We need to assess out data to:

* Test assumptions about
  + Values
  + Data Types
  + Shape
* Identify errors or outliers
* Find Missing Values

Tabular Data - Spreadsheets

CSV:

* Lightweight
* Fields are separated by a Delimeter(comma typically)
* Just the data and the delimeters
* Do not need special software to use
* All spreadsheet Apps can Read/Write CSV

Python Module ‘xlrd’ allows you to work with older (.xls) and newer (.xlsx) excel formats. Allows python to read in any excel file and work with it programatically. ‘Xlwt’ module allows us to create Excel files as we need.

JSON:

* Items may have different fields
* May have nested objects
* May have nested arrays

XML:

* Data Transfer (Producer App and Consumer App can be in different languages and can communicate so long as they “speak XML” to eachother)
* Easy to Write Code to Read/Write
* Document Validation
* Human Readable
* Supports a Wide Variety of Apps
* XML Standard
  + Has robust parsers in most languages (Python included)
  + Allows us to focus on our app
    - Means that we don’t have to create parsers for ad hoc data types
* Useful parsing code
  + Very generalized, must be tailored to what kind of XML

# Will get the root element of XML doc, then will get it’s child elements

import xml.etree.ElementTree as et

import pprint

tree = et.parse('exampleResearchArticle.xml')

root = tree.getroot()

print "\nChildren of root:"

for child in root:

print child.tag

------------------------------------------------------------------------------------------------------------------

#Searches for the title of article and prints it

import xml.etree.ElementTree as et

import pprint

tree = et.parse('exampleResearchArticle.xml')

root = tree.getroot()

title = root.find('./fm/bibl/title')

title\_text = ""

for p in title:

title\_text += p.text

print "\nTitle:\n", title\_text

#Searches for Authors in the Author group of the bibliography section

print "\nAuthor email addresses:"

for a in root.findall('./fm/bibl/aug/au'):

email = a.find('email')

if email is not None:

print email.text

---------------------------------------------------------------------------------------------------------------

Beautiful Soup:

* a Python library for pulling data out of HTML and XML files.

----------------------------------------------------------------------------------------------------------------

SCREEN SCRAPING:

* Typically required when dealing with a govt organization or a small grass-roots campaign of some kind. I.E. [transtats.bts.gov](https://www.transtats.bts.gov/)

Best Practice for Scraping:

1. Look at how a browser makes requests. (Dev tools in browser are good method)
2. Emulate in code
3. If stuff blows up, look at the HTTP traffic
4. Return to 1 until it works

**Data Quality/Data Cleaning**

Sources of Dirty Data

* User Entry Errors
* Poorly Applied Coding Standards
* Different Schemas
* Legacy Systems
* Evolving Applications
* No unique Identifiers
* Data Migration
* Programmer Error
* Corrupted in Transmission/Storage

**Measures of Data Quality**

* Validity: The degree to which our data conforms to a schema
* Accuracy: The degree data conforms to gold standard
* Completeness: Do we have all records needed?
* Consistency: Matches other data
* Uniformity: Do we use the same units (Miles vs Kilometers)

**Blueprint for Cleaning Data**

* Audit your data
  + Programatically checking the data. Create a report on the quality of the data.
  + Possibly run a stat analysis, for say, outliers
* Create a data cleaning plan
  + Identify causes (why is the data dirty?)
  + Define operations (ways to correct this)
  + TEST
* Execute the plan
  + Likely running a script to correct what is dirty
* Manually Correct