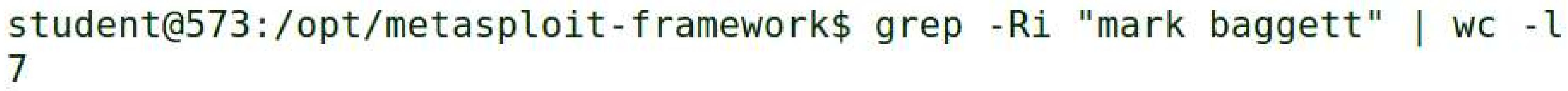
getting the most our of freq and domainstats.py

@markbaggett

`Get-ADUser -Filter "Mark Baggett"| fl -Properties *`

- Mark Baggett
- Penetration Testing and Incident Response Consulting
- Senior SANS Instructor
- Author of SANS SEC573 Automating InfoSec with Python
- Masters in Information Security Engineering
- GSE #15
- DoD Advisor, Former CISO 18 years commercial



```
student@573:/opt/metasploit-framework$ grep -Ri "mark baggett" | wc -l
7
```

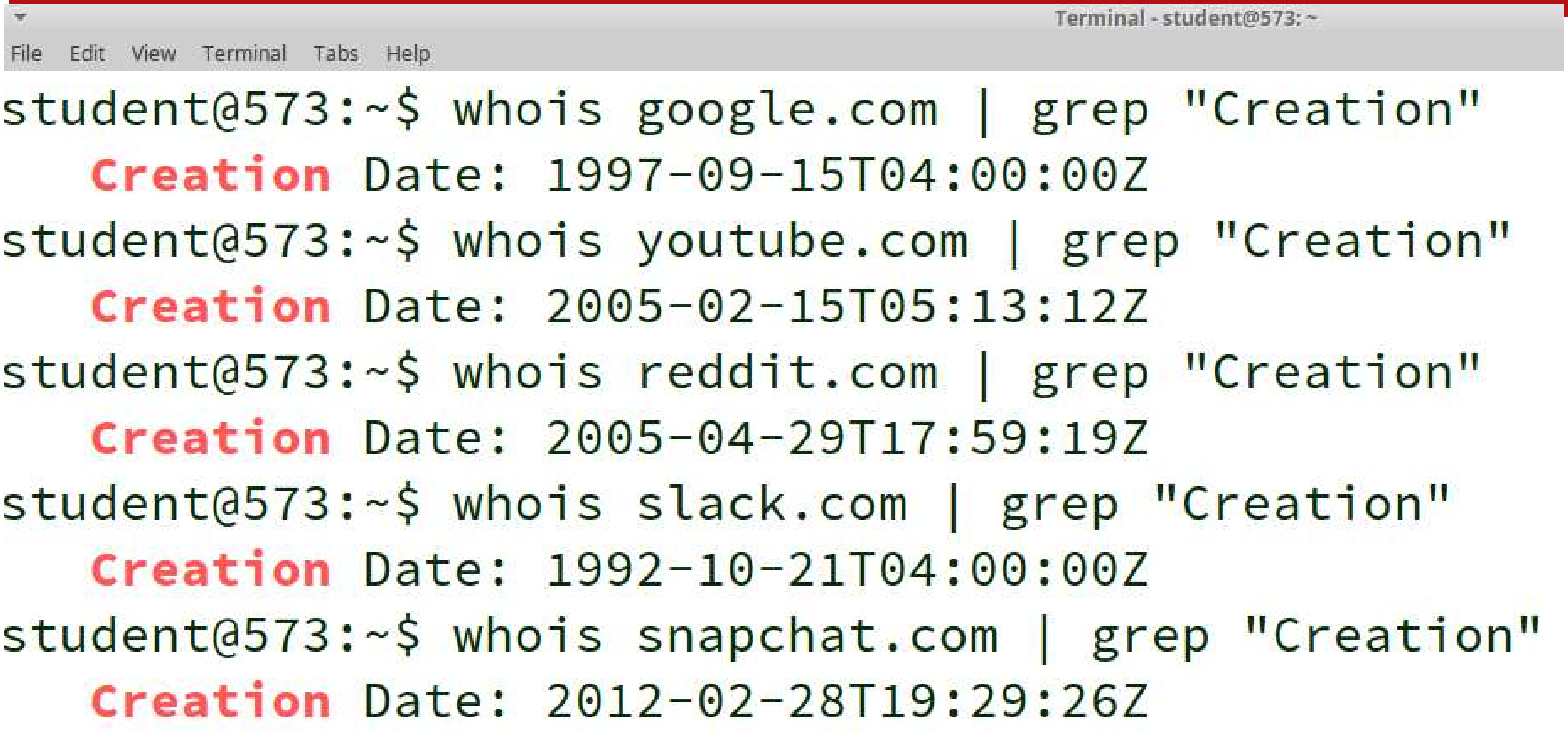
Todays Topic

- Using freq.py and freq_server
  - Help Analysts Using Security Onion to interpret "FREQ SCORES"
  - Help Administrators "tweek" their configurations to do more with the tool that the "out of the box" configuration
- Same thing for domain_stats.py

Intro to Domain Stats

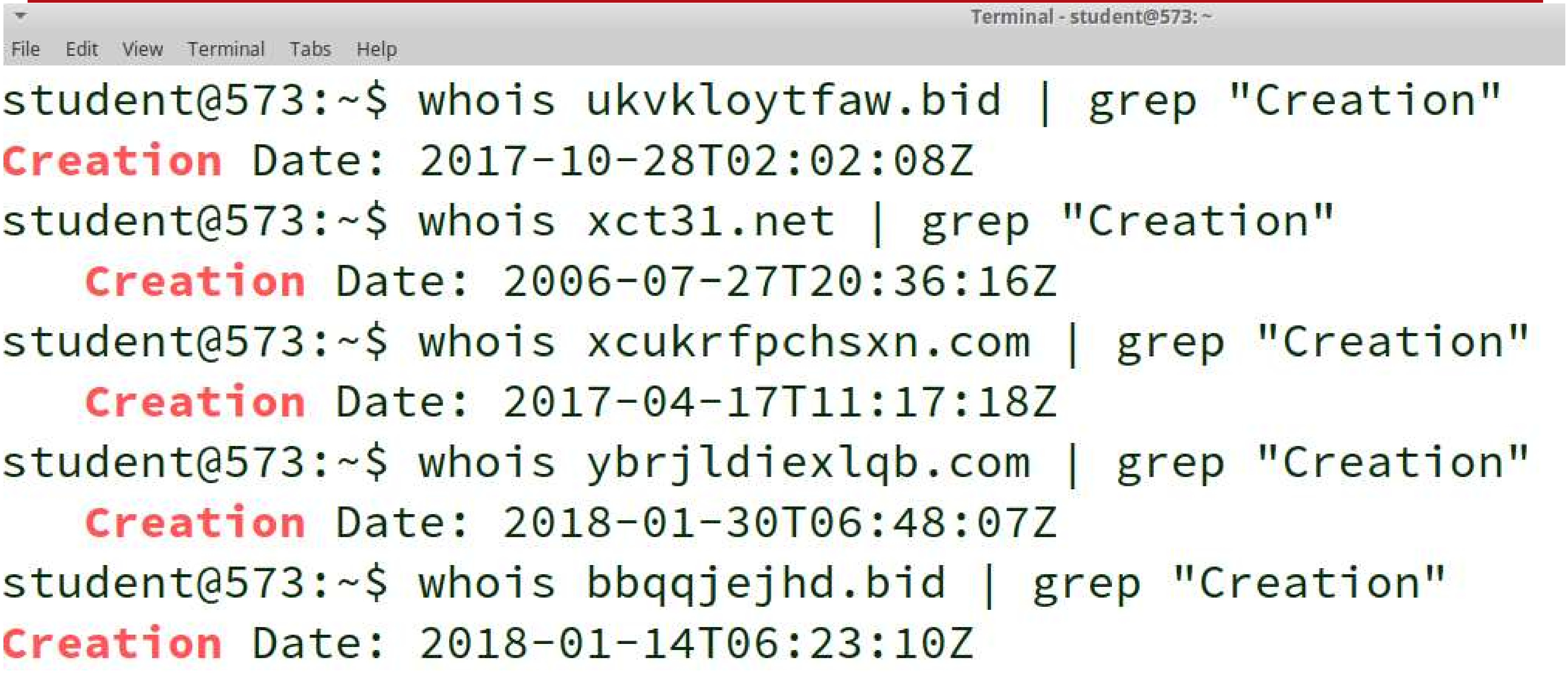
- SEC555 "SEIM with Tactical Analysis" with Justin Henderson and Baby Domains
  - Malware domain are typically much "younger" than legitimate domains!
  - Looking up every domain via whois is slow and can get you blacklisted.
  - Querying whois from a SEIM is non-trivial
- Domain_stats.py was born!
  - Solves problem by caching and prefetching common domains.
  - Provides an easy to use API for SEIM integration

"Normal" Domain Creation Dates



```
student@573:~$ whois google.com | grep "Creation"
   Creation Date: 1997-09-15T04:00:00Z
student@573:~$ whois youtube.com | grep "Creation"
   Creation Date: 2005-02-15T05:13:12Z
student@573:~$ whois reddit.com | grep "Creation"
   Creation Date: 2005-04-29T17:59:19Z
student@573:~$ whois slack.com | grep "Creation"
   Creation Date: 1992-10-21T04:00:00Z
student@573:~$ whois snapchat.com | grep "Creation"
   Creation Date: 2012-02-28T19:29:26Z
```

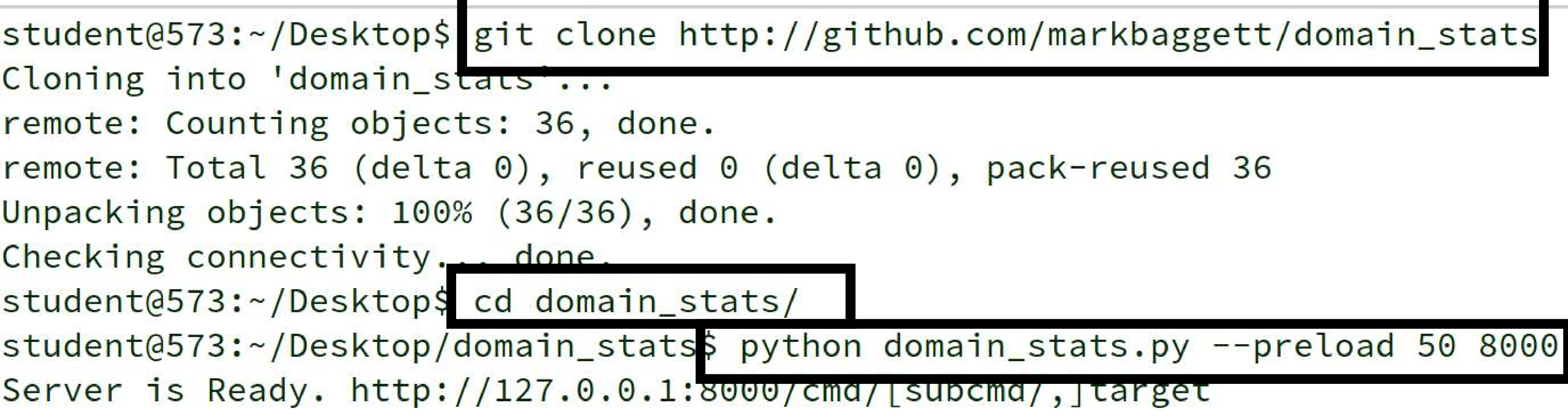
Malware Domain Creation Dates



```
student@573:~$ whois ukvkloytfaw.bid | grep "Creation"
Creation Date: 2017-10-28T02:02:08Z
student@573:~$ whois xct31.net | grep "Creation"
   Creation Date: 2006-07-27T20:36:16Z
student@573:~$ whois xcukrfpchsxn.com | grep "Creation"
   Creation Date: 2017-04-17T11:17:18Z
student@573:~$ whois ybrjldiexlqb.com | grep "Creation"
   Creation Date: 2018-01-30T06:48:07Z
student@573:~$ whois bbqqjejhd.bid | grep "Creation"
Creation Date: 2018-01-14T06:23:10Z
```

Installing and Running Domain_stats

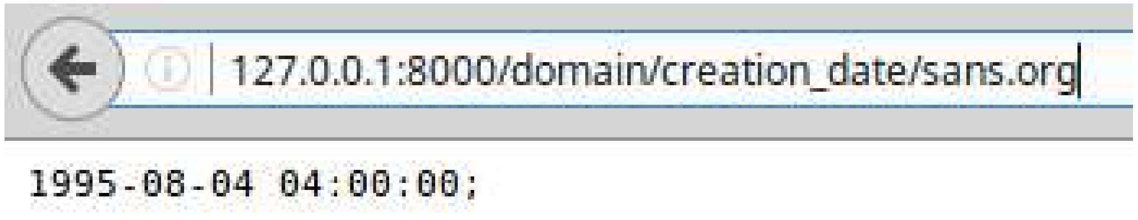
- Run "`python -m pip install python-whois`"



```
student@573:~/Desktop$ git clone http://github.com/markbaggett/domain_stats
Cloning into 'domain_stats'...
remote: Counting objects: 36, done.
remote: Total 36 (delta 0), reused 0 (delta 0), pack-reused 36
Unpacking objects: 100% (36/36), done.
Checking connectivity... done.
student@573:~/Desktop$ cd domain_stats/
student@573:~/Desktop/domain_stats$ python domain_stats.py --preload 50 8000
Server is Ready. http://127.0.0.1:8000/cmd/[subcmd/,]target
```

Query the Creation date from SEIM APIs

- Now you can query whois via an easy web request



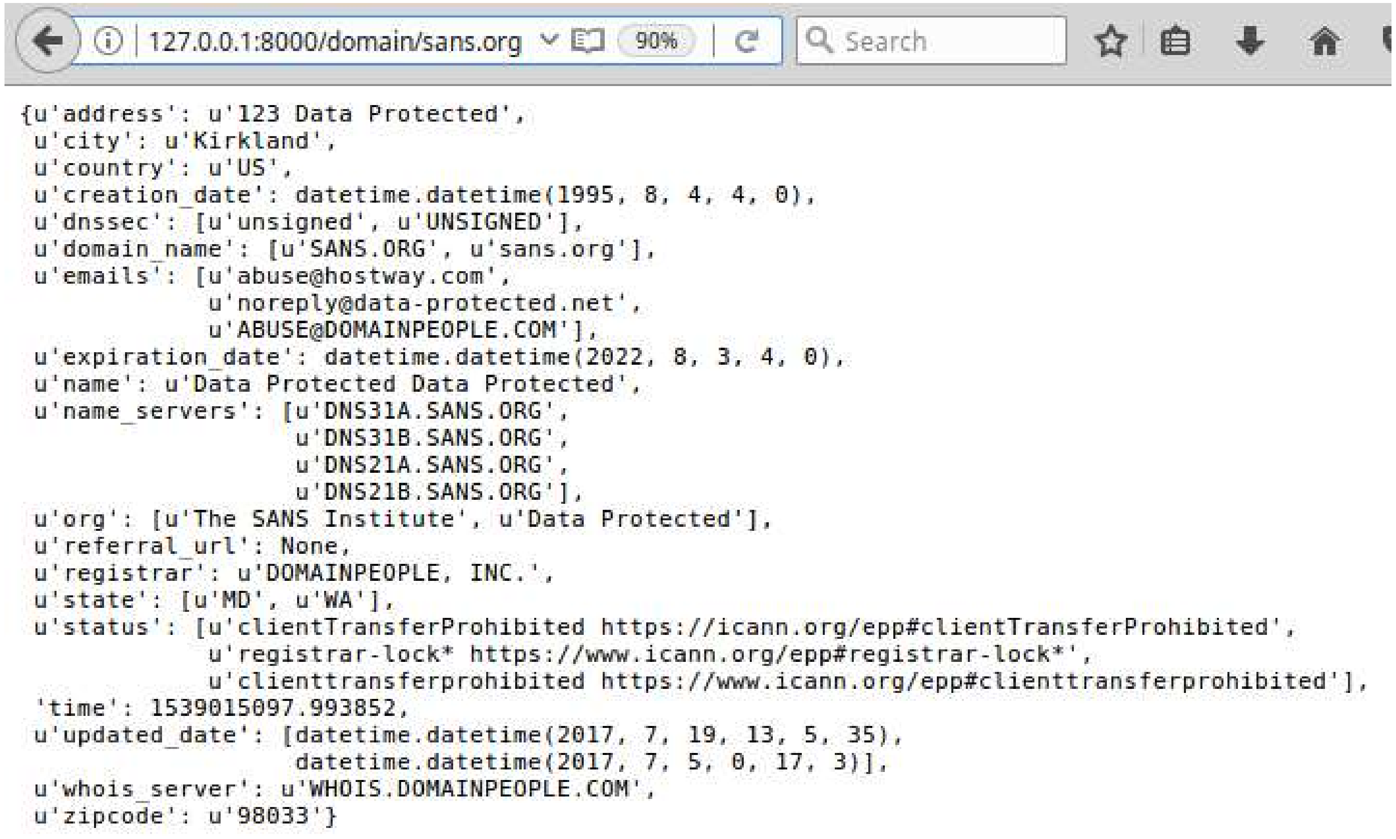
```
127.0.0.1:8000/domain/creation_date/sans.org
```

```
1995-08-04 04:00:00;
```

- Domains are cached locally for speed and minimizing use of whois servers
- So Security Onion can consume this data and present it to the analyst!
- Justin Henderson has config for many SEIM products

Not just CREATION_DATE

- Every field in the whois record is available via the API.
- You can ask for all of it
- You can ask for one field
- You can ask for multiple fields

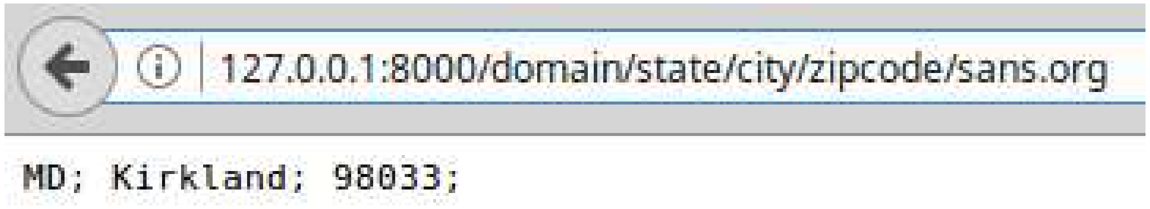


```
{u'address': u'123 Data Protected',
 u'city': u'Kirkland',
 u'country': u'US',
 u'creation_date': datetime.datetime(1995, 8, 4, 4, 0),
 u'dnssec': [u'unsigned', u'UNSIGNED'],
 u'domain_name': [u'SANS.ORG', u'sans.org'],
 u'emails': [u'abuse@hostway.com',
             u'noreply@data-protected.net',
             u'ABUSE@DOMAINPEOPLE.COM'],
 u'expiration_date': datetime.datetime(2022, 8, 3, 4, 0),
 u'name': u'Data Protected Data Protected',
 u'name_servers': [u'DNS31A.SANS.ORG',
                   u'DNS31B.SANS.ORG',
                   u'DNS21A.SANS.ORG',
                   u'DNS21B.SANS.ORG'],
 u'org': [u'The SANS Institute', u'Data Protected'],
 u'referral_url': None,
 u'registrar': u'DOMAINPEOPLE, INC.',
 u'state': [u'MD', u'WA'],
 u'status': [u'clientTransferProhibited https://icann.org/epp#clientTransferProhibited',
             u'registrar-lock* https://www.icann.org/epp#registrar-lock*',
             u'clienttransferprohibited https://www.icann.org/epp#clienttransferprohibited'],
 'time': 1539015097.993852,
 u'updated_date': [datetime.datetime(2017, 7, 19, 13, 5, 35),
                   datetime.datetime(2017, 7, 5, 0, 17, 3)],
 u'whois_server': u'WHOIS.DOMAINPEOPLE.COM',
 u'zipcode': u'98033'}
```

You can ask for more than just CREATION_DATE

- Full API documentation on http://github.com/markbaggett/domain_stats

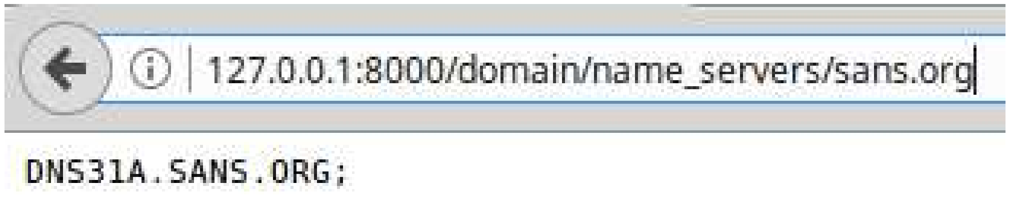
Query Multiple Fields



```
127.0.0.1:8000/domain/state/city/zipcode/sans.org
```

```
MD; Kirkland; 98033;
```

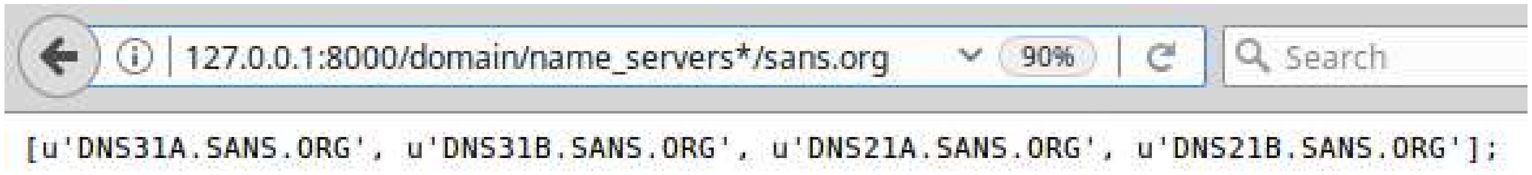
or one



```
127.0.0.1:8000/domain/name_servers/sans.org
```

```
DNS31A.SANS.ORG;
```

* to access multi-value fields



```
127.0.0.1:8000/domain/name_servers*/sans.org
```

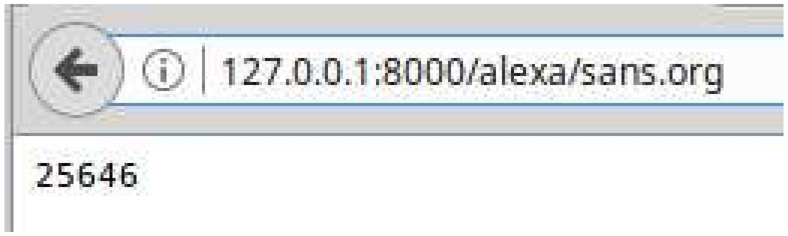
```
[u'DNS31A.SANS.ORG', u'DNS31B.SANS.ORG', u'DNS21A.SANS.ORG', u'DNS21B.SANS.ORG'];
```

Alexa Ranking of Domains

- Use DOMAIN_STATS to see what the Alexa rank of a domain is

```
python domain_stats.py -a top-1m.csv --preload 0 8000
```

- As soon as you give DOMAIN_STATS an Alexa file it will attempt to preload its cache with most common domains
- Controllable with --preload



```
127.0.0.1:8000/alexa/sans.org

25646
```

- Update Top 1M at http://s3-us-west-1.amazonaws.com/umbrella-static/index.html

BETA TESTING A NEW FEATURE

- Punycode/IDN Domain resolution:

```
$ curl http://127.0.0.1:8000/punycode/xn--n28h
😉
$ curl http://127.0.0.1:8000/punycode/xn--g6h8599noea
👁❤🐍
```

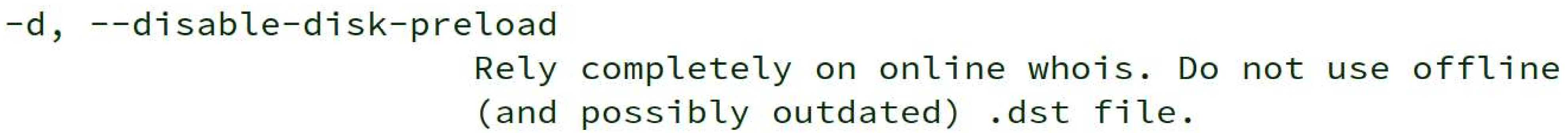
- Feature requests by N7FAA52318.
- Implemented but not committed to main branch
- If you are interested in this feature I am seeking testers.

