In [1]:
```
pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.1.2.tar.gz (212.4 MB)
     |████████████████████████████████| 212.4 MB 50 kB/s s eta 0:00:01     |███
     | 24.4 MB 4.4 MB/s eta 0:00:43
Collecting py4j==0.10.9
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 51.1 MB/s eta 0:00:01
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.1.2-py2.py3-none-any.whl size=212880768 sha256=9a66496492c4545fdbd0aba7ff09b1f3ee9c18003652c3a8231e4f4c19de865b
  Stored in directory: /root/.cache/pip/wheels/a5/0a/c1/9561f6fecb759579a7d863dcd846daaa95f598744e71b02c77
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9 pyspark-3.1.2
WARNING: Running pip as root will break packages and permissions. You should install packages reliably by using venv: https://pip.pypa.io/warnings/venv
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
from pyspark.sql import functions as f
from pyspark.sql import DataFrameNaFunctions as DFna
from pyspark.sql.functions import udf, col, when
import matplotlib.pyplot as plt
import pyspark as ps
import os, sys, requests, json
from pyspark.sql.functions import col,size,regexp_replace,lit
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS

from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml import Pipeline
from pyspark.sql import Row
import numpy as np
import math
from pyspark.sql.functions import regexp_replace
from pyspark.sql import *



spark = ( SparkSession.builder.appName("ModelTraining") .config("spark.executor.memory","16g") .getOrCreate() )

sc = spark.sparkContext
```

In [3]:
```
import numpy as np
import pandas as pd
from pyspark.sql import functions as f
from operator import add
from pyspark.ml.feature import RegexTokenizer, CountVectorizer, Tokenizer
from pyspark.ml.feature import StopWordsRemover, VectorAssembler
from pyspark.ml.feature import Word2Vec, Word2VecModel
from pyspark.ml.feature import IDF

from pyspark.ml import Pipeline, PipelineModel


from pyspark.sql.functions import *

from pyspark.sql.types import *

import folium
import html
```

In [4]:
```
#Data cleaning question
def datacleaning_ques(test):
    col_name='questions_body'
    user_regex=r"(@\w{1,15})"
    clean_test=test.withColumn( 'user_mentioned',f.array_remove(
    f.array(
    f.regexp_extract(f.col(col_name),user_regex,1),
    f.regexp_extract(
    f.col(col_name), "".join([f"{user_regex}.*?" for i in range(0,2)]),2),
    f.regexp_extract(
    f.col(col_name), "".join([f"{user_regex}.*?" for i in range(0,3)]),3),
    f.regexp_extract(
    f.col(col_name), "".join([f"{user_regex}.*?" for i in range(0,4)]),4),
    f.regexp_extract(
    f.col(col_name), "".join([f"{user_regex}.*?" for i in range(0,5)]),5),
    f.regexp_extract(
    f.col(col_name), "".join([f"{user_regex}.*?" for i in range(0,6)]),6),), "",).alias('user_mentioned'))
    clean_test=clean_test.withColumn( 'original_text',f.col(col_name)).withColumn(col_name,f.regexp_replace(f.col(col_name), user_regex,"").alias(col_name))
    # remove hastag
    hashtag_user_regex="#(\w{1,})"
    clean_test=clean_test.withColumn( 'hashtags',f.array_remove(
    f.array(
    f.regexp_extract(f.col(col_name),hashtag_user_regex,1),
    f.regexp_extract(
    f.col(col_name), "".join([f"{hashtag_user_regex}.*?" for i in range(0,2)]),2),
    f.regexp_extract(
    f.col(col_name), "".join([f"{hashtag_user_regex}.*?" for i in range(0,3)]),3),
    f.regexp_extract(
    f.col(col_name), "".join([f"{hashtag_user_regex}.*?" for i in range(0,4)]),4),
    f.regexp_extract(
    f.col(col_name), "".join([f"{hashtag_user_regex}.*?" for i in range(0,5)]),5),
    f.regexp_extract(
    f.col(col_name), "".join([f"{hashtag_user_regex}.*?" for i in range(0,6)]),6),), "",).alias('hashtags'))
    clean_test=clean_test.withColumn(col_name,f.regexp_replace(f.col(col_name), hashtag_user_regex,"$1").alias(col_name))
    url_regex=r"((https?|ftp|file):\/{2,3})+([-\w+&@#/%=~|$?!:,.]*)|(www.)+([-\w+&@#/%=~|$?!:,.]*)"
    clean_test=clean_test.withColumn( col_name,f.regexp_replace(f.col(col_name), url_regex,""))

    email_regex=r"[\w.-]+@[\w.-]+\.[a-zA-Z]{1,}"
    clean_test=clean_test.withColumn(col_name,f.regexp_replace(f.col(col_name), email_regex,""))
    #clean_test=clean_test.withColumn('text', html_unescape("text"))

    test_cleaned=(clean_test.withColumn( col_name,f.regexp_replace(f.col(col_name),"[^a-zA-Z]"," ")).withColumn(col_name,f.regexp_replace(f.col(col_name)," +"," ")).withColumn( col_name,f.trim(f.col(col_name))).filter(col(col_name) !=''))
    test_data=test_cleaned.select('professionals_id',col_name).coalesce(2).cache()
    return test_data
```

