KevinSmithMidterm506

November 8, 2023

0.1 Kevin Smith CS506 Midterm Project

[144]:

```python
import pandas as pd
trainingSet = pd.read_csv("./data/train.csv")
```

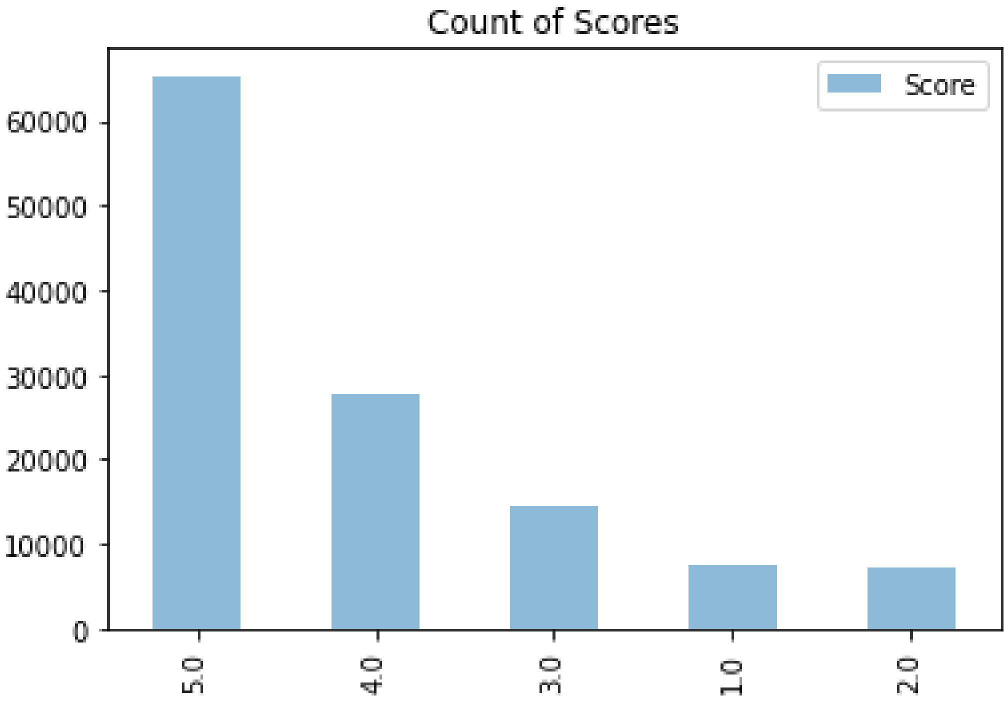
0.2 Exploration

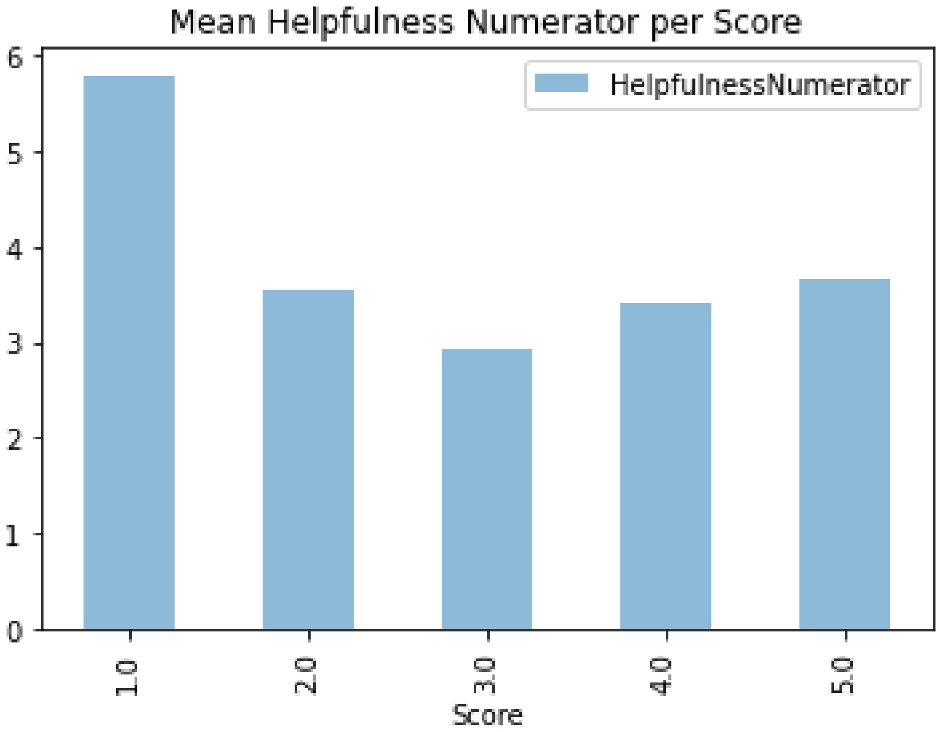
[49]:

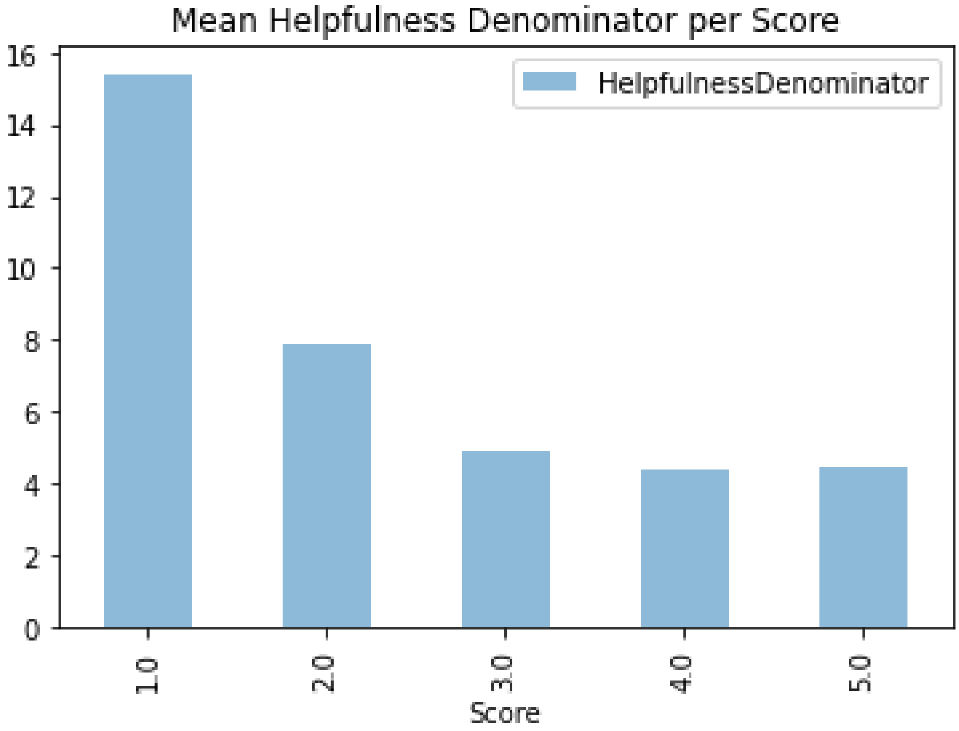
```python
import matplotlib.pyplot as plt
import math

print("Each review has the following attributes:")
print(trainingSet.dtypes)
print()
print(trainingSet.head())
print()

#COUNT OF EACH SCORE
trainingSet['Score'].value_counts().plot(kind='bar', legend=True, alpha=.5)
plt.title("Count of Scores")
plt.show()

#MEAN HELPFULNESS NUMERATOR PER SCORE
trainingSet[['Score', 'HelpfulnessNumerator']].groupby('Score').mean().plot(kind='bar', legend=True, alpha=.5)
plt.title("Mean Helpfulness Numerator per Score")
plt.show()

#MEAN HELPFULNESS DENOMINATOR PER SCORE
trainingSet[['Score', 'HelpfulnessDenominator']].groupby('Score').mean().plot(kind='bar', legend=True, alpha=.5)
plt.title("Mean Helpfulness Denominator per Score")
plt.show()

#MEAN HELPFULNESS PER SCORE
trainingSet['Helpfulness'] = trainingSet['HelpfulnessNumerator'] / trainingSet['HelpfulnessDenominator']
trainingSet['Helpfulness'] = trainingSet['Helpfulness'].fillna(0)
```

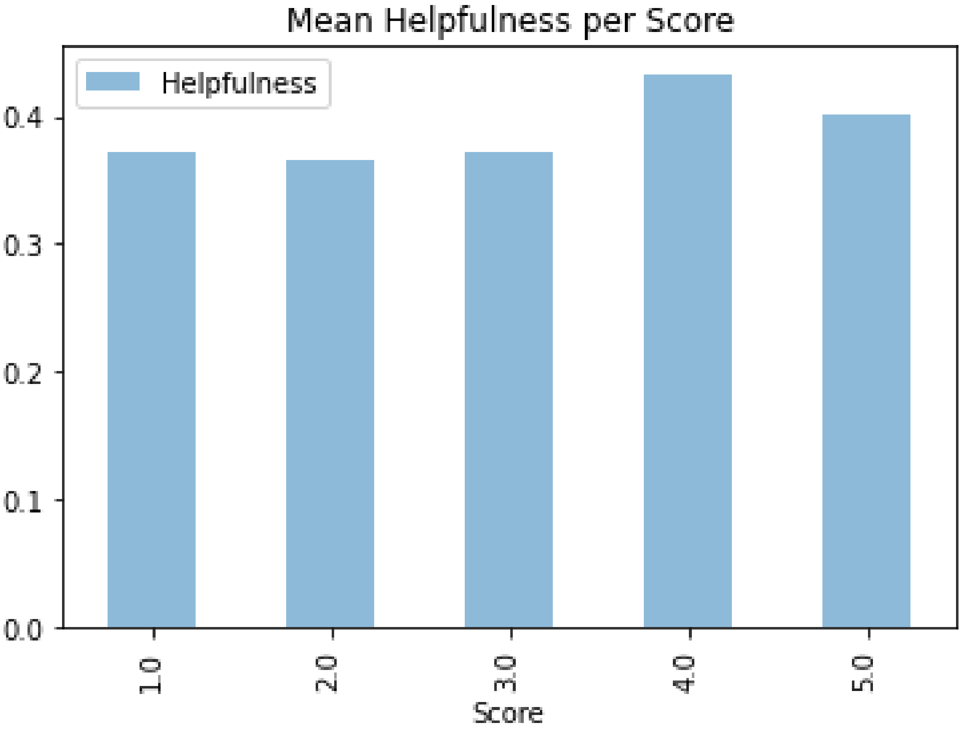
```
trainingSet[['Score', 'Helpfulness']].groupby('Score').mean().plot(kind='bar',␣
 ↪legend=True, alpha=.5)
plt.title("Mean Helpfulness per Score")
plt.show()

def getHour(time):
    return math.floor(time/(1000*3600))%24

#TIME DISTRIBUTION OF 1-STAR REVIEWS
one_star_reviews = trainingSet[trainingSet['Score'] == 1]
one_star_reviews.loc[:, 'Hour'] = one_star_reviews['Time'].apply(getHour)
hourly_counts = one_star_reviews['Hour'].value_counts().sort_index()
hourly_index = range(24)
hourly_counts = hourly_counts.reindex(hourly_index, fill_value=0)
plt.figure(figsize=(10, 6))
plt.bar(hourly_counts.index, hourly_counts.values, tick_label=hourly_counts.
 ↪index)
plt.xlabel('Hour of the Day')
plt.ylabel('Number of 1-Star Reviews')
plt.title('Time Distribution of 1-Star Reviews')
plt.xticks(hourly_counts.index)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.show()

#TIME DISTRIBUTION OF 2-STAR REVIEWS
two_star_reviews = trainingSet[trainingSet['Score'] == 2]
two_star_reviews.loc[:, 'Hour'] = two_star_reviews['Time'].apply(getHour)
hourly_counts = two_star_reviews['Hour'].value_counts().sort_index()
hourly_index = range(24)
hourly_counts = hourly_counts.reindex(hourly_index, fill_value=0)
plt.figure(figsize=(10, 6))
plt.bar(hourly_counts.index, hourly_counts.values, tick_label=hourly_counts.
 ↪index)
plt.xlabel('Hour of the Day')
plt.ylabel('Number of 1-Star Reviews')
plt.title('Time Distribution of 1-Star Reviews')
plt.xticks(hourly_counts.index)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.show()

#TIME DISTRIBUTION OF 3-STAR REVIEWS
three_star_reviews = trainingSet[trainingSet['Score'] == 3]
three_star_reviews.loc[:, 'Hour'] = three_star_reviews['Time'].apply(getHour)
hourly_counts = three_star_reviews['Hour'].value_counts().sort_index()
hourly_index = range(24)
hourly_counts = hourly_counts.reindex(hourly_index, fill_value=0)
plt.figure(figsize=(10, 6))
plt.bar(hourly_counts.index, hourly_counts.values, tick_label=hourly_counts.
 ↪index)
plt.xlabel('Hour of the Day')
plt.ylabel('Number of 1-Star Reviews')
plt.title('Time Distribution of 1-Star Reviews')
plt.xticks(hourly_counts.index)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.show()

#TIME DISTRIBUTION OF 4-STAR REVIEWS
four_star_reviews = trainingSet[trainingSet['Score'] == 4]
four_star_reviews.loc[:, 'Hour'] = four_star_reviews['Time'].apply(getHour)
hourly_counts = four_star_reviews['Hour'].value_counts().sort_index()
hourly_index = range(24)
hourly_counts = hourly_counts.reindex(hourly_index, fill_value=0)
plt.figure(figsize=(10, 6))
plt.bar(hourly_counts.index, hourly_counts.values, tick_label=hourly_counts.
 ↪index)
plt.xlabel('Hour of the Day')
plt.ylabel('Number of 1-Star Reviews')
plt.title('Time Distribution of 1-Star Reviews')
plt.xticks(hourly_counts.index)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.show()

#TIME DISTRIBUTION OF 5-STAR REVIEWS
five_star_reviews = trainingSet[trainingSet['Score'] == 5]
five_star_reviews.loc[:, 'Hour'] = five_star_reviews['Time'].apply(getHour)
hourly_counts = five_star_reviews['Hour'].value_counts().sort_index()
hourly_index = range(24)
hourly_counts = hourly_counts.reindex(hourly_index, fill_value=0)
plt.figure(figsize=(10, 6))
plt.bar(hourly_counts.index, hourly_counts.values, tick_label=hourly_counts.
 ↪index)
plt.xlabel('Hour of the Day')
plt.ylabel('Number of 1-Star Reviews')
plt.title('Time Distribution of 1-Star Reviews')
plt.xticks(hourly_counts.index)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.show()

#MEAN REVIEW LENGTH PER SCORE
trainingSet['ReviewLength'] = trainingSet.apply(lambda row : len(row['Text'].
 ↪split()) if type(row['Text']) == str else 0, axis = 1)
trainingSet[['Score', 'ReviewLength']].groupby('Score').mean().plot(kind='bar',␣
 ↪legend=True, alpha=.5)
plt.title("Mean Review Length per Score")
plt.show()
```

```
Each review has the following attributes:
Id                          int64
ProductId                  object
UserId                     object
HelpfulnessNumerator        int64
HelpfulnessDenominator      int64
Time                        int64
Summary                    object
Text                       object
Score                     float64
Helpfulness               float64
dtype: object

        Id   ProductId          UserId  HelpfulnessNumerator  \
0   195370  1890228583  A3VLX5Z090RQ0V                     1
1  1632470  B00BEIYSL4   AUDXDMFM49NGY                     0
2     9771  0767809335  A3LFIA97BUU5IE                     3
3   218855  6300215792  A1QZM75342ZQVQ                     1
4   936225  B000B5XOZW   ANM2SCEUL3WL1                     1

   HelpfulnessDenominator        Time  \
0                       2  1030838400
1                       1  1405036800
2                      36   983750400
3                       1  1394841600
4                       1  1163721600

                                             Summary  \
0                        An Unexplained Anime Review
1                                         not great.
2                    Technical problem with this DVD
3                          Heeeeyyyyy LAAAAADEEE!!!!
4  Herzog the Great Traveler of both natural and …

                                                Text  Score  Helpfulness
0  I was very anxious to see the Uncut version of…    2.0     0.500000
1                        Movie was okay…not great.    3.0     0.000000
2  Like the Dinosaur Collector's Edition DVD, thi…    1.0     0.083333
3  Come on, now…  this has to be, by far, the…    5.0     1.000000
4  I've always been a great admirer of Herzog's o…    4.0     1.000000
```

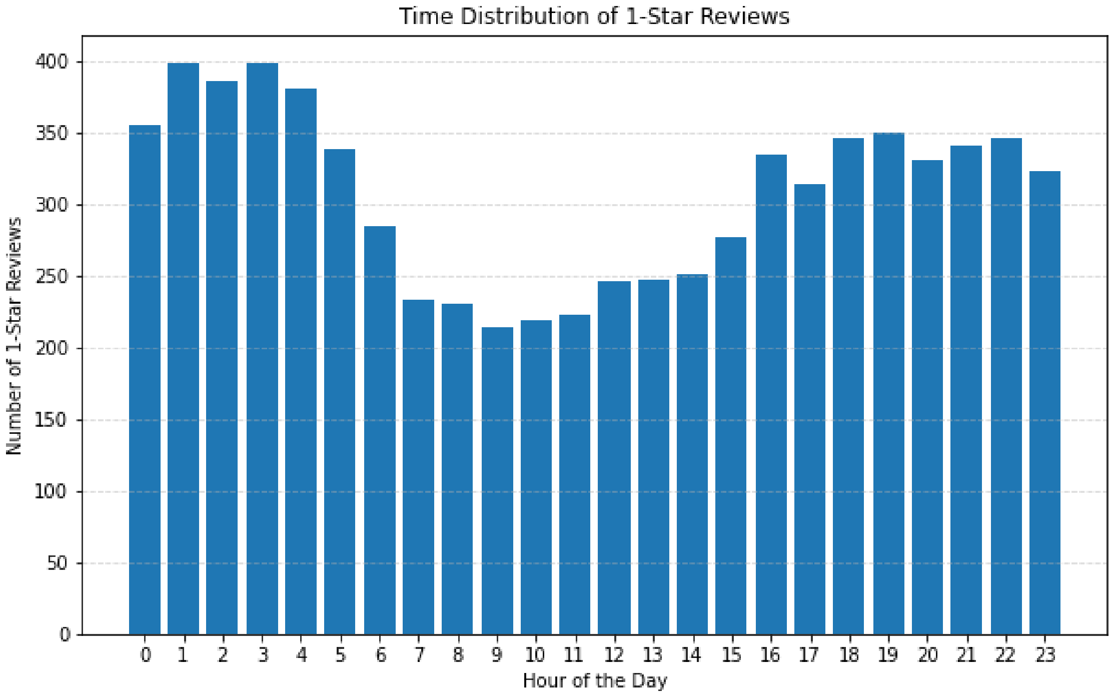

Count of Scores
Score
60000
50000
40000
30000
20000
10000
0
5.0
4.0
3.0
1.0
2.0


Mean Helpfulness Numerator per Score
HelpfulnessNumerator
6
5
4
3
2
1
0
1.0
2.0
3.0
4.0
5.0
Score

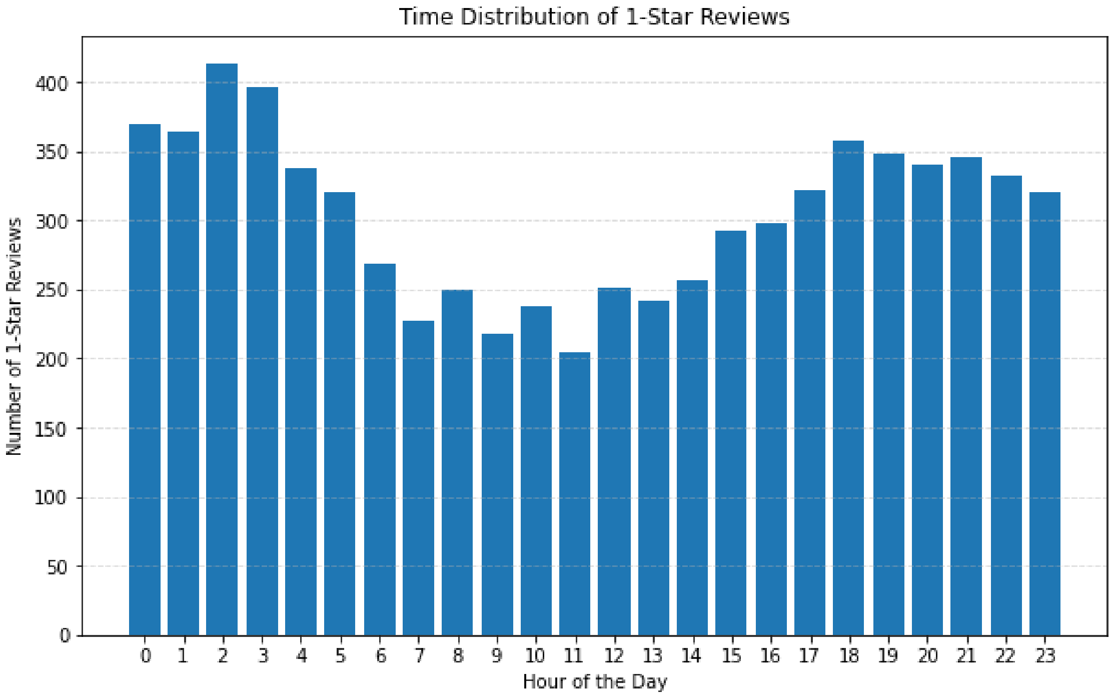

Mean Helpfulness Denominator per Score
HelpfulnessDenominator
0
2
4
6
8
10
12
14
16
1.0
2.0
3.0
4.0
5.0
Score


Mean Helpfulness per Score
Helpfulness
0.0
0.1
0.2
0.3
0.4
1.0
2.0
3.0
4.0
5.0
Score

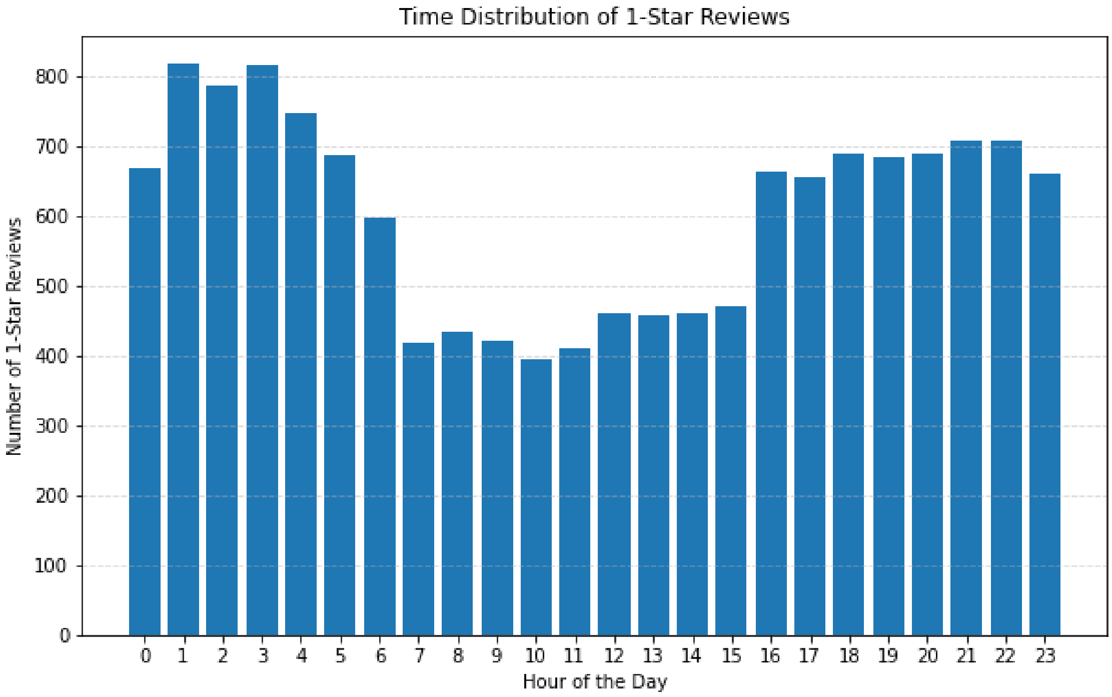
```
/var/folders/n1/mp2n6fsd09963xzzjvzq12g80000gn/T/ipykernel_73938/3394742300.py:3
7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  one_star_reviews.loc[:, 'Hour'] = one_star_reviews['Time'].apply(getHour)
```



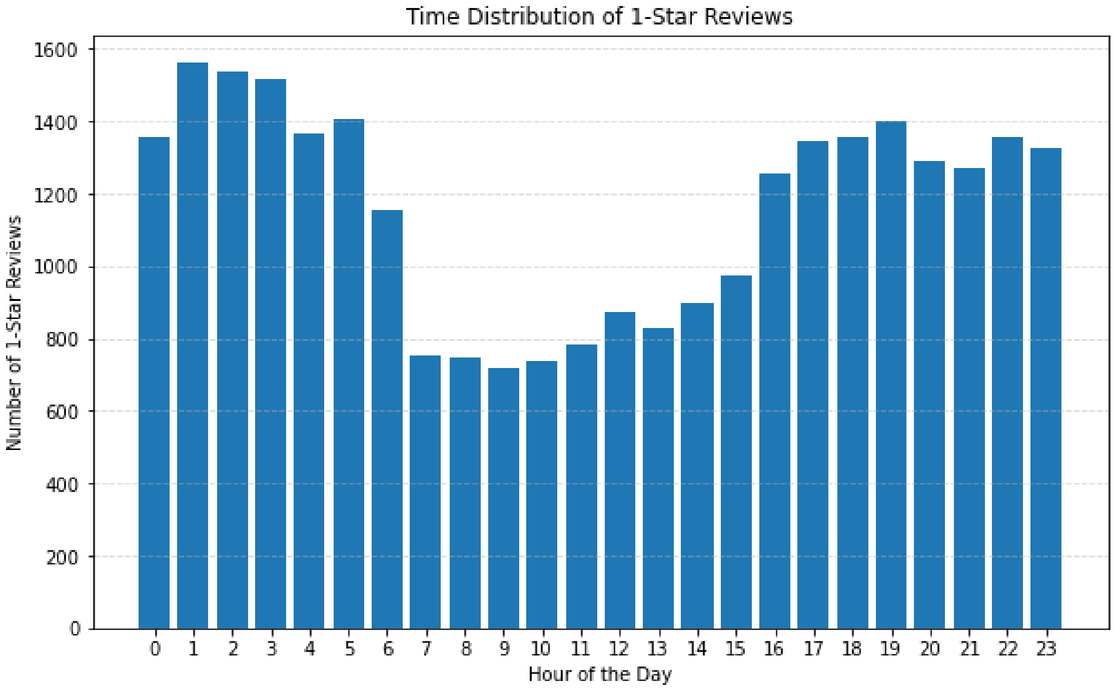
```
/var/folders/n1/mp2n6fsd09963xzzjvzq12g80000gn/T/ipykernel_73938/3394742300.py:5
2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  two_star_reviews.loc[:, 'Hour'] = two_star_reviews['Time'].apply(getHour)
```



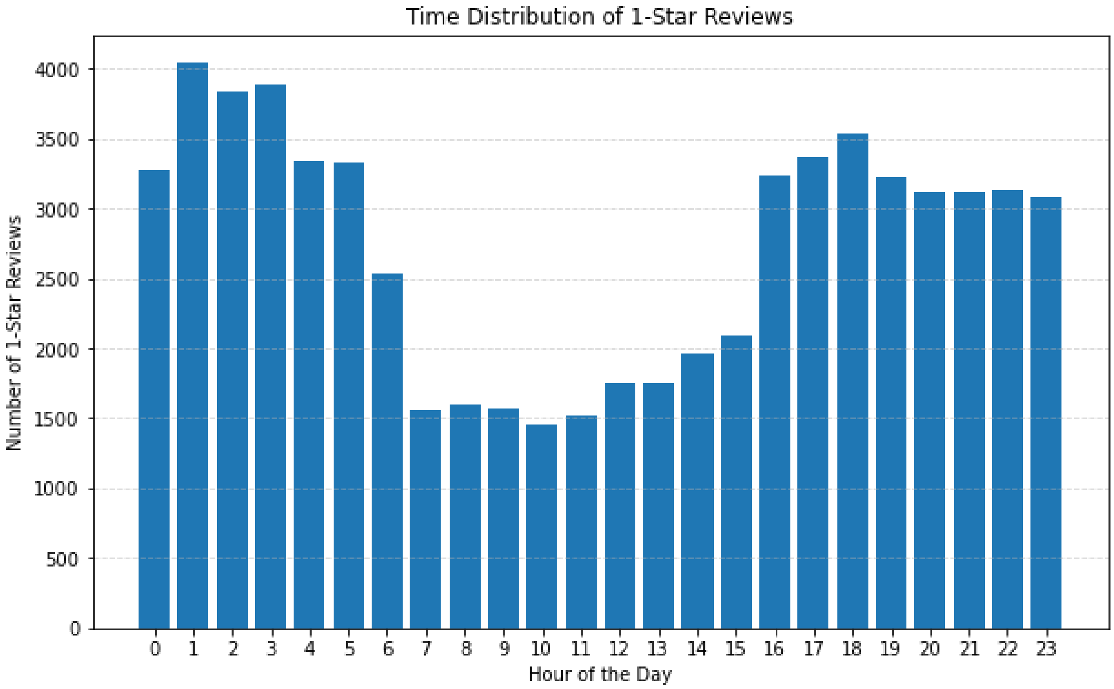
```
/var/folders/n1/mp2n6fsd09963xzzjvzq12g80000gn/T/ipykernel_73938/3394742300.py:6
7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  three_star_reviews.loc[:, 'Hour'] = three_star_reviews['Time'].apply(getHour)
```

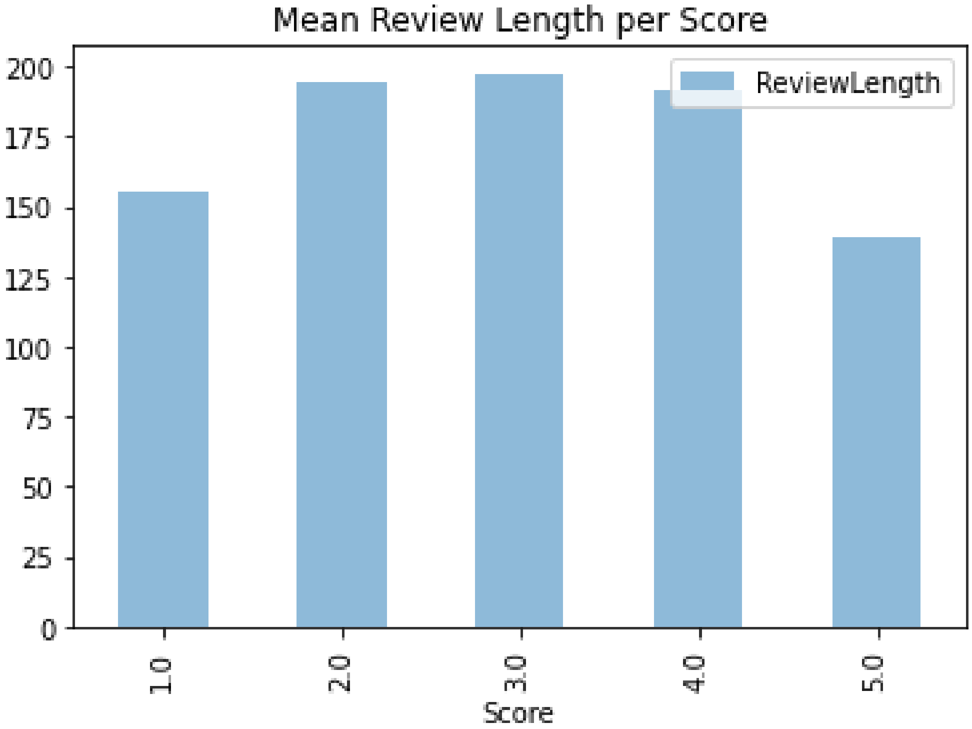


```
/var/folders/n1/mp2n6fsd09963xzzjvzq12g80000gn/T/ipykernel_73938/3394742300.py:8
2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  four_star_reviews.loc[:, 'Hour'] = four_star_reviews['Time'].apply(getHour)
```



```
/var/folders/n1/mp2n6fsd09963xzzjvzq12g80000gn/T/ipykernel_73938/3394742300.py:9
7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  five_star_reviews.loc[:, 'Hour'] = five_star_reviews['Time'].apply(getHour)
```


Time Distribution of 1-Star Reviews
Number of 1-Star Reviews
0
500
1000
1500
2000
2500
3000
3500
4000
Hour of the Day
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23


Mean Review Length per Score
ReviewLength
0
25
50
75
100
125
150
175
200
Score
1.0
2.0
3.0
4.0
5.0

0.3 Feature Extraction

[147]:

```
import pandas as pd
import numpy as np
import re
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

#NEW FEATURES ARE EXTRACTED HERE
def process(df):

    ## SIMPLE EDITS ##

    df = df.rename(columns={'ProductId': 'MovieID'})
    df = df.rename(columns={'UserId': 'UserID'})

    df['Summary'] = df['Summary'].fillna('neutral')
    df['Text'] = df['Text'].fillna('neutral')
    df['Summary'] = df['Summary'].apply(lambda x : x.lower())
    df['Text'] = df['Text'].apply(lambda x : x.lower())

    ## NEW FIELDS ##

    df['ReviewLength'] = df.apply(lambda row : len(row['Text'].split()) if␣
 ↪type(row['Text']) == str else 0, axis = 1)

    df['AvgScoreByUser'] = df.groupby('UserID')['Score'].transform('mean')

    ## COMPLEXITY ANALYSIS (COLEMAN-LIAU)##

    def coleman_liau_index(text):
        letters = len(re.findall(r'[a-zA-Z]', text))
        sentences = len(re.split(r'[.!?]', text))
        L = (letters / len(text)) * 100
        S = (sentences / len(text)) * 100
        return 0.0588 * L - 0.296 * S - 15.8

    df['ColemanLiauIndex'] = df['Text'].apply(coleman_liau_index)

    ## SENTIMENT ANALYSIS ##

    analyzer = SentimentIntensityAnalyzer()

    def get_positivity_score(text):
        sentiment = analyzer.polarity_scores(text)
        return sentiment['compound']
    textPS = df['Text'].apply(get_positivity_score)
    summaryPS = df['Summary'].apply(get_positivity_score)

    df['PositivityScore'] = textPS

    return df

# Load the dataset
trainingSet = pd.read_csv("./data/train.csv")
testingSet = pd.read_csv("./data/test.csv")

# Process the DataFrames
train_processed = process(trainingSet)

# Load test set
submissionSet = pd.read_csv("./data/test.csv")

# Merge on Id so that the test set can have feature columns as well
testX= pd.merge(train_processed, submissionSet, left_on='Id', right_on='Id')
testX = testX.drop(columns=['Score_x'])
testX = testX.rename(columns={'Score_y': 'Score'})

# The training set is where the score is not null
trainX =  train_processed[train_processed['Score'].notnull()]

# Save the datasets with the new features for easy access later
testX.to_csv("./data/X_test.csv", index=False)
trainX.to_csv("./data/X_train.csv", index=False)
```

0.4 Creating your model

[153]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, mean_squared_error, classification_report
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import hstack
from sklearn.ensemble import AdaBoostClassifier
from catboost import CatBoostClassifier
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Load training set with new features into DataFrame
trainingSet = pd.read_csv("./data/X_train.csv").sample(10000)
```

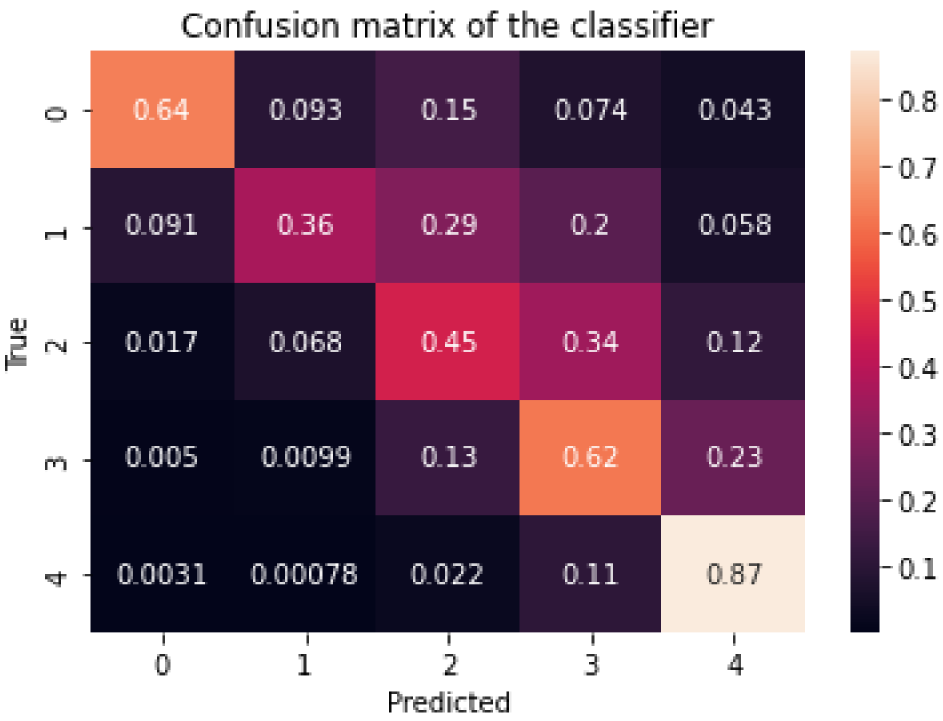
```
# SPLIT DATA INTO TRAINING & TESTING SETS
X_text = trainingSet['Text']
X_other = trainingSet[['HelpfulnessNumerator',␣
 ↪'HelpfulnessDenominator','PositivityScore','ColemanLiauIndex','ReviewLength','AvgScoreByUse
Y = trainingSet['Score']

