
Storing Tweets

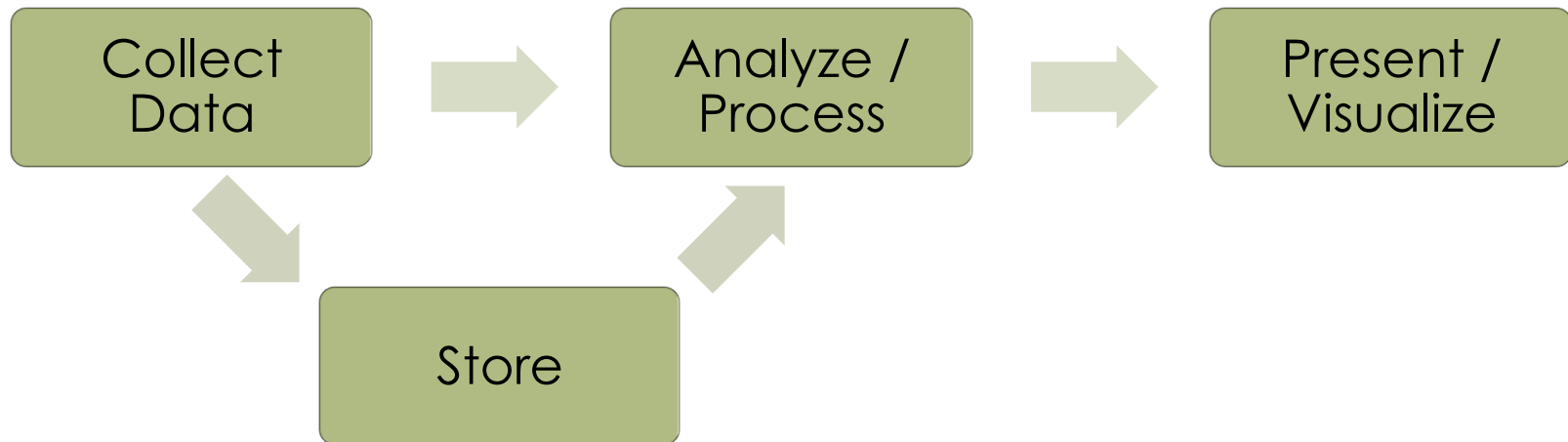
Pickling

Connecting to MySQL through Python

Outline

- Storing or Streaming
 - Simple Storage: Pickling in Python
 - Connecting and using MySQL
-

Processing pipeline



- Streaming vs Storing
-

Simple Storage

- Tweet data response – JSON
 - This is text
 - What does this look like?

Simple Storage

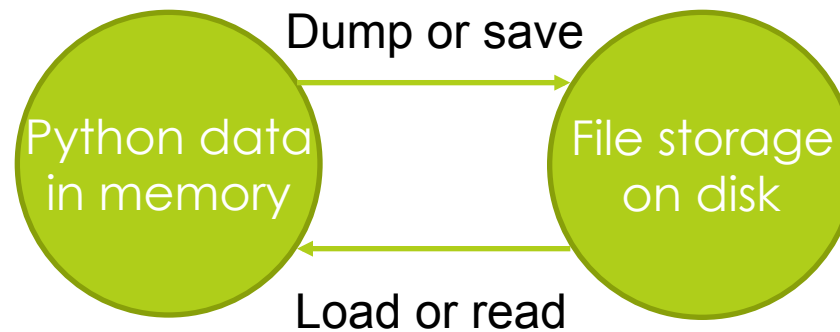
```
{
  "created_at": "Wed Jan 15 05:34:43 +0000 2014",
  "id": 423327363959492610,
  "id_str": "423327363959492610",
  "text": "RT @nflnetwork: Anyone up for some Football?\n\n@Seahawks vs.
@49ers. Week 14. Right now. #NFLReplay http://t.co/12bqr3He9f",
  "user": {
    "followers_count": 87,
    "friends_count": 400,
    "id": 245167474,
    "id_str": "245167474",
    "location": "Dallas,Tx",
    "name": "La Troy Woodruff",
    "screen_name": "LockDown_22",
    "verified": false
  }
}
```

Simple Storage

- Tweet data response – JSON
 - This is text
 - What does this look like?
 - JSON is more useful as a python dictionary
 - Using the json module
 - `import json`
 - Sample read a JSON text file and convert to python dictionary
-

Storage

- Loading and dumping



Read A JSON File (example 1)

```
import sys, json, pickle, cPickle

def read_json_file(fname=""):
    jsdict = dict()
    fdoc = ""
    if fname:
        fin = open(fname, "r")
        for line in fin:
            fdoc=fdoc+line
        fin.close()
        jsdict = json.loads(fdoc)
    else:
        print "Must supply a filename"
    return jsdict
```


Read A JSON File (example 2)

```
import sys, json, pickle, cPickle

def read_json_jsonload (fname=""):
    jsdict = {}
    if fname:
        fin = open(fname, "r")
        jsdict = json.load(fin)
    else:
        print "Must supply a filename"
    return jsdict
```

Write a JSON File

- What would it take to save a JSON file?

What would **you** do?

Write a JSON File

- ▣ What would it take to save a JSON file?
 - ▣ JSON is just text
 - ▣ We saw sample code to read and write text
 - ▣ Could that work?

Alternatives to JSON?

