# Movies

May 18, 2021

[1]:
```python
import warnings
warnings.filterwarnings('ignore')


import pandas as pd
import numpy as np
from plotnine import *
from plotnine.data import mtcars
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler #Z-score variables

from sklearn.model_selection import train_test_split # simple TT split cv
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA

%matplotlib inline
```

[2]:
```python
data = pd.read_csv("./movies.csv")
```

[3]:
```python
data.head()
```

[3]:
```
     budget                              company country         director  \
0   8000000          Columbia Pictures Corporation     USA       Rob Reiner
1   6000000                     Paramount Pictures     USA      John Hughes
2  15000000                     Paramount Pictures     USA       Tony Scott
3  18500000  Twentieth Century Fox Film Corporation     USA    James Cameron
4   9000000                    Walt Disney Pictures     USA   Randal Kleiser

       genre  genre_encoded      gross                     name rating  \
0  Adventure              0   52287414               Stand by Me      R
1     Comedy              1   70136369  Ferris Bueller's Day Off  PG-13
2     Action              2  179800601                   Top Gun     PG
3     Action              2   85160248                    Aliens      R
4  Adventure              0   18564613   Flight of the Navigator     PG
```

```
   rating_encoded    released  runtime  score               star   votes  \
0               3  1986-08-22       89    8.1        Wil Wheaton  299174
1               2  1986-06-11      103    7.8  Matthew Broderick  264740
2               1  1986-05-16      110    6.9         Tom Cruise  236909
3               3  1986-07-18      137    8.4   Sigourney Weaver  540152
4               1  1986-08-01       90    6.9        Joey Cramer   36636

          writer  year released
0   Stephen King           1986
1    John Hughes           1986
2       Jim Cash           1986
3  James Cameron           1986
4  Mark H. Baker           1986
```

[48]:
```
data.columns
```

[48]:
```
Index(['budget', 'company', 'country', 'director', 'genre', 'genre_encoded',
       'gross', 'name', 'rating', 'rating_encoded', 'released', 'runtime',
       'score', 'star', 'votes', 'writer', 'year released',
       'year_assignments'],
      dtype='object')
```

[5]:
```
data.isnull().sum(axis=0) # checked to make sure there is no missing data
```

[5]:
```
budget            0
company           0
country           0
director          0
genre             0
genre_encoded     0
gross             0
name              0
rating            0
rating_encoded    0
released          0
runtime           0
score             0
star              0
votes             0
writer            0
year released     0
dtype: int64
```

[6]:
```
movies_before_2000 = data[data['year released'] < 2000]
print("There are " + str(len(movies_before_2000)) + " movies from the dataset␣
 ↪that were released before 2000.")
```

```
movies_before_2000.tail()
```

There are 3080 movies from the dataset that were released before 2000.

[6]:
```
         budget                           company country           director  \
3075          0                    3B Productions  France       Bruno Dumont
3076          0                      C.E.O. Films     USA        George Haas
3077     312000                   Spanky Pictures     USA      Gavin O'Connor
3078          0  Cinerenta Medienbeteiligungs KG     USA       Scott Elliott
3079    7500000                   Code Productions     USA  Robert Marcarelli

        genre  genre_encoded     gross              name         rating  \
3075    Drama              3    113495          Humanité  Not specified
3076    Drama              3     94633   Friends & Lovers              R
3077    Drama              3   1281176        Tumbleweeds          PG-13
3078    Drama              3    544538  A Map of the World              R
3079   Action              2  12614346     The Omega Code          PG-13

      rating_encoded    released  runtime  score               star  votes  \
3075               5  1999-10-27      148    6.9   Emmanuel Schotté   3105
3076               3  1999-04-16      100    4.5    Stephen Baldwin   1330
3077               2  2000-03-03      102    6.7       Janet McTeer   3018
3078               3  2000-01-21      125    6.7   Sigourney Weaver   3659
3079               2  1999-08-27      100    3.5    Casper Van Dien   4762

              writer  year released
3075    Bruno Dumont           1999
3076     Neill Barry           1999
3077  Angela Shelton           1999
3078   Jane Hamilton           1999
3079   Stephan Blinn           1999
```

[7]:
```
movies_2000_and_after = data[data['year released'] >= 2000]
print("There are " + str(len(movies_2000_and_after)) + " movies from the dataset␣
↪that were released after 2000.")
movies_2000_and_after.tail()
```

There are 3740 movies from the dataset that were released after 2000.

[7]:
```
         budget                     company country         director      genre  \
6815          0    Fox Searchlight Pictures      UK   Mandie Fletcher     Comedy
6816          0    Siempre Viva Productions     USA   Paul Duddridge      Drama
6817    3500000              Warner Bros. 7     USA          Sam Liu  Animation
6818          0         Borderline Presents     USA    Nicolas Pesce      Drama
6819          0  Les Productions du Trésor  France    Nicole Garcia      Drama

      genre_encoded    gross                            name rating  \
```

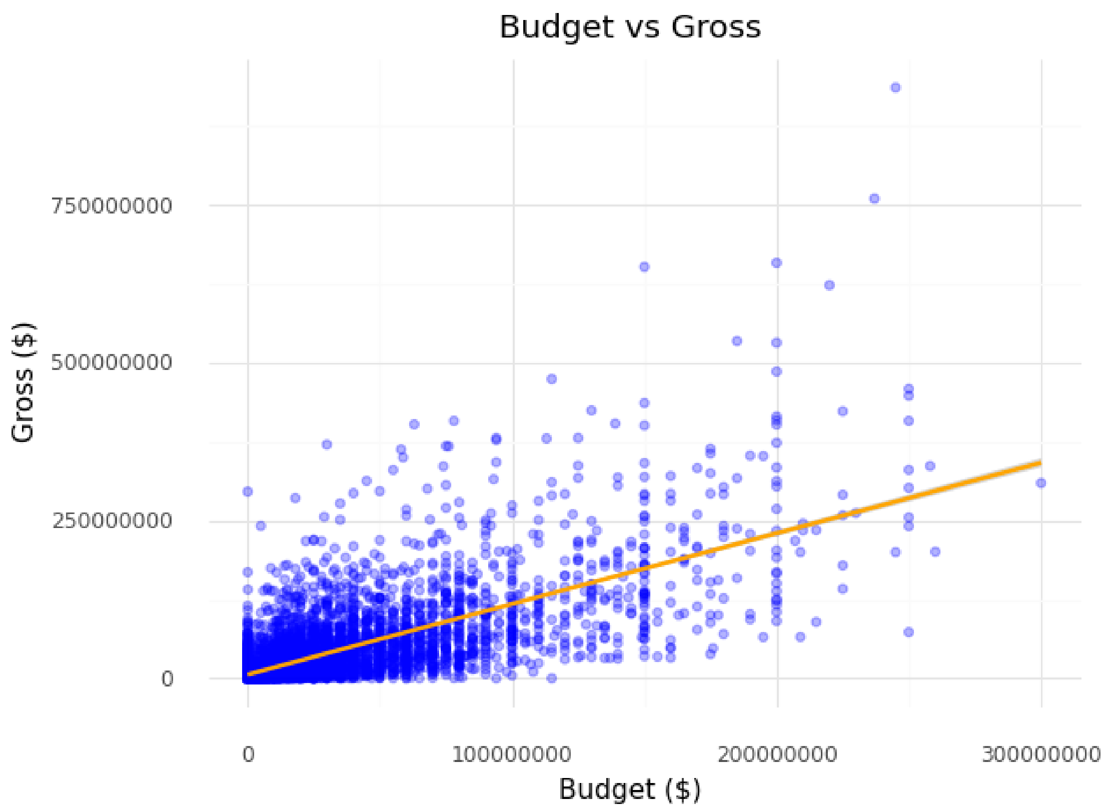
```
6815             1  4750497  Absolutely Fabulous: The Movie      R
6816             3    28368           Mothers and Daughters  PG-13
6817             7  3775000         Batman: The Killing Joke      R
6818             3    25981            The Eyes of My Mother      R
6819             3    37757        From the Land of the Moon      R

      rating_encoded    released  runtime  score                star  votes  \
6815               3  2016-07-22       91    5.4  Jennifer Saunders   9161
6816               2  2016-05-06       90    4.9        Selma Blair   1959
6817               3  2016-07-25       76    6.5       Kevin Conroy  36333
6818               3  2016-12-02       76    6.2     Kika Magalhães   6947
6819               3  2017-07-28      120    6.7   Marion Cotillard   2411

                 writer  year released
6815  Jennifer Saunders           2016
6816      Paige Cameron           2016
6817    Brian Azzarello           2016
6818      Nicolas Pesce           2016
6819        Milena Agus           2016
```

# 1 Question 1:

## 1.1 Part 1 What is the relationship between movie budget and revenue (gross),

## 1.2 Part 2: and is that relationship different for movies that came out before 2000 compared to movies that came out after 2000?

[8]:
```
(ggplot(data, aes(x = 'budget', y = 'gross'))
 + geom_point(colour = 'blue', alpha = 0.30)
 + theme_minimal() + ggtitle("Budget vs Gross")
 + labs(x = "Budget ($)", y = "Gross ($)")
 + geom_smooth(method = "lm", colour = 'orange')
)
```




[8]: <ggplot: (322879570)>

*Caption: Scatterplot showing the distribution of data when plotting movie budget vs. movie gross (i.e., revenue). A line of best fit has been included on the graph. There is a positive linear relationship between budget and gross, meaning that the more it costs to make a movie, the more likely it is for that movie to do well at the box office.*

# 2 Question 1 Discussion Part 1

## 2.1 What is the (general) relationship between movie budget and revenue (gross)?

- To understand the relationship between a movie's budget and its gross, the scatter plot was created above. The scatter plot has budget on the x-axis and gross on the y-axis. Budget is on the x-axis because it is the predictor variable (AKA independent variable) and gross is on the y-axis because it is the outcome that is being analyzed based on the budget. It can observed that generally as x (budget) increases, it can be expected the y (gross) value to increase. To confirm this trend, a best fit linear regression line was added. The regression line aligns with the observation that as x increases, it can be expected for y to increase because the slope of the regression line is positive and greater than 1. This is because the slope is slanted upwards. This slope that is positive and greater than 1 indicates that the relationship between budget and revenue is a positive linear relationship.

- Our results are not too suprising because it makes sense for a movie to make more money (gross) if the movie's producers have more money (budget). Having a higher budget allows movie producers to have more resources, experiment with different ideas, and ultimately gives them a lot more opportunities to make a more successful (in terms of gross) movie.
- In the next section, the question of whether this positive linear relationship is similar or different across movies made before 2000 vs movies made 2000 and after is explored.

[9]:
```
# labsList = ['Movies Released Before 2000', 'Movies Released 2000 and After']
year_assignments = []
for i in data['year released']:
    if i >= 2000:
        year_assignments.append(1)
    else:
        year_assignments.append(0)
len(year_assignments)
```

[9]: 6820

[10]:
```
data['year_assignments'] = year_assignments
data.tail()
```

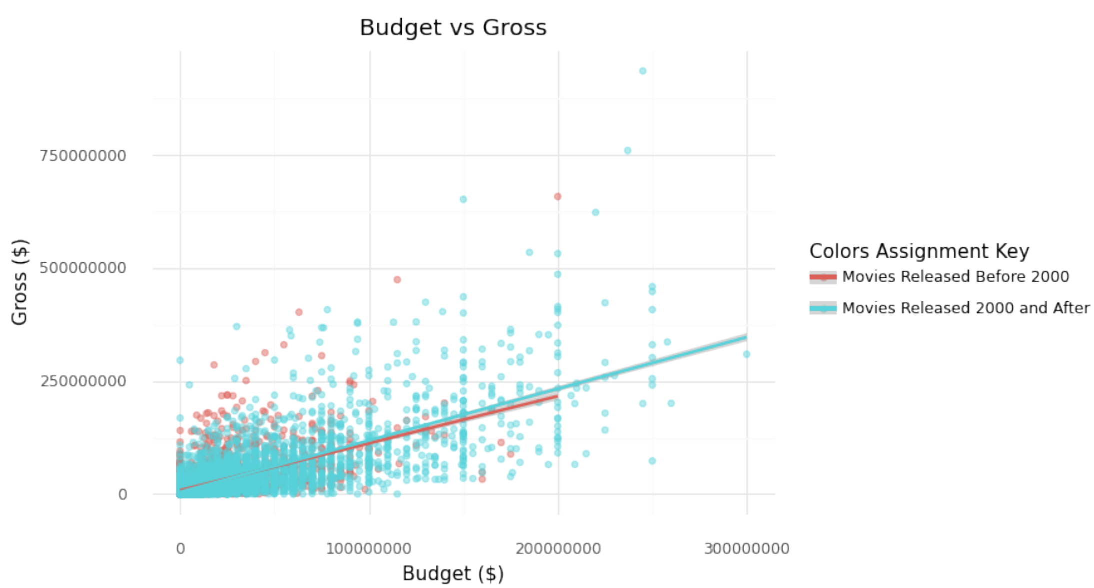
[10]:
```
       budget                  company country        director      genre  \
6815        0   Fox Searchlight Pictures      UK  Mandie Fletcher     Comedy
6816        0   Siempre Viva Productions     USA   Paul Duddridge      Drama
6817  3500000             Warner Bros. 7     USA          Sam Liu  Animation
6818        0        Borderline Presents     USA    Nicolas Pesce      Drama
6819        0  Les Productions du Trésor  France    Nicole Garcia      Drama

      genre_encoded    gross                          name rating  \
6815              1  4750497  Absolutely Fabulous: The Movie      R
6816              3    28368         Mothers and Daughters  PG-13
6817              7  3775000        Batman: The Killing Joke      R
6818              3    25981           The Eyes of My Mother      R
6819              3    37757        From the Land of the Moon      R

      rating_encoded    released  runtime  score              star  votes  \
6815               3  2016-07-22       91    5.4  Jennifer Saunders   9161
6816               2  2016-05-06       90    4.9        Selma Blair   1959
6817               3  2016-07-25       76    6.5      Kevin Conroy  36333
6818               3  2016-12-02       76    6.2    Kika Magalhães   6947
6819               3  2017-07-28      120    6.7  Marion Cotillard   2411

                 writer  year released  year_assignments
6815  Jennifer Saunders           2016                 1
6816      Paige Cameron           2016                 1
6817    Brian Azzarello           2016                 1
6818      Nicolas Pesce           2016                 1
```

```
6819        Milena Agus          2016                   1
```

[11]:
```
labsList = ["Cluster " + str(i) for i in year_assignments]
label_titles = ['Movies Released Before 2000', 'Movies Released 2000 and After']
print("Testing out the labsList list: ")
print(labsList[1000])
print(labsList[6000])
```

```
Testing out the labsList list:
Cluster 0
Cluster 1
```

[12]:
```
(ggplot(data, aes(x = 'budget', y = 'gross', colour =␣
↪"factor(year_assignments)"))
 + geom_point(alpha = 0.45)
 + theme_minimal()
 + ggtitle("Budget vs Gross")
 + labs(x = "Budget ($)", y = "Gross ($)")
 + geom_smooth(method = "lm")
 + scale_color_discrete(name = "Colors Assignment Key",
                        labels = label_titles)
)
```




[12]: <ggplot: (323070943)>

*Caption: Scatterplot showing the distribution of data when plotting movie budget vs. movie gross (i.e., revenue) for movies released before 2000 and movies released 2000 and after. A line of best fit for each movie category has been included on the graph. The linear relationship between budget and gross for movies* *released 2000 and after appears to be slightly more positive than that of movies released before 2000.*

# 3 Question 1 Discussion Part 2

## 3.1 Is the relationship between movie budget and revenue (gross) different for movies that came out before 2000 compared to movies that came out after 2000?

- To understand if the relationship between budget and gross are different among movies made before 2000 vs movies made 2000 and after, another scatter plot was created. Once again, the x-axis is budget and the y-axis is gross. The main difference in this graph is that the 2 groups (movies before 2000s and movies 2000s+) are distinguised by different colors. The salmon/pink colored data points represent movies that were made before 2000. The light/sky blue color data points represent movies that were made 2000 and beyond. Each of the groups have their own best fit regression lines and are colored relative to their group colors.
- It can be seen from the graph above, that 2 groups have very similar results. Both regression lines are positive and greater than 1 because they are slanted up and continue to go up as budget increases. These results suggest that generally it can be expected for a movie's gross to increase as its budget increases.
- Although the regression lines are very similar, it should be noted that the group of movies released in 2000 and after is higher (in terms of gross) than the other regression line. This is most likely due to the fact that movies made 2000s and after are much more likely to have made more money/gross compared to movies made before 2000. This is not too suprising because the movie industry and its audience grows significantly over years especially over the last couple of years. Inflation may also be another reason as to why the movies made 2000 and after have a higher regression line.

# 4 Question 1 Explicit Answers to Parts 1 and 2

## 4.1 Part 1: What is the (general) relationship between movie budget and revenue (gross)?

- **The relationship between movie budget and revenue is positive linear relationship.** Generally, as a movie's budget increases, it can be expected that its respected gross to increase as well.

## 4.2 Part 2: Is the relationship between movie budget and revenue (gross) different for movies that came out before 2000 compared to movies that came out after 2000?

- **The relationship between movie budget and revenue is not different, except that for the movies made in 2000 and after CAN (not necessarily always) have higher grosses.**

# 5 Question 2

## 5.1 Using the number of user votes as a proxy for movie popularity, are certain genres, (action, drama, and adventure), of movies more popular than others?

### 5.1.1 Part 1: Boxplot of users votes across different genres

### 5.1.2 Part 2: Barplot of average number of user votes across different genres

### 5.1.3 Part 3: Barplot of count of movies from dataset across different genres

[13]:
```
print("The dataset contains all of these genres: \n")
for i in data['genre'].unique():
    print(i)
print("\n")
print("However, only action, drama, and adventure are of interest for this␣
 ↪question.")
```

```
The dataset contains all of these genres:

Adventure
Comedy
Action
Drama
Crime
Thriller
Horror
Animation
Biography
Sci-Fi
Musical
Family
Fantasy
Mystery
War
Romance
Western


However, only action, drama, and adventure are of interest for this question.
```

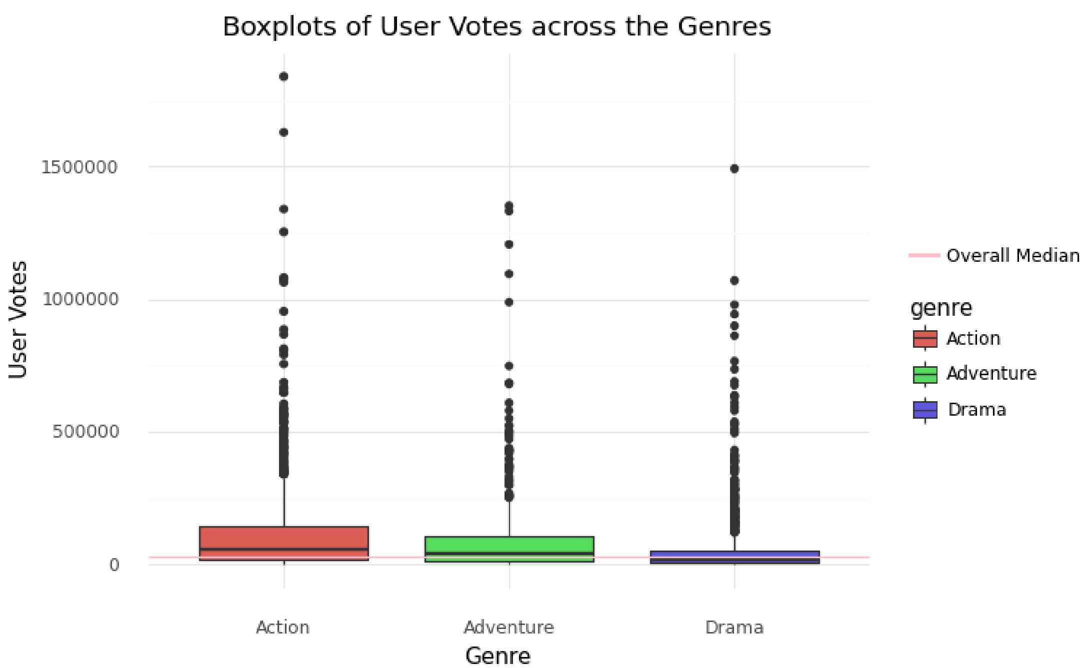
[14]:
```
print("All of the movies that are considered an action, drama, or adventure␣
 ↪genre\nare found and then stored in the variable called data_genre_filtered")
desired_genres = ['Action', 'Adventure', 'Drama']
data_genre_filtered = data[data['genre'].isin(desired_genres)]
data_genre_filtered.head()
```

```
All of the movies that are considered an action, drama, or adventure genre
are found and then stored in the variable called data_genre_filtered
```

[14]:
```
      budget                           company country         director  \
0    8000000       Columbia Pictures Corporation     USA       Rob Reiner
2   15000000                  Paramount Pictures     USA       Tony Scott
3   18500000  Twentieth Century Fox Film Corporation USA    James Cameron
4    9000000               Walt Disney Pictures     USA   Randal Kleiser
5    6000000                            Hemdale      UK     Oliver Stone

       genre  genre_encoded      gross                      name rating  \
0  Adventure              0   52287414               Stand by Me      R
2     Action              2  179800601                   Top Gun     PG
3     Action              2   85160248                    Aliens      R
4  Adventure              0   18564613  Flight of the Navigator     PG
5      Drama              3  138530565                   Platoon      R

   rating_encoded    released  runtime  score              star   votes  \
0               3  1986-08-22       89    8.1       Wil Wheaton  299174
2               1  1986-05-16      110    6.9        Tom Cruise  236909
3               3  1986-07-18      137    8.4  Sigourney Weaver  540152
4               1  1986-08-01       90    6.9       Joey Cramer   36636
5               3  1987-02-06      120    8.1     Charlie Sheen  317585

          writer  year released  year_assignments
0   Stephen King           1986                 0
2       Jim Cash           1986                 0
3  James Cameron           1986                 0
4  Mark H. Baker           1986                 0
5   Oliver Stone           1986                 0
```

[15]:
```
action_median_user_votes = data_genre_filtered[data_genre_filtered['genre'] ==␣
 ↪'Action']['votes'].median()
adventure_median_user_votes = data_genre_filtered[data_genre_filtered['genre']␣
 ↪== 'Adventure']['votes'].median()
drama_median_user_votes = data_genre_filtered[data_genre_filtered['genre'] ==␣
 ↪'Drama']['votes'].median()
median_user_votes = data_genre_filtered['votes'].median()
```

[16]:
```
(ggplot(data_genre_filtered, aes(x = 'genre', y='votes', fill = "genre"))
 + geom_boxplot(stat = "boxplot")
 + theme_minimal()
 + ggtitle("Boxplots of User Votes across the Genres")
 + labs(x = "Genre", y = "User Votes")
 + geom_hline(aes(yintercept = median_user_votes, color=["Overall Median"]),
 ↪show_legend=True)
 + scale_color_manual(values="pink",name=' ')
)
```




[16]: <ggplot: (323079708)>

*Caption: Boxplots showing the total number of user votes across three movie genres: action, adventure, and drama. The median for action is slightly higher than that of adventure, which is slightly higher than that of drama, although all three medians are relatively similar.*

[17]:
```
print("The median number of user votes for Action movies is:\n" +␣
↪str(action_median_user_votes) + "\n")
print("The median number of user votes for Adventure movies is:\n" +␣
↪str(adventure_median_user_votes)+ "\n")
print("The median number of user votes for Drama movies is:\n" +␣
↪str(drama_median_user_votes)+ "\n")
```

```
The median number of user votes for Action movies is:
55046.0

The median number of user votes for Adventure movies is:
39098.5

The median number of user votes for Drama movies is:
16435.5
```

## 6 Question 2 Part 1 Discussion

- A boxplot (shown above) was created to help gain insight or a better understanding of the user votes variability across the 3 different genres of interest (Action - red, Adventure - green, Drama - blue). The boxplot above is an effective visualization of the distribution of user votes for each genre because it allows us to see the shape of the distribution of the data. The black horizontal lines within each box (AKA inner-quartile range) represent the median (value separating the higher half from the lower half of a data sample) number of user votes for that genre. It can be observed that the medians for each category is very small compared to the high points that fall out of their inner-quartile range's container. The pink line represents the median number of votes across all 3 genres. This overall median value is small like each genre's median which what is expected given the small medians for each genre.
- These observations tell us that the large majority of movies do not have a great amount of votes. The very high user votes values across the different genres are outliers because these movies performed exceptionally well in regards to user votes.
- The size of the inner-quartile range provide us with insight in terms of the amount of variability observed in each genre. Generally speaking, the larger a inner-quartile range is of a boxplot means that there is more variability of data in that sample. The Action genre has the largest inner-quartile range of all 3 genres above, which indicates that the Action genre has greater variability in terms of user votes. Adventure has the second most variability and drama has the least variability of these 3 genres.
- Concrete conslusions cannot be drawn with this knowledge but it is insightful to learn and expand upon to gain a better overall understanding of the data. For example, perhaps Action has the largest amount of variability because there are a lot of action movies. If there are a lot of movies in a certain genre, it may be more difficult for smaller movies of that genre to stand out and get user votes. To gain a better understanding of the different genres, barplots of the data is created next.

[18]:
```
# init dict that will hold avg votes for each desired genre
user_votes_genres = {
    "Action": {
        "avg_user_votes": 0,
        "genre": 'Action',
        "count": 0
    },
    "Adventure": {
        "avg_user_votes": 0,
        "genre": "Adventure",
        "count": 0
    },
    "Drama": {
        "avg_user_votes": 0,
        "genre": "Drama",
        "count": 0
    }
}

# populate dict with avg votes for each genre
```

```
for key in user_votes_genres:
    user_votes_genres[key]['avg_user_votes'] = np.
 ↪mean(data_genre_filtered[data_genre_filtered['genre'] == key]['votes'])
    user_votes_genres[key]['count'] =␣
 ↪len(data_genre_filtered[data_genre_filtered['genre'] == key])
user_votes_genres
```

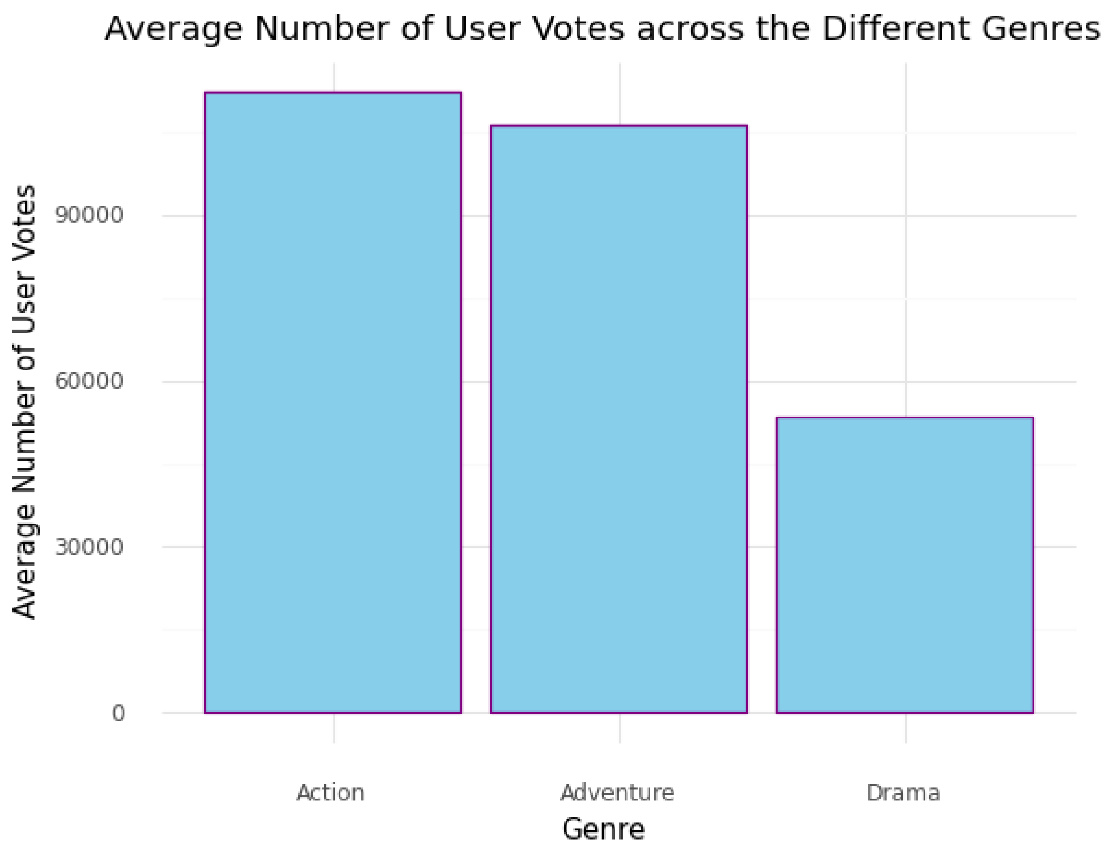
[18]:
```
{'Action': {'avg_user_votes': 112157.26897069873,
  'genre': 'Action',
  'count': 1331},
 'Adventure': {'avg_user_votes': 106109.04081632652,
  'genre': 'Adventure',
  'count': 392},
 'Drama': {'avg_user_votes': 53389.16966759003,
  'genre': 'Drama',
  'count': 1444}}
```

[19]:
```
# plt.bar(avg_user_votes_genres.keys(), avg_user_votes_genres.values())
DF_user_votes_genres = pd.DataFrame.from_dict(user_votes_genres, orient =␣
 ↪'index')
DF_user_votes_genres
```

[19]:

| | avg_user_votes | genre | count |
|---|---|---|---|
| Action | 112157.268971 | Action | 1331 |
| Adventure | 106109.040816 | Adventure | 392 |
| Drama | 53389.169668 | Drama | 1444 |

[20]:
```
(ggplot(DF_user_votes_genres, aes(x = 'genre', y='avg_user_votes'))
 + geom_bar(stat = "identity", color = "purple", fill="skyblue")
 + theme_minimal()
 + ggtitle("Average Number of User Votes across the Different Genres")
 + labs(x = "Genre", y = "Average Number of User Votes")
)
```




[20]: <ggplot: (323228799)>

*Caption: Bargraph showing the average number of user votes across three movie genres: action, adventure, and drama. Action has the highest average number of user votes, adventure has a similar and the second highest average number, and drama has the lowest average number.*

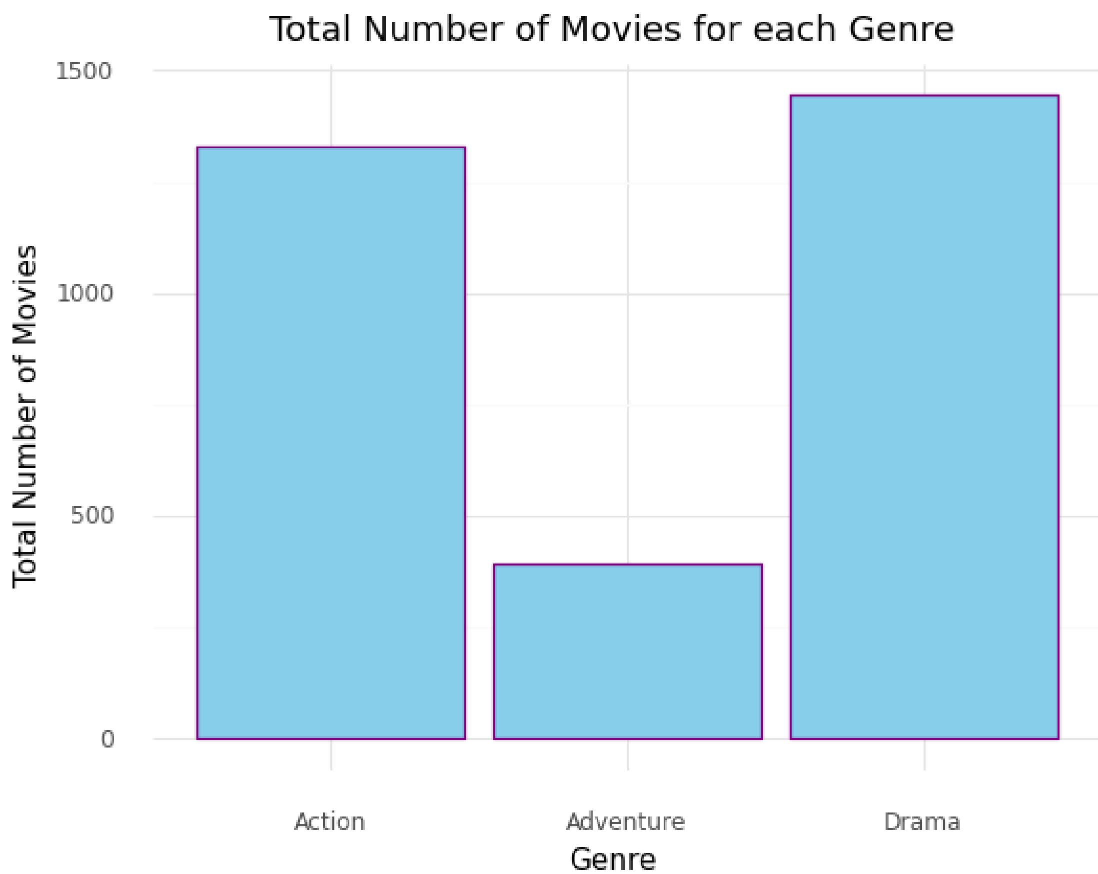
[21]:
```
for key in user_votes_genres:
    print(key
          + " has an average of about "
          + str(round(user_votes_genres[key]['avg_user_votes']))
          + " user votes\n")
```

```
Action has an average of about 112157 user votes

Adventure has an average of about 106109 user votes

Drama has an average of about 53389 user votes
```

# 7 Question 2 Part 2 Discussion

- A barplot (shown above) was created with each bar representing the average number of user votes for its respected genre. Action has the highest average number of votes with about 112,157 user votes. Adventure is the second highest average number of votes with about 106,109 votes and Drama has the least average number of user votes of about 53,389 votes.
- This graph was created to help give a better understanding of the data that is being worked with for these genres. In the previous part, it was discovered that action has the greatest amount of variability, adventure the second, and drama the last. This descending order pattern is the same pattern observed with the average number of user votes above. Perhaps this is the case because of the existance of very high user votes outliers under the Action genre. Looking back at the boxplot from part 1, it can be observed that there are a siginificant number of outliers that are above the Action's inner-quartile range. Those high user votes values under Action are most likely responsible for Action's high average number of user votes as well as Action's large variability of user votes.

[22]:
```
(ggplot(DF_user_votes_genres, aes(x = 'genre', y='count'))
 + geom_bar(stat = "identity", color = "purple", fill="skyblue")
 + theme_minimal()
 + ggtitle("Total Number of Movies for each Genre")
 + labs(x = "Genre", y = "Total Number of Movies")
)
```




[22]: <ggplot: (323071877)>

*Caption: Bargraph showing the total number of movies across three movie genres: action, adventure, and drama. Drama has the most movies, action has the second most, and adventure has the least. Drama has over triple the number of movies compared to adventure.*

[23]:
```
for key in user_votes_genres:
    print(key
          + " has "
          + str(round(user_votes_genres[key]['count']))
          + " total number of movies\n")
```

```
Action has 1331 total number of movies

Adventure has 392 total number of movies

Drama has 1444 total number of movies
```

## 8 Question 2 Part 3 Discussion

- A barplot (shown above) was created with each bar representing the total number of movies in its respected genre. Drama has the most (1444) number of movies, Action has the second most (1331), and Adventure has the smallest number (392). This information is valuable because it provides insight about the data that is being worked with in these genres. Generally speaking, the more data a sample has, the more likely that the analysis or obersvations of/from that sample are reliable. Reliable meaning that the results or calculations of the dataset are not easily affected by outliers. With this in mind, it should be noted that the movies dataset provides significantly more Drama and Action movies compared to Adventure. And so observations/calculations of the Adventure sample are not as reliable as the other genres. The Adventure genre is much more susceptible to outliers compared to the other genres. Looking back at the boxplot, there are a significant number of outliers under the Adventure genre. So the Adventure genre's calculations are most likely being easily affected by its outliers.

## 9 Question 2 Explicit Answer to the Question - are certain genres, (action, drama, and adventure), of movies more popular than others?

- **According to this dataset, Action movies seem to be the most popular, Adventure the second, and drama the third. The main reasoning behind this answer is that Action movies have both the highest average and the highest median user votes of the 3 genres from the dataset provided.**
- Although those are good reasons to believe Action is the highest, it should be noted that Action and Adventure movies have significantly greater variation in user votes compared to Drama movies. This is important to consider because these 2 genres (Action & Adventure) have signficant number of outliers with very high user votes (can be seen in the boxplot graph). Additionally, it should be noted that there are far less Adventure movies than the other 2 genres in this dataset which could potentially mean that the Adventure movies data is not as reliable in providing insight on the genre outside of the dataset. The lack in number of Adventure movies could also potentially indicate that there are less Adventure movies in general (not just this dataset) than the other genres.

# Final Project

May 18, 2021

[2]:
```
# import necessary packages

import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from plotnine import *

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score

from sklearn.model_selection import train_test_split

%matplotlib inline
```

## 0.1 3. Do movies with a runtime over 100 minutes have a higher probability of grossing at least 20 million dollars than movies with a IMDb user score of at least 7.0?

## 0.2 a)

[3]:
```
df = pd.read_csv("/Users/emmachen/Downloads/CPSC392/HW:Projects/Final/movies.
 ↪csv")
df = df.dropna() # remove missing values
df = df.reset_index(drop=True)
df.head()
```

[3]:
```
       budget                            company country         director  \
0   8000000.0        Columbia Pictures Corporation     USA       Rob Reiner
1   6000000.0                   Paramount Pictures     USA      John Hughes
2  15000000.0                   Paramount Pictures     USA       Tony Scott
3  18500000.0  Twentieth Century Fox Film Corporation     USA    James Cameron
4   9000000.0                  Walt Disney Pictures     USA   Randal Kleiser
```

```
      genre        gross                      name rating    released  \
0  Adventure   52287414.0               Stand by Me      R  1986-08-22
1     Comedy   70136369.0  Ferris Bueller's Day Off  PG-13  1986-06-11
2     Action  179800601.0                   Top Gun     PG  1986-05-16
3     Action   85160248.0                    Aliens      R  1986-07-18
4  Adventure   18564613.0   Flight of the Navigator     PG  1986-08-01

   runtime  score               star   votes         writer  year
0       89    8.1        Wil Wheaton  299174   Stephen King  1986
1      103    7.8  Matthew Broderick  264740    John Hughes  1986
2      110    6.9         Tom Cruise  236909       Jim Cash  1986
3      137    8.4   Sigourney Weaver  540152  James Cameron  1986
4       90    6.9        Joey Cramer   36636  Mark H. Baker  1986
```

[4]:

```
# filter dataset given condition of grossing over 20 million
gross = (df["gross"] >= 20000000)
df_gross = df[gross] # store our filtered data in df_gross

# ----------------------

# filter dataset given conditions of having a runtime greater than 100min &␣
↪grossing over 20 million
runtime_gross = ((df["runtime"] > 100) & (df["gross"] >= 20000000))
df_runtime_gross = df[runtime_gross] # store our filtered data in df_runtime100

prob_runtime = df_runtime_gross.shape[0] / df_gross.shape[0] # calculate␣
↪probability
print("Probability of a movie with a runtime over 100 minutes grossing at least␣
↪20 million dollars:", prob_runtime)

# ----------------------

# filter dataset given conditions of having a IMDb user score of at least 7 &␣
↪grossing over 20 million
score_gross = ((df["score"] >= 7.0) & (df["gross"] >= 20000000))
df_score_gross = df[score_gross] # store our filtered data in df_score_gross

prob_score = df_score_gross.shape[0] / df_gross.shape[0] # calculate probability
print("Probability of a movie with a score of at least 7 grossing at least 20␣
↪million dollars:", prob_score)
```

```
Probability of a movie with a runtime over 100 minutes grossing at least 20
million dollars: 0.6511371973587674
Probability of a movie with a score of at least 7 grossing at least 20 million
dollars: 0.30117388114453414
```

[5]:
```
# make a new DF with just the probabilities above
Group = ["Runtime over 100 minutes", "IMDb score of at least 7"] # column 1 of␣
↪the df
Probability = [prob_runtime, prob_score] # column 2 of the df
myNewDF = {"Group": Group, "Probability": Probability} # create a dictionary for␣
↪df
probDF = pd.DataFrame(myNewDF) # create the df using pandas
probDF.head()
```

[5]:

| | Group | Probability |
|---|---|---|
| 0 | Runtime over 100 minutes | 0.651137 |
| 1 | IMDb score of at least 7 | 0.301174 |

## 0.3 b)

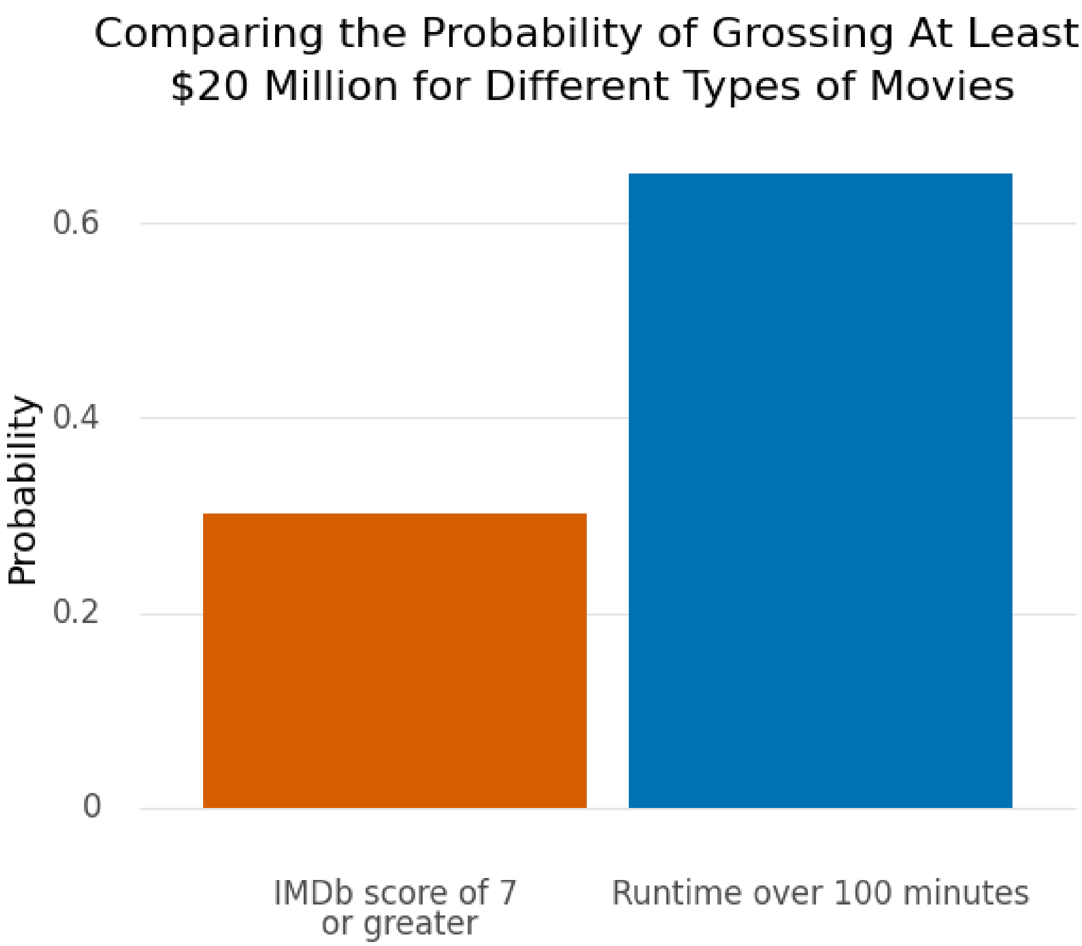
Based on the analyses above, movies with a runtime over 100 minutes have a higher probability (approx 0.65) of grossing at least 20,000,000 dollars than movies with an IMDb user score of at least 7.0 (probability of approx 0.30). In other words, based on this dataset, movies with a runtime over 100 minutes are more likely to gross 20,000,000 dollars or more compared to movies with an IMDb user score of at least 7.0. So, when trying to make a movie that will gross at least 20,000,000 dollars, it might be better to focus on achieving a longer runtime than aiming towards a good IMDb score.

