Identifying keywords, topics and summaries using text data from Stack Exchange sites

Sayan Das

1. Introduction and background

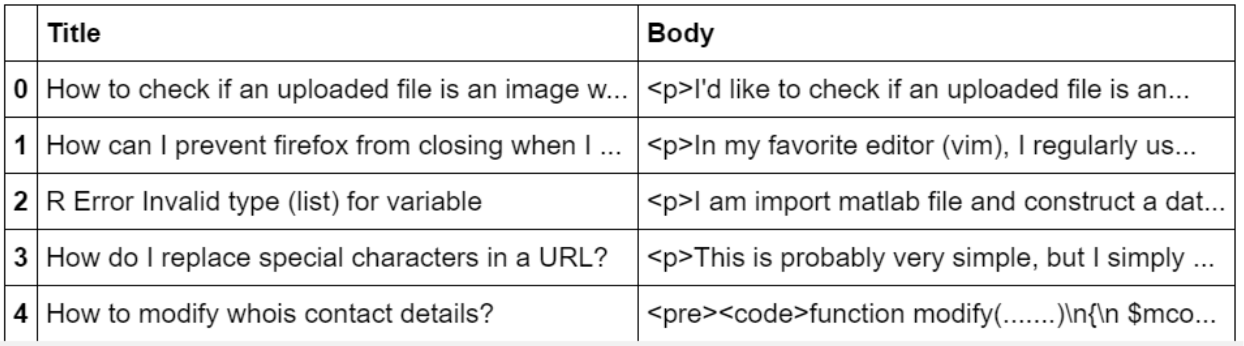
Stack Exchange and its associated websites provide a massive platform for sharing technical knowledge and collaboration on projects. Encompassing actively used sites such as *Stack Overflow*, *Super Users* and *Ask Ubuntu*, the Stack Exchange environment is a network for questions and answers across a wide range of topics. These forums serve as hubs for basic as well as complex queries providing a rich collection of technical text data.

Such information often covers important topics and the latest trends in various industries. A way to summarize this text as well as identify topics in the data could help search engines sort user queries and provide quicker and more accurate results. This would not only help the companies hosting the platform optimize their results but also help other similar forums categorize future questions in an automated manner where users can more easily find solutions to their queries. In this project, we propose topic modeling techniques to analyze this text data and identify keywords related to each question. As part of the analysis, different algorithms such as *Latent Dirichlet Allocation (LDA)* as well as *lda2vec* using word embedding are evaluated in order to optimize the categorization.

Stack Exchange as well as similar platforms such as *Quora*, *Reddit*, *LinkedIn* groups, *Facebook* groups etc, could use this model or even adapt the model on their respective platforms to categorize text data. Since text data is platform agnostic, this model and analysis can be extended to other sites and search engines such as Google and Bing. The analysis can also be applied to any sort of text data such as comments, reviews, item descriptions in platforms such as Yelp, Amazon, Twitter etc.

2. Data Description

The dataset is acquired from Kaggle and consists of questions from Stack Exchange users, spanning **420,668 rows**. Each instance contains a column that has the question title and another column containing the body of the question which could consist of text as well as code. The text in the body is in HTML format which was processed to extract only the meaningful text. A snippet of the dataset is shown in *Fig 1*.



	Title	Body
0	How to check if an uploaded file is an image w...	<p>I'd like to check if an uploaded file is an...
1	How can I prevent firefox from closing when I ...	<p>In my favorite editor (vim), I regularly us...
2	R Error Invalid type (list) for variable	<p>I am import matlab file and construct a dat...
3	How do I replace special characters in a URL?	<p>This is probably very simple, but I simply ...
4	How to modify whois contact details?	<pre><code>function modify(.......)\n{\n $mco...

Fig 1. Snippet of dataset showing the *Title* and *Body* columns

There was an additional column which contained the topics related to each document, serving as the target variable. The original kaggle competition was structured to predict these topics/labels. However, for our application, we ignore the target variable and conduct our own analysis on simply the title and the question treating the data as unstructured.

3. Data Cleaning

a. Handling missing values

An initial look at the data using the revealed 420,556 non-null values out of the 420,666 total instances indicating 110 missing/unknown rows. Since this was a relatively amount, we were able to simply remove these instances without affecting the overall quality of the dataset.

b. Removing HTML tags

HTML tags are special characters within the source code of websites that encompass different types of data about the features on the page. All such tags generally come in pairs of complimentary tags that indicate the start and end of sequences. The start tag is embedded within the **<>** notation whereas the end notation is **</>** emphasize by a forward slash. For example, the *<p>* and *</p>* notations indicate a **paragraph** while the *<b>* and *<b/>* tags specify **bold text**.

All the HTML tags in the dataset were identified and extracted using the *re* regular expressions library in Python. The *Body* variable contained a total of **6,231,982 tags** out of which **91 were unique** tags. One thing to note would be that there could be more than one type of tag in an instance.

In *Fig 2*, we can see the 25 highest occurring tags in the *Body* column. One interesting thing to note from the plot is that all the tag compliments have the same frequencies as expected. However, there are rare cases where a tag could have a different frequency than its compliment. For example, **<Code>** has 3 occurrences while **</Code>** has 2 occurrences.

