# Python for Data Analysis, 3rd edition

Data Wrangling with pandas, NumPy, and Jupyter

Stephen CUI[1]

January 4, 2022

[1] cuixuanStephen@gmail.com

# Contents

# Chapter 1

# Getting Started with pandas

pandas will be a major tool of interest throughout much of the rest of the book. It contains data structures and data manipulation tools designed to make data cleaning and analysis fast and convenient in Python. pandas is often used in tandem with numerical computing tools like NumPy and SciPy, analytical libraries like statsmodels and scikit-learn, and data visualization libraries like matplotlib. pandas adopts significant parts of NumPy's idiomatic style of array-based computing, especially array-based functions and a preference for data processing without for loops.

While pandas adopts many coding idioms from NumPy, the biggestabout difference is that pandas is designed for working with tabular or heterogeneous data. NumPy, by contrast, is best suited for working with homogeneously typed numerical array data.

## 1.1 Introduction to pandas Data Structures

To get started with pandas, you will need to get comfortable with its two workhorse data structures: *Series* and *DataFrame*.

### 1.1.1 Series

A Series is a one-dimensional array-like object containing a sequence of values (of similar types to NumPy types) of the same type and an associated array of data labels, called its *index*.

1. Since we did not specify an index for the data, a default one consisting of the integers 0 through $N - 1$ (where $N$ is the length of the data) is created. You can get the array representation and index object of the Series via its array and index attributes, respectively.
2. Often, you'll want to create a Series with an index identifying each data point with a label. Compared with NumPy arrays, you can use labels in the index when selecting single values or a set of values.
3. 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.

4. Another way to think about a Series is as a fixed-length, ordered dictionary, as it is a mapping of index values to data values. Should you have data contained in a Python dictionary, you can create a Series from it by passing the dictionary.
5. A Series can be converted back to a dictionary with its `to_dict` method. You can override this by passing an index with the dictionary keys in the order you want them to appear in the resulting Series(索引里没找到的将称为缺失值NaN，没有包含的原索引将被删除).
6. The `isna` and `notna` functions in pandas should be used to detect missing data(`missing`、`NA`、`null`). Series also has these as instance methods. I discuss working with missing data in more detail in Chapter 3.

A useful Series feature for many applications is that it automatically aligns by index label in arithmetic operations.(与数据中的连接相似)

1. Both the Series object itself and its index have a name attribute, which integrates with other areas of pandas functionality.
2. A Series's index can be altered in place by assignment.

### 1.1.2 DataFrame

A DataFrame represents a rectangular table of data and contains an ordered, named 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; it can be thought of as a dictionary of Series all sharing the same index.

1. There are many ways to construct a DataFrame, though one of the most common is from a dictionary of equal-length lists or NumPy arrays.
2. The head method selects only the first five rows. Similarly, tail returns the last five rows.
3. If you 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 dictionary, it will appear with missing values in the result.
4. A column in a DataFrame can be retrieved as a Series either by dictionary-like notation or by using the dot attribute notation.

> Notes
>
> frame2[column] works for any column name, but frame2.column works only when the column name is a valid Python variable name and does not conflict with any of the method names in DataFrame. For example, if a column's name contains whitespace or symbols other than underscores, it cannot be accessed with the dot attribute method.

Rows can also be retrieved by position or name with the special iloc and loc attributes.

1. 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 index values not present.

2. Assigning a column that doesn't exist will create a new column.
3. The del keyword will delete columns like with a dictionary.

Warnings

New columns cannot be created with the frame2.eastern dot attribute notation.

Warnings

The column returned from indexing a DataFrame is a view on the underlying data, not a copy. Thus, any in-place modifications to the Series will be reflected in the DataFrame. The column can be explicitly copied with the Series's copy method.

- Another common form of data is a nested dictionary of dictionaries. If the nested dictionary is passed to the DataFrame, pandas will interpret the outer dictionary keys as the columns, and the inner keys as the row indices.
- You can transpose the DataFrame (swap rows and columns) with similar syntax to a NumPy array.
- The keys in the inner dictionaries are combined to form the index in the result. This isn't true if an explicit index is specified.
- Dictionaries of Series are treated in much the same way.

Warnings

Note that transposing discards the column data types if the columns do not all have the same data type, so transposing and then transposing back may lose the previous type information. The col umns become arrays of pure Python objects in this case.

For a list of many of the things you can pass to the DataFrame constructor, see Table 1.1.

1. If a DataFrame's index and columns have their name attributes set, these will also be displayed.
2. Unlike Series, DataFrame does not have a name attribute. DataFrame's `to_numpy` method returns the data contained in the DataFrame as a two-dimensional ndarray.
3. If the DataFrame's columns are different data types, the data type of the returned array will be chosen to accommodate all of the columns

### 1.1.3 Index Objects

pandas's Index objects are responsible for holding the axis labels (including a DataFrame's column names) and other metadata (like the axis name or names).

1. Any array or other sequence of labels you use when constructing a Series or DataFrame is internally converted to an Index
2. Index objects are immutable and thus can't be modified by the user.

Table 1.1: Possible data inputs to the DataFrame constructor

| Type | Description |
|---|---|
| 2D ndarray | A matrix of data, passing optional row and column labels Dictionary 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 “dictionary of arrays” case |
| Dictionary of Series | Each value becomes a column; indexes from each Series are unioned together to form theresult’s row index if no explicit index is passed |
| Dictionary of dictionaries | Each inner dictionary becomes a column; keys are unioned to form the row index as in the “dictionary of Series” case |
| List of dictionaries or Series | Each item becomes a row in the DataFrame; unions of dictionary 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 are missing in the DataFrame result |

3. In addition to being array-like, an Index also behaves like a fixed-size set.
4. Unlike Python sets, a pandas Index can contain duplicate labels.

Some useful ones are summarized in Table 1.2.

## 1.2 Essential Functionality

### 1.2.1 Reindexing

An important method on pandas objects is reindex, which means to create a new object with the values rearranged to align with the new index.

1. Calling reindex on this Series rearranges the data according to the new index introducing missing values if any index values were not already present.
2. For ordered data like time series, you may want to do some interpolation or filling of values when reindexing. The method option allows us to do this, using a method such as ffill, which forward-fills the values.
3. With DataFrame, reindex can alter the (row) index, columns, or both. When passed only a sequence, it reindexes the rows in the result. The columns can be reindexed with the columns keyword.
4. Another way to reindex a particular axis is to pass the new axis labels as a positional argument and then specify the axis to reindex with the axis keyword

Table 1.2: Some Index methods and properties

| Method/Property | Description |
|---|---|
| `append()` | Concatenate with additional Index objects, producing a new Index |
| `difference()` | 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`(单调的) | 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 |

See Table 1.3 for more about the arguments to `reindex`.

As we'll explore later in **??**, you can also reindex by using the loc operator, and many users prefer to always do it this way. This works only if all of the new index labels already exist in the DataFrame (whereas reindex will insert missing data for new labels)

### 1.2.2 Dropping Entries from an Axis

Dropping one or more entries from an axis is simple if you already have an index array or list without those entries, since you can use the reindex method or .loc-based indexing. 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.

1. With DataFrame, index values can be deleted from either axis.
2. Calling drop with a sequence of labels will drop values from the row labels (axis 0).
3. To drop labels from the columns, instead use the columns keyword
4. You can also drop values from the columns by passing axis=1 (which is like NumPy) or axis="columns"

### 1.2.3 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.

While you can select data by label this way, the preferred way to select index values is with the special `loc` operator.

