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 3](#_Toc531651079)

[1.1 Dataset 3](#_Toc531651080)

[1.1.1 Source 3](#_Toc531651081)

[1.1.2 Structure and content 3](#_Toc531651082)

[1.1.3 Data quality 3](#_Toc531651083)

[1.2 Data cleaning goals 4](#_Toc531651084)

[1.3 Use-cases and usability 4](#_Toc531651085)

[1.3.1 Use-cases where this data can be used in its current form 4](#_Toc531651086)

[1.3.2 Target use-cases 4](#_Toc531651087)

[2 Data cleaning with OpenRefine and Pandas 4](#_Toc531651088)

[2.1 Data processing using OpenRefine 4](#_Toc531651089)

[2.1.1 Movies data – Issues and corresponding fixes 4](#_Toc531651090)

[2.1.2 Credits data – Issues and corresponding fixes 7](#_Toc531651091)

[2.2 Data processing using Pandas 7](#_Toc531651092)

[2.2.1 Movie data – Wrong movie release dates issue 7](#_Toc531651093)

[2.2.2 Movie data – Multi-value categorical attribute ‘Genre’ issue 7](#_Toc531651094)

[2.2.3 Credits data – JSON data of ‘cast’ and ‘crew’ columns issue 8](#_Toc531651095)

[3 Relational database(SQLite) schema 8](#_Toc531651096)

[3.1 Database Schema 8](#_Toc531651097)

[3.2 Data integrity and constraints 9](#_Toc531651098)

[4 Workflow Model 12](#_Toc531651099)

[4.1 Overall workflow Model (Workflow1) 12](#_Toc531651100)

[4.2 OpenRefine workflow Model 12](#_Toc531651101)

[4.2.1 OpenRefine workflow on the ‘Movies’ dataset (Workflow2) 12](#_Toc531651102)

[4.2.2 OpenRefine workflow on the ‘Credits’ dataset (Workflow3) 13](#_Toc531651103)

[4.3 Pandas workflow Model 13](#_Toc531651104)

[4.3.1 Pandas workflow on the ‘Movies’ dataset (Workflow4) 13](#_Toc531651105)

[4.3.2 Pandas workflow on the ‘Credits’ dataset (Workflow5) 14](#_Toc531651106)

[5 Conclusion 14](#_Toc531651107)

[6 Attribution 14](#_Toc531651108)

[Appendix 1 15](#_Toc531651109)

[Appendix 2 15](#_Toc531651110)

[Appendix 3 15](#_Toc531651111)

# Overview

## Dataset

### 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>

### 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.

**Credits**

|  |  |  |
| --- | --- | --- |
| SL | Attribute name | Type |
| 1 | movie\_id | Numeric |
| 2 | title | Text |
| 3 | cast | Text |
| 4 | crew | Text |

**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 |

### 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.

## 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.

## Use-cases and usability

### 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.

### 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.

# 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.

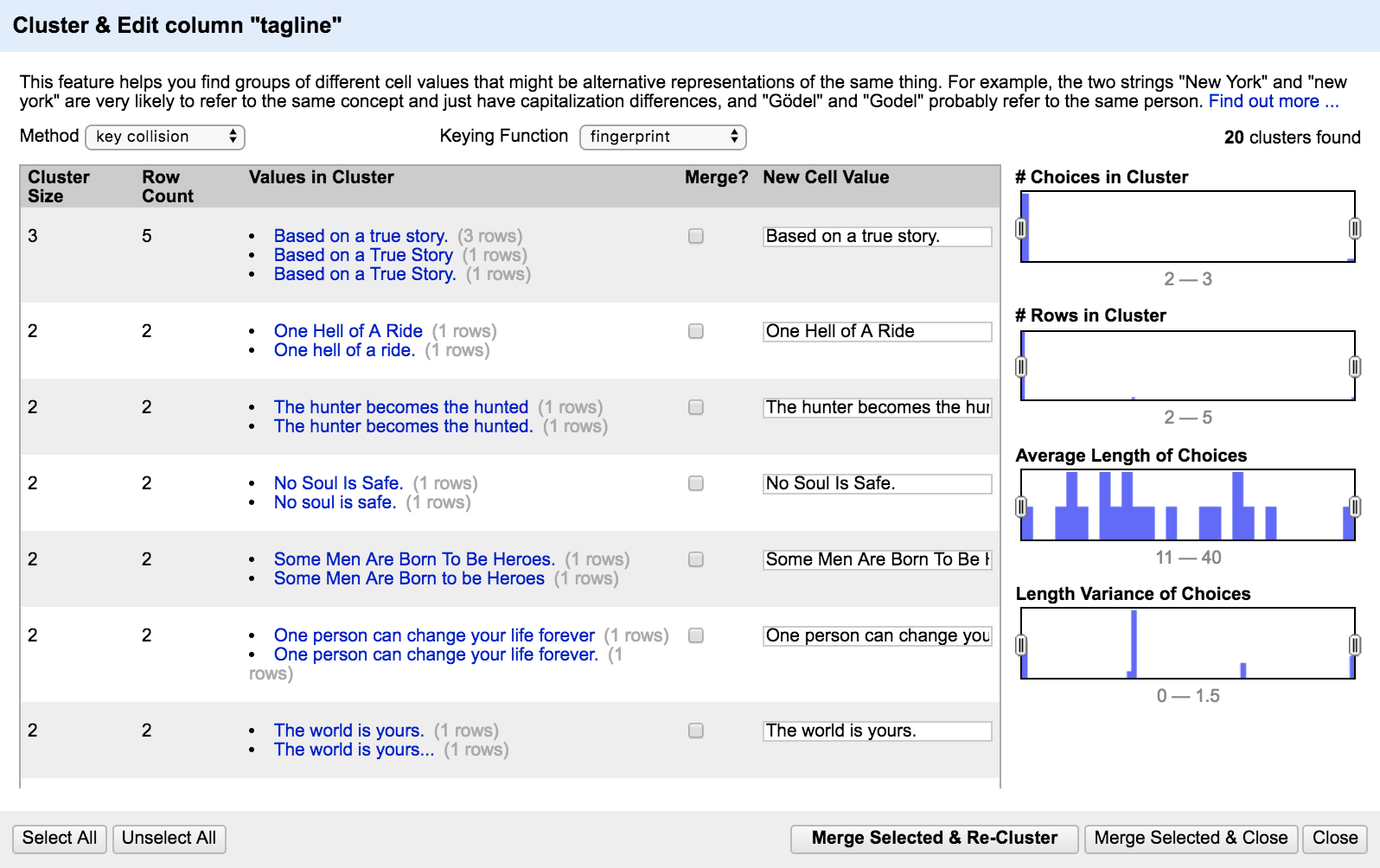
Here is the detailed description of the operations made on the two data files.

## Data processing using OpenRefine

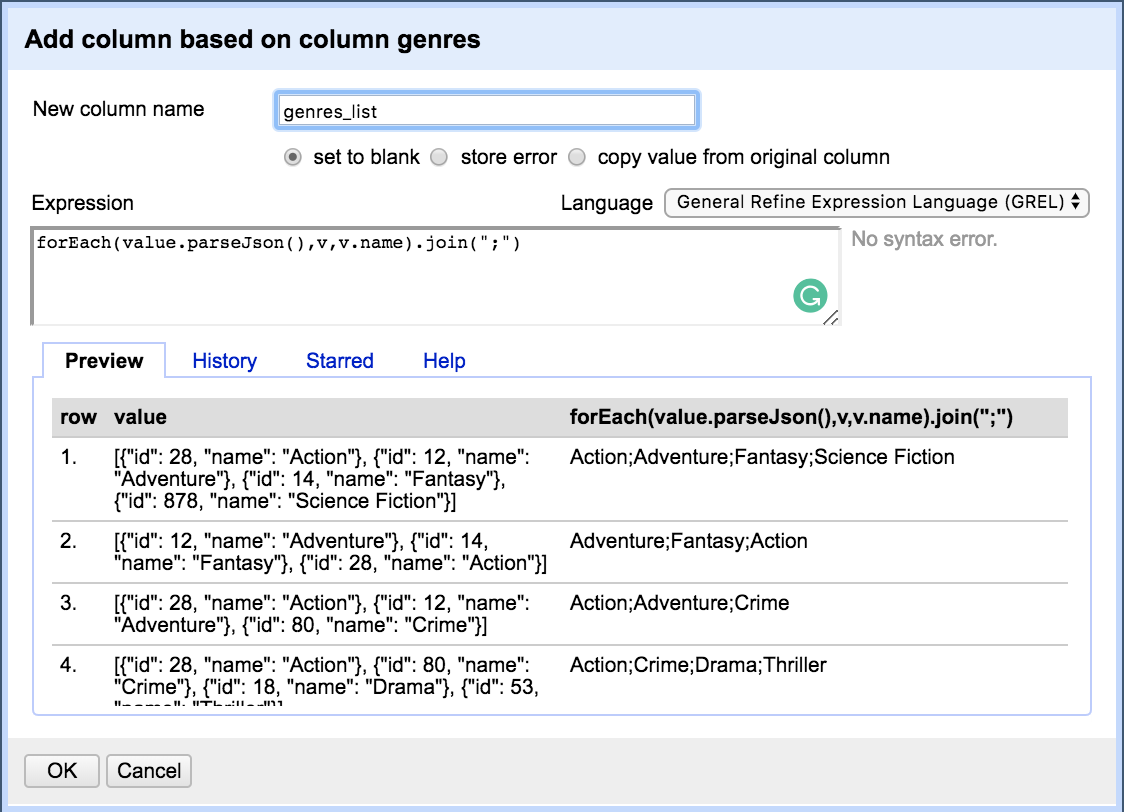
### Movies data – Issues and corresponding fixes

|  |  |  |  |
| --- | --- | --- | --- |
| 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.



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’.



### 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(";") |

## Data processing using Pandas

### 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}](https://api.themoviedb.org/3/movie/%7bmovie_id%7d/release_dates?api_key=%7bapi_key_givenby_tmdb%7d)

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.

### 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.

A close up of a device

Description automatically generated

### 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 |

# Relational database(SQLite) schema

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

## 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.

A screenshot of a cell phone

Description automatically generated

## 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.

1. 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.

1. 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

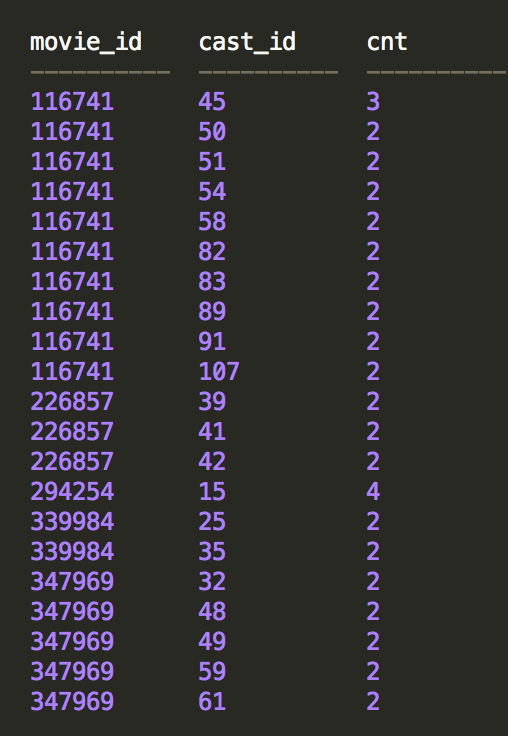
PN – Correct relese\_dates of 14 movies were fetched using TMDB API and updated in the main dataset.   
  
A screen shot of a computer

Description automatically generated

1. 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.



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;

A screenshot of a cell phone

Description automatically generated

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>

1. 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’.

1. 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);

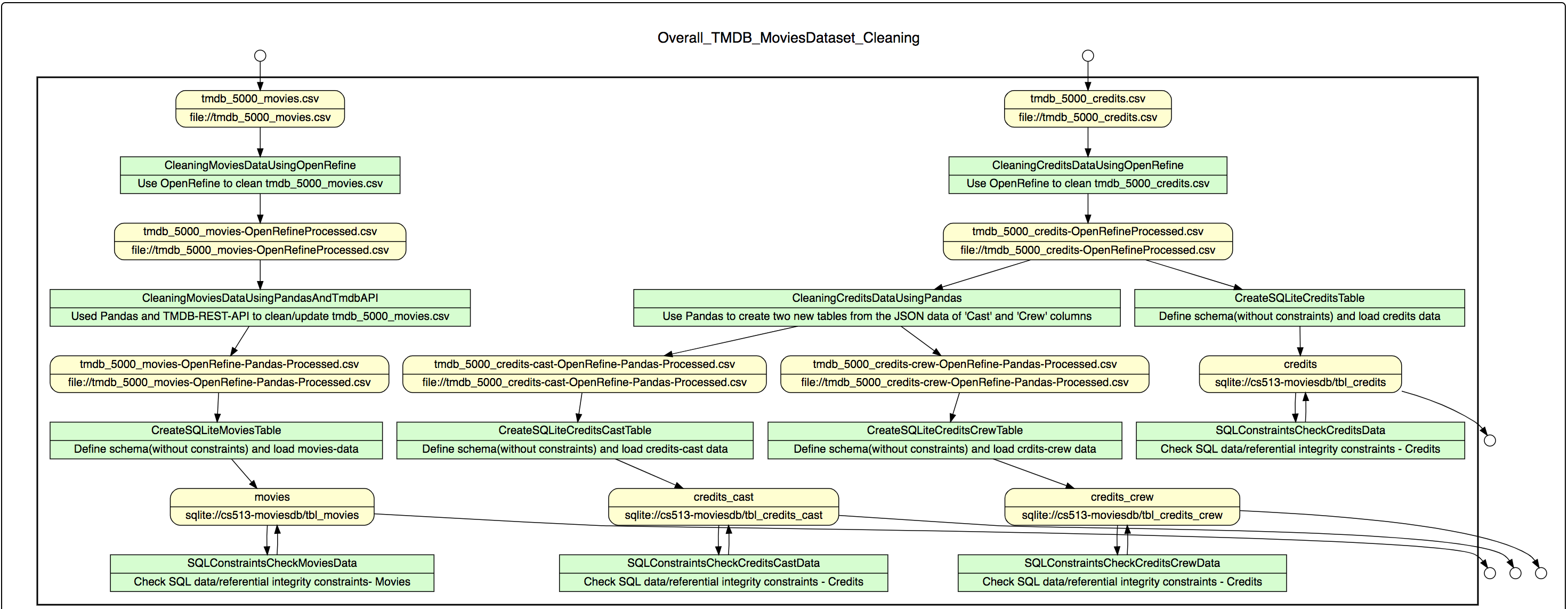
Since it didn’t return any result, it suggests 1:1 relation between these two tables. Hence, the ‘movie\_id’ can be used as primarykey in the ‘tbl\_movie’ and primarkey/foreignkey in the ‘tbl\_credit’ table.

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.

# Workflow Model

## Overall workflow Model (Workflow1)

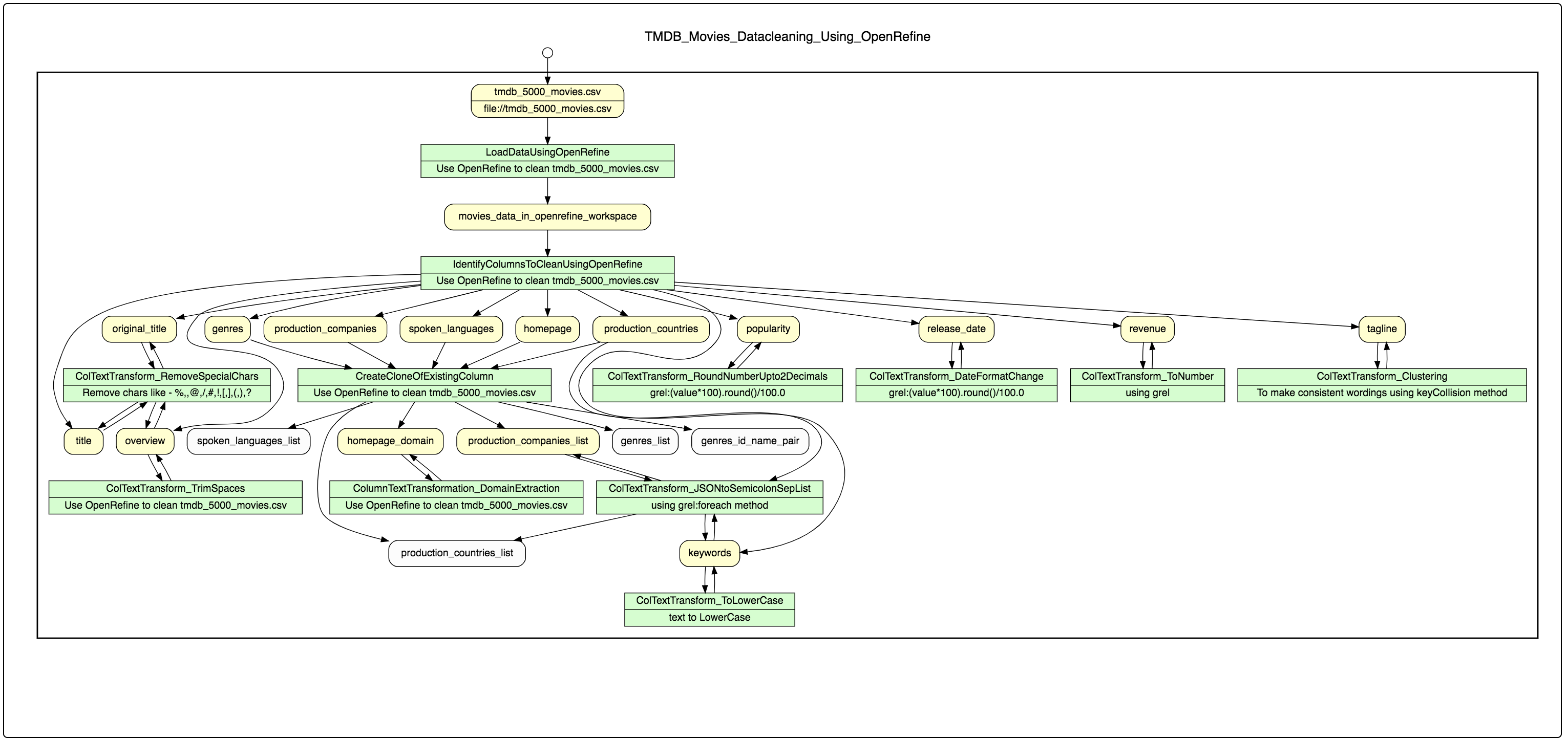
This workflow diagram shows how two input CSV files are processed using OpenRefine, Pandas and SQLite database.



## OpenRefine workflow Model

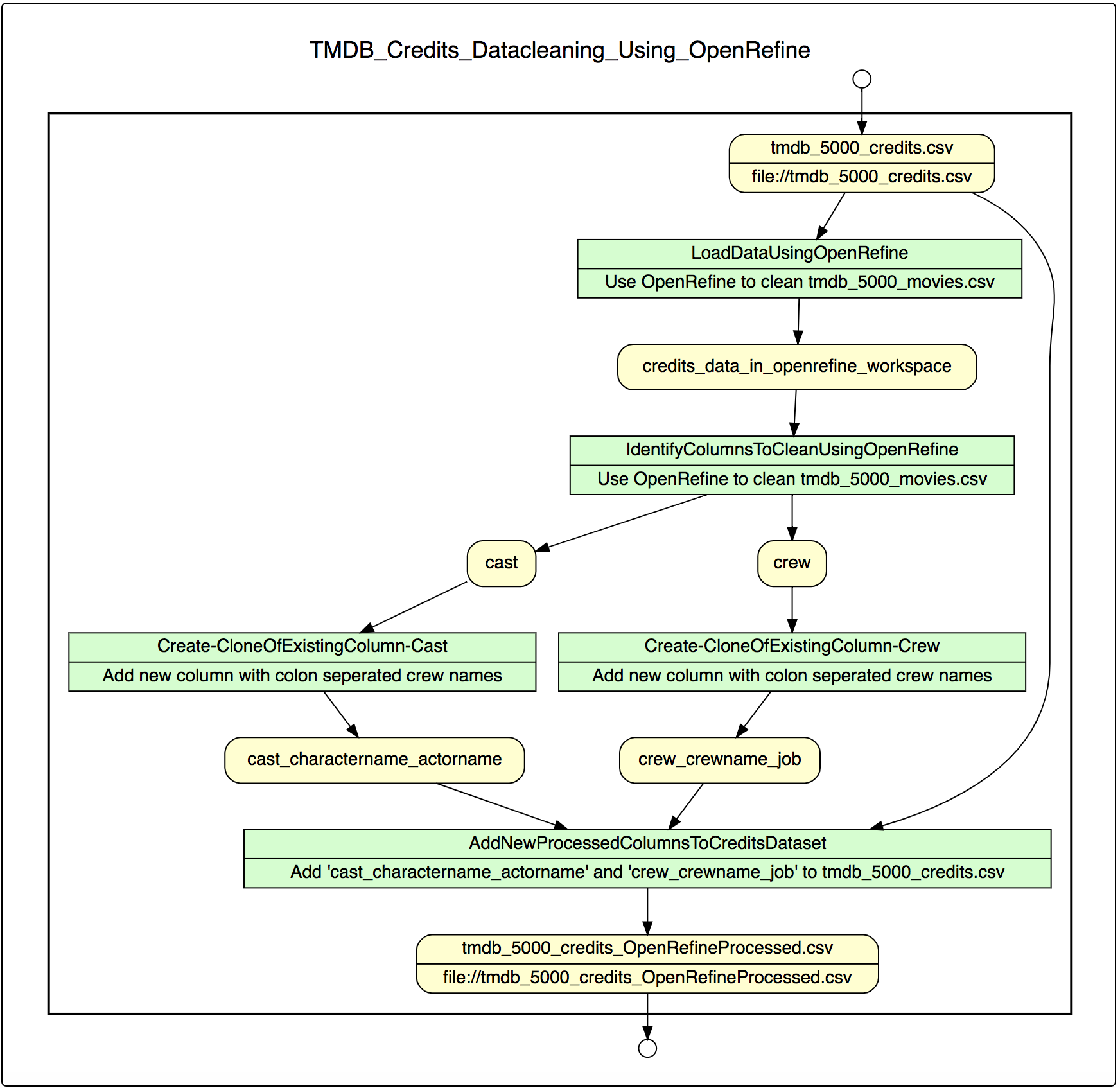
### OpenRefine workflow on the ‘Movies’ dataset (Workflow2)

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



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

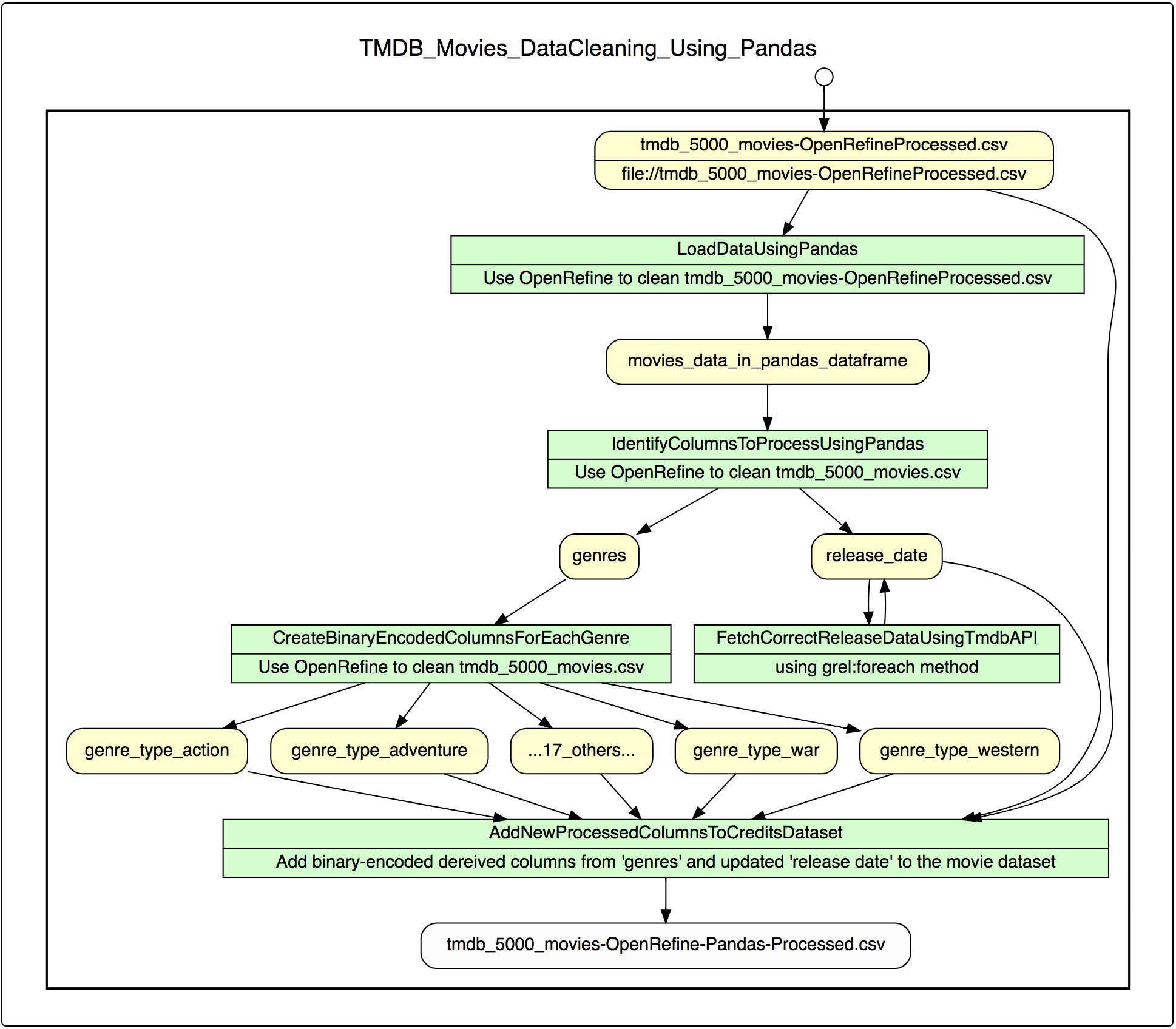
The workflow shows how OpenRefine is used to process the ‘Credits’ dataset.



## Pandas workflow Model

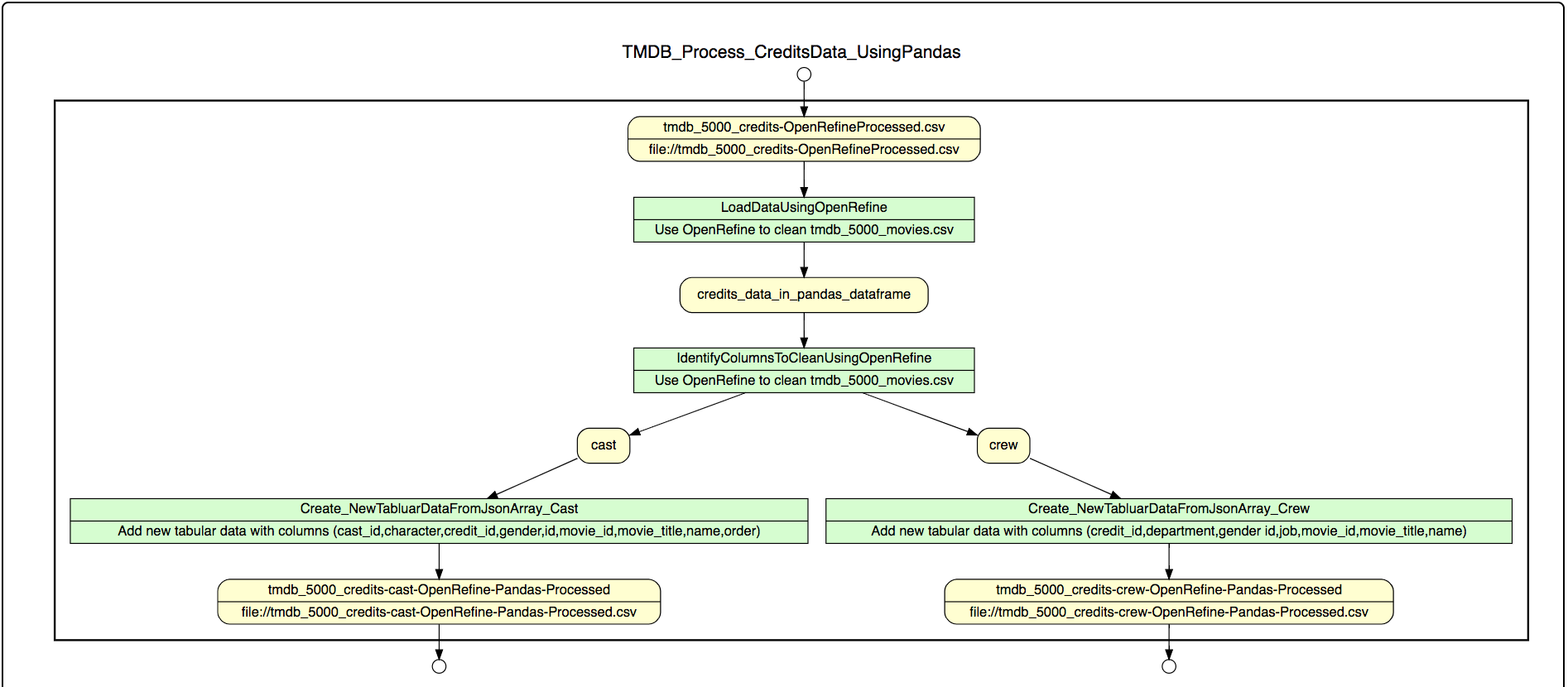
### Pandas workflow on the ‘Movies’ dataset (Workflow4)

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



### 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.



# 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.

# 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>



* 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/tree/master/CourseProject>