# StackOverflow_EDA_and_Data_Preparation

June 29, 2019

In [2]:
```
# general purpose packages
import pandas as pd
import numpy as np
import sqlite3
import re
import os

from datetime import datetime
import pickle

# visualization related packages
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from wordcloud import WordCloud

# text preprocessing related packages
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer

# feature extraction related packages
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

# partition data to train & test
from sklearn.model_selection import train_test_split
# saving sparse matrix
import scipy.sparse
```

## 1 Stack Overflow: Tag Prediction

1. Business Problem

1.1 Description

Description

Stack Overflow is the largest, most trusted online community for developers to learn, share their programming knowledge, and build their careers. Stack Overflow is something which every

programmer use one way or another. Each month, over 50 million developers come to Stack Overflow to learn, share their knowledge, and build their careers. It features questions and answers on a wide range of topics in computer programming. The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers in a fashion similar to a wiki or Digg. As of April 2014 Stack Overflow has over 4,000,000 registered users, and it exceeded 10,000,000 questions in late August 2015. Based on the type of tags assigned to questions, the top eight most discussed topics on the site are: Java, JavaScript, C#, PHP, Android, jQuery, Python and HTML.

Problem Statemtent

Suggest the tags based on the content that was there in the question posted on Stackoverflow.

Source: https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/

1.2 Source / useful links

Data Source : https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data Youtube : https://youtu.be/nNDqbUhtIRg Research paper : https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tagging-1.pdf Research paper : https://dl.acm.org/citation.cfm?id=2660970&dl=ACM&coll=DL

1.3 Real World / Business Objectives and Constraints

1. Predict as many tags as possible with high precision and recall.
2. Incorrect tags could impact customer experience on StackOverflow.
3. No strict latency constraints.

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

Refer: https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/data All of the data is in 2 files: Train and Test.

The questions are randomized and contains a mix of verbose text sites as well as sites related to math and programming. The number of questions from each site may vary, and no filtering has been performed on the questions (such as closed questions).

**Data Field Explaination**

Dataset contains 6,034,195 rows. The columns in the table are:

2.1.2 Example Data point

2.2 Mapping the real-world problem to a Machine Learning Problem

It is a multi-label classification problem Multi-label Classification: Multilabel classification assigns to each sample a set of target labels. This can be thought as predicting properties of a datapoint that are not mutually exclusive, such as topics that are relevant for a document. A question on Stackoverflow might be about any of C, Pointers, FileIO and/or memory-management at the same time or none of these. **Credit**: http://scikit-learn.org/stable/modules/multiclass.html

2.2.2 Performance metric

Micro-Averaged F1-Score (Mean F Score) : The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

F1 = 2 * (precision * recall) / (precision + recall)

In the multi-class and multi-label case, this is the weighted average of the F1 score of each class.

'Micro f1 score': Calculate metrics globally by counting the total true positives, false negatives and false positives. This is a better metric when we have class imbalance.

'Macro f1 score': Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

https://www.kaggle.com/wiki/MeanFScore http://scikit-learn.org/stable/modules/generated/sklearn.me
Hamming loss : The Hamming loss is the fraction of labels that are incorrectly predicted. https://www.kaggle.com/wiki/HammingLoss

## 2 Configs

```
In [3]: df_path = '/media/amd_3/20DAD539DAD50BC2/DSET_REPO/DataSets/CS06_STACK_OVERFLOW_TAG/Trai
        raw_db_path = './data/raw_data.db'
        cleaned_csv_path = './data/cleaned_data.csv'
        all_tags_csv_path = './data/all_tags_df.csv'

        # path of final features frames
        final_train_feat_path = './data/train_feat_sparse_matrix.npz'
        final_test_feat_path = './data/test_feat_sparse_matrix.npz'

        #
        sample_size= 300000 # the size of sample to be used for Final DF generation
        title_weight= 3 # the weight to be assigned for title field for vectorization
```

## 3 Read the data

```
In [11]: def load_to_db(df_path, raw_db_path):

             print(datetime.now(), ' Loading the raw data to database')

             if os.path.isfile(raw_db_path):
                 print('Database already present in the path !!!')
                 all_tags_df = pd.read_csv(all_tags_csv_path, index_col=False)
                 return all_tags_df

             # establish a connection to DB
             conn = sqlite3.connect(raw_db_path)

             # set the chunk size
             chunk_size = 100000

             # read data frame chunk by chunk
             df_chunk_reader = pd.read_csv(df_path, names=['Id', 'Title', 'Body', 'Tags'],
                                           chunksize=chunk_size, encoding='utf-8')

             # process one chunk at a time
             for chunk_number, chunk_df in enumerate(df_chunk_reader):
                 chunk_df.to_sql('raw_data', conn, if_exists='append', index=False)
```

```
            print(datetime.now(), ' Processed chunk: ', chunk_number + 1)

        # count the number of rows in the data base
        num_rows = pd.read_sql_query("""SELECT count(*) FROM raw_data""", conn)
        print('Number of rows in the database before deduping:', '\n',
              num_rows['count(*)'].values[0])

        # Dedupe the data
        df_no_dup = pd.read_sql_query("""SELECT Id, Title, Body, Tags, COUNT(*) as
                                         cnt_dup FROM raw_data GROUP BY Title, Body, Tags""
                                      conn)

        print(datetime.now(), 'dedupe operation completed ')

        print("number of duplicate questions :",
              num_rows['count(*)'].values[0]- df_no_dup.shape[0],
              "(",(1-((df_no_dup.shape[0])/(num_rows['count(*)'].values[0])))*100,"% )")


        print('Number of times each question appeared: \n',
              df_no_dup.cnt_dup.value_counts())


        # keep only the required columns
        df_no_dup = df_no_dup[['Id', 'Title', 'Body', 'Tags']]

        # create a data frame of all tags df
        all_tags_df = df_no_dup[['Id', 'Tags']]
        all_tags_df.to_csv(all_tags_csv_path, index=False)

        df_no_dup.to_sql('raw_data', conn, if_exists='replace', index=False)
        print(datetime.now(), ' Completed loading of raw data to DB')

        # close the connection
        conn.close()

        return all_tags_df
```

