



Lecture 4: Getting Started with pandas

Alpar Sultan, PhD
Associate professor

```
In [1]: import pandas as pd
```

Thus, whenever you see `pd.` in code, it's referring to pandas. You may also find it easier to import `Series` and `DataFrame` into the local namespace since they are so frequently used:

```
In [2]: from pandas import Series, DataFrame
```

Introduction to pandas Data Structures Series.

- A Series is a one-dimensional array-like object containing a sequence of values (of similar types to NumPy types) and an associated array of data labels, called its index. The simplest Series is formed from only an array of data:

```
In [11]: obj = pd.Series([4, 7, -5, 3])
```

```
In [12]: obj
```

```
Out[12]:
```

```
0    4
```

```
1    7
```

```
2   -5
```

```
3    3
```

```
dtype: int64
```

- Using NumPy functions or NumPy-like operations, such as filtering with a boolean array, scalar multiplication, or applying math functions, will preserve the index-value link:

```
In [21]: obj2[obj2 > 0]
```

```
Out[21]:
```

```
d      6
```

```
b      7
```

```
c      3
```

```
dtype: int64
```

- Another way to think about a Series is as a fixed-length, ordered dict, as it is a map- ping of index values to data values. It can be used in many contexts where you might use a dict:

```
In [24]: 'b' in obj2  
Out[24]: True
```

```
In [25]: 'e' in obj2  
Out[25]: False
```

- Should you have data contained in a Python dict, you can create a Series from it by passing the dict:

```
In [26]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
```

```
In [27]: obj3 = pd.Series(sdata)
```

Data Frame

- A DataFrame represents a rectangular table of data and contains an ordered collection of columns, each of which can be a different value type (numeric, string, boolean, etc.).

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],  
        'year': [2000, 2001, 2002, 2001, 2002, 2003],  
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}  
frame = pd.DataFrame(data)
```

Data Frame

- When you are assigning lists or arrays to a column, the value's length must match the length of the DataFrame. If you assign a Series, its labels will be realigned exactly to the DataFrame's index, inserting missing values in any holes:

```
In [58]: val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
```

```
In [59]: frame2['debt'] = val
```

```
In [60]: frame2
```

```
Out[60]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7
six	2003	Nevada	3.2	NaN

Possible data inputs to Data Frame constructor

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

Index Objects

- pandas's Index objects are responsible for holding the axis labels and other metadata (like the axis name or names). Any array or other sequence of labels you use when constructing a Series or DataFrame is internally converted to an Index:

```
In [76]: obj = pd.Series(range(3), index=['a', 'b', 'c'])
```

```
In [77]: index = obj.index
```

```
In [78]: index
```

```
Out[78]: Index(['a', 'b', 'c'], dtype='object')
```

```
In [79]: index[1:]
```

```
Out[79]: Index(['b', 'c'], dtype='object')
```

Index methods and properties

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

Reindexing

- An important method on pandas objects is `reindex`, which means to create a new object with the data conformed to a new index.

```
In [91]: obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])
```

```
In [93]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
```

Reindex function arguments

Argument	Description
<code>index</code>	New sequence to use as index. Can be <code>Index</code> instance or any other sequence-like Python data structure. An <code>Index</code> will be used exactly as is without any copying.
<code>method</code>	Interpolation (fill) method; <code>'ffill'</code> fills forward, while <code>'bfill'</code> fills backward.
<code>fill_value</code>	Substitute value to use when introducing missing data by reindexing.
<code>limit</code>	When forward- or backfilling, maximum size gap (in number of elements) to fill.
<code>tolerance</code>	When forward- or backfilling, maximum size gap (in absolute numeric distance) to fill for inexact matches.
<code>level</code>	Match simple <code>Index</code> on level of <code>MultiIndex</code> ; otherwise select subset of.
<code>copy</code>	If <code>True</code> , always copy underlying data even if new index is equivalent to old index; if <code>False</code> , do not copy the data when the indexes are equivalent.

Dropping Entries from an Axis

- Dropping one or more entries from an axis is easy if you already have an index array or list without those entries. As that can require a bit of munging and set logic, the drop method will return a new object with the indicated value or values deleted from an axis:

```
In [105]: obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
```

```
In [107]: new_obj = obj.drop('c')
```

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
- Slicing with labels behaves differently than normal Python slicing in that the end- point is inclusive
- Setting using these methods modifies the corresponding section of the Series
- Indexing into a DataFrame is for retrieving one or more columns either with a single value or sequence

Selection with loc and iloc

- For DataFrame label-indexing on the rows, the special indexing operators loc and iloc are introduced. 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).

Indexing options with DataFrame

Type	Notes
<code>df[val]</code>	Select single column or sequence of columns from the DataFrame; special case conveniences: boolean array (filter rows), slice (slice rows), or boolean DataFrame (set values based on some criterion)
<code>df.loc[val]</code>	Selects single row or subset of rows from the DataFrame by label
<code>df.loc[:, val]</code>	Selects single column or subset of columns by label
<code>df.loc[val1, val2]</code>	Select both rows and columns by label
<code>df.iloc[where]</code>	Selects single row or subset of rows from the DataFrame by integer position
<code>df.iloc[:, where]</code>	Selects single column or subset of columns by integer position
<code>df.iloc[where_i, where_j]</code>	Select both rows and columns by integer position
<code>df.at[label_i, label_j]</code>	Select a single scalar value by row and column label
<code>df.iat[i, j]</code>	Select a single scalar value by row and column position (integers)
reindex method	Select either rows or columns by labels
get_value, set_value methods	Select single value by row and column label

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. This is similar to an automatic outer join on the index labels
- The internal data alignment introduces missing values in the label locations that don't overlap. Missing values will then propagate in further arithmetic computations.

Flexible arithmetic methods

Method	Description
<code>add, radd</code>	Methods for addition (+)
<code>sub, rsub</code>	Methods for subtraction (-)
<code>div, rdiv</code>	Methods for division (/)
<code>floordiv, rfloordiv</code>	Methods for floor division (//)
<code>mul, rmul</code>	Methods for multiplication (*)
<code>pow, rpow</code>	Methods for exponentiation (**)

Function Application and Mapping

- NumPy ufuncs (element-wise array methods) also work with pandas objects
- Another frequent operation is applying a function on one-dimensional arrays to each column or row. DataFrame's apply method does exactly this
- Many of the most common array statistics (like sum and mean) are DataFrame methods, so using apply is not necessary.

Sorting and Ranking

- Sorting a dataset by some criterion is another important built-in operation. To sort lexicographically by row or column index, use the `sort_index` method, which returns a new, sorted object:

```
In [201]: obj = pd.Series(range(4), index=['d', 'a', 'b', 'c'])
```

```
In [202]: obj.sort_index()
```

```
Out[202]:
```

```
a    1
```

```
b    2
```

```
c    3
```

```
d    0
```

```
dtype: int64
```

- Ranking assigns ranks from one through the number of valid data points in an array. The rank methods for Series and DataFrame are the place to look; by default rank breaks ties by assigning each group the mean rank:

```
In [215]: obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
```

```
In [216]: obj.rank()
```

```
Out[216]:
```

```
0    6.5
```

```
1    1.0
```

```
2    6.5
```

```
3    4.5
```

```
4    3.0
```

```
5    2.0
```

```
6    4.5
```

```
dtype: float64
```

Tie-breaking methods with rank

Method	Description
'average'	Default: assign the average rank to each entry in the equal group
'min'	Use the minimum rank for the whole group
'max'	Use the maximum rank for the whole group
'first'	Assign ranks in the order the values appear in the data
'dense'	Like <code>method='min'</code> , but ranks always increase by 1 in between groups rather than the number of equal elements in a group

Axis Indexes with Duplicate Labels

```
In [222]: obj = pd.Series(range(5), index=['a', 'a', 'b', 'b', 'c'])
```

```
In [223]: obj
```

```
Out[223]:
```

```
a    0
```

```
a    1
```

```
b    2
```

```
b    3
```

```
c    4
```

```
dtype: int64
```

```
In [224]: obj.index.is_unique
```

```
Out[224]: False
```

```
In [225]: obj['a']
```

```
Out[225]:
```

```
a    0
```

```
a    1
```

```
dtype: int64
```

```
In [226]: obj['c']
```

```
Out[226]: 4
```

Summarizing and Computing Descriptive Statistics

```
In [232]: df.sum()
```

```
Out[232]:
```

```
one      9.25
```

```
two     -5.80
```

```
dtype: float64
```

```
In [233]: df.sum(axis='columns')
```

```
Out[233]:
```

```
a      1.40
```

```
b      2.60
```

```
c      NaN
```

```
d     -0.55
```

```
dtype: float64
```

```
In [234]: df.mean(axis='columns', skipna=False)
```

```
Out[234]:
```

```
a      NaN
```

```
b      1.300
```

```
c      NaN
```

Method	Description
--------	-------------

axis	Axis to reduce over; 0 for DataFrame's rows and 1 for columns
------	---

skipna	Exclude missing values; True by default
--------	---

level	Reduce grouped by level if the axis is hierarchically indexed (MultiIndex)
-------	--

Descriptive and summary statistics

Method	Description
count	Number of non-NA values
describe	Compute set of summary statistics for Series or each DataFrame column
min, max	Compute minimum and maximum values
argmin, argmax	Compute index locations (integers) at which minimum or maximum value obtained, respectively
idxmin, idxmax	Compute index labels at which minimum or maximum value obtained, respectively
quantile	Compute sample quantile ranging from 0 to 1
sum	Sum of values
mean	Mean of values
median	Arithmetic median (50% quantile) of values
mad	Mean absolute deviation from mean value
prod	Product of all values
var	Sample variance of values
std	Sample standard deviation of values
skew	Sample skewness (third moment) of values
kurt	Sample kurtosis (fourth moment) of values
cumsum	Cumulative sum of values
cummin, cummax	Cumulative minimum or maximum of values, respectively
cumprod	Cumulative product of values
diff	Compute first arithmetic difference (useful for time series)
pct_change	Compute percent changes

Unique Values, Value Counts and Membership

```
In [252]: uniques = obj.unique()
```

```
In [253]: uniques
```

```
Out[253]: array(['c', 'a', 'd', 'b'], dtype=object)
```

```
In [254]: obj.value_counts()
```

```
Out[254]:
```

```
c      3
```

```
a      3
```

```
b      2
```

```
d      1
```

```
dtype: int64
```

Method	Description
<code>isin</code>	Compute boolean array indicating whether each Series value is contained in the passed sequence of values
<code>match</code>	Compute integer indices for each value in an array into another array of distinct values; helpful for data alignment and join-type operations
<code>unique</code>	Compute array of unique values in a Series, returned in the order observed
<code>value_counts</code>	Return a Series containing unique values as its index and frequencies as its values, ordered count in descending order