Phase 4 Project Submission

Student name: Felix Limo.

Student pace: Part Time-DS-PT08.

Scheduled project review date/time:

Instructor names: Samuel Karu & Daniel Ekale.

1.0 Introduction ¶

This project involves analysis of movies data to build a recommendation system model that provides diverse options and accurate recommendations to customers that improves their shopping experience and increase engagement with shop catalogs, subsequently increasing sales. The research follows cross industry standard procedures (CRISP-DM) methodlogy fo the movies industry.

2.0 Business Understanding

In the midst of modern business competion,the new film shop purposes to increase its customer interaction by providing personalized recommendations of individual films. This recommendation system will be important for improvement in customer experience, increase customer engagement and driving sale. By offering personal suggestions based on customers' preferences, previous behavior and film ratings, the shop expects to increase customer engagement, increase sales and improve customer retention.

2.1 Objective

The research mainly aims at developing a movie recommendation system, which would be helpful in recommending other similar movies to customers depending on the preference that a customer may have for a particular movie. A customer interested in a particular movie-he asks questions about it or looks at it in a catalog-the system should suggest other movies similar to the target movie.

3.0 The Data

The dataset for modelling was drawn from [https://grouplens.org/datasets/movielens/latest/ (https://grouplens.org/datasets/movielens/latest/)](https://grouplens.org/datasets/movielens/latest/). Merged dataset contains 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users.

Content

- **userId:** Unique identifier for the user.
- **movieId:** Unique identifier for movie.
- **rating:** Ratings given by the user to the movie.
- **timestamp:** Time at which the rating was given by user.
- **title:** Name of the movie.
- **genres:** The genres for which movies belong.
- **tag:** A glimpse of what the movie is about or like.

3.1 Data Understanding

Data Preview

This is important as it provides a snapshot of the type of information contained in the dataset for analysis.

Import relevant python libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Loading of the MovieLens datasets for preview

In [2]:
```
links = pd.read_csv("links.csv")
movies = pd.read_csv("movies.csv")
ratings = pd.read_csv("ratings.csv")
tags = pd.read_csv("tags.csv")
```

In [3]:
```
print(f'Links dataset first 3 records \n {links.head(3)} ' )
print('------------')
print(f'Movies dataset first 3 records \n  {movies.head(3)}' )
print('------------')
print(f'Ratings dataset first 3 records \n  {ratings.head(3)}' )
print('------------')
print(f'Tags dataset first 3 records \n  {tags.head(3)}' )
```

```
Links dataset first 3 records 
    movieId  imdbId   tmdbId
0        1  114709    862.0
1        2  113497   8844.0
2        3  113228  15602.0 
------------
Movies dataset first 3 records 
     movieId                    title  \
0        1         Toy Story (1995)   
1        2           Jumanji (1995)   
2        3  Grumpier Old Men (1995)   

                                        genres  
0  Adventure|Animation|Children|Comedy|Fantasy  
1                   Adventure|Children|Fantasy  
2                               Comedy|Romance  
------------
Ratings dataset first 3 records 
     userId  movieId  rating  timestamp
0       1        1     4.0  964982703
1       1        3     4.0  964981247
2       1        6     4.0  964982224
------------
Tags dataset first 3 records 
     userId  movieId              tag   timestamp
0       2    60756            funny  1445714994
1       2    60756  Highly quotable  1445714996
2       2    60756     will ferrell  1445714992
```

Observations

- Movies, Ratings and Tags datasets will be merged to form data enriched dataset for analysis. Merging criteria on *movieId* with an *inner joint*.
- Links datasets only contains unique identifies (IDs) and may not be useful for this study, thus will not be utilized.

In [4]:
```
#Merge movie and ratings datasets on movieId with an inner joint and assign mo
movie_ratings = pd.merge(ratings,movies, on='movieId', how='inner')

#Merge the resultant movie_ratings with tags on movieId with inner joint and a
movie_rating_tags = pd.merge(movie_ratings, tags, on=['movieId'], how='inner'

#Remove duplicates if any
movie_rating_tags = movie_rating_tags.drop_duplicates()

#Check the first 5 rows of the merged dataset
movie_rating_tags.head()
```

Out[4]:

	userId_x	movieId	rating	timestamp_x	title	genres	us
0	1	1	4.0	964982703	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	
1	1	1	4.0	964982703	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	
2	1	1	4.0	964982703	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	
3	5	1	4.0	847434962	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	
4	5	1	4.0	847434962	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	

