Diagram

Description automatically generated

Spidering and Modeling Email Data - Introduction

This week we do the first half of a project to download, process, and visualize an email corpus from the Sakai open source project from 2004-2011:

<http://mbox.dr-chuck.net/>

This is a large amount of data and requires significant cleanup to make sense of the data before we visualize it.

**Important:** You do not have to download all of the data to complete this project. Depending on your Internet connection, downloading nearly a gigabyte of data might be impossible. All we want to do is to have you download a small subset of the data and run the steps to process the data.

Here is the software we will be using to retrieve and process the email data:

<https://www.py4e.com/code3/gmane.zip>

If you have a fast network connection with no bandwidth charge - you can download all the data. If you try to download all the data it may take well over 24 hours to pull the data. The good news is that because there are separate crawl, clean, model, and visualization steps, you can start and stop the crawl process as often as you like and run the other processes on the data that has been downloaded so far.

Welcome Jeannie Kim-Schreiber from Capstone: Retrieving, Processing, and Visualizing Data with Python

#### Mailing List Data - Part I

In this assignment you will download some of the mailing list data from http://mbox.dr-chuck.net/ and run the data cleaning / modeling process and take some screen shots. All screenshots should use the same data set.

Don't take off points for little mistakes. If they seem to have done the assignment give them full credit. Feel free to make suggestions if there are small mistakes. Please keep your comments positive and useful. The course staff will assign the last 30% of the grade and it make take a few days to get graded.

**Please Upload Your Submission:**

Top of Form

A screen shot of your SQLiteBrowser showing messages downloaded from mbox.dr-chuck.net into the content.sqlite database including all data fields and the bottom of sqlite browser.(Please use PNG or JPG files 1024KB max)

A screen shot of you running the gmodel.py application to produce the index.sqlite database. Please include the allsenders line at the beginning of the output. Please do not include code in the screenshot.(Please use PNG or JPG files 1024KB max)

A screen shot of your SQLiteBrowser showing messages in the index.sqlite database after the gmodel.py has executed including all data fields and the bottom of sqlite browser..(Please use PNG or JPG files 1024KB max)

A screen shot of you running the gbasic.py program to compute basic histogram data on the messages you have retrieved. Please include the "How many to dump" and "Loaded messages" lines in the output. For a dump number, please use a number less than 25. Do not include code in the screenshot.(Please use PNG or JPG files 1024KB max)

Enter optional comments below

Bottom of Form

**gmane.py**

**import** sqlite3  
**import** time  
**import** ssl  
**import** urllib.request, urllib.parse, urllib.error  
**from** urllib.parse **import** urljoin  
**from** urllib.parse **import** urlparse  
**import** re  
**from** datetime **import** datetime, timedelta  
  
*# Not all systems have this so conditionally define parser***try**:  
 **import** dateutil.parser **as** parser  
**except**:  
 **pass  
  
def** parsemaildate(md) :  
 *# See if we have dateutil* **try**:  
 pdate = parser.parse(tdate)  
 test\_at = pdate.isoformat()  
 **return** test\_at  
 **except**:  
 **pass** *# Non-dateutil version - we try our best* pieces = md.split()  
 notz = **" "**.join(pieces[:4]).strip()  
  
 *# Try a bunch of format variations - strptime() is \*lame\** dnotz = **None  
 for** form **in** [ **'%d %b %Y %H:%M:%S'**, **'%d %b %Y %H:%M:%S'**,  
 **'%d %b %Y %H:%M'**, **'%d %b %Y %H:%M'**, **'%d %b %y %H:%M:%S'**,  
 **'%d %b %y %H:%M:%S'**, **'%d %b %y %H:%M'**, **'%d %b %y %H:%M'** ] :  
 **try**:  
 dnotz = datetime.strptime(notz, form)  
 **break  
 except**:  
 **continue  
  
 if** dnotz **is None** :  
 *# print 'Bad Date:',md* **return None** iso = dnotz.isoformat()  
  
 tz = **"+0000"  
 try**:  
 tz = pieces[4]  
 ival = int(tz) *# Only want numeric timezone values* **if** tz == **'-0000'** : tz = **'+0000'** tzh = tz[:3]  
 tzm = tz[3:]  
 tz = tzh+**":"**+tzm  
 **except**:  
 **pass  
  
 return** iso+tz  
  
*# Ignore SSL certificate errors*ctx = ssl.create\_default\_context()  
ctx.check\_hostname = **False**ctx.verify\_mode = ssl.CERT\_NONE  
  
conn = sqlite3.connect(**'content.sqlite'**)  
cur = conn.cursor()  
  
baseurl = **"http://mbox.dr-chuck.net/sakai.devel/"**cur.execute(**'''CREATE TABLE IF NOT EXISTS Messages  
 (id INTEGER UNIQUE, email TEXT, sent\_at TEXT,  
 subject TEXT, headers TEXT, body TEXT)'''**)  
  
