In [2]:

```
#cp322 project
#march 24,2022

#imports
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier # KNN
from sklearn.naive_bayes import GaussianNB # Gaussian Naive Bayes

import pylab as pl
import matplotlib.pyplot as plt

from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report, confusion_matrix


from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer #root words#Lemmatization rec
lem = WordNetLemmatizer()
```

In [3]:

```
#read files #split text into good or bad meaning

def readFile(file_name):#read files for it
        file = open(file_name, "r")
        words = file.read().splitlines()
        file.close()
        return words


badWords= readFile("bad.txt")
print (badWords)

goodWords= readFile("good.txt")
print (goodWords)

#neutralWords = ["nice", "comfortable", "cozy","elegant", "impress","like","a
expressionWords =["!","super", "lot","too","very","really"] #eg !,too: #if th
negateWords = ["no","not","don't","won't","but","didn't"] #"not" - check if n
```

```
['bad', 'deceive', 'flaw', 'disappoint', 'bore', 'huge', 'large', 'bi
g', 'poor', 'awful', 'terrible', 'delicate', 'scratch', 'itch', 'itch
y', 'expensive', 'sad', 'ugly', 'aggravate', 'anger', 'annoy', 'awful',
'awkward', 'bother', 'break', 'bug', 'burden', 'challenge', 'chaos', 'c
omplain', 'concen', 'conflict', 'confuse', 'con', 'costly', 'crazy', 'd
egrade', 'deprive', 'desperate', 'deteriorate', 'detest', 'Hate', 'disa
gree', 'disapprove', 'disaster', 'discouraging', 'disdain', 'disgust',
'disgrace', 'dishonest', 'dislike', 'disregard', 'disrespect', 'distast
eful', 'distraught', 'doubt', 'dull', 'error', 'excuse', 'excessive',
'exhaust', 'expire', 'fail', 'fake', 'fall', 'frustrate', 'greed', 'gri
eve', 'gross', 'hard', 'harsh', 'harm', 'hate', 'horrid', 'horrible',
'hostile', 'hurt', 'impatient', 'impossible', 'inadequate', 'inaccurat
e', 'inconsistent', 'inconvenience', 'incorrect', 'ineffective', 'insan
e', 'insignificant', 'insult', 'intense', 'intolerable', 'irresponsibl
e', 'liar', 'lie', 'loath', 'limit', 'lose', 'loss', 'mad', 'mess', 'mi
serable', 'miss', 'nasty', 'odd', 'offend', 'overpriced', 'overrated',
'overstatement', 'oversize', 'overwhelm', 'pain', 'panic', 'paranoid',
'pathetic', 'peeve', 'poor', 'pretend', 'problem', 'protest', 'punish',
'provoke', 'rage', 'rant', 'refuse', 'regret', 'reject', 'remorse', 're
pulse', 'resent', 'revolt', 'rip-off', 'ripoff', 'ruin', 'rough', 'sa
d', 'severe', 'shock', 'sick', 'shun', 'slow', 'sorry', 'stain', 'stres
s', 'stupid', 'suck', 'suffer', 'terrible', 'threat', 'tricky', 'unacce
ptable', 'unable', 'unavailable', 'unavoidably', 'unbearably', 'unbelie
vable', 'uncertain', 'uncomfy', 'unfortunate', 'unhappy', 'unnatural',
'unlucky', 'unpopular', 'unpleasant', 'unreasonable', 'unsatisfactory',
'untrue', 'unusual', 'upset', 'violate', 'weak', 'waste', 'weird', 'wor
thless', 'wreak', 'wretch', 'wrong', 'unflatter', 'small', 'flat', 'dis
appointment', 'return', 'massive', 'short', 'bulk', 'issue', 'shrunk',
'crazy']
['good', 'style', 'best', 'gorgeous', 'amaze', 'flatter', 'sweet', 'bea
utiful', 'great', 'perfect', 'nice', 'love', 'lovely', 'cute', 'cozy',
'comfy', 'comfort', 'comfortable', 'elegant', 'impressed', 'gorgeous',
'stunning', 'stun', 'pretty', 'sexy', 'fun', 'like', 'favorite', 'attra
ctive', 'fabulous', 'stunning', 'happy', 'bright', 'admire', 'adore',
```

```
'adorable', 'affordable', 'amaze', 'appreciate', 'attract', 'awesome',
'bargain', 'best', 'enjoy', 'fantastic', 'flawless', 'friendly', 'fres
h', 'splendid', 'success', 'wonderful', 'stylish', 'better', 'clean',
'cool', 'chic', 'gain', 'gentle', 'glad', 'glee', 'great', 'help', 'jo
y', 'nice', 'neat', 'pleasant', 'positive', 'recommend', 'remarkable',
'rich', 'right', 'satisfy', 'soft', 'luxury', 'approve', 'classic', 'cl
assy', 'vibrant', 'unique', 'excite', 'compliment', 'easy', 'fab', 'ar
t', 'elegant', 'cozy', 'nice', 'like', 'attract', 'impress']
```

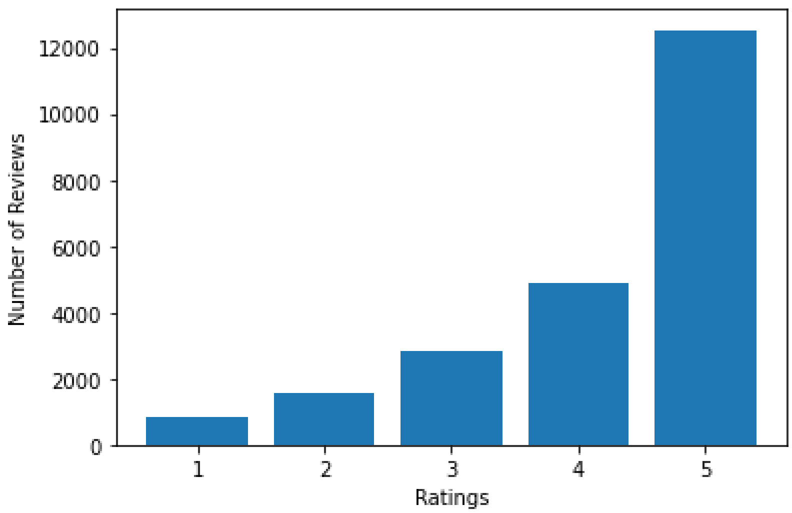
In [4]:

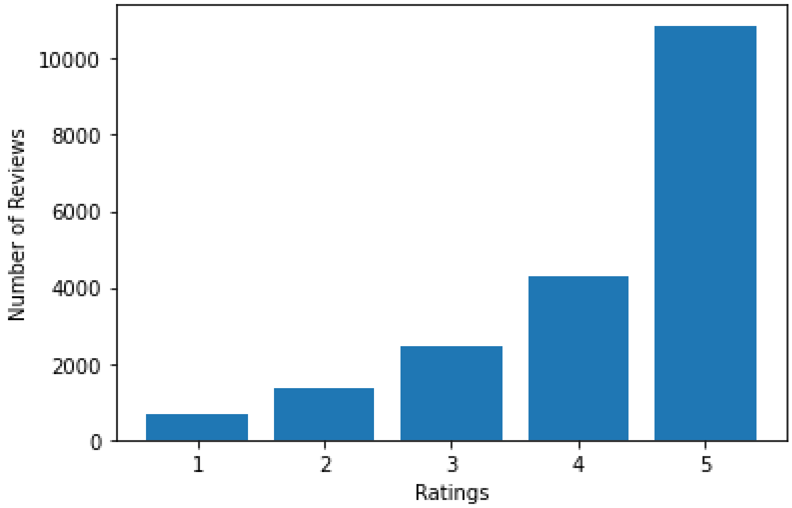
```
#Load the original data
sentiment_df= pd.read_csv("Womens Clothing E-Commerce Reviews.csv",delimiter
df = sentiment_df #copy

print(sentiment_df.head())
#print(df.head())
```

```
   Unnamed: 0  Clothing ID  Age                    Title  \
0           0          767   33                      NaN
1           1         1080   34                      NaN
2           2         1077   60  Some major design flaws
3           3         1049   50         My favorite buy!
4           4          847   47         Flattering shirt

                                         Review Text  Rating  Recommended I
ND  \
0  Absolutely wonderful - silky and sexy and comf...       4
1
1  Love this dress!  it's sooo pretty.  i happene...       5
1
2  I had such high hopes for this dress and reall...       3
0
3  I love, love, love this jumpsuit. it's fun, fl...       5
1
4  This shirt is very flattering to all due to th...       5
1

   Positive Feedback Count   Division Name Department Name Class Name
0                        0       Initmates        Intimate  Intimates
1                        4         General         Dresses    Dresses
2                        0         General         Dresses    Dresses
3                        0  General Petite         Bottoms      Pants
4                        6         General            Tops    Blouses
```

In [5]:

```
#graphs - showing distriubtions of Ratings based on dif text features #showin

Sentiment_count=df.groupby('Rating').count()

plt.bar(Sentiment_count.index.values,  Sentiment_count['Review Text'])
plt.xlabel('Ratings')
plt.ylabel('Number of Reviews')
plt.show()


plt.bar(Sentiment_count.index.values,  Sentiment_count['Title'])
plt.xlabel('Ratings')
plt.ylabel('Number of Reviews')
plt.show()
```





In [6]:

```
# sentiment classify -text and title

sentiment_df["Title"] =sentiment_df["Title"].fillna(0)#replace nan with 0
sentiment_df["Review Text"] =sentiment_df["Review Text"].fillna(0)

sentiment_df_title=sentiment_df["Title"]
sentiment_df_text=sentiment_df["Review Text"]


print(sentiment_df_title.head())
print()
print(sentiment_df_text.head())
```

```
0                          0
1                          0
2    Some major design flaws
3           My favorite buy!
4           Flattering shirt
Name: Title, dtype: object

0    Absolutely wonderful - silky and sexy and comf...
1    Love this dress!  it's sooo pretty.  i happene...
2    I had such high hopes for this dress and reall...
3    I love, love, love this jumpsuit. it's fun, fl...
4    This shirt is very flattering to all due to th...
Name: Review Text, dtype: object
```

In [7]:

```
#title
title_evals= []


for i in range(len(sentiment_df_title)):#for each sentence
    num_good_words = 0
    num_bad_words = 0

    final_eval = 0

    was_negated = False
    was_expression = False


    if (sentiment_df_title.values[i]!=0):#not na

        sentence = sentiment_df_title.values[i]
        print(sentence)#debug
        words=word_tokenize(sentence)

        for j in range(len(words)): #for each word

            word = words[j].lower()
            rootWord = lem.lemmatize(word,"v") #root words

            #meaning of word
            if rootWord in goodWords :
                num_good_words+=1

            elif rootWord  in badWords:
                num_bad_words+=1



            elif rootWord in negateWords:
                was_negated = True
            elif rootWord in expressionWords:
                was_expression = True


    #calc final eval
    final_eval = num_good_words-num_bad_words

    #check for negation or expresion
    #if a word in negationWords: ->increase other counter
    #if a word in expressionWords: ->increase the counter more?

    if(was_negated):
        final_eval = -final_eval

    if(was_expression):
        final_eval = final_eval*2#*5#*10 #nomralize them after?

    print("was_negated, was_expression = ",was_negated,was_expression)
    print("num_good_words, num_bad_words = ",num_good_words, num_bad_words   )
    print("final_eval =",final_eval,"\n"   )
```

```
    #put final eval in list or somthing to add to descript features
    title_evals.append(final_eval)
```

```
was_negated, was_expression =  False False
num_good_words, num_bad_words =  0 0
final_eval = 0

was_negated, was_expression =  False False
num_good_words, num_bad_words =  0 0
final_eval = 0

Some major design flaws
was_negated, was_expression =  False False
num_good_words, num_bad_words =  0 1
final_eval = -1

My favorite buy!
was_negated, was_expression =  False True
num_good_words, num_bad_words =  1 0
final_eval = 2

Flattering shirt
```

In [8]:

```
#text
text_evals= []


for i in range(len(sentiment_df_text)):#for each sentence
    num_good_words = 0
    num_bad_words = 0

    final_eval = 0

    was_negated = False
    was_expression = False

    if (sentiment_df_text.values[i]!=0):#not na

        sentence = sentiment_df_text.values[i]
        words=word_tokenize(sentence)

        for j in range(len(words)): #for each word


            word = words[j].lower()
            rootWord = lem.lemmatize(word,"v") #root words

            #meaning of word
            if rootWord in goodWords :
                num_good_words+=1

            elif rootWord  in badWords:
                num_bad_words+=1


            elif rootWord in negateWords:
                was_negated = True
            elif rootWord in expressionWords:
                was_expression = True


    #calc final eval
    final_eval = num_good_words-num_bad_words


    #check for negation or expresion
    if(was_negated):
        final_eval = -final_eval

    if(was_expression):
        final_eval = final_eval*2#*5 #*10

    #print("was_negated, was_expression",was_negated,was_expression)
    #print("num_good_words, num_bad_words =",num_good_words, num_bad_wor
    print("final_eval =",final_eval,"\n"   )


    #put final eval in list or somthing to add to descript features
    text_evals.append(final_eval)
```

```
final_eval = 3

final_eval = 8

final_eval = 6

final_eval = -14

final_eval = 6

final_eval = -2

final_eval = -4

final_eval = 0

final_eval = -6

final_eval = -4
```

In [9]:

```
#preprocess data #find descriptive vs target features

all_features  = df
#print(all_features.shape)

all_features = all_features.iloc[: , 1:]  #drop first unnamed col of datafram
#print (all_features.head())


#Rating is target

target_features = np.array(all_features['Rating'])
print("target_features===============\n")
print(target_features,"\n\n")
print(target_features.shape)


#descriptive_features: want - title, reivew text, Positive Feedback Count,Rec

descriptive_features=all_features.drop('Rating', axis = 1)#drop target
descriptive_features=descriptive_features.drop('Clothing ID', axis = 1)#drop
descriptive_features=descriptive_features.drop('Age', axis = 1)
descriptive_features=descriptive_features.drop('Division Name', axis = 1)
descriptive_features=descriptive_features.drop('Department Name', axis = 1)
descriptive_features=descriptive_features.drop('Class Name', axis = 1)


#replace text and title - with final eval

descriptive_features=descriptive_features.drop('Title', axis = 1)
descriptive_features=descriptive_features.drop('Review Text', axis = 1)

descriptive_features.insert(2, "titleNew", title_evals)
descriptive_features.insert(3, "textNew", text_evals)

print("descriptive_features===============\n")
print(descriptive_features.head(),"\n")
descriptive_features = np.array(descriptive_features)

print(descriptive_features,"\n\n")
print(descriptive_features.shape)
```

```
target_features===============

[4 5 3 ... 3 3 5]


(23486,)
descriptive_features===============

   Recommended IND  Positive Feedback Count  titleNew  textNew
0                1                        0         0        3
1                1                        4         0        8
2                0                        0        -1        6
```

```
3                1                        0          2       -14
4                1                        6          1         6

[[ 1  0  0  3]
 [ 1  4  0  8]
 [ 0  0 -1  6]
 ...
 [ 0  1 -1 -4]
 [ 1  2  4 -4]
 [ 1 22  2  8]]


(23486, 4)
```

In [10]:

```
#Randomly split original data #70% for training : 30% for testing

descriptive_train, descriptive_test, target_train, target_test = train_test_s

print('descriptive_train:', descriptive_train.shape)
print('target_train:', target_train.shape)
print('descriptive_test:', descriptive_test.shape)
print('target_test:', target_test.shape)
```

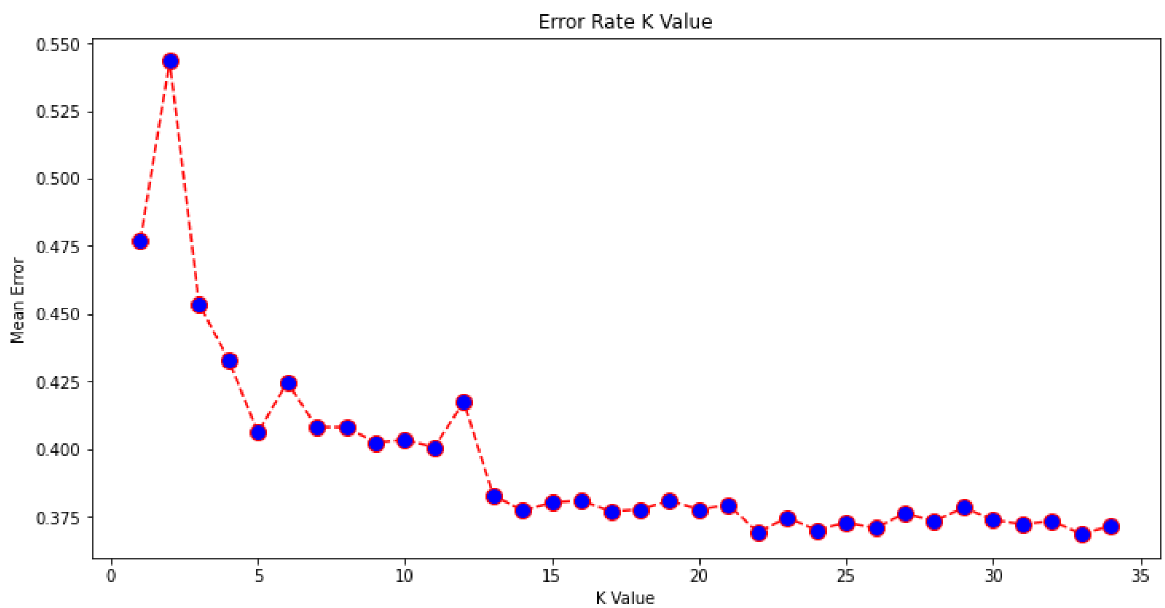
```
descriptive_train: (16440, 4)
target_train: (16440,)
descriptive_test: (7046, 4)
target_test: (7046,)
```

In [11]:

```
#find a k value for knn- look for lowest point(error) on graph

minError = 1 #where is least erorr
minErrorK = 0 #which k to choose
errors = [0] #errors = []

# Calculate error for dif K values
for i in range(1,35):#(5,35):#(1, 40)

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(descriptive_train, target_train)
    pred_i = knn.predict(descriptive_test)
    errors.append(np.mean(pred_i != target_test))

    #print(errors[i-1],np.mean(pred_i != target_test),minError,minErrorK)
    if(errors[i]<minError): #if(errors[i-1]<minError):
        minError=errors[i]
        minErrorK=i

errors=errors[1:]
print("choose k = ", minErrorK)

plt.figure(figsize=(12, 6))
plt.plot(range(1, 35), errors, color='red', linestyle='dashed', marker='o', m

plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

```
choose k =  33
```

Out[11]: Text(0, 0.5, 'Mean Error')



In [12]:

```
#K-Nearest Neighbors - predictive models

print("K-Nearest Neighbors\n")

#model = KNeighborsClassifier(n_neighbors = 32) #make modle #17 #10,50,20

model = KNeighborsClassifier(n_neighbors = minErrorK) #make modle  #more flex
#model = KNeighborsClassifier(n_neighbors = minErrorK,weights='distance')

model = model.fit(descriptive_train, target_train) #train
target_predict = model.predict(descriptive_test) #predict

print("target_predict===============\n")
print(target_predict)
print(target_predict.shape,"\n")
```

```
K-Nearest Neighbors

target_predict===============

[5 5 5 ... 4 5 5]
(7046,)
```

In [13]:

```
# eval model # recall,f1 score

fscore = f1_score(target_test, target_predict, average='weighted')
print('F1score: {:f}'.format( fscore))


#precision = precision_score(target_test, target_predict, average='weighted')
#print('precision: {:f}'.format( precision))

recall = recall_score(target_test, target_predict, average='weighted')
print('recall: {:f}'.format( recall))
```

```
F1score: 0.564304
recall: 0.631422
```

In [14]:

```
#Naive Bayes - predictive models #not as good

print("Naive Bayes\n")

model = GaussianNB() #make modle
model = model.fit(descriptive_train, target_train) #train
target_predict = model.predict(descriptive_test) #predict

print("target_predict===============\n")
print(target_predict)
print(target_predict.shape,"\n")


fscore = f1_score(target_test, target_predict, average='weighted')
print('F1score: {:f}'.format( fscore))
recall = recall_score(target_test, target_predict, average='weighted')
print('recall: {:f}'.format( recall))
```

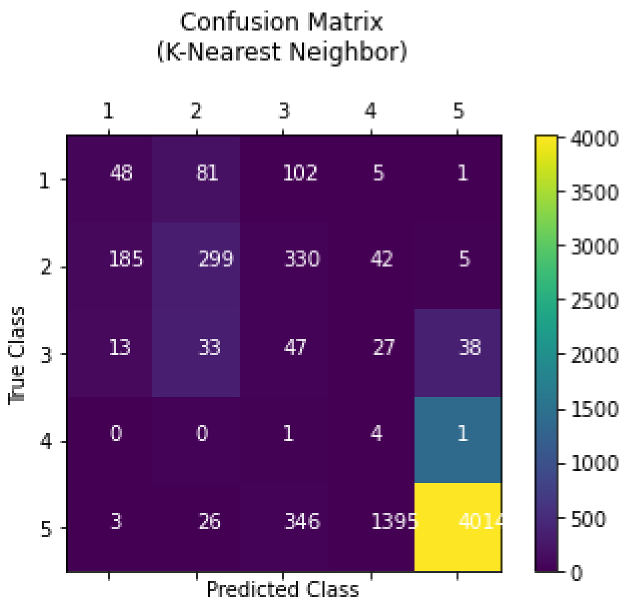
```
Naive Bayes

target_predict===============

[5 5 5 ... 5 5 5]
(7046,)

F1score: 0.517817
recall: 0.626171
```

In [15]:

```
#visualizations
#eg
#1,1 -> true pos of rating 1
#5,5 -> true pos of rating 5


def draw_confusion_matrices(confusion_matrices, class_names):
    labels = list(class_names)

    for cm in confusion_matrices:
        fig = pl.figure()
        ax = fig.add_subplot(111)

        cax = ax.matshow(cm[1])
        pl.title('Confusion Matrix\n(%s)\n' % cm[0])
        fig.colorbar(cax)
        ax.set_xticklabels([''] + labels)
        ax.set_yticklabels([''] + labels)
        pl.xlabel('Predicted Class')
        pl.ylabel('True Class')

        for i,j in ((x,y) for x in range(len(cm[1])) for y in range(len(cm[1]
            ax.annotate(str(cm[1][i][j]), xy=(i,j), color='white')

        pl.show()


y= target_test
y = np.array(y)
class_names = np.unique(y)


print(classification_report(target_predict, target_test))
print(confusion_matrix(target_predict, target_test))


confusion_matrices = [ ("K-Nearest Neighbor", confusion_matrix(y, target_prec
draw_confusion_matrices(confusion_matrices, class_names)
```

```
              precision    recall  f1-score   support

           1       0.19      0.20      0.20       237
           2       0.68      0.35      0.46       861
           3       0.06      0.30      0.10       158
           4       0.00      0.67      0.01         6
           5       0.99      0.69      0.82      5784

    accuracy                           0.63      7046
   macro avg       0.38      0.44      0.31      7046
weighted avg       0.90      0.63      0.73      7046

[[  48   81  102    5    1]
 [ 185  299  330   42    5]
 [  13   33   47   27   38]
 [   0    0    1    4    1]
 [   3   26  346 1395 4014]]
```

```
C:\Users\Morgan\AppData\Local\Temp/ipykernel_14728/3686583269.py:17: UserWa
rning: FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels([''] + labels)
C:\Users\Morgan\AppData\Local\Temp/ipykernel_14728/3686583269.py:18: UserWa
rning: FixedFormatter should only be used together with FixedLocator
  ax.set_yticklabels([''] + labels)
```



In [16]:

```
'''
IN CONCLUSIONS...we get results similar to :

F1score: 0.555832
recall: 0.619075

                  precision    recall  f1-score   support

           1       0.05      0.30      0.08        44
           2       0.27      0.36      0.31       371
           3       0.33      0.39      0.36       713
           4       0.15      0.45      0.22       506
           5       0.96      0.69      0.80      5412

    accuracy                           0.62      7046
   macro avg       0.35      0.44      0.35      7046
weighted avg       0.79      0.62      0.68      7046



therefore our model is about 62% accurate, F1score is 56%, and recall is 62%

In addition:
f1 score of each target is different.
eg:
rating 1 with fscore= 0.08
vs
rating 5 with fscore= 0.80

this is either because the data is highly imbalanced or because we are better

'''
```

Out[16]:

```
'\nIN CONCLUSIONS...we get results similar to : \n\nF1score: 0.555832\nprec
ision: 0.559843\nrecall: 0.619075\n\n                 precision    recall
f1-score   support\n\n           1       0.05      0.30      0.08        44
\n           2       0.27      0.36      0.31       371\n           3
0.33      0.39      0.36       713\n           4       0.15      0.45
0.22       506\n           5       0.96      0.69      0.80      5412\n\n
accuracy                           0.62      7046\n   macro avg       0.35
0.44      0.35      7046\nweighted avg       0.79      0.62      0.68
7046\n\n\n \ntherefore our model is about 62% accurate, F1score is 56%, and
recall is 62% .\n\nIn addition:\nf1 score of each target is different. \ne
g: \nrating 1 with fscore= 0.08 \nvs \nrating 5 with fscore= 0.80 \n\nthis
is either because the data is highly imbalanced or because we are better at
predicting the 5 star ratings.\n\n'
```