Python Program

Chapter 3: Getting Started with Pandas

Chapter Objectives

In this chapter, we will introduce:

- Pandas data structures
- Essential functionality of pandas
- Reading from data sources
- Summarizing and computing descriptive statistics
- Handling missing data

Chapter Concepts

Introduction to Pandas Data

Essential Functionality

Reading From Data Sources

Summarizing and Computing Descriptive Statistics

Handling Missing Data

Chapter Summary

Introducing Pandas

- *Pandas* is an open-source, BSD-licensed library
  - Provides high-performance, easy-to-use data structures and data analysis tools

- Common usages
  - `import pandas as pd`
  - `from pandas import DataFrame`
  - `from pandas import Series`
  - `from pandas import DataFrame, Series`

Pandas `Series` Data Structure

- Pandas provides the `Series` data structure
  - A one-dimensional array-like object
- Contains:
  - Array of data
  - Array of data labels known as the index
- `Series` object has `values` and `index` properties
- Can be thought of as a fixed-length dictionary

`Series` Example

```
from pandas import Series
data = Series([1,2,3,4])
print(data)
0     1
1     2
2     3
3     4
dtype: int64

print(data.values)
array([1, 2, 3, 4])

print(data.index)
RangeIndex(start=0, stop=4, step=1)

print(data[1])
2
```

Each data value is assigned an index from `N` through to `N-1`

`Series` Index

- It is possible to create a `Series` with a user-defined index for each data point

```
data = Series([1,2,3,4], index=['a','b','c','d'])
print(data.index)
Index([u'a', u'b', u'c', u'd'], dtype='object')

print(data['c'])
3
```

Index can be used to access data points

`Series` and Dictionaries

- A `Series` can be created by passing a dictionary
  - Dictionary keys are used for `Series` index by default
    - Separate keys can be provided

```
cities = {'Dublin': 200000, 'Athlone': 15000, 'Galway': 700000}
series1 = Series(cities)
print (series1)
Athlone      15000
Dublin      200000
Galway      700000
dtype: int64
```

`Series` and Dictionaries (continued)

- If no key is provided, then `NaN` is used
  - Can use `isnull()` and `notnull()` functions to detect missing data

```
cities = {'Dublin': 200000, 'Athlone': 15000, 'Galway':
700000}

indexes = ['Dublin', 'Athlone', 'Waterford']

series2 = Series(cities, index=indexes)

print (series2)

Dublin        200000.0
Athlone        15000.0
Waterford          NaN
dtype: float64
```

Missing value

Detecting Missing Data

```
print(series2)
Dublin        200000.0
Athlone        15000.0
Waterford          NaN
dtype: float64

print(series2.isnull())
Dublin        False
Athlone       False
Waterford      True
dtype: bool

print(series2.notnull())
Dublin         True
Athlone        True
Waterford     False
dtype: bool
```

DataFrame

- `DataFrame` represents a tabular data structure
  – Similar to spreadsheet
  – Contains ordered collection of rows and columns
- Has both a row and column index
- Most common way to construct is from a dictionary of lists or NumPy arrays
  – Must be equal length
  – Index will be provided automatically
  – Columns placed in sorted order by default

DataFrame Example

```
from pandas import DataFrame

data = {'team':['Leicester', 'Manchester City',
'Arsenal'], 'player':['Vardy', 'Aguero', 'Sanchez'],
'goals':[24,22,19]}

football = DataFrame(data)

print(football)
   goals   player             team
0     24    Vardy        Leicester
1     22   Aguero  Manchester City
2     19  Sanchez          Arsenal
```

`DataFrame` Indexes

- Column order can be specified when creating `DataFrame`

```
data = {'team':['Leicester', 'Manchester City',
'Arsenal'], 'player':['Vardy', 'Aguero', 'Sanchez'],
'goals':[24,22,19]}

football = DataFrame(data,
       columns=['player','team','goals','played'],
       index=['one','two','three'])

print(football)

        player            team  goals played
one      Vardy       Leicester     24    NaN
two     Aguero  Manchester City    22    NaN
three  Sanchez         Arsenal     19    NaN
```

