Movies and GDP

Nullbusters
Oscar Ortiz, Rissa Jackson

Problem

Is a movie's budget, gross income, and ratings correlated to the economic health (through gross domestic product) of the country the movie was produced in?

Does a country's population affect the number of movies produced in that country and, therefore, their worldwide box office revenue?

Datasets

imdb

Movies
- imdb_title_id
- title
- year
- budget
- worlwide_gross_income

Ratings
- imdb_title_id
- weighted_average_vote
- mean_vote
- us_voters_rating
- non_us_voters_rating

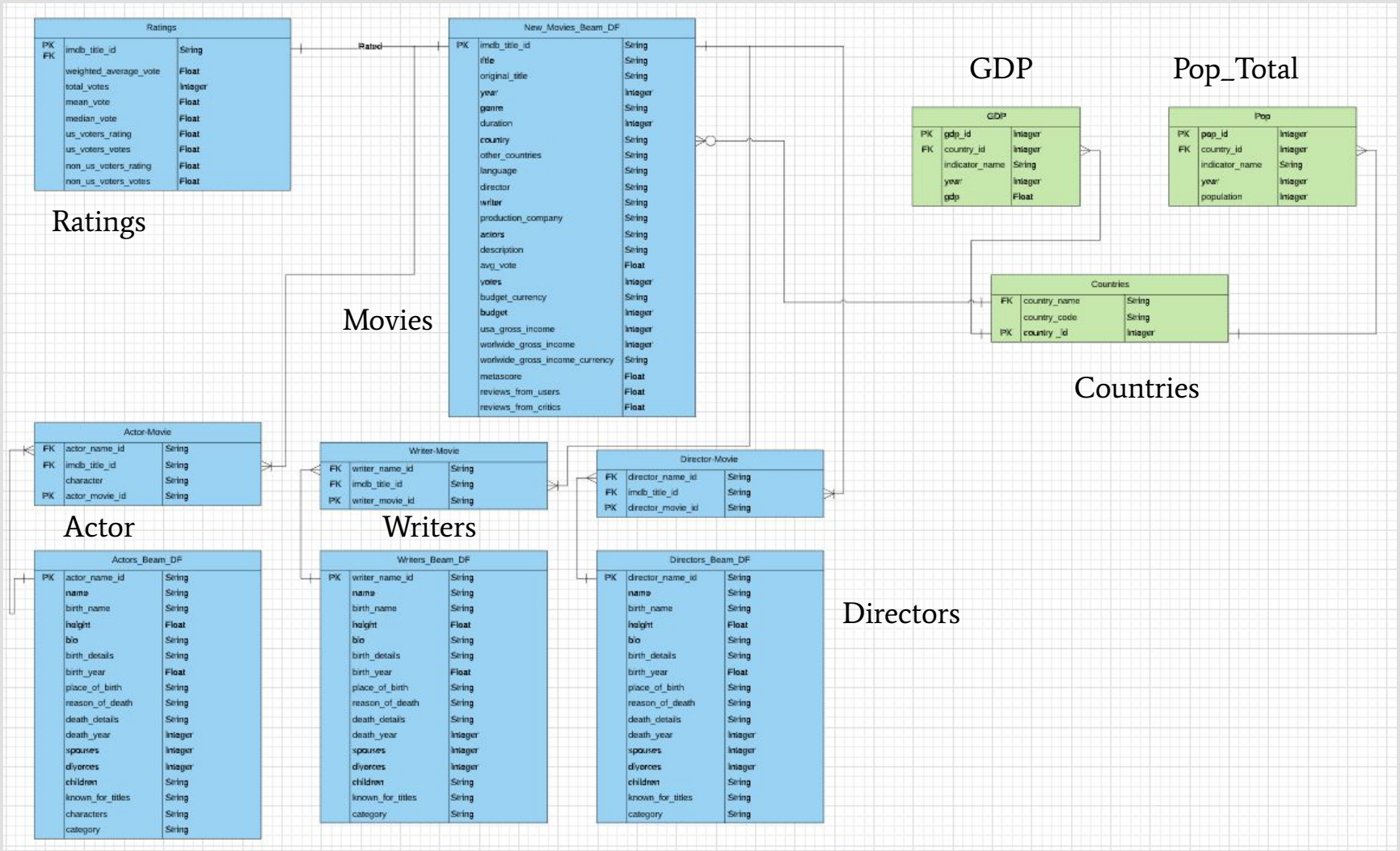
Names
- imdb_name_id
- name
- primary_profession
- known_for_titles
- children

Title_Principals
- imdb_title_id
- imdb_name_id
- category
- job
- characters

worldbank

GDP
- Country_Name
- Country_Code
- Indicator_Name
- Indicator_Code
- Year (1960 - 2018)

Population
- Country_Name
- Country_Code
- Indicator_Name
- Indicator_Code
- Year (1960 - 2018)


Ratings
PK imdb_title_id String
FK
weighted_average_vote Float
total_votes Integer
mean_vote Float
median_vote Float
us_voters_rating Float
us_voters_votes Float
non_us_voters_rating Float
non_us_voters_votes Float
Rated
New_Movies_Beam_DF
PK imdb_title_id String
title String
original_title String
year Integer
genre String
duration Integer
country String
other_countries String
language String
director String
writer String
production_company String
actors String
description String
avg_vote Float
votes Integer
budget_currency String
budget Integer
usa_gross_income Integer
worlwide_gross_income Integer
worlwide_gross_income_currency String
metascore Float
reviews_from_users Float
reviews_from_critics Float
GDP
Pop_Total
GDP
PK gdp_id Integer
FK country_id Integer
indicator_name String
year Integer
gdp Float
Pop
PK pop_id Integer
FK country_id Integer
indicator_name String
year Integer
population Integer
Countries
FK country_name String
country_code String
PK country_id Integer
Movies
Countries
Actor-Movie
FK actor_name_id String
FK imdb_title_id String
character String
PK actor_movie_id String
Writer-Movie
FK writer_name_id String
FK imdb_title_id String
PK writer_movie_id String
Director-Movie
FK director_name_id String
FK imdb_title_id String
PK director_movie_id String
Actor
Writers
Directors
Actors_Beam_DF
PK actor_name_id String
name String
birth_name String
height Float
bio String
birth_details String
birth_year Float
place_of_birth String
reason_of_death String
death_details String
death_year Integer
spouses Integer
divorces Integer
children String
known_for_titles String
characters String
category String
Writers_Beam_DF
PK writer_name_id String
name String
birth_name String
height Float
bio String
birth_details String
birth_year Float
place_of_birth String
reason_of_death String
death_details String
death_year Integer
spouses Integer
divorces Integer
children String
known_for_titles String
category String
Directors_Beam_DF
PK director_name_id String
name String
birth_name String
height Float
bio String
birth_details String
birth_year Float
place_of_birth String
reason_of_death String
death_details String
death_year Integer
spouses Integer
divorces Integer
children String
known_for_titles String
category String

Beam Pipelines

```
# splitting budget into currency and budget columns
if (budget != None): # check if NULL
    money = str(budget)
    bg_curr, bg_budget = money.split(" ")
    bg_budget = int(bg_budget)
else:
    bg_curr = None
    bg_budget = budget

# removing currency symbol from usa_gross_income
if (usa_gross_income != None): # check if NULL
    money = str(usa_gross_income)
    usa_currency, income = money.split(" ")
    new_usa_income = int(income) # new income as integer
else:
    new_usa_income = usa_gross_income

# splitting worlwide_gross_income into currency and income
if (worlwide_gross_income != None): # check if NULL
    money = str(worlwide_gross_income)
    data = money.split(" ")
    worl_currency = data[0]
    new_worl_income = data[1]
else:
    new_worl_income = new_usa_income
    worl_currency = None
```

```
# splitting country into two columns
if (country != None): # check if NULL
    country_list = country.split(", ")
    for i, country_type in enumerate(country_list):
        if "USA" == country_type:
            country_list[i] = "United States"
        if "UK" == country_type:
            country_list[i] = "United Kingdom"
    main_country = country_list.pop(0)
    other_countries = ", ".join(country_list) or None
else:
    main_country = country
    other_countries = None


# turn floats into ints and checking for NULL

if birth_year != None:
    new_birth = int(birth_year)
else:
    new_birth = birth_year

if death_year != None:
    new_death = int(death_year)
else:
    new_death = death_year
```

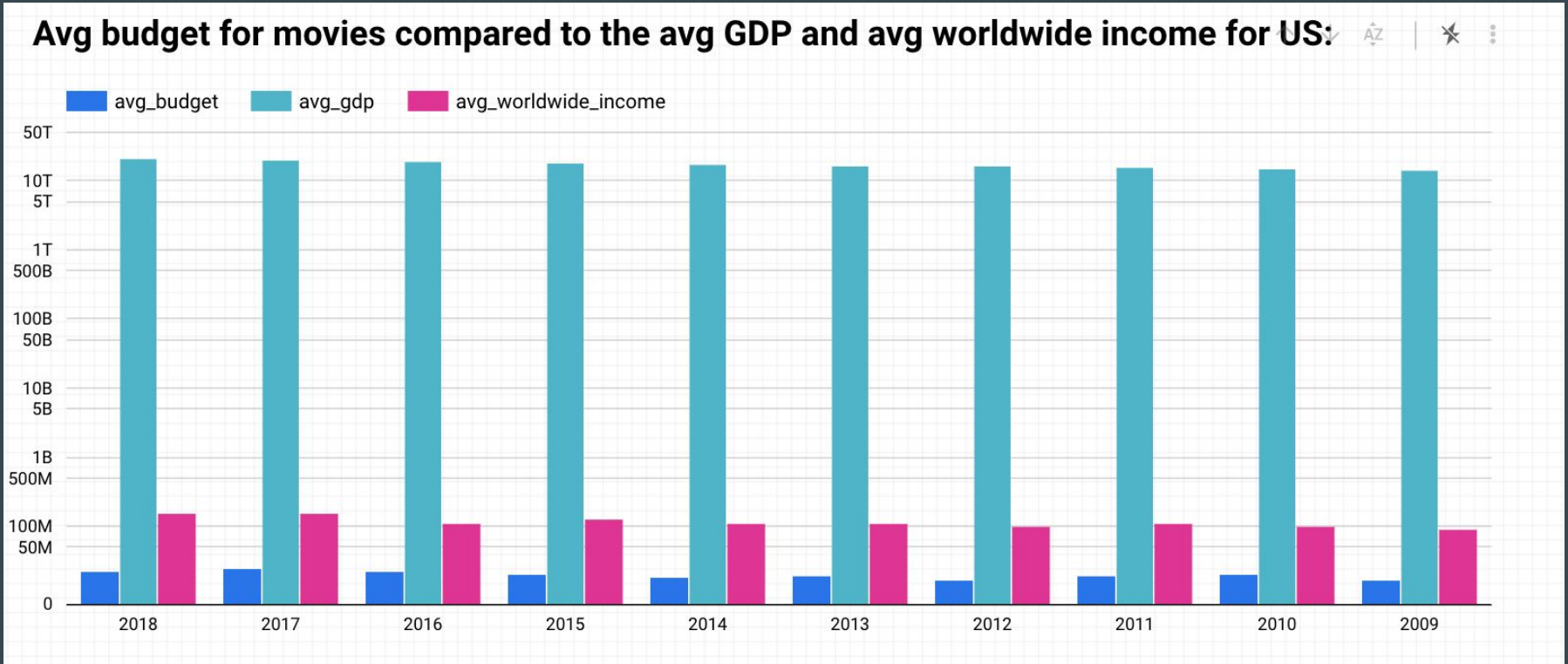
Cross-dataset Queries

```
%%bigquery
select avg(gdp) as avg_gdp, avg(budget) as avg_budget, avg(worlwide_gross_income) as avg_worldwide_income, m.year
from worldbank_modeled.GDP p join worldbank_modeled.Countries c using (country_id)
join imdb_modeled.New_Movies_Beam_DF m on (c.country_name = m.country)
where m.year = p.year and m.country = "United States" and budget is not null
group by m.year
order by m.year
```

