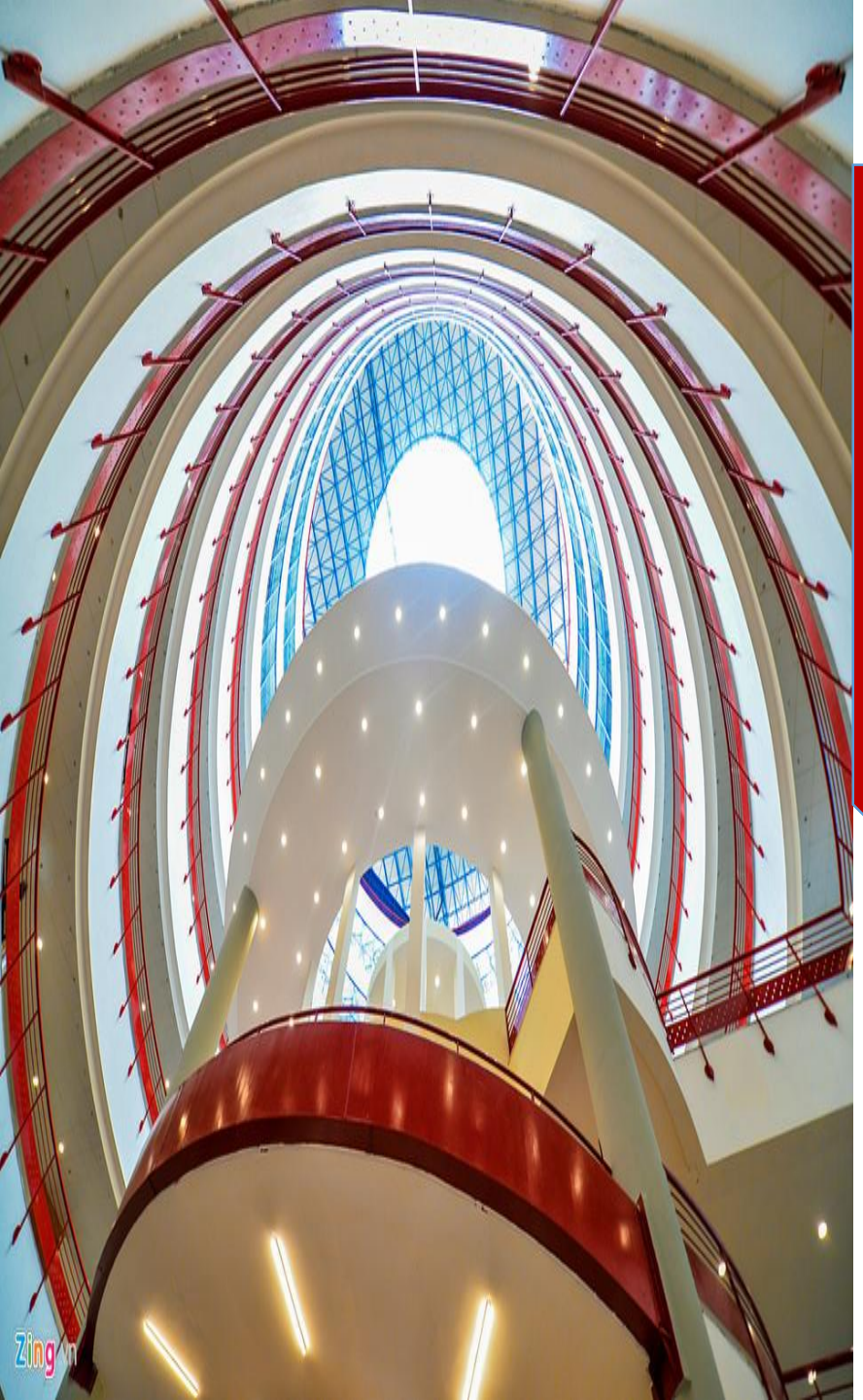