Extra column

Extra column

Index Objects

- Index objects are immutable and cannot be modified
  - Can be shared across data structures
    - Act as a set

```
print(football)
        player               team  goals played
one      Vardy          Leicester     24    NaN
two     Aguero  Manchester City     22    NaN
three  Sanchez            Arsenal     19    NaN

print('player' in football.columns)
True

print('three' in football.index)
True
```

Set operations

- Index has a number of methods found at:
  - http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Index.html

Chapter Concepts

`Series` Indexing and Selection

- Data from `Series` can be retrieved by index using integers and indexes

```
data = Series(np.arange(4.0), index=['a','b','c','d'])
```

```
print(data)
a     0.0
b     1.0
c     2.0
d     3.0
dtype: float64

print(data[2])
2.0
```

```
print(data[['b','d']])
b     1.0
d     3.0
dtype: float64
```

```
print(data<2)
a     True
b     True
c    False
d    False

print(data[data<2])
a     0.0
b     1.0
dtype: float64
```

`DataFrame` Indexing and Selection

- Indexing retrieves one or more columns

```
data = DataFrame(np.arange(9).reshape((3,3)),
          index=['a','b','c'], columns=['one','two','three'])
```

```
print(data)
    one  two  three
a     0    1      2
b     3    4      5
c     6    7      8
print(data['three'])
a     2
b     5
c     8
Name: three, dtype:
int64
```

```
print(data[:2])
    one  two  three
a     0    1      2
b     3    4      5
```

```
print(data['two']>1)
a    False
b     True
c     True
Name: two, dtype:bool
print(data[data['two']>1])
    one  two  three
b     3    4      5
c     6    7      8
```

`DataFrame` Indexing with `ix`, `loc`, and `iloc`

- `DataFrame` has several indexing fields: `ix`, `loc`, and `iloc`
  - Allows selecting subset of rows and columns

```
data = DataFrame(np.arange(9).reshape((3,3)),
         index=['a','b','c'], columns=['one','two','three'])
   one  two  three
a    0    1      2
b    3    4      5
c    6    7      8
```

- All these could be used to retrieve the first row
  - `data.ix[0] data.ix['a'] data.loc['a'] data.iloc[0]`
- All these would retrieve the second column for all rows
  - `data.ix[:,'two'], data.ix[:,1], data.loc[:,'two'], data.iloc[:,1]`
- All these would retrieve the first two rows and the second column
  - `data.ix[['a','b'],'two'], data.ix[0:2,1], data.loc['a':'c':,'two'], data.iloc[0:2,1]`

Series
Arithmetic and Data Alignment

- When adding together objects if the index pairs are not the same, then index in result is the union of the index pairs

```
data1 = Series([1.0,2.0,3.0], index=['a','d','e'])
data2 = Series([2.0,3.0,4.0, 5.0], index=['a','b','c','e'])
```

```
print(data1)
a     1.0
d     2.0
e     3.0
dtype: float64

print(data2)
a     2.0
b     3.0
c     4.0
e     5.0
dtype: float64
```

```
print(data1 + data2)
a     3.0
b     NaN
c     NaN
d     NaN
e     8.0
dtype: float64
```

`NaN` for indices that do not overlap

DataFrame
Arithmetic and Data Alignment

- DataFrame's alignment is performed on columns and rows

```
data1 = DataFrame(np.arange(9.0).reshape((3,3)),
columns=list('abc'), index=['one','two','three'])

data2 = DataFrame(np.arange(12.0).reshape((4,3)),
columns=list('ace'), index=['one','two','three','four'])
```

```
print(data1)
          a    b    c
one     0.0  1.0  2.0
two     3.0  4.0  5.0
three   6.0  7.0  8.0
```

```
print(data2)
          a     c     e
one     0.0   1.0   2.0
two     3.0   4.0   5.0
three   6.0   7.0   8.0
four    9.0  10.0  11.0
```

```
print(data1 + data2)
          a    b     c    e
four    NaN  NaN   NaN  NaN
one     0.0  NaN   3.0  NaN
three  12.0  NaN  15.0  NaN
two     6.0  NaN   9.0  NaN
```

Index and columns are unions of data1 and data2

Arithmetic with Fill Values

- For arithmetic between differently indexed objects, can use `fill_value` to prevent missing values appearing in resultant data structure

```
print(data1)
         a    b    c
one    0.0  1.0  2.0
two    3.0  4.0  5.0
three  6.0  7.0  8.0
```

```
print(data2)
         a    c     d     e
one    0.0  1.0   2.0   3.0
two    4.0  5.0   6.0   7.0
three  8.0  9.0  10.0  11.0
```

```
data1.add(data2, fill_value=0)
          a    b     c     d     e
one     0.0  1.0   3.0   2.0   3.0
two     7.0  4.0  10.0   6.0   7.0
three  14.0  7.0  17.0  10.0  11.0
```

Function Application and Mapping

- A frequent operation is to apply a function to each column or row of `DataFrame`

```
data = DataFrame(np.random.randn(4,4))
print(data)
          0         1         2         3
0  0.546781 -0.862394 -2.384923 -0.098065
1 -0.219738  0.172776 -1.558335 -0.124880
2 -1.016070 -0.670825 -1.602997 -0.018526
3 -0.050491  0.258218  0.073574  0.144686
```

```
f = lambda x: x.max() - x.min()
```

```
print(data.apply(f))
print(data.apply(f, axis=0))
0     1.562851
1     1.120612
2     2.458497
3     0.269566
dtype: float64
```

Apply per column

```
print(data.apply(f, axis=1))
0     2.931705
1     1.731110
2     1.584471
3     0.308709
dtype: float64
```

Along row

Sorting

- Sorting by index on either axis is available, ascending order by default

```
print(data)
          0         1         2         3
0  0.546781 -0.862394 -2.384923 -0.098065
1 -0.219738  0.172776 -1.558335 -0.124880
2 -1.016070 -0.670825 -1.602997 -0.018526
3 -0.050491  0.258218  0.073574  0.144686
```

Specify axis = 1 for column sorting

```
print(data.sort_index(ascending=False))
          0         1         2         3
3 -0.050491  0.258218  0.073574  0.144686
2 -1.016070 -0.670825 -1.602997 -0.018526
1 -0.219738  0.172776 -1.558335 -0.124880
0  0.546781 -0.862394 -2.384923 -0.098065
```

Descending order on rows

Could supply two sort keys with a list

- Can also sort by values instead of index

```
print(data.sort_values(by=[1], ascending=False))
```

Ranking

- Ranking assigns ranks from 1 to the number of valid data points in an array

```
data = DataFrame({'b':[1,4,3,2], 'a':[6,9,20,3], 'c':[7,2,8,15]})
print(data)

    a  b   c
0   6  1   7
1   9  4   2
2  20  3   8
3   3  2  15
```

Rank on column

```
print(data.rank())

     a    b    c
0  2.0  1.0  2.0
1  3.0  4.0  1.0
2  4.0  3.0  3.0
3  1.0  2.0  4.0
```

Rank on row

```
print(data.rank(axis=1))

     a    b    c
0  2.0  1.0  3.0
1  3.0  2.0  1.0
2  3.0  1.0  2.0
3  2.0  1.0  3.0
```

Chapter Concepts

Introduction to Pandas Data

Essential Functionality

Reading From Data Sources

Summarizing and Computing Descriptive Statistics

Handling Missing Data

Chapter Summary

Data

- To use the tools in this course, we need data to work with
- Data can be used from a variety of sources:
  - Text format
    - CSV
    - JSON
    - XML/HTML
  - Binary formats
    - HDF5
    - Excel
  - Databases
    - MongoDB

Working with Text Formats

- We will introduce pandas during this course
  - It provides a number of features for working with data in a tabular format
    - Known as a `DataFrame`
      - We will use this in our examples here with details to follow
- Pandas provides the following functions for reading data:
  - `read_csv`
    - Load data from delimited file or URL, comma is default delimiter
  - `read_table`
    - Load delimited data from a file or URL, tab is default delimiter
  - `read_fwf`
    - Read fixed-width column formatted file or URL

Reading a Text File Source

- The following code loads a `csv` file

```
data = pd.read_csv('sample.csv')
print(data)

   1   2   3   4          hello
0  5   6   7   8          world
1  9  10  11  12   some message
```

Returns a `DataFrame`

- `read_table` would work for this file too, but it's been deprecated

```
pd.read_table('sample.csv',sep=',')

   1   2   3   4          hello
0  5   6   7   8          world
1  9  10  11  12   some message
```

Delimeter to use

Reading Large Files

- It is possible to read large files in smaller fragments
  - Specify a `chunksize` to `read_csv`
    - Size is number of lines to supply

```
fragment = pd.read_csv('sample.csv', chunksize=1)

for line in fragment:
    print(line)

   1   2   3   4   hello
0  5   6   7   8   world
   1    2    3    4          hello
0  9   10   11   12   some message
```

Read a line at a time

Writing Data Out to Text Files

- Data can be exported to files in a delimited format

```
print(data)

    1    2    3    4            hello
0   5    6    7    8            world
1   9   10   11   12     some message

data.to_csv('file1.csv')
```

Write to file

- Can specify a separator too

```
data.to_csv('file1.csv', sep = '|')
```

Separator parameter

JSON Data

- Can read in JSON data using Python
  - Create `DataFrame` from data
- Consider the following JSON file:

```
{ "name":"jayne",
  "role":"sales",
   "customers" :
     [{"name":"Andersons","product":"Bosch","quantity":100},
      {"name":"ElectricalDirect","product":"Miele",
                "quantity":200}]}
```

```
import json
data = json.loads(open('example.json').read())
customers = DataFrame(data['customers'])
print(customers)

                  name product  quantity
0            Andersons   Bosch       100
1   Electrical Direct    Miele       200
```

Read JSON file

SQL Data

- Can read from SQL databases using a standard recipe
  - Import the package for the particular database
  - Open a connection
  - Create a cursor
  - Execute a query
  - Iterate through the cursor or fetch the results to a list
  - Close the connection when done

```
import sqlite3
cn = sqlite3.connect('test.sqlite')
curs = cn.cursor()
curs.execute("create table names (id int, name varchar(20))")
curs.execute("insert into names values(1, 'Alice'), (2, 'Bob')")
cn.commit()
curs.execute("select * from names")
names = curs.fetchall()
print(names)
names2 = pd.read_sql_query("select * from names", cn)
print(names2)
cn.close()

[(1, 'Alice'), (2, 'Bob')]
```

```
   id   name
0   1  Alice
1   2    Bob
```

Other Formats

- Python has many libraries for reading and writing different formats/sources
  – HTML and XML
- We will not cover the details here, but at a high level data can be processed from:
  – HTML/XML sources
  – Binary formats
  – Microsoft Excel files
  – Web APIs (JSON)
  – Databases
    - Relational
    - MongoDB

Chapter Concepts

Summarizing and Computing Descriptive Statistics

- Pandas objects have a set of common mathematical and statistical methods
- Most are reductions or summary statistics
  - Extract a single value, e.g., `sum()`
  - They exclude missing data

```
data = DataFrame([[1,np.nan],[3,4],[5,np.nan]],
                 columns=['a','b'])
print(data)

   a    b
0  1  NaN
1  3  4.0
2  5  NaN
```

Sums columns

```
print(data.sum())

a     9.0
b     4.0
dtype: float64
```

Sums rows

```
print(data.sum(axis=1))

0     1.0
1     7.0
2     5.0
dtype: float64
```

Pandas Mathematical Methods

- A few methods return multiple values, e.g., `describe()`

```
print (data)

   a    b
0  1  NaN
1  3  4.0
2  5  NaN
```

```
print (data.describe())

          a    b
count   3.0  1.0
mean    3.0  4.0
std     2.0  NaN
min     1.0  4.0
25%     2.0  NaN
50%     3.0  NaN
75%     4.0  NaN
max     5.0  4.0
```

Produces summary statistics

- Full list of DataFrame methods found at:
  – http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html

Correlation and Covariance

- Correlation and covariance are computed from pairs of arguments
- Consider fetching data from Yahoo! Finance

```
import pandas_datareader.data as web
all_data = {ticker: web.get_data_yahoo(ticker) for ticker in
['AAPL', 'IBM', 'MSFT', 'GOOG']}
print(all_data['AAPL'])
```

Data returned

```
[2418 rows x 6 columns],
'AAPL':
Date          Open        High        Low         Close       Volume

2010-01-04   626.951088  629.511067  624.241073  626.751061   3927000
2010-01-05   627.181073  627.841071  621.541045  623.991055   6031900
....
Date          Adj Close
2010-01-04   313.062468
2010-01-05   311.683844
```

