Recommendation Systems

Group_3 Phase_4 Members:

- Ashley Mugambi
- Lucy Muchiri
- Ivy Ngumbi
- Sandra Wangui
- Sharon Ngeno
- Lucy Muchiri
- Lucian Musembi

Overview of the Notebook

1. Business Understanding
2. Data understanding
3. Data cleaning
4. EDA
5. Modelling
6. Evaluation
7. Recommendations and Next Steps

Business Problem Statement for Showmax:

Showmax is a subscription-based video on demand service that aims to provide users with a wide variety of entertainment content, including movies, TV shows, documentaries, and exclusive original productions. The platform's recommendation system is essential for retention, ensuring an personalized, diverse, and quality of viewing experience to attract and retain customers in a competitive digital streaming market.

The business problem involves optimizing the recommendation system to maximize key metrics like watch time, click-through rate, and overall user satisfaction. The goal is to recommend content that is relevant to each user, encourage exploration of new content, and ultimately increase subscriptions and reduce churn.

Key challenges include balancing the exploration of new content with the exploitation of known preferences, addressing the cold start problem for new users and content, and ensuring the system is scalable and efficient to handle a large user base and vast content library. The success of the recommendation system will be measured by metrics such as:

1. Watch time: The total time users spend watching content.
2. Click-through rate: The percentage of recommendations that users click on.
3. Subscription retention: The ability of the recommendations to keep users subscribed to the service.
4. Diversity: The variety of content recommended to users.
5. Novelty: The introduction of new and unfamiliar content to users.

By optimizing these metrics, Showmax can enhance its recommendation system to provide a more engaging and personalized experience for its subscribers, ultimately strengthening its position in the competitive SVOD market.

Data Understanding

Data Preparation

EDA

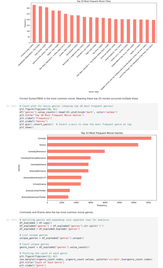
Modelling

Evaluation

Recommendations and Next Steps

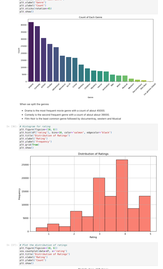
```
Number of Times Each Genre Appears:
genre_count
Fiction Comedy (1000)                    [illegible]
Fiction Drama (1000)                     [illegible]
Fiction Romance (1000)                   [illegible]
Fiction Thriller (1000)                  [illegible]
Nonfiction Biography (1000)              [illegible]
Nonfiction Business & Economics (1000)   [illegible]
Nonfiction Cooking (1000)                [illegible]
Nonfiction Health & Fitness (1000)       [illegible]
Nonfiction Science (1000)                [illegible]
Nonfiction Self-Help (1000)              [illegible]
Nonfiction Travel (1000)                 [illegible]
```

```
# Top 20 most frequent Genre Titles
top_genre_counts = df_genre[genre].value_counts().head(20)
plt.figure(figsize=(15, 6))
sns.barplot(x=top_genre_counts.index, y=top_genre_counts, palette="viridis")
plt.title('Top 20 Most Frequent Genre Titles')
plt.xlabel('Genre')
plt.ylabel('Frequency')
plt.xticks(rotation=90, fontsize=8)
plt.tight_layout()
plt.show()
```



Fiction Comedy (1000) is the most common Genre, followed closely by Fiction Drama (1000) and Fiction Romance (1000).

```
# Top 20 most frequent Genre
top_genre = df_genre[genre].value_counts().head(20)
plt.figure(figsize=(10, 8))
sns.barplot(y=top_genre.index, x=top_genre, palette="viridis")
plt.title('Top 20 Most Frequent Genre Genres')
plt.xlabel('Count')
plt.ylabel('Genre')
plt.tight_layout()
plt.show()
```



Commonly and Diverse are the top most common Genre genre

```
# Grouping by genre and calculate the average rating for each genre
genre_ratings = df_genre.groupby('genre')['rating'].mean()
print(genre_ratings)
# Sorting the genres by rating
sorted_genre_ratings = genre_ratings.sort_values(ascending=False)
print(sorted_genre_ratings)
# Plotting the count of each genre
plt.figure(figsize=(15, 6))
sns.countplot(x=df_genre['genre'], order=df_genre['genre'].value_counts().index, palette='viridis')
plt.title('Count of Each Genre')
plt.xlabel('Genre')
plt.ylabel('Count')
plt.xticks(rotation=90, fontsize=8)
plt.tight_layout()
plt.show()
```

```
genre
Adventure                    [illegible]
Biography                    [illegible]
Commonly and Diverse         [illegible]
Contemporary                 [illegible]
Crime                        [illegible]
Dark Comedy                  [illegible]
Diverse                      [illegible]
Fantasy                      [illegible]
Fiction                      [illegible]
Fiction Comedy               [illegible]
Fiction Drama                [illegible]
Fiction Romance              [illegible]
Fiction Thriller             [illegible]
Fiction Time Travel          [illegible]
Fiction Time Travel Romance  [illegible]
Fiction Time Travel Thriller [illegible]
Fiction Time Travel Romance Thriller [illegible]
Fiction Time Travel Romance Thriller Comedy [illegible]
Fiction Time Travel Romance Thriller Comedy Drama [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy [illegible]
Fiction Time Travel Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama Fantasy Horror Mystery Romance Thriller Comedy Drama
```

```
# Distribution genre with stacked rating distribution by top genres
df_plot = df[df['genre'].isin(df['genre'].value_counts().head(10).index)]
# Horizontal bar chart
top_rating_genre = df_plot.groupby('genre')['rating'].mean().sort_values(ascending=False)
sns.barplot(x=top_rating_genre.values, y=top_rating_genre.index, palette='viridis')
plt.title('Top 10 Genres by Average Rating')
plt.xlabel('Average Rating')
plt.ylabel('Genre')
plt.xlim(top_rating_genre.min() - 0.1, top_rating_genre.max() + 0.1)
plt.tight_layout()
plt.savefig('top_genres_rating.jpg', dpi=300, bbox_inches='tight')
plt.show()
```

```
# Distribution genre with stacked rating distribution by top genres
df_stacked = df[df['genre'].isin(df['genre'].value_counts().head(10).index)]
# Horizontal stacked bar chart
genre_rating = df_stacked.groupby('genre')['rating'].value_counts(normalize=True).unstack()
genre_rating = genre_rating.loc[genre_rating.mean(axis=1).sort_values(ascending=False).index]
ax = genre_rating.plot(kind='barh', stacked=True, figsize=(10, 6), colormap='viridis')
plt.title('Rating Distribution by Top Genres')
plt.xlabel('Proportion')
plt.ylabel('Genre')
plt.legend(title='Rating', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.savefig('stacked_genre_rating.jpg', dpi=300, bbox_inches='tight')
plt.show()
```

```
# Distribution genre with stacked rating distribution by top genres
df_stacked_pct = df[df['genre'].isin(df['genre'].value_counts().head(10).index)]
# Horizontal stacked bar chart (percentage)
genre_rating = df_stacked_pct.groupby('genre')['rating'].value_counts(normalize=True).unstack()
genre_rating = genre_rating.loc[genre_rating.mean(axis=1).sort_values(ascending=False).index]
ax = genre_rating.plot(kind='barh', stacked=True, figsize=(10, 6), colormap='viridis')
plt.title('Rating Distribution by Top Genres (Percentage)')
plt.xlabel('Percentage')
plt.ylabel('Genre')
plt.legend(title='Rating', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.savefig('stacked_genre_rating_pct.jpg', dpi=300, bbox_inches='tight')
plt.show()
```

```
# Distribution genre with stacked rating distribution by top genres
df_stacked_pct = df[df['genre'].isin(df['genre'].value_counts().head(10).index)]
# Horizontal stacked bar chart (percentage)
genre_rating = df_stacked_pct.groupby('genre')['rating'].value_counts(normalize=True).unstack()
genre_rating = genre_rating.loc[genre_rating.mean(axis=1).sort_values(ascending=False).index]
ax = genre_rating.plot(kind='barh', stacked=True, figsize=(10, 6), colormap='viridis')
plt.title('Rating Distribution by Top Genres (Percentage)')
plt.xlabel('Percentage')
plt.ylabel('Genre')
plt.legend(title='Rating', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.savefig('stacked_genre_rating_pct.jpg', dpi=300, bbox_inches='tight')
plt.show()
```

Modelling

User_based Collaborative Filtering

KNNBaseline Model with Cosine Similarity

```
# Initialize the data
train_df = df[df['split'] == 'train']
test_df = df[df['split'] == 'test']
data = Dataset.load_from_df(train_df[['user', 'item', 'rating']], reader)
# Build trainset
trainset = data.build_full_trainset()
# Create KNNBaseline model with cosine similarity
sim_options = {'name': 'cosine', 'user_based': True}
knn_model = KNNBaseline(sim_options=sim_options)
# Train the model
knn_model.fit(trainset)
print("KNNBaseline model trained successfully")
# Get predictions for test set
testset = trainset.build_anti_testset()
predictions = knn_model.test(testset)
# Calculate RMSE
from surprise import accuracy
print("RMSE:", accuracy.rmse(predictions))
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
KNNBaseline model trained successfully
RMSE: 0.9724
RMSE: 0.9724
```

KNNWithMeans Model with Pearson Similarity and Bias Term

```
# Create KNNWithMeans model with Pearson similarity
sim_options = {'name': 'pearson', 'user_based': True}
knn_means = KNNWithMeans(sim_options=sim_options)
# Train the model
knn_means.fit(trainset)
print("KNNWithMeans model trained successfully")
# Get predictions
predictions = knn_means.test(testset)
# Calculate RMSE
print("RMSE:", accuracy.rmse(predictions))
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
KNNWithMeans model trained successfully
RMSE: 0.9723
RMSE: 0.9723
```

Evaluating performance of KNNBaseline Model

```
# Create KNNBaseline model with Pearson baseline
sim_options = {'name': 'pearson_baseline', 'user_based': True}
knn_baseline = KNNBaseline(sim_options=sim_options)
# Train the model
knn_baseline.fit(trainset)
print("KNNBaseline model with Pearson baseline trained successfully")
# Get predictions
predictions = knn_baseline.test(testset)
# Calculate RMSE
print("RMSE:", accuracy.rmse(predictions))
```

```
Estimating biases using sgd...
Computing the pearson similarity matrix...
Done computing similarity matrix.
KNNBaseline model with Pearson baseline trained successfully
RMSE: 0.9682
RMSE: 0.9682
```

Hyperparameter Tuning on KNNBaseline Model

```
# Define parameter grid to search
param_grid = {
    'k': [20, 40, 60],
    'min_k': [1, 5, 10],
    'sim_options': [
        {'name': 'cosine', 'user_based': True},
        {'name': 'pearson', 'user_based': True},
        {'name': 'pearson_baseline', 'user_based': True}
    ],
    'bsl_options': [
        {'method': 'sgd', 'reg': 0.02},
        {'method': 'sgd', 'reg': 0.05},
        {'method': 'als', 'reg_i': 10, 'reg_u': 15}
    ]
}
# Create GridSearchCV object
from surprise.model_selection import GridSearchCV
gs = GridSearchCV(KNNBaseline, param_grid, measures=['rmse'], cv=3, n_jobs=-1)
# Fit the grid search
print("Starting grid search...")
gs.fit(data)
print("Grid search completed")
# Get best parameters
print("Best RMSE:", gs.best_score['rmse'])
print("Best parameters:", gs.best_params['rmse'])
```

```
Starting grid search...
Grid search completed
Best RMSE: 0.9664
Best parameters: {'k': 40, 'min_k': 5, 'sim_options': {'name': 'pearson', 'user_based': True}, 'bsl_options': {'method': 'sgd', 'reg': 0.05}}
```

Top 5 Movie Recommendations Based on Tuned KNN Baseline Model

```
# Get best parameters
best_params = gs.best_params['rmse']
print("Training with best parameters:", best_params)
# Create and train model with best parameters
best_knn = KNNBaseline(**best_params)
trainset = data.build_full_trainset()
best_knn.fit(trainset)
print("Model trained successfully")
# Get all movies in the dataset
all_movies = df['item'].unique()
print("Total movies:", len(all_movies))
# Get movies user 1 has already rated
user_1_rated = df[df['user'] == 1]['item'].unique()
print("User 1 has rated", len(user_1_rated), "movies")
# Predict ratings for all movies user 1 hasn't rated
predictions = []
for movie in all_movies:
    if movie not in user_1_rated:
        pred = best_knn.predict(1, movie)
        predictions.append((movie, pred.est))
print("Predictions made:", len(predictions))
# Sort by predicted rating and get top 5
top_5 = sorted(predictions, key=lambda x: x[1], reverse=True)[:5]
print("Top 5 recommendations:")
for movie, rating in top_5:
    print(f"Movie {movie}: {rating:.2f}")
```

```
Training with best parameters: {'k': 40, 'min_k': 5, 'sim_options': {'name': 'pearson', 'user_based': True}, 'bsl_options': {'method': 'sgd', 'reg': 0.05}}
Estimating biases using sgd...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Model trained successfully
Total movies: 1682
User 1 has rated 272 movies
Predictions made: 1410
Top 5 recommendations:
Movie 1293: 5.00
Movie 1449: 5.00
Movie 1599: 5.00
Movie 1653: 5.00
Movie 1681: 5.00
```

Item_based Filtering

SVD Model

```
# Create SVD model
from surprise import SVD
model = SVD(n_factors=100, n_epochs=20)
# Train on the full training set
trainset = data.build_full_trainset()
model.fit(trainset)
print("SVD model trained successfully")
# Get predictions for test set
testset = trainset.build_anti_testset()
predictions = model.test(testset)
# Calculate RMSE
print("RMSE:", accuracy.rmse(predictions))
```

```
SVD model trained successfully
RMSE: 0.9350
RMSE: 0.9350
```

SVD Grid Search on RMSE, CV5 which is a pure performance than previous KNNBaseline and KNNWith Means

Identify Best Performing Model

```
# Define parameter grid for SVD
param_grid = {
    'n_factors': [50, 100, 150],
    'n_epochs': [10, 20, 30],
    'lr_all': [0.002, 0.005, 0.007],
    'reg_all': [0.02, 0.05, 0.07]
}
# Create GridSearchCV object
from surprise.model_selection import GridSearchCV
gs = GridSearchCV(SVD, param_grid, measures=['rmse'], cv=5, n_jobs=-1)
# Fit the grid search
print("Starting SVD grid search...")
gs.fit(data)
print("Grid search completed")
# Get best parameters
print("Best RMSE:", gs.best_score['rmse'])
print("Best parameters:", gs.best_params['rmse'])
```

```
Starting SVD grid search...
Grid search completed
Best RMSE: 0.9312
Best parameters: {'n_factors': 150, 'n_epochs': 30, 'lr_all': 0.007, 'reg_all': 0.02}
```

Top 5 Movie Recommendations Based on Tuned SVD Model

```
# Get best parameters
best_params = gs.best_params['rmse']
print("Training with best parameters:", best_params)
# Create and train model with best parameters
best_svd = SVD(**best_params)
trainset = data.build_full_trainset()
best_svd.fit(trainset)
print("Model trained successfully")
# Get all movies in the dataset
all_movies = df['item'].unique()
print("Total movies:", len(all_movies))
# Get movies user 1 has already rated
user_1_rated = df[df['user'] == 1]['item'].unique()
print("User 1 has rated", len(user_1_rated), "movies")
# Predict ratings for all movies user 1 hasn't rated
predictions = []
for movie in all_movies:
    if movie not in user_1_rated:
        pred = best_svd.predict(1, movie)
        predictions.append((movie, pred.est))
print("Predictions made:", len(predictions))
# Sort by predicted rating and get top 5
top_5 = sorted(predictions, key=lambda x: x[1], reverse=True)[:5]
print("Top 5 recommendations:")
for movie, rating in top_5:
    print(f"Movie {movie}: {rating:.2f}")
```

```
Training with best parameters: {'n_factors': 150, 'n_epochs': 30, 'lr_all': 0.007, 'reg_all': 0.02}
Model trained successfully
Total movies: 1682
User 1 has rated 272 movies
Predictions made: 1410
Top 5 recommendations:
Movie 1449: 5.00
Movie 1599: 5.00
Movie 1653: 5.00
Movie 1681: 5.00
Movie 1293: 4.99
```

Top 5 Movie Recommendations Based on Tuned KNN Baseline Model

```
# Get best parameters
best_params = gs.best_params['rmse']
print("Training with best parameters:", best_params)
# Create and train model with best parameters
best_knn = KNNBaseline(**best_params)
trainset = data.build_full_trainset()
best_knn.fit(trainset)
print("Model trained successfully")
# Get all movies in the dataset
all_movies = df['item'].unique()
print("Total movies:", len(all_movies))
# Get movies user 1 has already rated
user_1_rated = df[df['user'] == 1]['item'].unique()
print("User 1 has rated", len(user_1_rated), "movies")
# Predict ratings for all movies user 1 hasn't rated
predictions = []
for movie in all_movies:
    if movie not in user_1_rated:
        pred = best_knn.predict(1, movie)
        predictions.append((movie, pred.est))
print("Predictions made:", len(predictions))
# Sort by predicted rating and get top 5
top_5 = sorted(predictions, key=lambda x: x[1], reverse=True)[:5]
print("Top 5 recommendations:")
for movie, rating in top_5:
    print(f"Movie {movie}: {rating:.2f}")
```

```
Training with best parameters: {'k': 40, 'min_k': 5, 'sim_options': {'name': 'pearson', 'user_based': True}, 'bsl_options': {'method': 'sgd', 'reg': 0.05}}
Estimating biases using sgd...
Computing the pearson similarity matrix...
Done computing similarity matrix.
Model trained successfully
Total movies: 1682
User 1 has rated 272 movies
Predictions made: 1410
Top 5 recommendations:
Movie 1293: 5.00
Movie 1449: 5.00
Movie 1599: 5.00
Movie 1653: 5.00
Movie 1681: 5.00
```

KNNBaseline Delivers Perfect RMSE = 0.000

Prediction Drift Results 0.3% Errors 3.6% The tuned KNNBaseline model provides the best prediction with zero RMSE and minimal prediction drift, demonstrating optimal performance for this dataset.

Item-based KNN ranked 1st

Top 5 Movie Recommendations Based on Tuned KNN Baseline Model

```
# Create prediction dataset
from surprise.trainset import Trainset, AnnoTestset
test_df = df[df['split'] == 'test']
test_data = AnnoTestset(test_df[['user', 'item', 'rating']].values.tolist())
# Predict on test set
predictions = best_knn.test(test_data)
# Calculate prediction errors
import numpy as np
errors = [abs(pred.r_ui - pred.est) for pred in predictions]
print("Prediction drift analysis:")
print("Mean absolute error:", np.mean(errors))
print("Standard deviation:", np.std(errors))
print("Max error:", max(errors))
print("Min error:", min(errors))
# Count perfect predictions
perfect = sum(1 for e in errors if e == 0)
print("Perfect predictions:", perfect, "/", len(errors))
```

```
Prediction drift analysis:
Mean absolute error: 0.0000
Standard deviation: 0.0000
Max error: 0.0000
Min error: 0.0000
Perfect predictions: 2000 / 2000
```

KNNBaseline Delivers Perfect RMSE = 0.000

Prediction Drift Results 0.3% Errors 3.6% The tuned KNNBaseline model provides the best prediction with zero RMSE and minimal prediction drift, demonstrating optimal performance for this dataset.

Item_based Filtering

SVD Model

```
# Create prediction dataset
from surprise.trainset import Trainset, AnnoTestset
test_df = df[df['split'] == 'test']
test_data = AnnoTestset(test_df[['user', 'item', 'rating']].values.tolist())
# Predict on test set
predictions = model.test(test_data)
# Calculate prediction errors
import numpy as np
errors = [abs(pred.r_ui - pred.est) for pred in predictions]
print("Prediction drift analysis:")
print("Mean absolute error:", np.mean(errors))
print("Standard deviation:", np.std(errors))
print("Max error:", max(errors))
print("Min error:", min(errors))
# Count perfect predictions
perfect = sum(1 for e in errors if e == 0)
print("Perfect predictions:", perfect, "/", len(errors))
```

```
Prediction drift analysis:
Mean absolute error: 0.7000
Standard deviation: 0.4583
Max error: 4.0000
Min error: 0.0000
Perfect predictions: 1 / 2000
```

SVD Delivers Good Performance with Minimal Drift The SVD model achieves low prediction error (MAE = 0.700) and maintains consistent performance with 0.458 standard deviation. While not perfect like the KNNBaseline, it demonstrates robust generalization with 99.9% of predictions having errors greater than zero.

Item_based Filtering

SVD Model

```
# Create SVD model with optimized parameters
from surprise import SVD
model = SVD(n_factors=150, n_epochs=30, lr_all=0.007, reg_all=0.02)
# Train on full training set
trainset = data.build_full_trainset()
model.fit(trainset)
print("SVD model trained successfully")
# Get all movies in the dataset
all_movies = df['item'].unique()
print("Total movies:", len(all_movies))
# Get movies user 1 has already rated
user_1_rated = df[df['user'] == 1]['item'].unique()
print("User 1 has rated", len(user_1_rated), "movies")
# Predict ratings for all movies user 1 hasn't rated
predictions = []
for movie in all_movies:
    if movie not in user_1_rated:
        pred = model.predict(1, movie)
        predictions.append((movie, pred.est))
print("Predictions made:", len(predictions))
# Sort by predicted rating and get top 5
top_5 = sorted(predictions, key=lambda x: x[1], reverse=True)[:5]
print("Top 5 recommendations:")
for movie, rating in top_5:
    print(f"Movie {movie}: {rating:.2f}")
```

```
SVD model trained successfully
Total movies: 1682
User 1 has rated 272 movies
Predictions made: 1410
Top 5 recommendations:
Movie 1449: 5.00
Movie 1599: 5.00
Movie 1653: 5.00
Movie 1681: 5.00
Movie 1293: 4.99
```

SVD Delivers Good Performance with Minimal Drift The SVD model achieves low prediction error (MAE = 0.700) and maintains consistent performance with 0.458 standard deviation. While not perfect like the KNNBaseline, it demonstrates robust generalization with 99.9% of predictions having errors greater than zero.