*# Pick up where we left off*start = **None**cur.execute(**'SELECT max(id) FROM Messages'** )  
**try**:  
 row = cur.fetchone()  
 **if** row **is None** :  
 start = 0  
 **else**:  
 start = row[0]  
**except**:  
 start = 0  
  
**if** start **is None** : start = 0  
  
many = 0  
count = 0  
fail = 0  
**while True**:  
 **if** ( many < 1 ) :  
 conn.commit()  
 sval = input(**'How many messages:'**)  
 **if** ( len(sval) < 1 ) : **break** many = int(sval)  
  
 start = start + 1  
 cur.execute(**'SELECT id FROM Messages WHERE id=?'**, (start,) )  
 **try**:  
 row = cur.fetchone()  
 **if** row **is not None** : **continue  
 except**:  
 row = **None** many = many - 1  
 url = baseurl + str(start) + **'/'** + str(start + 1)  
  
 text = **"None"  
 try**:  
 *# Open with a timeout of 30 seconds* document = urllib.request.urlopen(url, **None**, 30, context=ctx)  
 text = document.read().decode()  
 **if** document.getcode() != 200 :  
 print(**"Error code="**,document.getcode(), url)  
 **break  
 except** KeyboardInterrupt:  
 print(**''**)  
 print(**'Program interrupted by user...'**)  
 **break  
 except** Exception **as** e:  
 print(**"Unable to retrieve or parse page"**,url)  
 print(**"Error"**,e)  
 fail = fail + 1  
 **if** fail > 5 : **break  
 continue** print(url,len(text))  
 count = count + 1  
  
 **if not** text.startswith(**"From "**):  
 print(text)  
 print(**"Did not find From "**)  
 fail = fail + 1  
 **if** fail > 5 : **break  
 continue** pos = text.find(**"\n\n"**)  
 **if** pos > 0 :  
 hdr = text[:pos]  
 body = text[pos+2:]  
 **else**:  
 print(text)  
 print(**"Could not find break between headers and body"**)  
 fail = fail + 1  
 **if** fail > 5 : **break  
 continue** email = **None** x = re.findall(**'\nFrom: .\* <(\S+@\S+)>\n'**, hdr)  
 **if** len(x) == 1 :  
 email = x[0];  
 email = email.strip().lower()  
 email = email.replace(**"<"**,**""**)  
 **else**:  
 x = re.findall(**'\nFrom: (\S+@\S+)\n'**, hdr)  
 **if** len(x) == 1 :  
 email = x[0];  
 email = email.strip().lower()  
 email = email.replace(**"<"**,**""**)  
  
 date = **None** y = re.findall(**'\Date: .\*, (.\*)\n'**, hdr)  
 **if** len(y) == 1 :  
 tdate = y[0]  
 tdate = tdate[:26]  
 **try**:  
 sent\_at = parsemaildate(tdate)  
 **except**:  
 print(text)  
 print(**"Parse fail"**,tdate)  
 fail = fail + 1  
 **if** fail > 5 : **break  
 continue** subject = **None** z = re.findall(**'\Subject: (.\*)\n'**, hdr)  
 **if** len(z) == 1 : subject = z[0].strip().lower();  
  
 *# Reset the fail counter* fail = 0  
 print(**" "**,email,sent\_at,subject)  
 cur.execute(**'''INSERT OR IGNORE INTO Messages (id, email, sent\_at, subject, headers, body)  
 VALUES ( ?, ?, ?, ?, ?, ? )'''**, ( start, email, sent\_at, subject, hdr, body))  
 **if** count % 50 == 0 : conn.commit()  
 **if** count % 100 == 0 : time.sleep(1)  
  
conn.commit()  
cur.close()

**gmodel.py**

**import** sqlite3  
**import** time  
**import** re  
**import** zlib  
**from** datetime **import** datetime, timedelta  
  
