Pandas

Thomas Schwarz, SJ

Basics

- Tool for data sets:
  - Analysis
  - Aggregation
  - Cleaning
  - Merging
  - Pivoting

Basics

- Where to get Pandas
  - Install via pip or homebrew
  - Use a distribution like Anaconda
    - Comes with Jupyter (aka iPython) Notebooks which are popular among data scientists

Pandas Series

- A one-dimensional array
  - Can hold data of any type
  - Axis labels are called index

Pandas Series

- Can create using a scalar that is going to be repeated.
  - An index needs to be explicitly provided

```
pd.Series(5,index)
```

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

Pandas Series

- Default Index is np.arange(n), i.e. the numbers from 0, ...,
  - Example:

```
import pandas as pd
lit_it_isl = pd.Series(['elba', 'ischia', 'capri'])
```

- creates a Pandas Series

```
0      elba
1    ischia
2     capri
dtype: object
```

Pandas Series

- dtype is the type of the Series
  - In this case, it is object because the data consists of strings

Pandas Series

- We can create an explicit index

```
labels = ['nice', 'nicer', 'nicest']
data = ['elba', 'ischia', 'capri']
lit_it_isl = pd.Series(data = data, index = labels)
```

- When we print out the result, we now see the index

```
nice         elba
nicer      ischia
nicest      capri
dtype: object
```

Pandas Series

- There are a number of data sources
  - Can create using a Python list
  - Can create using a dictionary

```
isl_dic={'nice':'elba', 'nicer':'ischia', 'nicest':'capri'}
>>> lit_it_isl = pd.Series(isl_dict)
```

  - Can create using a numpy array

```
pd.Series(np.random.uniform(0,1,5))
0    0.644686
1    0.812248
2    0.496581
3    0.876687
4    0.280538
dtype: float64
```

Pandas Series

- There is no limit imposed on the objects that can be stored
  - For example, we can store functions

```
pd.Series([random.uniform, print, len, "".join])
0    <bound method Random.uniform of <random.Random...
1                                  <built-in function print>
2                                    <built-in function len>
3    <built-in method join of str object at 0x104c3...
dtype: object
```

Pandas Series

- To retrieve a data value, we give it the index

```
lit_it_isl['nicer']
'ischia'
```

Pandas Series

- The slice operation works differently

```
ex = pd.Series(['capri', 'ischia', 'elba',
                'giglia', 'giannutri'],
               index=list('abcde'))
```

```
a         capri
b       ischia
c         elba
d       giglia
e    giannutri
dtype: object
```

Pandas Series

- Both the beginning and the end are included

```
ex['b':'d']
b    ischia
c      elba
d    giglia
dtype: object
```

Pandas Series

- Slices:
  - Like in NumPy, a slice only creates a **reference**
    - If you change a slice, you change the original
      - Example: Create a series

```
ex = pd.Series(['capri', 'ischia', 'elba', 'giglia',
'giannutri'], index=list('abcde'))
```

      - Create a slice

```
my_slice = ex['b':'d']
```

Pandas Series

- Slices are references (cont.)
    - Change the slice

```
my_slice = ex['b':'d']
```

    - The original (as well as the slice) have changed

```
ex
a         capri
b       ischia
c       zanone
d       giglia
e    giannutri
dtype: object
```

Pandas Series

- If an index is not in the series, a KeyError is raised

```
ex['h']
Traceback (most recent call last):
...

KeyError: 'h'
>>>
```

Pandas Series

- As we have seen, we can use indexing to update a value

Pandas Series

- In addition to explicit indexing with the [ ] operator
  - Can use subsets referring explicit indices (offsets)
    - with the .loc operator
    - with the .iloc

Pandas Series

- Example:
  - Define a series based on the Olympic ice-hockey tournament 2018

```
icehockey2018 = pd.Series({'russia': 1, 'germany': 2,
'canada': 3, 'czech': 4, 'sweden':5})
>>> icehockey2018
russia     1
germany    2
canada     3
czech      4
sweden     5
dtype: int64
```

Pandas Series

- Example
  - Using .loc with a list of labels

```
icehockey2018.loc[['russia', 'sweden']]
russia    1
sweden    5
dtype: int64
```

Pandas Series

- Example
  - Accessing a sub-series with iloc by numerical index

```
icehockey2018.iloc[1:3]
```

```
germany    2
canada     3
dtype: int64
```

Pandas Series

- Example
  - Using a series of integer indices with .iloc

```
icehockey2018.iloc[[1,2,3,4]]
```

```
germany    2
canada     3
czech      4
sweden     5
dtype: int64
```

Pandas Series

- Just as for numpy arrays, we can use operations between series
- These are dependent on labels

Pandas Series

- Example: Olympic Ice-hockey results

```
icehockey2018 = pd.Series({'russia': 1, 'germany': 2,
'canada': 3, 'czech': 4, 'sweden':5})
>>> icehockey2018
russia     1
germany    2
canada     3
czech      4
sweden     5
dtype: int64
```