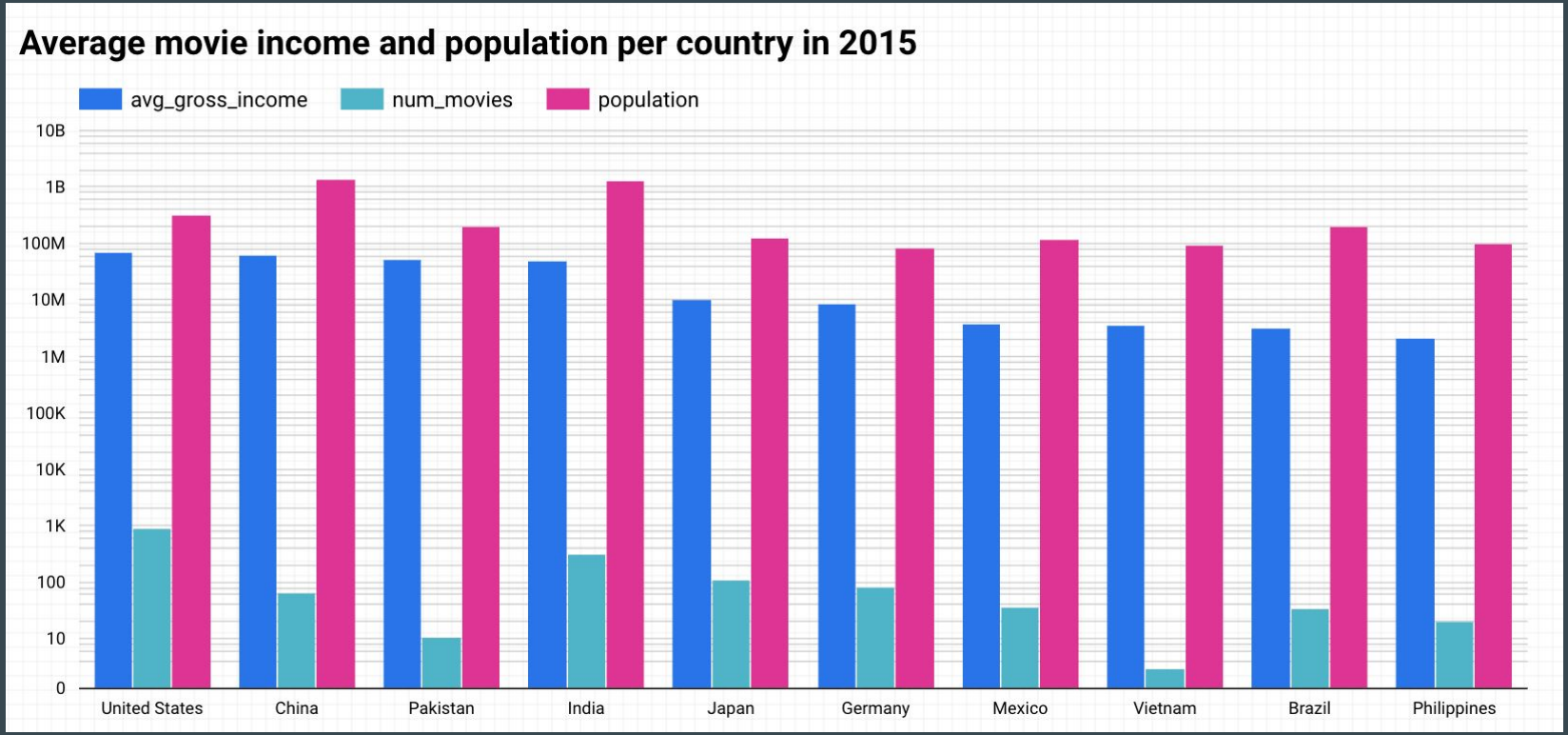
```
%%bigquery
select country, count(title) as num_movies, avg(population) as population, avg(worlwide_gross_income) as avg_gross_income
from imdb_modeled.New_Movies_Beam_DF m join worldbank_modeled.Countries c on (m.country = c.country_name)
join worldbank_modeled.Pop_Total p using (country_id)
where m.year = p.year and p.year = 2015
group by country
having avg_gross_income is not Null
order by population desc
limit 10
```

```
%%bigquery
select country, count(title) as num_movie, avg(gdp) as gdp, avg(weighted_average_vote) as avg_vote
from imdb_modeled.New_Movies_Beam_DF m join worldbank_modeled.Countries c on (m.country = c.country_name)
join worldbank_modeled.GDP g using (country_id) join imdb_modeled.Ratings using (imdb_title_id)
where m.year = g.year and g.year = 2015
group by country
order by gdp DESC
limit 10
```

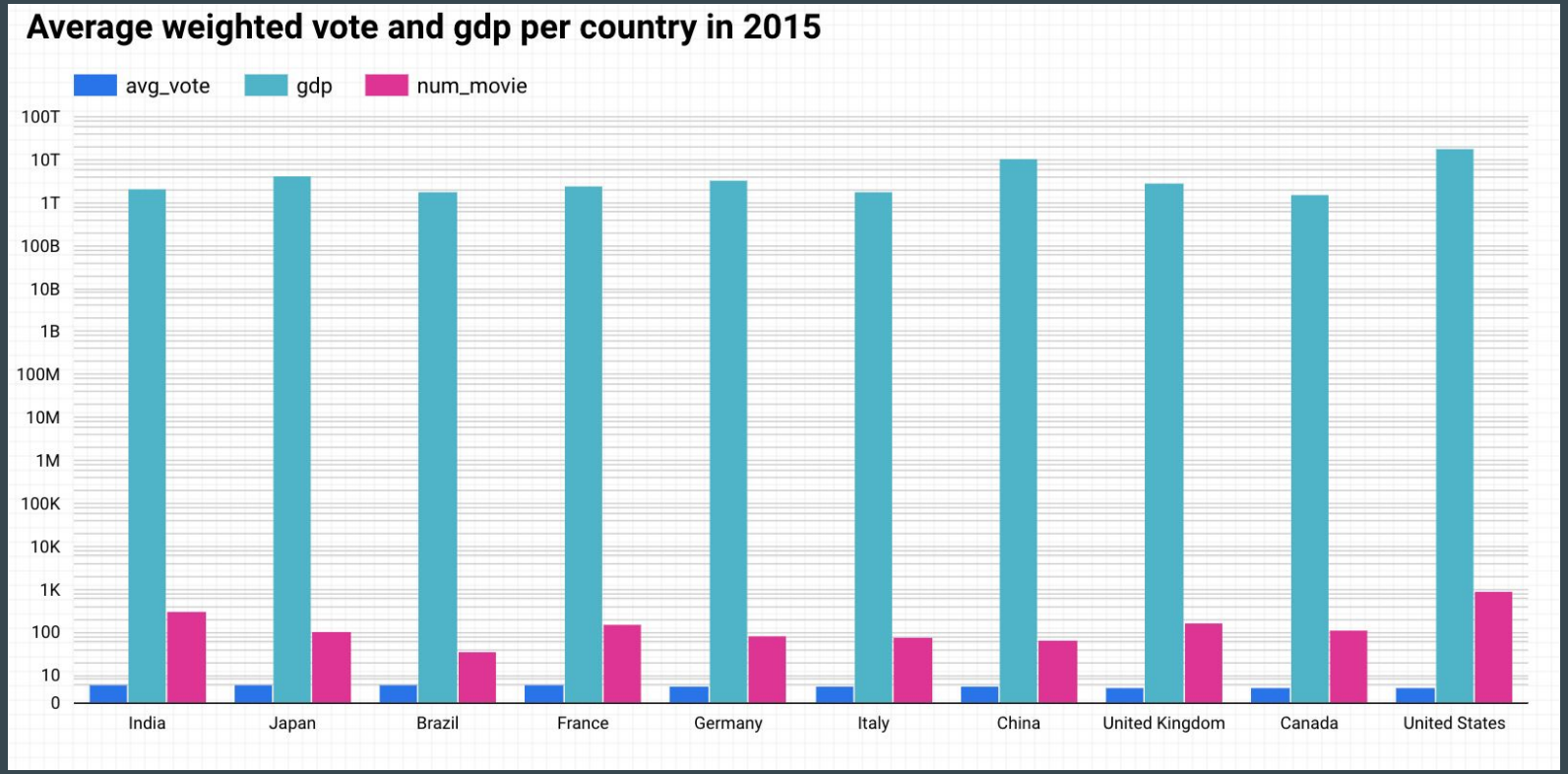
Visualizations



Visualizations



Visualizations



Workflow

```
staging_dataset = 'imdb_workflow_staging'
modeled_dataset = 'imdb_workflow_modeled'
orig_staging_dataset = 'imdb_staging'
movies_schema = '/home/jupyter/airflow/dags/movies_schema.json'
names_schema = '/home/jupyter/airflow/dags/names_schema.json'

bq_query_start = 'bq query --use_legacy_sql=false '

create_ratings = 'create or replace table ' + modeled_dataset + '''.Ratings as
                    select distinct *
                    from ''' + staging_dataset + '''.Ratings'''

create_movies = 'create or replace table ' + modeled_dataset + '''.Movies as
                    select distinct *
                    from ''' + staging_dataset + '''.Movies'''

create_actors = 'create or replace table ' + modeled_dataset + '''.Actors as
                    select distinct * except(primary_profession, job, ordering, actors, production_company,
place_of_death, reason_of_death, imdb_name_id, date_of_death), date_of_death as reason_of_death, generate_uuid() as
actor_name_id
                    from ''' + staging_dataset + '''.Names n inner join ''' + staging_dataset + '''.Title_Principals t
using (imdb_name_id) where t.category = "actor"'''

create_directors = 'create or replace table ' + modeled_dataset + '''.Directors as
                    select distinct * except(primary_profession, job, ordering, actors, production_company,
place_of_death, reason_of_death, imdb_name_id, date_of_death), date_of_death as reason_of_death, generate_uuid() as
director_name_id
                    from ''' + staging_dataset + '''.Names n inner join ''' + staging_dataset + '''.Title_Principals t
using (imdb_name_id) where t.category = "director"'''

create_writers = 'create or replace table ' + modeled_dataset + '''.Writers as
                    select distinct * except(primary_profession, job, ordering, actors, production_company,
place_of_death, reason_of_death, imdb_name_id, date_of_death), date_of_death as reason_of_death, generate_uuid() as
writer_name_id
                    from ''' + staging_dataset + '''.Names n inner join ''' + staging_dataset + '''.Title_Principals t
using (imdb_name_id) where t.category = "writer"'''
```

```
    create_directors_movie = BashOperator(
            task_id='create_directors_movie',
            bash_command=bq_query_start + "'" + create_directors_movie + "'",
            trigger_rule='one_success')

    dataflow_writers = BashOperator(
            task_id='dataflow_writers',
            bash_command='python /home/jupyter/airflow/dags/Writers_beam_dataflow.py')

    dataflow_actors = BashOperator(
            task_id='dataflow_actors',
            bash_command='python /home/jupyter/airflow/dags/Actors_beam_dataflow.py')

    dataflow_directors = BashOperator(
            task_id='dataflow_directors',
            bash_command='python /home/jupyter/airflow/dags/Directors_beam_dataflow.py')

    dataflow_movies = BashOperator(
            task_id='dataflow_movies',
            bash_command='python /home/jupyter/airflow/dags/Movies_beam_dataflow.py')

    dataflow_new_movies = BashOperator(
            task_id='dataflow_new_movies',
            bash_command='python
 /home/jupyter/airflow/dags/New_Movies_beam_dataflow.py')
```

```
    create_staging >> create_modeled >> branch
    branch >> load_movies >> create_movies >> dataflow_movies >>
dataflow_new_movies
    branch >> load_ratings >> create_ratings
    branch >> load_title_principals >> load_names >> create_actors >>
create_directors >> create_writers >> create_actors_movie >>
create_directors_movie >> create_writers_movie >> dataflow_directors >>
dataflow_writers >> dataflow_actors
```

Future Improvements

- The data is interesting but some of the comparisons involved numbers that were too far apart to make for interesting visualizations.
    - Could make separate chart for each attribute that was measured
- Could have separated the movie budgets and income into their own separate table, so the movies table would feel less cramped with the new currency columns.