*# Not all systems have this***try**:  
 **import** dateutil.parser **as** parser  
**except**:  
 **pass**dnsmapping = dict()  
mapping = dict()  
  
**def** fixsender(sender,allsenders=**None**) :  
 **global** dnsmapping  
 **global** mapping  
 **if** sender **is None** : **return None** sender = sender.strip().lower()  
 sender = sender.replace(**'<'**,**''**).replace(**'>'**,**''**)  
  
 *# Check if we have a hacked gmane.org from address* **if** allsenders **is not None and** sender.endswith(**'gmane.org'**) :  
 pieces = sender.split(**'-'**)  
 realsender = **None  
 for** s **in** allsenders:  
 **if** s.startswith(pieces[0]) :  
 realsender = sender  
 sender = s  
 *# print(realsender, sender)* **break  
 if** realsender **is None** :  
 **for** s **in** mapping:  
 **if** s.startswith(pieces[0]) :  
 realsender = sender  
 sender = mapping[s]  
 *# print(realsender, sender)* **break  
 if** realsender **is None** : sender = pieces[0]  
  
 mpieces = sender.split(**"@"**)  
 **if** len(mpieces) != 2 : **return** sender  
 dns = mpieces[1]  
 x = dns  
 pieces = dns.split(**"."**)  
 **if** dns.endswith(**".edu"**) **or** dns.endswith(**".com"**) **or** dns.endswith(**".org"**) **or** dns.endswith(**".net"**) :  
 dns = **"."**.join(pieces[-2:])  
 **else**:  
 dns = **"."**.join(pieces[-3:])  
 *# if dns != x : print(x,dns)  
 # if dns != dnsmapping.get(dns,dns) : print(dns,dnsmapping.get(dns,dns))* dns = dnsmapping.get(dns,dns)  
 **return** mpieces[0] + **'@'** + dns  
  
**def** parsemaildate(md) :  
 *# See if we have dateutil* **try**:  
 pdate = parser.parse(md)  
 test\_at = pdate.isoformat()  
 **return** test\_at  
 **except**:  
 **pass** *# Non-dateutil version - we try our best* pieces = md.split()  
 notz = **" "**.join(pieces[:4]).strip()  
  
 *# Try a bunch of format variations - strptime() is \*lame\** dnotz = **None  
 for** form **in** [ **'%d %b %Y %H:%M:%S'**, **'%d %b %Y %H:%M:%S'**,  
 **'%d %b %Y %H:%M'**, **'%d %b %Y %H:%M'**, **'%d %b %y %H:%M:%S'**,  
 **'%d %b %y %H:%M:%S'**, **'%d %b %y %H:%M'**, **'%d %b %y %H:%M'** ] :  
 **try**:  
 dnotz = datetime.strptime(notz, form)  
 **break  
 except**:  
 **continue  
  
 if** dnotz **is None** :  
 *# print('Bad Date:',md)* **return None** iso = dnotz.isoformat()  
  
 tz = **"+0000"  
 try**:  
 tz = pieces[4]  
 ival = int(tz) *# Only want numeric timezone values* **if** tz == **'-0000'** : tz = **'+0000'** tzh = tz[:3]  
 tzm = tz[3:]  
 tz = tzh+**":"**+tzm  
 **except**:  
 **pass  
  
 return** iso+tz  
  
*# Parse out the info...***def** parseheader(hdr, allsenders=**None**):  
 **if** hdr **is None or** len(hdr) < 1 : **return None** sender = **None** x = re.findall(**'\nFrom: .\* <(\S+@\S+)>\n'**, hdr)  
 **if** len(x) >= 1 :  
 sender = x[0]  
 **else**:  
 x = re.findall(**'\nFrom: (\S+@\S+)\n'**, hdr)  
 **if** len(x) >= 1 :  
 sender = x[0]  
  
 *# normalize the domain name of Email addresses* sender = fixsender(sender, allsenders)  
  
 date = **None** y = re.findall(**'\nDate: .\*, (.\*)\n'**, hdr)  
 sent\_at = **None  
 if** len(y) >= 1 :  
 tdate = y[0]  
 tdate = tdate[:26]  
 **try**:  
 sent\_at = parsemaildate(tdate)  
 **except** Exception **as** e:  
 *# print('Date ignored ',tdate, e)* **return None** subject = **None** z = re.findall(**'\nSubject: (.\*)\n'**, hdr)  
 **if** len(z) >= 1 : subject = z[0].strip().lower()  
  
 guid = **None** z = re.findall(**'\nMessage-ID: (.\*)\n'**, hdr)  
 **if** len(z) >= 1 : guid = z[0].strip().lower()  
  