Pandas Series

```
icehockey2014= pd.Series({'canada':1, 'sweden':2,
'finland':3, 'usa': 4, 'czech':5})
```

Pandas Series

- Calculate the average, and we get lot's of Not a Number (NaN)

```
(icehockey2018+icehockey2014)/2
```

```
canada      2.0
czech       NaN
finland     NaN
germany     NaN
russia      3.0
sweden      3.5
usa         NaN
dtype: float64
```

Pandas Dataframe

- A two-dimensional table

```
example = pd.DataFrame(np.random.randn(5,4),
['a','b','c','d','e'],['w','x','y','z'])
```

```
>>> example
          w         x         y         z
a  0.968015 -0.292712 -0.456712  0.478160
b -0.182741  0.801120  1.466134  0.883498
c  0.497248 -0.170697 -0.487031  3.018604
d  0.948902 -0.878197  0.796428 -0.479922
e -1.420614  0.200272  1.111076 -0.283730
```

Pandas Dataframe

- Access to data uses the bracket [ ] operation
  - Example (continued):

```
example['w']
```

```
a    0.968015
b   -0.182741
c    0.497248
d    0.948902
e   -1.420614
Name: w, dtype: float64
```

Pandas Dataframe

- Example (continued)

```
example[['w','z']]
```

```
          w         z
a  0.968015  0.478160
b -0.182741  0.883498
c  0.497248  3.018604
d  0.948902 -0.479922
e -1.420614 -0.283730
```

Pandas Dataframe

- The rows are given by an "index"
  - Columns can be added

```
example['summa']=example['w']+example['x']+
                 example['y']+example['z']
```

```
          w         x         y         z     summa
a  0.968015 -0.292712 -0.456712  0.478160  0.696751
b -0.182741  0.801120  1.466134  0.883498  2.968011
c  0.497248 -0.170697 -0.487031  3.018604  2.858124
d  0.948902 -0.878197  0.796428 -0.479922  0.387211
e -1.420614  0.200272  1.111076 -0.283730 -0.392997
```

Pandas Dataframe

- Columns can also be deleted
  - Use drop
    - drop has a parameter axis
      - Axis 0: drop an index
      - Axis 1: drop a column

Pandas Dataframe

- Example:
  - Drop the first column with label 'w'

```
example.drop('w',axis=1)
```

```
          x         y         z      summa
a -0.292712 -0.456712  0.478160  0.696751
b  0.801120  1.466134  0.883498  2.968011
c -0.170697 -0.487031  3.018604  2.858124
d -0.878197  0.796428 -0.479922  0.387211
e  0.200272  1.111076 -0.283730 -0.392997
```

Pandas Dataframe

- Example (continued)
  - But this does not change the original dataframe

```
example
```

	w	x	y	z	summa
a	0.968015	-0.292712	-0.456712	0.478160	0.696751
b	-0.182741	0.801120	1.466134	0.883498	2.968011
c	0.497248	-0.170697	-0.487031	3.018604	2.858124
d	0.948902	-0.878197	0.796428	-0.479922	0.387211
e	-1.420614	0.200272	1.111076	-0.283730	-0.392997

Pandas Dataframe

- To make the change to the original, need to specify that the inplace parameter is True
    - Otherwise, we are just making a copy
    - This is really a bit of a headache
        - Need to lookup manual to figure out whether an operation makes a copy or changes the original

Pandas Dataframe

- Example: With inplace being True, we change the dataframe itself

```
example.drop('w',axis=1, inplace=True)
```

```
example
          x         y         z      summa
a -0.292712 -0.456712  0.478160  0.696751
b  0.801120  1.466134  0.883498  2.968011
c -0.170697 -0.487031  3.018604  2.858124
d -0.878197  0.796428 -0.479922  0.387211
e  0.200272  1.111076 -0.283730 -0.392997
```

Pandas Dataframe Selftest

- Drop a row from an example dataframe

Pandas Dataframe Self-test Solution

- Just use axis = 0

```
example.drop('e', axis=0, inplace = True)
```

```
          x         y         z     summa
a -0.292712 -0.456712  0.478160  0.696751
b  0.801120  1.466134  0.883498  2.968011
c -0.170697 -0.487031  3.018604  2.858124
d -0.878197  0.796428 -0.479922  0.387211
```

Pandas Dataframe

- How to select rows
  - Use the .loc operation

```
example.loc[['a','c']]
```

```
          x         y         z     summa
a -0.292712 -0.456712  0.478160  0.696751
c -0.170697 -0.487031  3.018604  2.858124
```

  - Use the .iloc operation

Pandas Dataframe

- Just as for numpy arrays, can use multi-indices

```
example.loc[['a','c'],['x','y']]
```

```
          x         y
a -0.292712 -0.456712
c -0.170697 -0.487031
```

Pandas Dataframe

- Just as in numpy, we can create boolean selections

```
boolex = example > 1
```

```
       x      y      z  summa
a  False  False  False  False
b  False   True  False   True
c  False  False   True   True
d  False  False  False  False
```

Pandas Dataframe

- And use the boolean selection to select values from the frame

```
example[boolex]

    x           y           z      summa
a NaN         NaN         NaN        NaN
b NaN    1.466134         NaN   2.968011
c NaN         NaN    3.018604   2.858124
d NaN         NaN         NaN        NaN
```

Pandas Dataframe

- Or do so in a single step

```
example[example>1]
```

	x	y	z	summa
a	NaN	NaN	NaN	NaN
b	NaN	1.466134	NaN	2.968011
c	NaN	NaN	3.018604	2.858124
d	NaN	NaN	NaN	NaN

- Notice that the numbers not fitting are NaNs

Pandas Dataframe

- A more typical selection uses a column
- Example: The example dataframe

```
          x         y         z     summa
a -0.292712 -0.456712  0.478160  0.696751
b  0.801120  1.466134  0.883498  2.968011
c -0.170697 -0.487031  3.018604  2.858124
d -0.878197  0.796428 -0.479922  0.387211
```

- Select the rows where the 'z' value is positive:

```
example[example['z']>0]
```

```
          x         y         z     summa
a -0.292712 -0.456712  0.478160  0.696751
b  0.801120  1.466134  0.883498  2.968011
c -0.170697 -0.487031  3.018604  2.858124
```

Pandas Dataframe

- Compound Conditions
  - We can combine conditions for selection
    - Unlike classical Python, we **cannot** use and / or
    - Need to use single ampersand or vertical bar & | for and, or

Pandas Dataframe

- Example:
  - Create a random frame

```
example = pd.DataFrame(np.random.randn(4,3),
['a','b','c','d'],['x','y','z'])
```

```
print(example)
```

```
          x         y         z
a  1.411543  2.160431 -1.891248
b -1.062715 -0.831573  0.440250
c -1.157673 -0.963104  1.817167
d -0.162145  0.140711 -0.016717
```

Pandas Dataframe

- Select the rows where 'x' is negative and 'z' is positive

```
          x         y         z
a  1.411543  2.160431 -1.891248
b -1.062715 -0.831573  0.440250
c -1.157673 -0.963104  1.817167
d -0.162145  0.140711 -0.016717
```

- These are rows b and c

Pandas Dataframe

- Use the ampersand
  - Parentheses are necessary

```
new_frame = example[(example['x']<0) & (example['z']>0)]
print(new_frame)
```

```
          x         y         z
b -1.062715 -0.831573  0.440250
c -1.157673 -0.963104  1.817167
```

Pandas Dataframe

- We can make our own Boolean conditions
    - We do so by using the dataframe .apply method that applies a function along an axis of the data frame
        - axis=0: index
            - Applies function to each column
            - default
        - axis=1: columns
            - Applies function to each row

Pandas Dataframe

- Example:
  - Find all entries that have US somewhere in the 'company_name' of a dataframe df

```
return df[df[company_name].apply(
    lambda x: 'US' in x)]
```

- Here the function returns a Boolean value

Pandas Dataframe

- We can change the index
  - Example: Import the following .csv file

```
First Name, Last Name, Position, Year
Salmon, Chase, CJUS, 1865
Salmon, Chase, CJUS, 1866
Salmon, Chase, CJUS, 1867
Salmon, Chase, CJUS, 1868
Salmon, Chase, CJUS, 1869
Ulysses, Grant, PotUS, 1869
Ulysses, Grant, GUSA, 1866
Ulysses, Grant, GUSA, 1867
Ulysses, Grant, GUSA, 1868
Andrew, Johnson, PotUS, 1865
Andrew, Johnson, PotUS, 1866
Andrew, Johnson, PotUS, 1867
Andrew, Johnson, PotUS, 1868
Andrew, Johnson, PotUS, 1869
William, Sherman, GUSA, 1869
```

Pandas Dataframe

- Import using read_csv

```
ex = pd.read_csv('example.csv')
```

```
   First Name  Last Name  Position   Year
0       Salmon      Chase      CJUS   1865
1       Salmon      Chase      CJUS   1866
2       Salmon      Chase      CJUS   1867
```

Pandas Dataframe

- Create a new column
  - Needs to have the same number of elements as there are data rows:

```
ex['entry']=['a','b','c','d','e','f','g','h','i',
             'j','k','l','m','n','o']
```

	First Name	Last Name	Position	Year	entry
0	Salmon	Chase	CJUS	1865	a
1	Salmon	Chase	CJUS	1866	b
2	Salmon	Chase	CJUS	1867	c
3	Salmon	Chase	CJUS	1868	d

Pandas Dataframe

- Now set the index to this new column

```
ex.set_index('entry')
```

```
      First Name  Last Name  Position   Year
entry
a         Salmon      Chase      CJUS   1865
b         Salmon      Chase      CJUS   1866
c         Salmon      Chase      CJUS   1867
d         Salmon      Chase      CJUS   1868
```

- add parameter inplace = True if these changes are intended to be permanent