5 - Pandas-Intro

March 22, 2023

Table of Contents

From the documentation. https://pandas.pydata.org/docs/getting_started/overview.html

The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, DataFrame provides everything that R's data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Here are just a few of the things that pandas does well:

- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Intuitive merging and joining data sets
- Flexible reshaping and pivoting of data sets
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast HDF5 format

[1]:

```
import numpy as np
import pandas as pd
from numpy.random import randn
```

```
#import os
#for dirname, _, filenames in os.walk('/kaggle/input'):
#    for filename in filenames:
#        print(os.path.join(dirname, filename))
```

1 Series and Dataframes

1.1 Series

Series is a one-dimensional labeled array capable of holding any data type. The **axis labels** are collectively referred to as the index. The basic method to create a Series is to call:

s = pd.Series(data, index=index)

1.1.1 Create a Series

[2]:

```
# Use the Series method: s = pd.Series(data, index=index)
# Shift + Tab t osee other parameters

s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
s
```

[2]:

```
a   -1.577340
b   -1.298409
c   -0.642891
d    1.564212
e   -0.243558
dtype: float64
```

[3]:

```
# Index is optional

s = pd.Series(randn(5))  # Don't need np.random because randn was imported.
s
```

[3]:

```
0    0.643103
1    0.977564
2   -1.314306
3   -0.312168
4    1.237914
dtype: float64
```

[4]:

```
# A list, array or dictionary can be used to create a series.

my_list = [5,3,0]
my_arr = np.array([5,3,0])
my_dictionary = {'a':5,'b':3,'c':0}
```

[5]:

```
# Use a list w/o an index

pd.Series(my_list)
```

[5]:

```
0    5
1    3
2    0
dtype: int64
```

[6]:

```
# Use a list w/ an index

pd.Series(my_list, index=['a','b','c'])
```

[6]:

```
a    5
b    3
c    0
dtype: int64
```

[7]:

```
# Use a list w/ a list for the index
i_names = [['a','b','c']]

pd.Series(my_list, i_names)
```

[7]:

```
a    5
b    3
c    0
dtype: int64
```

[8]:

```
# Use an array
my_arr = np.array([5,3,0])
pd.Series(my_arr, index=['a','b','c'])
```

[8]:

```
a    5
b    3
c    0
dtype: int64
```

[9]:

```
# Use a dictionary
my_dictionary = {'a':5,'b':3,'c':0}
pd.Series(my_dictionary, index=['a','b','c'])

# What happens if the index list is changed to hold x,y and z?
```

[9]:

```
a    5
b    3
c    0
dtype: int64
```

[10]:
```
# Using strings
my_cities = ['Chicago','Atlanta','Boston']
pd.Series(my_cities, i_names)
```

[10]:
```
a    Chicago
b    Atlanta
c     Boston
dtype: object
```

[11]:
```
# Use the cities as the labels
my_cities = ['Chicago','Atlanta','Boston']
state = ['IL','GA','MA']
cities = pd.Series(state, my_cities)
cities
```

[11]:
```
Chicago    IL
Atlanta    GA
Boston     MA
dtype: object
```

1.1.2 Using the Series index

[12]:
```
cities['Chicago']
```

[12]:
```
'IL'
```

1.2 Dataframes

DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object.

1.2.1 Create a DataFrame

[13]:
```
np.random.seed(1234)
df = pd.
  ↪DataFrame(randn(4,5),index=['IL','GA','MA','VT'],columns=['Sent','Used','Expired','Lost','D
df
```

[13]:
```
        Sent      Used   Expired      Lost  Destroyed
IL  0.471435 -1.190976  1.432707 -0.312652  -0.720589
GA  0.887163  0.859588 -0.636524  0.015696  -2.242685
MA  1.150036  0.991946  0.953324 -2.021255  -0.334077
VT  0.002118  0.405453  0.289092  1.321158  -1.546906
```

[14]:
```
# A little shortcut
np.random.seed(1234)
```

```
df = pd.DataFrame(randn(4,5),index='IL GA MA VT'.split(),columns='S U E L D'.
 ↪split())
df
```

[14]:

```
           S         U         E         L         D
IL  0.471435 -1.190976  1.432707 -0.312652 -0.720589
GA  0.887163  0.859588 -0.636524  0.015696 -2.242685
MA  1.150036  0.991946  0.953324 -2.021255 -0.334077
VT  0.002118  0.405453  0.289092  1.321158 -1.546906
```

[15]:

```
# Create a DataFrame

data = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2]
}
sales = pd.DataFrame(data)
sales
```

[15]:

```
   apples  oranges
0       3        0
1       2        3
2       0        7
3       1        2
```

1.2.2 Using the DataFrame index

[16]:

```
df
```

[16]:

```
           S         U         E         L         D
IL  0.471435 -1.190976  1.432707 -0.312652 -0.720589
GA  0.887163  0.859588 -0.636524  0.015696 -2.242685
MA  1.150036  0.991946  0.953324 -2.021255 -0.334077
VT  0.002118  0.405453  0.289092  1.321158 -1.546906
```

[17]:

```
# Select a column
df['S']
```

[17]:

```
IL    0.471435
GA    0.887163
MA    1.150036
VT    0.002118
Name: S, dtype: float64
```

[18]:

```
# Select multiple columns
df[['S','E']]            # Outer brackets: [ expecting an arguement] inner␣
 ↪brackets: passing in a list ['a','b']
```

```
[18]:             S         E
      IL  0.471435  1.432707
      GA  0.887163 -0.636524
      MA  1.150036  0.953324
      VT  0.002118  0.289092
```

```
[19]: # Getting a row
      df.loc['IL']
```

```
[19]: S    0.471435
      U   -1.190976
      E    1.432707
      L   -0.312652
      D   -0.720589
      Name: IL, dtype: float64
```

```
[20]: df.iloc[0]
```

```
[20]: S    0.471435
      U   -1.190976
      E    1.432707
      L   -0.312652
      D   -0.720589
      Name: IL, dtype: float64
```

```
[21]: df.iloc[1:3]
```

```
[21]:             S         U         E         L         D
      GA  0.887163  0.859588 -0.636524  0.015696 -2.242685
      MA  1.150036  0.991946  0.953324 -2.021255 -0.334077
```