X_text_train, X_text_test, X_other_train, X_other_test, Y_train, Y_test =␣
 ↪train_test_split(
    X_text, X_other, Y, test_size=0.25, random_state=33)

# APPLY TF-IDF VECTORIZATION TO THE 'Text' COLUMN
tfidf_vectorizer = TfidfVectorizer(max_df=0.5)
X_text_train_tfidf = tfidf_vectorizer.fit_transform(X_text_train)
X_text_test_tfidf = tfidf_vectorizer.transform(X_text_test)

# COMBINE TF-IDF VECTORIZED 'Text' COLUMN WITH OTHER COLUMNS
X_train = hstack((X_text_train_tfidf, X_other_train))
X_test = hstack((X_text_test_tfidf, X_other_test))

# SCALING & NORMALIZATION
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train.toarray())
X_test = scaler.transform(X_test.toarray())

#TRAIN CATBOOST CLASSIFIER
model = CatBoostClassifier(iterations=100, loss_function='MultiClass',␣
 ↪custom_loss=['Accuracy'])
model.fit(X_train, Y_train)

# Evaluate your model on the testing set
Y_test_predictions = model.predict(X_test)
print("Accuracy on testing set = ", accuracy_score(Y_test, Y_test_predictions))
print("RMSE on testing set = ", mean_squared_error(Y_test,␣
 ↪Y_test_predictions)**(1/2))
print("Classification Report:\n", classification_report(Y_test,␣
 ↪Y_test_predictions))

# Plot a confusion matrix
cm = confusion_matrix(Y_test, Y_test_predictions, normalize='true')
sns.heatmap(cm, annot=True)
plt.title('Confusion matrix of the classifier')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
Learning rate set to 0.5
0:      learn: 1.0227662        total: 1.5s     remaining: 2m 28s
```

```
1:      learn: 0.9040791        total: 3.34s    remaining: 2m 43s
2:      learn: 0.8232865        total: 4.91s    remaining: 2m 38s
3:      learn: 0.7877554        total: 6.48s    remaining: 2m 35s
4:      learn: 0.7692956        total: 7.94s    remaining: 2m 30s
5:      learn: 0.7524978        total: 9.4s     remaining: 2m 27s
6:      learn: 0.7466980        total: 10.9s    remaining: 2m 24s
7:      learn: 0.7432049        total: 12.3s    remaining: 2m 21s
8:      learn: 0.7329321        total: 13.7s    remaining: 2m 18s
9:      learn: 0.7237881        total: 15s      remaining: 2m 14s
10:     learn: 0.7177149        total: 16.3s    remaining: 2m 11s
11:     learn: 0.7139312        total: 18s      remaining: 2m 11s
12:     learn: 0.7103950        total: 19.5s    remaining: 2m 10s
13:     learn: 0.7018809        total: 21.2s    remaining: 2m 9s
14:     learn: 0.7010618        total: 23.2s    remaining: 2m 11s
15:     learn: 0.6974500        total: 24.7s    remaining: 2m 9s
16:     learn: 0.6959011        total: 27.4s    remaining: 2m 13s
17:     learn: 0.6893979        total: 29s      remaining: 2m 12s
18:     learn: 0.6871888        total: 30.8s    remaining: 2m 11s
19:     learn: 0.6783016        total: 32.1s    remaining: 2m 8s
20:     learn: 0.6672144        total: 33.5s    remaining: 2m 6s
21:     learn: 0.6617618        total: 35s      remaining: 2m 3s
22:     learn: 0.6567507        total: 36.5s    remaining: 2m 2s
23:     learn: 0.6520099        total: 38s      remaining: 2m
24:     learn: 0.6510253        total: 39.3s    remaining: 1m 57s
25:     learn: 0.6474944        total: 40.5s    remaining: 1m 55s
26:     learn: 0.6440026        total: 42s      remaining: 1m 53s
27:     learn: 0.6432366        total: 43.6s    remaining: 1m 52s
28:     learn: 0.6396755        total: 44.9s    remaining: 1m 50s
29:     learn: 0.6387831        total: 46.2s    remaining: 1m 47s
30:     learn: 0.6368998        total: 47.6s    remaining: 1m 45s
31:     learn: 0.6359230        total: 49s      remaining: 1m 44s
32:     learn: 0.6347502        total: 50.3s    remaining: 1m 42s
33:     learn: 0.6336555        total: 51.5s    remaining: 1m 39s
34:     learn: 0.6295426        total: 52.7s    remaining: 1m 37s
35:     learn: 0.6276315        total: 54.2s    remaining: 1m 36s
36:     learn: 0.6266400        total: 55.5s    remaining: 1m 34s
37:     learn: 0.6256232        total: 56.7s    remaining: 1m 32s
38:     learn: 0.6205476        total: 58.1s    remaining: 1m 30s
39:     learn: 0.6199783        total: 59.3s    remaining: 1m 29s
40:     learn: 0.6164242        total: 1m       remaining: 1m 27s
41:     learn: 0.6124089        total: 1m 1s    remaining: 1m 25s
42:     learn: 0.6112853        total: 1m 3s    remaining: 1m 23s
43:     learn: 0.6104291        total: 1m 4s    remaining: 1m 22s
44:     learn: 0.6081746        total: 1m 5s    remaining: 1m 20s
45:     learn: 0.6063285        total: 1m 7s    remaining: 1m 18s
46:     learn: 0.6027341        total: 1m 8s    remaining: 1m 17s
47:     learn: 0.6019201        total: 1m 9s    remaining: 1m 15s
48:     learn: 0.6008069        total: 1m 11s   remaining: 1m 13s
```

```
49:     learn: 0.5996646        total: 1m 12s   remaining: 1m 12s
50:     learn: 0.5988495        total: 1m 13s   remaining: 1m 10s
51:     learn: 0.5965917        total: 1m 15s   remaining: 1m 9s
52:     learn: 0.5949341        total: 1m 16s   remaining: 1m 7s
53:     learn: 0.5936881        total: 1m 17s   remaining: 1m 6s
54:     learn: 0.5900777        total: 1m 19s   remaining: 1m 4s
55:     learn: 0.5865426        total: 1m 20s   remaining: 1m 3s
56:     learn: 0.5855877        total: 1m 22s   remaining: 1m 1s
57:     learn: 0.5845208        total: 1m 23s   remaining: 1m
58:     learn: 0.5833049        total: 1m 24s   remaining: 59s
59:     learn: 0.5827105        total: 1m 26s   remaining: 57.7s
60:     learn: 0.5816953        total: 1m 28s   remaining: 56.8s
61:     learn: 0.5803613        total: 1m 33s   remaining: 57.1s
62:     learn: 0.5782501        total: 1m 35s   remaining: 55.9s
63:     learn: 0.5756970        total: 1m 37s   remaining: 54.7s
64:     learn: 0.5733750        total: 1m 38s   remaining: 53.3s
65:     learn: 0.5717652        total: 1m 40s   remaining: 51.7s
66:     learn: 0.5711472        total: 1m 42s   remaining: 50.3s
67:     learn: 0.5689667        total: 1m 43s   remaining: 48.8s
68:     learn: 0.5683983        total: 1m 45s   remaining: 47.3s
69:     learn: 0.5670043        total: 1m 47s   remaining: 45.9s
70:     learn: 0.5663849        total: 1m 48s   remaining: 44.4s
71:     learn: 0.5638185        total: 1m 50s   remaining: 42.8s
72:     learn: 0.5609397        total: 1m 51s   remaining: 41.3s
73:     learn: 0.5605175        total: 1m 53s   remaining: 39.8s
74:     learn: 0.5601120        total: 1m 54s   remaining: 38.2s
75:     learn: 0.5580418        total: 1m 55s   remaining: 36.6s
76:     learn: 0.5566895        total: 1m 58s   remaining: 35.3s
77:     learn: 0.5560560        total: 2m       remaining: 34s
78:     learn: 0.5552949        total: 2m 2s    remaining: 32.6s
79:     learn: 0.5545130        total: 2m 5s    remaining: 31.3s
80:     learn: 0.5538048        total: 2m 6s    remaining: 29.8s
81:     learn: 0.5496052        total: 2m 8s    remaining: 28.2s
82:     learn: 0.5491877        total: 2m 10s   remaining: 26.6s
83:     learn: 0.5474389        total: 2m 12s   remaining: 25.2s
84:     learn: 0.5464119        total: 2m 13s   remaining: 23.6s
85:     learn: 0.5459179        total: 2m 15s   remaining: 22.1s
86:     learn: 0.5441824        total: 2m 17s   remaining: 20.6s
87:     learn: 0.5431781        total: 2m 19s   remaining: 19s
88:     learn: 0.5425184        total: 2m 21s   remaining: 17.5s
89:     learn: 0.5416384        total: 2m 23s   remaining: 15.9s
90:     learn: 0.5404718        total: 2m 24s   remaining: 14.3s
91:     learn: 0.5362668        total: 2m 26s   remaining: 12.7s
92:     learn: 0.5352288        total: 2m 28s   remaining: 11.2s
93:     learn: 0.5342561        total: 2m 30s   remaining: 9.57s
94:     learn: 0.5322217        total: 2m 31s   remaining: 7.97s
95:     learn: 0.5316609        total: 2m 33s   remaining: 6.39s
96:     learn: 0.5311609        total: 2m 35s   remaining: 4.79s
```

```
97:     learn: 0.5301761        total: 2m 36s   remaining: 3.19s
98:     learn: 0.5296231        total: 2m 38s   remaining: 1.6s
99:     learn: 0.5275473        total: 2m 39s   remaining: 0us
Accuracy on testing set =  0.7148
RMSE on testing set =  0.7655063683601855
Classification Report:
               precision    recall  f1-score   support

         1.0       0.80      0.64      0.71       162
         2.0       0.57      0.36      0.44       154
         3.0       0.43      0.45      0.44       293
         4.0       0.58      0.62      0.60       606
         5.0       0.85      0.87      0.86      1285

    accuracy                           0.71      2500
   macro avg       0.65      0.59      0.61      2500
weighted avg       0.72      0.71      0.71      2500
```



Confusion matrix of the classifier

	0	1	2	3	4
0	0.64	0.093	0.15	0.074	0.043
1	0.091	0.36	0.29	0.2	0.058
2	0.017	0.068	0.45	0.34	0.12
3	0.005	0.0099	0.13	0.62	0.23
4	0.0031	0.00078	0.022	0.11	0.87

0.5 Create the Kaggle submission

```
[ ]: from scipy.sparse import csr_matrix

tfidf_vectorizer = TfidfVectorizer(max_df=0.5)
X_text_train_tfidf = tfidf_vectorizer.fit_transform(X_text_train)
X_text_test_tfidf = tfidf_vectorizer.transform(X_text_test)

X_submission = pd.read_csv("./data/X_test.csv")

X_submission_tfidf = tfidf_vectorizer.transform(X_submission['Text'])
X_submission_tfidf = csr_matrix(X_submission_tfidf)

X_submission_other = X_submission[['HelpfulnessNumerator',␣
 ↪'HelpfulnessDenominator','PositivityScore','ColemanLiauIndex','ReviewLength','AvgScoreByUser
X_submission = hstack([X_submission_tfidf, X_submission_other])

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_submission = scaler.transform(X_submission)

submission_predictions = model.predict(X_submission)

submission_predictions = submission_predictions.flatten()
submission = pd.DataFrame({'Id': X_submission['Id'], 'Score':␣
 ↪submission_predictions})

submission.to_csv("./data/submission.csv", index=False)
print("Submission saved to submission.csv.")
```

Now you can upload the submission.csv to kaggle