EDS DATASET ACTIVITY

Name:- Tanish A Rayalkar

PRN:- 202401070062

Roll No:- ET1-34

Division:- ET1

Problem 1: Find the total number of unique movies.

Solution: total_movies =

movies['movieId'].nunique()

print(total_movies)

Problem 2: Find the total number of unique users.

Solution: total_users =

ratings['userId'].nunique()

print(total_users)

Problem 3: Find the average rating across all movies.

Solution: average_rating =

ratings['rating'].mean()

print(average_rating)

Problem 4: Find the highest rated movie(s) (with at least 50 ratings).

Solution: movie_ratings = ratings.groupby('movieId').agg({'rating':

['mean', 'count']}) filtered = movie_ratings[movie_ratings[('rating', 'count')]

>= 50] highest_rated = filtered[('rating', 'mean')].idxmax()

print(movies[movies['movieId'] == highest_rated])

Problem 5: List the top 5 movies with the most ratings.

Solution: most_rated =

ratings['movieId'].value_counts().head(5) top_movies =

movies[movies['movieId'].isin(most_rated.index)]

print(top_movies)

Problem 6: Find how many distinct genres are there.

Solution: genres =

movies['genres'].str.split('|').explode().unique()

print(len(genres), genres)

Problem 7: Find the number of movies with 'Comedy' genre.

Solution: comedy_movies =

movies[movies['genres'].str.contains('Comedy')]

print(len(comedy_movies))

Problem 8: Find the earliest and latest year of movies in the dataset.

Solution:

movies['year'] = movies['title'].str.extract(r'\((\d{4})\)')

movies['year'] = pd.to_numeric(movies['year'], errors='coerce')

earliest = movies['year'].min() latest = movies['year'].max()

print(earliest, latest)

Problem 9: Find the average rating per genre.

Solution:

movies_genres = movies.copy() movies_genres =

movies_genres.assign(genres=movies_genres['genres'].str.split('|')).explode('genres') merged =

pd.merge(ratings, movies_genres, on='movieId') average_rating_per_genre =

merged.groupby('genres')['rating'].mean() print(average_rating_per_genre)

Problem 10: Find which user has given the maximum number of ratings.

Solution: top_user =

ratings['userId'].value_counts().idxmax()

print(top_user)

Problem 11: How many users have rated more than 1000 movies?

Solution: heavy_raters =

ratings['userId'].value_counts()

print((heavy_raters > 1000).sum())

Problem 12: Find the most common tag assigned to any movie.

Solution: most_common_tag =

tags['tag'].value_counts().idxmax()

print(most_common_tag)

Problem 13: Find the number of movies which have at least one tag.

Solution: movies_with_tags =

tags['movieId'].nunique()

print(movies_with_tags)

Problem 14: Find movies that have never been rated.

Solution: rated_movie_ids = ratings['movieId'].unique()

unrated_movies = movies[~movies['movieId'].isin(rated_movie_ids)]

print(unrated_movies)

Problem 15: Find users who have rated only a single movie.

Solution: single_rating_users =

ratings['userId'].value_counts() single_users =

single_rating_users[single_rating_users == 1]

print(single_users.index.tolist())

Problem 16: Find the top 5 movies with highest average rating (at least 100 ratings).

Solution:
avg_rating = ratings.groupby('movieId').agg({'rating': ['mean', 'count']})

avg_rating.columns = ['avg_rating', 'rating_count']

top_avg_movies = avg_rating[avg_rating['rating_count'] >= 100] .sort_values('avg_rating',

ascending=False).head(5) print(movies[movies['movieId'].isin(top_avg_movies.index)])

Problem 17: Create a pivot table of user vs movie with ratings as values.

Solution:

pivot_table = ratings.pivot_table(index='userId', columns='movieId', values='rating')

print(pivot_table)

Problem 18: Find the correlation between number of ratings and average rating for movies.

Solution: rating_stats = ratings.groupby('movieId').agg({'rating':

['mean', 'count']}) rating_stats.columns = ['avg_rating', 'num_ratings']

```
correlation = rating_stats['avg_rating'].corr(rating_stats['num_ratings'])

print(correlation)
```

Problem 19: Find how many movies have average rating above 4.5.

Solution: avg_ratings =

```
ratings.groupby('movieId')['rating'].mean()

high_avg_movies = avg_ratings[avg_ratings > 4.5]

print(len(high_avg_movies))
```

Problem 20: Calculate the percentage of movies that belong to the 'Action' genre.

Solution:

```
action_movies = movies[movies['genres'].str.contains('Action')]

percentage_action = (len(action_movies) / len(movies)) * 100

print(f"{percentage_action:.2f}%")
```