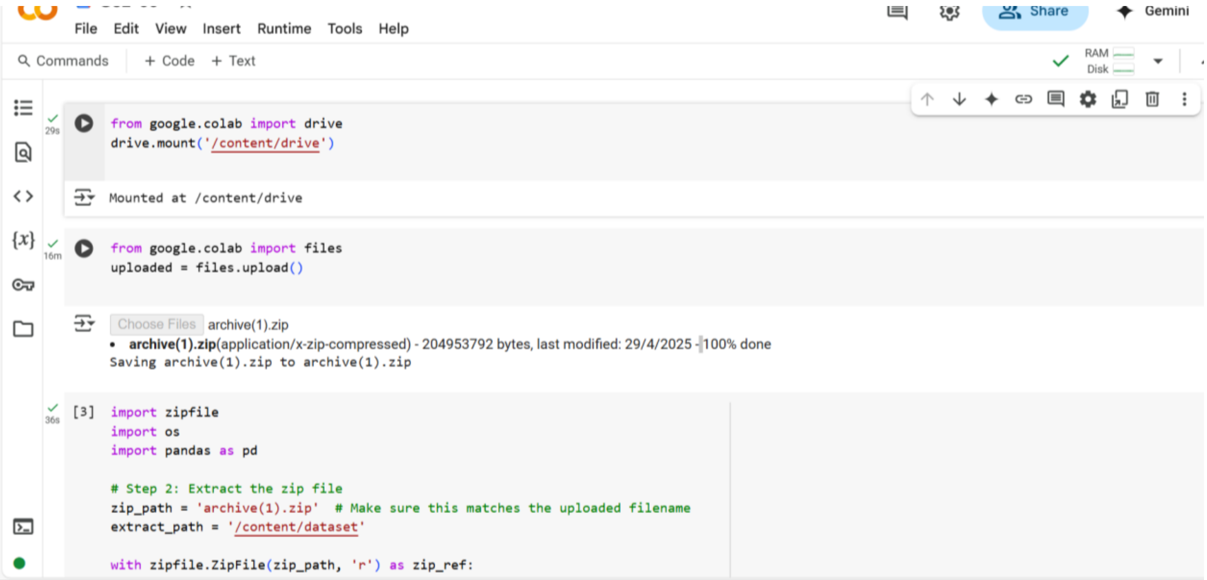
Name:-Priyal S. Mahajan

Roll No.:-CS3-76

PRN:- 202401040203

- EDS ACTIVITY-2



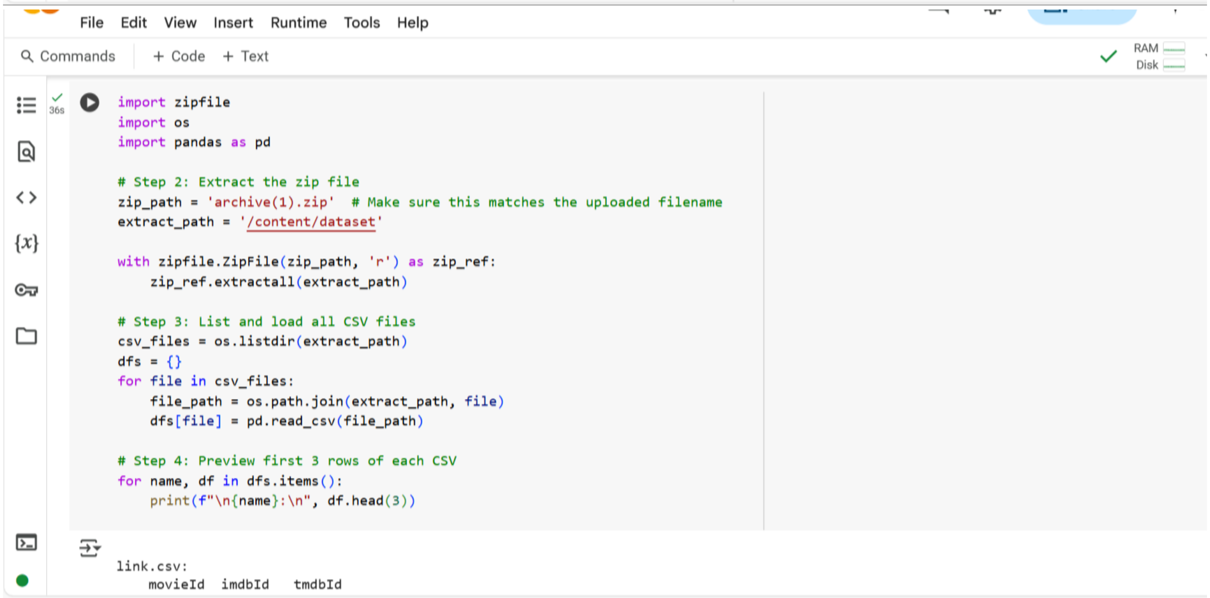
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from google.colab import files
uploaded = files.upload()
```

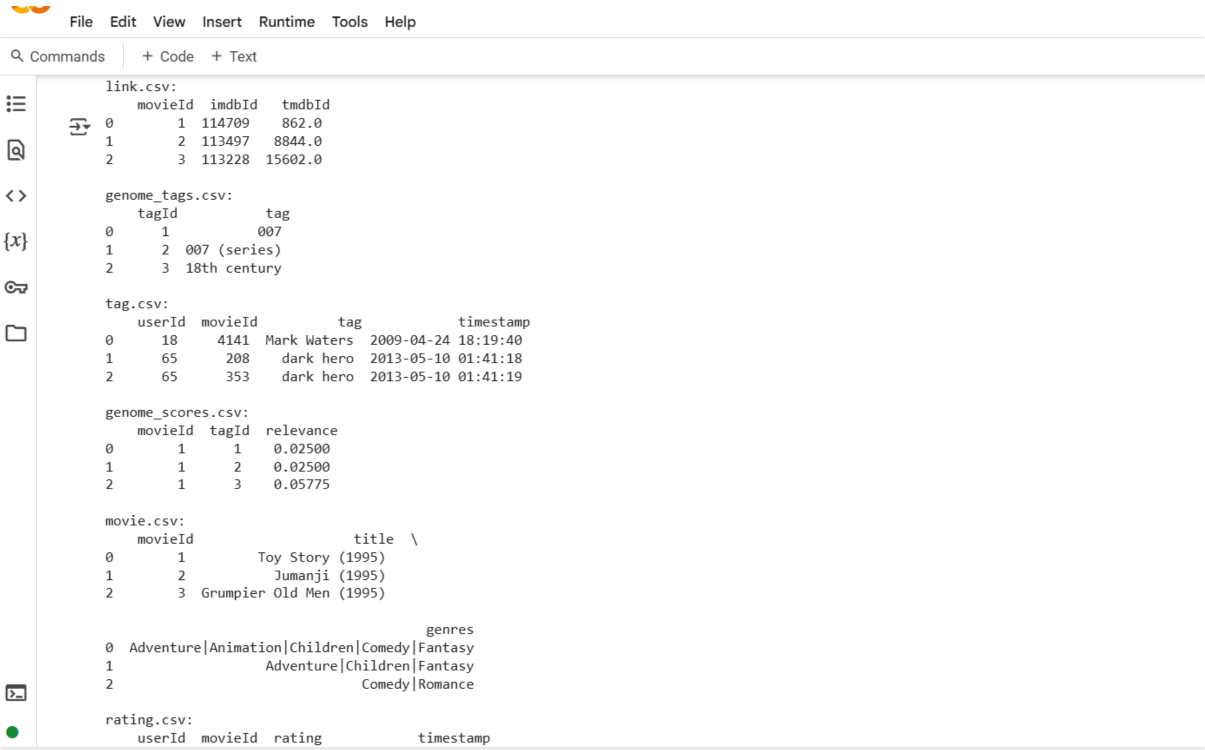
```
Choose Files archive(1).zip
• archive(1).zip(application/x-zip-compressed) - 204953792 bytes, last modified: 29/4/2025 - 100% done
Saving archive(1).zip to archive(1).zip
```

```
[3] import zipfile
    import os
    import pandas as pd

    # Step 2: Extract the zip file
    zip_path = 'archive(1).zip'  # Make sure this matches the uploaded filename
    extract_path = '/content/dataset'

    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
```

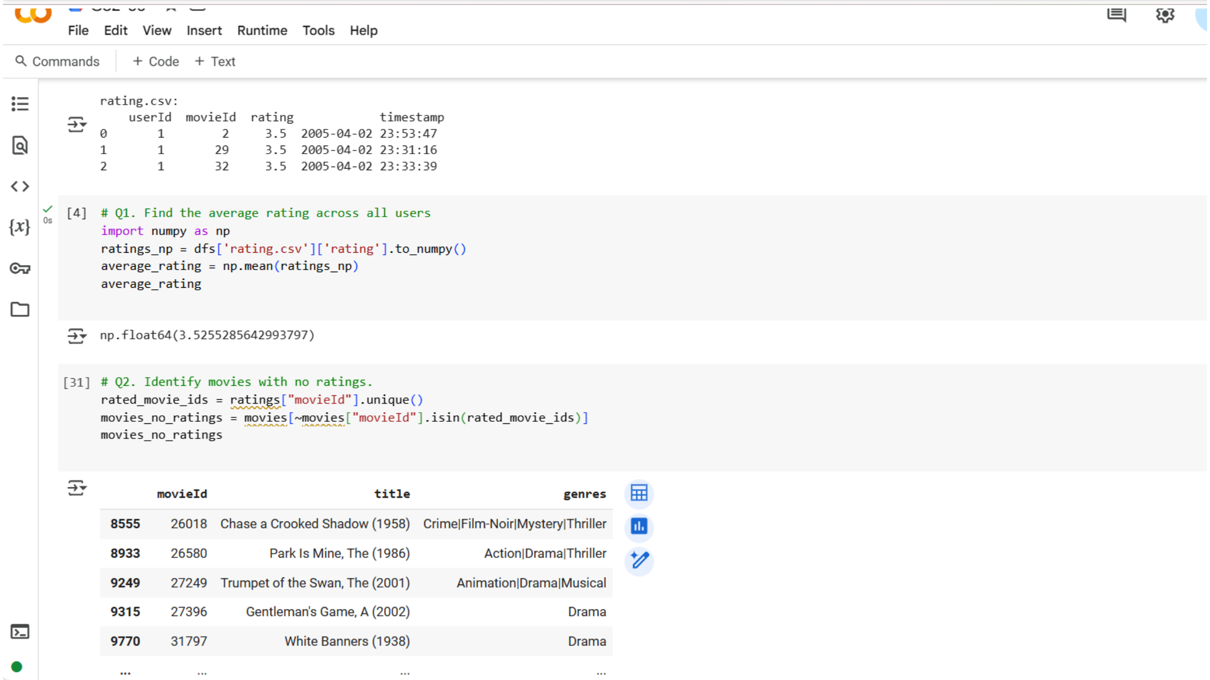


```
import zipfile
import os
import pandas as pd

# Step 2: Extract the zip file
zip_path = 'archive(1).zip'  # Make sure this matches the uploaded filename
extract_path = '/content/dataset'

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# Step 3: List and load all CSV files
csv_files = os.listdir(extract_path)
dfs = {}
for file in csv_files:
    file_path = os.path.join(extract_path, file)
    dfs[file] = pd.read_csv(file_path)

# Step 4: Preview first 3 rows of each CSV
for name, df in dfs.items():
    print(f"\n{name}:\n", df.head(3))
```

```
link.csv:
    movieId  imdbId   tmdbId
```



```
link.csv:
    movieId  imdbId   tmdbId
0        1  114709    862.0
1        2  113497   8844.0
2        3  113228  15602.0

genome_tags.csv:
    tagId           tag
0      1           007
1      2  007 (series)
2      3  18th century

tag.csv:
    userId  movieId          tag            timestamp
0      18     4141  Mark Waters  2009-04-24 18:19:40
1      65      208    dark hero  2013-05-10 01:41:18
2      65      353    dark hero  2013-05-10 01:41:19

genome_scores.csv:
    movieId  tagId  relevance
0        1      1    0.02500
1        1      2    0.02500
2        1      3    0.05775

movie.csv:
    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

rating.csv:
    userId  movieId  rating            timestamp
```



```
rating.csv:
    userId  movieId  rating            timestamp
0        1        2     3.5  2005-04-02 23:53:47
1        1       29     3.5  2005-04-02 23:31:16
2        1       32     3.5  2005-04-02 23:33:39
```

[4]

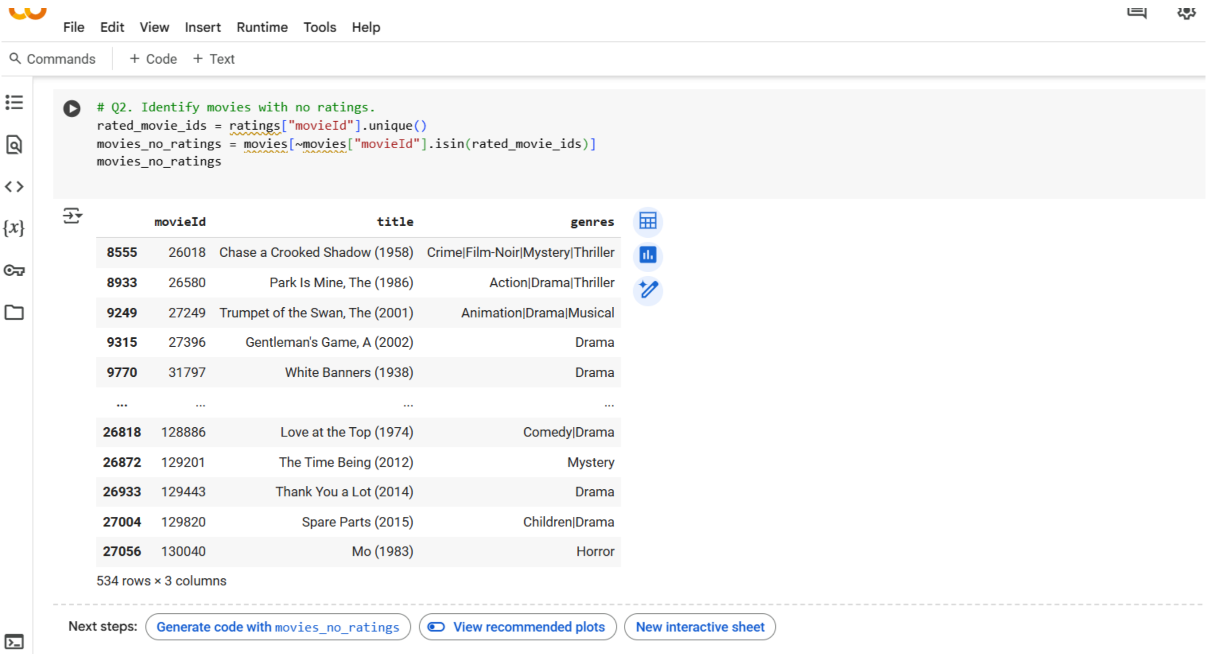
```
# Q1. Find the average rating across all users
import numpy as np
ratings_np = dfs['rating.csv']['rating'].to_numpy()
average_rating = np.mean(ratings_np)
average_rating
```

```
np.float64(3.5255285642993797)
```

[31]

```
# Q2. Identify movies with no ratings.
rated_movie_ids = ratings["movieId"].unique()
movies_no_ratings = movies[~movies["movieId"].isin(rated_movie_ids)]
movies_no_ratings
```

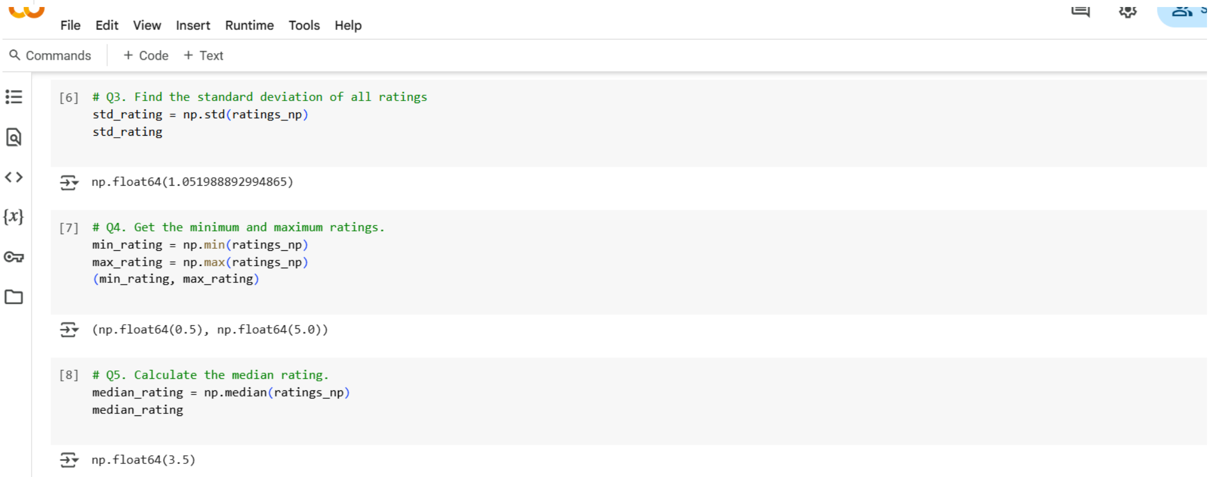
	movieId	title	genres
8555	26018	Chase a Crooked Shadow (1958)	Crime|Film-Noir|Mystery|Thriller
8933	26580	Park Is Mine, The (1986)	Action|Drama|Thriller
9249	27249	Trumpet of the Swan, The (2001)	Animation|Drama|Musical
9315	27396	Gentleman's Game, A (2002)	Drama
9770	31797	White Banners (1938)	Drama
...	...	...	...



```python
# Q2. Identify movies with no ratings.
rated_movie_ids = ratings["movieId"].unique()
movies_no_ratings = movies[~movies["movieId"].isin(rated_movie_ids)]
movies_no_ratings
```

	movieId	title	genres
8555	26018	Chase a Crooked Shadow (1958)	Crime|Film-Noir|Mystery|Thriller
8933	26580	Park Is Mine, The (1986)	Action|Drama|Thriller
9249	27249	Trumpet of the Swan, The (2001)	Animation|Drama|Musical
9315	27396	Gentleman's Game, A (2002)	Drama
9770	31797	White Banners (1938)	Drama
...	...	...	...
26818	128886	Love at the Top (1974)	Comedy|Drama
26872	129201	The Time Being (2012)	Mystery
26933	129443	Thank You a Lot (2014)	Drama
27004	129820	Spare Parts (2015)	Children|Drama
27056	130040	Mo (1983)	Horror

534 rows × 3 columns



```python
[6] # Q3. Find the standard deviation of all ratings
    std_rating = np.std(ratings_np)
    std_rating
```

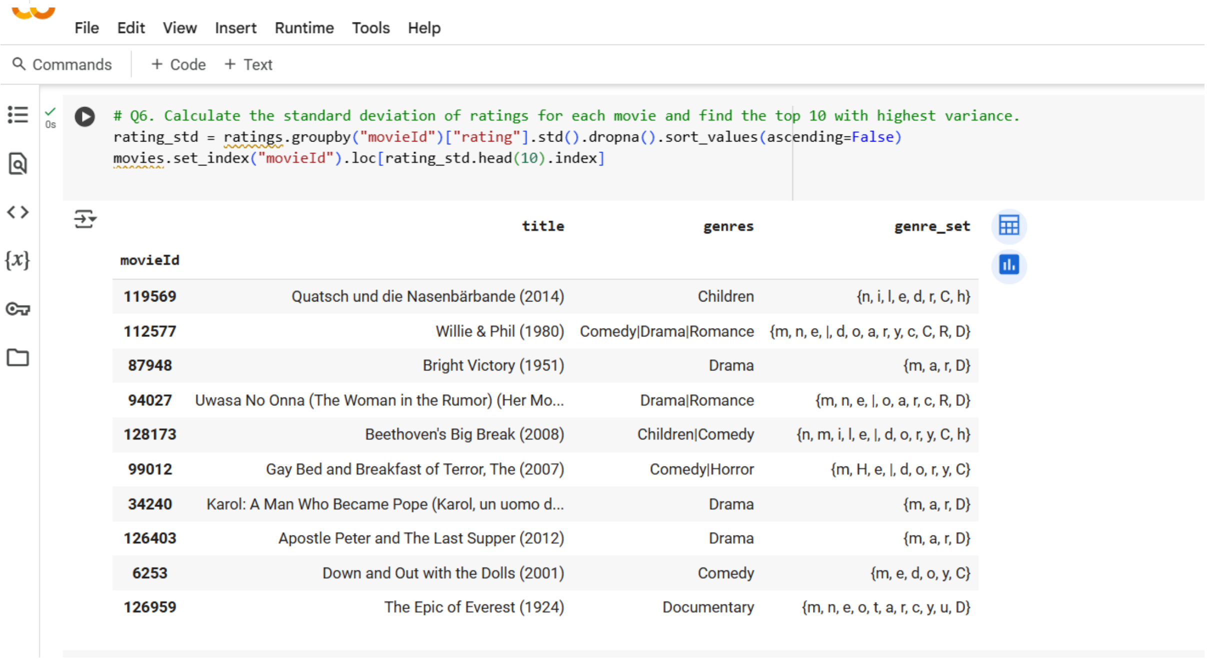
```
np.float64(1.051988892994865)
```

```python
[7] # Q4. Get the minimum and maximum ratings.
    min_rating = np.min(ratings_np)
    max_rating = np.max(ratings_np)
    (min_rating, max_rating)
```

```
(np.float64(0.5), np.float64(5.0))
```

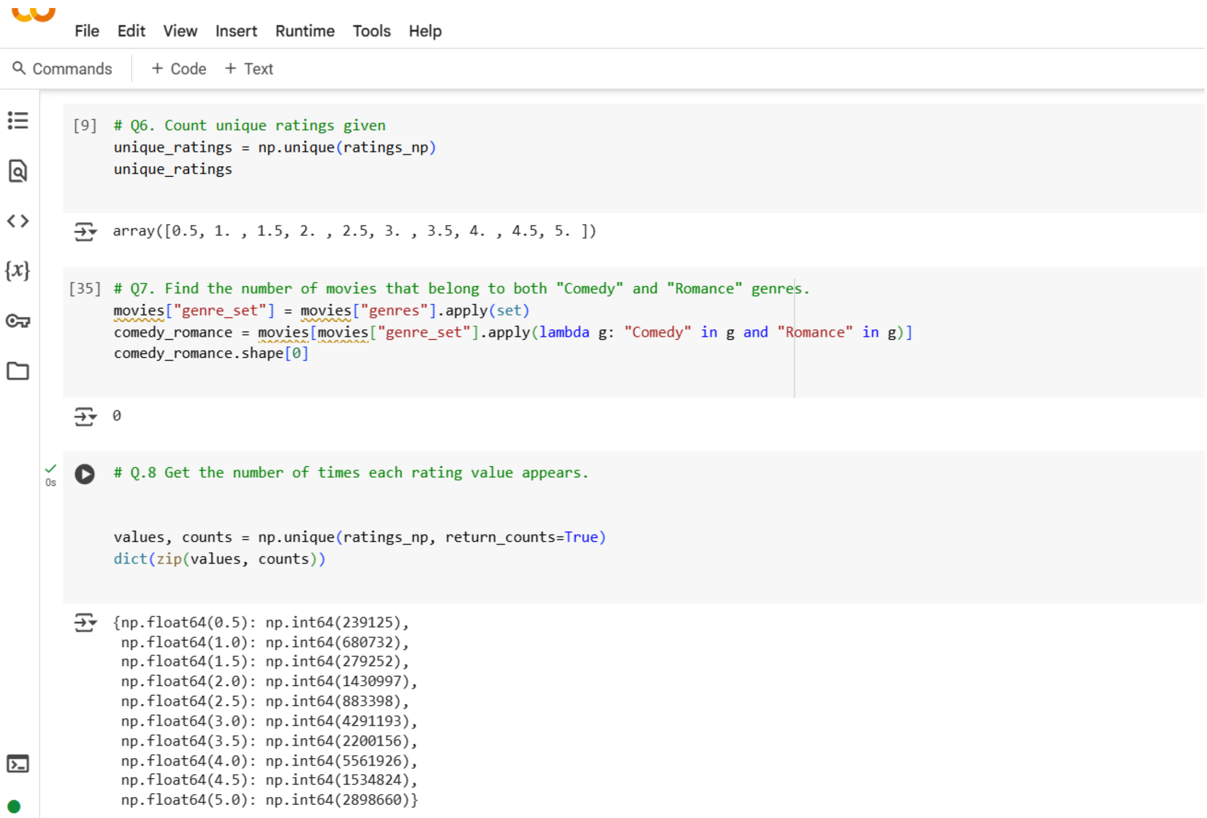
```python
[8] # Q5. Calculate the median rating.
    median_rating = np.median(ratings_np)
    median_rating
```

```
np.float64(3.5)
```



```python
# Q6. Calculate the standard deviation of ratings for each movie and find the top 10 with highest variance.
rating_std = ratings.groupby("movieId")["rating"].std().dropna().sort_values(ascending=False)
movies.set_index("movieId").loc[rating_std.head(10).index]
```

	title	genres	genre_set
movieId			
119569	Quatsch und die Nasenbärbande (2014)	Children	{n, i, l, e, d, r, C, h}
112577	Willie & Phil (1980)	Comedy|Drama|Romance	{m, n, e, |, d, o, a, r, y, c, C, R, D}
87948	Bright Victory (1951)	Drama	{m, a, r, D}
94027	Uwasa No Onna (The Woman in the Rumor) (Her Mo...	Drama|Romance	{m, n, e, |, o, a, r, c, R, D}
128173	Beethoven's Big Break (2008)	Children|Comedy	{n, m, i, l, e, |, d, o, r, y, C, h}
99012	Gay Bed and Breakfast of Terror, The (2007)	Comedy|Horror	{m, H, e, |, d, o, r, y, C}
34240	Karol: A Man Who Became Pope (Karol, un uomo d...	Drama	{m, a, r, D}
126403	Apostle Peter and The Last Supper (2012)	Drama	{m, a, r, D}
6253	Down and Out with the Dolls (2001)	Comedy	{m, e, d, o, y, C}
126959	The Epic of Everest (1924)	Documentary	{m, n, e, o, t, a, r, c, y, u, D}



```
[9]  # Q6. Count unique ratings given
     unique_ratings = np.unique(ratings_np)
     unique_ratings
```

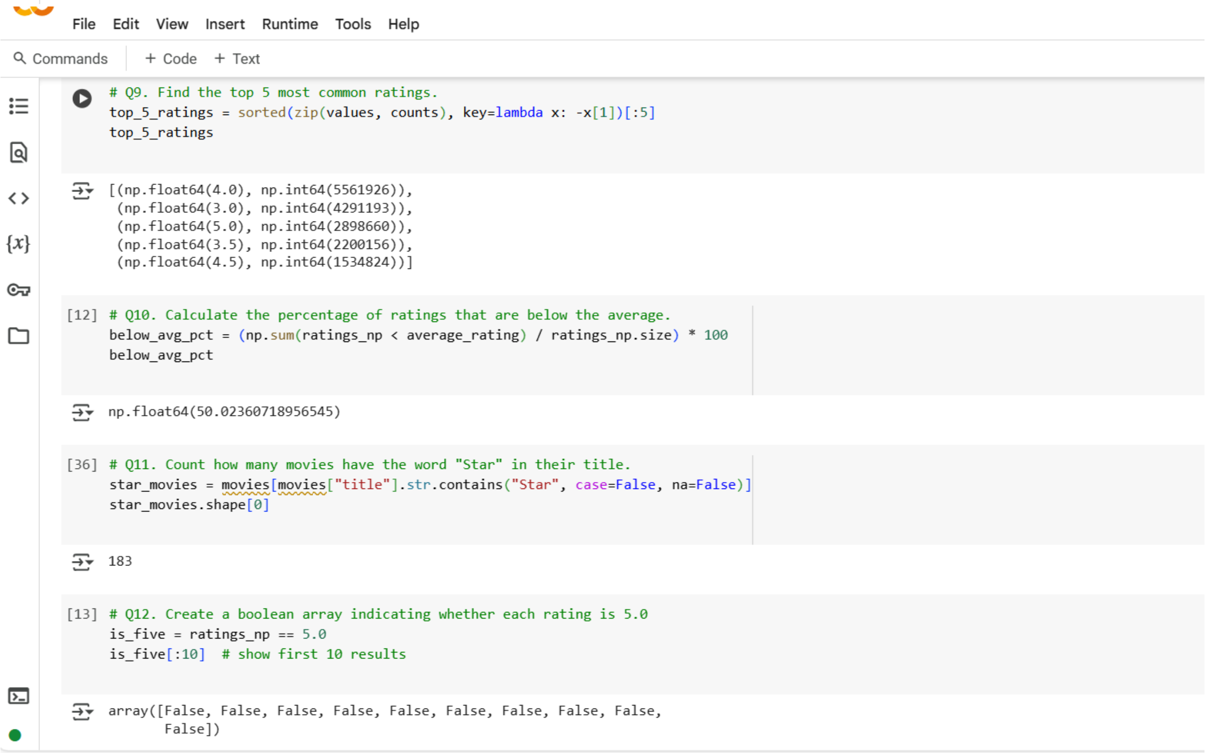
```
array([0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
```

```
[35] # Q7. Find the number of movies that belong to both "Comedy" and "Romance" genres.
     movies["genre_set"] = movies["genres"].apply(set)
     comedy_romance = movies[movies["genre_set"].apply(lambda g: "Comedy" in g and "Romance" in g)]
     comedy_romance.shape[0]
```

```
0
```

```
# Q.8 Get the number of times each rating value appears.


values, counts = np.unique(ratings_np, return_counts=True)
dict(zip(values, counts))
```

```
{np.float64(0.5): np.int64(239125),
 np.float64(1.0): np.int64(680732),
 np.float64(1.5): np.int64(279252),
 np.float64(2.0): np.int64(1430997),
 np.float64(2.5): np.int64(883398),
 np.float64(3.0): np.int64(4291193),
 np.float64(3.5): np.int64(2200156),
 np.float64(4.0): np.int64(5561926),
 np.float64(4.5): np.int64(1534824),
 np.float64(5.0): np.int64(2898660)}
```



```
# Q9. Find the top 5 most common ratings.
top_5_ratings = sorted(zip(values, counts), key=lambda x: -x[1])[:5]
top_5_ratings
```

```
[(np.float64(4.0), np.int64(5561926)),
 (np.float64(3.0), np.int64(4291193)),
 (np.float64(5.0), np.int64(2898660)),
 (np.float64(3.5), np.int64(2200156)),
 (np.float64(4.5), np.int64(1534824))]
```

```
[12] # Q10. Calculate the percentage of ratings that are below the average.
     below_avg_pct = (np.sum(ratings_np < average_rating) / ratings_np.size) * 100
     below_avg_pct
```

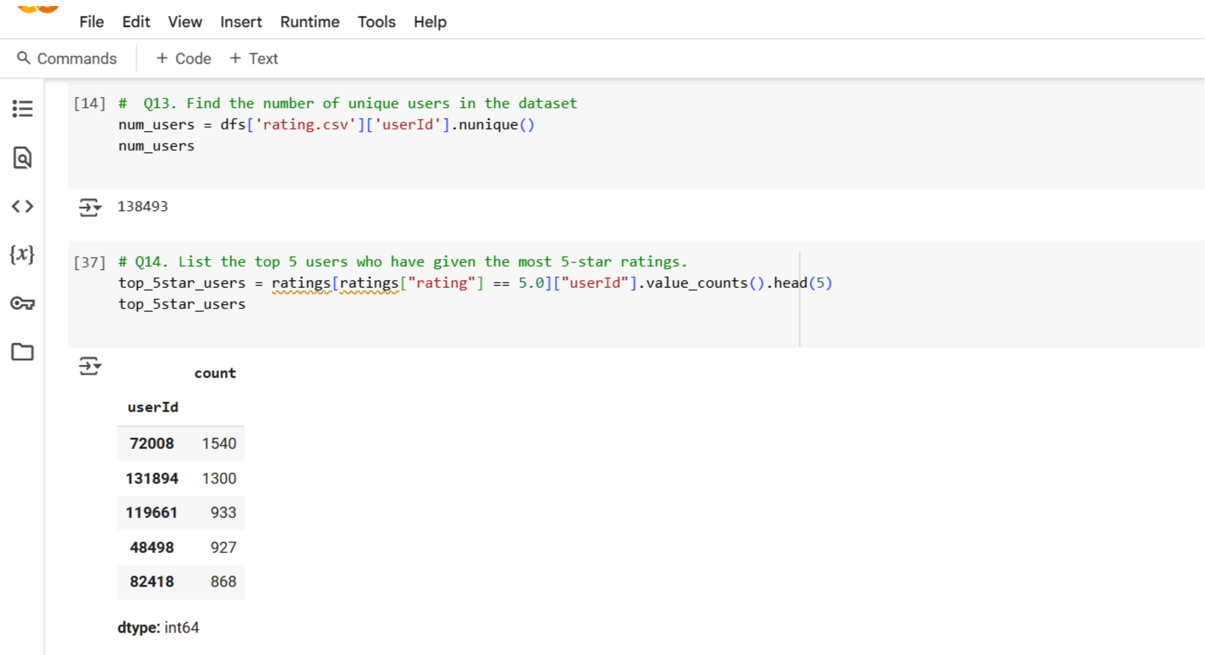
```
np.float64(50.02360718956545)
```

```
[36] # Q11. Count how many movies have the word "Star" in their title.
     star_movies = movies[movies["title"].str.contains("Star", case=False, na=False)]
     star_movies.shape[0]
```

```
183
```

```
[13] # Q12. Create a boolean array indicating whether each rating is 5.0
     is_five = ratings_np == 5.0
     is_five[:10]  # show first 10 results
```

```
array([False, False, False, False, False, False, False, False, False,
       False])
```



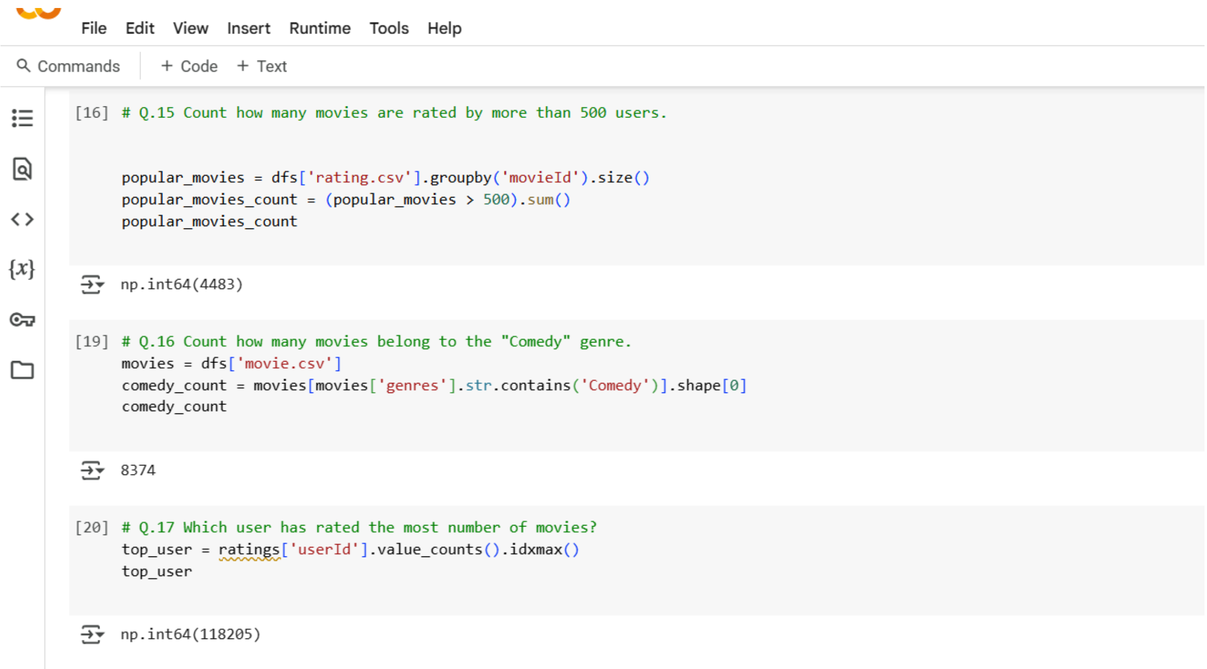
```
[14] #  Q13. Find the number of unique users in the dataset
     num_users = dfs['rating.csv']['userId'].nunique()
     num_users
```

```
138493
```

```
[37] # Q14. List the top 5 users who have given the most 5-star ratings.
     top_5star_users = ratings[ratings["rating"] == 5.0]["userId"].value_counts().head(5)
     top_5star_users
```

	count
userId	
72008	1540
131894	1300
119661	933
48498	927
82418	868

dtype: int64



```
[16] # Q.15 Count how many movies are rated by more than 500 users.


     popular_movies = dfs['rating.csv'].groupby('movieId').size()
     popular_movies_count = (popular_movies > 500).sum()
     popular_movies_count
```

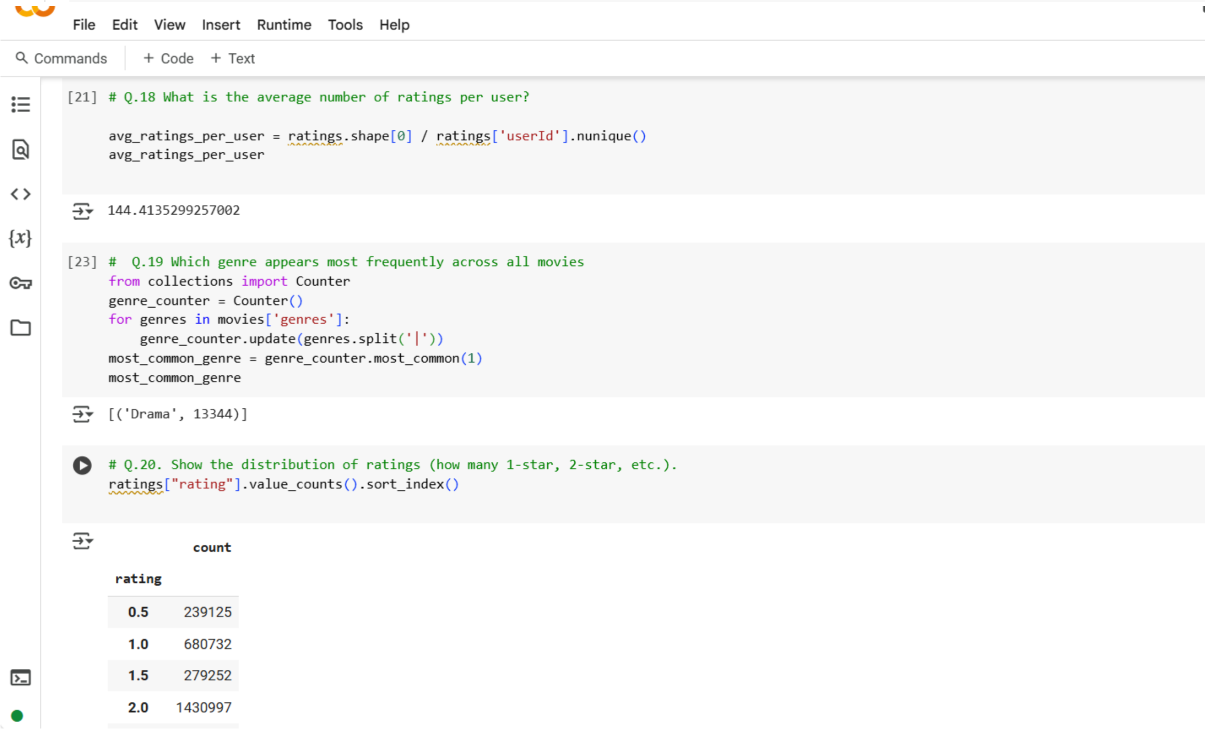
```
np.int64(4483)
```

```
[19] # Q.16 Count how many movies belong to the "Comedy" genre.
     movies = dfs['movie.csv']
     comedy_count = movies[movies['genres'].str.contains('Comedy')].shape[0]
     comedy_count
```

```
8374
```

```
[20] # Q.17 Which user has rated the most number of movies?
     top_user = ratings['userId'].value_counts().idxmax()
     top_user
```

```
np.int64(118205)
```



```
[21] # Q.18 What is the average number of ratings per user?

     avg_ratings_per_user = ratings.shape[0] / ratings['userId'].nunique()
     avg_ratings_per_user
```

```
144.4135299257002
```

```
[23] #  Q.19 Which genre appears most frequently across all movies
     from collections import Counter
     genre_counter = Counter()
     for genres in movies['genres']:
         genre_counter.update(genres.split('|'))
     most_common_genre = genre_counter.most_common(1)
     most_common_genre
```

```
[('Drama', 13344)]
```

```
# Q.20. Show the distribution of ratings (how many 1-star, 2-star, etc.).
ratings["rating"].value_counts().sort_index()
```

	count
rating	
0.5	239125
1.0	680732
1.5	279252
2.0	1430997

```
[27] # Q21. Find the 10 movies that received the most ratings.
     most_rated = ratings["movieId"].value_counts().head(10)
     movies.set_index("movieId").loc[most_rated.index]
```

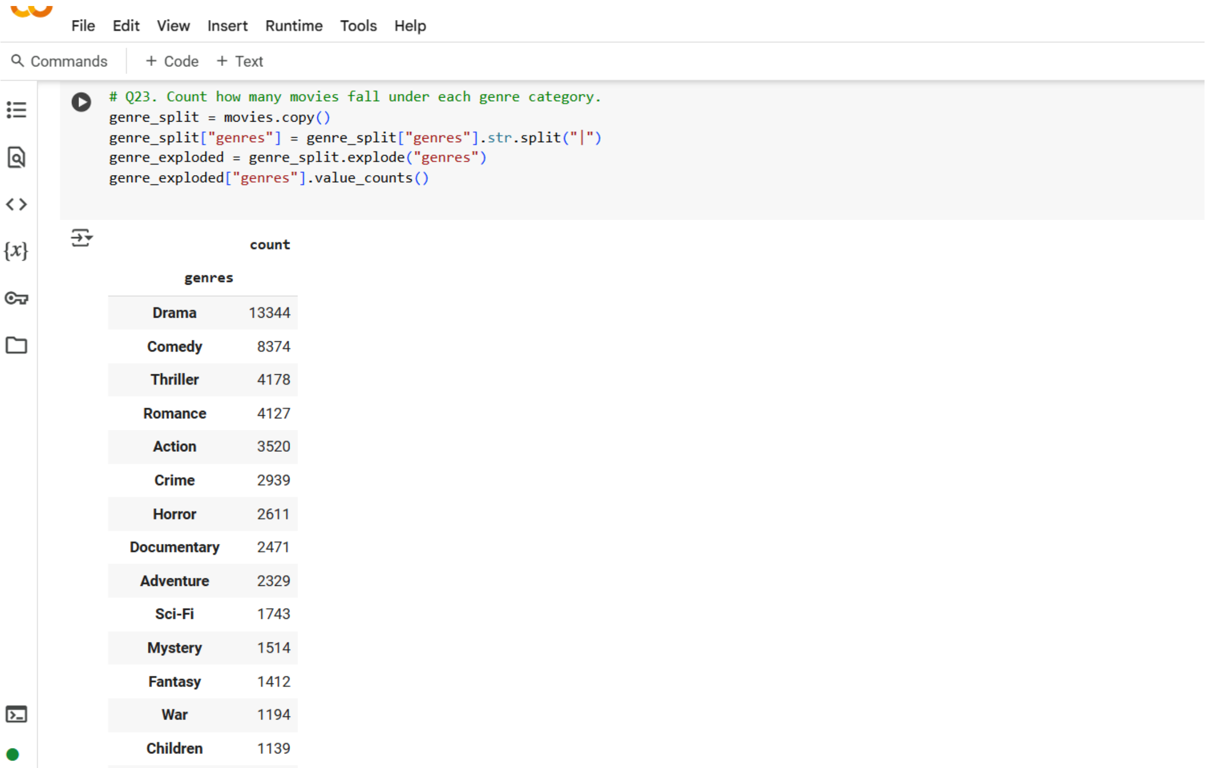
	title	genres
movieId		
296	Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller
356	Forrest Gump (1994)	Comedy|Drama|Romance|War
318	Shawshank Redemption, The (1994)	Crime|Drama
593	Silence of the Lambs, The (1991)	Crime|Horror|Thriller
480	Jurassic Park (1993)	Action|Adventure|Sci-Fi|Thriller
260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi
110	Braveheart (1995)	Action|Drama|War
589	Terminator 2: Judgment Day (1991)	Action|Sci-Fi
2571	Matrix, The (1999)	Action|Sci-Fi|Thriller
527	Schindler's List (1993)	Drama|War

```
# Q22. Identify users who have rated more than 100 movies.
active_users = ratings["userId"].value_counts()
active_users[active_users > 100]
```

	count
userId	
118205	9254
8405	7515
82418	5646
121535	5520
125794	5491
...	...
74293	101
20433	101
20427	101
3440	101
115563	101

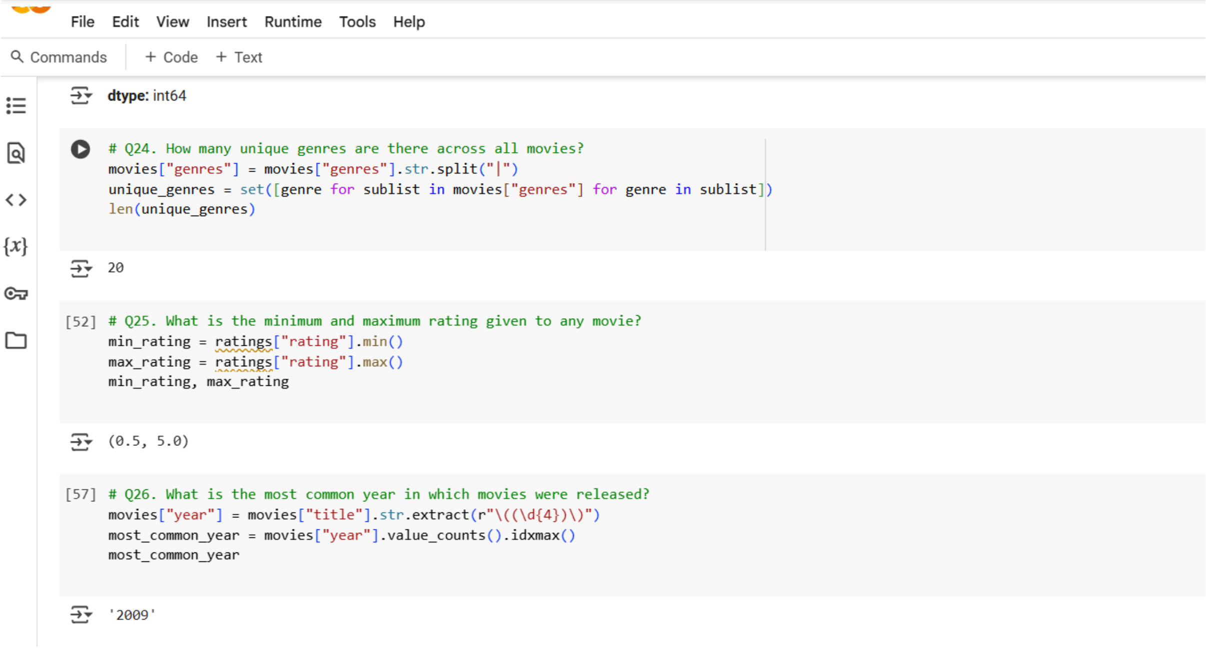
51869 rows × 1 columns

dtype: int64



```
# Q23. Count how many movies fall under each genre category.
genre_split = movies.copy()
genre_split["genres"] = genre_split["genres"].str.split("|")
genre_exploded = genre_split.explode("genres")
genre_exploded["genres"].value_counts()
```

	count
genres	
Drama	13344
Comedy	8374
Thriller	4178
Romance	4127
Action	3520
Crime	2939
Horror	2611
Documentary	2471
Adventure	2329
Sci-Fi	1743
Mystery	1514
Fantasy	1412
War	1194
Children	1139



dtype: int64

```
# Q24. How many unique genres are there across all movies?
movies["genres"] = movies["genres"].str.split("|")
unique_genres = set([genre for sublist in movies["genres"] for genre in sublist])
len(unique_genres)
```

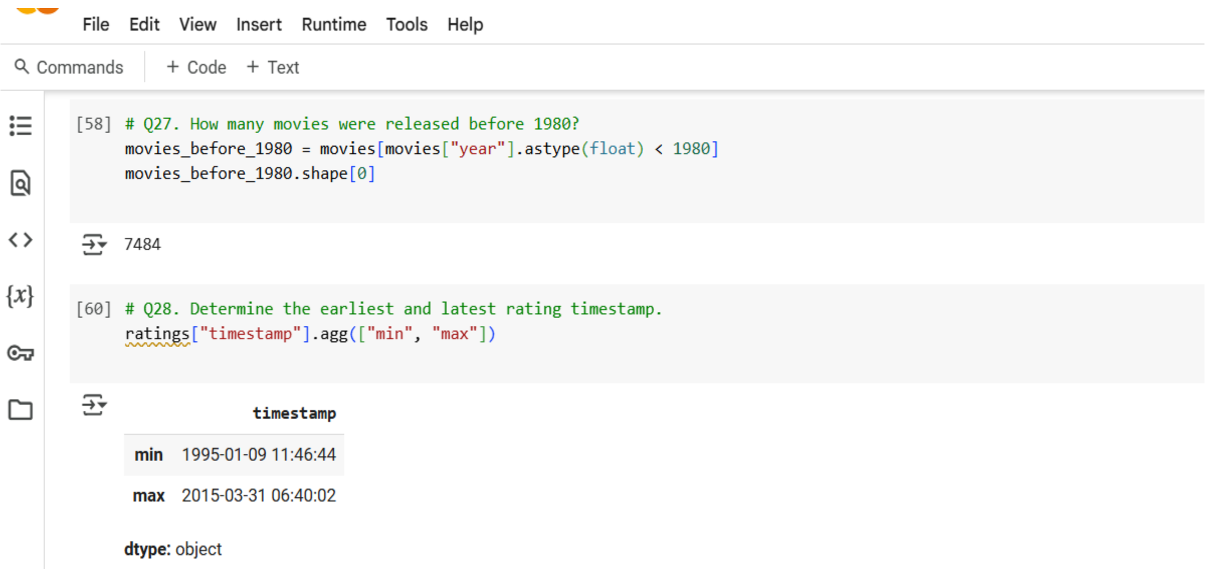
```
20
```

```
[52] # Q25. What is the minimum and maximum rating given to any movie?
min_rating = ratings["rating"].min()
max_rating = ratings["rating"].max()
min_rating, max_rating
```

```
(0.5, 5.0)
```

```
[57] # Q26. What is the most common year in which movies were released?
movies["year"] = movies["title"].str.extract(r"\((\d{4})\)")
most_common_year = movies["year"].value_counts().idxmax()
most_common_year
```

```
'2009'
```

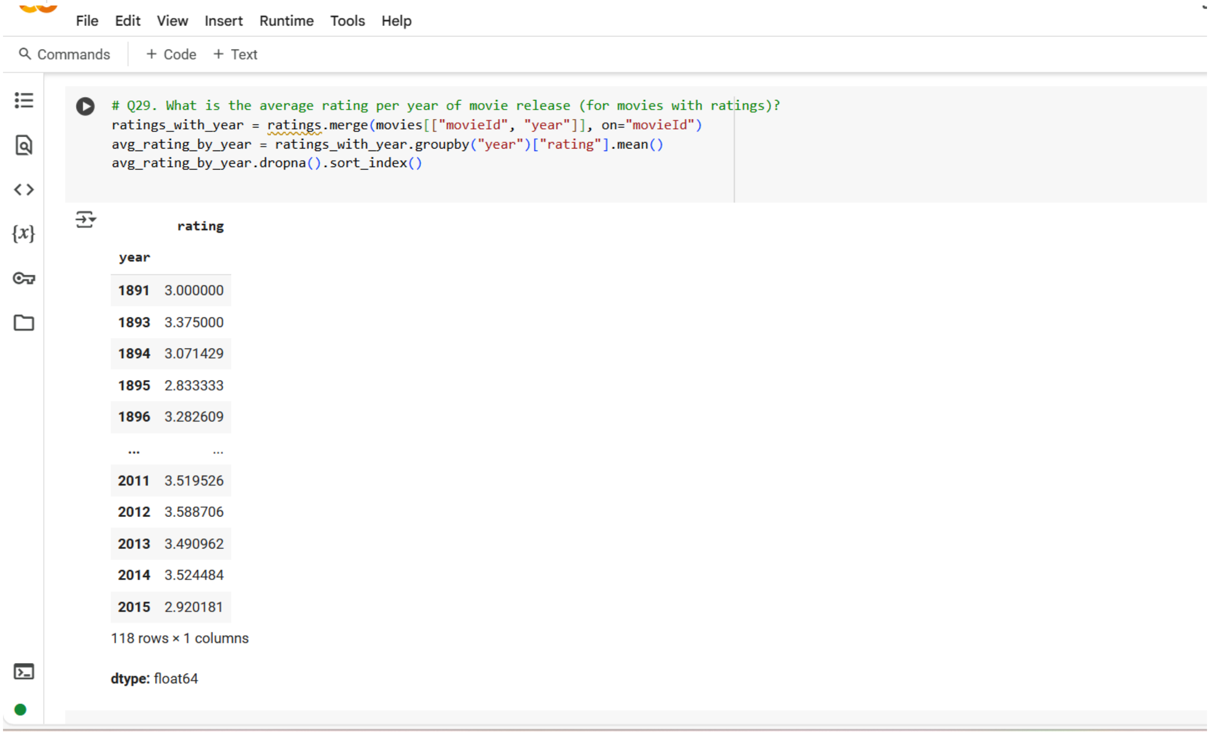


```
[58] # Q27. How many movies were released before 1980?
     movies_before_1980 = movies[movies["year"].astype(float) < 1980]
     movies_before_1980.shape[0]
```

```
7484
```

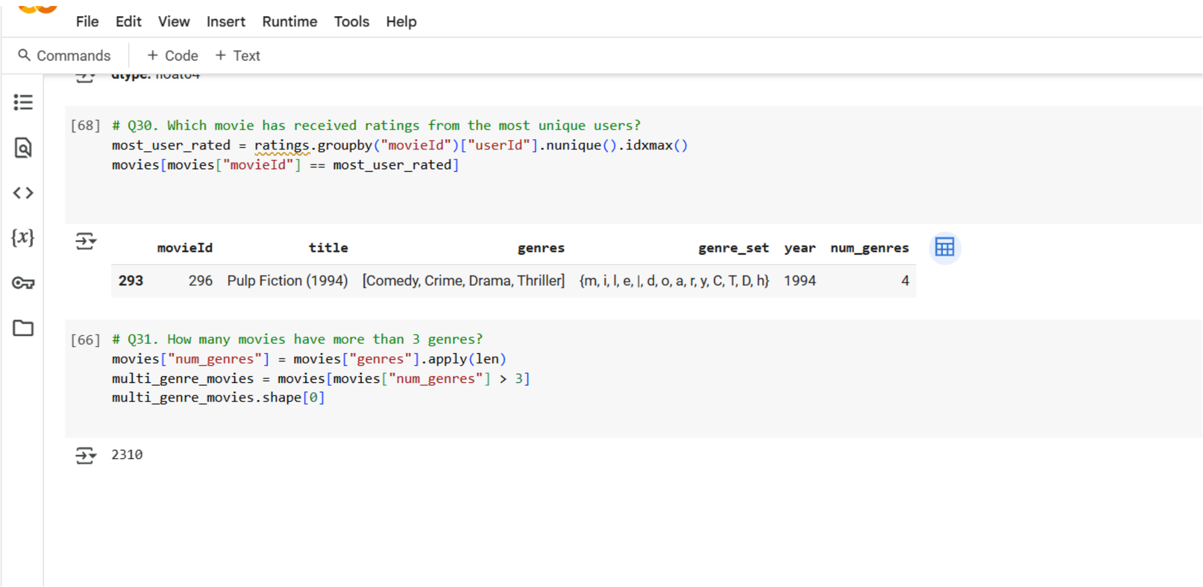
```
[60] # Q28. Determine the earliest and latest rating timestamp.
     ratings["timestamp"].agg(["min", "max"])
```

```
               timestamp
min   1995-01-09 11:46:44
max   2015-03-31 06:40:02

dtype: object
```



```
# Q29. What is the average rating per year of movie release (for movies with ratings)?
ratings_with_year = ratings.merge(movies[["movieId", "year"]], on="movieId")
avg_rating_by_year = ratings_with_year.groupby("year")["rating"].mean()
avg_rating_by_year.dropna().sort_index()
```

```
        rating
year
1891  3.000000
1893  3.375000
1894  3.071429
1895  2.833333
1896  3.282609
 ...       ...
2011  3.519526
2012  3.588706
2013  3.490962
2014  3.524484
2015  2.920181
118 rows × 1 columns

dtype: float64
```



```
dtype: float64
```

```
[68] # Q30. Which movie has received ratings from the most unique users?
     most_user_rated = ratings.groupby("movieId")["userId"].nunique().idxmax()
     movies[movies["movieId"] == most_user_rated]
```

	movieId	title	genres	genre_set	year	num_genres
293	296	Pulp Fiction (1994)	[Comedy, Crime, Drama, Thriller]	{m, i, l, e, |, d, o, a, r, y, C, T, D, h}	1994	4

```
[66] # Q31. How many movies have more than 3 genres?
     movies["num_genres"] = movies["genres"].apply(len)
     multi_genre_movies = movies[movies["num_genres"] > 3]
     multi_genre_movies.shape[0]
```

```
2310
```