Introduction to Pandas

Continuum Analytics

Pandas

- Data analysis library
- Panel Data Structures
- Written by Wes McKinney

Key Features

- Indexed Arrays
- Data alignment
- Date Time handling
- Resampling/frequency conversion
- Statistical modeling
- Group by, joining, merging, pivoting

CONTINUUM
ANALYTICS

Added Goodies

- Moving window statistics
- Easy and fast data access (hdf5, csv, sql)
- Integration with Matplotlib

CONTINUUM
ANALYTICS

Core Concepts

- `Series` and `DataFrame` are the main structures
- Operations on `Series` and `DataFrame` generally produce more `Series` and `DataFrame`
- No matter where you are in your analysis, you always have your full arsenal at your disposal
- Library is designed around common data analysis tasks

CONTINUUM
ANALYTICS

Recalling NumPy...

- Indexing - `a[2,3]`, selecting one value
- Slicing - `a[2:10]`, using slicing notation to select many values
- Fancy Indexing - `a[[2,3,4,5], 3]` or `a[[True, True], 3]`

CONTINUUM ANALYTICS

Pandas Series

- like a NumPy array but with an index

```
>>> index = ['a','b','c','d','e']

>>> series = pandas.Series(np.arange(5), index=index)
a    0
b    1
c    2
d    3
e    4
```

CONTINUUM ANALYTICS

Pandas Series

- Refer to content by named index or by range

```
>>> series['a']
0

>>> series['b']
1

>>> series['c']
2

>>> series[2:4]
c    2
d    3

>>> series * 2
a    0
b    2
c    4
d    6
e    8
```

CONTINUUM ANALYTICS

Pandas Series

- `Series` extends NumPy array
- If you don't pass in an index, one is created for you (equivalent of range(N), where N is the length of your data)
- Index used to implement fast lookups, data alignment and join operations
- *Try to avoid integer index names*

Series Construction

- Can be constructed with an array like object, or with a dict
  - With a dict, the keys are *sorted* and used as the index
  - With an array-like object, you can pass in another array-like object as the index

Series Indexing

- Indexing looks up value using the index (row label)
  - `myseries[0]` or `myseries['a']`
- Slicing with integers defaults to ignoring the index
  - `myseries[2:4]`
- Slicing with non-integers uses the index, and is inclusive
  - `myseries['a':'c']`
- Order matters
  - `myseries['a':'c']` is different from `myseries['c':'a']`
- *Try to avoid integer index names*

CONTINUUM ANALYTICS

Series Operations

- You can do math using series
  - When index values are different, default to an outer join

```
>>> myseries
a    1
b    2
d    4
e    5

>>> myseries2
a      1
b      2
d      4
f     10

>>> myseries + myseries2
a      2
b      4
d      8
e    NaN
f    NaN
```

CONTINUUM ANALYTICS

Pandas DataFrames

- `DataFrame` is a collection of Pandas `Series`
  - joined on index: DateTime, AlphaNumerical Index, etc

- This index is also referred to as a row label

- Pandas `DataFrame` objects have column names:
  - accessed attribute style: `prices.Close`
  - dictionary style: `prices['Adj Close']`

CONTINUUM ANALYTICS

DataFrame Example

```
>>> rawdata = {'a': np.random.random(5), 'b': np.rando
m.random(5)}

>>> data = pandas.DataFrame(rawdata)
>>> data
          a         b
0  0.266826  0.602288
1  0.338174  0.294303
2  0.019489  0.473737
3  0.876180  0.518681
4  0.901697  0.370186

>>> data[1:3]
          a         b
1  0.338174  0.294303
2  0.019489  0.473737

>>> data[1:3].a
1    0.338174
2    0.019489
Name: a
```

CONTINUUM ANALYTICS

DataFrame Indexing

- `DataFrame` is dict-like in referring to column names
- Using a list of column names selects that list of columns
- Similar to `Series`, mathematical operations on `DataFrame` objects default to outer-joins
  - joins occur both row-wise, and column-wise
- `df.ix` for NumPy like indexing semantics
- `df.xs` for cross-section along a row

CONTINUUM ANALYTICS

DataFrame Updates

- Adding New Columns:
  - zero fill: `df['var'] = 0`
  - values from NumPy array: `df['my_data'] = data`
  - note: `df.var` construct can not create a column by that name; only used to access existing columns by name

- Deleting Columns:
  - `df.drop(['var','new_data'], axis=1)`

CONTINUUM ANALYTICS

DataFrame Construction

- Dict of array like objects -- keys are column names
- Nested dict of values
- CSV
  - Excel Files (requires `xlrd`)
- HDF5
- SQL

DataFrame Extraction

- Important to get data out of `DataFrame`
- `df.values` returns underlying NumPy array
- to_method
  - `df.to_csv`
  - `df.to_excel`
  - `df.to_html`
  - `df.to_latex`
  - ...

