Task 3B

December 22, 2020

[26]:
```python
#connecting to database
import pymongo
from pymongo import MongoClient
from pprint import pprint
import xmltodict
client = pymongo.MongoClient("mongodb://user1:123@172.17.0.2")
db=client['db']
```

[ ]:
```python
#Script to insert data from XML into MongoDB
import json
file=open("Badges.xml", "r")
badge_collection=db['badges']
count=0
while True:
    count += 1
    print(count)
    line = file.readline()
    if not line:
        break
    if count <=2 or line.startswith("</badges"):
        continue
    ins=xmltodict.parse(line)
    badge_collection.insert_one(ins['row'])
```

[13]:
```python
#Due to some errors, duplicates had entered the database, and this code removes␣
↪them
import pymongo
from pymongo import MongoClient
client = pymongo.MongoClient("mongodb://user1:123@172.17.0.2")
db=client['db']
post_collection=db['posts']
cursor=post_collection.find_one()
print(cursor.keys())
cursor=post_collection.find({})
```

```python
arrayid=[]
count=0
counter=0
copyid=[]
for document in cursor:
    if document['@Id'] in arrayid:
        count+=1
    arrayid.append(document['@Id'])
    counter+=1
    if(counter%1000==0):
        print(count)
        print(counter)
print(count)
```

```
AY
dict_keys(['_id', '@Id', '@PostTypeId', '@AcceptedAnswerId', '@CreationDate',
'@Score', '@ViewCount', '@Body', '@OwnerUserId', '@LastEditorUserId',
'@LastEditDate', '@LastActivityDate', '@Title', '@Tags', '@AnswerCount',
'@CommentCount', '@FavoriteCount', '@ClosedDate'])
0
1000
0
2000
0
3000
0
4000
0
5000
0
6000
0
7000
0
8000
0
9000
0
10000
0
11000
0
12000
0
13000
0
14000
```

```
0
15000
0
16000
0
17000
0
18000
0
19000
0
20000
```

```
       ␣
 ↪---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call␣
 ↪last)

        <ipython-input-13-f438f6f73690> in <module>
         13 copyid=[]
         14 for document in cursor:
    ---> 15     if document['@Id'] in arrayid:
         16         count+=1
         17     arrayid.append(document['@Id'])


        KeyboardInterrupt:
```

[28]:
```
import pymongo
from pymongo import MongoClient
client = pymongo.MongoClient("mongodb://user1:123@172.17.0.2")
db=client['db']
post_collection=db['posts']
cursor=post_collection.find_one()
print(cursor.keys())
```

```
dict_keys(['_id', '@Id', '@PostTypeId', '@AcceptedAnswerId', '@CreationDate',
'@Score', '@ViewCount', '@Body', '@OwnerUserId', '@LastEditorUserId',
'@LastEditDate', '@LastActivityDate', '@Title', '@Tags', '@AnswerCount',
'@CommentCount', '@FavoriteCount', '@ClosedDate'])
```

[14]:
```
#Finding top 10 hashtags
cursor=post_collection.find({})
dic={}
total_count=0
```

