```
!pip install fuzzywuzzy
```

```
Requirement already satisfied: fuzzywuzzy in
/usr/local/lib/python3.10/dist-packages (0.18.0)
```

Imports

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
import re
import resource
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk import word_tokenize, ngrams
import warnings

color = sns.color_palette()
stopwords = set(stopwords.words('english'))

%matplotlib inline
warnings.filterwarnings('ignore')
pd.options.mode.chained_assignment = None
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Data reading

```
df = pd.read_csv('/content/quora_duplicate_questions.tsv', sep='\t')
```

Data Understanding

```
print("Number of data points:",df.shape[0])
print("Number of features:",df.shape[1])
```

```
Number of data points: 404290
Number of features: 6
```

```
df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

```
df.tail()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 5,\n  \"fields\": [\n
{\n      \"column\": \"id\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 404285,\
n        \"max\": 404289,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          404286,\n          404289,\n
404287\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"qid1\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 225059,\n        \"min\": 18840,\n        \"max\": 537932,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          18840,\n
537932,\n          537928\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"qid2\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 167894,\n        \"min\": 155606,\n
\"max\": 537933,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          155606,\n          537933,\n
537929\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"question1\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"Do you believe there is life after death?\",\n
\"What is like to have sex with cousin?\",\n          \"What is one
coin?\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"question2\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"Is it true that there is life after death?\",\n
\"What is it like to have sex with your cousin?\",\n          \"What's
this coin?\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"is_duplicate\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


##EDA

```
len(df)
```

```
404290
```

Dataset with question pairs of 404287 size

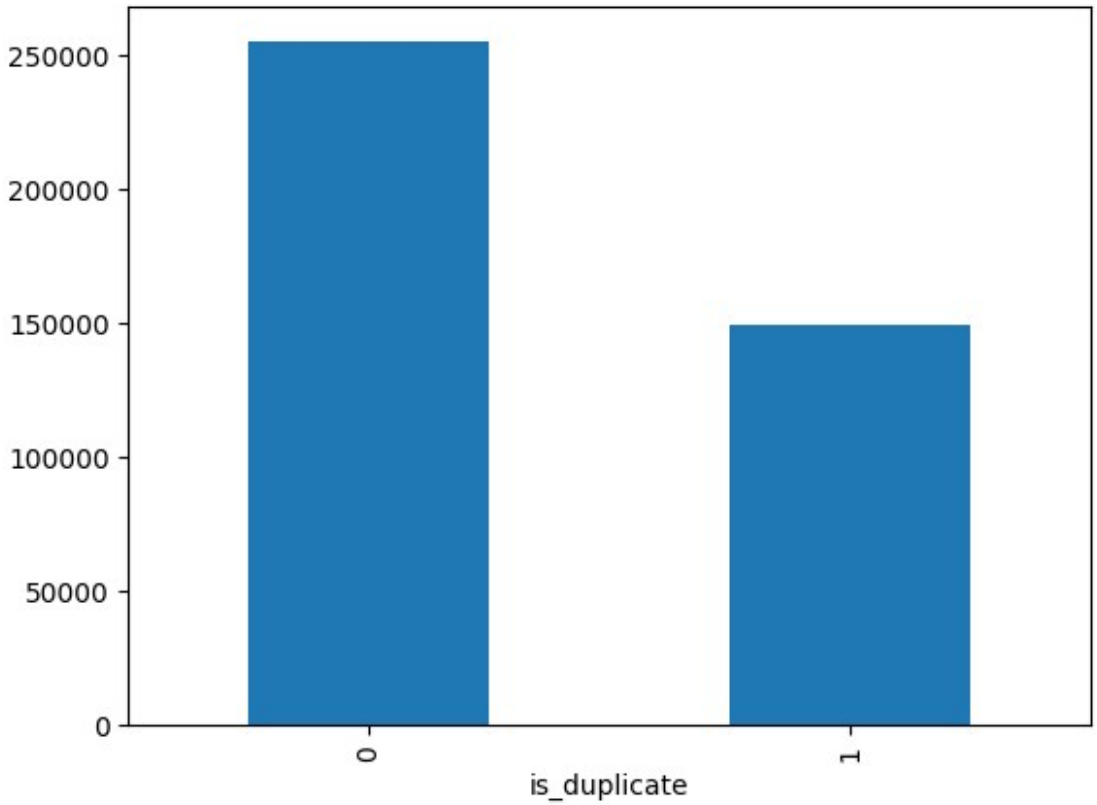
Target Variable Distribution

```
df['is_duplicate'].value_counts()
```

```
is_duplicate
0    255027
1    149263
Name: count, dtype: int64
```

```
df['is_duplicate'].value_counts().plot(kind='bar')
```

```
<Axes: xlabel='is_duplicate'>
```




```
len(df[df['is_duplicate'] == 0])/len(df) * 100
```

```
63.08021469737069
```

```
len(df[df['is_duplicate'] == 1])/len(df) * 100
```

```
36.9197853026293
```

We have 63% of non-similar question pairs and 37% of similar question pairs from the calculations above

This indicates that possibly our dataset is imbalanced.

Unique questions details

```
print(len(df[df.isnull().any(axis=1)]))
df = df.fillna('')
print(len(df[df.isnull().any(axis=1)]))
```

```
3
0
```

```
questions_combined = np.concatenate([df['question1'],
df['question2']])

full_questions_df = pd.DataFrame(questions_combined,
columns=['questions'])

full_questions_df['questions'].nunique()
```

```
537361
```

```
question_wise_value_counts =
full_questions_df['questions'].value_counts()

print(np.sum(question_wise_value_counts > 1))
print(round(np.sum(question_wise_value_counts >
1)/len(set(questions_combined)) * 100, 2))
```

```
111873
20.82
```

```
print(np.sum(question_wise_value_counts == 1))
print(round(np.sum(question_wise_value_counts ==
1)/len(set(questions_combined)) * 100, 2))
```

```
425488
79.18
```

So there are total 537361 unique questions in our dataset.

And there are 111872 questions repeated more than once which is 20.82% of unique questions.

