In [1]:
```
import pandas as pd
import numpy as np
from  sklearn  import tree
from sklearn import preprocessing
from sklearn import tree
```

In [2]:
```
df = pd.read_csv('...........\\drugsCom_raw\\Process.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Unnamed: 0	Id	rating	ratingSentiment	ratingSentimentLabel	vaderReviewScore	vaderSer
0	0	206461	9	2	positive	0.0000	
1	1	95260	8	2	positive	0.9070	
2	2	92703	5	0	neutral	0.7096	
3	3	138000	8	2	positive	0.7184	
4	4	35696	9	2	positive	0.9403	

In [4]:
```
data=df.iloc[:,:-1]
data
```

Out[4]:

	Unnamed: 0	Id	rating	ratingSentiment	ratingSentimentLabel	vaderReviewScore	vac
0	0	206461	9	2	positive	0.0000	
1	1	95260	8	2	positive	0.9070	
2	2	92703	5	0	neutral	0.7096	
3	3	138000	8	2	positive	0.7184	
4	4	35696	9	2	positive	0.9403	
...	...	...	...	...	...	...	...
215058	53761	159999	10	2	positive	-0.8924	
215059	53762	140714	9	2	positive	0.9223	
215060	53763	130945	8	2	positive	-0.8471	
215061	53764	47656	1	1	negative	-0.8175	
215062	53765	113712	9	2	positive	0.0000	

215063 rows × 7 columns

In [5]:
```
data_df=data.drop(['Unnamed: 0'],axis=1)
```

In [6]:
```
data_df.head()
```

Out[6]:

	Id	rating	ratingSentiment	ratingSentimentLabel	vaderReviewScore	vaderSentiment
0	206461	9	2	positive	0.0000	0
1	95260	8	2	positive	0.9070	2
2	92703	5	0	neutral	0.7096	2
3	138000	8	2	positive	0.7184	2
4	35696	9	2	positive	0.9403	2

In [7]:
```
x0 = data_df.iloc[ : , 0 :-1]
x0
```

Out[7]:

	Id	rating	ratingSentiment	ratingSentimentLabel	vaderReviewScore
0	206461	9	2	positive	0.0000
1	95260	8	2	positive	0.9070
2	92703	5	0	neutral	0.7096
3	138000	8	2	positive	0.7184
4	35696	9	2	positive	0.9403
...	...	...	...	...	...
215058	159999	10	2	positive	-0.8924
215059	140714	9	2	positive	0.9223
215060	130945	8	2	positive	-0.8471
215061	47656	1	1	negative	-0.8175
215062	113712	9	2	positive	0.0000

215063 rows × 5 columns

In [8]:
```
y0 = df['vaderTarget']
y0
```

Out[8]:
```
0          neutral
1         positive
2         positive
3         positive
4         positive
             ...
215058    negative
215059    positive
215060    negative
215061    negative
215062     neutral
Name: vaderTarget, Length: 215063, dtype: object
```

In [9]:
```
l1 = preprocessing.LabelEncoder()
l1.fit(['neutral', 'positive','negative'])
```

Out[9]:
```
LabelEncoder()
```

In [10]:
```
x = x0
x.iloc[:,3] = l1.transform(x0.iloc[:,3])
x0
```

Out[10]:

	Id	rating	ratingSentiment	ratingSentimentLabel	vaderReviewScore
0	206461	9	2	2	0.0000
1	95260	8	2	2	0.9070
2	92703	5	0	1	0.7096
3	138000	8	2	2	0.7184
4	35696	9	2	2	0.9403
...	...	...	...	...	...
215058	159999	10	2	2	-0.8924
215059	140714	9	2	2	0.9223
215060	130945	8	2	2	-0.8471
215061	47656	1	1	0	-0.8175
215062	113712	9	2	2	0.0000

215063 rows × 5 columns

In [11]:
```
l0 = preprocessing.LabelEncoder()
l0.fit(['neutral', 'positive','negative'])
y= l0.transform(y0)
```

In [12]:
```
y
```

Out[12]:
```
array([1, 2, 2, ..., 0, 0, 1])
```

In [13]:
```
x0
```

Out[13]:

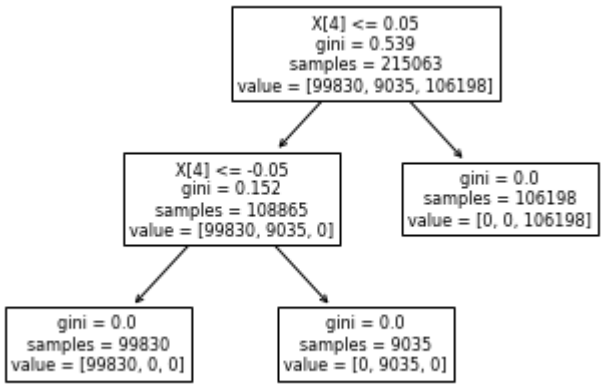
	Id	rating	ratingSentiment	ratingSentimentLabel	vaderReviewScore
0	206461	9	2	2	0.0000
1	95260	8	2	2	0.9070
2	92703	5	0	1	0.7096
3	138000	8	2	2	0.7184
4	35696	9	2	2	0.9403
...	...	...	...	...	...
215058	159999	10	2	2	-0.8924
215059	140714	9	2	2	0.9223
215060	130945	8	2	2	-0.8471
215061	47656	1	1	0	-0.8175
215062	113712	9	2	2	0.0000

215063 rows × 5 columns

In [14]:
```
clf = tree.DecisionTreeClassifier()
```

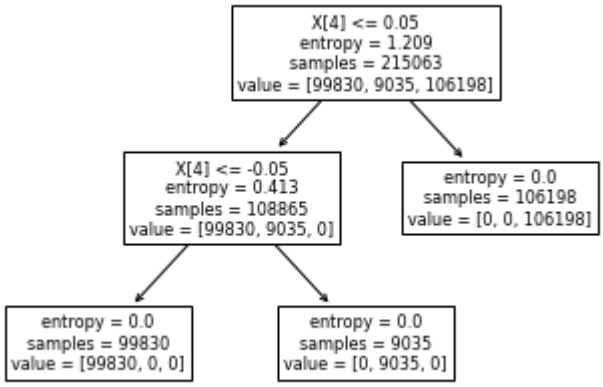
In [15]:
```
clf_fit =clf.fit(x,y)
tree.plot_tree(clf_fit)
```

Out[15]:
```
[Text(200.88000000000002, 181.2, 'X[4] <= 0.05\ngini = 0.539\nsamples = 21506
3\nvalue = [99830, 9035, 106198]'),
 Text(133.92000000000002, 108.72, 'X[4] <= -0.05\ngini = 0.152\nsamples = 108
865\nvalue = [99830, 9035, 0]'),
 Text(66.96000000000001, 36.23999999999998, 'gini = 0.0\nsamples = 99830\nval
ue = [99830, 0, 0]'),
 Text(200.88000000000002, 36.23999999999998, 'gini = 0.0\nsamples = 9035\nval
ue = [0, 9035, 0]'),
 Text(267.84000000000003, 108.72, 'gini = 0.0\nsamples = 106198\nvalue = [0,
0, 106198]')]
```



In [16]:
```
clf = tree.DecisionTreeClassifier(criterion='entropy')
clf_fit =clf.fit(x,y)
tree.plot_tree(clf_fit)
```

Out[16]:
```
[Text(200.88000000000002, 181.2, 'X[4] <= 0.05\nentropy = 1.209\nsamples = 21
5063\nvalue = [99830, 9035, 106198]'),
 Text(133.92000000000002, 108.72, 'X[4] <= -0.05\nentropy = 0.413\nsamples =
108865\nvalue = [99830, 9035, 0]'),
 Text(66.96000000000001, 36.23999999999998, 'entropy = 0.0\nsamples = 99830\n
value = [99830, 0, 0]'),
 Text(200.88000000000002, 36.23999999999998, 'entropy = 0.0\nsamples = 9035\n
value = [0, 9035, 0]'),
 Text(267.84000000000003, 108.72, 'entropy = 0.0\nsamples = 106198\nvalue =
[0, 0, 106198]')]
```



In [ ]:
```
```

In [17]:
```
data_bayes=x0
```

==========Bayse

In [18]:

```
import nltk
import nltk.classify.util
from nltk.classify import NaiveBayesClassifier
import numpy
data_bayes = numpy.random.rand(100, 5)
numpy.random.shuffle(data_bayes)
training, test = data_bayes[:80,:], data_bayes[80:,:]
print(training,test)

classifier = nltk.NaiveBayesClassifier.train(training)

print("Naive Bayes Algo Accuracy:", (nltk.classify.accuracy(classifier,test))*
100)
```

```
[[0.1987314  0.42732063 0.017934   0.48355671 0.88976405]
 [0.95625451 0.83869059 0.58444718 0.483578   0.60848297]
 [0.93290672 0.41525814 0.04435639 0.31632065 0.28320288]
 [0.0079684  0.13635072 0.4119271  0.68644579 0.29982104]
 [0.71688913 0.85154104 0.33101245 0.43789394 0.71421863]
 [0.80600855 0.94204977 0.47519367 0.70235195 0.71647505]
 [0.59324282 0.24600582 0.48833661 0.77905403 0.45290581]
 [0.86901787 0.88650532 0.29820666 0.15383178 0.10102257]
 [0.71105297 0.90547558 0.25856708 0.97045145 0.36643718]
 [0.44674251 0.28574926 0.44007382 0.63242112 0.8943617 ]
 [0.10764886 0.43782926 0.06106357 0.79996478 0.25417637]
 [0.26973525 0.35633953 0.15239139 0.42699778 0.18289258]
 [0.38539125 0.76629529 0.92904816 0.39433936 0.43297949]
 [0.15077982 0.75152652 0.42934695 0.43757632 0.65372024]
 [0.19386324 0.93873607 0.36949066 0.34578056 0.00973919]
 [0.43506068 0.28552208 0.43112344 0.75920094 0.97395591]
 [0.20544947 0.02975253 0.49278616 0.61192342 0.79600785]
 [0.68103656 0.55577749 0.48543763 0.6894822  0.00611269]
 [0.97610139 0.6132361  0.66597078 0.47175556 0.76519224]
 [0.78846559 0.4925709  0.42624058 0.71795348 0.20838012]
 [0.73896757 0.20566151 0.77188587 0.9931629  0.61638495]
 [0.77034644 0.92232831 0.38280191 0.69545778 0.24006618]
 [0.74530033 0.97125701 0.28156835 0.46299507 0.89771024]
 [0.71480134 0.19416242 0.67969427 0.7520433  0.83523939]
 [0.41454664 0.37947255 0.20893891 0.57061552 0.43108823]
 [0.35271621 0.57640987 0.49916637 0.13927789 0.49132826]
 [0.64062622 0.99947021 0.33778034 0.24268674 0.82657957]
 [0.20892474 0.70818529 0.17372495 0.13586797 0.82127536]
 [0.42746927 0.12836126 0.29179508 0.33471331 0.90504211]
 [0.98102266 0.41711403 0.47263982 0.25655951 0.81702587]
 [0.04470089 0.19932754 0.54040771 0.17948521 0.32250253]
 [0.68778269 0.78512757 0.6014559  0.64841307 0.14917062]
 [0.16330046 0.15454635 0.64550989 0.57904109 0.1699142 ]
 [0.83662088 0.1654576  0.37405428 0.83151545 0.80832527]
 [0.47038844 0.58787664 0.37142521 0.63742645 0.81167772]
 [0.41753171 0.03074204 0.14777788 0.58265328 0.68864474]
 [0.7417414  0.09497896 0.79578047 0.06988502 0.77655853]
 [0.0380863  0.80396203 0.6790935  0.97404049 0.82722785]
 [0.60158368 0.8563304  0.47550157 0.53672904 0.10236156]
 [0.18991074 0.87732236 0.14691153 0.73138078 0.0119456 ]
 [0.73304646 0.03456975 0.24254736 0.07470384 0.26855602]
 [0.92385712 0.43839623 0.52635943 0.71729463 0.59898008]
 [0.06458471 0.47635871 0.32584605 0.73391254 0.14140035]
 [0.39992422 0.96281869 0.64030949 0.36352585 0.08868739]
 [0.11163672 0.71920788 0.4249832  0.28750093 0.81994071]
 [0.55160994 0.90399881 0.98256402 0.45332239 0.30551598]
 [0.3927967  0.30586902 0.78648651 0.40748515 0.60854771]
 [0.71398244 0.27440222 0.50615823 0.63932478 0.28651849]
 [0.5466323  0.61937368 0.05381277 0.87255218 0.86407465]
 [0.62264966 0.50145709 0.75318069 0.3954933  0.86735189]
 [0.77319697 0.3906041  0.63662098 0.76133168 0.59737839]
 [0.14498935 0.71985726 0.49377174 0.90843815 0.83733614]
 [0.46192079 0.84259087 0.35033483 0.37919365 0.80939513]
 [0.54774695 0.90427238 0.71353022 0.55586177 0.75281321]
 [0.49775839 0.43593283 0.03783536 0.75717415 0.53529124]
 [0.43917625 0.45209916 0.79908777 0.47237572 0.14399208]
 [0.05764024 0.72289893 0.13409123 0.14943555 0.6384656 ]
```

```
 [0.03007382 0.61362324 0.27830348 0.21569088 0.64996995]
 [0.64284805 0.23982988 0.90466956 0.96670754 0.45153251]
 [0.88422624 0.84213773 0.46770018 0.62068862 0.66467454]
 [0.20376109 0.60127081 0.55130106 0.80472192 0.74761018]
 [0.6651257  0.59195701 0.17591359 0.82832703 0.34214189]
 [0.8465664  0.68714652 0.53589154 0.08100845 0.82581404]
 [0.11465939 0.29785085 0.20142301 0.89209125 0.53880633]
 [0.32606412 0.75908311 0.68168815 0.52973848 0.98624572]
 [0.86847037 0.21487867 0.31347089 0.01404097 0.28500295]
 [0.44108529 0.18550331 0.46705039 0.76341985 0.25044702]
 [0.09761508 0.0765833  0.37004113 0.75109821 0.42130316]
 [0.74138401 0.2809752  0.32469163 0.01357106 0.52142124]
 [0.53480228 0.78858818 0.58579898 0.18588654 0.30510707]
 [0.04930724 0.45179031 0.47923661 0.36188637 0.37938368]
 [0.10801993 0.01873972 0.60705863 0.45467734 0.74342215]
 [0.72205252 0.30758791 0.11506704 0.23928553 0.61878128]
 [0.07877932 0.30351881 0.04021957 0.45414869 0.22796198]
 [0.82576791 0.2084232  0.80238714 0.88172441 0.3765393 ]
 [0.03422453 0.34370821 0.01472601 0.46969362 0.83928303]
 [0.88636258 0.65464099 0.02456791 0.34253764 0.29669475]
 [0.21933173 0.12407516 0.12691338 0.15616086 0.31167972]
 [0.56666419 0.65096238 0.58705586 0.58189422 0.82715386]
 [0.929805   0.17569903 0.58909125 0.69127787 0.96481886]] [[0.35129893 0.160
91783 0.23983071 0.2326217  0.30208545]
 [0.6519387  0.77115595 0.35982234 0.0419286  0.51237224]
 [0.69723223 0.93344665 0.94506643 0.20873452 0.54632045]
 [0.55511262 0.85029227 0.18046981 0.21736623 0.71462834]
 [0.2588538  0.24123908 0.33425884 0.44829054 0.68236612]
 [0.89149955 0.85126769 0.00742118 0.97800619 0.5451166 ]
 [0.74244276 0.17043572 0.69638212 0.10313888 0.65061195]
 [0.99225125 0.74299373 0.6575256  0.02642428 0.03968807]
 [0.96462928 0.75629185 0.60755774 0.15124636 0.17134491]
 [0.7936344  0.62879122 0.32759768 0.68113727 0.16267059]
 [0.46583548 0.66927051 0.62513801 0.85126738 0.41581308]
 [0.52853113 0.39561652 0.63193773 0.18483001 0.12217484]
 [0.26928101 0.78925504 0.91113607 0.71868629 0.8288641 ]
 [0.64145647 0.80593769 0.43957713 0.33648589 0.31928644]
 [0.36491374 0.81039943 0.7820562  0.53464259 0.50075916]
 [0.86327357 0.67323135 0.88035101 0.2665163  0.07812713]
 [0.38345766 0.38497878 0.02569732 0.74713833 0.19876888]
 [0.31081514 0.41687017 0.40863755 0.46916287 0.26848491]
 [0.04589767 0.49453076 0.47903392 0.24079867 0.40830959]
 [0.76702358 0.4709328  0.19107987 0.10501978 0.68284603]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-18-bd71c789f1e2> in <module>
      8 print(training,test)
      9
---> 10 classifier = nltk.NaiveBayesClassifier.train(training)
     11
     12 print("Naive Bayes Algo Accuracy:", (nltk.classify.accuracy(classifie
r,test))*100)

C:\ProgramData\Anaconda3\lib\site-packages\nltk\classify\naivebayes.py in tra
in(cls, labeled_featuresets, estimator)
    205         # Count up how many times each feature value occurred, given
    206         # the label and featurename.
--> 207         for featureset, label in labeled_featuresets:
    208             label_freqdist[label] += 1
    209             for fname, fval in featureset.items():

ValueError: too many values to unpack (expected 2)
```

In [ ]:
```
#============================
```

==========KNN

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
```

In [ ]:
```
dataset=x0
print(len(dataset))
dataset.head(22)
```

In [ ]:
```
zero_not_accepted =['Id','rating','ratingSentiment','ratingSentimentLabel','va
derReviewScore']
```

In [ ]:
```
for column in zero_not_accepted:
    dataset[column]=dataset[column].replace(0,np.NaN)
    mean = int(dataset[column].mean(skipna=True))
    dataset[column] = dataset[column].replace(np.NAN,mean)
```

In [ ]:
```
print(dataset['Id'])
```

In [ ]:
```
x=dataset.iloc[:,0:5]
y=dataset.iloc[:, 4]
x_train , x_test , y_train , y_test =train_test_split(x,y,random_state=0 ,test
_size=0.2)
```

In [ ]:
```
sc_x = StandardScaler()
x_train = sc_x.fit_transform(x_train)
x_test = sc_x.fit_transform(x_test)
```

In [ ]:
```
x_train
```

In [ ]:
```
x_test
```

In [ ]:
```
y_train
```

In [ ]:
```
y_test
```

In [ ]:
```
import math
```

In [ ]:
```
math.sqrt(len(y_train))
```

In [ ]:
```
math.sqrt(len(y_test))
```

In [ ]:
```
#classifier = KNeighborsClassifier(n_neighbors=13,p=2,metric='euclidean')
#classifier.fit(x_train,y_train)
#cm = confusion_matrix(y_test , y_pred)
#print(cm)
```

==========MLP

In [ ]:
```
from sklearn.neural_network import MLPClassifier

from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split


X, y = make_classification(n_samples=100, random_state=1)


X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,random_st
ate=1)


clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)

clf.predict_proba(X_test[:1])

clf.predict(X_test[:5, :])


clf.score(X_test, y_test)
```

==========

In [ ]:
```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor

# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import r2_score
```

In [ ]:
```
print(x0.shape)
x0.describe().transpose()
```

In [ ]:
```
x0
```

In [ ]:
```
target_column = ['vaderReviewScore']
predictors = list(set(list(x0.columns))-set(target_column))
x0[predictors] = x0[predictors]/x0[predictors].max()
x0.describe().transpose()
```

In [ ]:
```
X = x0[predictors].values
y = x0[target_column].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, rand
om_state=40)
print(X_train.shape); print(X_test.shape)
```

In [ ]:
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(8,8,8), activation='relu', solver='ada
m', max_iter=500)
mlp.fit(X_train,y_train)

predict_train = mlp.predict(X_train)
predict_test = mlp.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_train,predict_train))
print(classification_report(y_train,predict_train))
```

==========Logistic Regression

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

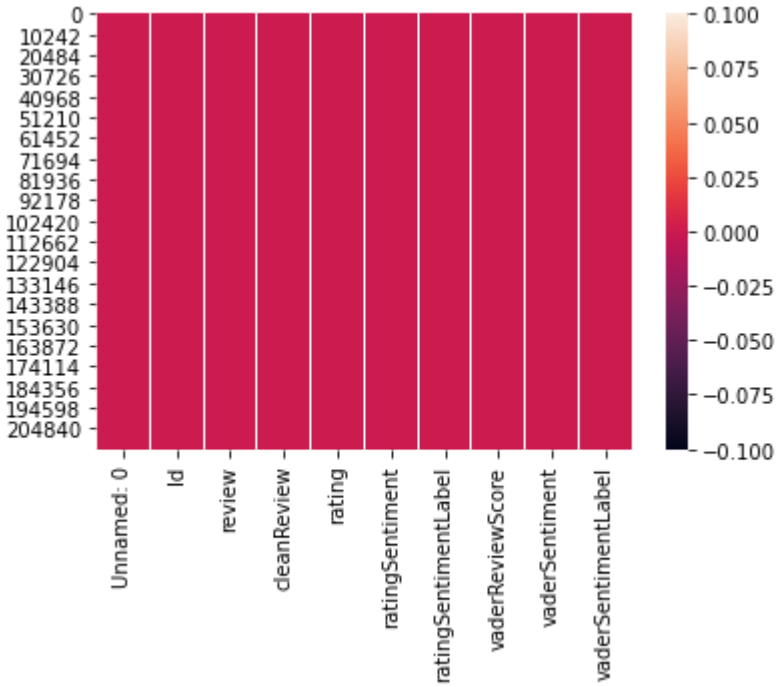
In [ ]:
```
train = pd.read_csv('..............\\Proj_paython_drug\\processed.csv.gz')

train.head()
```

In [ ]:
```
train.isnull()
```

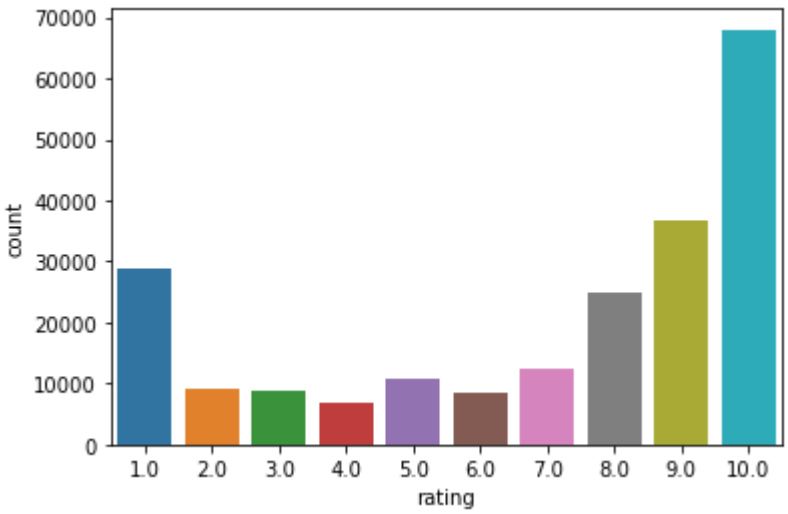
In [5]:
```
sns.heatmap(train.isnull())
```

Out[5]: <AxesSubplot:>



In [6]:
```
sns.countplot(x='rating',data=train)
```

Out[6]: <AxesSubplot:xlabel='rating', ylabel='count'>



In [ ]:
```
sns.countplot(x='review',hue='rating',data=train)
```

In [ ]:
```
plt.figure(figsize=(12, 7))
sns.boxplot(x='rating',y='ratingSentimentLabel',data=train,palette='winter')
```

In [ ]:
```
def impute_age(cols):
    Id = cols[0]
    rating = cols[1]
    if pd.isnull(Id):

        if rating == 1:
            return 37
        elif rating == 2:
            return 29
        else:
            return 24
    else:
        return Id
```

In [ ]:
```
train['Id'] = train[['Id','rating']].apply(impute_age,axis=1)
```

In [ ]:
```
#sns.heatmap(train.isnull(),yticklabels=False,cbar=False)
```

In [ ]:
```
train.drop('ratingSentiment',axis=1,inplace=True)
```

In [ ]:
```
train.info()
```

In [ ]:
```
train
```

In [ ]:
```
#review = pd.get_dummies(train['review'],drop_first=True)
#cleanReview = pd.get_dummies(train['cleanReview'],drop_first=True)
```

In [ ]:
```
train.drop(['Id','review','cleanReview'],axis=1,inplace=True)
```

In [ ]:
```
#train = pd.concat([train,sex,embark],axis=1)
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train.drop('vaderSentimen
t',axis=1), train['vaderSentiment'], test_size=0.30, random_state=101)
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
predictions = logmodel.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```