 **if** sender **is None or** sent\_at **is None or** subject **is None or** guid **is None** :  
 **return None  
 return** (guid, sender, subject, sent\_at)  
  
conn = sqlite3.connect(**'index.sqlite'**)  
cur = conn.cursor()  
  
cur.execute(**'''DROP TABLE IF EXISTS Messages '''**)  
cur.execute(**'''DROP TABLE IF EXISTS Senders '''**)  
cur.execute(**'''DROP TABLE IF EXISTS Subjects '''**)  
cur.execute(**'''DROP TABLE IF EXISTS Replies '''**)  
  
cur.execute(**'''CREATE TABLE IF NOT EXISTS Messages  
 (id INTEGER PRIMARY KEY, guid TEXT UNIQUE, sent\_at INTEGER,  
 sender\_id INTEGER, subject\_id INTEGER,  
 headers BLOB, body BLOB)'''**)  
cur.execute(**'''CREATE TABLE IF NOT EXISTS Senders  
 (id INTEGER PRIMARY KEY, sender TEXT UNIQUE)'''**)  
cur.execute(**'''CREATE TABLE IF NOT EXISTS Subjects  
 (id INTEGER PRIMARY KEY, subject TEXT UNIQUE)'''**)  
cur.execute(**'''CREATE TABLE IF NOT EXISTS Replies  
 (from\_id INTEGER, to\_id INTEGER)'''**)  
  
conn\_1 = sqlite3.connect(**'mapping.sqlite'**)  
cur\_1 = conn\_1.cursor()  
  
cur\_1.execute(**'''SELECT old,new FROM DNSMapping'''**)  
**for** message\_row **in** cur\_1 :  
 dnsmapping[message\_row[0].strip().lower()] = message\_row[1].strip().lower()  
  
mapping = dict()  
cur\_1.execute(**'''SELECT old,new FROM Mapping'''**)  
**for** message\_row **in** cur\_1 :  
 old = fixsender(message\_row[0])  
 new = fixsender(message\_row[1])  
 mapping[old] = fixsender(new)  
  
*# Done with mapping.sqlite*conn\_1.close()  
  
*# Open the main content (Read only)*conn\_1 = sqlite3.connect(**'file:content.sqlite?mode=ro'**, uri=**True**)  
cur\_1 = conn\_1.cursor()  
  
allsenders = list()  
cur\_1.execute(**'''SELECT email FROM Messages'''**)  
**for** message\_row **in** cur\_1 :  
 sender = fixsender(message\_row[0])  
 **if** sender **is None** : **continue  
 if 'gmane.org' in** sender : **continue  
 if** sender **in** allsenders: **continue** allsenders.append(sender)  
  
print(**"Loaded allsenders"**,len(allsenders),**"and mapping"**,len(mapping),**"dns mapping"**,len(dnsmapping))  
  
cur\_1.execute(**'''SELECT headers, body, sent\_at  
 FROM Messages ORDER BY sent\_at'''**)  
  
senders = dict()  
subjects = dict()  
guids = dict()  
  
count = 0  
  
**for** message\_row **in** cur\_1 :  
 hdr = message\_row[0]  
 parsed = parseheader(hdr, allsenders)  
 **if** parsed **is None**: **continue** (guid, sender, subject, sent\_at) = parsed  
  
 *# Apply the sender mapping* sender = mapping.get(sender,sender)  
  
 count = count + 1  
 **if** count % 250 == 1 : print(count,sent\_at, sender)  
 *# print(guid, sender, subject, sent\_at)* **if 'gmane.org' in** sender:  
 print(**"Error in sender ==="**, sender)  
  
 sender\_id = senders.get(sender,**None**)  
 subject\_id = subjects.get(subject,**None**)  
 guid\_id = guids.get(guid,**None**)  
  
