```
import pandas as pd
links_df = pd.read_csv('links_small.csv')
```

```
links_df = links_df.dropna(subset = ['tmdbId'])
links_df
```

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0
...	...	...	...
9120	162672	3859980	402672.0
9121	163056	4262980	315011.0
9122	163949	2531318	391698.0
9123	164977	27660	137608.0
9124	164979	3447228	410803.0

9112 rows × 3 columns

```
ratings_df = pd.read_csv('ratings_small.csv')
ratings_df
```

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205
...	...	...	...	...
99999	671	6268	2.5	1065579370
100000	671	6269	4.0	1065149201
100001	671	6365	4.0	1070940363
100002	671	6385	2.5	1070979663
100003	671	6565	3.5	1074784724

100004 rows × 4 columns

```
average_ratings = ratings_df.groupby('movieId')['rating'].mean().round(1)
average_ratings
```

```
movieId
1         3.9
2         3.4
3         3.2
4         2.4
5         3.3
         ...
161944    5.0
162376    4.5
162542    5.0
162672    3.0
163949    5.0
Name: rating, Length: 9066, dtype: float64
```

```
average_ratings = average_ratings.apply(lambda x: round(x * 2) / 2)
average_ratings
```

```
movieId
1         4.0
2         3.5
3         3.0
4         2.5
```

```
5         3.5
         ...
161944    5.0
162376    4.5
162542    5.0
162672    3.0
163949    5.0
Name: rating, Length: 9066, dtype: float64
```

```
average_ratings_df = pd.DataFrame({'movieId': average_ratings.index,
'average_rating': average_ratings.values})
average_ratings_df
```

	movieId	average_rating
0	1	4.0
1	2	3.5
2	3	3.0
3	4	2.5
4	5	3.5
...	...	...
9061	161944	5.0
9062	162376	4.5
9063	162542	5.0
9064	162672	3.0
9065	163949	5.0

9066 rows × 2 columns

```
result_df = pd.merge(links_df, average_ratings_df, on='movieId',
how='inner')
# The original Data is displayed
result_df
```

	movieId	imdbId	tmdbId	average_rating
0	1	114709	862.0	4.0
1	2	113497	8844.0	3.5
2	3	113228	15602.0	3.0
3	4	114885	31357.0	2.5
4	5	113041	11862.0	3.5
...	...	...	...	...
9048	161944	255313	159550.0	5.0
9049	162376	4574334	410612.0	4.5
9050	162542	5165344	392572.0	5.0
9051	162672	3859980	402672.0	3.0
9052	163949	2531318	391698.0	5.0

9053 rows × 4 columns

```
result_df.to_csv('my_dataframe.csv', index=False)
result_df
```

	movieId	imdbId	tmdbId	average_rating
0	1	114709	862.0	4.0
1	2	113497	8844.0	3.5
2	3	113228	15602.0	3.0
4	5	113041	11862.0	3.5

```
movies_metadata = pd.read_csv('/content/movies_metadata.csv', low_memory=False)
```

```
import pandas as pd


result_df = pd.read_csv('my_dataframe.csv', low_memory=False)


movies_df = pd.read_csv('movies_metadata.csv', low_memory=False)

# Limit to the first 9054 rows
movies_df = movies_df.head(9054)

movies_df['release_date'] = pd.to_datetime(movies_df['release_date'], errors='coerce')

# 1. SeasonReleased as Integers
movies_df['SeasonReleased'] = pd.to_datetime(movies_df['release_date']).dt.month

# 2. Simplified Popularity (as an integer)
default_popularity_value = 0  # You can set this to any default value you prefer
movies_df['simplifiedPopularity'] = pd.to_numeric(movies_df['popularity'], errors='coerce').fillna(default_popularity_value).round(0).as

# Convert 'revenue' and 'budget' columns to numeric values, coercing errors to NaN
movies_df['revenue'] = pd.to_numeric(movies_df['revenue'], errors='coerce')
movies_df['budget'] = pd.to_numeric(movies_df['budget'], errors='coerce')

# Calculate profit
movies_df['profit'] = movies_df['revenue'] - movies_df['budget']

# Assign numerical values to profitability based on the 'profit' column
movies_df['profitability_numeric'] = pd.cut(movies_df['profit'],
                                             bins=[float('-inf'), 0, float('inf')],
                                             labels=[-1, 0])

# Handling data types and missing values
movies_df['vote_count'] = pd.to_numeric(movies_df['vote_count'], errors='coerce')
movies_df['vote_average'] = pd.to_numeric(movies_df['vote_average'], errors='coerce')

# Calculate the maximum vote count for normalization
max_vote_count = movies_df['vote_count'].max()

# Calculate the weighted popularity score
movies_df['popularity_score'] = (movies_df['vote_count'] * movies_df['vote_average']) / max_vote_count

# Round the 'popularity_score' to 2 decimal places
movies_df['popularity_score'] = movies_df['popularity_score'].round(2)

# Extracting specific columns from 'movies_df'
new_data = movies_df[['SeasonReleased', 'simplifiedPopularity', 'profitability_numeric', 'popularity_score']]

# Concatenate the 'result_df' and 'new_data' DataFrames along the columns (axis=1)
result_df = pd.concat([result_df, new_data], axis=1)

# Ensure that 'result_df' has a maximum of 9054 rows
result_df = result_df.head(9054)

# Save the updated 'result_df' to a CSV file
result_df.to_csv('attributeresult.csv', index=False)

# Displaying the updated DataFrame
print(result_df)
```

```
       movieId     imdbId    tmdbId  average_rating  SeasonReleased  \
0          1.0   114709.0     862.0             4.0            10.0   
1          2.0   113497.0    8844.0             3.5            12.0   
2          3.0   113228.0   15602.0             3.0            12.0   
3          4.0   114885.0   31357.0             2.5            12.0   
4          5.0   113041.0   11862.0             3.5             2.0   
...        ...        ...       ...             ...             ...   
9049  162376.0  4574334.0  410612.0             4.5            11.0   
9050  162542.0  5165344.0  392572.0             5.0             1.0   
9051  162672.0  3859980.0  402672.0             3.0             9.0   
9052  163949.0  2531318.0  391698.0             5.0             1.0   
9053       NaN        NaN       NaN             NaN            12.0   
```

```
      simplifiedPopularity profitability_numeric  popularity_score
0                       22                     0              4.31
1                       17                     0              1.72
2                       12                    -1              0.06
3                        4                     0              0.02
4                        8                     0              0.10
...                    ...                   ...               ...
9049                     2                    -1              0.02
9050                     0                    -1              0.00
9051                     3                    -1              0.02
9052                     0                    -1              0.00
9053                     6                     0              0.09

[9054 rows x 8 columns]
```