Percentage Price Change

- Consider calculating the percentage change in the daily price of the stocks

```
price = DataFrame({ticker:data['Adj Close'] for
        ticker, data in all_data.items()})

print(price)

Date          AAPL       GOOG        IBM         MSFT
2010-01-04    27.727039  313.062468  111.405000  25.555485
2010-01-05    27.774976  311.683844  110.059232  25.563741
2010-01-06    27.333178  303.826685  109.344283  25.406859
```

```
returns = price.pct_change()

print(returns.tail())

                AAPL      GOOG       IBM      MSFT
Date
2017-03-23 -0.003536 -0.014477  0.000229 -0.002460
2017-03-24 -0.001987 -0.003853 -0.005663  0.001696
2017-03-27  0.001707  0.006238 -0.000345  0.001847
```

Daily price change

Correlation and Covariance

- DataFrame's `corr()` and `cov()` methods return a correlation or covariance matrix as a `DataFrame`

```
print(returns.corr())

          AAPL      GOOG       IBM      MSFT
AAPL  1.000000  0.409814  0.382086  0.389641
GOOG  0.409814  1.000000  0.402671  0.471145
IBM   0.382086  0.402671  1.000000  0.495369
MSFT  0.389641  0.471145  0.495369  1.000000
```

```
print(returns.cov())

          AAPL      GOOG       IBM      MSFT
AAPL  0.000267  0.000104  0.000075  0.000092
GOOG  0.000104  0.000242  0.000075  0.000106
IBM   0.000075  0.000075  0.000143  0.000085
MSFT  0.000092  0.000106  0.000085  0.000208
```

Chapter Concepts

Handling Missing Data

- Missing data is common in most data analysis applications
- Pandas tries to make working with missing data as painless as possible
  - The `NaN (NA)` value is used to represent missing data
- Two approaches to working with missing data:
  - Filter out missing data
  - Fill in missing data

Filtering Out Missing Data: `Series`

```
from numpy import nan as NA
data = Series([1,NA,2,3,4,NA])
print(data)
0     1.0
1     NaN
2     2.0
3     3.0
4     4.0
5     NaN
dtype: float64

print(data.dropna())
0     1.0
2     2.0
3     3.0
4     4.0
dtype: float64

data = data.dropna()
data.dropna(inplace=True)
```

Returns only non-null values and index values but does not modify the original data set

Reassign it back to the same variable to modify it or use `inplace=True`

Filtering Out Missing Data: `DataFrame`

- Useful parameters include:
  - `axis` which defaults to 0 for rows or 1 for columns
  - `how` which drops the row or column if `any` one value is NA or `all` are NA

```
data = DataFrame
([[1,2,3],[NA,5,NA],[NA,NA,NA],
[10,11,12]])
```

```
data = DataFrame
([[1,2,NA],[NA,5,NA],[NA,12,NA],
[10,11,NA]])
```

```
print(data.dropna(how='all'))

      0     1     2
0   1.0   2.0   3.0
1   NaN   5.0   NaN
3  10.0  11.0  12.0
```

```
print(data.dropna(how='all',axis=1))

      0     1
0   1.0   2.0
1   NaN   5.0
2   NaN  12.0
3  10.0  11.0
```

```
print(data.dropna(how='any'))

      0     1     2
0   1.0   2.0   3.0
3  10.0  11.0  12.0
```

```
print(data.dropna(how='any',axis=1))
      1
0   2.0
1   5.0
2  12.0
3  11.0
```

Filling In Missing Data

```
print(data)

      0     1     2
0   1.0   2.0   3.0
1   NaN   5.0   NaN
2   NaN   NaN   NaN
3  10.0  11.0  12.0
```

```
filled = data.fillna(0)
print(filled)

      0     1     2
0   1.0   2.0   3.0
1   0.0   5.0   0.0
2   0.0   0.0   0.0
3  10.0  11.0  12.0
```

```
filled = data.fillna({0:10, 1:11, 2:12})
print(filled)

      0     1     2
0   1.0   2.0   3.0
1  10.0   5.0  12.0
2  10.0  11.0  12.0
3  10.0  11.0  12.0
```

Supply dictionary with column:value pair for different fill values per column

Chapter Concepts

Chapter Summary

In this chapter, we have introduced:

- Pandas data structures
- Essential functionality of pandas
- Reading from data sources
- Summarizing and computing descriptive statistics
- Handling missing data