3-Pandas

September 13, 2019

___ ## Pedram Jahangiry (Fall 2019)

1 Introduction to Pandas

Topics to be covered:

1. Series
2. DataFrames
3. Missing Variables
4. Operations
5. Data import and export

Make sure you have access to the pandas cheatsheet provided in the course folder.

1.1 1. Series

Series are very similar to NumPy arrays. The difference is that a Series can have axis labels, meaning it can be indexed by a label, instead of just a number location. We can convert a list,numpy array, or dictionary to a Series.

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

[2]:
```
my_list = [1,2,3]
pd.Series(data=my_list)
```

[2]:
```
0    1
1    2
2    3
dtype: int64
```

[3]:
```
labels = ['a','b','c']
pd.Series(data=my_list,index=labels)
```

[3]:
```
a    1
b    2
c    3
dtype: int64
```

[4]:
```
my_array = np.array([1,2,3])
pd.Series(my_array)
```

[4]:
```
0    1
1    2
2    3
dtype: int32
```

[5]:
```
pd.Series(my_array,labels)
```

[5]:
```
a    1
b    2
c    3
dtype: int32
```

[6]:
```
my_dict = {'a':1,'b':2,'c':3}
pd.Series(my_dict)
```

[6]:
```
a    1
b    2
c    3
dtype: int64
```

[7]:
```
my_series = pd.Series(my_dict)
```

[8]:
```
my_series[0]          # unlike dictionaries, we can extract info by index number␣
 ↪and lable.
```

[8]:
```
1
```

[9]:
```
my_series['b']
```

[9]:
```
2
```

1.2 2. DataFrames

DataFrames are directly inspired by the R programming language and are the workhorse of pandas.

[10]:
```
np.random.seed(100)  # do this if you want to see the same results as mine
```

[11]:
```
df = pd.DataFrame(np.random.randn(4,4),index='A B C D'.split(),columns='W X Y␣
 ↪Z'.split())
df
```

[11]:
```
          W         X         Y         Z
A -1.749765  0.342680  1.153036 -0.252436
B  0.981321  0.514219  0.221180 -1.070043
C -0.189496  0.255001 -0.458027  0.435163
D -0.583595  0.816847  0.672721 -0.104411
```

[12]:
```
df.describe()
```

[12]:
```
              W         X         Y         Z
count  4.000000  4.000000  4.000000  4.000000
mean  -0.385384  0.482187  0.397227 -0.247932
std    1.126511  0.247722  0.685467  0.622657
```

```
min   -1.749765  0.255001 -0.458027 -1.070043
25%   -0.875138  0.320761  0.051378 -0.456838
50%   -0.386545  0.428450  0.446950 -0.178424
75%    0.103208  0.589876  0.792800  0.030483
max    0.981321  0.816847  1.153036  0.435163
```

[15]:
```
df.describe().transpose()              # or equivalently, df.describe().T
```

[15]:

	count	mean	std	min	25%	50%	75%	max
W	4.0	-0.385384	1.126511	-1.749765	-0.875138	-0.386545	0.103208	0.981321
X	4.0	0.482187	0.247722	0.255001	0.320761	0.428450	0.589876	0.816847
Y	4.0	0.397227	0.685467	-0.458027	0.051378	0.446950	0.792800	1.153036
Z	4.0	-0.247932	0.622657	-1.070043	-0.456838	-0.178424	0.030483	0.435163

1.2.1 Indexing and extraction

[38]:
```
df['W']       # this is equivalent to df.W (which I don't recommend you to use
 ↪it). == df$W in R
```

[38]:
```
A   -1.749765
B    0.981321
C   -0.189496
D   -0.583595
Name: W, dtype: float64
```

[39]:
```
df[['W']]
```

[39]:

	W
A	-1.749765
B	0.981321
C	-0.189496
D	-0.583595

[36]:
```
df[['W','Y']]
```

[36]:

	W	Y
A	-1.749765	1.153036
B	0.981321	0.221180
C	-0.189496	-0.458027
D	-0.583595	0.672721

[55]:
```
df['new'] = df['W'] + df['Y']
```

[56]:
```
df
```

[56]:

	W	X	Y	Z	new
A	-1.749765	0.342680	1.153036	-0.252436	-0.596730
B	0.981321	0.514219	0.221180	-1.070043	1.202500
C	-0.189496	0.255001	-0.458027	0.435163	-0.647523
D	-0.583595	0.816847	0.672721	-0.104411	0.089126