And 425488 questions appeared only once means 79.18% of total unique questions

```
print(question_wise_value_counts.index[0])
question_wise_value_counts[0]
```

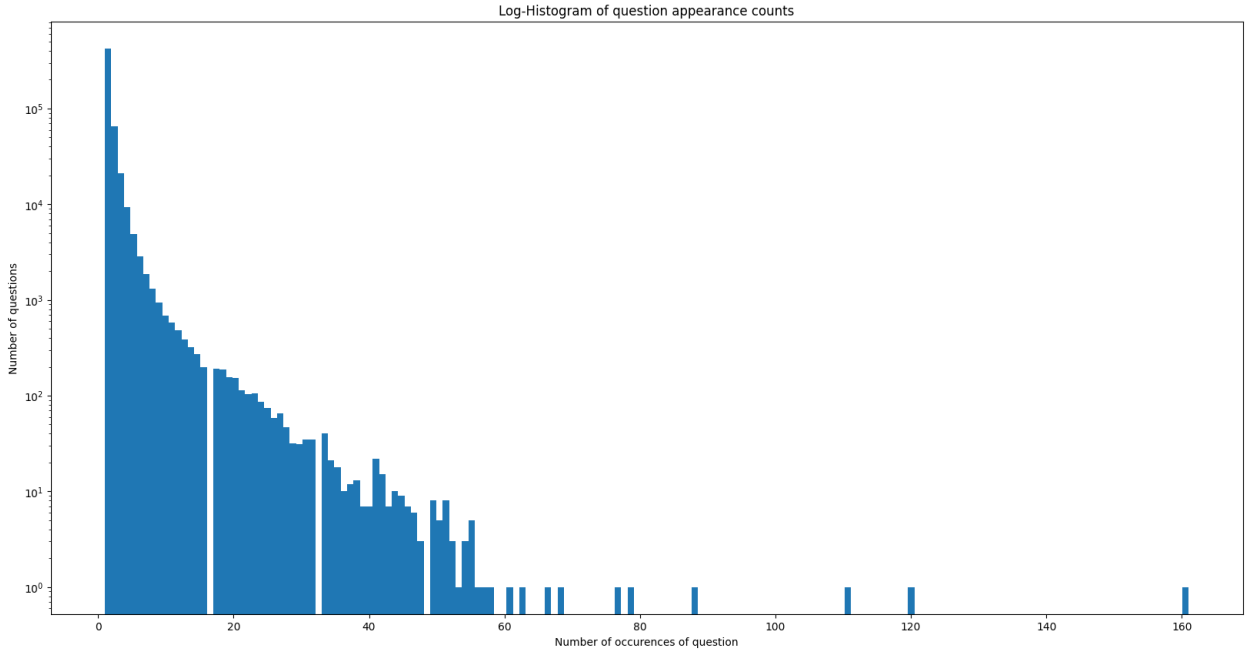
```
What are the best ways to lose weight?

161
```

Most common question: **What are the best ways to lose weight?**

It appeared 161 times

Each question occurences count

```
plt.figure(figsize=(20, 10))
plt.hist(question_wise_value_counts, bins=170)
plt.yscale('log')
plt.title('Log-Histogram of question appearance counts')
plt.xlabel('Number of occurences of question')
plt.ylabel('Number of questions')

Text(0, 0.5, 'Number of questions')
```



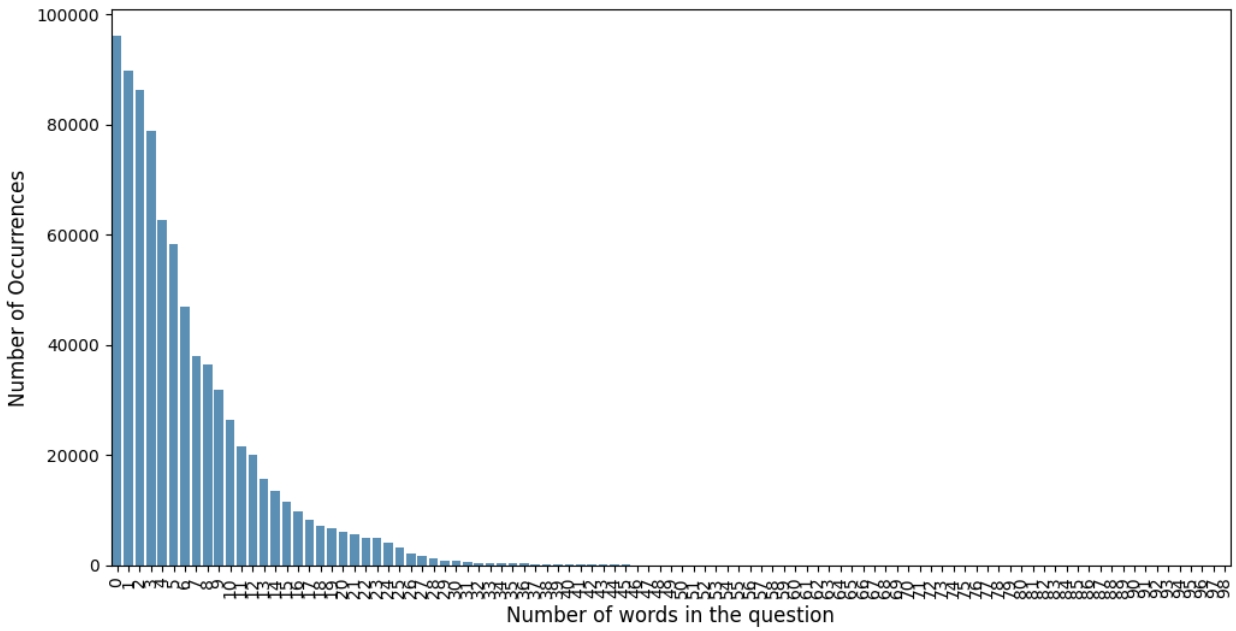
From the above histogram graph we can see that most of the questions repeated maxium of 50 times. Other questions those appeared more than 60 times are very less. As we know the most repeated question appeared to be 161 times.

Word distribution of questions

How words are distributed in question pairs. This feature allows us to understand word level distribution of the questions.

```
full_questions_df['number_of_words'] =
full_questions_df["questions"].str.split().apply(len)

cnt_srs = full_questions_df['number_of_words'].value_counts()

plt.figure(figsize=(12,6))
sns.barplot(cnt_srs.values, alpha=0.8, color=color[0])
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Number of words in the question', fontsize=12)
```

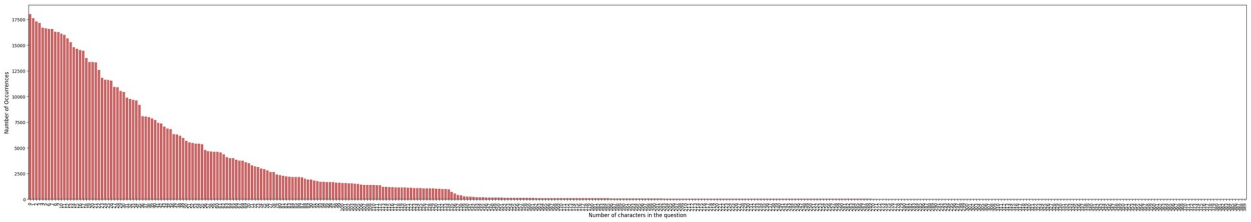
```
plt.xticks(rotation='vertical')
plt.show()
```



Character distribution

How character legths varies in question pairs. This feature allows us to understand character level distribution of the questions.

