



Data Series and Frames

LECTURE 6

Organization of Lecture 6

- Getting Used to Pandas Data Structures
- Reshaping Data
- Handling Missing Data
- Combining Data
- Ordering and Describing Data
- Transforming Data
- Taming Pandas File I/O

Working with Data Series and Frames

- Data scientists *love* tabular data
 - Nice shape and convenient access to elements, rows and columns
 - Supercomputers can perform vectorized operations (broadcasting)
- Module numpy fails to tie numeric data back to data attributes (e.g., the column and row names and indexes)
- Module pandas to the rescue
 - Adds to Python the *abstractions of data series and frames* (core of R)
 - Built on top of numpy (extends and partially re-implements numpy)

pandas data frame

- “smart” spreadsheet
- Labeled table with columns (variables), observations (rows)
- Built-in operations
- Cells in data part actually implemented in numpy
- Operations: reshaping, aggregation and ufuncs
- Access to row and column labels
 - Allows combining frames by merging/concatenating (think join in relational databases)
- Integrates with pyplot – plotting data visualization system

pandas data structures

- Two containers
 - Series – 1D, labeled vector
 - DataFrame – table with labeled rows and columns
- Provide support for
 - Single-level and hierarchical indexing
 - Handling missing data
 - Arithmetic and Boolean operations on entire columns and tables
 - Database type operations (merging and aggregation)
 - Plotting individual columns and whole tables
 - Reading data from files and writing data to files

Series 1		Series 2		Series 3		DataFrame			
	Mango		Apple		Banana		Mango	Apple	Banana
0	4	0	5	0	2	0	4	5	2
1	5	1	4	1	3	1	5	4	3
2	6	2	3	2	5	2	6	3	5
3	3	3	0	3	2	3	3	0	2
4	1	4	2	4	7	4	1	2	7

Series

- 1D homogeneous vectors
- Can create from list, tuple or array
- Example: pass a n-tuple to Series constructor (inflation for many years in a row)

```
import pandas as pd
inflation = pd.Series((2.2, 3.4, 2.8, 1.6, 2.3, 2.7, 3.4, 3.2, 2.8, 3.8, \
-0.4, 1.6, 3.2, 2.1, 1.5, 1.5))
print (inflation)
print (len(inflation))
```

- Series has default integer index attribute, starting at 0 `inflation.index.values`
- The values attribute has the values in a numpy array, `inflation.values`
- These arrays are mutable, can change their values, e.g.,
`inflation.values[-1] = 1.6` will change the last value in the series.

Console	Shell
0	2.2
1	3.4
2	2.8
3	1.6
4	2.3
5	2.7
6	3.4
7	3.2
8	2.8
9	3.8
10	-0.4
11	1.6
12	3.2
13	2.1
14	1.5
15	1.5
dtype: float64	
16	

Series with customized indices

- Series looks and act like arrays
 - If you want to associate inflation values with a year array, have to keep the series together (think parallel arrays from CS1)
 - Better to have something like an array of objects ... or a *dictionary*?

```
import numpy as np
import pandas as pd
inflation = pd.Series({1999:2.2, 2014: 1.6, 2015: np.nan})
print (inflation)
```

- This is a series with a customized index(based on keys)

Console	Shell
1999	2.2
2014	1.6
2015	NaN
dtype: float64	
>	

Naming series and indices

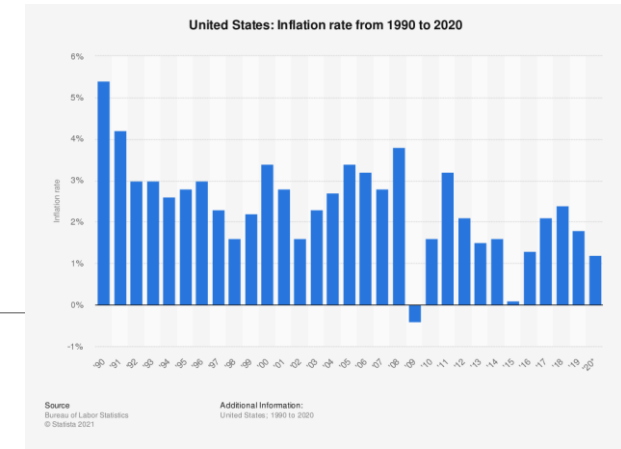
```
import numpy as np
import pandas as pd
inflation = pd.Series({1999:2.2, 2014: 1.6, 2015: np.nan})
inflation.index.name = "Year"
inflation.name = "Inflation Data"
print (inflation)
```

- `inflation.head()` returns first five rows of a series
- `inflation.tail()` returns last five rows of a series

```
Year
1999    2.2
2014    1.6
2015    NaN
Name: Inflation Data, dtype: float64
> []
```