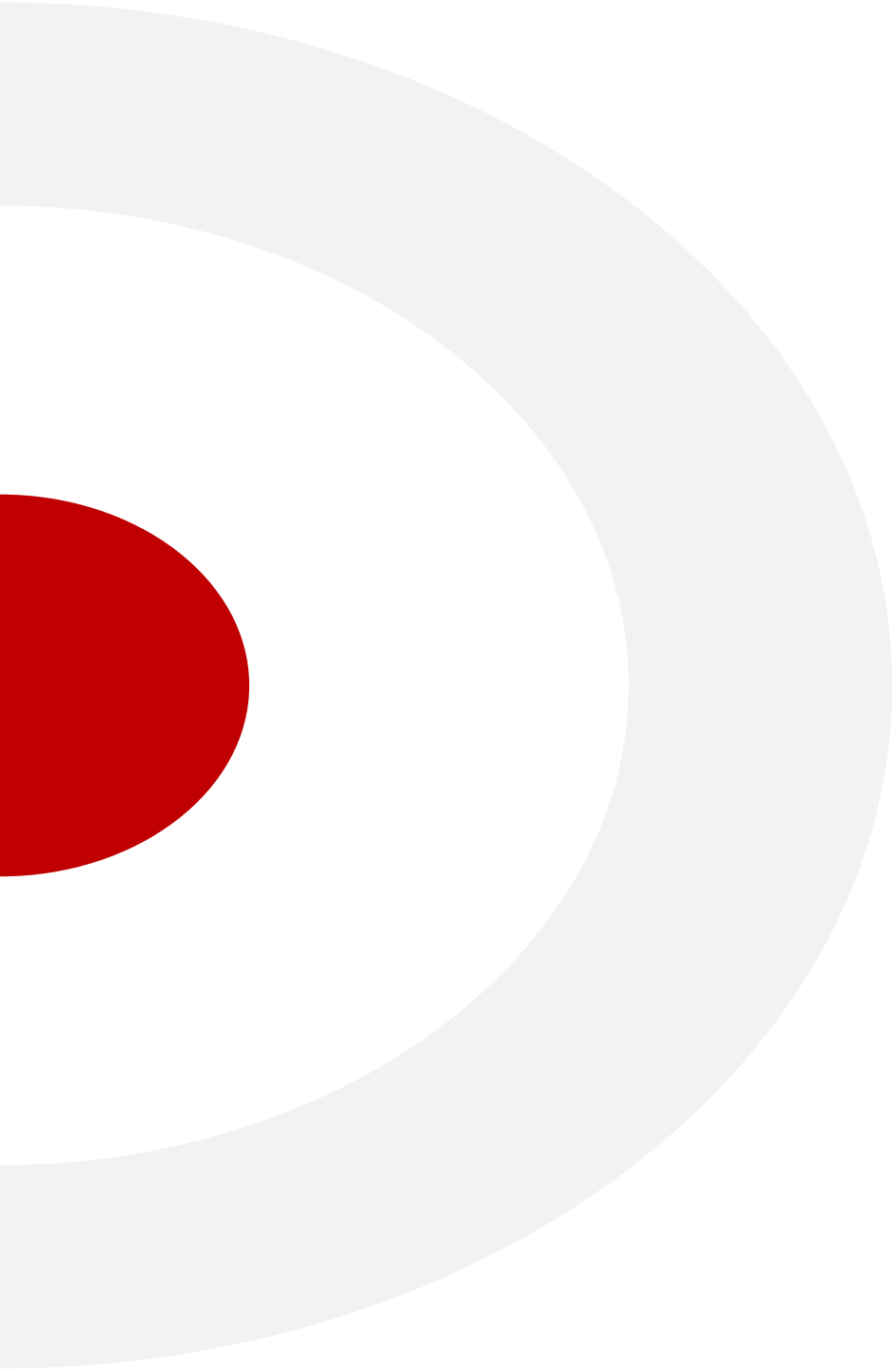


DATA PREPARATION AND VISUALIZATION

Mathematical Economics Faculty

National Economics University
https://www.neu.edu.vn/



Chapter 2: Getting Started with Pandas

Objectives

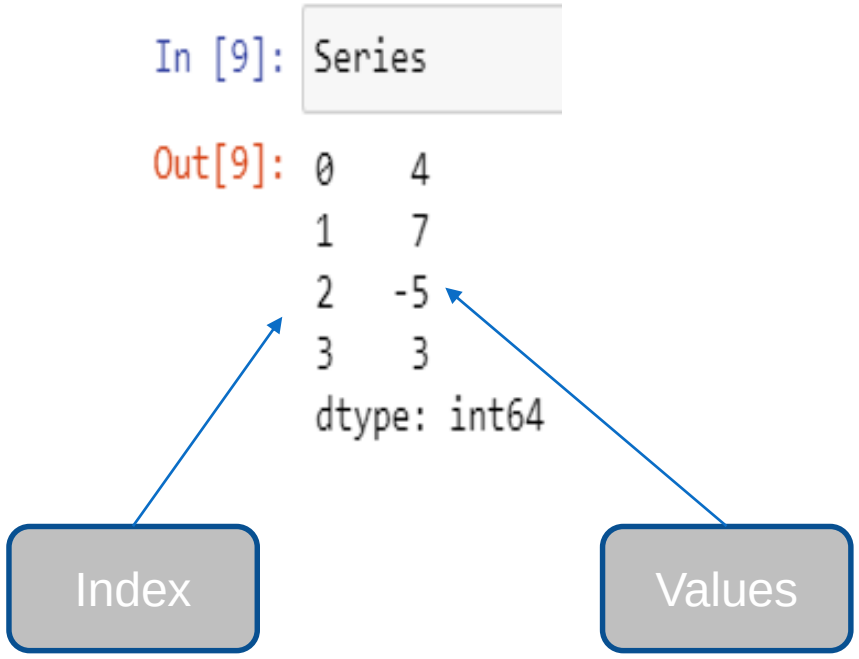
- Introduction to pandas Data Structures
  - Series
  - DataFrame
  - Index Objects
- Essential Functionality
  - Reindexing
  - Dropping Entries from an Axis
  - Indexing, Selection, and Filtering
  - Integer Indexes
  - Arithmetic and Data Alignment
  - Function Application and Mapping
  - Sorting and Ranking

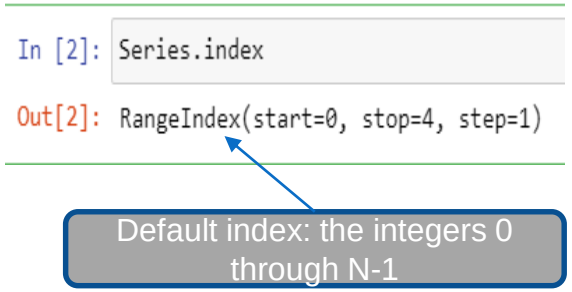
Objectives

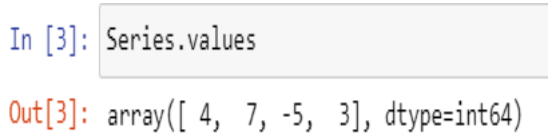
- Summarizing and Computing Descriptive Statistic
  - Correlation and Covariance
  - Unique Values, Value Counts, and Membership
  - Index Objects

Introduction to pandas Data Structures

- **Series:** 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
- Example: *Series = pd.Series([4, 7, -5, 3])*







Introduction to pandas Data Structures

- Create a Series with an index identifying each data point with a label
  - *Series = pd.Series([4, 7, -5, 3], index = ['a','b','c','d'])*

- How can we select single value or a set of values using labels in the index?
  - We can use labels in the index
- Using Numpy functions or Numpy-like operations such as filtering with Boolean array, scalar multiplication, or applying math functions, will preserve the index-value link
- Another way to think about a Series: a fixed-length, ordered dict, as it is a mapping of index values to data values
- You can create a Series from a dictionary
  - By default: the index will have the dict's keys conserving the order of keys in dictionary
  - You can override this by passing the dict keys in the order you want them to appear in the resulting Series

DataFrame

- 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
- The DataFrame has both a row and column index
- The most common way to construct a DataFrame is from a dict of equal-length lists or Numpy array
  - The resulting DataFrame will have its index assigned automatically as with Series, and the columns are placed in the original order
  - You can pass the index and specify a sequence of columns, the DataFrame's columns will be arranged in that order
  - If you pass a column that isn't contained in the dict, it will appear with missing values in the results

DataFrame

Access DataFrames's elements

- A column in a DataFrame can be retrieved as a Series either by:
  - Dict-like notation: DataFrame[column]
  - Attribute-like access: DataFrame.column
- Rows can be also be retrived by position or name with the special loc attribute
  - DataFrame.loc[column]
- Columns can be modified by assignment. Ex:
  - Df['debt'] = 16.5 -> create a new column named debt, this column is assigned a scalar value
  - Df['debt'] = np.array(6.) -> the 'debt' column is assigned an array of values
- If you assign a Series, its labels will be realigned exactly to the DataFrame's index, inserting missing values in any holes. Ex:
  - Create a Series: value = [-1.2,-1.5,-1.7,0,0]
  - Assign this Series to the 'debt' column
  - What happen?

