Q. Does user's demographic(location, age, gender) impact type of anime(genre, episodes, anime type(OVA, movie), Rank) they are prefer? For solving this question, I am trying to determine the rating of an anime based on user's demographic(Age, Location, Gender) and anime's features(genre, type and episodes)

The first algorithm I am trying is OrderedModel. I am trying to classify a given combination of user demographic and anime features to the probable rating by user. This is because rating is a multiclass column. Plain logistic regression would be a bit lacking to predict the expected rating. OrderedModel is more suitable for ordinal multiclass classification.

```
import pandas as pd
import numpy as np
from statsmodels.miscmodels.ordinal_model import OrderedModel
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer

data = pd.read_csv('../../joined_datasets/joined_rating_dataset.csv')
cleaned_dataset =
pd.read_csv("../../cleaned_datasets/users_details_dataset_cleaned.csv"
)
data.dropna()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5207686 entries, 0 to 5207685
Data columns (total 21 columns):
 #   Column            Dtype  
---  ------            -----  
 0   Unnamed: 0        int64  
 1   user_id           int64  
 2   anime_id          int64  
 3   rating            int64  
 4   Gender            object 
 5   Location          object 
 6   Birthday_Date     object 
 7   Joined_Date       object 
 8   Age_Join          float64
 9   Episodes Watched  float64
 10  Age               float64
 11  Name              object 
 12  Genres            object 
 13  Type              object 
 14  Start Date        object 
 15  End Date          object 
 16  Studios           object 
 17  Source            object 
 18  Rank              object 
 19  Episodes          float64
 20  Episodes_Norm     float64
```

```
dtypes: float64(5), int64(4), object(12)
memory usage: 834.4+ MB
```

Performing undersampling as number of rows was a bit too high to compute using my laptop. Considering only the top 10 countries

```
df = cleaned_dataset.copy()
import pandas as pd
import matplotlib.pyplot as plt

location_user_counts = df['Location'].value_counts()

top_countries = df['Location'].value_counts().head(10)

# # Get the list of top 20 countries
top_10_countries = top_countries.index.tolist()
data = data[data['Location'].isin(top_10_countries)]
data.info()

<class 'pandas.core.frame.DataFrame'>
Index: 3357313 entries, 0 to 5207685
Data columns (total 21 columns):
 #   Column            Dtype  
---  ------            -----  
 0   Unnamed: 0        int64  
 1   user_id           int64  
 2   anime_id          int64  
 3   rating            int64  
 4   Gender            object 
 5   Location          object 
 6   Birthday_Date     object 
 7   Joined_Date       object 
 8   Age_Join          float64
 9   Episodes Watched  float64
 10  Age               float64
 11  Name              object 
 12  Genres            object 
 13  Type              object 
 14  Start Date        object 
 15  End Date          object 
 16  Studios           object 
 17  Source            object 
 18  Rank              object 
 19  Episodes          float64
 20  Episodes_Norm     float64
dtypes: float64(5), int64(4), object(12)
memory usage: 563.5+ MB

# Undersampling
# Count entries for each location
```

```
location_counts = data['Location'].value_counts()

# Decide the target sample size (e.g., use the minimum count of the
top 5 locations)
target_sample_size = location_counts.nsmallest(5).min()  # Choose the
minimum of the top 5

# Under-sample the DataFrame
under_sampled_df = data.groupby('Location').apply(lambda x:
x.sample(n=min(len(x), target_sample_size),
random_state=42)).reset_index(drop=True)

# Count occurrences of each class in the target variable (assuming
your target variable is 'rating')
target_counts = under_sampled_df['Location'].value_counts()
# Output the results
print("Under-sampled DataFrame shape:", under_sampled_df.shape)
print("Counts of each class in the target variable:")
print(target_counts)
```

```
/var/folders/dz/fg9tl53x4y16ytgmdhwdt0kr0000gn/T/
ipykernel_85581/807053095.py:9: DeprecationWarning:
DataFrameGroupBy.apply operated on the grouping columns. This behavior
is deprecated, and in a future version of pandas the grouping columns
will be excluded from the operation. Either pass
`include_groups=False` to exclude the groupings or explicitly select
the grouping columns after groupby to silence this warning.
  under_sampled_df = data.groupby('Location').apply(lambda x:
x.sample(n=min(len(x), target_sample_size),
random_state=42)).reset_index(drop=True)
```

```
Under-sampled DataFrame shape: (1075590, 21)
Counts of each class in the target variable:
Location
Australia        107559
Brazil           107559
Canada           107559
France           107559
Germany          107559
Philippines      107559
Poland           107559
Russia           107559
Sweden           107559
United States    107559
Name: count, dtype: int64
```

Checking if more undersampling is needed based on ratings.

```
from collections import Counter
```

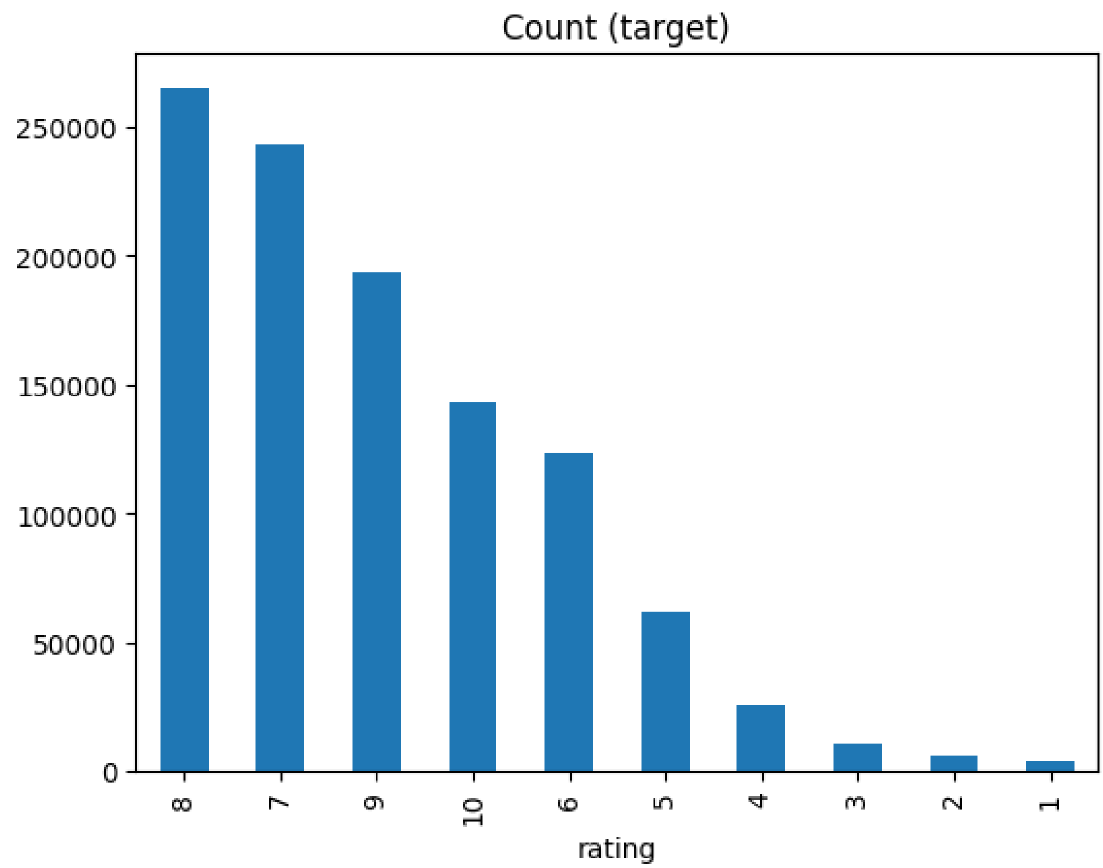
```
counter = Counter(under_sampled_df["rating"])

for k, v in counter.items():
    per = 100*v/len(under_sampled_df["rating"])
    print(f"Class = {k}, n={v} ({per:.2f}%)")

target_count = under_sampled_df["rating"].value_counts()
target_count.plot(kind='bar', title='Count (target)')
```

```
Class = 10, n=142811 (13.28%)
Class = 9, n=193198 (17.96%)
Class = 6, n=123165 (11.45%)
Class = 8, n=264913 (24.63%)
Class = 7, n=243352 (22.62%)
Class = 5, n=61613 (5.73%)
Class = 4, n=25777 (2.40%)
Class = 2, n=5855 (0.54%)
Class = 3, n=10901 (1.01%)
Class = 1, n=4005 (0.37%)

<Axes: title={'center': 'Count (target)'}, xlabel='rating'>
```




```
data.columns
```

```
Index(['Unnamed: 0', 'user_id', 'anime_id', 'rating', 'Gender',
'Location',
       'Birthday_Date', 'Joined_Date', 'Age_Join', 'Episodes Watched',
'Age',
       'Name', 'Genres', 'Type', 'Start Date', 'End Date', 'Studios',
'Source',
       'Rank', 'Episodes', 'Episodes_Norm'],
      dtype='object')
```

