In [145]:
```
import sys
import pandas as pd
import nltk
from sklearn.metrics import classification_report
#Data Preprocessing
import html
import numpy as np
import re
import warnings
warnings.filterwarnings('ignore')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

In [146]:
```
df = pd.read_csv("processed_reviews_split_surnamesG_minimal.csv")
df.head()
```

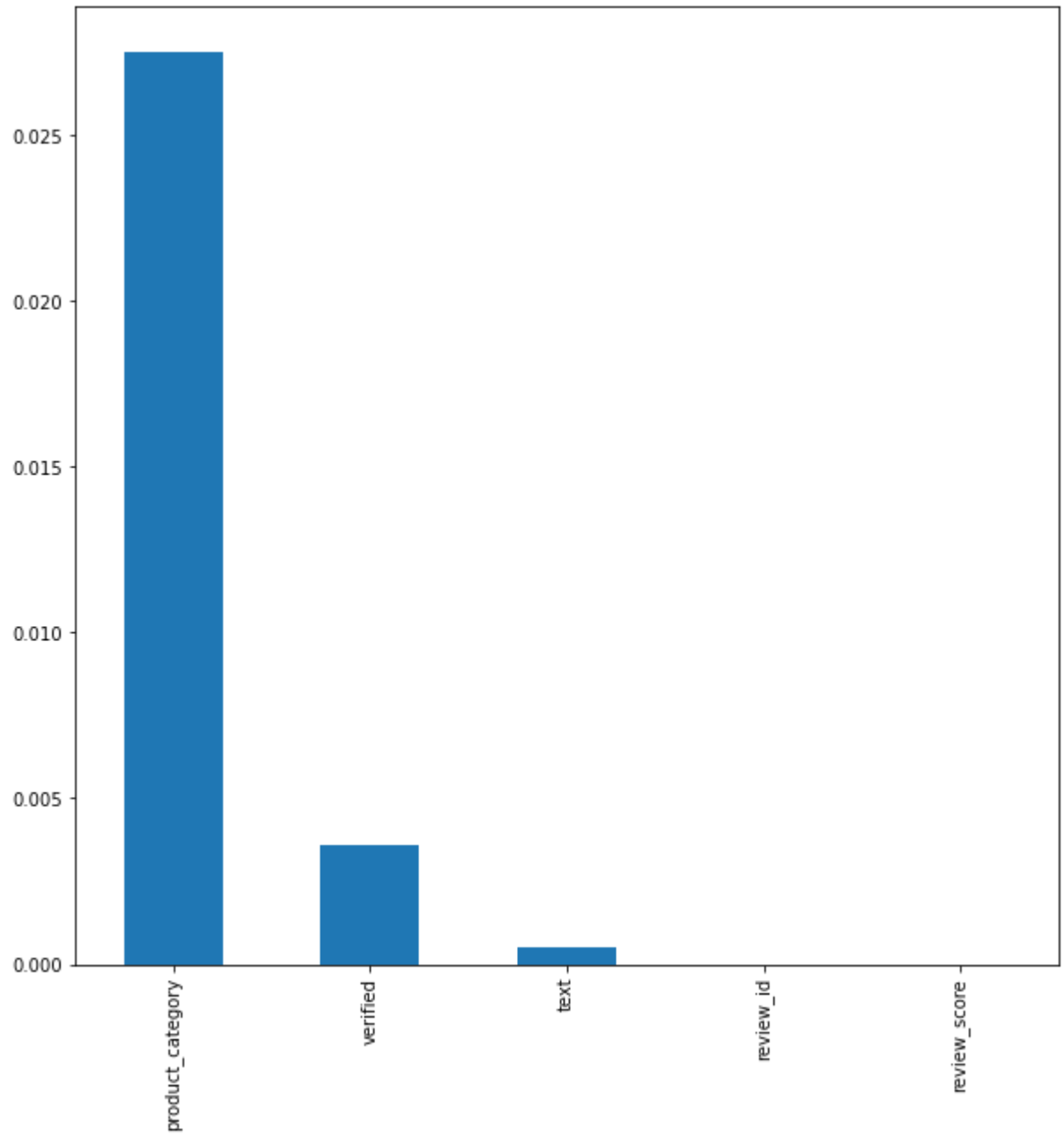
Out[146]:

	review_id	text	verified	review_score	product_category
0	product_review_000000	I had received my copy of this new version of ...	True	3.0	video_games
1	product_review_000001	That ever since i purchased my DS when it was ...	False	5.0	video_games
2	product_review_000002	Simple, and just a bit goes a long way..	True	5.0	musical_instruments
3	product_review_000003	I have not played any of the other games in th...	True	4.0	video_games
4	product_review_000004	i got it quick and it was in great shape works...	True	5.0	video_games

In [147]:
```
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df.columns,
                                 'percent_missing': percent_missing})
```

In [148]:
```
#missing values percentage plot for all the features
import matplotlib.pyplot as plt
Miss_val = df.isna().sum()/df.shape[0]
Miss_val.sort_values(ascending=False, inplace=True)
Miss_val
plt.figure(figsize=(10,10))
Miss_val.plot.bar()
```

Out[148]: <AxesSubplot:>



In [ ]:

In [149]:
```
drp=df[df.isnull().any(axis=1)]
idd=drp['review_id']
```

In [150]:
```
idd=list(idd)
```

In [151]:
```
l=[]
for i in df['review_id']:
    if i in idd:
        exc='1 reason = missing'
        l.append(exc)
    else:
        exc='0 reason = complete rows'
        l.append(exc)
```

In [152]:
```
exclusivedf=df.copy()
exclusivedf['excluded']=l
```

In [153]:
```
exclusivedf.to_csv('exclusivedataset.csv')
```

In [154]:
```
#dropping  missing values rows
df = df.dropna(axis=0)
df=df.reset_index(drop=True)
df
```

Out[154]:

	review_id	text	verified	review_score	product_category
0	product_review_000000	I had received my copy of this new version of ...	True	3.0	video_games
1	product_review_000001	That ever since i purchased my DS when it was ...	False	5.0	video_games
2	product_review_000002	Simple, and just a bit goes a long way..	True	5.0	musical_instruments
3	product_review_000003	I have not played any of the other games in th...	True	4.0	video_games
4	product_review_000004	i got it quick and it was in great shape works...	True	5.0	video_games
...	...	...	...	...	...
31875	product_review_032913	I've been playing Skyrim on the PS3 for about ...	False	-1.0	video_games
31876	product_review_032914	If you are looking for the overall best PGA To...	True	5.0	video_games
31877	product_review_032915	Another figure to add to your Disney Infinity ...	True	5.0	video_games
31878	product_review_032916	Another glad to find! Have looked and looked ...	True	5.0	video_games
31879	product_review_032917	like the game.	True	4.0	video_games

31880 rows × 5 columns

In [ ]:

replacing null values with mode for object type features

Cleaning the data

Let's first clean the data, remove stopwords etc and perform basic pre-processing

Removing weird spaces

In [155]:
```
def remove_spaces(text):
    text=text.strip()
    text=text.split()
    return ' '.join(text)
```

Contraction

In [156]:
```
contraction = {'cause':'because',
               'aint': 'am not',
               'aren\'t': 'are not'}

