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 4](#__RefHeading___Toc2569_331549091)

[1. Introduction 5](#__RefHeading___Toc2573_331549091)

[2. Top Movie Data Analysis 6](#__RefHeading___Toc2577_331549091)

[3. Conclusion 7](#__RefHeading___Toc2609_331549091)

[References 8](#__RefHeading___Toc3692_331549091)

Table of Figures:

# 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 |  |
| Data Indexing and Selection |  |
| Operating on Data in Pandas |  |
| Handling Missing Data |  |
| Hierarchical Indexing |  |
| Combining Datasets: Concat and Append |  |
| Combining Datasets: Merge and Join |  |
| Aggregation and Grouping |  |
| Pivot Tables |  |
| Vectorized String Operations |  |
| Working with Time Series |  |
| High-Performance Pandas: eval() and query() |  |

# 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. 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 from the year( year). 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.

Fig

Using *`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 ## shows the code and output.

Fig.

To better understand the data, I used the *`shape()`* , *`size()`,* and `*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 ## shows the code and output.

Fig

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 ### shows this output.

Fig

After viewing the first few rows of data and the last few rows using the *`head()`* and *`tail()`* methods respectively, it seemed 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#), I decided to add a rank column as seen in Figure #.

Fig

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 # shows the output.

Fig

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 # shows the code and output

Fig

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 ## shows the code and results.

Fig

I didn’t want the Certificate column or the Poster\_Link column, so I decided to remove both. Figure ## 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.

Fig

*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 ## shows the code and output. This reduces the number of entries to 750, meaning I lost 25% of my data.

Fig

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

Fig

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 ## shows my updated code and output and shows that the Gross column is now a type int64.

Fig

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 ## shows the results.

Fig

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 ## shows the code and result.

Fig

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

Fig

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 ## shows the code and results. The top movie genres are drama, comedy, romance, thriller, and crime.

Fig

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. Fig ## shows the code and results.

Fig

I then do the same thing but to see which movies had the least number of votes. Figure ## 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.

Fig

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 ## and ## show the respective code along with the results.

Fig

Fig

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 ## shows the results.

Fig

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 ## shows the code and results.

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 shows this code and output.

Fig

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 ## shows the results.

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 ## shows the results.

Fig

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 # shows the code and the results.

Fig

To demonstrate high-performance pandas using `*eval()`* and `*query()`* I restructured the code from the pivot tables example to use these

For this project I attempt to begin the programming of a recommendation system for movies. Due to the scope ocf this class and time limitations, only the beginning movie data analysis will be completed. In subsequent projects, there may be an attempt to implement a complete recommendation guide.

Pandas and NumPy were imported into the file and the dataset for movies’ credits and movies’ metadata were read into the python script as csv files, creating two DataFrames.

# 3. Conclusion

This report documents my journey in mastering NumPy which involved grasping fundamental concepts such as arrays, computations using universal functions and broadcasting, incorporating comparisons and boolean logic into arrays, indexing and sorting arrays, and learning how to create structured arrays.

Through the process of coding, many errors were encountered. Due to the simplicity of the projects and exercises, all of the errors were syntax errors. My experience is mainly in C and C++ so I am used to the nuance of adding semicolons to the end of each line. Many times in Python, I found myself accidentally adding semicolons to the ends of blocks and statements, confused why my code block continually showed errors. Other examples of syntax errors I encountered include forget a quotation, forgetting a colon when defining a function or loop, or simply misspelling a variable that I named.

Engaging in mini-projects and working through the exercises has proved to be a valuable source in aiding me to take the theory of Python fundamentals and the NumPy package and put the concepts into practice and think about practical data science applications. This has allowed me to grasp the concepts that make the NumPy package a valuable tool for efficiently managing large sets of data.

# References

1. VanderPlas, J. (*2016*).  *Python Data Science Handbook*. O’Reilly Media. Retrieved from https://jakevdp.github.io/PythonDataScienceHandbook/index.html