 **if** sender\_id **is None** :  
 cur.execute(**'INSERT OR IGNORE INTO Senders (sender) VALUES ( ? )'**, ( sender, ) )  
 conn.commit()  
 cur.execute(**'SELECT id FROM Senders WHERE sender=? LIMIT 1'**, ( sender, ))  
 **try**:  
 row = cur.fetchone()  
 sender\_id = row[0]  
 senders[sender] = sender\_id  
 **except**:  
 print(**'Could not retrieve sender id'**,sender)  
 **break  
 if** subject\_id **is None** :  
 cur.execute(**'INSERT OR IGNORE INTO Subjects (subject) VALUES ( ? )'**, ( subject, ) )  
 conn.commit()  
 cur.execute(**'SELECT id FROM Subjects WHERE subject=? LIMIT 1'**, ( subject, ))  
 **try**:  
 row = cur.fetchone()  
 subject\_id = row[0]  
 subjects[subject] = subject\_id  
 **except**:  
 print(**'Could not retrieve subject id'**,subject)  
 **break** *# print(sender\_id, subject\_id)* cur.execute(**'INSERT OR IGNORE INTO Messages (guid,sender\_id,subject\_id,sent\_at,headers,body) VALUES ( ?,?,?,datetime(?),?,? )'**,  
 ( guid, sender\_id, subject\_id, sent\_at,  
 zlib.compress(message\_row[0].encode()), zlib.compress(message\_row[1].encode())) )  
 conn.commit()  
 cur.execute(**'SELECT id FROM Messages WHERE guid=? LIMIT 1'**, ( guid, ))  
 **try**:  
 row = cur.fetchone()  
 message\_id = row[0]  
 guids[guid] = message\_id  
 **except**:  
 print(**'Could not retrieve guid id'**,guid)  
 **break**cur.close()  
cur\_1.close()

**gbasic.py**

**import** sqlite3  
**import** time  
**import** zlib  
  
howmany = int(input(**"How many to dump? "**))  
  
conn = sqlite3.connect(**'index.sqlite'**)  
cur = conn.cursor()  
  
cur.execute(**'SELECT id, sender FROM Senders'**)  
senders = dict()  
**for** message\_row **in** cur :  
 senders[message\_row[0]] = message\_row[1]  
  
cur.execute(**'SELECT id, subject FROM Subjects'**)  
subjects = dict()  
**for** message\_row **in** cur :  
 subjects[message\_row[0]] = message\_row[1]  
  
*# cur.execute('SELECT id, guid,sender\_id,subject\_id,headers,body FROM Messages')*cur.execute(**'SELECT id, guid,sender\_id,subject\_id,sent\_at FROM Messages'**)  
messages = dict()  
**for** message\_row **in** cur :  
 messages[message\_row[0]] = (message\_row[1],message\_row[2],message\_row[3],message\_row[4])  
  
print(**"Loaded messages="**,len(messages),**"subjects="**,len(subjects),**"senders="**,len(senders))  
  
sendcounts = dict()  
sendorgs = dict()  
**for** (message\_id, message) **in** list(messages.items()):  
 sender = message[1]  
 sendcounts[sender] = sendcounts.get(sender,0) + 1  
 pieces = senders[sender].split(**"@"**)  
 **if** len(pieces) != 2 : **continue** dns = pieces[1]  
 sendorgs[dns] = sendorgs.get(dns,0) + 1  
  
print(**''**)  
print(**'Top'**,howmany,**'Email list participants'**)  
  
x = sorted(sendcounts, key=sendcounts.get, reverse=**True**)  
**for** k **in** x[:howmany]:  
 print(senders[k], sendcounts[k])  
 **if** sendcounts[k] < 10 : **break**print(**''**)  
print(**'Top'**,howmany,**'Email list organizations'**)  
  
x = sorted(sendorgs, key=sendorgs.get, reverse=**True**)  
**for** k **in** x[:howmany]:  
 print(k, sendorgs[k])  
 **if** sendorgs[k] < 10 : **break**

**gline.py**

**import** sqlite3  
**import** time  
**import** zlib  
  
conn = sqlite3.connect(**'index.sqlite'**)  
cur = conn.cursor()  
  
cur.execute(**'SELECT id, sender FROM Senders'**)  
senders = dict()  
**for** message\_row **in** cur :  
 senders[message\_row[0]] = message\_row[1]  
  
cur.execute(**'SELECT id, guid,sender\_id,subject\_id,sent\_at FROM Messages'**)  
messages = dict()  
**for** message\_row **in** cur :  
 messages[message\_row[0]] = (message\_row[1],message\_row[2],message\_row[3],message\_row[4])  
  
print(**"Loaded messages="**,len(messages),**"senders="**,len(senders))  
  
sendorgs = dict()  
**for** (message\_id, message) **in** list(messages.items()):  
 sender = message[1]  
 pieces = senders[sender].split(**"@"**)  
 **if** len(pieces) != 2 : **continue** dns = pieces[1]  
 sendorgs[dns] = sendorgs.get(dns,0) + 1  
  