In [5]:
```
#Check merged dataset info
movie_rating_tags.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 233213 entries, 0 to 233212
Data columns (total 9 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   userId_x     233213 non-null  int64  
 1   movieId      233213 non-null  int64  
 2   rating       233213 non-null  float64
 3   timestamp_x  233213 non-null  int64  
 4   title        233213 non-null  object 
 5   genres       233213 non-null  object 
 6   userId_y     233213 non-null  int64  
 7   tag          233213 non-null  object 
 8   timestamp_y  233213 non-null  int64  
dtypes: float64(1), int64(5), object(3)
memory usage: 16.0+ MB
```

Observations

- The dataset has 233213 rows and 9 columns, although there,s duplication of columns (userId & timestamp).
- It has 6 numerical features and 3 object features.
- Dataset has equal number of non_null counts in all columns, indicates that there are no missing values.
- Contains movieId and userId making the dataset suitable for building recommendation system(user-based and content-based).

3.2 Problem Statement

A new movie shop opens a branch in a new town with an aim to invent better interaction with customers by offering personalized movie recommendations. The company aims to recommend movies in which the customers have shown interest, liked, or even inquired about. This customized service will expose the customer to films they might not have considered but will likely enjoy based on the films they browse or inquire about. It would, therefore, be able to provide personalized recommendations through customer data on movie preference, past queries, and behavior to enhance customer experience, thereby commanding high satisfaction, loyalty, and repeat visits.

General Objective

- To build a model that provides top 5 movie recommendations to a user, based on their ratings of other movies.

Specific Objectives

- **Personalized Recommendations:** Build a system that will be able to recommend movies based on what customers have done, liked, or searched.
- **Enhanced Discovery:** Help customers discover movies that they may have never considered but might like and thus increase their tastes and knowledge of films.
- **Customer Engagement:** Incentivize customers to spend more time on the website with value-added recommendations relevant to their interests.
- **Increased Sales and Retention:** Personalized suggestions will increase sales and improve customer retention, as they will revisit your site for more and remain longer-term.
- **Enhanced User Experience:** Facilitate an easy and smooth recommendation experience for your customers.

3.3 Metrics of success

This project will be deemed successful if the built models will be able to predict top 5 movie recommendations to a user, based on their ratings of other movies.

4.0 Data Preparation

4.1 Data Cleaning

Involves checking and removal of duplicates,checking for missing values and mitigation, and feature engineering.

Dataset preview revealed duplicated columns and non-uniform feature naming. Therefore, all feature names will be converted to lowercase and remove the duplicated columns(userId_y, timestamp_y). Subsequently, rename 'userId_x' and 'timestamp_x' features to remove the suffixes.

In [6]:
```
#Check for duplicates if any and print out
print(f'Duplicates: \n......\n{movie_rating_tags.duplicated().sum()}')
#Check for missing values duplicates if any and print out
print(f'Missing values: \n....... \n {movie_rating_tags.isna().sum()}')
```

```
Duplicates:
......
0
Missing values:
.......
 userId_x      0
movieId       0
rating        0
timestamp_x   0
title         0
genres        0
userId_y      0
tag           0
timestamp_y   0
dtype: int64
```

Observation

There are no duplicated rows and no missing values in all columns. Further cleaning to ensure uniformity and feature selection.

In [7]:
```
#Remove 'userId_y' and 'timestap' features
movie_rating_tags = movie_rating_tags.drop(["userId_y","timestamp_x","timesta
#Rename 'userId_x' as 'userid'
movie_rating_tags = movie_rating_tags.rename(columns={"userId_x": "userid", "
#Convert feature lowercase for uniformity
movie_rating_tags.columns = movie_rating_tags.columns.str.strip().str.lower()
#Remove duplicates if any
movie_rating_tags = movie_rating_tags.drop_duplicates()
```

4.1.1 Save cleaned dataset to df

In [8]:
```
#Making a copy of cleaned dataset and save as df
df = movie_rating_tags.copy(deep=True)
```

Preview the clean data set.

In [9]:
```
df.head()
```

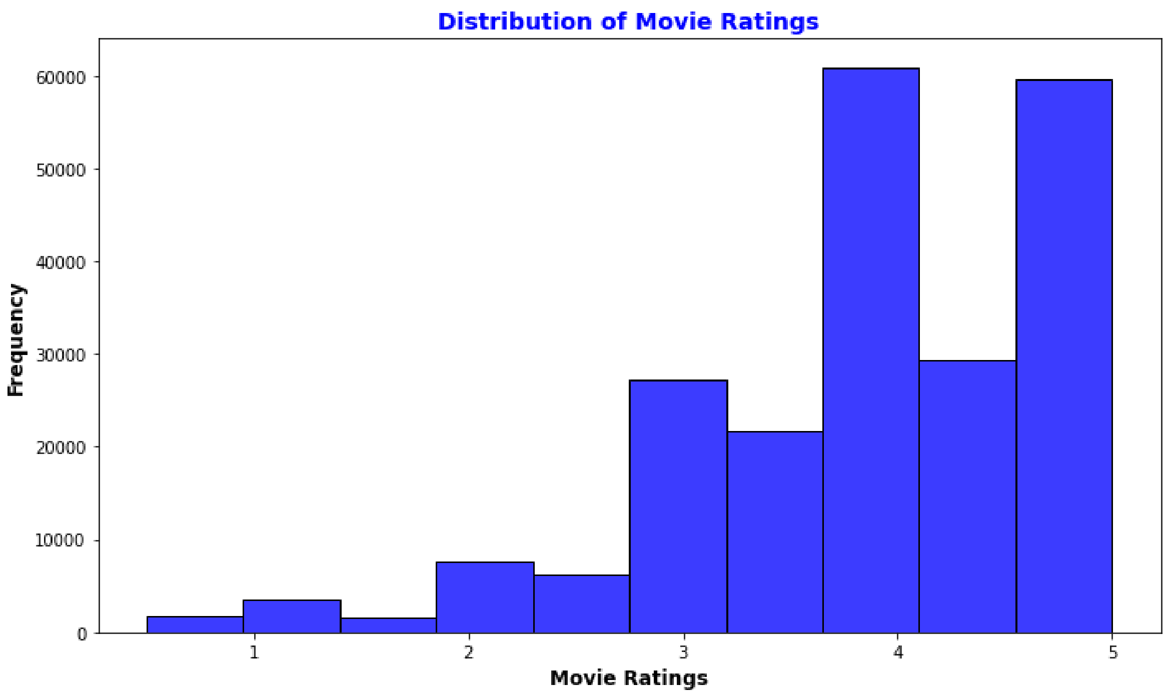
Out[9]:

	userid	movieid	rating	title	genres	tag
0	1	1	4.0	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	pixar
2	1	1	4.0	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	fun
3	5	1	4.0	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	pixar
5	5	1	4.0	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	fun
6	7	1	4.5	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	pixar

4.2 Data Exploration

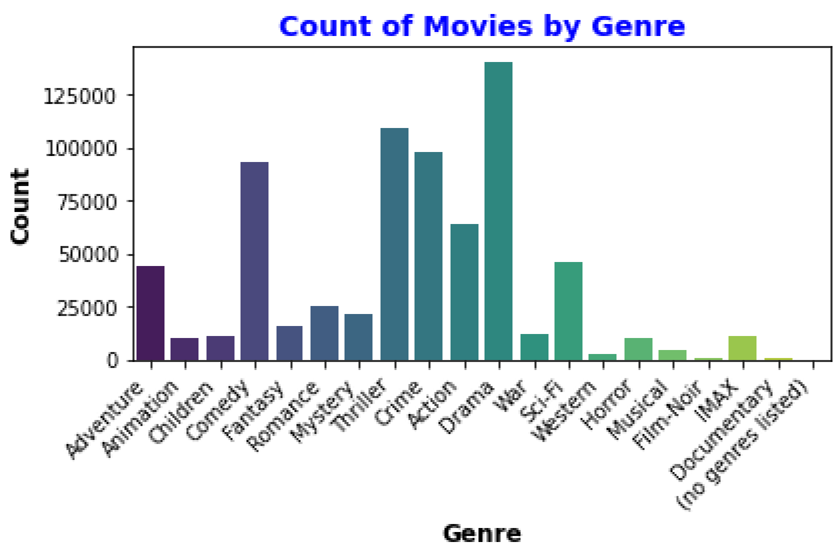
4.2.1 Visualization of distribution of movies based on their ratings.

In [10]:
```
#Distributio of movie ratings
plt.figure(figsize=(10, 6))

# Histplot plot for the ratings
sns.histplot(df['rating'], bins=10, color='blue')

# Title and labels
plt.title('Distribution of Movie Ratings', color= 'blue', size=14, weight='bo
plt.xlabel('Movie Ratings',color= 'black', size=12, weight='bold')
plt.ylabel('Frequency',color= 'black', size=12, weight='bold')

# Display
plt.tight_layout()
plt.show()
```



4.2.2 Count of movie distribution by genre.

In [11]:
```
#Count plot
# Creating series of genres
genres_series = df['genres'].str.split('|').explode()

# Count plot
sns.countplot(data=genres_series, x=genres_series, palette='viridis')

# Title and labels
plt.title('Count of Movies by Genre', color= 'blue', size=14, weight='bold')
plt.xlabel('Genre',color= 'black', size=12, weight='bold')
plt.ylabel('Count',color= 'black', size=12, weight='bold')

# Rotate labels for better readability
plt.xticks(rotation=45, ha='right')

#Display
plt.tight_layout()
plt.show()
```



5.0 Modelling

Build a model that provides top 5 movie recommendations to a user, based on their ratings of other movies. This will be deployed to ....

Modelling packages

In [12]:
```
#Modelling packages
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.metrics import mean_squared_error, mean_absolute_error,accuracy_
from sklearn.metrics.pairwise import cosine_similarity
from surprise import KNNBasic, Reader, Dataset
from surprise.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from surprise import accuracy, SVD, Reader, Dataset
```

5.1. Item-Based Collaborative Filtering (item-item CF)

The model recommends a movie based on the similarity between items (movies).

Initializing Reader class and using .min() and .max() to set the rating range of the dataset(df), and convert the dataset(df) to Surprise dataset(data).

In [13]:
```
# Define the rating scale
reader = Reader(rating_scale=(df['rating'].min(), df['rating'].max()))

# Convert the df dataset to a Surprise dataset
data = Dataset.load_from_df(df[["userid", "movieid", "rating"]], reader)
```

Splitting Surprise dataset (data) into training and testing datasets, setting test size to 20% of the dataset and random state to 42 for reproducibility.

In [14]:
```
# Splitting the surprise dataset
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)
```

Defining cosine similarity for KNNBasic to measure the similarity between items (setting user_based = False to imply item_based).

In [15]:
```
# Define similarity options
sim_options = {
    'name': 'cosine',  # cosine similarity option to measure the similarity be
    'user_based': False  # Setting to False for item-based filtering
}
```

Initialize item_based collaborative filtering model using KNNBasic algorithm and train the

In [16]:
```
# Build the model using the KNNBasic
item_cf_model = KNNBasic(sim_options=sim_options)

# Train the model on the training set
item_cf_model.fit(trainset)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

Out[16]:
```
<surprise.prediction_algorithms.knns.KNNBasic at 0x22c621b4b50>
```

Creating a dictionary that maps movie IDs to coresponding movie titles, convert neighbors indices(indexing in the trainset) to original raw item ID and retrieve coresponding movie titles. Print out top 5 (k=5) similar items for a given items (say movie Id=296).

In [17]:
```
# Retrieve the neighbors for the specified movie ID
movieid =296 # item
k = 5        # Number of neighbors
neighbors = item_cf_model.get_neighbors(movieid, k=k)  # Get neighbors for a g
# Creating a dictionary of movieId to title
movie_titles_dict = dict(zip(df['movieid'], df['title']))

# Convert the neighbors indices back to original IDs
neighbors_original_ids = [trainset.to_raw_iid(i) for i in neighbors]

# Map the neighbor movie IDs to movie titles using list comprehension
neighbors_titles = [movie_titles_dict.get(movie_id) for movie_id in neighbors_

#Top 5 neighbors for item 296
print("Top 5 neighbors for item 296:")
for i, title in enumerate(neighbors_titles, 1):
    movie_id = neighbors_original_ids[i - 1]  # Getting the original movie Id
    print(f"{i}. Movie ID: {movie_id}, Title: {title}")
```

```
Top 5 neighbors for item 296:
1. Movie ID: 4024, Title: House of Mirth, The (2000)
2. Movie ID: 58047, Title: Definitely, Maybe (2008)
3. Movie ID: 3330, Title: Splendor in the Grass (1961)
4. Movie ID: 892, Title: Twelfth Night (1996)
5. Movie ID: 2390, Title: Little Voice (1998)
```

5.2. Content Based Filtering Recommendation System

This model uses item features to recommend other items similar to user preferences, based on their previous ratings.

Limit the original dataset to 10000 rows to reduce computational complexity, and create and feature engineer 'content' which gives an enriched textual description of each movie based on its genres and associated tags for feature extraction.

In [18]:
```
# Subset the original dataset to 10000 rows and feature engineer 'content' fe
df_subset = df.head(10000)
df_subset['content'] = df_subset['genres'] + ' ' + df_subset['tag']
```

Create TF-IDF(Term Frequency-Inverse Document Frequency) to transform the combined textual content into a matrix of numerical features.

In [19]:
```
#Create a TF-IDF representation of the 'content' column
tfidf = TfidfVectorizer(stop_words='english')
# Convert the content column into a matrix of TF-IDF features
tfidf_matrix = tfidf.fit_transform(df_subset['content'])
```

Compute cosine similarity between all pairs of movies.

In [20]:
```
# Calculate Cosine Similarity between movies
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

Remove duplicates based on movie Id to ensure that each movie only appears once in the DataFrame,useful for getting distinct movie recommendations, and reset index.

In [21]:
```
# Drop duplicates based on 'movieid', keeping only one entry per movie
df_unique = df.drop_duplicates(subset=['movieid'])
# Reset the index of df_unique
df_unique = df_unique.reset_index(drop=True)
```

Define a function to recommend top 5 similar movies based on a given movie Id. The function takes movie Id as as an argument and returns a list of the top 5 most similar movies based on cosine similarity.

In [22]:
```
# Create a function to recommend movies based on movieid
def recommend_content_based(movieid, cosine_sim=cosine_sim, top_n=5):
    # Get the index of the movie that matches the movieid
    idx = df_unique.index[df_unique['movieid'] == movieid].tolist()[0]

    # Get the pairwise similarity scores for the movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on similarity score (highest first)
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the top 5 most similar movies
    sim_scores = sim_scores[1:top_n+1]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 5 most similar movies
    recommended_content_based = df_unique.iloc[movie_indices][['movieid', 'ti

    return recommended_content_based
```

Calling the function to recommend movies for movieid = 3.

In [23]:
```
recommend_content_based(3)
```

Out[23]:

	movieid	title
3	50	Usual Suspects, The (1995)
5	110	Braveheart (1995)
7	223	Clerks (1994)
9	260	Star Wars: Episode IV - A New Hope (1977)
11	316	Stargate (1994)

5.3. Matrix Factorization with Singular Value Decomposition (SVD)

The model predicts the ratings for the unrated movies, and recommend movies to users based on their past ratings and similar preferences of other users.

Instantiating reader and loading data.

In [24]:
```
# Instantiate reader and load the data
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(df[['userid', 'movieid', 'rating']], reader)
```

Generate training set, initialize and train SVD model on train set.

In [25]:
```
#Generate trainset and train an SVD model
trainset = data.build_full_trainset()
svd = SVD()
svd.fit(trainset)
```

Out[25]: <surprise.prediction_algorithms.matrix_factorization.SVD at 0x22c5fc94a60>

Extracting a list of unique movie Ids from the dataframe.

In [26]:
```
# List of all movie Ids
all_movie_ids = df['movieid'].unique()
```

Defining a function to recommend top 5 movies to a given user based on predictions made by the trained Singular Value Decomposition (SVD) model.

In [27]:
```
# Define the svd function
def svd_recommendations(user_id, svd_model, all_movie_ids, top_n=5):
    # Predict ratings for all movies for the given user
    rating_pred = [svd_model.predict(user_id, movie_id) for movie_id in all_m

    # Sort predictions
    sorted_pred = sorted(rating_pred, key=lambda x: x.est, reverse=True)

    # Extract top_n recommended movie Ids
    recommended_movie_ids = [prediction.iid for prediction in sorted_pred[:to

    # Remove duplicates from DataFrame
    clean_df = df.drop_duplicates(subset='movieid')

    # Map movie Ids to movie titles
    svd_recommended_movies = clean_df[clean_df['movieid'].isin(recommended_mo

    return svd_recommended_movies
```

Generate and print top 5 movies recommended for a specified user.

In [28]:
```
# Extract top 5 movie recommendations for a specific user
user_id = 5
top_5_movies_svd = svd_recommendations(user_id, svd, all_movie_ids, top_n=5)

# Top 5 movie recommendations
print("Top 5 Recommended Movies: \n", top_5_movies_svd)
```

```
Top 5 Recommended Movies: 
        movieid                                          title  rating  \
12388      296                             Pulp Fiction (1994)     3.0   
134038    2324  Life Is Beautiful (La Vita è bella) (1997)     1.0   
187651    1193          One Flew Over the Cuckoo's Nest (1975)     4.0   
207585  104879                                Prisoners (2013)     3.0   
227909  174053           Black Mirror: White Christmas (2014)     5.0   

                                   genres  
12388          Comedy|Crime|Drama|Thriller  
134038            Comedy|Drama|Romance|War  
187651                               Drama  
207585              Drama|Mystery|Thriller  
227909  Drama|Horror|Mystery|Sci-Fi|Thriller  
```

6.0 Model Evaluations

Evaluations for all models shall be based on mean absolute error (MAE) and mean squared error(MSE). Comparison of the two parameter and criteria for picking the best performing model shall be on the lowest MAE and MSE.

6.1. Evaluation of Item-Based Recommendation Model

In [29]:
```

# Predict ratings on the testset
item_cf_predictions = item_cf_model.test(testset)

# Evaluate using MSE and MAE
item_cf_mse = accuracy.mse(item_cf_predictions)
item_cf_mae = accuracy.mae(item_cf_predictions)
#

# print(f'Item-based CF - MSE: {item_cf_mse:.4f}')
# print(f'Item-based CF - MAE: {item_cf_mae:.4f}')
```

```
MSE: 0.3424
MAE:  0.3594
```

6.2. Evaluation of Content-Based Recommendation Model

In [30]:
```
# Extract the actual ratings from the 'rating' column of df_unique
actual_ratings = df_unique['rating'].values

# Initialize an empty list to store the predicted ratings
predicted_ratings = []

# Loop through each movie in the dataset get recommendations and evaluate(Lim
for idx in range(1000):
    movieid = df_unique.iloc[idx]['movieid']  # Extracting movieid for the mo

    # Getting the recommended movies using content-based filtering
    recommended_movies = recommend_content_based(movieid, cosine_sim, top_n=5

    # Predict the rating based on the mean of the ratings of the recommended
    # List of movie ids for the recommended movies
    recommended_movie_ids = recommended_movies['movieid'].values

    # Ratings for the recommended movie ids
    recommended_ratings = df_unique[df_unique['movieid'].isin(recommended_mov

    # Taking the mean
    predicted_rating = np.nanmean(recommended_ratings)
    predicted_ratings.append(predicted_rating)

# Calculate MSE and MAE
content_based_mse = mean_squared_error(actual_ratings[:1000], predicted_ratin
content_based_mae = mean_absolute_error(actual_ratings[:1000], predicted_rati

# Print out the evaluation metrics
print(f"Mean Squared Error (MSE): {content_based_mse: .4f}")
print(f"Mean Absolute Error (MAE): {content_based_mae: .4f}")
```

```
Mean Squared Error (MSE):  1.2797
Mean Absolute Error (MAE):  0.8500
```

6.3. Evaluation of Singular Value Decomposition (SVD) Model

In [31]:
```
# Predict ratings on the testset
svd_predictions = svd.test(testset)

# Evaluate using MSE and MAE
svd_mse = accuracy.mse(svd_predictions)
svd_mae = accuracy.mae(svd_predictions)
```

```
MSE: 0.0929
MAE:  0.2071
```

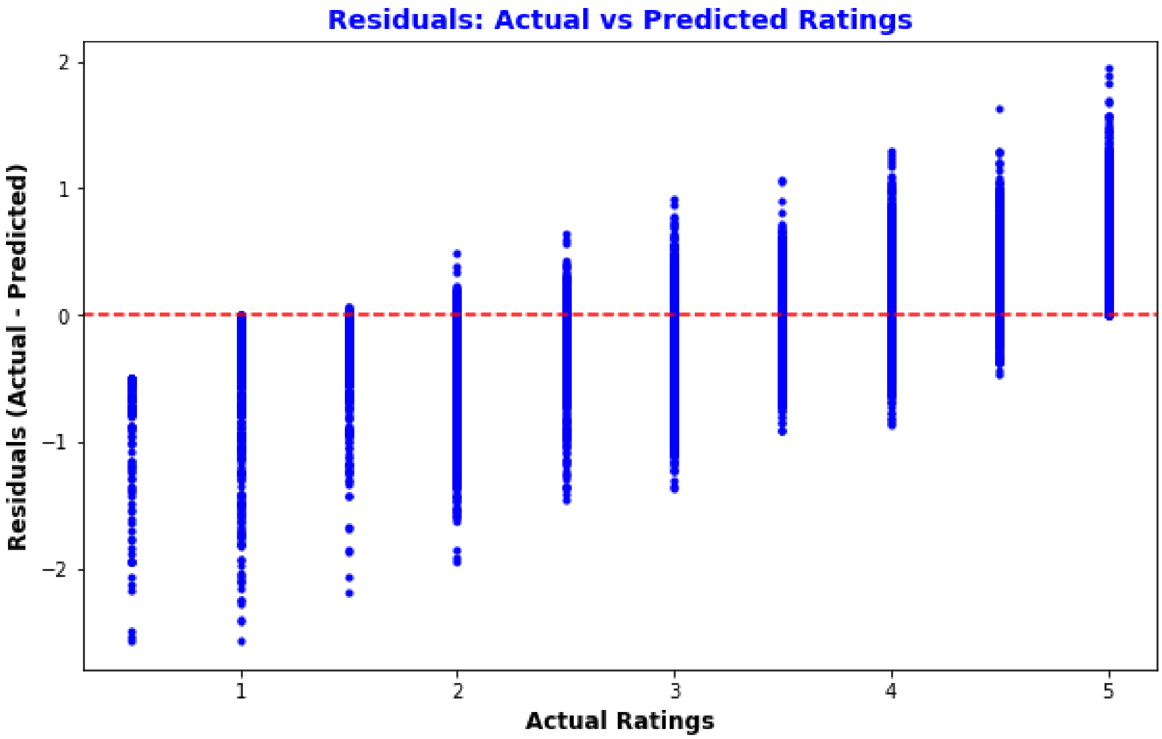
6.4 Best Model Performances

The best performing model comparing mean squared error(MSE) and mean absolute error(MAE) among the three models is the **Matrix Factorization with Singular Value Decomposition(SVD) model**. It had the lowest MSE and MAE.

6.4.1. Visual on Residuals

In [32]:
```
# Prepare for storing predictions and actuals
predictions = []
actuals = []

# Loop through testset to predict ratings for each user-item pair
for uid, iid, true_r in testset:
    # Predict the rating for the user-item pair
    prediction = svd.predict(uid, iid)

    # Store predicted and actual ratings
    predictions.append(prediction.est)
    actuals.append(true_r)

# Convert lists to numpy arrays for easier handling
predictions = np.array(predictions)
actuals = np.array(actuals)

# Calculate residuals (actual - predicted)
residuals = actuals - predictions
```

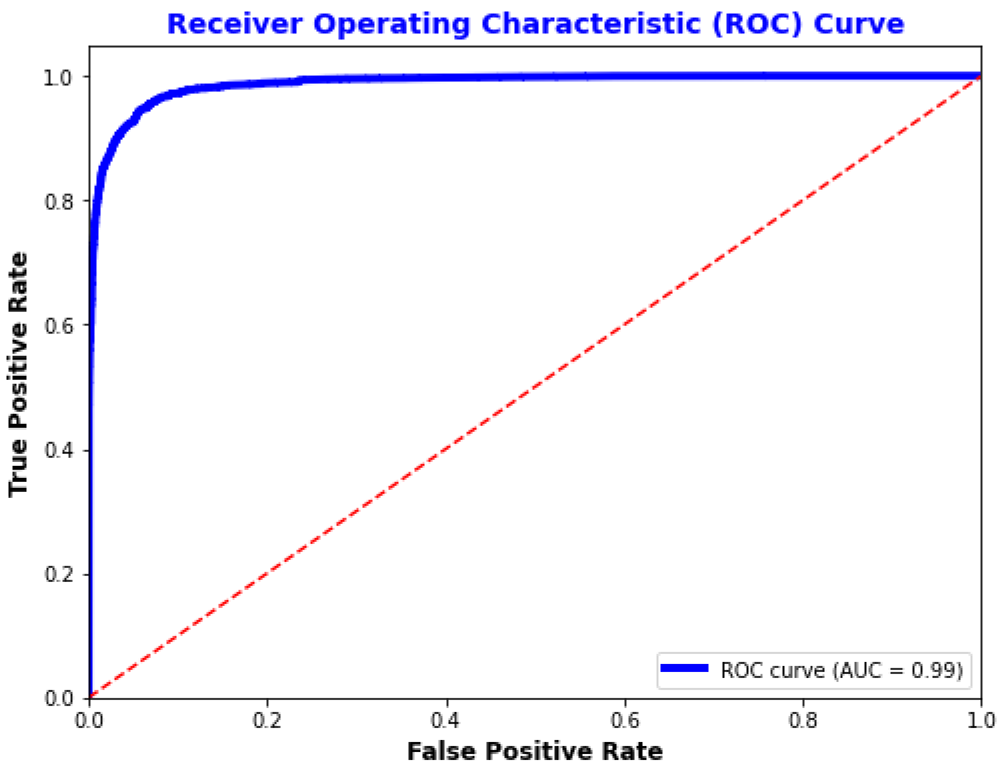
In [33]:
```
# Visualize the residuals
plt.figure(figsize=(10, 6))
plt.scatter(actuals, residuals, color='blue', s=10)
plt.axhline(y=0, color='red', linestyle='--')
plt.title('Residuals: Actual vs Predicted Ratings', color='blue', size=14, we
plt.xlabel('Actual Ratings', color='black', size=12, weight='bold')
plt.ylabel('Residuals (Actual - Predicted)',color='black', size=12, weight='b
plt.show()
```



6.4.2. Receiver Operating Characteristic (ROC) Curve, Area under the Curve (AUC) visual

In [34]:
```
# Consider ratings > 3 as relevant)
threshold = 3.0

# Predict ratings for each user-item pair in the testset using the trained SVI
predictions = svd.test(testset)

# Create true labels and predicted scores
y_true = []
y_pred = []

# Loop through the predictions to calculate true labels
for uid, iid, true_r, est, _ in predictions:
    # 1 for relevant (rating > threshold), 0 for irrelevant (rating <= thresh
    y_true.append(1 if true_r > threshold else 0)
    y_pred.append(est)  # Predicted rating

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_true, y_pred)
roc_auc = auc(fpr, tpr)
```

In [35]:
```
# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=4, label=f'ROC curve (AUC = {roc_auc:.2f}
plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # Diagonal line for ra
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.title('Receiver Operating Characteristic (ROC) Curve', color='blue', size
plt.xlabel('False Positive Rate',color='black', size=12, weight='bold')
plt.ylabel('True Positive Rate',color='black', size=12, weight='bold')

plt.legend(loc='lower right')
plt.show()

# Output AUC
print(f'AUC: {roc_auc:.2f}')
```




```
AUC: 0.99
```

Observations made when comparing genres of the top 5 rated movies by user (userid =5) with that of the model prediction of similar movies, shows clearly there exists a big overlap.

In [36]:
```
#Genres for top 5 rated movies by user (5)
df[df['userid']==5]['genres'].value_counts().head(5)
```

Out[36]:
```
genres
Comedy|Crime|Drama|Thriller    178
Action|Drama|War                10
Comedy|Romance                   9
Action|Sci-Fi                    8
Crime|Drama                      8
Name: count, dtype: int64
```

In [37]:
```
#Genres of top 5 similar movies predicted for user(5) by the model.
top_5_movies_svd['genres']
```

Out[37]:
```
12388              Comedy|Crime|Drama|Thriller
134038                Comedy|Drama|Romance|War
187651                                   Drama
207585                  Drama|Mystery|Thriller
227909    Drama|Horror|Mystery|Sci-Fi|Thriller
Name: genres, dtype: object
```

7.0 Conclusions and Recommendations

Best recommendtion system model for deployment is **Matrix Factorization with Singular Value Decomposistion(SVD)** by the movie shop.

Model performance with an AUC (Area Under the Curve) = 0.97 indicates that the model highly distinguishes between relevant and irrelevant recommendations, which translates to higher precision and higher recall. The model ranks the relevant movies higher in the list, ensuring that the user gets personalized and accurate recommendations.

This ensures that;

- It best recommends movies based on what customers have done, liked, or searched (personalized recommendations).
- Help customers discover movies that they may have never considered but might like and thus increase their tastes and knowledge of films.
- Incentivize customers to spend more time on the website with value-added recommendations relevant to their interests.
- Personalized suggestions will increase sales and improve customer retention, as they will revisit your site for more and remain longer-term.
- Facilitate an easy and smooth recommendation experience for your customers.