[3]`
- `s2 = pd.Series(np.arange(4), index=['a','b','c','d'])`
- `s2[3]`

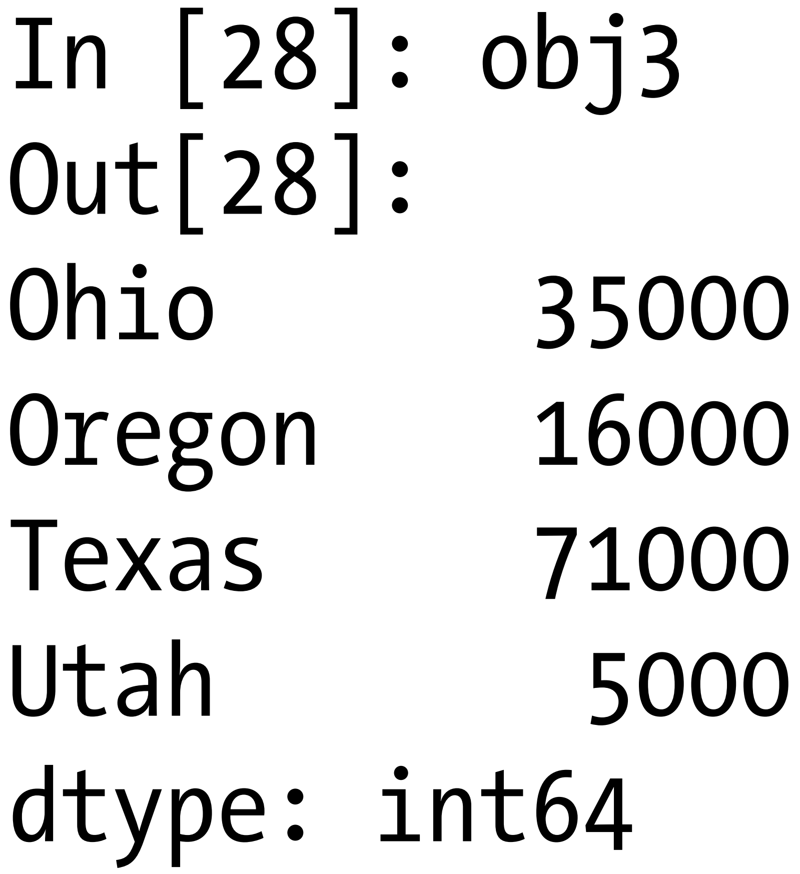
Filtering

- Same as with numpy arrays but allows use of column-based criteria
  - `data[data < 5] = 0`
  - `data[data['three'] > 5]`

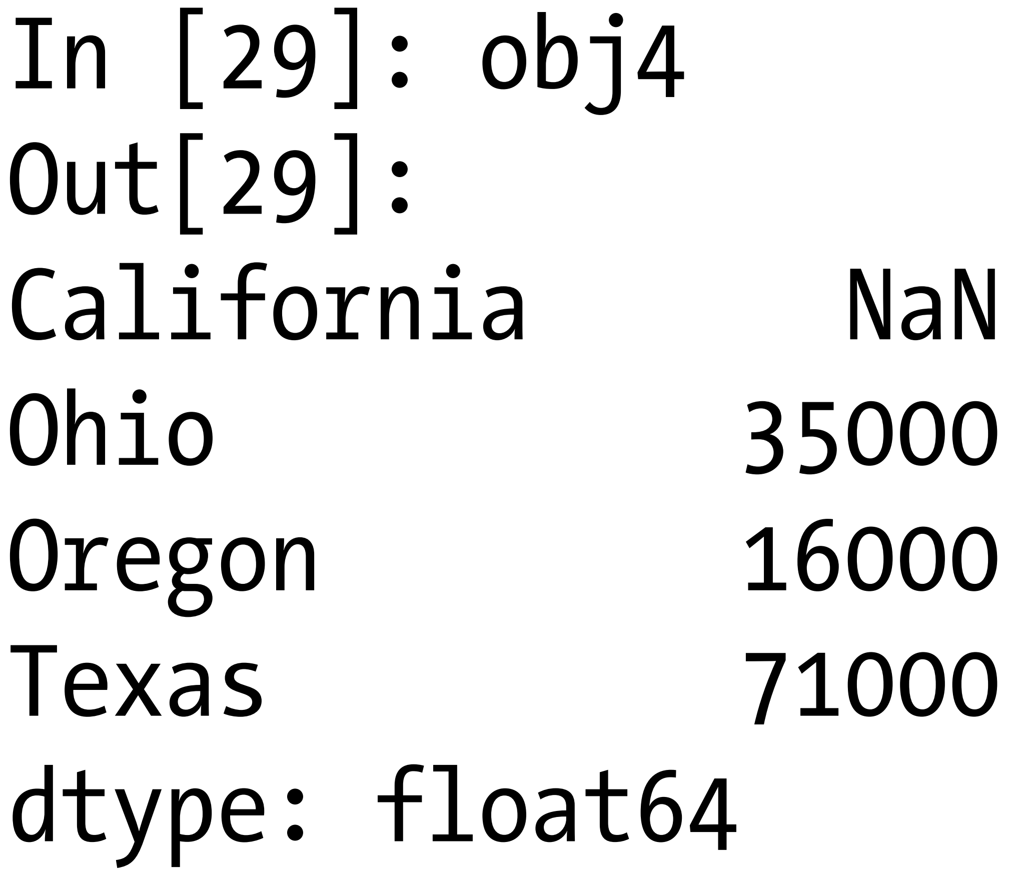
- `data < 5` → boolean data frame, can be used to select specific elements