[47]:
```
df.drop('A',axis=0)
```

[47]:
```
          W         X         Y         Z
B  0.981321  0.514219  0.221180 -1.070043
C -0.189496  0.255001 -0.458027  0.435163
D -0.583595  0.816847  0.672721 -0.104411
```

[43]:
```
df.drop('new',axis=1)
```

[43]:
```
          W         X         Y         Z
A -1.749765  0.342680  1.153036 -0.252436
B  0.981321  0.514219  0.221180 -1.070043
C -0.189496  0.255001 -0.458027  0.435163
D -0.583595  0.816847  0.672721 -0.104411
```

[44]:
```
df
```

[44]:
```
          W         X         Y         Z       new
A -1.749765  0.342680  1.153036 -0.252436 -0.596730
B  0.981321  0.514219  0.221180 -1.070043  1.202500
C -0.189496  0.255001 -0.458027  0.435163 -0.647523
D -0.583595  0.816847  0.672721 -0.104411  0.089126
```

[45]:
```
df.drop('new',axis=1,inplace=True)
# or alternatively use: df = df.drop('new', 1)
```

[46]:
```
df
```

[46]:
```
          W         X         Y         Z
A -1.749765  0.342680  1.153036 -0.252436
B  0.981321  0.514219  0.221180 -1.070043
C -0.189496  0.255001 -0.458027  0.435163
D -0.583595  0.816847  0.672721 -0.104411
```

[50]:
```
# we can select a row by calling its label or by selecting based on its␣
 ↪position instead of label
df.loc['A']
```

[50]:
```
W   -1.749765
X    0.342680
Y    1.153036
Z   -0.252436
Name: A, dtype: float64
```

[21]:
```
df.iloc[0]
```

[21]:
```
W   -1.749765
X    0.342680
Y    1.153036
Z   -0.252436
Name: A, dtype: float64
```

[22]:
```
df.iloc[np.arange(2)]
```

[22]:
```
          W         X         Y         Z
A -1.749765  0.342680  1.153036 -0.252436
B  0.981321  0.514219  0.221180 -1.070043
```

[24]:
```
df.loc[['A','D'],['W','Z']]
```

[24]:
```
          W         Z
A -1.749765 -0.252436
D -0.583595 -0.104411
```

1.2.2 Conditional extraction

This is very similar to numpy conditional extraction

[61]:
```
df
```

[61]:
```
          W         X         Y         Z
A -1.749765  0.342680  1.153036 -0.252436
B  0.981321  0.514219  0.221180 -1.070043
C -0.189496  0.255001 -0.458027  0.435163
D -0.583595  0.816847  0.672721 -0.104411
```

[62]:
```
df>0
```

[62]:
```
       W     X      Y      Z
A  False  True   True  False
B   True  True   True  False
C  False  True  False   True
D  False  True   True  False
```

[63]:
```
df[df>0]
```

[63]:
```
          W         X         Y         Z
A       NaN  0.342680  1.153036       NaN
B  0.981321  0.514219  0.221180       NaN
C       NaN  0.255001       NaN  0.435163
D       NaN  0.816847  0.672721       NaN
```

[68]:
```
df[df['Y']>0]
```

[68]:
```
          W         X         Y         Z
A -1.749765  0.342680  1.153036 -0.252436
B  0.981321  0.514219  0.221180 -1.070043
D -0.583595  0.816847  0.672721 -0.104411
```

[69]:
```
df[df['Y']>0]['X']
```

[69]:
```
A    0.342680
B    0.514219
D    0.816847
Name: X, dtype: float64
```

[71]:
```
df[df['Y']>0][['Y','Z']]
```

[71]:
```
          Y         Z
A  1.153036 -0.252436
B  0.221180 -1.070043
D  0.672721 -0.104411
```

[72]:
```
df[(df['Y']>0) & (df['Z'] < -0.5)]
```

[72]:
```
          W         X        Y         Z
B  0.981321  0.514219  0.22118 -1.070043
```

1.3 3. Missing variables

[25]:
```
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})
df
```

[25]:
```
     A    B  C
0  1.0  5.0  1
1  2.0  NaN  2
2  NaN  NaN  3
```

[160]:
```
df.isnull()
```

[160]:
```
       A      B      C
0  False  False  False
1  False   True  False
2   True   True  False
```

[26]:
```
df.dropna() # by default axis=0,   # this is similar to df[complete.cases(df),]␣
↪in R
```