In [12]:
```
all_tags_df = load_to_db(df_path, raw_db_path)
```

```
2019-06-25 00:17:17.157805  Loading the raw data to database
2019-06-25 00:17:23.779355  Processed chunk:  1
2019-06-25 00:17:29.241675  Processed chunk:  2
2019-06-25 00:17:34.722642  Processed chunk:  3
2019-06-25 00:17:40.121638  Processed chunk:  4
2019-06-25 00:17:45.494259  Processed chunk:  5
2019-06-25 00:17:51.231266  Processed chunk:  6
2019-06-25 00:17:56.858416  Processed chunk:  7
```

```
2019-06-25 00:18:02.142860  Processed chunk:  8
2019-06-25 00:18:07.866116  Processed chunk:  9
2019-06-25 00:18:13.454907  Processed chunk:  10
2019-06-25 00:18:18.797847  Processed chunk:  11
2019-06-25 00:18:24.587638  Processed chunk:  12
2019-06-25 00:18:30.222038  Processed chunk:  13
2019-06-25 00:18:35.818718  Processed chunk:  14
2019-06-25 00:18:41.417359  Processed chunk:  15
2019-06-25 00:18:47.008024  Processed chunk:  16
2019-06-25 00:18:52.655092  Processed chunk:  17
2019-06-25 00:18:58.291011  Processed chunk:  18
2019-06-25 00:19:03.975847  Processed chunk:  19
2019-06-25 00:19:09.539295  Processed chunk:  20
2019-06-25 00:19:15.292618  Processed chunk:  21
2019-06-25 00:19:20.935390  Processed chunk:  22
2019-06-25 00:19:25.988573  Processed chunk:  23
2019-06-25 00:19:31.518365  Processed chunk:  24
2019-06-25 00:19:37.139675  Processed chunk:  25
2019-06-25 00:19:42.746568  Processed chunk:  26
2019-06-25 00:19:48.317718  Processed chunk:  27
2019-06-25 00:19:54.019963  Processed chunk:  28
2019-06-25 00:19:59.543530  Processed chunk:  29
2019-06-25 00:20:06.165705  Processed chunk:  30
2019-06-25 00:20:11.996643  Processed chunk:  31
2019-06-25 00:20:17.864587  Processed chunk:  32
2019-06-25 00:20:24.198337  Processed chunk:  33
2019-06-25 00:20:29.688234  Processed chunk:  34
2019-06-25 00:20:35.463909  Processed chunk:  35
2019-06-25 00:20:41.028179  Processed chunk:  36
2019-06-25 00:20:46.538084  Processed chunk:  37
2019-06-25 00:20:52.214990  Processed chunk:  38
2019-06-25 00:20:57.987430  Processed chunk:  39
2019-06-25 00:21:03.711676  Processed chunk:  40
2019-06-25 00:21:10.567795  Processed chunk:  41
2019-06-25 00:21:16.167951  Processed chunk:  42
2019-06-25 00:21:21.879857  Processed chunk:  43
2019-06-25 00:21:28.353104  Processed chunk:  44
2019-06-25 00:21:34.016077  Processed chunk:  45
2019-06-25 00:21:40.018170  Processed chunk:  46
2019-06-25 00:21:45.676298  Processed chunk:  47
2019-06-25 00:21:51.310538  Processed chunk:  48
2019-06-25 00:21:57.710072  Processed chunk:  49
2019-06-25 00:22:03.953905  Processed chunk:  50
2019-06-25 00:22:09.737008  Processed chunk:  51
2019-06-25 00:22:15.260773  Processed chunk:  52
2019-06-25 00:22:20.974857  Processed chunk:  53
2019-06-25 00:22:26.488638  Processed chunk:  54
2019-06-25 00:22:32.752469  Processed chunk:  55
```

```
2019-06-25 00:22:38.712969  Processed chunk:  56
2019-06-25 00:22:44.927671  Processed chunk:  57
2019-06-25 00:22:50.566247  Processed chunk:  58
2019-06-25 00:22:56.390840  Processed chunk:  59
2019-06-25 00:23:02.110278  Processed chunk:  60
2019-06-25 00:23:04.456308  Processed chunk:  61
Number of rows in the database before deduping:
 6034196
2019-06-25 00:38:12.127770 dedupe operation completed
number of duplicate questions : 1827881 ( 30.292038906260256 % )
Number of times each question appeared:
 1    2656284
2    1272336
3     277575
4         90
5         25
6          5
Name: cnt_dup, dtype: int64
2019-06-25 01:07:19.271026  Completed loading of raw data to DB
```

# 4 Exploratoty Data Analysis

```
In [4]: all_tags_df = pd.read_csv(all_tags_csv_path, index_col=False)
        print(all_tags_df.shape)
        all_tags_df.head()
```

```
(4206315, 2)
```

```
/home/amd_3/anaconda3/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2785: DtypeWa
  interactivity=interactivity, compiler=compiler, result=result)
```

```
Out[4]:         Id                                     Tags
        0  1078065                                    c++ c
        1   940626          c# silverlight data-binding
        2  1484628  c# silverlight data-binding columns
        3  1074875                              jsp jstl
        4  3954566                             java jdbc
```

```
In [11]: print('Id column contains NaN :', all_tags_df['Id'].isnull().any())
         print('Tags column contains NaN :', all_tags_df['Tags'].isnull().any())
```

```
Id column contains NaN : False
Tags column contains NaN : True
```

```
In [12]: all_tags_df = all_tags_df.fillna('__')
         print('Tags column contains NaN :', all_tags_df['Tags'].isnull().any())
```

```
Tags column contains NaN : False
```

### 4.1 1. Number of Unique Tags

```
In [25]: vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='true')
         tag_vectorized = vectorizer.fit_transform(all_tags_df['Tags'])
```

```
In [26]: print('Total number of questions ', tag_vectorized.shape[0])
         print('Total number of Tags ', tag_vectorized.shape[1])
```

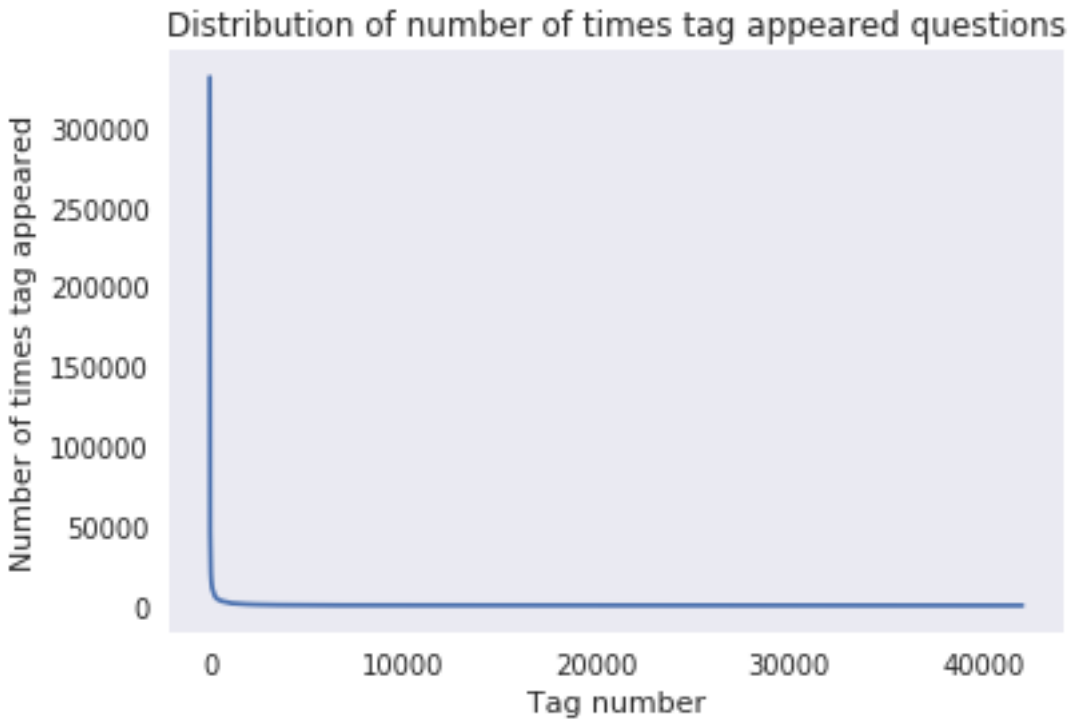
```
Total number of questions  4206315
Total number of Tags  42049
```