Arithmetic

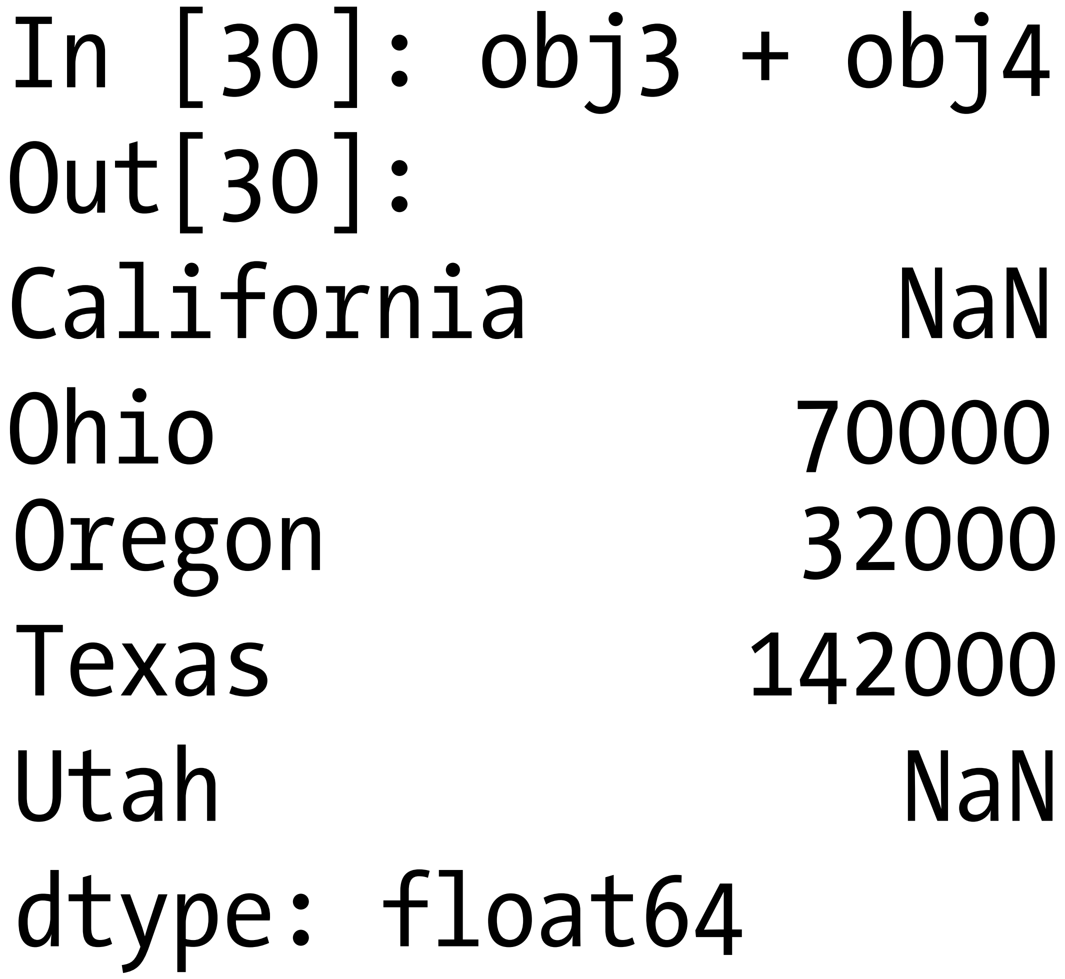
- Add, subtract, multiply, and divide are element-wise like numpy
- …but use labels to align
- …and missing labels lead to `NaN` (not a number) values



```
In [28]: obj3
Out[28]:
Ohio      35000
Oregon    16000
Texas     71000
Utah       5000
dtype: int64
```



```
In [29]: obj4
Out[29]:
California      NaN
Ohio          35000
Oregon        16000
Texas         71000
dtype: float64
```



```
In [30]: obj3 + obj4
Out[30]:
California       NaN
Ohio           70000
Oregon         32000
Texas         142000
Utah             NaN
dtype: float64
```

- also have `.add`, `.subtract`, … that allow `fill_value` argument
- `obj3.add(obj4, fill_value=0)`

Arithmetic between DataFrames and Series

- Broadcasting: e.g. apply single row operation across all rows
- Example:

```
In [148]: frame
Out[148]:
        b   d   e
Utah    0   1   2
Ohio    3   4   5
Texas   6   7   8
Oregon  9  10  11
```

```
In [149]: series
Out[149]:
b    0
d    1
e    2
Name: Utah, dtype: float64
```

```
In [150]: frame - series
Out[150]:
        b  d  e
Utah    0  0  0
Ohio    3  3  3
Texas   6  6  6
Oregon  9  9  9
```

- To broadcast over **columns**, use methods (`.add`, …)

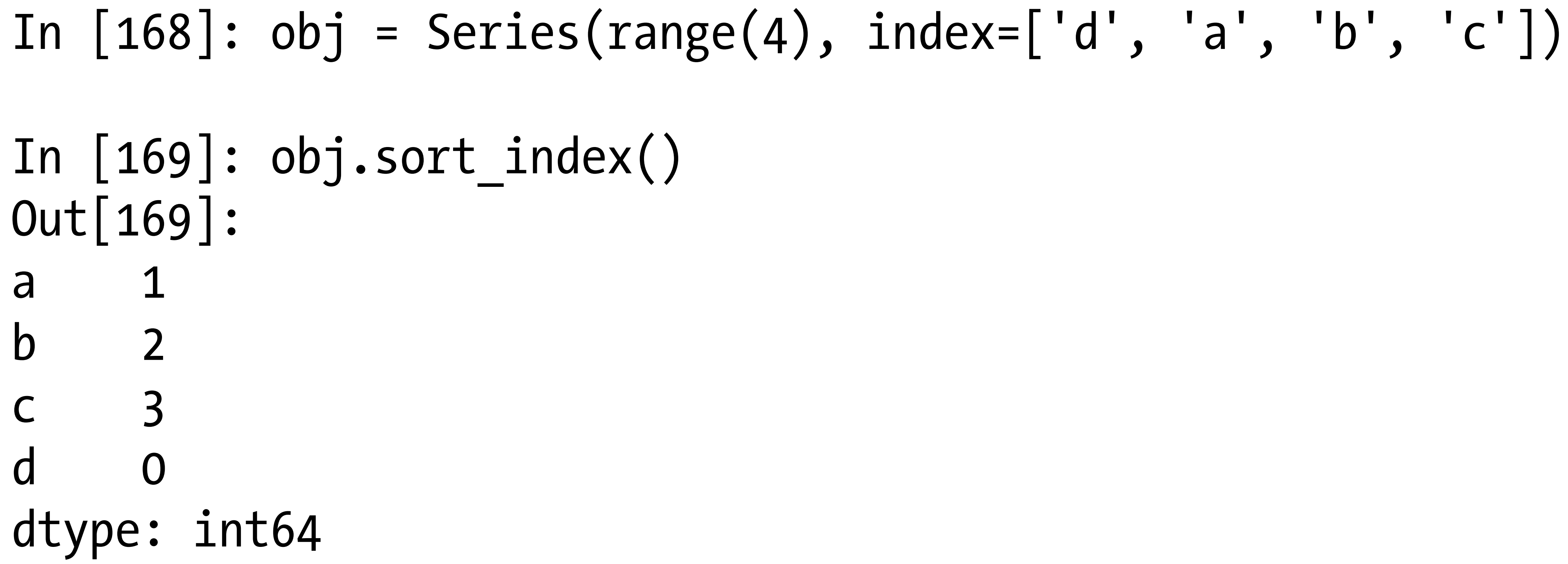
```
In [154]: frame
Out[154]:
        b   d   e
Utah    0   1   2
Ohio    3   4   5
Texas   6   7   8
Oregon  9  10  11
```

```
In [155]: series3
Out[155]:
Utah       1
Ohio       4
Texas      7
Oregon    10
Name: d, dtype: float64
```

```
In [156]: frame.sub(series3, axis=0)
Out[156]:
         b  d  e
Utah    -1  0  1
Ohio    -1  0  1
Texas   -1  0  1
Oregon  -1  0  1
```

Sorting by Index (sort_index)

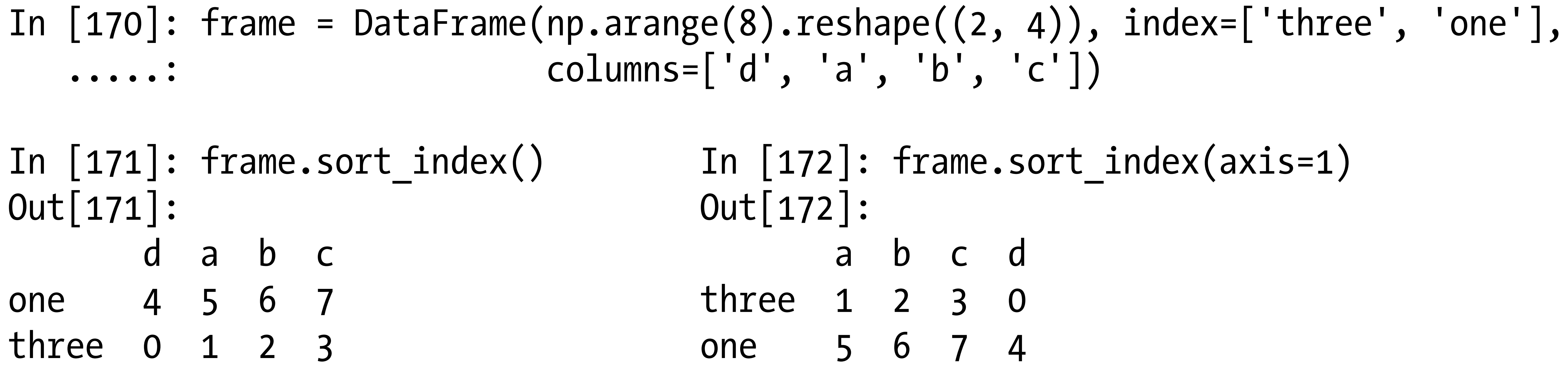
- Sort by index (lexicographical):



```
In [168]: obj = Series(range(4), index=['d', 'a', 'b', 'c'])

In [169]: obj.sort_index()
Out[169]:
a    1
b    2
c    3
d    0
dtype: int64
```

- DataFrame sorting:



```
In [170]: frame = DataFrame(np.arange(8).reshape((2, 4)), index=['three', 'one'],
   .....:                   columns=['d', 'a', 'b', 'c'])

In [171]: frame.sort_index()        In [172]: frame.sort_index(axis=1)
Out[171]:                           Out[172]:
       d  a  b  c                          a  b  c  d
one    4  5  6  7                   three  1  2  3  0
three  0  1  2  3                   one    5  6  7  4
```

- axis controls sort rows (0) vs. sort columns (1)

Sorting by Value (sort_values)

- `sort_values` method on series
  - `obj.sort_values()`
- Missing values (`NaN`) are at the end by default (`na_position` controls, can be first)
- `sort_values` on DataFrame:
  - `df.sort_values(<list-of-columns>)`
  - `df.sort_values(by=['a', 'b'])`
  - Can also use `axis=1` to sort by index labels

Reading

- Wednesday
- Discussing paper:
  - "Wrangler: Interactive Visual Specification of Data Transformation Scripts"
  - Kandel et al.
  - http://vis.stanford.edu/files/wrangler.pdf
- Read
- Come prepared with questions, thoughts
  - Compare with how things work in pandas