EDA_SC1_done

October 13, 2020

1 1.Problem Statement

In this project i have to develop stack overflow search engine based on semantic meaning so when user search any question on search engine then it should give relevant and and semantically most similar results. In order to understand data i need following:

```
1. Title
2. Question body
3. Answer of the question
4. Votes of each answers
```

2 2. Business Objective

```
1. We have to find top n result based in semantic similariy and it should be most relevant to user's
2. Top results should be based on both question and answer, that means if user search for any questi
```

3 2. Business Constraints

```
1. Low latency Requirement.
2. Search engine should display Most relevant result.
```

4 3.Data Collection

1. Google BigQuery dataset includes an archive of stackoverflow contents.it includes posts, votes, tags and badges.This dataset is also available on stack exchange internet archive, here is link : https://archive.org/details/stackexchange , For more understanding, Data is also availabe on kaggle, link: https://www.kaggle.com/stackoverflow/stackoverflow
2. i will obtain data by using GoogleBigQuery for that we have to create project on GCP to get project ID and Google Application credentials.
3. Dataset contains many tables but we only use 'post_question' and 'post_answer' tables.

- post_question contains fields like, question_id, title, body, creation date etc..
- post_answer has fields like, answer_id, answer body, comment count,creation date etc

4. We will join these 2 tables on id's of question and answer and we will get id, title, Question body, question answer and votes for the answer and i will restrict data only for python and sports related question
5. To get data from google cloud dataset we should have google cloud account and have to use google's big query
6. Using googles's big query we will get the data and store it to drive

[27]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# Authenticate google account for acces google public dataset
from google.colab import auth
auth.authenticate_user()
print('Authenticated')
```

```
Authenticated
```

[4]:
```
from google.cloud import bigquery
```

[5]:
```
# project id which we can get from GCP
project_id = 'eminent-facet-283205'
```

[6]:
```
# Google credentials which we can get fromGCP as json file and store it to drive
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]="My First Project-79ec2149eeab.
 ↪json"
```

[7]:
```
client = bigquery.Client()
dataset_reference = client.dataset('stackoverflow',␣
 ↪project='bigquery-public-data')
dataset = client.get_dataset(dataset_reference)
```

[8]:
```
# list of tables availbale in stackoverflow dataset
# reference : https://www.kaggle.com/fluffyhamster/stack-overflow-data
tables = list(client.list_tables(dataset))
for table in tables:
    print(table.table_id)
```

```
badges
comments
post_history
```

```
post_links
posts_answers
posts_moderator_nomination
posts_orphaned_tag_wiki
posts_privilege_wiki
posts_questions
posts_tag_wiki
posts_tag_wiki_excerpt
posts_wiki_placeholder
stackoverflow_posts
tags
users
votes
```

[ ]:
```
# we will only use post question and post question tables for joining.
```

[9]:
```
# The pandas-gbq library is a community led project by the pandas community.
# It covers basic functionality, such as writing a DataFrame to BigQuery and␣
 ↪running a query,
# but as a third-party library it may not handle all BigQuery features or use␣
 ↪cases.
# reference: https://colab.research.google.com/notebooks/bigquery.
 ↪ipynb#scrollTo=oKNxsRvuKtAz

df = pd.io.gbq.read_gbq('''
  SELECT q.id, q.title, q.body, q.tags, a.body as answers,
  a.score FROM `bigquery-public-data.stackoverflow.posts_questions` AS q
  INNER JOIN `bigquery-public-data.stackoverflow.posts_answers` AS a
  ON q.id = a.parent_id
  LIMIT 1000000
  ''', project_id=project_id, dialect='standard')
```

[10]:
```
df.head()
```

[10]:

	id	…	score
0	27880607	…	14
1	23482748	…	34
2	19487576	…	16
3	25157511	…	75
4	1160711	…	-2

[5 rows x 6 columns]

[11]:
```
# we have taken 500000 rows
df.shape
```

[11]:
```
(1000000, 6)
```

4.1 Data Analysis and EDA

[ ]:
```
df = pd.read_csv('/content/drive/My Drive/self_case_study1/data/data1.csv')
df
```

[ ]:
```
              id  … score
0       17256604  …    -2
1       22938679  …    33
2       22938679  …   153
3        7160737  …   124
4        7160737  …    63
…            …  …   …
499995  56198802  …    -1
499996  50935665  …    -1
499997  12458198  …    -1
499998  39927950  …    -1
499999  57554512  …    -1

[500000 rows x 6 columns]
```

[13]:
```
# print dataframe info.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 6 columns):
 #   Column   Non-Null Count    Dtype
---  ------   --------------    -----
 0   id       1000000 non-null  int64
 1   title    1000000 non-null  object
 2   body     1000000 non-null  object
 3   tags     1000000 non-null  object
 4   answers  1000000 non-null  object
 5   score    1000000 non-null  int64
dtypes: int64(2), object(4)
memory usage: 45.8+ MB
```

[ ]:
```
# we have no null values in any of the columns
```

[14]:
```
# checking for null values
df.isna().sum()
```

[14]:
```
id         0
title      0
body       0
tags       0
answers    0
```

```
score      0
dtype: int64
```

[15]:
```
# checking for duplicate values
df.duplicated().any()
```

[15]:
```
False
```

[16]:
```
# checking for duplicate values
print(df.duplicated('id').any())
print(df.duplicated('title').any())
print(df.duplicated('body').any())
print(df.duplicated('tags').any())
print(df.duplicated('answers').any())
```

```
True
True
True
True
True
```

4.2 Data aggregation

1. As we can se here are some duplicate rows in dataframe.
2. so i will concatnate answers based on duplicate id.
3. i will sum up score also based on duplicate id.

[17]:
```
# create corpus
# combine answers and take sum of votes with duplicate id,questions,title and␣
 ↪tags
aggregated = {'answers': lambda x: "\n".join(x), 'score': 'sum'}
# aggregate df
grouped = df.groupby(['id','title', 'body','tags'],as_index=False).
 ↪agg(aggregated)
# make dataframe
final_deduplicate_df = pd.DataFrame(grouped)
```

[18]:
```
# Look at shape after aggregation
final_deduplicate_df.shape
```

[18]:
```
(572554, 6)
```

[19]:
```
print('Total duplicate rows we have removed is = ',(df.
 ↪shape[0]-final_deduplicate_df.shape[0]))
```

```
Total duplicate rows we have removed is =  427446
```

[20]:
```
# maximum score before and after aggregation so we can see effect of aggregation
print('Max score of votes before aggregation',np.max(df.score.values))
print('Max score of votes after aggregation',np.max(final_deduplicate_df.score.
 ↪values))
```

```
Max score of votes before aggregation 11021
Max score of votes after aggregation 14645
```

[21]:
```
print('Columns of Dataset',final_deduplicate_df.columns)
```

```
Columns of Dataset Index(['id', 'title', 'body', 'tags', 'answers', 'score'],
dtype='object')
```

[22]:
```
print('Number of rows in dataframe',final_deduplicate_df.shape[0])
```

```
Number of rows in dataframe 572554
```

5 Analysis on Tags

[23]:
```
final_deduplicate_df['tag_count'] = final_deduplicate_df['tags'].apply(lambda x␣
 ↪: len(x.split('|')))
# apply function will make change in values of series
```

[24]:
```
final_deduplicate_df[['tags','tag_count']]
```

[24]:

	tags	tag_count
0	c#|.net|datetime	3
1	c#|datetime|time|datediff|relative-time-span	5
2	java|php|oop|theory	4
3	c++|oop|class|nested-class	4
4	python|django|views|oop	4
…	…	…
572549	c#|backend|dto	3
572550	c++	1
572551	postgresql|indexing|partial-index	3
572552	python|python-3.x|string|replace	4
572553	python|python-3.6	2

[572554 rows x 2 columns]

[28]:
```
# this is the directory where we will put our data
currentDirectory = "/content/drive/My Drive/self_case_study1/"
# this is for data directory where we will store data.csv
dataDirectory = currentDirectory + "data/"
# store dataframe as csv to working the directory
```

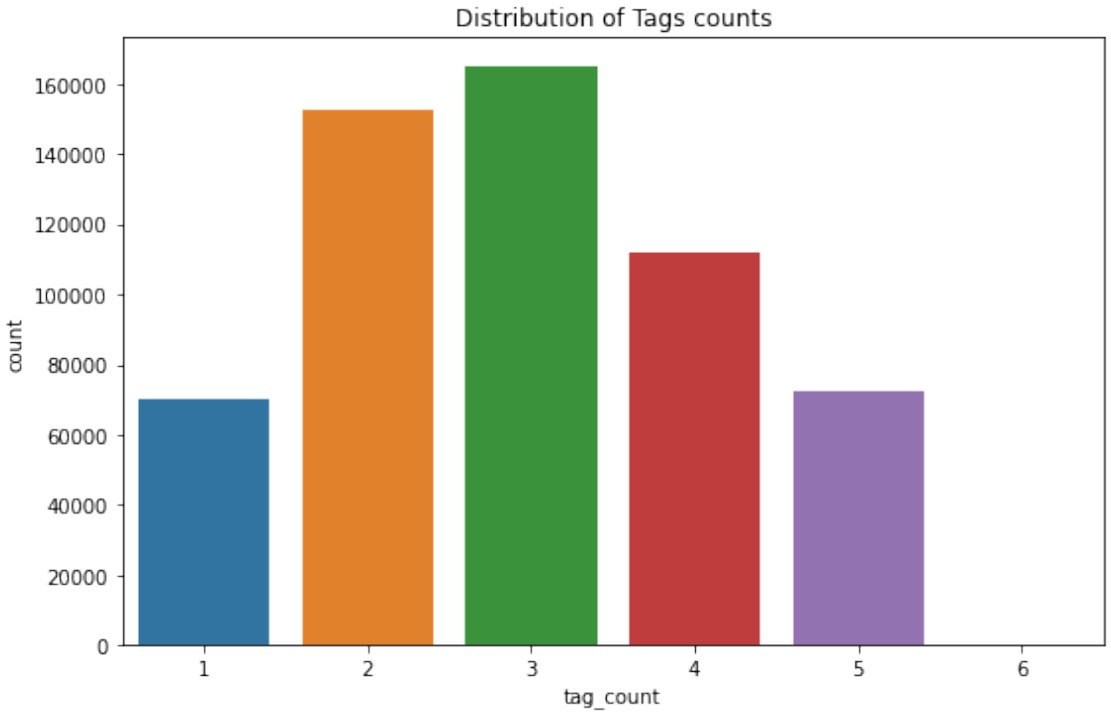
```
final_deduplicate_df.to_csv(dataDirectory + 'all_data.
 ↪csv',encoding='utf-8',index=False)
```

[25]:
```
# maximum tags in question
print('Maximum number of tag in any question is =',np.
 ↪max(final_deduplicate_df['tag_count']))
# minimum tags in question
print('Mimimum number of tag in any question is =',np.
 ↪min(final_deduplicate_df['tag_count']))
```

```
Maximum number of tag in any question is = 6
Mimimum number of tag in any question is = 1
```

[29]:
```
import seaborn as sns
```

[30]:
```
# define figure size
fig = plt.figure(figsize=(8,5))
# countplot usinf seaborn
sns.countplot(x='tag_count',data=final_deduplicate_df)
# for better visulization use tight layout
plt.tight_layout()
# title of the plot
plt.title('Distribution of Tags counts')
# showing plot
plt.show()
```



5.0.1 Observation:

1. By watching above plot we can observe that most of the questions have 3 or 2 tags, and very small amount of questions have only 1 tag in it. 2.Question with 1 and 5 tags are almost same.
2. Maximum number of tag in any question is 5.
3. Minimum number of tag in any question is 1.

Analysis on unique tags and Most frequent tag

[31]:
```
# import CountVectorizer from sklearn
from sklearn.feature_extraction.text import CountVectorizer
# instantiate object to do count of tags
CV = CountVectorizer(tokenizer = lambda x : x.split('|'))
# Bag of words for tags with frequencies of each tag
tag_bag_words = CV.fit_transform(final_deduplicate_df['tags'])
```

[32]:
```
tag_bag_words.shape
```

[32]:
```
(572554, 35571)
```

[33]:
```
print('Total Number of Unique Tags',tag_bag_words.shape[1])
```

```
Total Number of Unique Tags 35571
```

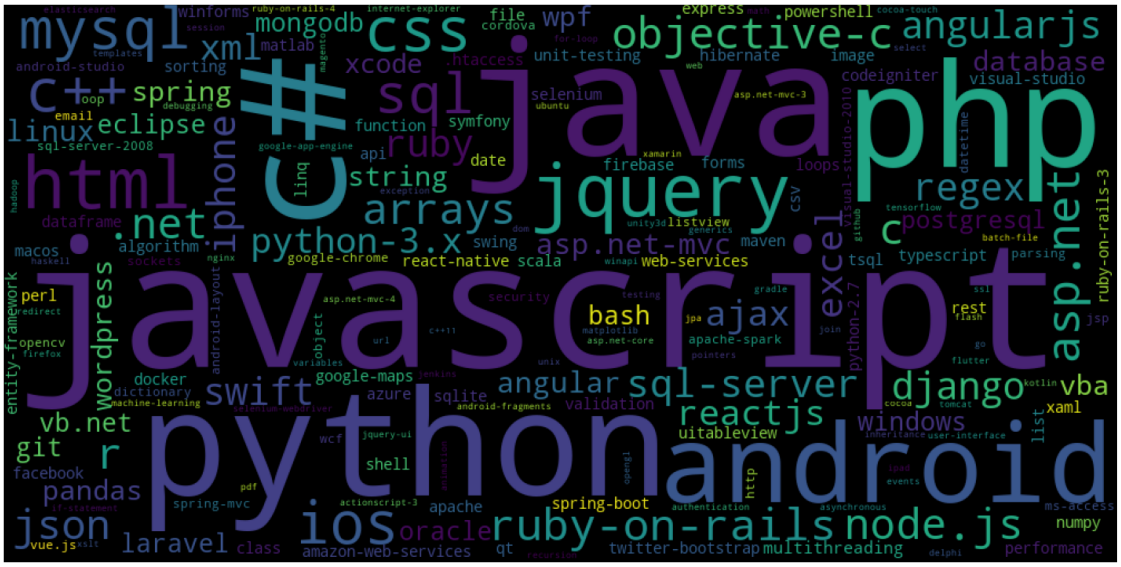
[34]:
```
# taking sum of each tags
# below line will sum the each column of sparce matrix and A1 converts matrix␣
 ↪into array
tag_column_sum = tag_bag_words.sum(axis=0).A1
# make dictionary of each unique tag frequency count
tag_freq_count = dict(zip(CV.get_feature_names(),tag_column_sum))
# sort above dict in Descending order of tag count
tag_freq_count_sorted = dict(sorted(tag_freq_count.items(), key = lambda x:␣
 ↪x[1], reverse = True))
```

[35]:
```
# top 10 most frequent tags
list(tag_freq_count_sorted.items())[:10]
```

[35]:
```
[('javascript', 57610),
 ('java', 47535),
 ('python', 40462),
 ('c#', 40064),
 ('php', 39090),
 ('android', 36325),
 ('jquery', 30018),
```

```
('html', 29473),
('css', 20017),
('c++', 18425)]
```

[38]:
```
# generate wordcloud from frequencies of tag counts
# Reference : https://stackoverflow.com/questions/16645799/
 ↪how-to-create-a-word-cloud-from-a-corpus-in-python
from wordcloud import WordCloud
Wcloud = WordCloud(width=1000,height=500)
Wcloud.generate_from_frequencies(tag_freq_count)
plt.figure(figsize=(20,20))
plt.imshow(Wcloud)
plt.axis('off')
plt.tight_layout()
plt.show()
```



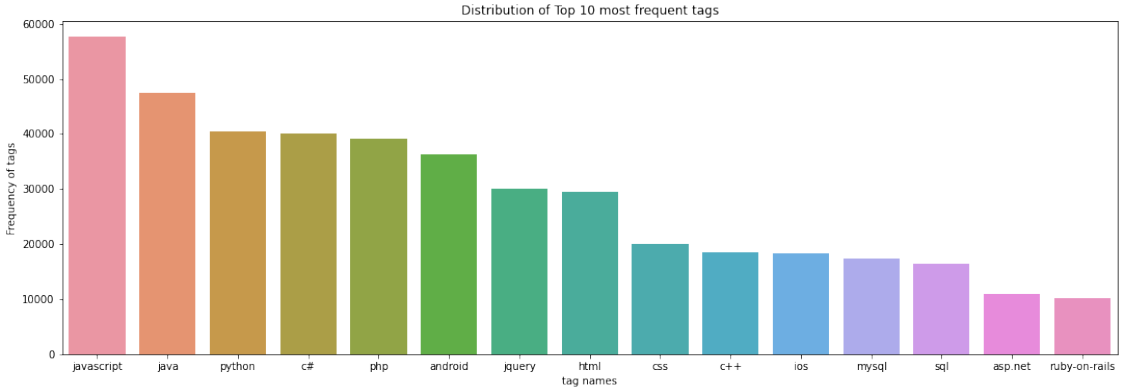
Observation

1. From above word cloud we can observ that size of Python word is very big that means **javascript** is most frequent word.
2. Java is second most frequent word and python and c# is 3rd and 4th most frequent words.

Bar Plot of Top 10 tags vs frquency

[43]:
```
tags = list(tag_freq_count_sorted.keys())[:15]
freq = list(tag_freq_count_sorted.values())[:15]
```

[46]:
```
# define figure size
fig = plt.figure(figsize=(15,5))
# Barplot using seaborn
sns.barplot(x=tags,y=freq)
# for better visulization use tight layout
plt.tight_layout()
# title of the plot
plt.title('Distribution of Top 10 most frequent tags')
plt.xlabel('tag names')
plt.ylabel('Frequency of tags')
# showing plot
plt.show()
```



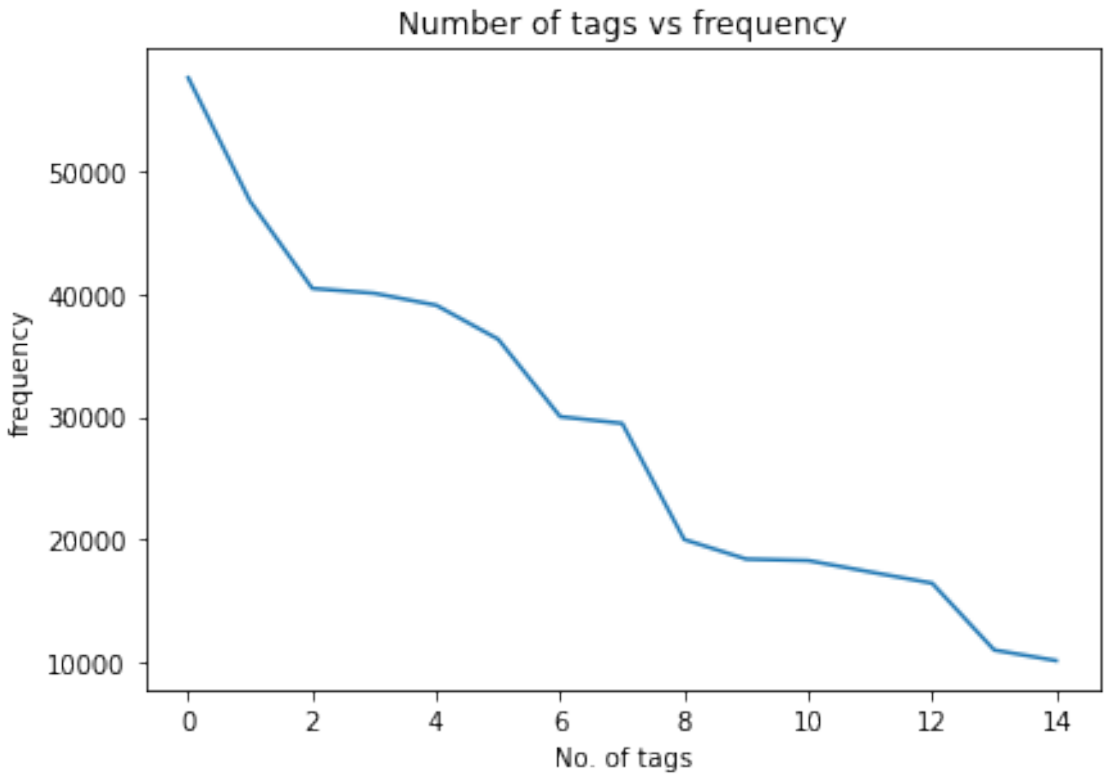
5.0.2 Observation

- Most frequent tags are javascript and java and python

Plot for frequency distribution

[47]:
```
plt.plot(freq)
plt.tight_layout()
plt.title('Number of tags vs frequency')
plt.xlabel('No. of tags')
plt.ylabel('frequency')
```

[47]:
```
Text(11.625, 0.5, 'frequency')
```

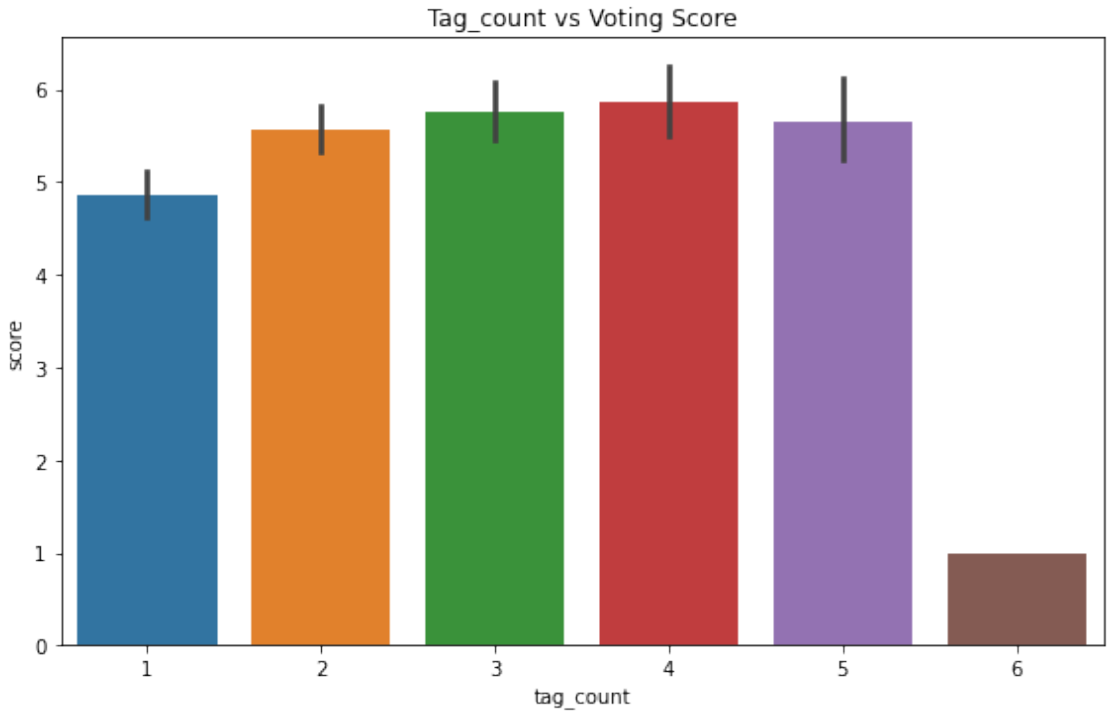


Obseravtion

- From above plot we can observe that there are very few tags whose frequency is very high, Actually we have seen that in bar plot that only javascript tag is most frequent.
- Except javascript tag all other tags frequenct is below 50000.
- Two tags(python and c#) have frequency is around 40,000.

5.0.3 countplot for Voting Score

[48]:
```
# define figure size
fig = plt.figure(figsize=(8,5))
# Barplot using seaborn
sns.barplot(x=final_deduplicate_df['tag_count'],y=final_deduplicate_df['score'])
# for better visulization use tight layout
plt.tight_layout()
# title of the plot
plt.title('Tag_count vs Voting Score')
# showing plot
plt.show()
```



Obseravtion

- From above plot we observes that question with 6 tag count has very low voting score.
- Questions with tag count 4 has highest rating amontgst all
- Questions with 2,3 and 5 tag counts have almost same voting score.