Frames

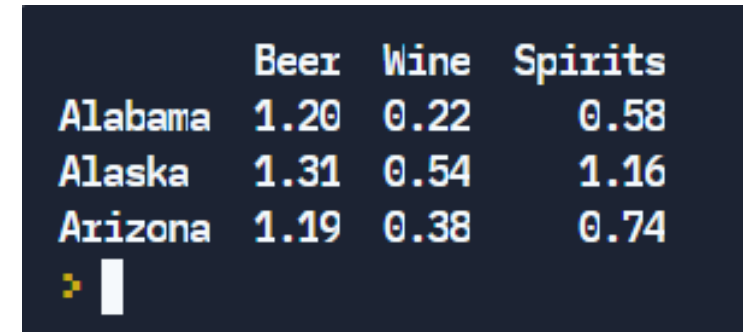
- Table with labeled rows and columns
- Construct from 2D numpy array, list of tuples, dictionary or frame
 - Dictionary case: keys serve as column names and values are column values.
 - Array case: you can supply the column names through optional parameter columns
 - Frame case: copies column names from source to new frame
- Frame index is called axis 0 (vertical) and frame columns are called axis 1 (horizontal)



Frames Example

- Data from Natl Inst on Alcohol Abuse and Alcoholism 2019
 - Shows per capita alcohol consumption per state, category (beer, wine, spirits) and per year between 1970-2019.
 - Can create a simple frame with column names and an index by passing a list of row tuples or other s sequences of the same length to the constructors.

```
alco2009 = pd.DataFrame([(1.20, 0.22, 0.58), \
(1.31, 0.54, 1.16), \
(1.19, 0.38, 0.74)], \
columns = ("Beer", "Wine", "Spirits"), \
index = ("Alabama", "Alaska", "Arizona"))
```

A terminal window with a dark background and light-colored text. It displays a pandas DataFrame with three columns: 'Beer', 'Wine', and 'Spirits'. The rows are indexed by state names: 'Alabama', 'Alaska', and 'Arizona'. The values are: Alabama (1.20, 0.22, 0.58), Alaska (1.31, 0.54, 1.16), and Arizona (1.19, 0.38, 0.74). A cursor is visible at the bottom left.

	Beer	Wine	Spirits
Alabama	1.20	0.22	0.58
Alaska	1.31	0.54	1.16
Arizona	1.19	0.38	0.74

- Can also use a dictionary of columns {“Beer”: (1.20, 1.31, 1.19, ...), ...
index = ("Alabama", "Alaska", "Arizona")

Accessing Frame Columns

- `frame[column heading].head()` returns tuple with 5 values in column at beginning of column
 - `Alco2009["Wine"].head()`
- `frame[column heading].tail()` returns tuple with 5 values in column at end of column
 - `Alco2009["Wine"].tail()`

```
Head is
Alabama    0.22
Alaska     0.54
Arizona    0.38
Name: Wine, dtype: float64
```

```
Tail is
Alabama    0.22
Alaska     0.54
Arizona    0.38
Name: Wine, dtype: float64
```

```
>
```

Broadcasting Frames

- Can assign value to all rows of a column in 1 assignment statement.
- Column doesn't exist, panda creates it

```
alco2009["Total"] = 0  
print (alco2009.head())
```

	Beer	Wine	Spirits	Total
Alabama	1.20	0.22	0.58	0
Alaska	1.31	0.54	1.16	0
Arizona	1.19	0.38	0.74	0

```
> 
```

Reshaping Data

- Main contribution of pandas is data labeling – association of labels with columns and rows
- Association is flexible – just like numpy arrays can be reshaped, pandas frames' data labels can be reorganized.
- Example: if a hierarchical index of one frame has two levels (“Year” and “State”), but another frame has a flat “State” index, you can convert the “Year” labels into *column names*.

Indexing

- Frame index
 - collection of labels assigned to frame rows
 - Have to be same type
 - Do not have to be unique
 - Optional parameter to DataFrame() constructor
- Can access and change column names (frame.column.values) and the index (frame.index.values)
- Any column in frame can become an index

```
column values: ['Beer' 'Wine' 'Spirits']  
index values: ['Alabama' 'Alaska' 'Arizona']
```

Indexing cont.

- Any column in frame can become an index
 - Function `reset_index()` – deposits existing index, if any
 - Function `set_index()` – declares a new index
 - Both return a new frame, unless optional parameter `inplace=True` is supplied.

```
alco2009.index.names=["State"] # just added an index name
print ("Original: \n ", alco2009)
```

```
#added State index heading #old index added as column/new sequential index is used instead
print ("\nIndex reset: \n ", alco2009.reset_index())
```

```
#drops sequential index and replaces with Beer index
print ("\nBeer set as index: \n ", alco2009.reset_index().set_index("Beer"))
```

Output for Indexing Example

Original:

	Beer	Wine	Spirits
State			
Alabama	1.20	0.22	0.58
Alaska	1.31	0.54	1.16
Arizona	1.19	0.38	0.74

Index reset:

	State	Beer	Wine	Spirits
0	Alabama	1.20	0.22	0.58
1	Alaska	1.31	0.54	1.16
2	Arizona	1.19	0.38	0.74

Beer set as index:

	State	Wine	Spirits
Beer			
1.20	Alabama	0.22	0.58
1.31	Alaska	0.54	1.16
1.19	Arizona	0.38	0.74

>

Reindexing

- Create new frame or series from existing frame or series
 - Can permute rows, columns or both
 - Counterpart to numpy “smart” indexing, *except*
 - if row or column not found, creates it (them) and populates with Nans
- Example if you were only interested in states whose names begin with ‘S’

```
s_states = [state for state in alco2009.index if state[0] == 'S']  
drinks = list(alco2009.columns) + ["Water"]  
nan_alcoS = alco2009.reindex(s_states, columns=drinks)
```

Hiearchical Indexing

- Multilevel indexing/multilevel column names
- Consists of 3 lists: level names, labels per level, list of actual values
- Example below has multiindex by State and Year (sorted by state and then year)

```
alco = pd.DataFrame([(1.20, 0.22, 0.58), \
                     (1.31, 0.54, 1.16), \
                     (1.19, 0.38, 0.74), \
                     (1.20, 0.22, 0.58), \
                     (1.31, 0.54, 1.16), \
                     (1.19, 0.38, 0.74), \
                     (1.20, 0.22, 0.58), \
                     (1.31, 0.54, 1.16), \
                     (1.19, 0.38, 0.74)], \
                    columns = ("Beer", "Wine", "Spirits"), \
                    index = multi)
alco.index.names=["State", "Year"]
print("\n", alco)
```

		Beer	Wine	Spirits
State	Year			
Alabama	1977	1.20	0.22	0.58
	1978	1.31	0.54	1.16
	1979	1.19	0.38	0.74
Alaska	1977	1.20	0.22	0.58
	1978	1.31	0.54	1.16
	1979	1.19	0.38	0.74
Arizona	1977	1.20	0.22	0.58
	1978	1.31	0.54	1.16
	1979	1.19	0.38	0.74

Hierarchical Indexing

- Multiindex can be used same was as a flat index.
- Partial selection produces a frame, a complete selection a series.

```
#Partial Selection
print ("\n", alco.loc['Arizona'].head())

#complete Selection
print ("\n", alco.loc['Arizona', 1979].head())
```

```
      Beer  Wine  Spirits
Year
1977  1.20  0.22   0.58
1978  1.31  0.54   1.16
1979  1.19  0.38   0.74

      Beer      1.19
      Wine      0.38
      Spirits  0.74
Name: (Arizona, 1979), dtype: float64
>
```

Stacking and Pivoting

- `stack()` function
 - increments the number of levels in the index (hierarchical indexing)
 - Makes frame taller and more narrow
- `unstack()` function
 - flattens (decrements number of levels and simultaneously increments number of levels in column names)
 - Makes frame shorter and wider
- `pivot(index, columns, values)` function
 - Converts a frame into another frame: column is new index, columns is new list of column names, and values is the data

Example

```
tall_alco.stack()  
tall_alco.index.names = ["State", "Year", "Drink"]  
print (alco.stack())
```

```
wide_alco = alco.unstack()  
print (wide_alco, "\n")
```

	Beer			Wine			Spirits		
Year	1977	1978	1979	1977	1978	1979	1977	1978	1979
State									
Alabama	1.20	1.31	1.19	0.22	0.54	0.38	0.58	1.16	0.74
Alaska	1.25	1.31	1.17	1.22	0.54	1.38	0.32	1.16	1.74
Arizona	1.20	1.31	1.19	0.22	0.54	0.38	0.58	1.16	0.74

State	Year	Drink	
Alabama	1977	Beer	1.20
		Wine	0.22
		Spirits	0.58
	1978	Beer	1.31
		Wine	0.54
		Spirits	1.16
	1979	Beer	1.19
		Wine	0.38
		Spirits	0.74
Alaska	1977	Beer	1.25
		Wine	1.22
		Spirits	0.32
	1978	Beer	1.31
		Wine	0.54
		Spirits	1.16
	1979	Beer	1.17
		Wine	1.38
		Spirits	1.74
Arizona	1977	Beer	1.20
		Wine	0.22
		Spirits	0.58
	1978	Beer	1.31
		Wine	0.54
		Spirits	1.16
	1979	Beer	1.19
		Wine	0.38
		Spirits	0.74

dtype: float64

Imperfect Data

- Incorrect data, outliers, not in order, missing, etc.
- pandas uses `numpy.nan`
- Must take care of imperfect data
 - Can clean up files first (e.g., open .xls files)
 - Steps
 - Delete missing data (all nan values, columns with any nan values, rows with any nan values)
 - Can input missing data
 - replace them with “clean” values, e.g. 0’s, 1’s, with an average, “undecided”, etc.

Imperfect Data

- Have panda take care of missing data with built-in functions
 - `frame.dropna(how="all")` or `frame.dropna(how="all", axis=1)`
 - `Frame.isnull()` or `frame.notnull()`
 - `isnull()` returns True if value is a nan, False otw
- Have panda fill in missing data with built-in functions
 - `frame.fillna()`
 - `frame.replace(val_or_list, new_val)` #if new_val is a list, must be same size
 - Both return a new frame, unless you specify `inplace = True`