In [14]:
```
import numpy as np
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy import sparse
from sklearn.metrics import accuracy_score, confusion_matrix, mean_squared_err
from sklearn.model_selection import GridSearchCV
```

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt

trainingSet = pd.read_csv("./data/train.csv")
testingSet = pd.read_csv("./data/test.csv")

print("train.csv shape is ", trainingSet.shape)
print("test.csv shape is ", testingSet.shape)

print()

print(trainingSet.head())
print()
print(testingSet.head())

print()

print(trainingSet.describe())

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

trainingSet['ProductId'].value_counts().nlargest(25).plot(kind='bar', legend=T
plt.title("Top 25 most rated Products")
plt.show()

trainingSet['ProductId'].value_counts().nsmallest(25).plot(kind='bar', legend=
plt.title("Top 25 least rated Products")
plt.show()

trainingSet['UserId'].value_counts().nlargest(25).plot(kind='bar', legend=True
plt.title("Top 25 Reviewers")
plt.show()

trainingSet['UserId'].value_counts().nsmallest(25).plot(kind='bar', legend=Tru
plt.title("Lowest 25 Reviewers")
plt.show()

trainingSet[['Score', 'HelpfulnessNumerator']].groupby('Score').mean().plot(ki
plt.title("Mean Helpfulness Numerator per Score")
plt.show()

trainingSet[['Score', 'ProductId']].groupby('ProductId').mean().nlargest(25, '
plt.title("Top 25 best rated Products")
plt.show()

trainingSet[['Score', 'ProductId']].groupby('ProductId').mean().nsmallest(25,
plt.title("Top 25 worst rated Products")
plt.show()

trainingSet[['Score', 'UserId']].groupby('UserId').mean().nlargest(25, 'Score'
plt.title("Top 25 kindest Reviewers")
plt.show()

trainingSet[['Score', 'UserId']].groupby('UserId').mean().nsmallest(25, 'Score
plt.title("Top 25 harshest Reviewers")
```

```
plt.show()

trainingSet[trainingSet['ProductId'].isin(trainingSet['ProductId'].value_count
plt.title("Mean of top 25 most rated Products")
plt.show()
```

```
C:\Users\Admin\anaconda3\lib\site-packages\pandas\core\computation\express
ions.py:20: UserWarning: Pandas requires version '2.7.3' or newer of 'nume
xpr' (version '2.7.1' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED

train.csv shape is  (139753, 9)
test.csv shape is  (17470, 2)

        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
```

In [3]:
```
trainX = pd.read_csv('data/X_train.csv')
testX = pd.read_csv('data/X_test.csv')
```

In [4]:
```
# Replace NaN values with an empty string in 'Text' and 'Summary' columns
trainX['Text'].fillna('', inplace=True)
testX['Text'].fillna('', inplace=True)
trainX['Summary'].fillna('', inplace=True)
testX['Summary'].fillna('', inplace=True)
```

In [5]:
```
import re
def preprocess_2(text):
    text = str(text)
    text = re.sub(r'[0-9]', "", text)
    return text
```

In [6]:
```
trainX['Summary'] = trainX['Summary'].apply(preprocess_2)
trainX['Text'] = trainX['Text'].apply(preprocess_2)
```

In [7]:
```
X = trainX.drop(columns=['Score'])
y = trainX['Score']-1
```

In [8]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ran
```

In [9]:
```
text_vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 3), stop_
summary_vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 3), st
```

In [10]:
```
x_train_text = text_vectorizer.fit_transform(x_train['Text'])
x_test_text = text_vectorizer.transform(x_test['Text'])
```

In [11]:
```
x_train_summary = summary_vectorizer.fit_transform(x_train['Summary'])
x_test_summary = summary_vectorizer.transform(x_test['Summary'])
```

In [12]:
```
x_train_stacked = sparse.hstack([x_train_text, x_train_summary])
x_test_stacked = sparse.hstack([x_test_text, x_test_summary])
```

In [ ]:
```
from xgboost import XGBRegressor
xgb_grid = XGBRegressor()

# Define the hyperparameters and their possible values for Grid Search
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    # Add more hyperparameters and their values as needed
}

# Create the Grid Search object
grid_search_xgb = GridSearchCV(estimator=xgb_grid, param_grid=param_grid, scor

# Fit the Grid Search to the data
grid_search_xgb.fit(x_train_stacked, y_train)
```

In [ ]:
```
best_params = grid_search_xgb.best_params_
best_xgb_model = grid_search_xgb.best_estimator_
```

In [ ]:
```
#grid search xgb model
y_test_preds = best_xgb_model.predict(x_test_svd)
print("RMSE on testing set = ", mean_squared_error(y_test, y_test_preds)**0.5)
```

In [47]:
```
xgb = XGBRegressor()
xgb.fit(x_train_svd, y_train)
```

Out[47]:
```
XGBRegressor

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=No
ne,
             enable_categorical=False, eval_metric=None, feature_types=No
ne,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=No
ne,
             max_cat_threshold=None, max_cat_to_onehot=None,
```

In [48]:
```
#xgb regressor
y_test_preds = xgb.predict(x_test_svd)
print("RMSE on testing set = ", mean_squared_error(y_test, y_test_preds)**0.5)
```

```
RMSE on testing set =  0.9338581347338172
```

In [52]:
```
from sklearn.linear_model import Lasso
lasso = Lasso()
lasso.fit(x_train_stacked, y_train)
```

Out[52]:
```
Lasso

Lasso()
```

In [53]:
```
#lasso regressiob
y_test_preds = lasso.predict(x_test_stacked)
print("RMSE on testing set = ", mean_squared_error(y_test, y_test_preds)**0.5)
```

```
RMSE on testing set =  1.1916252680217625
```

In [44]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train_svd, y_train)
```

Out[44]:
```
LinearRegression

LinearRegression()
```

In [45]:
```
#linear regression
y_test_preds = lr.predict(x_test_svd)
print("RMSE on testing set = ", mean_squared_error(y_test, y_test_preds)**0.5)
```

```
RMSE on testing set =  0.9540656315158077
```

In [49]:
```
from sklearn.linear_model import Ridge
ridge = Ridge()
ridge.fit(x_train_stacked, y_train)
```

Out[49]:
```
Ridge
Ridge()
```

In [51]:
```
#ridge regression
y_test_preds = ridge.predict(x_test_stacked)
print("RMSE on testing set = ", mean_squared_error(y_test, y_test_preds)**0.5)
```

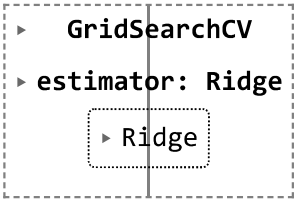
```
RMSE on testing set =  0.853371721000955
```

In [85]:
```
#grid search using ridge

param_grid = {
    'alpha': [0.1, 1.0, 10.0],
    'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga']
}
#creating the grid search
grid_search = GridSearchCV(estimator=ridge, param_grid=param_grid, cv=5, scori
#fitting the grid search to the train data
grid_search.fit(x_train_stacked, y_train)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\model_selection\_validatio
n.py:425: FitFailedWarning:
45 fits failed out of a total of 105.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting erro
r_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
15 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\model_selection\_v
alidation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\base.py", line 115
2, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_ridg
e.py", line 1131, in fit
    return super().fit(X, y, sample_weight=sample_weight)
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_ridg
e.py", line 823, in fit
    raise ValueError(
ValueError: solver='svd' does not support fitting the intercept on sparse dat
a. Please set the solver to 'auto' or 'lsqr', 'sparse_cg', 'sag', 'lbfgs' or
set `fit_intercept=False`

--------------------------------------------------------------------------------
15 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\model_selection\_v
alidation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\base.py", line 115
2, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_ridg
e.py", line 1131, in fit
    return super().fit(X, y, sample_weight=sample_weight)
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_ridg
e.py", line 823, in fit
    raise ValueError(
ValueError: solver='cholesky' does not support fitting the intercept on spars
e data. Please set the solver to 'auto' or 'lsqr', 'sparse_cg', 'sag', 'lbfg
s' or set `fit_intercept=False`

--------------------------------------------------------------------------------
15 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\model_selection\_v
alidation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\base.py", line 115
```

```
2, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_ridg
e.py", line 1131, in fit
    return super().fit(X, y, sample_weight=sample_weight)
  File "C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_ridg
e.py", line 823, in fit
    raise ValueError(
ValueError: solver='saga' does not support fitting the intercept on sparse da
ta. Please set the solver to 'auto' or 'lsqr', 'sparse_cg', 'sag', 'lbfgs' or
set `fit_intercept=False`

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:979: UserWarning: One or more of the test scores are non-finite: [-0.761284
03         nan         nan -0.76110944 -0.76128403 -0.76128403
         nan -0.73552904         nan         nan -0.73557645 -0.73552904
 -0.73552904         nan -0.71391604         nan         nan -0.71392304
 -0.71391604 -0.71391604         nan]
  warnings.warn(
```

Out[85]:



In [86]:
```
best_params = grid_search.best_params_
best_ridge_model = grid_search.best_estimator_
```

In [87]:
```
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'alpha': 10.0, 'solver': 'auto'}
```

In [88]:
```
y_pred_grid_search = best_ridge_model.predict(x_test_stacked)
print("RMSE on testing set = ", mean_squared_error(y_test, y_pred_grid_search))
```

```
RMSE on testing set =  0.843763386638613
```

In [77]:
```
import pickle
with open('ridge_movie_rating_v2.pkl', 'wb') as f:
    pickle.dump(best_ridge_model, f)
```

In [ ]:
```
# import pickle
# with open('xgb_new.pkl', 'wb') as f:
#     pickle.dump(xgb, f)
```

In [31]:
```
X_submission = pd.read_csv("./data/X_test.csv")
X_submission['Summary'] = X_submission['Summary'].apply(preprocess_2)
X_submission['Text'] = X_submission['Text'].apply(preprocess_2)
X_submission['Text'].fillna('', inplace=True)
X_submission['Summary'].fillna('', inplace=True)
```

In [32]:
```
X_submission.head()
```

Out[32]:

	Id	ProductId	UserId	HelpfulnessNumerator	HelpfulnessDenominator
0	786781	B0000VD02Y	A1UL8PS42M5DM8	1	7
1	17153	0767823931	A2OP1HD9RGX5OW	3	6
2	1557328	B008JFUNTG	AY113687D8YK1	1	8
3	1242666	B001UWOLQG	A2MVTAEGBP08RB	0	1
4	1359242	B003QS0E54	ALGAE0IGE4DBP	99	103

In [78]:
```
#creating the submission vectors using the text and summary tfidf vectorizers.
x_submission_text = text_vectorizer.transform(X_submission['Text'])
x_submission_summary = summary_vectorizer.transform(X_submission['Summary'])
```

In [79]:
```
#stacking the two vectors together
x_submission_stacked = sparse.hstack([x_submission_text, x_submission_summary]
```

In [80]:
```
#submission
submission = X_submission[['Id']].copy()
submission['Score'] = best_ridge_model.predict(x_submission_stacked) + 1  # Sh
submission.to_csv('./data/submission_v2.csv', index=False)
```