Project 3: Data Manipulation with Pandas

John Wesley Mathis

Dr. Anthony Choi

June 08, 2024

ECE/SSE 591, Summer 2024

Table of Contents:

[Deliverable Table 5](#__RefHeading___Toc2569_331549091)

[1. Introduction 6](#__RefHeading___Toc2573_331549091)

[2. Top Movie Data Analysis 7](#__RefHeading___Toc2577_331549091)

[3. Conclusion 33](#__RefHeading___Toc2609_331549091)

[References 34](#__RefHeading___Toc3692_331549091)

Table of Figures

[Figure 1: Using Pandas objects and importing CSV files 7](#Figure!0|sequence)

[Figure 2: Collecting information about the data 7](#Figure!1|sequence)

[Figure 3: Collecting information about the dataset 8](#Figure!2|sequence)

[Figure 4: Changing the index of the dataset 9](#Figure!3|sequence)

[Figure 5: Adding a Rank column 10](#Figure!4|sequence)

[Figure 6: Moving 'Rank' column to the beginning of the dataset 11](#Figure!5|sequence)

[Figure 7: Sorting the data by IMDB rating 12](#Figure!6|sequence)

[Figure 8: Finding what data is missing 13](#Figure!7|sequence)

[Figure 9: Removing unused columns from the dataset 14](#Figure!8|sequence)

[Figure 10: Dealing with missing data 15](#Figure!9|sequence)

[Figure 11: Checking for duplicated data 16](#Figure!10|sequence)

[Figure 12: Converting 'Gross' column from String to Integer 16](#Figure!11|sequence)

[Figure 13: Cleaning up the 'Runtime' column 17](#Figure!12|sequence)

[Figure 14: Converting release year to datatime format 18](#Figure!13|sequence)

[Figure 15: Cleaning up the genre column 18](#Figure!14|sequence)

[Figure 16: Determining top movie genres 19](#Figure!15|sequence)

[Figure 17: Determining top 5 most voted movies 19](#Figure!16|sequence)

[Figure 18: Determining bottom 5 least voted movies 20](#Figure!17|sequence)

[Figure 19: Determining which movie has the longest runtime 20](#Figure!18|sequence)

[Figure 20: Determining which movie has the shortest runtime 21](#Figure!33|sequence)

[Figure 21: Using `query()` to filter data 21](#Figure!19|sequence)

[Figure 22: Using `eval()` to calculate new data 22](#Figure!20|sequence)

[Figure 23: Determining which year had the most top movies 22](#Figure!21|sequence)

[Figure 24: Determining highest grossing movie 23](#Figure!22|sequence)

[Figure 25: Determining which genre has the IMDB rating and which has the highest gross 24](#Figure!23|sequence)

[Figure 26: Discovering which directors consistently produce top movies using pivot tables 25](#Figure!24|sequence)

[Figure 27: Demonstrating hierarchical indexing 26](#Figure!25|sequence)

[Figure 28: Importing two new datasets and showing information about dataset 1 27](#Figure!26|sequence)

[Figure 29: Dataset 2 information 27](#Figure!27|sequence)

[Figure 30: Merging the two datasets on 'id' 28](#Figure!28|sequence)

[Figure 31: Removing unused columns 29](#Figure!29|sequence)

[Figure 32: Setting the index to 'genres' and 'release\_date' 30](#Figure!30|sequence)

[Figure 33: Code for vectorized string operations 30](#Figure!31|sequence)

[Figure 34: Output to Figure 32 code 31](#Figure!32|sequence)

[Figure 35: Demonstrating `concat()` 32](#Figure!34|sequence)

[Figure 36: Using `concat()` to append new data 33](#Figure!35|sequence)

# Deliverable Table

The purpose of this table is to provide a complete view of the concepts covered in chapter 3 of *"Python Data Science Handbook"* (VanderPlas, 2016) and provide a general page location for where the topic was demonstrated.

|  |  |
| --- | --- |
| Deliverables | Location |
| Introducing Pandas Objects | 6 |
| Data Indexing and Selection | 6 |
| Operating on Data in Pandas | 6-7 |
| Handling Missing Data | 12-14 |
| Hierarchical Indexing | 24 |
| Combining Datasets: Concat and Append | 31-32 |
| Combining Datasets: Merge and Join | 25-30 |
| Aggregation and Grouping | 22 |
| Pivot Tables | 23 |
| Vectorized String Operations | 25-30 |
| Working with Time Series | 16 |
| High-Performance Pandas: eval() and query() | 20-21 |

Additionally, here is a link to my GitHub were the datasets and the Jupyter Notebook for the project can be downloaded: https://github.com/jwmathis/SSE591\_Project3.git. In order to run the file, Python and Pandas package must be installed.

# 1. Introduction

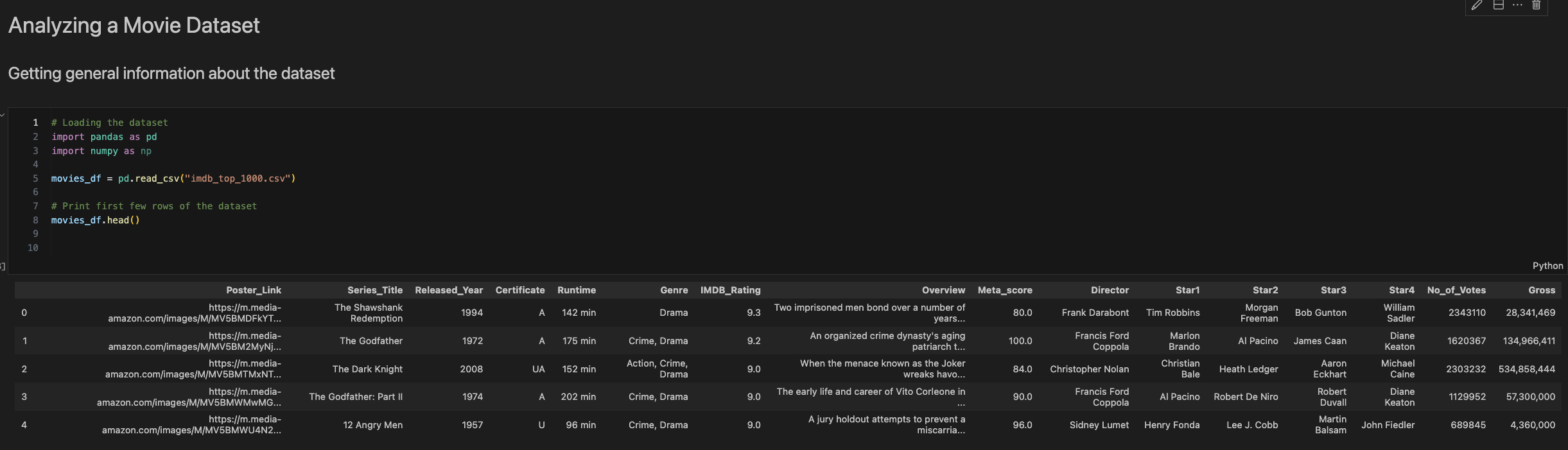
Python has a rich repository of libraries that aid scientists and researchers in data analysis and manipulation. One of the most common libraries in use is Pandas, which is built on top of NumPy and provides a higher-level, and more flexible interface for data handling. While NumPy excels at efficient numerical computations with arrays, Pandas introduces data structures like Series and DataFrame that offer a more intuitive means to work with structured data.

Because of Pandas’ Series and DataFrame objects, data scientists have an indispensable tool to handle, clean and manipulate data in tabular form. These objects support a wide range of operations, from simple data aggregation and filtering to complex time-series analysis. The library’s ability to handle missing data, merge datasets, and perform group-by operations adds significant value to Python’s data manipulation kit.

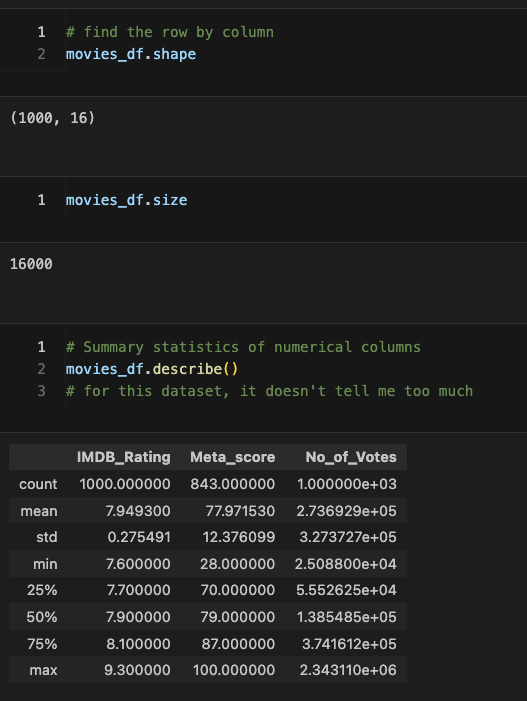
This report aims to demonstrate my proficiency in Python data manipulation techniques as covered in Chapter 3 of the “Python Data Science Handbook” by Jake VanderPlas (2016). This report attempts to illustrate the core concepts and functionalities of the Pandas library by implementing the concepts into a single project. The code presented in this report was developed using Visual Studio Code with Jupyter Notebook extensions. I will provide detailed explanations, highlighting key features and operations that make Pandas an essential tool for data analysis.

# 2. Top Movie Data Analysis

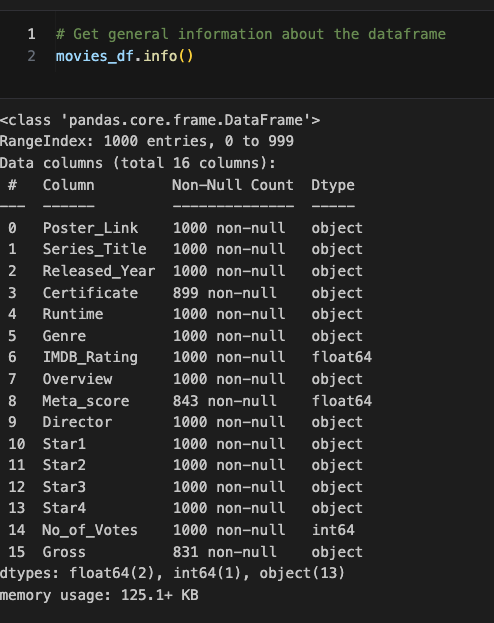
I chose the IMDB Top 1000 movies dataset to analyze. This dataset contains information about the top 1000 movies. After importing the dataset, I viewed the first few rows by outputting it to the screen using *`head()`.* Figure 1 below shows the code and output.

Figure 1: Using Pandas objects and importing CSV files

Using *shape()`* , *`size()`, and`describe()`* I obtained summary statistics of the numerical columns. There are only three numerical columns, IMDB rating, metascore, and number of votes. For this particular data set this doesn’t tell me to much. However, we find out that the average IMDB rating is a 7.94 and the average metascore for movies is a 77.97. However, because we don’t know much about the data, we don’t understand how skewed this information may be. Figure 2 shows the code and output.

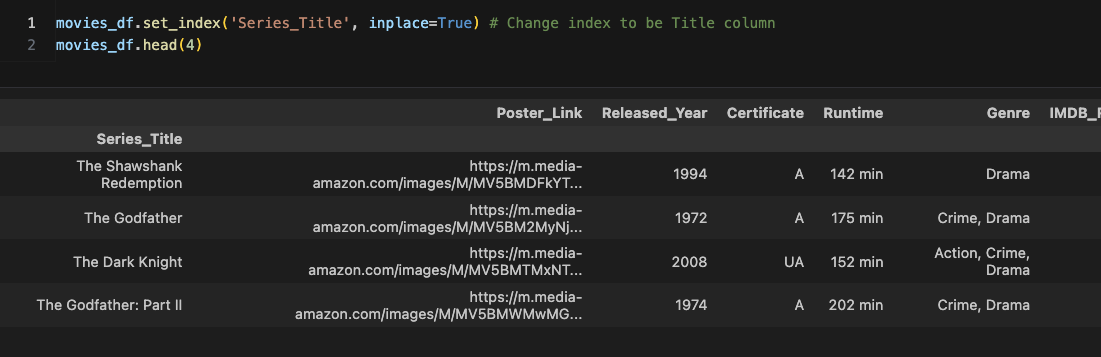
Figure 2: Collecting information about the data

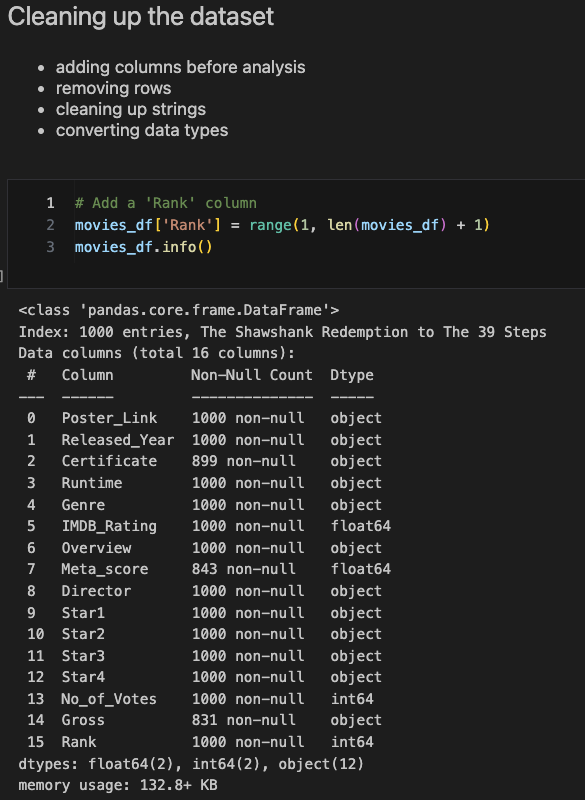
To better understand the data, I used `*info()`* to obtain general information about the DataFrame. From this view, I learned the column names, the data type for each column, and if each column contained all the information. The *Certificate, Metascore, and Gross* columns were missing some data. Figure 3 shows the code and output.

Figure 3: Collecting information about the dataset

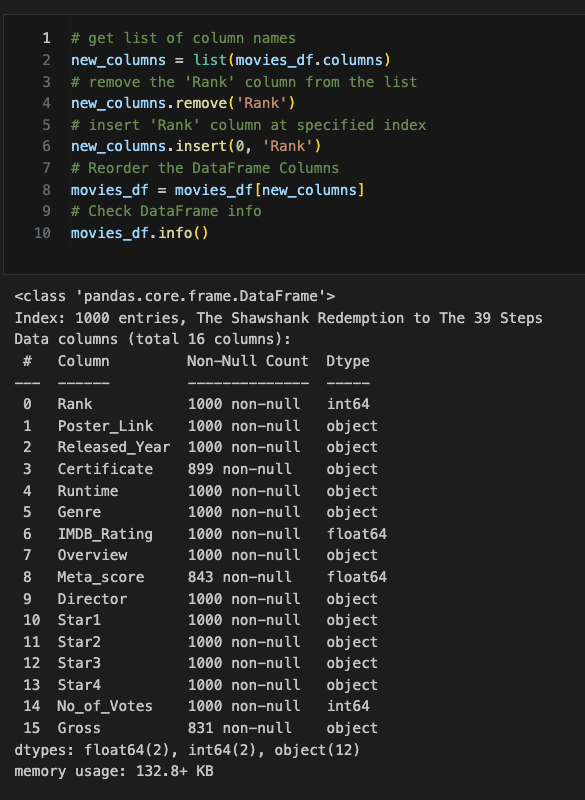
Now with this understanding of the data, I begin to clean it up. Because the data seems to be un-ranked, just simply the top 1000 movies in an arbitrary order, I changed the index from the basic 0-999 using the *`set\_index()`* method. I changed the index to be the series title. Figure 4 shows this output.

After viewing the first few rows of data and the last few rows using the *`head()`* and *`tail()`* methods respectively, I confirmed that the data was un-ranked The dataset is simply a list of the top 1000 movies. After reviewing the data info from before (Fig 4), I decided to add a rank column as seen in Figure 5.

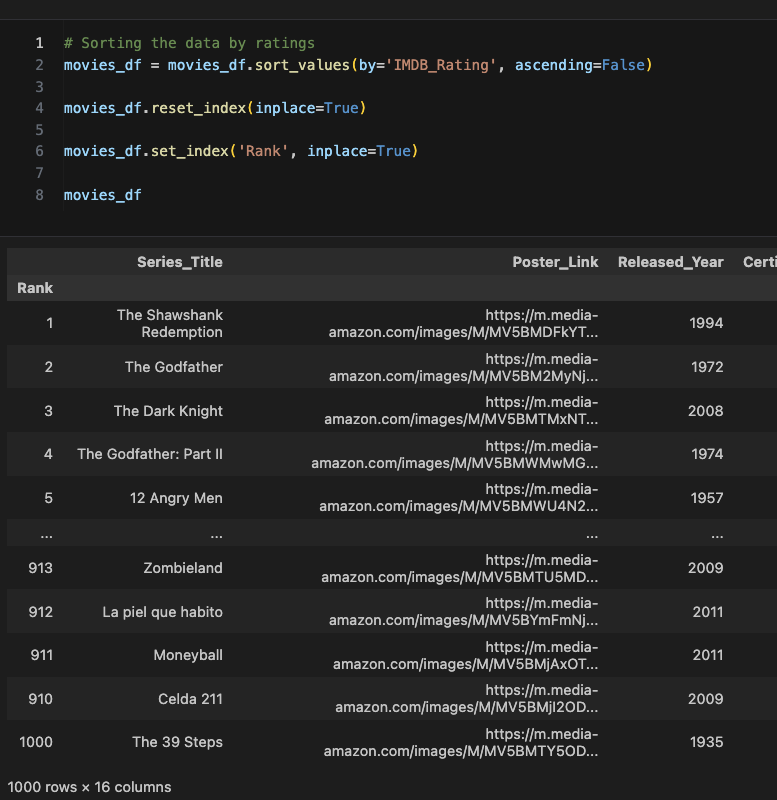
Figure 4: Changing the index of the dataset

Figure 5: Adding a Rank column

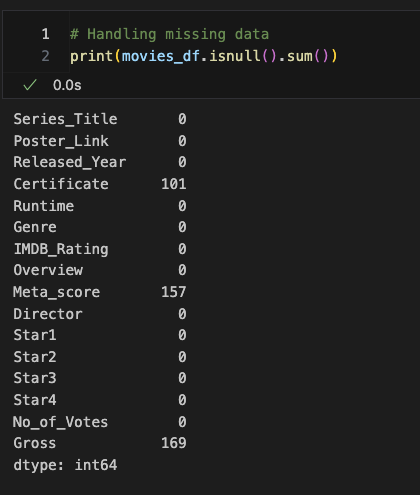
However, there are a lot of columns in this dataset, and I wanted to be able to quickly and easily view the ranks. So I retrieved the columns and converted them to a list and stored the information in a variable for access. I removed the string *`Rank`* from the list and inserted it again into the list at position 0. Then I re-ordered the DataFrame using the variable that stored the columns. Figure 6 shows the output.

Figure 6: Moving 'Rank' column to the beginning of the dataset

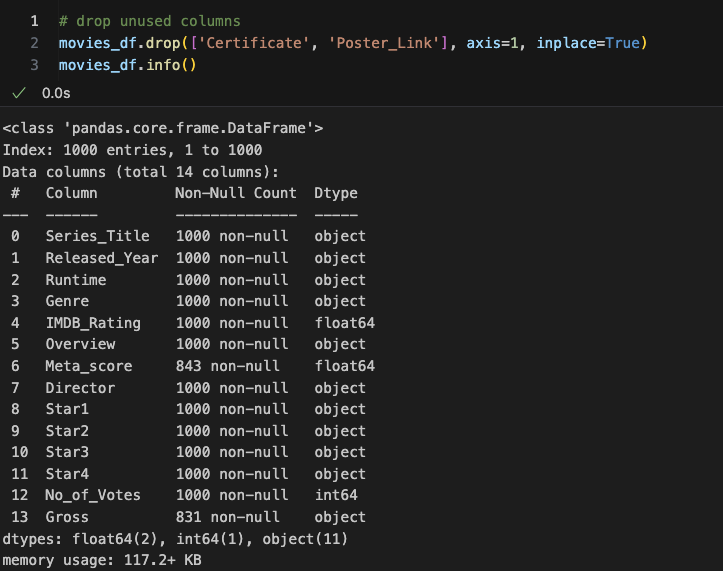
After re-ordering the DataFrame, I used *`sort\_values()`*  using the IMDB rating to sort the movies in ascending order. Then I set the index to be the *Rank* column. Figure 7 shows the code and output

Figure 7: Sorting the data by IMDB rating

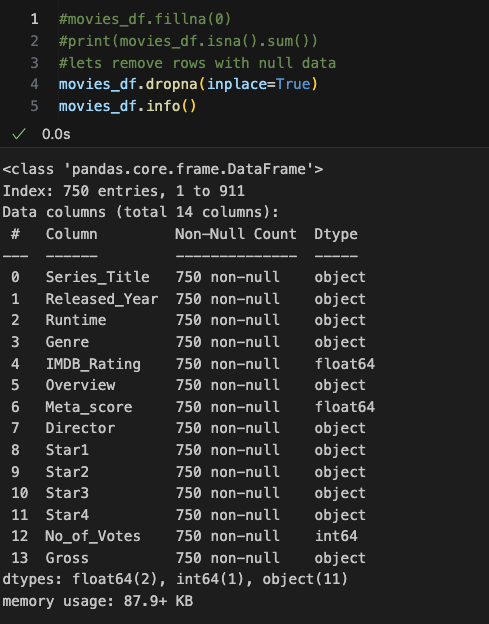
From figure ## I already saw that some of the data was missing. So I wanted to clean that up some. So I first checked what columns contain null values using *`isna()`* and *`sum()`* to get a number of missing data in each column. From this I verify that the only columns missing data are *Certifcate, Meta\_score, and Gross.* Figure 8 shows the code and results.

Figure 8: Finding what data is missing

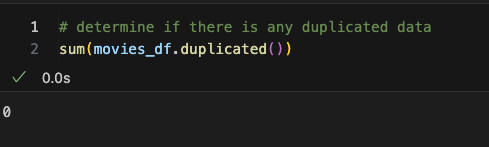
I didn’t want the Certificate column or the Poster\_Link column, so I decided to remove both. Figure 9 shows the code and the output. I remove these columns by using *`drop()`*. With *Certificate* gone, there is one less column of missing data to deal with.

Figure 9: Removing unused columns from the dataset

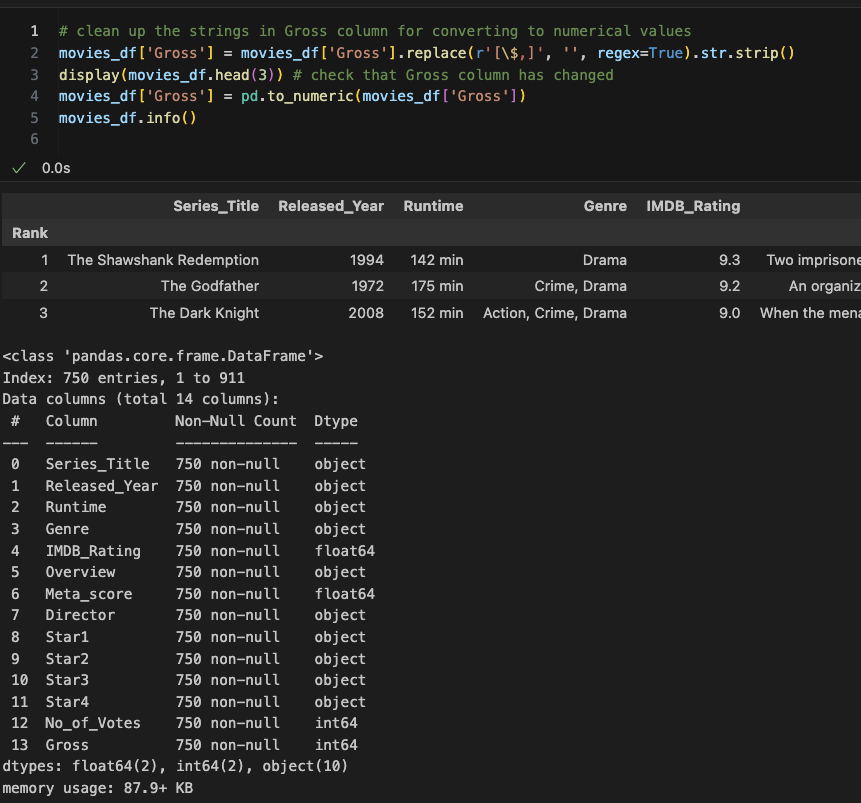
*Gross*  represents the amount of money the movie made. Rather than finding appropriate data to fill in these missing values for the movies, I decide to simply remove all rows that are missing data in either the metascore column or the gross column. I used *`dropna()`* to remove these rows. Figure 10 shows the code and output. This reduces the number of entries to 750, meaning I lost 25% of my data.

Figure 10: Dealing with missing data

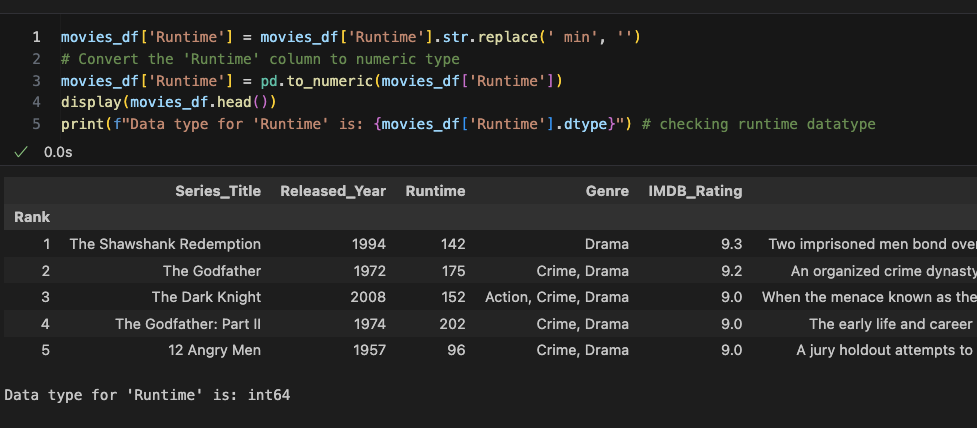
I wanted to also check if there are any duplicates in the dataset. Figure 11 shows the code and output. I used *`duplicated()`* to determine this. The result return 0, meaning there are no duplicated rows.

Figure 11: Checking for duplicated data

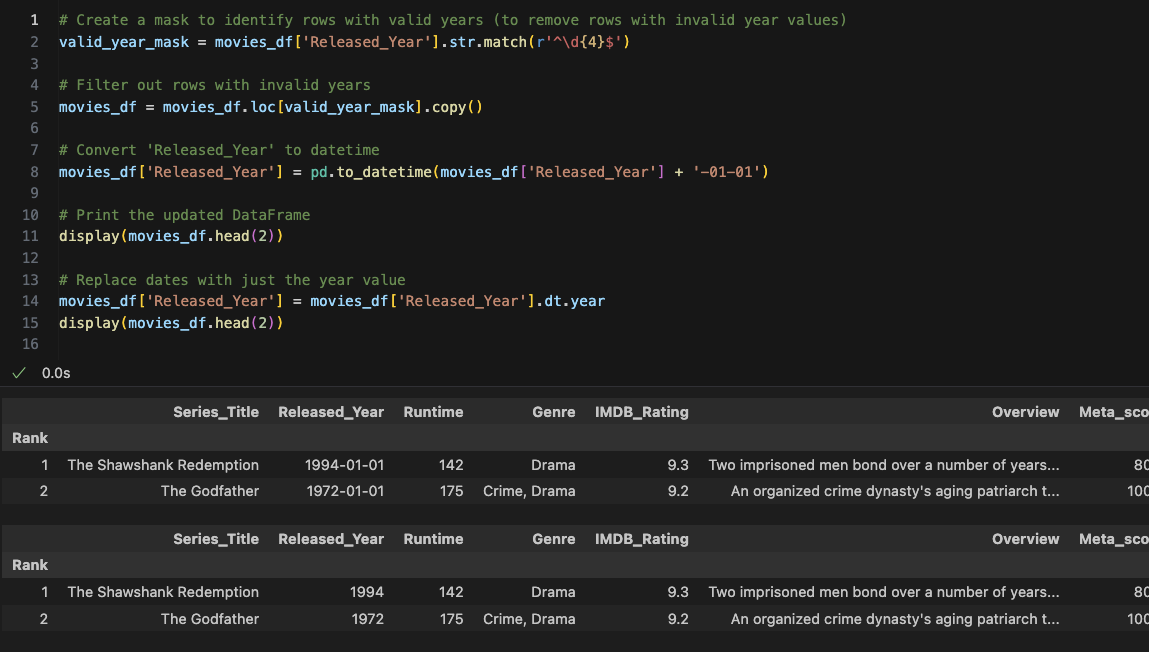
Next, I attempted to convert the data in Gross to numerical data since its data type is still an object. However, this produced a *ValueError.* So I updated the code to coerce errors to NaN. Unfortunately this removed all the data from that column. So I backtracked and decided to clean the strings up. After I cleaned the strings, I then converted the values to numerical values. Figure 12 shows my updated code and output and shows that the Gross column is now a type int64.

Figure 12: Converting 'Gross' column from String to Integer

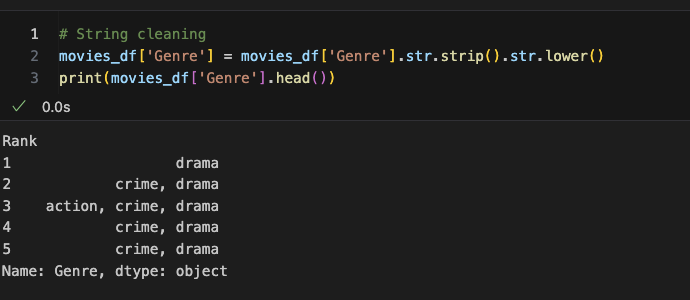
Next, I cleaned up the Runtime column. I wanted it to be numerical data as well. To accomplish this I first removed the word minute from the columns by using *`str.replace()`*. Then I used *`pd.to\_numeric`* to convert the strings to numerical values. Figure 13 shows the results.

Figure 13: Cleaning up the 'Runtime' column

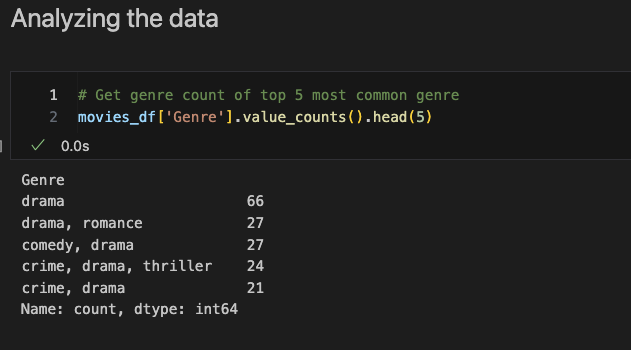
There is a release year column so I wanted to turn this into the datetime format. Using *`pd.to\_datetime()`* I attempted to do this. Unfortunately, this only works if it is a full date and this column of data only contains the year. So to demonstrate this functionality, I decided to add the full date to the column by filling it with the correct year and a placeholder date of ‘01-01’. However, all of the rows do not contain a valid year, but it does contain some type of information that is not considered NaN or null. To remove invalid rows, I created a mask to find only the rows with a valid year. Then, I updated the DataFrame to only include these valid year rows. Afterwards, I converted the release year to the proper datetime format. After this was accomplished, I reset the column to show only the year values by using the *`dt.year`*. Figure 14 shows the code and result.

Figure 14: Converting release year to datatime format

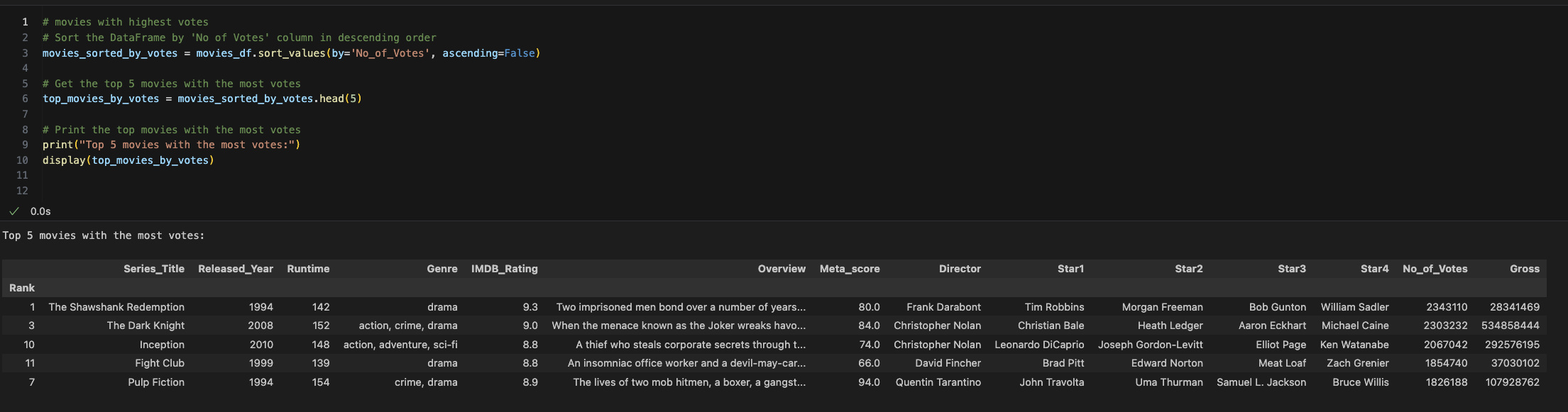
Lastly, I clean up the genre column by using *`str.strip()`* and *`str.lower()`* to ensure there was nothing weird with the column. Figure 15 shows the code and the results.

Figure 15: Cleaning up the genre column

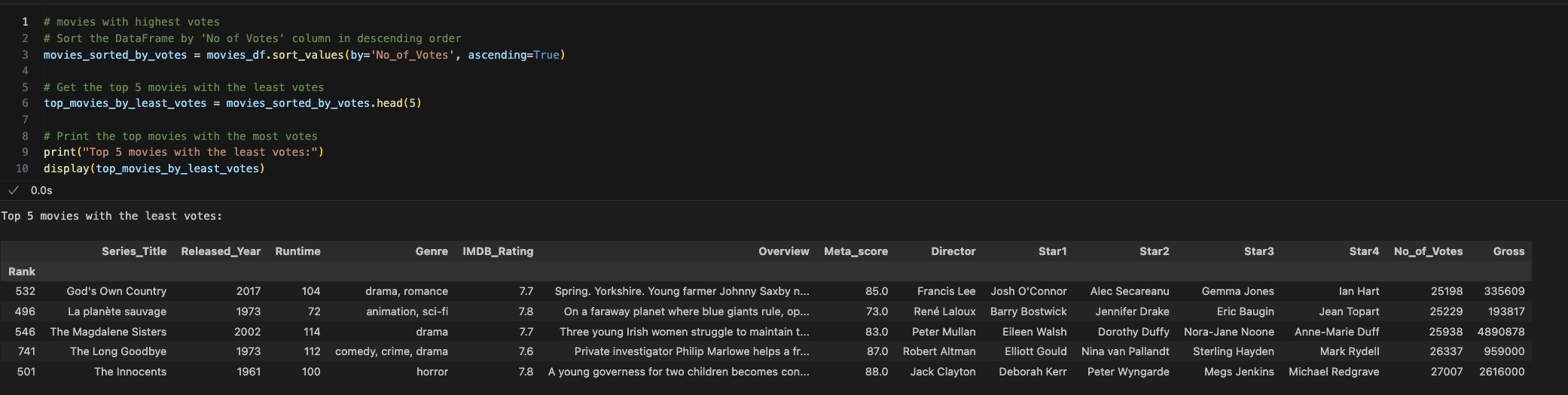
After cleaning up the data, I begin to analyze the dataset. First I determine what are the top five most common genre movies that made it to this list. I use *`value\_counts().head(5)`*  to display the top 5 results. Figure 16 shows the code and results. The top movie genres are drama, comedy, romance, thriller, and crime.

Figure 16: Determining top movie genres

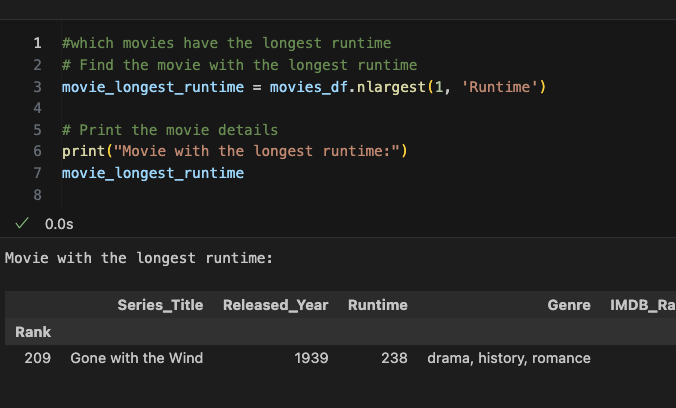
I then use *`sort\_values`* to sort the DataFrame by the Number of votes column to determine what top 5 movies had the most votes. Figure 17 shows the code and results.

Figure 17: Determining top 5 most voted movies

I then do the same thing but to see which movies had the least number of votes. Figure 18 shows the results. The least number of votes is 25,198. This lets me know that none of the movies have a small number of votes that could truly skew the data such as one 5 star vote that makes the movie rank in the top ten. However, this does not mean the results aren’t skewed. Further analysis would need to be completed to correctly determine this.

Figure 18: Determining bottom 5 least voted movies

The next question I answered was which movies had the longest runtime and which movies had the shortest runtime. I useed the built-in*`nlargest()`* and *`nsmallest()`* to determine this. Figures 19 and 20 show the respective code along with the results.

Figure 19: Determining which movie has the longest runtime

To demonstrate the usefulness of *`query()`,* I made a vert simple example that filers the movie dataset. I have the code print out the movies that have a released year past 2004, with an IMDB rating greater than 8.0 and a metascore higher than 90.0. For this only 11 movies meet the criteria, and three of the eleven are animated movies. Figure 21 shows the results.

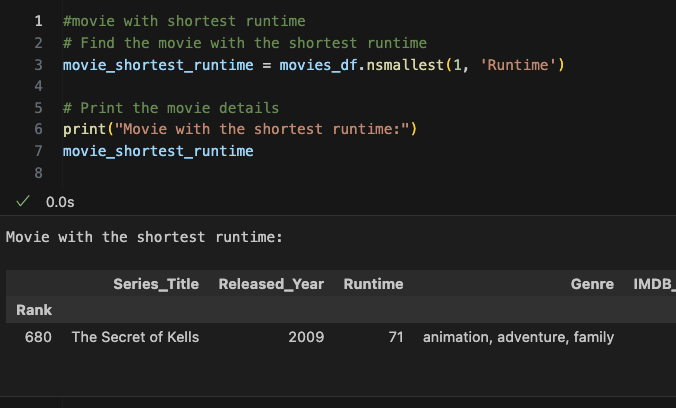
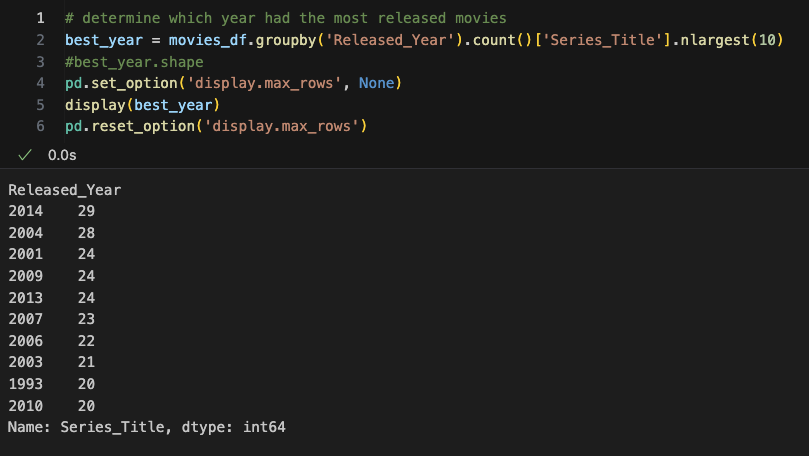
Figure 20: Determining which movie has the shortest runtime

Figure 21: Using `query()` to filter data

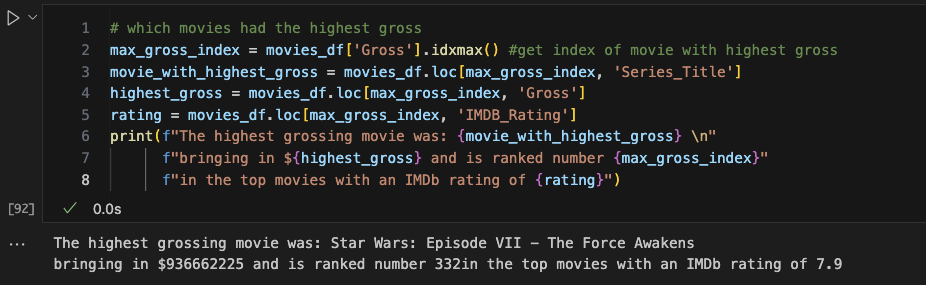
To demonstrate using eval, I calculated what the total earnings would be by multiplying the gross column by the number of votes. This is not going to be realistic or is necessary. This is simply to show the usefulness of using *`eval()`.* Figure 22 shows the code and results.

Figure 22: Using `eval()` to calculate new data

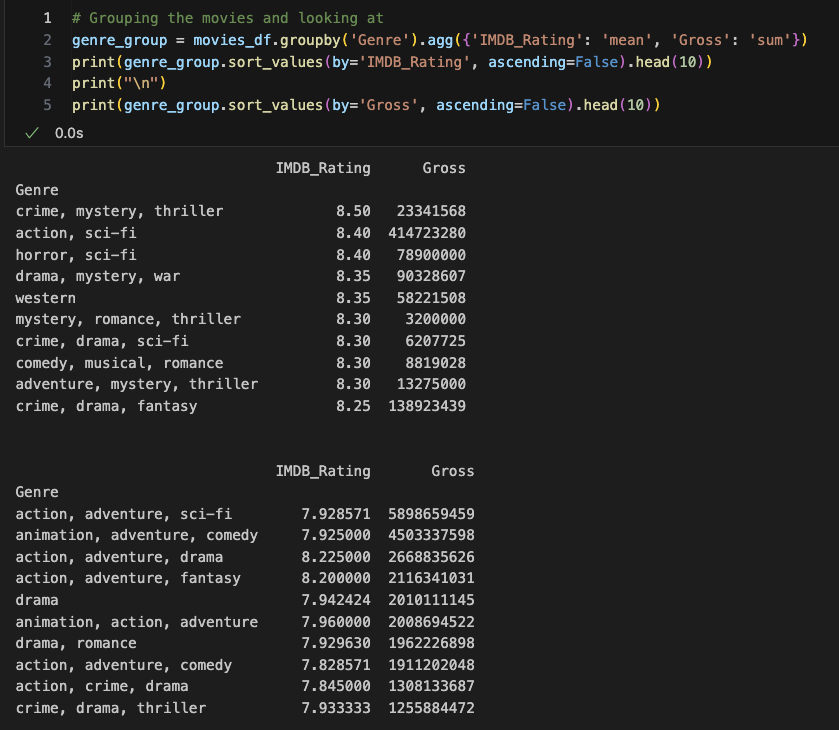
The next question I answer is which year had the most top movies released. I accomplish this by using the *`groupby()`* method and *`count()`* method. I store the results in a variable named *`best\_year`*. From this information I determined that 2014 had the most released movies that were considered top movies. Figure 23 shows this code and output.

Figure 23: Determining which year had the most top movies

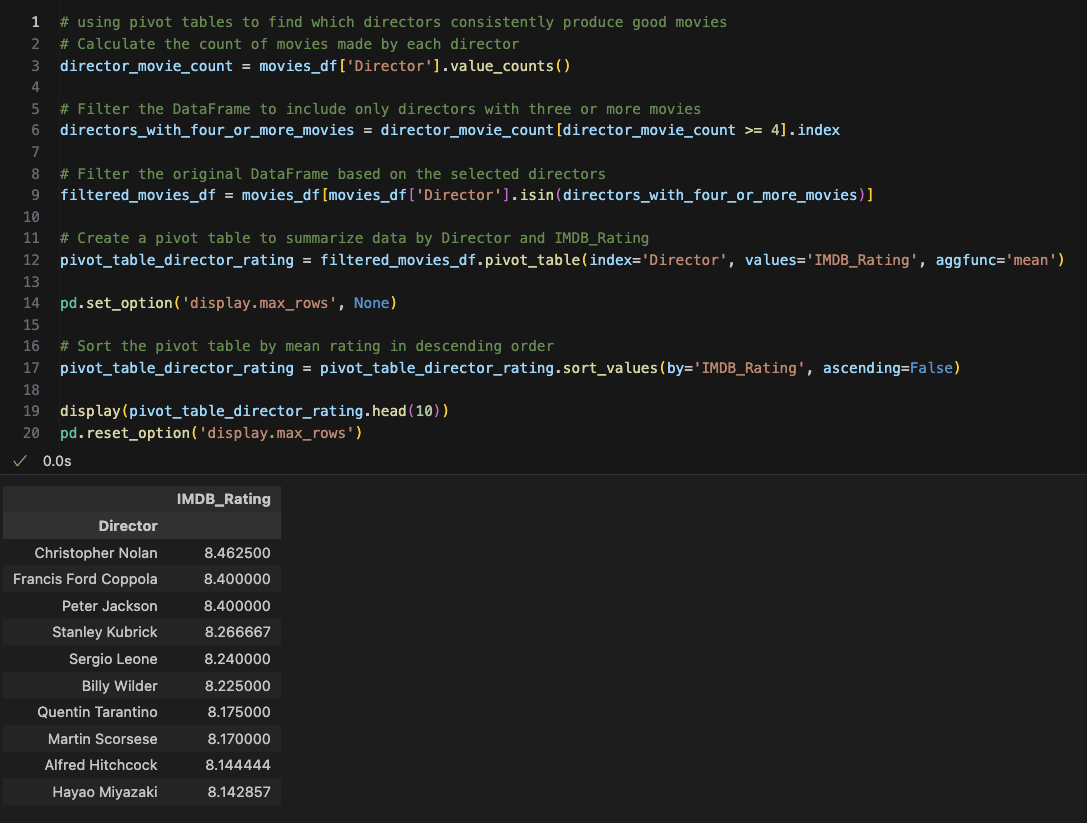
Next, I determine which movie was the highest grossing movie. To accomplish this task, I used *`idxmax()`* to get the index value of the movie that has the largest number in the column *Gross*. Funnily enough the highest grossing movie is ranked 332 with an IMDB rating of 7.9. Figure 24 shows the results.

Figure 24: Determining highest grossing movie

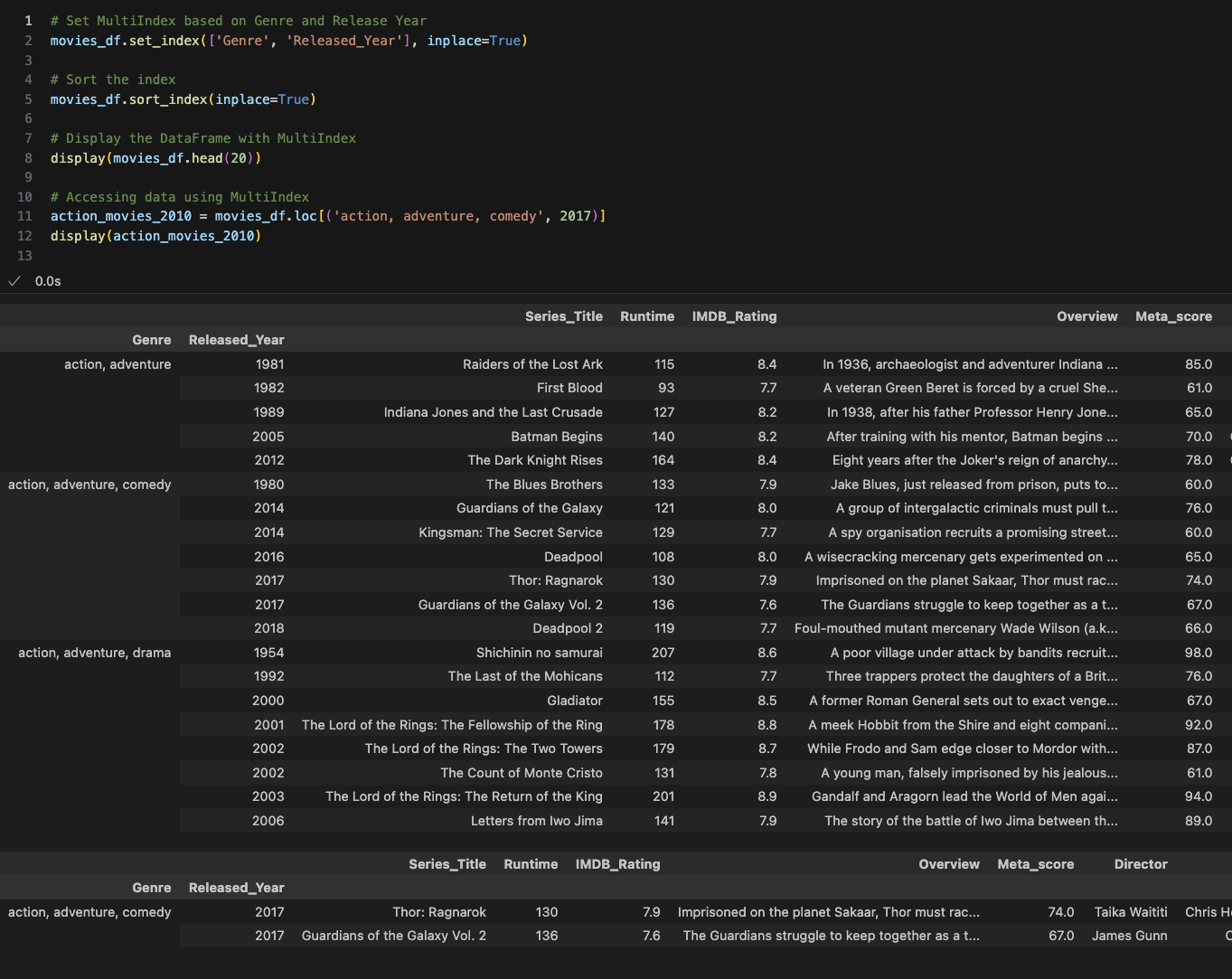
I next determine which movie genres have the highest ratings. I grouped the movies by genre and then used *`agg()`*  to look at the mean of the IMDB ratings and for fun the sum of how much money the genre brings in. Interestingly, the category of *crime, mystery, thriller*  has the highest rating at 8.50 but *action, adventure, sci-fi*  brings in the most money. Figure 25 shows the results.

Figure 25: Determining which genre has the IMDB rating and which has the highest gross

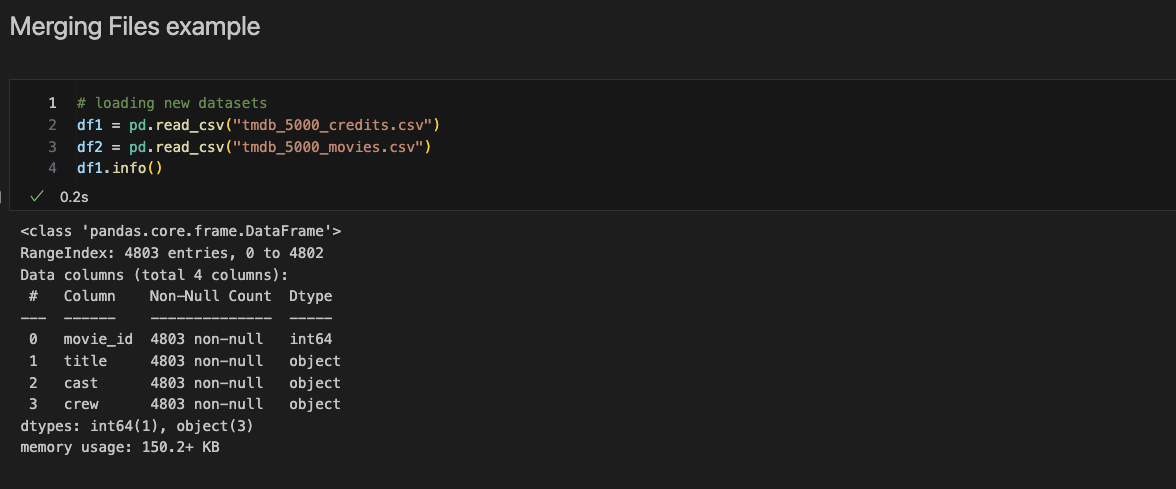
I created a pivot table to analyze the movie ratings by director to help identify which directors consistently produce highly rated movies. I filtered the data to ensure only directors who have made four or more movies would be included. Figure 26 shows the code and the results.

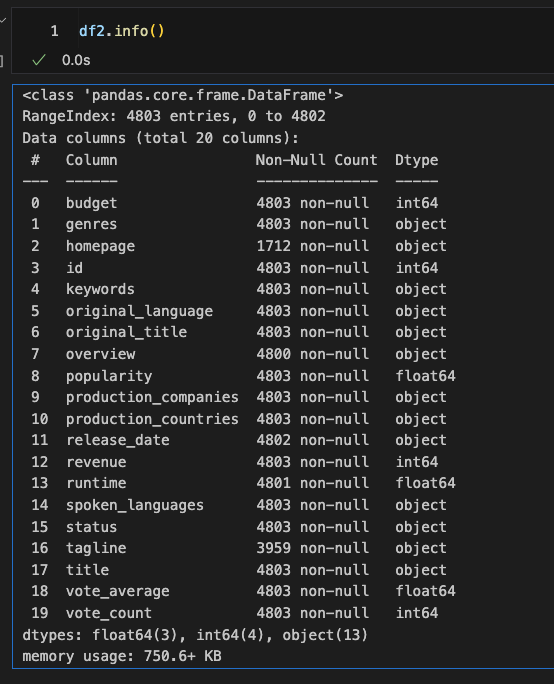
Figure 26: Discovering which directors consistently produce top movies using pivot tables

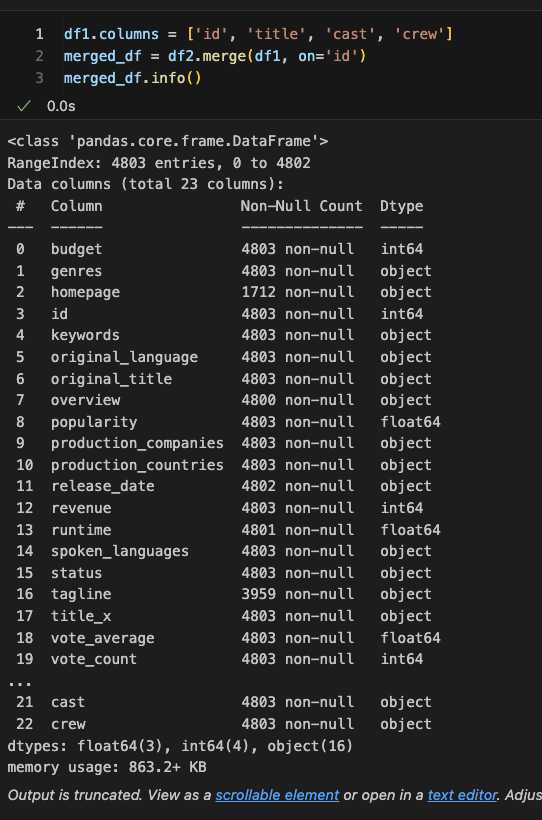
To demonstrate hierarchical indexing, I created a multi-index based on the genre and release year of the movies. This was accomplished by setting the index to *`Genre`* and *`Released\_Year`*. Then an example was written to access the data within the DataFrame using hierarchical indexing. Figure 27 shows the code and the output.

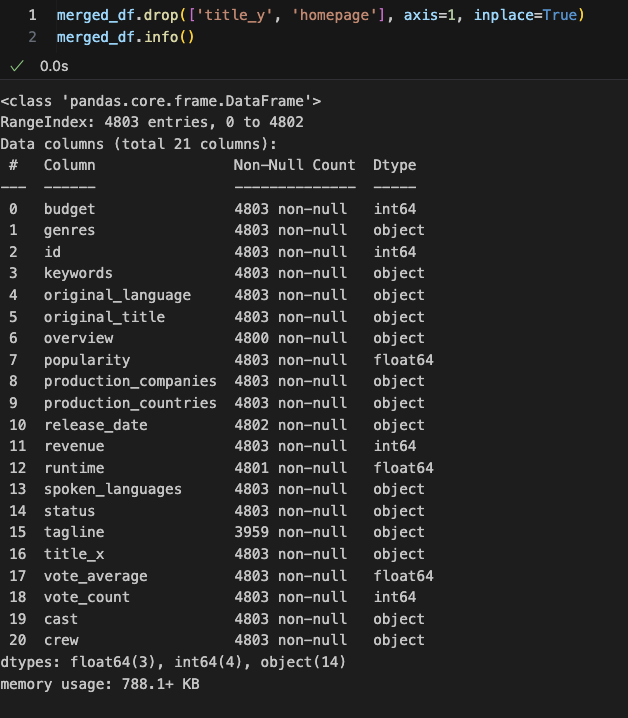
Figure 27: Demonstrating hierarchical indexing

In order to demonstrate merging files, two new data sets were imported, TMDB 5000 Credits.csv and TMDB 5000 Movies.csv(see Figure 28-29). The datasets were then merged on the movie id and the new DataFrame was created(see Figure 30). Two columns were removed from the newly merged data(see Figure 31) and the index was set to the genre and release date column(see Figure 32). To demonstrate vectorized string operations, the first word of all the movie titles was extracted and added to a new column(see figure 33-34).

Figure 28: Importing two new datasets and showing information about dataset 1

Figure 29: Dataset 2 information

Figure 30: Merging the two datasets on 'id'

Figure 31: Removing unused columns

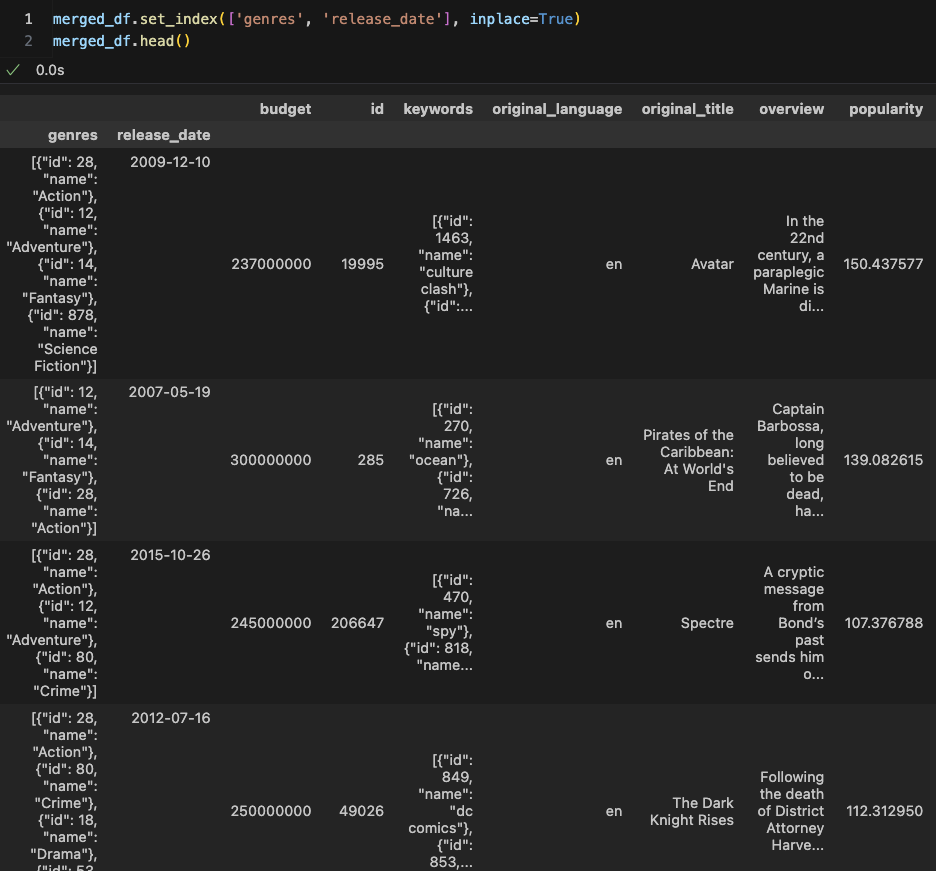
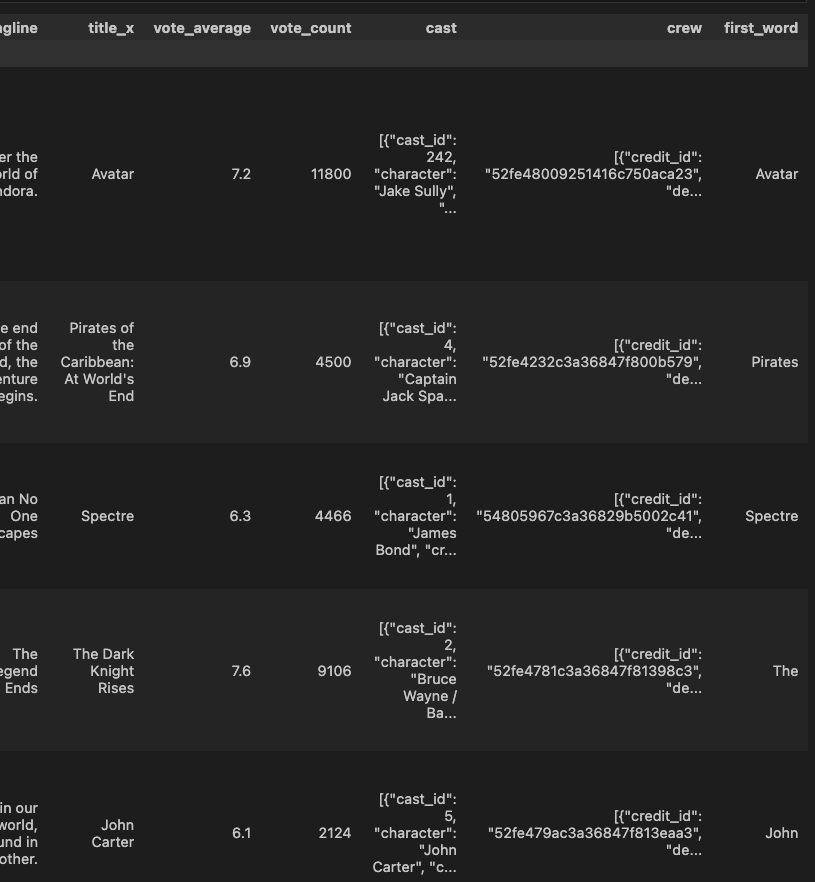
Figure 32: Setting the index to 'genres' and 'release\_date'

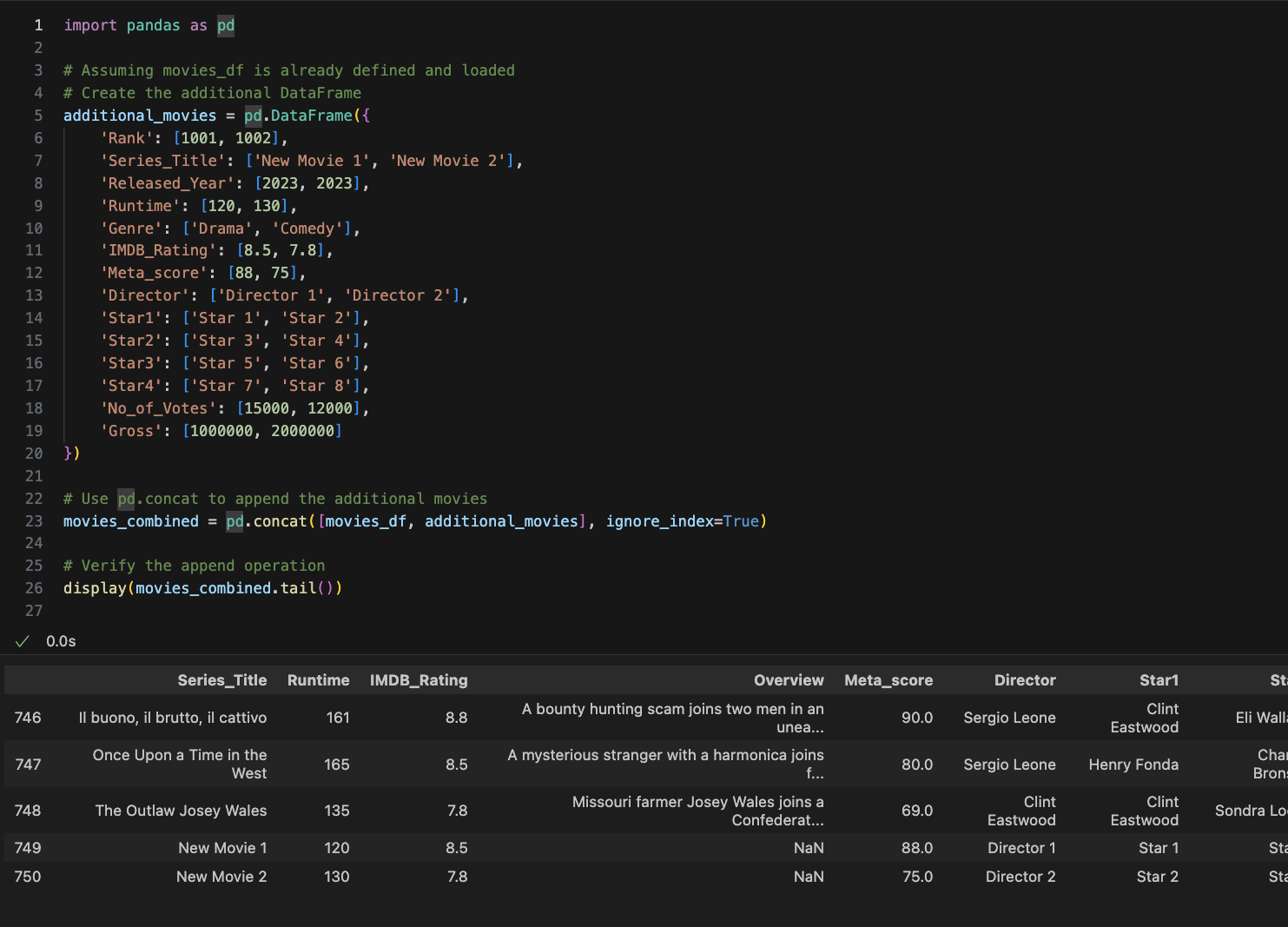
Figure 33: Code for vectorized string operations

Figure 34: Output to Figure 32 code

To demonstrate the *`concat()`* method, I split the *`movie\_df`* dataset into two halves stored in two different variables. I then combined the two halves back together in a new DataFrame using `*concat()`.* To verify this worked, I print the shape of the new DataFrame to ensure it is the same shape as the original DataFrame. Figure 35 shows the code and output.

Figure 35: Demonstrating `concat()`

To demonstrate using *`concat()`* to append new rows to an existing DataFrame, I define two generic Movies to add to the end of the list. Figure 36 shows the code and the output after appending the two new movies to the original dataset of *`movies\_df`*.

Figure 36: Using `concat()` to append new data

# 3. Conclusion

This report documents my journey in learning Pandas, a powerful data manipulation library in Python. Key concepts I explored include data structures essential for handling and analyzing structured data. I learned to perform various data operations, including data indexing, merging datasets, grouping data and more.

By using real data, I was able to explore how to go about cleaning the data up properly before beginning to analyze it. Many errors I encountered were related to missing values and data types, which are significantly different from syntax errors I encountered in previous projects. However, through practice and persistence, I was able to clean the data up and obtain datasets that could be analyzed properly using Pandas.

# References

1. Harshit Shankhdhar. (2021). IMDB Dataset of Top 1000 Movies and TV Shows. Retrieved from https://www.kaggle.com/datasets/harshitshankhdhar/IMDB-dataset-of-top-1000-movies-and-tv-shows
2. The Movie Database (TMDB). (2018). TMDB 5000 Movie Dataset. Retrieved from https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata
3. VanderPlas, J. (*2016*).  *Python Data Science Handbook*. O’Reilly Media. Retrieved from https://jakevdp.github.io/PythonDataScienceHandbook/index.html