Dictionaries and Series

Python Module 5

Template by: Laurent Lessard

Dictionaries

Dictionaries (`dict`) are similar to lists, except they are indexed using strings rather than integers. Syntax:

```python
ages = {'Alice': 19, 'Bob': 24}
ages['Bob']
```

```
24
```

These are called (`key`, `value`) pairs.

Works for arbitrary data types:

```python
scores = {'Alice': [75, 86, 92], 'Bob': [71, 95, 84]}
scores['Alice']
```

```
[75, 86, 92]
```

More information: https://realpython.com/python-dicts/

Structured data

Use simple and intuitive data structures when possible!

This allows use of libraries *designed* to handle this simple data

```
database = {'Alice': {'high_school': 'West HS',
                      'college': 'UW',
                      'status': 'sophomore'},
            'Bob': {'high_school': 'East HS',
                    'college': 'UW',
                    'status': 'alum'}}

database['Bob']['college']
```

```
UW
```

Indentation only matters for the first line. You can space out and align your code like this to make it more readable.

Creating a dictionary

- Empty dictionary: use `D = dict()` or `D = {}`.
- Creating from data:
    - Using the curly brace syntax:
      ```
      D = { 'a':1, 'b':2, 'c':3 }
      ```
    - Using a list of lists (or a list of tuples, or a tuple of tuples, etc.):
      ```
      D = dict( [('a',1), ('b',2), ('c',3)] )
      ```
    - Using an optional argument for each (key,value) pair:
      ```
      D = dict( a=1, b=2, c=3 )
      ```
    - Creating a blank dictionary and adding values later:
      ```
      D = {};  D['a'] = 1   # and so on...
      ```

Tabular data example

State	Population (in millions)	Gross Domestic Product (in millions of $)
CA	39.6	2.7
TX	28.7	1.7
NY	19.5	1.5
…	…	…

Each row: one observation
Each column: features of each observation

Many possible representations

- Row-first:

```
state_data = { 'CA': { 'pop':39.6, 'gdp':2.7 },
               'TX': { 'pop':28.7, 'gdp':1.7 },
               'NY': { 'pop':19.5, 'gdp':1.5 }}
```

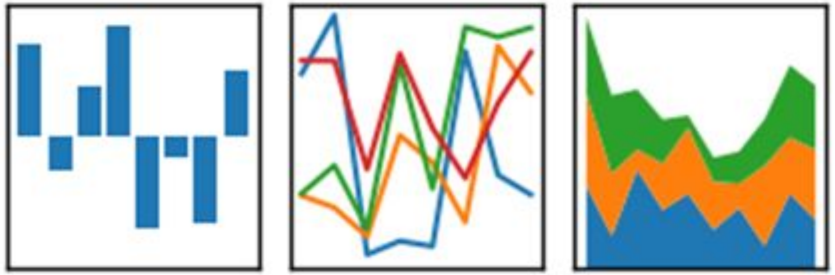
- Column-first:

```
state_data = { 'pop': { 'CA':39.6, 'TX':28.7, 'NY':19.5 },
               'gdp': { 'CA':2.7,  'TX':1.7,  'NY':1.5  }}
```

- What is the standard way to represent tabular data?

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Pandas is a Python package providing **fast, flexible, and expressive data structures** designed to make working with **“relational” or “labeled” data** both easy and intuitive.

It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

Importing Pandas

- Pandas must be imported before it can be used
- Pandas functions are called using the dot operator

```
import pandas
pandas.merge(...)
pandas.concat(...)
```

- It's standard to use the alias **pd** for pandas

```
import pandas as pd
pd.merge(...)
pd.concat(...)
```

You don't have to call it "pd", but this name is an agreed-upon convention.

Warning: Namespaces

- You can also import specific functions

```
from pandas import merge, concat
merge(...)
concat(...)
```

- Or just import everything

```
from pandas import *
merge(...)
concat(...)
```

Seems like the simplest solution, but pandas is a LARGE package with lots of functions. Having the "**pd.**" prefix is a helpful reminder that the function you're calling is a pandas function.

Pandas Series

Series (`pd.Series`) are the building block of Pandas. They…

- …are very similar to dictionaries (`dict`) and lists (`list`).
- …provide efficient computation and storage
- …provide additional advanced functionality (more later!)

Pandas Series work best with flat data structures (e.g., lists of floats) though they can be used with arbitrary data structures.

Series indexing

- Series can be indexed like a dict, like a list, or using dot:

```
import pandas as pd

population = { 'CA':39.6, 'TX':28.7, 'NY':19.5 }
s = pd.Series( population )

s['CA']       # returns 39.6
s.TX          # returns 28.7
s[-1]         # returns 19.5
```

- data and index can be specified separately

```
state_names = ['CA', 'TX', 'NY']
pop_values = [39.6, 28.7, 19.5 ]
s = pd.Series( data=pop_values, index=state_names )
```

Series operations

- Adding Series together automatically aligns indices

```
s1 = pd.Series( {'CA':39.6,'TX':28.7,'NY':19.5} )
s2 = pd.Series( {'TX':10.1,'CA':10.2,'MA':11.6} )
s1 + s2
```

```
CA     49.8
MA      NaN
NY      NaN
TX     38.8
dtype: float64
```

NaN stands for "not a number". Used by pandas to indicate missing or unknown data entries.

dtype is the data type of the Series.

- This also works with * and other operations as well…