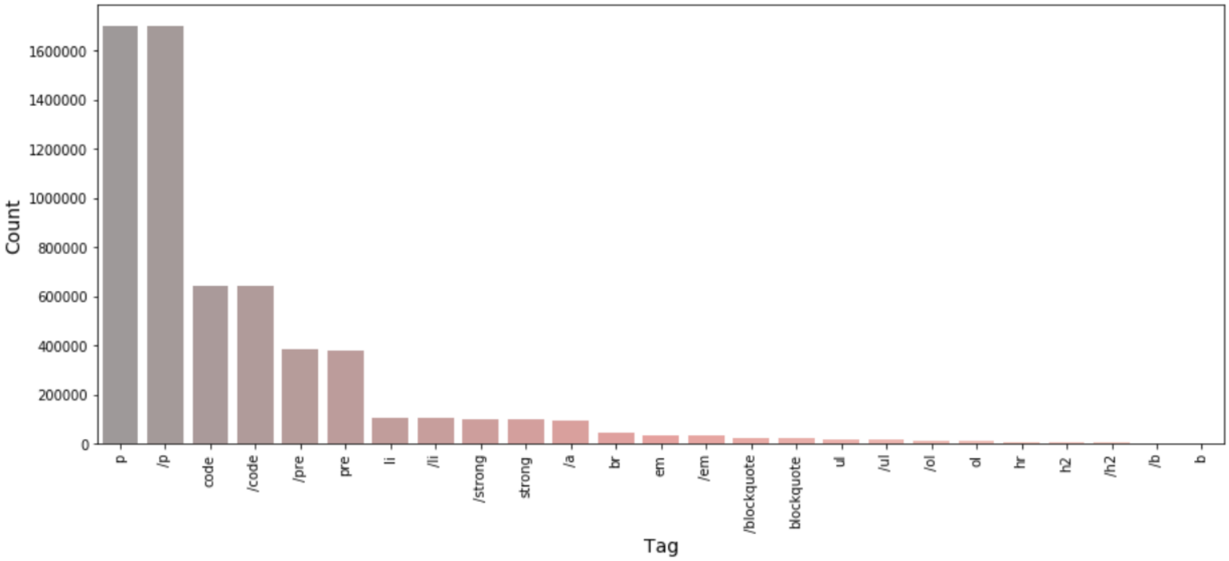


Fig 2. Distribution of top 25 tags in the *Body* column.

To somewhat visualize the occurrence of these tags in each instance, we can plot the distribution for each tag stacked on top of another as shown in *Fig 3*. From this plot, it is evident that the frequency of **<p>** in general is higher. Also, we can see that some tags have unusually high frequencies in certain documents.

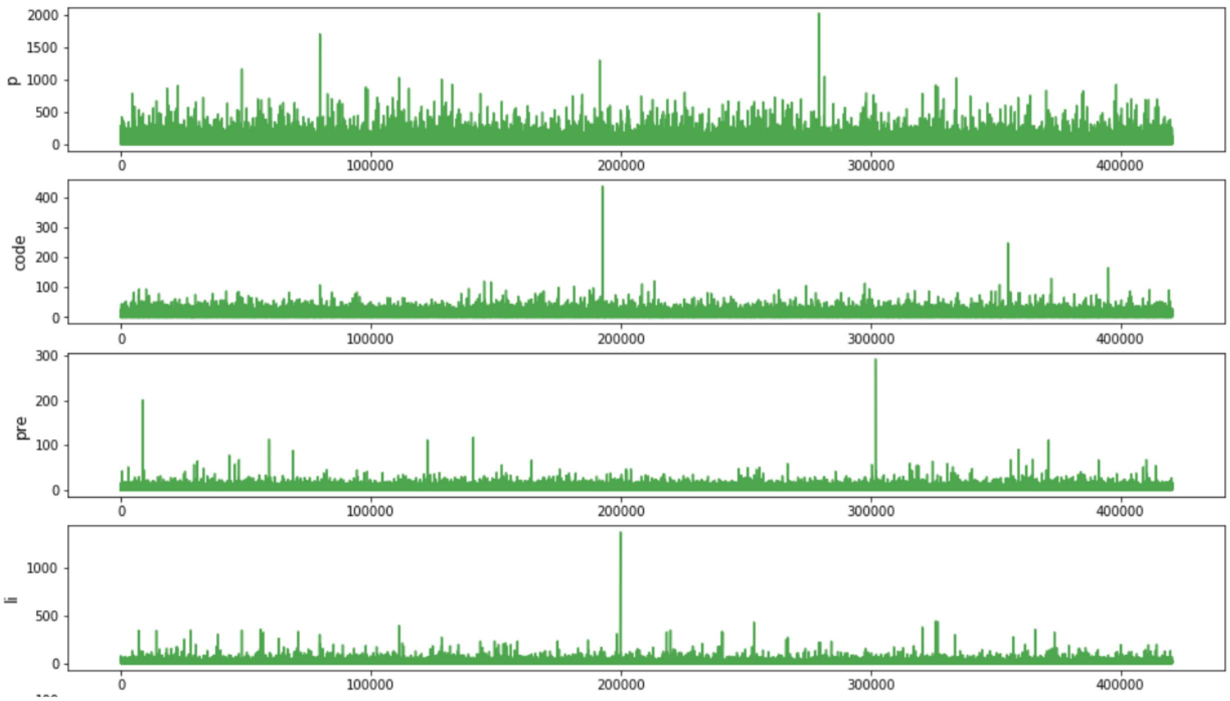


Fig 3. Occurrences of tags across each instance in the *Body* documents.

The distribution of tag frequencies for *Title*, as shown in *Fig 4*, are far more uneven than the *Body* case. There are no complimentary tags and some of these words are not even actual HTML tags such as **script, img,** and **object**. Most of these words could have simply had **<>** and **<>** around them because the user wanted to emphasize them. Considering that, most of these words don't seem to be keywords either. Therefore, we get rid of these words completely.

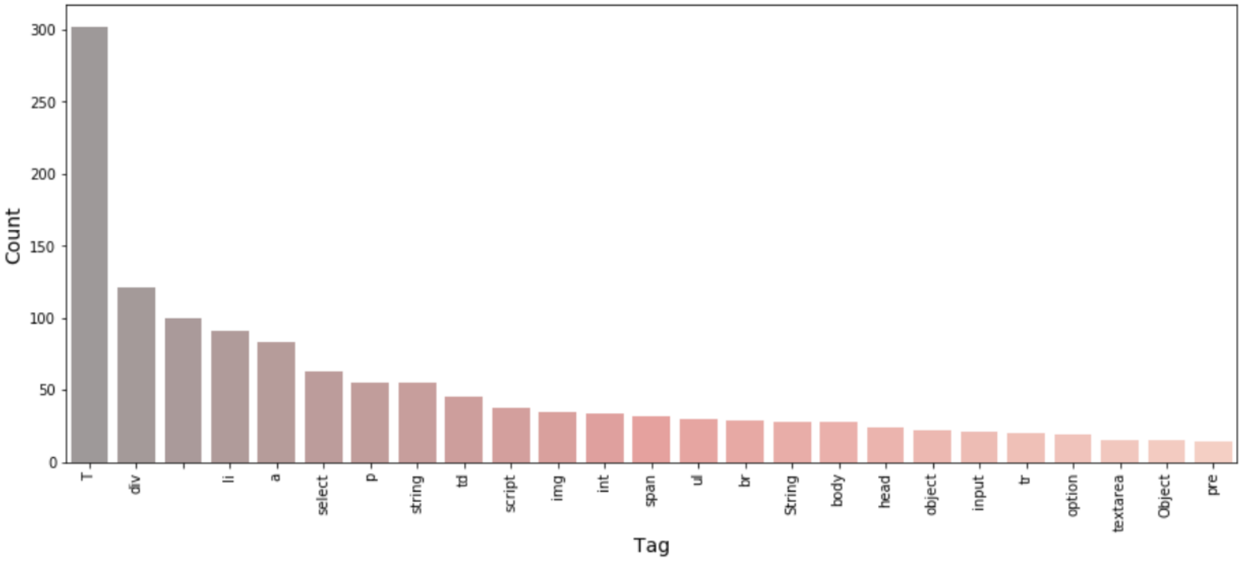


Fig 4. Distribution of top 25 tags in the *Body* column.

c. Removing newline characters

Newline characters (/n) offer no real value for topic modeling, hence they were removed from the dataset.

4. Pre-processing

a. Feature selection

It turns out that tokenizing over 450,000 text documents for both **Title** and **Body** takes up too much computational resources. Further processing can cause the kernel to crash. From some initial analysis done on the frequently occurring words using just 500 instances from the dataset, it was discovered that the title in fact contains more crucial keywords than the body. Therefore, for our analysis, we are simply going to use the *Title* variable.

Fig 5 shows the 50 most frequent words in the title documents while *Fig 6* shows the same for the body. From the title distribution, we can see that words such as **create, jquery, server** and **java** are some of the highest occurring words. These words could serve as crucial keywords. The highest occurring words in the body column are **http, new, like, try** and **work**. Most of the frequent words in this case are more conversational with a lot of verbs as compared to the words in the title column. This makes sense as the body contains more detailed descriptions while the title tends to contain more keywords.

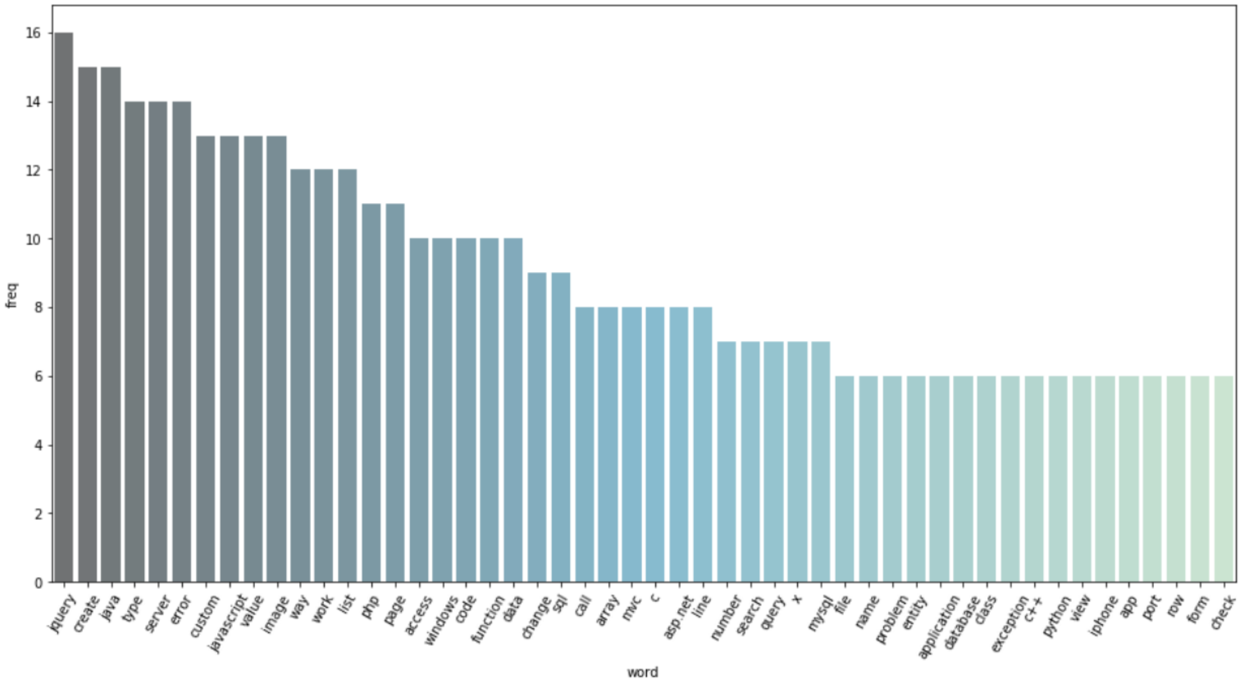


Fig 5. Distribution of top 50 highest occurring words in the title.

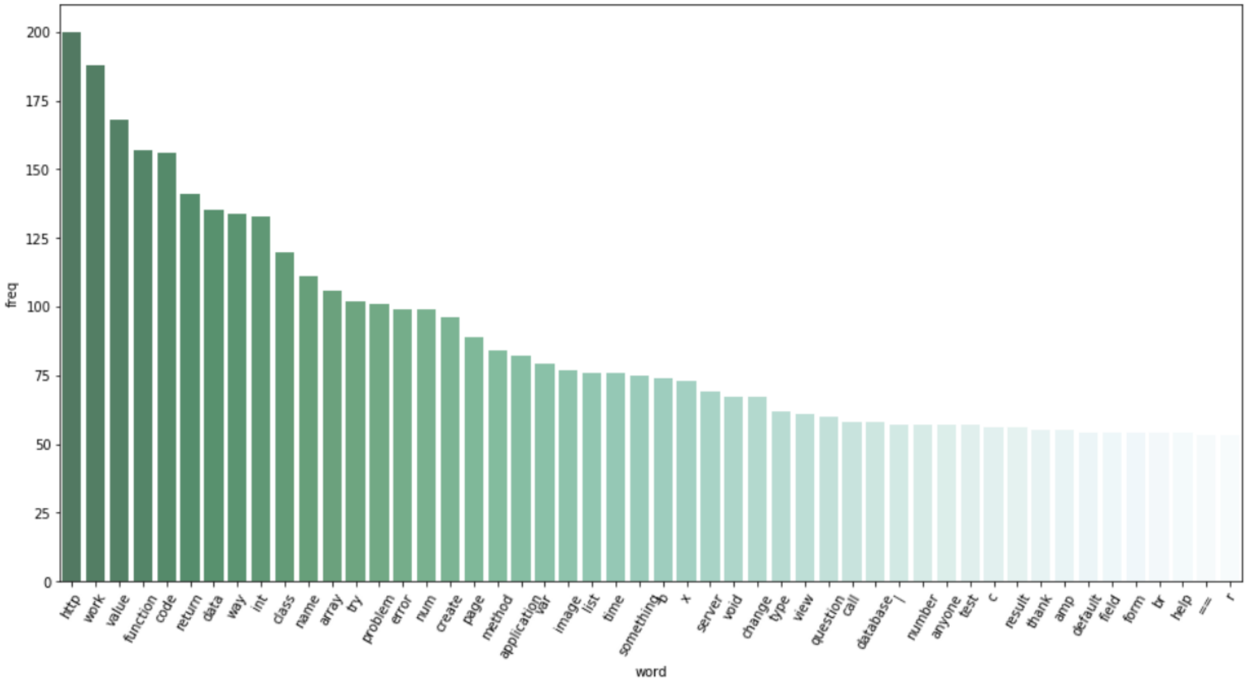
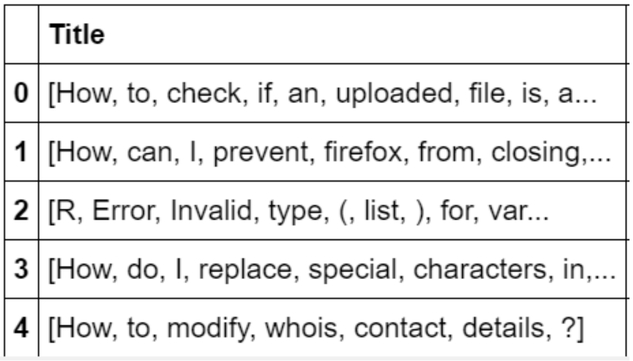


Fig 6. Distribution of top 50 highest occurring words in the title.

b. Word tokenization

Tokenization splits a sentence or text document into tokens which could be words, special characters, punctuations etc. Hence, it's more effective than simply using the ***.****split()*** function. This technique is applied to the entire dataset using the *word_tokenize()* function of Python’s *nltk* library. A snippet of the tokenized documents is shown in *Fig 7*.



	Title
0	[How, to, check, if, an, uploaded, file, is, a...
1	[How, can, I, prevent, firefox, from, closing,...
2	[R, Error, Invalid, type, (, list,), for, var...
3	[How, do, I, replace, special, characters, in,...
4	[How, to, modify, whois, contact, details, ?]

Fig 7. Snippet of tokenized words for each document in the title.

c. Changing words to lowercase

All words in the dataset are changed to lowercase so that the topic model can recognize two words such as ‘Java’ and ‘java’ as the same for practical purposes.

d. Handling stopwords, special characters and symbols

The stopwords were removed in two stages. First we extract a list of all common stopwords using the *stopwords.words(‘English’)* function from the *nltk* library. However, there were still words such as **I, The** and **'d** which also behave as potential stopwords. For these special cases, we iteratively collected as many such strings by checking the highest token frequencies in the dataset.

Tokens such as **(, $, “, +** etc. would show up as some of the highest frequency tokens but offer little in terms of topic modeling. Therefore, such tokens manually added to a list which would then be used to filter out these stopwords from the dataset in a single step. This process was repeated two to three times to get rid of the special characters, symbols and other stopwords that the *nltk* function missed.

e. Lemmatization

Lemmatization was used to reduce different variations of a word to a single word. For example, **runs, ran** and **running** were all changed to the root word **run**. However, we needed to provide a context for the change. In our case, the context or part of speech was set to a **verb** which allowed all the variations of a word to be changed to the simplest verb form.

The verb context was selected since a lot of the questions are about the user wanting to do something or something that they have already tried. This allowed us to group all the variations of a word together for topic modeling. By default, *lemmatizer* uses the noun form. Later, we get rid of verbs in the dataset as nouns are better keywords for topic modeling. When it comes to more meaningful keywords as such as JQuery or Java, lemmatizer will keep them in the default form as the verbs for these words don’t exist. This is another reason why choosing the verb POS for lemmatization is more effective as it would detect all the verbs which would then be filtered out properly.

We have not opted for the *PorterStemmer* method as that tends to reduce words to root words that may not be in the standard English language.

f. Part of Speech (POS) tagging

First we explored all types of POS tags that exist in the dataset for both title and body.

Common POS tags*:*

- *Nouns*: NN
- *Plural nouns*: NNS
- *Proper nouns*: NNP, NNPS
- *Verbs*: VBP, VBN and VBD
- *Adjectives*: JJ, JJR, JJS

- *Adverbs*: RB, RBR (comparative), RBS (superlative)

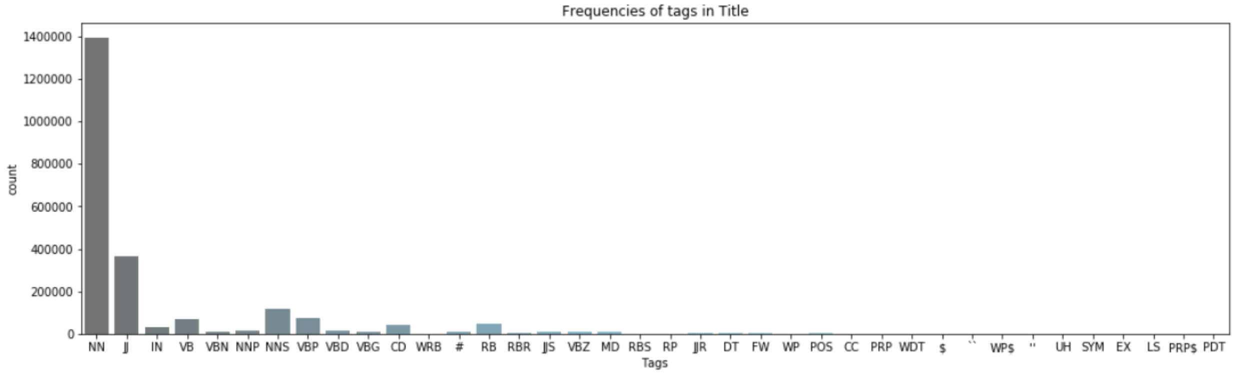
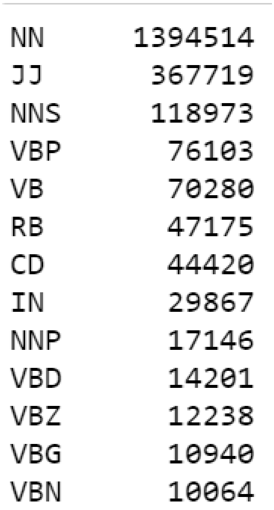


Fig 8. Distribution of POS tags in the title documents.

Fig 8 shows the distribution of the different POS tags in the dataset while *Fig 9* shows the exact frequencies in a table format. From the distributions, we can see that **nouns are by far the most frequent (1,394,514)** with **adjectives coming in at second (367719)**. **Verbs (VB, VBN, VBP, VBD etc) come up to around 19,000 to 20,000** in total while adverbs (RB, RBS, RBR) account for less than 50,000 of the instances.



```
NN     1394514
JJ      367719
NNS     118973
VBP      76103
VB       70280
RB       47175
CD       44420
IN       29867
NNP      17146
VBD      14201
VBZ      12238
VBG      10940
VBN      10064
```

Fig 9. Table showing exact POS tag frequencies for the top 13 tags.

The frequencies of the 40 highest occurring verbs are shown in *Fig 10*. We can see that most of the words do not add much value for topic modeling. However, there are certain words that should really be in the **noun** category such as **windows** and **android**. On the other hand, certain words could fall under both verb and noun depending on the meaning and context. Such words are **object** and **string**. The POS tagger is not going to be able to handle these intricacies. Although, there is a good chance that some of these words also show up as nouns so we don't have to delete them.

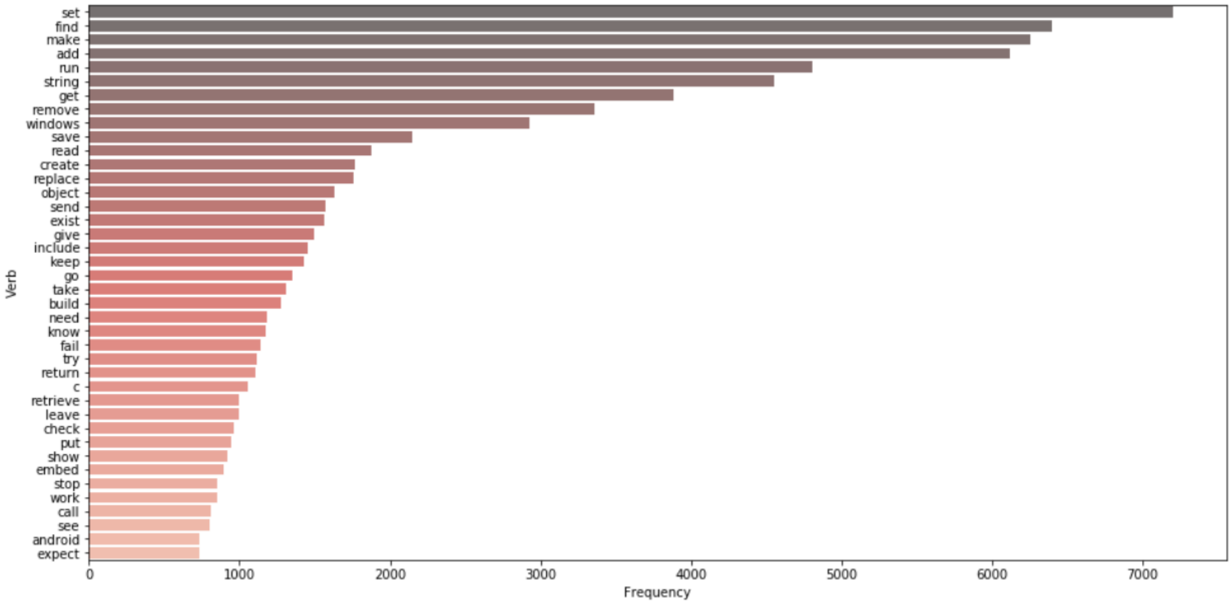


Fig 10. Distribution of 40 highest occurring verbs.

When it comes to adjectives the highest occurring words, shown in *Fig 11*, are accurate to a good extent but there are some keywords such as **android, library, c, url and php** that need to be preserved as they could serve as potential keywords. Overall, the adjectives seem to be more useful for topic modeling than verbs. Hence, we may end up retaining this category.