To answer this question, we completed a number of calculations. We first calculated the probability of movies with a runtime over 100 minutes having a gross of at least 20,000,000 dollars. To do this, we divided the total number of movies with a gross of at least 20,000,000 dollars by the number of movies with both a gross of at least 20,000,000 dollars and a runtime over 100 minutes. Then, we calculated the probability of movies with an IMDb score of at least 7.0 having a gross of at least 20,000,000 dollars. To do this, we divided the total number of movies with a gross of at least 20,000,000 dollars by the number of movies with both a gross of at least 20,000,000 dollars and a score of at least 7.0. Once we calculated these two probabilities, we plotted them on a bar graph and pie chart in order to directly compare them to each other. A higher probability indicates a greater likelihood that something will occur; in the context of this question, movies with a runtime over 100 minutes have a probability of 0.65, versus the smaller 0.30 probability of movies with an IMDb score of at least 7, of grossing at least 20,000,000 dollars.

## 0.4 c)

[124]:
```
(ggplot(probDF, aes(x = "Group", y = "Probability", fill = "Group")) +
     geom_bar(stat="identity") +
     theme_minimal() +
     labs(x = "", y = "Probability") +
     scale_x_discrete(labels = ("IMDb score of 7\n or greater", "Runtime over␣
↪100 minutes")) +
     ggtitle("Comparing the Probability of Grossing At Least \n$20 Million for␣
↪Different Types of Movies") +
     scale_fill_manual(["#d55e00", "#0072b2"]) +
```

```
    theme(panel_grid_minor_x = element_blank(),
    panel_grid_minor_y = element_blank(),
    panel_grid_major_x = element_blank(),
    axis_text_x = element_text(size = 12),
    axis_title_x = element_text(size = 14),
    axis_text_y = element_text(size = 12),
    axis_title_y = element_text(size = 14),
    plot_title = element_text(lineheight = 1.5, size = 16),
    legend_text = element_text(size = 12),
    legend_position = "none"))
```




[124]: <ggplot: (8761979648793)>

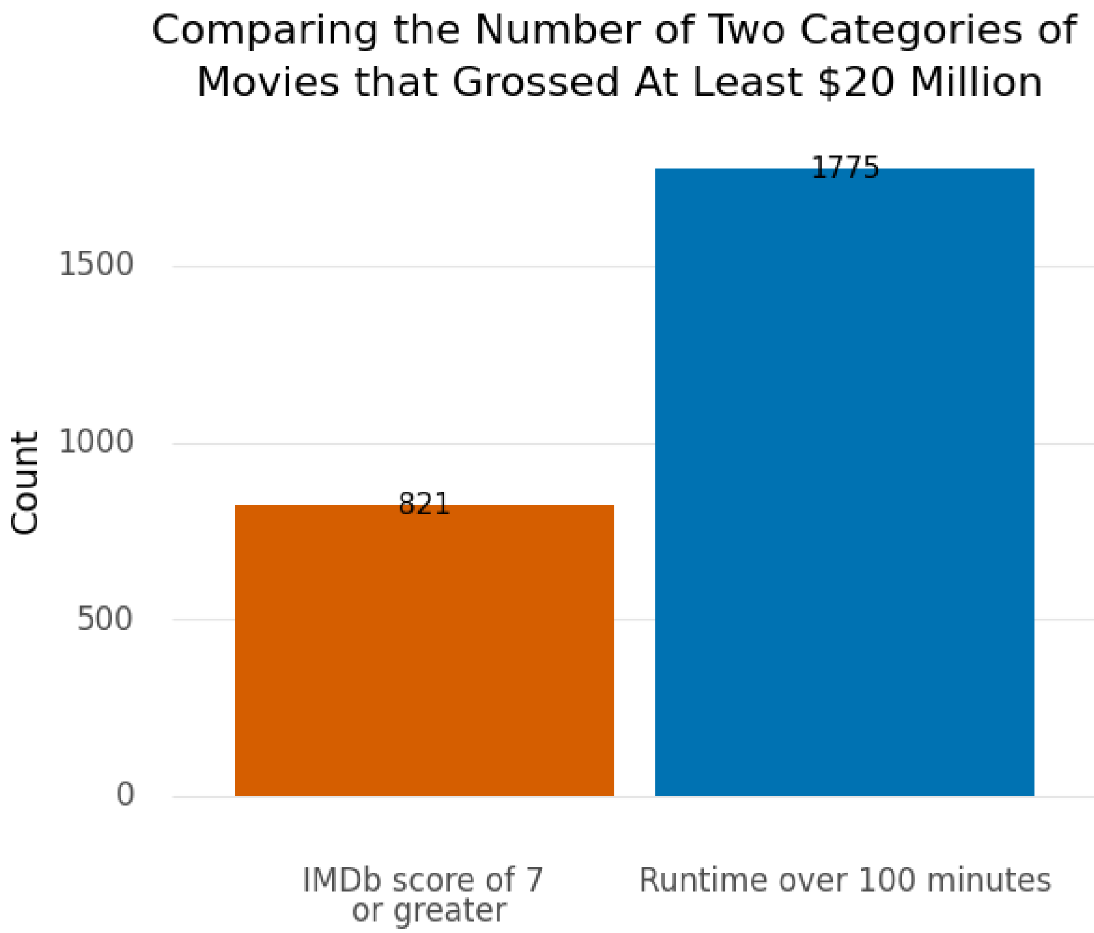
*Caption: Comparing the probability of grossing at least 20,000,000 dollars between movies that have an IMDb score of at least 7.0 and movies that have a runtime over 100 minutes. The greater the probability, as plotted on the y-axis, the more likely a movie will gross at least 20,000,000 dollars. As the graph shows, movies with a runtime over 100 minutes have a higher probability of grossing at least 20,000,000 dollars than movies with an IMDb score of at least 7.0.*

[9]:
```
count_runtime = len(df_runtime_gross)
count_score = len(df_score_gross)

Group = ["Runtime over 100 minutes", "IMDb score of at least 7"] # column 1 of␣
↪the df
Count = [count_runtime, count_score] # column 2 of the df

NewDF = {"Group": Group, "Count": Count} # create a dictionary for df
countDF = pd.DataFrame(NewDF) # create the df using pandas
countDF.head()
```

[9]:

| | Group | Count |
|---|---|---|
| 0 | Runtime over 100 minutes | 1775 |
| 1 | IMDb score of at least 7 | 821 |

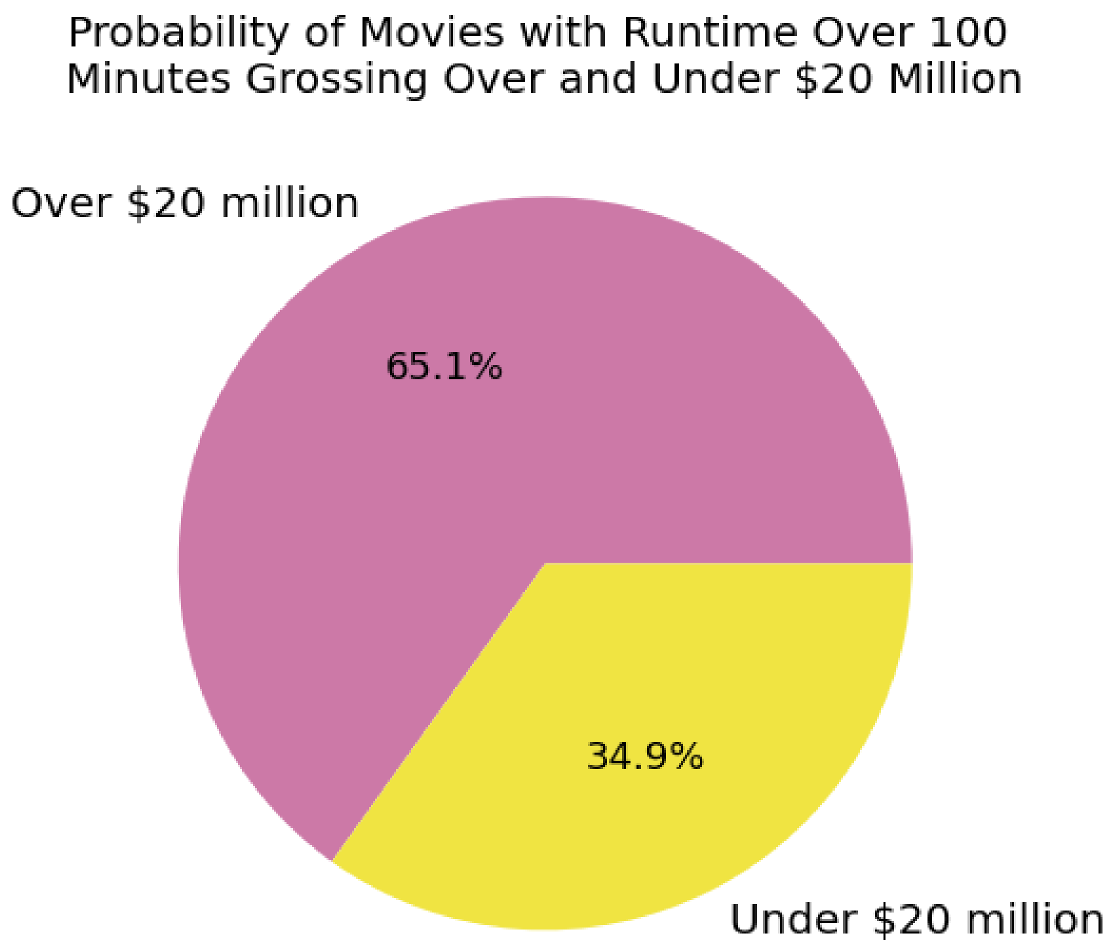
[23]:
```
(ggplot(countDF, aes(x = "Group", y = "Count", fill = "Group")) +
     geom_bar(stat="identity") +
     theme_minimal() +
     labs(x = "", y = "Count") +
     geom_text(aes(label = Count)) +
     scale_x_discrete(labels = ("IMDb score of 7\n or greater", "Runtime over␣
↪100 minutes")) +
     ggtitle("Comparing the Number of Two Categories of \nMovies that Grossed At␣
↪Least $20 Million") +
     scale_fill_manual(["#d55e00", "#0072b2"]) +
     theme(panel_grid_minor_x = element_blank(),
     panel_grid_minor_y = element_blank(),
     panel_grid_major_x = element_blank(),
     axis_text_x = element_text(size = 12),
     axis_title_x = element_text(size = 14),
     axis_text_y = element_text(size = 12),
     axis_title_y = element_text(size = 14),
     plot_title = element_text(lineheight = 1.5, size = 16),
     legend_text = element_text(size = 12),
     legend_position = "none"))
```




[23]: <ggplot: (8765016219742)>

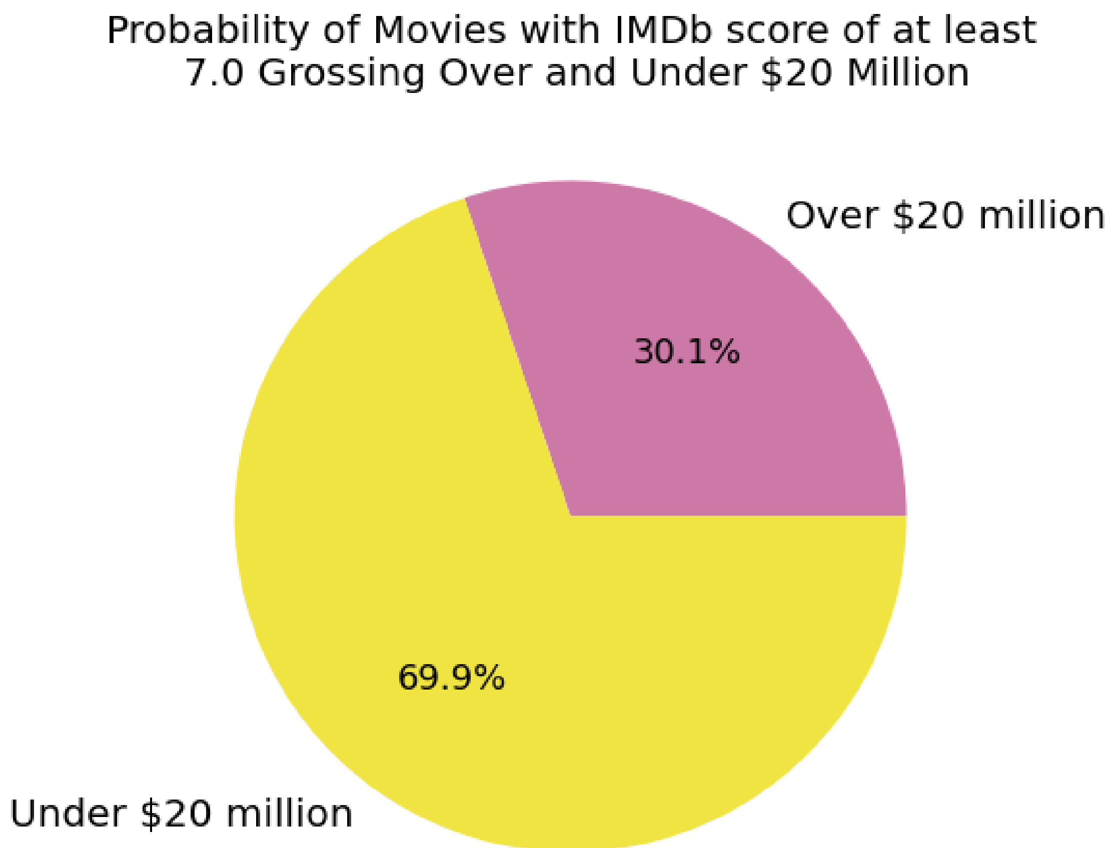
*Caption: Comparing how many movies that grossed over 20,000,000 dollars had an IMDb score of at least 7, as well as how many had a runtime over 100 minutes. As the chart shows, there was over double the number of movies that grossed at least 20,000,000 dollars that also had a runtime over 100 minutes, compared to the number of movies that grossed at least 20,000,000 dollars that also had an IMDb score of at least 7.*

[138]:
```
prob = [0.651137, 0.348863]
labels = ["Over $20 million", "Under $20 million"]
c1 = ["#cc79a7", "#f0e442"]
SMALL_SIZE = 18

plt.rc('font', size=SMALL_SIZE)
plt.figure(figsize=(8,8))
plt.title("Probability of Movies with Runtime Over 100 \nMinutes Grossing Over␣
 ↪and Under $20 Million")
plt.pie(prob, labels = labels, autopct = '%2.1f%%', colors = c1)
plt.show()
```




*Caption: Comparing the probability of grossing over and under 20,000,000 dollars for movies that have a runtime over 100 minutes. As the chart shows, movies with a runtime over 100 minutes are more likely to gross over 20,000,000 dollars than they are to gross under 20,000,000 dollars.*

[139]:
```
prob = [0.301174, 0.698826]
labels = ["Over $20 million", "Under $20 million"]
c1 = ["#cc79a7", "#f0e442"]
SMALL_SIZE = 18

plt.rc('font', size=SMALL_SIZE)
plt.figure(figsize=(8,8))
plt.title("Probability of Movies with IMDb score of at least\n 7.0 Grossing Over␣
↪and Under $20 Million")
plt.pie(prob, labels = labels, autopct = '%2.1f%%', colors = c1)
plt.show()
```




*Caption: Comparing the probability of grossing over and under 20,000,000 dollars for movies that have an IMDb score over 7.0. As the chart shows, movies with a runtime over 100 minutes are more likely to gross under 20,000,000 dollars than they are to gross over 20,000,000 dollars.*

### 0.4.1 4. Out of all the continuous/interval variables (budget, runtime, score, votes, year), which are the strongest predictors of gross?

## 0.5 a)

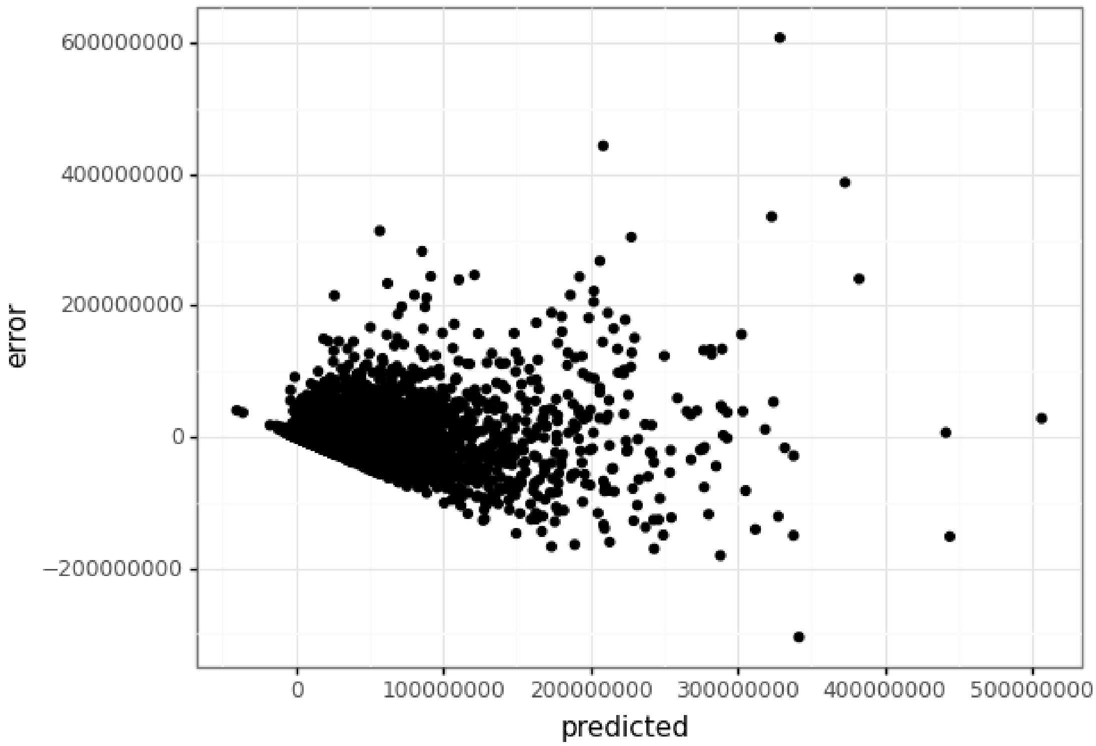
[6]:
```
features = ["budget", "runtime", "score", "votes", "year"]
X = df[features]
y = df["gross"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1) #
↪train test split

# z-scoring
zScore = StandardScaler()
```

```
Xz_train = zScore.fit_transform(X_train)
Xz_test = zScore.transform(X_test)

# create and fit model with training data
lr = LinearRegression()
lr.fit(Xz_train,y_train)

# mse
trainmse = mean_squared_error(y_train, lr.predict(Xz_train))
testmse = mean_squared_error(y_test, lr.predict(Xz_test))

# r^2
trainr2 = lr.score(Xz_train, y_train)
testr2 = lr.score(Xz_test, y_test)

print("\033[4mTRAINING SET\033[0m\nmse:", trainmse, "\nr2 :", trainr2)
print("\n\033[4mTEST SET\033[0m\nmse:", testmse, "\nr2 :", testr2)
```

```
TRAINING SET
mse: 1235341799227197.2
r2 : 0.6434543709820107

TEST SET
mse: 1243313117144092.5
r2 : 0.5362030598628513
```

[7]:

```
# check assumptions for linear regression
pred = lr.predict(Xz_train)

assump = pd.DataFrame({"error": y_train - pred, "predicted": pred})

ggplot(assump, aes(x = "predicted", y = "error")) + geom_point() + theme_bw()
```




[7]: <ggplot: (8761979023002)>

[27]:
```
# fit model with all non-z scored data
lr_all = LinearRegression()
lr_all.fit(X, y)

# create dataframe with coefficients
coef = pd.DataFrame({"Predictor": features, "Coefficient": lr_all.coef_})
coef.head()
```

[27]:

| | Predictor | Coefficient |
|---|---|---|
| 0 | budget | 0.828589 |
| 1 | runtime | -151477.874352 |
| 2 | score | 54683.160058 |
| 3 | votes | 189.052445 |
| 4 | year | -358206.084588 |

### 0.6 b)

Based on the analyses above, year and runtime are the strongest predictors of a movie's gross. Year had the coefficient of greatest magnitude of about -358,206. In other words, the release date of a movie increasing by 1 year is associated with a 358,206 dollar decrease in gross. Runtime had the coefficient of the second greatest magnitude of about -151,478. In other words, the runtime of a movie increasing by 1 minute is associated with a 151,478 dollar decrease in gross. In contrast, budget, score, and votes were much weaker predictors of gross. What this suggests is that as time passes, movies are likely to see a decrease in gross, and longer movies also tend to gross less.

