**5310 SQL & Relational Database**

**Group Project IMDb**

Team Members: Jiaman Li, Jie Ming, Kai Hu, Tsai-Hsuan Cheng,

Wenpei Sun, Xiaotong Sui, Zhichen Xia

Columbia University

School of Professional Studies

Instructor: Michael Coakley

November 29, 2022

**Table of Contents**

Problem Statement …………………………………………………………………………….. P.3

Proposal ………………………………………………………………………………………... P.3

Normalization Plan………………………………………………………………………………P.5

ETL Process………………………………………………………………………………..……P.7

Team Structure and Timeline ...………………………………………………………………... P.8

Database Schema …………………………………………………………………………….... P.9

Interactions With the Database System ………………………………………………………. P.12

Analytics Applications ……………………………………………………………………….. P.14

Conclusion …………………………………………………………………………………… P.18

Appendix I (ETL Code)……………………………………………..………………………... P.19

Appendix II (Insight Code)………………….………………………………………………... P.32

**Problem Statement**

Zebra is a new rising movie and television production company that started after the Covid-19 pandemic. While seeking the idea for their next work, they noticed the phenomenon that trends powerfully rule the film industry. While a specific trend is popular among society, films and television series that follow it usually have satisfactory performance at their box office and viewership ratings. Therefore, they decided to apply the strategy of following the trend for their next work to maximize the opportunity for their next work to succeed. To better predict the trends and produce movies and television works accordingly to attract the public, the filmmakers and industry executives at Zebra decided to analyze the data of previous films and television works. By getting insights from the factors associated with the success and failures of previous films, filmmakers and industry executives believed that they could capture future market trends and recruit best-fit film crews accordingly, such as writers, directors, and actors, to launch the next most popular film and television work. If the plan is administered perfectly, this would be the key that boosts their revenue and leads to the company’s business success. However, lacking professional analysts and data of past films and television works, Zebra had no idea where or how to start their plan. Therefore, to achieve their goal, Zebra hired us, a professional analysis group, to help them solve the problem.

**Proposal**

Since our client was a new rising movie and television production company that lacked their private data of past films and television works, we would develop our project with the help of open data from IMDb. In detail, IMDb is a website that provides users with comprehensive information about movies and television shows, such as ratings, rankings, release calendar, and so on. It provides subsets of its own IMDb data as open-source online data that includes all the up-to-date information on trend movies, famous writers, directors, actors, and their related information, such as personal biographies, plot summaries, ratings, fans, and critical reviews.

Our project could be briefly divided into four main parts. First, we would do the data preparation to reorganize the datasets to make them suitable for analyzing and importing into the database system. Originally, each dataset was contained in a gzipped, tab-separated-values (TSV) formatted file in the UTF-8 character set. There were seven files in total, including “name.basics,” “title.akas,” “title.basics,” “title.crews,” “title.episode,” “title.principals,” and “title.rating.” The “name.basics” file contained 12,019,991 rows and 6 columns. The “title.akas” file contained 33,574,168 rows and 8 columns. The “title.basics” file contained 9,313,429 rows and 9 columns. The “title.crews” file contained 9,308,649 rows and 3 columns. The “title.episode” file contained 7,026,974 rows and 4 columns. The “title.principals” file contained 52,644,470 rows and 6 columns. The “title.rating” file contained 1,239,291 rows and 3 columns. Each file's first line included headers describing what is in each column. A *'\N'* denoted that a particular field was missing or null for that title/name. The data preparation process included a lot of steps, such as dealing with the *'\N'* values, separating the values from cells that contain multiple values, splitting and creating tables to ensure that the data was following the structure of 3NF, and so on. More details will be described in the following sections of our normalization plan and ETL process.

Second, we would visualize the data to show some general facts about past trends in the industry. Here we would use Tableau as our tool for the visualization. The most important part here was that, considering how to maximize our client's profit, we would take not only the box office gross but also the production budget into consideration. In other words, we would determine the top choice of movie genre for our client by the ratio of box office gross to budget rather than just the box office gross itself.

Then, in the third part, we would import the organized data into the database system. Here we would use pgAdmin as our tool. Then, with the help of the work we had built until now, we could easily get a list of names of directors or actors who are suitable for the target movie genre that we just determined in the previous part, with just several lines of easy queries in pgAdmin.

Last but not least, in the fourth part, we would provide our customers a complete list of the ideal crew for the target movie genre. Here we would use Metabase as our tool to show the visualized result, with bar charts showing the ranking of different positions in the crew. Then, by combining these results together, we would get the full list of crew members that are most likely to succeed in producing a high-quality movie of the target movie genre, maximizing the possible amount of profit that Zebra could earn from that work and the opportunity of Zebra reaching its business success.

The benefit that our customers could receive from our work is even more than what just mentioned above. Whenever in the future Zebra detects a change of the trend and plans to produce a different target movie genre, our database system would allow them to access all kinds of information quickly and filter out the best-fit film crews to again achieve their business goals of maximizing the profit. Such a data-based decision-making process will help the company to effectively reduce the risks of producing a failed movie and provide decision-makers with more criteria to judge.

**Normalization Plan**

The 7 raw datasets we extracted from IMDb contain massive information such as titles, crews, genres, and ratings of movies and TV, etc. However, the structure of the raw datasets is unorganized. Since our primary goal is to help the company build a perfect production team efficiently for the intended movie, it is important for us to first separate some key information from the existing tables with appropriate normalization procedures according to the 3rd normal form principles.

In these raw datasets, all tables have variables of either “tconst”, which is a primary key of table “title.basics”, or “nconst”, which is a primary key of table “name.basics”. We decided to start with the tables that related to titles to conduct 3rd normalization.

There are 9 variables in table ‘title\_basics’ including tconst, titleType, primaryTitle, originalTitle, isAdult, startYear, endYear, runtimeMinutes, and genres. All the variables conform to 1st and 2nd normalization except genres since some titles may have more than one genre. Therefore, we created table ‘title’ for the remaining eight variables, and separated genre from the table ‘title’. Then we created another two tables called ‘genre’ and ‘title\_genre’. Table ‘genre’ consists of variable genre\_id as the primary key and genre\_name to show a specific genre. Table ‘title\_genres’ is a bridge table to depict the relationship between genres and titles, consisting of variables tconst as the primary key and genre\_id to connect with different kinds of genres.

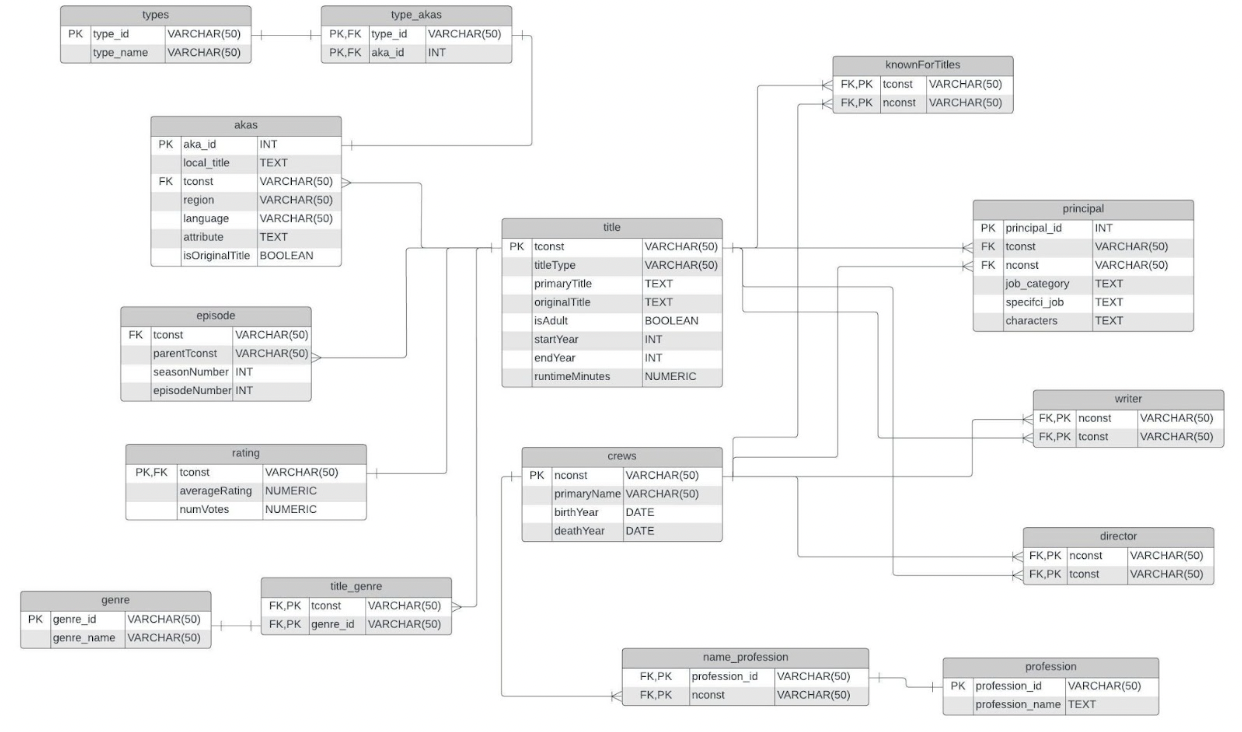
Table ‘title.akas’ consists of several variables including titleId, ordering, title, region, language, types, attributes, and isOriginalTitle. In this table, the combination of titleId and ordering is a composite key to identify the uniqueness of each row. To enhance the efficiency and make the relationship more clear, we added a new variable aka\_id as the primary key. Considering that we may have frequent queries on types, we separated variable types from the original table and created new table types to store different types with their type\_id as the primary key. Then we picked up the remaining variables to build a new table called ‘akas’ with aka\_id as the primary key and titleId as the foreign key from table ‘title’. Another bridge table ‘aka\_types’ was created as well, which contains aka\_id, type\_id to show the connection between akas and types.

Table ‘title.episode’ contains variables including tconst, parentTconst, seasonNumber, and episodeNumber. This follows the principles of the 3rd normalization form, so we can use this table directly for our table ‘episode’ which have tconst as the primary key and parentTconst as the foreign key of table title.

Table ‘title.ratings’ have variables including tconst, averageRating, and numVotes. This follows the principles of the 3rd normalization form so that we can use this table directly for our table rating, with tconst as the primary key and the foreign key of table title.

Table ‘name.basics’ contains variables including nconst, primaryName, birthYear, deathYear, primaryProfession, and knownForTitles. The variable primaryProfession is not in accord with 1st normalization form since it is not atomic, so we separated this variable in a new table called ‘profession’. We extracted knownForTitles from the table and created new tables for them to query more efficiently. The remaining variables were formed as table ‘crews’ with nconst as primary key. Table ‘profession’ was created to store different professions with profession\_id as its primary key. The bridge table ‘name\_profession’ was used to represent the exact profession for each nconst in table crews. Table ‘knownForTitles’ is also a bridge table to link table ‘title’ and table ‘crews’.

Table ‘title.crew’ contains the tconst, directors, and writers variables. The values of directors and writers are nconst in table crews. So we divided this table into two tables – table ‘director’ and table ‘writer’. Both of the tables have variables of tconst and nconst. All the variables accord with the 3rd normalization form.

Table ‘title.principals’ consists of 6 variables including tconst, ordering, nconst, category, job, and characters. In this table, the combination of tconst and ordering is a composite key of the table. We added a new variable principal\_id as the primary key and removed the variable ordering to table ‘principals’. The rest of the variables are perfect for the 3rd normalization form.

**ETL Process**

Our team's whole ETL process is mainly implemented through python. We gain a lot of data from IMDb, but they are all tsv files which are hard to import to the database directly.

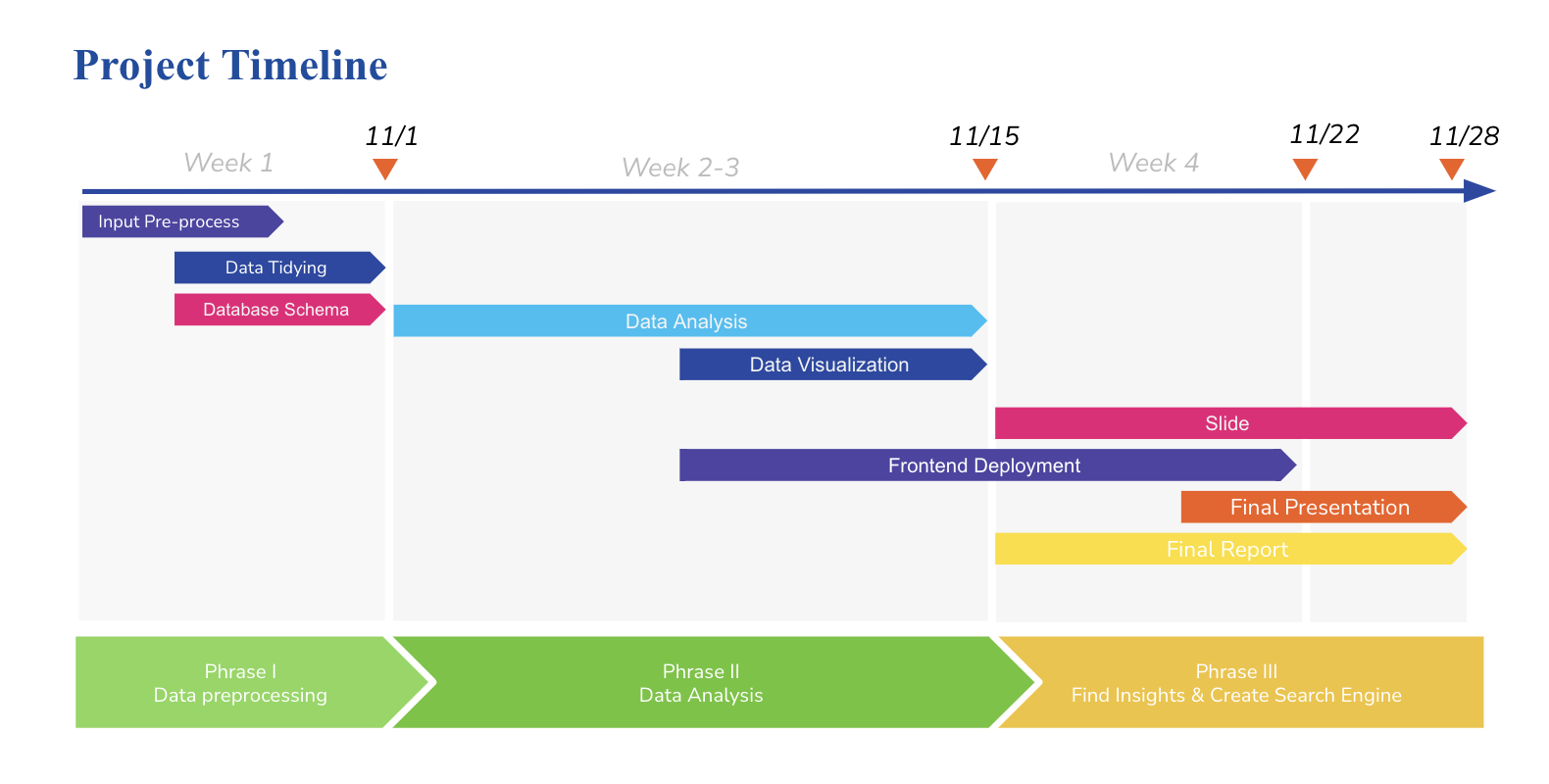
The first step in the ETL process is to extract the data that is useful for the project. We import the data from IMDb through pandas function in python and save the file type as csv file, because csv file can help us to view the composition and variable names of the dataset more easily through excel. In the exploration of the data, we found some missing data in the dataset, but we found in the dictionary of IMDb data that these missing data are not a mistake in collecting the data, but there is no specific data to show. So we replaced the missing data in the dataset with none, which can more accurately help customers understand the search results. Before data conversion, our main task was to split our dataset into title, akas, episode, type\_akas, types, title\_genre, genre, writer, director, crews, name\_profession, profession, principal, knownforTitles, and rating according to the third database normalization we developed. For tables like type\_akas, title genre, writer, director, name\_profession, we established the relationships in the main data table. We set new unique values for specific variables such as IDs via python. For example, for the genre of a film or TV production, we extract the data separately and split the genre by a comma to get unique data for the genre of the film or TV production. We add IDs to the movie genres by sorting them to link the data tables. After determining the linkage in the data table, we found that the data in the primary key of the data table may not cover the foreign keys of the other data tables, so we used the primary key of the core data table as the standard and removed the redundant data from the rest of the data tables. We spent a lot of time in the conversion step for data import and data splitting into the data frames we needed because of the large volume of data.

The second part of the ETL process is to transform and save the imported data in python to the database. In python, we used the create\_engine in the sqlalchemy package to connect to our local server's pgadmin for storing the data we prepared for our client's analysis of the film production. We used SQL code to create the tables in our local database and import the data. We created 15 tables following the third database normalization determined before the ETL, and for each data we needed to convert into the database, we determined the data type and character space. We set the primary and foreign keys for each table, which are unique data like IDs. We imported our processed data frames into pgadmin via the to\_sql function.

In the third step of the ETL process, we decided to use Metabase as a data warehouse to load our data. We connected to the local database via Metabase and loaded the data tables. Metabase also supports SQL queries to get the relevant data. (We upload the entire ETL code to Appenddix I)

**Team Structure and Timeline**

| Tasks | Description | Date | Responsibilities |
| --- | --- | --- | --- |
| Data Tidying | Load packages about cleaning data  Clean and organize data for future investigation | 11/01 | Zhichen Xia |
| Database Schema | Create database and insert data into SQL database | 11/01 | Kai Hu |
| Data Analysis | Use python to analyze the data and discover the details and insights, i.e. find out the trend of most high-rated type of movies in specific time period | 11/15 | Jiaman Li |
| Data Visualization | Summarize data analysis with visualization to generate more insights | 11/15 | Jie Ming |
| Frontend Development | ETL and connect database and website | 11/22 | Wenpei Sun |
| Presentation | Deliver a presentation of the whole project including client scenario, original data sample, explanation of normalization plan, ETL process as well as a brief demo of database interaction. | 12/06 | All |
| Project Report | Deliver a professional report including client scenario, team contract, a sample of data, normalization plan, ETL process, and analytical procedures | 11/28 | Xiaotong Sui & Tsai-Hsuan Cheng |



**Database Schema**

CREATE TABLE types

(

type\_id varchar(50) NOT NULL,

types varchar(50) ,

PRIMARY KEY ( type\_id )

);

CREATE TABLE title

(

tconst varchar(50) NOT NULL,

title\_type varchar(50) ,

primary\_title text ,

original\_title text ,

is\_adult boolean,

start\_year int ,

end\_year int ,

run\_time\_minutes numeric ,

PRIMARY KEY ( tconst ),

CONSTRAINT title CHECK (is\_adult IN (True,False))

);

CREATE TABLE akas

(

aka\_id int PRIMARY KEY,

tconst varchar(50) NOT NULL,

local\_title text ,

region varchar(50) ,

language varchar(50) ,

attributes text ,

is\_original\_title boolean ,

FOREIGN KEY ( tconst ) REFERENCES title ( tconst ),

CONSTRAINT akas CHECK (is\_original\_title IN ('1','0'))

);

CREATE TABLE type\_akas

(

aka\_id int NOT NULL,

type\_id varchar (50) NOT NULL,

PRIMARY KEY ( type\_id, aka\_id),

FOREIGN KEY ( aka\_id ) REFERENCES akas ( aka\_id),

FOREIGN KEY ( type\_id )REFERENCES types ( type\_id )

);

CREATE TABLE episode

(

episode\_id varchar(50) NOT NULL PRIMARY KEY,

tconst varchar(50) ,

season\_number int ,

episode\_number int ,

FOREIGN KEY ( tconst ) REFERENCES title ( tconst )

);

CREATE TABLE genres

(

genre\_id varchar(50) NOT NULL,

genre\_name varchar(50) ,

PRIMARY KEY ( genre\_id )

);

CREATE TABLE title\_genres

(

tconst varchar(50) NOT NULL,

genre\_id varchar(50) NOT NULL,

PRIMARY KEY ( tconst, genre\_id),

FOREIGN KEY ( tconst ) REFERENCES title ( tconst ),

FOREIGN KEY ( genre\_id ) REFERENCES genres ( genre\_id )

);

CREATE TABLE rating

(

tconst varchar(50) NOT NULL,

averagerating numeric ,

numvotes numeric ,

PRIMARY KEY ( tconst ),

FOREIGN KEY ( tconst ) REFERENCES title ( tconst )

);

CREATE TABLE profession

(

profession\_id varchar(50) NOT NULL,

profession\_name text ,

PRIMARY KEY ( profession\_id )

);

CREATE TABLE crews

(

nconst varchar(50) NOT NULL,

primaryName text ,

birthYear int ,

deathYear int ,

PRIMARY KEY ( nconst )

);

CREATE TABLE name\_profession

(

nconst varchar(50) NOT NULL,

profession\_id varchar(50) NOT NULL,

PRIMARY KEY ( nconst, profession\_id),

FOREIGN KEY ( profession\_id ) REFERENCES profession ( profession\_id ),

FOREIGN KEY ( nconst ) REFERENCES crews ( nconst )

);

CREATE TABLE principal

(

principal\_id int PRIMARY KEY,

tconst varchar(50) NOT NULL,

nconst varchar(50) NOT NULL,

job\_category text ,

specific\_job text ,

characters text ,

FOREIGN KEY ( tconst ) REFERENCES title ( tconst ),

FOREIGN KEY ( nconst ) REFERENCES crews ( nconst )

);

CREATE TABLE known\_for\_titles

(

tconst varchar(50) NOT NULL,

nconst varchar(50) NOT NULL,

PRIMARY KEY ( tconst, nconst),

FOREIGN KEY ( tconst ) REFERENCES title ( tconst ),

FOREIGN KEY ( nconst ) REFERENCES crews ( nconst )

);

CREATE TABLE directors

(

tconst varchar(50) NOT NULL,

nconst varchar(50) ,

PRIMARY KEY ( tconst, nconst),

FOREIGN KEY ( nconst ) REFERENCES crews ( nconst ),

FOREIGN KEY ( tconst ) REFERENCES title ( tconst )

);

CREATE TABLE writers

(

tconst varchar(50) NOT NULL,

nconst varchar(50) ,

PRIMARY KEY ( tconst, nconst),

FOREIGN KEY ( nconst ) REFERENCES crews ( nconst ),

FOREIGN KEY ( tconst ) REFERENCES title ( tconst )

);

"""

**Interactions With the Database System**

The customer will interact with the database system we designed using Metabase, which will retrieve the data we collect in pgAdmin and present it to our clients. Our clients can query the data they need from the Metabase and visualize data in tables and charts based on their needs.

The Metabase will allow our clients to search for information about film ratings in general. The average rating of a film is used to reflect the public preference for films. The clients can check the average score of different types of films and associated personals, such as specific directors, writers, and actors/actresses. Our clients can understand the film market well by taking this information as quantitative indicators. For example, the clients can understand the recent popular film types and the type of films at which the director, writer, and actors/actresses are good. This way, our clients can make better decisions in selecting film types and building film production teams.

For the further investigation of our data, our clients can use predictive analysis to predict the public rating score of their coming films, which could help our clients to avoid heavily investing in movies that themes can not meet the public tastes and preferences and with teams that have no previous experience and advantages.

As an advanced tool that helps everyone dig deeper with an easy-to-use visual query builder, Metabase can handle the complexity of joins, aggregations, etc., allowing our clients to visualize trends in data easily, focus more on data analysis, and get practical insights to optimize their business operation rather than cherry-picking through SQL.

The results that we would implement for analysts and “C” level officers are completely different. For analysts, we will first compile an explanation of the glossary to introduce each relation and the columns in the relations, so they can understand what they could gain from our database and how to extract data under certain scenarios. Then we will create and save several questions related to our proposed objectives using Metabase's question function. Analysts could access most data they need through our prepared questions. Suppose these questions cannot satisfy their need. In that case, they could also implement queries using the question function even though they need help understanding SQL since Metabase could help them create queries as long as they understand simple language. After they obtained the data, they could also make some visualizations to show the results, finding patterns or summarizing information for film planning suggestions and strategies.

To ensure that the “C” level officers understand our work and the profit it could bring to the company comprehensively, we would generate the details of our work into a well-organized report. The content of this report could be briefly divided into four parts. First, we would describe the data the company originally had, including the number of data sets, the sizes, the variables, and all the situations, such as redundancy and the null values. Second, we would describe the work that we have done to organize the data to meet the requirement of 3NF, such as (1) using python for data cleaning and splitting the tables and columns and (2) importing the data into our selected database system, pgAdmins, for future data query. Third, we would describe several examples of how the work we created could benefit the company. For example, if the “C” level officers decide to produce a new active movie, they could easily know the candidates of directors and actors by querying in the database system. Last but not least, since we would display the visualization on the Metabase, we would also mention that in the last part of our report, along with the detailed guide and description.

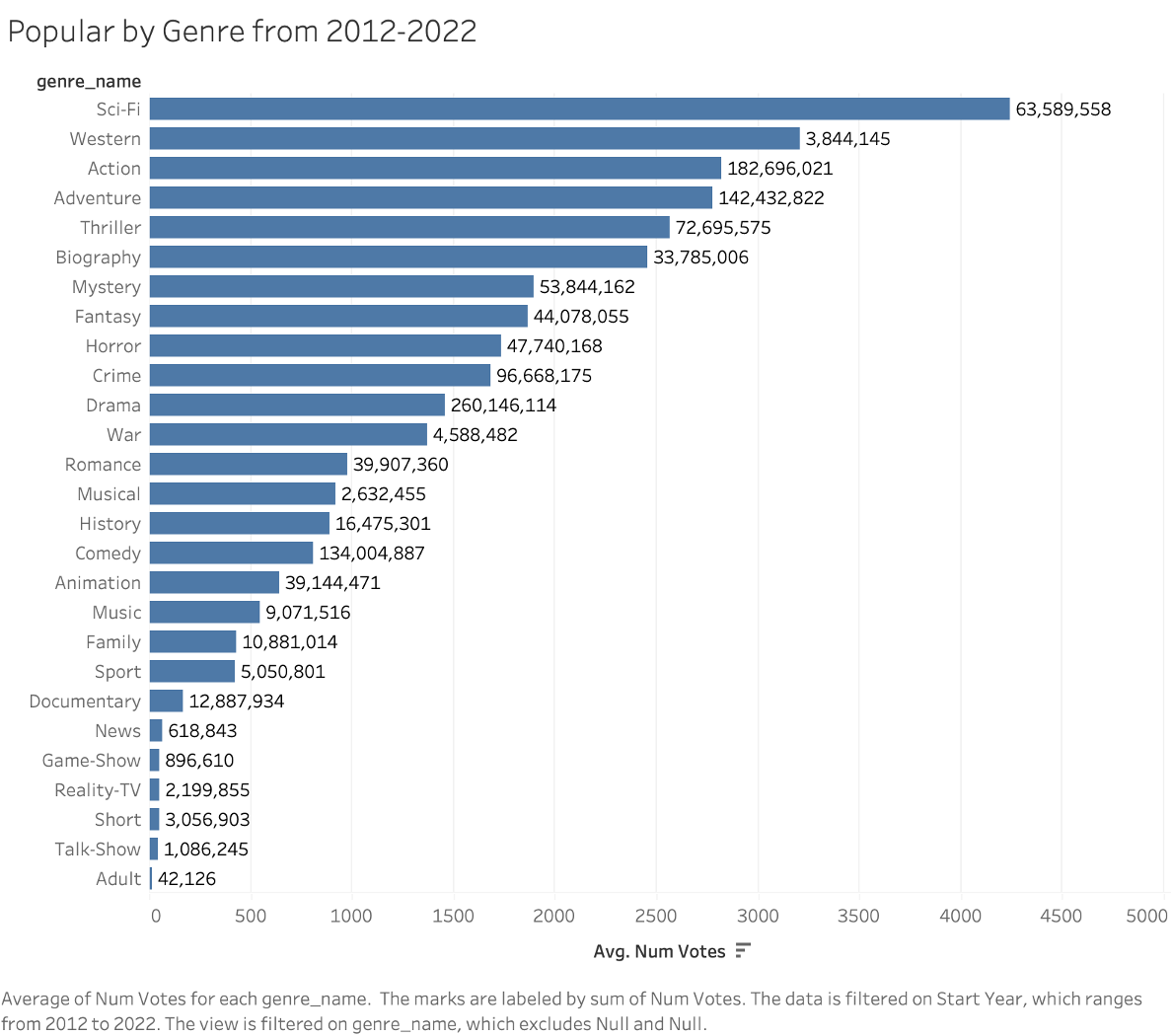
There are plenty of business intelligence tools available. To satisfy the client's requirements, firstly, we selected candidates from Tableau, Metabase, and PowerBI. All three can access multiple databases and provide certain features to make the interactive dashboard. Metabase is a flexible software integrating analytical tools, which has the advantage of a user-friendly interface for non-technical users and beginners to create data visualizations easily. Also, Metabase can generate reports and update its system automatically and make visualization and calculation suggestions for users, such as the mean value by a variable. Therefore, our team believes Metabase is the most suitable tool used for the project.

As to resolve the redundancy and improve our database’s performance, for the ER-Diagram, our group created categorized and standardized tables and bridge tables to reduce the redundancy and used ERD cardinality and primary and foreign keys to show the relationships among these tables. Moreover, we also defined data constraints for each attribute, such as data length, type, and existence, which could benefit future data management when inserting or updating new data into the database. Also, the well-structured database system with clear relationships and categorized tables could help our clients to retrieve the data they want fast and use it for future analysis. For the coding part, our group mainly used groupby(), unique(), isin(), etc. built-in functions and functions from the pandas package to standardize the attributes, reduce redundant data, and ensure the consistency of data in each table.

**Analytics Application**

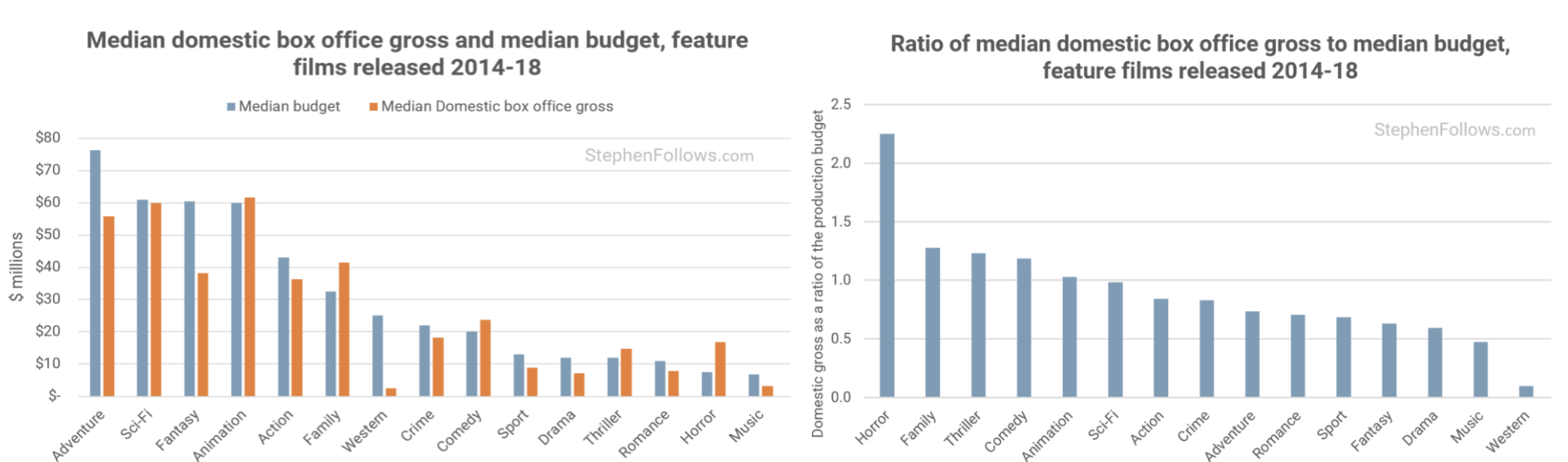
Zebra, a new rising movie and television production company, wants to select a movie genre that could capture the market trends for its initial production as well as recruit a best-fit film production team, such as writers, directors, and actors to ensure the success. Besides, as a new company, Zebra wants its new movie to enhance brand awareness, meet budget constraints, and have a low risk of investment.

To better predict the trends and produce movies, the filmmakers and industry executives at Zebra decided to hire a business analyst team for better decision-making by analyzing the IMDb, which is an online database that includes all the up-to-date information on trend movies, famous writers, directors, actors, and their related information, such as personal biographies, plot summaries, ratings, fans, and critical reviews.

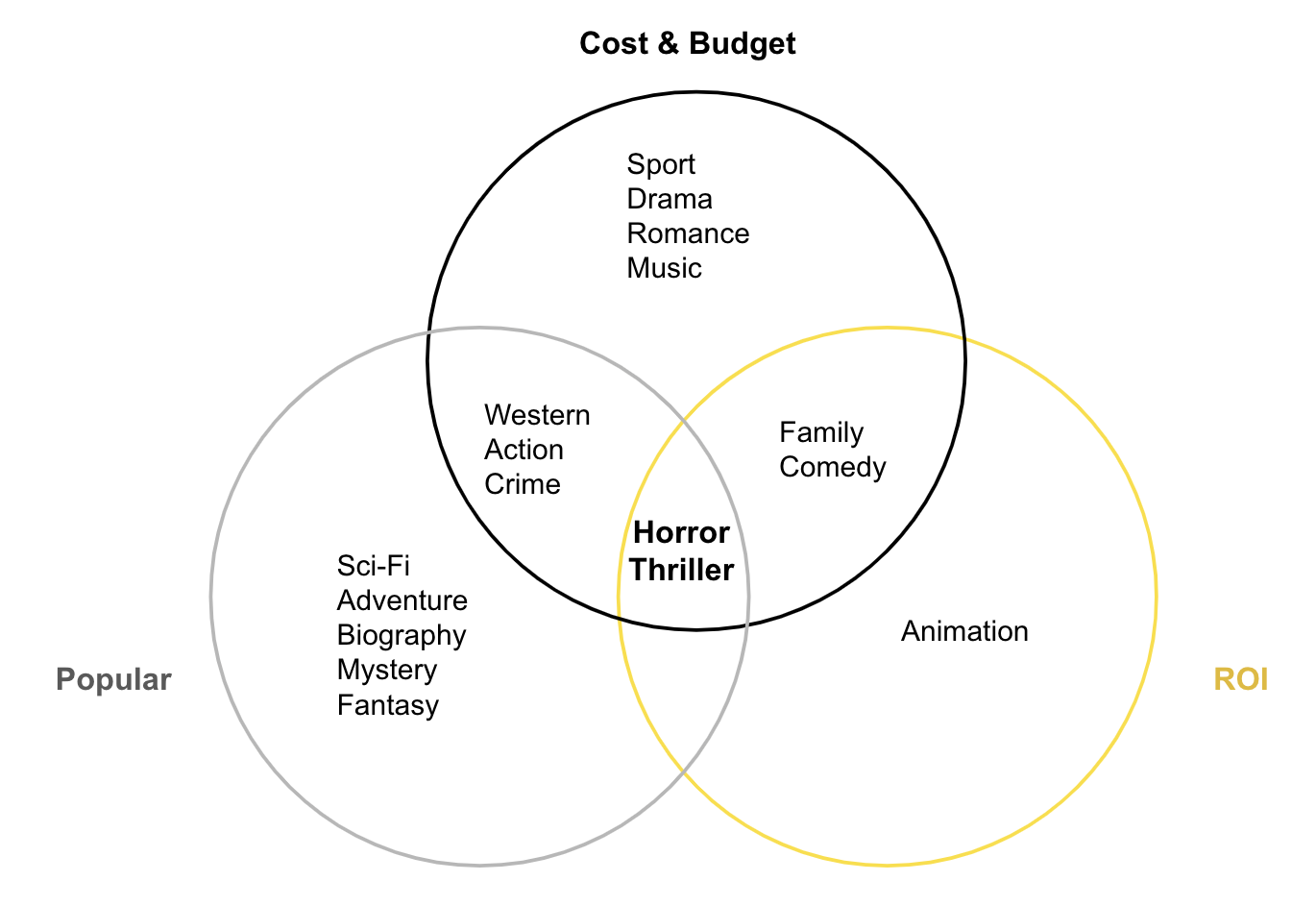
During the analysis process, firstly we, the business analyst team, selected variables: Num Votes, Start Year, and genre and made a bar chart by using Tableau, which reflects the popularity of the different genres of movies from the nearest 10 years. The top 10 most popular genre movies are Sci-Fi, Western, Action, Adventure, Thriller, Biography, Mystery, Fantasy, Horror, and Crime. In other words, these genres of movies can meet the client’s expectations for enhancing brand awareness (see Chart I).

*Chart I*

Then, the business analyst team conducted research online and found the rank for the median budget and the ratio of median box office gross (U.S. & Canada) to the median budget of movies from 2014 - 2018 under different genres. The prior rank shows the genres of movies that have a median budget under $50 million dollars are action, family, western, crime, comedy, sport, drama, thriller, romance, horror, and music (see Chart II). The later rank shows that genres of movies that are most likely to be profitable and low in risk of investment, ratio larger than 1, are horror, family, thriller, comedy, and animation (see Chart III)

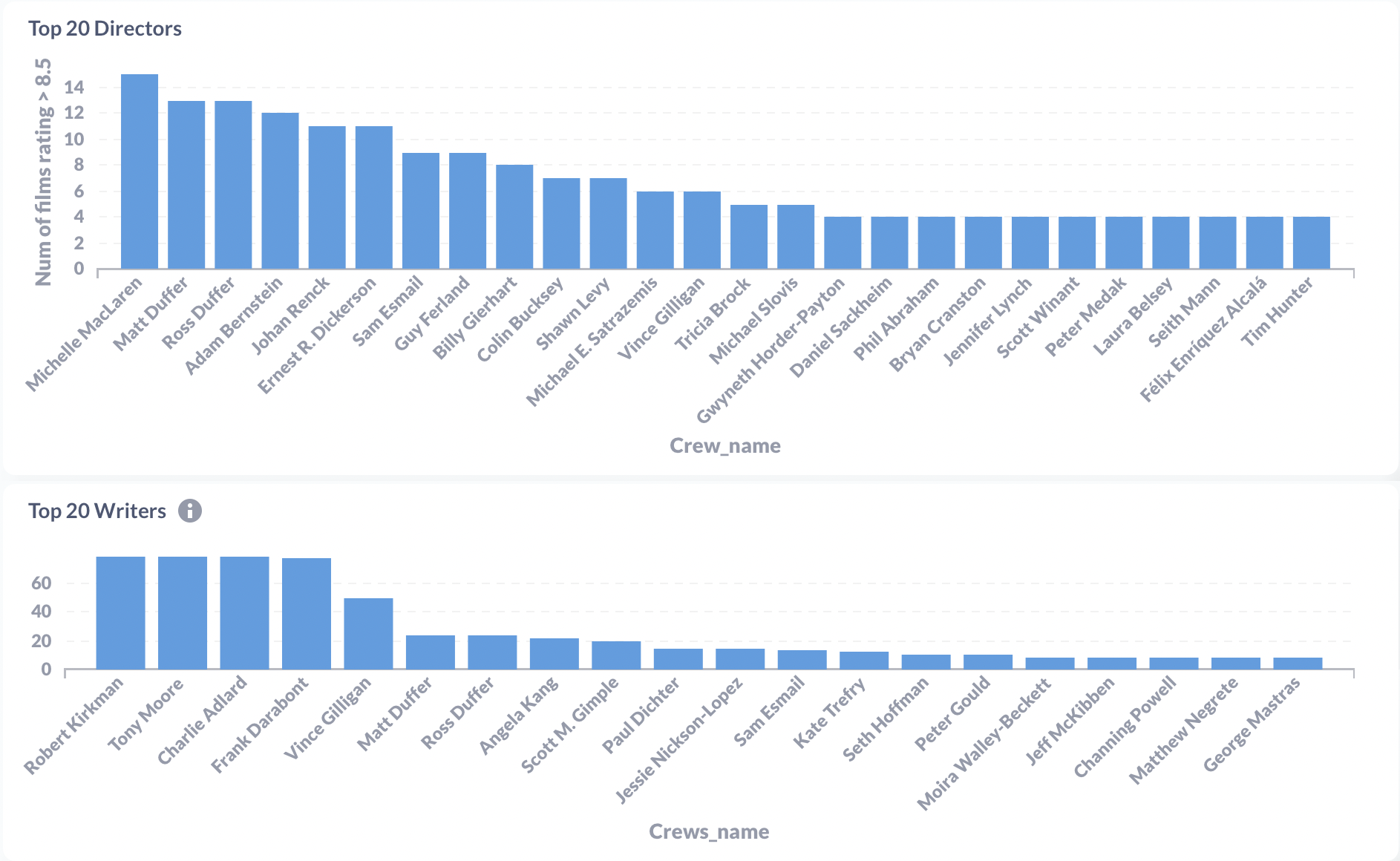


*Chart II Chart III*

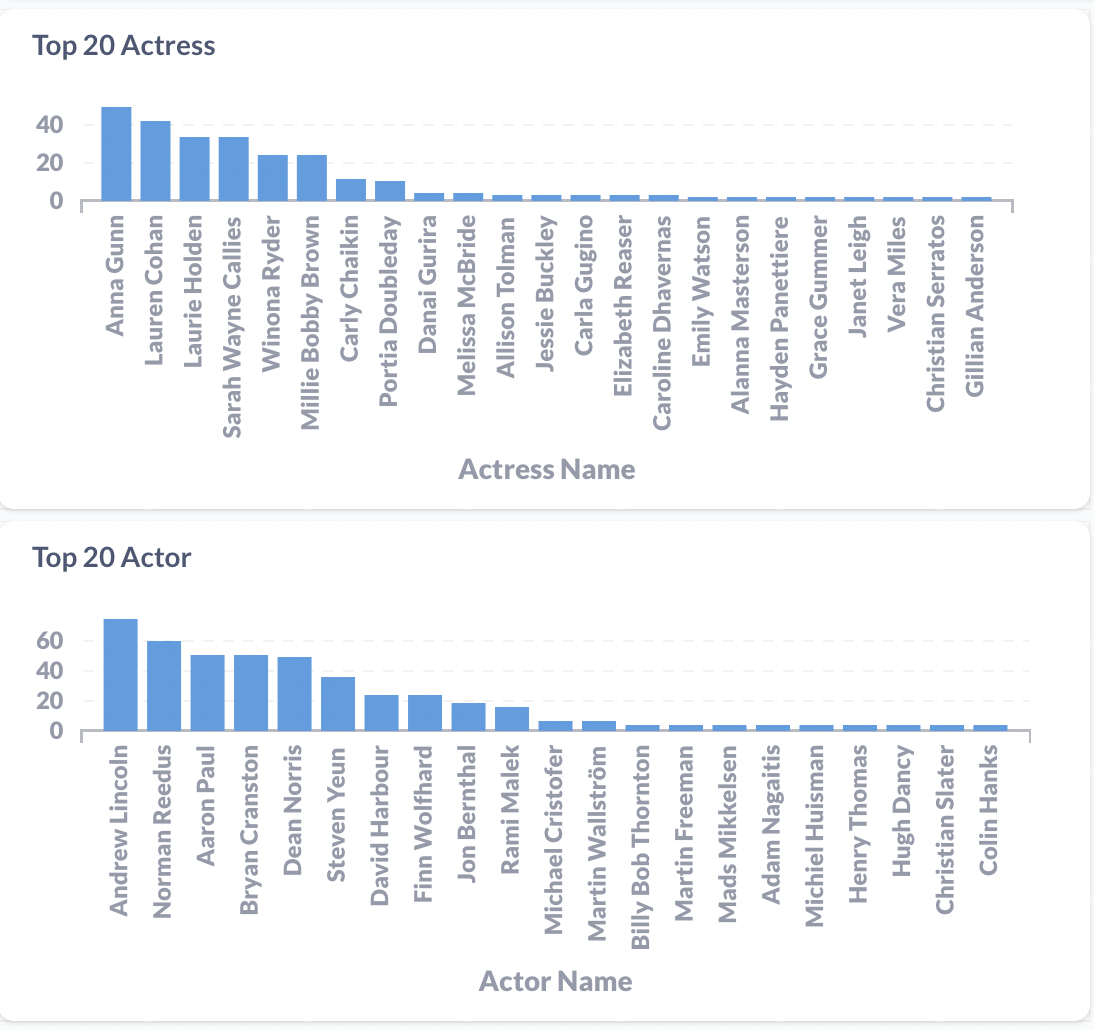
Looking at the correlation of the genres of movies that can meet the popularity, budget constraints, and low risk of investment, our analysis team found horror and thriller are the genres that can satisfy all three requirements. In detail, thriller movies are more popular and have higher budgets, and are less profitable than those horror movies generally (see Chart IV). 

Therefore, our analysis team decided to recommend Zebra to produce horror or thriller genre movies,

After selecting the genre of the film, our next step is to select the appropriate film team for the company. We have two methods to achieve this goal.



The first method is to filter the database for finding the best directors, writers, actresses, and actors who have made horror or thriller films. The indicator of excellent crews is the number of movies with an average rating score of over 8.5 and over 10,000 ratings. Therefore, we get four charts showing the top 20 directors, writers, actors, and actresses respectively. These tables represent the teams that are best at working on horror or thriller films. And they also represent the teams most likely to create a great film and avoid risk. The company can just discuss with people at the tables and make decisions about the film team.



The second method is to choose the best director who has made horror or thriller films. The indicator of the best director is how many movies he/she made with an average rating of over 8.5 and over 10,000 ratings. Then rank his best films by the average rating score and the number of ratings. We can try to build the same team that got better performance before. This method may be more representative, the audience would prefer to see the original cast starring in the film.

We believe that both approaches are effective in helping Zebra choose the most suitable team according to the budget. Our approach helps companies provide the most cost-effective team while reducing the risk. (The code for 10 analytical procedures that led to valuable insights is in Appendix II.)

**Conclusion**

In this project, we completed our client’s goal by providing two core recommendations and a reusable relational database management system to our client, Zebra. The two core recommendations include (1) recommending two genres that capture the market trend and (2) recommending the film crew accordingly to achieve success, while the database system would allow them to access all kinds of information quickly and filter out the best-fit film crews to again achieve their business goals of maximizing the profit.

In order to achieve the goal, we have created tables including genres, genre-related, film crews, crew-related, and KPI (i.e. rating), which led us to create 15 tables that followed the rules of the third normal form to do a comprehensive analysis. In the ETL part, we utilized python with the Pandas package to import data into our chosen relational database management system, pgAdmin. Meanwhile, the data was stored as csv files before we appended them into the database by sqlalchemy, which saves us plenty of time and avoids crashing the system. Then, the database was connected to the intelligence tool, Metabase, where we performed our two analyses there. Through the analysis, we found a crucial insight that the thriller and horror genres are the two most suitable choices among the aspects of popularity, cost & budget, and ROI. Based on the first insight, we then generated a list of film crews that would form the best team. Before the analysis is open to the public and the market becomes saturated, our client can take advantage of data science over other competitors, which enables them to have better exposure to the audience, financial performance, and resources.

**Appendix I** (ETL Code)

import pandas as pd

df\_crew = pd.read\_csv ("df\_crew.csv")

#####################. title

title\_basics=pd.read\_csv('df\_tb.csv')

tb\_title = title\_basics.drop(['Unnamed: 0','genres'], axis=1)

tb\_title.head()

tb\_title.info()

tb\_title[tb\_title['tconst'].duplicated() == True]

tb\_title['titleType'].unique()

title\_basics['isAdult'].unique()

tb\_title['isAdult'] = tb\_title['isAdult'].replace([0, 1], ['0','1'])

tb\_title[tb\_title['isAdult'].isin([2019, 1981, 2020, 2017, 2014, 2005])]

tb\_title['isAdult'] = tb\_title['isAdult'].replace([2019, 1981, 2020, 2017, 2014, 2005],

[None, None, None, None, None, None])

tb\_title['runtimeMinutes'].unique()

tb\_title['runtimeMinutes'] = tb\_title['runtimeMinutes'].replace(['Talk-Show','Reality-TV','Game-Show','Animation,Comedy,Family','Game-Show,Reality-TV','Documentary'],

[None, None, None, None, None, None])

tb\_title = tb\_title.rename(columns={"titleType":"title\_type",

"primaryTitle":"primary\_title",

"originalTitle":"original\_title",

"isAdult":"is\_adult",

"startYear":"start\_year",

"endYear":"end\_year",

"runtimeMinutes":"run\_time\_minutes"})

title\_akas = pd.read\_csv('df\_aka.csv')

title\_akas.head()

tb\_akas = title\_akas.drop(['ordering', 'types','Unnamed: 0'], axis=1)

tb\_akas

tb\_akas['region'].unique()

tb\_akas['language'].unique()

tb\_akas.groupby(['attributes'])['attributes'].count()

tb\_akas['isOriginalTitle'].unique()

tb\_akas['isOriginalTitle'] = tb\_akas['isOriginalTitle'].replace([0, 1], ['0','1'])

tb\_akas = tb\_akas.rename(columns={"title":"local\_title",

"titleId":"tconst",

"isOriginalTitle":"is\_original\_title"})

tb\_akas[~tb\_akas['tconst'].isin(title\_basics['tconst'])]

######## delete rows that tconst is not in Title Table

tb\_akas = tb\_akas[tb\_akas['tconst'].isin(title\_basics['tconst'])]

tb\_akas.insert(0, 'aka\_id', range(1, 1 + len(tb\_akas)))

tb\_akas

######epi

title\_episode = pd.read\_csv('df\_epi.csv')

title\_episode = title\_episode.drop(['Unnamed: 0'], axis=1)

title\_episode

title\_episode = title\_episode.rename(columns={"tconst":"episode\_id",

"parentTconst":"tconst",

"seasonNumber":"season\_number",

"episodeNumber":"episode\_number"})

title\_episode[title\_episode['tconst'].duplicated() == True]

title\_episode.groupby(['tconst'])['tconst'].count().sort\_values(ascending=False)

title\_episode.groupby(['episode\_id'])['episode\_id'].count().sort\_values(ascending=False)

title\_episode[~title\_episode['tconst'].isin(title\_basics['tconst'])]['tconst'].unique()

############ Delete rows that tconst is not in title

title\_episode = title\_episode[title\_episode['tconst'].isin(title\_basics['tconst'])]

title\_episode

############## type

df\_aka=pd.read\_csv('df\_aka.csv')

df\_aka= df\_aka.rename(columns={'titleId':'tconst'})

df\_aka = df\_aka[df\_aka['tconst'].isin(tb\_title['tconst'])]

df\_aka

df\_aka.insert(0,'aka\_id',range(1,1+len(df\_aka)))

df\_aka\_types = df\_aka[['aka\_id','types']]

df\_aka\_types = df\_aka\_types.rename(columns={"types":"type\_name"})

df\_aka\_types

df\_aka\_type = df\_aka\_types.drop(['types'],axis=1).join(df\_aka\_types['types'].str.split('',expand = True).stack().reset\_index(level=1,drop=True).rename('types'))

######## Type table

temp\_type\_df = pd.DataFrame(df\_aka\_type.types.unique(), columns=['types'])

# types table

temp\_type\_df.insert(0, 'type\_id', range(1, 1 + len(temp\_type\_df)))

temp\_type\_df

#merge two tables

type\_akas = pd.merge(df\_aka\_type, temp\_type\_df, on = ['types'],how='left')[['aka\_id', 'type\_id', 'types']]

#type\_akas

type\_akas = type\_akas[['aka\_id','type\_id']]

type\_akas

##################### name basic

df\_nb=pd.read\_csv('df\_nb.csv')

df\_nb

df\_nb = df\_nb[['nconst','primaryName','birthYear','deathYear']]

df\_nb = df\_nb.rename(columns={"primaryName":"primaryname",

"birthYear":"birthyear",

"deathYear":"deathyear"})

df\_nb

############# rating

df\_rating=pd.read\_csv('df\_rating.csv')

df\_rating

columns = [column.lower() for column in list(df\_rating.columns)]

df\_rating.columns= columns

df\_rating = df\_rating[['tconst','averagerating','numvotes']]

df\_rating

######### delete rows that tconst is not in Title Table

df\_rating = df\_rating[df\_rating['tconst'].isin(tb\_title['tconst'])]

df\_rating

################# principal table

df\_pri=pd.read\_csv('df\_pri.csv')

df\_pri

df\_pri = df\_pri[['tconst','nconst', 'category', 'job','characters']]

df\_pri

####### delete rows that tconst is not in Title Table

df\_pri = df\_pri[df\_pri['tconst'].isin(tb\_title['tconst'])]

####### delete rows that nconst is not in Crews Table

df\_pri = df\_pri[df\_pri['nconst'].isin(df\_nb['nconst'])]

df\_pri.insert(0, 'principal\_id', range(1, 1 + len(df\_pri)))

df\_pri

columns = ['principal\_id','tconst','nconst', 'job\_category', 'specific\_job','character']

df\_pri.columns= columns

df\_pri

########### genres

df\_tb=pd.read\_csv('df\_tb.csv')

df\_tb

genre\_name = df\_tb[['tconst','genres']]

genre\_name = genre\_name.rename(columns={"tconst":"tconst",

"genres":"genre\_name"})

genre\_name

lst\_col = 'genre\_name'

genre\_name = genre\_name.assign(\*\*{lst\_col:genre\_name[lst\_col].str.split(',')})

genre\_name= genre\_name.explode(['genre\_name'])

genre\_name

genres = df\_tb["genres"].str.split(',', expand=True)

genres.head()

series1 = pd.Series(genres[0])

genre = series1.unique()

genre = pd.DataFrame(genre, columns = ["genre\_name"])

genre.insert(0, 'genre\_id', range(1, 1 + len(genre)))

############# title\_genre

title\_genres = df\_tb[['tconst','genres']]

title\_genres = title\_genres.rename(columns={"tconst":"tconst",

"genres":"genre\_name"})

title\_genres.head()

lst\_col = 'genre\_name'

title\_genres = genre\_name.assign(\*\*{lst\_col:genre\_name[lst\_col].str.split(',')})

title\_genres= genre\_name.explode(['genre\_name'])

title\_genres.head()

title\_genres = pd.merge(title\_genres, genre,on = 'genre\_name',how='left')[['tconst', 'genre\_id', 'genre\_name']]

title\_genres = title\_genres.drop(['genre\_name'], axis=1)

title\_genres['genre\_id'] = title\_genres['genre\_id'].astype('Int64')

title\_genres

################# profession

df\_nb1=pd.read\_csv('df\_nb.csv')

name\_profession= df\_nb1[["nconst", "primaryProfession"]]

name\_profession = name\_profession.rename(columns={"nconst":"nconst",

"primaryProfession":"profession\_name"})

name\_profession.head()

lst\_col = 'profession\_name'

name\_profession = name\_profession.assign(\*\*{lst\_col:name\_profession[lst\_col].str.split(',')})

name\_profession = name\_profession.explode(['profession\_name'])

name\_profession.head()

# split values in each cell and assign to multiple columns

primaryProfession = df\_nb1["primaryProfession"].str.split(',', expand=True)

primaryProfession.head()

# define the profession table

series1 = pd.Series(primaryProfession[0])

profession = series1.unique()

profession = pd.DataFrame(profession, columns = ["profession\_name"])

profession.insert(0, 'profession\_id', range(1, 1 + len(profession)))

# profession table complete

profession

#################### name\_profession table

# combine name\_profession and profession tables

name\_profession = pd.merge(name\_profession, profession, on = 'profession\_name',how='left')[['nconst', 'profession\_id', 'profession\_name']]

name\_profession

# drop profession\_name column

name\_profession = name\_profession.drop(['profession\_name'], axis=1)

name\_profession

# define name\_profession

name\_profession['profession\_id'] = name\_profession['profession\_id'].astype('Int64')

name\_profession

name\_profession = name\_profession[name\_profession['nconst'].isin(df\_nb['nconst'])]

name\_profession # name\_profession table delete 0 rows

################### knownForTitles

df\_pri1=pd.read\_csv('df\_pri.csv')

knownForTitles = df\_pri1[["tconst","nconst"]]

knownForTitles

####### delete rows that tconst is not in Title Table

knownForTitles = knownForTitles[knownForTitles['tconst'].isin(tb\_title['tconst'])]

knownForTitles

####### delete rows that nconst is not in Crews Table

knownForTitles = knownForTitles[knownForTitles['nconst'].isin(df\_nb['nconst'])]

knownForTitles

#### Director

df\_director = df\_crew[['tconst', 'directors']]

lst\_col = 'directors'

df\_director = df\_director.assign(\*\*{lst\_col:df\_director[lst\_col].str.split(',')})

df\_director = df\_director.explode(['directors'])

df\_director

####### delete rows that tconst is not in Title Table

df\_director = df\_director[df\_director['tconst'].isin(tb\_title['tconst'])]

####### delete rows that nconst is not in Crews Table

df\_director = df\_director[df\_director['directors'].isin(df\_nb['nconst'])]

####### Director Table Data

df\_director

#### Writer

df\_writer = df\_crew[['tconst', 'writers']]

lst\_col = 'writers'

df\_writer = df\_writer.assign(\*\*{lst\_col:df\_writer[lst\_col].str.split(',')})

df\_writer = df\_writer.explode(['writers'])

df\_writer

####### delete rows that tconst is not in Title Table

df\_writer = df\_writer[df\_writer['tconst'].isin(tb\_title['tconst'])]

####### delete rows that nconst is not in Crews Table

df\_writer = df\_writer[df\_writer['writers'].isin(df\_nb['nconst'])]

df\_writer

#### rename column

df\_director.rename(columns = {'directors':'nconst'}, inplace = True)

df\_writer.rename(columns = {'writers':'nconst'}, inplace = True)

####---------------------------connect pgadmin

import pandas as pd

from sqlalchemy import create\_engine

# Pass the connection string to a variable, conn\_url

conn\_url = 'postgresql://postgres:123@localhost/test4'

# Create an engine that connects to PostgreSQL server

engine = create\_engine(conn\_url)

# Establish a connection

connection = engine.connect()

drop = '''

DROP TABLE IF EXISTS writers CASCADE;

DROP TABLE IF EXISTS directors CASCADE;

DROP TABLE IF EXISTS knownForTitles CASCADE;

DROP TABLE IF EXISTS principal CASCADE;

DROP TABLE IF EXISTS name\_profession CASCADE;

DROP TABLE IF EXISTS crews CASCADE;

DROP TABLE IF EXISTS profession CASCADE;

DROP TABLE IF EXISTS rating CASCADE;

DROP TABLE IF EXISTS title\_genres CASCADE;

DROP TABLE IF EXISTS genres CASCADE;

DROP TABLE IF EXISTS episode CASCADE;

DROP TABLE IF EXISTS type\_akas CASCADE;

DROP TABLE IF EXISTS akas CASCADE;

DROP TABLE IF EXISTS title CASCADE;

DROP TABLE IF EXISTS types CASCADE;

'''

# Execute the statement to create tables

connection.execute(drop)

# Pass the SQL statements that create all tables

stmt = """

CREATE TABLE types

(

type\_id varchar(50) NOT NULL,

type\_name varchar(50) ,

PRIMARY KEY ( type\_id )

);

CREATE TABLE title

(

tconst varchar(50) NOT NULL,

title\_type varchar(50) ,

primary\_title text ,

original\_title text ,

is\_adult boolean ,

start\_year int ,

end\_year int ,

run\_time\_minutes numeric ,

PRIMARY KEY ( tconst ),

CONSTRAINT title CHECK (is\_adult IN ('1','0'))

);

CREATE TABLE akas

(

aka\_id int PRIMARY KEY,

local\_title text ,

tconst varchar(50) NOT NULL,

region varchar(50) ,

language varchar(50) ,

attributes text ,

is\_original\_title boolean ,

FOREIGN KEY ( tconst ) REFERENCES title ( tconst ),

CONSTRAINT akas CHECK (is\_original\_title IN ('1','0'))

);

CREATE TABLE type\_akas

(

aka\_id int NOT NULL,

type\_id varchar (50) NOT NULL,

PRIMARY KEY ( type\_id, aka\_id),

FOREIGN KEY ( aka\_id )REFERENCES akas ( aka\_id),

FOREIGN KEY ( type\_id )REFERENCES types ( type\_id )

);

CREATE TABLE episode

(

episode\_id varchar(50) NOT NULL PRIMARY KEY,

tconst varchar(50) ,

season\_number int ,

episode\_number int ,

FOREIGN KEY ( tconst ) REFERENCES title ( tconst )

);

CREATE TABLE genres

(

genre\_id varchar(50) NOT NULL,

genre\_name varchar(50) ,

PRIMARY KEY ( genre\_id )

);

CREATE TABLE title\_genres

(

tconst varchar(50) NOT NULL,

genre\_id varchar(50) NOT NULL,

PRIMARY KEY ( tconst, genre\_id),

FOREIGN KEY ( tconst ) REFERENCES title ( tconst ),

FOREIGN KEY ( genre\_id ) REFERENCES genres ( genre\_id )

);

CREATE TABLE rating

(

tconst varchar(50) NOT NULL,

averageRating numeric ,

numVotes numeric ,

PRIMARY KEY ( tconst ),

FOREIGN KEY ( tconst ) REFERENCES title ( tconst )

);

CREATE TABLE profession

(

profession\_id varchar(50) NOT NULL,

profession\_name text ,

PRIMARY KEY ( profession\_id )

);

CREATE TABLE crews

(

nconst varchar(50) NOT NULL,

primaryName text ,

birthYear int ,

deathYear int ,

PRIMARY KEY ( nconst )

);

CREATE TABLE name\_profession

(

profession\_id varchar(50) NOT NULL,

nconst varchar(50) NOT NULL,

PRIMARY KEY ( nconst, profession\_id),

FOREIGN KEY ( profession\_id ) REFERENCES profession ( profession\_id ),

FOREIGN KEY ( nconst ) REFERENCES crews ( nconst )

);

CREATE TABLE principal

(

principal\_id int PRIMARY KEY,

tconst varchar(50) NOT NULL,

nconst varchar(50) NOT NULL,

job\_category text ,

specific\_job text ,

characters text ,

FOREIGN KEY ( tconst ) REFERENCES title ( tconst ),

FOREIGN KEY ( nconst ) REFERENCES crews ( nconst )

);

CREATE TABLE knownForTitles

(

tconst varchar(50) NOT NULL,

nconst varchar(50) NOT NULL,

PRIMARY KEY ( tconst, nconst),

FOREIGN KEY ( tconst ) REFERENCES title ( tconst ),

FOREIGN KEY ( nconst ) REFERENCES crews ( nconst )

);

CREATE TABLE directors

(

tconst varchar(50) NOT NULL,

nconst varchar(50) ,

PRIMARY KEY ( tconst, nconst),

FOREIGN KEY ( nconst ) REFERENCES crews ( nconst ),

FOREIGN KEY ( tconst ) REFERENCES title ( tconst )

);

CREATE TABLE writers

(

tconst varchar(50) NOT NULL,

nconst varchar(50) ,

PRIMARY KEY ( tconst, nconst),

FOREIGN KEY ( nconst ) REFERENCES crews ( nconst ),

FOREIGN KEY ( tconst ) REFERENCES title ( tconst )

);

"""

# Execute the statement to create tables

connection.execute(stmt)

####insert value ---------- title

# Insert values to table title

tb\_title.to\_sql(name='title', con=engine, if\_exists='append', index=False)

######insert value ---------- genre

genre.to\_sql(name='genre', con=engine, if\_exists = 'append', index=False)

#######insert value ---------- title\_genre

title\_genres.to\_sql(name='title\_genre', con=engine, if\_exists = 'append', index=False)

####insert value ---------- crews

df\_nb.to\_sql(name='crews',con = engine, if\_exists = 'append', index = False)

######insert value ---------- principal

df\_pri.to\_sql(name='principal',con = engine, if\_exists = 'append', index = False)

# Insert values to table akas

tb\_akas.to\_sql(name='akas', con=engine, if\_exists='append', index=False)

#insert Value --------- episode table

title\_episode.to\_sql(name='episode', con=engine, if\_exists='append', index=False)

#insert Value --------- profession table

profession.to\_sql(name='profession', con=engine, if\_exists='append', index=False)

#insert Value --------- rating table

df\_rating.to\_sql(name='rating',con = engine, if\_exists = 'append', index = False)

#insert Value --------- name\_profession table

name\_profession.to\_sql(name='name\_profession', con=engine, if\_exists='append', index=False)

#insert Value --------- types table

# insert into type

temp\_type\_df.to\_sql(name='types', con=engine, if\_exists='append', index=False)

#insert Value --------- type\_akas table

#insert type\_akas

type\_akas.to\_sql(name='type\_akas', con=engine, if\_exists='append', index=False)

#insert Value --------- knownForTitles table

knownForTitles.to\_sql(name='knownForTitles', con=engine, if\_exists='append', index=False)

#insert Value --------- directors table

# directors table

director\_df = df\_director[['tconst','nconst']]

# insert into directors

director\_df.to\_sql('directors', con=engine, if\_exists='append', index=False)

#insert Value --------- writers table

# writers table

writer\_df = df\_writer[['tconst','nconst']]

# insert into writers

writer\_df.to\_sql('writers', con=engine, if\_exists='append', index=False)

**Appendix II** (Code for 10 Analytical Procedures Ihat Led to Valuable Insights)

**What is the most successful format of title in recent two years?**

select title\_type, count (title\_type) from

(select \* ,rank () over (order by averagerating DESC, numvotes DESC) from title\_genres

left join genres on genres.genre\_id = title\_genres.genre\_id

left join rating on rating.tconst = title\_genres.tconst

left join title on title.tconst = title\_genres.tconst

where averagerating is not null

and numvotes is not null

and start\_year > 2020) as x

where x.rank <100

group by title\_type

**What genres of movies are more popular?**

select genre\_name , count(genres.genre\_name) from title\_genres

left join genres on genres.genre\_id = title\_genres.genre\_id

left join rating on rating.tconst = title\_genres.tconst

left join title on title.tconst = title\_genres.tconst

where numvotes > 500000

group by genres.genre\_name

ORDER BY count DESC;

**The most popular fillm genre of the 10 years**

select genre\_name , count(genres.genre\_name) from title\_genres

left join genres on genres.genre\_id = title\_genres.genre\_id

left join rating on rating.tconst = title\_genres.tconst

left join title on title.tconst = title\_genres.tconst

where numvotes > 500000

and start\_year >= 2012

group by genres.genre\_name

ORDER BY count DESC;

**Top 20 Horror or Thriller Film Directors**

select primaryname , count from (

select primaryname , count(\*) as count, rank () over (order by count(\*) DESC) as rank from title\_genres

left join genres on genres.genre\_id = title\_genres.genre\_id

left join rating on rating.tconst = title\_genres.tconst

left join directors on directors.tconst = rating.tconst

left join crews on crews.nconst = directors.nconst

where genres.genre\_name in ('Horror','Thriller')

and averagerating >= 8.5

and numvotes > 10000

group by primaryname

order by COUNT DESC) as x

where x.rank > 1

and x.rank <= 21

**Top 20 the Busiest Horror or Thriller Film Writers in 2022**

select primaryname , count from (

select primaryname , count(\*) as count, rank () over (order by count(\*) DESC) as rank from title\_genres

left join genres on genres.genre\_id = title\_genres.genre\_id

left join writers on writers.tconst = title\_genres.tconst

left join crews on crews.nconst = writers.nconst

left join title on title.tconst = writers.tconst

where genres.genre\_name in ('Horror','Thriller')

and start\_year > 2021

group by primaryname

order by COUNT DESC) as x

where x.rank <= 20

**Top 20 Horror or Thriller Film Writers**

select primaryname , count from (

select primaryname , count(\*) as count, rank () over (order by count(\*) DESC) as rank from title\_genres

left join genres on genres.genre\_id = title\_genres.genre\_id

left join rating on rating.tconst = title\_genres.tconst

left join writers on writers.tconst = rating.tconst

left join crews on crews.nconst = writers.nconst

where genres.genre\_name in ('Horror','Thriller')

and averagerating >= 8.5

and numvotes >10000

group by primaryname

order by COUNT DESC) as x

where x.rank <= 20

**Top 20 Horror or Thriller Film Actress**

select primaryname , count from (

select primaryname , count(\*) as count, rank () over (order by count(\*) DESC) as rank from title\_genres

left join genres on genres.genre\_id = title\_genres.genre\_id

left join rating on rating.tconst = title\_genres.tconst

left join principal on principal.tconst = rating.tconst

left join crews on crews.nconst = principal.nconst

where genres.genre\_name in ('Horror','Thriller')

and averagerating >= 8.5

and numvotes > 10000

and job\_category = 'actress'

group by primaryname

order by COUNT DESC) as x

where x.rank <= 20

**Top 20 Horror or Thriller Film Actor**

select primaryname , count from (

select primaryname , count(\*) as count, rank () over (order by count(\*) DESC) as rank from title\_genres

left join genres on genres.genre\_id = title\_genres.genre\_id

left join rating on rating.tconst = title\_genres.tconst

left join principal on principal.tconst = rating.tconst

left join crews on crews.nconst = principal.nconst

where genres.genre\_name in ('Horror','Thriller')

and averagerating >= 8.5

and numvotes >10000

and job\_category = 'actor'

group by primaryname

order by COUNT DESC) as x

where x.rank <= 20

**Rank the best director’s films**

select \* , rank () over (order by averagerating DESC, numvotes DESC) as rank from principal

left join rating on rating.tconst = principal.tconst

left join crews on crews.nconst = principal.nconst

where primaryname = 'Michelle MacLaren'

and averagerating is not null

**Method 2 the best director’s Top 2 films Team Members**

select primaryname, job\_category from principal

left join crews on crews.nconst = principal.nconst

where tconst in ('tt2301449')

order by job\_category

select primaryname, job\_category from principal

left join crews on crews.nconst = principal.nconst

where tconst in ('tt2301449')

order by job\_category