*# pick the top schools*orgs = sorted(sendorgs, key=sendorgs.get, reverse=**True**)  
orgs = orgs[:10]  
print(**"Top 10 Organizations"**)  
print(orgs)  
  
counts = dict()  
months = list()  
*# cur.execute('SELECT id, guid,sender\_id,subject\_id,sent\_at FROM Messages')***for** (message\_id, message) **in** list(messages.items()):  
 sender = message[1]  
 pieces = senders[sender].split(**"@"**)  
 **if** len(pieces) != 2 : **continue** dns = pieces[1]  
 **if** dns **not in** orgs : **continue** month = message[3][:7]  
 **if** month **not in** months : months.append(month)  
 key = (month, dns)  
 counts[key] = counts.get(key,0) + 1  
  
months.sort()  
*# print counts  
# print months*fhand = open(**'gline.js'**,**'w'**)  
fhand.write(**"gline = [ ['Month'"**)  
**for** org **in** orgs:  
 fhand.write(**",'"**+org+**"'"**)  
fhand.write(**"]"**)  
  
**for** month **in** months:  
 fhand.write(**",\n['"**+month+**"'"**)  
 **for** org **in** orgs:  
 key = (month, org)  
 val = counts.get(key,0)  
 fhand.write(**","**+str(val))  
 fhand.write(**"]"**);  
  
fhand.write(**"\n];\n"**)  
fhand.close()  
  
print(**"Output written to gline.js"**)  
print(**"Open gline.htm to visualize the data"**)

**gword.py**

**import** sqlite3  
**import** time  
**import** zlib  
**import** string  
  
conn = sqlite3.connect(**'index.sqlite'**)  
cur = conn.cursor()  
  
cur.execute(**'SELECT id, subject FROM Subjects'**)  
subjects = dict()  
**for** message\_row **in** cur :  
 subjects[message\_row[0]] = message\_row[1]  
  
*# cur.execute('SELECT id, guid,sender\_id,subject\_id,headers,body FROM Messages')*cur.execute(**'SELECT subject\_id FROM Messages'**)  
counts = dict()  
**for** message\_row **in** cur :  
 text = subjects[message\_row[0]]  
 text = text.translate(str.maketrans(**''**,**''**,string.punctuation))  
 text = text.translate(str.maketrans(**''**,**''**,**'1234567890'**))  
 text = text.strip()  
 text = text.lower()  
 words = text.split()  
 **for** word **in** words:  
 **if** len(word) < 4 : **continue** counts[word] = counts.get(word,0) + 1  
  
x = sorted(counts, key=counts.get, reverse=**True**)  
highest = **None**lowest = **None  
for** k **in** x[:100]:  
 **if** highest **is None or** highest < counts[k] :  
 highest = counts[k]  
 **if** lowest **is None or** lowest > counts[k] :  
 lowest = counts[k]  
print(**'Range of counts:'**,highest,lowest)  
  
*# Spread the font sizes across 20-100 based on the count*bigsize = 80  
smallsize = 20  
  
fhand = open(**'gword.js'**,**'w'**)  
fhand.write(**"gword = ["**)  
first = **True  
for** k **in** x[:100]:  
 **if not** first : fhand.write( **",\n"**)  
 first = **False** size = counts[k]  
 size = (size - lowest) / float(highest - lowest)  
 size = int((size \* bigsize) + smallsize)  
 fhand.write(**"{text: '"**+k+**"', size: "**+str(size)+**"}"**)  
fhand.write( **"\n];\n"**)  
fhand.close()  
  
print(**"Output written to gword.js"**)  
print(**"Open gword.htm in a browser to see the vizualization"**)