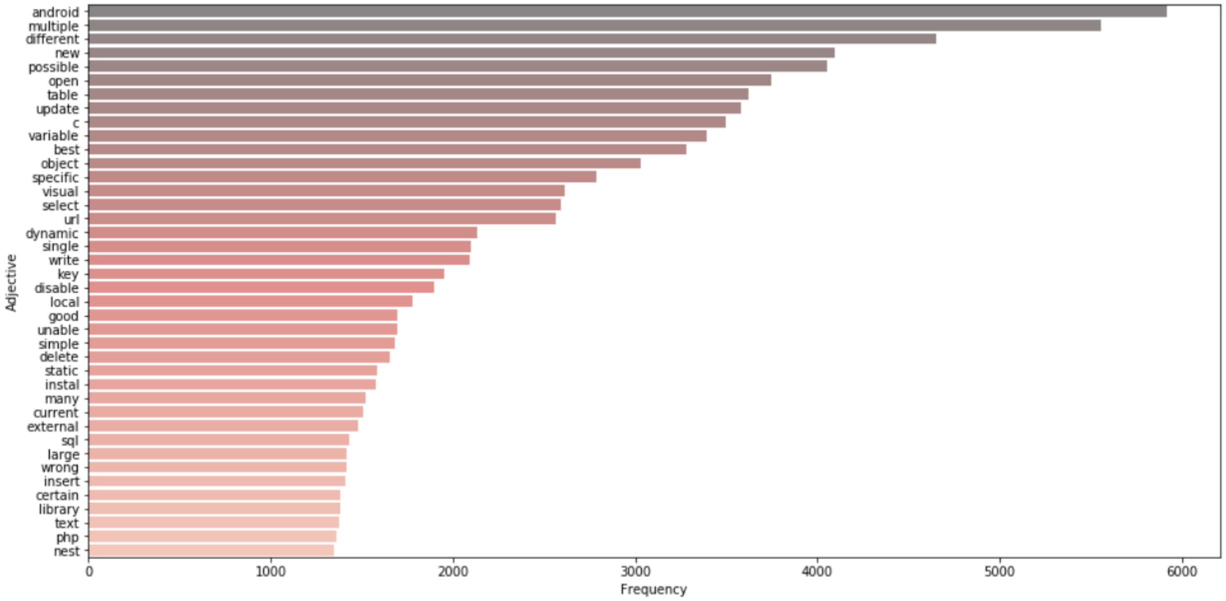


Fig 11. Distribution of 40 highest occurring adjectives.

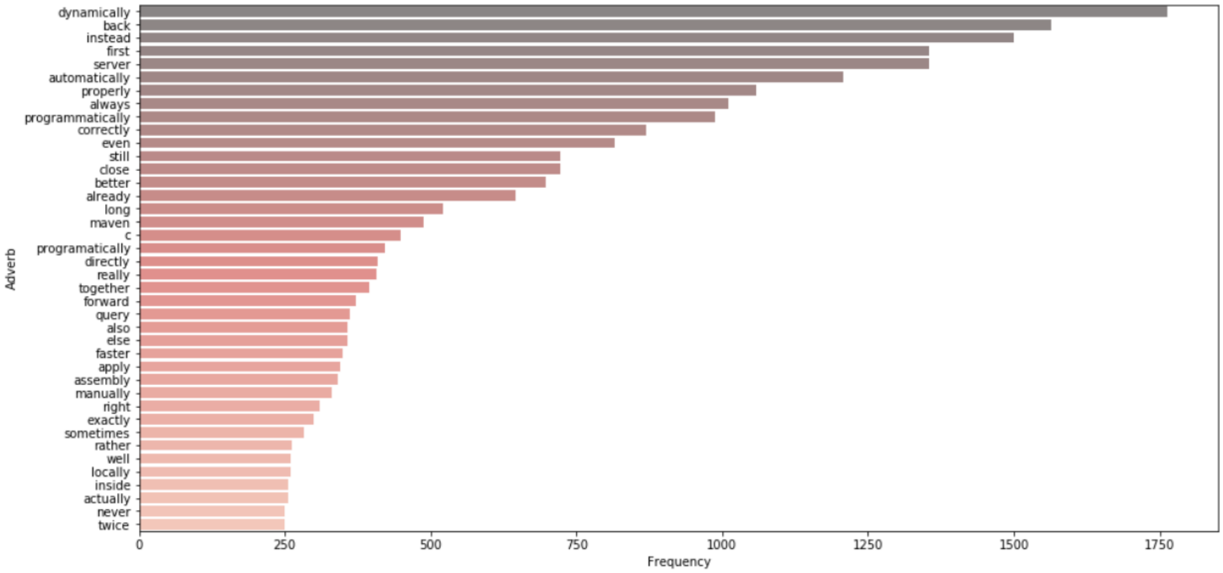


Fig 12. Distribution of 40 highest occurring adverbs.

The distribution of adverbs are shown in *Fig 12*. Since adverbs describe an action (or verb), a lot of these words would be similar to verbs themselves. Hence, similar to verbs, adverbs don't really provide a lot of keywords for our analysis. We can possibly get rid of these.

Fig 13 shows the top 40 nouns in the dataset and we can immediately see that most of these words could be crucial words for topic modeling. Words like **jquery, image, array, java** etc. are technical terms and would provide more meaning to our model than action words.

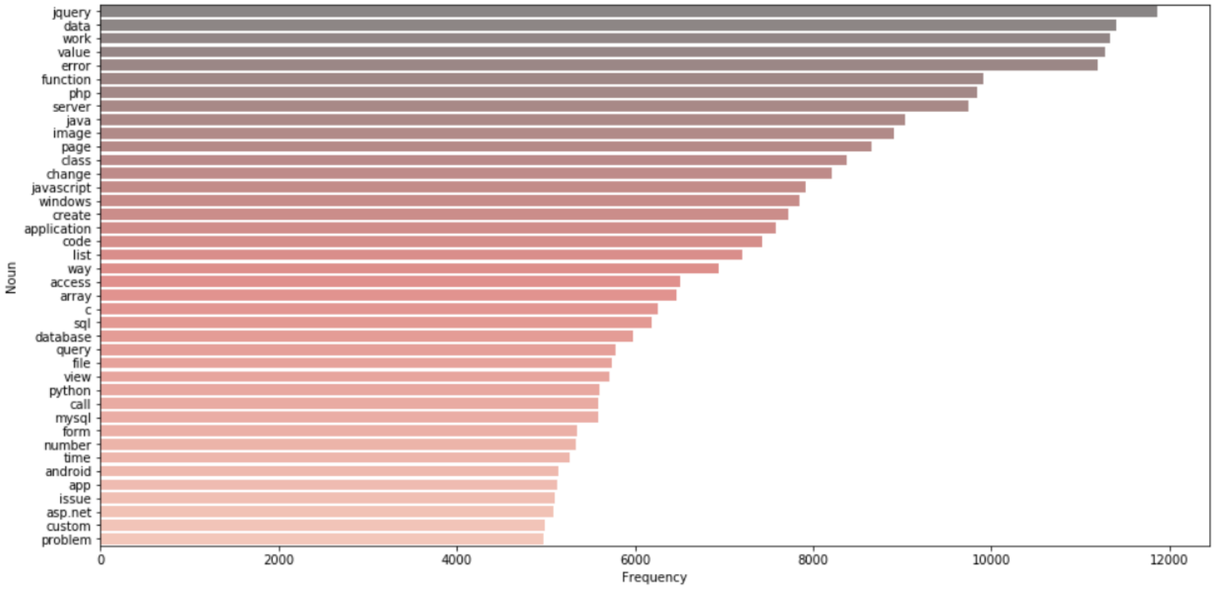


Fig 13. Distribution of 40 highest occurring nouns.

For our analysis, we only kept the nouns and got rid of other tags such as verbs, adverbs and adjectives. We did this by retaining all words with the POS tags *NN, NNS, NNP* and *NNPS*.

5. Exploratory Data Analysis

From the distribution in *Fig 14*, we can see that words such as **create, jquery, server** and **java** are some of the highest occurring words in the title column and they match our earlier visualization that contains just nouns. This makes sense as we only have nouns in our dataset.

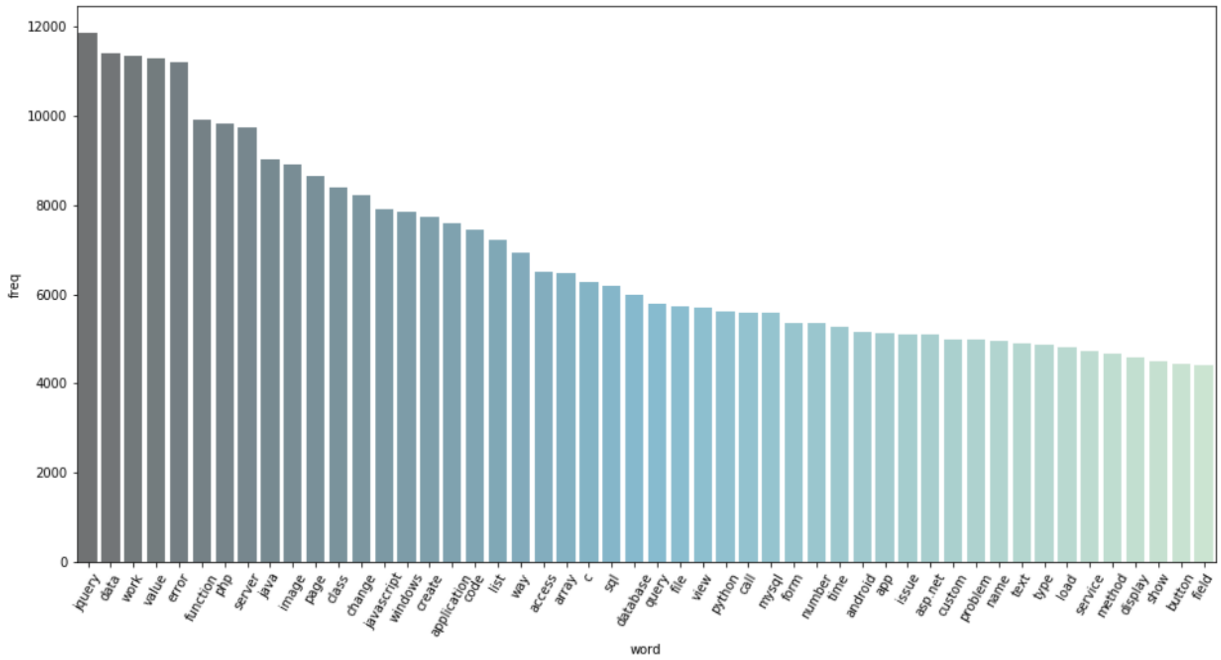


Fig 14. Distribution of highest occurring words in the preprocessed dataset.