The reason to prefer loc is because of the different treatment of integers when indexing with []. Regular []-based indexing will treat integers as labels if the index contains integers, so the behavior

Table 1.3: reindex function arguments

| Argument | Description |
|---|---|
| `labels` | New sequence to use as an index. Can be Index instance or any other sequence-like Python data structure. An Index will be used exactly as is without any copying. |
| `index` | Use the passed sequence as the new index labels. |
| `columns` | Use the passed sequence as the new column labels. |
| `axis` | The axis to reindex, whether "index" (rows) or "columns". The default is "index". You can alternately do reindex(`index=new_labels`) or reindex(`columns=new_labels`). |
| `method` | Interpolation (fill) method; "ffill" fills forward, while "bfill" fills backward. |
| `fill_value` | Substitute value to use when introducing missing data by reindexing. Use `fill_value="missing"` (the default behavior) when you want absent labels to have null values in the result. |
| `limit` | When forward filling or backfilling, the maximum size gap (in number of elements) to fill. |
| `tolerance` | When forward filling or backfilling, the 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 the new index is equivalent to the old index; if False, do not copy the data when the indexes are equivalent. |

differs depending on the data type of the index.

When using loc, the expression `obj.loc[[0, 1, 2]]` will fail when the index does not contain integers.

Since loc operator indexes exclusively with labels, there is also an iloc operator that indexes exclusively with integers to work consistently whether or not the index contains integers.

> Warnings
>
> You can also slice with labels, but it works differently from normal Python slicing in that the endpoint is inclusive

索引数据框可以获得其一列（单一值）或者多列（序列值）：

像上面这样的索引有几个特殊的小案例：

- 用由Boolean组成的数组来进行切片（slice）或者选择（select）数据：
- 用由Boolean组成的数据框来进行索引（index）数据：

### Selection on DataFrame with loc and iloc

Like Series, DataFrame has special attributes loc and iloc for label-based and integer-based indexing, respectively.

仅选择数据框中的某一行返回的是Series（带有索引，即原数据的columns）。如果想要选择多行，并创建一个数据框，需要指定一个序列。

You can combine both row and column selection in loc by separating the selections with a comma：

这些对`iloc`同样适用，只要你提供整数来作为筛选器：

除了单一的标签或者有标签组成的列表，切片对`loc`和`iloc`也同样奏效:

Boolean组成的数组可以和`loc`一起使用但是不能和`iloc`配合使用：

There are many ways to select and rearrange the data contained in a pandas object. For DataFrame, Table 1.4 provides a short summary of many of them. As you will see later, there are a number of additional options for working with hierarchical indexes.

### Integer indexing pitfalls

Working with pandas objects indexed by integers can be a stumbling block for new users since they work differently from built-in Python data structures like lists and tuples.

### Pitfalls with chained indexing

A common gotcha for new pandas users is to chain selections when assigning. like this:

```
data.loc[data.three == 5]["three"] = 6
# <ipython-input-11-0ed1cf2155d5>:1: SettingWithCopyWarning:
```

Table 1.4: Indexing options with DataFrame

| Argument | Notes |
|---|---|
| `df[column]` | 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[rows]` | Select single row or subset of rows from the DataFrame by label |
| `df.loc[:, cols]` | Select single column or subset of columns by label |
| `df.loc[rows, cols]` | Select both row(s) and column(s) by label |
| `df.iloc[rows]` | Select single row or subset of rows from the DataFrame by integer position |
| `df.iloc[:, cols]` | Select single column or subset of columns by integer position |
| `df.iloc[rows, cols]` | Select both row(s) and column(s) by integer position |
| `df.at[row, col]` | Select a single scalar value by row and column label |
| `df.iat[row, col]` | Select a single scalar value by row and column position (integers) |
| reindex method | Select either rows or columns by labels |

```
    # A value is trying to be set on a copy of a slice from a DataFrame.
    # Try using .loc[row_indexer,col_indexer] = value instead
```

Depending on the data contents, this may print a special SettingWithCopyWarning, which warns you that you are trying to modify a temporary value (the nonempty result of `data.loc[data.three == 5]`) instead of the original DataFrame data, which might be what you were intending. Here, data was unmodified.

**A good rule of thumb is to avoid chained indexing when doing assignments.**

### 1.2.4 Arithmetic and Data Alignment

# Chapter 2

# Data Loading, Storage, and File Formats

# Chapter 3

# Data Cleaning and Preparation

Chapter 4

# Appendix A