02-DataFrames

October 19, 2022

---

1 DataFrames

DataFrames are the workhorse of pandas and are directly inspired by the R programming language. We can think of a DataFrame as a bunch of Series objects put together to share the same index. Let's use pandas to explore this topic!

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
from numpy.random import randn
np.random.seed(101)
```

[3]:
```
df = pd.DataFrame(randn(5,4),index='A B C D E'.split(),columns='W X Y Z'.
 ↪split())
```

[4]:
```
df
```

[4]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

1.1 Selection and Indexing

Let's learn the various methods to grab data from a DataFrame

[5]:
```
df['W']
```

```
[5]: A    2.706850
     B    0.651118
     C   -2.018168
     D    0.188695
     E    0.190794
     Name: W, dtype: float64
```

```
[6]: # Pass a list of column names
     df[['W','Z']]
```

```
[6]:           W         Z
     A  2.706850  0.503826
     B  0.651118  0.605965
     C -2.018168 -0.589001
     D  0.188695  0.955057
     E  0.190794  0.683509
```

```
[7]: # SQL Syntax (NOT RECOMMENDED!)
     df.W
```

```
[7]: A    2.706850
     B    0.651118
     C   -2.018168
     D    0.188695
     E    0.190794
     Name: W, dtype: float64
```

DataFrame Columns are just Series

```
[8]: type(df['W'])
```

```
[8]: pandas.core.series.Series
```

1.1.1 Creating a new column:

```
[9]: df['new'] = df['W'] + df['Y']
```

```
[10]: df
```

```
[10]:           W         X         Y         Z       new
      A  2.706850  0.628133  0.907969  0.503826  3.614819
      B  0.651118 -0.319318 -0.848077  0.605965 -0.196959
      C -2.018168  0.740122  0.528813 -0.589001 -1.489355
      D  0.188695 -0.758872 -0.933237  0.955057 -0.744542
      E  0.190794  1.978757  2.605967  0.683509  2.796762
```

1.1.2 Removing Columns

[11]:
```
df.drop('new',axis=1)
```

[11]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[12]:
```
# Not inplace unless specified!
df
```

[12]:
```
          W         X         Y         Z       new
A  2.706850  0.628133  0.907969  0.503826  3.614819
B  0.651118 -0.319318 -0.848077  0.605965 -0.196959
C -2.018168  0.740122  0.528813 -0.589001 -1.489355
D  0.188695 -0.758872 -0.933237  0.955057 -0.744542
E  0.190794  1.978757  2.605967  0.683509  2.796762
```

[13]:
```
df.drop('new',axis=1,inplace=True)
```

[14]:
```
df
```

[14]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

Can also drop rows this way:

[15]:
```
df.drop('E',axis=0)
```

[15]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
```

1.1.3 Selecting Rows

[16]:
```
df.loc['A']
```

[16]:
```
W    2.706850
X    0.628133
Y    0.907969
Z    0.503826
Name: A, dtype: float64
```

Or select based off of position instead of label

[17]:
```
df.iloc[2]
```

[17]:
```
W   -2.018168
X    0.740122
Y    0.528813
Z   -0.589001
Name: C, dtype: float64
```

1.1.4 Selecting subset of rows and columns

[18]:
```
df.loc['B','Y']
```

[18]:
```
-0.8480769834036315
```

[19]:
```
df.loc[['A','B'],['W','Y']]
```

[19]:
```
          W         Y
A  2.706850  0.907969
B  0.651118 -0.848077
```

1.1.5 Conditional Selection

An important feature of pandas is conditional selection using bracket notation, very similar to numpy:

[20]:
```
df
```

[20]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[21]:
```
df>0
```

```
[21]:        W      X      Y      Z
      A   True   True   True   True
      B   True  False  False   True
      C  False   True   True  False
      D   True  False  False   True
      E   True   True   True   True
```

```
[22]: df[df>0]
```

```
[22]:          W         X         Y         Z
      A  2.706850  0.628133  0.907969  0.503826
      B  0.651118       NaN       NaN  0.605965
      C       NaN  0.740122  0.528813       NaN
      D  0.188695       NaN       NaN  0.955057
      E  0.190794  1.978757  2.605967  0.683509
```

```
[23]: df[df['W']>0]
```

```
[23]:          W         X         Y         Z
      A  2.706850  0.628133  0.907969  0.503826
      B  0.651118 -0.319318 -0.848077  0.605965
      D  0.188695 -0.758872 -0.933237  0.955057
      E  0.190794  1.978757  2.605967  0.683509
```

```
[24]: df[df['W']>0]['Y']
```

```
[24]: A    0.907969
      B   -0.848077
      D   -0.933237
      E    2.605967
      Name: Y, dtype: float64
```

```
[25]: df[df['W']>0][['Y','X']]
```

```
[25]:          Y         X
      A  0.907969  0.628133
      B -0.848077 -0.319318
      D -0.933237 -0.758872
      E  2.605967  1.978757
```

For two conditions you can use | and & with parenthesis:

```
[26]: df[(df['W']>0) & (df['Y'] > 1)]
```

```
[26]:          W         X         Y         Z
      E  0.190794  1.978757  2.605967  0.683509
```

1.2 More Index Details

Let's discuss some more features of indexing, including resetting the index or setting it something else. We'll also talk about index hierarchy!

[27]:
```
df
```

[27]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[28]:
```
# Reset to default 0,1...n index
df.reset_index()
```

[28]:
```
  index         W         X         Y         Z
0     A  2.706850  0.628133  0.907969  0.503826
1     B  0.651118 -0.319318 -0.848077  0.605965
2     C -2.018168  0.740122  0.528813 -0.589001
3     D  0.188695 -0.758872 -0.933237  0.955057
4     E  0.190794  1.978757  2.605967  0.683509
```

[29]:
```
newind = 'CA NY WY OR CO'.split()
```

[30]:
```
df['States'] = newind
```

[31]:
```
df
```

[31]:
```
          W         X         Y         Z States
A  2.706850  0.628133  0.907969  0.503826     CA
B  0.651118 -0.319318 -0.848077  0.605965     NY
C -2.018168  0.740122  0.528813 -0.589001     WY
D  0.188695 -0.758872 -0.933237  0.955057     OR
E  0.190794  1.978757  2.605967  0.683509     CO
```

[32]:
```
df.set_index('States')
```

[32]:
```
               W         X         Y         Z
States
CA      2.706850  0.628133  0.907969  0.503826
NY      0.651118 -0.319318 -0.848077  0.605965
WY     -2.018168  0.740122  0.528813 -0.589001
OR      0.188695 -0.758872 -0.933237  0.955057
CO      0.190794  1.978757  2.605967  0.683509
```

[33]:
```
df
```

[33]:
```
          W         X         Y         Z States
A  2.706850  0.628133  0.907969  0.503826     CA
B  0.651118 -0.319318 -0.848077  0.605965     NY
C -2.018168  0.740122  0.528813 -0.589001     WY
D  0.188695 -0.758872 -0.933237  0.955057     OR
E  0.190794  1.978757  2.605967  0.683509     CO
```

[34]:
```
df.set_index('States',inplace=True)
```

[35]:
```
df
```

[35]:
```
               W         X         Y         Z
States
CA      2.706850  0.628133  0.907969  0.503826
NY      0.651118 -0.319318 -0.848077  0.605965
WY     -2.018168  0.740122  0.528813 -0.589001
OR      0.188695 -0.758872 -0.933237  0.955057
CO      0.190794  1.978757  2.605967  0.683509
```

1.3 DataFrame Summaries

There are a couple of ways to obtain summary data on DataFrames. df.describe() provides summary statistics on all numerical columns. df.info and df.dtypes displays the data type of all columns.

[36]:
```
df.describe()
```

[36]:
```
              W         X         Y         Z
count  5.000000  5.000000  5.000000  5.000000
mean   0.343858  0.453764  0.452287  0.431871
std    1.681131  1.061385  1.454516  0.594708
min   -2.018168 -0.758872 -0.933237 -0.589001
25%    0.188695 -0.319318 -0.848077  0.503826
50%    0.190794  0.628133  0.528813  0.605965
75%    0.651118  0.740122  0.907969  0.683509
max    2.706850  1.978757  2.605967  0.955057
```

[38]:
```
df.dtypes
```

[38]:
```
W    float64
X    float64
Y    float64
Z    float64
dtype: object
```

[40]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5 entries, CA to CO
Data columns (total 4 columns):
W    5 non-null float64
X    5 non-null float64
Y    5 non-null float64
Z    5 non-null float64
dtypes: float64(4)
memory usage: 200.0+ bytes
```

2 Great Job!