What Types of Movies Have Been Popular Over the Past 100 Years? — A Bayesian Analysis

Final Report
Chenfei Wang (chenfeiw)
12/18/2024

1. Introduction

As the eighth art form, movies have existed for over 120 years since their birth in 1895. Over this time, movies have gone through much development, with various types of movies emerging and people's preferences for movie genres evolving alongside societal changes.

IMDb is a large database that includes movies, TV shows, dramas, family videos, video games, and online streaming media content information, such as cast, crew, personal biographies, plot summaries, reviews, and ratings. As of December 2024, the database contained some 22 million titles (including television episodes), 11.5 million person records, and 83 million registered users.

On the IMDb website, we can see trends in the popularity and ratings of different types of movies over the years. To some extent, this allows us to study how the audience's reception of different genres has evolved over time. This paper uses Bayesian methods to explore trends in movie genre popularity over time.

2. Dataset

2.1. Dataset Overview

We collected an IMDb dataset from the Kaggle website. The dataset includes information on the top 1,000 highest-grossing movies on IMDb, including name, release year, genre, gross revenue, and ratings. The dataset was last updated in 2024.

2.2. Data Cleaning

We performed data preprocessing to remove irrelevant variables and clean the dataset. We noticed that our goal was to explore the trend of changes in movie types, so the influence of individual movies was not great. Therefore, we directly deleted the rows with missing items. We noticed that some movies corresponded to more than one category, and the first category best summarized the characteristics of the movie. Therefore, we divided the movies according to the first category.

After data cleaning, there are 828 data entries left. We simply draw a line chart of the ratings and total box office of different types of movie over time, as shown in Figure 1 and Figure 2 at the end of the article. From the figure, we can see some obvious trends, such as action movies and animation having become more and more popular in the past 20 years, etc.

3. Method

To determine the popularity of movies, we selected two variables: rating and gross. We analyzed how these two variables changed over the years. As for the specific analysis method, we used two models: the first one is a simple Bayesian model, and the second one is a Bayesian hierarchical model, which is used to analyze the change trend of each specific category of movies. We wrote the code for the stan model and used the cmdstanpy library for analysis.

3.1. Analysis of Rating

First, we normalized the data (e.g., readjust the year to improve the numerical stability of Bayesian modeling). Then, we used a simple Bayesian model to analyze the rating variable. The specific model is as follows:

$$\text{Likelihood: } y_i \sim \mathcal{N}(\alpha + \beta x_i, \sigma) \tag{1}$$

$$\begin{aligned} \text{Prior: } \alpha &\sim \mathcal{N}(0,1), \\ \beta &\sim \mathcal{N}(0,1), \\ \sigma &\sim \mathcal{N}(0,1), \quad \sigma > 0 \end{aligned} \tag{2}$$

This representation indicates that through Bayesian inference, we can jointly consider the observed data $\{x_i, y_i\}$ and the prior knowledge of the parameters to infer the posterior distributions of the model parameters α, β, and σ.

We then used a Bayesian hierarchical model. We introduced independent intercepts and slopes for each genre, while setting global priors. The model structure is as follows:

$$\text{rating}_i \sim \mathcal{N}(\alpha_{\text{genre}[i]} + \beta_{\text{genre}[i]} \cdot \text{year}_{\text{scaled},i}, \sigma)$$

- α_{genre}: Intercept for each genre
- β_{genre}: Slope for each genre
- σ: Noise standard deviation (residual standard deviation)

Through this model, we can get the trend of the ratings of each type of movies changing with the year.

3.2. Analysis of Gross

When performing Bayesian modeling on gross variables, we adopted a similar approach as that of rating variables. We also established two models: a simple Bayesian model and a hierarchical model. The specific modeling process will not be repeated here.

4. Results

4.1. Result of Rating

For a simple Bayesian analysis of the rating variable, we obtain an intercept (alpha) with a mean rating of approximately 7.94, a slope (beta) with a mean of -0.052, and a 95% posterior confidence interval that is completely below zero (-0.066 to -0.034). This shows that the movie ratings show a significant downward trend with the increase of years. We can also see this trend from Figure 3. The residual standard deviation is about 0.28, indicating that the data has certain volatility, but the overall fitting effect of the model is relatively stable. The Rhat values of all parameters are close to 1.0 (the maximum is 1.001), indicating that the model has converged and the sampling quality is good. The effective sample size NEff of all parameters is high, which further illustrates the stability of the posterior distribution estimation.

For the hierarchical Bayes model of the rating variable, we obtain the intercept and slope for each movie genre (1-14), from which we can see that the genres with the most obvious drop in ratings are Crime ($\beta = 0.0710$) and Western ($\beta = 0.0724$). These two categories saw the largest declines in ratings over the years. The types with relatively stable scores are Action ($\beta = 0.0388$) and Animation ($\beta = 0.0397$), and the score trend decreases slowly. We found that the overall movie ratings gradually decreased over the years, and there were differences in the performance of different types of movies. Crime and Western genres saw the most significant declines in ratings. The Action and Animation types are relatively stable, with little change in their ratings.

4.2. Result of Gross

For a simple Bayesian analysis of the gross variable, we get an average box office of about 60.72, $\beta = 23.26$: the box office shows a significant growth trend with the increase of years. The 95% confidence interval is [17.95, 28.65], indicating that this trend is Statistically very significant. Overall trend: The box office shows a significant growth trend over the years, which shows that the overall level of movie box office continues to improve over time. We can also see this from the picture.

For the hierarchical Bayesian model of the gross variable, we obtain the intercept and slope corresponding to each movie type (1-14). The intercepts of different movie types fluctuate greatly, indicating that there are large differences in the average box office levels of each type. There are also obvious fluctuations in the slopes of different types of movies, showing the differences in box office trends between types. Specifically, both Action and Animation have high intercepts and slopes, indicating high box office levels and rapid growth. The growth rates of Comedy and Western are slower.

4.3. Model Checking and Evaluation

We performed a posterior prediction check on the results, as shown in Figure 7. We found that the hierarchical model is close to the simple model in the low box office area, but performs better in the high box office area and is closer to the real data distribution. In the medium and high box office area , the points of the hierarchical model are closer to y=x, indicating that the model prediction is more accurate. However, the hierarchical model is complex and may be sensitive to the data distribution, especially when there are fewer samples, it may be unstable.

5. Limitations

The following limitations are identified in the model:

1. **Inflation Adjustment**: The model does not account for inflation over time, which may lead to earlier movies having lower gross values and affect accuracy.

2. **Historical Context**: The model does not incorporate specific historical contexts, which could reveal changes in the popularity of certain movie genres over time.

3. **Number of Screenings**: Movies released in earlier decades typically had more screenings, while modern movies may rely on different distribution strategies, creating potential biases.

4. **Small Sample Sizes for Certain Genres**: For some genres with limited data, the Bayesian hierarchical model may overfit, leading to unreliable estimates.

6. Conclusion and Future Work

Overall, we used Bayesian inference to analyze the changes in the popularity of different types of movies over time. Bayesian analysis has a good application for this type of problem. In the future, I will continue to improve the model based on the limitations mentioned above.

References

[1] IMDb. (n.d.). *Press Room - Statistics*. Retrieved December 18, 2024, from `https://www.imdb.com/pressroom/stats`

[2] CmdStanPy Documentation. Retrieved December 18, 2024, from `https://mc-stan.org/cmdstanpy/`

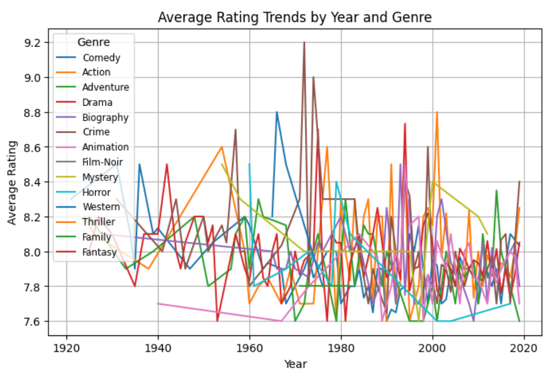


Figure 1: Average Rating Trends by Year and Genre.