Converting the columns with string data to 1 hot encoded columns. Location to multiple columns with country name, gender to male, non-binary, genres to Adventure etc. Splitting genres was a bit more tricky as there are multiple genres in each entry of genres column.

```
# Select features for the model - adjust based on your dataset
features = ['Gender', 'Location', 'Age', 'Genres', 'Type', 'Episodes',
'Rank']
target = 'rating'

# One-hot encode categorical features
df = pd.get_dummies(under_sampled_df, columns=['Gender', 'Location',
'Type'],drop_first=True)


# Split the 'Genres' column into lists of individual genres
df['Genres'] = df['Genres'].str.strip().str.replace(', ',
',').str.replace(' ,', ',').str.split(',')

# Initialize MultiLabelBinarizer
mlb = MultiLabelBinarizer()

# One-hot encode the genres and create a DataFrame with separate
columns for each genre
genres_one_hot = pd.DataFrame(mlb.fit_transform(df['Genres']),
columns=mlb.classes_, index=df.index)
genres_one_hot = genres_one_hot.add_prefix("Genre_")


# Concatenate the one-hot encoded genres back to the original
DataFrame and drop the original 'Genres' column
df = pd.concat([df, genres_one_hot], axis=1).drop('Genres', axis=1)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 808423 entries, 0 to 1075589
Data columns (total 52 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   Unnamed: 0                808423 non-null  float64
 1   user_id                   808423 non-null  float64
 2   anime_id                  808423 non-null  float64
 3   rating                    808423 non-null  float64
```

```
 4   Birthday_Date            808423 non-null  object
 5   Joined_Date              808423 non-null  object
 6   Age_Join                 808423 non-null  float64
 7   Episodes Watched         808423 non-null  float64
 8   Age                      808423 non-null  float64
 9   Name                     808423 non-null  object
 10  Start Date               808423 non-null  object
 11  End Date                 808423 non-null  object
 12  Studios                  808423 non-null  object
 13  Source                   808423 non-null  object
 14  Rank                     808423 non-null  object
 15  Episodes                 808423 non-null  float64
 16  Episodes_Norm            808423 non-null  float64
 17  Gender_Male              808423 non-null  float64
 18  Gender_Non-Binary        808423 non-null  float64
 19  Location_Brazil          808423 non-null  float64
 20  Location_Canada          808423 non-null  float64
 21  Location_France          808423 non-null  float64
 22  Location_Germany         808423 non-null  float64
 23  Location_Philippines     808423 non-null  float64
 24  Location_Poland          808423 non-null  float64
 25  Location_Russia          808423 non-null  float64
 26  Location_Sweden          808423 non-null  float64
 27  Location_United States   808423 non-null  float64
 28  Type_Music               808423 non-null  float64
 29  Type_ONA                 808423 non-null  float64
 30  Type_OVA                 808423 non-null  float64
 31  Type_Special             808423 non-null  float64
 32  Type_TV                  808423 non-null  float64
 33  Genre_Action             808423 non-null  float64
 34  Genre_Adventure          808423 non-null  float64
 35  Genre_Avant Garde        808423 non-null  float64
 36  Genre_Award Winning      808423 non-null  float64
 37  Genre_Boys Love          808423 non-null  float64
 38  Genre_Comedy             808423 non-null  float64
 39  Genre_Drama              808423 non-null  float64
 40  Genre_Ecchi              808423 non-null  float64
 41  Genre_Fantasy            808423 non-null  float64
 42  Genre_Girls Love         808423 non-null  float64
 43  Genre_Gourmet            808423 non-null  float64
 44  Genre_Horror             808423 non-null  float64
 45  Genre_Mystery            808423 non-null  float64
 46  Genre_Romance            808423 non-null  float64
 47  Genre_Sci-Fi             808423 non-null  float64
 48  Genre_Slice of Life      808423 non-null  float64
 49  Genre_Sports             808423 non-null  float64
 50  Genre_Supernatural       808423 non-null  float64
 51  Genre_Suspense           808423 non-null  float64
```

```
dtypes: float64(44), object(8)
memory usage: 326.9+ MB
```

```
df = df.astype({col: 'float' for col in
df.select_dtypes(include=['bool']).columns})
df = df.astype({col: 'float' for col in
df.select_dtypes(include=['int64', 'float64']).columns})
df = df.dropna()
df = df[df["Rank"]!="UNKNOWN"]
```

Dropping columns with no data

```
genre_columns = [col for col in df.columns if
col.startswith("Genre_")]


genre_counts = df[genre_columns].sum()


zero_count_genres = genre_counts[genre_counts == 0].index
zero_count_genres
df = df.drop(columns=zero_count_genres)
```

Dropping type columns with no data

```
type_columns = [col for col in df.columns if col.startswith("Type_")]


type_counts = df[type_columns].sum()


zero_count_types = type_counts[type_counts == 0].index
df = df.drop(columns=zero_count_types)
```