- JSON
 - Easy to read, edit – it's just text
 - Slow to load, read and convert to dict (python dictionary)
 - Pickle
 - Raw data, dictionary
 - Quick to load, no conversion
 - Both are built into Python, simple
 - More sophisticated DBs require more work
-

Save data (pickle)

```
import sys, json, pickle, cPickle

def pickle_dictionary(fname="",d={}):
    if fname:
        fout = open(fname, "w")
        pickle.dump(d,fout)
        #cPickle.dump(d,fout)
    else:
        print "Must supply a filename"
    return
```

Load data (unpickle)

```
import sys, json, pickle, cPickle

def unpickle_dictionary(fname=""):
    d = {}
    if fname:
        fin = open(fname, "r")
        d = pickle.load(fin)
        #d = cPickle.load(fin)
    else:
        print "Must supply a filename"
    return d
```

Little demo

- Pickling and unpickling

Connecting to MySQL

Using the HCDE user data module

Using MySQL in Python

- Need to have MySQL installed
 - Need a DB (a schema, a table structure)
 - Need a Python to MySQL connector
 - Nice to have, an Object-Relation Manager (ORM)
-

Using MySQL in Python

- Need MySQL installed
 - You just need to install this ("community edition" is free)
 - Need a DB (a schema, a table structure)
 - Sample provided in `hcde.data.db.schemas`
 - e.g. "tweet_hcde530-db-schemas.sql"
 - Need a Python to MySQL connector
 - `pymysql`
 - Nice to have a Object-Relation Manager (ORM)
 - `SQLAlchemy`
-

Test the install

```
[Tiki:~Development/python/] dwmc% python
Python 2.7.10 (default, Oct 23 2015, 18:05:06)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import pymysql
>>> import sqlalchemy
>>>
```

Basic Twitter SQL Schema

- Tweets Table
 - User Table
 - Friends Table
 - Followers Table
 - User Meta Table
-

Simple Twitter Schema Demo

- Browse - `hcde.data.db.schemas`
 - `tweet_hcde530-db-schemas.sql`

ORM (Object-Relation Manager)

- An ORM Manages
 - The relation between the data in the memory of the machine (while the program is running) and the more permanent storage of that data in a database
 - Changes to an 'object' in memory is (eventually) mirrored to the database.

Working With sqlalchemy (ORM)

- Need to develop an Python Object abstraction for each database table
 - See `hcde.data.db.base`
 - `tweetObj.py`
 - `userObj.py`
 - `userMetaObj.py`
 - `friendObj.py`
 - `followerObj.py`
- Also, convenient to have an abstraction for the DB itself
 - `TweetsDB.py`
- These are all usable base classes, we probably want specialized versions for a real data collection

Specializing ORM Classes

- Specialized “example” classes

- See `hcde.data.db.example`

- `ExampleTweetObj.py`

- `ExampleUserObj.py`

- `ExampleUserMetaObj.py`

- `ExampleFriendObj.py`

- `ExampleFollowerObj.py`

- `ExampleTweetsDB.py`

- These could be used for your project if you wanted to use MySQL for your storage

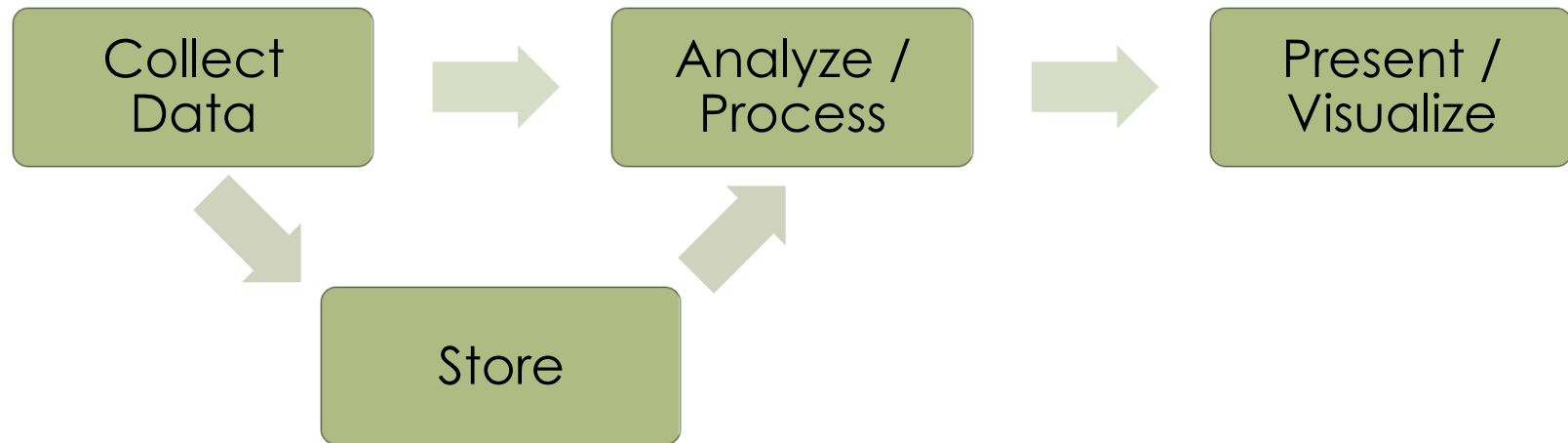
Demo Connecting

- HCDE user module tries to hide some of the complexity of using ORM
 - `hcde.data.db.base`
 - `dbConfig()` object
 - `baseDB()` object
 - `hcde.data.db.fitness`
 - `FitTweetsDB()`
 - `FitTweetObj()`
 - `settings_db.py`

Demo Using Fitness Data

- HCDE user module has four datasets
 - `hcde.data.election_2012`
 - `hcde.data.election_2016`
 - `hcde.data.fitness`
 - `hcde.data.oscar_2016`
 - Let's try working with the fitness data
 - The dates of the collection are in the file "constants.py" in the directory for the collection.
-

Processing pipeline



- When using the fitness data, what path are we following?
-