[26]:
```
     A    B  C
0  1.0  5.0  1
```

[27]:
```
df.dropna(axis=1)
```

[27]:
```
   C
0  1
1  2
2  3
```

[28]:
```
df.dropna(thresh=2) # how many elements in an observation is NaN?
```

[28]:
```
     A    B  C
0  1.0  5.0  1
1  2.0  NaN  2
```

[29]:
```
df.fillna(value='new value')
```

[29]:
```
           A          B  C
0          1          5  1
1          2  new value  2
```

```
2  new value  new value  3
```

[32]:
```
df['A'].fillna(value=df['A'].mean()) # filling the value with the mean of a␣
↪column
```

[32]:
```
0    1.0
1    2.0
2    1.5
Name: A, dtype: float64
```

1.4 4. Operations

[33]:
```
df = pd.DataFrame({'names': 'PJ PJ TJ MJ'.split() ,
                   'GPA': [4,4,3.8,3.5]}, index='A B C D'.split())
df
```

[33]:
```
  names  GPA
A    PJ  4.0
B    PJ  4.0
C    TJ  3.8
D    MJ  3.5
```

[168]:
```
df.head(3)
```

[168]:
```
  names  GPA
A    PJ  4.0
B    PJ  4.0
C    TJ  3.8
```

[169]:
```
df.tail(1)
```

[169]:
```
  names  GPA
D    MJ  3.5
```

[170]:
```
# Unique Values
df['names'].unique()
```

[170]:
```
array(['PJ', 'TJ', 'MJ'], dtype=object)
```

[172]:
```
# number of unique values
df['GPA'].nunique()
```

[172]:
```
3
```

[173]:
```
df['names'].value_counts()      # this is table(df$names) in R
```

[173]:
```
PJ    2
MJ    1
TJ    1
Name: names, dtype: int64
```

[176]:
```
# Applying Functions
df['GPA'].mean()
```

[176]:
```
3.825
```

[179]:
```
round(df['GPA'].std(), 2)
```

[179]:
```
0.24
```

[36]:
```
df['GPA_100'] = df['GPA'].apply(lambda x: x*25)   # of ocurse we are looking␣
 ↪for some special functions not just *25
df
```

[36]:
```
  names  GPA  GPA_100
A    PJ  4.0    100.0
B    PJ  4.0    100.0
C    TJ  3.8     95.0
D    MJ  3.5     87.5
```

[38]:
```
df['is_pass'] = ['pass' if x > 3.5 else 'fail' for x in df['GPA']]
# in R: df <- mutate(df,is_pass = ifelse(GPA>3.5, "pass", "fail"))
df
```

[38]:
```
  names  GPA  GPA_100  pass
A    PJ  4.0    100.0  pass
B    PJ  4.0    100.0  pass
C    TJ  3.8     95.0  pass
D    MJ  3.5     87.5  fail
```

[184]:
```
df.columns            # names(df) in R
```

[184]:
```
Index(['names', 'GPA', 'GPA_100'], dtype='object')
```

[185]:
```
df.index
```

[185]:
```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

[186]:
```
df.sort_values(by='GPA') #inplace=False by default (what does this mean?)
```

[186]:
```
  names  GPA  GPA_100
D    MJ  3.5     87.5
C    TJ  3.8     95.0
A    PJ  4.0    100.0
B    PJ  4.0    100.0
```

[190]:
```
df.reset_index(inplace=True)
```

[191]:
```
df
```

[191]:
```
  index names  GPA  GPA_100
0     A    PJ  4.0    100.0
1     B    PJ  4.0    100.0
2     C    TJ  3.8     95.0
3     D    MJ  3.5     87.5
```

1.5 5. Data import and export

[194]:
```
# reading from CSV file
df = pd.read_csv('GDP.csv')   # reading excel files: pd.read_excel('GDP.
 ↪xlsx',sheetname='Sheet1')
df.tail(5)
```

[194]:

	DATE	GDP
285	2018-04-01	20510.177
286	2018-07-01	20749.752
287	2018-10-01	20897.804
288	2019-01-01	21098.827
289	2019-04-01	21339.121

[196]:
```
# Writing to CSV file
df.to_csv('GDP_new.csv',index=False)   # writing to excel files: df.
 ↪to_excel('GDP.xlsx',sheet_name='raw data')
```

[198]:
```
df.to_excel('GDP.xlsx',sheet_name='raw data', index=False)
```