Pandas Notes

February 22, 2022

1 Pandas

Pandas (derived from the term "**pan**el **da**ta") is Python's primary data analysis library. Built on NumPy, it provides a vast range of data-wrangling capabilites that are fast, flexible, and intuitive. Unlike NumPy, pandas allows for the ingestion of *heterogeneous* data types *via* its two main data structures: pandas **series** and pandas **data frames**.

To begin, execute the following command to import pandas. (Let's also import NumPy for good measure.)

[1]:
```
import pandas as pd

import numpy as np
```

1.1 pandas Series

A pandas *series* is a *one-dimensional* array-like object that allows us to index data is various ways. It acts much like an `ndarray` in NumPy, but supports many more data types such as *integers*, *strings*, *floats*, *Python objects*, etc. The basic syntax to create a pandas series is

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

where

- `data` can be e.g. a Python dictionary, list, or ndarray.
- `index` is a list of axis labels the *same length* as `data`.

Note that Series is like a NumPy array, but we can prescribe *custom indices* instead of the usual numeric 0 to $N-1$.

Creating pandas Series

[26]:
```
# Example: create series using ndarray

s1 = pd.Series(np.arange(0,5), index = ['I', 'II', 'III', 'IV', 'V'])

print(s1)
```

```
I      0
II     1
III    2
IV     3
V      4
dtype: int64
```

One important difference from NumPy is that the entries in `data` do not need to be of the same type.

[27]:
```
# Example: heterogeneous data types

s2 = pd.Series(data = [0.1, 12, 'Bristol', 1000], index = ['a', 'b', 'c', 'd'])

print(s2)
```

```
a        0.1
b         12
c    Bristol
d       1000
dtype: object
```

We can also create a Series from **Python dictionaries**. Note that when a Series is substantiated from a dictionary, *we do not specify the index.*

[4]:
```
d1 = {'q': 8, 'r': 16, 's': 24} # create dictionary

s3 = pd.Series(d1)

print(s3)
```

```
q     8
r    16
s    24
dtype: int64
```

Retrieving the names of Series indices

We can retrieve the Series indices as follows:

[28]:
```
s1.index
```

[28]:
```
Index(['I', 'II', 'III', 'IV', 'V'], dtype='object')
```

Extract elements from Series by index name

To call/extract elements, we use the `.loc[index name]` command. Note the use of *square brackets*. If a label is used that is not in the Series, an exception is raised.

[29]:
```
s2.loc['a']
```

[29]: 0.1

To access multiple entries, we use

[30]:
```
s2.loc[['d', 'c']]
```

[30]:
```
d       1000
c    Bristol
dtype: object
```

Extract elements from Series by integer location (`.iloc`)

Alternatively, we can use the integer-based `.iloc` command that extracts elements based on their numeric index.

[31]:
```
s2.iloc[[2, 3, 0]]
```

[31]:
```
c    Bristol
d       1000
a        0.1
dtype: object
```

1.2 pandas DataFrame

A pandas *DataFrame* is a two-dimensional data structure that supports heterogeneous data with labelled axes for rows and columns. The columns can have different types. DataFrames's are the more commonly used pandas data structures. It can be useful to think of a DataFrame as being analogous to something like a spreadsheet in Excel.

Creating DataFrames

One way to create a pandas DataFrame is through a dictionary of Python Series.

[32]:
```
# Create a DataFrame from dictionary of Python series

d = {'X' : pd.Series(np.arange(0,5), index = ['cheese', 'wine', 'bread',␣
↪'olives', 'gin']),
    'Y' : pd.Series(data = ['Glasgow', 'London', 'Bristol'], index = ['wine',␣
↪'cheese', 'cider'])}

dF = pd.DataFrame(d)
dF
```

[32]:

	X	Y
bread	2.0	NaN
cheese	0.0	London
cider	NaN	Bristol
gin	4.0	NaN
olives	3.0	NaN

```
wine      1.0   Glasgow
```

Let's pause to think a little about the ouput here. In particular, note the occurence of the values NaN in both columns. We note that the indices are the *union* of the indices of the various Series that make up our data frame. In other words, the indices are merged.

There are numerous other ways to construct DataFrames in pandas. In the **Worksheet**, you will learn how to create a DataFrame from a *list of Python dictionaries*.

Retrieving DataFrame index and column names

To obtain the DataFrame index and column names, we execute:

[35]:
```
dF.index
```

[35]:
```
Index(['bread', 'cheese', 'cider', 'gin', 'olives', 'wine'], dtype='object')
```

[36]:
```
dF.columns
```

[36]:
```
Index(['X', 'Y'], dtype='object')
```

[37]:
```
dF['X']
```

[37]:
```
bread      2.0
cheese     0.0
cider      NaN
gin        4.0
olives     3.0
wine       1.0
Name: X, dtype: float64
```

Indexing & selection

Indexing DataFrames follows essentially the same syntax as Series. To access:

- a column, we use `dF[column name]` OR dF.column name
- a row, we use either (i) its index label `dF.loc[index label]` or (ii) its integer location `dF.iloc[integer location]`
- multiple rows, we use slice indexing e.g. `dF[0:3]`. **Note**: if you try to use a single integer, `dF[0]` say, an exception will be thrown as pandas thinks you're trying to access a column called 0.

[38]:
```
# By column

print(dF['X'])
print()
print(dF.X)
print()
```

```python
# By row, index

print(dF.loc['bread'])
print()

# By row, integer location

print(dF.iloc[1])
print()

# Multiple rows by integer location

print(dF[0:3])
print()
```

```
bread     2.0
cheese    0.0
cider     NaN
gin       4.0
olives    3.0
wine      1.0
Name: X, dtype: float64

bread     2.0
cheese    0.0
cider     NaN
gin       4.0
olives    3.0
wine      1.0
Name: X, dtype: float64

X      2
Y    NaN
Name: bread, dtype: object

X         0
Y    London
Name: cheese, dtype: object

          X        Y
bread   2.0      NaN
cheese  0.0   London
cider   NaN  Bristol
```

Boolean indexing

Like in NumPy we can apply *Boolean filtering/indexing* to extract specific elements in a DataFrame.

[39]:
```
dF
```

[39]:
```
           X        Y
bread    2.0      NaN
cheese   0.0   London
cider    NaN  Bristol
gin      4.0      NaN
olives   3.0      NaN
wine     1.0  Glasgow
```

[40]:
```
# Extract the rows of dF where the values in the column X are greater than 2.

dF_new = dF[dF['X'] > 2]
dF_new
```

[40]:
```
           X    Y
gin      4.0  NaN
olives   3.0  NaN
```

Here we apply a Boolean filter `dF['X'] > 2` which gives the values True or False for each value in the column `X` depending on whether the condition is satisfied or not. We then provide this indexing to the DataFrame dF to extract the rows where the condition is satisfied, giving a new DataFrame `dF`.

1.3 Data ingestion

Pandas really comes into its own when dealing with large data sets with potentially millions of entries of different data types and formats.

We will concentrate here on the NBA Players Database (called `NBA_Stats.csv`), a publicly available database of NBA statistics on the website Kaggle, which provides basic statistics on NBA basketball players up to the year 2020. To import the `.csv` file, we use the pandas function `.read_csv()`.

[41]:
```
NBA = pd.read_csv('./NBA_Stats.csv', sep = ',')

print(type(NBA))
```

```
<class 'pandas.core.frame.DataFrame'>
```

We can get some information about our DataFrame NBA using the `.info()` command. This shows us that the DataFrame has 22 columns of information and 11700 rows. Note the data types of each column. Further, notice that the indices in this DataFrame are just the integers 0 to 11700.

[42]:
```
NBA.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11700 entries, 0 to 11699
Data columns (total 22 columns):
```

```
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Unnamed: 0          11700 non-null  int64
 1   player_name         11700 non-null  object
 2   team_abbreviation   11700 non-null  object
 3   age                 11700 non-null  float64
 4   player_height       11700 non-null  float64
 5   player_weight       11700 non-null  float64
 6   college             11700 non-null  object
 7   country             11700 non-null  object
 8   draft_year          11700 non-null  object
 9   draft_round         11700 non-null  object
 10  draft_number        11700 non-null  object
 11  gp                  11700 non-null  int64
 12  pts                 11700 non-null  float64
 13  reb                 11700 non-null  float64
 14  ast                 11700 non-null  float64
 15  net_rating          11700 non-null  float64
 16  oreb_pct            11700 non-null  float64
 17  dreb_pct            11700 non-null  float64
 18  usg_pct             11700 non-null  float64
 19  ts_pct              11700 non-null  float64
 20  ast_pct             11700 non-null  float64
 21  season              11700 non-null  object
dtypes: float64(12), int64(2), object(8)
memory usage: 2.0+ MB
```

We can view the first few rows using the `.head()` function (which prints the first 5 rows by default) or the last few rows using `.tail()`.

[43]:
```python
# Print the first 10 rows

NBA.head()
```

[43]:
```
   Unnamed: 0       player_name team_abbreviation   age  player_height  \
0           0     Travis Knight               LAL  22.0         213.36
1           1         Matt Fish               MIA  27.0         210.82
2           2      Matt Bullard               HOU  30.0         208.28
3           3      Marty Conlon               BOS  29.0         210.82
4           4  Martin Muursepp               DAL  22.0         205.74

   player_weight                    college country draft_year draft_round  \
0      106.59412                Connecticut     USA       1996           1
1      106.59412  North Carolina-Wilmington     USA       1992           2
2      106.59412                       Iowa     USA  Undrafted   Undrafted
3      111.13004                 Providence     USA  Undrafted   Undrafted
4      106.59412                       None     USA       1996           1
```

```
   …  pts  reb  ast  net_rating  oreb_pct  dreb_pct  usg_pct  ts_pct  \
0  …  4.8  4.5  0.5         6.2     0.127     0.182    0.142   0.536
1  …  0.3  0.8  0.0       -15.1     0.143     0.267    0.265   0.333
2  …  4.5  1.6  0.9         0.9     0.016     0.115    0.151   0.535
3  …  7.8  4.4  1.4        -9.0     0.083     0.152    0.167   0.542
4  …  3.7  1.6  0.5       -14.5     0.109     0.118    0.233   0.482

   ast_pct   season
0    0.052  1996-97
1    0.000  1996-97
2    0.099  1996-97
3    0.101  1996-97
4    0.114  1996-97

[5 rows x 22 columns]
```

[44]:

```python
# Print the last 10 rows

NBA.tail()
```

[44]:

```
       Unnamed: 0          player_name team_abbreviation   age  player_height  \
11695       11695  Matthew Dellavedova               CLE  30.0         190.50
11696       11696     Maurice Harkless               SAC  28.0         200.66
11697       11697            Max Strus               MIA  25.0         195.58
11698       11698    Marcus Morris Sr.               LAC  31.0         203.20
11699       11699         Aaron Gordon               DEN  25.0         203.20

       player_weight                           college    country draft_year  \
11695      90.718400  St.Mary's College of California  Australia  Undrafted
11696      99.790240                        St. John's        USA       2012
11697      97.522280                            DePaul        USA  Undrafted
11698      98.883056                            Kansas        USA       2011
11699     106.594120                           Arizona        USA       2014

      draft_round  …   pts  reb  ast  net_rating  oreb_pct  dreb_pct  \
11695   Undrafted  …   2.8  1.8  4.5        -3.1     0.029     0.085
11696           1  …   5.2  2.4  1.2        -2.9     0.017     0.097
11697   Undrafted  …   6.1  1.1  0.6        -4.2     0.011     0.073
11698           1  …  13.4  4.1  1.0         4.2     0.025     0.133
11699           1  …  12.4  5.7  3.2         2.1     0.055     0.150

       usg_pct  ts_pct  ast_pct   season
11695    0.125   0.312    0.337  2020-21
11696    0.114   0.527    0.071  2020-21
11697    0.179   0.597    0.074  2020-21
11698    0.194   0.614    0.056  2020-21
11699    0.204   0.547    0.165  2020-21
```

```
[5 rows x 22 columns]
```