DataFrame

Possible data inputs to DataFrame 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
Dict of Series	Each value becomes a column; indexes from each Series are unioned together to 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
Another DataFrame	The DataFrame's indexes are used unless different ones are passed

Essential Functionality

Reindexing

- Reindex: create a new object with the data conformed to a new index

```
DataFrame.reindex(labels=None, index=None, columns=None, axis=None, method=None,
copy=True, level=None, fill_value=nan, limit=None, tolerance=None)        [source]
```

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

Essential Functionality

Dropping Entries from an Axis

Syntax

DataFrame.drop(*labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise'*)

Parameters:

labels : ***single label or list-like***

Index or column labels to drop. A tuple will be used as a single label and not treated as a list-like.

axis : ***{0 or 'index', 1 or 'columns'}, default 0***

Whether to drop labels from the index (0 or 'index') or columns (1 or 'columns').

index : ***single label or list-like***

Alternative to specifying axis (`labels, axis=0` is equivalent to `index=labels`).

columns : ***single label or list-like***

Alternative to specifying axis (`labels, axis=1` is equivalent to `columns=labels`).

level : ***int or level name, optional***

For MultiIndex, level from which the labels will be removed.

inplace : ***bool, default False***

If False, return a copy. Otherwise, do operation inplace and return None.

errors : ***{'ignore', 'raise'}, default 'raise'***

If 'ignore', suppress error and only existing labels are dropped.

Returns:

DataFrame or None

DataFrame without the removed index or column labels or None if `inplace=True`.

Essential Functionality

Dropping Entries from an Axis

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

Data.drop(??)

	one	three	four
Ohio	0	2	3
Colorado	4	6	7
Utah	8	10	11
New York	12	14	15

Data.drop(??)

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

Essential Functionality

Selection with loc and iloc

Type	Notes
`df[val]`	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)
`df.loc[val]`	Selects single row or subset of rows from the DataFrame by label
`df.loc[:, val]`	Selects single column or subset of columns by label
`df.loc[val1, val2]`	Select both rows and columns by label
`df.iloc[where]`	Selects single row or subset of rows from the DataFrame by integer position

Type	Notes
`df.iloc[:, where]`	Selects single column or subset of columns by integer position
`df.iloc[where_i, where_j]`	Select both rows and columns by integer position
`df.at[label_i, label_j]`	Select a single scalar value by row and column label
`df.iat[i, j]`	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

Essential Functionality

Selection with loc and iloc

Create a dataframe look like the following table

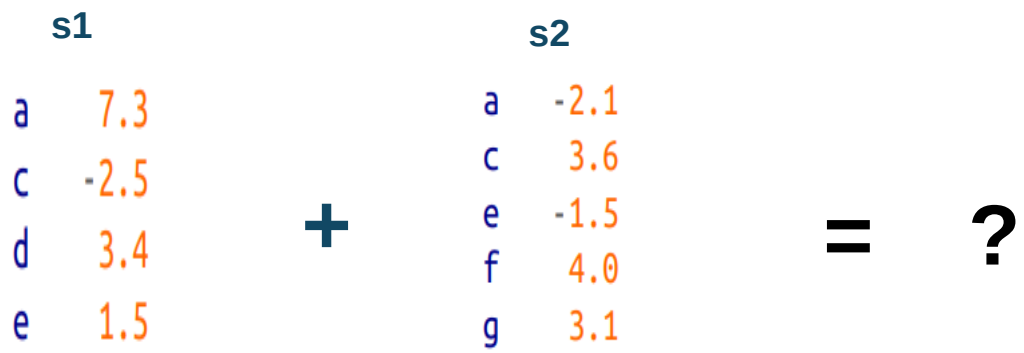
	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

- Select the column ‘one’ using loc
- Select the first row and the columns ‘one’ and ‘two’ by label
- Perform similar selections with integers using iloc
- Select the first 3 rows and the columns ‘one and ‘two’ using loc and iloc

Essential Functionality

Arithmetic and Data Alignment

- An important pandas feature for some applications is the behavior of arithmetic between objects with different indexes. For ex:



- 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:
  - df1 = pd.DataFrame({'A' : [1,2]})
  - df2 = pd.DataFrame({'B' : [3,4]})
  - Try df1 – df2

Essential Functionality

Arithmetic and Data Alignment

Flexible arithmetic methods

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

Using the arithmetic methods like add, sub, div…you can fill with a special value, like 0, when an axis label is found in one object but not the other

Essential Functionality

Arithmetic and Data Alignment

Operations between DataFrame and Series

- By default, arithmetic between DataFrame and Series matches the index of Series on the DataFrame's columns, broading down the rows
- If an index value is not found in either the DataFrame's columns or the Series's index, the objects will be reindexed to form the union
- If you want to instead broadcast over the columns, matching on the rows, you have to use one of the arithmetic methods

Essential Functionality

Arithmetic and Data Alignment

Operations between DataFrame and Series

Create the following dataframe

```
df = pd.DataFrame(np.arange(12).reshape(4,3),\
                  columns = list('bcd'),\
                  index = ['i1','i2','i3','i4'])
series1 = df.iloc[0]
series2 = df['b']
```

Try df – series1 and df – series2

Essential Functionality

Function Application and Mapping

- Numpy ufuncs (element-wise methods) also work with pandas objects:
  - *Ex: np.abs(dataframe)*
- Another frequent operation is applying a function on one-dimensional arrays to each column or row
  - *df.apply(func,axis = 0)*
- The function passed to apply need not return a scalar value; it can also return a Series with multiple values
- We can use *applymap* for applying a function to a Dataframe elementwise.

Essential Functionality

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
- To sort a Series by its value, use its sort_values method

Sort_index method

Parameters	Description
Axis: {0 or 'index', 1 or 'columns', Default 0	The axis along which to sort
Ascending: bool or list-like of bools, default True	Sort ascending vs descending
Na_position {'first', 'last'}, default 'last'	Puts NaNs at the beginning if first, last puts NaNs at the end.

Essential Functionality

Sorting and Ranking

Sort_values method

Parameters	Description
Axis: {0 or ‘index’, 1 or ‘columns’, Default 0	The axis along which to sort
Ascending: bool or list-like of bools, default True	Sort ascending vs descending
Na_position {‘first’, ‘last’}, default ‘last’	Puts NaNs at the beginning if first, last puts NaNs at the end.
By: str or list of str	Name or list of names to sort by

Essential Functionality

Sorting and Ranking

- Ranking assigns ranks from one through the number of valid data points in an array
- Use pd.Series.rank() method

Parameters	Description
Axis: {0 or 'index', 1 or 'columns', Default 0	Index to direct ranking
Method: {'average', 'min', 'max', 'first','dense'}, default 'average'	How to rank the group of records that have the same value: • Average: average rank of the group • Min: lowest rank in the group • Max: highest rank in the group • First: ranks assigned in order they appear in the array • Dense: like 'min', but rank always increases by 1 between groups
Na_position {'first', 'last'}, default 'last'	
Na_option: {'keep', 'top', 'bottom'}, default 'keep'	How to rank NaN values • Keep: assign NaN rank to NaN values • Top: assign lowest rank to NaN values • Bottom: assign highest rank to NaN values

Essential Functionality

Sorting and Ranking

- Example

```
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
```

Summarizing and Computing Descriptive Statistics

- Pandas objects are equipped with a set of common mathematical and statistical methods
- Most of these fall into the category of reductions or summary statistics, method that extract single value (like the sum or mean) from a Series or Series of values from the rows or columns of a DataFrame

Table 5-7. Options for reduction methods

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)

Summarizing and Computing Descriptive Statistics

Table 5-8. 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

Summarizing and Computing Descriptive Statistics

Unique Values, Value Counts, and Membership

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