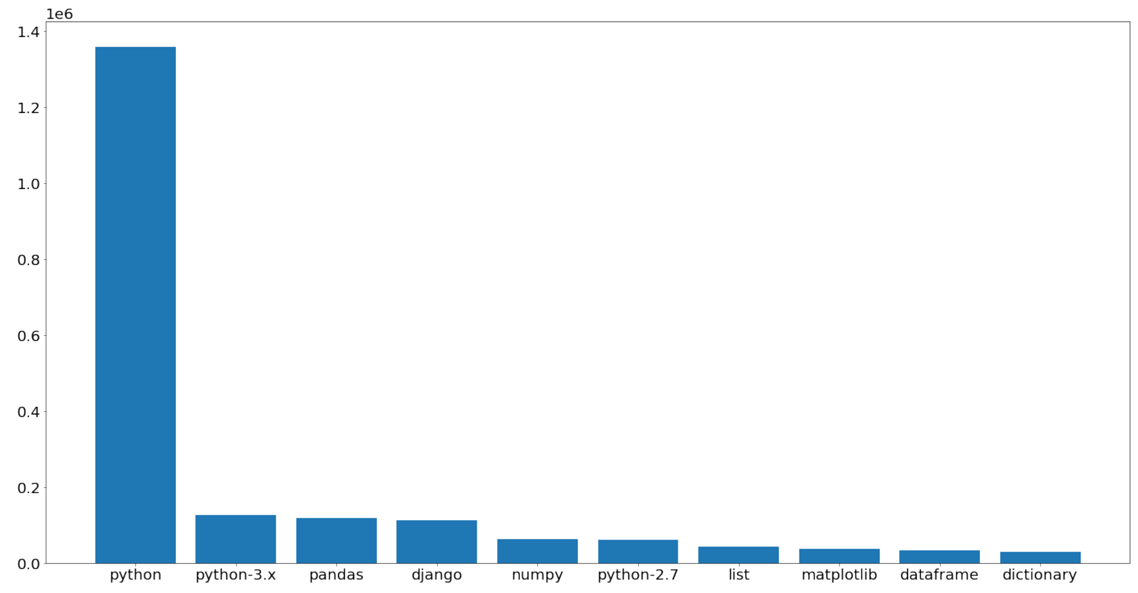
```
for document in cursor:
    if '@Tags' in document:
        tags=document['@Tags']
        tags=tags.split("><")
        tags[0]=tags[0][1:]
        tags[-1]=tags[-1][:-1]
        for tag in tags:
            if tag in dic:
                dic[tag]+=1
            else:
                dic[tag]=1
    total_count+=1
dic={k: v for k, v in sorted(dic.items(), key=lambda item: item[1],␣
 ↪reverse=True)}
```

[15]:
```
#Printing top 10 hashtags
top_10_list=list(dic.items())[:10]
print(top_10_list)
print(total_count)
```

```
[('python', 1358860), ('python-3.x', 125609), ('pandas', 119220), ('django',
111953), ('numpy', 62820), ('python-2.7', 60590), ('list', 42883),
('matplotlib', 38246), ('dataframe', 33927), ('dictionary', 29821)]
3380601
```

[16]:
```
#Plotting top 10 hashtags
tag_list=[]
value_list=[]
for i in top_10_list:
    tag_list.append(i[0])
    value_list.append(i[1])
print(tag_list)
print(value_list)
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"]=20,10
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(tag_list,value_list)
plt.show()
```

```
['python', 'python-3.x', 'pandas', 'django', 'numpy', 'python-2.7', 'list',
'matplotlib', 'dataframe', 'dictionary']
[1358860, 125609, 119220, 111953, 62820, 60590, 42883, 38246, 33927, 29821]
```



The above plot shows the top 10 tags, and it is evident that frequency of the tag ‘python’ exceeds all the other tags.

