MCSDS: CS-513 – Data Cleaning Project

Cleaning TMDB 5000 Movie Dataset

Manas Kumar Mukherjee
2018-12-01

Executive Summary

This is the final project report of the ‘CS-513 – Theory and Practice of Data Cleaning’ course. This project has practical application of all the tools and techniques like OpenRefine, SQLite, YesWorkflow etc which were taught by the professor in this graduate course. The ‘5000 Movies’ dataset is obtained from the Kaggle website.

Reference - https://www.kaggle.com/tmdb/tmdb-movie-metadata

Table of Contents

1 Overview

1.1 Dataset

1.1.1 Source

This dataset is taken from the Kaggle's website. Kaggle team created this dataset using the TMDb's ('The movie database') API.
Ref - https://www.kaggle.com/tmdb/tmdb-movie-metadata

1.1.2 Structure and content

The dataset comes with the following two csv files i.e. tmdb_5000_movies and tmdb_5000_credits.
The movies-data has 4803 records with 20 attributes/entry and credits_data has 4803 of records with only 4 attributes/entry.

Here is the list of attributes of the two data files.

Movies

SL	Attribute name	Type
1	budget	Numeric
2	genres	Text
3	homepage	Text
4	id	Numeric
5	keywords	Text
6	original_language	Text
7	original_title	Text
8	overview	Text
9	popularity	Text
10	production_companies	Text
11	production_countries	Text
12	release_date	Text
13	revenue	Numeric
14	runtime	Text
15	spoken_languages	Text
16	status	Text
17	tagline	Text
18	title	Text
19	vote_average	Text
20	vote_count	Numeric

Credits

SL	Attribute name	Type
1	movie_id	Numeric
2	title	Text
3	cast	Text
4	crew	Text

1.1.3 Data quality

Overall structure of the data is good. A significant number of attributes like genres, keywords etc. contain JSON data which made this data a little complex. It seems that multiple normalized tables were merged while creating this dataset. Since all fields are filled out by users, there are some inconsistencies on the keywords, genres, taglines etc. 'Budget' value is marked as zero for a significant number of movies. As per the guideline given by the Kaggle team, zero values should be considered as missing values.

1.2 Data cleaning goals

- Make this data more readable for the end-users.
- Systematically process the data to make it consumable by users for machine learning projects.
- Get rid of JSON values and create normalized tables where possible.

1.3 Use-cases and usability

1.3.1 Use-cases where this data can be used in its current form

It can be used in simple analytics and/or data visualization project where all attributes are not required and data quality is not a major concern.
Ex – Report on number of movies released per year, Average budget and revenue increase over time, Correlations with runtime and rating etc.

1.3.2 Target use-cases

After cleaning the dataset, it will be more readable and readily useful for advanced analytics and machine learning projects.
Ex – To produce data backed answers for question like – "Do star actors drive the success of movies" ? Here is a reference to HBS post on this topic. Ref - http://www.people.hbs.edu/aelberse/papers/hbs_06-002.pdf
It can also be used in recommender systems, topic modeling based on title, revenue forecasting of production companies etc.

2 Data cleaning with OpenRefine and Pandas

The web-interface of the OpenRefine tool is used to identify the data issues exist in this dataset. It helps to parse JSON, remove inconsistencies in words using clustering, standardize data format, removed leading/trailing spaces from text fields etc.

Here is the detailed description of the operations made on the two data files.

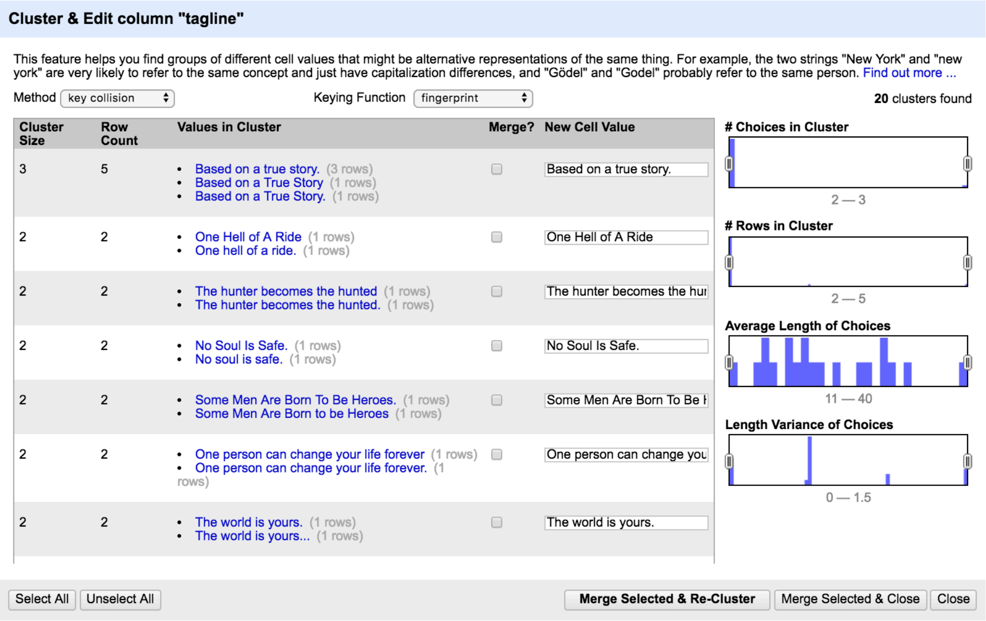
2.1 Data processing using OpenRefine

2.1.1 Movies data – Issues and corresponding fixes

SL	Attribute name	Data issues	Operations
1	budget	~25% of the movies have missing budget information. This data is not readily available in the TMDb or IMDb datasets.	It can be fetched by scrapping the movie specific IMDB page but this is *outside of the scope of this project.*
2	genres	Hard to read/consume JSON data	JSON data is parsed using OpenRefine and two new columns called genres_id_name_pair and genres_list are created. 'genres_list' will help to interpret the data easily. It was further processed by pandas to encode this categorical data using binary encoding(useful for ML dataset)

3	homepage	NA	A new attribute is created using the extracted domain. Might be useful for the consumers of this dataset. Clojure function - ‘.getHost()’ is used to extract the domain names.
4	keywords	Hard to read/consume JSON data. Unwanted leading/trailing spaces	JSON data is parsed using OpenRefine. An in-place update is done with the list of keywords associated with each movie.
5	original_title	Unwanted characters like “%,,@,/,#,!,[,],(,),\?”	All unwanted characters are removed using OpenRefine
6	overview	Unwanted characters like “%,,@,/,#,!,[,],(,),\?” and leading/trailing spaces	All unwanted characters and spaces are removed using OpenRefine
7	popularity	Represented as text and too many digits after the decimal point.	Type is converted from ‘Text’ to ‘Number’ and data is rounded up to 2 decimal places. Formula(GREL) used - value*100).round()/100.0
8	production_companies	Represented in JSON	A new column called “production_companies_list” list created that contains semicolon separated names of the production companies.
9	production_countries	Represented in JSON	A new column called “production_countries_list” list created that contains semicolon separated country code(iso-639-1) of the production countries.
10	release_date	Data is represented in ‘MM/DD/YY’.	An in-place update is done after converting each date in the standard ISO-8601 recommended format i.e ‘YYYY-MM-DD’.
11	revenue	Represented as ‘Text’	Using OpenRefine, datatype is converted from ‘Text’ to ‘Number’
12	spoken_languages	Represented as JSON	A new column called “ spoken_languages_list” is created that contains semicolon separated language code(iso-639-1) of the spoken languages.
13	tagline	Inconsistent wordings used by users	Using OpenRefine’s Text-Facet/Clustering feature, wordings are made consistent across 44 taglines.
14	title	Unwanted characters like “%,,@,/,#,!,[,],(,),\?” and leading/trailing spaces	All unwanted characters are removed for 50 movies.
15	vote_average	NA	NA
16	vote_count	NA	NA

Here is the snapshot of the clustering step where using ‘Key collision’ method and ‘fingerprint’ function, inconsistent wordings issue across taglines are fixed.



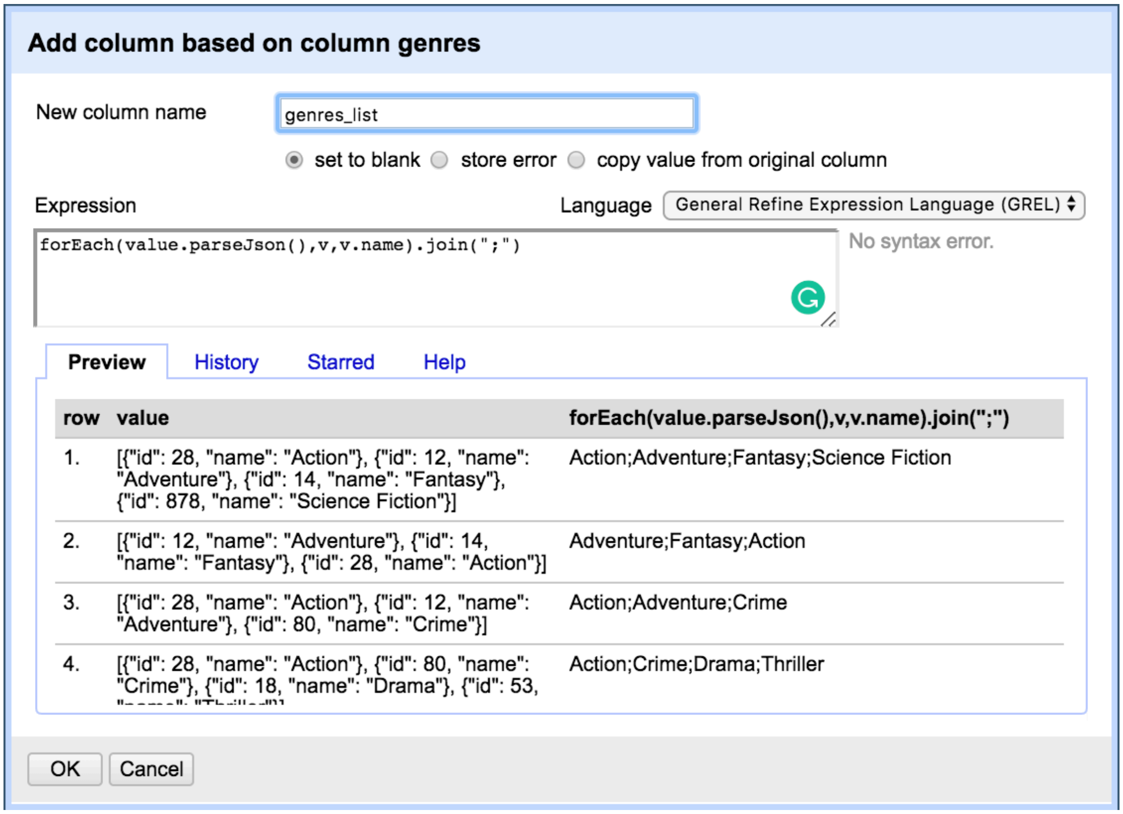
Cluster & Edit column "tagline"

This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strings "New York" and "new york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer to the same person. Find out more ...

Method: key collision | Keying Function: fingerprint | **20** clusters found

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
3	5	Based on a true story. (3 rows) Based on a True Story (1 rows) Based on a True Story. (1 rows)	□	Based on a true story.
2	2	One Hell of A Ride (1 rows) One hell of a ride. (1 rows)	□	One Hell of A Ride
2	2	The hunter becomes the hunted (1 rows) The hunter becomes the hunted. (1 rows)	□	The hunter becomes the hur
2	2	No Soul Is Safe. (1 rows) No soul is safe. (1 rows)	□	No Soul Is Safe.
2	2	Some Men Are Born To Be Heroes. (1 rows) Some Men Are Born to be Heroes (1 rows)	□	Some Men Are Born To Be I
2	2	One person can change your life forever (1 rows) One person can change your life forever. (1 rows)	□	One person can change you
2	2	The world is yours. (1 rows) The world is yours... (1 rows)	□	The world is yours.

Here the reference snapshot of the step where the JSON data of 'Genres' attribute is parsed and used to create a new derived column called 'genres_list'.



Add column based on column genres

New column name: genres_list

● set to blank ○ store error ○ copy value from original column

Expression | Language: General Refine Expression Language (GREL)

```
forEach(value.parseJson(),v,v.name).join(";")
```

No syntax error.

row	value	forEach(value.parseJson(),v,v.name).join(";")
1.	[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]	Action;Adventure;Fantasy;Science Fiction
2.	[{"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 28, "name": "Action"}]	Adventure;Fantasy;Action
3.	[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 80, "name": "Crime"}]	Action;Adventure;Crime
4.	[{"id": 28, "name": "Action"}, {"id": 80, "name": "Crime"}, {"id": 18, "name": "Drama"}, {"id": 53, "name": "Thriller"}]	Action;Crime;Drama;Thriller

2.1.2 Credits data – Issues and corresponding fixes

SL	Attribute name	Data issues	Operations
1	movie_id	NA	NA
2	title	Unwanted characters like “%,@,/,#,!,[,],(,),\?”	All unwanted characters are removed using OpenRefine and regex. Ref - value.replace(/[\%\@\#\!\\\[\]\(\)\?]/, “")
3	cast	Hard to read/consume JSON data	Each movie has long ‘cast’ list with metadata that can’t/shouldn’t be represented in the same datafile. For quick reference, a new column called ‘cast_charactername_actorname’ is created using this formula - forEach(value.parseJson(),v,v.character+"-"+v.name).join(";")
4	crew	Hard to read/consume JSON data	Each movie has long ‘crew’ list with metadata that can’t/shouldn’t be represented in the same datafile. For quick reference, a new column called ‘crew_crewname_job’ is created using this formula - forEach(value.parseJson(),v,v.name+"-"+v.job).join(";")

2.2 Data processing using Pandas

2.2.1 Movie data – Wrong movie release dates issue

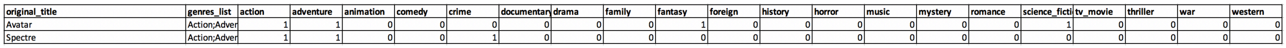
There were some movie-entries with wrong release date (like ‘2031-06-01’ of Pandora’s box).
A subset of those entries where ‘release date’ is greater than the current_date (today – ‘2018-12-01’) have been replaced by correct ‘release date’ fetched using the TMDB’s open-source API.
Ref API -
https://api.themoviedb.org/3/movie/{movie_id}/release_dates?api_key={api_key_givenby_tmdb}
PN – To minimized the number of REST API calls, a subset(14) of wrong release dates are fixed. If required, all 5000 release dates can be checked and updated by fetching correct release dates using API.

2.2.2 Movie data – Multi-value categorical attribute ‘Genre’ issue

Movies ‘Genres’ is an important attribute which can be important while using this data for any analytics/data-viz or machine-learning projects. OpenRefine has helped to parse the JSON data and create a derived column with comma separated genre names. To make this data more consumable for ML projects, dummy variables are created for each genre and this categorical column(genres_list) is represented using binary encoding. As a result, 20 new columns are added are created and added to the ‘Movies’ dataset.

Reference column names -
genre_type_action, genre_type_adventure, genre_type_animation, genre_type_comedy, genre_type_crime, genre_type_documentary, genre_type_drama, genre_type_family, genre_type_fantasy, genre_type_foreign, genre_type_history, genre_type_horror,genre_type_music, genre_type_mystery, genre_type_romance, genre_type_science_fiction, genre_type_tv_movie

Example – Genre representation of two movies (Avatar and Spectre) using encoding.



original_title	genres_list	action	adventure	animation	comedy	crime	documentary	drama	family	fantasy	foreign	history	horror	music	mystery	romance	science_ficti	tv_movie	thriller	war	western
Avatar	Action;Adver	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
Spectre	Action;Adver	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2.2.3 Credits data – JSON data of ‘cast’ and ‘crew’ columns issue

The ‘cast’ and ‘crew’ columns contain complex JSON data with many attributes. Their one-to-many relationships of ‘Movie with its Casts’ and ‘Movie with its Crew’ can’t be properly represented in a single table.

For these two attributes, two new data files(‘Credit_cast’ and ‘Credit_crew’) are created.
These tables represent each movie’s 1:n relationship with its crews and ‘cast’ using ‘movie_id’ as the foreign key.

Here are a few sample rows the two new datasets - Credit_cast and Credit_crew.

- Credit_cast table view

credit_id	department	gender	id	job	movie_id	movie_title	name
52fe48009251416c750aca23	Editing	0	1721	Editor	19995	Avatar	Stephen E. Rivkin
539c47ecc3a36810e3001f87	Art	2	496	Production Design	19995	Avatar	Rick Carter

- Credit_crew table view

cast_id	character	credit_id	gender	id	movie_id	movie_title	name	order
242	Jake Sully	5602a8a7c3a3685532001c9a	2	65731	19995	Avatar	Sam Worthington	0
3	Neytiri	52fe48009251416c750ac9cb	1	8691	19995	Avatar	Zoe Saldana	1

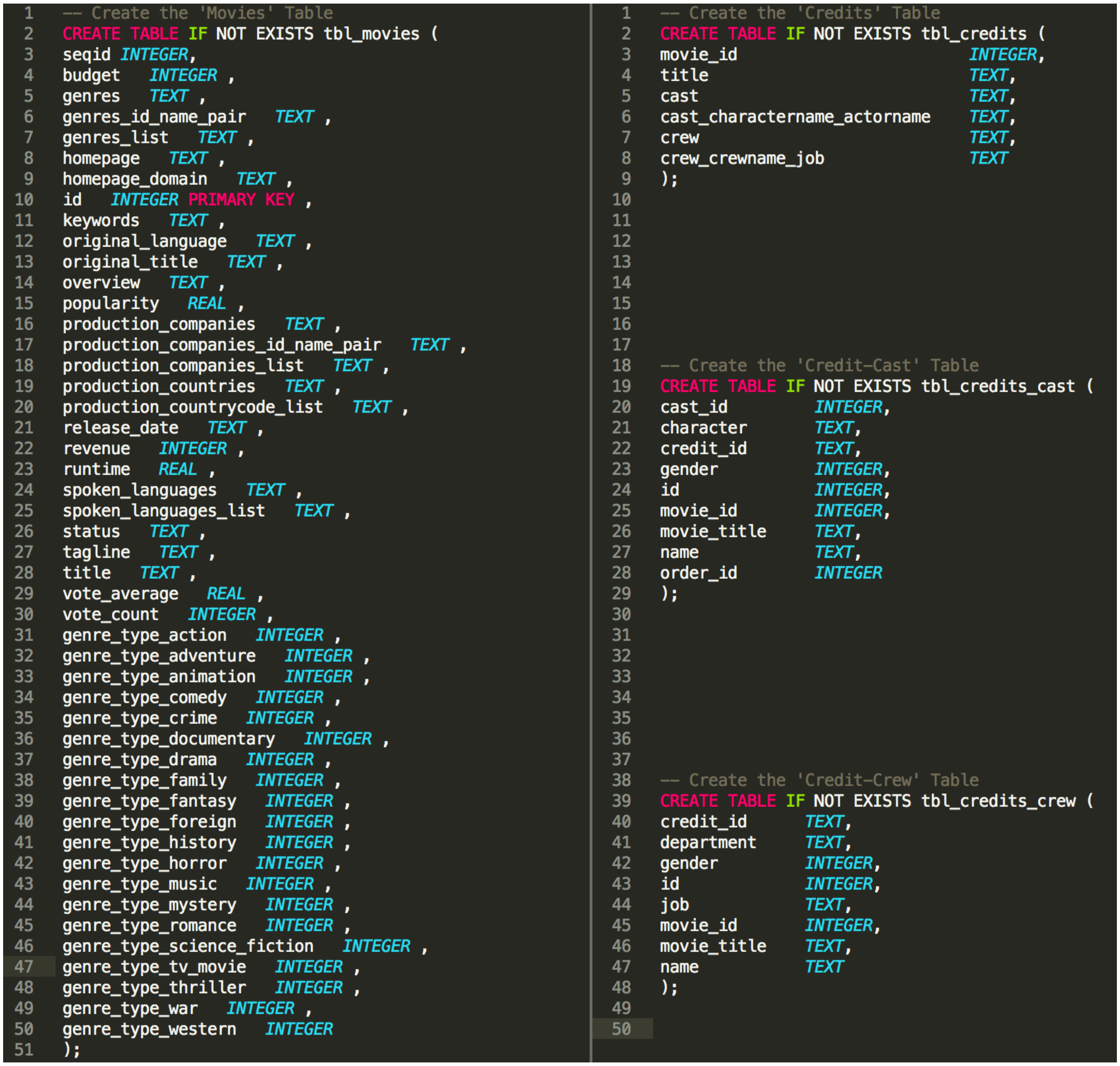
3 Relational database(SQLite) schema

Four cleaned and processed CSV files are loaded to a SQLite database using SQL scripts and SQLite’s ‘.import’ command.

3.1 Database Schema

Four database tables corresponding to the CSV files are created in the project database[Ref - cs513-moviesdb.db]

PN – Initially DB tables were created without any foreign-key constraints. After ensuring(using SQL) that there is no foreign-key violations in the data, foreign-keys were added back to the tables.



```
-- Create the 'Movies' Table
CREATE TABLE IF NOT EXISTS tbl_movies (
seqid INTEGER,
budget   INTEGER ,
genres   TEXT ,
genres_id_name_pair   TEXT ,
genres_list   TEXT ,
homepage   TEXT ,
homepage_domain   TEXT ,
id   INTEGER PRIMARY KEY ,
keywords   TEXT ,
original_language   TEXT ,
original_title   TEXT ,
overview   TEXT ,
popularity   REAL ,
production_companies   TEXT ,
production_companies_id_name_pair   TEXT ,
production_companies_list   TEXT ,
production_countries   TEXT ,
production_countrycode_list   TEXT ,
release_date   TEXT ,
revenue   INTEGER ,
runtime   REAL ,
spoken_languages   TEXT ,
spoken_languages_list   TEXT ,
status   TEXT ,
tagline   TEXT ,
title   TEXT ,
vote_average   REAL ,
vote_count   INTEGER ,
genre_type_action   INTEGER ,
genre_type_adventure   INTEGER ,
genre_type_animation   INTEGER ,
genre_type_comedy   INTEGER ,
genre_type_crime   INTEGER ,
genre_type_documentary   INTEGER ,
genre_type_drama   INTEGER ,
genre_type_family   INTEGER ,
genre_type_fantasy   INTEGER ,
genre_type_foreign   INTEGER ,
genre_type_history   INTEGER ,
genre_type_horror   INTEGER ,
genre_type_music   INTEGER ,
genre_type_mystery   INTEGER ,
genre_type_romance   INTEGER ,
genre_type_science_fiction   INTEGER ,
genre_type_tv_movie   INTEGER ,
genre_type_thriller   INTEGER ,
genre_type_war   INTEGER ,
genre_type_western   INTEGER
);
```

```
-- Create the 'Credits' Table
CREATE TABLE IF NOT EXISTS tbl_credits (
movie_id                        INTEGER,
title                           TEXT,
cast                            TEXT,
cast_charactername_actorname    TEXT,
crew                            TEXT,
crew_crewname_job               TEXT
);

-- Create the 'Credit-Cast' Table
CREATE TABLE IF NOT EXISTS tbl_credits_cast (
cast_id         INTEGER,
character       TEXT,
credit_id       TEXT,
gender          INTEGER,
id              INTEGER,
movie_id        INTEGER,
movie_title     TEXT,
name            TEXT,
order_id        INTEGER
);

-- Create the 'Credit-Crew' Table
CREATE TABLE IF NOT EXISTS tbl_credits_crew (
credit_id      TEXT,
department     TEXT,
gender         INTEGER,
id             INTEGER,
job            TEXT,
movie_id       INTEGER,
movie_title    TEXT,
name           TEXT
);
```

3.2 Data integrity and constraints

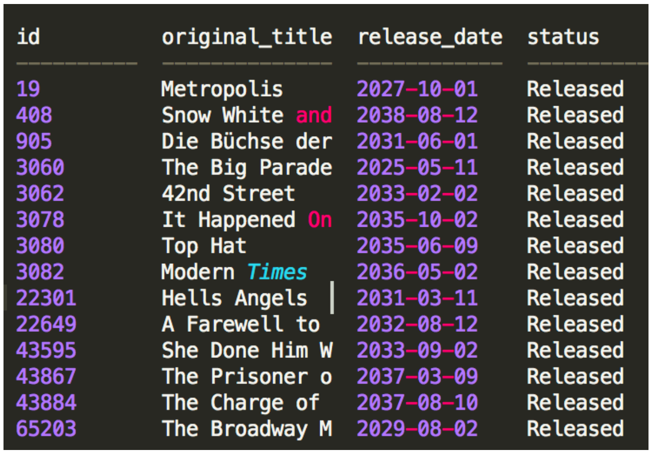
Following data and referential integrity constraints are checked using various SQL queries.

1. All data files are successfully loaded to the corresponding data tables.
   Ref Queries -
   select count(*) from tbl_movies; --Output : 4803 records
   select count(*) from tbl_credits; -- Output : 4803 records
   select count(*) from tbl_credits_cast; --Output : 106257
   select count(*) from tbl_credits_crew; --Output : 129581

   These row-counts are matching with corresponding row-count of the csv files. It suggests that all files entries are loaded successfully.

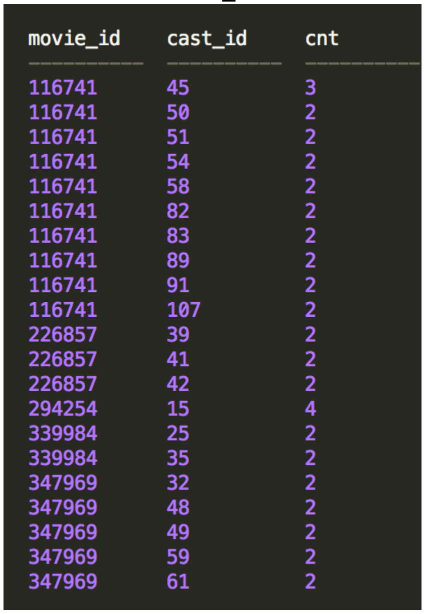
2. Check all movie_ids are unique in the movie table(tbl_movies)
   Query - select id, count(id) as cnt from tbl_movies group by id having cnt>1;
   This query doesn’t return any output. It proves that all movie_ids are unique in the movie table.

3. Check if release data is greater than today's date.
   Query - select id, original_title, release_date,status from tbl_movies where release_date > '2018-11-25';
   Output: 0 records found
   PN – Correct relese_dates of 14 movies were fetched using TMDB API and updated in the main dataset.



```
id          original_title  release_date  status
----------  --------------  ------------  ----------
19          Metropolis      2027-10-01    Released
408         Snow White and  2038-08-12    Released
905         Die Büchse der  2031-06-01    Released
3060        The Big Parade  2025-05-11    Released
3062        42nd Street     2033-02-02    Released
3078        It Happened On  2035-10-02    Released
3080        Top Hat         2035-06-09    Released
3082        Modern Times    2036-05-02    Released
22301       Hells Angels    2031-03-11    Released
22649       A Farewell to   2032-08-12    Released
43595       She Done Him W  2033-09-02    Released
43867       The Prisoner o  2037-03-09    Released
43884       The Charge of   2037-08-10    Released
65203       The Broadway M  2029-08-02    Released
```

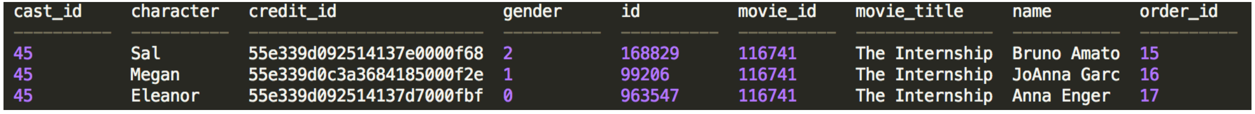
4. Check if same cast_id is used for differnt casts for any movie (ref table tbl_credits_cast)
   select movie_id, credit_id, count(*) as cnt from tbl_credits_cast group by movie_id, credit _id having cnt>1;
   Query - select movie_id, cast_id, count(*) as cnt from tbl_credits_cast group by movie_id, cast_id having cnt>1;
   Output – 20 records found
   It indicates that cast_id values are reused in a few movies.



```
movie_id    cast_id     cnt
----------  ----------  ----------
116741      45          3
116741      50          2
116741      51          2
116741      54          2
116741      58          2
116741      82          2
116741      83          2
116741      89          2
116741      91          2
116741      107         2
226857      39          2
226857      41          2
226857      42          2
294254      15          4
339984      25          2
339984      35          2
347969      32          2
347969      48          2
347969      49          2
347969      59          2
347969      61          2
```

Using the following query, we see that same ‘cast_id’ to represent 3 different casts corresponding to 3 different actors for the movie ‘The Internship’.

Query - select * from tbl_credits_cast where movie_id=116741 and cast_id=45;



```
cast_id     character   credit_id                 gender      id          movie_id    movie_title      name         order_id
----------  ----------  ------------------------  ----------  ----------  ----------  ---------------  -----------  ----------
45          Sal         55e339d092514137e0000f68  2           168829      116741      The Internship   Bruno Amato  15
45          Megan       55e339d0c3a3684185000f2e  1           99206       116741      The Internship   JoAnna Garc  16
45          Eleanor     55e339d092514137d7000fbf  0           963547      116741      The Internship   Anna Enger   17
```

It suggests that ‘cast_id’ may not be considered as a reliable field for many use-cases. The following thread reinforces this assumption. Ref - https://www.themoviedb.org/talk/537250c1c3a368434300134e

5. Check if same ‘credit_id’ is used multiple times for any given movie.
   Query:
   select movie_id, credit_id, count(*) as cnt from tbl_credits_cast group by movie_id, credit_id having cnt>1;
   Output : 0 records.
   It suggests that no ‘credit_id’ is reused for any movie. This field should be used instead of ‘cast_id’.

6. Check referential constraints between movie and credit tables.
   Queries:
   select movie_id, title from tbl_credits where movie_id not in (select id from tbl_movies);
   select id from tbl_movies where id not in (select movie_id from tbl_credits);

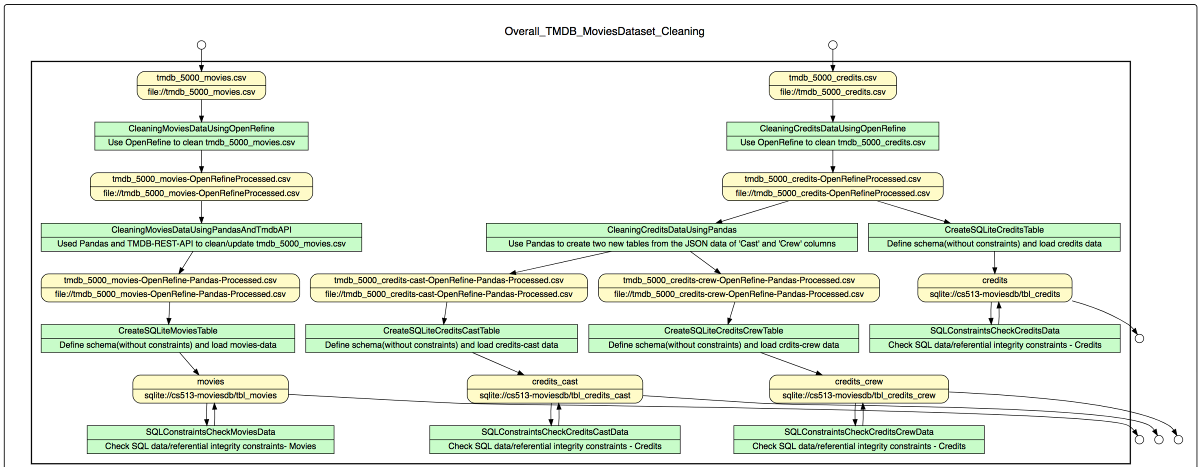
   Since it didn’t return any result, it suggests 1:1 relation between these two tables. Hence, the ‘movie_id’ can be used as primarykey in the ‘tbl_movie’ and primarkey/foreignkey in the ‘tbl_credit’ table.

   While creating the other two tables – ‘credits_cast’ and ‘credits_crew’ from the ‘credit’ table, movie_id column was added programmatically. This process explicitly created the one-to-many relationships between the movie with its casts and crews. No further referential integrity checks are made for these two tables.

4 Workflow Model

4.1 Overall workflow Model (Workflow1)

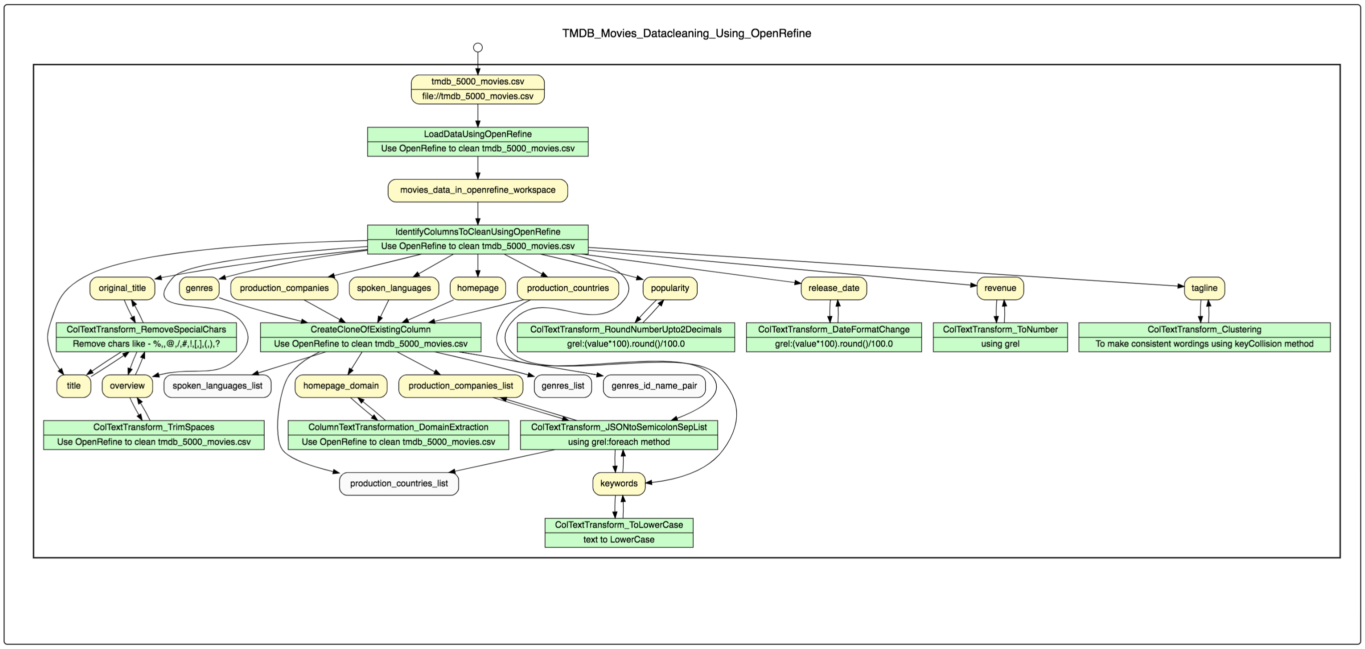
This workflow diagram shows how two input CSV files are processed using OpenRefine, Pandas and SQLite database.


Overall_TMDB_MoviesDataset_Cleaning
tmdb_5000_movies.csv
file://tmdb_5000_movies.csv
CleaningMoviesDataUsingOpenRefine
Use OpenRefine to clean tmdb_5000_movies.csv
tmdb_5000_movies-OpenRefineProcessed.csv
file://tmdb_5000_movies-OpenRefineProcessed.csv
CleaningMoviesDataUsingPandasAndTmdbAPI
Used Pandas and TMDB-REST-API to clean/update tmdb_5000_movies.csv
tmdb_5000_movies-OpenRefine-Pandas-Processed.csv
file://tmdb_5000_movies-OpenRefine-Pandas-Processed.csv
CreateSQLiteMoviesTable
Define schema(without constraints) and load movies-data
movies
sqlite://cs513-moviesdb/tbl_movies
SQLConstraintsCheckMoviesData
Check SQL data/referential integrity constraints- Movies
tmdb_5000_credits.csv
file://tmdb_5000_credits.csv
CleaningCreditsDataUsingOpenRefine
Use OpenRefine to clean tmdb_5000_credits.csv
tmdb_5000_credits-OpenRefineProcessed.csv
file://tmdb_5000_credits-OpenRefineProcessed.csv
CleaningCreditsDataUsingPandas
Use Pandas to create two new tables from the JSON data of 'Cast' and 'Crew' columns
CreateSQLiteCreditsTable
Define schema(without constraints) and load credits data
tmdb_5000_credits-cast-OpenRefine-Pandas-Processed.csv
file://tmdb_5000_credits-cast-OpenRefine-Pandas-Processed.csv
tmdb_5000_credits-crew-OpenRefine-Pandas-Processed.csv
file://tmdb_5000_credits-crew-OpenRefine-Pandas-Processed.csv
credits
sqlite://cs513-moviesdb/tbl_credits
CreateSQLiteCreditsCastTable
Define schema(without constraints) and load credits-cast data
CreateSQLiteCreditsCrewTable
Define schema(without constraints) and load crdits-crew data
SQLConstraintsCheckCreditsData
Check SQL data/referential integrity constraints - Credits
credits_cast
sqlite://cs513-moviesdb/tbl_credits_cast
credits_crew
sqlite://cs513-moviesdb/tbl_credits_crew
SQLConstraintsCheckCreditsCastData
Check SQL data/referential integrity constraints - Credits
SQLConstraintsCheckCreditsCrewData
Check SQL data/referential integrity constraints - Credits

4.2 OpenRefine workflow Model

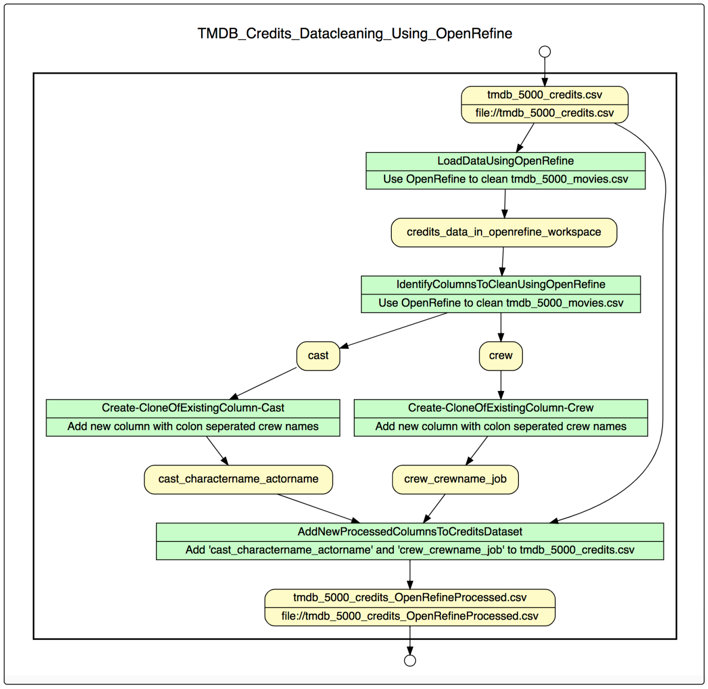
4.2.1 OpenRefine workflow on the 'Movies' dataset (Workflow2)

It covers what processing are done using OpenRefine on the individual attributes of the 'Movies' dataset.


TMDB_Movies_Datacleaning_Using_OpenRefine
tmdb_5000_movies.csv
file://tmdb_5000_movies.csv
LoadDataUsingOpenRefine
Use OpenRefine to clean tmdb_5000_movies.csv
movies_data_in_openrefine_workspace
IdentifyColumnsToCleanUsingOpenRefine
Use OpenRefine to clean tmdb_5000_movies.csv
original_title
genres
production_companies
spoken_languages
homepage
production_countries
popularity
release_date
revenue
tagline
ColTextTransform_RemoveSpecialChars
Remove chars like - %,,@,/,#,!,[,],(,),?
CreateCloneOfExistingColumn
Use OpenRefine to clean tmdb_5000_movies.csv
ColTextTransform_RoundNumberUpto2Decimals
grel:(value*100).round()/100.0
ColTextTransform_DateFormatChange
grel:(value*100).round()/100.0
ColTextTransform_ToNumber
using grel
ColTextTransform_Clustering
To make consistent wordings using keyCollision method
title
overview
spoken_languages_list
homepage_domain
production_companies_list
genres_list
genres_id_name_pair
ColTextTransform_TrimSpaces
Use OpenRefine to clean tmdb_5000_movies.csv
ColumnTextTransformation_DomainExtraction
Use OpenRefine to clean tmdb_5000_movies.csv
ColTextTransform_JSONtoSemicolonSepList
using grel:foreach method
production_countries_list
keywords
ColTextTransform_ToLowerCase
text to LowerCase

4.2.2 OpenRefine workflow on the ‘Credits’ dataset (Workflow3)

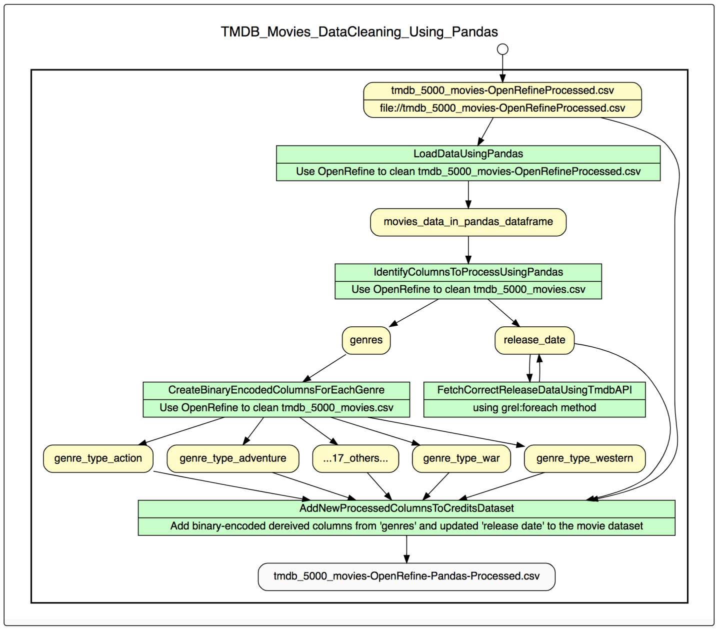
The workflow shows how OpenRefine is used to process the ‘Credits’ dataset.


TMDB_Credits_Datacleaning_Using_OpenRefine
tmdb_5000_credits.csv
file://tmdb_5000_credits.csv
LoadDataUsingOpenRefine
Use OpenRefine to clean tmdb_5000_movies.csv
credits_data_in_openrefine_workspace
IdentifyColumnsToCleanUsingOpenRefine
Use OpenRefine to clean tmdb_5000_movies.csv
cast
crew
Create-CloneOfExistingColumn-Cast
Add new column with colon seperated crew names
Create-CloneOfExistingColumn-Crew
Add new column with colon seperated crew names
cast_charactername_actorname
crew_crewname_job
AddNewProcessedColumnsToCreditsDataset
Add 'cast_charactername_actorname' and 'crew_crewname_job' to tmdb_5000_credits.csv
tmdb_5000_credits_OpenRefineProcessed.csv
file://tmdb_5000_credits_OpenRefineProcessed.csv

4.3 Pandas workflow Model

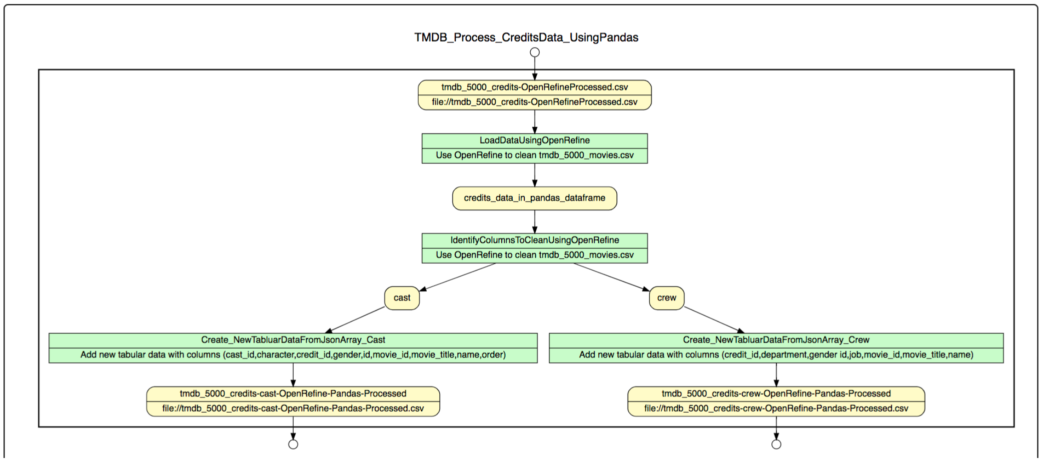
4.3.1 Pandas workflow on the ‘Movies’ dataset (Workflow4)

It covers what processing(binary encoding of ‘genre’ and imputing wrong release dates) are done using Pandas on the ‘Movies’ dataset.


TMDB_Movies_DataCleaning_Using_Pandas
tmdb_5000_movies-OpenRefineProcessed.csv
file://tmdb_5000_movies-OpenRefineProcessed.csv
LoadDataUsingPandas
Use OpenRefine to clean tmdb_5000_movies-OpenRefineProcessed.csv
movies_data_in_pandas_dataframe
IdentifyColumnsToProcessUsingPandas
Use OpenRefine to clean tmdb_5000_movies.csv
genres
release_date
CreateBinaryEncodedColumnsForEachGenre
Use OpenRefine to clean tmdb_5000_movies.csv
FetchCorrectReleaseDataUsingTmdbAPI
using grel:foreach method
genre_type_action
genre_type_adventure
...17_others...
genre_type_war
genre_type_western
AddNewProcessedColumnsToCreditsDataset
Add binary-encoded dereived columns from 'genres' and updated 'release date' to the movie dataset
tmdb_5000_movies-OpenRefine-Pandas-Processed.csv

4.3.2 Pandas workflow on the ‘Credits’ dataset (Workflow5)

The workflow shows how two new tables were created from the ‘cast’ and ‘crew’ data of the ‘Credits’ dataset.


TMDB_Process_CreditsData_UsingPandas
tmdb_5000_credits-OpenRefineProcessed.csv
file://tmdb_5000_credits-OpenRefineProcessed.csv
LoadDataUsingOpenRefine
Use OpenRefine to clean tmdb_5000_movies.csv
credits_data_in_pandas_dataframe
IdentifyColumnsToCleanUsingOpenRefine
Use OpenRefine to clean tmdb_5000_movies.csv
cast
crew
Create_NewTabluarDataFromJsonArray_Cast
Add new tabular data with columns (cast_id,character,credit_id,gender,id,movie_id,movie_title,name,order)
Create_NewTabluarDataFromJsonArray_Crew
Add new tabular data with columns (credit_id,department,gender id,job,movie_id,movie_title,name)
tmdb_5000_credits-cast-OpenRefine-Pandas-Processed
file://tmdb_5000_credits-cast-OpenRefine-Pandas-Processed.csv
tmdb_5000_credits-crew-OpenRefine-Pandas-Processed
file://tmdb_5000_credits-crew-OpenRefine-Pandas-Processed.csv

5 Conclusion

There are a few attributes like ‘revenue’, ‘budget’ etc. where further data cleaning and imputation are required. Depending on the use-cases, these data can be sourced and leveraged using proprietary movie-metadata related APIs.

This data can now be used in different types of ML and advanced data-visualization projects. After processing the datasets, the quality and readability of the dataset have been improved significantly. Example - the ‘cast’ and ‘crew’ data of the ‘credits data’ are now available in separate tables/CSVs which are much more clean and readable than its original JSON form.

TBD – We have plan to share these processed data files via Kaggle so that many others researchers and students can use this data for their projects.

6 Attribution

- In this project, ‘themoviedb’ API is used to source correct movie-release-dates of 14 movies.
  We are thankful to ‘themoviedb’ for letting us use their great APIs.
  Ref - https://www.themoviedb.org/about/logos-attribution
  POWERED BY
  THE MOVIE DB
- Data source - https://www.kaggle.com/tmdb/tmdb-movie-metadata/home

Appendix 1 – Python code used encode the categorical values of the ‘genre’ attribute.

```
# Binary encoded column values for the categorical attributes are common practices in ML
projects
genres_list_with_dummies = data_movies['genres_list'].str.get_dummies(sep=';')
genres_list_with_dummies.head(15)

# For each genre value of the 'Genre' column, create a binary encoded column with prefix
'genre_type_'
genres_list_with_dummies.columns = /
["genre_type_"+col_name.lower().replace(' ','_') for col_name in
genres_list_with_dummies.columns]

# Add the new binary-encoded columns(generated from 'genre' column) are concatenated with the
made movie dataset
data_movies_with_genres_dummies = pd.concat([data_movies, genres_list_with_dummies], axis=1);
```

Appendix 2 – Python code used get correct ‘release dates’ using the TMDB REST APIs.

```
import http.client
import json
from pprint import pprint
from IPython.display import display

conn = http.client.HTTPSConnection("api.themoviedb.org")

def get_correct_releasedate_for_movie(movie_id):
    conn.request("GET", "/3/movie/" + movie_id + "/release_dates?api_key={*****}", "{}")
    res = conn.getresponse()
    data = res.read()
    str_data = data.decode("UTF-8")
    json_data = json.loads(str_data)
    #pprint(json_data)

    results = json_data['results']
    for res in results:
        if res['iso_3166_1'] == 'US':
            return res['release_dates'][0]['release_date'][:10]

    return 'NA'
```

Appendix 3 – Project Repo

https://github.com/manas-mukherjee/MCSDS-CS513