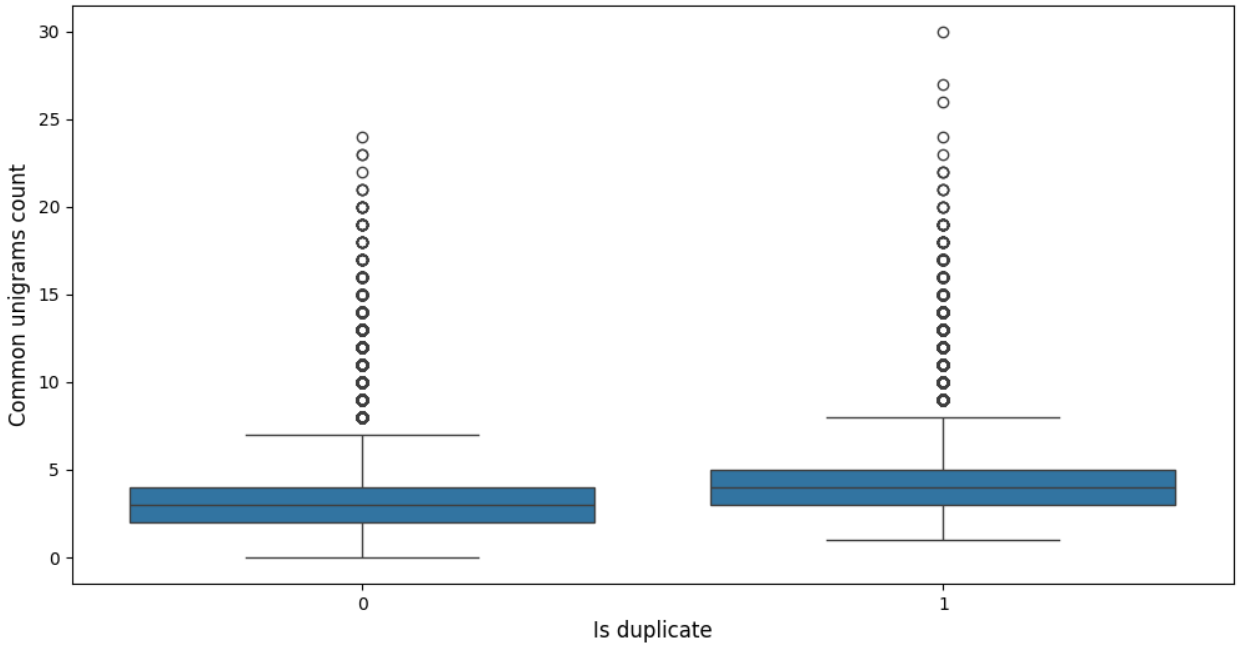
```
full_questions_df['number_of_chars'] =
full_questions_df["questions"].apply(len)

cnt_srs = full_questions_df['number_of_chars'].value_counts()

plt.figure(figsize=(50,8))
sns.barplot(cnt_srs.values, alpha=0.8, color=color[3])
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Number of characters in the question', fontsize=12)
plt.xticks(rotation='vertical')
plt.show()
```



Unigrams

By understanding the unigrams present in our question pairs we can understand common unigrams present in our questions.

```
def unigrams(question):
    return [word for word in word_tokenize(question.lower()) if word
not in stopwords]

def common_unigrams(row_data):
    return len(
set(row_data["unigrams_ques1"]).intersection(set(row_data["unigrams_qu
es2"])) )

def common_unigram_ratio(row_data):
    return float(row_data["unigrams_common_count"]) / max(len(
set(row_data["unigrams_ques1"]).union(set(row_data["unigrams_ques2"]))
),1)

df["unigrams_ques1"] = df['question1'].apply(lambda x:
unigrams(str(x)))
df["unigrams_ques2"] = df['question2'].apply(lambda x:
unigrams(str(x)))
df["unigrams_common_count"] = df.apply(lambda row:
common_unigrams(row),axis=1)
df["unigrams_common_ratio"] = df.apply(lambda row:
common_unigram_ratio(row), axis=1)

plt.figure(figsize=(12,6))
sns.boxplot(x="is_duplicate", y="unigrams_common_count", data=df)
plt.xlabel('Is duplicate', fontsize=12)
plt.ylabel('Common unigrams count', fontsize=12)
plt.show()
```



We see there is some count of common unigrams present in our question and we have some significant change in count among two target classes which will be useful for the modelling.

Data Preprocessing

Missing values

```
#Checking whether there are any rows with null values
null_rows = df[df.isnull().any(axis=1)]
null_rows
```

```
{"repr_error":"Out of range float values are not JSON compliant:
nan","type":"dataframe","variable_name":"null_rows"}
```

```
# Rows with null values are removed
print(df.shape)
df = df[~df.isnull().any(axis=1)]
print(df.shape)
```

```
(404290, 10)
(404290, 10)
```

Data Cleaning

Text data cleaning needs to be done to make sure we deal with proper data. Different NLP stratergies listed below are followed

```
def clean_sentence(text):
    if pd.isnull(text):
        return ''
```

```
    if type(text) != str or text=='':
        return ''
    ps = PorterStemmer()

    text = re.sub("\'s", " ", text)
    text = re.sub("i'm", "i am", text, flags=re.IGNORECASE)
    text = text.replace("can't", "can not")
    text = re.sub("\'re", " are ", text)
    text = text.replace(" whats ", " what is ")
    text = re.sub("\'ve", " have ", text)
    text = re.sub("n't", " not ", text)
    text = text.replace("b\.g\.", " bg ")
    text = re.sub("\'d", " would ", text)
    text = text.replace("\'ll", " will ")
    text = re.sub("i'm", "i am", text, flags=re.IGNORECASE)
    text = text.replace("e\.g\.", " eg ")
    text = re.sub("(the[\s]+|The[\s]+)?U\.S\.A\.", " America ", text,
flags=re.IGNORECASE)
    text = re.sub("(\d+)(kK)", " \g<1>000 ", text)
    text = re.sub("\(s\)", " ", text, flags=re.IGNORECASE)
    text = re.sub("[c-fC-F]\:\/", " disk ", text)

    text = re.sub('(?<=[0-9])\,(?=[0-9])', "", text)
    text = text.replace('\$', " dollar ")
    text = text.replace('\%', " percent ")
    text = text.replace('\&', " and ")

    text = ' '.join([word for word in text.split(" ") if word not in
stopwords]).lower()
    #text = ' '.join([ps.stem(word) for word in text])
    return text

df["question1"] = df["question1"].fillna("").apply(clean_sentence)
df["question2"] = df["question2"].fillna("").apply(clean_sentence)
```

Feature Engineering

Basic Textual Features

```
def get_unique_words(sentence):
  sent_split = sentence.split(" ")
  sent_split_processed = [word.lower().strip() for word in sent_split]
  return set(sent_split_processed)


def word_common_details(row):
    words1 = get_unique_words(row['question1'])
    words2 = get_unique_words(row['question2'])
    return 1.0 * len(words1 & words2)
```

```
def word_count_details(row):
    words1 = get_unique_words(row['question1'])
    words2 = get_unique_words(row['question2'])
    return 1.0 * (len(words1) + len(words2))

def word_share_data(row):
    words1 = get_unique_words(row['question1'])
    words2 = get_unique_words(row['question2'])
    return 1.0 * len(words1 & words2)/(len(words1) + len(words2))

df['q1len'] = df['question1'].str.len()
df['q2len'] = df['question2'].str.len()
df['diff_len'] = df.q1len - df.q2len

df['freq_qid1'] = df.groupby('qid1')['qid1'].transform('count')
df['freq_qid2'] = df.groupby('qid2')['qid2'].transform('count')
df['freq_q1+q2'] = df['freq_qid1']+df['freq_qid2']
df['freq_q1-q2'] = abs(df['freq_qid1']-df['freq_qid2'])

df['q1_n_words'] = df['question1'].apply(lambda row: len(row.split('
')))
df['q2_n_words'] = df['question2'].apply(lambda row: len(row.split('
')))

df['len_char_q1'] = df.question1.apply(lambda x:
len(''.join(set(str(x)))))
df['len_char_q2'] = df.question2.apply(lambda x:
len(''.join(set(str(x)))))

df['len_word_q1'] = df.question1.apply(lambda x: len(str(x).split()))
df['len_word_q2'] = df.question2.apply(lambda x: len(str(x).split()))

df['word_Common'] = df.apply(word_common_details, axis=1)
df['word_Total'] = df.apply(word_count_details, axis=1)
df['word_share'] = df.apply(word_share_data, axis=1)
```

Similarity Features

To get semantic features we used 'word2vec-google-news-300' model to get the vector representation of the questions first then find the similarity score using the different distance measures

```
import gensim.downloader as api
from scipy.spatial.distance import cosine, cityblock, jaccard,
canberra, euclidean, minkowski, braycurtis

model = api.load('word2vec-google-news-300')
```

```
def get_sent_vector_repr(sent):
    tokens = str(sent).lower()
    tokens = word_tokenize(tokens)
    tokens = [w for w in tokens if w.isalpha()]
    sentence_vector = []
    for w in tokens:
        try:
            sentence_vector.append(model[w])
        except:
            continue
    sentence_vector = np.array(sentence_vector)
    vector = sentence_vector.sum(axis=0)
    return vector / np.sqrt((vector ** 2).sum())

from tqdm.notebook import tqdm

vectors_question1 = np.zeros((df.shape[0], 300))
for index, question in enumerate(tqdm(df['question1'].values)):
    vectors_question1[index, :] = get_sent_vector_repr(question)

vectors_question2  = np.zeros((df.shape[0], 300))
for index, question in enumerate(tqdm(df['question2'].values)):
    vectors_question2[index, :] = get_sent_vector_repr(question)
```

{"model_id":"c2d0a9bb021d4120a1d2f9975f479387","version_major":2,"vers
ion_minor":0}

{"model_id":"3d9eaf22191846e49a4e95c25b5c432d","version_major":2,"vers
ion_minor":0}

```
df['cosine_distance'] = [cosine(x, y) for (x, y) in
zip(np.nan_to_num(vectors_question1),
np.nan_to_num(vectors_question2))]
df['cityblock_distance'] = [cityblock(x, y) for (x, y) in
zip(np.nan_to_num(vectors_question1),
np.nan_to_num(vectors_question2))]
df['jaccard_distance'] = [jaccard(x, y) for (x, y) in
zip(np.nan_to_num(vectors_question1),
np.nan_to_num(vectors_question2))]
df['canberra_distance'] = [canberra(x, y) for (x, y) in
zip(np.nan_to_num(vectors_question1),
np.nan_to_num(vectors_question2))]
df['euclidean_distance'] = [euclidean(x, y) for (x, y) in
zip(np.nan_to_num(vectors_question1),
np.nan_to_num(vectors_question2))]
df['minkowski_distance'] = [minkowski(x, y, 3) for (x, y) in
zip(np.nan_to_num(vectors_question1),
np.nan_to_num(vectors_question2))]
df['braycurtis_distance'] = [braycurtis(x, y) for (x, y) in
```

```
zip(np.nan_to_num(vectors_question1),
np.nan_to_num(vectors_question2))]
```

Substring level features

```
df['fuzz_qratio'] = df.apply(lambda x:
fuzz.QRatio(str(x['question1']), str(x['question2'])), axis=1)
df['fuzz_partial_ratio'] = df.apply(lambda x:
fuzz.partial_ratio(str(x['question1']), str(x['question2'])), axis=1)
# df['fuzz_WRatio'] = df.apply(lambda x:
fuzz.WRatio(str(x['question1']), str(x['question2'])), axis=1)
# df['fuzz_partial_token_set_ratio'] = df.apply(lambda x:
fuzz.partial_token_set_ratio(str(x['question1']),
str(x['question2'])), axis=1)
# df['fuzz_partial_token_sort_ratio'] = df.apply(lambda x:
fuzz.partial_token_sort_ratio(str(x['question1']),
str(x['question2'])), axis=1)
# df['fuzz_token_set_ratio'] = df.apply(lambda x:
fuzz.token_set_ratio(str(x['question1']), str(x['question2'])),
axis=1)
# df['fuzz_token_sort_ratio'] = df.apply(lambda x:
fuzz.token_sort_ratio(str(x['question1']), str(x['question2'])),
axis=1)
```

```
df.head()
```


```
{"type":"dataframe","variable_name":"df"}
```


```
df.columns
```

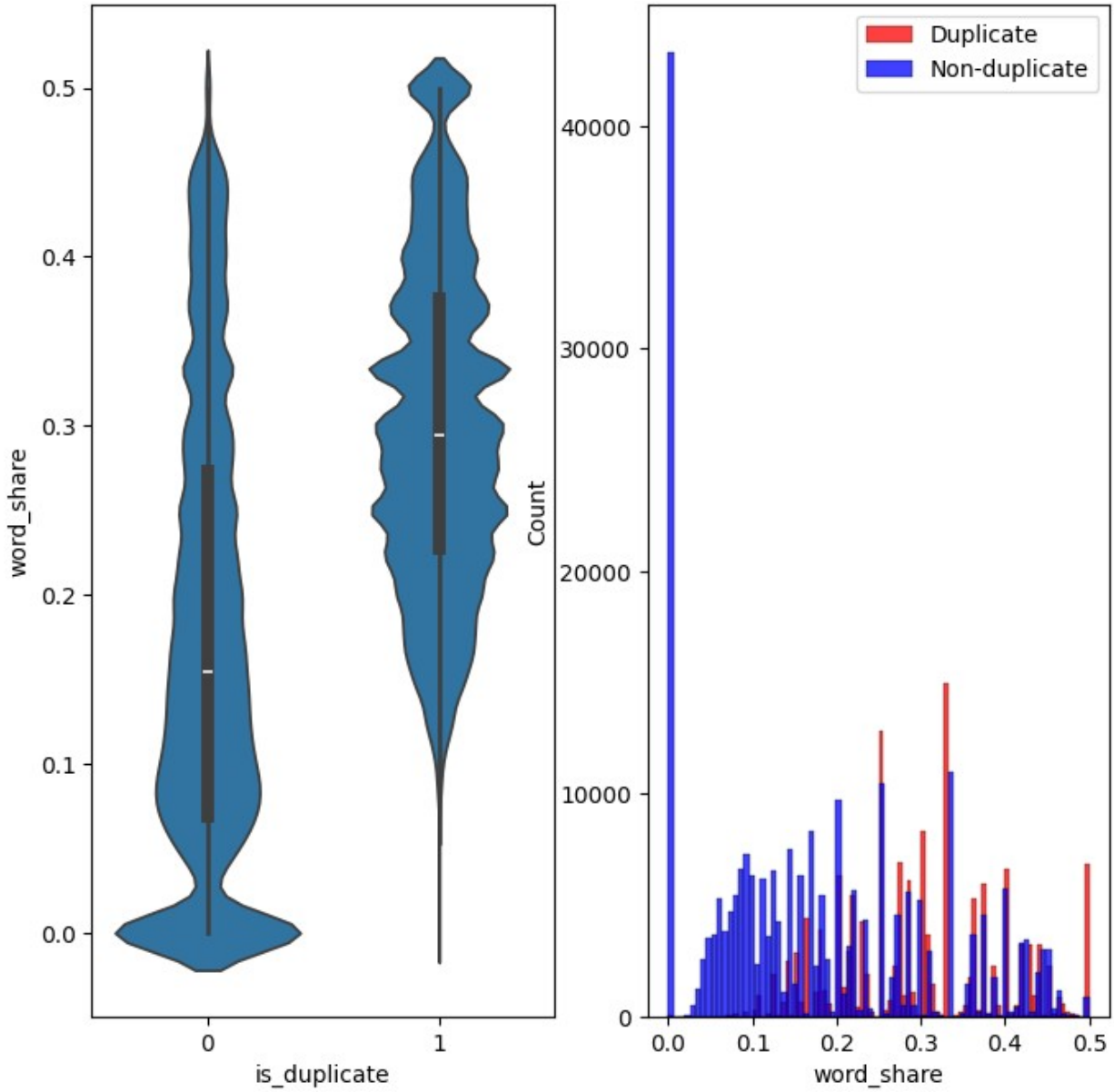
```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'unigrams_ques1', 'unigrams_ques2', 'unigrams_common_count',
       'unigrams_common_ratio', 'q1len', 'q2len', 'diff_len',
'freq_qid1',
       'freq_qid2', 'freq_q1+q2', 'freq_q1-q2', 'q1_n_words',
'q2_n_words',
       'len_char_q1', 'len_char_q2', 'len_word_q1', 'len_word_q2',
       'word_Common', 'word_Total', 'word_share', 'cosine_distance',
       'cityblock_distance', 'jaccard_distance', 'canberra_distance',
       'euclidean_distance', 'minkowski_distance',
'braycurtis_distance',
       'fuzz_qratio'],
      dtype='object')
```

Plots understand features extracted

shared words plot and distribution

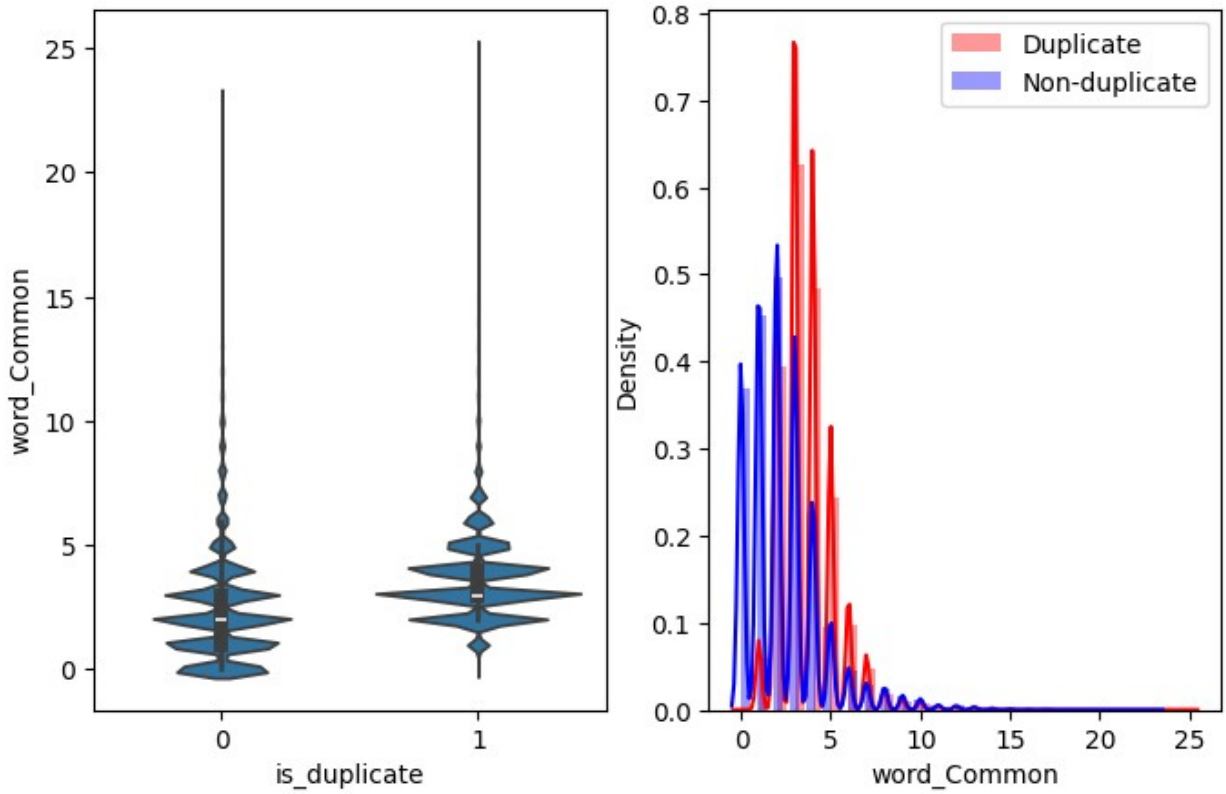
```
plt.figure(figsize=(8, 8))
```

```
plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'word_share', data = df[0:])

plt.subplot(1,2,2)
sns.histplot(df[df['is_duplicate'] == 1.0]['word_share'][0:] , label =
"Duplicate", color = 'red')
sns.histplot(df[df['is_duplicate'] == 0.0]['word_share'][0:] , label =
"Non-duplicate" , color = 'blue' )
plt.legend()
plt.show();
```



common words plot & distribution

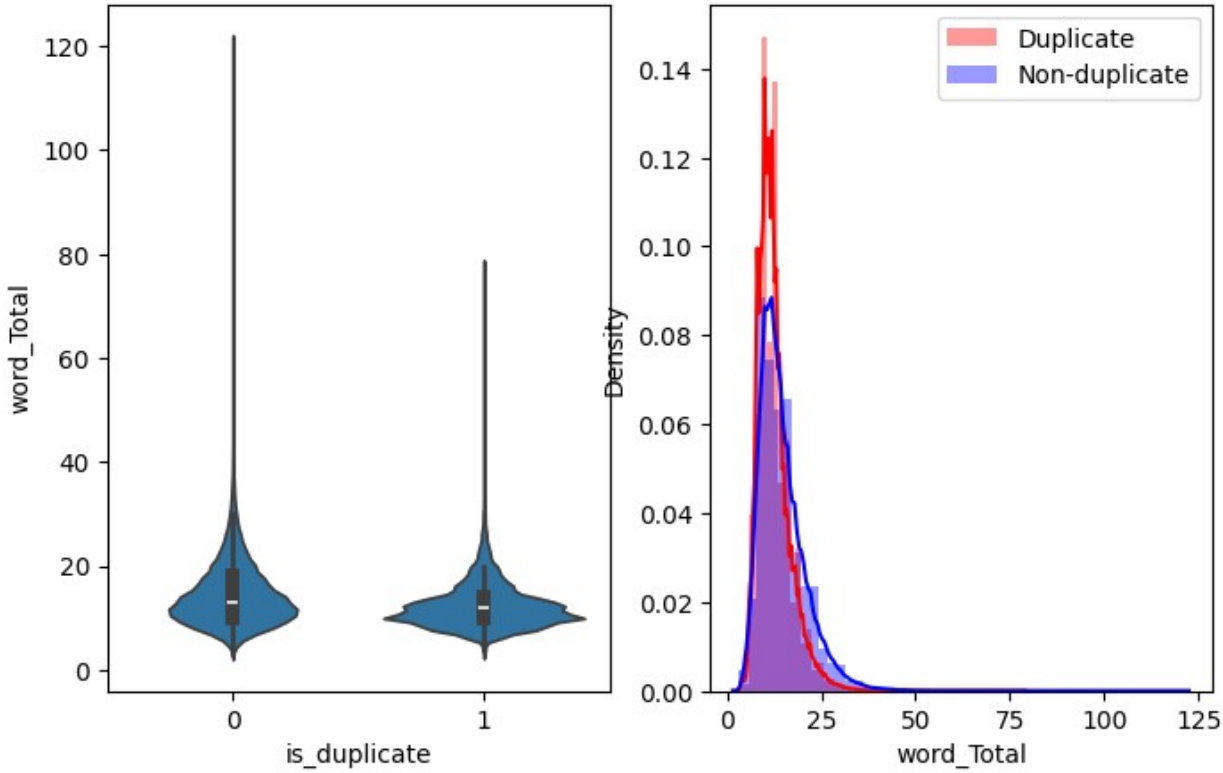
```
plt.figure(figsize=(8,5))

plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'word_Common', data = df[0:])

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['word_Common'][0:] , label
= "Duplicate", color = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['word_Common'][0:] , label
= "Non-duplicate" , color = 'blue' )
plt.legend()
plt.show();
```



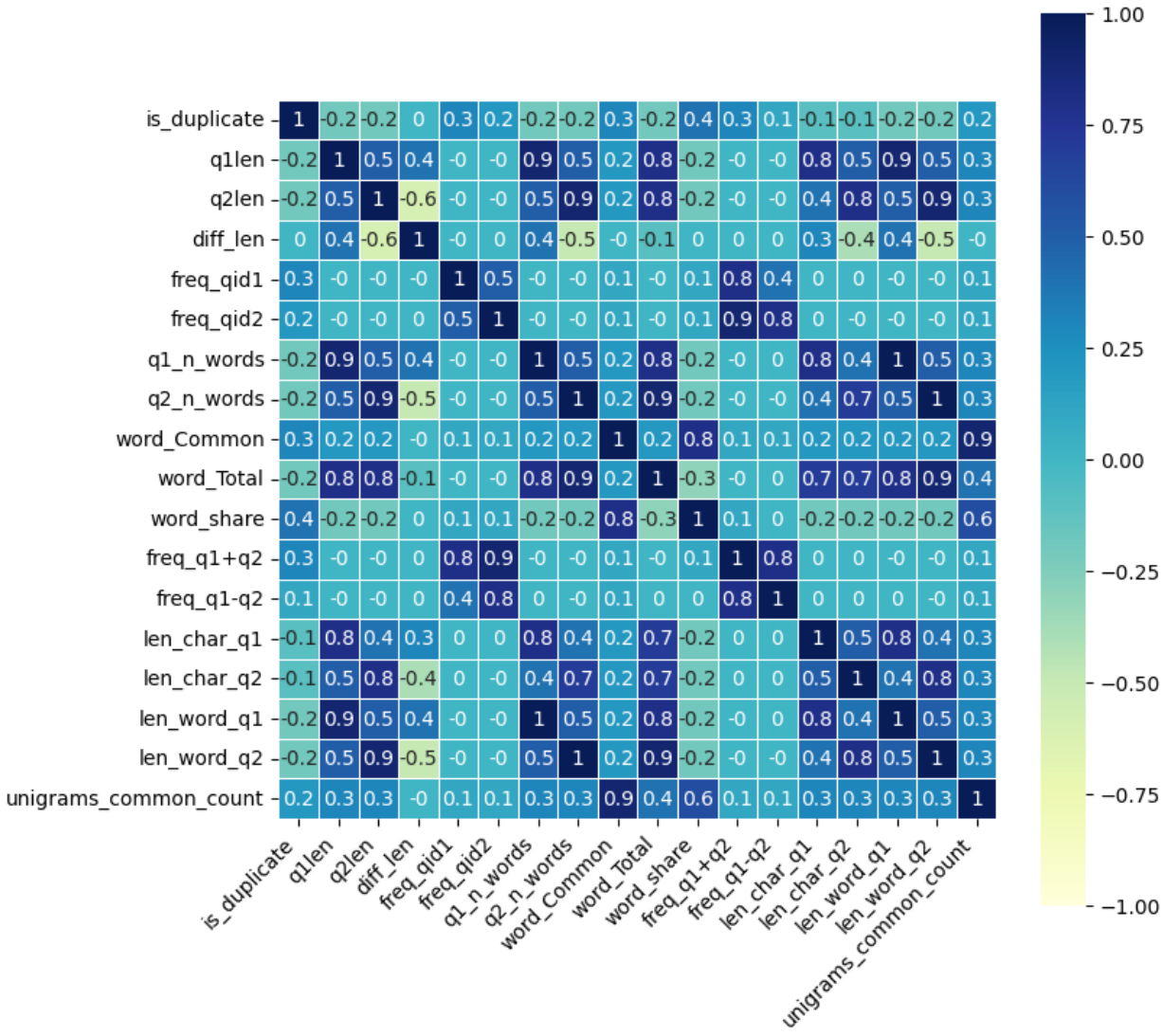
total words plots and distribution

```
plt.figure(figsize=(8, 5))

plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'word_Total', data = df[0:])

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

```
"Non-duplicate" , color = 'blue' )
plt.legend()
plt.show();
```




```
corr_columns = ['is_duplicate', 'q1len', 'q2len', 'diff_len',
'freq_qid1',
                'freq_qid2', 'q1_n_words', 'q2_n_words',
'word_Common', 'word_Total',
                'word_share', 'freq_q1+q2', 'freq_q1-q2',
'len_char_q1', 'len_char_q2', 'len_word_q1', 'len_word_q2',
'unigrams_common_count']
correlation_data = df[corr_columns].corr()
fig, ax = plt.subplots(figsize=(8,8))
ax = sns.heatmap(
    round(correlation_data,1),
    vmin=-1, vmax=1, center=0,annot=True,linewidths=.5,
    cmap="YlGnBu",
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
```



Model Building

```
df_original = df.copy()

# Removing features those are not useful and non-numeric. And also
braycurtis_distance as it has all null values
df.drop(['qid1','qid2','question1','question2', 'unigrams_ques1',
'unigrams_ques2', 'braycurtis_distance'], axis=1, inplace=True)

df.dropna(axis = 0,inplace = True)

y_true = df['is_duplicate']
y_true = list(map(int, y_true.values))
df.drop(['id','is_duplicate'], axis=1, inplace=True)
```

```
cols = list(df.columns)
for i in cols:
    df[i] = df[i].apply(pd.to_numeric)
```

Test-train split

```
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(df, y_true,
stratify=y_true, test_size=0.2)

from collections import Counter

print("-"*10, "Output variable distribution in train data", "*"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",round(int(train_distr[0])*100/train_len,2),"Class 1:
", round(int(train_distr[1])*100/train_len,2))

print("-"*10, "output variable distribution in validation data",
"*"*10)
test_distr = Counter(y_test)
test_len = len(y_test)
print("Class 1: ",round(int(test_distr[0])*100/test_len,2), "Class 1:
",round(int(test_distr[1])*100/test_len,2))
```

```
---------- Output variable distribution in train data **********
Class 0:  63.08 Class 1:  36.92
---------- output variable distribution in validation data **********
Class 1:  63.08 Class 1:  36.92
```

Logistic Regression

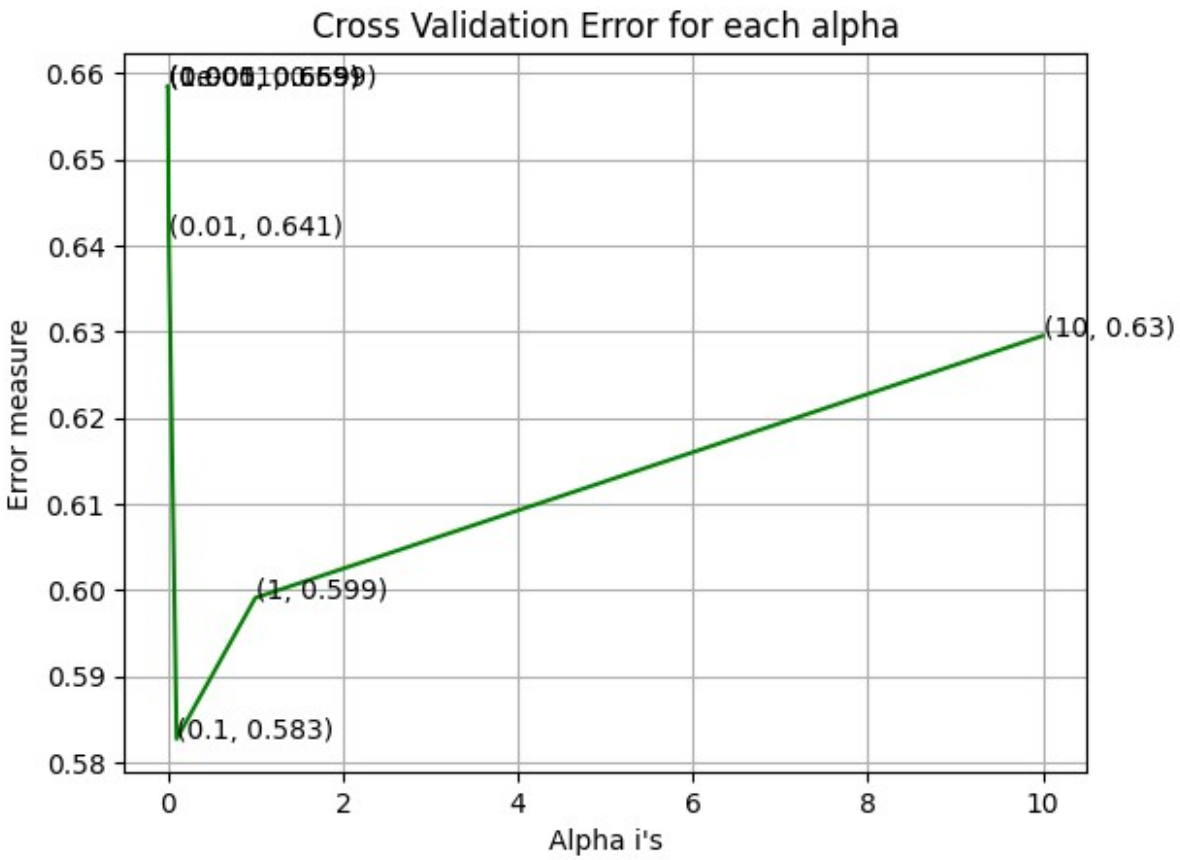
SGD is a optimization method SGDClassifier is a linear classifier with SGD.

Logistic Regression == SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, log_loss

alpha_values = [10 ** x for x in range(-5, 2)]

log_error_values=[]
for i in alpha_values:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log',
random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_values.append(log_loss(y_test, predict_y,
labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss
is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

fig, ax = plt.subplots()
ax.plot(alpha_values, log_error_values,c='g')
for i, txt in enumerate(np.round(log_error_values,3)):
    ax.annotate((alpha_values[i],np.round(txt,3)),
(alpha_values[i],log_error_values[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(log_error_values)
clf = SGDClassifier(alpha=alpha_values[best_alpha], penalty='l2',
loss='log', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)
```

```
For values of alpha =  1e-05 The log loss is: 0.6585300338196388
For values of alpha =  0.0001 The log loss is: 0.6585300338196388
For values of alpha =  0.001 The log loss is: 0.6585300338196388
For values of alpha =  0.01 The log loss is: 0.6412728467929882
For values of alpha =  0.1 The log loss is: 0.5827239969320069
For values of alpha =  1 The log loss is: 0.5990993513923978
For values of alpha =  10 The log loss is: 0.6295331603050178
```




```
CalibratedClassifierCV(estimator=SGDClassifier(alpha=0.1, loss='log',
                                               random_state=42))
```

```
print("Best value of alpha - ", alpha_values[best_alpha])
predict_y = sig_clf.predict_proba(X_train)
print("The train log loss is:",log_loss(y_train, predict_y,
labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print("The test log loss is:",log_loss(y_test, predict_y,
labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
```

```
Best value of alpha -  0.1
The train log loss is: 0.5837994604444074
The test log loss is: 0.5827239969320069
```