### 4.2 2. Number of times a Tag appeared

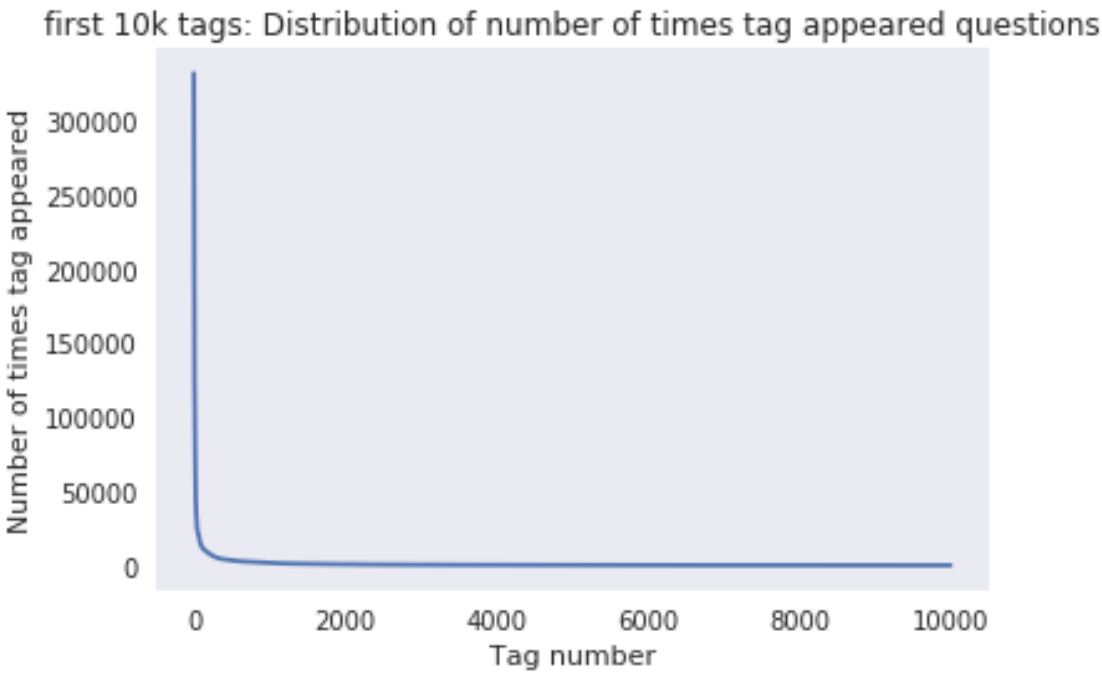
```
In [27]: tag_count_dict = dict(zip(vectorizer.get_feature_names(),
                                   tag_vectorized.sum(axis=0).A1))
         tag_count_df = pd.DataFrame({'Tag':vectorizer.get_feature_names(),
                                      'Count':tag_vectorized.sum(axis=0).A1},
                                     columns=['Tag', 'Count'])
         tag_count_df = tag_count_df.sort_values(['Count'], ascending=False)
         tag_count_df.head()
```

```
Out[27]:
                Tag   Count
4338            c#  331505
18070         java  299414
27250          php  284103
18158  javascript  265423
1235      android  235436
```

```
In [28]: plt.plot(tag_count_df['Count'].values)
         #plt.xticks(tag_count_df['Tag'].values)
         plt.title("Distribution of number of times tag appeared questions")
         plt.grid()
         plt.xlabel("Tag number")
         plt.ylabel("Number of times tag appeared")
         plt.show()
         plt.close()
```



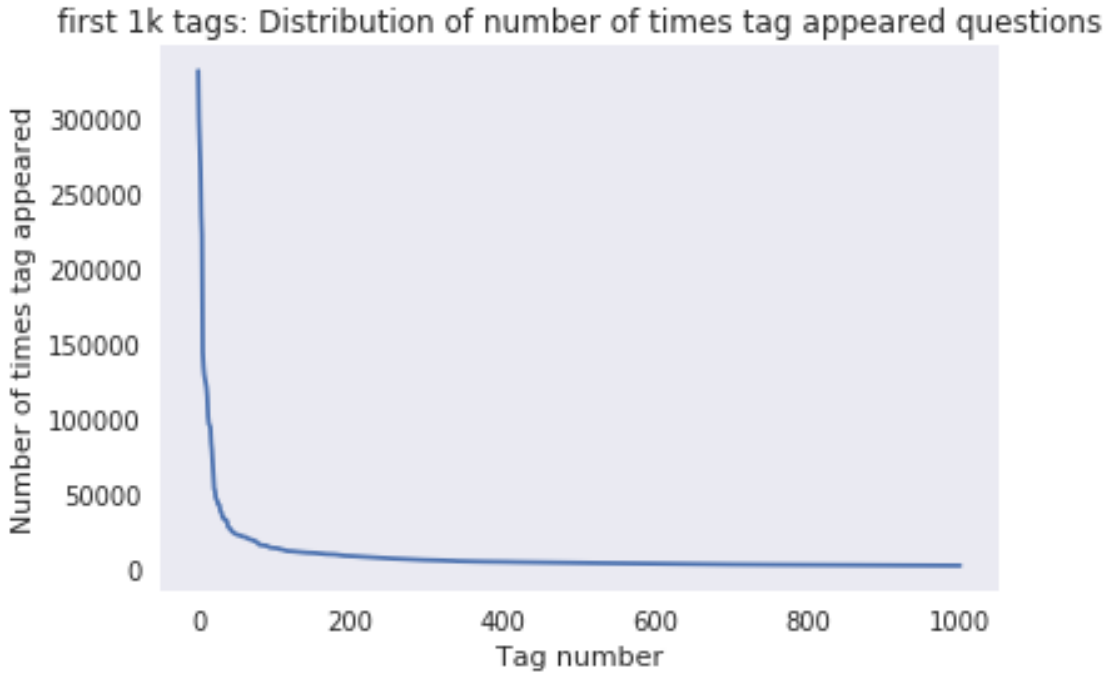

```
In [29]: plt.plot(tag_count_df['Count'].values[0:10000])
         plt.title('first 10k tags: Distribution of number of times tag appeared questions')
         plt.grid()
         plt.xlabel("Tag number")
         plt.ylabel("Number of times tag appeared")
         plt.show()
         plt.close()
         print(len(tag_count_df['Count'].values[0:10000:25]),
               tag_count_df['Count'].values[0:10000:25])
```




```
400 [331505   44829   22429   17728   13364   11162   10029    9148    8054    7151
    6466    5865    5370    4983    4526    4281    4144    3929    3750    3593
    3453    3299    3123    2986    2891    2738    2647    2527    2431    2331
    2259    2186    2097    2020    1959    1900    1828    1770    1723    1673
    1631    1574    1532    1479    1448    1406    1365    1328    1300    1266
    1245    1222    1197    1181    1158    1139    1121    1101    1076    1056
    1038    1023    1006     983     966     952     938     926     911     891
     882     869     856     841     830     816     804     789     779     770
     752     743     733     725     712     702     688     678     671     658
     650     643     634     627     616     607     598     589     583     577
     568     559     552     545     540     533     526     518     512     506
     500     495     490     485     480     477     469     465     457     450
     447     442     437     432     426     422     418     413     408     403
     398     393     388     385     381     378     374     370     367     365
     361     357     354     350     347     344     342     339     336     332
     330     326     323     319     315     312     309     307     304     301
     299     296     293     291     289     286     284     281     278     276
     275     272     270     268     265     262     260     258     256     254
     252     250     249     247     245     243     241     239     238     236
     234     233     232     230     228     226     224     222     220     219
     217     215     214     212     210     209     207     205     204     203
     201     200     199     198     196     194     193     192     191     189
     188     186     185     183     182     181     180     179     178     177
     175     174     172     171     170     169     168     167     166     165
     164     162     161     160     159     158     157     156     156     155
```

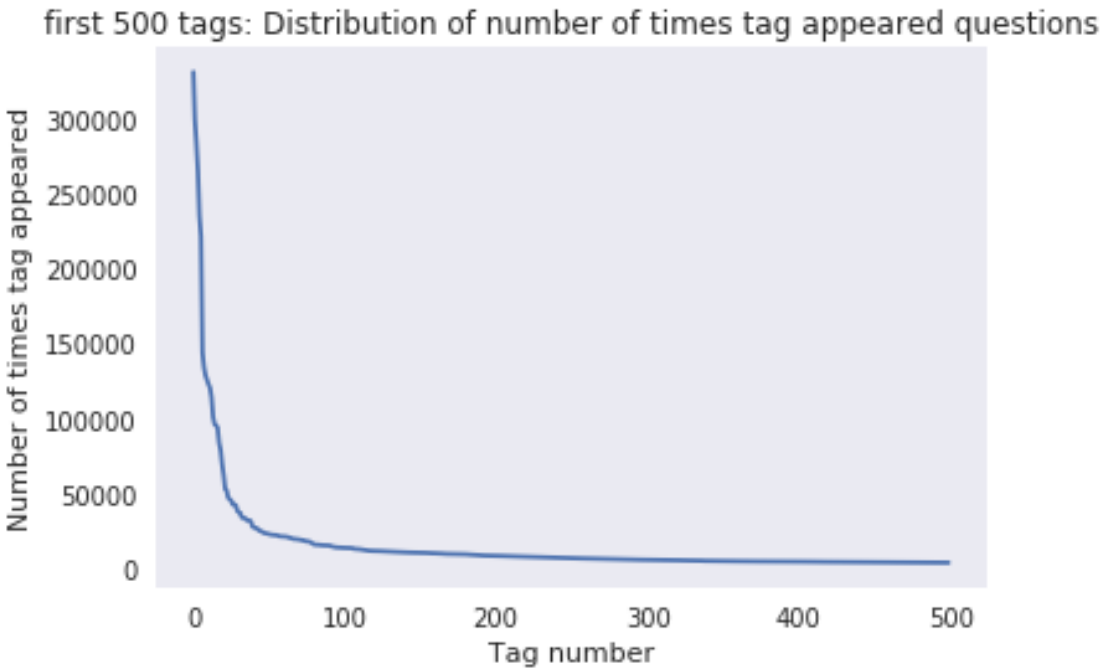
```
154   153   152   151   150   149   149   148   147   146
145   144   143   142   142   141   140   139   138   137
137   136   135   134   134   133   132   131   130   130
129   128   128   127   126   126   125   124   124   123
123   122   122   121   120   120   119   118   118   117
117   116   116   115   115   114   113   113   112   111
111   110   109   109   108   108   107   106   106   106
105   105   104   104   103   103   102   102   101   101
100   100    99    99    98    98    97    97    96    96
 95    95    94    94    93    93    93    92    92    91
 91    90    90    89    89    88    88    87    87    86
 86    86    85    85    84    84    83    83    83    82
 82    82    81    81    80    80    80    79    79    78
 78    78    78    77    77    76    76    76    75    75
 75    74    74    74    73    73    73    73    72    72]
```

In [30]:
```
plt.plot(tag_count_df['Count'].values[0:1000])
plt.title('first 1k tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
plt.close()
print(len(tag_count_df['Count'].values[0:1000:5]),
      tag_count_df['Count'].values[0:1000:5])
```



```
200 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   7867   7702   7564   7274   7151   7052   6847   6656   6553
   6466   6291   6183   6093   5971   5865   5760   5577   5490   5411
   5370   5283   5207   5107   5066   4983   4891   4785   4658   4549
   4526   4487   4429   4335   4310   4281   4239   4228   4195   4159
   4144   4088   4050   4002   3957   3929   3874   3849   3818   3797
   3750   3703   3685   3658   3615   3593   3564   3521   3505   3483
   3453   3427   3396   3363   3326   3299   3272   3232   3196   3168
   3123   3094   3073   3050   3012   2986   2983   2953   2934   2903
   2891   2844   2819   2784   2754   2738   2726   2708   2681   2669
   2647   2621   2604   2594   2556   2527   2510   2482   2460   2444
   2431   2409   2395   2380   2363   2331   2312   2297   2290   2281
   2259   2246   2222   2211   2198   2186   2162   2142   2132   2107
   2097   2078   2057   2045   2036   2020   2011   1994   1971   1965
   1959   1952   1940   1932   1912   1900   1879   1865   1855   1841
   1828   1821   1813   1801   1782   1770   1760   1747   1741   1734
   1723   1707   1697   1688   1683   1673   1665   1656   1646   1639]
```

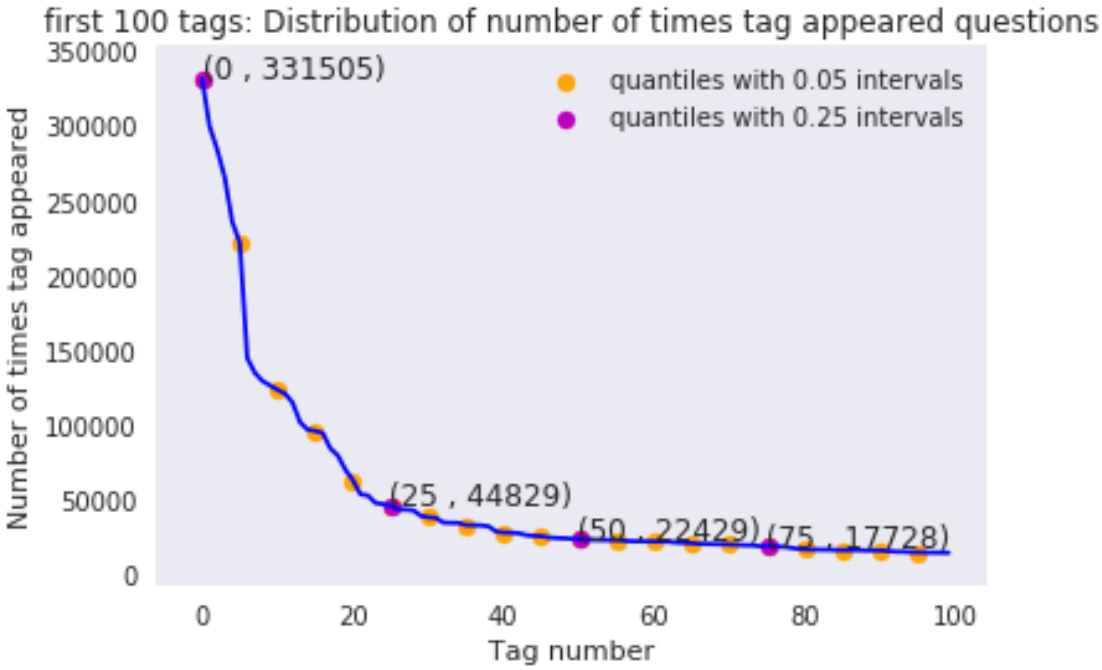
```
In [31]: plt.plot(tag_count_df['Count'].values[0:500])
         plt.title('first 500 tags: Distribution of number of times tag appeared questions')
         plt.grid()
         plt.xlabel("Tag number")
         plt.ylabel("Number of times tag appeared")
         plt.show()
         plt.close()
         print(len(tag_count_df['Count'].values[0:500:5]),
               tag_count_df['Count'].values[0:500:5])
```




```
100 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703
  13364  13157  12407  11658  11228  11162  10863  10600  10350  10224
  10029   9884   9719   9411   9252   9148   9040   8617   8361   8163
   8054   7867   7702   7564   7274   7151   7052   6847   6656   6553
   6466   6291   6183   6093   5971   5865   5760   5577   5490   5411
   5370   5283   5207   5107   5066   4983   4891   4785   4658   4549
   4526   4487   4429   4335   4310   4281   4239   4228   4195   4159
   4144   4088   4050   4002   3957   3929   3874   3849   3818   3797
   3750   3703   3685   3658   3615   3593   3564   3521   3505   3483]
```

```
In [32]: plt.plot(tag_count_df['Count'].values[0:100], c='b')
         plt.scatter(x=list(range(0,100,5)), y=tag_count_df['Count'].values[0:100:5],
                     c='orange', label='quantiles with 0.05 intervals')
         # quantiles with 0.25 difference
         plt.scatter(x=list(range(0,100,25)), y=tag_count_df['Count'].values[0:100:25],
                     c='m', label='quantiles with 0.25 intervals')

         for x,y in zip(list(range(0,100,25)), tag_count_df['Count'].values[0:100:25]):
             plt.annotate(s="({} , {})".format(x,y), xy=(x,y), xytext=(x-0.05, y+500))

         plt.title('first 100 tags: Distribution of number of times tag appeared questions')
         plt.grid()
         plt.xlabel('Tag number')
```

```
plt.ylabel('Number of times tag appeared')
plt.legend()
plt.show()
plt.close()
print(len(tag_count_df['Count'].values[0:100:5]),
      tag_count_df['Count'].values[0:100:5])
```




```
20 [331505 221533 122769  95160  62023  44829  37170  31897  26925  24537
  22429  21820  20957  19758  18905  17728  15533  15097  14884  13703]
```

```
In [33]: # Store tags greater than 10K in one list
         lst_tags_gt_10k = tag_count_df[tag_count_df['Count']> 10000].Tag
         #Print the length of the list
         print ('{} Tags are used more than 10000 times'.format(len(lst_tags_gt_10k)))
         # Store tags greater than 100K in one list
         lst_tags_gt_100k = tag_count_df[tag_count_df['Count']>100000].Tag
         #Print the length of the list.
         print ('{} Tags are used more than 100000 times'.format(len(lst_tags_gt_100k)))
```

```
153 Tags are used more than 10000 times
14 Tags are used more than 100000 times
```

Observations: 1. There are total 153 tags which are used more than 10000 times. 2. 14 tags are used more than 100000 times. 3. Most frequent tag (i.e. c#) is used 331505 times. 4. Since some tags occur much more frequenctly than others, Micro-averaged F1-score is the appropriate metric for this probelm.

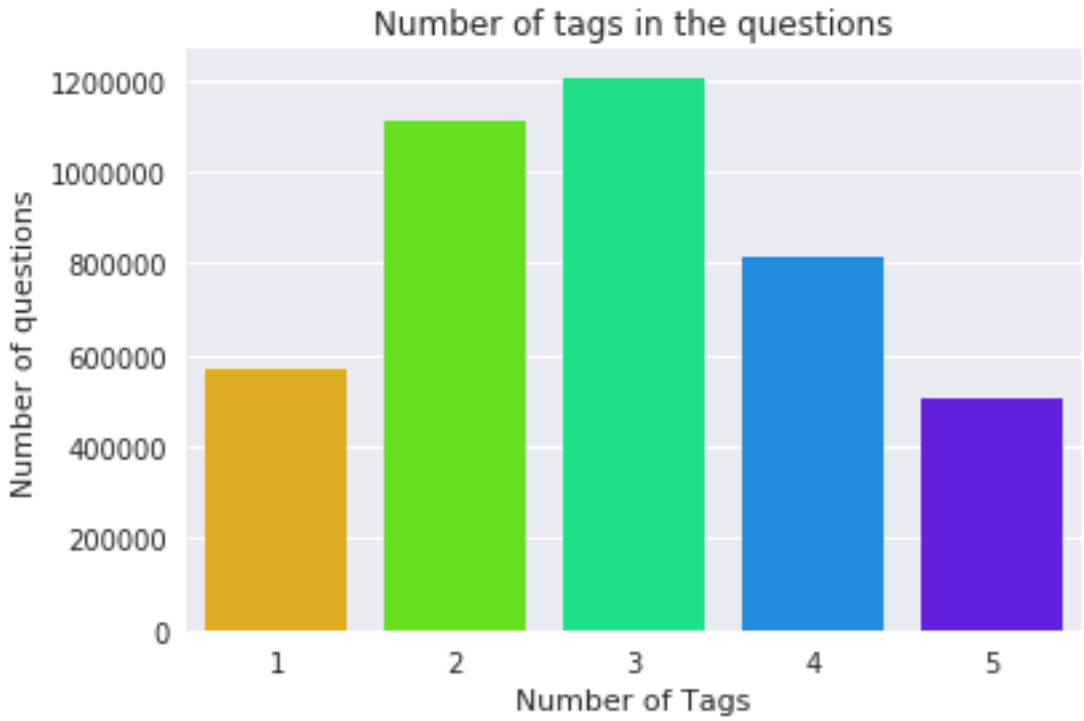
### 4.3 3.2.4 Tags Per Question

```
In [34]: tag_per_question = tag_vectorized.sum(axis=1).A1

         question_tag_count_df = pd.DataFrame({'Question': range(0, tag_per_question.shape[0]),
                                               'Tag_Count':tag_per_question})
```

```
In [35]: print('Maximum number of tags per question: %d'%(question_tag_count_df['Tag_Count'].max
         print('Minimum number of tags per question: %d'%(question_tag_count_df['Tag_Count'].min
         print('Avg. number of tags per question: %f'%(question_tag_count_df['Tag_Count'].mean()
```

```
Maximum number of tags per question: 5
Minimum number of tags per question: 1
Avg. number of tags per question: 2.899439
```

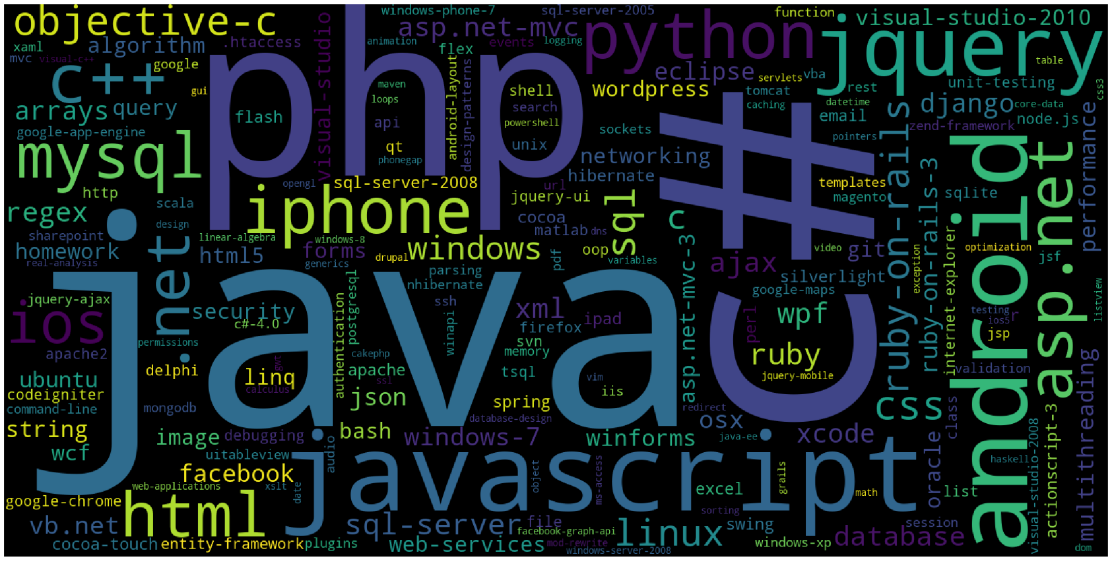
```
In [36]: sns.countplot(data=question_tag_count_df, x='Tag_Count',
                      palette='gist_rainbow')
         plt.title('Number of tags in the questions')
         plt.xlabel('Number of Tags')
         plt.ylabel('Number of questions')
         plt.show()
         plt.close()
```




Observations: 1. Maximum number of tags per question: 5 2. Minimum number of tags per question: 1 3. Avg. number of tags per question: 2.899 4. Most of the questions are having 2 or 3 tags

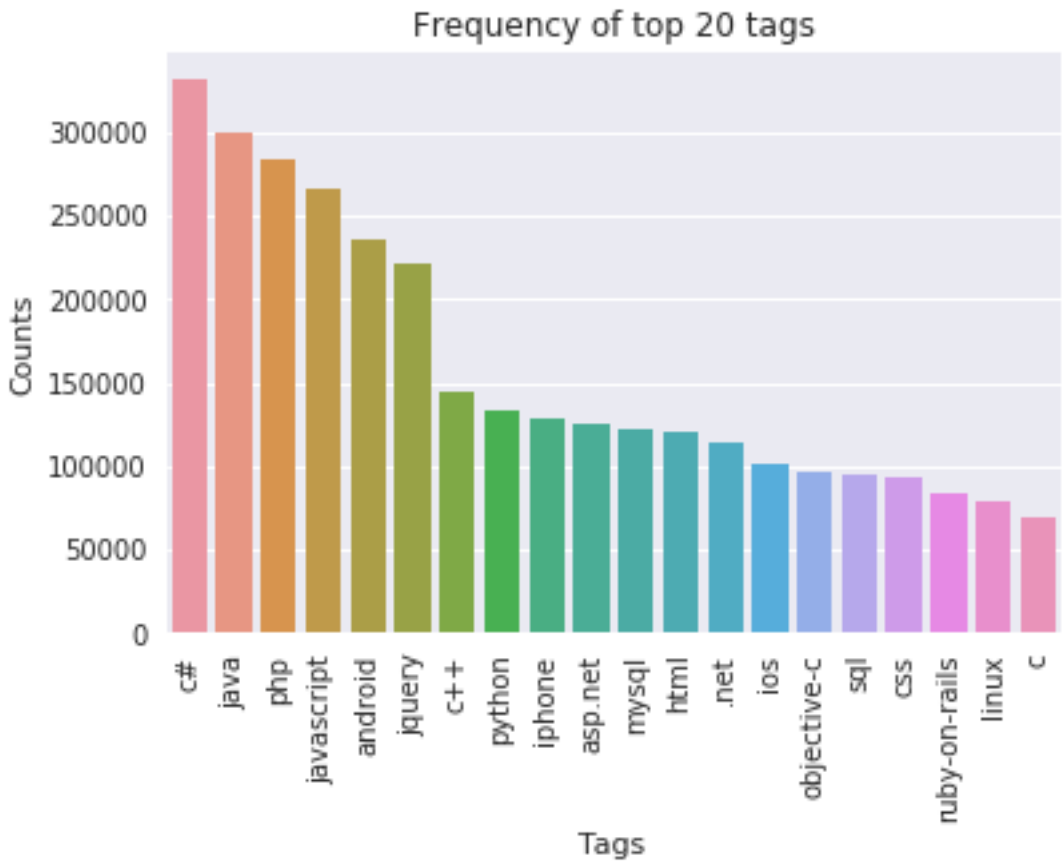
## 4.4 3.2.5 Most Frequent Tags

```
In [37]: # Ploting word cloud
         start = datetime.now()

         #Initializing WordCloud using frequencies of tags.
         wordcloud = WordCloud(background_color='black',
                               width=1600,
                                   height=800,
                              ).generate_from_frequencies(tag_count_dict)

         fig = plt.figure(figsize=(30,20))
         plt.imshow(wordcloud)
         plt.axis('off')
         plt.tight_layout(pad=0)
         fig.savefig('./data/tag.png', dpi=300)
         plt.show()
         plt.close()
```




Observations: A look at the word cloud shows that "c#", "java", "php", "asp.net", "javascript", "c++" are some of the most frequent tags.

```
In [38]: sns.barplot(x='Tag', y='Count', data=tag_count_df.iloc[0:20])
         plt.xlabel('Tags')
         plt.ylabel('Counts')
         plt.title('Frequency of top 20 tags')
         plt.xticks(rotation=90)
         plt.show()
         plt.close()
```




Observations: 1. Majority of the most frequent tags are programming language. 2. C# is the top most frequent programming language. 3. Android, IOS, Linux and windows are among the top most frequent operating systems.

### 4.5 Questions Covered by Tags

```
In [39]: # create dictionary having tag name & its index
         tag_index_dict = {tag:index for index, tag
                                 in enumerate(vectorizer.get_feature_names())}

         # get row, column indices of non-zero entries in CSR matrix
         row_indices, col_indices = tag_vectorized.nonzero()
         non_zero_indices_df = pd.DataFrame(list(zip(row_indices, col_indices)),
                                            columns=['row', 'column'])
         non_zero_indices_df.head()
```

Out[39]:

| | row | column |
|---|---|---|
| 0 | 0 | 4337 |
| 1 | 0 | 4347 |
| 2 | 1 | 7927 |

```
3    1   33112
4    1    4338
```

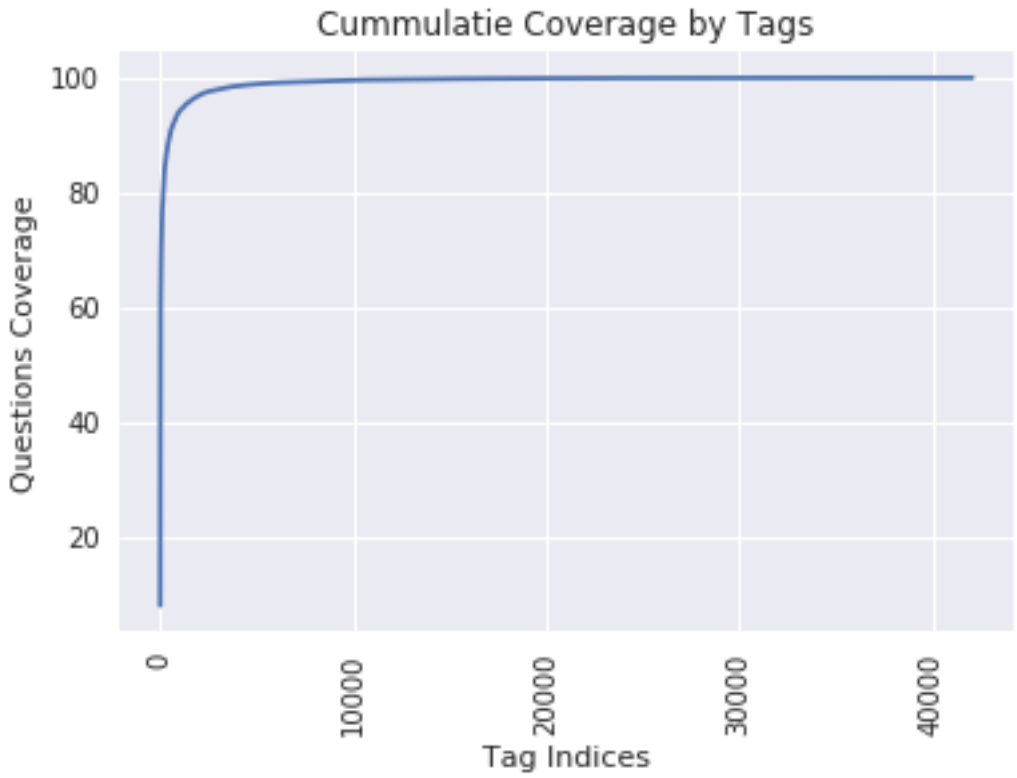
In [5]:
```
if os.path.exists('./data/Coverage_info.csv'):
    coverage_info_df = pd.read_csv('./data/Coverage_info.csv', index_col=False)

else:
    covered_questions_dict = dict()
    total_ques = tag_vectorized.shape[0]
    coverage_info_list = list()

    # reset index of tag_count df
    tag_count_df = tag_count_df.reset_index(drop=True)

    for index, row in tag_count_df.iterrows():
        # get index of tag
        tag_index = tag_index_dict[row['Tag']]
        # get total number of questions covered by this
        temp_df = non_zero_indices_df[non_zero_indices_df['column'] == tag_index]
        questions_indices = set(temp_df['row'])

        # add to covered questions
        for item in questions_indices:
            covered_questions_dict[item] = 1

        # get cummultive coverage so far
        coverage = (len(covered_questions_dict) * 100.0) / total_ques

        coverage_info_list.append((row['Tag'], coverage,))

        if (index+1) % 100000 == 0:
            print(datetime.now(), 'Processed rows', index+1)

        #print('Questions coverage by top %d tags : %f '%(index+1, coverage,))
    coverage_info_df = pd.DataFrame(coverage_info_list,
                                    columns=['Tag', 'Cumm_Coverage'])

    coverage_info_df.to_csv('./data/Coverage_info.csv', index=False)

coverage_info_df.head()
```

Out[5]:

| | Tag | Cumm_Coverage |
|---|---|---|
| 0 | c# | 7.881126 |
| 1 | java | 14.932548 |
| 2 | php | 21.619826 |
| 3 | javascript | 27.287519 |
| 4 | android | 32.041847 |

In [45]:
```
plt.plot(coverage_info_df['Cumm_Coverage'])
```

```
x_ticks = coverage_info_df['Tag'].tolist()
#plt.set_xticklabels(coverage_info_df['Tag'])
plt.xticks(rotation=90)
plt.xlabel('Tag Indices')
plt.ylabel('Questions Coverage')
plt.title('Cummulatie Coverage by Tags')
plt.show()
plt.close()
```




In [6]:
```
# we select the tags that cover upto 95%
#coverage_info_df = coverage_info_df[coverage_info_df['Cumm_Coverage'] <= 95.0]
# selecting the top 500 tags
coverage_info_df = coverage_info_df.iloc[0:500]
coverage_info_df.tail()
```

Out[6]:

| | Tag | Cumm_Coverage |
|---|---|---|
| 495 | amazon-s3 | 89.952845 |
| 496 | attributes | 89.956221 |
| 497 | reference-request | 90.007406 |
| 498 | functions | 90.051981 |
| 499 | pdo | 90.053051 |

```
In [7]: required_tags = sorted(list(set(coverage_info_df['Tag'])))
        print('Total tags selected', len(required_tags))

        # save model to disk
        pickle_out = open("./data/so_multilabels.pkl","wb")
        pickle.dump(required_tags, pickle_out)
        pickle_out.close()
```

```
Total tags selected 500
```

```
In [9]: print("some tags selected :\n", required_tags[50:100])
```

```
some tags selected :
 ['backup', 'bash', 'batch', 'batch-file', 'binding', 'blackberry', 'bluetooth', 'boost', 'boot'
```

```
In [13]: required_tags_set = set(required_tags)
         # prepare a boolean array for the rows which contain the required tags
         required_indices = [True if set(item.split()).intersection(required_tags_set) else Fals
                             in all_tags_df['Tags']]
         print(required_indices[0:4])
```

```
[True, True, True, True]
```

## 5 Sample from DB & Clean it

```
In [36]: stop_words = set(stopwords.words('english'))
         stemmer = SnowballStemmer('english')
         html_tags = re.compile('<.*?>')
         special_chars = re.compile(r'[^A-Za-z#+]')
```

```
In [37]: def clean_text(title, body_text, title_weight=1):

             # get code section from the body text
             code = str(re.findall(r'<code>(.*?)</code>', body_text,
                                   flags=re.DOTALL))

             # get non code section
             question = re.sub('<code>(.*?)</code>', ' ', body_text,
                               flags=re.MULTILINE|re.DOTALL)

             # concatenate the title & question
             if title_weight == 1:
                 text_data = title + ' ' + question
             else:
                 text_data =str()
```

```
            for _ in range(0, title_weight):
                text_data += ' ' + title

            text_data = text_data + ' ' + question

        # remove HTML tags
        text_data = re.sub(html_tags, ' ', text_data)

        # remove special characters
        text_data = re.sub(special_chars, ' ', text_data)

        # convert to lower case
        text_data = text_data.lower()

        # remove stop words
        text_data_words = text_data.split()
        text_data_words = list(filter(lambda x : x not in stop_words,
                                      text_data_words))

        # remove single letter words except c
        text_data_words = list(filter(lambda x : x=='c' or len(x) > 1,
                                      text_data_words))

        # Find root word - stem the word
        text_data_words = [stemmer.stem(item) for item in text_data_words]

        # combine all words with space to get the cleaned text
        cleaned_text = ' '.join(text_data_words)

        return (cleaned_text, code,)
```

In [38]:
```
def sample_and_clean_DB(db_in_path, required_indices, sample_size, title_weight=1):

    if os.path.isfile(cleaned_csv_path):
        print('Cleaned data csv already present in the path !!!')
        return

    conn = sqlite3.connect(db_in_path)
    df_no_dup = pd.read_sql_query("""SELECT Id, Title, Body, Tags FROM raw_data""",
                                  conn)
    conn.close()

    # select only those rows where atleast one required tag is present
    print('Number of rows in the entire dataset :', df_no_dup.shape[0])
    df_no_dup = df_no_dup[required_indices]
    print('Number of rows selected based on required tags:', df_no_dup.shape[0])
```

```
            # take sample of data frame if opted
            sample_count = min(sample_size, df_no_dup.shape[0])
            df_no_dup = df_no_dup.sample(n=sample_count)
            df_no_dup = df_no_dup.reset_index(drop=True)
            # save id into a set
            id_list = df_no_dup['Id']
            tag_list = df_no_dup['Tags']

            print(datetime.now(), ' Sample of size %d taken'%(sample_count,))

            # STEP 2 : Separate out text & code from Body, also combine Title & Question as tex
            print(datetime.now(), ' Cleaning data frame started !!!')
            df_no_dup = df_no_dup.apply(lambda x : clean_text(x['Title'], x['Body'], title_weig
                                        axis=1)
            print(datetime.now(), ' Cleaning data frame completed !!!')

            # separate out code and text
            df_no_dup_txt = [item[0] for item in df_no_dup]
            df_no_dup_code = [item[1] for item in df_no_dup]

            # create cleaned data frame
            df_no_dup = pd.DataFrame({ 'Id' : id_list,
                                       'Question' : df_no_dup_txt,
                                      'Code' : df_no_dup_code,
                                      'Tags' : tag_list},
                                      index=range(df_no_dup.shape[0]))

            # re-order the column name
            df_no_dup = df_no_dup[['Id', 'Question', 'Code', 'Tags']]

            # get the number of rows where code is present
            code_count = len(list(filter(lambda x : x != str(), df_no_dup['Code'])))
            per_info = (code_count/ df_no_dup.shape[0]) * 100.0
            print('Size of the Sampled data frame: ', df_no_dup.shape[0])
            print('Number of rows having code: %d , Percentage: %f'%(code_count,
                                                                     per_info,))

            # dump the deduped data to disk
            df_no_dup.to_csv(cleaned_csv_path, index=False)
            #
            print('Cleaned & Sampled DF :\n', df_no_dup.head())

            print(datetime.now(), ' Done !!!')
```

In [39]:
```
sample_and_clean_DB(raw_db_path, required_indices, sample_size, title_weight)
```

```
Number of rows in the entire dataset : 4206315
Number of rows selected based on required tags: 3787914
```

```
2019-06-26 21:01:02.937691  Sample of size 300000 taken
2019-06-26 21:01:02.938300  Cleaning data frame started !!!
2019-06-26 21:05:28.440158  Cleaning data frame completed !!!
Size of the Sampled data frame:  300000
Number of rows having code: 300000 , Percentage: 100.000000
Cleaned & Sampled DF :
         Id                                           Question  \
0  2909527  adob effect script use chroma key filter layer...
1  2895202  run applic eclips generat class find load exce...
2  1345329  get post valu textarea use tinymc get post val...
3  5449688  retri method call generic way retri method cal...
4  1713764  forc repaint wxpython canva forc repaint wxpyt...

                                                 Code  \
0                                                  []
1  ['&lt;directory&gt;path_to_target_folder_on_ra...
2  [' tinyMCE.init({\n    // General options\n   ...
3  ['try {\n ClassA objA = remoteServiceA.call(pa...
4                                                  []

                                          Tags
0    scripting effects extendscript after-effects
1                          java eclipse maven run
2                        php jquery html tinymce
3  java reflection frameworks openframeworks retry
4                        python drawing wxpython
2019-06-26 21:05:36.198371  Done !!!
```

## 6 Prepare Train, Test Data Sets

```
In [45]: def prepare_datasets(cleaned_csv_path):

             if os.path.exists(final_train_feat_path) and os.path.exists(final_test_feat_path):
                 print('Final Train, Test data frames features already found in the path !!!')
                 return


             print(datetime.now(), 'Preparing the final features of train, test data frames')
             df = pd.read_csv(cleaned_csv_path, index_col=False)

             # partition to train test data
             df_train, df_test = train_test_split(df, test_size=0.30, shuffle=False)
             df_train = df_train.reset_index(drop=True)
             df_test = df_test.reset_index(drop=True)
             # save train test data to disk
             df_train[['Id', 'Tags']].to_csv('./data/Final_train_df_label.csv', index=False)
```

```
    df_test[['Id', 'Tags']].to_csv('./data/Final_test_df_label.csv', index=False)

    # slice the dataframe to required columns
    df_train = df_train['Question']
    df_test = df_test['Question']

    # create text vectorizer object
    vectorizer = CountVectorizer(min_df=0.0005, max_df=0.95, max_features=20000,
                                 ngram_range=(1,4), tokenizer=lambda x : x.split())


    # 1- Featurization of the questions data

    # fit to the train data
    print(datetime.now(), ' Start: Vectorizing train, test questions ...')
    vectorizer.fit(df_train)
    feat_names_list = list(vectorizer.get_feature_names())

    # save the feature names list to disk
    pickle_out = open("./data/feature_names_list.pkl","wb")
    pickle.dump(feat_names_list, pickle_out)
    pickle_out.close()

    # transform the feature columns
    df_train = vectorizer.transform(df_train)
    df_test = vectorizer.transform(df_test)
    print(datetime.now(), ' End: Vectorizing train questions ')

    print('Train DF shape :', df_train.shape)
    print('Test DF shape :', df_test.shape)

    # write the train, test numpy sparse matrices to disk
    scipy.sparse.save_npz(final_train_feat_path, df_train)
    scipy.sparse.save_npz(final_test_feat_path, df_test)

    print(datetime.now(), ' Done !!!')
```

In [46]:
```
prepare_datasets(cleaned_csv_path)
```

```
2019-06-26 21:36:08.640200 Preparing the final features of train, test data frames
2019-06-26 21:36:12.058076  Start: Vectorizing train, test questions ...
2019-06-26 21:39:59.634761  End: Vectorizing train questions
Train DF shape : (210000, 12526)
Test DF shape : (90000, 12526)
2019-06-26 21:40:08.274760  Done !!!
```

# 7 Procedure Summary

Read raw data from the csv file and save it to DB

Basic EDA based on number of tags, questions, tags per questions, most frequent tags etc

Idenified the top tags that has maximal coverage, the tags that covers maority of the questions

Due to limited computation power restricted the number of top tags to 500 and the sample size of 0.3 M

Cleaned all the data points in the sample dataset

Given high weightage to title than the question (3:1) to improve the performance of model

The data set is partitioned into train & test

The text data is vectorized using count vectorizer

The final train, test dataset is prepared for multilabeled classification

# 8 Conclusion

The top 500 tags selected covers 90+ % of the questions

The train, test dataset for multi-labeled classification problem is prepared