Box Office Projections Using Box Office Mojo

Box Office Mojo is a website owned and operated by IMBD.com. It hosts box office information, including domestic and foreign earnings, as well as basic information about the film, such as rating, runtime, or budget. We can utilize information from this site to create a predictive model that will estimate the earnings of a movie based on a few basic inputs about the movie. To do so, we will need to gather enough data, understand the trends present in the data, create a predictive model, and then make that information available to others through the use of a Shiny App.

Collecting the Data

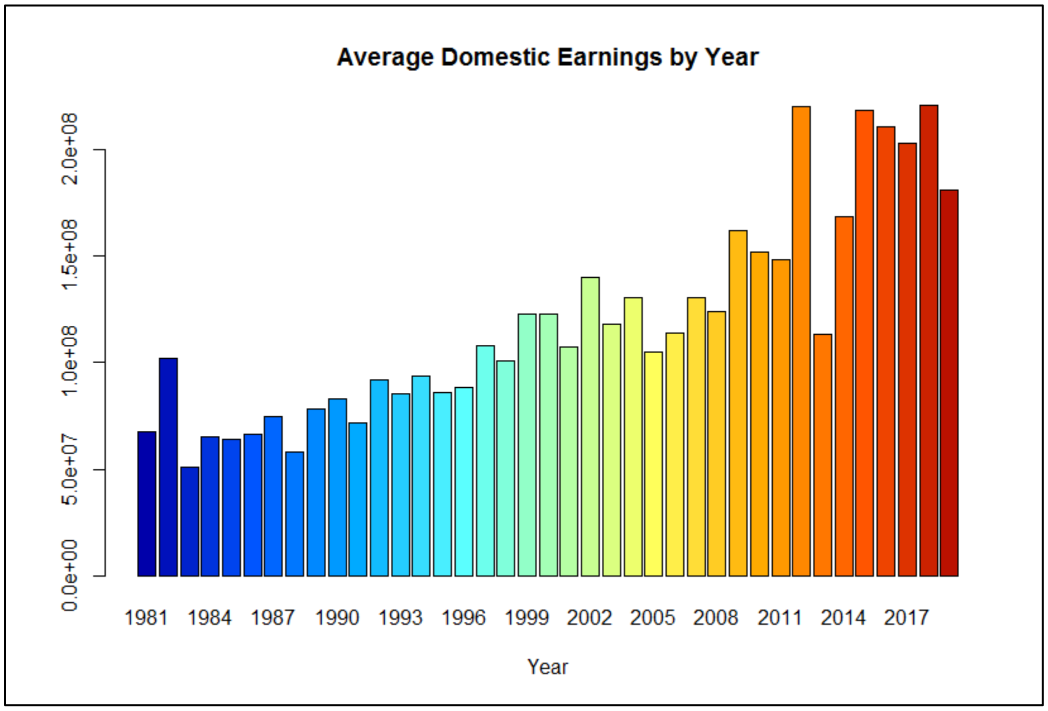
Box Office Mojo (BOM) doesn't make their data readily accessible. In order to collect enough information, we have to methodically scrape the data from their site movie-by-movie. Further complicating matters, BOM uses a movie id to identify each movie, which doesn't always correlate with the name of the film. So we need to collect the proper movie ids and then use those to collect the movie data.

To do this, I wrote two functions in R. The first function grabs the data for a specific movie id. It collects the domestic earnings, the foreign earnings, the worldwide earnings, the widest theater opening, the MPAA rating, the budget, and the month and year of release. The second function looks at the top earning weekly films from any given year and collects all of the movie ids from that year. I decided to only look at the top movies in order to weed out independent films and other such outliers.

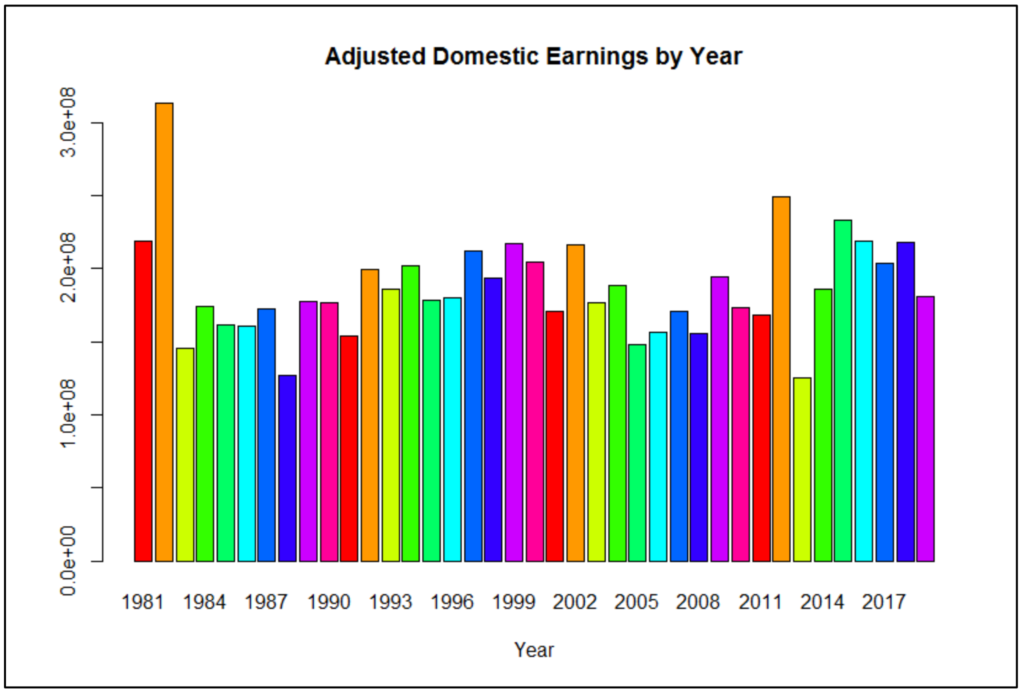
I used to second function to collect the ids of the top weekly films over the last 30 years. I limited this to 30 years because the data gets spotty as we go further back in time. Once I had collected the movie ids, I then ran those in a loop through the first function to collect all of the movie data for each id. In order to prevent my IP address getting banned, I artificially put a delay in the code between each subsequent data pull. This means that it takes a long time to collect all of the data with my code.

Exploratory Data Analysis and Data Processing

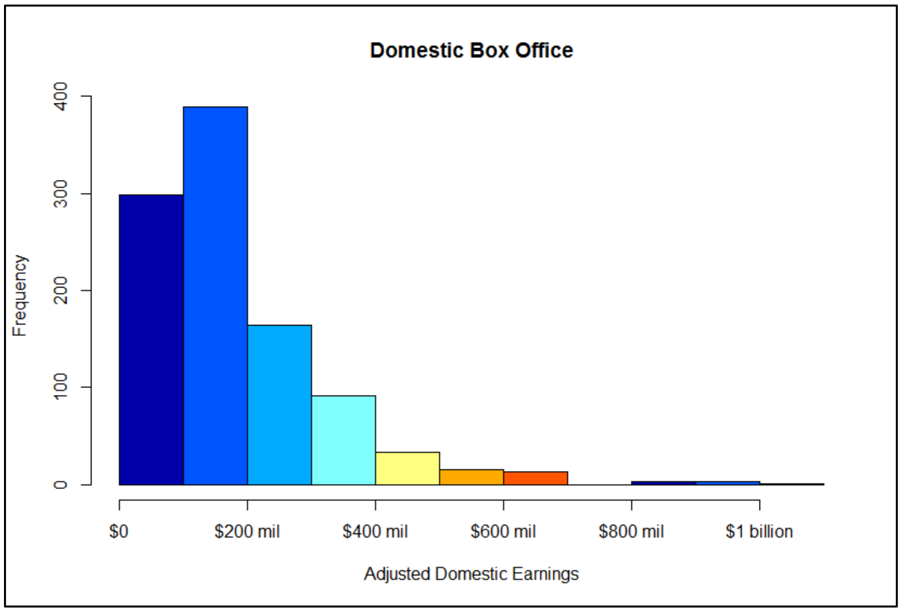
With the data collected, I needed to make sure that it was viable for use in a predictive model. I started with the response variable, the domestic earnings of the movie. By plotting the average domestic earnings by year, we can see the very apparent effect of inflation:



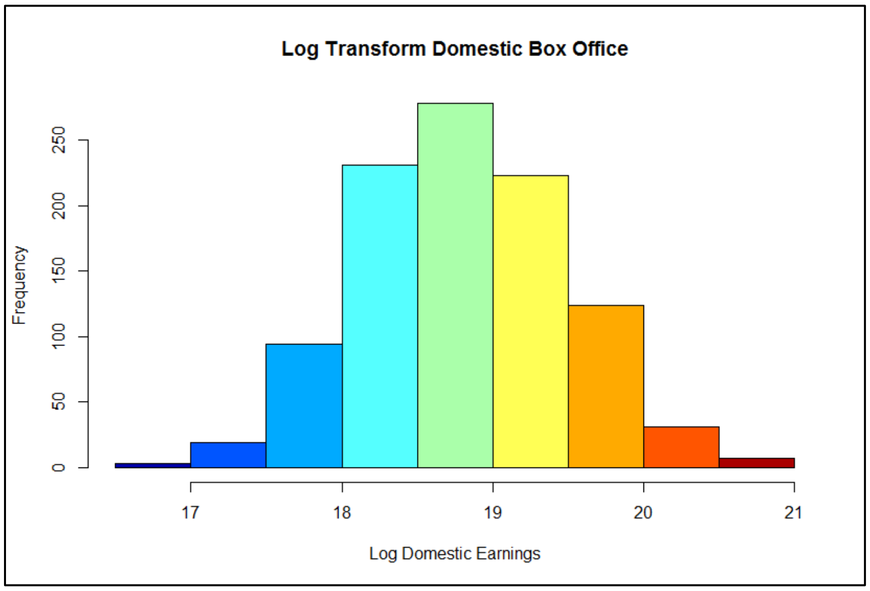
There is a very clear trend that the average earnings increase as time goes on. To make the model viable, we need to adjust the earnings for inflation. Luckily, BOM provides the average ticket price per year. We can divide a movie's earnings by the ticket price for that year to get an estimate of the number of tickets sold. We can then multiply that by the average price of a ticket in 2019 to get a decent estimate of how the earnings should be adjusted for inflation. Here is what the same plot looks like after making this adjustment:



Now that we've adjusted for inflation, we can look at a histogram of the adjusted earnings:

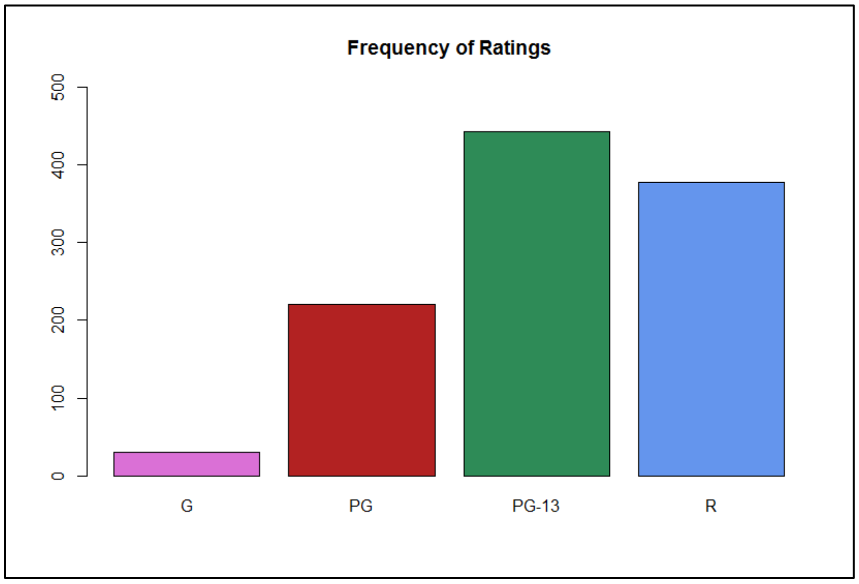


This graph shows how heavily right-skewed the data is. We can adjust for this by using a log transformation:



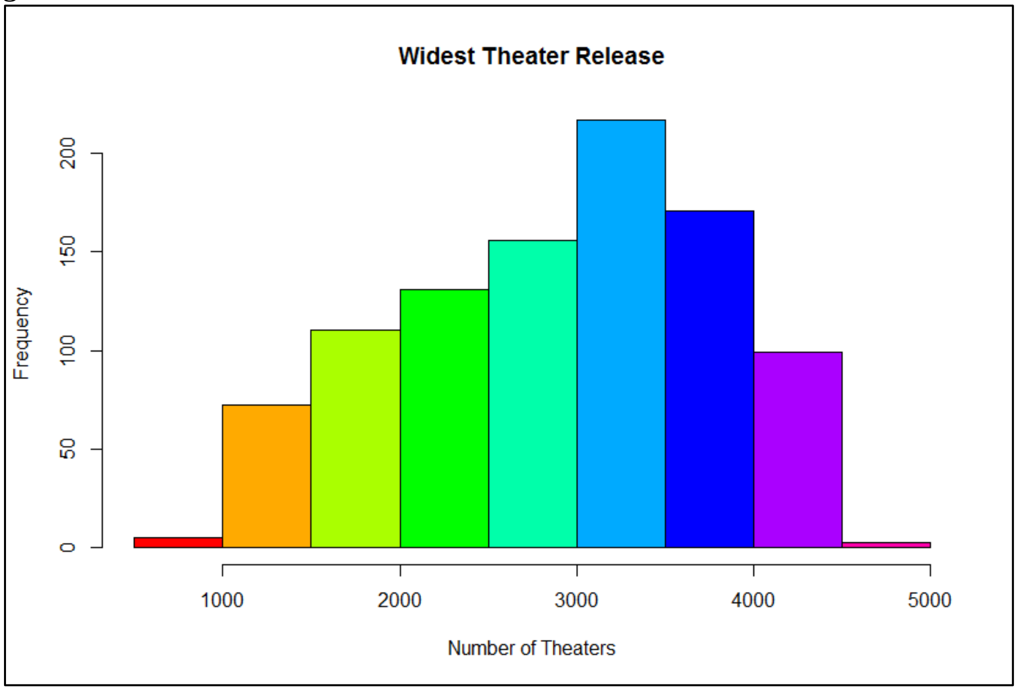
This transformed data now appears to have a normal distribution and is much more suitable for modeling.

Next is a frequency plot to see how many films of each MPAA rating we have in the data:

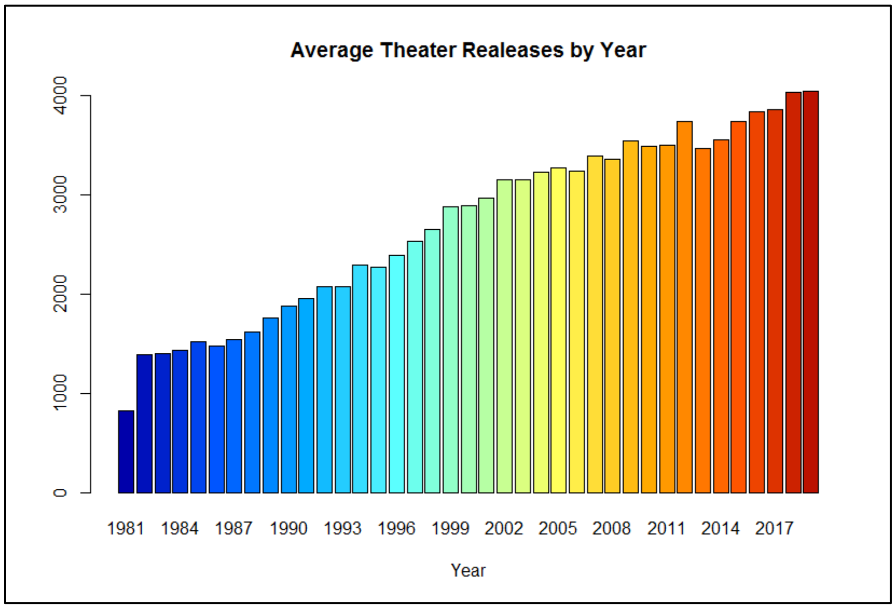


The majority of the top earning films are PG-13 and R rated, which seems reasonable. There are very few G rated movies in the data, which will likely cause some issues – especially since these are mostly Disney films which tend to perform exceptionally well, almost to the point of being outliers. This will likely bias the model toward predicting G rated films to have higher earnings.

The distribution of the widest theater release appears to have a normal distribution, which is good:

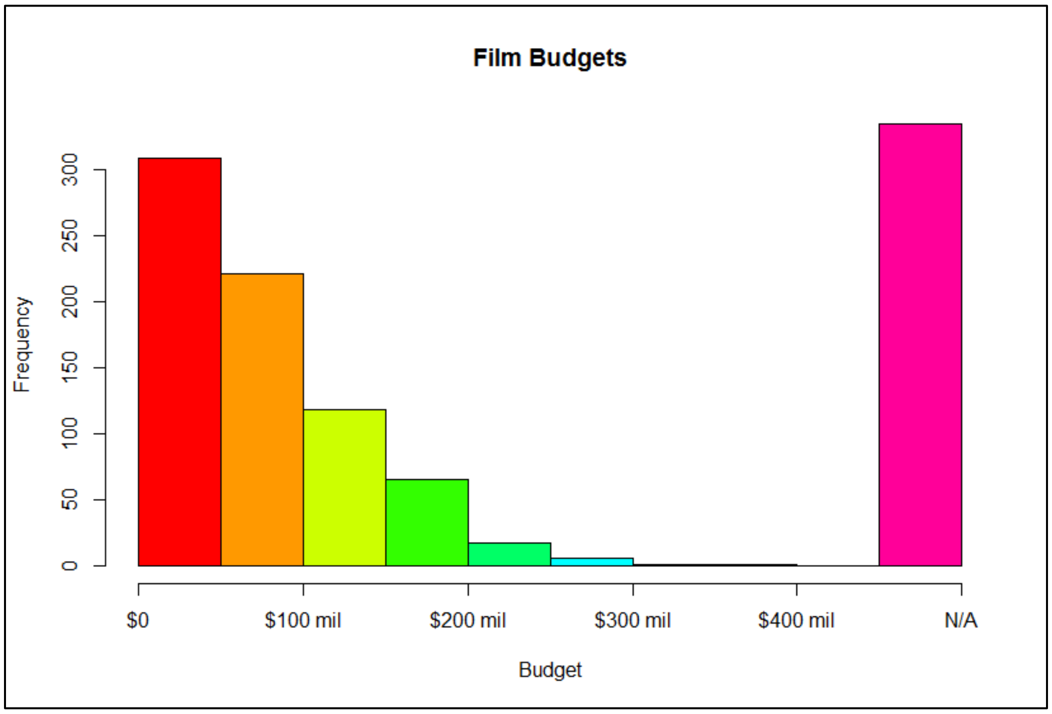


Unfortunately, the widest theater release poses a different problem that is not very easy to adjust for. Much like inflation, the number of theaters in the US has been increasing over time, as seen in this plot:

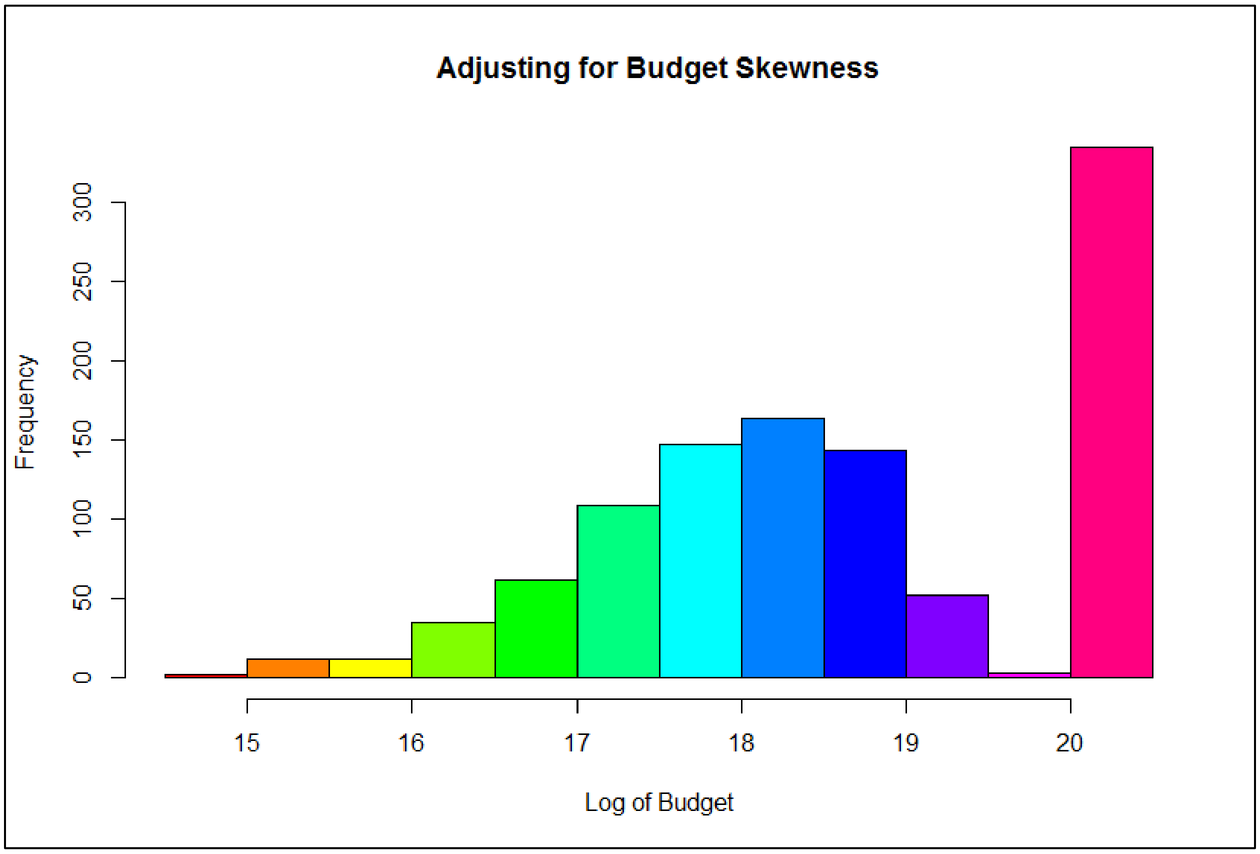


This will likely bias the model, but we are going to leave it for lack of a simple fix.

Looking at the film budgets, there are a fair number of movies that don't have a budget listed on BOM:



These are often older films or some recent Disney films that choose not to disclose their budget. Missing values aside, this data is completely skewed right. We will definitely want to use a log transformation to adjust for this:

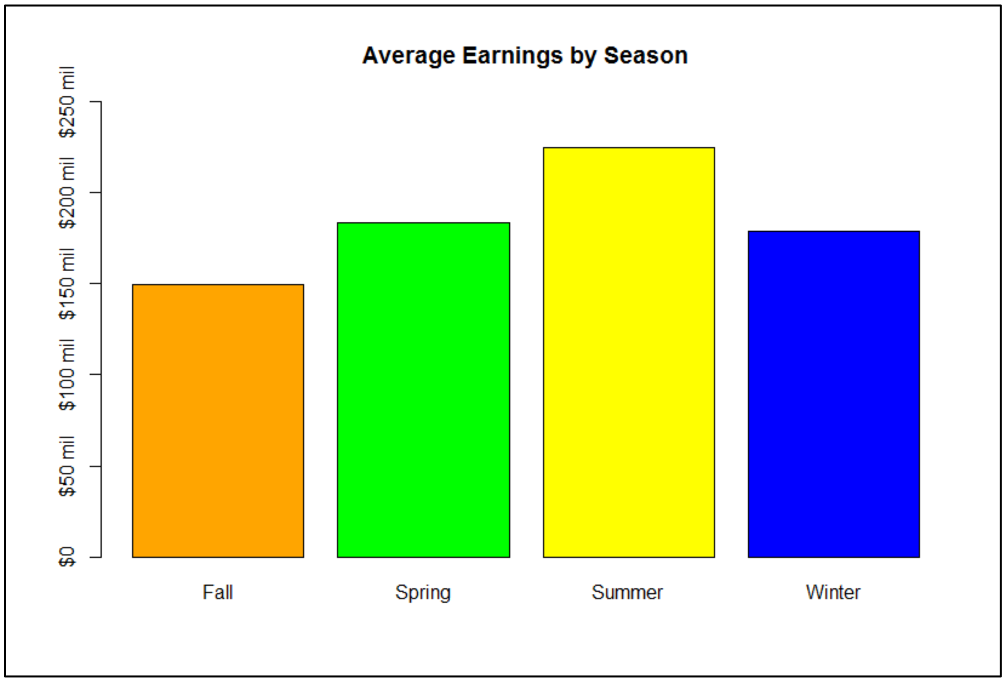


The pink bar on the right represents the missing values. Aside from that, this distribution appears to have a more normal distribution, which will help in our modeling.

The time of year when a movie is released tends to have an impact on its performance. Rather than looking at each month of the year, we will split the months into seasons and use that as a new feature in the model. I placed the months into the following seasons:

Spring	**Summer**	**Fall**	**Winter**
March	June	September	December
April	July	October	January
May	August	November	February

We can then look at the average earnings by season:



As expected, summer films tend to earn the most, with spring and winter not too far behind. This seems like a fine addition to the model.

Fitting a Model

Now that we've looked through our data and adjusted as necessary, we can try fitting a linear regression model. I ran through several iterations and decided on the following model:

log(gross_adj) ~ log(budget) + rating + season + widest_opening

Here are some of the model diagnostics:

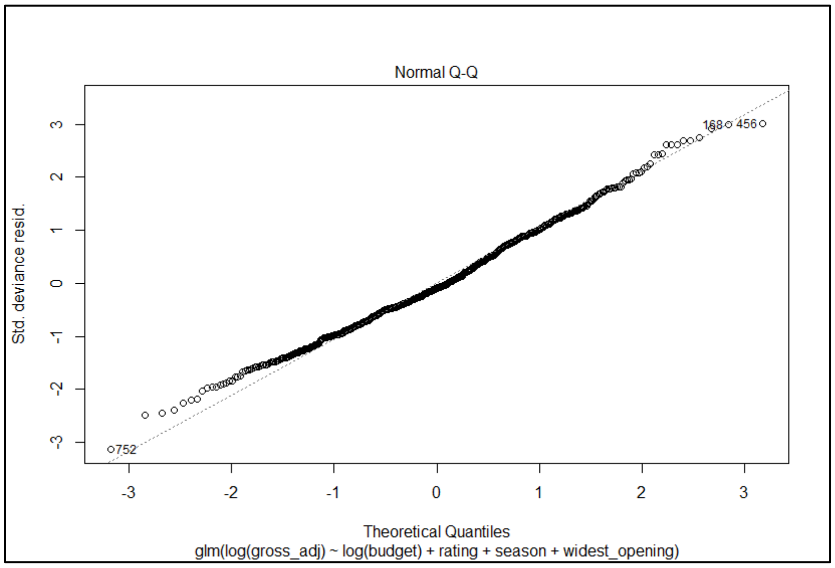
```
Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)     1.430e+01  5.521e-01  25.908  < 2e-16 ***
log(budget)     2.596e-01  3.354e-02   7.742 3.70e-14 ***
ratingPG       -1.075e-01  1.586e-01  -0.678  0.49796
ratingPG-13    -2.722e-01  1.529e-01  -1.781  0.07542 .
ratingR        -4.261e-01  1.566e-01  -2.721  0.00667 **
seasonSpring    1.275e-01  6.180e-02   2.064  0.03944 *
seasonSummer    3.632e-01  6.043e-02   6.009 3.08e-09 ***
seasonWinter    8.422e-02  6.448e-02   1.306  0.19193
widest_opening  1.793e-05  4.231e-05   0.424  0.67189
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 0.3097568)

    Null deviance: 285.04  on 667  degrees of freedom
```

```
Residual deviance: 204.13  on 659  degrees of freedom
  (404 observations deleted due to missingness)
AIC: 1123.8
```

Here is a QQ plot of the residuals on this model, which appears to be normally distributed:



While it is by no means a perfect fit, it seems to be a suitable model for giving a rough estimate of domestic earnings.

Creating a Shiny App

Now that we've collected the data and fit a model, it's time to build the app that users can interact with. We will do so using Shiny Apps, a framework for creating interactive apps in R.

To create a Shiny app, we need to create two functions in R. The first is a ui function that controls which elements the user can interact with and determines the layout of the app. My ui involves two slider inputs, one for the movie's budget and another for the widest theater opening. It also includes two drop-down menus where the use can select the film's MPAA rating and the season of the release. These drop-down menus will also control the appearance of two histograms that appear on the app.

The second function is a server function that controls what the server is calculating behind the scenes. This is where the calculation of the model result happens and where the histograms are generated according to the value in the drop-down menu.

The goal of these two functions here is to create a system by which the user can select their own budget, widest theater release, rating, and release season and receive an estimate for how much money that movie would make. The code that I wrote also includes some additional histograms to give the user more information about how films with similar attributes tend to perform.

Once these two functions are completed, we can compile the app locally by running the following function: shinyApp(ui, server)

This compiles both the ui and the server code into a functioning app. This version is only running locally, but we can use this code to deploy a version of this app to the cloud via Shiny Apps.

Deploying the Shiny App

Deploying the app requires a Shiny Apps account, which can be created here: https://shinyapps.io. This process is made possible through the rsconnect package, which you will need to install if you want to do this process yourself. Once you have created an account, you will receive an access token and a secret token. These are used to establish a connection between Shiny Apps and your computer. The following function establishes this connection:

```
rsconnect::setAccountInfo(name='<your-name>', token='<your-token>', secret='<your-secret>')
```

I have removed my information from this, but if you want to deploy your own version, you can use your own token and secret to do so.

Once a connection is successfully established, we can use the following code to deploy the app:

```
rsconnect::deployApp("<your-file-path>/Shiny App")
```

You will have to replace your file path to the location of the Shiny App folder on your local drive. Simply run this code and rsconnect will take care of the rest. The deployed app will automatically pop up in a browser on your machine. You can then access it from any computer with access to the internet.