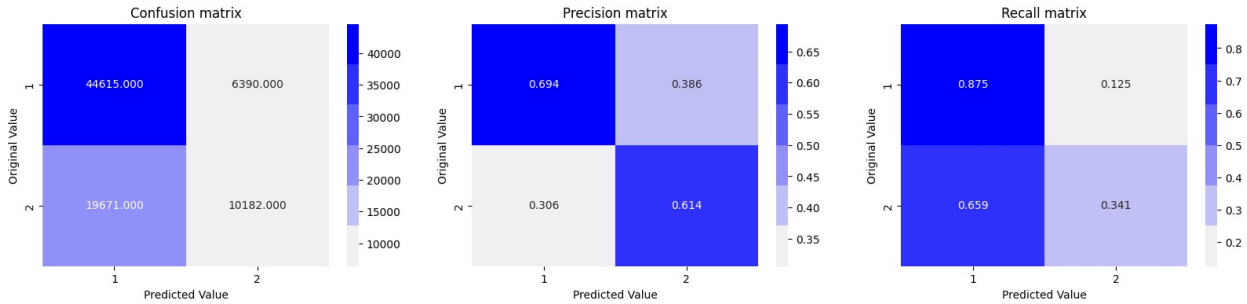
```
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score
print("Accuracy Score - ", accuracy_score(y_test, predicted_y))
```

```
Accuracy Score -  0.6776942293897944
```

```
print("Classification Report")
print(classification_report(y_test, predicted_y))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.69      0.87      0.77     51005
           1       0.61      0.34      0.44     29853

    accuracy                           0.68     80858
   macro avg       0.65      0.61      0.61     80858
weighted avg       0.66      0.68      0.65     80858
```

```
def confusion_matrix_plot(test_values, prediction_values):
    cf_matrix = confusion_matrix(test_values, prediction_values)

    precision_details =(((cf_matrix.T)/(cf_matrix.sum(axis=1))).T)

    recall_details = (cf_matrix/cf_matrix.sum(axis=0))
    plt.figure(figsize=(20,4))

    labels = [1,2]
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(cf_matrix, annot=True, cmap=cmap, fmt=".3f",
xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Value')
    plt.ylabel('Original Value')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(recall_details, annot=True, cmap=cmap, fmt=".3f",
xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Value')
    plt.ylabel('Original Value')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    sns.heatmap(precision_details, annot=True, cmap=cmap, fmt=".3f",
xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Value')
    plt.ylabel('Original Value')
    plt.title("Recall matrix")

    plt.show()

confusion_matrix_plot(y_test, predicted_y)
```



SVM

```
clf = SGDClassifier(alpha=alpha_values[best_alpha], penalty='l1',
loss='hinge', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)
```

```
CalibratedClassifierCV(estimator=SGDClassifier(alpha=0.1,
penalty='l1',
                                               random_state=42))
```

```
predict_y = sig_clf.predict_proba(X_test)
predicted_y =np.argmax(predict_y,axis=1)
print(accuracy_score(y_test, predicted_y))
print(classification_report(y_test, predicted_y))
```

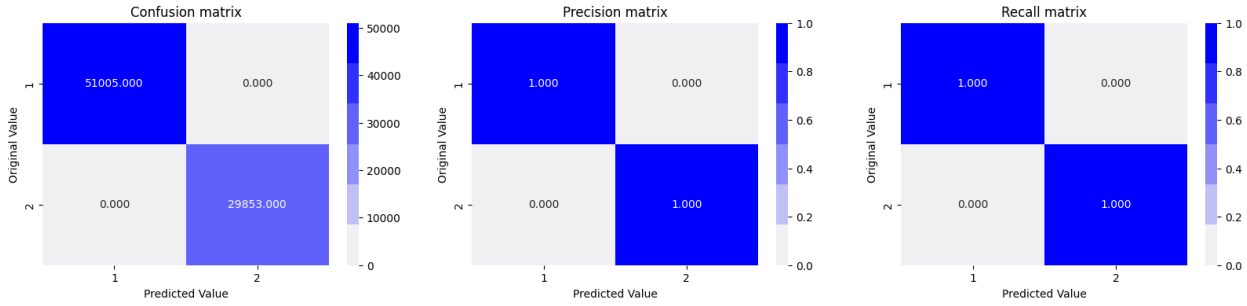
```
0.6307972000296816
              precision    recall  f1-score   support

           0       0.63      1.00      0.77     51005
           1       0.00      0.00      0.00     29853

    accuracy                           0.63     80858
   macro avg       0.32      0.50      0.39     80858
weighted avg       0.40      0.63      0.49     80858
```

XGBoost

```
import xgboost as xgb

clf=xgb.XGBClassifier(n_jobs=-
1,random_state=25,max_depth=5,n_estimators=300)
clf.fit(X_train,y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None,
early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None,
```

```
feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None,
max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan,
monotone_constraints=None,
              multi_strategy=None, n_estimators=300, n_jobs=-1,
              num_parallel_tree=None, random_state=25, ...)

y_pred_test=clf.predict_proba(X_test)
y_pred_train=clf.predict_proba(X_train)
log_loss_train = log_loss(y_train, y_pred_train, eps=1e-15)
log_loss_test=log_loss(y_test,y_pred_test,eps=1e-15)
print('Train log loss = ',log_loss_train,' Test log loss =
',log_loss_test)
predicted_y=np.argmax(y_pred_test,axis=1)
confusion_matrix_plot(y_test,predicted_y)

Train log loss =  6.091918669980021e-06  Test log loss =
6.091016389706224e-06
```




```
print("Classification Report")
print(classification_report(y_test, predicted_y))

Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     51005
           1       1.00      1.00      1.00     29853

    accuracy                           1.00     80858
   macro avg       1.00      1.00      1.00     80858
weighted avg       1.00      1.00      1.00     80858


print("Accuracy Score - ", accuracy_score(y_test, predicted_y))

Accuracy Score -  1.0
```

Seems XGBoosting is overfitting the data

These traditional models are giving moderate results so we can try deep learning models to get better results. Due to resource and time constraints couldn't those.