Standardizing the columns with high variance data, to a small range(0-1)

```
# Standardize the features
features = df.columns.difference(['user_id', 'anime_id', 'rating',
'Birthday_Date', 'Joined_Date', 'Age_Join', 'Episodes Watched',
                                  'Start Date', 'End Date', 'Name',
'Rank', 'Studios', 'Source', 'Episodes_Norm', 'Unnamed: 0'])
from sklearn.preprocessing import LabelEncoder, StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df[features])



# # Split the data
X = df_scaled
```

```
y = df[target]

y = y.astype('category')


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


# Fit an Ordinal Logistic Regression model
model = OrderedModel(y_train, X_train, distr='logit')
model_fit = model.fit(method='bfgs')
print(model_fit.summary())

# Coefficient significance
print(model_fit.pvalues)
print(model_fit.conf_int())
```

```
/Users/ramachandrank/Repos/MS/sem1/.venv/lib/python3.12/site-
packages/statsmodels/miscmodels/ordinal_model.py:205: Warning: the
endog has ordered == False, risk of capturing a wrong order for the
categories. ordered == True preferred.
  warnings.warn("the endog has ordered == False, "

Optimization terminated successfully.
         Current function value: 1.809041
         Iterations: 76
         Function evaluations: 77
         Gradient evaluations: 77
                             OrderedModel Results

======================================================================
========
Dep. Variable:                 rating   Log-Likelihood:            -
1.1700e+06
Model:                   OrderedModel   AIC:
2.340e+06
Method:            Maximum Likelihood   BIC:
2.341e+06
Date:                Tue, 05 Nov 2024

Time:                        01:05:53

No. Observations:              646738

Df Residuals:                  646693

Df Model:                          36
```

```
==============================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
--------
x1            -0.0982      0.002    -42.858      0.000      -0.103
-0.094
x2             0.0701      0.003     27.653      0.000       0.065
0.075
x3            -0.0363      0.002    -15.650      0.000      -0.041
-0.032
x4            -0.0243      0.002    -10.639      0.000      -0.029
-0.020
x5             0.0275      0.003      9.927      0.000       0.022
0.033
x6             0.0138      0.003      5.198      0.000       0.009
0.019
x7            -0.0196      0.002     -8.279      0.000      -0.024
-0.015
x8             0.1707      0.002     71.253      0.000       0.166
0.175
x9            -0.0328      0.002    -14.029      0.000      -0.037
-0.028
x10           -0.0252      0.003     -9.267      0.000      -0.030
-0.020
x11            0.0896      0.002     36.281      0.000       0.085
0.094
x12           -0.1601      0.002    -66.880      0.000      -0.165
-0.155
x13           -0.0662      0.003    -24.345      0.000      -0.071
-0.061
x14           -0.0462      0.002    -20.985      0.000      -0.051
-0.042
x15           -0.0067      0.002     -3.064      0.002      -0.011
-0.002
x16           -0.0536      0.002    -22.364      0.000      -0.058
-0.049
x17            0.0138      0.002      5.616      0.000       0.009
0.019
x18           -0.0108      0.003     -4.180      0.000      -0.016
-0.006
x19           -0.0712      0.003    -28.282      0.000      -0.076
-0.066
x20            0.0007      0.002      0.305      0.761      -0.004
0.005
x21            0.0276      0.002     11.525      0.000       0.023
0.032
x22            0.0557      0.003     21.536      0.000       0.051
```

```
0.061
x23             0.0901      0.002     37.126      0.000       0.085
0.095
x24             0.0997      0.003     33.413      0.000       0.094
0.106
x25             0.0049      0.003      1.679      0.093      -0.001
0.011
x26            -0.1212      0.003    -41.186      0.000      -0.127
-0.115
x27            -0.0015      0.003     -0.523      0.601      -0.007
0.004
x28             0.2139      0.003     71.678      0.000       0.208
0.220
x29             0.0054      0.003      1.809      0.070      -0.000
0.011
x30            -0.0021      0.003     -0.722      0.470      -0.008
0.004
x31            -0.0591      0.003    -20.133      0.000      -0.065
-0.053
x32             0.0511      0.003     17.317      0.000       0.045
0.057
x33             0.0819      0.022      3.793      0.000       0.040
0.124
x34             0.2518      0.046      5.474      0.000       0.162
0.342
x35             0.1143      0.031      3.744      0.000       0.054
0.174
x36             0.3265      0.057      5.683      0.000       0.214
0.439
1.0/2.0        -5.7747      0.021   -272.373      0.000      -5.816
-5.733
2.0/3.0        -0.0874      0.018     -4.879      0.000      -0.122
-0.052
3.0/4.0        -0.2641      0.013    -20.543      0.000      -0.289
-0.239
4.0/5.0        -0.1622      0.008    -19.599      0.000      -0.178
-0.146
5.0/6.0        -0.0942      0.005    -17.702      0.000      -0.105
-0.084
6.0/7.0        -0.0924      0.004    -25.372      0.000      -0.100
-0.085
7.0/8.0         0.1040      0.002     42.815      0.000       0.099
0.109
8.0/9.0         0.0904      0.002     40.083      0.000       0.086
0.095
9.0/10.0        0.1427      0.003     52.318      0.000       0.137
0.148
======================================================================
========
```

```
x1            0.000000e+00
x2          2.577699e-168
x3            3.321227e-55
x4            1.955738e-26
x5            3.176869e-23
x6            2.012625e-07
x7            1.246599e-16
x8            0.000000e+00
x9            1.029523e-44
x10           1.920620e-20
x11         3.235854e-288
x12           0.000000e+00
x13         6.571681e-131
x14           8.905023e-98
x15           2.182299e-03
x16         8.816428e-111
x17           1.959507e-08
x18           2.920828e-05
x19         5.724506e-176
x20           7.606672e-01
x21           9.828134e-31
x22         7.141908e-103
x23         1.054674e-301
x24         8.738269e-245
x25           9.317266e-02
x26           0.000000e+00
x27           6.010196e-01
x28           0.000000e+00
x29           7.049801e-02
x30           4.700492e-01
x31           3.813837e-90
x32           3.520680e-67
x33           1.487744e-04
x34           4.396056e-08
x35           1.808022e-04
x36           1.323051e-08
1.0/2.0       0.000000e+00
2.0/3.0       1.067125e-06
3.0/4.0       8.949131e-94
4.0/5.0       1.588290e-85
5.0/6.0       4.073080e-70
6.0/7.0     5.102751e-142
7.0/8.0       0.000000e+00
8.0/9.0       0.000000e+00
9.0/10.0      0.000000e+00
dtype: float64
                 0         1
x1       -0.102683 -0.093702
x2        0.065154  0.075095
```

```
x3        -0.040892 -0.031789
x4        -0.028831 -0.019861
x5         0.022085  0.032951
x6         0.008585  0.018977
x7        -0.024244 -0.014962
x8         0.166049  0.175442
x9        -0.037368 -0.028207
x10       -0.030481 -0.019838
x11        0.084786  0.094470
x12       -0.164840 -0.155453
x13       -0.071488 -0.060835
x14       -0.050527 -0.041895
x15       -0.011061 -0.002431
x16       -0.058337 -0.048936
x17        0.008960  0.018567
x18       -0.015927 -0.005758
x19       -0.076156 -0.066284
x20       -0.003877  0.005304
x21        0.022886  0.032264
x22        0.050661  0.060805
x23        0.085310  0.094819
x24        0.093861  0.105559
x25       -0.000828  0.010715
x26       -0.126932 -0.115400
x27       -0.007290  0.004219
x28        0.208073  0.219772
x29       -0.000447  0.011149
x30       -0.007860  0.003626
x31       -0.064889 -0.053376
x32        0.045356  0.056933
x33        0.039603  0.124290
x34        0.161626  0.341912
x35        0.054466  0.174110
x36        0.213920  0.439149
1.0/2.0   -5.816211 -5.733103
2.0/3.0   -0.122499 -0.052284
3.0/4.0   -0.289350 -0.238945
4.0/5.0   -0.178450 -0.146003
5.0/6.0   -0.104648 -0.083784
6.0/7.0   -0.099545 -0.085268
7.0/8.0    0.099216  0.108735
8.0/9.0    0.085951  0.094788
9.0/10.0   0.137371  0.148065
```

It seems the accuracy is not very high. This is probably due to the difference in the number of ratings for higher values(>5) than ratings(<5>)

```
import numpy as np
from sklearn.metrics import mean_squared_error, accuracy_score
```

```python
# Predict probabilities on the test set
predicted_probs = model_fit.predict(X_test)

# Convert probabilities to predicted ratings by selecting the category
with the highest probability
predicted_ratings = np.argmax(predicted_probs, axis=1)

y_test = y_test.cat.codes

# Calculate evaluation metrics
mse = mean_squared_error(y_test, predicted_ratings)
accuracy = accuracy_score(y_test, predicted_ratings)

print("Mean Squared Error:", mse)
print("Accuracy:", accuracy)
```

```
Mean Squared Error: 2.736833967282061
Accuracy: 0.260828153508365
```

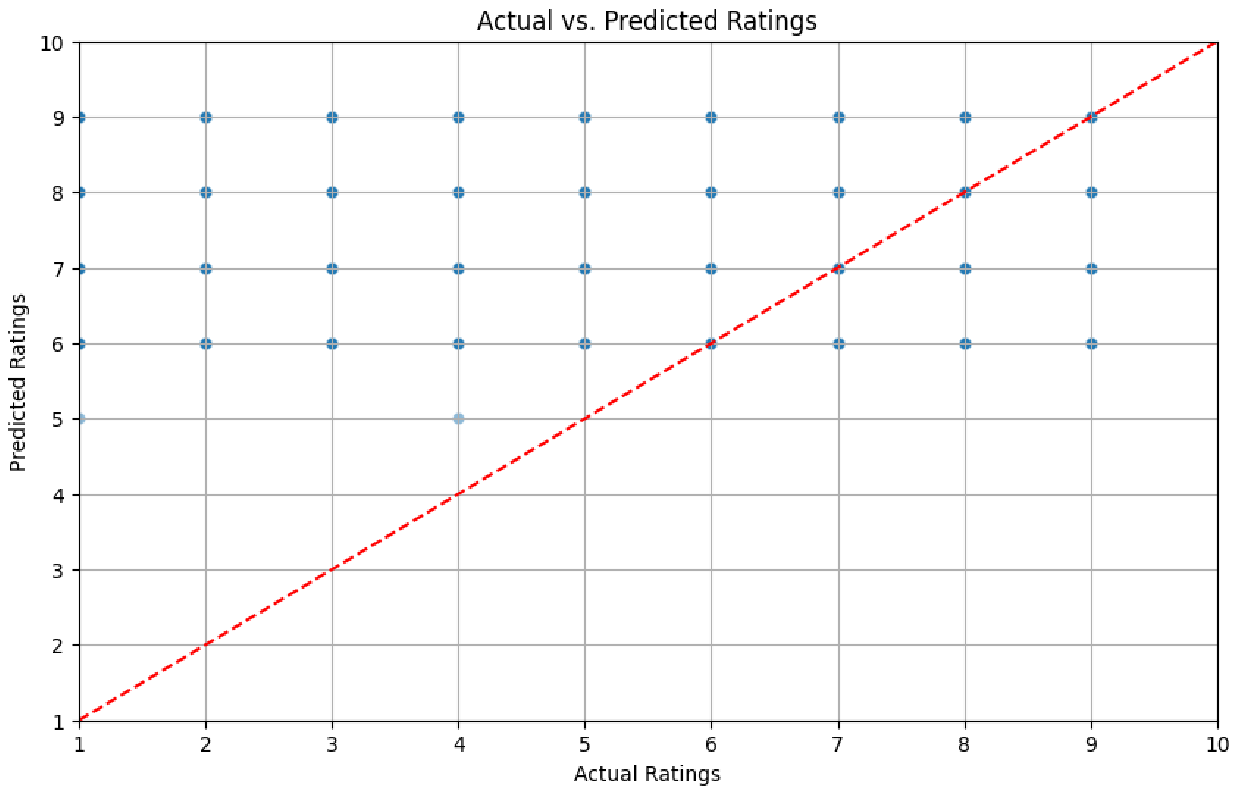
```python
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=predicted_ratings, alpha=0.5)

plt.plot([1, 10], [1, 10], color='red', linestyle='--')


plt.title('Actual vs. Predicted Ratings')
plt.xlabel('Actual Ratings')
plt.ylabel('Predicted Ratings')
plt.xlim(1, 10)
plt.ylim(1, 10)
plt.xticks(range(1, 11))
plt.yticks(range(1, 11))

# Show plot
plt.grid()
plt.show()
```



Undersampling the data based on rating to reduce the difference in number of ratings

```
rating_counts = df['rating'].value_counts()


target_sample_size = rating_counts.nlargest(5).min()


rating_under_sampled_df = df.groupby('rating').apply(lambda x:
x.sample(n=min(len(x), target_sample_size),
random_state=42)).reset_index(drop=True)


target_counts = rating_under_sampled_df['rating'].value_counts()

print("Under-sampled DataFrame shape:", rating_under_sampled_df.shape)
print("Counts of each class in the target variable:")
print(target_counts)
```

```
Under-sampled DataFrame shape: (521841, 51)
Counts of each class in the target variable:
rating
6.0     88831
7.0     88831
8.0     88831
9.0     88831
```

```
10.0    88831
5.0     44137
4.0     18756
3.0      7887
2.0      4136
1.0      2770
Name: count, dtype: int64

/var/folders/dz/fg9tl53x4y16ytgmdhwdt0kr0000gn/T/
ipykernel_85581/1598312855.py:7: DeprecationWarning:
DataFrameGroupBy.apply operated on the grouping columns. This behavior
is deprecated, and in a future version of pandas the grouping columns
will be excluded from the operation. Either pass
`include_groups=False` to exclude the groupings or explicitly select
the grouping columns after groupby to silence this warning.
  rating_under_sampled_df = df.groupby('rating').apply(lambda x:
x.sample(n=min(len(x), target_sample_size),
random_state=42)).reset_index(drop=True)
```

Second iteration with undersampled rating data

```
# Standardize the features
features = rating_under_sampled_df.columns.difference(['user_id',
'anime_id', 'rating', 'Birthday_Date', 'Joined_Date', 'Age_Join',
'Episodes Watched',
                                  'Start Date', 'End Date', 'Name',
'Rank', 'Studios', 'Source', 'Episodes_Norm', 'Unnamed: 0'])


# # Split the data
X = rating_under_sampled_df[features]

y = rating_under_sampled_df[target]

# Ordinal Logistic Regression expects integer targets as levels
y = y.astype('category')

# # Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Fit an Ordinal Logistic Regression model
model_under_rating = OrderedModel(y_train, X_train, distr='logit')
model_under_rating_fit = model_under_rating.fit(method='bfgs')
print(model_under_rating_fit.summary())

# Coefficient significance
print(model_under_rating_fit.pvalues)
print(model_under_rating_fit.conf_int())
```

```
/Users/ramachandrank/Repos/MS/sem1/.venv/lib/python3.12/site-
packages/statsmodels/miscmodels/ordinal_model.py:205: Warning: the
endog has ordered == False, risk of capturing a wrong order for the
categories. ordered == True preferred.
  warnings.warn("the endog has ordered == False, "

Optimization terminated successfully.
         Current function value: 1.921322
         Iterations: 205
         Function evaluations: 210
         Gradient evaluations: 210
                             OrderedModel Results

======================================================================
========
Dep. Variable:                 rating   Log-Likelihood:            -
8.0210e+05
Model:                   OrderedModel   AIC:
1.604e+06
Method:            Maximum Likelihood   BIC:
1.605e+06
Date:                Tue, 05 Nov 2024

Time:                        02:19:00

No. Observations:              417472

Df Residuals:                  417427

Df Model:                          36

======================================================================
====================
                             coef    std err          z      P>|z|
[0.025      0.975]
----------------------------------------------------------------------
--------------------
Age                       -0.0247      0.001    -36.055      0.000
-0.026      -0.023
Episodes                   0.0015   6.44e-05     23.744      0.000
0.001       0.002
Gender_Male               -0.0734      0.006    -12.010      0.000
-0.085      -0.061
Gender_Non-Binary         -0.3315      0.033     -9.961      0.000
-0.397      -0.266
Genre_Action               0.0646      0.007      9.348      0.000
0.051       0.078
Genre_Adventure            0.0407      0.008      4.906      0.000
0.024       0.057
Genre_Avant Garde         -0.2243      0.032     -7.093      0.000
```

```
-0.286      -0.162
Genre_Award Winning          0.8839      0.014     62.960      0.000
0.856       0.911
Genre_Boys Love             -0.4206      0.028    -14.848      0.000
-0.476      -0.365
Genre_Comedy                -0.0372      0.007     -5.471      0.000
-0.050      -0.024
Genre_Drama                  0.2091      0.007     30.387      0.000
0.196       0.223
Genre_Ecchi                 -0.5282      0.009    -59.781      0.000
-0.546      -0.511
Genre_Fantasy               -0.1668      0.008    -21.968      0.000
-0.182      -0.152
Genre_Girls Love            -0.4568      0.025    -17.968      0.000
-0.507      -0.407
Genre_Gourmet               -0.0705      0.031     -2.259      0.024
-0.132      -0.009
Genre_Horror                -0.2788      0.013    -21.321      0.000
-0.304      -0.253
Genre_Mystery                0.0442      0.009      4.728      0.000
0.026       0.062
Genre_Romance               -0.0192      0.007     -2.782      0.005
-0.033      -0.006
Genre_Sci-Fi                -0.1965      0.008    -25.039      0.000
-0.212      -0.181
Genre_Slice of Life          0.0237      0.013      1.840      0.066
-0.002       0.049
Genre_Sports                 0.1740      0.017     10.308      0.000
0.141       0.207
Genre_Supernatural           0.1495      0.008     18.768      0.000
0.134       0.165
Genre_Suspense               0.4901      0.014     34.748      0.000
0.462       0.518
Location_Brazil              0.3828      0.012     31.132      0.000
0.359       0.407
Location_Canada              0.0645      0.012      5.293      0.000
0.041       0.088
Location_France             -0.4132      0.012    -34.098      0.000
-0.437      -0.389
Location_Germany             0.0018      0.012      0.148      0.882
-0.022       0.026
Location_Philippines         0.8435      0.012     68.816      0.000
0.819       0.867
Location_Poland              0.0393      0.012      3.194      0.001
0.015       0.063
Location_Russia              0.0039      0.012      0.316      0.752
-0.020       0.028
Location_Sweden             -0.1822      0.012    -15.093      0.000
-0.206      -0.159
```

```
Location_United States      0.2110      0.012     17.251      0.000
0.187       0.235
Type_ONA                    0.3814      0.200      1.907      0.057
-0.011       0.773
Type_OVA                    0.7052      0.199      3.545      0.000
0.315       1.095
Type_Special                0.3728      0.199      1.871      0.061
-0.018       0.763
Type_TV                     0.7456      0.199      3.753      0.000
0.356       1.135
1.0/2.0                    -5.3759      0.201    -26.723      0.000
-5.770      -4.982
2.0/3.0                    -0.0690      0.018     -3.860      0.000
-0.104      -0.034
3.0/4.0                    -0.2473      0.013    -19.321      0.000
-0.272      -0.222
4.0/5.0                    -0.1396      0.008    -17.000      0.000
-0.156      -0.123
5.0/6.0                    -0.0503      0.005     -9.572      0.000
-0.061      -0.040
6.0/7.0                     0.0244      0.004      6.893      0.000
0.017       0.031
7.0/8.0                    -0.2750      0.003    -78.730      0.000
-0.282      -0.268
8.0/9.0                    -0.2826      0.003    -80.926      0.000
-0.289      -0.276
9.0/10.0                   -0.0231      0.004     -6.544      0.000
-0.030      -0.016
======================================================================
====================
Age                        1.169237e-284
Episodes                   1.257567e-124
Gender_Male                 3.141376e-33
Gender_Non-Binary           2.249364e-23
Genre_Action                8.926337e-21
Genre_Adventure             9.297422e-07
Genre_Avant Garde           1.315083e-12
Genre_Award Winning         0.000000e+00
Genre_Boys Love             7.129536e-50
Genre_Comedy                4.486878e-08
Genre_Drama                8.231411e-203
Genre_Ecchi                 0.000000e+00
Genre_Fantasy              5.862816e-107
Genre_Girls Love            3.448285e-72
Genre_Gourmet               2.388877e-02
Genre_Horror               7.316307e-101
Genre_Mystery               2.267776e-06
Genre_Romance               5.401458e-03
Genre_Sci-Fi               2.277079e-138
```

```
Genre_Slice of Life        6.571469e-02
Genre_Sports               6.507695e-25
Genre_Supernatural         1.381168e-78
Genre_Suspense            1.469830e-264
Location_Brazil           8.905444e-213
Location_Canada            1.204592e-07
Location_France           7.868142e-255
Location_Germany           8.824292e-01
Location_Philippines       0.000000e+00
Location_Poland            1.402949e-03
Location_Russia            7.523721e-01
Location_Sweden            1.794018e-51
Location_United States     1.107131e-66
Type_ONA                   5.654060e-02
Type_OVA                   3.923281e-04
Type_Special               6.136839e-02
Type_TV                    1.747418e-04
1.0/2.0                   2.527731e-157
2.0/3.0                    1.135987e-04
3.0/4.0                    3.603350e-83
4.0/5.0                    8.244007e-65
5.0/6.0                    1.053507e-21
6.0/7.0                    5.466686e-12
7.0/8.0                    0.000000e+00
8.0/9.0                    0.000000e+00
9.0/10.0                   5.998021e-11
dtype: float64
                                0         1
Age                     -0.026047 -0.023361
Episodes                 0.001404  0.001657
Gender_Male             -0.085393 -0.061432
Gender_Non-Binary       -0.396676 -0.266242
Genre_Action             0.051034  0.078111
Genre_Adventure          0.024451  0.056985
Genre_Avant Garde       -0.286247 -0.162298
Genre_Award Winning      0.856408  0.911442
Genre_Boys Love         -0.476156 -0.365110
Genre_Comedy            -0.050486 -0.023852
Genre_Drama              0.195632  0.222609
Genre_Ecchi             -0.545526 -0.510891
Genre_Fantasy           -0.181644 -0.151887
Genre_Girls Love        -0.506618 -0.406965
Genre_Gourmet           -0.131694 -0.009332
Genre_Horror            -0.304466 -0.253201
Genre_Mystery            0.025864  0.062492
Genre_Romance           -0.032701 -0.005669
Genre_Sci-Fi            -0.211833 -0.181078
Genre_Slice of Life     -0.001538  0.048861
Genre_Sports             0.140917  0.207090
```

```
Genre_Supernatural        0.133878  0.165101
Genre_Suspense            0.462463  0.517751
Location_Brazil           0.358706  0.406906
Location_Canada           0.040638  0.088434
Location_France          -0.436949 -0.389447
Location_Germany         -0.022208  0.025833
Location_Philippines      0.819436  0.867482
Location_Poland           0.015194  0.063458
Location_Russia          -0.020302  0.028092
Location_Sweden          -0.205896 -0.158568
Location_United States    0.187005  0.234945
Type_ONA                 -0.010624  0.773351
Type_OVA                  0.315323  1.095049
Type_Special             -0.017761  0.763361
Type_TV                   0.356205  1.134945
1.0/2.0                  -5.770132 -4.981570
2.0/3.0                  -0.104109 -0.033983
3.0/4.0                  -0.272350 -0.222182
4.0/5.0                  -0.155652 -0.123471
5.0/6.0                  -0.060540 -0.039960
6.0/7.0                   0.017459  0.031332
7.0/8.0                  -0.281800 -0.268110
8.0/9.0                  -0.289480 -0.275790
9.0/10.0                 -0.029995 -0.016169
```

```python
# Predict probabilities on the test set
predicted_probs = model_under_rating_fit.predict(X_test)

# Convert probabilities to predicted ratings by selecting the category
with the highest probability
predicted_ratings = np.argmax(predicted_probs, axis=1)

y_test = y_test.astype('category').cat.codes + 1

# Calculate evaluation metrics
mse = mean_squared_error(y_test, predicted_ratings)
accuracy = accuracy_score(y_test, predicted_ratings)

print("Mean Squared Error:", mse)
print("Accuracy:", accuracy)
```

```
Mean Squared Error: 5.318926117908575
Accuracy: 0.163640544606157
```

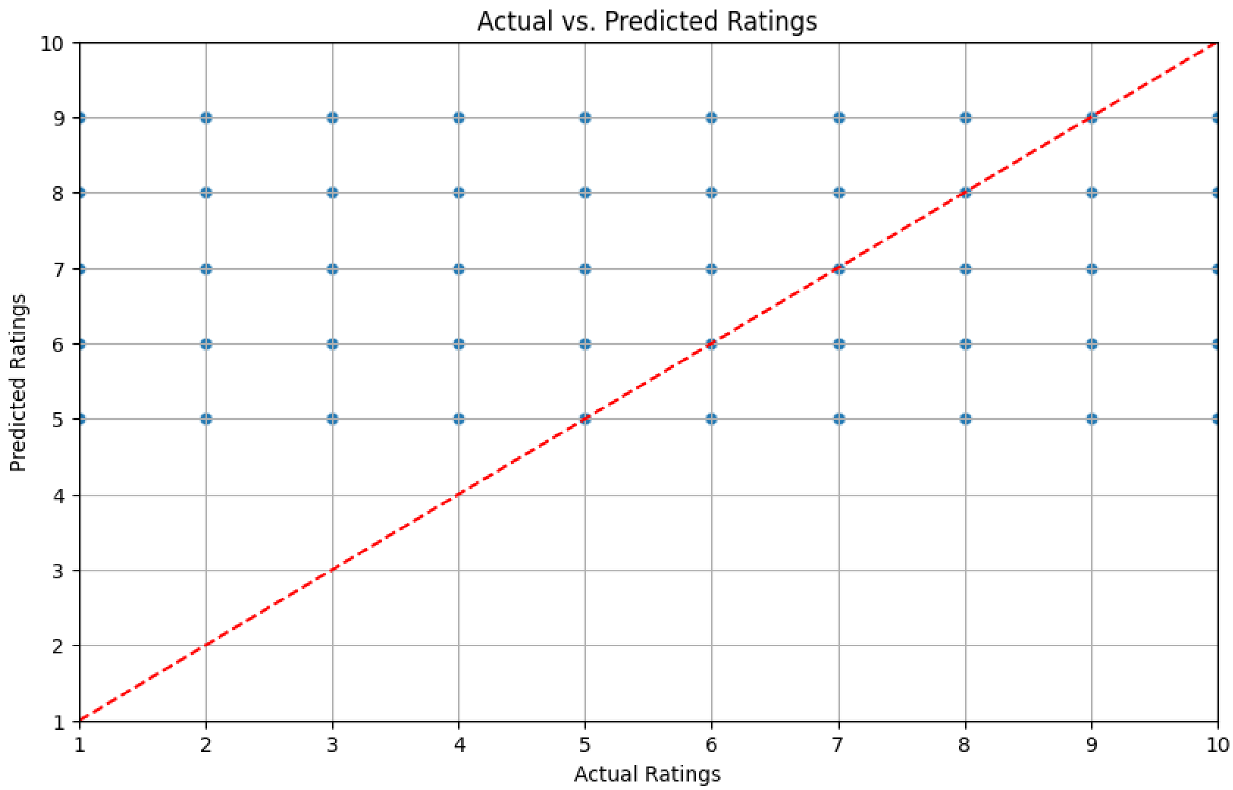
```python
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=predicted_ratings, alpha=0.5)

plt.plot([1, 10], [1, 10], color='red', linestyle='--')

plt.title('Actual vs. Predicted Ratings')
```

```
plt.xlabel('Actual Ratings')
plt.ylabel('Predicted Ratings')
plt.xlim(1, 10)
plt.ylim(1, 10)
plt.xticks(range(1, 11))
plt.yticks(range(1, 11))

# Show plot
plt.grid()
plt.show()
```




```
import pickle

# Save the model to a file
with open('./models/ordinal_logistic_model.pkl', 'wb') as f:
    pickle.dump(model_fit, f)

# Standardize the features
features = rating_under_sampled_df.columns.difference(['user_id',
'anime_id', 'rating', 'Birthday_Date', 'Joined_Date', 'Age_Join',
'Episodes Watched',
                                  'Start Date', 'End Date', 'Name',
'Rank', 'Studios', 'Source', 'Episodes_Norm', 'Unnamed: 0'])
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
scaler = StandardScaler()
df_scaled = scaler.fit_transform(rating_under_sampled_df[features])


# # Split the data
X = df_scaled

y = rating_under_sampled_df[target]

# Ordinal Logistic Regression expects integer targets as levels
y = y.astype('category')

# # Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Fit an Ordinal Logistic Regression model
model_under_rating_standard = OrderedModel(y_train, X_train,
distr='logit')
model_under_rating_standard_fit =
model_under_rating_standard.fit(method='bfgs')
print(model_under_rating_standard_fit.summary())

# Coefficient significance
print(model_under_rating_standard_fit.pvalues)
print(model_under_rating_standard_fit.conf_int())
```

```
/Users/ramachandrank/Repos/MS/sem1/.venv/lib/python3.12/site-
packages/statsmodels/miscmodels/ordinal_model.py:205: Warning: the
endog has ordered == False, risk of capturing a wrong order for the
categories. ordered == True preferred.
  warnings.warn("the endog has ordered == False, "
```

```
Optimization terminated successfully.
         Current function value: 1.921322
         Iterations: 70
         Function evaluations: 71
         Gradient evaluations: 71
                             OrderedModel Results

======================================================================
========
Dep. Variable:                 rating   Log-Likelihood:            -
8.0210e+05
Model:                   OrderedModel   AIC:
1.604e+06
Method:            Maximum Likelihood   BIC:
1.605e+06
Date:                Tue, 05 Nov 2024

Time:                        02:29:34
```

```
No. Observations:                 417472

Df Residuals:                     417427

Df Model:                             36

======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
x1            -0.1021      0.003    -36.055      0.000      -0.108
-0.097
x2             0.0753      0.003     23.745      0.000       0.069
0.082
x3            -0.0345      0.003    -12.010      0.000      -0.040
-0.029
x4            -0.0279      0.003     -9.960      0.000      -0.033
-0.022
x5             0.0320      0.003      9.345      0.000       0.025
0.039
x6             0.0161      0.003      4.891      0.000       0.010
0.023
x7            -0.0209      0.003     -7.099      0.000      -0.027
-0.015
x8             0.1884      0.003     62.968      0.000       0.182
0.194
x9            -0.0424      0.003    -14.848      0.000      -0.048
-0.037
x10           -0.0184      0.003     -5.476      0.000      -0.025
-0.012
x11            0.0930      0.003     30.385      0.000       0.087
0.099
x12           -0.1769      0.003    -59.767      0.000      -0.183
-0.171
x13           -0.0738      0.003    -21.956      0.000      -0.080
-0.067
x14           -0.0488      0.003    -17.949      0.000      -0.054
-0.043
x15           -0.0061      0.003     -2.252      0.024      -0.011
-0.001
x16           -0.0632      0.003    -21.315      0.000      -0.069
-0.057
x17            0.0143      0.003      4.736      0.000       0.008
0.020
x18           -0.0090      0.003     -2.788      0.005      -0.015
-0.003
x19           -0.0778      0.003    -25.039      0.000      -0.084
```

```
-0.072
x20            0.0053      0.003      1.839      0.066      -0.000
0.011
x21            0.0304      0.003     10.308      0.000       0.025
0.036
x22            0.0601      0.003     18.767      0.000       0.054
0.066
x23            0.1048      0.003     34.745      0.000       0.099
0.111
x24            0.1159      0.004     31.156      0.000       0.109
0.123
x25            0.0194      0.004      5.306      0.000       0.012
0.027
x26           -0.1250      0.004    -34.100      0.000      -0.132
-0.118
x27            0.0006      0.004      0.154      0.877      -0.007
0.008
x28            0.2550      0.004     68.830      0.000       0.248
0.262
x29            0.0118      0.004      3.211      0.001       0.005
0.019
x30            0.0012      0.004      0.323      0.747      -0.006
0.008
x31           -0.0551      0.004    -15.070      0.000      -0.062
-0.048
x32            0.0632      0.004     17.261      0.000       0.056
0.070
x33            0.0441      0.023      1.912      0.056      -0.001
0.089
x34            0.1724      0.049      3.553      0.000       0.077
0.267
x35            0.0612      0.033      1.876      0.061      -0.003
0.125
x36            0.2288      0.061      3.760      0.000       0.110
0.348
1.0/2.0       -5.3784      0.021   -250.668      0.000      -5.420
-5.336
2.0/3.0       -0.0688      0.018     -3.844      0.000      -0.104
-0.034
3.0/4.0       -0.2474      0.013    -19.327      0.000      -0.272
-0.222
4.0/5.0       -0.1395      0.008    -16.998      0.000      -0.156
-0.123
5.0/6.0       -0.0502      0.005     -9.570      0.000      -0.061
-0.040
6.0/7.0        0.0244      0.004      6.890      0.000       0.017
0.031
7.0/8.0       -0.2749      0.003    -78.727      0.000      -0.282
-0.268
```

```
8.0/9.0        -0.2827      0.003    -80.933      0.000      -0.290
-0.276
9.0/10.0       -0.0231      0.004     -6.539      0.000      -0.030
-0.016
======================================================================
========
x1          1.137272e-284
x2          1.240609e-124
x3           3.134745e-33
x4           2.278068e-23
x5           9.215770e-21
x6           1.004153e-06
x7           1.259340e-12
x8           0.000000e+00
x9           7.127258e-50
x10          4.350891e-08
x11         8.616374e-203
x12          0.000000e+00
x13         7.632627e-107
x14          4.921900e-72
x15          2.433157e-02
x16         8.164416e-101
x17          2.175851e-06
x18          5.308597e-03
x19         2.286305e-138
x20          6.585217e-02
x21          6.452772e-25
x22          1.411825e-78
x23         1.635496e-264
x24         4.265630e-213
x25          1.121340e-07
x26         7.419827e-255
x27          8.772316e-01
x28          0.000000e+00
x29          1.323391e-03
x30          7.470042e-01
x31          2.564371e-51
x32          9.237565e-67
x33          5.588277e-02
x34          3.814004e-04
x35          6.067350e-02
x36          1.696355e-04
1.0/2.0      0.000000e+00
2.0/3.0      1.208487e-04
3.0/4.0      3.175558e-83
4.0/5.0      8.549532e-65
5.0/6.0      1.064837e-21
6.0/7.0      5.593449e-12
7.0/8.0      0.000000e+00
```

```
8.0/9.0      0.000000e+00
9.0/10.0     6.177096e-11
dtype: float64
                 0         1
x1       -0.107699 -0.096594
x2        0.069124  0.081562
x3       -0.040172 -0.028900
x4       -0.033382 -0.022404
x5        0.025281  0.038700
x6        0.009628  0.022505
x7       -0.026611 -0.015096
x8        0.182493  0.194219
x9       -0.048000 -0.036806
x10      -0.024964 -0.011804
x11       0.086993  0.098990
x12      -0.182710 -0.171107
x13      -0.080335 -0.067168
x14      -0.054111 -0.043457
x15      -0.011492 -0.000796
x16      -0.068995 -0.057376
x17       0.008400  0.020260
x18      -0.015249 -0.002659
x19      -0.083943 -0.071755
x20      -0.000349  0.011009
x21       0.024636  0.036203
x22       0.053794  0.066341
x23       0.098892  0.110716
x24       0.108634  0.123220
x25       0.012221  0.026538
x26      -0.132231 -0.117857
x27      -0.006577  0.007703
x28       0.247776  0.262301
x29       0.004607  0.019044
x30      -0.005973  0.008326
x31      -0.062240 -0.047913
x32       0.056018  0.070370
x33      -0.001109  0.089405
x34       0.077282  0.267490
x35      -0.002742  0.125080
x36       0.109527  0.347990
1.0/2.0  -5.420454 -5.336347
2.0/3.0  -0.103820 -0.033706
3.0/4.0  -0.272443 -0.222274
4.0/5.0  -0.155633 -0.123452
5.0/6.0  -0.060534 -0.039954
6.0/7.0   0.017447  0.031321
7.0/8.0  -0.281789 -0.268099
8.0/9.0  -0.289509 -0.275818
9.0/10.0 -0.029980 -0.016153
```

```python
# Predict probabilities on the test set
predicted_probs = model_under_rating_standard_fit.predict(X_test)

# Convert probabilities to predicted ratings by selecting the category
with the highest probability
predicted_ratings = np.argmax(predicted_probs, axis=1) + 1

y_test = y_test.astype('category').cat.codes + 1

# Calculate evaluation metrics
mse = mean_squared_error(y_test, predicted_ratings)
accuracy = accuracy_score(y_test, predicted_ratings)

print("Mean Squared Error:", mse)
print("Accuracy:", accuracy)
```

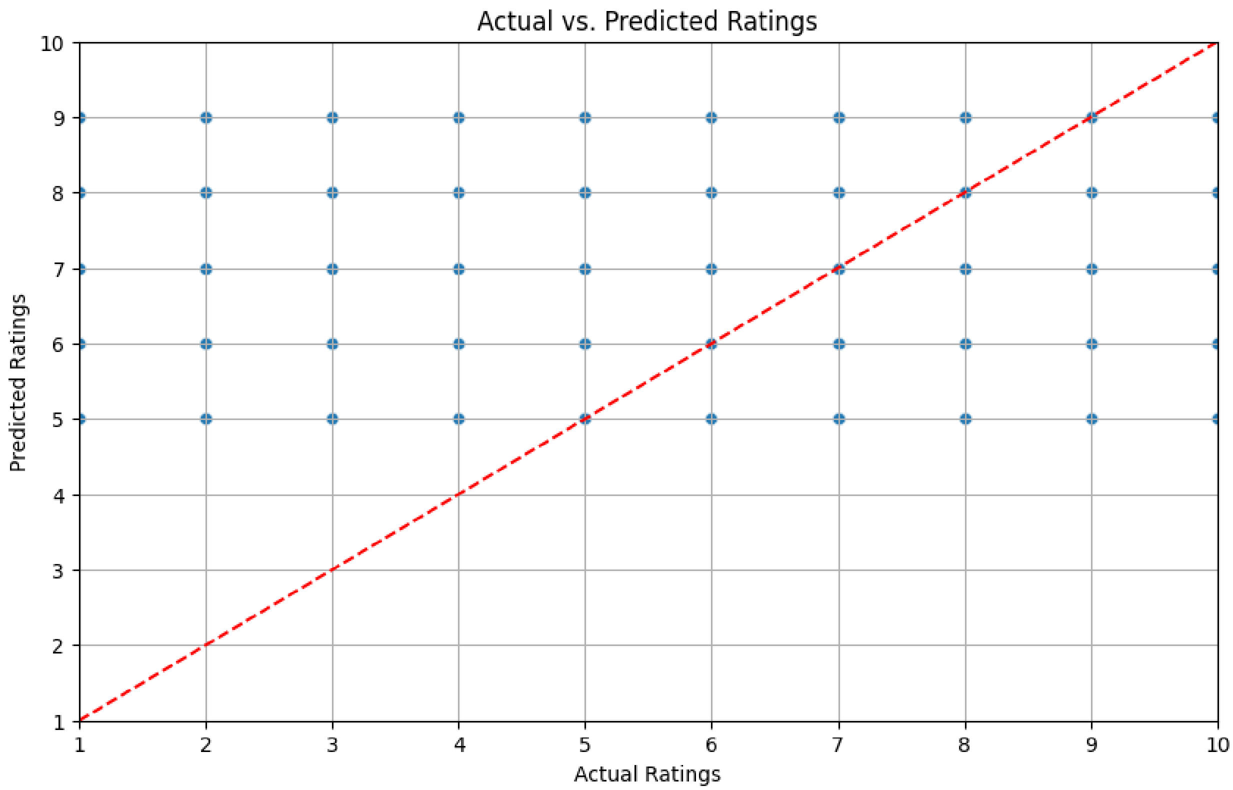
```
Mean Squared Error: 5.318926117908575
Accuracy: 0.163640544606157
```

```python
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=predicted_ratings, alpha=0.5)

plt.plot([1, 10], [1, 10], color='red', linestyle='--')

plt.title('Actual vs. Predicted Ratings')
plt.xlabel('Actual Ratings')
plt.ylabel('Predicted Ratings')
plt.xlim(1, 10)
plt.ylim(1, 10)
plt.xticks(range(1, 11))
plt.yticks(range(1, 11))

# Show plot
plt.grid()
plt.show()
```



Ordered Model was not very accurate with our data. This is mostly due to the sparseness of our data after 1 hot encoding multiple class columns like location and genres.