def mapping_replacer(x,dic):
    for words in dic.keys():
        if ' ' + words + ' ' in x:
            x=x.replace(' '+ words +' ' ,' '+dic[words]+' ' )
    return x
```

Stemming, lemmetisation and tokenisation

In [157]:
```
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem.lancaster import LancasterStemmer

nltk.LancasterStemmer
ls = LancasterStemmer()
lem = WordNetLemmatizer()
def lexicon_normalization(text):
    words = word_tokenize(text)


    # 1- Stemming
    words_stem = [ls.stem(w) for w in words]

    # 2- Lemmatization
    words_lem = [lem.lemmatize(w) for w in words_stem]
    return words_lem
```

Removing links, brackets, numbers, punctuations etc.

In [158]:
```
def clean_text(text):
    '''Make text lowercase, remove text in square brackets,remove links,remove pu
    and remove words containing numbers.'''
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('_', '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub('\'','', text)

    return text
```

Handling stopwords

In [159]:
```
from collections import Counter
def remove_stopword(text):
    stop_words = stopwords.words('english')
    stopwords_dict = Counter(stop_words)
    text = ' '.join([word for word in text.split() if word not in stopwords_dict]
    return text
```

Tokenisation

In [160]:
```
def tokenise(text):
    words = word_tokenize(text)
    return words
```

Applying data cleaning steps to data

Cleaning Regex Expressions from data

In [161]:
```
import re
df['text'] = df['text'].map(lambda x: re.sub(r'\W+', ' ', str(x)))
df['text'] = df['text'].replace(r'\W+', ' ', regex=True)
```

In [162]:
```
df['text']=df['text'].apply(lambda x: mapping_replacer(x, contraction))
```

In [163]:
```
df['text']=df['text'].apply(lambda x:clean_text(x))
```

In [164]:
```
df['text']=df['text'].apply(lambda x: remove_stopword(x))
```

In [165]:
```
df['text']=df['text'].apply(lambda x: lexicon_normalization(x))
```

Finding the most Common words in our Text

