IS532: Theory & Practice of Data Cleaning

Topic: Cleaning and Analysis of IMDb TV Series Dataset

Team: Akshay Bafna, Meghna Shrivastava, Tafseer Khan

University of Illinois at Urbana-Champaign Fall 2019

Abstract: *This report describes a data cleaning workflow using the IMDb dataset as an example to demonstrate various cleaning techniques and identifying use-cases for the cleaned dataset. This project will aim to clean the dataset and implement a few use cases. The following software and tools have been used to clean, organize and analyze the dataset: Python, Jupyter Notebook, Google Colab, YesWorkflow & Tableau.*

Introduction

IMDb (Internet Movie Database) is an online database of information related to films, television programs, home videos, video games, and streaming content online – including cast, production crew and personal biographies, plot summaries, trivia, fan and critical reviews, and ratings.. Originally a fan-operated website, the database is owned and operated by IMDb.com, Inc., a subsidiary of Amazon. Our dataset has been obtained from the official website of IMDb.

Initial Data Inspection

1. Dataset Initial Assessment

1.1. **Dataset structure, content and use-cases**:

The IMDb dataset for this project has been initially separated into 7 different files. Each dataset is contained in a gzipped, tab-separated-values (TSV) formatted file in the UTF-8 character set. The first line in each file contains headers that describe what is in each column. A *'\N'* is used to denote that a particular field is missing or null for that title/name.

The available datasets are as follows:

Title.akas.tsv.gz

- titleId (string) - a tconst, an alphanumeric unique identifier of the title
- ordering (integer) – a number to uniquely identify rows for a given titleId
- title (string) – the localized title
- region (string) - the region for this version of the title

- language (string) - the language of the title
- types (array) - Enumerated set of attributes for this alternative title. One or more of the following: "alternative", "dvd", "festival", "tv", "video", "working", "original", "imdbDisplay". New values may be added in the future without warning
- attributes (array) - Additional terms to describe this alternative title, not enumerated
- isOriginalTitle (boolean) – 0: not original title; 1: original title

Title.basics.tsv.gz

- tconst (string) - alphanumeric unique identifier of the title
- titleType (string) – the type/format of the title (e.g. movie, short, tvseries, tvepisode, video, etc)
- primaryTitle (string) – the more popular title / the title used by the filmmakers on promotional materials at the point of release
- originalTitle (string) - original title, in the original language
- isAdult (boolean) - 0: non-adult title; 1: adult title
- startYear (YYYY) – represents the release year of a title. In the case of TV Series, it is the series start year
- endYear (YYYY) – TV Series end year. '\N' for all other title types
- runtimeMinutes – primary runtime of the title, in minutes
- genres (string array) – includes up to three genres associated with the title

Title.crew.tsv.gz

- tconst (string) - alphanumeric unique identifier of the title
- directors (array of nconsts) - director(s) of the given title
- writers (array of nconsts) – writer(s) of the given title

Title.episode.tsv.gz

- tconst (string) - alphanumeric identifier of episode
- parentTconst (string) - alphanumeric identifier of the parent TV Series
- seasonNumber (integer) – season number the episode belongs to
- episodeNumber (integer) – episode number of the tconst in the TV series

Title.principals.tsv.gz

- tconst (string) - alphanumeric unique identifier of the title
- ordering (integer) – a number to uniquely identify rows for a given titleId
- nconst (string) - alphanumeric unique identifier of the name/person
- category (string) - the category of job that person was in
- job (string) - the specific job title if applicable, else '\N'
- characters (string) - the name of the character played if applicable, else '\N'

Title.ratings.tsv.gz

- tconst (string) - alphanumeric unique identifier of the title
- averageRating – weighted average of all the individual user ratings
- numVotes - number of votes the title has received

Name.basics.tsv.gz

- nconst (string) - alphanumeric unique identifier of the name/person
- primaryName (string)– name by which the person is most often credited
- birthYear – in YYYY format
- deathYear – in YYYY format if applicable, else '\N'
- primaryProfession (array of strings)– the top-3 professions of the person
- knownForTitles (array of tconsts) – titles the person is known for

1.2. Use-Cases defined

- Relationship between the runtime and popularity of a TV Series.
- Popularity of a genre based on region.
- Analyze how the popularity of a TV series changed over time.
- Popularity of directors based on genre.
- Predict the movies rating based on multiple factors such as genre, director.

1.3. Achieving the desired fitness of the dataset from the use-cases defined

Talking about one of the use-cases for e.g. Popularity of Directors based on genres, there has been a lot of data-wrangling and pre-processing steps involved. To achieve the results of this use-case, we realized that there has to be a clear understanding about how the 7 datasets needs to be handled and merged together. Before that, proper analysis of each attribute with respect to what insights it tries to communicate was done to design the overall merging process and achieve the desired concatenated outcome. We also sensed that it was necessary to check for the redundancy and duplication of records in each dataset and handle those before merging them together. Also, we focused on merging of the data frames based on most desired attributes to make our final dataset more concise and precise. We also focused on tweaking the datasets in a way that can help clearly classify the directors with respect to genres for each TV series. To achieve results for each use-case mentioned, there wasn't any cleaned dataset for it. We had to really tweak, concatenate and clean the row values to get the results for each use-case.

2. Initial Data Quality issues

Initial Analysis of the data files had quality issues related to duplicates, incomplete data, inconsistent formats and accessibility. It was a bit difficult to initially understand the data files when scrutinized separately. Individual attributes made more sense when combined with other supporting information but to make each attribute more complete and meaningful, we spent time identifying the desired goals based on which we defined the purpose of each attribute present in 7 different datafiles and merged them. Identifying the duplicate records for each TV series was a bit difficult since the naming conventions were different for each duplicate record but had the same unique title_ids and other attribute values. We also focused on supporting field values to confirm the presence of duplicate records. We had to thoughtfully analyze and choose what naming conventions were most appropriate for our dataset. Also, certain records in the data files were incomplete to an extent that it was easier to interpret the additional information needed to make it complete and understandable. We also came across inconsistent formatting while loading and reading these datafiles in the pandas dataframe.

Data Cleaning Methods and Process

We chose to work with the Python script and Tableau to achieve the results for this project. The reason behind choosing to work with python is: while OpenRefine is specifically used for data cleaning and transformation process, we were more eager to explore the strengths of python being an object-oriented programming language in data cleaning process. We wanted to work through developing the scripts instead of a tool based approach. Python is known for its elegant syntax and readable code as well as supports many data analysis, cleaning and transforming packages and libraries such as pandas etc. which helped us achieve the same goal with more elaborative effort. In short, we chose to work by tweaking things from scratch instead of selecting and performing well- defined operations/processes already present in the OpenRefine.

We also made use of the tableau at the end to visualize the results. We chose tableau since it integrates well with python.

Process

- Initially there were 7 datafiles namely:
  - ❏ Title.basics.tsv
  - ❏ Title.akas.tsv
  - ❏ Title.crew.tsv
  - ❏ Title.episode.tsv
  - ❏ Title.principals.tsv
  - ❏ Title.ratings.tsv
  - ❏ Name.basics.tsv
- We began with reading each datafile in pandas dataframe. While loading the files, we made sure to make use of sep='\t' and 'UTF-8' encoding to load the information in a proper format. This step is repeated for each datafile. We also narrowed the choice to only those attributes that we wanted to keep in order to achieve the goals e.g. we chose to remove the 'writers' column since there wasn't any key information present that would have led to loss of information. We made sure to do the selection with minimal or no loss of information.
- We then performed analysis on the dataframe 'Title.akas.tsv' to first check whether we had desired title names for each Television series. We wanted to only keep the records having original titles and hence chose the attribute' isOriginalTitle' to filter out the records that had original names in them. We performed the regular expression operation to remove the special characters, white spaces and other irrelevant information.

- After this initial stage, we spent a lot of time studying about each feature present in different dataframes. From here onwards, we had to make sure that our analysis should be proper to an extent that we get just one desired meaningful dataframe at the end. We defined our use cases and performed the concatenation operation based on the final requirements as follows:
    - ❏ Concatenated Title.principals.tsv and Name.basics.tsv data frames based on attribute ‘nconst’. Read it into the pandas dataframe to ensure correct format of the dataset. Name: ‘Master_table1’
    - ❏ Combined ‘Master_table1’ with dataframe Title.basics.tsv on the attribute 'tconst'. Named it as: ‘Master_table2’.
    - ❏ Combined ‘Master_table2’ with dataframe Title.crew.tsv on the attribute 'tconst'. Named it as: ‘Master_table3’.
    - ❏ Combined ‘Master_table3’ with dataframe Title.episode.tsv on the attribute 'tconst'. Named it as: ‘Master_table4’.
    - ❏ Combined ‘Master_table4’ with dataframe Title.ratings.tsv on the attribute 'tconst'. Named it as: ‘Master_table5’.
    - ❏ Master_table5 is a single dataframe formed by concatenating Title.principals.tsv, Name.basics.tsv, Title.basics.tsv, Title.crew.tsv, Title.episode.tsv and Title.ratings.tsv. Before merging Master_table5 with the Title.akas.tsv, we found that there isn’t any common attribute present between them. Question arises: How to merge these two dataframes into one? We analyzed every single attribute present in Master_table5, Title.akas.tsv and derived an interesting insight i.e. the attributes ‘titleId’ and ‘tconst’ are the same just the naming is different. To help non-technical audiences understand the process better, we chose to replace the name ‘tconst’ with ‘titleId’ in Master_table5. Once these results were achieved, we combined the Master_table5 with Title.akas.tsv and generated the final dataset ‘final_dataframe’.
    - ❏ There are 32 attributes present in the ‘final_dataframe’. We again performed a quick check to see if there are any repetition of the records or any columns. Proper analysis for the dataset helped us realize that there were attributes having different column names but has the same information namely ‘primaryTitle’, ‘originalTitle’, ‘title’. We had to choose any one of the attributes and narrowed it down to ‘primaryTitle’ since it had more refined names that could be well understood by others. Hence, dropped ‘originalTitle’ and ‘title’. We also got rid of ‘originalTitle’ as its main purpose of differentiating original titles from regional title names was already handled.

- We removed the columns that had no further significance with our project namely 'ordering_x', 'ordering_y', 'types' etc and formed a final_dataframe having 26 most relevant features/attributes. We also ran regular expression code to get rid of the special characters and undesired values.

```
final_dataframe.dtypes
```

```
titleId                object
nconst                 object
region                 object
language               object
attributes             object
category               object
job                    object
characters             object
primaryName            object
birthYear              object
deathYear              object
primaryProfession      object
knownForTitles         object
titleType              object
primaryTitle           object
isAdult                 int64
startYear              object
endYear                object
runtimeMinutes         object
genres                 object
directors              object
parentTconst           object
seasonNumber           object
episodeNumber          object
averageRating         float64
numVotes                int64
dtype: object
```

Fig (a): final_dataframe: 26 attributes

Integrity Constraints Identified

We defined the set of rules for final_dataframe to ensure that the data insertion, updation, and other processes gets performed in such a way that data integrity doesn't get affected. We defined the following set of rules:

- **title_id** should be unique.
- **startyear** should not be zero 0
- **runtimeMinutes** should not zero
- **seasonNumber** cannot be 0
- **averageRating** cannot be greater than 10 and smaller than or equal to zero.

We then converted the final_dataframe into the CSV file to proceed with the visualization of the use-cases.

Tableau Visualization

We tried visualizing the results for 2 of our defined uses cases:

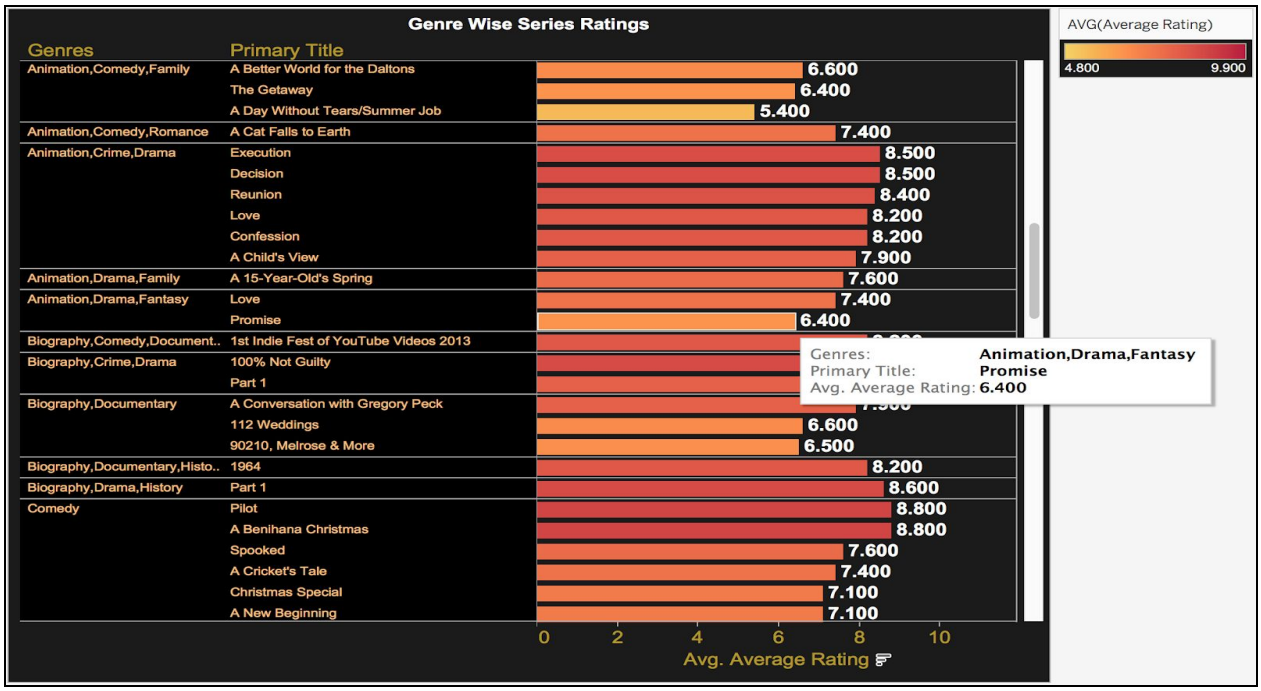


Fig (b): Use-case: Define genre wise series ratings

In this Visualization, We represented different TV series in various categories of genres. The TV series are not confined to one genre and can be considered in multiple categories. Hence, there is a wide range of list of genre considering multiple category in one. The visualization depicts the ratings corresponding to the particular TV series. The color variation shows the type of rating with dark red color shows higher rating and light yellow color shows lower rating.

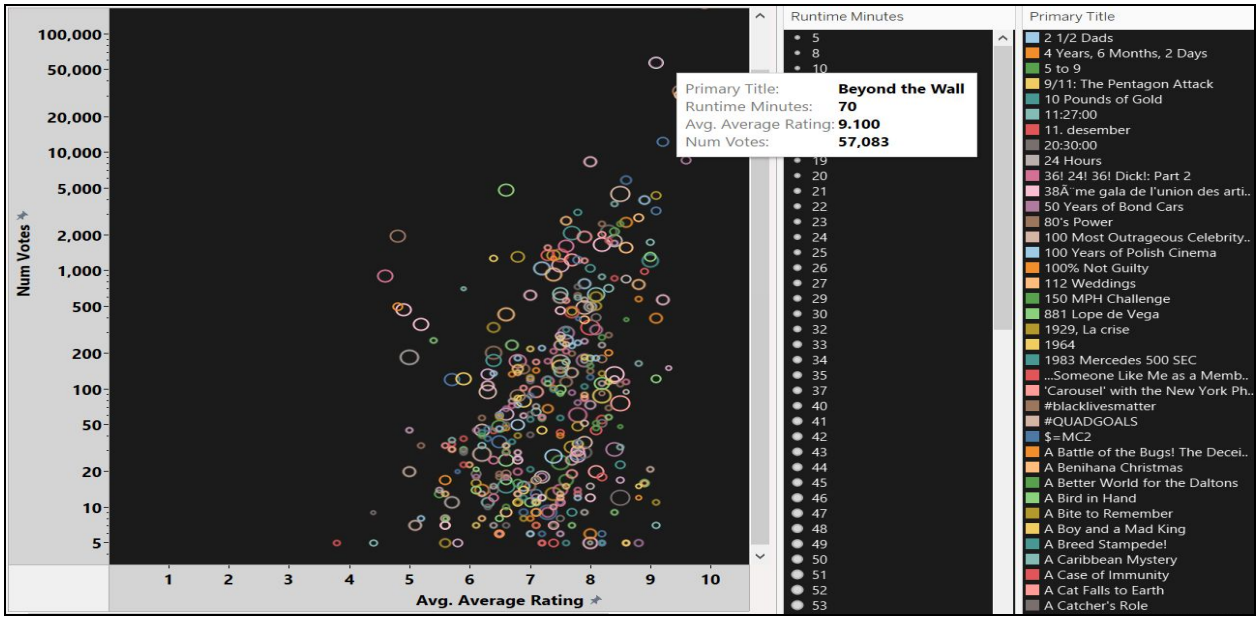


Fig (c): use-case: identifying popularity based on number of votes, average rating and runtime minutes

This visualization represents the major findings from the IMDb dataset by keeping into consideration Number of votes, Average Rating and the Runtime Minutes. The graph helps us to understand the importance of number of votes to see the popularity of TV series with the ratings assigned.

Looking at visualization, it seems that movies with larger runtime have received large number of votes as well as higher rating, whereas low runtime TV series are not preferred by viewers leading to less number of votes, hence, the ratings for such TV series can be doubted.

An interesting point which can be noted from the graph is that no TV series has an average rating lower than 3 .

Results

Before Cleaning

titleId	ordering	title	region	language	types	attributes	isOriginalTitle
tt0000001	1	Carmencit	HU	\N	imdbDispla	\N	0
tt0000001	2	Î šÎ±Î¼Î¼Î½	GR	\N	\N	\N	0
tt0000001	3	ÐšÐ°Ñ€Ð¼	RU	\N	\N	\N	0
tt0000001	4	Carmencit	US	\N	\N	\N	0
tt0000001	5	Carmencit	\N	\N	original	\N	1
tt0000002	1	Le clown e	\N	\N	original	\N	1
tt0000002	2	A bohÃ³c Ã	HU	\N	imdbDispla	\N	0
tt0000002	3	Le clown e	FR	\N	\N	\N	0
tt0000002	4	Clovnul si	RO	\N	imdbDispla	\N	0
tt0000002	5	ÐšÐ»Ð¾ÑƒÐ	RU	\N	\N	\N	0

tconst	directors	writers
tt0000001	nm000569	\N
tt0000002	nm072152	\N
tt0000003	nm072152	\N
tt0000004	nm072152	\N
tt0000005	nm000569	\N
tt0000006	nm000569	\N
tt0000007	nm000569	\N
tt0000008	nm000569	\N
tt0000009	nm008515	nm0085156

tconst	parentTco	seasonNur	episodeNumber
tt0041951	tt0041038	1	9
tt0042816	tt0989125	1	17
tt0042889	tt0989125	\N	\N
tt0043426	tt0040051	3	42
tt0043631	tt0989125	2	16
tt0043693	tt0989125	2	8

tconst	averageRa	numVotes
tt0000001	5.6	1545
tt0000002	6.1	187
tt0000003	6.5	1202
tt0000004	6.2	114
tt0000005	6.1	1924
tt0000006	5.2	102

tconst	titleType	primaryTit	originalTitl	isAdult	startYear	endYear	runtimeMi	genres
tt0000001	short	Carmencit	Carmencit	0	1894	\N	1	Documentary,Short
tt0000002	short	Le clown e	Le clown e	0	1892	\N	5	Animation,Short
tt0000003	short	Pauvre Pie	Pauvre Pie	0	1892	\N	4	Animation,Comedy,Romance
tt0000004	short	Un bon bo	Un bon bo	0	1892	\N	\N	Animation,Short
tt0000005	short	Blacksmith	Blacksmith	0	1893	\N	1	Comedy,Short

Fig (d): Original Datasets before Cleaning: few captured screenshots

Trimming all the leading and trailing white spaces in the all the columns results

```
In [38]: ### Define function
def trim_head_tail_space(df_column):
    clean_df_column = df_column.str.strip()
    print("row changes in column " + str(df_column.name) +": " ,sum(df_column!=clean_df_column))
    return clean_df_column


### Run function
table['titleId'] = trim_head_tail_space(table['titleId'])
table['nconst'] = trim_head_tail_space(table['nconst'])
table['region'] = trim_head_tail_space(table['region'])
table['language'] = trim_head_tail_space(table['language'])
table['attributes'] = trim_head_tail_space(table['attributes'])
table['category'] = trim_head_tail_space(table['category'])
table['job'] = trim_head_tail_space(table['job'])
table['characters'] = trim_head_tail_space(table['characters'])
table['primaryName'] = trim_head_tail_space(table['primaryName'])
table['birthYear'] = trim_head_tail_space(table['birthYear'])
table['deathYear'] = trim_head_tail_space(table['deathYear'])
table['primaryProfession'] = trim_head_tail_space(table['primaryProfession'])
table['primaryTitle'] = trim_head_tail_space(table['primaryTitle'])
table['startYear'] = trim_head_tail_space(table['startYear'])
table['endYear'] = trim_head_tail_space(table['endYear'])
table['runtimeMinutes'] = trim_head_tail_space(table['runtimeMinutes'])
table['genres'] = trim_head_tail_space(table['genres'])
table['directors'] = trim_head_tail_space(table['directors'])
table['parentTconst'] = trim_head_tail_space(table['parentTconst'])
table['seasonNumber'] = trim_head_tail_space(table['seasonNumber'])
table['episodeNumber'] = trim_head_tail_space(table['episodeNumber'])
```

```
row changes in column titleId:  0
row changes in column nconst:  0
row changes in column region:  0
row changes in column language:  0
row changes in column attributes:  0
row changes in column category:  0
row changes in column job:  0
row changes in column characters:  0
row changes in column primaryName:  0
row changes in column birthYear:  0
row changes in column deathYear:  0
row changes in column primaryProfession:  0
row changes in column primaryTitle:  3
row changes in column startYear:  5057
row changes in column endYear:  0
row changes in column runtimeMinutes:  4204
row changes in column genres:  0
row changes in column directors:  0
row changes in column parentTconst:  0
row changes in column seasonNumber:  4787
row changes in column episodeNumber:  4787
```

Collapse consecutive white spaces

```
In [39]: ### Define function
def remove_consecutive_spaces(df_column):
    clean_df_column = df_column.replace('\s+', ' ', regex=True)
    print("row changes in column " + str(df_column.name) +": " ,sum(df_column!=clean_df_column))
    return clean_df_column

### Run function
table['titleId'] = remove_consecutive_spaces(table['titleId'])
table['nconst'] = remove_consecutive_spaces(table['nconst'])
table['region'] = remove_consecutive_spaces(table['region'])
table['language'] = remove_consecutive_spaces(table['language'])
table['attributes'] = remove_consecutive_spaces(table['attributes'])
table['category'] = remove_consecutive_spaces(table['category'])
table['job'] = remove_consecutive_spaces(table['job'])
table['characters'] = remove_consecutive_spaces(table['characters'])
table['primaryName'] = remove_consecutive_spaces(table['primaryName'])
table['birthYear'] = remove_consecutive_spaces(table['birthYear'])
table['deathYear'] = remove_consecutive_spaces(table['deathYear'])
table['primaryProfession'] = remove_consecutive_spaces(table['primaryProfession'])
table['primaryTitle'] = remove_consecutive_spaces(table['primaryTitle'])
table['startYear'] = remove_consecutive_spaces(table['startYear'])
table['endYear'] = remove_consecutive_spaces(table['endYear'])
table['runtimeMinutes'] = remove_consecutive_spaces(table['runtimeMinutes'])
table['genres'] = remove_consecutive_spaces(table['genres'])
table['directors'] = remove_consecutive_spaces(table['directors'])
table['parentTconst'] = remove_consecutive_spaces(table['parentTconst'])
table['seasonNumber'] = remove_consecutive_spaces(table['seasonNumber'])
table['episodeNumber'] = remove_consecutive_spaces(table['episodeNumber'])
```

```
row changes in column titleId:  0
row changes in column nconst:  0
row changes in column region:  0
row changes in column language:  0
row changes in column attributes:  0
row changes in column category:  0
row changes in column job:  2
row changes in column characters:  1
row changes in column primaryName:  4
row changes in column birthYear:  0
row changes in column deathYear:  0
row changes in column primaryProfession:  0
row changes in column primaryTitle:  22
row changes in column startYear:  5057
row changes in column endYear:  0
row changes in column runtimeMinutes:  4204
row changes in column genres:  0
row changes in column directors:  0
row changes in column parentTconst:  0
row changes in column seasonNumber:  4787
row changes in column episodeNumber:  4787
```

After Cleaning

	titleId	primaryTitle	startYear	runtimeMi	genres	directors	parentTco	seasonNur	episodeNu	averageRa	numVotes	isAdult	category	nconst
0	tt0045960	King Lear	1953	75	Drama,His	nm056717	tt0044284	2	3	7	147	0	writer||ac	nm000063
1	tt0046855	A Christmas Carol	1954	60	Adventure	nm050658	tt0046643	1	4	6.3	107	0	writer||ac	nm000204
2	tt0048378	The Miracle on 34th Street	1955	46	Drama	nm082903	tt0047702	1	6	5.9	182	0	director||	nm082903
3	tt0048462	The Ox-Bow Incident	1955	60	Drama	nm065263	tt0047702	1	3	7.2	11	0	actor||sel	nm028592
4	tt0057671	The Forgotten Army	1963	25	Document	nm065191	tt1027590	2	18	7	29	0	director||	nm065191
5	tt0057828	Because I Love You	1964	56	Drama	nm065191	tt1030630	1	24	8	6	0	actor||act	nm075495
6	tt0058180	A Rebel's Fortress	1964	25	Document	nm065191	tt1027590	2	64	3.8	5	0	actor||dir	nm086580
7	tt0059753	The Cage	1966	63	Action,Adv	nm012511	tt0060028	1	0	7.7	4676	0	actor||act	nm039846
8	tt0063100	How to Irritate People	1969	68	Talk-Show	nm028599	tt0260607	No Inform	No Inform	6.9	1965	0	actress||v	nm051119
9	tt0064648	Portrait of a Dead Girl	1970	98	Action,Crir	nm017161	tt0065317	1	0	6.3	109	0	actor||act	nm030188
10	tt0064678	Mao Tse-Tung and the Cultural Revolution	1969	50	Document	nm065191	tt1052491	No Inform	No Inform	5	5	0	director||	nm065191
11	tt0064705	My Dog, the Thief: Part 1	1969	88	Adventure	nm082903	tt0046593	16	2	6.6	81	0	actress||d	nm000647
12	tt0064725	Pilot	1969	98	Drama,Far	nm075596	tt0065327	1	0	7.6	1459	0	actress||a	nm000107
13	tt0065074	Wer klingelt schon zur Fernsehzeit	1970	60	Crime,Drai	nm087227	tt0242211	8	1	6.7	6	0	productior	nm025756
14	tt0065381	Alias Smith and Jones	1971	74	Western	nm050621	tt0066625	1	1	7.7	444	0	actor||wr	nm087586
15	tt0065487	The Boy Who Stole the Elephant: Part 1	1970	60	Adventure	nm012843	tt0046593	17	2	7.4	23	0	actress||p	nm036989
16	tt0066235	Powderkeg	1971	93	Action,Adv	nm038222	tt0066631	1	0	6.7	135	0	editor||ac	nm077446
17	tt0066521	Vendetta for the Saint: Part 1	1969	95	Action,Crir	nm064022	tt0055701	6	15	6.6	226	0	writer||pr	nm015354
18	tt0066931	Dead Weight	1971	76	Crime,Drai	nm080691	tt1466074	1	3	6.9	1933	0	actor||pro	nm000073
19	tt0066932	Murder by the Book	1971	76	Crime,Drai	nm000022	tt1466074	1	1	7.7	3147	0	cinematog	nm000579
20	tt0066933	Ransom for a Dead Man	1971	95	Crime,Drai	nm041032	tt1466074	1	0	7.7	2092	0	actress||d	nm033551
21	tt0066934	Suitable for Framing	1971	76	Crime,Drai	nm000219	tt1466074	1	4	7.7	2095	0	cinematog	nm000579
22	tt0066964	Il 2 giugno	1971	42	Drama	nm000112	tt6074004	1	2	8.1	8	0	director||	nm000112
23	tt0067209	The Homecoming: A Christmas Story	1971	100	Drama,Far	nm017703	tt0068149	1	0	8.7	772	0	actor||act	nm000194
24	tt0067509	O'Hara, U.S. Treasury	1971	120	Drama	nm091613	tt0066693	1	0	7.8	32	0	composer	nm000612
25	tt0067743	She Stoops to Conquer	1971	No Inform	Drama	nm025455	tt2162303	No Inform	No Inform	9.1	15	0	writer||ac	nm032611
26	tt0068395	Blueprint for Murder	1972	75	Crime,Drai	nm000039	tt1466074	1	7	7.6	1820	0	actress||c	nm065671
27	tt0068396	Dagger of the Mind	1972	98	Crime,Drai	nm070368	tt1466074	2	4	7.1	1754	0	actor||dir	nm040503
28	tt0068397	Death Lends a Hand	1971	76	Crime,Drai	nm046860	tt1466074	1	2	7.8	2207	0	actor||cin	nm000153

Fig (e): Screenshot of the final_dataframe

Integrity constraints checked results

```
In [37]: #check integrity constraints
         #Check if titleid Identifier is unique
         id_series_titleid = table['titleId']
         if (len(set(id_series)) == len(id_series)):
             print ("ids are unique!")
         else:
             print ("there is a duplicate!")

         Yes, all ids are unique!
```

```
In [41]: #startyear constraint check
         id_series_startyear = table['startYear']
         if (len(id_series2) !='0'):
             print ("It's valid")
         else:
             print ("it's not valid")

         It's valid
```

```
In [42]: #runtime constraint check
         id_series_runtime = table['runtimeMinutes']
         if (len(id_series_runtime) !='0'):
             print ("It's valid")
         else:
             print ("it's not valid")

         It's valid
```

```
In [56]: ##averagerating constraint check
         id_series_averageRating = table['averageRating']
         if (0.0<=len(id_series_averageRating)<= 10.0):
             print ("It's valid")
         else:
             print ("it's not valid")
         #(len(id_series_averageRating))

         it's valid
```

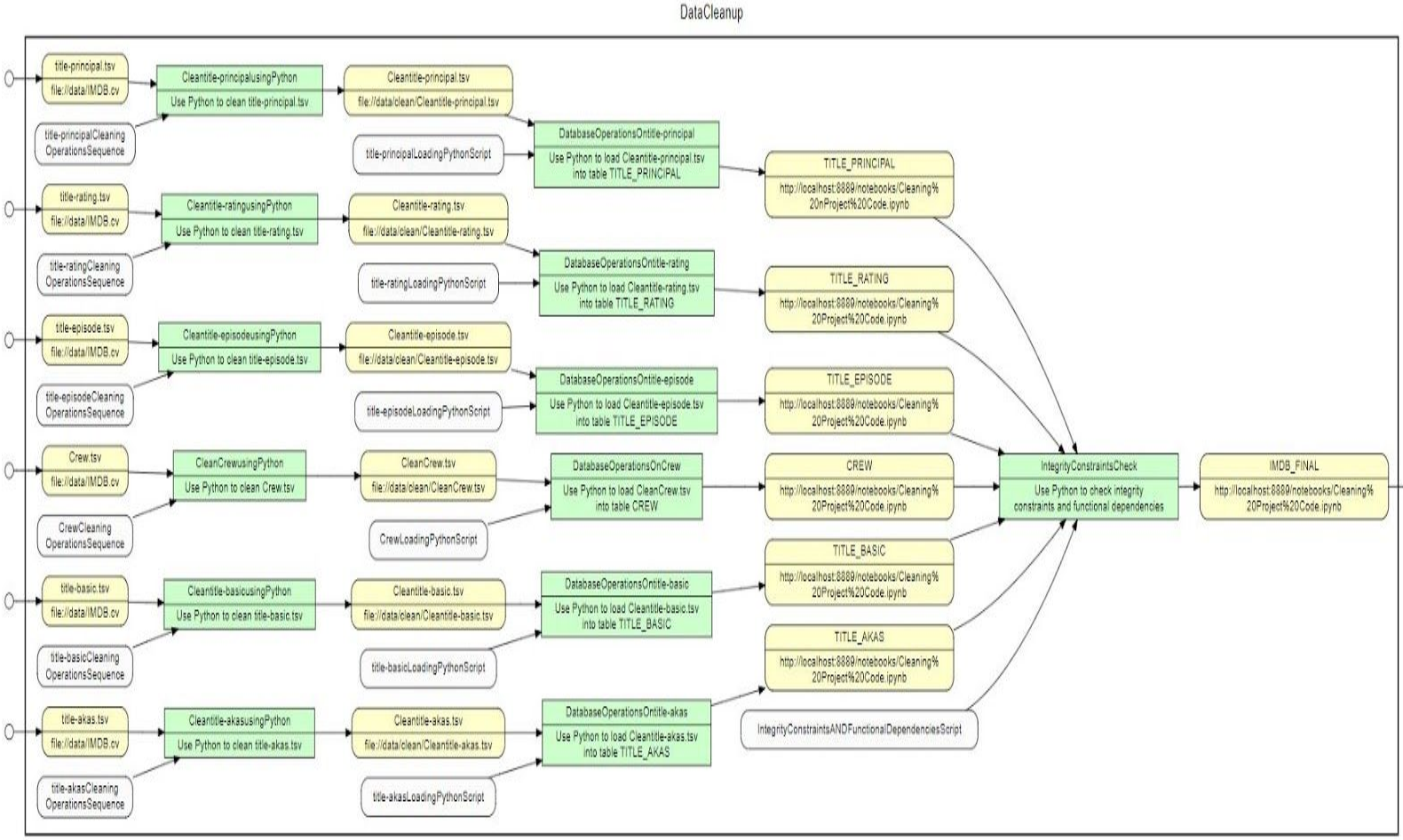
```
In [57]: ##season_number constraint check
         id_series_seasonnumber = table['seasonNumber']
         if (len(id_series_seasonnumber) !='0'):
             print ("It's valid")
         else:
             print ("it's not valid")

         It's valid
```

Conclusion and Future Work

Data Cleaning or Data Wrangling is not only an effective technique for removing the unwanted data 'dirty' data, but also the medium to make data in our system selective, concise and appropriate in order to perform better data analysis. Based on our experience, we can say Data Cleaning is one of the most time-consuming steps in any Data Analysis. We tried to clean the data using Python and implement a couple of use cases with the cleaned data using Tableau. Identifying integrity constraints and implementing them in Python took a lot of time. When data is this huge, there is no way to completely verify the integrity of our steps since the IMDb dataset is continuously growing on a daily basis. But we hope that the cleaned dataset that has been obtained can be used in the future for further analysis without much thought on cleaning it. Further work certainly involves experimenting with new methods to achieve the best results for this dataset.

WorkFlow Model



YesWorkFlow was used to create a simple workflow to provide an overview of the steps that were taken to clean our dataset. YesWorkFlow is a tool used for annotating data workflow. It's easy to script and doesn't require you to re-write any of the existing code. Simply add special (YW) comments to your existing script. Later, these comments are used to declare how the data was transformed. YesWorkflow requires neither the use of a workflow engine nor the overhead of adapting code to run effectively in such a system. Instead, YesWorkflow enables scientists to annotate existing scripts with special comments that reveal the computational modules and dataflows otherwise implicit in these scripts. YesWorkflow tools extract and analyze these comments, represent the scripts in terms of entities based on the typical scientific workflow model, and provide graphical renderings of this workflow-like view of the scripts. Future versions of YesWorkflow also will allow the prospective provenance of the data products of these scripts to be queried in ways similar to those available to users of scientific workflow systems.

References

1. https://arxiv.org/abs/1502.02403 “YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts”
2. https://pandas.pydata.org/pandas-docs/stable/
3. https://docs.scipy.org/doc/