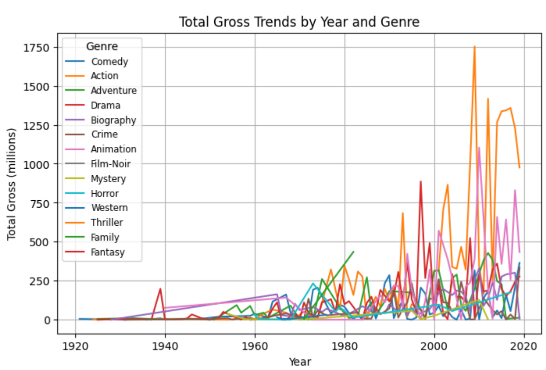


Figure 2: Total Gross Trends by Year and Genre.

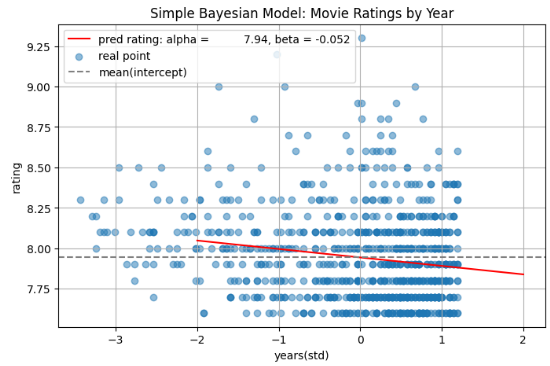


Figure 3: Simple Bayesian Model: Movie Ratings by Year

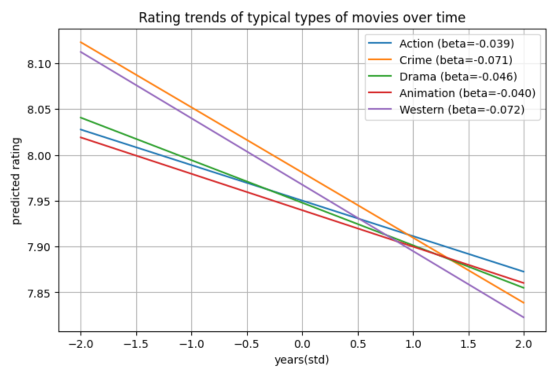


Figure 4: Rating trends of typical types of movies over time.

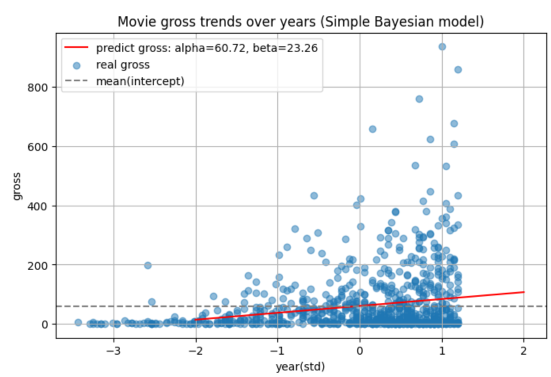


Figure 5: Movie gross trends over years (Simple Bayesian model)

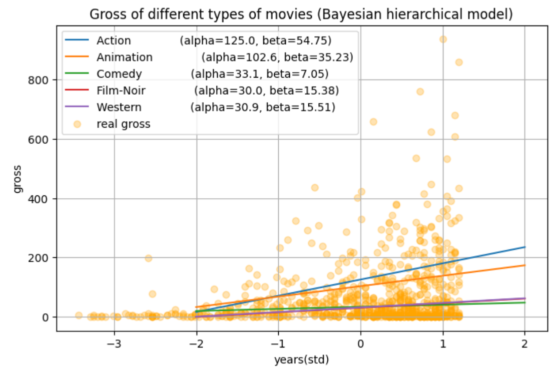


Figure 6: Gross of different types of movies (Bayesian hierarchical model)

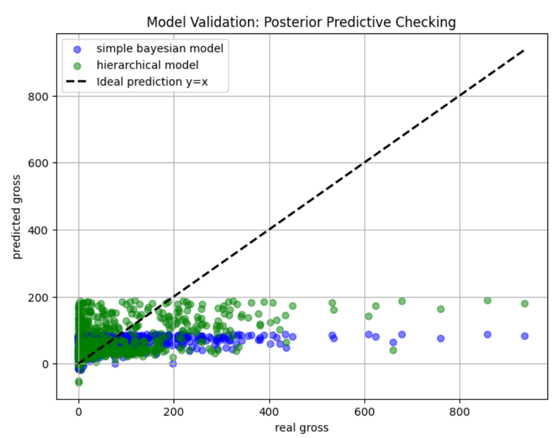


Figure 7: Model Validation: Posterior Predictive Checking

STATS 551 Final Project

Uniqname: chenfeiw

UMID: 73271527

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
from cmdstanpy import CmdStanModel
```

In [ ]:
```
file_path = 'IMDB_movie_reviews_details.csv'
data = pd.read_csv(file_path)
```

In [ ]:
```
data.head()
```

Out[ ]:

	Unnamed: 0	name	year	runtime	genre	rating	metascore	timeline
0	0	The Shawshank Redemption	1994	142	Drama	9.3	80.0	Two imprisoned men bond over a number of years...
1	1	The Godfather	1972	175	Crime, Drama	9.2	100.0	An organized crime dynasty's aging patriarch t...
2	2	Soorarai Pottru	2020	153	Drama	9.1	NaN	Nedumaaran Rajangam "Maara" sets out to make t...
3	3	The Dark Knight	2008	152	Action, Crime, Drama	9.0	84.0	When the menace known as the Joker wreaks havo...
4	4	The Godfather: Part II	1974	202	Crime, Drama	9.0	90.0	The early life and career of Vito Corleone in ...

Data Cleaning

In [ ]:
```
data_cleaned = data.drop(columns=["Unnamed: 0", "timeline"])
```

In [ ]:
```
data_cleaned['gross'] = data_cleaned['gross'].\
    str.replace('[\$,M]', '', regex=True).astype(float)
data_cleaned['metascore'] = data_cleaned['metascore'].\
    fillna(data_cleaned['metascore'].mean())
data_cleaned = data_cleaned.dropna(subset=['gross'])
data_cleaned['genre'] = data_cleaned['genre'].\
    fillna('Unknown')
```

In [ ]:
```
data_cleaned = data_cleaned.drop_duplicates()
```

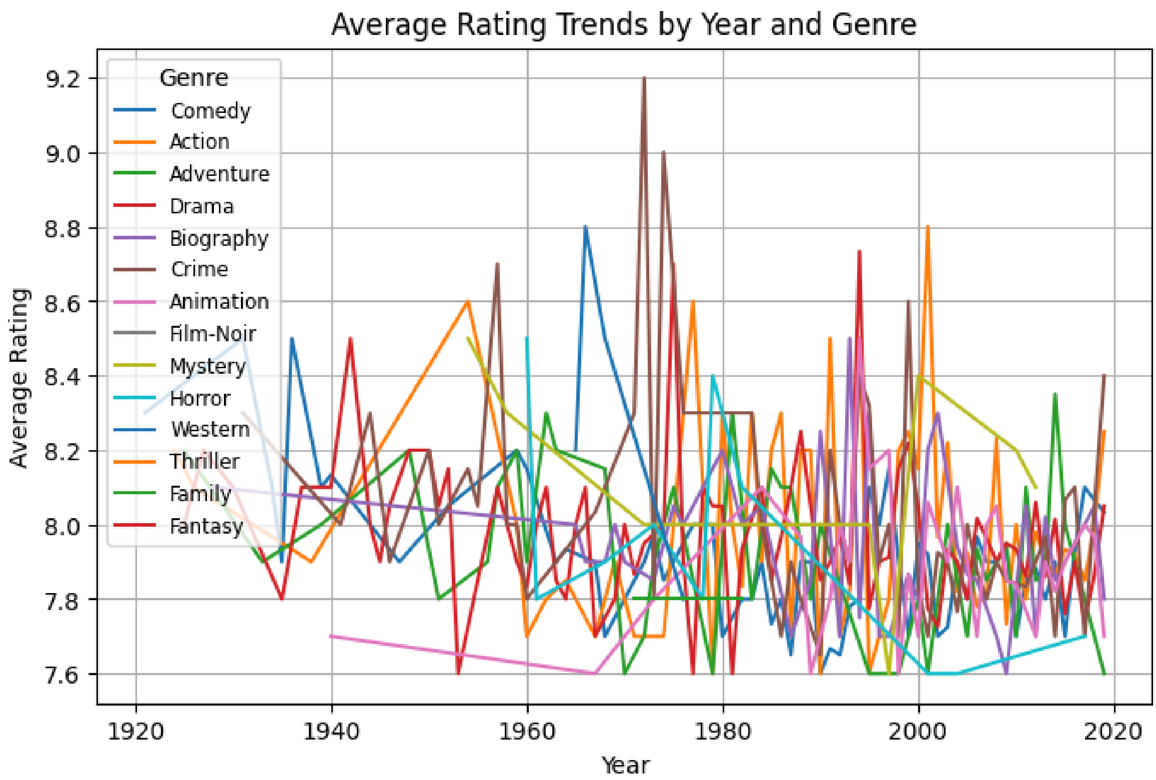
In [ ]:
```
data_cleaned['year'] = data_cleaned['year'].\
    str.extract('(\d{4})').astype(float)
data_cleaned = data_cleaned.dropna(subset=['year'])
data_cleaned['year'] = data_cleaned['year'].astype(int)
```

In [ ]:
```
data_expanded = data_cleaned.copy()
data_expanded = data_expanded.\
    assign(genre=data_cleaned['genre'].str.split(',')).explode('genre')
data_expanded['genre'] = data_expanded['genre'].str.strip()
```

In [ ]:
```
data_expanded = data_expanded.drop_duplicates(subset=['name'], keep='first')
```

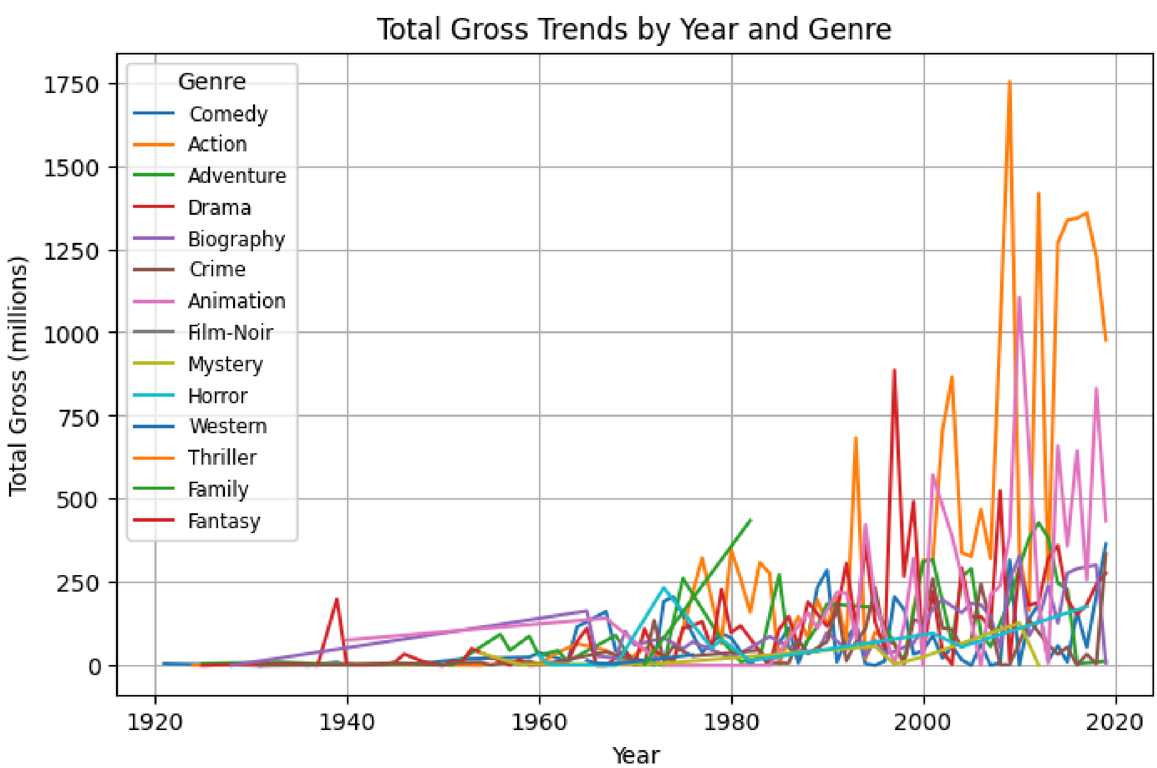
In [ ]:
```
year_genre = data_expanded.groupby(['year', 'genre']).agg({
    'rating': 'mean',
    'votes': 'sum',
    'gross': 'sum'
}).reset_index()
```

In [ ]:
```
plt.figure(figsize=(8, 5))
for genre in year_genre['genre'].unique():
    subset = year_genre[year_genre['genre'] == genre]
    plt.plot(subset['year'], subset['rating'], label=genre)

plt.title('Average Rating Trends by Year and Genre')
plt.xlabel('Year')
plt.ylabel('Average Rating')
plt.legend(loc='best', fontsize='small', title='Genre')
plt.grid(True)
plt.show()
```



In [ ]:

```
plt.figure(figsize=(8, 5))
for genre in year_genre['genre'].unique():
    subset = year_genre[year_genre['genre'] == genre]
    plt.plot(subset['year'], subset['gross'], label=genre)

plt.title('Total Gross Trends by Year and Genre')
plt.xlabel('Year')
plt.ylabel('Total Gross (millions)')
plt.legend(loc='best', fontsize='small', title='Genre')
plt.grid(True)
plt.show()
```



As can be seen from the figure, the overall box office of movies shows an upward trend as the year increases.

In [ ]:
```
data_simple = data_expanded[['year', 'genre', 'rating', 'gross']]
```

In [ ]:
```
data_simple.to_csv("movie_analyse.csv", index=False)
```

Bayesian Inference

About Rating

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
scaler = StandardScaler()
data_simple['year_scaled'] = scaler.fit_transform(data_simple[['year']])
genre_codes = pd.Categorical(data_simple['genre']).codes + 1

stan_data = {
    'N': len(data_simple),
    'J': 14,
    'K': 1,
    'x': data_simple['year_scaled'].values,
    'y': data_simple['rating'].values,
    'genre': pd.Categorical(data_simple['genre']).codes + 1
}
```

```
C:\Users\84207\AppData\Local\Temp\ipykernel_74180\2225896691.py:2: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_simple['year_scaled'] = scaler.fit_transform(data_simple[['year']])
```

simple bayesian model

In [ ]:
```
stan_model_code = """
data {
  int<lower=1> N;
  vector[N] x;
  vector[N] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
  alpha ~ normal(0, 1);
  beta ~ normal(0, 1);
  sigma ~ normal(0, 1);

  y ~ normal(alpha + beta * x, sigma);
```

```
}
"""
```

In [ ]:
```
stan_model_file = 'movie_rating_model.stan'
with open(stan_model_file, 'w') as file:
    file.write(stan_model_code)
```

In [ ]:
```
stan_model = CmdStanModel(stan_file="movie_rating_model.stan")
```

```
17:11:06 - cmdstanpy - INFO - compiling stan file C:\Users\84207\Desktop\SI618\si
618fa23-student-main\inclass\movie_rating_model.stan to exe file C:\Users\84207\D
esktop\SI618\si618fa23-student-main\inclass\movie_rating_model.exe
17:12:12 - cmdstanpy - INFO - compiled model executable: C:\Users\84207\Desktop\S
I618\si618fa23-student-main\inclass\movie_rating_model.exe
```

In [ ]:
```
stan_data = {
    'N': len(data_simple),
    'x': data_simple['year_scaled'].values,
    'y': data_simple['rating'].values
}
```

In [ ]:
```
fit = stan_model.sample(data=stan_data, chains=4, \
                        iter_sampling=1000, iter_warmup=500)
```

```
17:16:45 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status
```

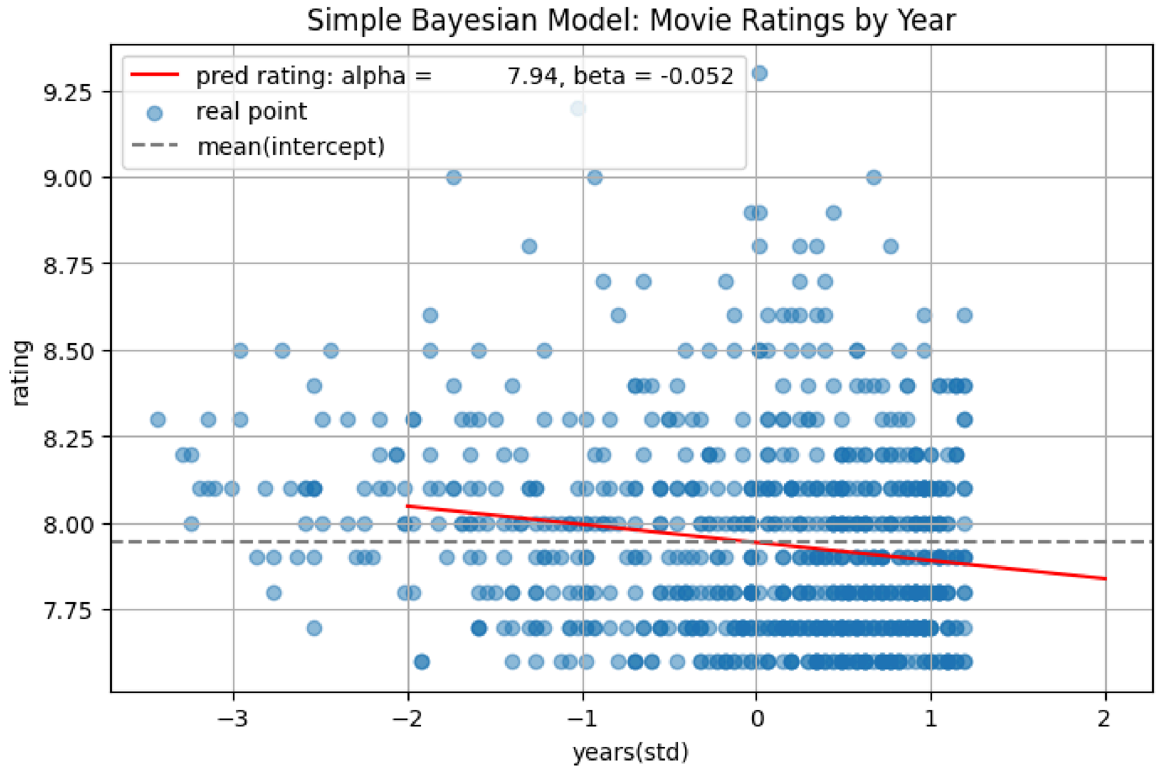
```
17:16:45 - cmdstanpy - INFO - CmdStan done processing.
17:16:45 - cmdstanpy - WARNING - Non-fatal error during sampling:
Exception: normal_lpdf: Scale parameter is 0, but must be positive! (in 'movie_ra
ting_model.stan', line 19, column 2 to column 38)
Exception: normal_lpdf: Scale parameter is 0, but must be positive! (in 'movie_ra
ting_model.stan', line 19, column 2 to column 38)
        Exception: normal_lpdf: Scale parameter is 0, but must be positive! (in
'movie_rating_model.stan', line 19, column 2 to column 38)
Consider re-running with show_console=True if the above output is unclear!
```

In [ ]:
```
print(fit.summary())
```

```
             Mean      MCSE    StdDev          5%         50%         95%  \
lp__   613.945000  0.025513  1.198510  611.572000  614.259000  615.236000
alpha    7.948390  0.000140  0.009342    7.933350    7.948260    7.963860
beta    -0.050258  0.000141  0.009852   -0.066835   -0.050332   -0.034069
sigma    0.278008  0.000122  0.006810    0.266825    0.277926    0.289453

         N_Eff   N_Eff/s     R_hat
lp__   2206.81   7689.22  1.001750
alpha  4478.91  15606.00  1.000530
beta   4850.38  16900.30  0.999851
sigma  3116.11  10857.50  0.999556
```

In [ ]:
```
alpha_mean = 7.9439
beta_mean = -0.052258
x = np.linspace(-2, 2, 100)
y_pred = alpha_mean + beta_mean * x
```

```
plt.figure(figsize=(8, 5))
plt.plot(x, y_pred, label=f'pred rating: alpha = \
         {alpha_mean:.2f}, beta = {beta_mean:.3f}', color='red')
plt.scatter(data_simple['year_scaled'], data_simple['rating'], \
            alpha=0.5, label='real point')
plt.axhline(alpha_mean, linestyle='--', color='gray', \
            label='mean(intercept)')
plt.title('Simple Bayesian Model: Movie Ratings by Year')
plt.xlabel('years(std)')
plt.ylabel('rating')
plt.legend()
plt.grid()
plt.show()
```



As the years go by, the ratings of movies show a slight downward trend.

Bayesian Hierarchical Model

In [ ]:
```
stan_model_hier = """
data {
  int<lower=1> N;
  int<lower=1> J;
  array[N] int<lower=1, upper=J> genre;
  vector[N] x;
  vector[N] y;
}
parameters {
  vector[J] alpha;
  vector[J] beta;
  real mu_alpha;
  real mu_beta;
  real<lower=0> sigma_alpha;
  real<lower=0> sigma_beta;
  real<lower=0> sigma;
}
```

```
model {
  mu_alpha ~ normal(0, 1);
  mu_beta ~ normal(0, 1);
  sigma_alpha ~ normal(0, 1);
  sigma_beta ~ normal(0, 1);
  sigma ~ normal(0, 1);

  alpha ~ normal(mu_alpha, sigma_alpha);
  beta ~ normal(mu_beta, sigma_beta);

  for (n in 1:N)
    y[n] ~ normal(alpha[genre[n]] + beta[genre[n]] * x[n], sigma);
}
"""
```

In [ ]:
```
stan_file_hier = "genre_rating_hierarchical.stan"
with open(stan_file_hier, "w") as f:
    f.write(stan_model_hier)
```

In [ ]:
```
stan_data = {
    'N': len(data_simple),
    'J': len(set(genre_codes)),
    'genre': genre_codes,
    'x': data_simple['year_scaled'].values,
    'y': data_simple['rating'].values
}
```

In [ ]:
```
stan_model = CmdStanModel(stan_file="genre_rating_hierarchical.stan")
```

```
17:35:27 - cmdstanpy - INFO - compiling stan file C:\Users\84207\Desktop\SI618\si
618fa23-student-main\inclass\genre_rating_hierarchical.stan to exe file C:\Users
\84207\Desktop\SI618\si618fa23-student-main\inclass\genre_rating_hierarchical.exe
17:35:43 - cmdstanpy - INFO - compiled model executable: C:\Users\84207\Desktop\S
I618\si618fa23-student-main\inclass\genre_rating_hierarchical.exe
```

In [ ]:
```
fit = stan_model.sample(data=stan_data, chains=4, \
                        iter_sampling=1000, iter_warmup=500)
```

```
17:35:44 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status
```

```
17:35:47 - cmdstanpy - INFO - CmdStan done processing.
17:35:47 - cmdstanpy - WARNING - Non-fatal error during sampling:
Exception: normal_lpdf: Scale parameter is 0, but must be positive! (in 'genre_ra
ting_hierarchical.stan', line 26, column 2 to column 37)
        Exception: normal_lpdf: Scale parameter is 0, but must be positive! (in
'genre_rating_hierarchical.stan', line 29, column 4 to column 66)
Exception: normal_lpdf: Scale parameter is 0, but must be positive! (in 'genre_ra
ting_hierarchical.stan', line 25, column 2 to column 40)
Exception: normal_lpdf: Scale parameter is 0, but must be positive! (in 'genre_ra
ting_hierarchical.stan', line 26, column 2 to column 37)
Consider re-running with show_console=True if the above output is unclear!
17:35:47 - cmdstanpy - WARNING - Some chains may have failed to converge.
        Chain 1 had 13 divergent transitions (1.3%)
        Chain 2 had 45 divergent transitions (4.5%)
        Chain 3 had 11 divergent transitions (1.1%)
        Chain 4 had 218 divergent transitions (21.8%)
        Use the "diagnose()" method on the CmdStanMCMC object to see further info
rmation.
```

In [ ]:

```
category_mapping = pd.Categorical(data_simple['genre']).categories
for code, category in enumerate(category_mapping):
    print(f"{code+1} -> {category}")
```

```
1 -> Action
2 -> Adventure
3 -> Animation
4 -> Biography
5 -> Comedy
6 -> Crime
7 -> Drama
8 -> Family
9 -> Fantasy
10 -> Film-Noir
11 -> Horror
12 -> Mystery
13 -> Thriller
14 -> Western
```

In [ ]:

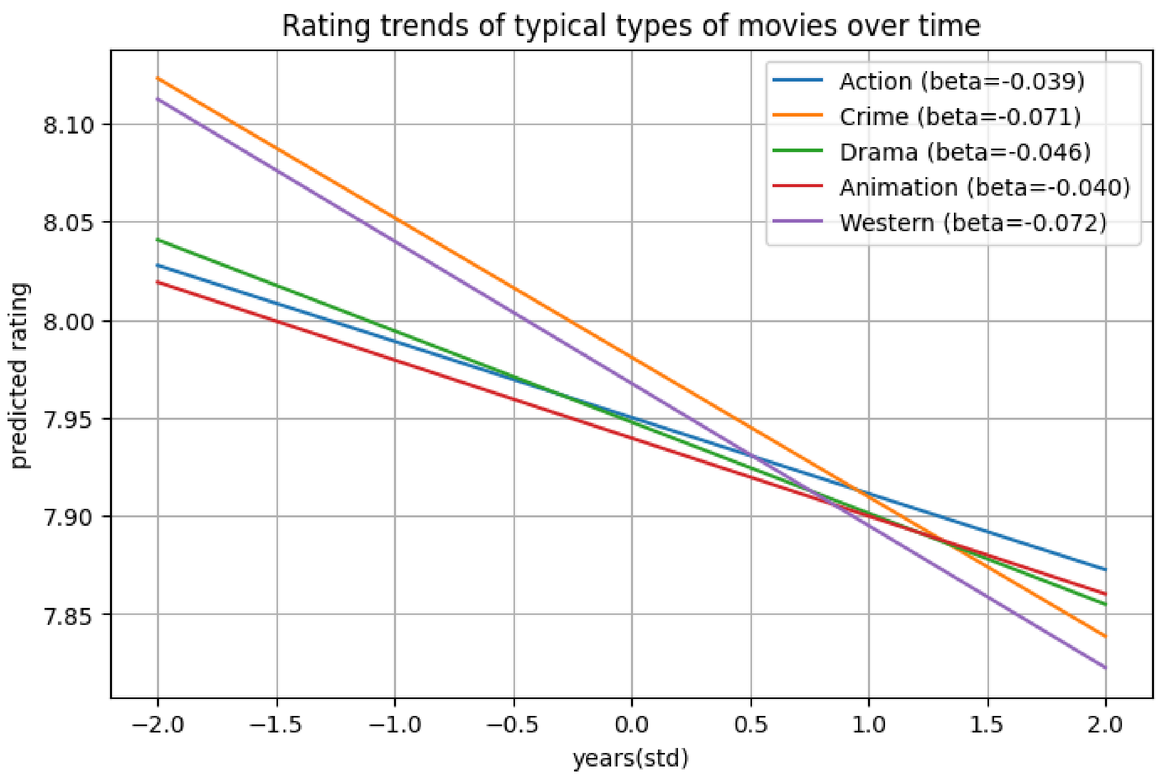
```
print(fit.summary())
```

```
                    Mean      MCSE     StdDev         5%         50%  \
lp__          702.215000  1.801650  12.290900  683.787000  700.828000
alpha[1]        7.950140  0.000771   0.019822    7.919230    7.949190
alpha[2]        7.938030  0.001231   0.026360    7.895360    7.938680
alpha[3]        7.939650  0.003211   0.027627    7.889420    7.941300
alpha[4]        7.949230  0.000595   0.022483    7.912790    7.949500
alpha[5]        7.915060  0.001273   0.023884    7.873270    7.917470
alpha[6]        7.980830  0.000914   0.026074    7.942180    7.977590
alpha[7]        7.947810  0.001955   0.016884    7.917690    7.948170
alpha[8]        7.938550  0.001086   0.045458    7.863800    7.940190
alpha[9]        7.940250  0.001108   0.043701    7.863660    7.943180
alpha[10]       7.946360  0.001340   0.044788    7.880810    7.945100
alpha[11]       7.935620  0.001662   0.038274    7.870060    7.938370
alpha[12]       7.966240  0.003956   0.044139    7.906510    7.961090
alpha[13]       7.940380  0.002227   0.045114    7.869690    7.941960
alpha[14]       7.967530  0.002512   0.048542    7.910370    7.957890
beta[1]        -0.038750  0.000634   0.020012   -0.067752   -0.040513
beta[2]        -0.052514  0.000380   0.019701   -0.085394   -0.051452
beta[3]        -0.039697  0.000811   0.026579   -0.077116   -0.042829
beta[4]        -0.048318  0.000429   0.022503   -0.085331   -0.047675
beta[5]        -0.054643  0.000511   0.016878   -0.084562   -0.053539
beta[6]        -0.071053  0.002173   0.024489   -0.116329   -0.067587
beta[7]        -0.046411  0.000335   0.014065   -0.069401   -0.045853
beta[8]        -0.051116  0.000842   0.035857   -0.106417   -0.050094
beta[9]        -0.053965  0.000790   0.036494   -0.110833   -0.052044
beta[10]       -0.056040  0.000754   0.034099   -0.114361   -0.051868
beta[11]       -0.062488  0.001447   0.032221   -0.123785   -0.056603
beta[12]       -0.061644  0.001383   0.034793   -0.128576   -0.056156
beta[13]       -0.050133  0.000654   0.034739   -0.103402   -0.049483
beta[14]       -0.072390  0.002665   0.043243   -0.154264   -0.061539
mu_alpha        7.945850  0.001500   0.018980    7.918990    7.945840
mu_beta        -0.054300  0.000590   0.016510   -0.081790   -0.052870
sigma_alpha     0.036230  0.001130   0.021970    0.010230    0.031170
sigma_beta      0.025710  0.002220   0.020550    0.003100    0.020680
sigma           0.276410  0.000110   0.006470    0.266070    0.276040

                     95%       N_Eff     N_Eff/s     R_hat
lp__          724.701000    46.54030    8.49896  1.109960
alpha[1]        7.984660   660.38000  120.59500  1.011780
alpha[2]        7.980620   458.69600   83.76490  1.016130
alpha[3]        7.982530    74.03550   13.52000  1.067970
alpha[4]        7.986370  1425.63000  260.34100  1.003950
alpha[5]        7.950010   352.15500   64.30870  1.023790
alpha[6]        8.027870   814.11200  148.66900  1.008610
alpha[7]        7.974980    74.59340   13.62190  1.069140
alpha[8]        8.004250  1751.94000  319.93100  1.006210
alpha[9]        8.004180  1556.83000  284.30000  1.002870
alpha[10]       8.016950  1116.74000  203.93400  1.009960
alpha[11]       7.996280   530.16400   96.81600  1.010420
alpha[12]       8.047750   124.48500   22.73280  1.052530
alpha[13]       8.012710   410.29700   74.92630  1.015830
alpha[14]       8.059530   373.56700   68.21900  1.031740
beta[1]        -0.002367   997.75100  182.20400  1.003150
beta[2]        -0.020580  2689.26000  491.10000  1.004550
beta[3]         0.010986  1075.22000  196.35100  1.002740
beta[4]        -0.009790  2748.91000  501.99300  0.999928
beta[5]        -0.027711  1092.08000  199.43000  1.007780
beta[6]        -0.039493   127.01000   23.19400  1.027610
beta[7]        -0.022788  1758.35000  321.10200  1.001070
beta[8]         0.003752  1812.58000  331.00500  1.001860
```

```
beta[9]        0.000428  2133.77000  389.65800  1.001940
beta[10]      -0.005680  2044.40000  373.33800  1.001320
beta[11]      -0.021700   496.14700   90.60400  1.009870
beta[12]      -0.016576   632.96200  115.58800  1.004670
beta[13]       0.007115  2817.20000  514.46200  1.001220
beta[14]      -0.026411   263.24300   48.07220  1.013020
mu_alpha       7.975320   159.90183   29.20048  1.042910
mu_beta       -0.031270   787.36436  143.78458  1.008870
sigma_alpha    0.075990   378.91934   69.19637  1.028870
sigma_beta     0.066890    85.88172   15.68329  1.035870
sigma          0.287560  3633.44307  663.52138  1.000740
```

In [ ]:

```
selected_genres = {"Action": 1, "Crime": 6, "Drama": \
                   7, "Animation": 3, "Western": 14}
selected_params = {
    genre: {"alpha": fit.stan_variable('alpha')[:, index - 1].mean(),
            "beta": fit.stan_variable('beta')[:, index - 1].mean()}
    for genre, index in selected_genres.items()
}

plt.figure(figsize=(8, 5))
x = np.linspace(-2, 2, 100)
for genre, params in selected_params.items():
    y_pred = params['alpha'] + params['beta'] * x
    plt.plot(x, y_pred, label=f"{genre} (beta={params['beta']:.3f})")
plt.title("Rating trends of typical types of movies over time")
plt.xlabel("years(std)")
plt.ylabel("predicted rating")
plt.legend()
plt.grid()
plt.show()
```



The downward trend in the ratings of westerns and crime films is more obvious, which may be related to the fact that the "golden years" of these two types of movies have passed.

About Gross

simple bayesian model

In [ ]:
```
data_simple['year_scaled'] = scaler.fit_transform(data_simple[['year']])
```

```
C:\Users\84207\AppData\Local\Temp\ipykernel_74180\311116061.py:1: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_simple['year_scaled'] = scaler.fit_transform(data_simple[['year']])
```

In [ ]:
```
stan_data_gross = {
    'N': len(data_simple),
    'x': data_simple['year_scaled'].values,
    'y': data_simple['gross'].values
}
```

In [ ]:
```
simple_gross_model_code = """
data {
  int<lower=1> N;
  vector[N] x;
  vector[N] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
  alpha ~ normal(0, 10);
  beta ~ normal(0, 10);
  sigma ~ normal(0, 10);

  y ~ normal(alpha + beta * x, sigma);
}
"""
```

In [ ]:
```
simple_gross_model_file = "simple_gross_model.stan"
with open(simple_gross_model_file, "w") as f:
    f.write(simple_gross_model_code)
```

In [ ]:
```
simple_gross_model_file
```

Out[ ]:
```
'simple_gross_model.stan'
```

In [ ]:
```
simple_gross_model = CmdStanModel(stan_file='simple_gross_model.stan')
```

```
18:17:26 - cmdstanpy - INFO - compiling stan file C:\Users\84207\Desktop\SI618\si
618fa23-student-main\inclass\simple_gross_model.stan to exe file C:\Users\84207\D
esktop\SI618\si618fa23-student-main\inclass\simple_gross_model.exe
18:17:43 - cmdstanpy - INFO - compiled model executable: C:\Users\84207\Desktop\S
I618\si618fa23-student-main\inclass\simple_gross_model.exe
```

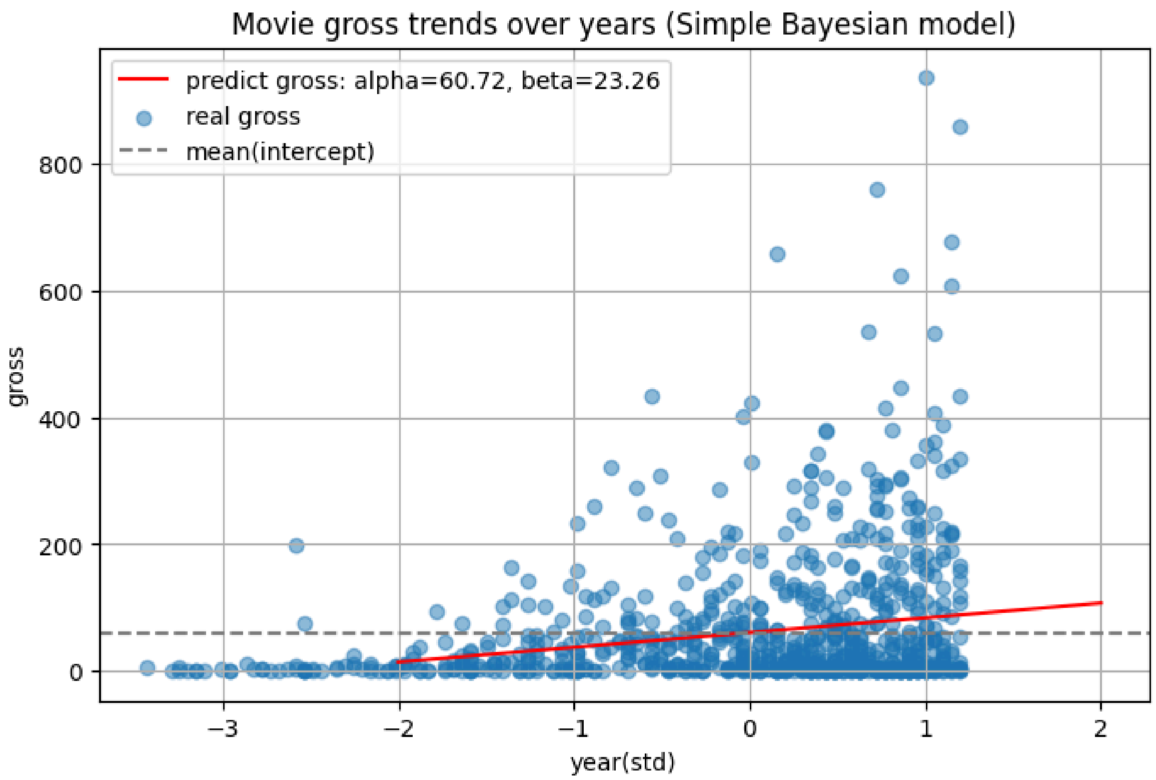
In [ ]:
```
fit_gross = simple_gross_model.sample(data=stan_data_gross, chains=4, \
                                      iter_sampling=1000, iter_warmup=500)
```

```
18:17:43 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status

18:17:44 - cmdstanpy - INFO - CmdStan done processing.
```

In [ ]:
```
print(fit_gross.summary())
```

```
            Mean      MCSE   StdDev         5%        50%        95%   N_Eff  \
lp__  -4355.7200  0.026192  1.19713 -4358.0000 -4355.4400 -4354.4200  2089.06
alpha    60.7234  0.054003  3.28721    55.2663    60.7039    66.0684  3705.27
beta     23.2628  0.051818  3.24083    17.9511    23.2288    28.6531  3911.64
sigma   101.2430  0.035294  2.24686    97.5857   101.2420   104.9990  4052.73

        N_Eff/s     R_hat
lp__    7487.67  1.001440
alpha  13280.50  1.000290
beta   14020.20  1.000260
sigma  14525.90  0.999667
```

In [ ]:
```
alpha_mean = 60.7234
beta_mean = 23.2628

x_years = np.linspace(-2, 2, 100)
y_pred = alpha_mean + beta_mean * x_years

plt.figure(figsize=(8, 5))
plt.plot(x_years, y_pred, color='red', label=f'predict gross: \
         alpha={alpha_mean:.2f}, beta={beta_mean:.2f}')
plt.scatter(data_simple['year_scaled'], data_simple['gross'], \
            alpha=0.5, label='real gross')
plt.title("Movie gross trends over years (Simple Bayesian model)")
plt.xlabel("year(std)")
plt.ylabel("gross")
plt.axhline(alpha_mean, linestyle='--', color='gray', \
            label='mean(intercept)')
plt.legend()
plt.grid()
plt.show()
```



Bayesian Hierarchical Model

In [ ]:
```
stan_gross_model_hier = """
data {
  int<lower=1> N;
  int<lower=1> J;
  array[N] int<lower=1, upper=J> genre;
  vector[N] x;
  vector[N] y;
}
parameters {
  vector[J] alpha;
  vector[J] beta;
  real mu_alpha;
  real mu_beta;
  real<lower=0> sigma_alpha;
  real<lower=0> sigma_beta;
  real<lower=0> sigma;
}
model {
  mu_alpha ~ normal(0, 10);
  mu_beta ~ normal(0, 10);
  sigma_alpha ~ normal(0, 10);
  sigma_beta ~ normal(0, 10);
  sigma ~ normal(0, 10);

  alpha ~ normal(mu_alpha, sigma_alpha);
  beta ~ normal(mu_beta, sigma_beta);

  for (n in 1:N)
    y[n] ~ normal(alpha[genre[n]] + beta[genre[n]] * x[n], sigma);
}
"""
```

In [ ]:
```
stan_gross_file_hier = "genre_gross_hierarchical.stan"
with open(stan_gross_file_hier, "w") as f:
    f.write(stan_gross_model_hier)
```

In [ ]:
```
stan_data_gross_hier = {
    'N': len(data_simple),
    'J': len(set(genre_codes)),
    'genre': genre_codes,
    'x': data_simple['year_scaled'].values,
    'y': data_simple['gross'].values
}
```

In [ ]:
```
stan_gross_model_hier
```

Out[ ]:
```
'\ndata {\n  int<lower=1> N;\n  int<lower=1> J;\n  array[N] int<lower=1, upper=
J> genre;\n  vector[N] x;\n  vector[N] y;\n}\nparameters {\n  vector[J] alph
a;\n  vector[J] beta;\n  real mu_alpha;\n  real mu_beta;\n  real<lower=0> sigma
_alpha;\n  real<lower=0> sigma_beta;\n  real<lower=0> sigma;\n}\nmodel {\n  mu_
alpha ~ normal(0, 10);\n  mu_beta ~ normal(0, 10);\n  sigma_alpha ~ normal(0, 1
0);\n  sigma_beta ~ normal(0, 10);\n  sigma ~ normal(0, 10);\n\n  alpha ~ norma
l(mu_alpha, sigma_alpha);\n  beta ~ normal(mu_beta, sigma_beta);\n\n  for (n in
1:N)\n    y[n] ~ normal(alpha[genre[n]] + beta[genre[n]] * x[n], sigma);\n}\n'
```

In [ ]:
```
hier_gross_model = CmdStanModel(stan_file='genre_gross_hierarchical.stan')
```

```
18:34:33 - cmdstanpy - INFO - compiling stan file C:\Users\84207\Desktop\SI618\si
618fa23-student-main\inclass\genre_gross_hierarchical.stan to exe file C:\Users\8
4207\Desktop\SI618\si618fa23-student-main\inclass\genre_gross_hierarchical.exe
18:34:49 - cmdstanpy - INFO - compiled model executable: C:\Users\84207\Desktop\S
I618\si618fa23-student-main\inclass\genre_gross_hierarchical.exe
```

In [ ]:
```
fit_gross_hier = hier_gross_model.sample(data=stan_data_gross_hier, chains=4, \
                                          iter_sampling=1000, iter_warmup=500)
```

```
18:34:50 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status

18:34:54 - cmdstanpy - INFO - CmdStan done processing.
```

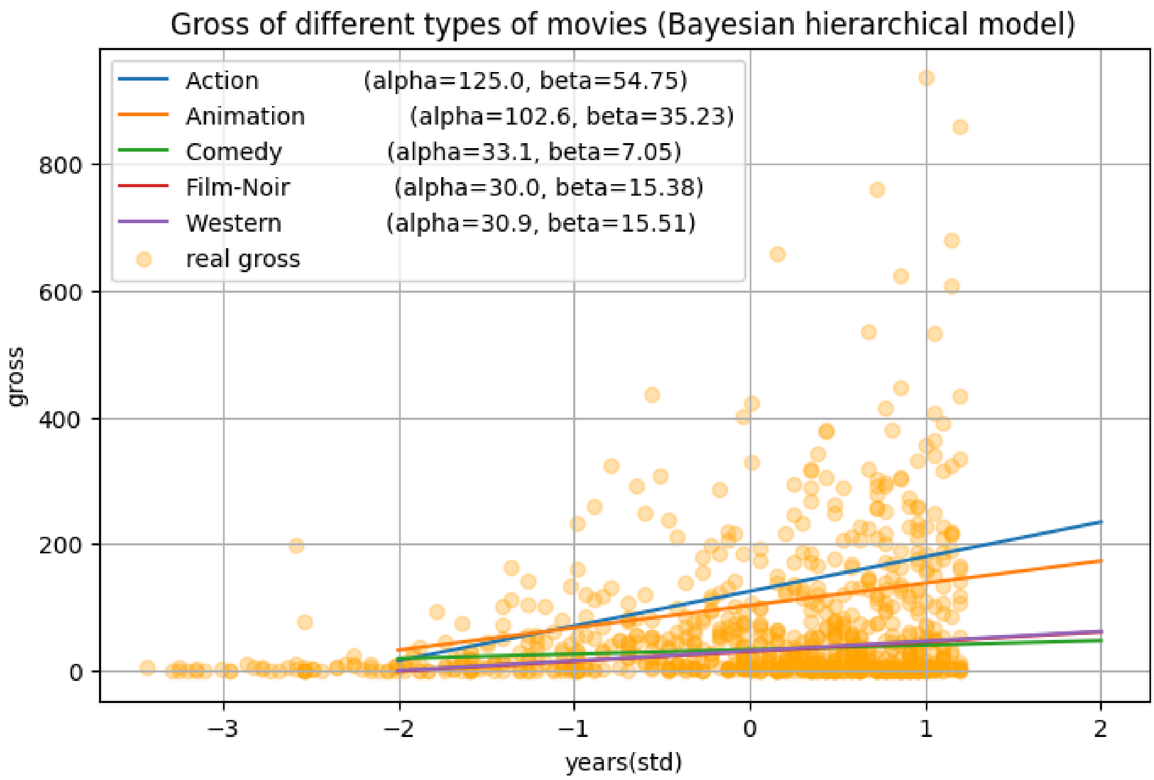
In [ ]:
```
print(fit_gross_hier.summary())
```

```
                    Mean      MCSE     StdDev          5%          50%  \
lp__        -4368.090000  0.123481   4.643790 -4376.27000 -4367.76000
alpha[1]      124.996000  0.107890   8.092550   111.48100   124.97200
alpha[2]       89.695200  0.151453  11.716800    70.47670    89.55970
alpha[3]      102.618000  0.177070  12.418400    82.04590   102.41100
alpha[4]       54.883200  0.126720  10.232600    38.07940    54.81000
alpha[5]       33.076800  0.101047   8.090600    20.02320    33.12390
alpha[6]       35.593700  0.121355   9.129320    20.57150    35.55600
alpha[7]       38.699100  0.078188   5.855980    29.18860    38.61340
alpha[8]       71.256700  0.440807  31.308500    19.86150    71.04570
alpha[9]       28.005600  0.459042  32.568100   -26.47110    28.02090
alpha[10]      29.977300  0.437572  32.603100   -24.68030    30.05380
alpha[11]      57.765100  0.286602  22.130900    21.82760    57.80400
alpha[12]      33.781500  0.342929  24.181900    -6.30403    34.41770
alpha[13]      31.201500  0.438419  33.158100   -25.68160    32.31250
alpha[14]      30.920900  0.384122  28.924100   -16.67610    31.45560
beta[1]        54.750900  0.150815   9.100520    39.86710    54.63550
beta[2]        29.062300  0.128455   9.421720    13.81650    29.02810
beta[3]        35.230400  0.206732  13.281200    14.53770    34.83370
beta[4]        16.129700  0.149172  10.725000    -1.65427    16.37080
beta[5]         7.052950  0.099377   7.226660    -4.75346     6.91377
beta[6]        13.313900  0.115537   8.631070    -1.02917    13.38820
beta[7]         7.611340  0.074760   5.457920    -1.50937     7.75075
beta[8]        10.187600  0.327183  20.164200   -24.99540    11.16240
beta[9]        15.565700  0.303805  20.128100   -18.35400    16.28430
beta[10]       15.384700  0.270151  17.689300   -13.64420    15.46020
beta[11]       17.225100  0.232601  16.485700   -10.45810    17.46610
beta[12]       15.041200  0.226674  16.375100   -12.07680    15.36760
beta[13]       14.866900  0.259975  18.900900   -17.49700    15.28210
beta[14]       15.509400  0.266981  18.033000   -14.78460    15.69190
mu_alpha       30.362127  0.143553   8.615695    16.30560    30.52020
mu_beta        15.536367  0.115288   6.156391     4.81748    15.88840
sigma_alpha    33.896440  0.094937   5.285749    25.73780    33.52840
sigma_beta     17.853146  0.105112   4.516761    10.95580    17.49270
sigma          93.226423  0.028982   2.067114    89.93270    93.14670

                   95%       N_Eff      N_Eff/s     R_hat
lp__        -4360.9900  1414.320000  572.135000  1.000420
alpha[1]      138.1920  5626.110000  2275.930000  0.999347
alpha[2]      108.7610  5984.960000  2421.100000  0.999254
alpha[3]      123.0990  4918.560000  1989.710000  0.999439
alpha[4]       71.7635  6520.480000  2637.730000  0.999508
alpha[5]       46.4332  6410.900000  2593.400000  0.999595
alpha[6]       50.6783  5659.290000  2289.360000  0.999748
alpha[7]       48.3350  5609.380000  2269.160000  0.999516
alpha[8]      124.3400  5044.590000  2040.690000  0.999667
alpha[9]       80.9300  5033.610000  2036.250000  1.000080
alpha[10]      82.1079  5551.600000  2245.790000  0.999483
alpha[11]      94.9846  5962.640000  2412.070000  0.999415
alpha[12]      73.4372  4972.470000  2011.520000  0.999652
alpha[13]      84.2327  5720.060000  2313.940000  0.999604
alpha[14]      76.5922  5669.960000  2293.670000  0.999754
beta[1]        69.8563  3641.200000  1472.980000  0.999898
beta[2]        44.6111  5379.700000  2176.250000  0.999266
beta[3]        57.9719  4127.260000  1669.600000  0.999613
beta[4]        33.5763  5169.180000  2091.090000  0.999417
beta[5]        19.1877  5288.200000  2139.240000  1.000760
beta[6]        27.1094  5580.720000  2257.570000  0.999480
beta[7]        16.6118  5329.800000  2156.070000  0.999359
beta[8]        41.3273  3798.250000  1536.510000  0.999412
```

```
beta[9]        48.1824  4389.500000  1775.690000  0.999540
beta[10]       44.0778  4287.530000  1734.440000  0.999551
beta[11]       43.7532  5023.340000  2032.090000  0.999501
beta[12]       41.6711  5218.700000  2111.120000  0.999746
beta[13]       45.4405  5285.710000  2138.230000  1.000080
beta[14]       44.8326  4562.240000  1845.570000  0.999786
mu_alpha       44.5923  3602.090859  1457.156496  1.000267
mu_beta        25.0098  2851.567574  1153.546753  1.001136
sigma_alpha    42.9658  3099.848245  1253.983918  1.000702
sigma_beta     25.8207  1846.518361   746.973447  0.999421
sigma          96.6784  5087.049414  2057.867886  0.999436
```

In [ ]:

```
selected_types = {1: "Action", 3: "Animation", 5: "Comedy", \
                  10: "Film-Noir", 14: "Western"}
selected_params = {
    genre: {
        "alpha": fit_gross_hier.stan_variable('alpha')[:, index - 1].mean(),
        "beta": fit_gross_hier.stan_variable('beta')[:, index - 1].mean()
    }
    for index, genre in selected_types.items()
}

plt.figure(figsize=(8, 5))
x_years = np.linspace(-2, 2, 100)
for genre, params in selected_params.items():
    y_pred = params["alpha"] + params["beta"] * x_years
    plt.plot(x_years, y_pred, label=f"{genre} \
             (alpha={params['alpha']:.1f}, beta={params['beta']:.2f})")
plt.scatter(data_simple['year_scaled'], data_simple['gross'], \
            alpha=0.3, color="orange", label="real gross")
plt.title("Gross of different types of movies (Bayesian hierarchical model)")
plt.xlabel("years(std)")
plt.ylabel("gross")
plt.legend()
plt.grid()
plt.show()
```



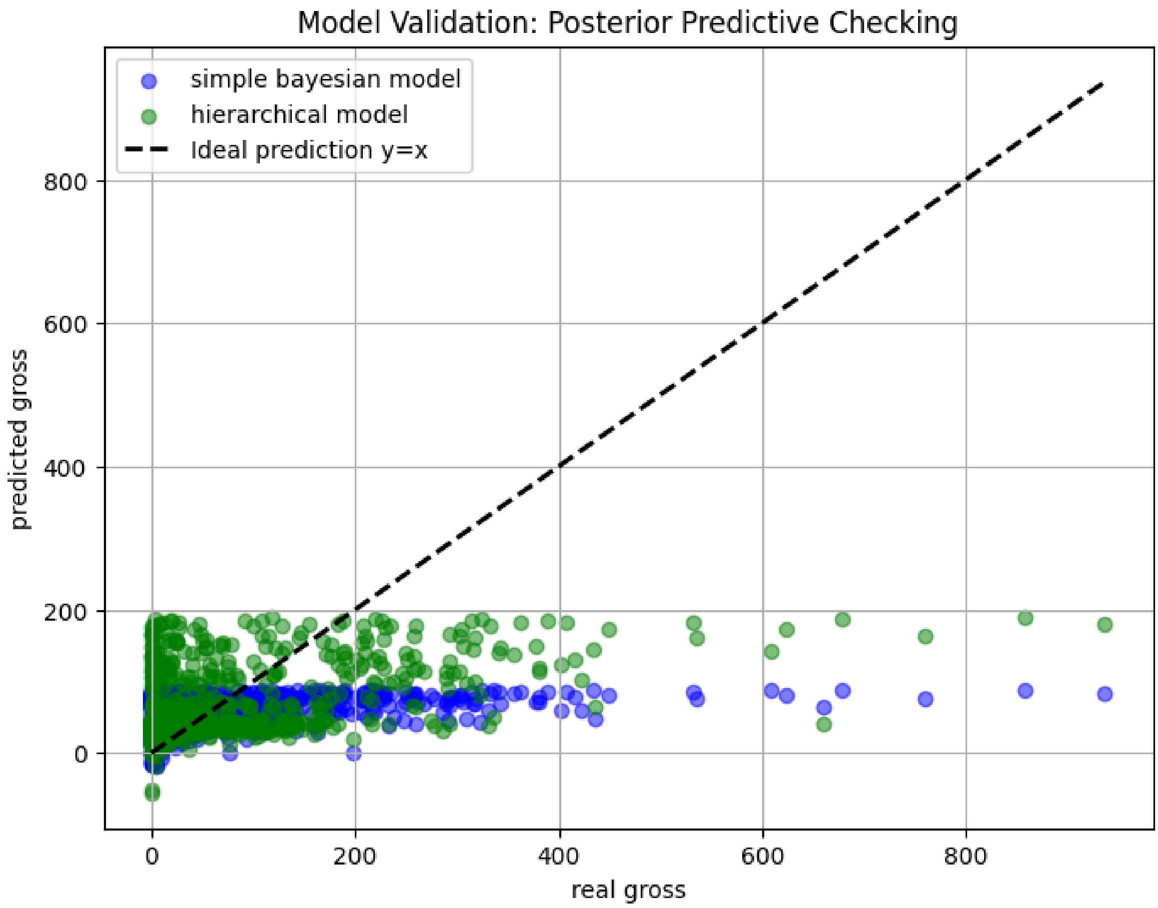
As can be seen from the figure, the box office of action movies has increased the fastest, followed by animation.

Model Checking and Evaluation

In [ ]:
```
import arviz as az
```

In [ ]:
```
y_actual = data_simple['gross'].values

alpha_simple = fit_gross.stan_variable('alpha')
beta_simple = fit_gross.stan_variable('beta')
y_pred_simple = alpha_simple[:, None] + beta_simple[:, None] * \
    data_simple['year_scaled'].values
y_pred_simple_mean = y_pred_simple.mean(axis=0)

genre_indices = stan_data_gross_hier['genre'] - 1
alpha_samples = fit_gross_hier.stan_variable('alpha')
beta_samples = fit_gross_hier.stan_variable('beta')

y_pred_hierarchical = np.array([
    alpha_samples[:, genre] + beta_samples[:, genre] * year
    for genre, year in zip(genre_indices, \
                           data_simple['year_scaled'].values)
])
y_pred_hierarchical_mean = y_pred_hierarchical.mean(axis=1)
plt.figure(figsize=(8, 6))
plt.scatter(y_actual, y_pred_simple_mean, alpha=0.5, \
            label="simple bayesian model", color="blue")
plt.scatter(y_actual, y_pred_hierarchical_mean, alpha=0.5, \
            label="hierarchical model", color="green")
plt.plot([y_actual.min(), y_actual.max()], \
         [y_actual.min(), y_actual.max()], \
            'k--', lw=2, label="Ideal prediction y=x")
```

```
plt.title("Model Validation: Posterior Predictive Checking")
plt.xlabel("real gross")
plt.ylabel("predicted gross")
plt.legend()
plt.grid()
plt.show()
```



As can be seen from the figure, the hierarchical model is closer to y=x, which means it is closer to the real data distribution and the model prediction accuracy is higher. However, it still fails to capture the trend of gross maximum. \

In addition, the hierarchical model runs relatively slowly compared to the simple Bayesian model.