In [5]:
```
pro=spark.read.csv('../input/data-science-for-good-careervillage/professionals.csv', header=True,quote='"',sep=",",multiLine=True)
email=spark.read.csv('../input/data-science-for-good-careervillage/emails.csv', header=True,quote='"',sep=",",multiLine=True)
ques=spark.read.csv('../input/data-science-for-good-careervillage/questions.csv', header=True,quote='"',sep=",",multiLine=True)
match=spark.read.csv('../input/data-science-for-good-careervillage/matches.csv', header=True,quote='"',sep=",",multiLine=True)
ans=spark.read.csv('../input/data-science-for-good-careervillage/answers.csv', header=True,quote='"',sep=",",multiLine=True)
ans_score=spark.read.csv('../input/data-science-for-good-careervillage/answer_scores.csv', header=True,quote='"',sep=",",multiLine=True)
```

In [6]:
```
pro_ans=pro.join(ans,pro.professionals_id==ans.answers_author_id).select('professionals_id','answers_question_id','answers_id')
```

In [7]:
```
pro_ans=pro_ans.join(ques,pro_ans.answers_question_id==ques.questions_id).select('professionals_id','questions_body','questions_title')
```

In [8]:
```
pro_ans=pro_ans.withColumn('combined_ques',concat(pro_ans.questions_title,lit(' '),pro_ans.questions_body))
```

In [9]:
```
pro_ques_final=pro_ans.groupBy('professionals_id').agg(f.collect_list('questions_title').alias('questions_body'))
```

In [10]:
```
from pyspark.sql.functions import udf, col
join_udf = udf(lambda x: ",".join(x))
pro_ques_final=pro_ques_final.withColumn("questions_body", join_udf(col("questions_body")))
```

In [11]:
```
pro_ques_final=datacleaning_ques(pro_ques_final)
```

In [12]:
```
pro_ques_final=pro_ques_final.na.drop(subset=["questions_body"])
```

In [13]:
```
# Build the pipeline
tokenizer = RegexTokenizer(gaps = False, pattern = '\w+',inputCol = 'questions_body', outputCol = 'token')
stopWordsRemover = StopWordsRemover(inputCol = 'token', outputCol = 'nostopwrd')
word2Vec = Word2Vec(vectorSize = 100, minCount = 5, inputCol = 'nostopwrd', outputCol = 'word_vec', seed=123)
pipeline = Pipeline(stages=[tokenizer, stopWordsRemover, word2Vec])

# fit the model
pipeline_mdl = pipeline.fit(pro_ques_final)
```

In [14]:
```
# transform the question data

ques_pipeline_df = pipeline_mdl.transform(pro_ques_final)
```

In [15]:
```
def CosineSim(vec1, vec2):
    return np.dot(vec1, vec2) / np.sqrt(np.dot(vec1, vec1)) / np.sqrt(np.dot(vec2, vec2))
```

In [16]:
```
all_ques_vecs = ques_pipeline_df.select('professionals_id', 'word_vec').rdd.map(lambda x: (x[0], x[1])).collect()
```

In [17]:
```
def getQuestionDetails(in_ques):

    a = in_ques.alias("a")
    b = pro.alias("b")

    return a.join(b, col("a.professionals_id") == col("b.professionals_id"), 'inner') \
            .select([col('a.score')]+[col('b.'+xx) for xx in b.columns] )
```

In [18]:
```
def getKeyWordsRecoms(key_words, sim_bus_limit):

    input_words_df = sc.parallelize([(0, key_words)]).toDF(['professionals_id', 'questions_body'])

    # transform the the key words to vectors
    input_words_df = pipeline_mdl.transform(input_words_df)

    # choose word2vec vectors
    input_key_words_vec = input_words_df.select('word_vec').collect()[0][0]

    # get similarity
    sim_bus_byword_rdd = sc.parallelize((i[0], float(CosineSim(input_key_words_vec, i[1]))) for i in all_ques_vecs)

    sim_bus_byword_df = spark.createDataFrame(sim_bus_byword_rdd) \
         .withColumnRenamed('_1', 'professionals_id') \
         .withColumnRenamed('_2', 'score') \
         .orderBy("score", ascending = False)

    sim_bus_byword_df=sim_bus_byword_df.na.drop(subset=["score"])

    # return top 10 similar
    a = sim_bus_byword_df.limit(sim_bus_limit)

    return getQuestionDetails(a).sort('score', ascending=False)
```

In [19]:
```
def acc_1ques(id_ques):
    key_words = str(np.array(ques.filter(ques.questions_id==id_ques).select('questions_title').collect())[0][0])
    keywords_recom_df = getKeyWordsRecoms(key_words, 20)
    list_truth=np.array(ans.filter(ans.answers_question_id==id_ques).select('answers_author_id').collect())
    list_recom=np.array(keywords_recom_df.select('professionals_id').collect())
    count=0
    sum_count=ans.filter(ans.answers_question_id==id_ques).count()
    for i in list_recom:
        for j in list_truth:
            if i==j:
                count=count+1
                break
    count=float(count/sum_count)
    return count
```

In [20]:
```
test=['01352c4d67fe435ca59e745ff2520d2a',

'03eee1ca07174470b160717027ab46d6',

'04a979f4e7fd49b9a07b6fae7a5727ee',

'062f49f153de4b8793e4e669ec5b5331',

'083965c88d894a9f9e4e71e521641338',

'09e3bdc69a6149aa8656bbc18162ac37',

'0d7fab391dc145a384da4af0a078b77f',

'0db6ed5d24df42f18d19958ccb32cd6e',

'1a039cb9f3064f76b386f84f303edc43',

'1a444e5e5824446eaf37f31effd72ce0']
sum=0
for t in test:
    sum=sum+acc_1ques(t)
score=float(sum/len(test))
score
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:2: RuntimeWarning: invalid value encountered in double_scalars
```

Out[20]: 0.2439203280379751

In [21]:
```
id_ques='8eb6ba7af57846acbfec5633e537192a'
key_words = str(np.array(ques.filter(ques.questions_id==id_ques).select('questions_body').collect())[0][0])
print(key_words)
keywords_recom_df = getKeyWordsRecoms(key_words, 10)
keywords_recom_df.toPandas()
```

```
Interested to be a  management consultant in the future. #consulting #management-consulting #strategic-consulting #career
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:2: RuntimeWarning: invalid value encountered in double_scalars
```

Out[21]:

	score	professionals_id	professionals_location	professionals_industry	professionals_headline	professionals_date_
0	0.878787	48d053b9dcb54579b243a57fbc81a3ad	Ithaca, New York	Financial Services	Associate Director at Cornell University	2018-03-01 04 UTC
1	0.765276	d7d98b518565418f9af9dbabb2c18cfc	Hong Kong	Investment Banking	Investment Banking Analyst at J.P. Morgan	2015-08-10 23 UTC
2	0.759056	9504260763e24a5191e687c2382f676c	San Francisco Bay Area	Management Consulting	Founder of The NYC & SF Restaurant Tech Groups...	2017-08-22 00 UTC
3	0.756288	abd17fd03c9a4441a5610b0d61067458	Chicago, Illinois	Online Publishing	Hotel Manager	2017-04-29 10 UTC
4	0.747003	c3f884b90ba242e3979764cf7cc4e656	Greater Detroit Area	Financial Services	Director of Sales, North America, ExxonMobil F...	2018-12-24 14 UTC
5	0.746175	df81d59e5a864eab830797e443b82010	Tampa/St. Petersburg, Florida Area	Management Consulting	Director of Enterprise Project Management Offi...	2018-10-15 20 UTC
6	0.745958	6beed7da13a34f3699bf8ee0a468cb50	Oak Lawn, Illinois	Legislative Office	Legislative Analyst | Researcher | Writer | Hi...	2017-05-01 19 UTC
7	0.745154	ccb9df2adf02464ebb8dc71febc28fd6	Nashville, Tennessee	Healthcare	Financial Assistance Coordinator at Fresenius ...	2014-09-23 13 UTC
8	0.745137	6758f047be9b432c98d3e4f13497ec1e	Philadelphia, Pennsylvania	Professional Training	Coach | Establisher | Builder	2016-03-04 15 UTC
9	0.742576	89ea8fc166844fa0aafb8df2144b8f74	Mullingar	Internet	Sales and Account Management at Google	2014-05-29 15 UTC

In [ ]: