Quora Question Pairs

1. Business Problem

1.1 Description

Quora is a place to gain and share knowledge—about anything. It’s a platform to ask questions and connect with people who contribute unique insights and quality answers. This empowers people to learn from each other and to better understand the world.

Over 100 million people visit Quora every month, so it's no surprise that many people ask similarly worded questions. Multiple questions with the same intent can cause seekers to spend more time finding the best answer to their question, and make writers feel they need to answer multiple versions of the same question. Quora values canonical questions because they provide a better experience to active seekers and writers, and offer more value to both of these groups in the long term.

> Credits: Kaggle

Problem Statement

- Identify which questions asked on Quora are duplicates of questions that have already been asked.
- This could be useful to instantly provide answers to questions that have already been answered.
- We are tasked with predicting whether a pair of questions are duplicates or not.

1.2 Sources/Useful Links

- Source : https://www.kaggle.com/c/quora-question-pairs (https://www.kaggle.com/c/quora-question-pairs)

Useful Links

- Discussions : https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments (https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments)
- Kaggle Winning Solution and other approaches: https://www.dropbox.com/sh/93968nfnrzh8bp5/AACZdtsApc1QSTQc7X0H3QZ5a?dl=0 (https://www.dropbox.com/sh/93968nfnrzh8bp5/AACZdtsApc1QSTQc7X0H3QZ5a?dl=0)
- Blog 1 : https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning (https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning)
- Blog 2 : https://towardsdatascience.com/identifying-duplicate-questions-on-quora-top-12-on-kaggle-4c1cf93f1c30 (https://towardsdatascience.com/identifying-duplicate-questions-on-quora-top-12-on-kaggle-4c1cf93f1c30)

1.3 Real world/Business Objectives and Constraints

1. The cost of a mis-classification can be very high.
2. You would want a probability of a pair of questions to be duplicates so that you can choose any threshold of choice.
3. No strict latency concerns.
4. Interpretability is partially important.

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

- Data will be in a file Train.csv
- Train.csv contains 5 columns : qid1, qid2, question1, question2, is_duplicate
- Size of Train.csv - 60MB
- Number of rows in Train.csv = 404,290

2.1.2 Example Data point

```
"id","qid1","qid2","question1","question2","is_duplicate"
"0","1","2","What is the step by step guide to invest in share market in india?","W
hat is the step by step guide to invest in share market?","0"
"1","3","4","What is the story of Kohinoor (Koh-i-Noor) Diamond?","What would happe
n if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?","0"
"7","15","16","How can I be a good geologist?","What should I do to be a great geol
ogist?","1"
"11","23","24","How do I read and find my YouTube comments?","How can I see all my
Youtube comments?","1"
```

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

It is a binary classification problem, for a given pair of questions we need to predict if they are duplicate or not.

2.2.2 Performance Metric

Source: https://www.kaggle.com/c/quora-question-pairs#evaluation (https://www.kaggle.com/c/quora-question-pairs#evaluation)

Metric(s):

- log-loss : https://www.kaggle.com/wiki/LogarithmicLoss (https://www.kaggle.com/wiki/LogarithmicLoss)
- Binary Confusion Matrix

2.3 Train and Test Construction

We build train and test by randomly splitting in the ratio of 70:30 or 80:20 whatever we choose as we have sufficient points to work with.

3. Exploratory Data Analysis

In [79]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
%matplotlib inline
import os
import warnings
warnings.filterwarnings("ignore")
from subprocess import check_output
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import gc
import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
from wordcloud import WordCloud, STOPWORDS
from PIL import Image
```

In [44]:
```
df = pd.read_csv("train.csv")
df.head()
```

Out[44]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

In [45]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 6 columns):
id              404290 non-null int64
qid1            404290 non-null int64
qid2            404290 non-null int64
question1       404290 non-null object
question2       404288 non-null object
is_duplicate    404290 non-null int64
dtypes: int64(4), object(2)
memory usage: 18.5+ MB
```

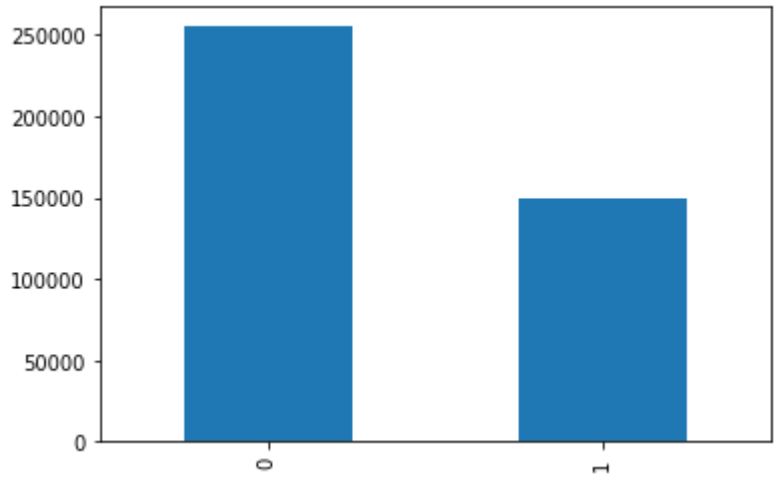
We are given a minimal number of data fields here, consisting of:

- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [46]:
```
df['is_duplicate'].value_counts().plot(kind = 'bar')
plt.show()
```



In [47]:
```
print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(1
00 - round(df['is_duplicate'].mean()*100, 2)))
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(rou
nd(df['is_duplicate'].mean()*100, 2)))
```

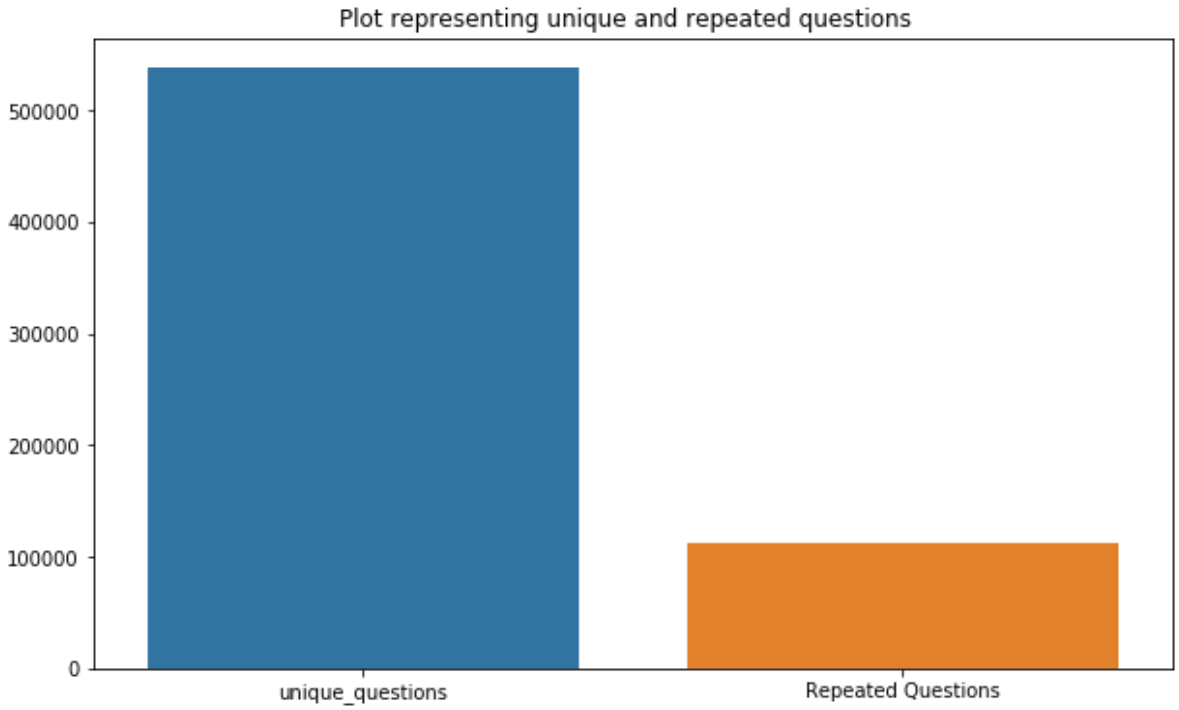
```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

~> Question pairs are Similar (is_duplicate = 1):
   36.92%
```

3.2.2 Number of unique questions

In [48]:
```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Question Pair are: {}\n'.format(unique_qs))

print ('Number of unique question Pair that appear more than one time: {} ({}
%)\n'.format(qs_morethan_onetime,qs_morethan_onetime/unique_qs*100))

print ('Max number of times a single question is repeated: {}\n'.format(max(qi
ds.value_counts())))

q_vals=qids.value_counts()

q_vals=q_vals.values
```

```
Total number of  Unique Question Pair are: 537933

Number of unique question Pair that appear more than one time: 111780 (20.779
53945937505%)

Max number of times a single question is repeated: 157
```

In [49]:
```
x = ["unique_questions" , "Repeated Questions"]
y =  [unique_qs , qs_morethan_onetime]

plt.figure(figsize=(10, 6))
plt.title ("Plot representing unique and repeated questions  ")
sns.barplot(x,y)
plt.show()
```



3.2.3 Checking for Duplicates

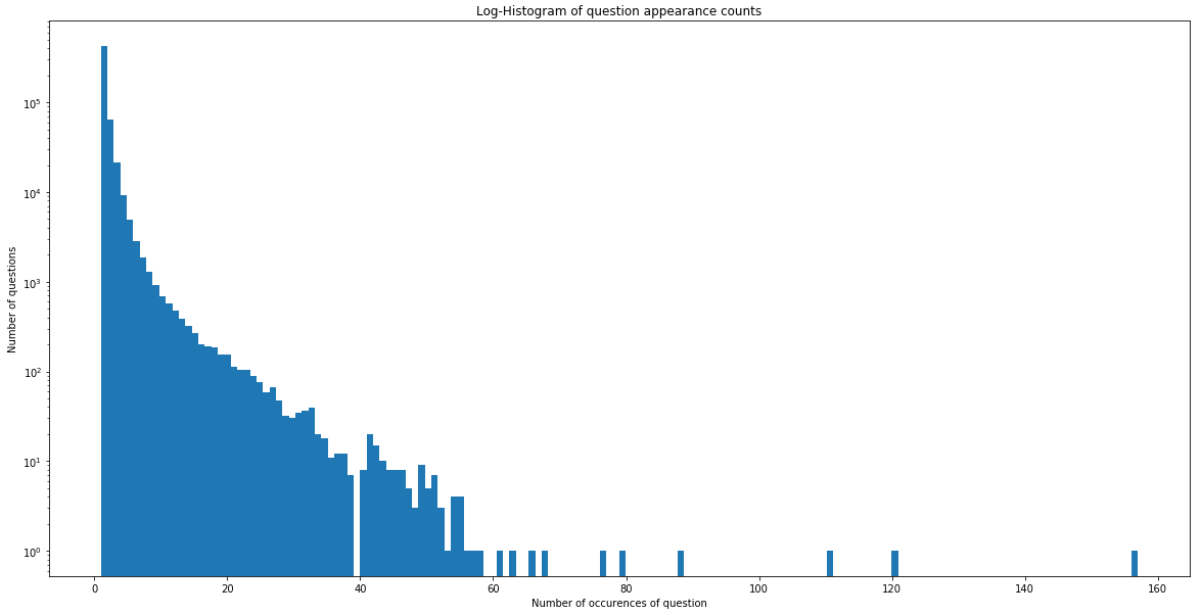
In [52]:
```
#No Such Duplicates in dataset
df.drop_duplicates(subset = {'qid1','qid2','is_duplicate'},keep = 'first' ,inp
lace = True)
print(df.shape)
```

```
(404290, 6)
```

3.2.4 Number of occurrences of each question

In [57]:
```
plt.figure(figsize=(20, 10))

plt.hist(qids.value_counts(), bins=160)

plt.yscale('log', nonposy='clip')

plt.title('Log-Histogram of question appearance counts')

plt.xlabel('Number of occurences of question')

plt.ylabel('Number of questions')

print ('Maximum number of times a single question is repeated: {}\n'.format(ma
x(qids.value_counts())))
```

```
Maximum number of times a single question is repeated: 157
```



3.2.5 Checking for NULL values

In [66]:
```
nan_rows = df.iloc[np.where(df.isnull())]
print(nan_rows)
```

```
Empty DataFrame
Columns: []
Index: []
```

In [67]:
```
# Filling the null values with ' '
df = df.fillna('')
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
```

```
Empty DataFrame
Columns: [id, qid1, qid2, question1, question2, is_duplicate]
Index: []
```

3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like:

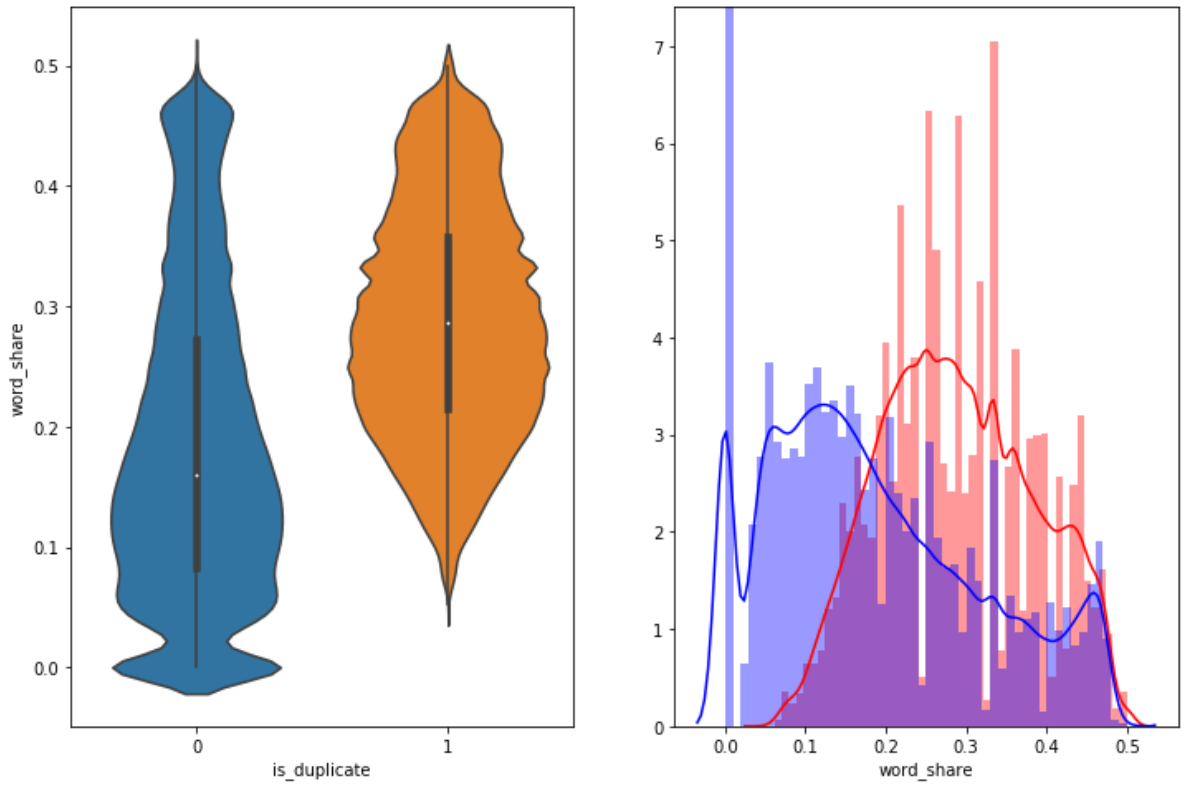
- **freq_qid1** = Frequency of qid1's
- **freq_qid2** = Frequency of qid2's
- **q1len** = Length of q1
- **q2len** = Length of q2
- **q1_n_words** = Number of words in Question 1
- **q2_n_words** = Number of words in Question 2
- **word_Common** = (Number of common unique words in Question 1 and Question 2)
- **word_Total** =(Total num of words in Question 1 + Total num of words in Question 2)
- **word_share** = (word_common)/(word_Total)
- **freq_q1+freq_q2** = sum total of frequency of qid1 and qid2
- **freq_q1-freq_q2** = absolute difference of frequency of qid1 and qid2

In [70]:
```
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-
1')
    df.fillna('')
else:
    df['freq_qid1'] = df.groupby('qid1')['qid1'].transform('count')
    df['freq_qid2'] = df.groupby('qid2')['qid2'].transform('count')
    df['q1len'] = df['question1'].str.len()
    df['q2len'] = df['question2'].str.len()
    df['q1_n_words'] = df['question1'].apply(lambda row: len(row.split(" ")))
    df['q2_n_words'] = df['question2'].apply(lambda row: len(row.split(" ")))

    def normalized_word_Common(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split
(" ")))
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split
(" ")))
        return 1.0 * len(w1 & w2)
    df['word_Common'] = df.apply(normalized_word_Common, axis=1)

    def normalized_word_Total(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split
(" ")))
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split
(" ")))
        return 1.0 * (len(w1) + len(w2))
    df['word_Total'] = df.apply(normalized_word_Total, axis=1)

    def normalized_word_share(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split
(" ")))
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split
(" ")))
        return 1.0 * len(w1 & w2)/(len(w1) + len(w2))
    df['word_share'] = df.apply(normalized_word_share, axis=1)

    df['freq_q1+q2'] = df['freq_qid1']+df['freq_qid2']
    df['freq_q1-q2'] = abs(df['freq_qid1']-df['freq_qid2'])

    df.to_csv("df_fe_without_preprocessing_train.csv", index=False)

df.head()
```

Out[70]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2le
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	1	73	59
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0	1	1	50	65
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39

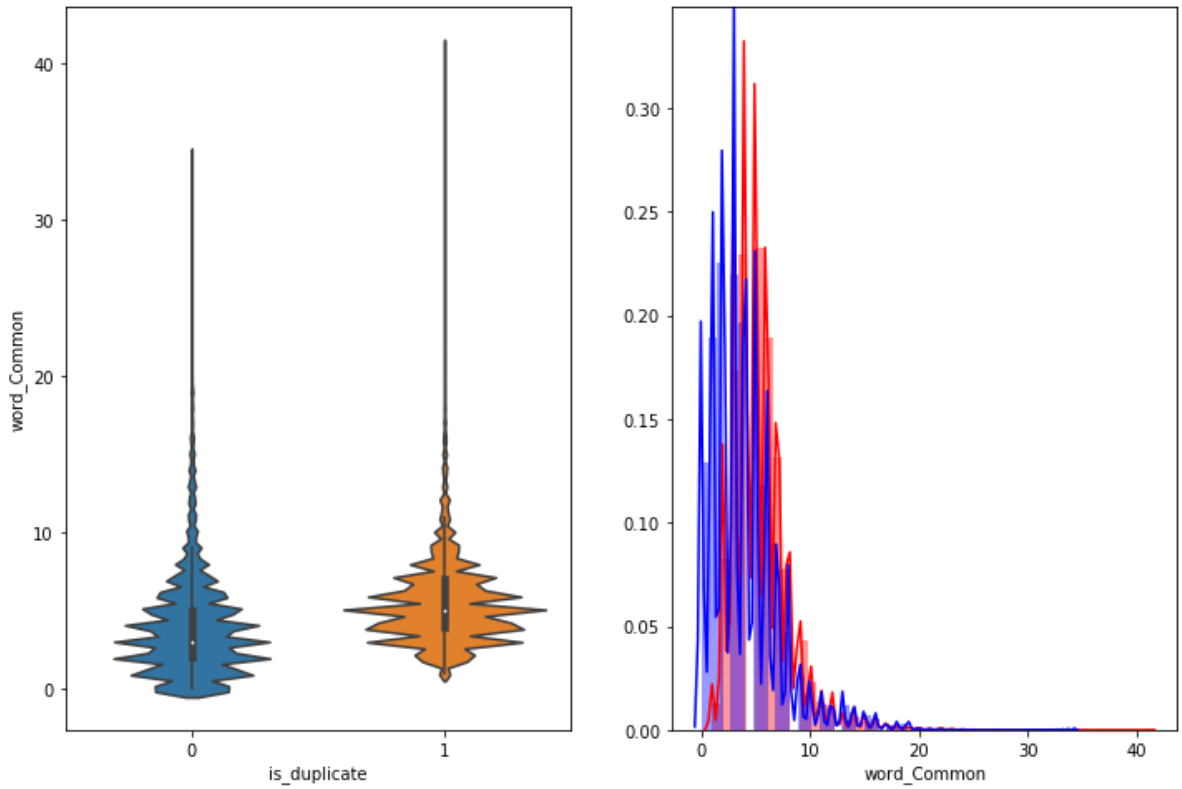
In [76]:
```
plt.figure(figsize=(12, 8))

plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'word_share', data = df)

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['word_share'] , label = "1", color
= 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['word_share'] , label = "0" , color
 = 'blue' )
plt.show()
```



- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

Feature: word_Common

In [77]:
```
plt.figure(figsize=(12, 8))

plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'word_Common', data = df)

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['word_Common'] , label = "1", color
 = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['word_Common'] , label = "0" , colo
r = 'blue' )
plt.show()
```



The distributions of the word_Common feature in similar and non-similar questions are highly overlapping

EDA: Advanced Feature Extraction.

In [85]:
```
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin_
1',engine = 'python')
    df = df.fillna('')
else:
    print("get df_fe_without_preprocessing_train.csv from drive or run the pre
vious notebook")
```

In [87]:
```
df.head(2)
```

Out[87]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88

Preprocessing of Text

- Preprocessing:
  - Removing html tags
  - Removing Punctuations
  - Performing stemming
  - Removing Stopwords
  - Expanding contractions etc.

In [100]:
```
STOP_WORDS = stopwords.words("english")
def preprocess(x):
    x = str(x).lower()
    x = x.replace(",000,000","m").replace(",000","k").replace("what's","what i
s").replace("won't","will not").replace("isn't","is not")\
                           .replace("’", "'").replace("’", "'").replace("n't"
," not").replace("cannot","can not").replace("can't","can not")\
                           .replace("'ve"," have").replace("i'm","i am").repl
ace("'re"," are")\
                           .replace("he's","he is").replace("she's","she is")
.replace("'s"," own")\
                           .replace("%", " percent ").replace("₹", " rupee ")
.replace("$", " dollar ")\
                          .replace("€", " euro ").replace("'ll", " will")

    x = re.sub(r"([0-9]+)000000",r"\1m",x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)

    porter =  PorterStemmer()
    pattern = re.compile("\W")

    if type(x) == type(''):
        x = re.sub(pattern,' ',x)

    if type(x) == type(''):
        x = porter.stem(x)
        bs4 = BeautifulSoup(x)
        x = bs4.get_text()

    return x
```

- Function to Compute and get the features : With 2 parameters of Question 1 and Question 2

Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

- **Token**: You get a token by splitting sentence a space
- **Stop_Word** : stop words as per NLTK.
- **Word** : A token that is not a stop_word

Features:

- **cwc_min** : Ratio of common_word_count to min lenghth of word count of Q1 and Q2
  cwc_min = common_word_count / (min(len(q1_words), len(q2_words))

- **cwc_max** : Ratio of common_word_count to max lenghth of word count of Q1 and Q2
  cwc_max = common_word_count / (max(len(q1_words), len(q2_words))

- **csc_min** : Ratio of common_stop_count to min lenghth of stop count of Q1 and Q2
  csc_min = common_stop_count / (min(len(q1_stops), len(q2_stops))

- **csc_max** : Ratio of common_stop_count to max lenghth of stop count of Q1 and Q2
  csc_max = common_stop_count / (max(len(q1_stops), len(q2_stops))

- **ctc_min** : Ratio of common_token_count to min lenghth of token count of Q1 and Q2
  ctc_min = common_token_count / (min(len(q1_tokens), len(q2_tokens))

- **ctc_max** : Ratio of common_token_count to max lenghth of token count of Q1 and Q2
  ctc_max = common_token_count / (max(len(q1_tokens), len(q2_tokens))

- **last_word_eq** : Check if First word of both questions is equal or not
  last_word_eq = int(q1_tokens[-1] == q2_tokens[-1])

- **first_word_eq** : Check if First word of both questions is equal or not
  first_word_eq = int(q1_tokens[0] == q2_tokens[0])

- **abs_len_diff** : Abs. length difference
  abs_len_diff = abs(len(q1_tokens) - len(q2_tokens))

- **mean_len** : Average Token Length of both Questions
  mean_len = (len(q1_tokens) + len(q2_tokens))/2

- **fuzz_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

- **fuzz_partial_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

- **token_sort_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

- **token_set_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

- **longest_substr_ratio** : Ratio of length longest common substring to min lenghth of token count of Q1 and Q2
  longest_substr_ratio = len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

In [105]:
```
SAFE_DIV = 0.0001
def get_token_feature(q1,q2):
    token_feature =[0.0]*10
    q1_token = q1.split()
    q2_token = q2.split()
    if len(q1_token) == 0 or len(q2_token) == 0:
        return token_feature
    else:
        q1_word = set([word for word in q1_token if word not in STOP_WORDS])
        q2_word = set([word for word in q2_token if word not in STOP_WORDS])
        q1_stop = set([stop for stop in q1_token if stop in STOP_WORDS])
        q2_stop = set([stop for stop in q2_token if stop in STOP_WORDS])

        common_word_count = len(q1_word.intersection(q2_word))
        common_token_count = len(set(q1_token).intersection(set(q2_token)))
        common_stop_count = len(q1_stop.intersection(q2_stop))

        token_feature[0] = common_word_count/(min(len(q1_word),len(q2_word)) +
 SAFE_DIV)
        token_feature[1] = common_word_count/(max(len(q1_word),len(q2_word)) +
 SAFE_DIV)
        token_feature[2] = common_stop_count/(min(len(q1_stop),len(q2_stop)) +
 SAFE_DIV)
        token_feature[3] = common_stop_count/(max(len(q1_stop),len(q2_stop)) +
 SAFE_DIV)
        token_feature[4] = common_token_count/(min(len(q1_token),len(q2_token
)) + SAFE_DIV)
        token_feature[5] = common_token_count/(max(len(q1_token),len(q2_token
)) + SAFE_DIV)

        token_feature[6] = int(q1_token[-1] == q2_token[-1])
        token_feature[7] = int(q1_token[0] == q2_token[0])

        token_feature[8] = abs(len(q1_token) - len(q2_token))
        token_feature[9] = (len(q1_token) + len(q2_token))/2
        return token_feature
```

In [119]:
```
from difflib import SequenceMatcher
def matchsubstring(m,n):
    seqMatch = SequenceMatcher(None,m,n)
    match = seqMatch.find_longest_match(0, len(m), 0, len(n))
    if (match.size!=0):
        return len(m[match.a: match.a + match.size])/(min(len(m), len(n)))
    else:
        return 0
```

In [120]:
```
def advance_feature(df):
    df['question1'] = df['question1'].fillna('').apply(preprocess)
    df['question2'] = df['question2'].fillna('').apply(preprocess)

    token_feature = df.apply(lambda x : get_token_feature(x['question1'],x['qu
estion2']),axis = 1)

    df["cwc_min"]       = list(map(lambda x: x[0], token_feature))
    df["cwc_max"]       = list(map(lambda x: x[1], token_feature))
    df["csc_min"]       = list(map(lambda x: x[2], token_feature))
    df["csc_max"]       = list(map(lambda x: x[3], token_feature))
    df["ctc_min"]       = list(map(lambda x: x[4], token_feature))
    df["ctc_max"]       = list(map(lambda x: x[5], token_feature))
    df["last_word_eq"]  = list(map(lambda x: x[6], token_feature))
    df["first_word_eq"] = list(map(lambda x: x[7], token_feature))
    df["abs_len_diff"]  = list(map(lambda x: x[8], token_feature))
    df["mean_len"]      = list(map(lambda x: x[9], token_feature))

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["q
uestion1"], x["question2"]), axis=1)
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x[
"question1"], x["question2"]), axis=1)
    #QRatio ----> Quick Ratio
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"
], x["question2"]), axis=1)
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["que
stion1"], x["question2"]), axis=1)
    df["longest_substr_ratio"]  = df.apply(lambda x:  matchsubstring(x["questi
on1"], x["question2"]), axis=1)
    return df
```

In [121]:
```
if os.path.isfile('nlp_features_train.csv'):
    df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv("train.csv")
    df = advance_feature(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head(2)
```

```
Extracting features for train:
```

Out[121]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.99
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	0.799984	0.399996	0.749981	0.59

2 rows × 21 columns

Analysis of extracted features

Plotting Word clouds

- Creating Word Cloud of Duplicates and Non-Duplicates Question pairs
- We can observe the most frequent occuring words

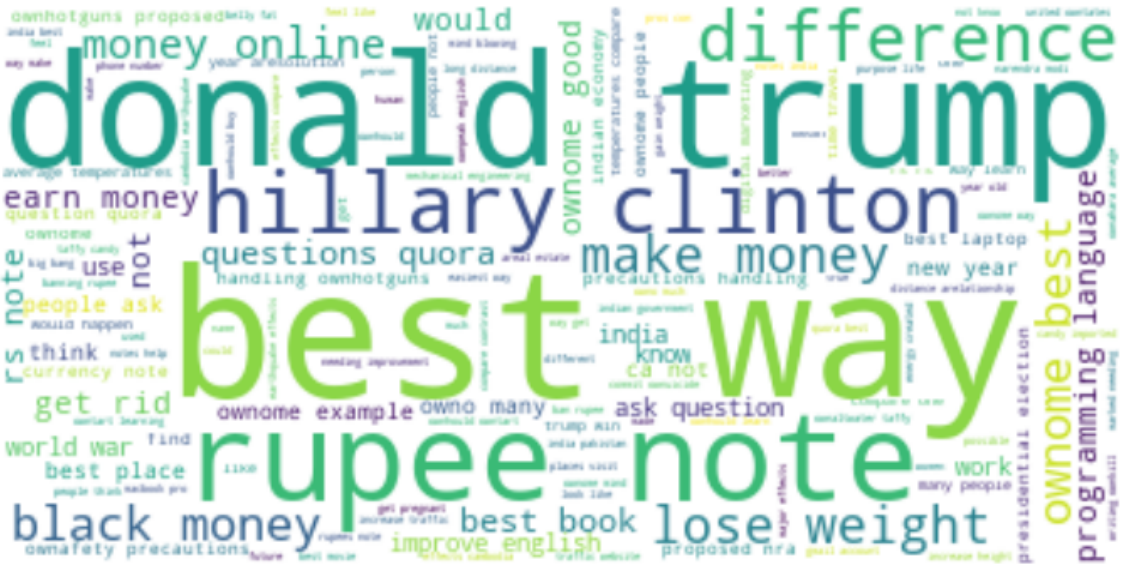
In [259]:
```
class_1 = df[df['is_duplicate'] == 1]
class_0 = df[df['is_duplicate'] == 0]

positive_qs_pair = np.array([class_1['question1'] , class_1['question2']]).fla
tten()
negative_qs_pair = np.array([class_0['question1'] , class_0['question2']]).fla
tten()

print ("Number of data points in class 1 (duplicate pairs) :",len(positive_qs_
pair))
print ("Number of data points in class 0 (non duplicate pairs) :",len(negative
_qs_pair))

stopword = set(STOP_WORDS)
stopword.add('said')
stopword.add('br')
stopword.add(' ')

stopword.remove('not')
stopword.remove('no')
stopword.add("what'")
stopword.add("how'")
stopword.add("is'")
stopword.add("do'")
#stopword.remove('like')
#print(stopword)
```

```
Number of data points in class 1 (duplicate pairs) : 298526
Number of data points in class 0 (non duplicate pairs) : 510054
```

Word Clouds generated from duplicate pair question's text

In [261]:
```
positive = []
for i in positive_qs_pair:
    x = i.split()
    for j in range(len(x)):
        if (x[j].isalpha()) :
            if type(x[j]) is type(''):
                positive.append(x[j])
wc = WordCloud(background_color="white", max_words = len(positive), stopwords=
stopword)
positive = preprocess(positive)
wc.generate(str(positive))
print ("Word Cloud for Duplicate Question pairs")
plt.figure(figsize=(12, 12))
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Word Cloud for Duplicate Question pairs



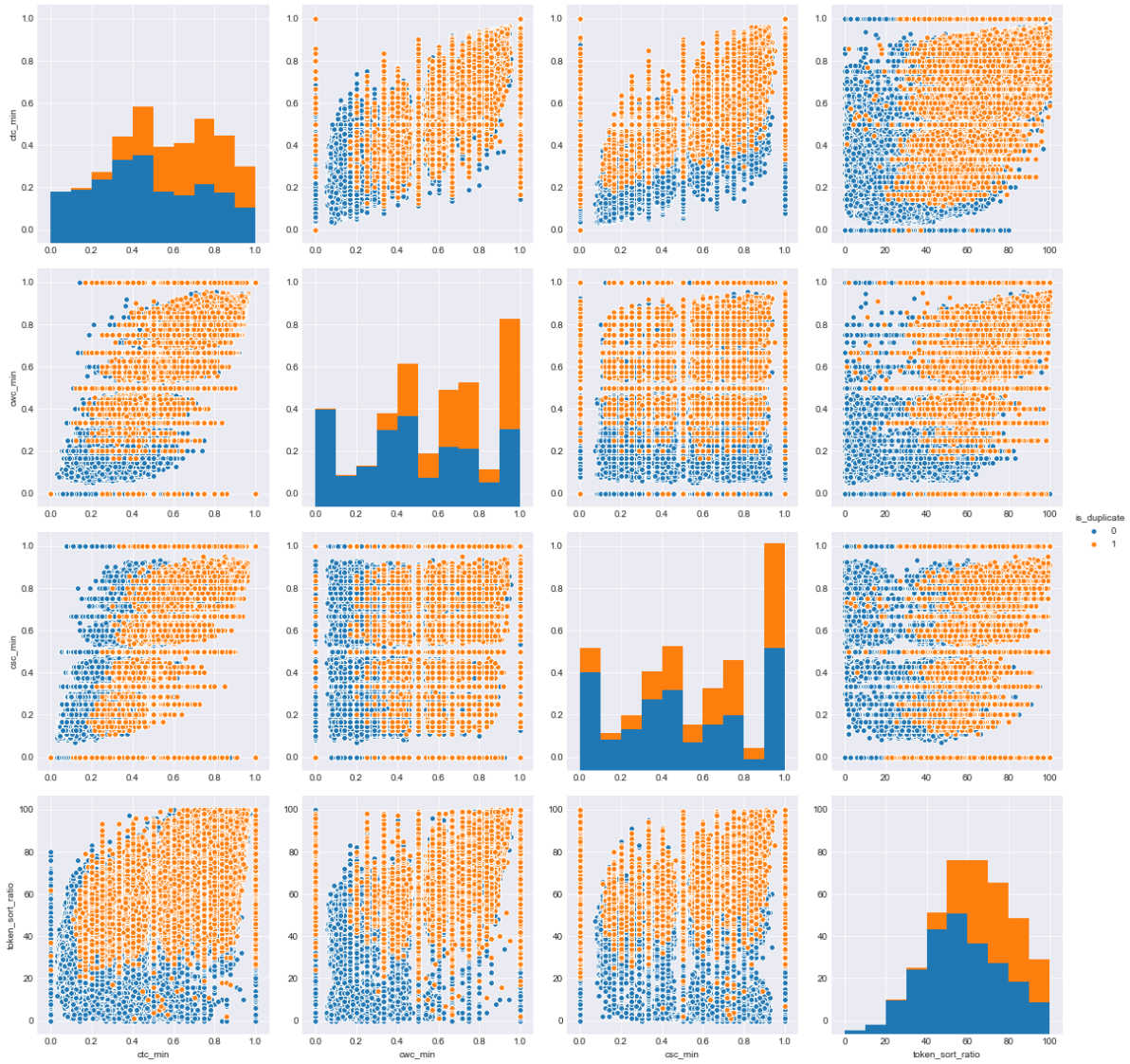
Word Clouds generated from non duplicate pair question's text

In [263]:
```
negative = []
for i in negative_qs_pair:
    x = i.split()
    for j in range(len(x)):
        if (x[j].isalpha()) :
            if type(x[j]) is type(''):
                negative.append(x[j])
wc = WordCloud(background_color="white", max_words = len(negative), stopwords=
stopword)
negative = preprocess(negative)
wc.generate(str(negative))
print ("Word Cloud for Duplicate Question pairs")
plt.figure(figsize=(12, 12))
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

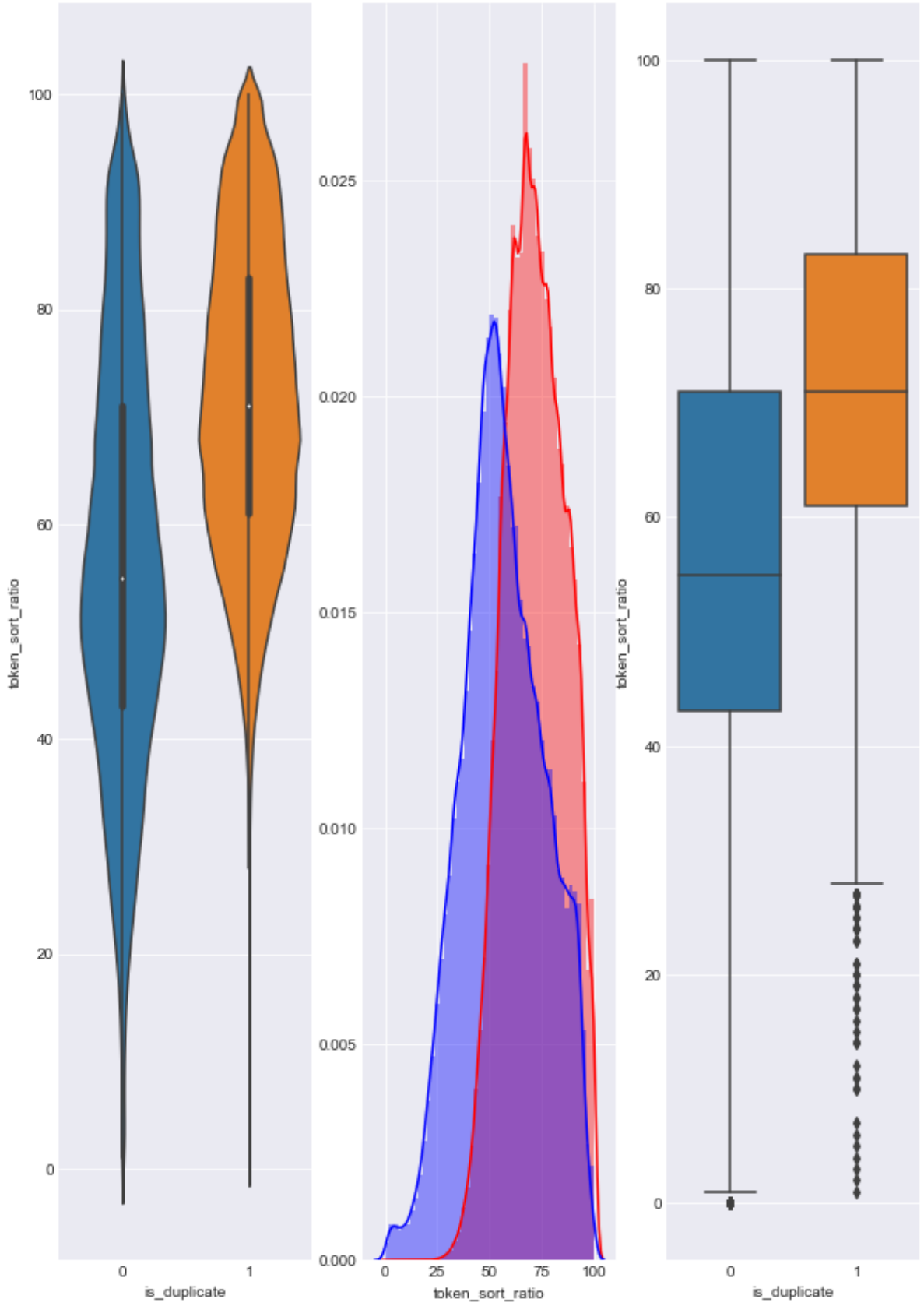
```
Word Cloud for Duplicate Question pairs
```

Pair plot of features ['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio']

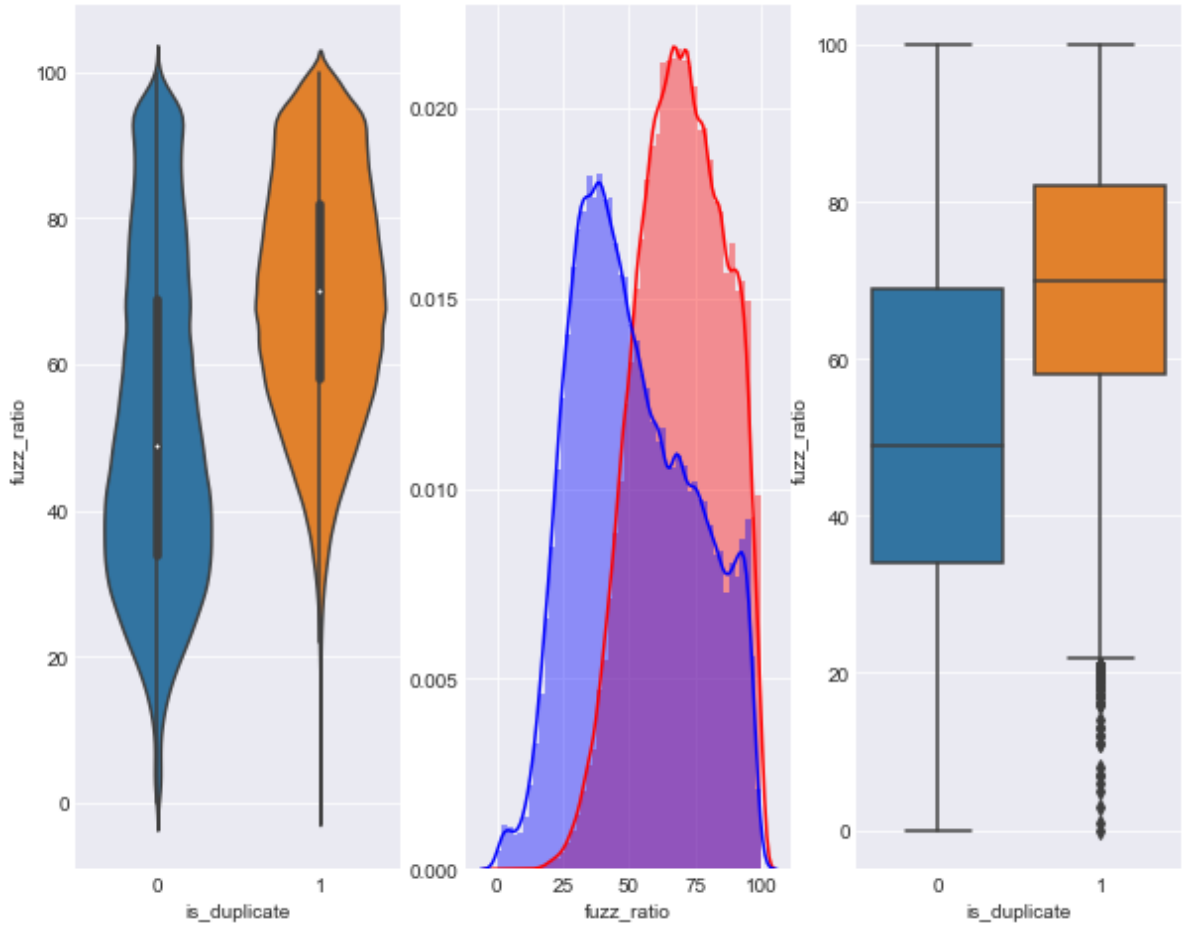
In [270]:
```
plt.close()
sns.set_style('darkgrid')
sns.pairplot(data = df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio','is_duplicate']],vars=['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio'], hue
 = 'is_duplicate' , size  = 4)
plt.show()
```



In [275]:
```
plt.figure(figsize=(10, 15))

plt.subplot(1,3,1)
sns.violinplot(x = 'is_duplicate', y = 'token_sort_ratio', data = df)

plt.subplot(1,3,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['token_sort_ratio'] , label = "1",
color = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'] , label = "0" ,
 color = 'blue' )

plt.subplot(1,3,3)
sns.boxplot(x = 'is_duplicate', y = 'token_sort_ratio', data = df)
plt.show()
```


token_sort_ratio
is_duplicate
0
1
0
20
40
60
80
100
0.000
0.005
0.010
0.015
0.020
0.025
0
25
50
75
100

In [276]:
```
plt.figure(figsize=(10, 8))

plt.subplot(1,3,1)
sns.violinplot(x = 'is_duplicate', y = 'fuzz_ratio', data = df)

plt.subplot(1,3,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['fuzz_ratio'] , label = "1", color
= 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['fuzz_ratio'] , label = "0" , color
 = 'blue' )

plt.subplot(1,3,3)
sns.boxplot(x = 'is_duplicate', y = 'fuzz_ratio', data = df)
plt.show()
```

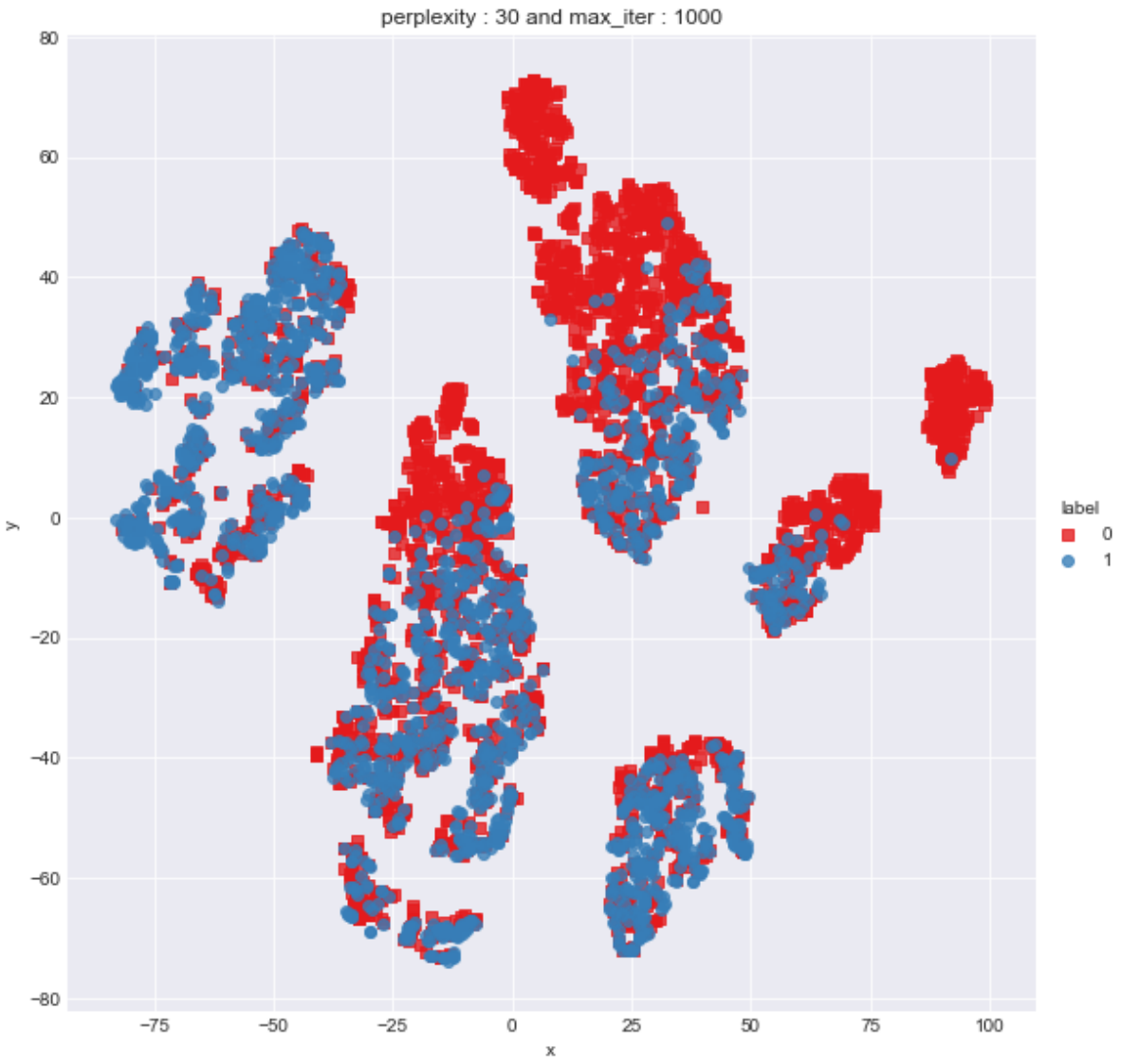


In [277]:
```
from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_mi
n', 'csc_max' , 'ctc_min' , 'ctc_max' , 'last_word_eq', 'first_word_eq' , 'abs
_len_diff' , 'mean_len' , 'token_set_ratio' , 'token_sort_ratio' ,  'fuzz_ratio
' , 'fuzz_partial_ratio' , 'longest_substr_ratio']])
y = dfp_subsampled['is_duplicate'].values
```

In [278]:
```
tsne2d = TSNE(
    n_components=2,
    init='random',#default
    random_state=101,
    method='barnes_hut',#default
    n_iter=1000,#default
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.058s...
[t-SNE] Computed neighbors for 5000 samples in 0.520s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.130781
[t-SNE] Computed conditional probabilities in 0.342s
[t-SNE] Iteration 50: error = 81.2424240, gradient norm = 0.0462601 (50 itera
tions in 9.863s)
[t-SNE] Iteration 100: error = 70.6049881, gradient norm = 0.0099202 (50 iter
ations in 6.731s)
[t-SNE] Iteration 150: error = 68.9059601, gradient norm = 0.0057159 (50 iter
ations in 6.456s)
[t-SNE] Iteration 200: error = 68.1044769, gradient norm = 0.0041706 (50 iter
ations in 6.863s)
[t-SNE] Iteration 250: error = 67.6028976, gradient norm = 0.0041267 (50 iter
ations in 6.836s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.602898
[t-SNE] Iteration 300: error = 1.7926643, gradient norm = 0.0011838 (50 itera
tions in 7.208s)
[t-SNE] Iteration 350: error = 1.3946527, gradient norm = 0.0004873 (50 itera
tions in 7.137s)
[t-SNE] Iteration 400: error = 1.2280741, gradient norm = 0.0002811 (50 itera
tions in 7.585s)
[t-SNE] Iteration 450: error = 1.1384543, gradient norm = 0.0001902 (50 itera
tions in 7.106s)
[t-SNE] Iteration 500: error = 1.0833004, gradient norm = 0.0001424 (50 itera
tions in 7.103s)
[t-SNE] Iteration 550: error = 1.0473055, gradient norm = 0.0001180 (50 itera
tions in 7.095s)
[t-SNE] Iteration 600: error = 1.0231537, gradient norm = 0.0000995 (50 itera
tions in 7.361s)
[t-SNE] Iteration 650: error = 1.0061477, gradient norm = 0.0000879 (50 itera
tions in 7.127s)
[t-SNE] Iteration 700: error = 0.9949846, gradient norm = 0.0000821 (50 itera
tions in 7.128s)
[t-SNE] Iteration 750: error = 0.9871543, gradient norm = 0.0000734 (50 itera
tions in 7.094s)
[t-SNE] Iteration 800: error = 0.9808751, gradient norm = 0.0000709 (50 itera
tions in 7.284s)
[t-SNE] Iteration 850: error = 0.9757624, gradient norm = 0.0000671 (50 itera
tions in 7.748s)
[t-SNE] Iteration 900: error = 0.9713756, gradient norm = 0.0000611 (50 itera
tions in 9.919s)
[t-SNE] Iteration 950: error = 0.9674607, gradient norm = 0.0000630 (50 itera
tions in 7.803s)
[t-SNE] Iteration 1000: error = 0.9642618, gradient norm = 0.0000562 (50 iter
ations in 8.371s)
[t-SNE] Error after 1000 iterations: 0.964262
```