[17]:
```
cursor=post_collection.find_one()
keys_array=(cursor.keys())
cursor=post_collection.find({})
total_count=0
for document in cursor:
    for k in keys_array:
        if k in document:
            print(k,document[k])
    total_count+=1
    if total_count==10:
        break
```

```
_id 5fe09f7966ded275b5a38a42
@Id 337
@PostTypeId 1
@AcceptedAnswerId 342
@CreationDate 2008-08-02T03:35:55.697
@Score 71
@ViewCount 8043
@Body <p>I am about to build a piece of a project that will need to construct
and post an XML document to a web service and I'd like to do it in Python, as a
means to expand my skills in it.  </p>

<p>Unfortunately, whilst I know the XML model fairly well in .NET, I'm uncertain
what the pros and cons are of the XML models in Python.  </p>
```

```
<p>Anyone have experience doing XML processing in Python? Where would you
suggest I start? The XML files I'll be building will be fairly simple.</p>

@OwnerUserId 111
@LastEditorUserId 2336654
@LastEditDate 2016-12-30T12:56:21.493
@LastActivityDate 2019-05-22T00:27:38.800
@Title XML Processing in Python
@Tags <python><xml>
@AnswerCount 12
@CommentCount 2
@FavoriteCount 7
@ClosedDate 2016-03-26T01:51:47.153
_id 5fe09f7966ded275b5a38a43
@Id 342
@PostTypeId 2
@CreationDate 2008-08-02T04:01:34.600
@Score 28
@Body <p>Personally, I've played with several of the built-in options on an XML-
heavy project and have settled on <a href="http://docs.python.org/lib/module-
xml.dom.pulldom.html" rel="noreferrer">pulldom</a> as the best choice for less
complex documents.</p>

<p>Especially for small simple stuff, I like the event-driven theory of parsing
rather than setting up a whole slew of callbacks for a relatively simple
structure.  <a href="http://www.prescod.net/python/pulldom.html"
rel="noreferrer">Here is a good quick discussion of how to use the API</a>.</p>

<p>What I like: you can handle the parsing in a <code>for</code> loop rather
than using callbacks.  You also delay full parsing (the "pull" part) and only
get additional detail when you call <code>expandNode()</code>.  This satisfies
my general requirement for "responsible" efficiency without sacrificing ease of
use and simplicity.</p>

@OwnerUserId 59
@LastActivityDate 2008-08-02T04:01:34.600
@CommentCount 1
_id 5fe09f7966ded275b5a38a44
@Id 469
@PostTypeId 1
@AcceptedAnswerId 3040
@CreationDate 2008-08-02T15:11:16.430
@Score 39
@ViewCount 2977
@Body <p>I am using the Photoshop's javascript API to find the fonts in a given
PSD.</p>
```

```
<p>Given a font name returned by the API, I want to find the actual physical
font file that font name corresponds to on the disc.</p>

<p>This is all happening in a python program running on OSX so I guess I'm
looking for one of:</p>

<ul>
<li>Some Photoshop javascript</li>
<li>A Python function</li>
<li>An OSX API that I can call from python</li>
</ul>

@OwnerUserId 147
@LastEditorUserId 1997093
@LastEditDate 2016-12-22T03:53:45.467
@LastActivityDate 2019-10-12T04:37:11.513
@Title How can I find the full path to a font from its display name on a Mac?
@Tags <python><macos><fonts><photoshop>
@AnswerCount 4
@CommentCount 0
@FavoriteCount 0
_id 5fe09f7966ded275b5a38a45
@Id 471
@PostTypeId 2
@CreationDate 2008-08-02T15:21:03.587
@Score 31
@Body <p><a href="http://effbot.org/zone/element-index.htm"
rel="noreferrer">ElementTree</a> has a nice pythony API.  I think it's even
shipped as part of python 2.5</p>

<p>It's in pure python and as I say, pretty nice, but if you wind up needing
more performance, then <a href="http://codespeak.net/lxml/"
rel="noreferrer">lxml</a> exposes the same API and uses libxml2 under the hood.
You can theoretically just swap it in when you discover you need it.</p>

@OwnerUserId 147
@LastActivityDate 2008-08-02T15:21:03.587
@CommentCount 2
_id 5fe09f7966ded275b5a38a46
@Id 497
@PostTypeId 2
@CreationDate 2008-08-02T16:56:53.893
@Score 8
@Body <p>open up a terminal (Applications->Utilities->Terminal) and type this
in:</p>

<pre><code>locate InsertFontHere
</code></pre>
```

```
<p>This will spit out every file that has the name you want.</p>

<p>Warning: there may be alot to wade through.</p>

@OwnerUserId 50
@LastActivityDate 2008-08-02T16:56:53.893
@CommentCount 0
_id 5fe09f7966ded275b5a38a47
@Id 502
@PostTypeId 1
@AcceptedAnswerId 7090
@CreationDate 2008-08-02T17:01:58.500
@Score 42
@ViewCount 14367
@Body <p>I have a cross-platform (Python) application which needs to generate a
JPEG preview of the first page of a PDF.</p>

<p>On the Mac I am spawning <a href="http://developer.apple.com/documentation/Da
rwin/Reference/ManPages/man1/sips.1.html" rel="noreferrer">sips</a>.  Is there
something similarly simple I can do on Windows?</p>

@OwnerUserId 147
@LastEditorUserId 63550
@LastEditDate 2011-04-08T11:42:03.807
@LastActivityDate 2019-10-16T08:06:32.043
@Title Get a preview JPEG of a PDF on Windows?
@Tags <python><windows><image><pdf>
@AnswerCount 3
@CommentCount 0
@FavoriteCount 13
_id 5fe09f7966ded275b5a38a48
@Id 518
@PostTypeId 2
@CreationDate 2008-08-02T17:42:28.607
@Score 6
@Body <p>I haven't been able to find anything that does this directly.  I think
you'll have to iterate through the various font folders on the system:
<code>/System/Library/Fonts</code>, <code>/Library/Fonts</code>, and there can
probably be a user-level directory as well <code>~/Library/Fonts</code>.</p>

@OwnerUserId 153
@LastEditorUserId 725306
@LastEditDate 2013-03-22T09:27:54.323
@LastActivityDate 2013-03-22T09:27:54.323
@CommentCount 0
_id 5fe09f7966ded275b5a38a49
@Id 525
```

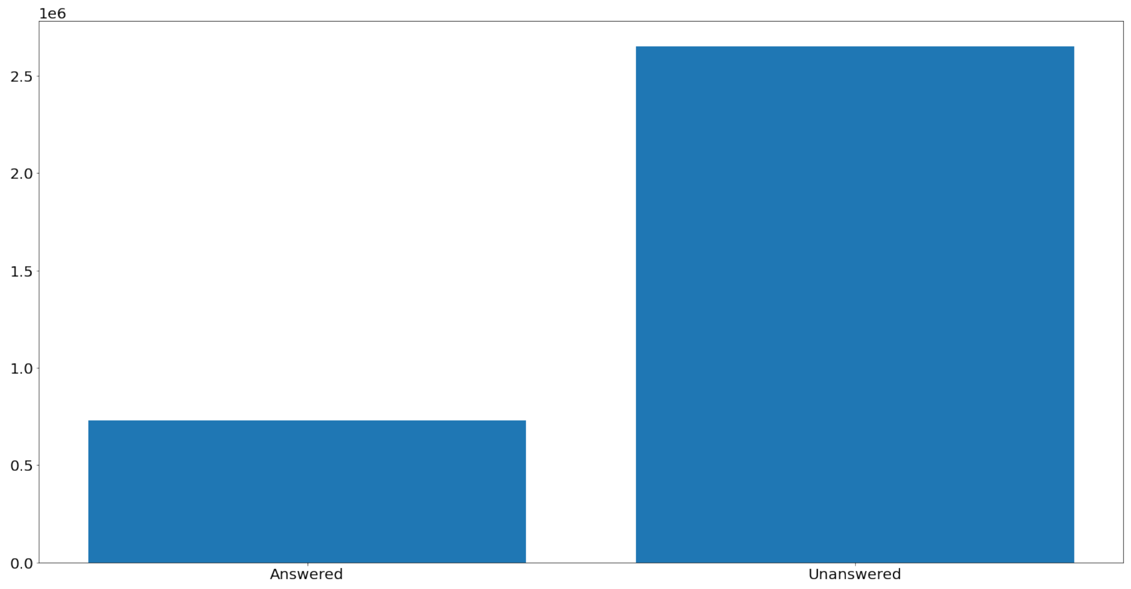
```
@PostTypeId 2
@CreationDate 2008-08-02T18:04:10.543
@Score 4
@Body <p>Since you mentioned that you'll be building "fairly simple" XML, the <a
href="http://www.python.org/doc/current/lib/module-xml.dom.minidom.html"
rel="noreferrer">minidom module</a> (part of the Python Standard Library) will
likely suit your needs. If you have any experience with the DOM representation
of XML, you should find the API quite straight forward.</p>

@OwnerUserId 154
@LastActivityDate 2008-08-02T18:04:10.543
@CommentCount 0
_id 5fe09f7966ded275b5a38a4a
@Id 535
@PostTypeId 1
@AcceptedAnswerId 541
@CreationDate 2008-08-02T18:43:54.787
@Score 57
@ViewCount 8879
@Body <p>I am starting to work on a hobby project with a <strong>Python</strong>
codebase and I would like to set up some form of continuous integration (i.e.
running a battery of test-cases each time a check-in is made and sending nag
e-mails to responsible persons when the tests fail) similar to
<strong>CruiseControl</strong> or <strong>TeamCity</strong>.</p>

<p>I realize I could do this with hooks in most <strong>VCSes</strong>, but that
requires that the tests run on the same machine as the version control server,
which isn't as elegant as I would like. Does anyone have any suggestions for a
small, user-friendly, open-source continuous integration system suitable for a
<strong>Python</strong> codebase?</p>

@OwnerUserId 154
@LastEditorUserId 7232508
@LastEditDate 2018-05-14T17:46:14.650
@LastActivityDate 2018-05-14T17:46:14.650
@Title Continuous Integration System for a Python Codebase
@Tags <python><continuous-integration><extreme-programming>
@AnswerCount 7
@CommentCount 0
@FavoriteCount 14
_id 5fe09f7966ded275b5a38a4b
@Id 536
@PostTypeId 2
@CreationDate 2008-08-02T18:49:07.740
@Score 12
@Body <p>You can use ImageMagick's convert utility for this, see some examples
in <a href="https://web.archive.org/web/20120413111338/http://studio.imagemagick
.org/pipermail/magick-users/2002-May/002636.html"
```

```
rel="noreferrer">http://studio.imagemagick.org/pipermail/magick-
users/2002-May/002636.html</a>
:</p>

<blockquote>
<pre><code>Convert taxes.pdf taxes.jpg
</code></pre>

  <p>Will convert a two page PDF file into [2] jpeg files: taxes.jpg.0,
  taxes.jpg.1</p>

  <p>I can also convert these JPEGS to a thumbnail as follows:</p>

<pre><code>convert -size 120x120 taxes.jpg.0 -geometry 120x120 +profile '*'
thumbnail.jpg
</code></pre>

  <p>I can even convert the PDF directly to a jpeg thumbnail as follows:</p>

<pre><code>convert -size 120x120 taxes.pdf -geometry 120x120 +profile '*'
thumbnail.jpg
</code></pre>

  <p>This will result in a thumbnail.jpg.0 and thumbnail.jpg.1 for the two
  pages.</p>
</blockquote>

@OwnerUserId 161
@LastEditorUserId 2581872
@LastEditDate 2015-08-29T23:20:48.107
@LastActivityDate 2015-08-29T23:20:48.107
@CommentCount 0
```

[18]:
```
#Checking
cursor=post_collection.find({})
answered=0
not_answered=0
for document in cursor:
    if '@AcceptedAnswerId' in document:
        answered+=1
    else:
        not_answered+=1
print(answered)
print(not_answered)
```

```
730441
2650160
```

[29]:
```
arr1=["Answered", "Unanswered"]
arr2=[answered, not_answered]
plt.rcParams["figure.figsize"]=20,10
plt.rc('xtick', labelsize=20)
plt.rc('ytick', labelsize=20)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(arr1, arr2)
plt.show()
```



As can be seen from the above plot, the number of questions that have an accepted answer are a lot lesser than the questions which don't have an accepted answer(questions without an accepted answer are 3.628 times the number of questions with an accepted answer).

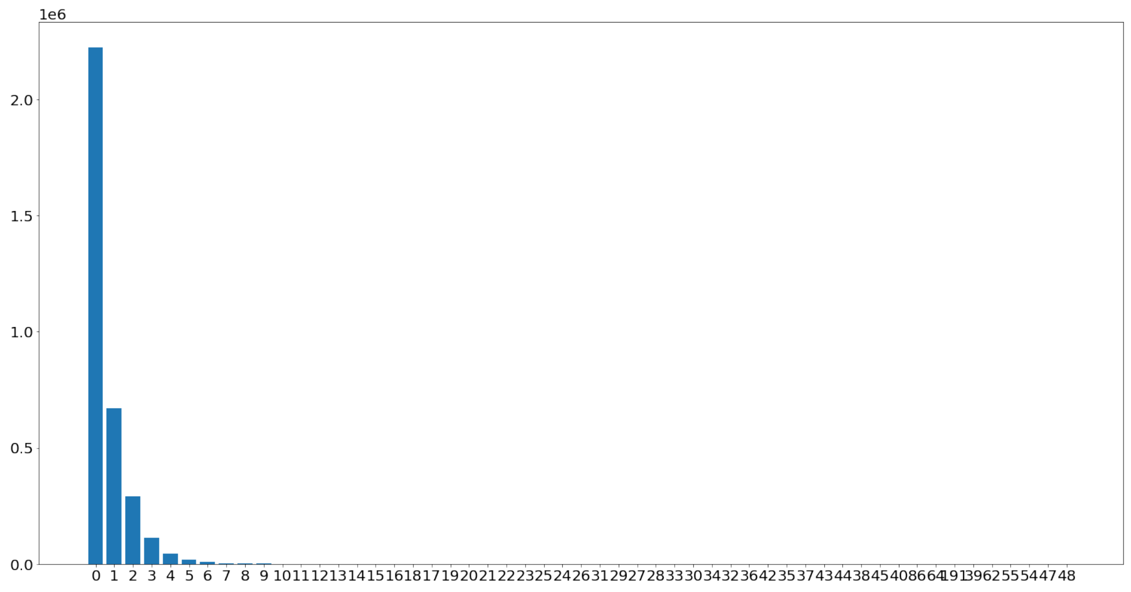
[20]:
```
cursor=post_collection.find({})
answer_dic={}
answer_dic['0']=0
for document in cursor:
    if '@AnswerCount' in document:
        if(document['@AnswerCount'] in answer_dic):
            answer_dic[document['@AnswerCount']]+=1
        else:
            answer_dic[document['@AnswerCount']]=1
    else:
        answer_dic['0']+=1
```

[21]:
```
answer_dic={k: v for k, v in sorted(answer_dic.items(), key=lambda item:␣
 ↪item[1], reverse=True)}
print(answer_dic)
```

```
{'0': 2222563, '1': 671088, '2': 292266, '3': 111599, '4': 44855, '5': 18596,
'6': 8320, '7': 4181, '8': 2283, '9': 1355, '10': 875, '11': 603, '12': 421,
'13': 335, '14': 237, '15': 185, '16': 157, '18': 86, '17': 84, '19': 81, '20':
66, '21': 46, '22': 45, '23': 40, '25': 34, '24': 32, '26': 25, '31': 22, '29':
19, '27': 19, '28': 16, '33': 9, '30': 8, '34': 7, '32': 6, '36': 6, '42': 5,
'35': 4, '37': 3, '43': 2, '44': 2, '38': 2, '45': 2, '40': 2, '86': 1, '64': 1,
'191': 1, '39': 1, '62': 1, '55': 1, '54': 1, '47': 1, '48': 1}
```

[22]:
```
arr1=(list(answer_dic.keys()))
arr2=(list(answer_dic.values()))
print(arr1)
print(arr2)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(arr1, arr2)
plt.show()
```

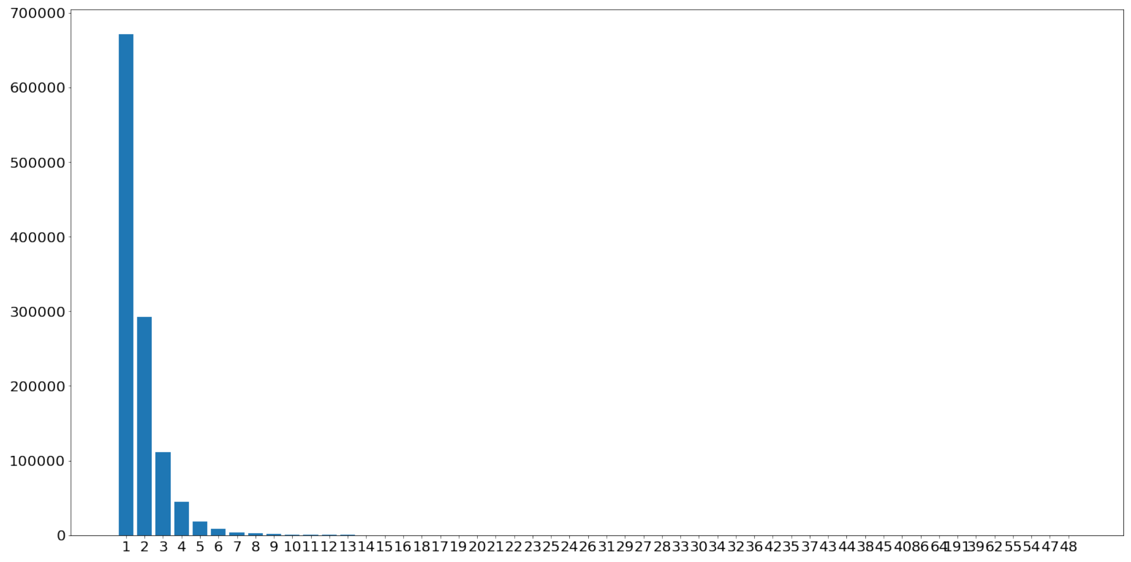
```
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14',
'15', '16', '18', '17', '19', '20', '21', '22', '23', '25', '24', '26', '31',
'29', '27', '28', '33', '30', '34', '32', '36', '42', '35', '37', '43', '44',
'38', '45', '40', '86', '64', '191', '39', '62', '55', '54', '47', '48']
[2222563, 671088, 292266, 111599, 44855, 18596, 8320, 4181, 2283, 1355, 875,
603, 421, 335, 237, 185, 157, 86, 84, 81, 66, 46, 45, 40, 34, 32, 25, 22, 19,
19, 16, 9, 8, 7, 6, 6, 5, 4, 3, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```



The above bar graph shows the number of answers on a question. We can see, as before that a very high number of posts which have zero answers, and this number keeps getting half after each value from 1, and after 4 answers, the number gets really small. This shows that most of the posts have 0-4 answers, and number of answers greater than that is rather uncommon. Another interesting observation that we can make is that number of questions with 0 nswers is 2222563, whereas number of questions without accepted answers is 2650160, which indicates that there are 427597 where a proposed answer(s) wasn't accepted.

[23]:
```python
arr1=arr1[1:]
arr2=arr2[1:]
print(arr1)
print(arr2)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(arr1, arr2)
plt.show()
```

```
['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14',
'15', '16', '18', '17', '19', '20', '21', '22', '23', '25', '24', '26', '31',
'29', '27', '28', '33', '30', '34', '32', '36', '42', '35', '37', '43', '44',
'38', '45', '40', '86', '64', '191', '39', '62', '55', '54', '47', '48']
[671088, 292266, 111599, 44855, 18596, 8320, 4181, 2283, 1355, 875, 603, 421,
335, 237, 185, 157, 86, 84, 81, 66, 46, 45, 40, 34, 32, 25, 22, 19, 19, 16, 9,
8, 7, 6, 6, 5, 4, 3, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```



The above bar graph gives a better visualisation of the questions with an answer, and shows the number of posts with a given answer.

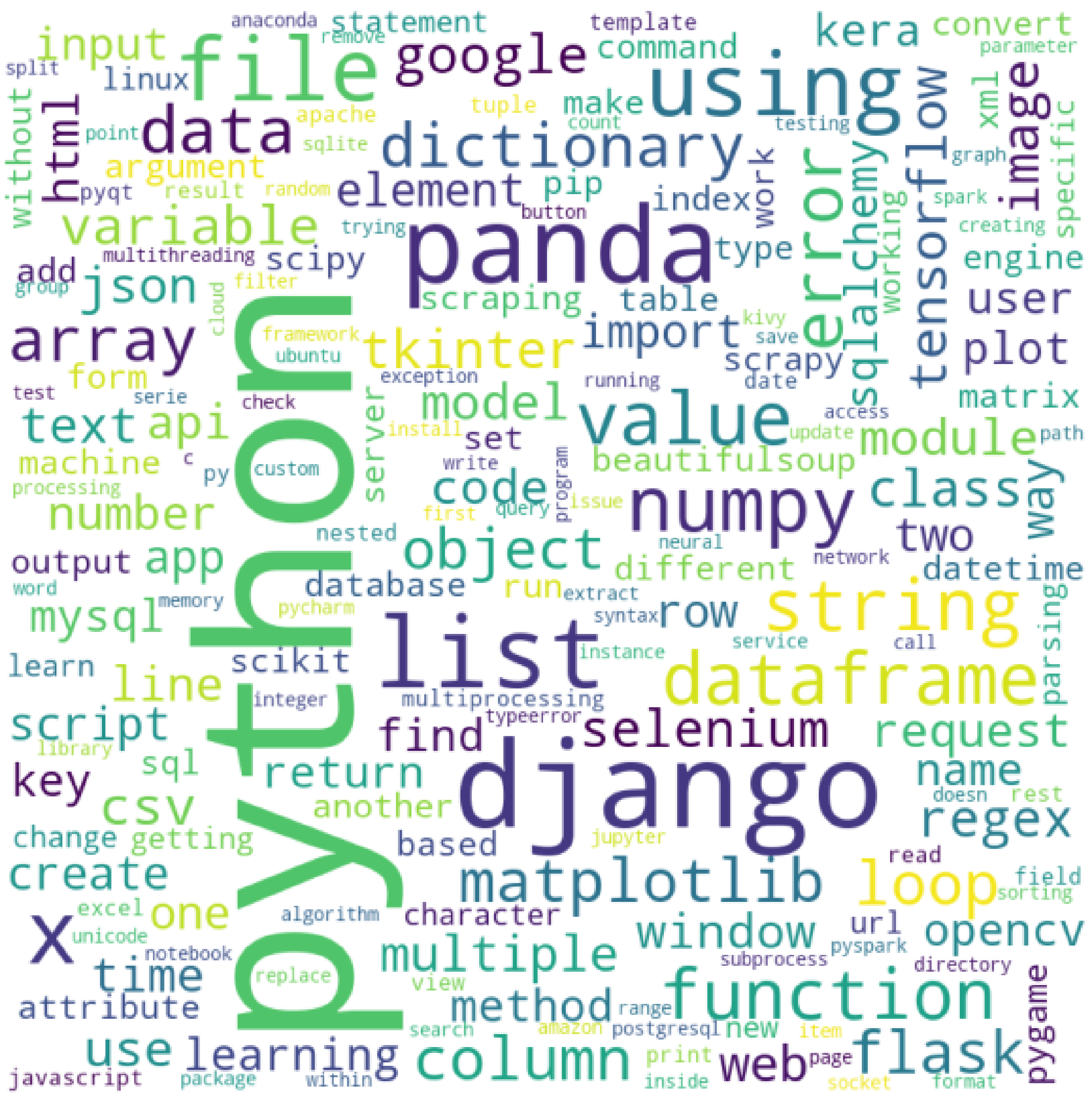
[24]:
```
cursor=post_collection.find()
print(cursor[0])
```

```
{'_id': ObjectId('5fe09f7966ded275b5a38a42'), '@Id': '337', '@PostTypeId': '1',
'@AcceptedAnswerId': '342', '@CreationDate': '2008-08-02T03:35:55.697',
'@Score': '71', '@ViewCount': '8043', '@Body': "<p>I am about to build a piece
of a project that will need to construct and post an XML document to a web
service and I'd like to do it in Python, as a means to expand my skills in it.
</p>\n\n<p>Unfortunately, whilst I know the XML model fairly well in .NET, I'm
uncertain what the pros and cons are of the XML models in Python.
</p>\n\n<p>Anyone have experience doing XML processing in Python? Where would
you suggest I start? The XML files I'll be building will be fairly
simple.</p>\n", '@OwnerUserId': '111', '@LastEditorUserId': '2336654',
'@LastEditDate': '2016-12-30T12:56:21.493', '@LastActivityDate':
'2019-05-22T00:27:38.800', '@Title': 'XML Processing in Python', '@Tags':
'<python><xml>', '@AnswerCount': '12', '@CommentCount': '2', '@FavoriteCount':
'7', '@ClosedDate': '2016-03-26T01:51:47.153'}
```

[ ]:
```
import re
from wordcloud import WordCloud, STOPWORDS
cursor=post_collection.find({})
arra=[]
cnt=0
for document in cursor:
    if '@Tags' in document:
        tags=document['@Tags']
        tags=tags.split("><")
        tags[0]=tags[0][1:]
        tags[-1]=tags[-1][:-1]
        for tag in tags:
            tag=tag.lower()
            arr_tag=tag.split()
            arra+=arr_tag
    if '@Title' in document:
        tit=document['@Title']
        tit=tit.lower()
        arra+=re.compile('\w+').findall(tit)
    cnt+=1
    if(cnt%1000==0):
        print(cnt)
print(len(arra))
```

[42]:
```
for x in range(len(arra)):
    arra[x]=arra[x].strip()
    if len(arra[x])==1:
        arra[x]=""
comment_words=" ".join(arra)
```

```
stopwords = set(STOPWORDS)
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10,
                collocations=False).generate(comment_words)
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



The above WordCloud shows the most important words in the title and tags of the posts, and we can mainly see Python related keywords in the above Wordcloud.

The observation that I made was that every post was realted to Python in the dataset, and thus the parameter behind the subsampling probably was taking the Python related posts.