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
1. Introduction	6
2. Top Movie Data Analysis	7
3. Conclusion	33
References	34

Table of Figures

Figure 1: Using Pandas objects and importing CSV files	7
Figure 2: Collecting information about the data	7
Figure 3: Collecting information about the dataset	8
Figure 4: Changing the index of the dataset	9
Figure 5: Adding a Rank column	10
Figure 6: Moving 'Rank' column to the beginning of the dataset	11
Figure 7: Sorting the data by IMDB rating	12
Figure 8: Finding what data is missing	13
Figure 9: Removing unused columns from the dataset	14
Figure 10: Dealing with missing data	15
Figure 11: Checking for duplicated data	16
Figure 12: Converting 'Gross' column from String to Integer	16
Figure 13: Cleaning up the 'Runtime' column	17
Figure 14: Converting release year to datatime format	18
Figure 15: Cleaning up the genre column	18
Figure 16: Determining top movie genres	19
Figure 17: Determining top 5 most voted movies	19
Figure 18: Determining bottom 5 least voted movies	20
Figure 19: Determining which movie has the longest runtime	20
Figure 20: Determining which movie has the shortest runtime	21
Figure 21: Using `query()` to filter data	21
Figure 22: Using `eval()` to calculate new data	22
Figure 23: Determining which year had the most top movies	22
Figure 24: Determining highest grossing movie	23
Figure 25: Determining which genre has the IMDB rating and which has the highest gross	24
Figure 26: Discovering which directors consistently produce top movies using pivot tables	25

Figure 27: Demonstrating hierarchical indexing	26
Figure 28: Importing two new datasets and showing information about dataset 1	27
Figure 29: Dataset 2 information	27
Figure 30: Merging the two datasets on 'id'	28
Figure 31: Removing unused columns	29
Figure 32: Setting the index to 'genres' and 'release_date'	30
Figure 33: Code for vectorized string operations	30
Figure 34: Output to Figure 32 code	31
Figure 35: Demonstrating `concat()`	32
Figure 36: Using `concat()` to append new data	33

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

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.

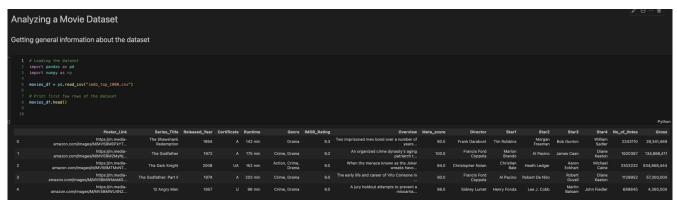


Figure 1: Using Pandas objects and importing CSV files

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.

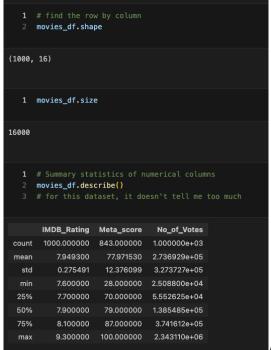


Figure 2:

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.

```
# Get general information about the dataframe
      movies_df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
    Column
                    Non-Null Count
                                    Dtype
0
    Poster Link
                    1000 non-null
                                    object
1
    Series_Title
                    1000 non-null
                                    object
2
    Released_Year
                    1000 non-null
                                    object
3
    Certificate
                    899 non-null
                                    object
4
    Runtime
                    1000 non-null
                                    object
5
    Genre
                    1000 non-null
                                    object
                    1000 non-null
                                    float64
    IMDB_Rating
7
    Overview
                    1000 non-null
                                    object
8
    Meta_score
                    843 non-null
                                    float64
                    1000 non-null
9
    Director
                                    object
10
    Star1
                    1000 non-null
                                    object
11
    Star2
                    1000 non-null
                                    object
12
    Star3
                    1000 non-null
                                    object
    Star4
                    1000 non-null
13
                                    object
    No_of_Votes
                    1000 non-null
                                    int64
14
15 Gross
                    831 non-null
                                    object
dtypes: float64(2), int64(1), object(13)
memory usage: 125.1+ KB
```

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.

<pre>1 movies_df.set_index('Series_Title', inplace=True) # Change index to be Title column 2 movies_df.head(4)</pre>										
	Poster_Link	Released_Year	Certificate	Runtime	Genre	IMDB_F				
Series_Title										
The Shawshank Redemption	https://m.media- amazon.com/images/M/MV5BMDFkYT	1994	А	142 min	Drama					
The Godfather	https://m.media- amazon.com/images/M/MV5BM2MyNj	1972	А	175 min	Crime, Drama					
The Dark Knight	https://m.media- amazon.com/images/M/MV5BMTMxNT	2008	UA	152 min	Action, Crime, Drama					
The Godfather: Part II	https://m.media- amazon.com/images/M/MV5BMWMwMG	1974	А	202 min	Crime, Drama					

Figure 4: Changing the index of the dataset

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.

```
Cleaning up the dataset
    · adding columns before analysis
    · removing rows
    · cleaning up strings

    converting data types

     1 # Add a 'Rank' column
     2 movies_df['Rank'] = range(1, len(movies_df) + 1)
     3 movies_df.info()
 <class 'pandas.core.frame.DataFrame'>
 Index: 1000 entries, The Shawshank Redemption to The 39 Steps
 Data columns (total 16 columns):
                      Non-Null Count Dtype
  # Column
  0 Poster_Link 1000 non-null object
  1 Released Year 1000 non-null object
  2 Certificate 899 non-null
  3 Runtime
                      1000 non-null object
                     1000 non-null object
  4 Genre
  5 IMDB_Rating 1000 non-null float64
      0verview
                         1000 non-null object
  6
      Meta_score
                        843 non-null
                                             float64

        8
        Director
        1000 non-null object

        9
        Star1
        1000 non-null object

        10
        Star2
        1000 non-null object

        11
        Star3
        1000 non-null object

        12
        Star4
        1000 non-null object

  13 No_of_Votes 1000 non-null int64
  14 Gross 831 non-null
                                            object
  15 Rank
                        1000 non-null int64
 dtypes: float64(2), int64(2), object(12)
 memory usage: 132.8+ KB
```

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.

```
1 # get list of column names
  2 new_columns = list(movies_df.columns)
  4 new_columns.remove('Rank')
  6 new_columns.insert(0, 'Rank')
  8 movies_df = movies_df[new_columns]
     # Check DataFrame info
  10 movies_df.info()
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, The Shawshank Redemption to The 39 Steps
Data columns (total 16 columns):
               Non-Null Count Dtype
# Column
0 Rank
                1000 non-null int64
1 Poster_Link 1000 non-null object
2 Released_Year 1000 non-null object
3 Certificate 899 non-null object
              1000 non-null object
1000 non-null object
4 Runtime
5 Genre
6 IMDB_Rating 1000 non-null float64
7 Overview
                1000 non-null object
8 Meta_score 843 non-null float64
9 Director
                1000 non-null object
10 Star1
                1000 non-null object
11 Star2
                1000 non-null object
11 Star3
                1000 non-null object
                1000 non-null object
14 No_of_Votes 1000 non-null int64
15 Gross
                 831 non-null
                                object
dtypes: float64(2), int64(2), object(12)
memory usage: 132.8+ KB
```

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

<pre># Sorting the data by ratings movies_df = movies_df.sort_values(by='IMDB_Rating', ascending=False) movies_df.reset_index(inplace=True) movies_df.set_index('Rank', inplace=True) movies_df</pre> movies_df									
Series_Title	Poster_Link	Released_Year	Certi						
Rank									
1 The Shawshank Redemption amaz	https://m.media- on.com/images/M/MV5BMDFkYT	1994							
2 The Godfather amaz	https://m.media- on.com/images/M/MV5BM2MyNj	1972							
3 The Dark Knight amazo	https://m.media- on.com/images/M/MV5BMTMxNT	2008							
4 The Godfather: Part II amazon.	https://m.media- .com/images/M/MV5BMWMwMG	1974							
5 12 Angry Men amazo	https://m.media- n.com/images/M/MV5BMWU4N2	1957							
913 Zombieland amazo	https://m.media- n.com/images/M/MV5BMTU5MD	2009							
912 La piel que habito amaz	https://m.media- on.com/images/M/MV5BYmFmNj	2011							
911 Moneyball amaz	https://m.media- zon.com/images/M/MV5BMjAxOT	2011							
910 Celda 211 ama:	https://m.media- zon.com/images/M/MV5BMjl2OD	2009							
1000 The 39 Steps amazo	https://m.media- on.com/images/M/MV5BMTY5OD	1935							
1000 rows × 16 columns									

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.

```
1 # Handling missing data
      print(movies_df.isnull().sum())
 ✓ 0.0s
Series_Title
                   0
Poster_Link
                   0
Released_Year
                   0
Certificate
                 101
Runtime
                   0
Genre
                   0
IMDB_Rating
Overview
Meta_score
                 157
Director
                   0
Star1
                   0
Star2
                   0
Star3
                   0
Star4
                   0
No_of_Votes
Gross
                 169
dtype: int64
```

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.

```
1 # drop unused columns
     movies_df.drop(['Certificate', 'Poster_Link'], axis=1, inplace=True)
     movies_df.info()
 ✓ 0.0s
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, 1 to 1000
Data columns (total 14 columns):
    Column
                 Non-Null Count Dtype
   Series_Title 1000 non-null
                                object
1 Released_Year 1000 non-null
                               object
2 Runtime
                 1000 non-null
                               object
   Genre
                 1000 non-null
                               object
   IMDB_Rating 1000 non-null float64
5 Overview
                1000 non-null object
   Meta_score 843 non-null
                               float64
   Director
               1000 non-null
                               object
                1000 non-null
   Star1
                               object
               1000 non-null
   Star2
                               object
                1000 non-null
10 Star3
                               object
11 Star4
                1000 non-null
                               object
12 No_of_Votes 1000 non-null
                                int64
13 Gross
                 831 non-null
                               object
dtypes: float64(2), int64(1), object(11)
memory usage: 117.2+ KB
```

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.

```
#movies_df.fillna(0)
   1
      #print(movies_df.isna().sum())
      #lets remove rows with null data
      movies_df.dropna(inplace=True)
      movies_df.info()
   0.0s
<class 'pandas.core.frame.DataFrame'>
Index: 750 entries, 1 to 911
Data columns (total 14 columns):
 #
     Column
                    Non-Null Count
                                     Dtype
     Series_Title
                    750 non-null
 0
                                     object
 1
     Released_Year
                    750 non-null
                                     object
 2
     Runtime
                    750 non-null
                                     object
 3
                    750 non-null
                                     object
     Genre
                    750 non-null
                                     float64
 4
     IMDB_Rating
 5
     Overview
                    750 non-null
                                     object
 6
                    750 non-null
                                     float64
     Meta_score
 7
     Director
                    750 non-null
                                     object
                    750 non-null
 8
     Star1
                                     object
 9
                    750 non-null
     Star2
                                     object
                    750 non-null
 10
    Star3
                                     object
 11
    Star4
                    750 non-null
                                     object
 12 No_of_Votes
                    750 non-null
                                     int64
 13 Gross
                    750 non-null
                                     object
dtypes: float64(2), int64(1), object(11)
memory usage: 87.9+ KB
```

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.

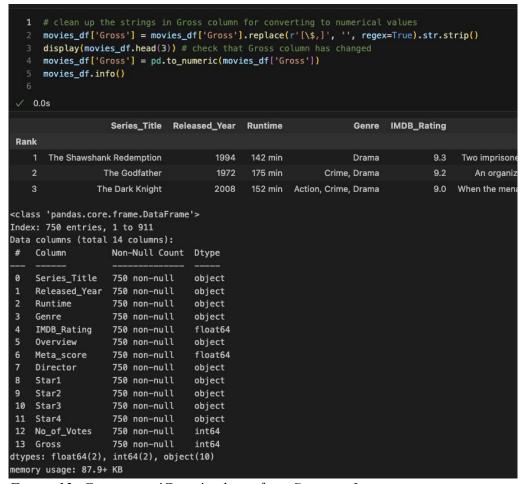


Figure 12: Converting 'Gross' column from String to Integer

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.

<pre>1 movies_df['Runtime'] = movies_df['Runtime'].str.replace(' min', '') 2 # Convert the 'Runtime' column to numeric type 3 movies_df['Runtime'] = pd.to_numeric(movies_df['Runtime']) 4 display(movies_df.head()) 5 print(f"Data type for 'Runtime' is: {movies_df['Runtime'].dtype}'') # checking runtime datatype</pre>										
	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating					
Rank										
1	The Shawshank Redemption	1994	142	Drama	9.3	Two imprisoned men bond over				
2	The Godfather	1972	175	Crime, Drama	9.2	An organized crime dynasty				
3	The Dark Knight	2008	152	Action, Crime, Drama	9.0	When the menace known as the				
4	The Godfather: Part II	1974	202	Crime, Drama	9.0	The early life and career				
5 Data t	,									

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.

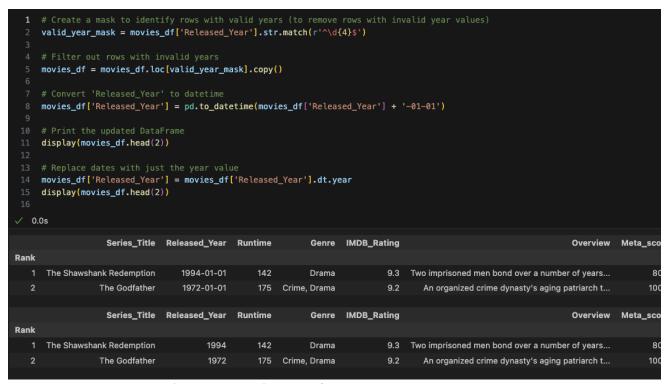


Figure 14: Converting release year to datatime format

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.

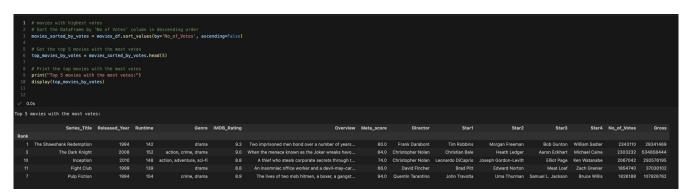


Figure 17: Determining top 5 most voted movies

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.

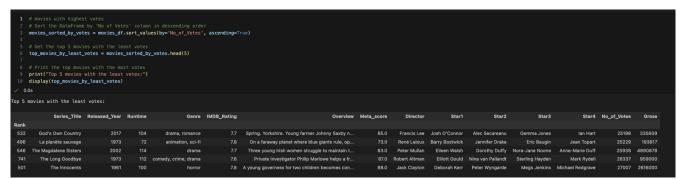


Figure 18: Determining bottom 5 least voted movies

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.

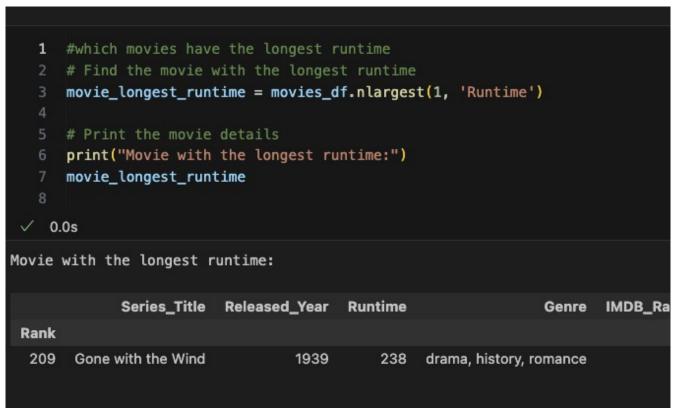


Figure 19: Determining which movie has the longest runtime

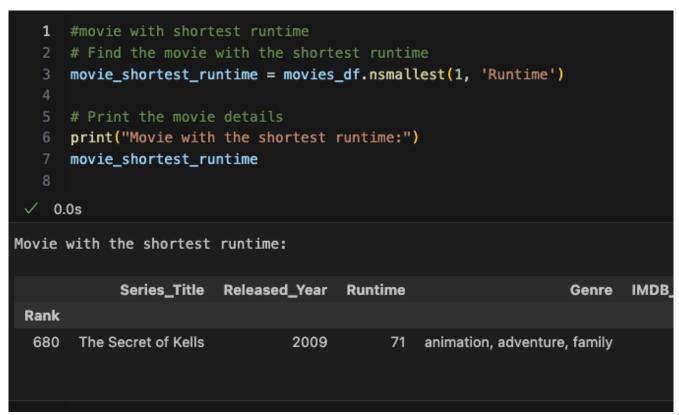


Figure 20: Determining which movie has the shortest runtime

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.

1 2 3	<pre>1 # showing query 2 filtered_movies = movies_df.query('IMDB_Rating > 8.0 and Released_Year > 2004 and Meta_score > 90.0') 3 filtered_movies</pre>										
✓ 0.0											
	Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Overview	Meta_score	D			
Rank											
21	Gisaengchung	2019	132	comedy, drama, thriller	8.6	Greed and class discrimination threaten the ne	96.0	Bong J			
57	WALL-E	2008	98	animation, adventure, family	8.4	In the distant future, a small waste-collectin	95.0	Andrew !			
72	Jodaeiye Nader az Simin	2011	123	drama	8.3	A married couple are faced with a difficult de	95.0	Asghar			
114	Toy Story 3	2010	103	animation, adventure, comedy	8.2	The toys are mistakenly delivered to a day-car	92.0	Lee l			
115	Pan's Labyrinth	2006	118	drama, fantasy, war	8.2	In the Falangist Spain of 1944, the bookish yo	98.0	Guillermo d			
116	There Will Be Blood	2007	158	drama	8.2	A story of family, religion, hatred, oil and m	93.0	Paul Thomas An			
143	Spotlight	2015	129	biography, crime, drama	8.1	The true story of how the Boston Globe uncover	93.0	Tom Mo			
145	Portrait de la jeune fille en feu	2019	122	drama, romance	8.1	On an isolated island in Brittany at the end o	95.0	Céline So			
146	12 Years a Slave	2013	134	biography, drama, history	8.1	In the antebellum United States, Solomon North	96.0	Steve Mo			
153	Inside Out	2015	95	animation, adventure, comedy	8.1	After young Riley is uprooted from her Midwest	94.0	Pete			
163	No Country for Old Men	2007	122	crime, drama, thriller	8.1	Violence and mayhem ensue after a hunter stumb	91.0	Etha			

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.

2	1 # showing eval() 2 df = movies_df.eval('Total_Earnings = Gross * No_of_Votes') 3 df.head(5) < 0.0s														
Rank		Released_Year	Runtime	Genre	IMDB_Rating	Overview	Meta_score	Director	Star1	Star2	Star3	Star4	No_of_Votes	Gross	Total_Earnings
1	The Shawshank Redemption	1994	142	drama	9.3	Two imprisoned men bond over a number of years	80.0	Frank Darabont	Tim Robbins	Morgan Freeman	Bob Gunton	William Sadler	2343110	28341469	66407179428590
2	The Godfather			crime, drama		An organized crime dynasty's aging patriarch t	100.0	Francis Ford Coppola	Marlon Brando	Al Pacino	James Caan	Diane Keaton	1620367	134966411	218695118492837
3	The Dark Knight	2008		action, crime, drama		When the menace known as the Joker wreaks havo	84.0	Christopher Nolan	Christian Bale	Heath Ledger	Aaron Eckhart	Michael Caine	2303232	534858444	1231903083691008
4	The Godfather: Part II	1974		crime, drama		The early life and career of Vito Corleone in		Francis Ford Coppola	Al Pacino	Robert De Niro	Robert Duvall	Diane Keaton	1129952	57300000	64746249600000
5	12 Angry Men	1957		crime, drama		A jury holdout attempts to prevent a miscarria	96.0	Sidney Lumet	Henry Fonda	Lee J. Cobb	Martin Balsam	John Fiedler	689845	4360000	3007724200000

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.

```
# determine which year had the most released movies
      best_year = movies_df.groupby('Released_Year').count()['Series_Title'].nlargest(10)
      #best_year.shape
   4 pd.set_option('display.max_rows', None)
      display(best_year)
      pd.reset_option('display.max_rows')
 ✓ 0.0s
Released_Year
2014
        29
2004
        28
2001
        24
2009
        24
2013
        24
2007
        23
2006
        22
2003
        21
1993
        20
2010
        20
Name: Series_Title, dtype: int64
```

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.

```
1 # Grouping the movies and looking at
    2 genre_group = movies_df.groupby('Genre').agg({'IMDB_Rating': 'mean', 'Gross': 'sum'})
    3 print(genre_group.sort_values(by='IMDB_Rating', ascending=False).head(10))
    4 print("\n")
    5 print(genre_group.sort_values(by='Gross', ascending=False).head(10))
  √ 0.0s
                                      IMDB_Rating
                                                           Gross
Genre
crime, mystery, thriller
                                              8.50 23341568
action, sci-fi
                                             8.40 414723280
horror, sci-fi
                                             8.40 78900000
drama, mystery, war
                                              8.35 90328607
                                              8.35
                                                     58221508
western
mystery, romance, thriller 8.30 3200000 crime, drama, sci-fi 8.30 6207725 comedy, musical, romance 8.30 8819028 adventure, mystery, thriller 8.30 13275000 crime, drama, fantasy 8.25 138923439
                                     IMDB_Rating
                                                            Gross
Genre
action, adventure, sci-fi
                                         7.928571 5898659459
animation, adventure, comedy
                                         7.925000 4503337598
action, adventure, drama 8.225000 2668835626
action, adventure, fantasy 8.200000 2116341031
drama 7.942424 20101111145
                                        7.942424 2010111145
animation, action, adventure 7.960000 2008694522
drama, romance 7.929630 1962226898 action, adventure, comedy 7.828571 1911202048 action, crime, drama 7.845000 1308133687 7.933333 1255884472
crime, drama, thriller
                                         7.933333 1255884472
```

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.

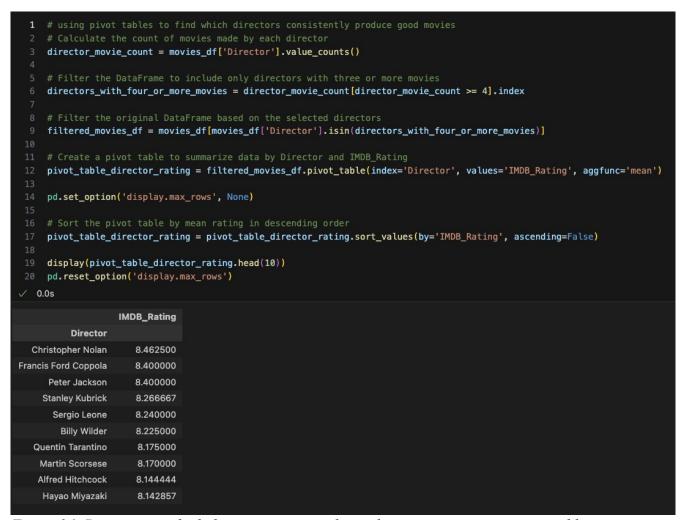


Figure 26: Discovering which directors consistently produce top movies using pivot tables

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.

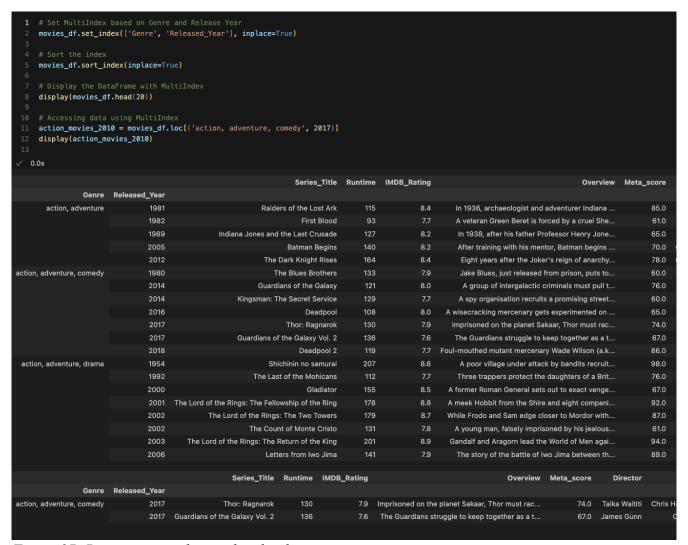


Figure 27: Demonstrating hierarchical indexing

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).

```
Merging Files example
   1 # loading new datasets
    df1 = pd.read_csv("tmdb_5000_credits.csv")
   3 df2 = pd.read_csv("tmdb_5000_movies.csv")
   4 df1.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 4 columns):
 # Column Non-Null Count Dtype
    movie_id 4803 non-null int64
    title 4803 non-null object
 2 cast
              4803 non-null object
 3 crew
              4803 non-null object
dtypes: int64(1), object(3)
memory usage: 150.2+ KB
```

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

```
1 df2.info()
 ✓ 0.0s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 20 columns):
    Column
                         Non-Null Count Dtype
                       4803 non-null
                                        int64
0
   budget
   genres
1
                        4803 non-null object
2
    homepage
                        1712 non-null
                                        object
                        4803 non-null int64
3
    id
    keywords
                        4803 non-null object
   original_language 4803 non-null
original_title 4803 non-null
                                        object
                                        object
                        4800 non-null object
   overview
   popularity
                        4803 non-null float64
    production_companies 4803 non-null
                                        object
10 production_countries 4803 non-null
                                        object
11 release_date
                        4802 non-null
                                        object
                         4803 non-null
12 revenue
                                        int64
13 runtime
                       4801 non-null
                                        float64
 14 spoken_languages
                       4803 non-null
                                        object
                       4803 non-null
 15 status
                                        object
                        3959 non-null
 16 tagline
                                        object
 17 title
                        4803 non-null
                                        object
                4803 non-null
18 vote_average
                                        float64
19 vote_count
                         4803 non-null
                                        int64
dtypes: float64(3), int64(4), object(13)
memory usage: 750.6+ KB
```

Figure 29: Dataset 2 information

_-

```
1 df1.columns = ['id', 'title', 'cast', 'crew']
   2 merged_df = df2.merge(df1, on='id')
   3 merged_df.info()
 ✓ 0.0s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 23 columns):
    Column
                          Non-Null Count
                                          Dtype
                          4803 non-null
                                          int64
 0
    budget
                          4803 non-null
 1
    genres
                                          object
 2
    homepage
                          1712 non-null
                                         object
                          4803 non-null int64
3
    id
 4
    keywords
                          4803 non-null
                                         object
                          4803 non-null object
 5
    original_language
                          4803 non-null
 6
    original_title
                                         object
7
    overview
                          4800 non-null
                                         object
    popularity
                          4803 non-null
 8
                                         float64
 9
    production_companies 4803 non-null
                                         object
 10
    production_countries 4803 non-null
                                         object
 11 release_date
                          4802 non-null
                                          object
 12 revenue
                          4803 non-null int64
                          4801 non-null
 13 runtime
                                         float64
                          4803 non-null object
 14 spoken_languages
 15 status
                          4803 non-null
                                         object
 16 tagline
                          3959 non-null
                                          object
 17 title_x
                          4803 non-null
                                          object
 18 vote average
                          4803 non-null
                                          float64
 19 vote_count
                          4803 non-null
                                          int64
. . . .
 21 cast
                          4803 non-null
                                          object
22 crew
                          4803 non-null
                                          object
dtypes: float64(3), int64(4), object(16)
memory usage: 863.2+ KB
Output is truncated. View as a scrollable element or open in a text editor. Adjus
```

Figure 30: Merging the two datasets on 'id'

```
merged_df.drop(['title_y', 'homepage'], axis=1, inplace=True)
   2 merged_df.info()
✓ 0.0s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 21 columns):
    Column
                         Non-Null Count Dtype
                          4803 non-null int64
    budget
0
                          4803 non-null
1
                                         object
    genres
                          4803 non-null int64
2
    id
3
                         4803 non-null object
    keywords
    original_language
                         4803 non-null object
4
5
    original title
                         4803 non-null object
6
    overview
                         4800 non-null object
7
    popularity
                         4803 non-null float64
8
    production_companies
                         4803 non-null
                                         object
    production_countries
                         4803 non-null object
9
10 release_date
                         4802 non-null object
                         4803 non-null int64
11 revenue
12 runtime
                         4801 non-null float64
    spoken_languages
                         4803 non-null object
13
14 status
                         4803 non-null object
15 tagline
                         3959 non-null object
                         4803 non-null
16 title_x
                                         object
17 vote_average
                         4803 non-null float64
18
    vote_count
                         4803 non-null int64
19 cast
                         4803 non-null
                                         object
20 crew
                         4803 non-null
                                         object
dtypes: float64(3), int64(4), object(14)
memory usage: 788.1+ KB
```

Figure 31: Removing unused columns

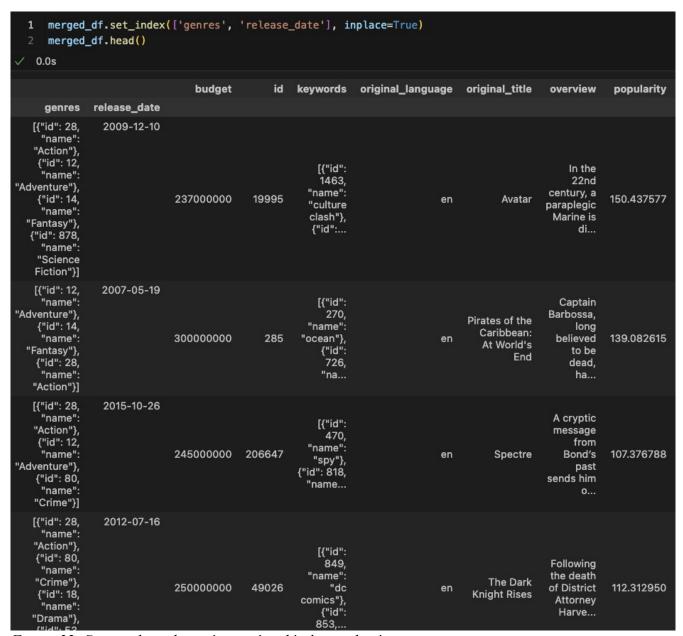


Figure 32: Setting the index to 'genres' and 'release date'

```
1  # vecterized string operations
2  merged_df['first_word'] = merged_df['title_x'].str.split().str[0]
3  merged_df

0.0s
```

Figure 33: Code for vectorized string operations

ıgline	title_x	vote_average	vote_count	cast	crew	first_word
er the orld of ndora.	Avatar	7.2	11800	[{"cast_id": 242, "character": "Jake Sully", "	[{"credit_id": "52fe48009251416c750aca23", "de	Avatar
e end of the d, the enture egins.	Pirates of the Caribbean: At World's End	6.9	4500	[{"cast_id": 4, "character": "Captain Jack Spa	[{"credit_id": "52fe4232c3a36847f800b579", "de	Pirates
an No One capes	Spectre	6.3	4466	[{"cast_id": 1, "character": "James Bond", "cr	[{"credit_id": "54805967c3a36829b5002c41", "de	Spectre
The egend Ends	The Dark Knight Rises	7.6	9106	[{"cast_id": 2, "character": "Bruce Wayne / Ba	[{"credit_id": "52fe4781c3a36847f81398c3", "de	The
in our world, und in other.	John Carter	6.1	2124	[{"cast_id": 5, "character": "John Carter", "c	[{"credit_id": "52fe479ac3a36847f813eaa3", "de	John

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'.

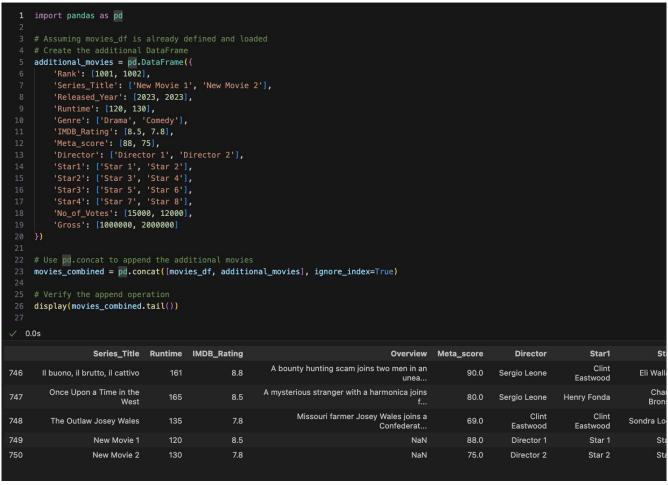


Figure 36: Using 'concat()' to append new data

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