In [280]:
```
df_tsne_2d = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] ,'label':y})

# draw the plot in appropriate place in the grid
sns.lmplot(data=df_tsne_2d, x='x', y='y', hue='label', fit_reg=False, size=8,p
alette="Set1",markers=['s','o'])
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```

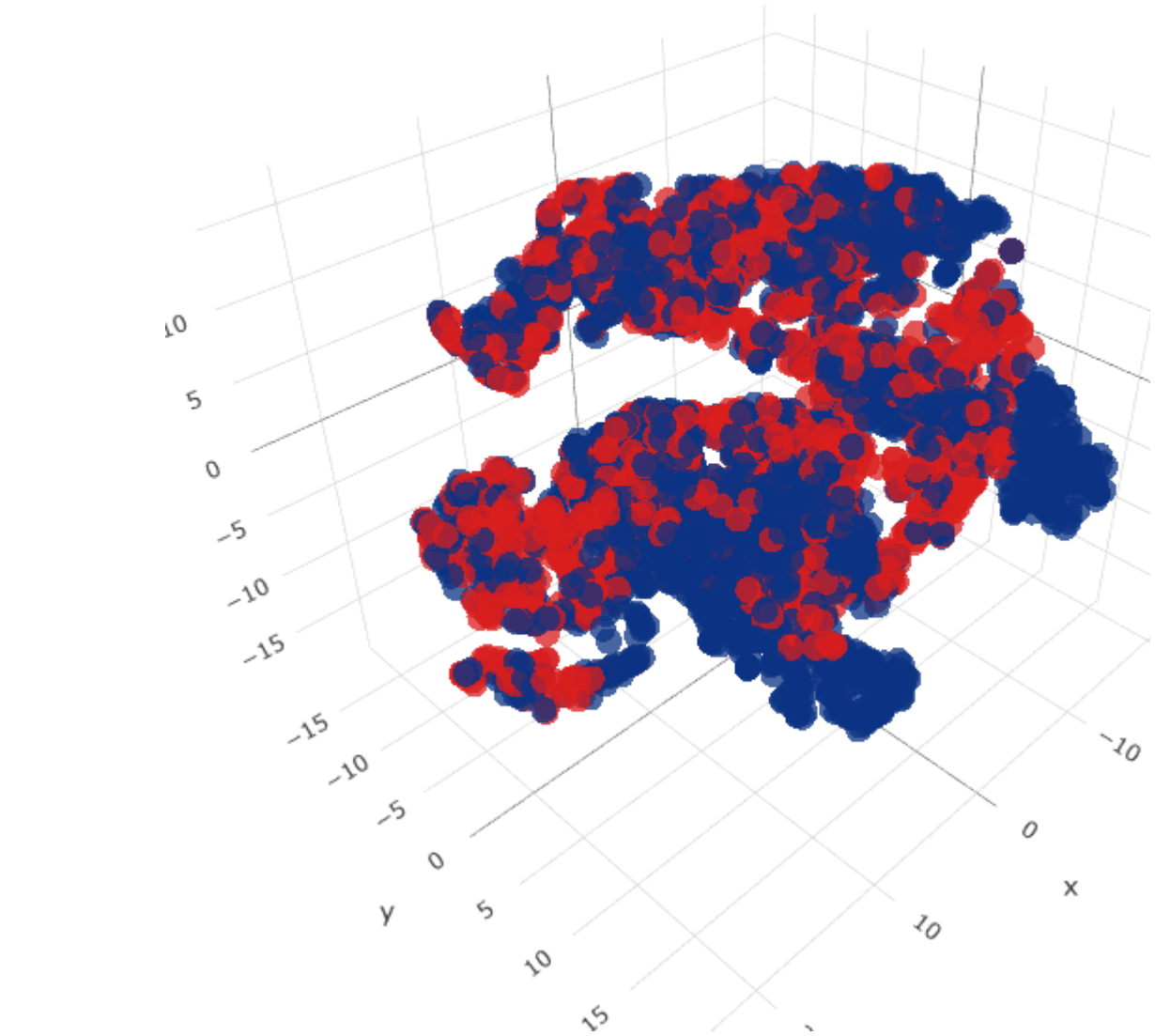


In [281]:
```
from sklearn.manifold import TSNE
tsne3d = TSNE(
    n_components=3,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.016s...
[t-SNE] Computed neighbors for 5000 samples in 0.516s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.130781
[t-SNE] Computed conditional probabilities in 0.262s
[t-SNE] Iteration 50: error = 80.4752350, gradient norm = 0.0326151 (50 itera
tions in 18.903s)
[t-SNE] Iteration 100: error = 69.3851929, gradient norm = 0.0036226 (50 iter
ations in 9.194s)
[t-SNE] Iteration 150: error = 67.9767609, gradient norm = 0.0016960 (50 iter
ations in 8.059s)
[t-SNE] Iteration 200: error = 67.4179535, gradient norm = 0.0011948 (50 iter
ations in 8.127s)
[t-SNE] Iteration 250: error = 67.1090927, gradient norm = 0.0008714 (50 iter
ations in 8.135s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.109093
[t-SNE] Iteration 300: error = 1.5234940, gradient norm = 0.0007018 (50 itera
tions in 10.011s)
[t-SNE] Iteration 350: error = 1.1842527, gradient norm = 0.0001983 (50 itera
tions in 13.716s)
[t-SNE] Iteration 400: error = 1.0398459, gradient norm = 0.0001072 (50 itera
tions in 13.894s)
[t-SNE] Iteration 450: error = 0.9680670, gradient norm = 0.0000699 (50 itera
tions in 13.855s)
[t-SNE] Iteration 500: error = 0.9299541, gradient norm = 0.0000523 (50 itera
tions in 13.674s)
[t-SNE] Iteration 550: error = 0.9095010, gradient norm = 0.0000447 (50 itera
tions in 12.652s)
[t-SNE] Iteration 600: error = 0.8966766, gradient norm = 0.0000448 (50 itera
tions in 15.772s)
[t-SNE] Iteration 650: error = 0.8865085, gradient norm = 0.0000384 (50 itera
tions in 13.414s)
[t-SNE] Iteration 700: error = 0.8792743, gradient norm = 0.0000320 (50 itera
tions in 12.576s)
[t-SNE] Iteration 750: error = 0.8733341, gradient norm = 0.0000327 (50 itera
tions in 14.452s)
[t-SNE] Iteration 800: error = 0.8682353, gradient norm = 0.0000285 (50 itera
tions in 11.695s)
[t-SNE] Iteration 850: error = 0.8631920, gradient norm = 0.0000250 (50 itera
tions in 11.617s)
[t-SNE] Iteration 900: error = 0.8587981, gradient norm = 0.0000240 (50 itera
tions in 11.694s)
[t-SNE] Iteration 950: error = 0.8549855, gradient norm = 0.0000229 (50 itera
tions in 11.981s)
[t-SNE] Iteration 1000: error = 0.8515424, gradient norm = 0.0000218 (50 iter
ations in 11.638s)
[t-SNE] Error after 1000 iterations: 0.851542
```

In [283]:
```
trace1 = go.Scatter3d(
    x=tsne3d[:,0],
    y=tsne3d[:,1],
    z=tsne3d[:,2],
    mode='markers',
    marker=dict(
        sizemode='diameter',
        color = y,
        colorscale = 'Portland',
        colorbar = dict(title = 'duplicate'),
        line=dict(color='rgb(255, 255, 255)'),
        opacity=0.75
    )
)

df_tsne_3d=[trace1]
layout=dict(height=800, width=800, title='3d embedding with engineered feature
s')
fig=go.Figure(data=df_tsne_3d, layout=layout)
py.iplot(fig, filename='3DBubble')
```

3d embedding with engineered features



Conclusion

- We have constructed some basic features without preprocessing and some advance features with preprocessing from the question1 and question2.
- Creating Word Cloud of Duplicates and Non-Duplicates Question pairs.
- We can observe the most frequent occuring words.
- Using TSNE for Visulization of 15 Features(Generated after cleaning the data) to 2 dimension and 3 dimention.