Common words for video product category

In [166]:
```
video=df[df['product_category']=='video_games']
```

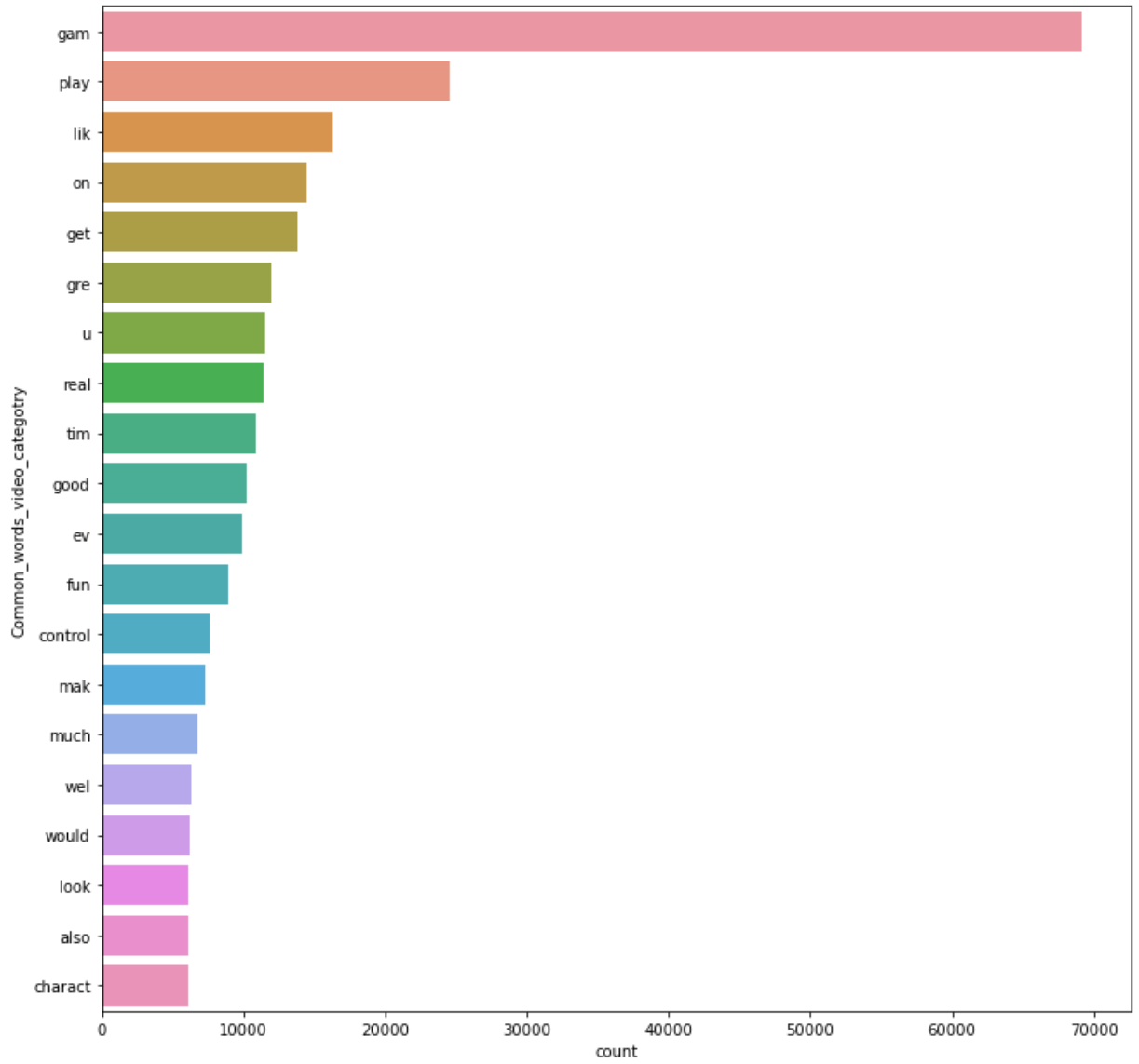
In [177]:
```
top = Counter([item for sublist in video['text'] for item in sublist])
temp = pd.DataFrame(top.most_common(20))
temp.columns = ['Common_words_video_categotry','count']
temp.style.background_gradient(cmap='Blues')
```

Out[177]:

	Common_words_video_categotry	count
0	gam	69209
1	play	24552
2	lik	16348
3	on	14409
4	get	13838
5	gre	11972
6	u	11496
7	real	11458
8	tim	10828
9	good	10195
10	ev	9859
11	fun	8880
12	control	7668
13	mak	7262
14	much	6714
15	wel	6356
16	would	6265
17	look	6146
18	also	6136
19	charact	6122

In [194]:
```
plt.figure(figsize=(12,12))
sns.barplot(temp['count'],temp['Common_words_video_categotry'])
```

Out[194]: <AxesSubplot:xlabel='count', ylabel='Common_words_video_categotry'>



Common words for music product category

In [196]:
```
music=df[df['product_category']=='musical_instruments']
```

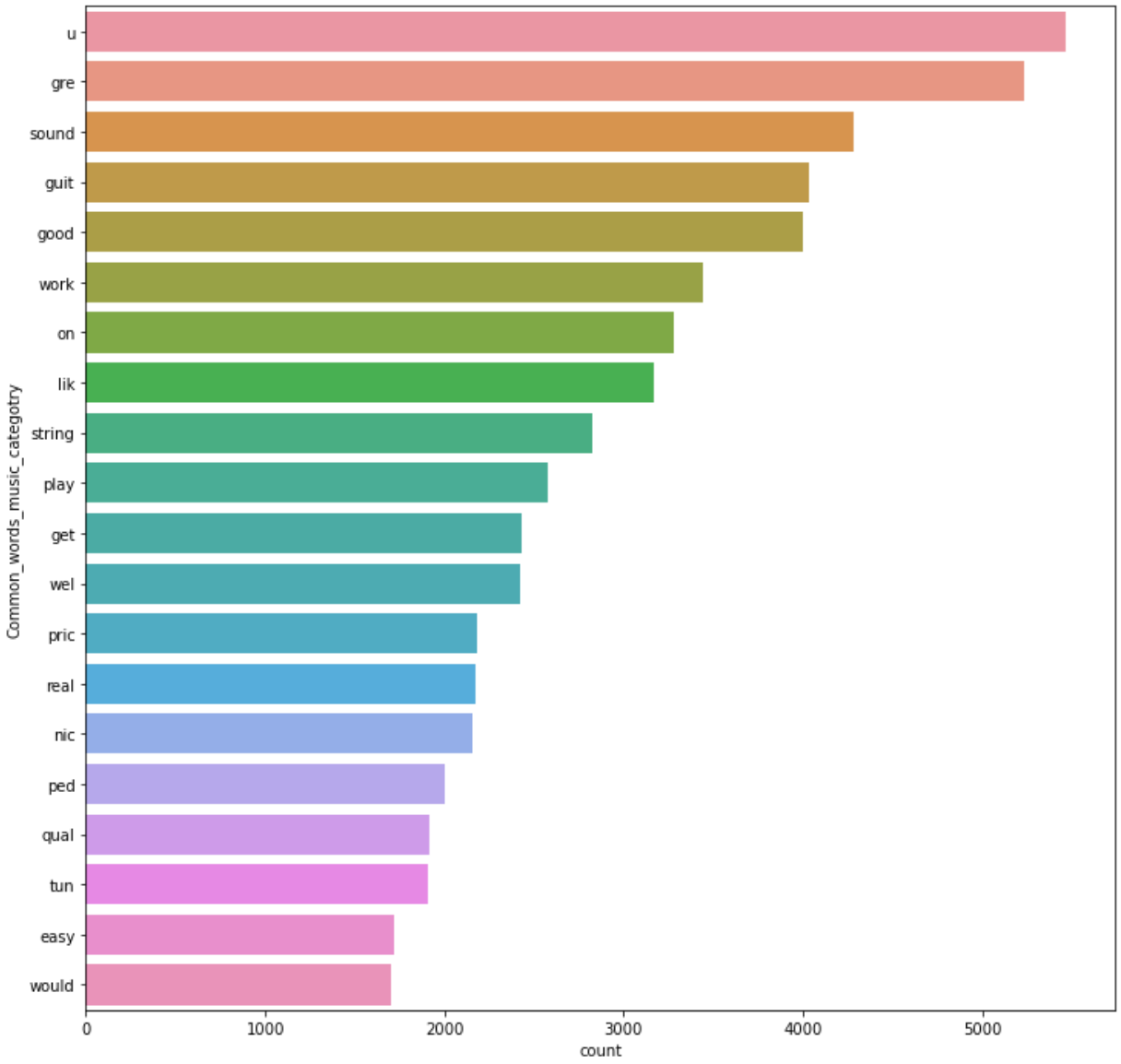
In [197]:
```
top2 = Counter([item for sublist in music['text'] for item in sublist])
temp2 = pd.DataFrame(top2.most_common(20))
temp2.columns = ['Common_words_music_categotry','count']
temp2.style.background_gradient(cmap='Blues')
```

Out[197]:

	Common_words_music_categotry	count
0	u	5467
1	gre	5238
2	sound	4279
3	guit	4037
4	good	3996
5	work	3440
6	on	3278
7	lik	3167
8	string	2825
9	play	2576
10	get	2427
11	wel	2420
12	pric	2180
13	real	2174
14	nic	2153
15	ped	2005
16	qual	1912
17	tun	1904
18	easy	1718
19	would	1703

In [199]:
```
plt.figure(figsize=(12,12))
sns.barplot(temp2['count'],temp2['Common_words_music_categotry'])
```

Out[199]: <AxesSubplot:xlabel='count', ylabel='Common_words_music_categotry'>



In [134]:
```
l=[]
for i in df['text']:
    str1 = " "
    t=str1.join(i)
    l.append(t)
df['text']=l
```

1:Product cagegory prediction

#Converting Tweet feature into a vector using Bag of words technique

In [33]:
```
 #Converting Tweet feature into a vector using Bag of words technique
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['text'])
b=vectorizer.get_feature_names()
```

In [34]:
```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.linear_model import LogisticRegression
```

In [35]:
```
df
```

Out[35]:

	review_id	text	verified	review_score	product_category
0	product_review_000000	receiv cop new vert igt slot gam almost masqu ...	True	3.0	video_games
1	product_review_000001	ev sint purchas d first releas knew would glad...	False	5.0	video_games
2	product_review_000002	simpl bit goe long way	True	5.0	musical_instruments
3	product_review_000003	play gam katamar sery own consol howev quit in...	True	4.0	video_games
4	product_review_000004	got quick gre shap work fin problem lik kind g...	True	5.0	video_games
...	...	...	...	...	...
31875	product_review_032913	play skyrim two year start get let play video ...	False	-1.0	video_games
31876	product_review_032914	look overal best pga tour gam want someth pret...	True	5.0	video_games
31877	product_review_032915	anoth fig ad disney infin collect sup ad disne...	True	5.0	video_games
31878	product_review_032916	anoth glad find look look glad fin find littl ...	True	5.0	video_games
31879	product_review_032917	lik gam	True	4.0	video_games

31880 rows × 5 columns

In [36]:
```
ddf=df.drop(['text','review_id'],axis=1)
```

#creating a dataframe of the array which was converted into a vector and concatinating it with other features

In [37]:
```
 #creating a dataframe of the array which was converted into a vector and concati
df1 = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names())
res = pd.concat([df1, ddf], axis=1)
```

Encoding categorical features

In [38]:
```
a=df.select_dtypes(include='object')
```

In [39]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
df['product_category']=le.fit_transform(df['product_category'])
```