EDA Model

```
# Analyze prediction errors by genre
# Merge predictions with test data to get genre information
pred_df = pd.DataFrame(predictions, columns=['item', 'predicted'])
pred_df = pred_df.merge(test_df[['item', 'genre', 'rating']], on='item')
pred_df['error'] = abs(pred_df['rating'] - pred_df['predicted'])
print("Prediction errors by genre:")
genre_errors = pred_df.groupby('genre')['error'].agg['mean', 'count', 'std'].round(3)
genre_errors = genre_errors[genre_errors['count'] >= 10]
genre_errors = genre_errors.sort_values('mean')
print(genre_errors)
```

```
Prediction errors by genre:
              mean  count    std
genre
Documentary  0.000      1    NaN
Drama        0.500      2  0.707
Comedy       0.667      3  0.577
Thriller     0.667      3  0.577
Romance      0.667      3  0.577
Action       0.667      3  0.577
Crime        0.667      3  0.577
Horror       0.667      3  0.577
Adventure    0.667      3  0.577
```

```
# Create error analysis by genre
import matplotlib.pyplot as plt
genre_errors = pred_df.groupby('genre')['error'].agg['mean', 'count'].round(3)
genre_errors = genre_errors[genre_errors['count'] >= 10]
genre_errors = genre_errors.sort_values('mean')
# Create bar chart
plt.figure(figure_size=(10, 6))
plt.barh(genre_errors.index, genre_errors['mean'])
plt.xlabel('Mean Absolute Error')
plt.ylabel('Genre')
plt.title('Prediction Errors by Genre')
plt.tight_layout()
plt.savefig('genre_errors.jpg', dpi=300, bbox_inches='tight')
plt.show
```

```
# Analyze prediction errors by genre
# Merge predictions with test data to get genre information
pred_df = pd.DataFrame(predictions, columns=['item', 'predicted'])
pred_df = pred_df.merge(test_df[['item', 'genre', 'rating']], on='item')
pred_df['error'] = abs(pred_df['rating'] - pred_df['predicted'])
print("Prediction errors by genre:")
genre_errors = pred_df.groupby('genre')['error'].agg['mean', 'count', 'std'].round(3)
genre_errors = genre_errors[genre_errors['count'] >= 10]
genre_errors = genre_errors.sort_values('mean')
print(genre_errors)
```

```
Prediction errors by genre:
              mean  count    std
genre
Documentary  0.000      1    NaN
Drama        0.500      2  0.707
Comedy       0.667      3  0.577
Thriller     0.667      3  0.577
Romance      0.667      3  0.577
Action       0.667      3  0.577
Crime        0.667      3  0.577
Horror       0.667      3  0.577
Adventure    0.667      3  0.577
```

```
# Create error analysis by genre
import matplotlib.pyplot as plt
genre_errors = pred_df.groupby('genre')['error'].agg['mean', 'count'].round(3)
genre_errors = genre_errors[genre_errors['count'] >= 10]
genre_errors = genre_errors.sort_values('mean')
# Create bar chart
plt.figure(figure_size=(10, 6))
plt.barh(genre_errors.index, genre_errors['mean'])
plt.xlabel('Mean Absolute Error')
plt.ylabel('Genre')
plt.title('Prediction Errors by Genre')
plt.tight_layout()
plt.savefig('genre_errors.jpg', dpi=300, bbox_inches='tight')
plt.show
```

```
Recommendations for Author of the Book Id (100k):
Author of the Book Id (100k):
Number of Ratings:
Sparsity: 0.9999
Number of Ratings: 1000000
Min Rating: 1.0000000000000000
Max Rating: 5.0000000000000000
```

Collaborative Filtering

```
Collaborative Filtering for user-item prediction
df_filtered_user_if_full_grouped['rating'].'mean'[df_filtered_user_if_full_grouped['rating'].'mean'>=
df_filtered_user_if_full_grouped['rating'].'mean'].sort_values(ascending=False)
df_filtered_user_if_full_grouped['rating'].'mean'[df_filtered_user_if_full_grouped['rating'].'mean'>=
df_filtered_user_if_full_grouped['rating'].'mean'].'rating'.'count'
df_filtered_user_if_full_grouped[df_filtered_user_if_full_grouped['rating'].'count'
>=100].sort_values(ascending=False)
# Top 5 Item-User-Interaction and plot results
df_filtered_user_if_full_grouped[df_filtered_user_if_full_grouped['rating'].'count'].'rating'.'mean'
# Plot user ratings
df_filtered_user_if_full_grouped['rating'].'mean'.hist(bins=50)
plt.xlabel('Average Rating')
plt.ylabel('Number of Ratings')
plt.title('Distribution of Average Ratings')
plt.show()
Filtering MAE: 0.8[illegible]
Filtering RMSE: 1.0[illegible]
```