To perform the analyses, we first split our data into test and training datasets. We included all the continuous variables as predictors for our analysis. Then, we standardized our data (i.e., put them on the same scale) by performing z-scoring. Next, we created a Linear Regression model, which is a type of machine learning model that can predict a continuous variable. In the context of this question, we used our model to predict gross. After fitting our model with the training dataset, we gauged our model's performance on both the training and test datasets using two metrics: mean squared error and r^2. Mean squared error (mse) reflects how successful our model is at predicting gross, with lower mse values reflecting better model performance. For both our training and test data, we got relatively large mse values, which suggests that our model was not very successful at predicting gross. R^2 reflects how much variation in the dataset is explained by our model, with r^2 values close to 1 reflecting more variation explained by our model and thus better model performance. For our training data, we got an r^2 of about 0.64, and for our test data, an r^2 of about 0.54. This suggests that our model's performance was relatively OK, but more importantly, it suggests that our model may be overfitted, or too specific to the data it was trained on. We can infer this from the difference between the two r^2 values; the r^2 value of our training set is somewhat greater than that of our test set, which suggests that our model performed well on the test data but slightly worse on unseen data.

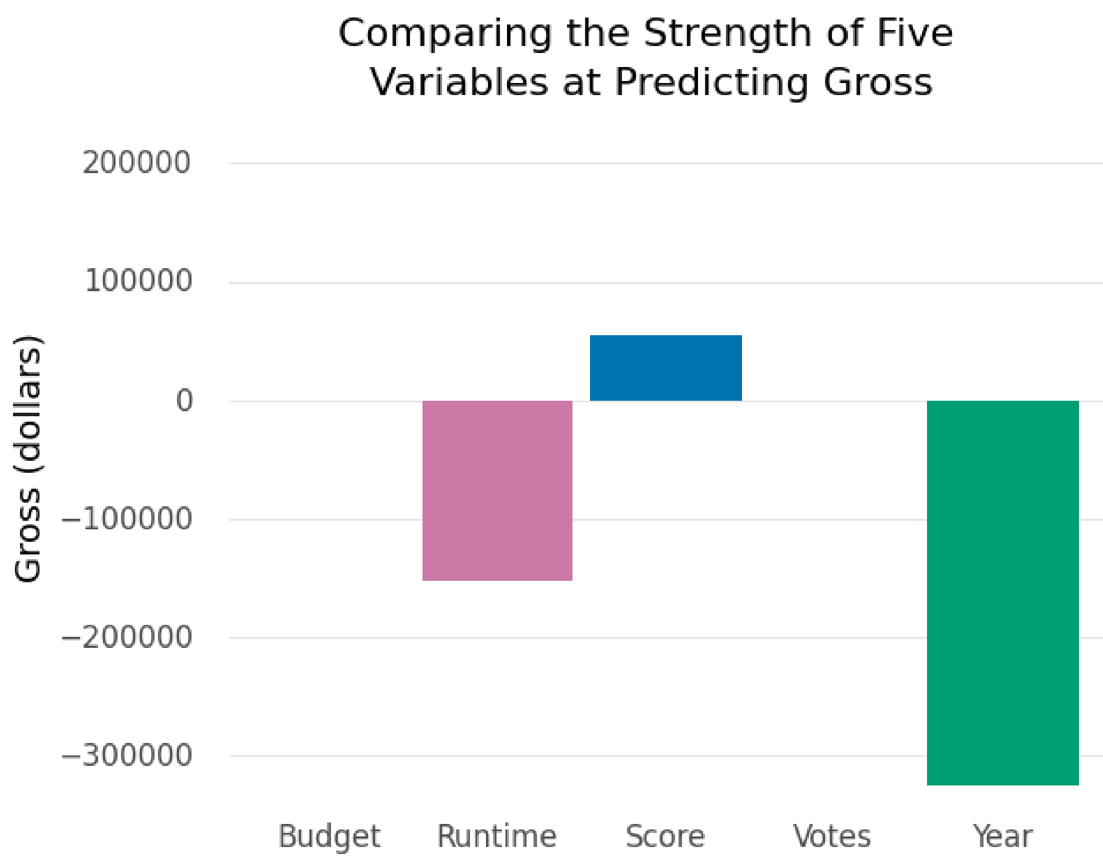
We then looked at the coefficients for each predictor. The coefficients represent the relationship between each predictor and the outcome variable (gross). We then plotted these values on bar graphs to directly compare them. The graphs confirm that year and runtime are the strongest predictors of (i.e., have the strongest relationship with) gross.

That being said, our model's predictions are questionable for several reasons. First, as explained above, our model's performance wasn't the best. Second, also as explained above, our model may have been overfitted. Third, and most importantly, our model may not have been best suited for this particular dataset in the first place. In order to use a Linear Regression model, several assumptions need to be made. We checked if our dataset met these assumptions by plotting the model's errors across all predicted values and looking at the variation in the graph. Based on this graph, we determined our dataset violated the assumption of homoskedasticity, which means that error should be evenly spread out throughout our model. In this case, it is violated because the error values clustered on one end of the graph and more spread out on the other. Given these issues, our answer to this question should be taken with a grain of salt.

### 0.7 c)

[100]:
```
(ggplot(coef, aes(x = "Predictor", y = "Coefficient", fill = "Predictor")) +
     geom_bar(stat="identity") +
     theme_minimal() +
     labs(x = "", y = "Gross (dollars)") +
     ggtitle("Comparing the Strength of Five \nVariables at Predicting Gross") +
     coord_cartesian(ylim = (-300000,200000)) +
     scale_fill_manual(["#d55e00", "#cc79a7", "#0072b2", "#f0e442", "#009E73"]) +
     scale_x_discrete(labels = ("Budget", "Runtime", "Score", "Votes", "Year")) +
     theme(panel_grid_minor_x = element_blank(),
```

```
    panel_grid_minor_y = element_blank(),
    panel_grid_major_x = element_blank(),
    axis_text_x = element_text(size = 12),
    axis_title_x = element_text(size = 14),
    axis_text_y = element_text(size = 12),
    axis_title_y = element_text(size = 14),
    plot_title = element_text(lineheight = 1.5, size = 16),
    legend_text = element_text(size = 12),
    legend_position = "none"))
```

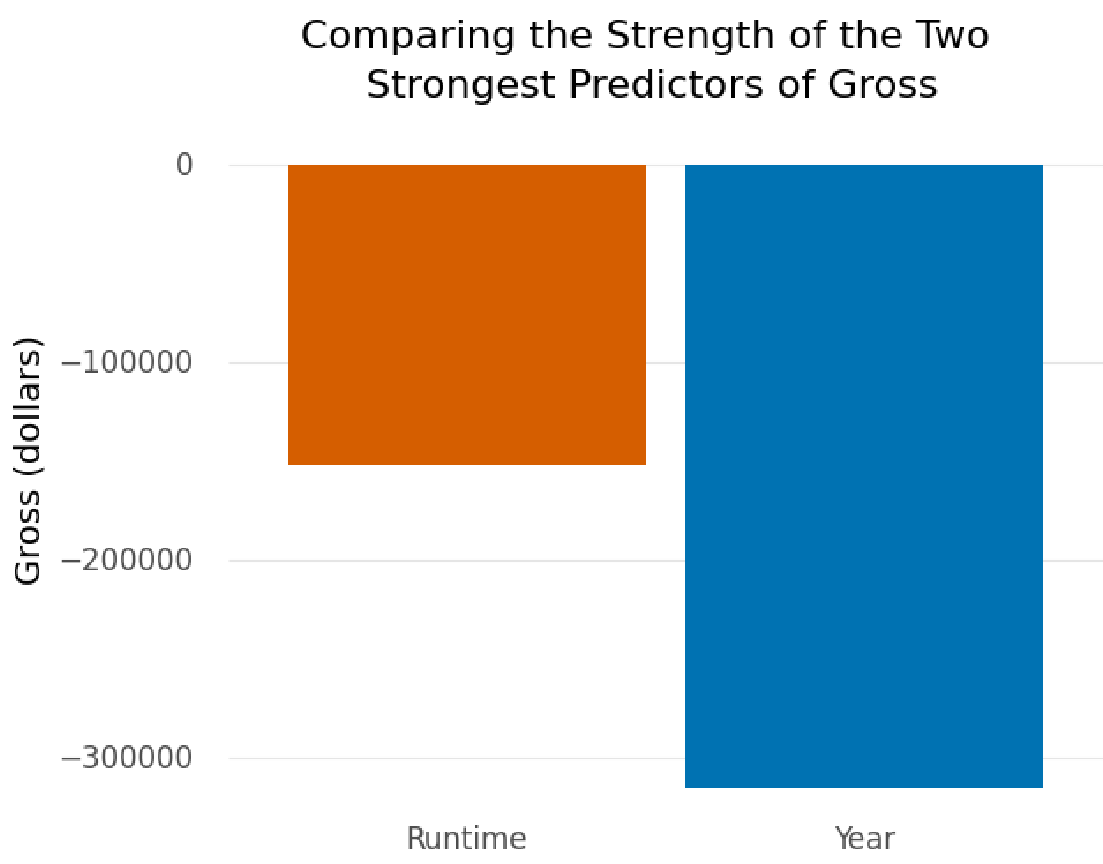



[100]: <ggplot: (8761979387709)>

*Caption: Comparing the strength of the relationship of all the predictors, which are all continuous variables, to gross, as determined by the Linear Regression model. As the graph shows, year and runtime are the strongest predictors of gross. Votes and budget do not appear to be visible because their coefficient values are much smaller than that of runtime, score, and year.*

[98]:
```
dfcoef = coef.loc[[1, 4], :]

(ggplot(dfcoef, aes(x = "Predictor", y = "Coefficient", fill = "Predictor")) +
```

```
    geom_bar(stat="identity") +
    theme_minimal() +
    labs(x = "", y = "Gross (dollars)") +
    ggtitle("Comparing the Strength of the Two \nStrongest Predictors of␣
↪Gross") +
    coord_cartesian(ylim = (-300000,0)) +
    scale_fill_manual(["#d55e00", "#0072b2"]) +
    scale_x_discrete(labels = ("Runtime", "Year")) +
    theme(panel_grid_minor_x = element_blank(),
    panel_grid_minor_y = element_blank(),
    panel_grid_major_x = element_blank(),
    axis_text_x = element_text(size = 12),
    axis_title_x = element_text(size = 14),
    axis_text_y = element_text(size = 12),
    axis_title_y = element_text(size = 14),
    plot_title = element_text(lineheight = 1.5, size = 16),
    legend_text = element_text(size = 12),
    legend_position = "none"))
```




[98]: <ggplot: (8761980383970)>

*Caption: Comparing the strength of the relationship of the two strongest predictors, runtime and year, to gross, as determined by the Linear Regression model. As the graph shows, year is the strongest predictor of gross, and both predictors have a negative relationship with gross, meaning that as each predictor increases, gross decreases.*

### 0.7.1 5. Is the model most accurate at predicting whether a movie grosses over or under 20 million dollars for movies with a PG rating, PG-13 rating, or R rating?

## 0.8 a)

[92]:
```
cont_vars = ["budget", "runtime", "score", "votes", "year", "gross", "rating"]
data_gross = df[cont_vars]

gross_over_20mil = []

for i in data_gross["gross"]:
    if i > 20000000:
        gross_over_20mil.append(1)
    if i < 20000000:
        gross_over_20mil.append(0)

data_gross["gross over 20mil"] = gross_over_20mil
data_gross.head()
```

[92]:
```
      budget  runtime  score   votes  year        gross rating  \
0  8000000.0       89    8.1  299174  1986   52287414.0      R
1  6000000.0      103    7.8  264740  1986   70136369.0  PG-13
2 15000000.0      110    6.9  236909  1986  179800601.0     PG
3 18500000.0      137    8.4  540152  1986   85160248.0      R
4  9000000.0       90    6.9   36636  1986   18564613.0     PG

   gross over 20mil
0                 1
1                 1
2                 1
3                 1
4                 0
```

[93]:
```
features1 = ["budget", "runtime", "score", "votes", "year"]
X1 = data_gross[features1]
y1 = data_gross["gross over 20mil"]

X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size = 0.1)␣
 ↪# train test split

# z-scoring
zScore1 = StandardScaler()
Xz_train1 = zScore1.fit_transform(X_train1)
```

```
Xz_test1 = zScore1.transform(X_test1)
```

[94]:
```
# create and fit logistic regression model with training data
myLogit = LogisticRegression()
myLogit.fit(Xz_train1, y_train1)

# acc
trainacc = accuracy_score(y_train1, myLogit.predict(Xz_train1))
testacc = accuracy_score(y_test1, myLogit.predict(Xz_test1))

print("\033[4mTRAINING SET ACC\033[0m", trainacc)
print("\n\033[4mTEST SET ACC\033[0m", testacc)
```

```
TRAINING SET ACC 0.8172043010752689

TEST SET ACC 0.8372434017595308
```

[95]:
```
# filter data based on PG rating
PG = data_gross["rating"] == "PG"
dfPG = data_gross.loc[PG]
xPG = dfPG[features1]
yPG = dfPG["gross over 20mil"]
acc_PG = accuracy_score(yPG, myLogit.predict(xPG))
print("Average accuracy for movies with a PG rating:", acc_PG)

# filter data based on PG-13 rating
PG13 = data_gross["rating"] == "PG-13"
dfPG13 = data_gross.loc[PG13]
xPG13 = dfPG13[features]
yPG13 = dfPG13["gross over 20mil"]
acc_PG13 = accuracy_score(yPG13, myLogit.predict(xPG13))
print("Average accuracy for movies with a PG-13 rating:", acc_PG13)

# filter data based on R rating
R = data_gross["rating"] == "R"
dfR = data_gross.loc[R]
xR = dfR[features]
yR = dfR["gross over 20mil"]
acc_R = accuracy_score(yR, myLogit.predict(xR))
print("Average accuracy for movies with an R rating:", acc_R)
```

```
Average accuracy for movies with a PG rating: 0.5488958990536278
Average accuracy for movies with a PG-13 rating: 0.5704260651629073
Average accuracy for movies with an R rating: 0.29274764150943394
```

## 0.9 b)

Based on the analyses above, the model is most accurate at predicting whether a movie grosses over or under 20,000,000 dollars for movies with a PG-13 rating. It is similar but slightly less accurate for movies with a PG rating and least accurate for movies with an R-rating. What this suggests is that it is harder to predict whether a movie will gross at least 20,000,000 dollars if a movie is rated R. For movies rated PG and PG-13 it is easier to predict if they will gross at least 20,000,000 dollars. So, the optimal use of this model would be to predict the gross of movies that are rated PG and PG-13, not rated R.

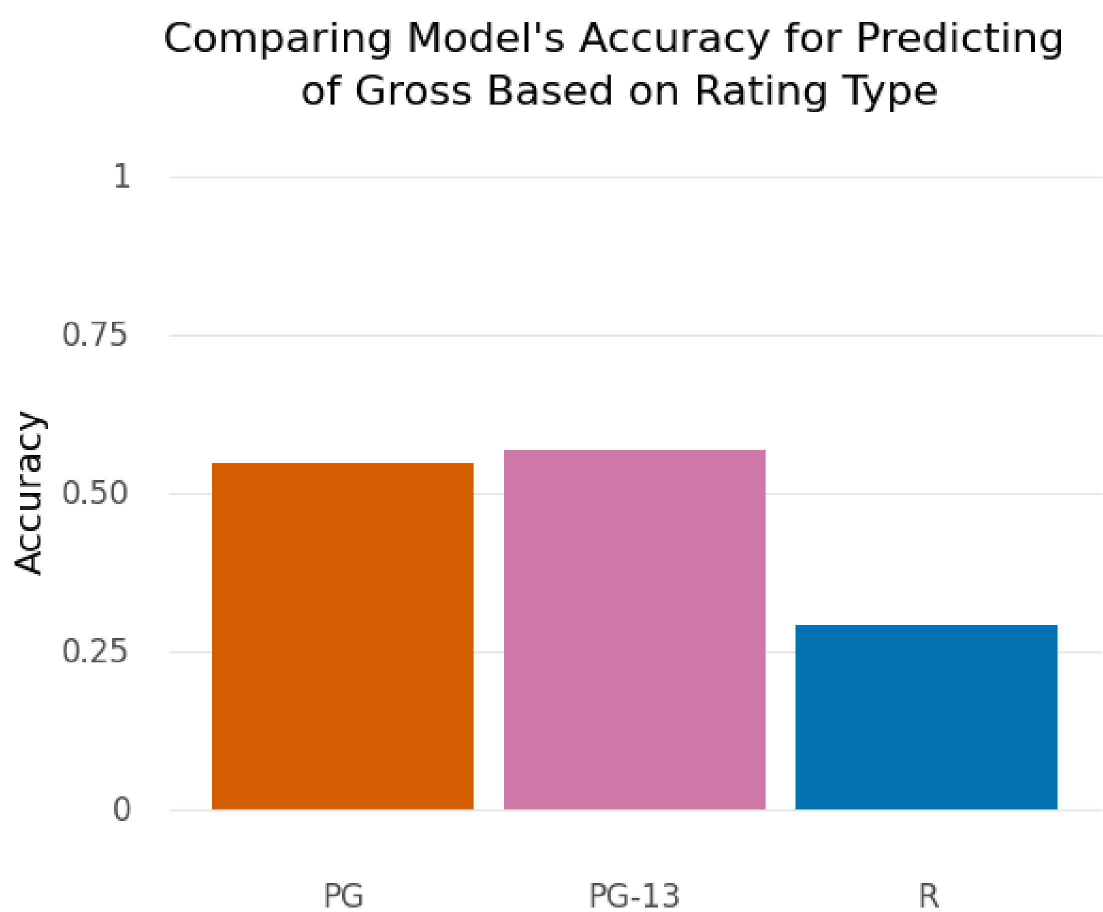
To perform the analyses, we first created a new column in our dataset that showed if each movie grossed over 20,000,000 dollars (1) or under 20,000,000 dollars (0). Then, after we split our data into test and training datasets, we standardized our data (i.e., put them on the same scale) by performing z-scoring. Next, we created a Logistic Regression model, which is a type of machine learning model that can predict things into two categories. In the context of this question, we used our model to predict if movies grossed above 20,000,000 dollars or below 20,000,000 dollars. After fitting our model with the training dataset, we calculated the accuracy of our model on both the training data and test data, and we found that our model achieved accuracies of about 0.82 and 0.84, respectively. Not only do these accuracies indicate that our model is relatively good at predicting whether a movie grossed above or below 20,000,000 dollars, but they also indicate that our model is not overfitted, or too specific to the data it was trained on. We can infer this because the two accuracies are similar, so there is not a big difference in performance for our model on seen and unseen data.

We then specifically looked at how our model's performance varies based on if the movie is rated PG, PG-13, or R by filtering our dataset based on these three rating types. Then, we calculated the model's accuracy at predicting gross for each of the rating types and plotted these values on a bar and line graph in order to directly compare them. The graphs confirm that our model performs best (i.e., accurately predict if a movie grossed above or below 20,000,000 dollars) for rated PG and PG-13 movies, while its performance is noticeably worse for rated R movies.

## 0.10 c)

[96]:
```
# make a new data frame with just the income groups and their accuracy scores
rating = ["PG", "PG-13", "R"]
acc = [acc_PG, acc_PG13, acc_R]
mylist = {"Rating": rating, "Accuracy": acc}
df_ratings = pd.DataFrame(mylist)

# used to set the x-axis in this specific order
positions = ("PG", "PG-13", "R")

(ggplot(df_ratings, aes(x = "Rating", y = "Accuracy", fill = "Rating")) +
     geom_bar(stat="identity") +
     theme_minimal() +
     labs(x = "", y = "Accuracy") +
     ggtitle("Comparing Model's Accuracy for Predicting \nof Gross Based on␣
 ↪Rating Type") +
     scale_fill_manual(["#d55e00", "#cc79a7", "#0072b2"]) +
```

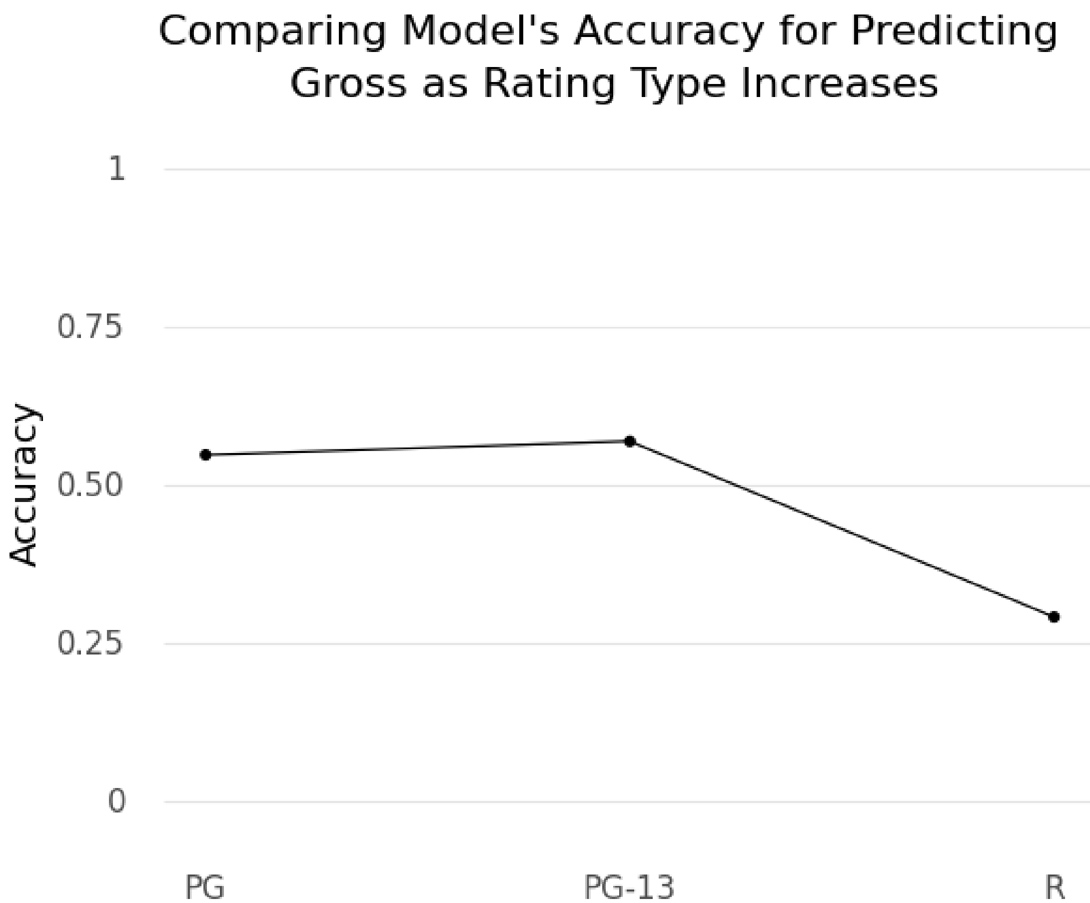
```
    scale_x_discrete(limits = positions) +
    coord_cartesian(ylim = (0, 1)) +
    theme(panel_grid_minor_x = element_blank(),
    panel_grid_minor_y = element_blank(),
    panel_grid_major_x = element_blank(),
    axis_text_x = element_text(size = 12),
    axis_title_x = element_text(size = 14),
    axis_text_y = element_text(size = 12),
    axis_title_y = element_text(size = 14),
    plot_title = element_text(lineheight = 1.5, size = 16),
    legend_text = element_text(size = 12),
    legend_position = "none"))
```




[96]: <ggplot: (8761979476914)>

*Caption: Comparing how the Logistic Regression model's accuracy at predicting whether a movie grossed over 20,000,000 dollars or under 20,000,000 dollars varies based on the rating type of the movie. An accuracy of 0 means that the model never got any of its predictions correct, while an accuracy of 1 means that a model got all of its predictions correct. As the graph shows, the model performs similarly for rated PG* *and PG-13 moves, but relatively worse for rated R movies.*

[217]:
```
rating1 = [1, 2, 3]
acc = [acc_PG, acc_PG13, acc_R]
mylist1 = {"Rating": rating1, "Accuracy": acc}
df_ratings1 = pd.DataFrame(mylist1)

(ggplot(df_ratings1, aes(x = "Rating", y = "Accuracy")) +
     geom_point() +
     geom_line() +
     theme_minimal() +
     labs(x = "", y = "Accuracy") +
     ggtitle("Comparing Model's Accuracy for Predicting \nGross as Rating Type␣
 ↪Increases") +
     coord_cartesian(ylim = (0, 1)) +
     scale_x_continuous(breaks = [1,2,3], labels = ("PG","PG-13","R")) +
     theme(panel_grid_minor_x = element_blank(),
     panel_grid_minor_y = element_blank(),
     panel_grid_major_x = element_blank(),
     axis_text_x = element_text(size = 12),
     axis_title_x = element_text(size = 14),
     axis_text_y = element_text(size = 12),
     axis_title_y = element_text(size = 14),
     plot_title = element_text(lineheight = 1.5, size = 16),
     legend_text = element_text(size = 12),
     legend_position = "none"))
```




[217]: <ggplot: (8761838690865)>

*Caption: Comparing how the Logistic Regression model's accuracy at predicting whether a movie grossed over 20,000,000 dollars or under 20,000,000 dollars varies as the rating type of the movie increases, or the movies become more restricted. An accuracy of 0 means that the model never got any of its predictions correct, while an accuracy of 1 means that a model got all of its predictions correct. As the graph shows, the model performs similarly for rated PG and PG-13 moves, but its performance declines for rated R movies.*

# 10

## Question 6

### 10.1 What is the minimum number of features needed to predict whether a movie will gross over 250K and over 500K with at least 70% explained variance?

[24]:
```
cont_features = ['budget', 'runtime', 'score', 'votes', 'year released', 'gross']
cont_predictors = ['budget', 'runtime', 'score', 'votes', 'year released']
```

[25]:
```
data_gross_cont_filtered = data[cont_features]
data_gross_cont_filtered.tail()
```

[25]:

| | budget | runtime | score | votes | year released | gross |
|---|---|---|---|---|---|---|
| 6815 | 0 | 91 | 5.4 | 9161 | 2016 | 4750497 |
| 6816 | 0 | 90 | 4.9 | 1959 | 2016 | 28368 |
| 6817 | 3500000 | 76 | 6.5 | 36333 | 2016 | 3775000 |
| 6818 | 0 | 76 | 6.2 | 6947 | 2016 | 25981 |
| 6819 | 0 | 120 | 6.7 | 2411 | 2016 | 37757 |

#### 10.1.1 Creating new columns of binary outcome type data. One column for whether a movie made over 250K or not and another column for whether a movie made over 500K or not. A value of 1 represents 'True' and a value of 0 represents 'False':

[26]:
```
z = StandardScaler()
gross_over_250k = []
gross_over_500k = []
for i in data_gross_cont_filtered['gross']:
    if i > 250000:
        gross_over_250k.append(1)
    else:
        gross_over_250k.append(0)
    if i > 500000:
        gross_over_500k.append(1)
    else:
        gross_over_500k.append(0)

print(len(gross_over_250k))
print(len(gross_over_500k))
```

```
6820
6820
```

## 10.2 Showing that the data frame named 'data_gross_cont_filtered' contains the 2 new binary outcome columns:

[27]:
```
data_gross_cont_filtered['gross_over_250k'] = gross_over_250k
data_gross_cont_filtered['gross_over_500k'] = gross_over_500k
data_gross_cont_filtered.tail(8)
```

[27]:
```
        budget  runtime  score  votes  year released     gross  \
6812         0       96    5.7   4439           2016     23020
6813         0      120    6.2   6054           2016    228894
6814  20000000      107    6.3  19084           2016  36874745
6815         0       91    5.4   9161           2016   4750497
6816         0       90    4.9   1959           2016     28368
6817   3500000       76    6.5  36333           2016   3775000
6818         0       76    6.2   6947           2016     25981
6819         0      120    6.7   2411           2016     37757

      gross_over_250k  gross_over_500k
6812                0                0
6813                0                0
6814                1                1
6815                1                1
6816                0                0
6817                1                1
6818                0                0
6819                0                0
```

## 10.3 PCA Models:

### 10.3.1 PCA Model for predicting gross over 250K

[28]:
```
PCA_LR_Model_250k = LogisticRegression() # init an empty Logistic Regression␣
↪model

# Use TTS with a 90/10 split (since data is large)
PCA_LR_X_train_250k, PCA_LR_X_test_250k, PCA_LR_y_train_250k, PCA_LR_y_test_250k␣
↪= train_test_split(data_gross_cont_filtered[cont_predictors],␣
↪data_gross_cont_filtered["gross_over_250k"], test_size=0.1)

# z-score predictors
PCA_LR_X_train_250k[cont_predictors] = z.
↪fit_transform(PCA_LR_X_train_250k[cont_predictors]) # z-score and fit bc model␣
↪is trained with train data
PCA_LR_X_test_250k[cont_predictors] = z.
↪transform(PCA_LR_X_test_250k[cont_predictors]) # z-score but do not fit bc do␣
↪not want to leak test data into model
```

```
PCA_Model_250k = PCA()
PCA_Model_250k.fit(PCA_LR_X_train_250k)
```

[28]: PCA()

[29]:
```
# mapping of both training and testing set to the PCA Model
PCA_LR_X_train_250k = PCA_Model_250k.transform(PCA_LR_X_train_250k)
PCA_LR_X_test_250k = PCA_Model_250k.transform(PCA_LR_X_test_250k)

# apply PCA to the training set
PCA_LR_Model_250k.fit(PCA_LR_X_train_250k, PCA_LR_y_train_250k) # fit the X and␣
↪y training data to the LR model

PCA_LR_y_pred_250k = PCA_LR_Model_250k.predict(PCA_LR_X_test_250k)

PCA_LR_mse_250k = mean_squared_error(PCA_LR_y_test_250k, PCA_LR_y_pred_250k)
PCA_LR_r2_250k = r2_score(PCA_LR_y_test_250k, PCA_LR_y_pred_250k)

print("PCA Logistic Regression Model ~ Mean Squared Error:\n" +␣
↪str(round(PCA_LR_mse_250k, 3)) + "\n")
print("PCA Logistic Regression Model ~ r2 score:\n" +␣
↪str(abs(round(PCA_LR_r2_250k, 3))))
```

```
PCA Logistic Regression Model ~ Mean Squared Error:
0.135

PCA Logistic Regression Model ~ r2 score:
0.189
```

## 10.4 Discussion of Logistic Regressin Model's (for predicting gross over 250K) results

### 10.4.1 What Mean Squared Error (mse) is and why it is important and used in this context:

- MSE is being used as a metric to measure the model's performance because it is a good metric to use to check how close the model's forecasts are to actual results. The mean squared error is sum of squared errors divided by the number of data points and is considered a loss function because it is a measure of well a model is doing. The mean squared error value tells approximately what error value can be expected from any data point on the Logistic Regression (LR) model. Like the sum of squared errors, the lower the mean squared error is (relative to the outcome units squared), the better the LR model is at predicting the outcome variable (y).

### 10.4.2 Interpretation of mse from the Linear Regression Model:

- The mean squared error for the logistic regression model is about 0.139 as shown above. As discussed before, the mse is in terms of the outcome units squared. In this LR model, the y-value is gross in US dollars and so the error is simply US dollars squared. This error value is very small given that the units are dollars squared, however, it is difficult to make conclusions from the mse. The mse will be more helpful later on when it is compared to the PCA model's mse value. This is because comparing the values will provide insight on how much the error changed from using less components. To help get a better idea of how well the LR model is doing without comparing it to another model (PCA), r2 is calculated next. r2 is generally more insightful since it gives a standardized score (between 0 and 1).

#### 10.4.3 What r2 is and why it is important and used in this context:

- R2 is being used as a metric to measure the model's performance because it provides an understanding of the strength of the relationship between the predictor variables (budget, score, votes, etc) and the outcome (gross) in a standard scale (0 - 1). r2 represents the percentage of variance that is explained by the model. The closer the percentage or decimal value of r2 is to 1.0, the more the variation is explained by the model (as opposed to external factors/noises). In constrast, an r2 of 0 or close to 0 is an indicator that the model does a poor job of predicting the outcome because the variance is not explained by the model.

#### 10.4.4 Interpretation of r2 from the Linear Regression Model:

- The r2 value is very low, 0.123, as shown above. This low r2 value indicates the model is performing very poorly at predicting the outcome variable (gross) because the variation in our model's results are not being explained from the model itself. It is desired for the variation of a model to be explained by the predictors/features because that implies that the features are great choices for predicting the outcome variable of interest.

#### 10.4.5 PCA Model for predicting gross over 500K

[30]:
```
PCA_LR_Model_500k = LogisticRegression() # init an empty Logistic Regression␣
↪model

# Use TTS with a 90/10 split (since data is large)
PCA_LR_X_train_500k, PCA_LR_X_test_500k, PCA_LR_y_train_500k, PCA_LR_y_test_500k␣
↪= train_test_split(data_gross_cont_filtered[cont_predictors],␣
↪data_gross_cont_filtered["gross_over_500k"], test_size=0.1)

# z-score predictors
PCA_LR_X_train_500k[cont_predictors] = z.
↪fit_transform(PCA_LR_X_train_500k[cont_predictors]) # z-score and fit bc model␣
↪is trained with train data
PCA_LR_X_test_500k[cont_predictors] = z.
↪transform(PCA_LR_X_test_500k[cont_predictors]) # z-score but do not fit bc do␣
↪not want to leak test data into model

PCA_Model_500k = PCA()
PCA_Model_500k.fit(PCA_LR_X_train_500k)
```

[30]:
```
PCA()
```

[31]:
```
# mapping of both training and testing set to the PCA Model
PCA_LR_X_train_500k = PCA_Model_250k.transform(PCA_LR_X_train_500k)
PCA_LR_X_test_500k = PCA_Model_250k.transform(PCA_LR_X_test_500k)

# apply PCA to the training set
PCA_LR_Model_500k.fit(PCA_LR_X_train_500k, PCA_LR_y_train_500k) # fit the X and␣
↪y training data to the LR model

PCA_LR_y_pred_500k = PCA_LR_Model_500k.predict(PCA_LR_X_test_500k)

PCA_LR_mse_500k = mean_squared_error(PCA_LR_y_test_500k, PCA_LR_y_pred_500k)
PCA_LR_r2_500k = r2_score(PCA_LR_y_test_500k, PCA_LR_y_pred_500k)

print("PCA Logistic Regression Model ~ Mean Squared Error:\n" +␣
↪str(round(PCA_LR_mse_500k, 3)) + "\n")
print("PCA Logistic Regression Model ~ r2 score:\n" +␣
↪str(round(abs(PCA_LR_r2_500k), 3)))
```

```
PCA Logistic Regression Model ~ Mean Squared Error:
0.145

PCA Logistic Regression Model ~ r2 score:
0.081
```

### 10.5 Discussion of Logistic Regressin Model's (for predicting gross over 500K) results

#### 10.5.1 Interpretation of mse from the Linear Regression Model:

- The mean squared error for the logistic regression model is about 0.151 as shown above. This error value is very small given that the units are dollars squared, however, it is difficult to make conclusions from the mse. The mse will be more helpful later on when it is compared to the PCA model's mse value. To help get a better idea of how well the LR model is doing without comparing it to another model (PCA), r2 is calculated next.

#### 10.5.2 Interpretation of r2 from the Linear Regression Model:

- The r2 value is very low, 0.085, as shown above. This extremely low r2 value indicates the model is performing very poorly at predicting the outcome variable (gross) because the variation in our model's results are not being explained from the model itself.

### 10.6 General commentary on the LR Models' performances

- The 2 Logistic Regression Models are performing very poorly in predicting their outcomes (gross). This tells us that the predictors - [budget, runtime, score, votes, year released] are terrible in predicting gross. And so **these LR Models are terrible in accomplishing what they were intended to predict, however, the Principle Component Analysis will still be performed next to demonstrate that nearly identical results can be achieved with less principle components (AKA predictors)**.

## 10.7 Creating Dataframes of Principle Components

### 10.7.1 Principle Components Dataframe for model predicting gross over 250K:

[32]:
```
PCA_DF_250k = pd.DataFrame({
    "Explained_Variance": PCA_Model_250k.explained_variance_ratio_,
    "Principle_Components": range(1, 6),
    "Cumulative_Variance": PCA_Model_250k.explained_variance_ratio_.cumsum()
})

PCA_DF_250k.head()
```

[32]:

| | Explained_Variance | Principle_Components | Cumulative_Variance |
|---|---|---|---|
| 0 | 0.422529 | 1 | 0.422529 |
| 1 | 0.220652 | 2 | 0.643182 |
| 2 | 0.157672 | 3 | 0.800854 |
| 3 | 0.127899 | 4 | 0.928753 |
| 4 | 0.071247 | 5 | 1.000000 |

### 10.7.2 Principle Components Dataframe for model predicting gross over 500K:

[33]:
```
PCA_DF_500k = pd.DataFrame({
    "Explained_Variance": PCA_Model_500k.explained_variance_ratio_,
    "Principle_Components": range(1, 6),
    "Cumulative_Variance": PCA_Model_500k.explained_variance_ratio_.cumsum()
})

PCA_DF_500k.head()
```

[33]:

| | Explained_Variance | Principle_Components | Cumulative_Variance |
|---|---|---|---|
| 0 | 0.424021 | 1 | 0.424021 |
| 1 | 0.218153 | 2 | 0.642174 |
| 2 | 0.160060 | 3 | 0.802234 |
| 3 | 0.126941 | 4 | 0.929175 |
| 4 | 0.070825 | 5 | 1.000000 |

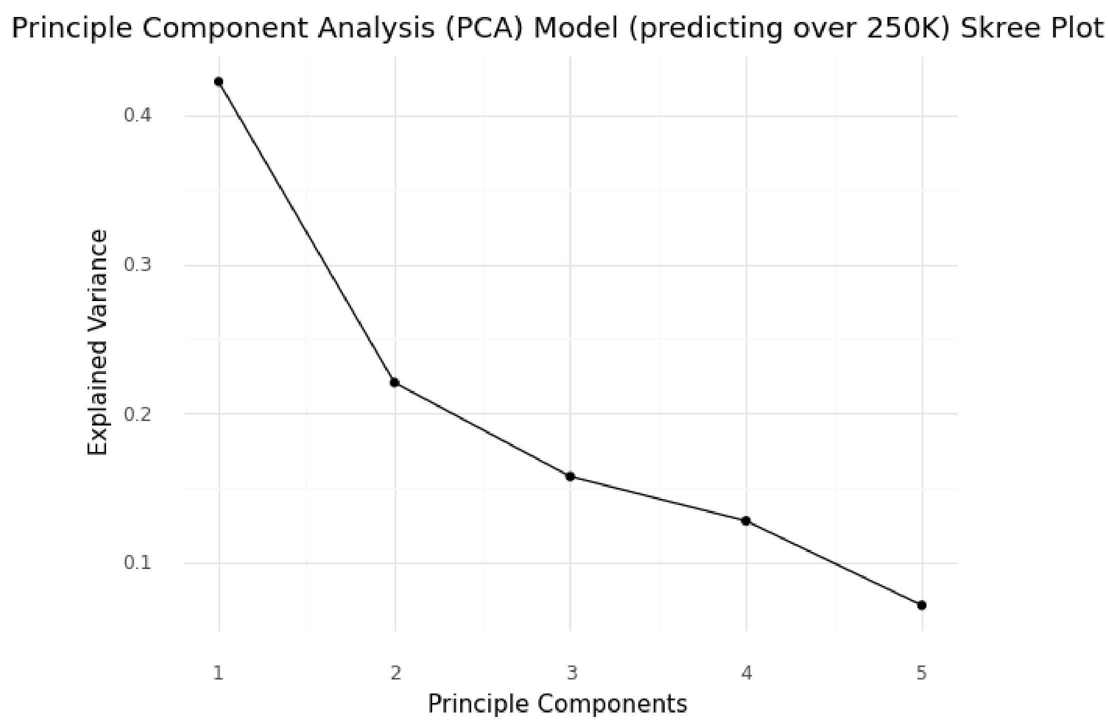
## 10.8 Explanation principle components and explained variance:

- The priniciple components are the features: budget, runtime, score, votes, and year released. The explained variance is the percentage of variance that is being explained by the model. It is desirable for the explained variance to be as close to 100% as possible. This is because we want variation from the model to be explained from the predictors (AKA principle components) as opposed to outside noise.

### 10.9 Creating the PCA Skree Plots

#### 10.9.1 PCA Skree Plots for Model predicting gross over 250K:

[34]:
```
# pca a scree plot
(ggplot(PCA_DF_250k, aes(x = "Principle_Components", y = "Explained_Variance"))
 + geom_point()
 + geom_line()
 + theme_minimal()
 + ggtitle("Principle Component Analysis (PCA) Model (predicting over 250K)␣
↪Skree Plot")
 + labs(x = "Principle Components", y = "Explained Variance")
)
```




[34]: `<ggplot: (323232419)>`

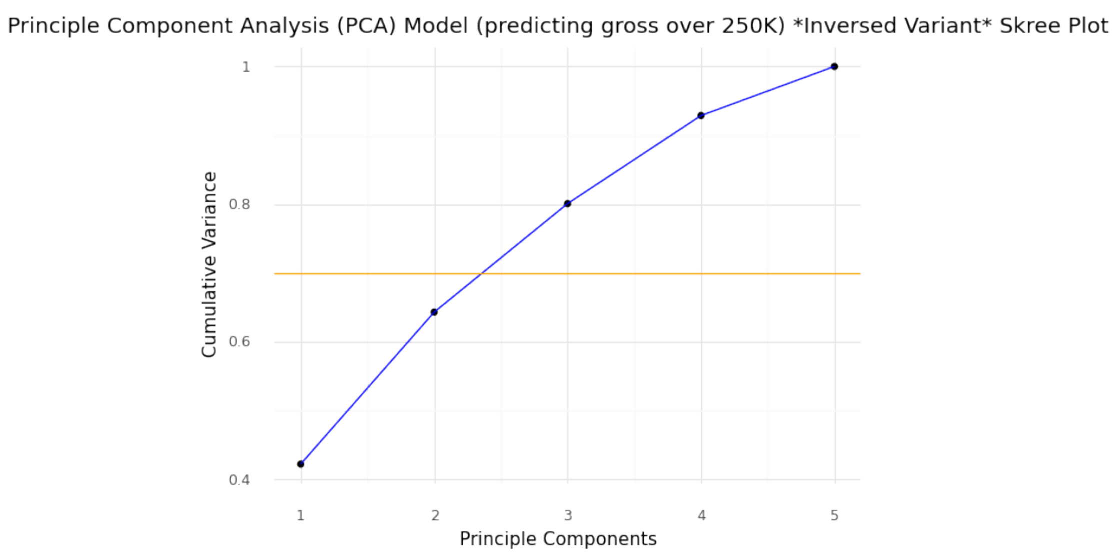
*Caption: Skree plot showing how much variance in the data is accounted for by each principle component in the model, as determined using PCA. The model is predicting whether a movie will gross over or under 250,000 dollars.*

### 10.10 Explanation and interpretation of the Skree Plot for the PCA Model (predicting over 250K)

- The skree plot is a scatter plot that visually represents how much much variation is being explained from the addition of a principle component. For example looking at the first principle component, it can observed that about 42% of the variance is being explained by just the first principle component. In other words if a PCA model were to be created with only that first principle component, the results of the PCA model's predictions would be about 42% explained from that one predictor. The second principle component has an explained variance of about 22%. This indicates that the second component can explain about 22% of the model's variance. If the first 2 principle components' explained variances are combined, a cumulative variance of about 64% would be achieved. This technique of principle component analysis is very powerful because it allows data scientists to minimize the number of principle components they use in a model to achieve a desired expected variation percentage. It is important to minimize these principle components because the more principle components that are used in a model, the more computationally expensive it is to get calculations and predictions from a model.

### 10.10.1 *Inversed Variant* PCA Skree Plots for Model predicting gross over 250K:

[35]:
```
# Figure out how many PCs you need to keep to retain 70% of the original␣
↪variance.
(ggplot(PCA_DF_250k, aes(x = "Principle_Components", y = "Cumulative_Variance"))
 + geom_point()
 + geom_line(color = "blue")
 + geom_hline(yintercept = 0.70, color = "orange")
 + theme_minimal() + ggtitle("Principle Component Analysis (PCA) Model␣
↪(predicting gross over 250K) *Inversed Variant* Skree Plot") + labs(x =␣
↪"Principle Components", y = "Cumulative Variance")
)
```




[35]: <ggplot: (322868502)>

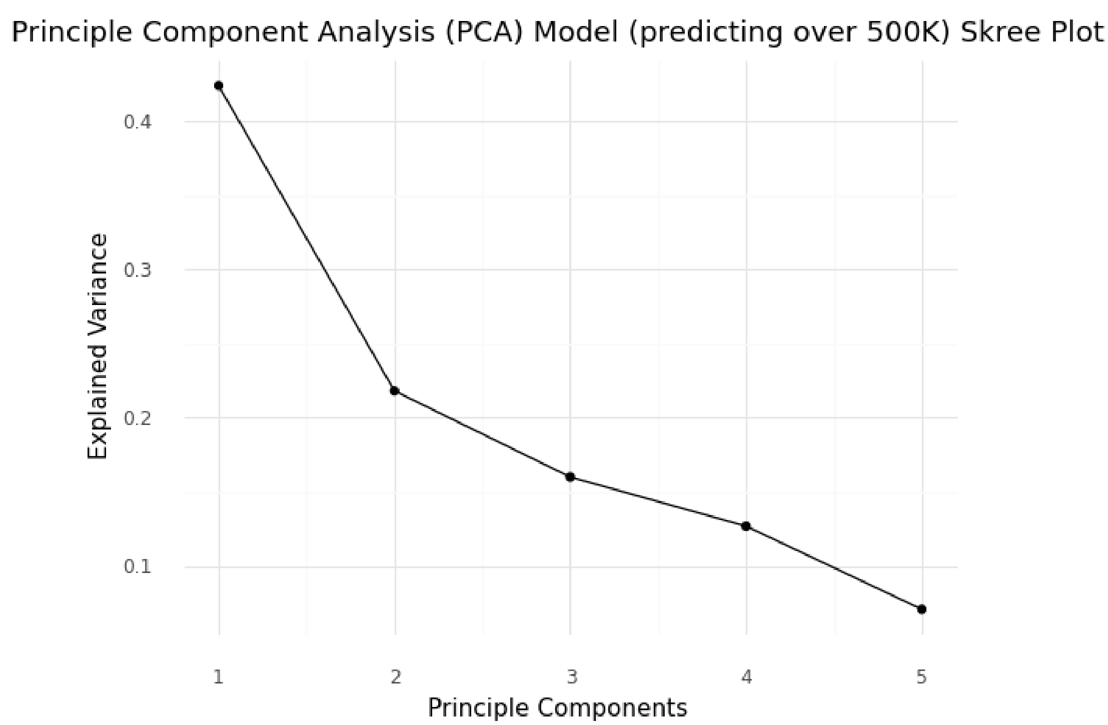
*Caption: Skree plot showing how much variance in the data is cumulatively accounted for by each principle* *component, as determined using PCA. The model is predicting whether a movie will gross over or under 250,000 dollars. The horizontal line at y = 0.7 represents 70% cumulative variance and can be used to determine how many principle components are needed to explain 70% of the variance in the data.*

## 10.11 Discussion of Inversed Variant Skree Plot for the PCA Model (predicting gross over 250K)

- The graph above is another way to visualize the amount of explained variance for principle components. This graph depicts the cumulative variance with the addition of each principle components. For example, at principle components equals to 2, the cumulative variance explained by the first 2 principle components is its respected y-value. The orange horizontal line is the minimum amount of explained variance, 70%, that is desired for this problem. **It can be observed that at least 3 principle components are needed to create a Logistic Regression model predicting a gross over 250K with at least 70% explained variance.**

### 10.11.1 PCA Skree Plots for Model predicting gross over 500K:

[36]:
```
# pca a scree plot
(ggplot(PCA_DF_500k, aes(x = "Principle_Components", y = "Explained_Variance"))
 + geom_point()
 + geom_line()
 + theme_minimal()
 + ggtitle("Principle Component Analysis (PCA) Model (predicting over 500K)
 ↪Skree Plot")
 + labs(x = "Principle Components", y = "Explained Variance")
)
```



[36]:
```
<ggplot: (323292925)>
```

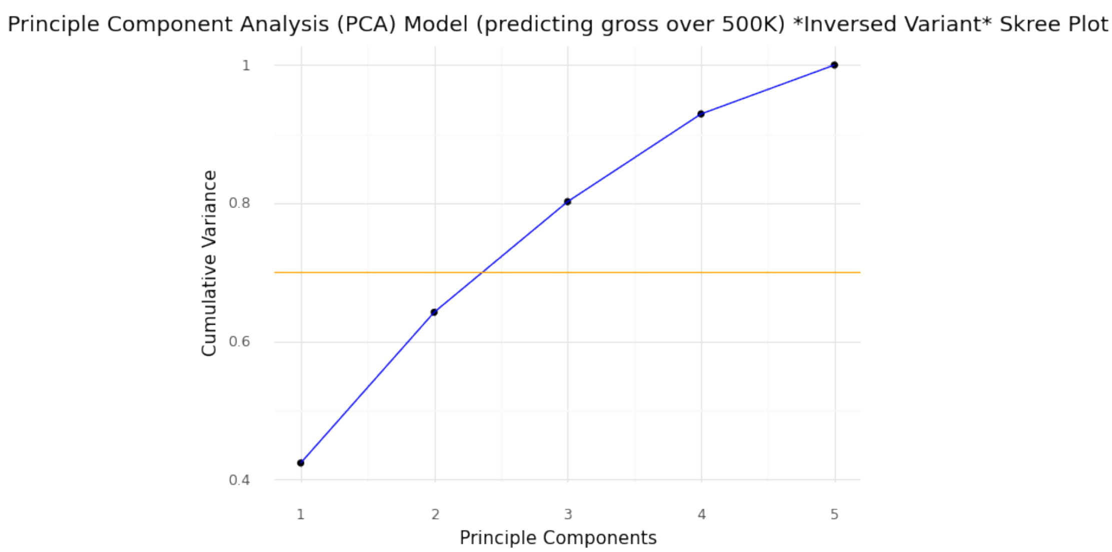
*Caption: Skree plot showing how much variance in the data is accounted for by each principle component in the model, as determined using PCA. The model is predicting whether a movie will gross over or under 500,000 dollars.*

### 10.12 Interpretation of the Skree Plot for the PCA Model (predicting over 500K)

- Looking at the first principle component, it can observed that about 42% of the variance is being explained by just the first principle component. The second principle component has an explained variance of about 22%, which indicates that the second component can explain about 22% of the model's variance. If the first 2 principle components' explained variances are combined, a cumulative variance of about 64% is achieved.

#### 10.12.1 *Inversed Variant* PCA Skree Plots for Model predicting gross over 500K:

[37]:
```
# Figure out how many PCs you need to keep to retain 70% of the original variance.
(ggplot(PCA_DF_500k, aes(x = "Principle_Components", y = "Cumulative_Variance"))
 + geom_point()
 + geom_line(color = "blue")
 + geom_hline(yintercept = 0.70, color = "orange")
 + theme_minimal() + ggtitle("Principle Component Analysis (PCA) Model (predicting gross over 500K) *Inversed Variant* Skree Plot")
 + labs(x = "Principle Components", y = "Cumulative Variance")
)
```



[37]: <ggplot: (323077807)>

*Caption: Skree plot showing how much variance in the data is cumulatively accounted for by each principle component, as determined using PCA. The model is predicting whether a movie will gross over or under 500,000 dollars. The horizontal line at y = 0.7 represents 70% cumulative variance and can be used to determine how many principle components are needed to explain 70% of the variance in the data.*

### 10.13 Discussion of Inversed Variant Skree Plot for the PCA Model (predicting gross over 500K)

- This graph depicts the cumulative explained variance with the addition of each principle components for the model predicting a movie to gross over 500K. The orange horizontal line is the minimum amount of explained variance, 70%, that is desired for this problem. **It can be observed that at least 3 principle components are needed to create a Logistic Regression model predicting a gross over 500K with at least 70% explained variance.**

[38]:
```
# method used to calculate the min number of principle components to achieve the␣
↪threshold cumulative accuracy
def calc_min_pc(data_frame, col_name, threshold):
    pc_index = 0
    for pc in data_frame[col_name]:
        pc_index += 1
        if pc >= threshold:
            return pc_index
```

[39]:
```
# figuring out how many PCs need to keep to retain 70% of the original variance
min_pc_250k = calc_min_pc(PCA_DF_250k, 'Cumulative_Variance', 0.70)
min_pc_500k = calc_min_pc(PCA_DF_500k, 'Cumulative_Variance', 0.70)

print("\n")
print("According to PCA, the Logistic Regression Model only needs " +␣
↪str(min_pc_250k) + " Principle Components to predict a movie will gross over␣
↪250K with at least 70% accuracy")
print("\n")
print("According to PCA, the Logistic Regression Model only needs " +␣
↪str(min_pc_500k) + " Principle Components to predict a movie will gross over␣
↪500K with at least 70% accuracy")
print("\n")
```

```
According to PCA, the Logistic Regression Model only needs 3 Principle
Components to predict a movie will gross over 250K with at least 70% accuracy


According to PCA, the Logistic Regression Model only needs 3 Principle
Components to predict a movie will gross over 500K with at least 70% accuracy
```

# 11 Question 6 Explicit Answer to the Question - What is the minimum number of features needed to predict whether a movie will gross over 250K and over 500K with at least 70% explained variance?

- **The minimum number of features needed to predict whether a movie will gross over 250K and over 500K with at least 70% explained variance is both 3 principle components.**

## 11.1 Creating new LR Models knowing now that only need 3 principle components

### 11.1.1 We are creating these models to ensure that these models are in fact predicting with at least 70% explained variance

[40]:
```
mod_PCA_Model_250k = PCA(n_components = min_pc_250k)
mod_PCA_Model_250k.fit(PCA_LR_X_train_250k)
```

[40]: PCA(n_components=3)

[41]:
```
mod_PCA_Model_500k = PCA(n_components = min_pc_500k)
mod_PCA_Model_500k.fit(PCA_LR_X_train_500k)
```

[41]: PCA(n_components=3)

[42]:
```
# 250k model
mod_train_y_pred_250k = PCA_LR_Model_250k.predict(PCA_LR_X_train_250k)

train_mod_mse_250k = mean_squared_error(PCA_LR_y_train_250k,␣
↪mod_train_y_pred_250k)
test_mod_mse_250k = mean_squared_error(PCA_LR_y_test_250k, PCA_LR_y_pred_250k)

train_mod_r2_250k = r2_score(PCA_LR_y_train_250k, mod_train_y_pred_250k)
test_mod_r2_250k = r2_score(PCA_LR_y_test_250k, PCA_LR_y_pred_250k)

# 500k model
mod_train_y_pred_500k = PCA_LR_Model_500k.predict(PCA_LR_X_train_500k)

train_mod_mse_500k = mean_squared_error(PCA_LR_y_train_500k,␣
↪mod_train_y_pred_500k)
test_mod_mse_500k = mean_squared_error(PCA_LR_y_test_500k, PCA_LR_y_pred_500k)

train_mod_r2_500k = r2_score(PCA_LR_y_train_500k, mod_train_y_pred_500k)
test_mod_r2_500k = r2_score(PCA_LR_y_test_500k, PCA_LR_y_pred_500k)
```

[43]:
```
print("PCA Model (250k model) MSE (Train): " + str(round(train_mod_mse_250k, 3)))
print("PCA Model (250k model) MSE (Test): " + str(round(test_mod_mse_250k, 3)) +␣
↪"\n")

print("PCA Model (250k model) r2 (Train): " + str(round(abs(train_mod_r2_250k),␣
↪3)))
print("PCA Model (250k model) r2 (Test): " + str(round(abs(test_mod_r2_250k),␣
↪3)))
```

```
PCA Model (250k model) MSE (Train): 0.111
PCA Model (250k model) MSE (Test): 0.135

PCA Model (250k model) r2 (Train): 0.109
PCA Model (250k model) r2 (Test): 0.189
```

### 11.2 Discussion of the Logistic Regression Model built with the min number of principle components to predict gross over 250K with at least 70% explained variance

- This model gave an mse of about 0.129 and an r2 of about 0.126. The original version of this model has the same exact mse and r2 values. **The results are identical which is great because that means the new logistic regression models are able to achieve identitcal results while using only 3 principle components instead of all 5 principle components.**

#### 11.2.1 NOTE: The train mse and r2 values were calculated to ensure that the train and test values are similar. If these values are not similar, it can indicate the model is overfitting. Since the train and test values are very similar here, overfitting is not a concern.

[44]:
```
print("PCA Model (500k model) MSE (Train): " + str(round(train_mod_mse_500k, 3)))
print("PCA Model (500k model) MSE (Test): " + str(round(test_mod_mse_500k, 3)) +␣
↪"\n")

print("PCA Model (500k model) r2 (Train): " + str(round(abs(train_mod_r2_500k),␣
↪3)))
print("PCA Model (500k model) r2 (Test): " + str(round(abs(test_mod_r2_500k),␣
↪3)))
```

```
PCA Model (500k model) MSE (Train): 0.153
PCA Model (500k model) MSE (Test): 0.145

PCA Model (500k model) r2 (Train): 0.131
PCA Model (500k model) r2 (Test): 0.081
```

### 11.3 Discussion of the Logistic Regression Model built with the min number of principle components to predict gross over 500K with at least 70% explained variance

- This model gave an mse of about 0.151 and an r2 of about 0.169. The original version of this model has the same exact mse and r2 values. **The results are identical which is great be-**

**cause that means the new logistic regression models are able to achieve identitcal results while using only 3 principle components instead of all 5 principle components.**

#### 11.3.1 NOTE: The train mse and r2 values were calculated for the same reasons mentioned before.

# Final Project Part 2

May 18, 2021

[1]:
```
# import necessary packages

import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
from plotnine import *

from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn import metrics
from sklearn.preprocessing import StandardScaler #Z-score variables
from sklearn.model_selection import KFold # k-fold cv
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import plot_confusion_matrix

from sklearn.model_selection import GridSearchCV
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.cluster import DBSCAN

from sklearn.metrics import silhouette_score
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch
from matplotlib import pyplot as plt

%precision %.7g
%matplotlib inline
```

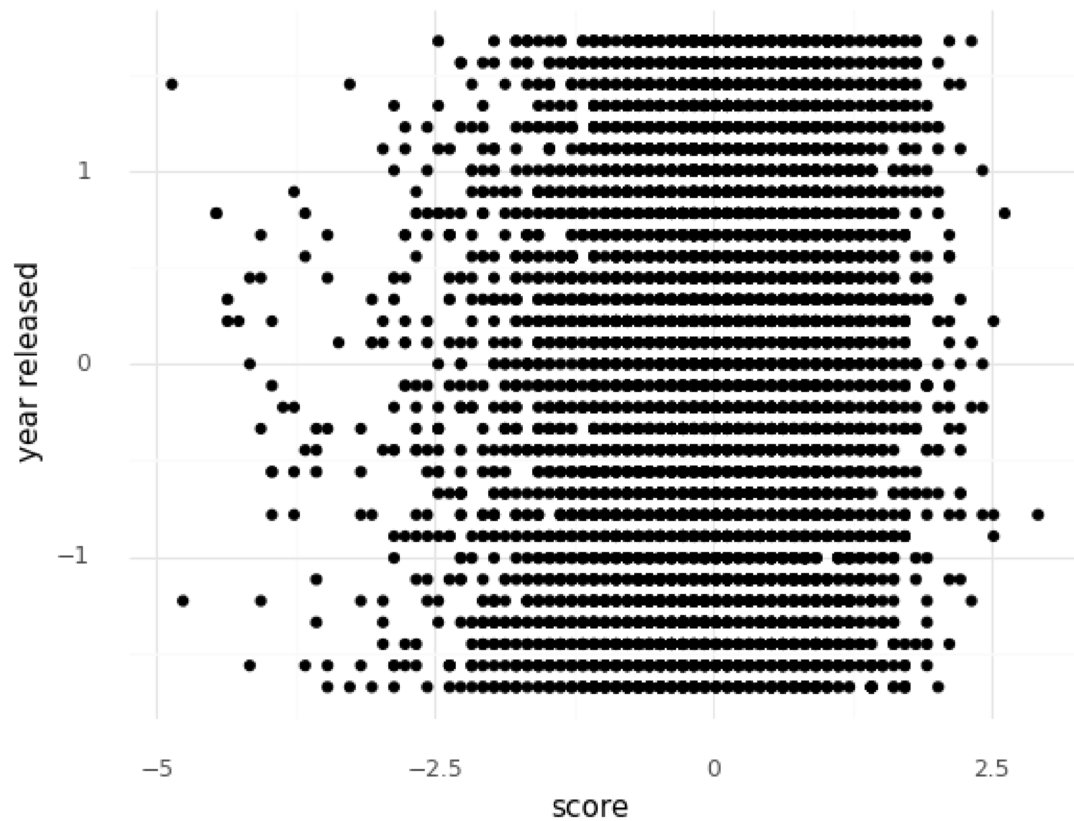
## 0.1 7. Using years and IMDB user score, which clustering models (K means, Gaussian Mixture Models, Hierarchical Clustering, or DBSCAN) would create the best clusters for our dataset?

## 0.2 a)

[2]:
```
data = pd.read_csv("https://raw.githubusercontent.com/tarekel96/MoviesData/
↪master/movies.csv")
data.head()
```

[2]:
```
      budget                                 company country         director  \
0    8000000            Columbia Pictures Corporation     USA       Rob Reiner
1    6000000                       Paramount Pictures     USA      John Hughes
2   15000000                       Paramount Pictures     USA       Tony Scott
3   18500000  Twentieth Century Fox Film Corporation     USA    James Cameron
4    9000000                    Walt Disney Pictures     USA  Randal Kleiser

       genre  genre_encoded      gross                     name rating  \
0  Adventure              0   52287414              Stand by Me      R
1     Comedy              1   70136369  Ferris Bueller's Day Off  PG-13
2     Action              2  179800601                  Top Gun     PG
3     Action              2   85160248                   Aliens      R
4  Adventure              0   18564613  Flight of the Navigator     PG

   rating_encoded    released  runtime  score                 star   votes  \
0               3  1986-08-22       89    8.1          Wil Wheaton  299174
1               2  1986-06-11      103    7.8   Matthew Broderick  264740
2               1  1986-05-16      110    6.9           Tom Cruise  236909
3               3  1986-07-18      137    8.4    Sigourney Weaver  540152
4               1  1986-08-01       90    6.9          Joey Cramer   36636

          writer  year released
0   Stephen King           1986
1    John Hughes           1986
2       Jim Cash           1986
3  James Cameron           1986
4  Mark H. Baker           1986
```

[3]:
```
features = ["year released", "score"]
X = data[features]

z = StandardScaler()
X[["year released", "score"]] = z.fit_transform(X)

(ggplot(X, aes("score", "year released")) + geom_point() + theme_minimal())
```




[3]: <ggplot: (8776549727476)>

[4]:
```
# GMM

EM = GaussianMixture(n_components = 2)
EM.fit(X[features])
```

[4]: GaussianMixture(n_components=2)

[5]:
```
cluster = EM.predict(X[features])
cluster
```

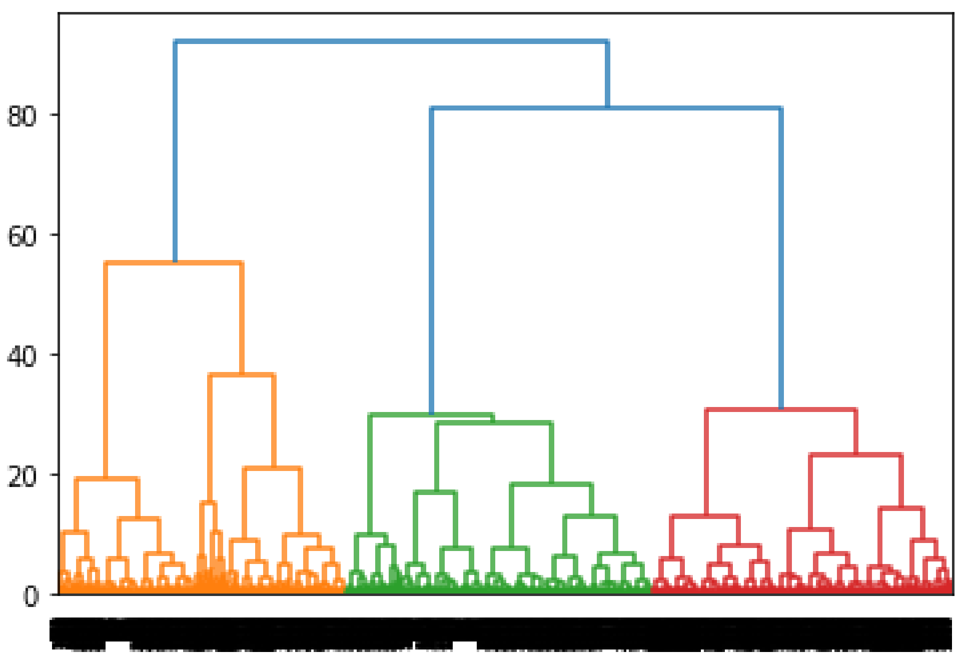
[5]: array([1, 1, 1, ..., 0, 0, 0])

[6]:
```
GMMnoruntimescore = silhouette_score(X[features], cluster)
print("SILHOUETTE: ", GMMnoruntimescore)
```

SILHOUETTE:  0.37352602092542114

[15]:
```
# HCA - 3 clusters (we'll find the optimal number of clusters later on),␣
 ↪affinity = euclidean, & linkage = ward
HCA_Model = AgglomerativeClustering(n_clusters = 3, affinity = "euclidean",
                             linkage = "ward")
# linkage - distance between the different clusters (method of calculating these␣
 ↪values ie average, furthest, nearest, etc)
#  affinity - intracluster distance calculating tuning

HCA_Model.fit(X[features])
# dendro - for showing the branches
dendro = sch.dendrogram(sch.linkage(X[features], method='ward'))
# very computationally expensive

membership = HCA_Model.labels_
HCA_ward_silhouette_score = silhouette_score(X[features], membership)
print("HCA Silhoutte Score (3 clusters, affinity - euclidean, & linkage - ward):
 ↪\n" + str(HCA_ward_silhouette_score))
```

```
HCA Silhoutte Score (3 clusters, affinity - euclidean, & linkage - ward):
0.3276350640789347
```



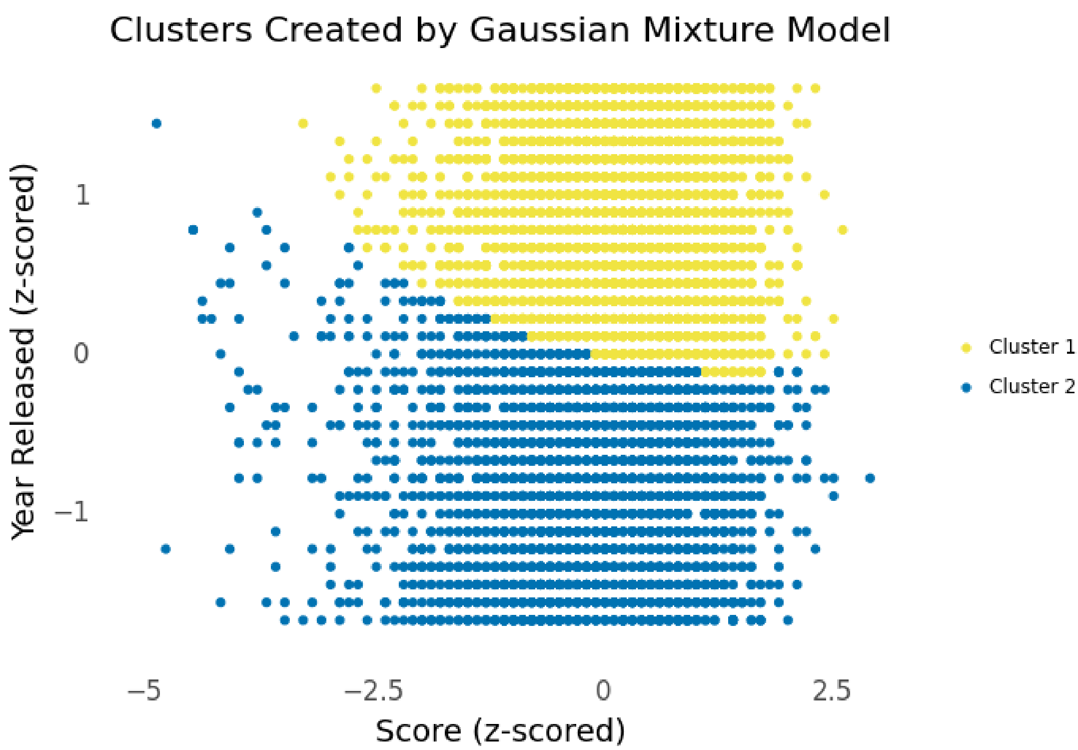

## 0.3 b)

Using years and IMDB user score, Gaussian Mixture Models (GMM) and Hierarchical Clustering (HC) would likely create the best clusters for our dataset. As shown by the first graph, in which all the data is plotted on a scatterplot, it appears that the data is very dense, so any clusters that exist within the data are likely overlapping. Given this, we eliminated both DBSCAN and K means as possible options, since both machine learning models do not perform well on data that has overlapping clusters. Instead, we used GMM and HC, which are more suited for handling overlapping clusters. In particular, GMM outperformed HC, likely because Gaussian mixture models uss probabilistic assignment, which means it assigns probabilities of belonging to each cluster to each data point. This is useful for this dataset in particular, as a Gaussian mixture model (EM) would be able to pick up the overlapping clusters using probabilistic assignment. We gauged our model's performance using their silhouette scores, which are a way for us to look at how well separated and cohesive the clusters created by the models are. GMM had a higher silhouette score of about 0.37, compared to HAC's score of about 0.33, and the higher the silhouette score, the better/more separated/more cohesive clusters the model has.

It is important to use models that create the best clusters possible in order to extrapolate accurate information from them. If we were to use DBSCAN or K means, the clusters created would likely not represent the actual clusters present in the data, which can lead to wrong assumptions being drawn from our data. Thus, by using GMM and HAC, we are more likely to draw meaningful information from the clusters that each model creates.

### 0.4 c)

[7]:
```
# GMM

X["cluster"] = cluster

(ggplot(X, aes(x = "score", y = "year released", color = "factor(cluster)")) +
     geom_point() +
     theme_minimal() +
     labs(x = "Score (z-scored)", y = "Year Released (z-scored)", color = "") +
     ggtitle("Clusters Created by Gaussian Mixture Model") +
     scale_color_manual(labels = ("Cluster 1", "Cluster 2"), values =␣
 ↪("#f0e442", "#0072b2")) +
     theme_minimal() +
     theme(panel_grid_minor_x = element_blank(),
     panel_grid_minor_y = element_blank(),
     panel_grid_major_x = element_blank(),
     panel_grid_major_y = element_blank(),
     axis_text_x = element_text(size = 12),
     axis_title_x = element_text(size = 14),
     axis_text_y = element_text(size = 12),
     axis_title_y = element_text(size = 14),
     plot_title = element_text(size = 16)))
```




[7]: <ggplot: (8776550103926)>

*Caption: Graph showing the two clusters created by GMM when plotting year released versus score. GMM divided the data horizontally in the center in order to create two equally-sized clusters.*

[8]:
```
HAC_scores = []
cluster_trials = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

for n in cluster_trials:
    # HCA - 3 clusters, affinity = euclidean, & linkage = ward
    HAC_Model = AgglomerativeClustering(n_clusters = n, affinity = "euclidean",
                                         linkage = "ward")
    # linkage - distance between the different clusters (method of calculating␣
 ↪these values ie average, furthest, nearest, etc)
    #  affinity - intracluster distance calculating tuning

    HAC_Model.fit(X[features])
    # dendro - for showing the branches
    # dendro = sch.dendrogram(sch.linkage(X[features], method='ward'))
    # very computationally expensive

    membership = HAC_Model.labels_
    HCA_ward_silhouette_score = silhouette_score(X[features], membership)
```

```
    cluster_dict = {
        "num_clusters": n,
        "score": HCA_ward_silhouette_score
    }
    HAC_scores.append(cluster_dict)
    print("HCA Silhoutte Score (" + str(n)
          + " clusters):\n"
          + str(HCA_ward_silhouette_score)
          + "\n")

HAC_scores
```

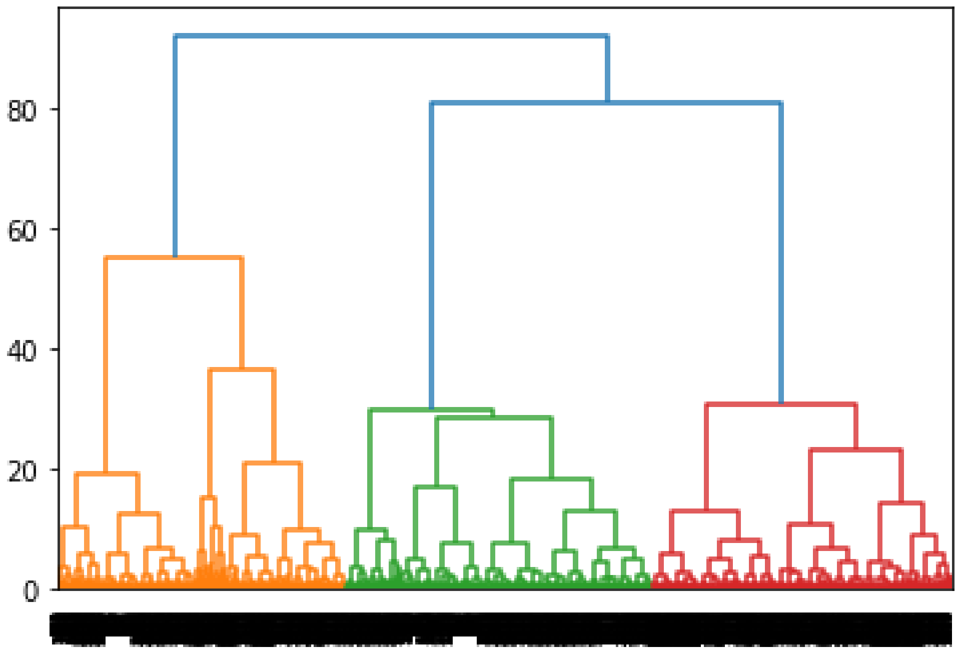
```
HCA Silhoutte Score (2 clusters):
0.30955369019164247

HCA Silhoutte Score (3 clusters):
0.3276350640789347

HCA Silhoutte Score (4 clusters):
0.3340925485827353

HCA Silhoutte Score (5 clusters):
0.32783739521912175

HCA Silhoutte Score (6 clusters):
0.2790097861404158

HCA Silhoutte Score (7 clusters):
0.25555022093770213

HCA Silhoutte Score (8 clusters):
0.2705647819387868

HCA Silhoutte Score (9 clusters):
0.2764890948152955

HCA Silhoutte Score (10 clusters):
0.27574224652822765

HCA Silhoutte Score (11 clusters):
0.2648636134951893

HCA Silhoutte Score (12 clusters):
0.26388664707234094

HCA Silhoutte Score (13 clusters):
0.26204030575302484
```

```
HCA Silhoutte Score (14 clusters):
0.2651498703191556

HCA Silhoutte Score (15 clusters):
0.25730398757135653
```

[8]:
```
[{'num_clusters': 2, 'score': 0.30955369019164247},
 {'num_clusters': 3, 'score': 0.3276350640789347},
 {'num_clusters': 4, 'score': 0.3340925485827353},
 {'num_clusters': 5, 'score': 0.32783739521912175},
 {'num_clusters': 6, 'score': 0.2790097861404158},
 {'num_clusters': 7, 'score': 0.25555022093770213},
 {'num_clusters': 8, 'score': 0.2705647819387868},
 {'num_clusters': 9, 'score': 0.2764890948152955},
 {'num_clusters': 10, 'score': 0.27574224652822765},
 {'num_clusters': 11, 'score': 0.2648636134951893},
 {'num_clusters': 12, 'score': 0.26388664707234094},
 {'num_clusters': 13, 'score': 0.26204030575302484},
 {'num_clusters': 14, 'score': 0.2651498703191556},
 {'num_clusters': 15, 'score': 0.25730398757135653}]
```

[41]:
```
# HCA - 4 clusters, affinity = euclidean, & linkage = ward
HAC_Model = AgglomerativeClustering(n_clusters = 4, affinity = "euclidean",
                                    linkage = "ward")
# linkage - distance between the different clusters (method of calculating these␣
 ↪values ie average, furthest, nearest, etc)
#  affinity - intracluster distance calculating tuning

HAC_Model.fit(X[features])
# dendro - for showing the branches
dendro = sch.dendrogram(sch.linkage(X[features], method='ward'))
# very computationally expensive

membership = HAC_Model.labels_
HCA_ward_silhouette_score = silhouette_score(X[features], membership)
print("HCA Silhoutte Score (4 clusters, affinity - euclidean, & linkage - ward):
 ↪\n" + str(HCA_ward_silhouette_score))
```

```
HCA Silhoutte Score (4 clusters, affinity - euclidean, & linkage - ward):
0.3340925485827353
```




*Caption: Dendrogram that shows the clusters created by our HAC model. Unlike GMM, which made two clusters, HAC produced three distinct clusters. It also appears that the clusters are both relatively well separated as well as cohesive, given that most of the density is towards the bottom of the graph, while the top of the graph is relatively sparse.*

## 0.5 8. Assess the performance of the best model from question 7. Additionally, how is the model performance affected when runtime is added as a variable?

### 0.6 a)

[9]:
```
new_features = ["year released", "score", "runtime"]
new_X = data[new_features]

z = StandardScaler()
new_X[new_features] = z.fit_transform(new_X)

new_X.head()
```

[9]:

| | year released | score | runtime |
|---|---|---|---|
| 0 | -1.677164 | 1.719825 | -0.973621 |
| 1 | -1.677164 | 1.420743 | -0.197002 |
| 2 | -1.677164 | 0.523496 | 0.191308 |
| 3 | -1.677164 | 2.018907 | 1.689073 |
| 4 | -1.677164 | 0.523496 | -0.918148 |

[10]:
```
EM = GaussianMixture(n_components = 3)
EM.fit(new_X)
```

[10]:
```
GaussianMixture(n_components=3)
```

[11]:
```
cluster = EM.predict(new_X)
cluster
```

[11]:
```
array([1, 1, 1, ..., 2, 2, 2])
```

[12]:
```
GMMruntimescore = silhouette_score(new_X, cluster)
print("SILHOUETTE: ", GMMruntimescore)
```

```
SILHOUETTE:  0.28610686925562406
```

### 0.7 b)

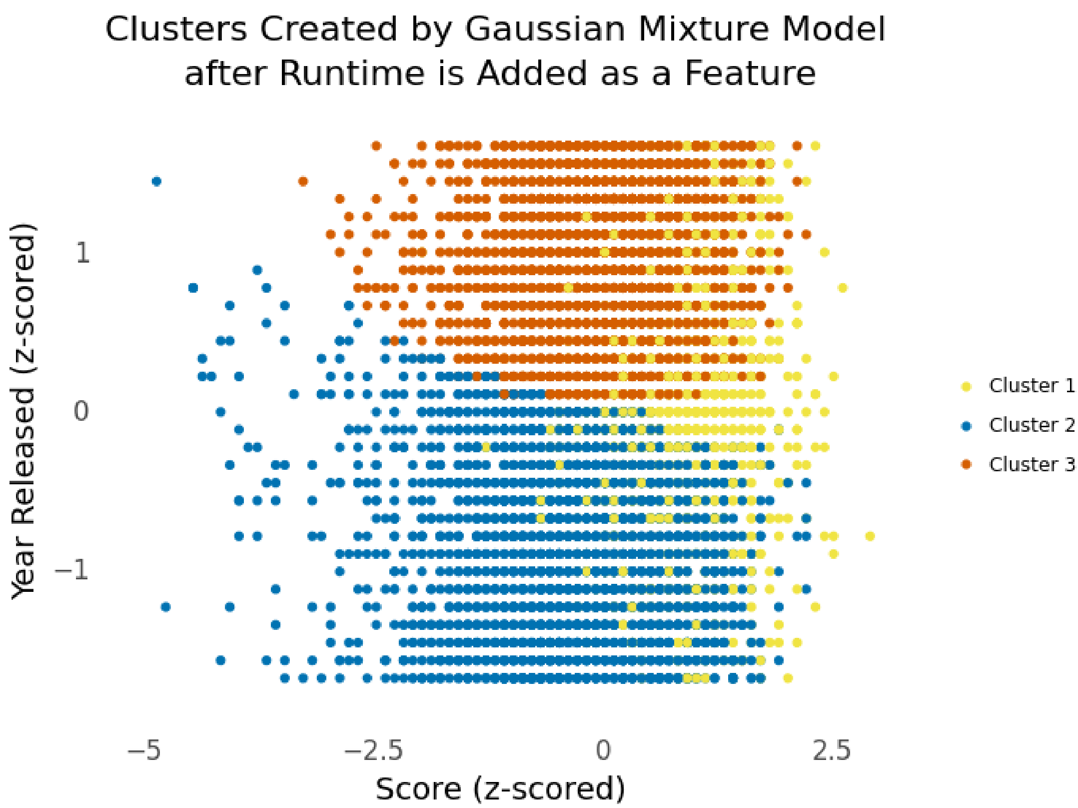
Based on the analyses run in part 7, the best model for this dataset is GMM. We can use the model's silhouette score to assess the model's performance. GMM had a silhouette score of about 0.37, which was greater than the approximately 0.33 silhouette score obtained by HAC. A silhouette score reflects how well separated (i.e., how far apart clusters are from each other) as well as how cohesive (i.e., how close together points in a single cluster are to each other). An ideal cluster is both well separated and cohesive, and a high silhouette score would reflect that good clusters have been obtained. Since GMM had a higher silhouette score than HAC, we can infer that GMM had the best model performance out of the two models.

When we added in runtime as a variable, the performance of GMM decreased from about 0.37 to about 0.29. Thus, runtime decreases our model's performance by causing the clusters that it creates to be less cohesive as well as less separated.

### 0.8 c)

[13]:
```
# First ggplot (years on the x-axis and scores on the y-axis)

new_X["cluster"] = cluster

(ggplot(new_X, aes(x = "score", y = "year released", color = "factor(cluster)"))␣
 ↪+
    geom_point() +
    theme_minimal() +
    labs(x = "Score (z-scored)", y = "Year Released (z-scored)", color = "") +
    ggtitle("Clusters Created by Gaussian Mixture Model \nafter Runtime is␣
 ↪Added as a Feature") +
    scale_color_manual(labels = ("Cluster 1", "Cluster 2", "Cluster 3"), values␣
 ↪= ("#f0e442", "#0072b2", "#d55e00")) +
    theme_minimal() +
    theme(panel_grid_minor_x = element_blank(),
    panel_grid_minor_y = element_blank(),
```

```
    panel_grid_major_x = element_blank(),
    panel_grid_major_y = element_blank(),
    axis_text_x = element_text(size = 12),
    axis_title_x = element_text(size = 14),
    axis_text_y = element_text(size = 12),
    axis_title_y = element_text(size = 14),
    plot_title = element_text(lineheight = 1.5, size = 16)))
```




[13]: <ggplot: (8776506420387)>

*Caption: Graph showing the clusters created by GMM after 'runtime' was added as a variable when plotting year released versus score. GMM divided the data into three clusters, whereas it divided the data into just two clusters when runtime was not included.*

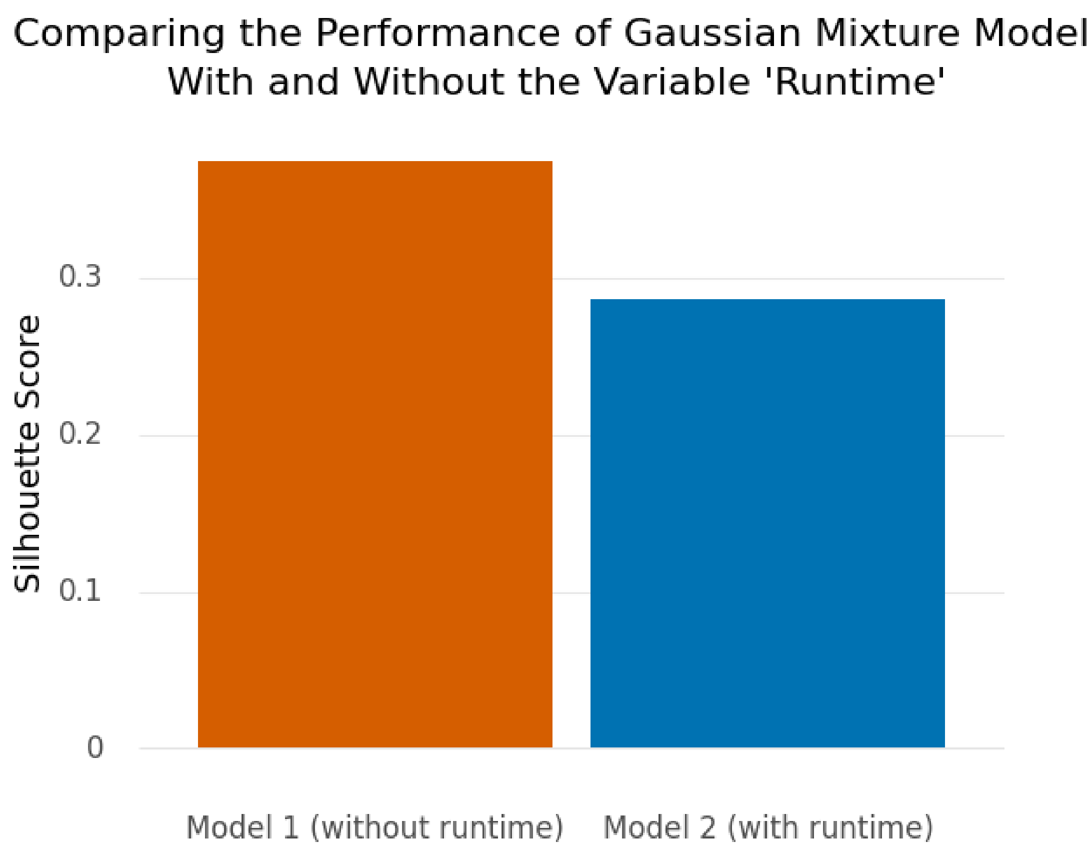
[14]:
```
# Second ggplot
# make a new DF with just the probabilities above

group = ["Model 2 (with Runtime)","Model 1"] # column 1 of the df
score = [GMMruntimescore, GMMnoruntimescore] # column 2 of the df
myDF = {"Model": group, "Score": score} # create a dictionary for df
probDF = pd.DataFrame(myDF) # create the df using pandas
probDF.head()
```

[14]:

```
                     Model     Score
0  Model 2 (with Runtime)  0.286107
1                 Model 1  0.373526
```

[16]:

```
(ggplot(probDF, aes(x = "Model", y = "Score", fill = "group")) +
     geom_bar(stat="identity") +
     theme_minimal() +
     labs(x = "", y = "Silhouette Score") +
     scale_x_discrete(labels = ("Model 1 (without runtime)", "Model 2 (with␣
↪runtime)")) +
     ggtitle("Comparing the Performance of Gaussian Mixture Model \nWith and␣
↪Without the Variable 'Runtime'") +
     scale_fill_manual(["#d55e00", "#0072b2"]) +
     theme(panel_grid_minor_x = element_blank(),
     panel_grid_minor_y = element_blank(),
     panel_grid_major_x = element_blank(),
     axis_text_x = element_text(size = 12),
     axis_title_x = element_text(size = 14),
     axis_text_y = element_text(size = 12),
     axis_title_y = element_text(size = 14),
     plot_title = element_text(lineheight = 1.5, size = 16),
     legend_text = element_text(size = 12),
     legend_position = "none"))
```




[16]: <ggplot: (8776506486673)>

*Caption: Comparing the performance of GMM after 'runtime' was added as a variable. Model 1, or GMM without 'runtime' as a variable, performed slightly better than Model 2, or GMM with 'runtime' as a variable.*

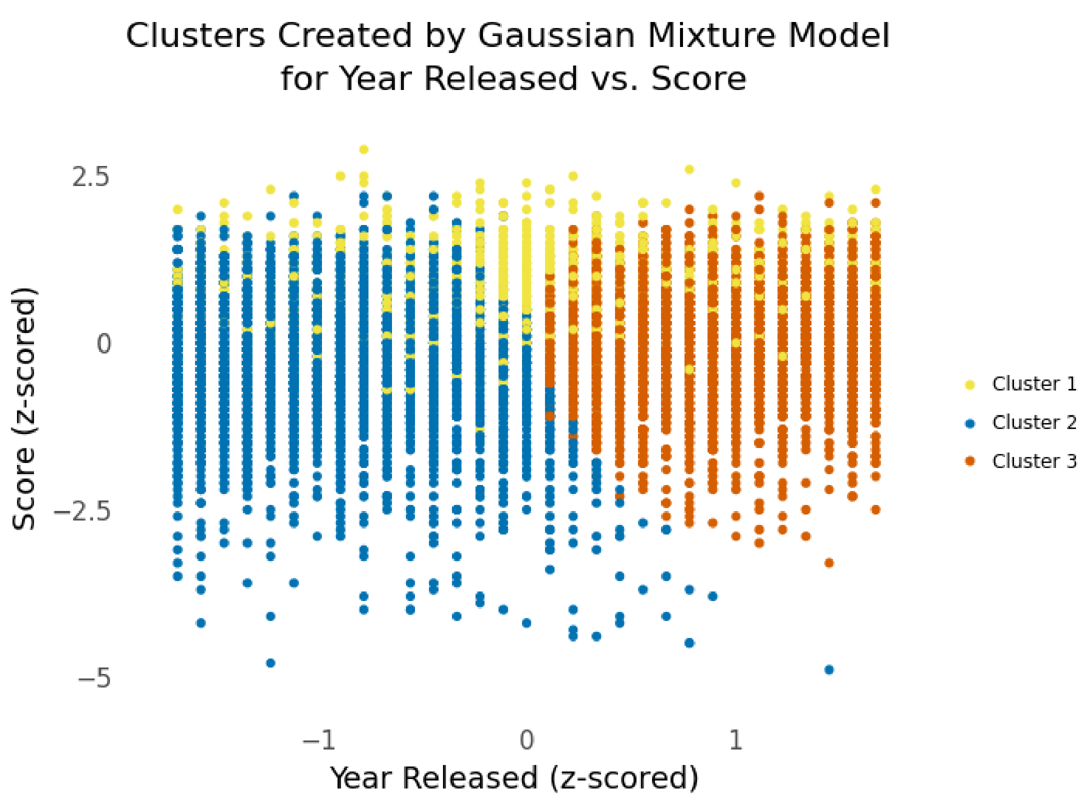
### 0.9 9) Using the model from question 8, create three scatter plots (years vs score, years vs runtime, score vs runtime) colored by cluster assignments. Describe what each cluster in each scatter plot represents.

### 0.10 a) AND c)

There is no analysis code for this section; instead, the code for this section is for generating the three ggplots.

[19]:
```
# First ggplot

(ggplot(new_X, aes(x = "year released", y = "score", color = "factor(cluster)"))␣
↪+
    geom_point() +
    theme_minimal() +
```

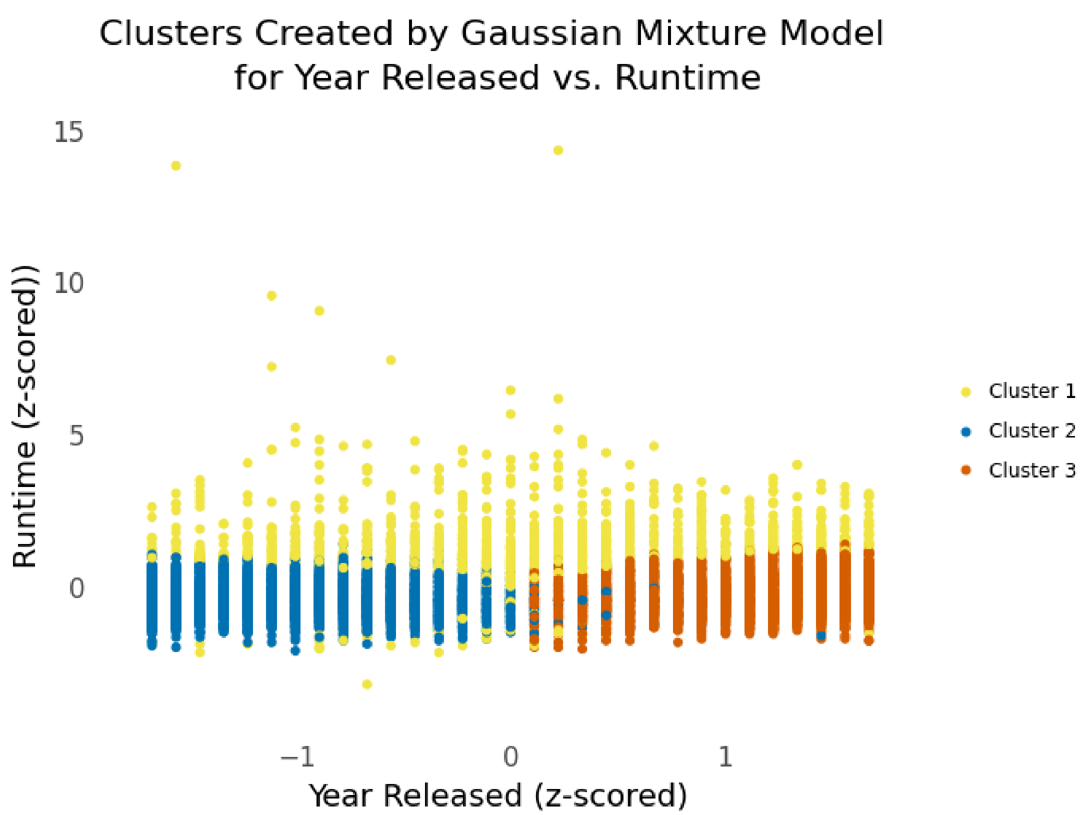
```
    labs(y = "Score (z-scored)", x = "Year Released (z-scored)", color = "") +
    ggtitle("Clusters Created by Gaussian Mixture Model \nfor Year Released vs.␣
↪Score") +
    scale_color_manual(labels = ("Cluster 1", "Cluster 2", "Cluster 3"), values␣
↪= ("#f0e442", "#0072b2", "#d55e00")) +
    theme_minimal() +
    theme(panel_grid_minor_x = element_blank(),
    panel_grid_minor_y = element_blank(),
    panel_grid_major_x = element_blank(),
    panel_grid_major_y = element_blank(),
    axis_text_x = element_text(size = 12),
    axis_title_x = element_text(size = 14),
    axis_text_y = element_text(size = 12),
    axis_title_y = element_text(size = 14),
    plot_title = element_text(lineheight = 1.5, size = 16)))
```




[19]: <ggplot: (8776405883266)>

*Caption: Scatterplot showing the clusters created by GMM when plotting year released versus score. GMM made three clusters, which appear to overlap to some extent.*

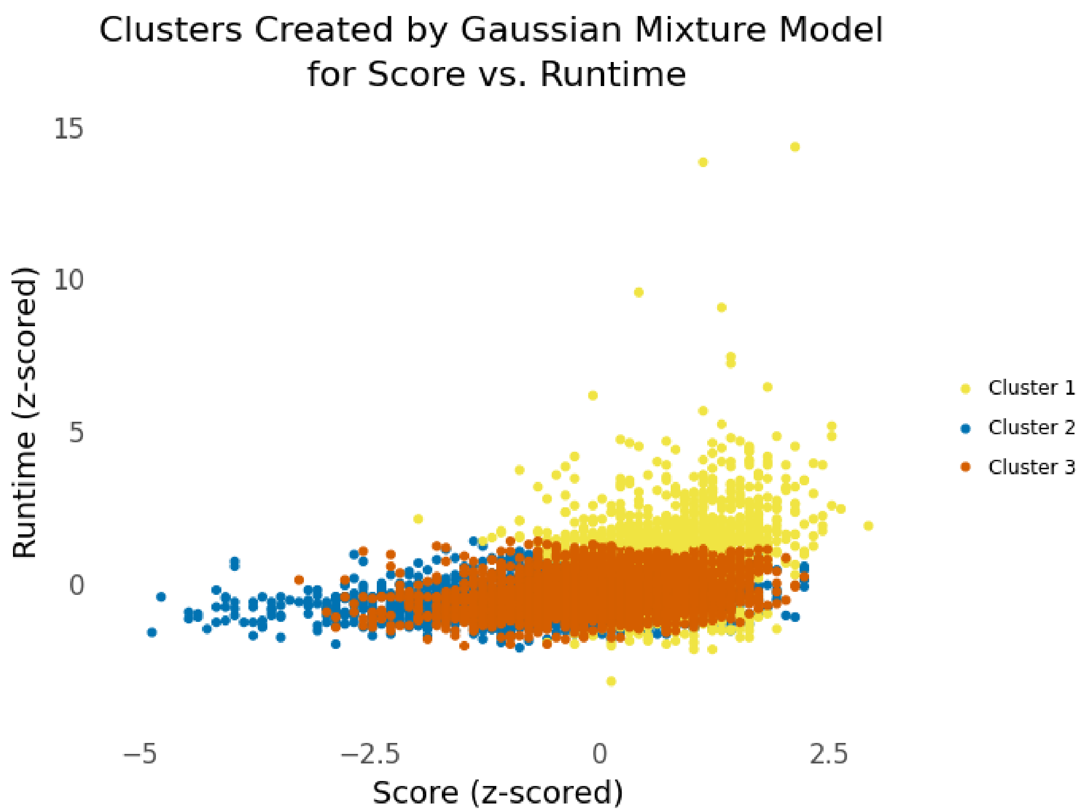
[20]:
```
# Second ggplot

(ggplot(new_X, aes(x = "year released", y = "runtime", color =␣
↪"factor(cluster)")) +
    geom_point() +
    theme_minimal() +
    labs(x = "Year Released (z-scored)", y = "Runtime (z-scored)", color = "") +
    ggtitle("Clusters Created by Gaussian Mixture Model \nfor Year Released vs.␣
↪Runtime") +
    scale_color_manual(labels = ("Cluster 1", "Cluster 2", "Cluster 3"), values␣
↪= ("#f0e442", "#0072b2", "#d55e00")) +
    theme_minimal() +
    theme(panel_grid_minor_x = element_blank(),
    panel_grid_minor_y = element_blank(),
    panel_grid_major_x = element_blank(),
    panel_grid_major_y = element_blank(),
    axis_text_x = element_text(size = 12),
    axis_title_x = element_text(size = 14),
    axis_text_y = element_text(size = 12),
    axis_title_y = element_text(size = 14),
    plot_title = element_text(lineheight = 1.5, size = 16)))
```



[20]: <ggplot: (8776506412029)>

*Caption: Scatterplot showing the clusters created by GMM when plotting year released versus runtime. GMM made three clusters, which appear to overlap to some extent.*

[21]:
```
# Third ggplot

(ggplot(new_X, aes(x = "score", y = "runtime", color = "factor(cluster)")) +
     geom_point() +
     theme_minimal() +
     labs(x = "Score (z-scored)", y = "Runtime (z-scored)", color = "") +
     ggtitle("Clusters Created by Gaussian Mixture Model \nfor Score vs.␣
 ↪Runtime") +
     scale_color_manual(labels = ("Cluster 1", "Cluster 2", "Cluster 3"), values␣
 ↪= ("#f0e442", "#0072b2", "#d55e00")) +
     theme_minimal() +
     theme(panel_grid_minor_x = element_blank(),
     panel_grid_minor_y = element_blank(),
     panel_grid_major_x = element_blank(),
     panel_grid_major_y = element_blank(),
     axis_text_x = element_text(size = 12),
     axis_title_x = element_text(size = 14),
     axis_text_y = element_text(size = 12),
     axis_title_y = element_text(size = 14),
     plot_title = element_text(lineheight = 1.5, size = 16)))
```




[21]: <ggplot: (8776406806363)>

*Caption: Scatterplot showing the clusters created by GMM when plotting year released versus runtime. GMM made three clusters, which appear to overlap to a noticeable extent.*

### 0.11 9b)

We used the model from question 8 (i.e., GMM with year released, score, and runtime as features) to generate three scatterplots.

In the first scatterplot, GMM created three clusters – a left blue cluster, a right red cluster, and a top yellow cluster. The left blue cluster represents movies that received a wide range of scores and came out relatively long ago, so this cluster can be labeled as "old movies". The right red cluster represents movies that also received a wide range of scores but came out relatively recently, so this cluster can be labeled as "new movies". The top yellow cluster represents movies that have a wide release date range but earned relatively good scores, so this cluster can be labeled as "good, timeless classics".

In the second scatterplot, GMM created three clusters – a left blue cluster, a right red cluster, and a top yellow cluster. The left blue cluster represents movies with a short runtime that came out relatively long ago, so this cluster can be labeled as "old, short movies". The right red cluster represents movies that also have a short runtime but came out relatively recently, so this cluster can be labeled as “new, short movies”. The top yellow cluster represents movies that have a wide release date range but are relatively long, so these can be labeled as “long movies”.

In the third scatterplot, GMM created three clusters – a bottom blue cluster, a bottom-right red cluster, and a middle-right yellow cluster. The bottom blue cluster represents movies that received a wide range of scores and have a relatively short runtime, so this cluster can be labeled as “short movies”. The bottom-right red cluster represents movies that also received relatively good scores and have a relatively short runtime, so this cluster can be labeled as “good short movies”. The middle-right yellow cluster represents movies that relatively received the best scores and also have a wide range of runtimes, so these can be labeled as “just good movies”.