defining inputs and outputs as X and y

respectively

In [40]:
```
X=df1
y=df['product_category']
```

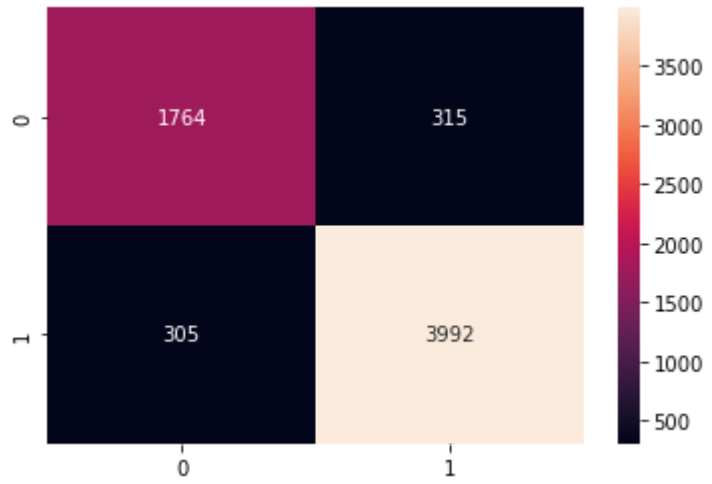
In [41]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20, random_s
```

In [42]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [43]:
```
import seaborn as sns
```

In [44]:
```
from sklearn.metrics import confusion_matrix

dt =DecisionTreeClassifier(random_state=10000)
dt.fit(X_train, y_train)
y_predict_dt = dt.predict(X_test)
y_predict_dt_train = dt.predict(X_train)
# confusion_matrix
cm = confusion_matrix(y_test, y_predict_dt)
sns.heatmap(cm, annot=True, fmt="d")
```

Out[44]: <AxesSubplot:>



In [45]:
```
print('train accuracy',accuracy_score((y_train), y_predict_dt_train))
print('test accuracy',accuracy_score((y_test), y_predict_dt))
print('precision',precision_score(y_test, y_predict_dt, average='macro'))
print('recall',recall_score(y_test, y_predict_dt, average='macro'))
```

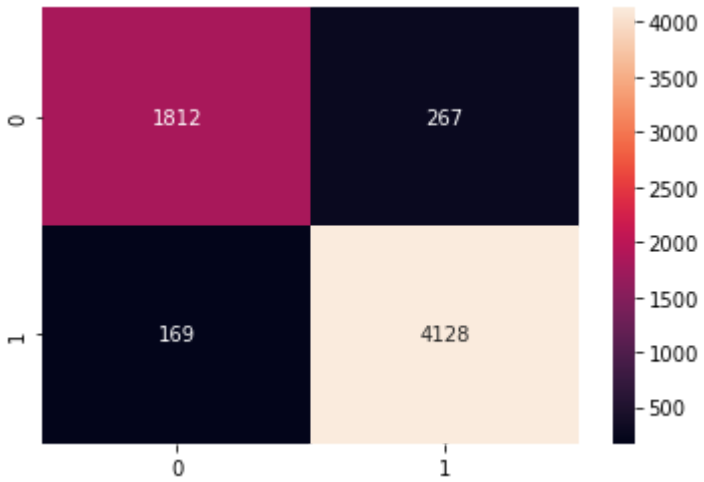
```
train accuracy 0.9955301129234629
test accuracy 0.9027603513174404
precision 0.8897245180578157
recall 0.8887525475842907
```

In [46]:
```
print(classification_report(y_test, y_predict_dt))
```

```
              precision    recall  f1-score   support

           0       0.85      0.85      0.85      2079
           1       0.93      0.93      0.93      4297

    accuracy                           0.90      6376
   macro avg       0.89      0.89      0.89      6376
weighted avg       0.90      0.90      0.90      6376
```

In [47]:
```
from sklearn.ensemble import RandomForestClassifier
rf =RandomForestClassifier(random_state=10000)
rf.fit(X_train, y_train)
y_predict_rf = rf.predict(X_test)
y_predict_rf_train = rf.predict(X_train)
# confusion_matrix
cm = confusion_matrix(y_test, y_predict_rf)
sns.heatmap(cm, annot=True, fmt="d")
```

Out[47]: <AxesSubplot:>



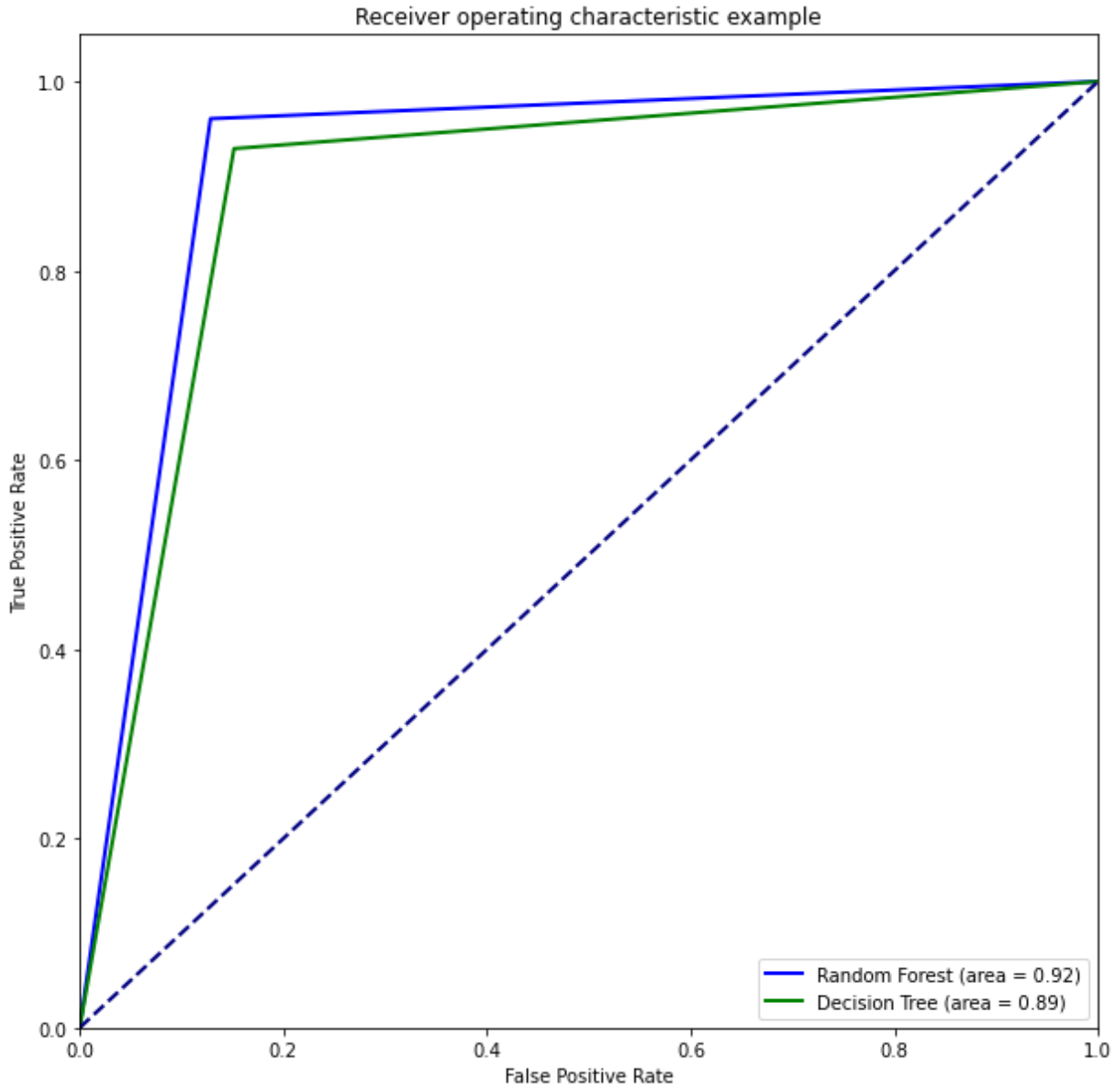
In [48]:
```
print(' train accuracy',accuracy_score((y_train), y_predict_rf_train))
print(' test accuracy',accuracy_score((y_test), y_predict_rf))
print('precision',precision_score(y_test, y_predict_rf))
print('recall',recall_score(y_test, y_predict_rf))
```

```
 train accuracy 0.9955301129234629
 test accuracy 0.9316185696361355
precision 0.9392491467576792
recall 0.9606702350477077
```

In [49]:
```
print(classification_report(y_test, y_predict_rf))
```

```
              precision    recall  f1-score   support

           0       0.91      0.87      0.89      2079
           1       0.94      0.96      0.95      4297

    accuracy                           0.93      6376
   macro avg       0.93      0.92      0.92      6376
weighted avg       0.93      0.93      0.93      6376
```

In [50]:
```
### Compute ROC curve and ROC area for predictions on validation set
from sklearn.metrics import roc_curve, auc

### Plot
plt.figure(figsize=(10,10))
lw = 2

fpr_rf, tpr_rf, _ = roc_curve(y_test, y_predict_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)
plt.plot(fpr_rf, tpr_rf, color='blue',
         lw=lw, label='Random Forest (area = %0.2f)' % roc_auc_rf)
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_predict_dt)
roc_auc_dt = auc(fpr_dt, tpr_dt)
plt.plot(fpr_dt, tpr_dt, color='green',
         lw=lw, label='Decision Tree (area = %0.2f)' % roc_auc_dt)

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```



2:review_score prediction

In [53]:
```
vectorizer2 = CountVectorizer(max_features=2500)
X2 = vectorizer2.fit_transform(df['text'])
b2=vectorizer2.get_feature_names()
```

In [55]:
```
 #creating a dataframe of the array which was converted into a vector and concati
df2 = pd.DataFrame(X2.toarray(), columns=vectorizer2.get_feature_names())
```

In [56]:
```
X2=df2
y2=df[['review_score']]
```

In [58]:
```
from collections import Counter
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
# summarize class distribution
print(Counter(y2))
# define resampling
# transform the dataset
strategy = {5.0:18681, 4.0:10000,3.0:10000,2.0:9000,1.0:8500,-1.0:8000}
over = RandomOverSampler(sampling_strategy=strategy)
# define pipeline
pipeline = Pipeline(steps=[('o', over)])
X2, y2 = pipeline.fit_resample(X2, y2)
# summarize class distribution
print(Counter(y2))
```

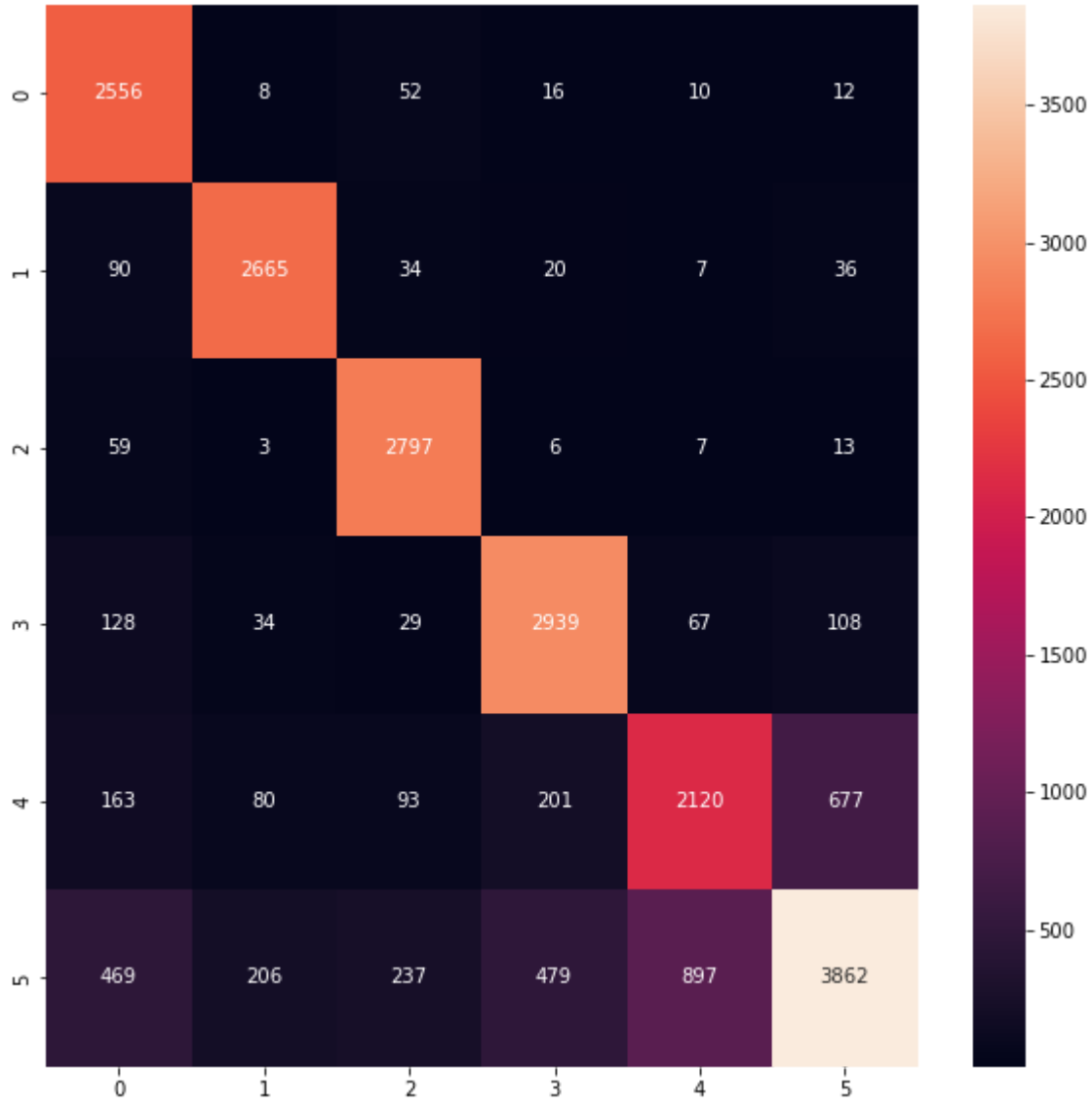
```
Counter({'review_score': 1})
Counter({'review_score': 1})
```