**gyear.py**

**import** sqlite3  
**import** time  
**import** urllib.request, urllib.parse, urllib.error  
**import** zlib  
  
conn = sqlite3.connect(**'index.sqlite'**)  
cur = conn.cursor()  
  
cur.execute(**'SELECT id, sender FROM Senders'**)  
senders = dict()  
**for** message\_row **in** cur :  
 senders[message\_row[0]] = message\_row[1]  
  
cur.execute(**'SELECT id, guid,sender\_id,subject\_id,sent\_at FROM Messages'**)  
messages = dict()  
**for** message\_row **in** cur :  
 messages[message\_row[0]] = (message\_row[1],message\_row[2],message\_row[3],message\_row[4])  
  
print(**"Loaded messages="**,len(messages),**"senders="**,len(senders))  
  
sendorgs = dict()  
**for** (message\_id, message) **in** list(messages.items()):  
 sender = message[1]  
 pieces = senders[sender].split(**"@"**)  
 **if** len(pieces) != 2 : **continue** dns = pieces[1]  
 sendorgs[dns] = sendorgs.get(dns,0) + 1  
  
*# pick the top schools*orgs = sorted(sendorgs, key=sendorgs.get, reverse=**True**)  
orgs = orgs[:10]  
print(**"Top 10 Organizations"**)  
print(orgs)  
*# orgs = ['total'] + orgs*counts = dict()  
months = list()  
*# cur.execute('SELECT id, guid,sender\_id,subject\_id,sent\_at FROM Messages')***for** (message\_id, message) **in** list(messages.items()):  
 sender = message[1]  
 pieces = senders[sender].split(**"@"**)  
 **if** len(pieces) != 2 : **continue** dns = pieces[1]  
 **if** dns **not in** orgs : **continue** month = message[3][:4]  
 **if** month **not in** months : months.append(month)  
 key = (month, dns)  
 counts[key] = counts.get(key,0) + 1  
 tkey = (month, **'total'**)  
 counts[tkey] = counts.get(tkey,0) + 1  
   
months.sort()  
*# print counts  
# print months*fhand = open(**'gline.js'**,**'w'**)  
fhand.write(**"gline = [ ['Year'"**)  
**for** org **in** orgs:  
 fhand.write(**",'"**+org+**"'"**)  
fhand.write(**"]"**)  
  
**for** month **in** months[1:-1]:  
 fhand.write(**",\n['"**+month+**"'"**)  
 **for** org **in** orgs:  
 key = (month, org)  
 val = counts.get(key,0)  
 fhand.write(**","**+str(val))  
 fhand.write(**"]"**);  
  
fhand.write(**"\n];\n"**)  
fhand.close()  
  
print(**"Output written to gline.js"**)  
print(**"Open gline.htm to visualize the data"**)

Welcome Jeannie Kim-Schreiber from Capstone: Retrieving, Processing, and Visualizing Data with Python

#### Mailing List Data - Part II

In this assignment you will visualize the mailing list data you have downloaded from http://mbox.dr-chuck.net/ and take some screen shots. Important: You do not have to download all of the data. Gbasic.py must use a count greater than 300. You can run gmane multiple times to download more messages. It is completely acceptable to visualize a small subset of the data in the gbasic screenshot. For the gbasic screenshot show the lines for dump?, loaded messages=, and both sections for Top Email list participants & Email list organizations. \*\*\*For students in mainland China that can't access the Google API for the timeline, take a screenshot of gline.js open in your editor showing the numbers, dates of messages at the top. Please add a note to your assignment.\*\*\*

Don't take off points for little mistakes. If they seem to have done the assignment give them full credit. Feel free to make suggestions if there are small mistakes. Please keep your comments positive and useful. The course staff will assign the last 30% of the grade and it may take a few days to get graded.

**Please Upload Your Submission:**

Top of Form

A screen shot of you running the gbasic.py program to compute basic histogram data on the messages you have retrieved.(Please use PNG or JPG files 1024KB max)

A screen shot of word cloud visualization for the messages you have retrieved.(Please use PNG or JPG files 1024KB max)

A screen shot of time line visualization for the messages you have retrieved showing messages per month.(Please use PNG or JPG files 1024KB max)

Optional Challenge: Change the gline.py program to show the message count by year instead of by month and take a screen shot of the by year visualization. You can switch from a by-month to a by-year visualization by changing only a few lines in gline.py. The puzzle is to figure out the smallest change to accomplish the change. If you do not want to do this optional challenge - just upload the above image a second time.(Please use PNG or JPG files 1024KB max)

Enter optional comments below

Bottom of Form