1 Getting Started with pandas

1.1 Introduction to pandas Data Structures

Series DataFrame Index Objects

1.2 Essential Functionality

Reindexing Dropping entries from an axis Indexing, selection, and filtering Arithmetic and data alignment Function application and mapping Sorting and ranking Axis indexes with duplicate values

1.3 Summarizing and Computing Descriptive Statistics

Correlation and Covariance Unique Values, Value Counts, and Membership

1.4 Handling Missing Data

Filtering Out Missing Data Filling in Missing Data

1.5 Hierarchical Indexing

Reordering and Sorting Levels Summary Statistics by Level Using a DataFrame's Columns

1.6 Other pandas Topics

Integer Indexing Panel Data

1.7 Manipulating indices -Reindexing

Reindexing allows users to manipulate the data labels in a DataFrame. It forces a DataFrame to conform to the new index, and optionally, fill in missing data if requested.

A simple use of reindex is to alter the order of the rows:

1.8 Missing data basics

Missing data The occurence of missing data is so prevalent that it pays to use tools like Pandas, which seamlessly integrates missing data handling so that it can be dealt with easily, and in the manner required by the analysis at hand.

Missing data are represented in Series and DataFrame objects by the NaN floating point value. However, None is also treated as missing, since it is commonly used as such in other contexts (e.g. NumPy).

When / why does data become missing?

Some might quibble over our usage of missing. By "missing" we simply mean null or "not present for whatever reason". Many data sets simply arrive with missing data, either because it exists and was not collected or it never existed. For example, in a collection of financial time series, some of the time series might start on different dates. Thus, values prior to the start date would generally be marked as missing.

In pandas, one of the most common ways that missing data is introduced into a data set is by reindexing. For example

```
In [1]: df = DataFrame(randn(5, 3), index=['a', 'c', 'e', 'f', 'h'],
   ...:                columns=['one', 'two', 'three'])
   ...:

In [2]: df['four'] = 'bar'

In [3]: df['five'] = df['one'] > 0

In [4]: df

        one       two     three four   five
a  0.059117  1.138469 -2.400634  bar   True
c -0.280853  0.025653 -1.386071  bar  False
e  0.863937  0.252462  1.500571  bar   True
f  1.053202 -2.338595 -0.374279  bar   True
h -2.359958 -1.157886 -0.551865  bar  False

In [5]: df2 = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

In [6]: df2

        one       two     three four   five
a  0.059117  1.138469 -2.400634  bar   True
b       NaN       NaN       NaN  NaN    NaN
c -0.280853  0.025653 -1.386071  bar  False
d       NaN       NaN       NaN  NaN    NaN
e  0.863937  0.252462  1.500571  bar   True
```

```
f  1.053202 -2.338595 -0.374279  bar   True
g       NaN       NaN       NaN  NaN    NaN
h -2.359958 -1.157886 -0.551865  bar  False
```

Values considered “missing”

As data comes in many shapes and forms, pandas aims to be flexible with regard to handling missing data. While NaN is the default missing value marker for reasons of computational speed and convenience, we need to be able to easily detect this value with data of different types: floating point, integer, boolean, and general object. In many cases, however, the Python None will arise and we wish to also consider that “missing” or “null”.

Until recently, for legacy reasons inf and -inf were also considered to be “null” in computations. This is no longer the case by default; use the mode.use_inf_as_null option to recover it.

To make detecting missing values easier (and across different array dtypes), pandas provides the isnull() and notnull() functions, which are also methods on Series objects:

```
In [7]: df2['one']

a    0.059117
b         NaN
c   -0.280853
d         NaN
e    0.863937
f    1.053202
g         NaN
h   -2.359958
Name: one, dtype: float64

In [8]: isnull(df2['one'])

a    False
b     True
c    False
d     True
e    False
f    False
g     True
h    False
Name: one, dtype: bool

In [9]: df2['four'].notnull()

a     True
b    False
c     True
d    False
e     True
```

```
f     True
g    False
h     True
dtype: bool
```

Summary: NaN and None (in object arrays) are considered missing by the isnull and notnull functions. inf and -inf are no longer considered missing by default.

1.9 Data summarization

We often wish to summarize data in Series or DataFrame objects, so that they can more easily be understood or compared with similar data. The NumPy package contains several functions that are useful here, but several summarization or reduction methods are built into Pandas data structures.