CONTINUUM ANALYTICS

Understanding Advanced Pandas

The Index

- `set_index` versus `reindex`
  - `set_index` replaces the index with some new values, you can refer to them by column name, or pass an array
  - `reindex` subselects from the `DataFrame`, padding any values that are necessary with `NA`

DateTime Indexing and Resampling

- Built-in logic for standard time chunks
  - microsecond, millisecond, minute, hour, etc.
  - `df.index.day`, `df.index.dayofweek`
- Resampling for non-standard time chunks (up- or down-sampling)
  - `df.resample(time,fill_method)`
- Can build index for any time chunk

```
df.resample('1min', fill_method='pad')
min35 = pandas.dateoffset(minutes=35)
df[datetime(2010,10,10,0,0,0) + min35]
```

CONTINUUM ANALYTICS

Split—Apply—Combine

- Split the data into chunks
- Apply some transformation or aggregation onto the chunks
- Pull the computed values back into a data structure
- Repeat again and again

Split—Apply—Combine (Cont...)

- Often end up with very simple code for each step
- Easy spelling easy debugging
- Can be tricky to get used to
- Similar mental process to understanding vectorized computing

CONTINUUM
ANALYTICS

Split with Group By

- Pass a column name
  - OR a List of column names
  - OR a function, which returns unique values

CONTINUUM
ANALYTICS

Group By Object

- `groups` attribute
- Dictionary with keys as the group names, and values the indexes which go in them
- `get_groups` call to retrieve each underlying group
- `agg`, `transform`, `apply` functions

```
    grouped = df.groupby(key)
    grouped = df.groupby([key1, key2])
```

CONTINUUM ANALYTICS

Apply and Combine

- Different options for the apply and combine steps
  - `agg`
  - `transform`
  - `apply`
- Each works slightly differently

CONTINUUM ANALYTICS

Aggregation

- Functions should take a `Series` and return a single aggregated value

- Results are combined into a `DataFrame`, where the index values are the group names

Aggregation on a Grouped DataFrame

- `grouped.agg(function)`
- Function is applied to each column
- If a list of functions is passed in, each function is called on each column
- Can also pass a dictionary of column names, where the values are functions or list of functions

```
grouped.agg(np.sum)
grouped.agg([np.sum, np.mean])
```

CONTINUUM ANALYTICS

Aggregation on a Grouped Series

- `grouped[column].agg(function)`
- Can pass a single function, in which case a series is returned, values are the output of the agg function, index is the group names
- Can pass a list of functions, in which case the list is fed on the group, and a `DataFrame` is created
- Again, the function expects a `Series`, and returns a number

CONTINUUM ANALYTICS

Apply

- Can be used to implement `transform` as well as `agg`
- Complete Reduction (single value/row)
  - Your index will be the group names
  - If you return one number, you will get a `Series`
  - If you return a `Series`, the index will be column names and each `Series` will be come a `DataFrame` row
- Same Index
  - If you return a `DataFrame` with the exact same index, that index is used

CONTINUUM ANALYTICS

Apply Example

- Apply used for non-aggregate function on groupby object

```
    def top10(group):
        return f.sort_index(by=key, ascending=False)[:10]

    grouped.apply(top10)
```

CONTINUUM ANALYTICS

Transformation

- `grouped.transform(function)`
  - Returns something like the original `DataFrame`
  - Cannot add any rows or columns
  - Cannot rename columns
- Compare with `apply` and `agg`
  - Promise of faster performance than `apply` (not always)
  - `agg` offers special use case with better performance than `apply`
  - `transform` offers special use case with no reduction

CONTINUUM ANALYTICS