Performance Over Accuracy: Understanding the Cache

- By Default DOMAIN_STATS preloads the top 1000 most frequently used domains from disk cache!
  - This is GREAT!! For CREATION_DATE which doesn't change
  - Undesirable if the company changes their DNS servers
- Items stay in cache for as long as you are querying that domain at least once a week
- Run "update_diskcache.py" at an interval you are happy with to make sure you have the latest data
- Requires that you restart your domain_stats server.

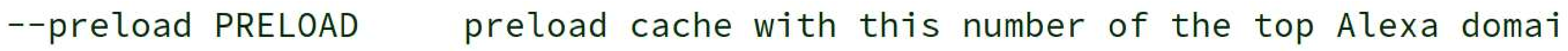
You: "Couldn't you do XYZ"   Me: "Yes, but performance"

- You have control of caching options on the CLI
- You can disable local disk cache of top 1000



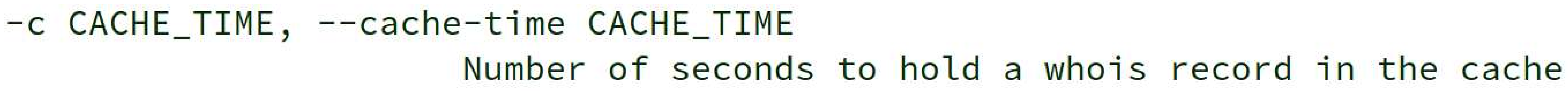
```
-d, --disable-disk-preload
                      Rely completely on online whois. Do not use offline
                      (and possibly outdated) .dst file.
```

- You can disable preloading common domains in background



```
--preload PRELOAD     preload cache with this number of the top Alexa domai
```

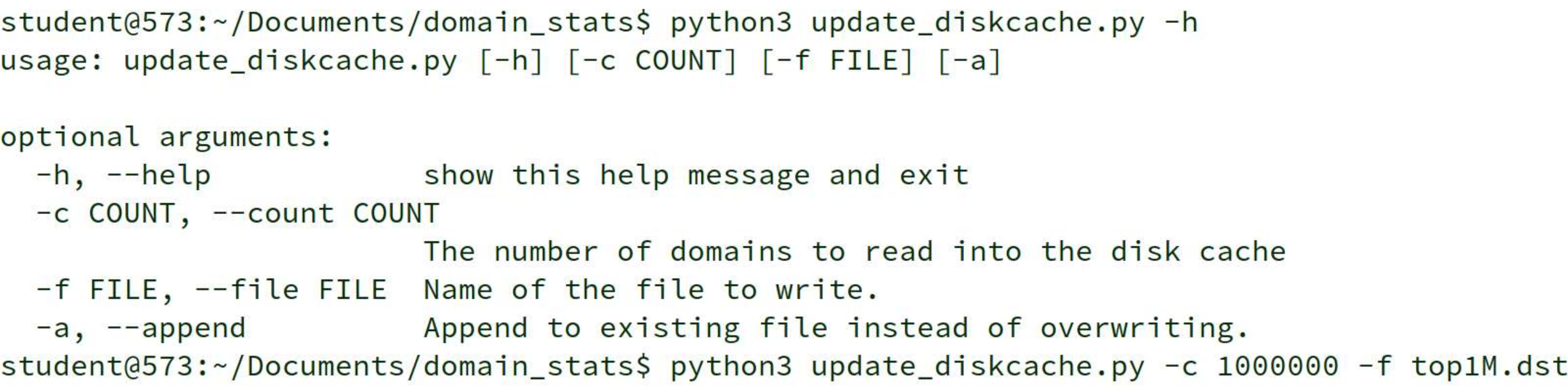
- You can control how long unused items are held in cache



```
-c CACHE_TIME, --cache-time CACHE_TIME
                      Number of seconds to hold a whois record in the cache
```

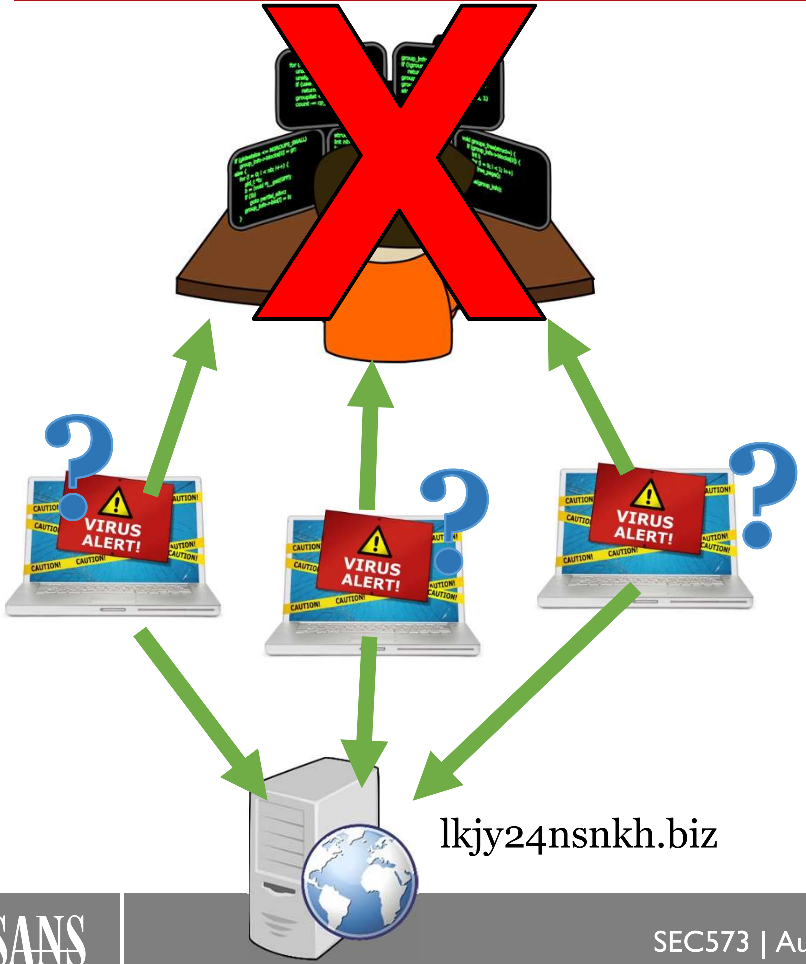
GO ALL IN!!

- If you only need creation_date then you don't need the online whois. PUMP UP THE DISK CACHE BABY!



```
student@573:~/Documents/domain_stats$ python3 update_diskcache.py -h
usage: update_diskcache.py [-h] [-c COUNT] [-f FILE] [-a]

optional arguments:
  -h, --help            show this help message and exit
  -c COUNT, --count COUNT
                        The number of domains to read into the disk cache
  -f FILE, --file FILE  Name of the file to write.
  -a, --append          Append to existing file instead of overwriting.
student@573:~/Documents/domain_stats$ python3 update_diskcache.py -c 1000000 -f top1M.dst
```

What are DGAs



- Imagine all the attacker bots are talking to mybotnet.com

- Law enforcement takes down mybotnet.com
- Network defenders block mybotnet.com

- Attackers would like bots to reconnect to new domain!

- New domain needs to be obscure enough to be available for purchase by the attacker. (ie , not already be owned)
- Need an almost infinite number of possibilities because defenders might keep blocking their domains!

- Use "Domain generation algorithms" to automatically choose new domains in a way that is predictable to the attacker.
- These domains typically look like random strings of characters
- Found in SSL certificates, DNS logs and HTTP headers more.

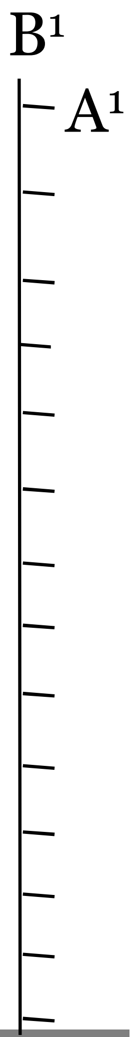
Intro to Freq.py and Freq_server

- SEC511 "Continuous Monitoring and Security Operations" with Seth Misenar

```
C:>echo "reddit.com" | ent.exe
Entropy = 3.640224 bits per byte.

C:\>echo "youtube.com" | ent.exe
Entropy = 3.625000 bits per byte.
```

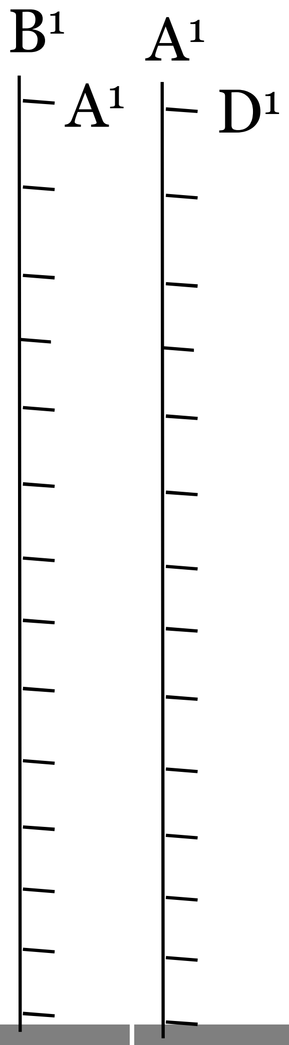
```
C:\>echo "ukvklo.bid" | ent.exe
Entropy = 3.640224 bits per byte.

C:\>echo "ybrjl.com" | ent.exe
Entropy = 3.664498 bits per byte.
```

- freq.py and freq_server.py were born!
  - Gives reliable "scores" to identify DGA domains
  - How does it work? Lets look.

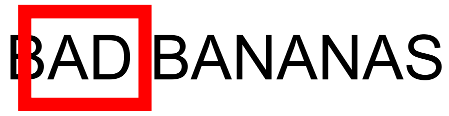


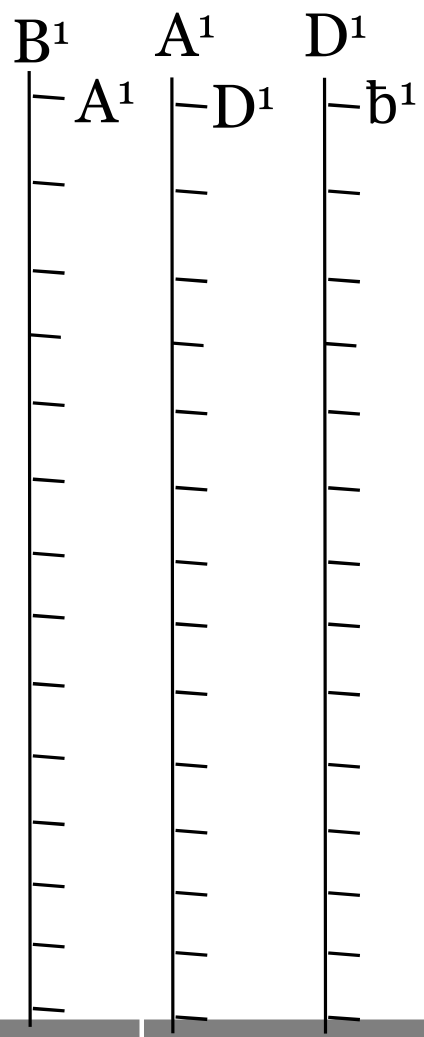
We analyze streams of legitimate text as character pairs to build a frequency table

$\boxed{\text{BA}}$D BANANAS



We analyze each pair counting the appearance of first and second characters.

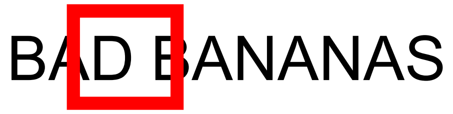


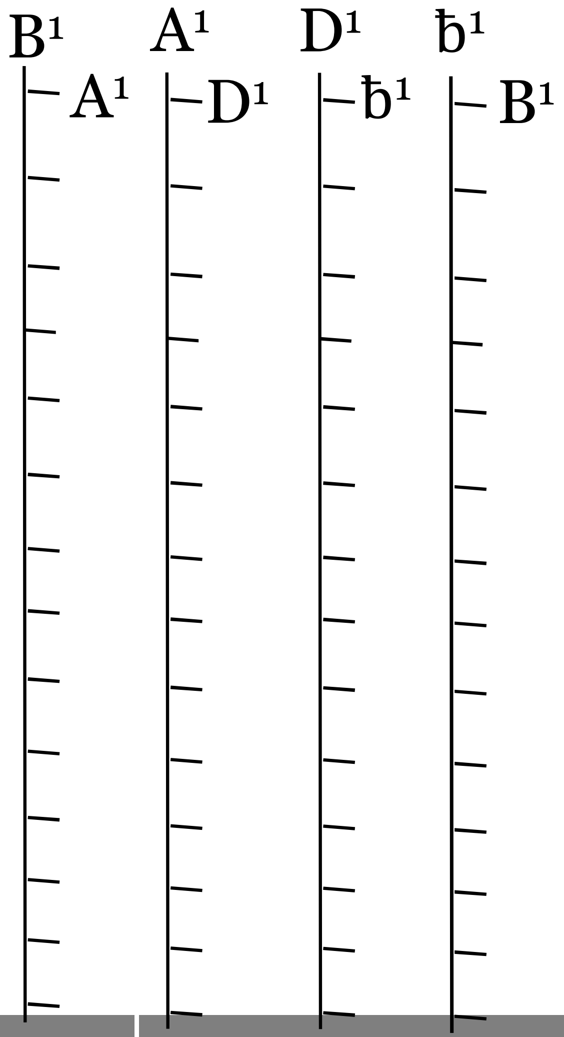


You may want to ignore some characters such as space, semicolon and other characters in the data you analyze if they do not commonly appear in the data you are searching. This is controllable by specifying the --ignorechars command line option.

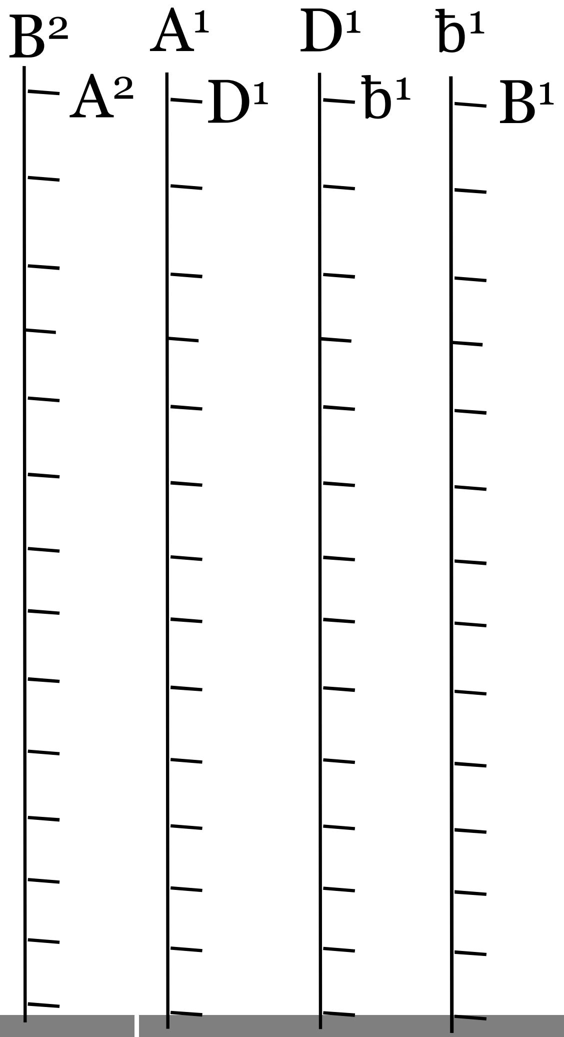
The original freq.py ignored these characters while building the tables.

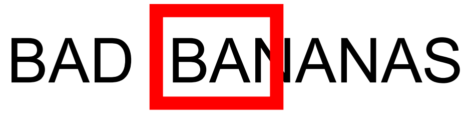
The new freq.py tallys every character are ignored specified characters during calculations. This means one table can work in all multiple situations.

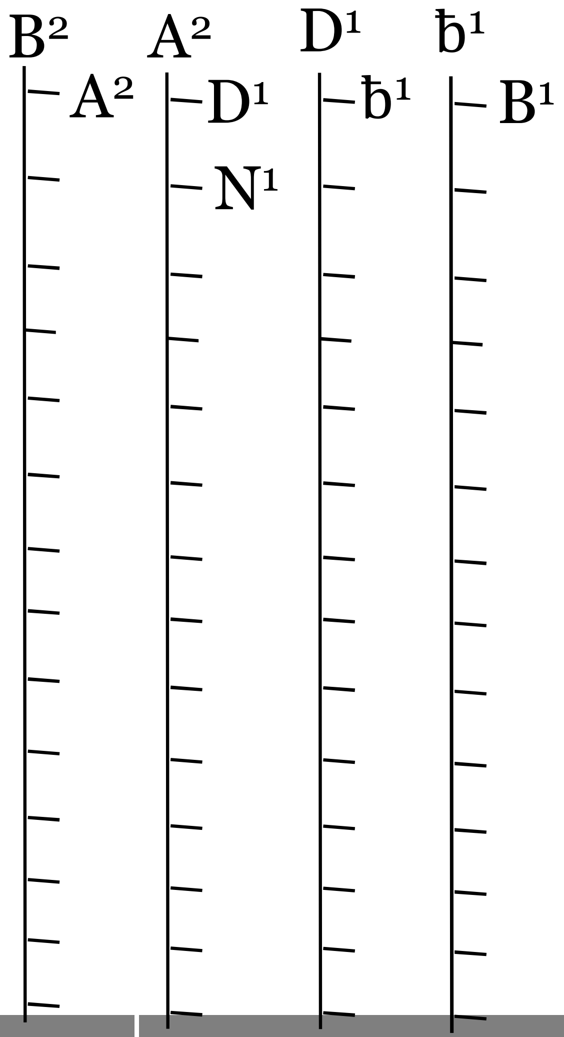


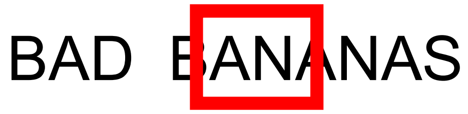


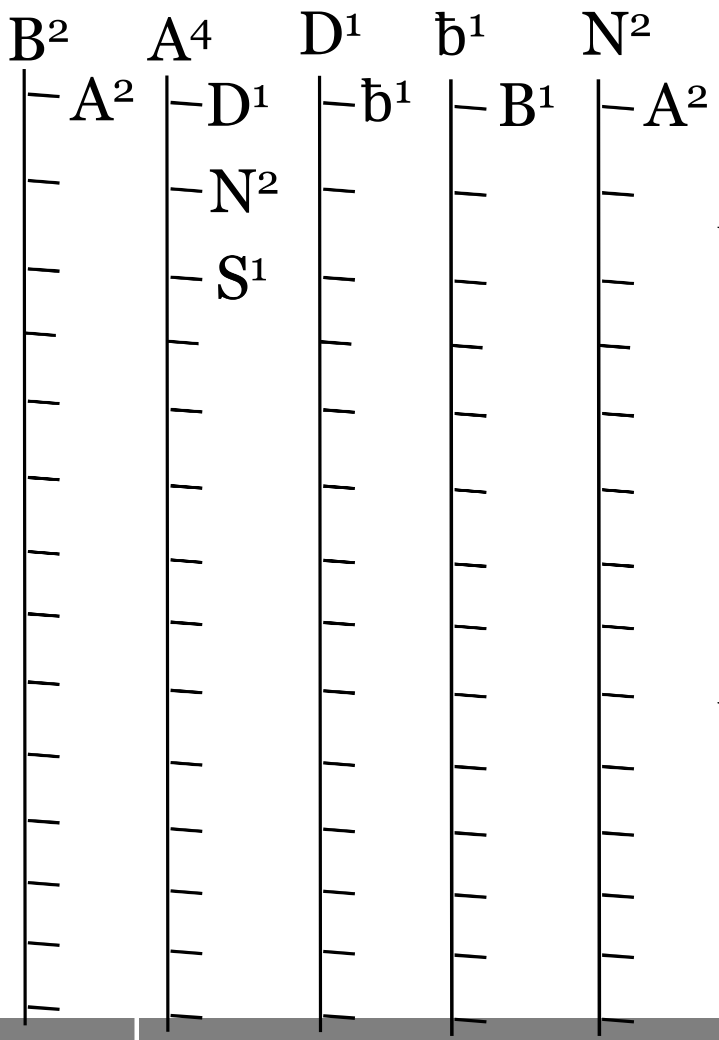
BAD BANANAS


B²
A²
A¹
D¹
D¹
ƀ¹
ƀ¹
B¹


BAD BANANAS


B² A² D¹ ƀ¹
A²
D¹
N¹
ƀ¹
B¹


BAD BANANAS



When we finish we get a table that looks like this.

For this small data set:

- 100% chance that B is followed by A
- 100% chance that N is followed by A
- 50% chance that A is followed by N
- 25% chance that A is followed by D
- 25% chance that A is followed by S

We can analyze large volumes of data to build probabilities of normal text.

BAD BANANAS

Two methods of measuring "Normal" text

- Method 1 - "Average Probability"
  - Built into Original freq.py
  - Based on average probability of pairs

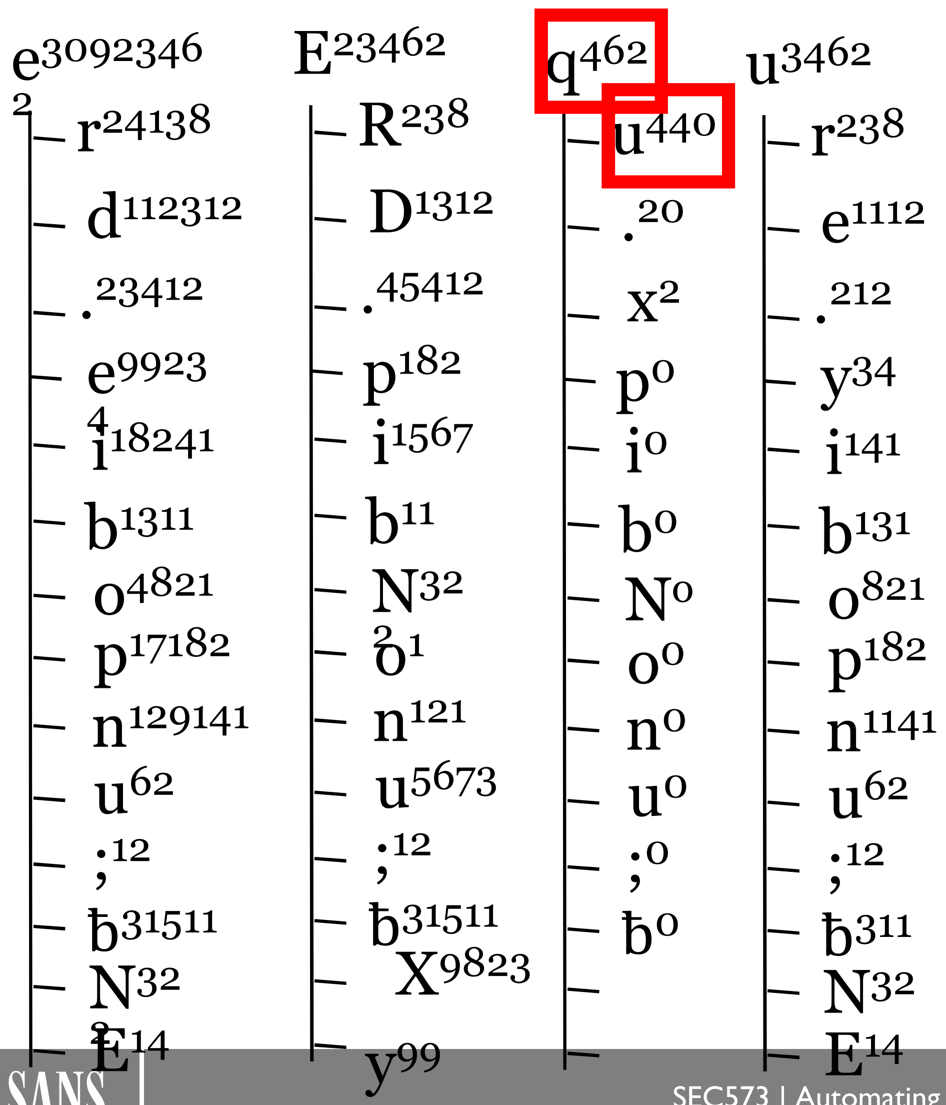
- Method 2 - "Word Probability"
  - Only available in latest update
  - Based on probability of the entire word

$e^{30923462}$	E^{23462}	q^{462}	u^{3462}
r^{24138}	R^{238}	u^{440}	r^{238}
d^{112312}	D^{1312}	$.^{20}$	e^{1112}
$.^{23412}$	$.^{45412}$	x^{2}	$.^{212}$
e^{99234}	p^{182}	p^{0}	y^{34}
i^{18241}	i^{1567}	i^{0}	i^{141}
b^{1311}	b^{11}	b^{0}	b^{131}
o^{4821}	N^{322}	N^{0}	o^{821}
p^{17182}	o^{1}	o^{0}	p^{182}
n^{129141}	n^{121}	n^{0}	n^{1141}
u^{62}	u^{5673}	u^{0}	u^{62}
$;^{12}$	$;^{12}$	$;^{0}$	$;^{12}$
ƀ^{31511}	ƀ^{31511}	ƀ^{0}	ƀ^{311}
N^{322}	X^{9823}		N^{32}
E^{14}	y^{99}		E^{14}

Here is what a portion of a complete table might look like.

Now lets measure probability of:

queen



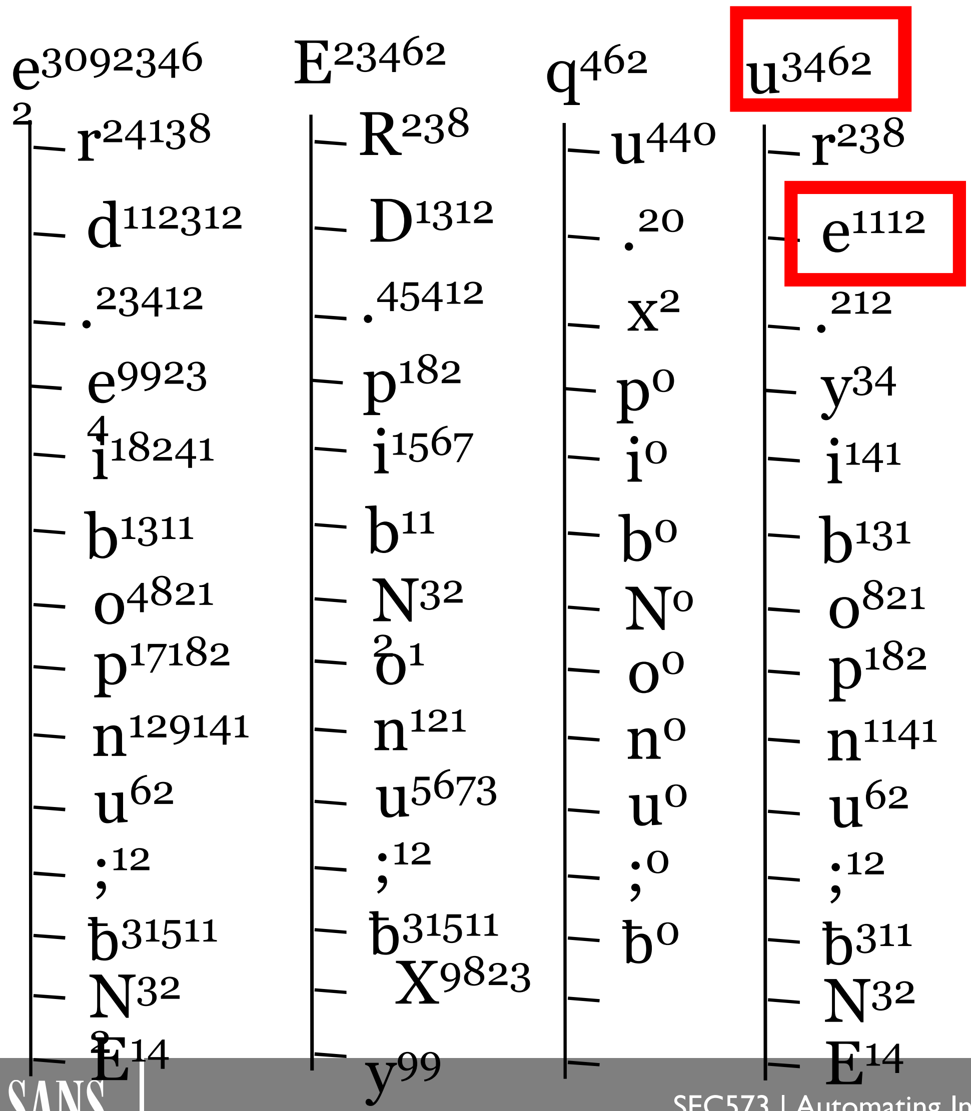
METHOD 1:

Here is what a portion of a complete table might look like.

Now lets measure probability of:

queen

qu = 440/462 = 0.952 or 95%



METHOD 1:

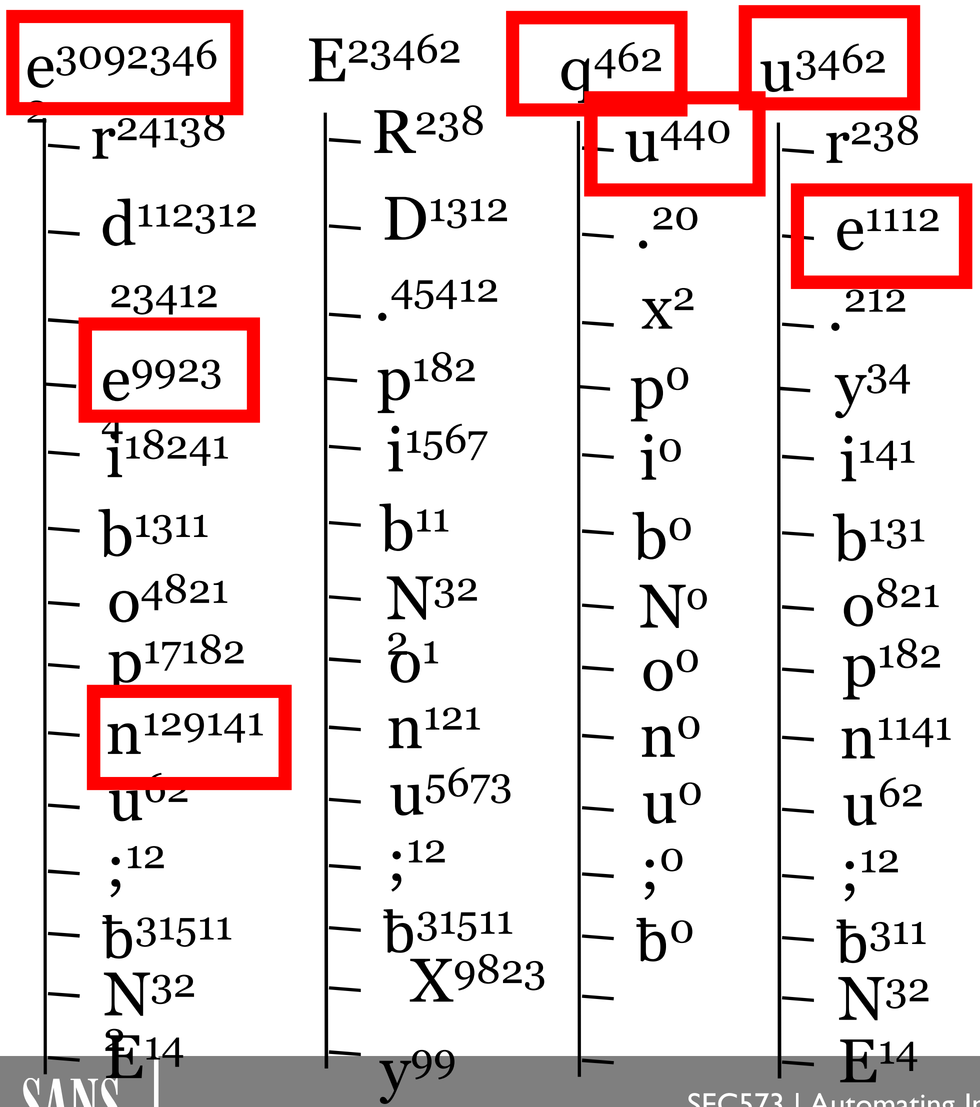
Here is what a portion of a complete table might look like.

Now lets measure probability of:

queen

qu = 440/462 = 0.952 or 95%

ue = 1112/3462 = 32%



Here is what a portion of a complete table might look like.

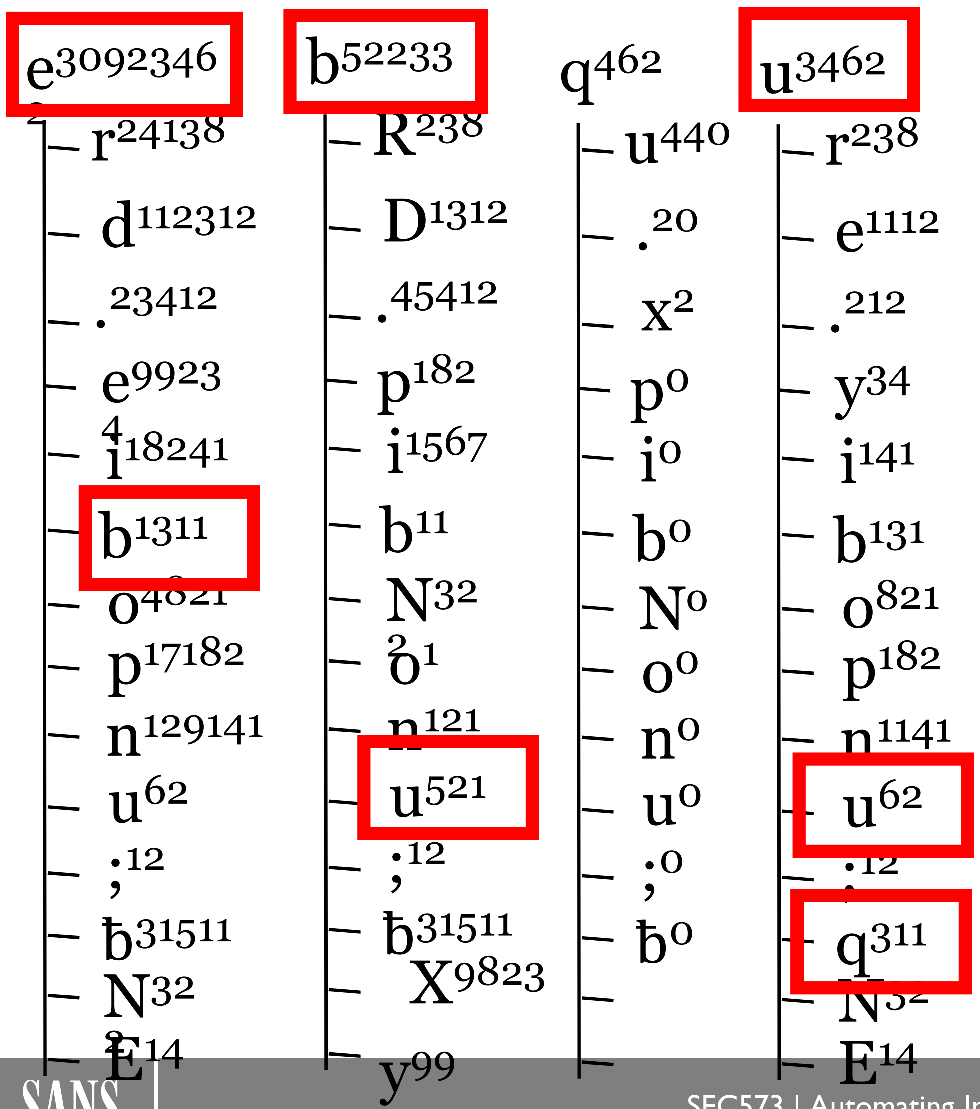
Now lets measure probability of:

queen

qu = 440/462 = 0.952 or 95%
ue = 1112/3462 = 32%
ee = 24126/30923462 = 1%
en = 129141/30923462 = 3%

Average Probability = 42%*

*put down calculators. All numbers are fictional examples



Here is what a portion of a complete table might look like.

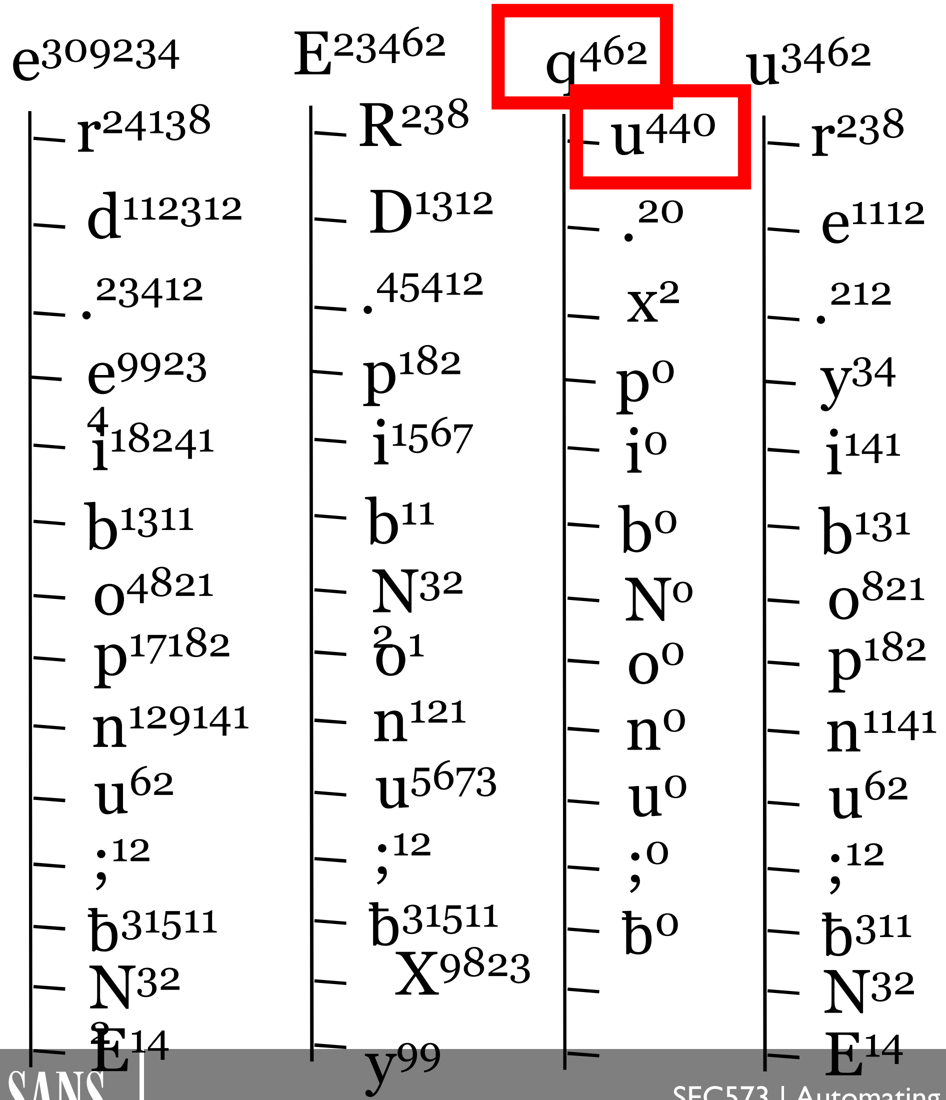
Now lets measure probability of:

ebuuq

eb = 1311/3092346 = .09%
bu = 52233/521 = 5%
uu = 62/3462 = 3%
uq = 331/3462 = 0.2%

Average Probability = 2.4%*

*put down calculators. All numbers are fictional examples



METHOD 2:

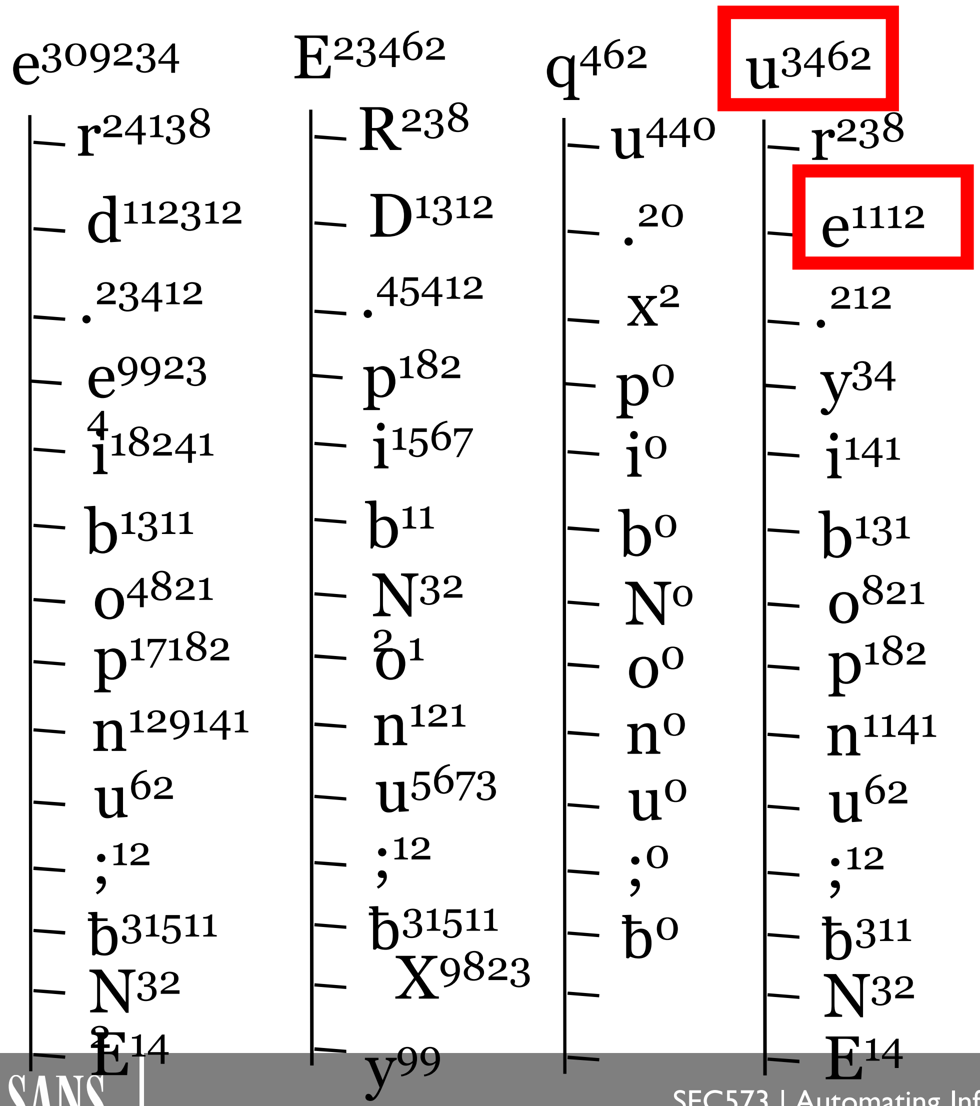
Here is what a portion of a complete table might look like.

Now lets measure probability of:

queen

qu , first = 462, sec = 440

Total first = 462
Total second = 440



METHOD 1:

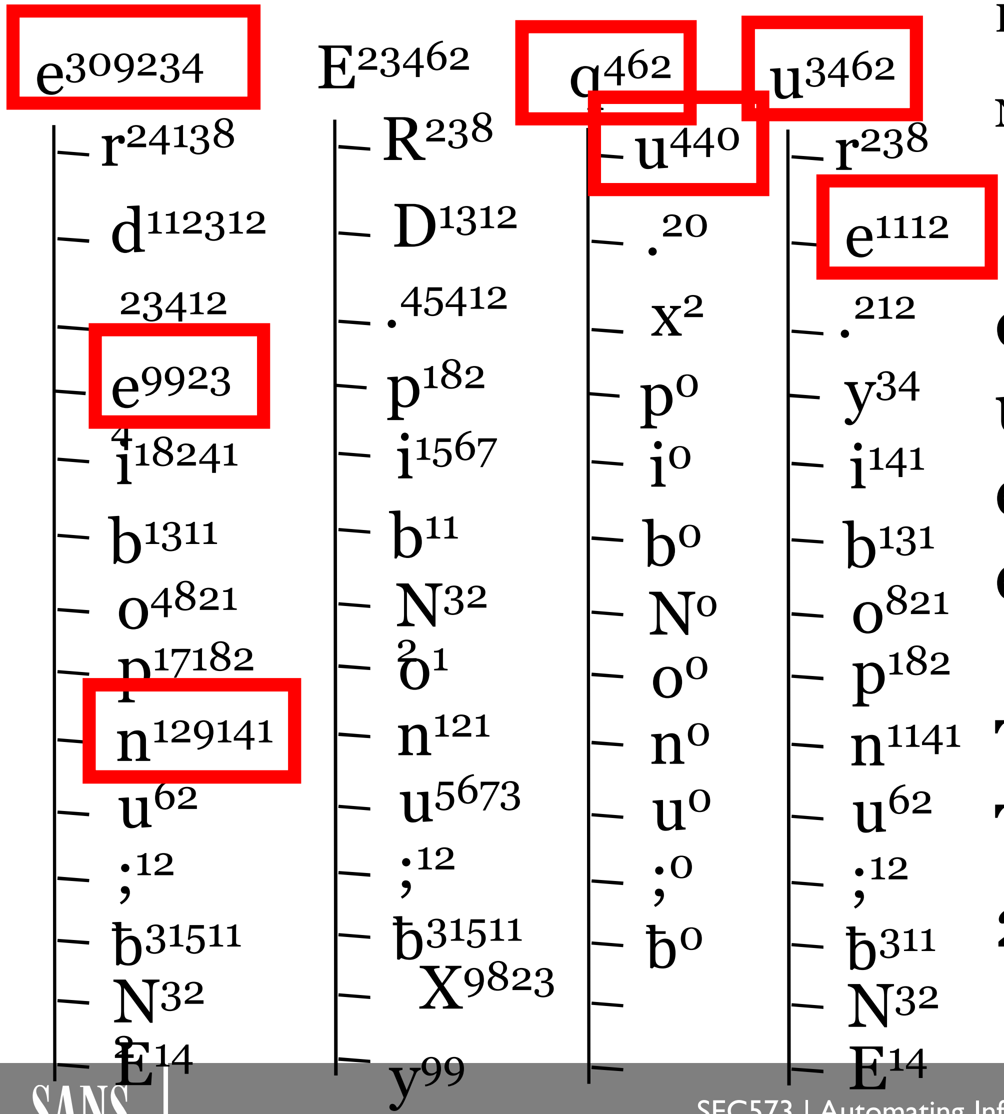
Here is what a portion of a complete table might look like.

Now lets measure probability of:

queen

qu , first = 462, sec = 440
ue, first = 3462, sec = 1112

Total first = 3942
Total second = 1552



Here is what a portion of a complete table might look like.

Now lets measure probability of:

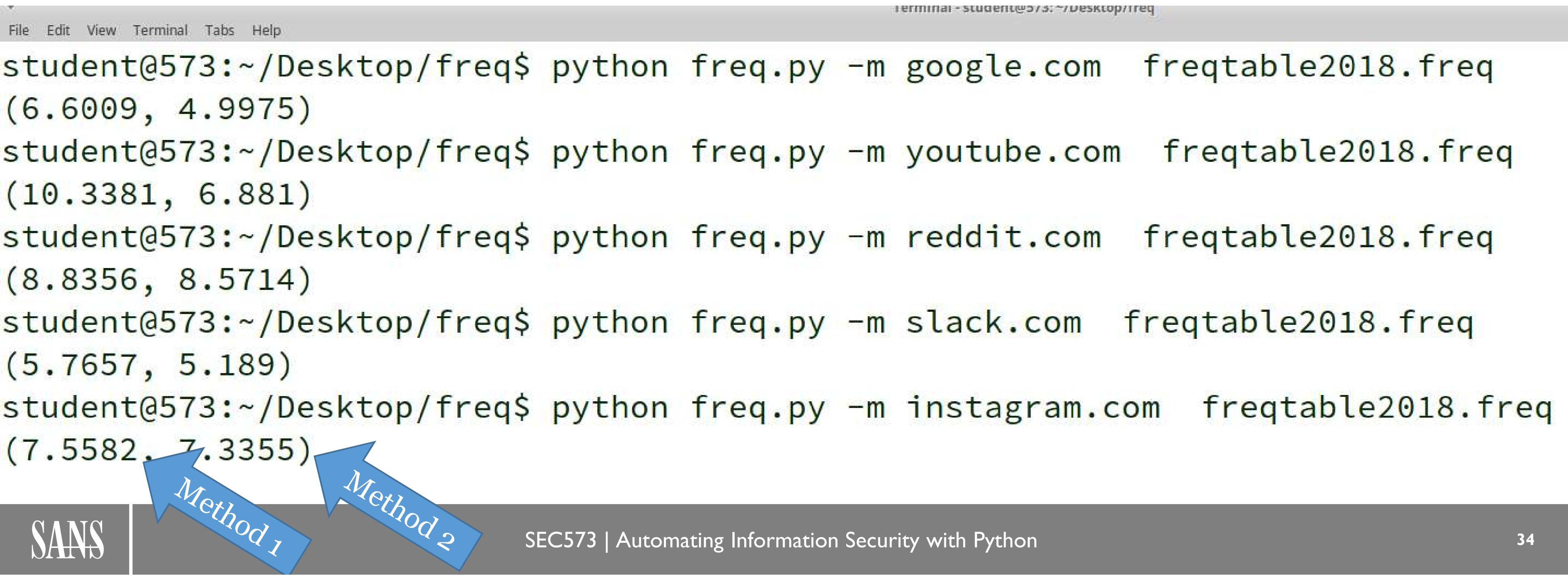
queen

qu , first = 462, sec = 440
ue, first = 3462, sec = 1112
ee, first=309234,sec=99234
en, first=309234,sec=129141

Total first = 4080930
Total second = 2299270
229927/4080930 = 6%

Where do "legit" domains score?

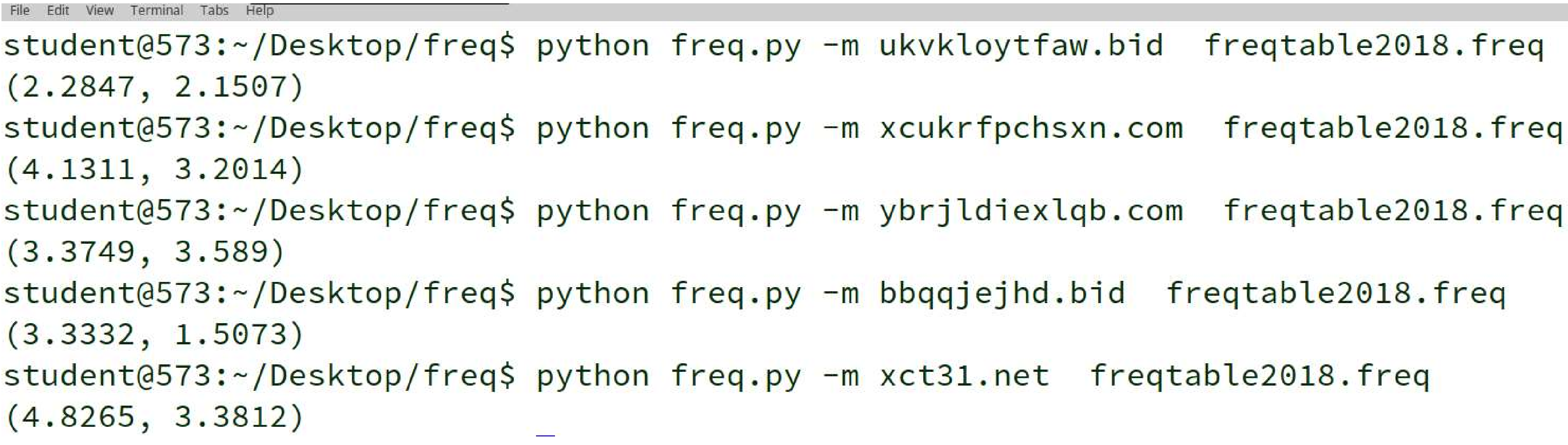
- Method 1 scores are greater than 5
- Method 2 scores are greater than 4


Terminal - student@573: ~/Desktop/freq
File Edit View Terminal Tabs Help
Method 1
Method 2

```
student@573:~/Desktop/freq$ python freq.py -m google.com  freqtable2018.freq
(6.6009, 4.9975)
student@573:~/Desktop/freq$ python freq.py -m youtube.com  freqtable2018.freq
(10.3381, 6.881)
student@573:~/Desktop/freq$ python freq.py -m reddit.com  freqtable2018.freq
(8.8356, 8.5714)
student@573:~/Desktop/freq$ python freq.py -m slack.com  freqtable2018.freq
(5.7657, 5.189)
student@573:~/Desktop/freq$ python freq.py -m instagram.com  freqtable2018.freq
(7.5582, 7.3355)
```

Scores for malicious domains?

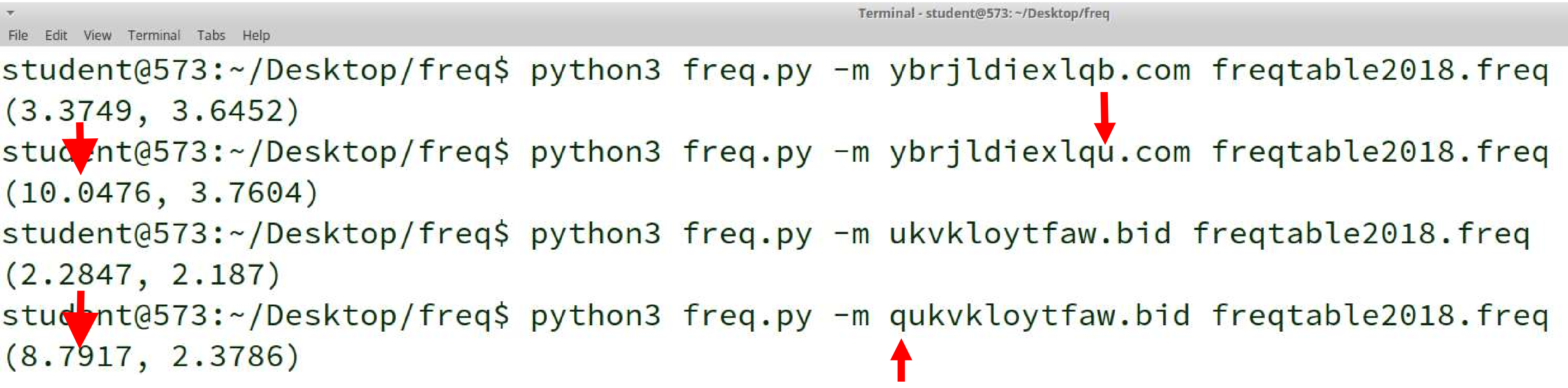
- Method 1 < 5
- Method2 < 4!



```
student@573:~/Desktop/freq$ python freq.py -m ukvkloytfaw.bid  freqtable2018.freq
(2.2847, 2.1507)
student@573:~/Desktop/freq$ python freq.py -m xcukrfpchsxn.com  freqtable2018.freq
(4.1311, 3.2014)
student@573:~/Desktop/freq$ python freq.py -m ybrjldiexlqb.com  freqtable2018.freq
(3.3749, 3.589)
student@573:~/Desktop/freq$ python freq.py -m bbqqjejhd.bid  freqtable2018.freq
(3.3332, 1.5073)
student@573:~/Desktop/freq$ python freq.py -m xct31.net  freqtable2018.freq
(4.8265, 3.3812)
```

Why is Method 2 better?

- A single "qu" pair can make the un-probable probable.

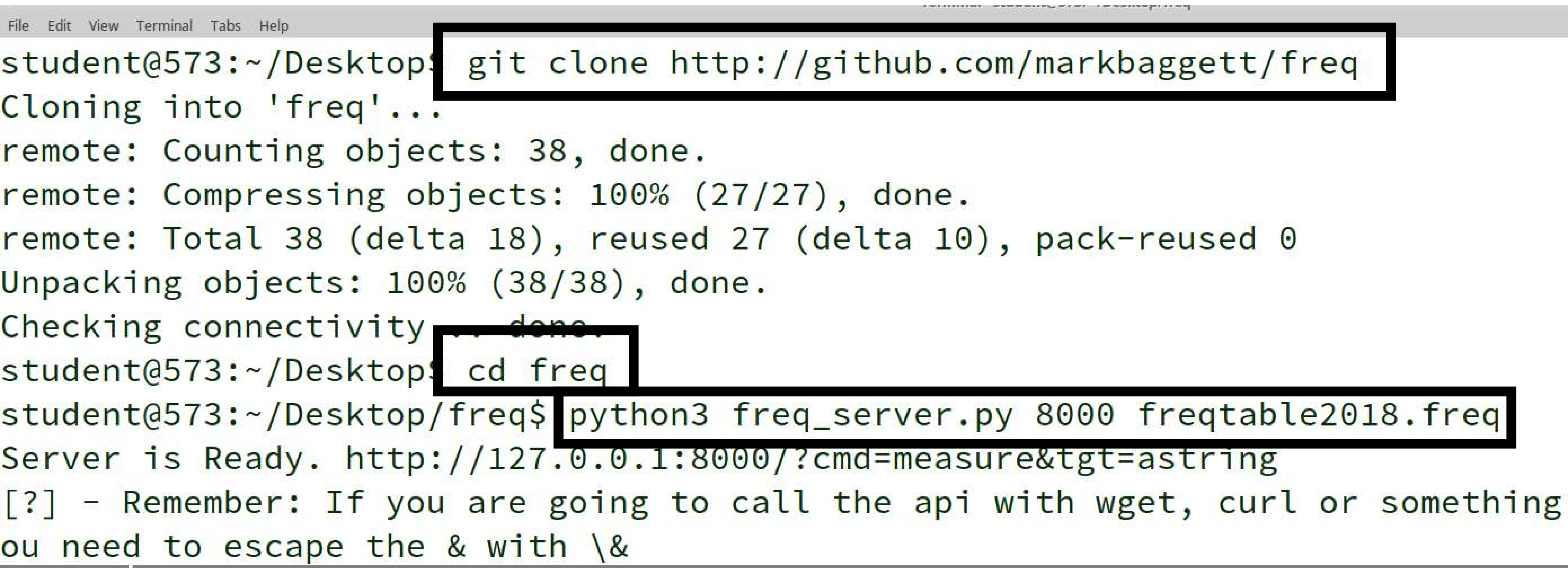

Terminal - student@573: ~/Desktop/freq
File Edit View Terminal Tabs Help

```
student@573:~/Desktop/freq$ python3 freq.py -m ybrjldiexlqb.com freqtable2018.freq
(3.3749, 3.6452)
student@573:~/Desktop/freq$ python3 freq.py -m ybrjldiexlqu.com freqtable2018.freq
(10.0476, 3.7604)
student@573:~/Desktop/freq$ python3 freq.py -m ukvkloytfaw.bid freqtable2018.freq
(2.2847, 2.187)
student@573:~/Desktop/freq$ python3 freq.py -m qukvkloytfaw.bid freqtable2018.freq
(8.7917, 2.3786)
```

- Letters are weighted base on how common they are in normal text. So "rstlne" have more effect on score than "qxz"

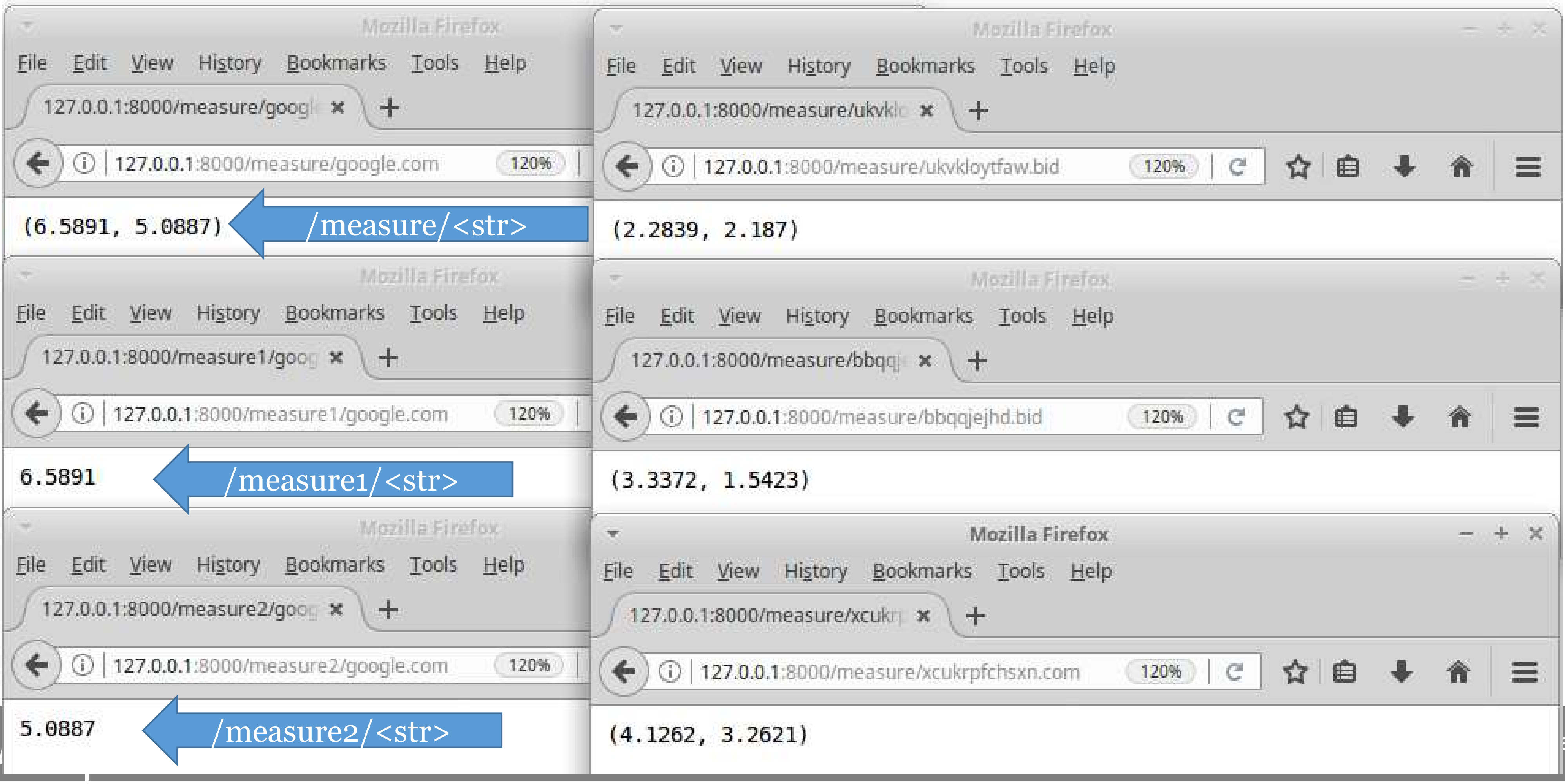
Installing and starting freq_server.py

- No module dependencies. Just download and execute!



```
student@573:~/Desktop$ git clone http://github.com/markbaggett/freq
Cloning into 'freq'...
remote: Counting objects: 38, done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 38 (delta 18), reused 27 (delta 10), pack-reused 0
Unpacking objects: 100% (38/38), done.
Checking connectivity... done.
student@573:~/Desktop$ cd freq
student@573:~/Desktop/freq$ python3 freq_server.py 8000 freqtable2018.freq
Server is Ready. http://127.0.0.1:8000/?cmd=measure&tgt=astring
[?] - Remember: If you are going to call the api with wget, curl or something
ou need to escape the & with \&
```

SEIM can Access the Server with /measure[1,2]/domain



127.0.0.1:8000/measure/google.com

```
(6.5891, 5.0887)
```

/measure/<str>

127.0.0.1:8000/measure1/google.com

```
6.5891
```

/measure1/<str>

127.0.0.1:8000/measure2/google.com

```
5.0887
```

/measure2/<str>

127.0.0.1:8000/measure/ukvkloytfaw.bid

```
(2.2839, 2.187)
```

127.0.0.1:8000/measure/bbqqjejhd.bid

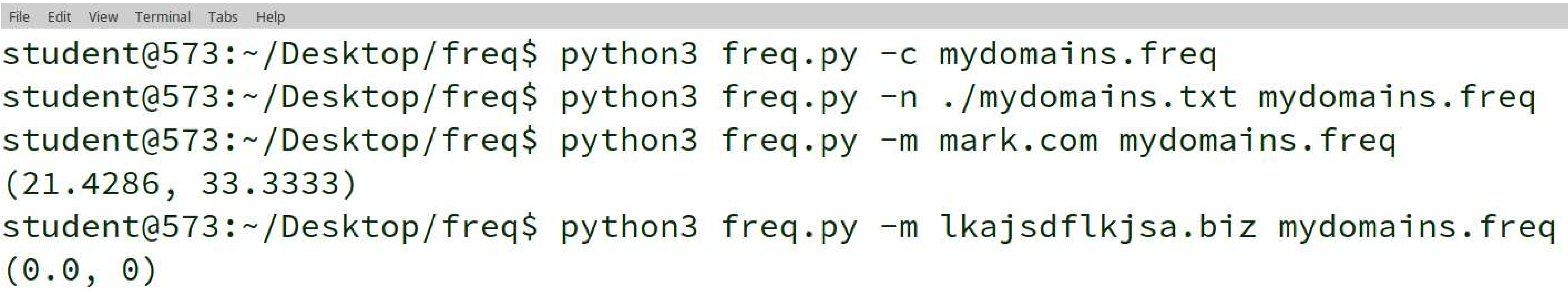
```
(3.3372, 1.5423)
```

127.0.0.1:8000/measure/xcukrpfchsxn.com

```
(4.1262, 3.2621)
```

Freq.py Makes freq_server.py Much Better!

- Customize your frequency tables for your specific environment!



```
student@573:~/Desktop/freq$ python3 freq.py -c mydomains.freq
student@573:~/Desktop/freq$ python3 freq.py -n ./mydomains.txt mydomains.freq
student@573:~/Desktop/freq$ python3 freq.py -m mark.com mydomains.freq
(21.4286, 33.3333)
student@573:~/Desktop/freq$ python3 freq.py -m lkajsdflkjsa.biz mydomains.freq
(0.0, 0)
```

- Build new frequency tables
- Adjust values by adding domains to freqtable2018.freq
- Measure domains from the CLI and other tools

Build Special Purpose frequency tables

1) Use Powershell to create a list of all files on a file system

```
PS C:\> gci -recurse | select -Property Name | Out-File -FilePath c:\allfiles.txt -Encoding ascii
```

2) Create a custom frequency table for filenames

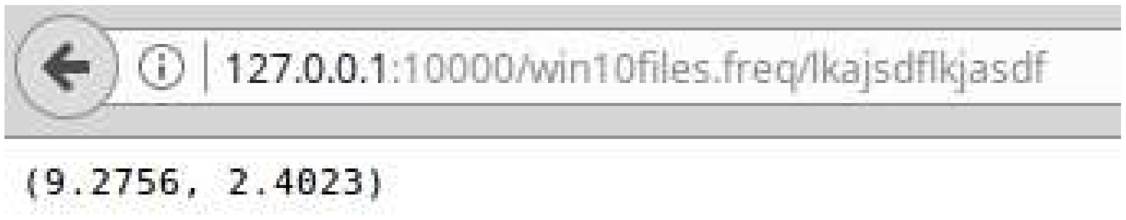
```
$ python3 freq.py -c win10files.freq
$ python3 freq.py -f ~/Desktop/allfiles.txt win10files.freq
$ python3 freq.py -m cmd.exe win10files.freq
(7.4876, 4.8509)
$ python3 freq.py -m aslkjfl.exe win10files.freq
(3.696, 2.9249)
```

Use Special Purpose Tables with the API

1) You can pass multiple frequency tables to freq_server

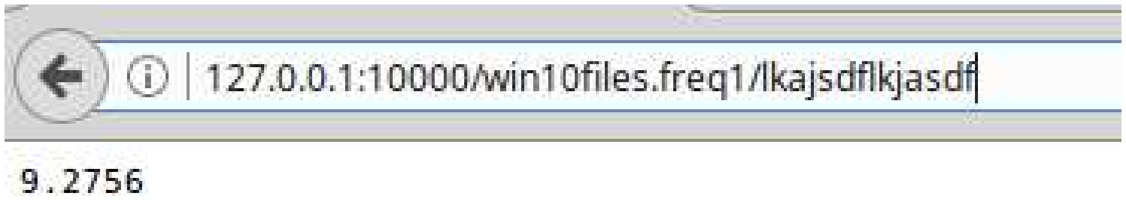
```
$ python3 freq_server.py 8080 freqtable2018.freq win10files.freq
```

2) Replace **measure**, **measure1** or **measure2** with the table name!



```
127.0.0.1:10000/win10files.freq/lkajsdflkjasdf
```

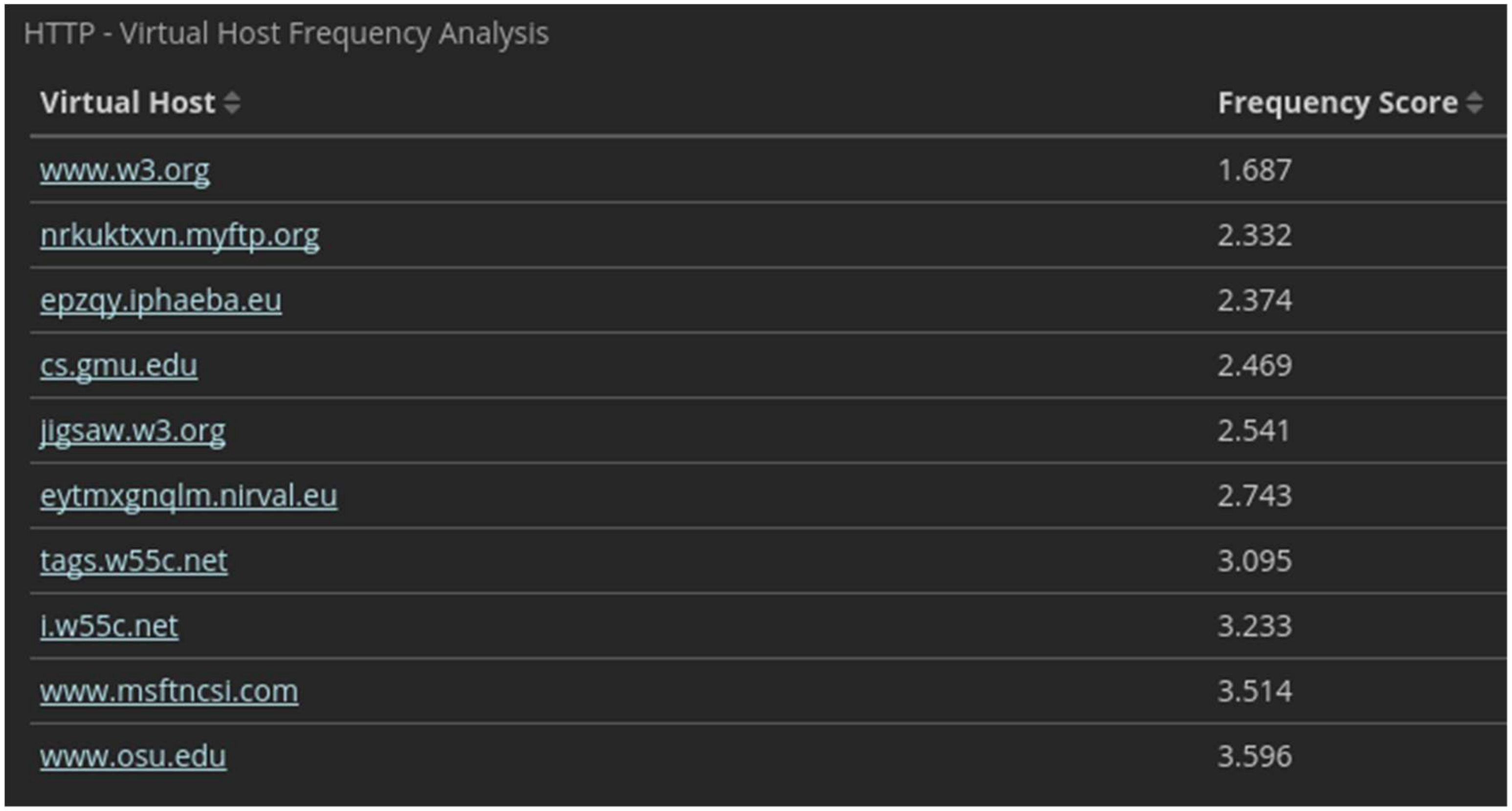
```
(9.2756, 2.4023)
```



```
127.0.0.1:10000/win10files.freq1/lkajsdflkjasdf
```

```
9.2756
```

Or Just Use Security Onion



HTTP - Virtual Host Frequency Analysis

Virtual Host	Frequency Score
www.w3.org	1.687
nrkuktxvn.myftp.org	2.332
epzqy.iphaeba.eu	2.374
cs.gmu.edu	2.469
jigsaw.w3.org	2.541
eytmxgnqlm.nirval.eu	2.743
tags.w55c.net	3.095
i.w55c.net	3.233
www.msftncsi.com	3.514
www.osu.edu	3.596

What if the tool doesn't do exactly what you need?

- Let me know. I'm happy to support these
- Come check out SEC573 and I'll show you how to customize any Python program to do exactly what you need!