INSY 6500
Information Systems for Operations

Introduction to Pandas

Fall 2019

Jeffrey S. Smith

Industrial and Systems Engineering Department
Auburn University
jsmith@auburn.edu

What is Pandas?

Pandas is a (newer) package built on top of NumPy, and provides an efficient implementation of a DataFrame. DataFrames are essentially multidimensional arrays with attached row and column labels, and often with heterogeneous types and/or missing data. As well as offering a convenient storage interface for labeled data, Pandas implements a number of powerful data operations familiar to users of both database frameworks and spreadsheet programs.

VanderPlas, Notebook 03.00

Our Pandas-related Resources

- Course Jupyter Notebook
  - 3.5.0 – Introduction to Pandas.ipynb
  - 3.5.1 – Concatenating, Appending, Joining Dataframes.ipynb
  - 3.5.2 – Pandas Time Series Modeling.ipynb
  - Pandas – Some Sample Datasets.ipynb
- VanderPlas Jupyter Notebooks
  - GitHub address: https://github.com/jakevdp/PythonDataScienceHandbook
  - Notebooks 03.00 – 03.13
- Pandas web page: https://pandas.pydata.org/
  - Documentation: http://pandas.pydata.org/pandas-docs/stable/

Fundamental Pandas Data Structures - Series

- A "one-dimensional array of indexed data."

```
data = pd.Series([0.25, 0.5, 0.75, 1.0])
```

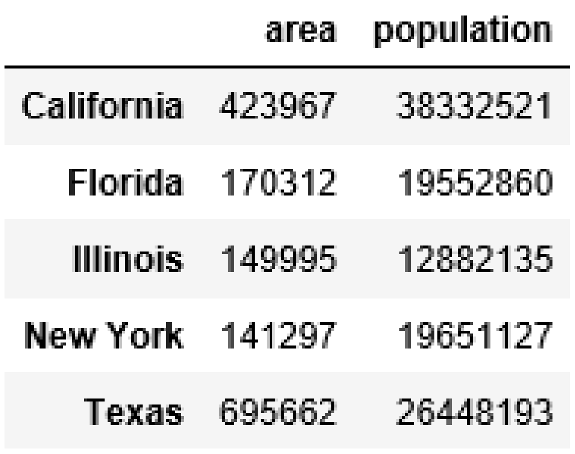
`data.values`	Returns a NumPy array containing the data values.
`data.index`	Returns a *Pandas Index object* (discussed soon) containing the index values.

- Examples in the VP 03.01 notebook and our general Pandas notebook (3.5.0)
- Series as either a *generalized NumPy array* or a *generalized Python Dictionary*

VanderPlas, Notebook 03.01

Fundamental Pandas Data Structures – DataFrame

- “If a Series is an analog of a one-dimensional array with flexible indices, a DataFrame is an analog of a two-dimensional array with both flexible row indices and flexible column names. Just as you might think of a two-dimensional array as an ordered sequence of aligned one-dimensional columns, you can think of a DataFrame as a sequence of aligned Series objects. Here, by ‘aligned’ we mean that they share the same index.”



	area	population
California	423967	38332521
Florida	170312	19552860
Illinois	149995	12882135
New York	141297	19651127
Texas	695662	26448193

VanderPlas, Notebook 03.01

Fundamental Pandas Data Structures – Index

- “Immutable ndarray implementing an ordered, sliceable set. The basic object storing axis labels for all pandas objects”
  - [https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Index.html]
- “This Index object is an interesting structure in itself, and it can be thought of either as an immutable array or as an ordered set (technically a multi-set, as Index objects may contain repeated values). Those views have some interesting consequences in the operations available on Index objects. ” [VanderPlas, 03.01].
- “Pandas objects are designed to facilitate operations such as joins across datasets, which depend on many aspects of set arithmetic. The Index object follows many of the conventions used by Python's built-in set data structure, so that unions, intersections, differences, and other combinations can be computed in a familiar way.” [VanderPlas, 03.01]

Handling Missing Data

- General methods to handle missing data:
  1. Masking – use a Boolean Mask to *mask* the missing values
  2. Sentinel Value – Use a special value to denote missing values – e.g, *NaN*
- Pandas uses the sentinel value approach with the floating point NaN and the Python None object.

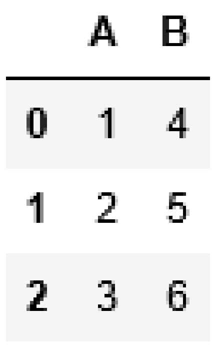
Detecting:

- isna()
- isnull()
- notna()
- notnull()

Changing:

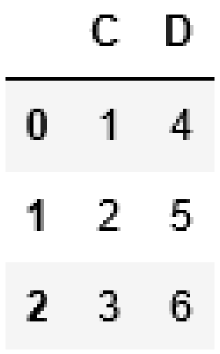
- dropna()
- fillna()

Concatenating, Appending, Joining DataFrames



	A	B
0	1	4
1	2	5
2	3	6

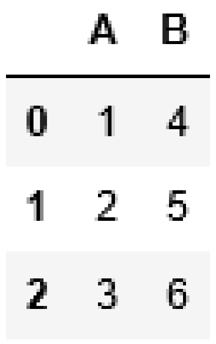
+



	C	D
0	1	4
1	2	5
2	3	6

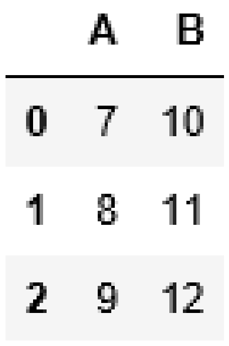
= ?

Concatenating, Appending, Joining DataFrames



	A	B
0	1	4
1	2	5
2	3	6

+

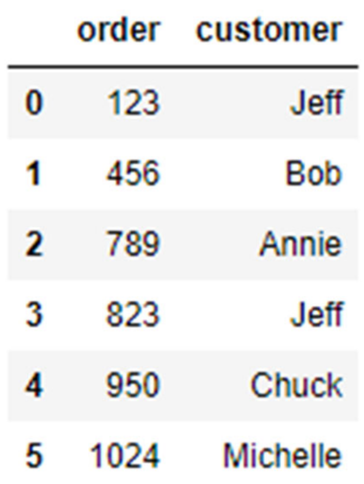


	A	B
0	7	10
1	8	11
2	9	12

= ?

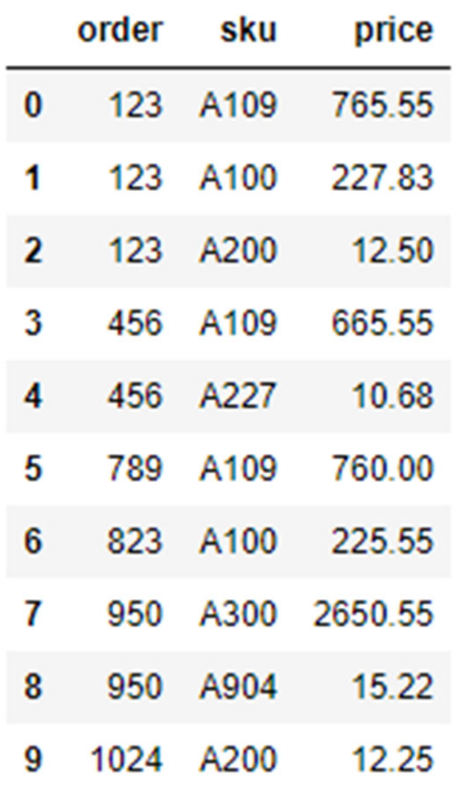
Merge (Join)

Orders:



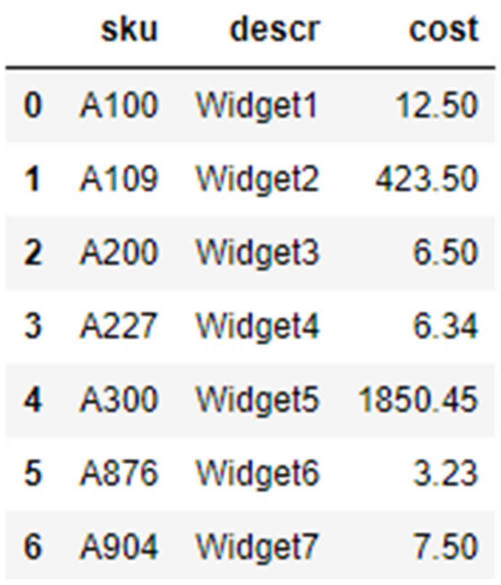
	order	customer
0	123	Jeff
1	456	Bob
2	789	Annie
3	823	Jeff
4	950	Chuck
5	1024	Michelle

Items:



	order	sku	price
0	123	A109	765.55
1	123	A100	227.83
2	123	A200	12.50
3	456	A109	665.55
4	456	A227	10.68
5	789	A109	760.00
6	823	A100	225.55
7	950	A300	2650.55
8	950	A904	15.22
9	1024	A200	12.25

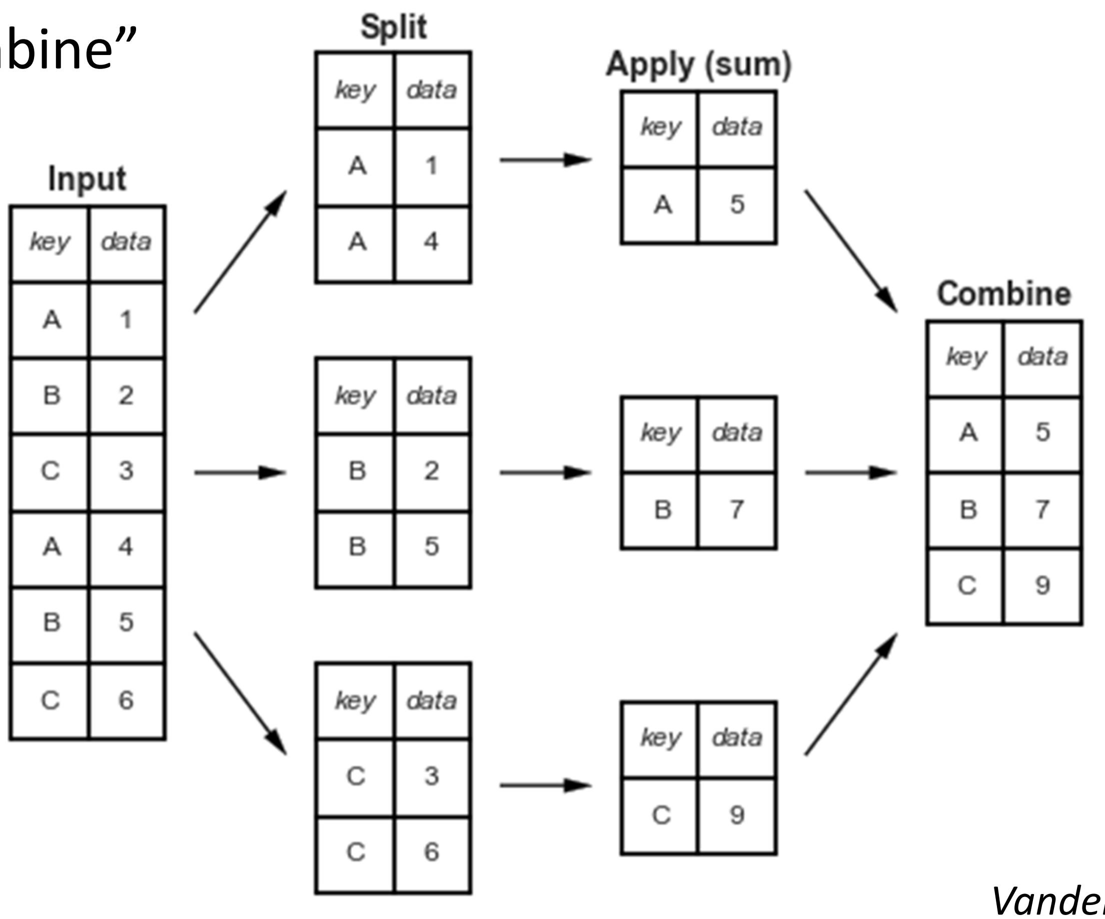
SKUS:



	sku	descr	cost
0	A100	Widget1	12.50
1	A109	Widget2	423.50
2	A200	Widget3	6.50
3	A227	Widget4	6.34
4	A300	Widget5	1850.45
5	A876	Widget6	3.23
6	A904	Widget7	7.50

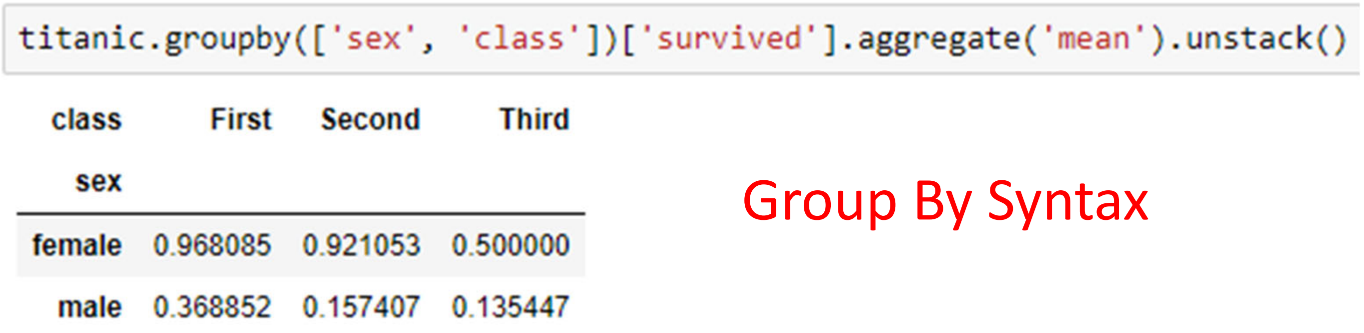
Group By: Aggregation and Grouping in Pandas

"Split, Apply, Combine" Method



VanderPlas, Notebook 03.08

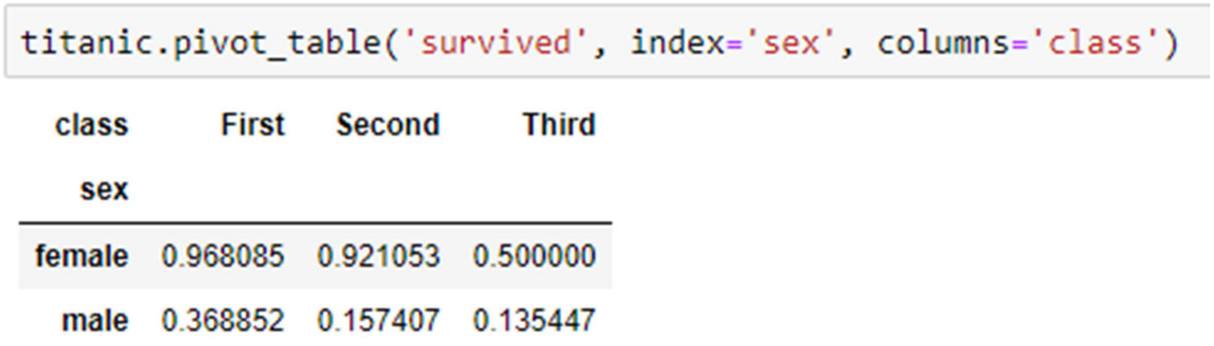
Pivot Tables



```python
titanic.groupby(['sex', 'class'])['survived'].aggregate('mean').unstack()
```

class	First	Second	Third
sex			
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447

Group By Syntax



```python
titanic.pivot_table('survived', index='sex', columns='class')
```

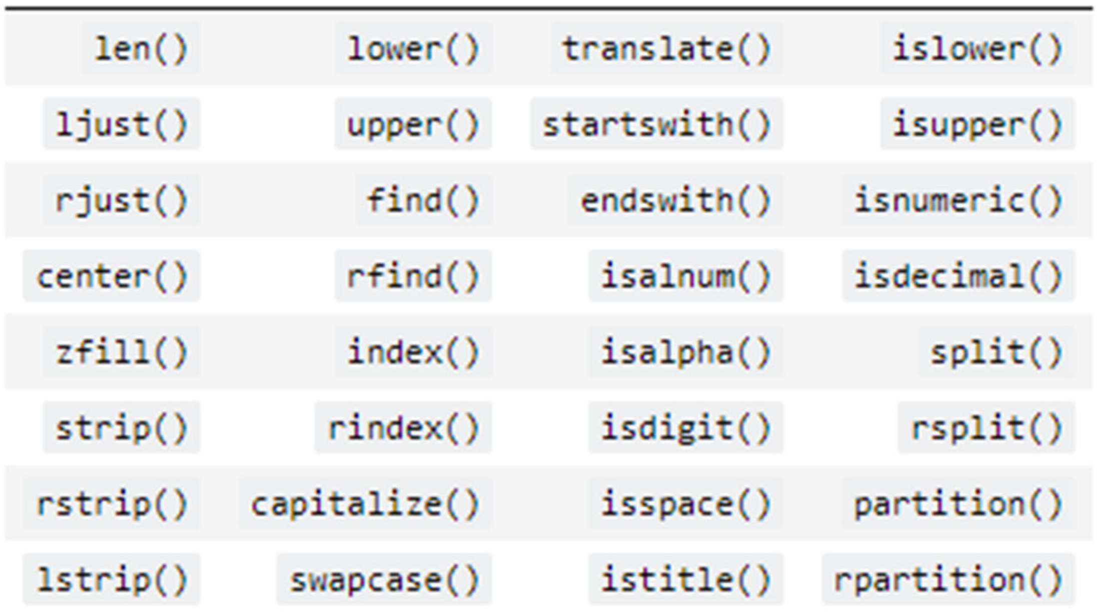
class	First	Second	Third
sex			
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447

Pivot Table Syntax

VanderPlas, Notebook 03.09

Vectorized String Operations in Pandas

Nearly all Python's built-in string methods are mirrored by a Pandas vectorized string method. Here is a list of Pandas `str` methods that mirror Python string methods:



`len()`	`lower()`	`translate()`	`islower()`
`ljust()`	`upper()`	`startswith()`	`isupper()`
`rjust()`	`find()`	`endswith()`	`isnumeric()`
`center()`	`rfind()`	`isalnum()`	`isdecimal()`
`zfill()`	`index()`	`isalpha()`	`split()`
`strip()`	`rindex()`	`isdigit()`	`rsplit()`
`rstrip()`	`capitalize()`	`isspace()`	`partition()`
`lstrip()`	`swapcase()`	`istitle()`	`rpartition()`

VanderPlas, Notebook 03.10

Dates, Times, and Time-series Modeling

- Resources
  - 03.11 notebook from VanderPlas, 3.3.0 and 3.5.2 notebooks from our repository
- Handling dates and times is a complex but important topic. We will just barely scratch the surface of the general topic ....
  - Python datetime module
  - datetime objects
  - strftime()
  - strptime()
  - timedelta()
  - Numpy's datetime object – datetime64