In [59]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X2 = scaler.fit_transform(X2)
```

In [60]:
```
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2,y2, test_size=0.33, ra
```

In [61]:
```
dt2 =DecisionTreeClassifier(random_state=1024)
dt2.fit(X_train2, y_train2)
y_predict_dt2 = dt2.predict(X_test2)
# confusion_matrix
cm = confusion_matrix(y_test2, y_predict_dt2)
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt="d")
```

Out[61]: <AxesSubplot:>

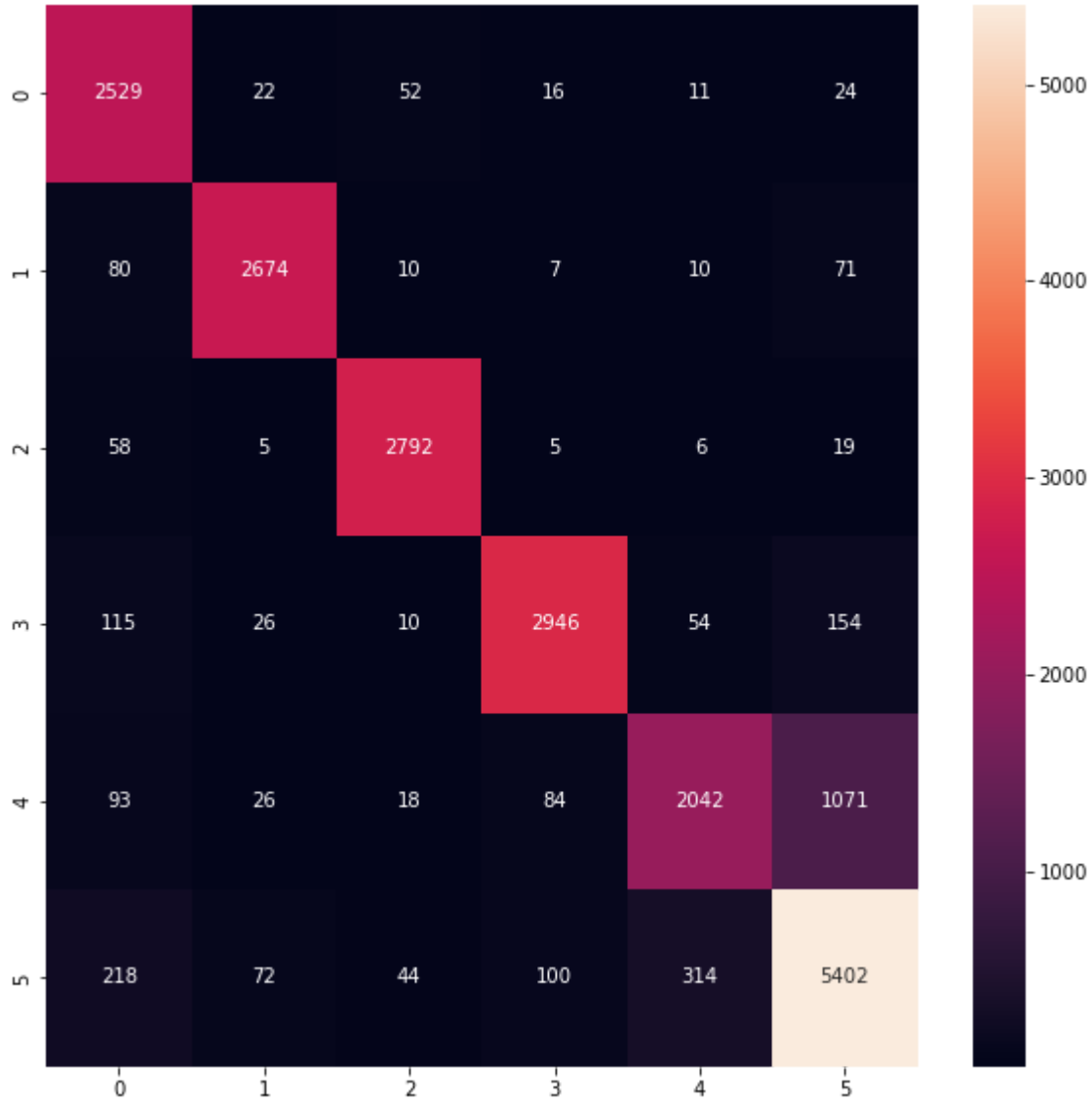


In [62]:
```
print('accuracy',accuracy_score((y_test2), y_predict_dt2))
print('precision',precision_score(y_test2, y_predict_dt2, average='macro'))
print('recall',recall_score(y_test2, y_predict_dt2, average='macro'))
```

```
accuracy 0.7997639282341832
precision 0.7991872358765787
recall 0.8366838310546162
```

In [64]:
```
rf2 =RandomForestClassifier(random_state=1000)
rf2.fit(X_train2, y_train2)
y_predict_rf2 = rf2.predict(X_test2)
# confusion_matrix
cm = confusion_matrix(y_test2, y_predict_rf2)
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt="d")
```

Out[64]: <AxesSubplot:>



In [65]:
```
print('accuracy',accuracy_score((y_test2), y_predict_rf2))
print('precision',precision_score(y_test2, y_predict_rf2, average='macro'))
print('recall',recall_score(y_test2, y_predict_rf2, average='macro'))
```

```
accuracy 0.8680358829084042
precision 0.881759047466947
recall 0.8734135711947347
```

In [ ]:

In [ ]: