Shortlisted Tags

In [33]:

```
tags = {'javascript', 'java', 'c#', 'php', 'python', 'android', 'jquery', 'html', 'c++', 'i
        'R', 'C', 'rails', 'node', 'net', 'objective-c', 'json', 'sql', 'angular', 'swift',
        'excel', 'ruby', 'ajax', 'angular', 'xml', 'asp.net', 'linux', 'react', 'spring', '
        'mongodb', 'bash', 'git', 'typescript', 'bootstrap', 'scala', 'matlab', 'apache','h
        'firebase', 'azure','shell', 'maven', 'selenium','.htaccess', 'docker', 'codeignite
        'cordova', 'express', 'tensorflow', 'unity', 'ubuntu', 'visual studio', 'vue', 'del
        'go', 'hadoop', 'jpa', 'tomcat', 'cocoa', 'nginx', 'curl', 'selenium', 'laravel', '

user_map = {}
d1 = {}
```

Parses out any tagging done via HTML tags

In [34]:

```
import re

def cleanhtml(raw_html):

    '''
    Considering each post consists of some <HTML> tags,
    this parses out the tags
    '''

    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, '', raw_html)
    return cleantext
```

In [35]:

```
cleanhtml("I love Python<Python>")
```

Out[35]:

```
'I love Python'
```

Pickling to cache data

In [36]:

```
import pickle
def pickle_file(data,name):

    '''
    Pickle files taking data and name of the file
    '''

    pkl = open(name,"wb")
    pickle.dump(data,pkl)
    pkl.close()

def unpickle_file(name):

    '''
    Unpickle files from name of the file
    '''

    try:
        pkl = open(name,"rb")
        d1 = pickle.load(pkl)
        pkl.close()
        return d1
    except:
        return "Error"
```

Jaccard Similarity calculator

In [37]:

```
def jaccard_similarity(question1,question2):

    '''
    Jaccard similarity between question1 and question2
    '''

    bigram1 = build_bigram(question1)
    bigram2 = build_bigram(question2)
    q1 = set(bigram1)
    q2 = set(bigram2)
    jacc = None
    try:
        jacc = float(len(q1.intersection(q2))/len(q1.union(q2)))
    except:
        jacc = 0.0
    return jacc
```

Trigram is more specific, and helps get extremely unique results. However, this requires the graph to be huge. As you will see later on, this does not perform well on small graphs

In [38]:

```
def build_trigram(sentence):

    '''
    Builds bigrams from list of words
    '''

    bigrams = []
    for i in range(2,len(sentence)):
        bigrams.append((sentence[i-2],sentence[i-1],sentence[i]))
    return bigrams
```

Build bigrams

In [39]:

```
def build_bigram(sentence):

    '''
    Builds bigrams from list of words
    '''

    bigrams = []
    for i in range(1,len(sentence)):
        bigrams.append((sentence[i-1],sentence[i]))
    return bigrams
```

Clearing punctuation and specific Regexing

In [40]:

```
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
stop_words = set(stopwords.words('english'))
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
def clean_punctuation_and_stopwords(question):

    '''
    Cleaning data for use for bigram usage
    '''

    question = re.sub(r'[^a-zA-Z0-9#+-]',' ',question.lower()).split()
    for i in range(len(question)):
        if question[i]=="c":
            question[i] = "C"
        elif question[i]=="r":
            question[i] = "R"
    question = list(filter(lambda x:x not in stop_words,question))
    for i in range(len(question)):
        if question[i] not in tags:
            question[i] = lemmatizer.lemmatize(question[i])
    return question
```

Builds data from dataset, this function replaces global variables

In [41]:

```
def data_builder(fname):

    global d1,user_map
    d1 = {}
    user_map = {}
    fp = open(fname,"r")
    x = fp.readline()
    cnt = 0
    while True:
        if not cnt%10000:
            print(cnt,"Users read")
        cnt+=1
        x = fp.readline()
        uid = x.split(",")[0]
        uname = x.split(",")[1]
        about = ",".join(x.split(",")[2::])
        d1[uid] = about
        user_map[uid] = uname
        try:
            if uid=="12462789" or uid=="25000019":
                break
        except:
            pass

data_builder("parsed_data.csv")
```

```
0 Users read
10000 Users read
20000 Users read
30000 Users read
40000 Users read
50000 Users read
60000 Users read
```

Calls jaccard and punctuation cleaner

In [42]:

```
def similar_questions(question1,question2):

    '''
    Checks how similar two questions are
    '''

    txt1 = clean_punctuation_and_stopwords(question1)
    txt2 = clean_punctuation_and_stopwords(question2)
    similarity = jaccard_similarity(txt1,txt2)
    return similarity
```

Sample a question

In [43]:

```
def check_about(question,small=False):
    max_jacc = 0
    profile = None
    itr = 0
    usr_arr = []
    for i in d1:
        if not itr%10000:
            print(itr)
        itr+=1

        jacc = similar_questions(d1[i],question)
        if small:
            usr_arr.append((jacc,i))
        else:
            if jacc > 0:
                usr_arr.append((jacc,i))

    return usr_arr
```

Prints working data

In [44]:

```
def full_data(usr_arr):
    full_str = []
    for i in sorted(usr_arr,reverse=True):
        full_str.extend(d1[i[1]].split())
        print(i[0],user_map[i[1]]+" : "+d1[i[1]])


    arr = list(filter(lambda x:x in tags,full_str))
    from collections import Counter
    print(Counter(arr).most_common())
```

Recommended users are written in order.

Recommended programming languages at the end of output

In [50]:

```
usr_arr = check_about("I enjoy ruby and php")
full_data(usr_arr)
```

```
0
10000
20000
30000
40000
50000
60000
0.3333333333333333 Lilli Lieberenz : ruby, php, javascript

0.3333333333333333 rfei : ruby php ubuntu

0.25 Eason : Know a little ruby and php

0.16666666666666666 Willem : Developer python, ruby, php, java and c++.

0.14285714285714285 ruby007 : want to be a web developer. learning ruby an
d php now a days

0.1111111111111111 ZyDevs : Young programming enthusiast... Mainly do java
```

Builds Language dependency matrix.

This could take a while

In [14]:

```
matrix = {i:{j:0 for j in tags} for i in tags}
fp = open("data.csv","r")
line = fp.readline()
cnt=0
while len(line)>0:
    if cnt%50000==0:
        print(cnt,"lines complete")
    line = fp.readline()
    language = re.sub(r'[^a-zA-Z0-9#+-]',' ',line.lower())
    language = language.split()
    for i in range(len(language)):
        if language[i]=="c":
            language[i] = "C"
        elif language[i]=="r":
            language[i] = "R"

    mappings = list(filter(lambda x:x in set(tags),language))
    for i in mappings:
        for j in mappings:
            if i!=j:
                matrix[i][j]+=1
    cnt+=1

fp.close()

# JavaScript, HTML, CSS p(JavaScript/HTML)
```

```
0 lines complete
50000 lines complete
100000 lines complete
150000 lines complete
200000 lines complete
250000 lines complete
300000 lines complete
350000 lines complete
400000 lines complete
450000 lines complete
500000 lines complete
550000 lines complete
600000 lines complete
650000 lines complete
700000 lines complete
750000 lines complete
800000 lines complete
850000 lines complete
900000 lines complete
```

Converts matrix to percentage weight

In [15]:

```
for i in matrix:
    matsum = 0
    for j in matrix[i]:
        matsum+=matrix[i][j]
    for j in matrix[i]:
        try:
            matrix[i][j] = (matrix[i][j]/matsum)
        except:
            matrix[i][j] = 0
```

Returns languages for input

In [16]:

```
def return_langs(fs):
    my_langs = list(filter(lambda x:x in tags,fs.lower().split()))
    for i in range(len(my_langs)):
        if my_langs[i]=="c":
            my_langs[i] = "C"
        elif my_langs[i]=="r":
            my_langs[i] = "R"
    print(my_langs)
    m1 = {i:0 for i in matrix.keys() if i not in my_langs}
    for i in my_langs:
        for j in matrix[i]:
            try:
                m1[j]+=matrix[i][j]
            except:
                pass
    orderedTp = [(i,m1[i]) for i in m1]
    sortsum  = sum([i[1] for i in orderedTp])
    for i in sorted(orderedTp,key = lambda x:x[1],reverse=True):
        print(i[0],(i[1]/sortsum)*100,sep="\t\t")
```

Friends data set to build a graph

In [17]:

```
data_builder("friends.csv")

friend_graph = {i:{j:0 for j in user_map.keys()} for i in user_map.keys()}

for i in d1:
    chk = check_about(d1[i],True)
    full_data(chk)
    for j in chk:

        if i!=j[1]:
            friend_graph[i][j[1]] = j[0]
```

```
0 Users read
0
1.0 roh : Thriving to learn and adapt to newer technologies.Heading toward
s learning and understanding while tackling real world problems. Currently
researching on Virtual Reality and Machine Learning. Well versed in Pytho
n, Javascript, C and Angular.

0.0625 shr : Has a diverse skill set in technologies and is interested in
the latest trends around the globe. Has worked with cloud technologies and
also in the field of IoT. Also interested in data science/analytics as wel
l as machine learning. Well versed in Django, Python and Mongo.

0.04 sre : I am graduate Student at USC with a strong focus on Machine Lea
rning and Data Science.Apart from this, I also have full stack software de
velopment experience in an agile environment gained during my 6 month inte
rnship at GE Digital. Well versed in Perl, Ruby and C++.

0.037037037037037035 And :  author of The Hundred-Page Machine Learning Bo
ok
```

Structure of friend graph

In [18]:

```
print(friend_graph)
```

```
{'25000002': {'25000002': 0, '25000003': 0.0, '25000004': 0.01851851851851
8517, '25000005': 0.0625, '25000006': 0.04, '24000001': 0.0, '24000002':
0.0, '24000003': 0.0, '24000004': 0.0, '24000005': 0.015625, '24000006':
0.0, '24000007': 0.0, '24000008': 0.01020408163265306, '24000009': 0.02083
3333333333332, '24000010': 0.013157894736842105, '24000011': 0.02, '240000
12': 0.0, '24000013': 0.0, '24000014': 0.015625, '24000015': 0.0, '2400001
6': 0.014084507042253521, '24000017': 0.0, '24000018': 0.0169491525423728
8, '24000019': 0.011627906976744186, '24000020': 0.0, '24000021': 0.0, '25
000007': 0.0, '25000008': 0.022727272727272728, '26000001': 0.037037037037
037035, '26000002': 0.0, '26000003': 0.0, '26000004': 0.0, '26000005': 0.
0, '26000006': 0.014705882352941176, '26000007': 0.022222222222222223, '25
000010': 0.0, '25000011': 0.0, '25000012': 0.0, '25000013': 0.024390243902
439025, '25000014': 0.023255813953488372, '25000015': 0.0, '25000016': 0.
0, '25000017': 0.0, '25000018': 0.0, '25000019': 0.025}, '25000003': {'250
00002': 0.0, '25000003': 0, '25000004': 0.024390243902439025, '25000005':
0.02702702702702703, '25000006': 0.0, '24000001': 0.0, '24000002': 0.0, '2
4000003': 0.0, '24000004': 0.0, '24000005': 0.0196078431372549, '2400000
6': 0.0, '24000007': 0.0, '24000008': 0.0, '24000009': 0.0, '24000010': 0.
0, '24000011': 0.0, '24000012': 0.0, '24000013': 0.0, '24000014': 0.0, '24
```

Person v Person comparison

In [19]:

```
for i in friend_graph:
    for j in friend_graph[i]:
        print(user_map[i],user_map[j],friend_graph[i][j],sep="\t\t")
```

```
roh             roh             0
roh             pri             0.0
roh             sur             0.018518518518518517
roh             shr             0.0625
roh             sre             0.04
roh             vai             0.0
roh             nish            0.0
roh             ish             0.0
roh             rhy             0.0
roh             rrt             0.015625
roh             sun             0.0
roh             mid             0.0
roh             nav             0.01020408163265306
roh             apa             0.020833333333333332
roh             aay             0.013157894736842105
roh             ave             0.02
roh             nee             0.0
roh             nih             0.0
roh             dwe             0.015625
```

Writes Edge graph for friend's matrix using Gephi format

In [20]:

```
'''
;A;B;C;D;E
A;0;1;0;1;0
B;1;0;0;0;0
C;0;0;1;0;0
D;0;1;0;1;0
E;0;0;0;0;0
'''
edg_list = open("edge.csv","w")
fkeys = ";"+";".join([user_map[i] for i in friend_graph.keys()])
edg_list.write(fkeys+"\n")
for i in friend_graph:
    edg_list.write(user_map[i])
    for j in friend_graph[i]:
        edg_list.write(";"+str(friend_graph[i][j]))
    edg_list.write("\n")
edg_list.close()
```

Programming language dependency matrix to Gephi format

In [21]:

```
edg_list = open("progmatrix.csv","w")
fkeys = ";"+";".join([i for i in matrix.keys()])
edg_list.write(fkeys+"\n")
for i in matrix:
    edg_list.write(i)
    for j in matrix[i]:
        edg_list.write(";"+str(matrix[i][j]))
    edg_list.write("\n")
edg_list.close()
```

Sample from language dependency matrix

C++ naturally suggests C language

In [22]:

```
return_langs("I love c++")
```

```
['c++']
C               12.859446884731732
java            12.61269296231057
python          9.341717020818004
c#              7.974908396322476
javascript              5.847027432793003
php             5.364668108541998
net             4.07961530171613
html            3.8878607475454676
linux           3.450095505659733
android         3.267259767962125
sql             3.129018112629788
css             2.9015882925669105
ruby            1.5020773410035158
asp             1.4983611674730768
jquery          1.4418753298104012
ios             1.3682950939077054
perl            1.3437683486068068
matlab          0.958772770853308
R               0.9060031067210715
swift           0.8294499319940246
objective-c             0.8227608196392341
go              0.8026934825748625
bash            0.7722208596252613
unix            0.7714776249191734
node            0.7670182166826464
xml             0.7157350219625858
delphi          0.6116821631102888
git             0.5455342742684713
angular         0.5217507636736606
shell           0.49573754896058636
rails           0.46972433424751214
unity           0.46675139542316074
react           0.4652649260109851
scala           0.4318193642370325
django          0.420670843645715
spring          0.41546820070310014
mongodb         0.3954008636387286
haskell         0.3901982206961137
bootstrap               0.3790497001047962
ubuntu          0.375333526574357
ajax            0.3745902918682692
wordpress               0.3604688324526003
apache          0.33073944420908685
typescript              0.2742536065464113
excel           0.27276713713423567
json            0.27053743301597216
docker          0.26830772889770865
laravel         0.23337569771158037
azure           0.2252001159446142
flash           0.20736248299850615
express         0.1709439824002022
hibernate               0.1649981047514995
hadoop          0.13452548180189824
codeigniter             0.12634990003493204
tensorflow              0.108512267088824
vue             0.10702579767664833
```

```
cocoa           0.10553932826447265
tomcat          0.09959345061576998
maven           0.09216110355489161
nginx           0.08324228708183759
selenium                0.07878287884531059
symfony         0.07358023590269575
firebase                0.06986406237225655
cordova         0.05871554178093902
jpa             0.05128319472006066
gradle          0.03047262294960126
curl            0.02601321471307425
asp.net         0.0
visual studio           0.0
.htaccess               0.0
```

Cocoa and Swift are programming languages for Apple app development.

Objective-C is by Apple as well

In [23]:

```
return_langs("I love cocoa and swift")
```

```
['cocoa', 'swift']
ios             20.19933902814618
objective-c             7.981451985764442
C               6.860615495585108
java            6.43815962596672
android         5.356668001301781
c++             4.268624149675835
javascript              4.111279507550652
php             3.9881573319964105
python          3.667740445215726
c#              3.4030547946740173
net             2.982878248884156
html            2.5102424351057753
css             2.2967708254464583
ruby            2.1502927657152298
sql             1.959764728017557
json            1.796489643438421
xml             1.4148200739961152
git             1.4062145764184797
jquery          1.3896187334021883
linux           1.121446668042257
react           1.090095464150556
rails           1.0784241544301367
node            1.0042582511260063
asp             0.9311247864522566
firebase                0.7453084935747084
angular         0.7407995886782186
perl            0.5588803647422189
go              0.5435131679118672
unity           0.5174954933473453
mongodb         0.5160606910537332
django          0.4716050332400196
wordpress               0.46443433795599637
R               0.3843313598298392
unix            0.3828982156282459
spring          0.3826920595205735
laravel         0.37101411743207924
bootstrap               0.34212849637235176
bash            0.33434430704136814
delphi          0.32430566526213944
ajax            0.3230770190761996
typescript              0.2650979627570536
flash           0.2624361723695205
ubuntu          0.22678885205707688
apache          0.19851672504430998
shell           0.1976970748896771
docker          0.19523812442577862
scala           0.17126874620182939
excel           0.16246040870055933
haskell         0.15897731006637453
azure           0.15631386158682253
cordova         0.14996281645743217
vue             0.13459727771909946
express         0.12435413792421728
codeigniter             0.11964239310409262
hibernate               0.11308684995904876
matlab          0.11226719980441595
```

```
symfony          0.06350879320188466
jpa              0.06064084670667914
maven            0.0542898015772888
nginx            0.05347015142265596
selenium                0.051626353097736784
tomcat           0.05142185508208329
hadoop           0.050602204927450455
tensorflow              0.027656974873787564
gradle           0.023969378223949223
curl             0.0036875966498383416
asp.net          0.0
visual studio            0.0
.htaccess                0.0
```

Gradle and Maven are built for Java, and Spring uses both Gradle and Maven for package dependency. Android pairs with Java, as does Hibernate

In [24]:

```
return_langs("I love gradle and maven")
```

```
['gradle', 'maven']
java            16.253541029350643
spring          10.69253285937903
android         5.219207707456439
hibernate               4.252913706332635
git             4.163974224108209
sql             3.9568021123143255
javascript              3.937601261840673
apache          3.1375907166322965
docker          2.595781936129478
linux           2.3839573355414205
html            2.2789435532332862
tomcat          2.274291064439112
jquery          2.264690639202286
css             2.2259934906065015
python          2.191948830804891
selenium                2.0491242428464087
xml             1.9490827470618464
mongodb         1.8595280194595456
jpa             1.8266651502518494
angular         1.4990445515249173
C               1.4792528057543073
php             1.4636460272242346
json            1.383396277210282
c++             1.3148394360534243
net             1.1109659485546424
c#              1.0098660359194862
ajax            1.0023580943029287
react           0.9710211634994624
bootstrap               0.9143049550239394
node            0.8946609330775688
ruby            0.7748033414414773
shell           0.7734494769424051
bash            0.7220010026389319
scala           0.7044494643128298
ios             0.6340955871841822
unix            0.6184888086541096
ubuntu          0.5976386460329058
typescript              0.5350881581692947
hadoop          0.5112348189014678
R               0.4921816922520541 6
perl            0.4905323801045034
firebase                0.4054824174721376
nginx           0.3786259015576876
asp             0.35012007349568686
express         0.33872261228707035
swift           0.32897446322600454
azure           0.32762059872693233
django          0.31637086134255504
go              0.27796916039524916
wordpress               0.22696385756449383
rails           0.2125632197092542
laravel         0.2125632197092542
objective-c             0.1867651206453979
vue             0.17731241923281074
codeigniter             0.16936130614353517
excel           0.14385865472815748
```

```
cordova          0.12300849210695375
tensorflow              0.10380764163330082
unity            0.08145589068878555
matlab           0.05910413974427025
symfony          0.05115302665499463
haskell          0.039903289270617356
delphi           0.03360148832889258
flash            0.03360148832889258
cocoa            0.004800212618413226
curl             0.004800212618413226
asp.net          0.0
visual studio           0.0
.htaccess               0.0
```

In [ ]:

In [ ]:

In [ ]: