Forecasting Broadway Show Gross Revenue

OIT 367 final project

Ian Boneysteele, Konstantine Buhler, James Kernochan, Mike Mester, and Soren Sudhof

You can keep our project for future use

Executive Summary

In this project, our team took on the task of predicting and identifying drivers of weekly gross revenue for Broadway shows. One group member's experience working with Broadway producers revealed the need for rigorous, data-driven analysis and decision making in the Broadway industry, which traditionally has not employed sophisticated quantitative analysis extensively. Broadway is a big business, with total annual ticket revenue above one billion dollars and a large number of Broadway-dependent businesses in New York's Theatre District contributing even more economic impact. Broadway shows also draw significant investment interest. Use of more advanced statistical models remains relatively rare on Broadway, however, with many producers relying primarily on domain expertise, experience, and intuition to make decisions. With this project, we hope to supplement those decision making methods with quantitative analysis.

Our key task was to predict weekly Broadway gross revenue, both on an aggregate basis across all shows and on an individual, per-show basis. Gross revenue is highly variable from week to week and influences many of the decisions producers and other Broadway stakeholders make. Predicting weekly gross within a commercially useful margin of error would therefore enable better decision making across the Broadway ecosystem, as well as revealing drivers of commercial success or failure of shows. To improve the accuracy of our predictions and to uncover novel relationships, we cast a wide net in selecting predictors. In addition to basic time series and show capacity / frequency data, we examined the predictive power of show genre, seasonality and holidays, weather and financial indicators, and social data in the form of Google Trends search popularity. Our goal was to use these predictors to generate a predictive model with minimum root-mean-squared error (RMSE).

Although our project used only public datasets available on the Internet, data collection and processing proved to be a large hurdle. No comprehensive, machine-readable database of Broadway shows exists, so we had to generate our own dataset by scraping Broadway industry websites. Similarly, collecting search popularity posed difficulties given the opaque nature of the Google Trends data and the lack of a convenient way to programmatically query the Google Trends database. The data we collected was also large in scale: weekly gross revenue for 1,087 shows back to 1985 with run lengths ranging from a few weeks to several years, for a total of nearly 39,000 individual rows of data. Including lag terms, interaction terms, and binned categorical variables, our final comprehensive model contained over 4,000 predictors for each of these rows, requiring a large amount of computational resources.

Our efforts resulted in impressive prediction accuracy for our comprehensive model. Our average out-of-sample RMSE for total weekly Broadway gross across 20 cross-validation folds was \$818k, less than 5% of a typical week's total gross. On a per-show basis, our average out-of-sample RMSE for weekly gross was \$85.5k, relative to a typical weekly gross for a larger show of above \$1M. Interestingly, most of the model's predictive power came from autocorrelation and interaction terms on the basic time series and capacity data rather than the external weather, economic, or search popularity data. The marginal accuracy provided by the external data was small to nonexistent, well below any reasonable threshold for statistical significance.

These results suggest three key recommendations for the Broadway ecosystem. First, our project has proven the viability of analyzing and predicting Broadway grosses quantitatively using multiple linear regression on a broad set of public predictor datasets, and we recommend that Broadway stakeholders adopt similar methods. For instance, Broadway producers could use our model's predictions to help them determine when to shut a poorly-performing show down and when to ride out a slow period in expectation of improved future performance. Second, the predictable cyclicality of Broadway grosses indicates an opportunity for Broadway ticket sellers to vary ticket pricing in response to expected demand, increasing revenue by charging more during peak periods and selling more tickets at a lower price during demand lulls. Third, the relative unimportance of Google search volume to Broadway grosses casts doubt on the viability of search keyword advertising for increasing Broadway ticket sales, though further work is required to make a definitive recommendation on this point.

1 Introduction and Description of Problem

Broadway theater in New York City is one of the major attractions in one of the world's most fashionable cities. Broadway is, along with the West End in London, considered to offer the highest level of quality and exposure for theatrical endeavors in the world. Shows on Broadway feature many of the world's biggest star actors and are massively popular both among residents in the New York metropolitan area and among tourists coming from all over North America and from across the globe. Consumers have shown a remarkable willingness to pay premium prices for a few hours of entertainment, with tickets prices approaching $500 a piece for exceptionally popular fare such as *The Book of Mormon*.

"Broadway" specifically refers to a set of approximately 30 theaters either on or adjacent to the street Broadway in midtown Manhattan. Every single show in a Broadway theatre must report, to the exact dollar, the amount of revenue earned from ticket sales in a given week. These numbers are then publicly released every Monday afternoon for anyone to peruse on the web. BroadwayWorld.com represents one news source for this data, providing a particularly comprehensive portrait of show data each week. As can be seen at http://www.broadwayworld.com/grosses.cfm, each show is listed by title along with its particular theatre's name (e.g., *The Book of Mormon* plays at the Eugene O'Neill Theatre), the current week's ticket gross, the prior week's gross, the difference between the two, the average ticket price, the top ticket price (i.e., the highest price paid for the best seat in the house), the number of performances in a week (usually eight, spread across six days), the number of available seats across all the performances, the number of those seats for which tickets were purchased, the percentage of available seats filled, and the percentage of maximum possible gross achieved. The last of these numbers is perhaps the trickiest to determine and the easiest to abuse, as it is a complicated and sometimes shifting metric; shows have sliding scales for ticket pricing, dipping as low as $20 for desperation-prompted discounts (below the "maximum" gross for the seat) and vaulting as high as $475 for a premium seat (well above the supposed "maximum" non-premium price for the seat).

Our project is looking to find the primary factors influencing ticket revenue for Broadway theater productions in New York City between 1984 and today, the full range for which ticketing data are available on the web. One of our target audiences is theatrical producers. This audience needs to understand revenue projections for their shows for budgeting purposes, and finding the factors with the most predictive power will be very helpful for resource allocation and planning. Shows must meet a given weekly running cost in order to stay open, and this often involves moving past weak weeks wherein a certain amount of money is lost. Show producers can better plan for these periods if they know how to predict revenue trends into the future. Producers can also get a sense of whether advertising should be increased along certain avenues, as word of mouth may build below or above the pattern given by our data analysis. Producers must also make critical decisions about ticket pricing, including whether to feature their respective shows at discount-ticketing booths (such as the well-known "TKTS Booth" featured prominently in Times Square). Most shows begin with very elaborate systems of discounting in order to pique consumer interest, such as special arrangements with major corporations in the area and flyers mailed to homes of previous patrons. More streamlined information about ticket revenue relative to benchmarks can provide producers with useful decision-making tools about how to structure, curtail, or alter discount or premium ticketing schemes.

There is also a broader economic ecosystem around Broadway whose revenues are directly tied to business from the shows. A large infrastructure of restaurants, gift shops, hotels, and convenience stores is situated in Midtown West. These economic stakeholders have access to the publicly available information about Broadway grosses, and so they could benefit from a general picture of how Broadway shows are likely to perform in a given week, taking into consideration other relevant factors such as weather and time of year.

2 Data Collection

All of the data we used for this project was taken from publicly available websites: Broadway World (http://www.broadwayworld.com), the Internet Broadway Database or IBDB (http://www.ibdb.com), weather and financial data from government sources, and Google Trends (http://www.google.com/trends). Unfortunately, none of these sites offers a public API to access its data, requiring us to scrape each one individually. The Python scripts we used to collect the data are presented in section §D. A complete copy of the processed, cleaned database we used in our analysis is available at http://tinyurl.com/n6qfyfw.

The Broadway World database provides our primary response variable, weekly grosses, and several other basic capacity and frequency data for shows, e.g., number of performances in a week, number of seats available, and average and maximum ticket prices. The data is mostly complete, except for seat and ticket price data before approximately 1996. Show runs often contain gaps and some shows leave Broadway for a long period before returning. Generally, Broadway World treats a revival of an old show as a separate show from the original, and we have followed that convention in our analysis.

There are 1,087 separate shows that have been on stage during the period from 1985 to the present. The shows range from very short runs (three weeks for total failures) to what is among the most commercially successful shows of all time, *The Phantom of the Opera*, which has run since 1988. In total, this dataset represents nearly 39,000 rows of data, each representing a week's worth of grosses for a single show. Weather, general economic data, and holidays are also incorporated into our analysis, in order to determine their particular impact on the flow of ticket sales throughout the weeks.

Broadway World's data is available by show at `http://www.broadwayworld.com/grossesbyshow.cfm`. Data for each show appears in a relatively standardized HTML table containing several columns. The main grosses page linked above contains an index of all shows available in the database. To collect this data, we wrote a Python script (see Script 3). This script first creates a list of all the shows available by crawling the Broadway World index. Then, the script issues an HTTP request for each of the shows and receives an HTML table containing Broadway World's data in response. The script then parses the HTML table using the BeautifulSoup Python library and outputs the result to an XLSX file. We save each of these files locally to avoid overloading the Broadway World site with requests each time we want to regenerate our data set. Finally, the script takes all of the per-show XLSX files and concatenates them into a combined XLSX file, which forms the foundation of our data set. Each row of this combined file represents a single week of a single show.

To supplement the Broadway World data, we added show type data from IBDB. Analogous to IMDB for movies, IBDB provides a comprehensive database of Broadway shows, including cast, credits, theatre, and categorical data about the show's genre. IBDB's data is based on playbills that accompany each production, provided by The Broadway League.[1] Although IBDB's data is rich, it is not structured in an easily machine-readable format. Subtle and unpredictable mismatches between names of shows in Broadway World and IBDB further complicate scraping of the IBDB dataset.[2] Given these complications, we focused on retrieving a single feature from IBDB: the category of shows. There are three such categories: play, musical, or other (typically short-running specials).

Our IBDB scraping used another Python script (see Script 4), this time using the Mechanize library to emulate a web browser. The script takes a list of show names (in our case, from Broadway World) and searches the IBDB database for each show. Because IBDB sometimes contains multiple entries for each show with different categories, we had to develop a heuristic for obtaining a single category. We chose simple majority voting, where we assigned each show the category that appeared most frequently in the IBDB results page. For most shows, the voting was unanimous, and spot-checking the non-unanimous shows indicated that the majority-rule heuristic was reasonable. Searching IBDB returned no results for 138 of the shows in our dataset, mostly for revivals where the Broadway World naming convention differs from IBDB. For these shows, we created a "NA" category to indicate missing data.

To include exogenous factors beyond Broadway itself, we supplemented the Broadway data with NOAA weather data for New York City (the Central Park weather station in particular) and the closing price of the Dow Jones Industrial Average stock index. We included weather predictors to test the intuitive hypothesis that poor weather would negatively affect Broadway grosses. Similarly, we included stock market data to test the sensitivity of Broadway revenue to broader economic trends. Given the important of the financial services industry to the New York City economy, Broadway revenues could be highly sensitive to stock market performance, which would be an important insight for prospective investors in Broadway shows.

The final data set we used was Google Trends data on search term popularity. We included Google Trends to identify the importance of "social" popularity for Broadway grosses and to expand our set of predictors beyond conventional Broadway data sources. Put simply, Google Trends data describes the relative popularity of search terms on a weekly or monthly basis back to 2004. Data are available for many search terms, but some are so rare that no Trends data exists.

Google Trends proved the most difficult of our four data sets to scrape, parse, and analyze. We focus here on the scraping and parsing. We discuss the analysis of the Google Trends data separately in section 4.5. The only public access to Google Trends data is through the interactive website `http://www.google.com/trends`, which is designed for casual human use and not for automated scraping. No API for Google Trends is available, despite Google announcements to the contrary. Examination of the Google Trends website, however, revealed a function that produced a CSV of Trends data for specified search terms with a relatively simple HTTP GET request. We then wrote a Python script to issue these requests for each of the shows in the Broadway World list (given that Trends data is only available post-2004, many shows had no data at all). For each show name, we issued four requests

[1] See `http://www.ibdb.com/about.php`.

[2] For instance, Broadway World typically appends the opening year to the name of a show that has been revived whereas IBDB does not.

with different search queries: the show name, the show name plus "tickets," the show name plus "Broadway," and all three of these queries together. The "tickets" and "Broadway" queries help to distinguish show-specific interest from general interest in a search term. Since many show names are common searches on their own (e.g. *Cats*), search interest in the show name alone may not be predictive. The final combined Trends request is needed to handle the normalization of the Trends data that Google provides, which is discussed in detail in section 4.5.

While not expressly prohibited, mechanized querying of Google Trends is discouraged by Google. The nearly 5,000 total queries we needed to make to the Trends website fell well outside what Google's unpublished quotas allowed. As a result, our script was initially throttled after the first few dozen requests, effectively disabling our access to the Trends dataset. To work around this limitation, we had our script wait between requests for between 10 and 30 seconds and we included cookies in our request that simulated a logged-in Google user. After some experimentation, we were able to complete our queries for all 1,087 shows with the script presented here. Ongoing retrieval of the Google Trends data would likely require either more sophisticated methods for avoiding throttling or explicit cooperation with Google.

Our script then parsed the Google Trends CSVs and created a table of combined Trends data for each of the four query types described above. To ease analysis of the data, we presented the Trends results for each show by week, relative to the show's opening. Weeks without Trends data were assigned a value of zero. Finally, we used R to generate several predictor columns from these tables, which are described in section 4.5.

3 Methodology

After we collected the data, we divided our analysis into two stages. First, we examined the predictive power and effect of each category of predictors separately: the basic capacity and time series data (section 4.2), show categories (section 4.2), seasonality and holidays (section 4.3), weather and financial data (section 4.4), and Google Trends data (section 4.5). These separate analyses helped reveal which of the many predictors we collected actually pertained to Broadway and which were not useful, which in turn guided our predictive model building. The analyses also helped us to understand the drivers of Broadway grosses, which provides useful insight for Broadway producers managing shows. We discuss each of these analyses individually and then discuss our final comprehensive predictive model (section 4.6).

For our initial analyses, we used multivariable linear regressions, usually with autocorrelation terms. Due to the large number of predictors involved, we switched to a LASSO linear regression for our comprehensive model.

All models were evaluated using mean RMSE from 20-fold cross-validation. The cross validation was performed by taking random starting points in the data set and then selecting the first 5,000 rows after that point as a training set and the following 2,500 rows as a test set for determining RMSE. This process was repeated 20 times for each model, using the same splits by setting a common random seed, and the model's RMSE was estimated as the mean of the RMSEs from the cross validation.

4 Analysis and Results

4.1 Baseline model

To facilitate our analysis of the various predictors, we created a simple baseline model of total weekly gross containing only three autocorrelation terms, basic capacity data like number of seats and performances, and a summary of the genres of the shows currently playing. This model produced an RMSE of 1.73e+06, and its code and results are presented in Figure 13.

4.2 Capacity predictors and autocorrelation

We started our detailed analysis of the data by examining the basic macro factors that potentially drive Broadway total gross sales. As we hypothesized that total gross sales would be increasing over the years (as Broadway has proved to be a growing industry), we started by regressing total gross sales on Week (our variable for time, see Figure 14). As we hypothesized, this was statistically significant and showed that total gross sales increased over time. This simplistic model had an RMSE of \$2.98M and, realizing the power of intuition, we added the week of the year (as Broadway is seasonal), total number of performances, and number of shows being performed to the linear regression model (Figure 15). This showed that all factors were statistically significant and drastically improved the RMSE to \$2.01M. As might be expected, the number of performances of a show in a week is positively correlated with total gross, showing that volume outweighs dilution effects. Contrary to expectation, though, the number of different shows playing at once was negatively correlated with total gross, showing potential dilution effects. Given the significance of the number of shows being performed, we added number of musicals being performed to the linear regression (Figure 16). This indicated that while the number of shows is negatively correlated with total gross, the number of musicals is positively correlated. After seeing the significance of this effect, we added in the ratio of the number of musicals to the number of all types of shows (Figure 17). This left RMSE relatively unchanged at \$2.02M in both models. Indeed, while this may suggest that the ratio is the key driver, removing the number of musicals being performed reduces the RMSE of the model and lowers the statistical significance of the ratio interaction variable.

Next we explored the impact of the total number of seats available on Broadway on total gross sales. However, because total number of seats information is only available for post-1996 (Figure 7), we restricted our analysis to this timeframe. We see that total number of seats is statistically significant and explanatory (Figure 18), reducing RMSE to \$2.59M from \$2.81M (which was the RMSE of model restricted to post-1996 data). As we suspected there could be a non-linear relationship we explored the square of total seats (not significant on its own, RMSE = \$2.61M), the cube of total seats (not significant on its own, RMSE = \$2.61M), and the square plus the cube of total seats (which is significant although increased RMSE to \$3.02M, see Figure 19). This indicated the declining impact on total gross of incremental seats as total seats increases. In line with later analysis, we also explored autocorrelation (Figure 20), although this idea combined with the significance of Total Seats, led to the idea that prior period average price could be a more powerful driver. Indeed, we see that the introduction of average price to the linear model eliminates the significance of prior period total gross (Figure 21). A PACF chart is presented below, indicating at least marginally significant autocorrelation up to approximately the $AR(20)$ term (and some beyond). Given this result, we included lags up to 20 weeks for our comprehensive model, described below in section 4.6.

Having established the importance of autocorrelation, we explored the impact of show aging (median and mean age of shows currently playing) and saw median show age's statistical significance (Figure 22 and Figure 23). In this analysis we see that the older and more established the Broadway line-up is, the larger Broadway's total gross sales. Finally, we explored the quadratic and cube for number of performances being performed as we suspected that there was a non-linear relationship similar to the total seats relationship (Figure 24 and Figure 25). Here we see that, unlike total seats, incremental total performances have a continuing positive impact on Broadway total gross.

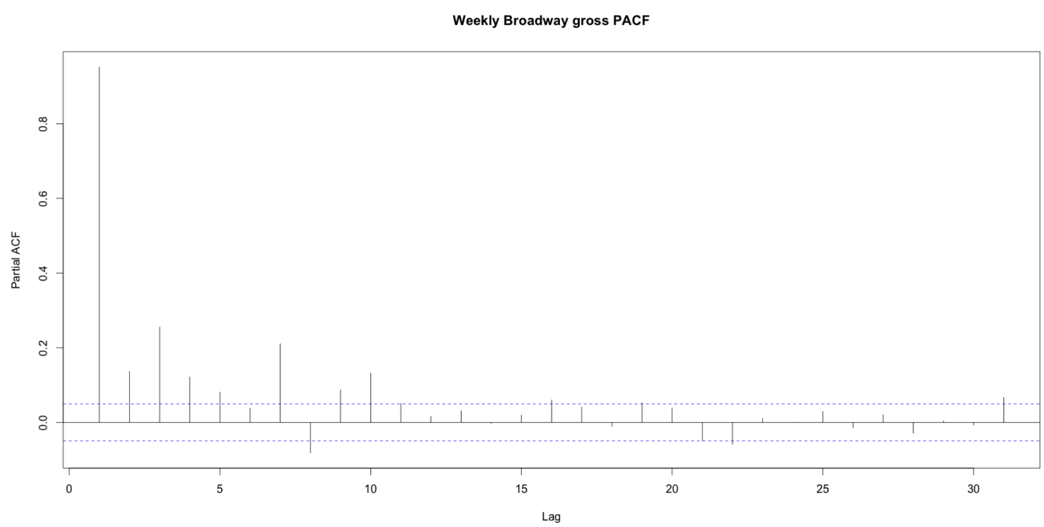

Figure 1: PACF chart for total weekly gross

4.3 Seasonality and holidays

A quick view of the data (Figure 2) immediately produces an impression of a strong seasonal element to bookings. Total Broadway grosses seem to peak in the summer, around May, and of course there is a general growth trend over the entire timeframe (see Figure 10 for an explicit presentation of secular growth in total weekly gross). One reason for the seasonality is the occurrence of public holidays and timing of school vacation schedules, which attract tourists and add to the number of performance slots. Another is the industry-specific calendar – entries for the Tony awards, usually held in June, have to be submitted and released by April and can influence a show's commercial success dramatically. And the long-term trend is likely the result of macro factors such as GDP growth and demographics.

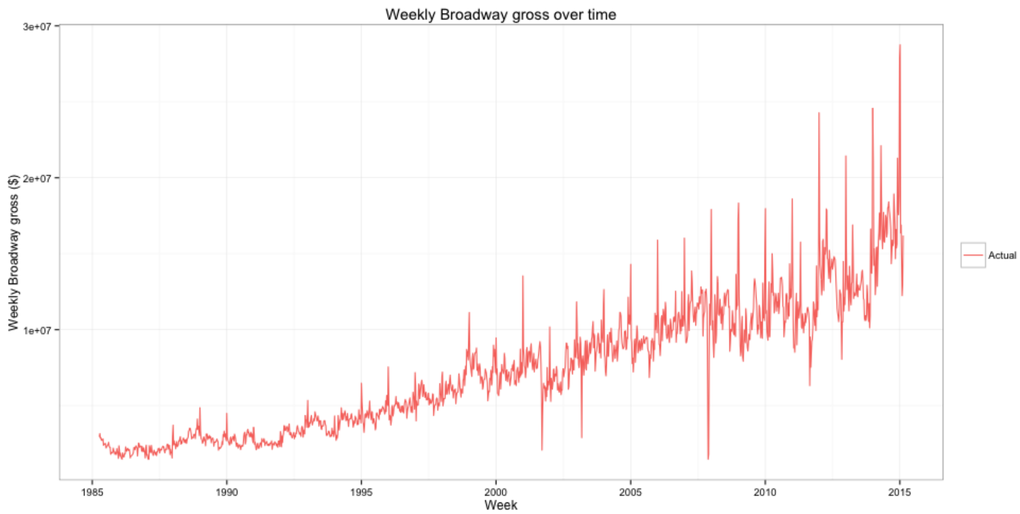

Figure 2: Weekly total Broadway gross over time

The amount of noise in weekly grosses makes quantitative predictions of these seasonal effects difficult, however. As an example, Figure 3 shows that even the intuitively obvious quarterly seasonal patterns are difficult to detect in the weekly gross data. With the exception of the fall season, differences between seasons appear much smaller than one might expect after looking at the overall time series. This noise made analysis of seasonal and holiday predictors in isolation difficult.

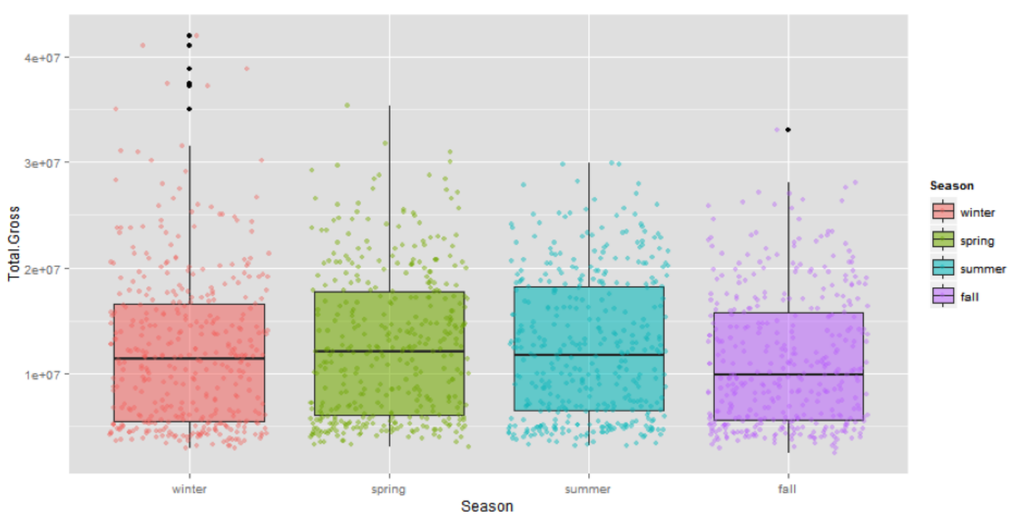

Figure 3: Seasonal variability in total weekly gross

Thus, in order to incorporate variables in our models that might improve our forecasting accuracy, we extracted week of year (sequentially), monthly, quarterly (effectively seasonal), and annual elements from the week start dates. Isolating the week number enabled us to regress much more granularly on specific points in the year, such as potential holiday dates. Month and quarterly variables were intended to capture more general fluctuations and holidays that occur in different weeks in different years (such as Easter). Using year as a variable was intended to capture the trend across the timeframe.

In an attempt to better isolate the specific holidays, we also used the `zoo` and `timeDate` R packages to import public holiday data for the last few decades and then flag weeks within a certain interval of days from the holiday dates. We experimented with two different lists of holidays: one of all G-7 public holidays (with the logic that some portion of Broadway revenues are tourist-driven and thus wealthy country holidays might be predictive), and one of NYSE public holidays. The former dataset proved not to improve our model, likely due to the high number of holidays it contains – nearly every week was flagged. We also created variables flagging weeks around Christmas and Thanksgiving, as these anecdotally are particularly important holidays on the Broadway calendar and likely drive the large spike in grosses that reliably occurs near the end of the year.

We evaluated the effect of these additional variables on our base model (Figure 13) by building a model incorporating all seasonality indicators. The results (Figure 8 and Figure 26) indicate that season variables, the months December, March, May, August, and November, the week of the year, the year itself, and the occurrence of NYSE holidays are all significant predictors. Surprisingly, Christmas and Thanksgiving variables were not significant by themselves. In the comprehensive model (discussed in section 4.6), however, the seasonal predictors are responsible for a decrease in RMSE of $251k, the largest and most significant effect among all the predictor sets we tested ($t = 3.85$, $p << 0.01$). Much of the improvement appears to have come from interaction terms between the seasonality variables and lagged weekly gross.

4.4 Weather and financial predictors

Before we put all the sources of data together and ran LASSO, we ran a few tests with a linear model. We found in this linear model that financial data was quite significant in predicting the revenue. In fact, it reduced our baseline RMSE over 6%. Weather, on the other hand, had only marginal implications for the linear model's RMSE.

However, when we ran the LASSO algorithm on the complete dataset, we found that weather made almost no difference in the accuracy of our model. Including weather and financial data in the comprehensive model decreased mean RMSE by only $28 per week, which is highly insignificant given standard errors of RMSE on the order of several thousand dollars ($t << 0.01$, $p > 0.99$). The irrelevance of weather could be due to much of the weather information being carried by the seasonality variables (e.g. winter in New York generally has worse weather than summer does) or simply because Broadway grosses are not as susceptible to weather as intuition might suggest.

4.5 Google Trends

To measure the predictive power of social engagement and interest for Broadway grosses, we included Google Trends data on the popularity of searches for the various Broadway shows. Our basic approach was to use search popularity for a show from prior weeks to help predict the gross for that show for a given week. As described in section §2, we tested the predictive power of several search strings: the show's name, the show's name plus "tickets," and the show's name plus "Broadway."

Google Trends data is a weekly or monthly (depending on search term popularity) normalized measure of the volume of searches for a set of search strings. Although Google does not clearly describe the normalization they use, examination of the data reveals the likely process. For Google Trends queries with a single search string, the Trends value for a given week is the search volume for that week divided by the maximum search volume observed for that term over the period of the query, rounded to an integer between zero and 100. Because all of our Trends queries were for the entire range of available data from 2004 to the present, the denominator is the maximum observed search volume for a given term. For Trends queries with multiple search strings (e.g. our combined query for the show name, the show name plus "tickets," and the show name plus "Broadway"), the normalization denominator appears to be the maximum search volume across all of the search strings. An example of the Google Trends results for *The Book of Mormon* is below in Figure 4, with annotations indicating key events.

The Trends normalization presents several challenges for analysis. First and most importantly, the raw Google Trends data from the past cannot be used for prediction, because it incorporates future information in the form of its normalization denominator. The true normalization denominator is not given, however, so we cannot simply multiply by it to recover the raw search volume. Instead, we divide each week's Trends data point by a fixed Trends value from before the show opened (e.g. a quarter or a year before open). This division causes the normalization terms to cancel and decontaminates the Trends data point from future information. For denominators, we tried average Trends data for one month prior to show open, three months prior, six months prior, and one year prior. Our query with multiple search strings also allows analysis of the relative frequency of different search strings, which is impossible with the normalized single-term queries. We accomplished this by simply dividing the "tickets" and "Broadway" Trends values by the value for the show name alone. Unfortunately, however, shows whose names are common search terms on their own (e.g. *Cats*) typically have "tickets" and "Broadway" search volume low enough

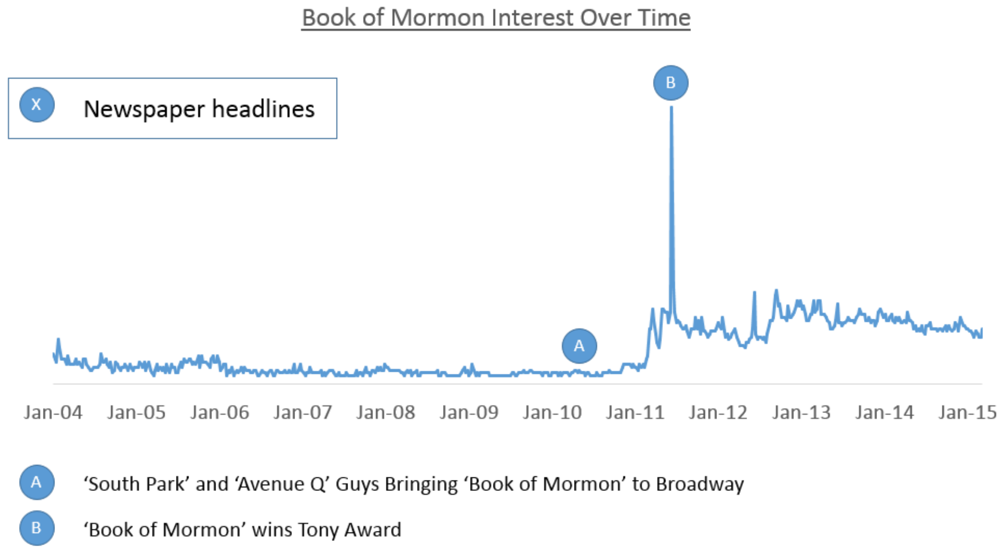

Figure 4: Google Trends results for *The Book of Mormon*

relative to the simple show name's search volume to round to zero in this relative data. Given the importance of lagged terms in the Broadway grosses data, we included lagged versions of these variables up to 10 weeks. We also added fixed average Trends values for two, four, six, 12, and 26 weeks prior to show opening. Any missing data (e.g. for shows before Trends data is available) was replaced with zero.

Because of the large number of predictors introduced (a total of approximately 150 including the different denominators and lags), we used a LASSO linear regression to determine the predictive effect of these Google Trends variables. We also included all of the basic capacity and frequency data in the Broadway World database along with lags of weekly gross back 20 weeks.

Only the sixth and seventh lag of the "tickets" search relative to searches for a show's name had a non-zero coefficient in the LASSO model among the non-interaction Google Trends terms (excluding a single average Trends term with a coefficient near zero). Both coefficients were positive, indicating a positive relationship between frequency of searches for a show's tickets and gross for that show. The lags perhaps suggest that the most important search activity for tickets occurs six to seven weeks before a customer attends a Broadway show, though more more analysis would be required to confirm that hypothesis. Interactions between the Trends and lags of grosses produced ten more non-zero coefficients (see table table 1 for details). These interaction terms capture the more natural notion that search interest in a show might bend the show's gross trend up or down relative to its existing trajectory, rather than produce a fixed increase or decrease in gross, as the coefficients for the non-interaction Trends terms imply. Nevertheless, removing the Trends data from our comprehensive model only increases RMSE by approximately \$8k, a statistically insignificant amount ($t = 0.16$, $p > 0.85$), indicating that the Google Trends data adds no meaningful predictive power to the model.

4.6 Comprehensive predictive model

The final step in our analysis was to combine all of the predictors we examined in the earlier sections into a complete, comprehensive model. Given the large number of predictors (a total of 239 including lag terms, with one predictor as a factor with over 1,000 levels), we used LASSO to select important variables and avoid over-fitting. We ran the model on a per-show basis both to improve its predictive performance versus an aggregated model and to improve its usefulness to show producers interested in predictions for a single show. R code for building this model is presented in Script 1.

Our comprehensive model included all of the predictors described above, plus interactions with all of the predictors and lagged grosses back to seven weeks. We included the interactions to capture the intuition that most of these predictors are primarily important relative to a show's existing trajectory of grosses. Given the high variability of typical weekly grosses among shows and from week-to-week for a single show, predictors were seldom important on their own. For instance, determining a single value for all shows to capture the additional gross around Christmas (corresponding to the coefficient of a Christmas term) is very difficult. A more natural notion is the amount by which Christmas increases grosses relative to last week or the week before that, which is captured by the interaction terms. We chose interactions back to a lag of seven weeks based on the PACF chart (Figure 1) indicating that lag terms up to $AR(7)$ are particularly significant. Ideally, we would include interactions with lag terms up to $AR(20)$ given that many of these terms are significant as well, but computational limitations prevented us from including that many variables. Even the model presented here takes several hours to run through its 20-fold cross validation. Adding the interactions terms improved the RMSE of the model by \$144k over a model without any interaction terms, which is significant ($t = 2.15$, $p < 0.05$).

Finally, we tried normalizing all of our variables using R's scale command, due to the large difference in magnitude between the weekly grosses (which are on the order of 10^6) and the many predictors between zero and one. Normalization only improved aggregate gross RMSE by \$3k, which was insignificant ($t = 0.06$, $p > 0.90$). Normalization also increased per-show RMSE by about \$1k, though this was also insignificant ($t = 0.13$, $p > 0.85$). Given the ambiguous performance improvement from normalization and the more natural interpretation that attaches to unnormalized variables, then, we chose to use the unnormalized model for prediction and analysis. A comparison of the model's performance on aggregate weekly gross with different predictor sets is presented in Figure 5 (a comparison of per-show model performance is presented in Figure 12).

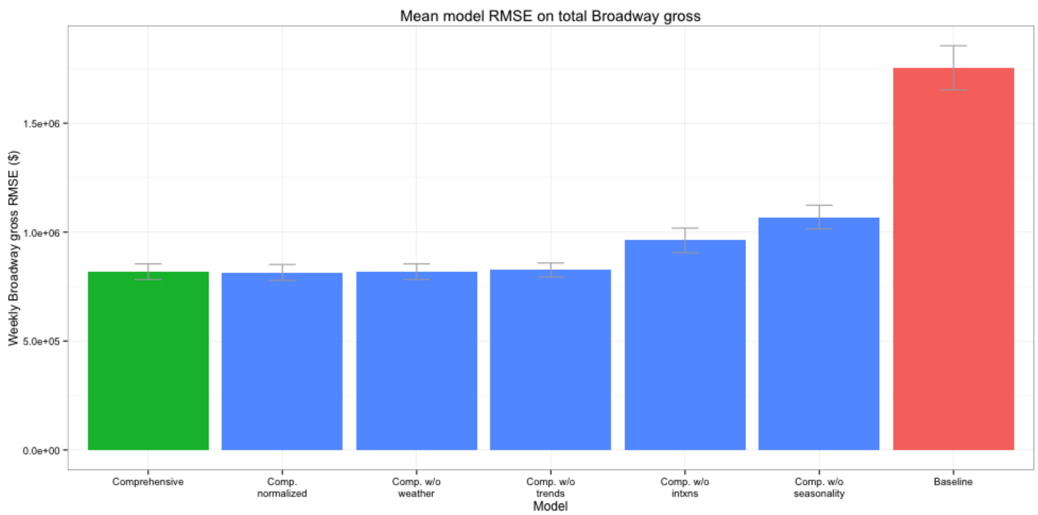

Figure 5: Comparison of RMSE on aggregate weekly gross across models

The predictions of this comprehensive model are presented in Figure 6. The model produces an RMSE for total weekly gross of \$818k (SE \$35.8k), which is less than 5% of a typical week's gross on Broadway. On a per-show basis, the model achieves an RMSE of \$85.5k (SE \$4.5k). For higher-grossing shows (e.g. musicals), this RMSE represents only about 5% of a typical week's gross. For the smallest shows, the per-show RMSE represents approximately 25% of a typical week's gross, although the model may have reduced error on these smaller grossing shows. The coefficients for the 305 variables selected by LASSO are presented in table table 1. There is no clear pattern to the selected variables except that interaction terms predominate (252 vs. 53 non-interaction), and the variables cover a wide range of the predictor types we examined.

5 Principal Findings and Recommendations

The most important finding of our work is that Broadway grosses are highly predictable even with widely available predictor variables. The ability to predict general Broadway grosses for a given week is a powerful tool for business owners who rely on the traffic of patrons for Broadway shows. A restaurant owner may purchase a certain amount of food for consumption in a given week based on a prediction of whether Broadway grosses will be weak or strong, and an accurate model can prevent either waste of food or inadequate resources. While not all of the information in our analysis can be gathered in advance (particularly an accurate prediction of the DJIA), the most important predictors are either publicly available prior data or reasonably estimable. Because the restaurant owner's business is directly tied to Broadway grosses, the model should also enable him or her to figure out resource demands within a similar level of accuracy.

Our comprehensive model's predictions are presented in Figure 6 and closely track the actual grosses for most periods. Although our model has a tendency to underestimate peak mid-summer and winter holiday weeks (despite including related terms in our predictor set), it is highly accurate for the vast majority of the year. The comprehensive model's RMSE is also almost $1M per week lower than the simple baseline model we used. This increased accuracy enables better demand forecasting and pricing decisions for Broadway producers, better capacity and supply decisions for Broadway-dependent businesses, and more sophisticated show-closing decisions for producers and investors. We would therefore recommend that these Broadway stakeholders use techniques like our comprehensive model to improve their decision making. In particular, producers should use similar predictive modeling to assist decisions about whether to close an underperforming show or keep it open, particularly for larger shows where our model's accuracy is higher.

Figure 6: Model predictions vs. actual total weekly Broadway grosses since 2004 (full time series in Figure 11)

Our analysis also produced several interesting insights into the dynamics of the Broadway ticket market. The negative coefficient on number of concurrent shows (discussed in section 4.2) indicates that demand for Broadway shows is relatively fixed. Each additional show offered at a given time reduces total weekly gross across all shows by nearly $100k, suggesting that incremental shows induce price competition among shows and harm overall Broadway revenue. This analysis recommends that existing Broadway stakeholders should strongly resist attempts to expand the number of "Broadway" theaters, as some have attempted in the past. Another interesting result is that Broadway shows do not systematically age, in the sense that weekly gross is positively correlated with weeks since a show is opened (discussed in section 4.2). Producers and theaters therefore should not disfavor or discount otherwise successful shows just because they have been running for a long time. Audiences evidently do not become tired of long-running shows.

The seasonality analysis discussed in section 4.3 mainly confirmed and quantified the common perceptions among Broadway stakeholders about the importance of holidays and seasonality to Broadway returns. Seasonality and holidays were critical to our model's predictive accuracy, however, improving RMSE by $251k. Our difficulty in completely capturing the holiday and mid-summer gross peaks despite the seasonality factors we included in our model also suggests an avenue for further improvement to our predictive model.

While weather and stock market performance are obviously outside the control of Broadway stakeholders, our analysis of these predictors still provides useful insight for the Broadway ecosystem. The relative unimportance of weather and financial variables in predicting weekly gross (discussed in section 4.4) means Broadway returns are less susceptible to these exogenous, unpredictable variables than conventional wisdom holds. That result should make Broadway investments more attractive to a wider group of investors, benefitting producers by reducing the cost of funding shows.

Finally, our Google Trends analysis indicates that focusing on search term popularity to predict Broadway grosses likely represents misplaced effort. Google Trends only improved the RMSE of our comprehensive model by a statistically insignificant amount, at the cost of considerable effort in collecting, processing, and analyzing the data. Most of the information contained in the Google Trends data apparently resides in the time series of grosses themselves, a much easier dataset to obtain and manage. The low predictive power of Google search volume may also suggest that search keyword advertising is not particularly useful for increasing Broadway show revenue. Although our analysis cannot squarely address the question of search advertising effectiveness for Broadway shows, the low predictive power of search volume for Broadway grosses at least raises doubts about a connection between Internet searches and Broadway ticket purchase behavior.

6 Conclusion

The variety of data analysis we have undertaken has resulted in a number of potent and useful conclusions, and provides an apparatus for more effective business enterprises in the Broadway ecosystem. The per-show RMSE is particularly relevant for big-budget musicals, where a deviation beyond our margin of error demonstrates that a show is either outperforming or underperforming where static demand would lie, evincing a trend driven by organic consumer sentiment rather than exogenous factors. Producers are often caught up in emotional, high-stakes decisions about the financial stability of their shows, and about the level of advertising and subsidizing that are required to sustain the shows, and so having a reliable metric against which to measure their performance can enable them to make more rational decisions.

The RMSE across all the shows at once shows that business owners relying on Broadway can also use data from public and readily available sources in order to make rational decisions about how to structure their own businesses. The general trends depicted in our report also enable them to make quicker calculations and judgments about their businesses as well – noting, for example, that over Christmas a Broadway lineup filled with long-running musicals is going to mean a very healthy stream of potential customers coming their way. While weather, financial, and Google Trends did not provide commercially or statistically relevant information, the data available from IBDB and Broadway World all interacted with consumer patterns (and often with each other) in robust and remarkable ways.

A Figures and Tables

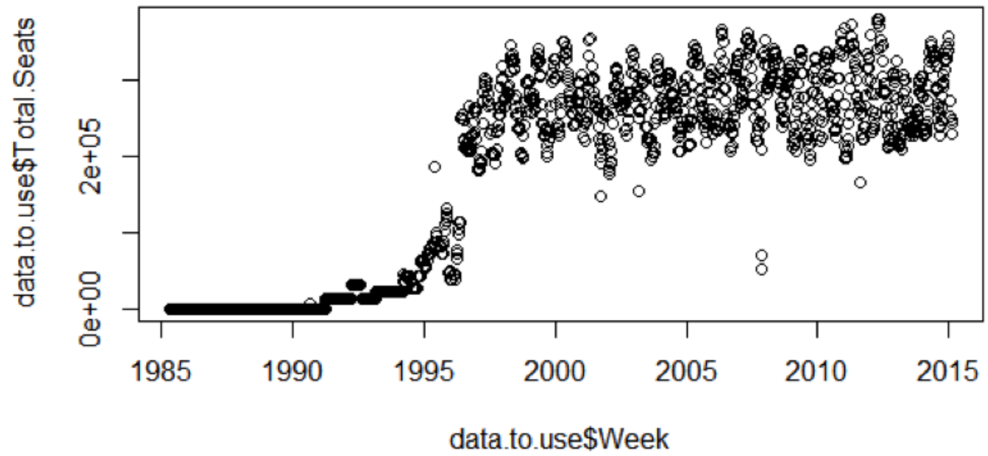

Figure 7: Total seats data availability by week

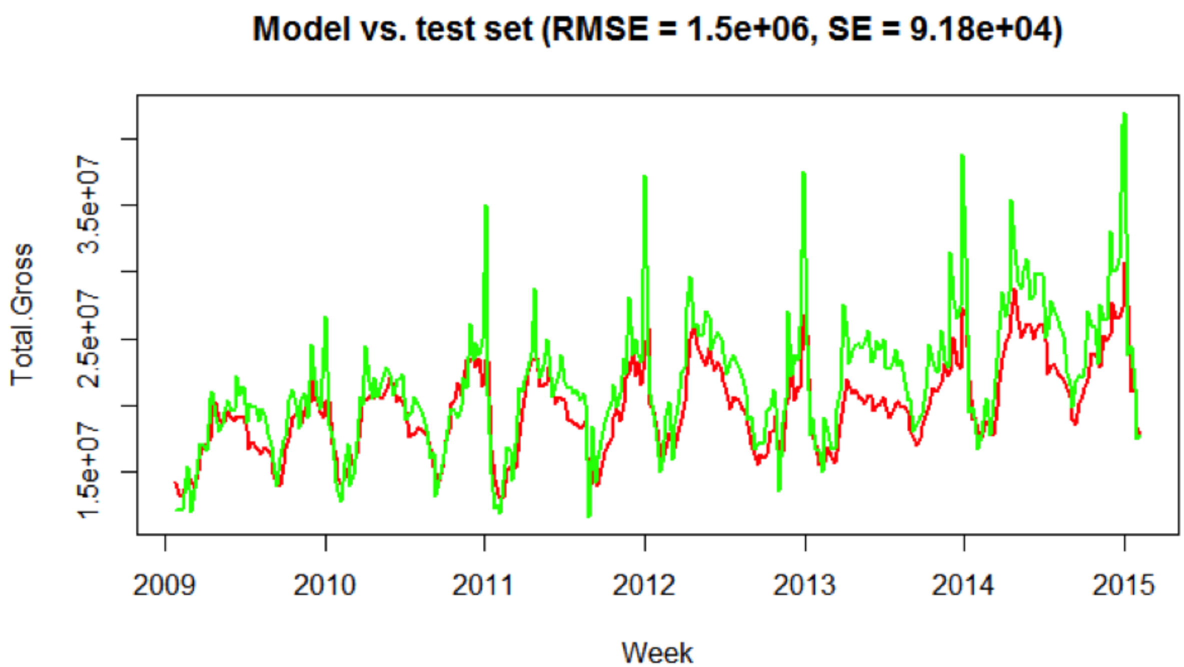

Figure 8: Out-of-sample predictions (in red) vs. actual (in green) for total weekly Broadway gross

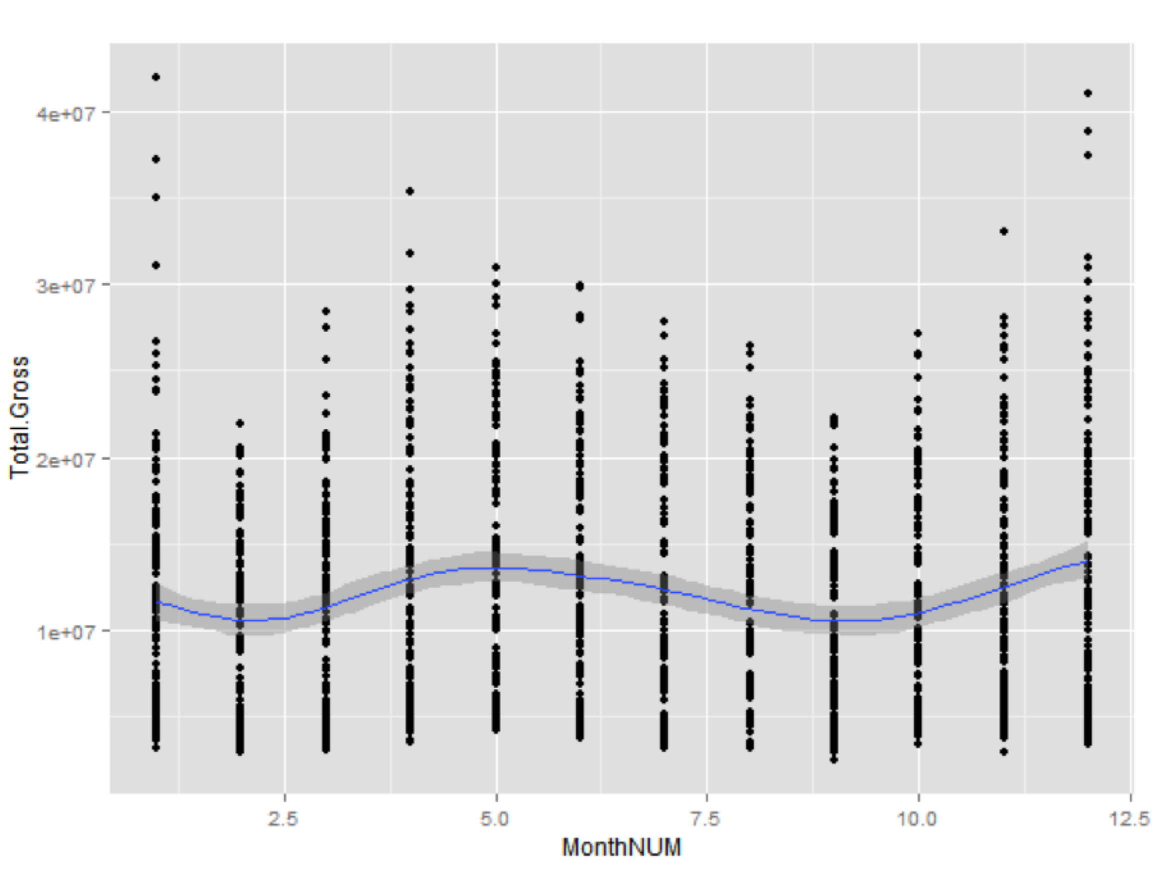

Figure 9: Monthly variability in total weekly gross

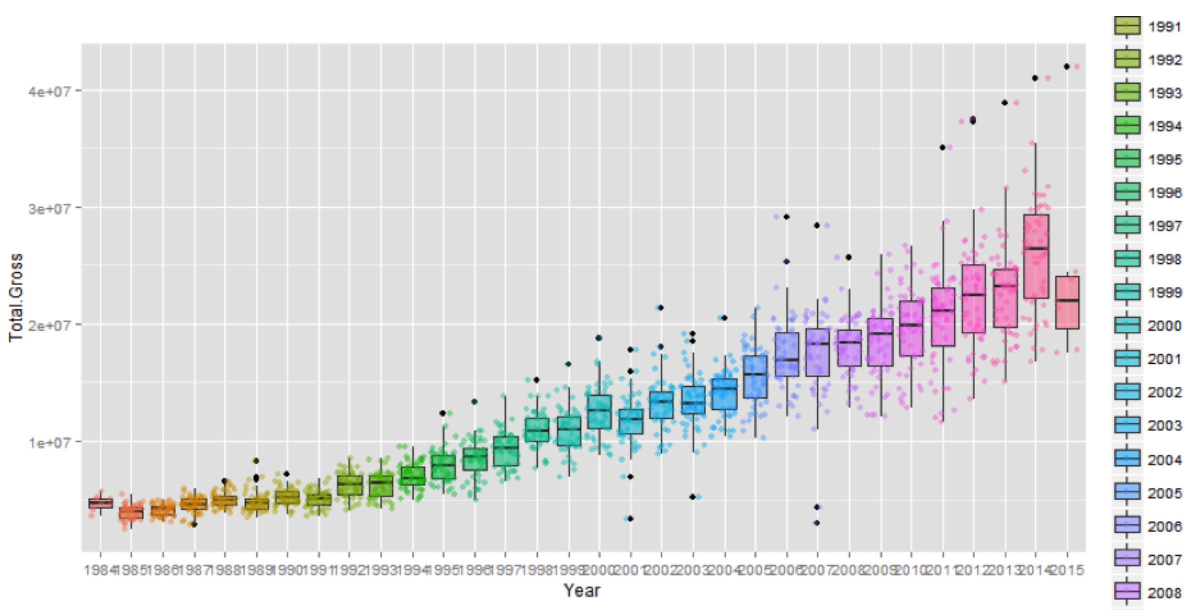

Figure 10: Annual variability in total weekly gross

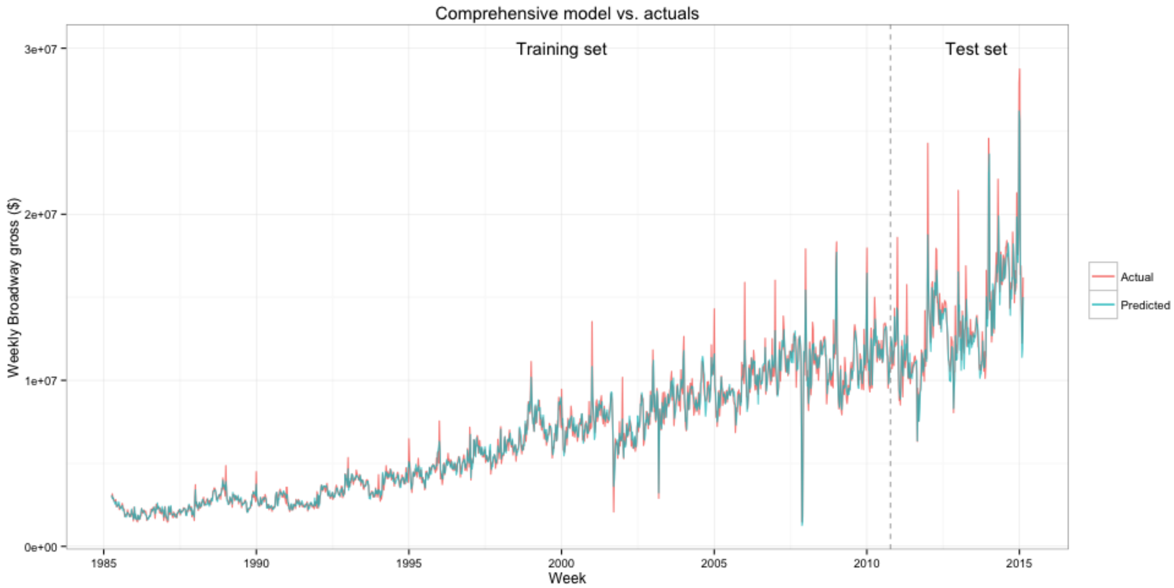

Figure 11: Model predictions vs. actual total weekly Broadway grosses since 1985

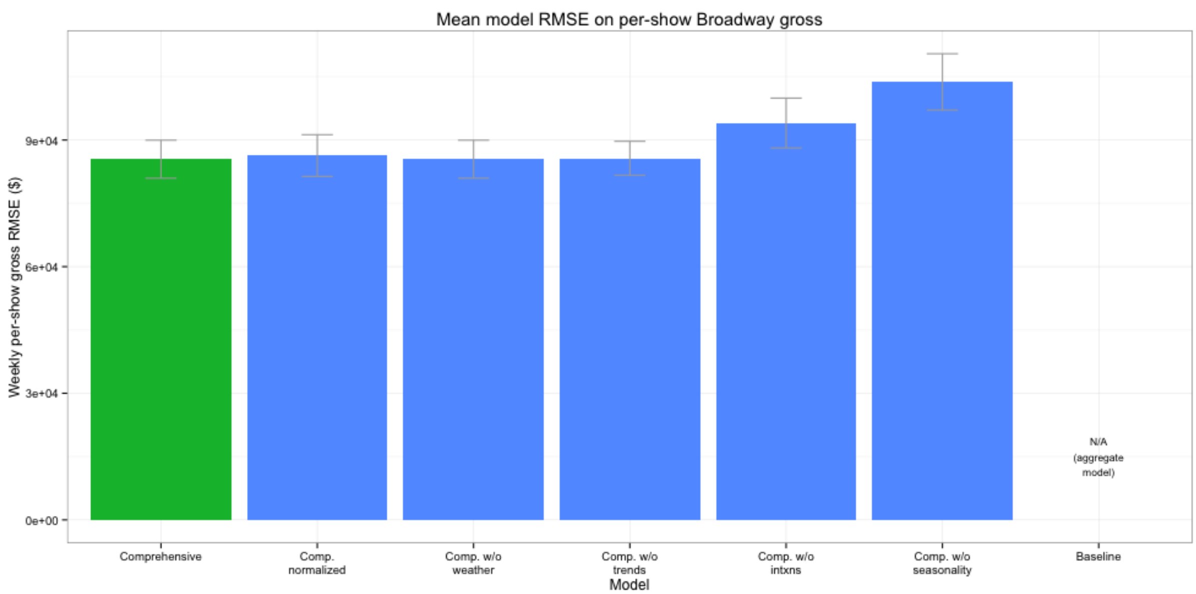

Figure 12: Comparison of RMSE on per-show weekly gross across models

B R Output Logs

```
lm(formula = Total.Gross ~ Week + Total.Gross_lag_1 + Total.Gross_lag_2 +
        Total.Gross_lag_3 + Total.Seats + Total.Performances + Num.Shows +
        Num.Musicals/Num.Shows, data = train)

Residuals:
      Min        1Q    Median        3Q       Max
-10237823   -619675     15472    582704  12580825

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)
(Intercept)            -5.162e+06  6.428e+05  -8.031 1.88e-15 ***
Week                    9.799e+02  5.048e+01  19.413  < 2e-16 ***
Total.Gross_lag_1       4.729e-01  2.318e-02  20.403  < 2e-16 ***
Total.Gross_lag_2      -5.203e-02  2.593e-02  -2.006   0.0450 *
Total.Gross_lag_3       4.951e-02  2.124e-02   2.331   0.0199 *
Total.Seats            -6.011e+00  8.585e-01  -7.002 3.73e-12 ***
Total.Performances      1.111e+05  7.616e+03  14.583  < 2e-16 ***
Num.Shows              -8.007e+05  6.295e+04 -12.720  < 2e-16 ***
Num.Musicals           -3.421e+05  4.321e+04  -7.915 4.62e-15 ***
Num.Shows:Num.Musicals  1.324e+04  1.467e+03   9.028  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1541000 on 1571 degrees of freedom
Multiple R-squared:  0.9501,     Adjusted R-squared:  0.9498
F-statistic:  3321 on 9 and 1571 DF,  p-value: < 2.2e-16
```

Figure 13: R `lm` command and output for baseline model

```
Call:
lm(formula = Total.Gross ~ Week, data = train)

Residuals:
      Min        1Q    Median        3Q       Max
-14884401  -1496808    -84878   1383208  18925054

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -9.869e+06  2.535e+05  -38.93   <2e-16 ***
Week         2.001e+03  2.208e+01   90.62   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Figure 14: R `lm` command output

```
Call:
lm(formula = Total.Gross ~ Week + Total.Performances + Num.Shows +
    Week.of.Year, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-7848316 -1196520   -59098   936486 15384268

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)        -1.440e+07  2.513e+05  -57.32  < 2e-16 ***
Week                1.591e+03  2.148e+01   74.04  < 2e-16 ***
Total.Performances  1.639e+05  9.929e+03   16.51  < 2e-16 ***
Num.Shows          -9.366e+05  7.886e+04  -11.88  < 2e-16 ***
Week.of.Year        1.178e+04  3.476e+03    3.39 0.000716 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 15: R `lm` command output

```
Call:
lm(formula = Total.Gross ~ Week + Total.Performances + Num.Shows +
    Week.of.Year + Num.Musicals, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-8907564 -1150691   -79437   941184 15288221

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)        -1.503e+07  2.850e+05 -52.734  < 2e-16 ***
Week                1.709e+03  3.371e+01  50.709  < 2e-16 ***
Total.Performances  1.668e+05  9.887e+03  16.869  < 2e-16 ***
Num.Shows          -9.095e+05  7.859e+04 -11.573  < 2e-16 ***
Week.of.Year        1.347e+04  3.474e+03   3.877  0.00011 ***
Num.Musicals       -1.285e+05  2.824e+04  -4.550 5.79e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 16: R `lm` command output

```
Call:
lm(formula = Total.Gross ~ Week + Total.Performances + Num.Shows +
    Week.of.Year + Num.Musicals + I(Num.Musicals/Num.Shows),
    data = train)

Residuals:
      Min        1Q    Median        3Q       Max
-11577961  -1033298    -58335    800989  15049012

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)               -2.514e+06  1.227e+06  -2.048  0.04075 *
Week                       1.733e+03  3.266e+01  53.045  < 2e-16 ***
Total.Performances         1.536e+05  9.641e+03  15.931  < 2e-16 ***
Num.Shows                 -1.356e+06  8.713e+04 -15.560  < 2e-16 ***
Week.of.Year               1.025e+04  3.373e+03   3.039  0.00241 **
Num.Musicals               7.560e+05  8.884e+04   8.510  < 2e-16 ***
I(Num.Musicals/Num.Shows) -2.055e+07  1.964e+06 -10.462  < 2e-16 ***
```

Figure 17: R `lm` command output

```
Call:
lm(formula = Total.Gross ~ Week + Total.Performances + Num.Shows +
    Week.of.Year + Num.Musicals + I(Num.Musicals/Num.Shows) +
    Total.Seats, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-5643189 -1106735  -143681   925065 13846532

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)               -1.697e+07  2.878e+06  -5.897 5.14e-09 ***
Week                       2.153e+03  4.543e+01  47.395  < 2e-16 ***
Total.Performances         5.495e+04  1.689e+04   3.253 0.001181 **
Num.Shows                 -9.009e+05  1.353e+05  -6.658 4.71e-11 ***
Week.of.Year               4.971e+03  4.638e+03   1.072 0.284139
Num.Musicals               5.691e+05  1.598e+05   3.562 0.000386 ***
I(Num.Musicals/Num.Shows) -1.496e+07  4.325e+06  -3.460 0.000565 ***
Total.Seats                6.359e+01  1.014e+01   6.274 5.35e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Figure 18: R `lm` command output

```
Call:
lm(formula = Total.Gross ~ Week + Total.Performances + Num.Shows +
    Week.of.Year + Num.Musicals + I(Num.Musicals/Num.Shows) +
    Total.Seats + I(Total.Seats^2) + I(Total.Seats^3), data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-5366299 -1128369  -127997   974197 14044775

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)               -2.349e+07  3.556e+06  -6.605 6.64e-11 ***
Week                       2.167e+03  4.519e+01  47.957  < 2e-16 ***
Total.Performances         5.315e+04  1.677e+04   3.169  0.00158 **
Num.Shows                 -1.198e+06  1.504e+05  -7.965 4.76e-15 ***
Week.of.Year               8.270e+03  4.659e+03   1.775  0.07623 .
Num.Musicals               9.831e+05  1.847e+05   5.323 1.28e-07 ***
I(Num.Musicals/Num.Shows) -2.695e+07  5.097e+06  -5.287 1.54e-07 ***
Total.Seats                2.636e+02  4.830e+01   5.458 6.14e-08 ***
I(Total.Seats^2)          -8.444e-04  1.949e-04  -4.332 1.63e-05 ***
I(Total.Seats^3)           1.152e-09  2.642e-10   4.362 1.43e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Figure 19: R `lm` command output

```
Call:
lm(formula = Total.Gross ~ Week + Total.Performances + Num.Shows +
    Week.of.Year + Num.Musicals + I(Num.Musicals/Num.Shows) +
    Total.Seats + I(Total.Seats^2) + I(Total.Seats^3) + Total.Gross_lag_1 +
    Total.Gross_lag_2 + Total.Gross_lag_3, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-6200144 -1011642   -62370   850534 12803068

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)               -1.638e+07  3.135e+06  -5.225 2.15e-07 ***
Week                       1.273e+03  7.090e+01  17.950  < 2e-16 ***
Total.Performances         5.724e+04  1.465e+04   3.906 0.000100 ***
Num.Shows                 -1.045e+06  1.317e+05  -7.936 5.91e-15 ***
Week.of.Year               1.407e+04  4.135e+03   3.404 0.000693 ***
Num.Musicals               7.409e+05  1.620e+05   4.573 5.45e-06 ***
I(Num.Musicals/Num.Shows) -2.099e+07  4.468e+06  -4.696 3.04e-06 ***
Total.Seats                2.260e+02  4.234e+01   5.337 1.18e-07 ***
I(Total.Seats^2)          -7.669e-04  1.706e-04  -4.495 7.82e-06 ***
I(Total.Seats^3)           1.039e-09  2.311e-10   4.498 7.73e-06 ***
Total.Gross_lag_1          4.141e-01  2.908e-02  14.237  < 2e-16 ***
Total.Gross_lag_2         -5.696e-02  3.176e-02  -1.794 0.073188 .
Total.Gross_lag_3          5.105e-02  2.644e-02   1.931 0.053794 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 20: R `lm` command output

```
Call:
lm(formula = Total.Gross ~ Week + Total.Performances + Num.Shows +
    Week.of.Year + Num.Musicals + I(Num.Musicals/Num.Shows) +
    Total.Seats + I(Total.Seats^2) + I(Total.Seats^3) + Total.Gross_lag_1 +
    Total.Gross_lag_2 + Total.Gross_lag_3 + Avg.Ticket_lag_1,
    data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-5217981  -977084     9913   763883 11588536

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)               -1.360e+07  3.031e+06  -4.488 8.09e-06 ***
Week                       8.279e+02  8.494e+01   9.747  < 2e-16 ***
Total.Performances         4.714e+04  1.414e+04   3.333 0.000891 ***
Num.Shows                 -1.042e+06  1.267e+05  -8.228 6.34e-16 ***
Week.of.Year               1.309e+04  3.979e+03   3.289 0.001042 **
Num.Musicals               8.837e+05  1.567e+05   5.641 2.24e-08 ***
I(Num.Musicals/Num.Shows) -2.450e+07  4.316e+06  -5.676 1.84e-08 ***
Total.Seats                2.190e+02  4.072e+01   5.378 9.52e-08 ***
I(Total.Seats^2)          -7.189e-04  1.642e-04  -4.379 1.33e-05 ***
I(Total.Seats^3)           1.037e-09  2.222e-10   4.665 3.54e-06 ***
Total.Gross_lag_1          4.495e-02  5.048e-02   0.890 0.373465
Total.Gross_lag_2         -3.021e-02  3.070e-02  -0.984 0.325235
Total.Gross_lag_3          6.192e-02  2.546e-02   2.433 0.015180 *
Avg.Ticket_lag_1           1.455e+05  1.656e+04   8.784  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 21: R `lm` command output

```
Call:
lm(formula = Total.Gross ~ Week + Total.Performances + Num.Shows +
    Week.of.Year + Num.Musicals + I(Num.Musicals/Num.Shows) +
    Total.Seats + I(Total.Seats^2) + I(Total.Seats^3) + Avg.Ticket_lag_1 +
    Avg.Ticket_lag_2 + Avg.Ticket_lag_3 + Median.Weeks.Since.Open,
    data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-4962737  -970840     7054   813978 11366602

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)              -1.116e+07  3.131e+06  -3.566 0.000381 ***
Week                      7.984e+02  8.980e+01   8.892  < 2e-16 ***
Total.Performances        4.151e+04  1.408e+04   2.948 0.003276 **
Num.Shows                -1.040e+06  1.263e+05  -8.228 6.34e-16 ***
Week.of.Year              9.329e+03  3.987e+03   2.340 0.019505 *
Num.Musicals              9.599e+05  1.556e+05   6.171 1.01e-09 ***
I(Num.Musicals/Num.Shows) -2.757e+07  4.346e+06  -6.344 3.48e-10 ***
Total.Seats               1.937e+02  4.127e+01   4.694 3.08e-06 ***
I(Total.Seats^2)         -6.003e-04  1.669e-04  -3.596 0.000340 ***
I(Total.Seats^3)          8.894e-10  2.259e-10   3.937 8.86e-05 ***
Avg.Ticket_lag_1          1.642e+05  1.000e+04  16.409  < 2e-16 ***
Avg.Ticket_lag_2         -2.040e+04  1.162e+04  -1.755 0.079537 .
Avg.Ticket_lag_3          2.821e+04  1.012e+04   2.789 0.005388 **
Median.Weeks.Since.Open   1.298e+04  4.717e+03   2.751 0.006059 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 22: R `lm` command output

```
Call:
lm(formula = Total.Gross ~ Week + Total.Performances + Num.Shows +
    Week.of.Year + Num.Musicals + I(Num.Musicals/Num.Shows) +
    Total.Seats + I(Total.Seats^2) + I(Total.Seats^3) + Avg.Ticket_lag_1 +
    Avg.Ticket_lag_2 + Avg.Ticket_lag_3 + Mean.Weeks.Since.Open,
    data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-4937150  -983322   -25087   808712 11457324

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)              -1.103e+07  3.370e+06  -3.275 0.001097 **
Week                      7.589e+02  9.548e+01   7.948 5.41e-15 ***
Total.Performances        4.541e+04  1.415e+04   3.208 0.001380 **
Num.Shows                -1.044e+06  1.268e+05  -8.238 5.83e-16 ***
Week.of.Year              1.152e+04  3.917e+03   2.942 0.003347 **
Num.Musicals              9.382e+05  1.558e+05   6.021 2.49e-09 ***
I(Num.Musicals/Num.Shows) -2.617e+07  4.321e+06  -6.056 2.02e-09 ***
Total.Seats               1.899e+02  4.470e+01   4.248 2.37e-05 ***
I(Total.Seats^2)         -6.012e-04  1.773e-04  -3.391 0.000726 ***
I(Total.Seats^3)          8.996e-10  2.370e-10   3.796 0.000157 ***
Avg.Ticket_lag_1          1.630e+05  1.008e+04  16.167  < 2e-16 ***
Avg.Ticket_lag_2         -2.192e+04  1.166e+04  -1.880 0.060351 .
Avg.Ticket_lag_3          2.525e+04  1.024e+04   2.466 0.013822 *
Mean.Weeks.Since.Open     5.680e+03  4.165e+03   1.364 0.172970
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 23: R `lm` command output

```
Call:
lm(formula = Total.Gross ~ Week + Total.Performances + Num.Shows +
    Week.of.Year + Num.Musicals + I(Num.Musicals/Num.Shows) +
    Total.Seats + I(Total.Seats^2) + I(Total.Seats^3) + Avg.Ticket_lag_1 +
    Avg.Ticket_lag_2 + Avg.Ticket_lag_3 + Median.Weeks.Since.Open +
    I(Total.Performances^2), data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-4818941  -934929   -27970   805614 11305373

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)               -5.137e+06  3.621e+06  -1.419  0.15637
Week                       7.684e+02  8.981e+01   8.556  < 2e-16 ***
Total.Performances        -1.511e+05  6.067e+04  -2.491  0.01292 *
Num.Shows                 -1.130e+06  1.287e+05  -8.777  < 2e-16 ***
Week.of.Year               8.014e+03  3.987e+03   2.010  0.04471 *
Num.Musicals               1.113e+06  1.618e+05   6.883 1.07e-11 ***
I(Num.Musicals/Num.Shows) -3.182e+07  4.516e+06  -7.047 3.54e-12 ***
Total.Seats                3.055e+02  5.347e+01   5.713 1.49e-08 ***
I(Total.Seats^2)          -6.963e-04  1.687e-04  -4.128 3.98e-05 ***
I(Total.Seats^3)           6.278e-10  2.386e-10   2.631  0.00865 **
Avg.Ticket_lag_1           1.641e+05  9.954e+03  16.487  < 2e-16 ***
Avg.Ticket_lag_2          -2.028e+04  1.156e+04  -1.754  0.07974 .
Avg.Ticket_lag_3           3.045e+04  1.009e+04   3.019  0.00261 **
Median.Weeks.Since.Open    1.105e+04  4.730e+03   2.336  0.01973 *
I(Total.Performances^2)    4.373e+02  1.340e+02   3.263  0.00114 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Figure 24: R `lm` command output

```
Call:
lm(formula = Total.Gross ~ Week + Total.Performances + Num.Shows +
    Week.of.Year + Num.Musicals + I(Num.Musicals/Num.Shows) +
    Total.Seats + I(Total.Seats^2) + I(Total.Seats^3) + Avg.Ticket_lag_1 +
    Avg.Ticket_lag_2 + Avg.Ticket_lag_3 + Median.Weeks.Since.Open +
    I(Total.Performances^3), data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-4849108  -936809   -26340   807118 11333756

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)               -6.804e+06  3.424e+06  -1.987  0.04724 *
Week                       7.695e+02  8.989e+01   8.560  < 2e-16 ***
Total.Performances        -5.128e+04  3.327e+04  -1.541  0.12365
Num.Shows                 -1.108e+06  1.278e+05  -8.675  < 2e-16 ***
Week.of.Year               8.004e+03  3.993e+03   2.005  0.04527 *
Num.Musicals               1.088e+06  1.603e+05   6.783 2.08e-11 ***
I(Num.Musicals/Num.Shows) -3.109e+07  4.475e+06  -6.947 6.98e-12 ***
Total.Seats                2.388e+02  4.362e+01   5.474 5.66e-08 ***
I(Total.Seats^2)          -4.671e-04  1.717e-04  -2.719  0.00666 **
I(Total.Seats^3)           3.711e-10  2.810e-10   1.321  0.18699
Avg.Ticket_lag_1           1.641e+05  9.960e+03  16.478  < 2e-16 ***
Avg.Ticket_lag_2          -2.041e+04  1.157e+04  -1.764  0.07802 .
Avg.Ticket_lag_3           3.032e+04  1.009e+04   3.004  0.00274 **
Median.Weeks.Since.Open    1.120e+04  4.732e+03   2.367  0.01812 *
I(Total.Performances^3)    6.205e-01  2.018e-01   3.075  0.00217 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Figure 25: R `lm` command output

```
lm(formula = Total.Gross ~ Week + Total.Gross_lag_1 + Total.Gross_lag_2 +
    Total.Gross_lag_3 + Total.Seats + Total.Performances + Num.Shows +
    Num.Musicals/Num.Shows + Seasonspring + Seasonsummer + Seasonfall +
    MonthDec + MonthMar + MonthJan + MonthMay + MonthAug + MonthFeb +
    MonthJul + MonthJun + MonthNov + MonthOct + MonthSep + Week.of.Year +
    YearNum + NYSEHolidayFlags + ThanksgivFlags + XmasFlags,
    data = train)

Residuals:
     Min        1Q    Median        3Q       Max
-10892841   -621113    17548    578096  11674022

Coefficients: (3 not defined because of singularities)
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)           -1.461e+08  2.055e+07  -7.112 1.74e-12 ***
Week                   3.041e+04  4.289e+03   7.091 2.02e-12 ***
Total.Gross_lag_1      4.615e-01  2.170e-02  21.272  < 2e-16 ***
Total.Gross_lag_2     -5.555e-02  2.387e-02  -2.328 0.020055 *
Total.Gross_lag_3      6.956e-02  2.087e-02   3.332 0.000881 ***
Total.Seats           -6.380e+00  8.334e-01  -7.656 3.35e-14 ***
Total.Performances     1.042e+05  7.048e+03  14.778  < 2e-16 ***
Num.Shows             -7.440e+05  5.882e+04 -12.650  < 2e-16 ***
Num.Musicals          -3.634e+05  4.059e+04  -8.954  < 2e-16 ***
Seasonspring          -9.198e+05  3.270e+05  -2.813 0.004970 **
Seasonsummer          -1.888e+06  5.556e+05  -3.398 0.000697 ***
Seasonfall            -3.947e+06  9.299e+05  -4.245 2.32e-05 ***
MonthDec              -5.242e+06  1.295e+06  -4.047 5.45e-05 ***
MonthMar               8.015e+05  2.266e+05   3.537 0.000417 ***
MonthJan               2.167e+05  2.292e+05   0.945 0.344563
MonthMay              -1.063e+06  2.180e+05  -4.876 1.19e-06 ***
MonthAug              -6.563e+05  3.218e+05  -2.040 0.041540 *
MonthFeb                      NA         NA      NA       NA
MonthJul              -4.190e+05  2.234e+05  -1.875 0.060938 .
MonthJun                      NA         NA      NA       NA
MonthNov              -9.408e+05  3.210e+05  -2.931 0.003431 **
MonthOct               2.014e+05  2.229e+05   0.904 0.366226
MonthSep                      NA         NA      NA       NA
Week.of.Year          -7.222e+04  7.916e+03  -9.123  < 2e-16 ***
YearNum               -1.075e+07  1.568e+06  -6.857 1.01e-11 ***
NYSEHolidayFlags       7.113e+05  1.022e+05   6.961 4.96e-12 ***
ThanksgivFlags        -1.119e+05  2.574e+05  -0.435 0.663833
XmasFlags             -1.433e+05  2.793e+05  -0.513 0.607856
Num.Shows:Num.Musicals 1.379e+04  1.370e+03  10.068  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1401000 on 1555 degrees of freedom
Multiple R-squared:  0.9591,    Adjusted R-squared:  0.9585
F-statistic:  1460 on 25 and 1555 DF,  p-value: < 2.2e-16
```

Figure 26: R `lm` command and output for seasonality model

Table 1: Terms and coefficients selected by LASSO in comprehensive model

Term	Value / coefficient	Type	Interaction
(Intercept)	-1.06E+05	Intercept	
Show.nameA FUNN…AY TO THE FORUM	-8.80E+03	Show name	
Show.nameAMERICAN IDIOT	4.16E+05	Show name	
Show.nameBEAUTY AND THE BEAST	7.49E+03	Show name	
Show.nameCATS	6.30E+02	Show name	
Show.nameCONTACT	-5.40E+03	Show name	
Show.nameCOPENHAGEN	5.07E+04	Show name	
Show.nameDEFENDING THE CAVEMAN	5.30E+04	Show name	
Show.nameFELA!	4.09E+02	Show name	
Show.nameGORE V…'S THE BEST MAN	4.54E+04	Show name	
Show.nameJACKIE…CALLY INCORRECT	1.34E+04	Show name	
Show.nameLES MIS\xc4RABLES	1.29E+04	Show name	
Show.nameMaster Class 1995	1.45E+04	Show name	
Show.nameNEWSIES	6.04E+03	Show name	
Show.namePROOF	-2.63E+03	Show name	
Show.nameSHOW BOAT	8.56E+02	Show name	
Show.nameSUNSET BOULEVARD	1.52E+03	Show name	
Show.nameSWEET CHARITY	7.62E+03	Show name	
Show.nameTHE BOOK OF MORMON	5.30E+04	Show name	
Show.nameTHE HEIRESS	3.77E+04	Show name	
Show.nameTHE PH…OM OF THE OPERA	1.40E+04	Show name	
Show.nameTHE REAL THING	6.11E+02	Show name	
Show.nameTHE WO…ACCORDING TO ME	4.92E+04	Show name	
Show.nameTITANIC	-1.83E+03	Show name	
Show.nameWICKED	3.73E+04	Show name	
Potential.Gross	1.83E-02	Capacity	
Top.Ticket	5.99E+01	Capacity	
Per	5.89E+03	Capacity	
This.Week.s.Gross_lag_9	-1.92E-02	Gross lag	
This.Week.s.Gross_lag_12	3.06E-02	Gross lag	
This.Week.s.Gross_lag_13	2.44E-02	Gross lag	
This.Week.s.Gross_lag_17	3.69E-02	Gross lag	
This.Week.s.Gross_lag_20	2.69E-02	Gross lag	
Is.PlayTRUE	-1.84E+02	Category	
Num.Musicals	3.18E+02	Category	
name.1.mo.1.yr	-4.65E+01	Google Trends	
tickets.6.percent.of.tot	1.74E+04	Google Trends	
tickets.7.percent.of.tot	8.12E+03	Google Trends	
MonthAug	9.28E+03	Seasonality	
MonthJan	1.18E+04	Seasonality	
MonthJul	-9.53E+03	Seasonality	
MonthJun	-1.09E+04	Seasonality	
MonthMay	-1.14E+04	Seasonality	
MonthOct	6.12E+03	Seasonality	
MonthSep	-4.88E+03	Seasonality	
QuarterNum	-6.66E+02	Seasonality	
NYSEHolidayFlags	6.75E+03	Seasonality	
XmasFlags	8.46E+03	Seasonality	
ThanksgivFlags	-1.30E+04	Seasonality	
Average.Ticket_lag_1	3.59E+02	Average ticket lag	
Average.Ticket_lag_9	-5.32E+02	Average ticket lag	

Average.Ticket_lag_18	-1.42E+00	Average ticket lag	
Average.Ticket_lag_19	-9.02E+01	Average ticket lag	
I(TotalSeats^2)	3.40E-05	Capacity	Yes
I(Per^2)	1.06E+03	Capacity	Yes
Show.nameANYTHI...k.s.Gross_lag_1	-3.88E-02	Show name	Yes
Show.nameBLACK ...k.s.Gross_lag_1	-1.06E-02	Show name	Yes
Show.nameBLOOD ...k.s.Gross_lag_1	-4.09E-02	Show name	Yes
Show.nameBRIGHT...k.s.Gross_lag_1	-2.10E-02	Show name	Yes
Show.nameBROADW...k.s.Gross_lag_1	-3.59E-02	Show name	Yes
Show.nameCATS:T...k.s.Gross_lag_1	1.46E-02	Show name	Yes
Show.nameCOPENH...k.s.Gross_lag_1	1.02E-02	Show name	Yes
Show.nameDIRTY ...k.s.Gross_lag_1	-2.91E-02	Show name	Yes
Show.nameFELA!:...k.s.Gross_lag_1	-7.22E-02	Show name	Yes
Show.nameFIDDLE...k.s.Gross_lag_1	-9.41E-03	Show name	Yes
Show.nameFOSSE:...k.s.Gross_lag_1	-1.37E-02	Show name	Yes
Show.nameGORE V...k.s.Gross_lag_1	2.26E-03	Show name	Yes
Show.nameHAIR:T...k.s.Gross_lag_1	-7.48E-02	Show name	Yes
Show.nameHOW TO...k.s.Gross_lag_1	1.50E-01	Show name	Yes
Show.nameKINKY ...k.s.Gross_lag_1	-5.38E-03	Show name	Yes
Show.nameKISS M...k.s.Gross_lag_1	-1.22E-02	Show name	Yes
Show.nameLEND M...k.s.Gross_lag_1	1.99E-03	Show name	Yes
Show.nameLES MI...k.s.Gross_lag_1	6.42E-02	Show name	Yes
Show.nameLOST I...k.s.Gross_lag_1	-2.49E-02	Show name	Yes
Show.nameMEMPHI...k.s.Gross_lag_1	-1.46E-02	Show name	Yes
Show.nameMILLIO...k.s.Gross_lag_1	-1.02E-01	Show name	Yes
Show.nameNERD:T...k.s.Gross_lag_1	-1.24E-01	Show name	Yes
Show.nameOH CAL...k.s.Gross_lag_1	-4.34E-01	Show name	Yes
Show.nameOKLAHO...k.s.Gross_lag_1	-1.63E-01	Show name	Yes
Show.nameONCE O...k.s.Gross_lag_1	-6.10E-03	Show name	Yes
Show.namePRISCI...k.s.Gross_lag_1	-2.51E-02	Show name	Yes
Show.nameRAGTIM...k.s.Gross_lag_1	-6.11E-03	Show name	Yes
Show.nameRENT:T...k.s.Gross_lag_1	1.19E-02	Show name	Yes
Show.nameSMOKEY...k.s.Gross_lag_1	-9.92E-03	Show name	Yes
Show.nameSPEED ...k.s.Gross_lag_1	1.54E-01	Show name	Yes
Show.nameSPEED-...k.s.Gross_lag_1	1.54E-01	Show name	Yes
Show.nameSTARLI...k.s.Gross_lag_1	-1.84E-03	Show name	Yes
Show.nameSWEENE...k.s.Gross_lag_1	1.81E-01	Show name	Yes
Show.nameSWEET ...k.s.Gross_lag_1	-5.15E-02	Show name	Yes
Show.nameSWING!...k.s.Gross_lag_1	-1.62E-02	Show name	Yes
Show.nameTHE GR...k.s.Gross_lag_1	-4.53E-02	Show name	Yes
Show.nameTHE HE...k.s.Gross_lag_1	-5.08E-02	Show name	Yes
Show.nameTHE HE...k.s.Gross_lag_1	7.67E-02	Show name	Yes
Show.nameTHE LI...k.s.Gross_lag_1	-4.45E-02	Show name	Yes
Show.nameTHE LI...k.s.Gross_lag_1	8.14E-05	Show name	Yes
Show.nameTHE LI...k.s.Gross_lag_1	5.01E-03	Show name	Yes
Show.nameTHE RE...k.s.Gross_lag_1	4.20E-02	Show name	Yes
Show.nameTHE SI...k.s.Gross_lag_1	-6.73E-02	Show name	Yes
Show.nameTHE TA...k.s.Gross_lag_1	-1.27E-02	Show name	Yes
Show.nameTHOROU...k.s.Gross_lag_1	-1.06E-02	Show name	Yes
Show.nameVICTOR...k.s.Gross_lag_1	1.24E-02	Show name	Yes
Potential.Gross...k.s.Gross_lag_1	1.48E-08	Capacity	Yes
Per:This.Week.s.Gross_lag_1	6.12E-02	Capacity	Yes
This.Week.s.Gro..._lag_1:MonthDec	-2.28E-01	Seasonality	Yes

This.Week.s.Gro... _lag_1:MonthFeb	3.56E-02	Seasonality	Yes
This.Week.s.Gro... _lag_1:MonthMar	-1.54E-02	Seasonality	Yes
This.Week.s.Gro... _lag_1:MonthNov	-3.60E-02	Seasonality	Yes
This.Week.s.Gro... _1:Seasonsummer	2.69E-02	Seasonality	Yes
This.Week.s.Gro...lag_1:XmasFlags	4.71E-02	Seasonality	Yes
Show.nameA CHOR...k.s.Gross_lag_2	-1.39E-05	Show name	Yes
Show.nameAN INS...k.s.Gross_lag_2	-1.18E-02	Show name	Yes
Show.nameANGELS...k.s.Gross_lag_2	-1.76E-03	Show name	Yes
Show.nameANNIE:...k.s.Gross_lag_2	2.91E-02	Show name	Yes
Show.nameBURN T...k.s.Gross_lag_2	-3.93E-02	Show name	Yes
Show.nameCABARE...k.s.Gross_lag_2	2.55E-02	Show name	Yes
Show.nameCOASTA...k.s.Gross_lag_2	-1.22E-02	Show name	Yes
Show.nameCURTAI...k.s.Gross_lag_2	-6.16E-02	Show name	Yes
Show.nameDOUBT:...k.s.Gross_lag_2	-9.88E-02	Show name	Yes
Show.nameEVITA:...k.s.Gross_lag_2	-4.42E-02	Show name	Yes
Show.nameGORE V...k.s.Gross_lag_2	8.96E-04	Show name	Yes
Show.nameGYPSY:...k.s.Gross_lag_2	1.52E-01	Show name	Yes
Show.nameGYPSY ...k.s.Gross_lag_2	-3.75E-02	Show name	Yes
Show.nameHOW TO...k.s.Gross_lag_2	2.11E-02	Show name	Yes
Show.nameI AM M...k.s.Gross_lag_2	1.41E-01	Show name	Yes
Show.nameLES MI...k.s.Gross_lag_2	1.06E-02	Show name	Yes
Show.nameNICE W...k.s.Gross_lag_2	-2.68E-02	Show name	Yes
Show.nameOH CAL...k.s.Gross_lag_2	-1.62E-01	Show name	Yes
Show.nameSOCIAL...k.s.Gross_lag_2	-3.32E-02	Show name	Yes
Show.nameTHE KI...k.s.Gross_lag_2	-4.32E-03	Show name	Yes
Show.nameTHE LA...k.s.Gross_lag_2	-2.80E-02	Show name	Yes
Show.nameTHE RE...k.s.Gross_lag_2	8.55E-03	Show name	Yes
Show.nameTHE SO...k.s.Gross_lag_2	-9.06E-03	Show name	Yes
Show.nameTHOROU...k.s.Gross_lag_2	-3.50E-03	Show name	Yes
Show.nameTITANI...k.s.Gross_lag_2	-2.41E-02	Show name	Yes
Top.Ticket:This...k.s.Gross_lag_2	4.51E-06	Capacity	Yes
Per:This.Week.s.Gross_lag_2	5.42E-03	Capacity	Yes
This.Week.s.Gro....percent.of.tot	6.41E-03	Google Trends	Yes
This.Week.s.Gro....percent.of.tot	8.85E-03	Google Trends	Yes
This.Week.s.Gro....percent.of.tot	1.75E-03	Google Trends	Yes
This.Week.s.Gro....percent.of.tot	2.88E-05	Google Trends	Yes
This.Week.s.Gro....percent.of.tot	1.87E-02	Google Trends	Yes
This.Week.s.Gro... _lag_2:MonthDec	9.22E-03	Seasonality	Yes
This.Week.s.Gro... _lag_2:MonthJan	-1.63E-01	Seasonality	Yes
This.Week.s.Gro... _lag_2:MonthMay	-1.16E-02	Seasonality	Yes
This.Week.s.Gro... _2:Seasonspring	-5.44E-03	Seasonality	Yes
This.Week.s.Gro...lag_2:XmasFlags	-6.90E-02	Seasonality	Yes
Show.nameAMERIC...k.s.Gross_lag_3	-1.31E-01	Show name	Yes
Show.nameASPECT...k.s.Gross_lag_3	-1.13E-02	Show name	Yes
Show.nameCOPENH...k.s.Gross_lag_3	2.59E-06	Show name	Yes
Show.nameCRAZY ...k.s.Gross_lag_3	4.20E-04	Show name	Yes
Show.nameDEFEND...k.s.Gross_lag_3	2.19E-02	Show name	Yes
Show.nameEVITA:...k.s.Gross_lag_3	4.25E-02	Show name	Yes
Show.nameFOOTLO...k.s.Gross_lag_3	-3.97E-02	Show name	Yes
Show.nameGORE V...k.s.Gross_lag_3	7.26E-04	Show name	Yes
Show.nameGYPSY ...k.s.Gross_lag_3	-9.25E-03	Show name	Yes
Show.nameHAIR:T...k.s.Gross_lag_3	-5.64E-03	Show name	Yes
Show.nameHAIRSP...k.s.Gross_lag_3	-3.00E-03	Show name	Yes

Show.nameHEDWIG. . . k.s.Gross_lag_3	1.41E-01	Show name	Yes
Show.nameIF/THE. . . k.s.Gross_lag_3	6.11E-02	Show name	Yes
Show.nameIN THE. . . k.s.Gross_lag_3	-1.38E-02	Show name	Yes
Show.nameMAMMA . . . k.s.Gross_lag_3	5.26E-03	Show name	Yes
Show.nameMARY P. . . k.s.Gross_lag_3	-5.66E-03	Show name	Yes
Show.nameMOVIN'. . . k.s.Gross_lag_3	-1.40E-02	Show name	Yes
Show.nameSISTER. . . k.s.Gross_lag_3	-4.17E-03	Show name	Yes
Show.nameSIX DE. . . k.s.Gross_lag_3	-1.72E-02	Show name	Yes
Show.nameSOCIAL. . . k.s.Gross_lag_3	-5.63E-02	Show name	Yes
Show.nameTHE 25. . . k.s.Gross_lag_3	-1.91E-02	Show name	Yes
Show.nameTHE 39. . . k.s.Gross_lag_3	-3.79E-02	Show name	Yes
Show.nameTHE AD. . . k.s.Gross_lag_3	-1.25E-02	Show name	Yes
Show.nameTHE LI. . . k.s.Gross_lag_3	-1.89E-02	Show name	Yes
Show.nameTHE RO. . . k.s.Gross_lag_3	-6.08E-03	Show name	Yes
Show.nameWAR HO. . . k.s.Gross_lag_3	1.88E-04	Show name	Yes
Per:This.Week.s.Gross_lag_3	1.10E-02	Capacity	Yes
Weeks.since.ope. . . k.s.Gross_lag_3	-1.39E-05	Capacity	Yes
This.Week.s.Gro. . . .percent.of.tot	8.18E-04	Google Trends	Yes
This.Week.s.Gro. . . _lag_3:MonthMar	1.65E-01	Seasonality	Yes
This.Week.s.Gro. . . _3:Seasonspring	5.12E-04	Seasonality	Yes
This.Week.s.Gro. . . lag_3:XmasFlags	-4.25E-02	Seasonality	Yes
Show.nameBEAUTI. . . k.s.Gross_lag_4	1.58E-02	Show name	Yes
Show.nameCOPENH. . . k.s.Gross_lag_4	8.75E-05	Show name	Yes
Show.nameEVITA:. . . k.s.Gross_lag_4	1.11E-01	Show name	Yes
Show.nameGORE V. . . k.s.Gross_lag_4	1.48E-03	Show name	Yes
Show.nameGYPSY . . . k.s.Gross_lag_4	-9.80E-03	Show name	Yes
Show.nameHOW TO. . . k.s.Gross_lag_4	-1.55E-01	Show name	Yes
Show.nameJERSEY. . . k.s.Gross_lag_4	5.96E-03	Show name	Yes
Show.nameME AND. . . k.s.Gross_lag_4	6.50E-03	Show name	Yes
Show.nameMISS S. . . k.s.Gross_lag_4	9.75E-03	Show name	Yes
Show.nameRIVERD. . . k.s.Gross_lag_4	9.44E-03	Show name	Yes
Show.nameSPRING. . . k.s.Gross_lag_4	-9.75E-03	Show name	Yes
Show.nameTARZAN. . . k.s.Gross_lag_4	-2.79E-02	Show name	Yes
Show.nameTHE AD. . . k.s.Gross_lag_4	-1.09E-02	Show name	Yes
Show.nameTHE BO. . . k.s.Gross_lag_4	1.38E-01	Show name	Yes
Show.nameWEST S. . . k.s.Gross_lag_4	-4.70E-03	Show name	Yes
Show.nameWONDER. . . k.s.Gross_lag_4	2.68E-02	Show name	Yes
Per:This.Week.s.Gross_lag_4	1.32E-02	Capacity	Yes
Weeks.since.ope. . . k.s.Gross_lag_4	-2.28E-07	Gross lag	Yes
This.Week.s.Gro. . . k.s.Gross_lag_5	-2.62E-08	Gross lag	Yes
This.Week.s.Gro. . . .percent.of.tot	-3.41E-02	Google Trends	Yes
This.Week.s.Gro. . . _lag_4:MonthAug	-8.93E-04	Seasonality	Yes
This.Week.s.Gro. . . _lag_4:MonthDec	1.97E-01	Seasonality	Yes
This.Week.s.Gro. . . _lag_4:MonthJan	-1.86E-01	Seasonality	Yes
This.Week.s.Gro. . . _lag_4:MonthJul	3.15E-04	Seasonality	Yes
This.Week.s.Gro. . . _lag_4:MonthJun	1.49E-02	Seasonality	Yes
This.Week.s.Gro. . . _lag_4:MonthMar	1.15E-01	Seasonality	Yes
This.Week.s.Gro. . . _lag_4:MonthSep	-6.10E-02	Seasonality	Yes
This.Week.s.Gro. . . _4:Seasonspring	3.56E-03	Seasonality	Yes
This.Week.s.Gro. . . :ThanksgivFlags	-6.14E-01	Seasonality	Yes
Show.nameA CHOR. . . k.s.Gross_lag_5	-2.10E-02	Show name	Yes
Show.nameAMERIC. . . k.s.Gross_lag_5	-3.52E-01	Show name	Yes
Show.nameASPECT. . . k.s.Gross_lag_5	-2.69E-02	Show name	Yes

Show.nameAUGUST...k.s.Gross_lag_5	-7.56E-02	Show name	Yes
Show.nameBILOXI...k.s.Gross_lag_5	-4.65E-02	Show name	Yes
Show.nameCAROUS...k.s.Gross_lag_5	1.12E-02	Show name	Yes
Show.nameCOPENH...k.s.Gross_lag_5	4.05E-03	Show name	Yes
Show.nameCRAZY ...k.s.Gross_lag_5	5.39E-04	Show name	Yes
Show.nameFELA!:...k.s.Gross_lag_5	3.35E-02	Show name	Yes
Show.nameFENCES...k.s.Gross_lag_5	-2.55E-02	Show name	Yes
Show.nameFIDDLE...k.s.Gross_lag_5	2.78E-04	Show name	Yes
Show.nameGOD OF...k.s.Gross_lag_5	-1.07E-01	Show name	Yes
Show.nameGORE V...k.s.Gross_lag_5	1.92E-03	Show name	Yes
Show.nameGREASE...k.s.Gross_lag_5	-1.09E-02	Show name	Yes
Show.nameIF/THE...k.s.Gross_lag_5	1.09E-02	Show name	Yes
Show.nameLES MI...k.s.Gross_lag_5	3.39E-02	Show name	Yes
Show.nameMATILD...k.s.Gross_lag_5	-4.30E-03	Show name	Yes
Show.nameMOTOWN...k.s.Gross_lag_5	-9.32E-03	Show name	Yes
Show.nameSEARCH...k.s.Gross_lag_5	1.58E-01	Show name	Yes
Show.nameSPAMAL...k.s.Gross_lag_5	-1.34E-03	Show name	Yes
Show.nameSPIDER...k.s.Gross_lag_5	1.84E-02	Show name	Yes
Show.nameTHE CO...k.s.Gross_lag_5	-1.24E-02	Show name	Yes
Show.nameTHE DR...k.s.Gross_lag_5	-5.56E-02	Show name	Yes
Show.nameTHE TA...k.s.Gross_lag_5	-3.86E-02	Show name	Yes
Potential.Gross...k.s.Gross_lag_5	1.68E-09	Capacity	Yes
Per:This.Week.s.Gross_lag_5	8.24E-03	Capacity	Yes
This.Week.s.Gro....percent.of.tot	-5.13E-02	Google Trends	Yes
This.Week.s.Gro..._lag_5:MonthAug	-2.27E-02	Seasonality	Yes
This.Week.s.Gro..._lag_5:MonthDec	6.46E-03	Seasonality	Yes
This.Week.s.Gro..._lag_5:MonthFeb	-1.68E-01	Seasonality	Yes
This.Week.s.Gro..._lag_5:MonthJan	5.18E-01	Seasonality	Yes
This.Week.s.Gro...YSEHolidayFlags	4.68E-02	Seasonality	Yes
Show.nameA LITT...k.s.Gross_lag_6	-5.00E-02	Show name	Yes
Show.nameANNIE ...k.s.Gross_lag_6	-1.80E-02	Show name	Yes
Show.nameCINDER...k.s.Gross_lag_6	-1.39E-02	Show name	Yes
Show.nameCONVER...k.s.Gross_lag_6	-4.15E-02	Show name	Yes
Show.nameCOPENH...k.s.Gross_lag_6	1.14E-03	Show name	Yes
Show.nameDEFEND...k.s.Gross_lag_6	7.34E-03	Show name	Yes
Show.nameForeve...k.s.Gross_lag_6	-4.26E-03	Show name	Yes
Show.nameGORE V...k.s.Gross_lag_6	1.44E-04	Show name	Yes
Show.nameLITTLE...k.s.Gross_lag_6	-1.72E-03	Show name	Yes
Show.nameMaster...k.s.Gross_lag_6	5.99E-02	Show name	Yes
Show.nameNEXT T...k.s.Gross_lag_6	-1.90E-02	Show name	Yes
Show.nameONCE:T...k.s.Gross_lag_6	-8.70E-03	Show name	Yes
Show.nameRAGTIM...k.s.Gross_lag_6	-9.08E-03	Show name	Yes
Show.nameTAKE M...k.s.Gross_lag_6	-8.72E-02	Show name	Yes
Show.nameTHE AD...k.s.Gross_lag_6	-1.38E-02	Show name	Yes
Show.nameTHE LI...k.s.Gross_lag_6	2.82E-02	Show name	Yes
Show.nameTHE PR...k.s.Gross_lag_6	-6.22E-03	Show name	Yes
Show.nameTHE RO...k.s.Gross_lag_6	-2.39E-02	Show name	Yes
Show.nameTHE SI...k.s.Gross_lag_6	-3.32E-02	Show name	Yes
Show.nameWEST S...k.s.Gross_lag_6	-1.50E-02	Show name	Yes
This.Week.s.Gro...g_6:Is.PlayTRUE	-7.04E-03	Category	Yes
This.Week.s.Gro...lag_6:Num.Shows	-8.62E-04	Capacity	Yes
This.Week.s.Gro..._lag_6:MonthDec	-1.21E-02	Seasonality	Yes
This.Week.s.Gro..._lag_6:MonthFeb	2.00E-02	Seasonality	Yes

This.Week.s.Gro. . . _lag_6:MonthJan	-3.03E-01	Seasonality	Yes
This.Week.s.Gro. . . _lag_6:MonthOct	2.72E-04	Seasonality	Yes
This.Week.s.Gro. . . _6:Seasonsummer	-1.65E-02	Seasonality	Yes
This.Week.s.Gro. . . s_lag_6:YearNum	-3.46E-04	Seasonality	Yes
This.Week.s.Gro. . . lag_6:XmasFlags	-4.96E-01	Seasonality	Yes
Show.nameALADDI. . . k.s.Gross_lag_7	2.83E-02	Show name	Yes
Show.nameAMERIC. . . k.s.Gross_lag_7	-3.56E-03	Show name	Yes
Show.nameANNIE:. . . k.s.Gross_lag_7	-5.05E-02	Show name	Yes
Show.nameCATS:T. . . k.s.Gross_lag_7	1.65E-04	Show name	Yes
Show.nameCOPENH. . . k.s.Gross_lag_7	1.39E-03	Show name	Yes
Show.nameFALSET. . . k.s.Gross_lag_7	-4.13E-02	Show name	Yes
Show.nameForeve. . . k.s.Gross_lag_7	-1.59E-02	Show name	Yes
Show.nameHEDWIG. . . k.s.Gross_lag_7	6.10E-02	Show name	Yes
Show.nameJELLY'. . . k.s.Gross_lag_7	-2.12E-02	Show name	Yes
Show.nameJERSEY. . . k.s.Gross_lag_7	1.03E-02	Show name	Yes
Show.nameLA CAG. . . k.s.Gross_lag_7	-8.77E-02	Show name	Yes
Show.nameLEGALL. . . k.s.Gross_lag_7	-1.36E-02	Show name	Yes
Show.nameLES MI. . . k.s.Gross_lag_7	-7.66E-02	Show name	Yes
Show.nameMAMMA . . . k.s.Gross_lag_7	4.85E-04	Show name	Yes
Show.nameOH CAL. . . k.s.Gross_lag_7	-7.45E-01	Show name	Yes
Show.namePIPPIN. . . k.s.Gross_lag_7	-2.43E-02	Show name	Yes
Show.nameSHREK . . . k.s.Gross_lag_7	-5.94E-02	Show name	Yes
Show.nameSPEED . . . k.s.Gross_lag_7	-1.64E-01	Show name	Yes
Show.nameSPEED-. . . k.s.Gross_lag_7	-1.64E-01	Show name	Yes
Show.nameSPIDER. . . k.s.Gross_lag_7	-1.44E-02	Show name	Yes
Show.nameTHE 39. . . k.s.Gross_lag_7	-1.47E-02	Show name	Yes
Show.nameTHE HE. . . k.s.Gross_lag_7	-5.65E-02	Show name	Yes
Show.nameTHOROU. . . k.s.Gross_lag_7	-2.10E-02	Show name	Yes
Per:This.Week.s.Gross_lag_7	1.10E-02	Capacity	Yes
This.Week.s.Gro. . . .s.Gross_lag_12	3.65E-10	Gross lag	Yes
This.Week.s.Gro. . . .s.Gross_lag_14	1.70E-09	Gross lag	Yes
This.Week.s.Gro. . . .s.Gross_lag_16	7.64E-09	Gross lag	Yes
This.Week.s.Gro. . . 12.week.average	1.55E-02	Google Trends	Yes
This.Week.s.Gro. . . 26.week.average	6.01E-03	Google Trends	Yes
This.Week.s.Gro. . . _lag_7:MonthFeb	4.17E-02	Seasonality	Yes
This.Week.s.Gro. . . _lag_7:MonthMar	-2.56E-01	Seasonality	Yes
This.Week.s.Gro. . . _lag_7:MonthOct	1.42E-02	Seasonality	Yes
This.Week.s.Gro. . . _7:Seasonwinter	4.71E-02	Seasonality	Yes
This.Week.s.Gro. . . lag_7:XmasFlags	3.08E-01	Seasonality	Yes
This.Week.s.Gro. . . :ThanksgivFlags	6.89E-01	Seasonality	Yes
YearNum:XmasFlags	2.39E+03	Seasonality	Yes
This.Week.s.Gro. . . arNum:XmasFlags	1.12E-02	Seasonality	Yes

C R Code

Listing 1: R script for building comprehensive model

```
rm( list=ls() )
set.seed( "20150226" )

sink("finalModel-console.out", append=FALSE, split=FALSE)

### CHANGE LINE BELOW TO YOUR WORKING DIRECTORY ###
setwd( "/Users/mike/Documents/Classes/OIT 367/Project/" )

source( 'http://www.stanford.edu/~bayati/oit367/T367_utilities_10.R' )
source( 'cv_utilities.R' ) # Some useful utilities for CV and data processing

normalize = FALSE # TRUE to normalize variables

# To save computation time, we cache the processed data. If given a filename,
# the script assumes that cached data exists there and reads it. Otherwise, we
# regenerate the processed data (from raw data) and save it to write.path
data.file = "cleaned-avg_trends-seasons-weather-avg_ticket.csv"
write.path = "cleaned-all.csv"
if ( is.null( data.file ) ) {
  all.data = read.data( lags=c(1:10) )
  if ( !is.null( write.path ) ) {
    write.csv( all.data, file=write.path, row.names=FALSE )
  }
} else {
  all.data = read.csv( data.file )
  all.data$Week = as.Date( all.data$Week, "%Y-%m-%d")
}

# Sort the data by increasing time and make sure the show name variable is
# stored as a factor
all.data = all.data[order( all.data$Week ),]
all.data$Show.name = as.factor( all.data$Show.name )

# Placeholder variables
data.to.use = all.data
response = "This.Week.s.Gross"

# The core model building logic
build.model = function( train, test, summarize=FALSE, return.rmse=TRUE ) {

  # First, get rid of any "forbidden" variables (i.e. those that contain future
  # information)
  train$X = NULL
  train$X.1 = NULL
  train$X.2 = NULL
  train$Diff.. = NULL
  train$Gross...of.Potential = NULL
  train$Average.Ticket = NULL
  train$This.Week.. = NULL
  train$Diff...1 = NULL
  train$Category = NULL
  train$Total.Gross = NULL
  train$Last.Week.. = NULL
```

```
train$SeatsSold = NULL
train$Last.Week.s.Gross = NULL
train$Year = NULL
train$MonthNUM = NULL
train$QuarterFull = NULL

test$X = NULL
test$X.1 = NULL
test$X.2 = NULL
test$Diff.. = NULL
test$Gross...of.Potential = NULL
test$Average.Ticket = NULL
test$This.Week.. = NULL
test$Diff...1 = NULL
test$Category = NULL
test$Total.Gross = NULL
test$Last.Week.. = NULL
test$SeatsSold = NULL
test$Last.Week.s.Gross = NULL
test$Year = NULL
test$MonthNUM = NULL
test$QuarterFull = NULL

# Clean and NAs in the data
train = fix.na( train )
test = fix.na( test )


# If we've been told to normalize the variables, replace each one with the
# normalization of itself (otherwise, this loop is effectively a noop)
cols = colnames( train )
types = sapply( train, class )
train.preds = train
for ( i in 1:length( cols ) ) {
  if ( types[i] == "numeric" || types[i] == "integer" && cols[i] != "Week" ) {
    scaled = scale( train[, cols[i]] )
    if ( attr( scaled, "scaled:scale" ) != 0 && normalize ) {
      train[, cols[i]] = scaled
      train.preds[, cols[i]] = scaled

      test[, cols[i]] = ( test[, cols[i]] -
                              attr( scaled, "scaled:center" ) ) /
                              attr( scaled, "scaled:scale" )
    }
  }
}

# Ensure that the training and test show name factors have the same number of
# levels, to avoid cryptic errors in predict
levels = unique( union( train$Show.name, test$Show.name ) )

# Build a model matrix for the training set -- the library buildModel function
# raised errors for some reason, so we have to roll our own matrix
train.response = train.preds$This.Week.s.Gross
train.preds$This.Week.s.Gross = NULL
x = model.matrix( ~ . +
```

```
                          (.)*This.Week.s.Gross_lag_1 +
                          (.)*This.Week.s.Gross_lag_2 +
                          (.)*This.Week.s.Gross_lag_3 +
                          (.)*This.Week.s.Gross_lag_4 +
                          (.)*This.Week.s.Gross_lag_5 +
                          (.)*This.Week.s.Gross_lag_6 +
                          (.)*This.Week.s.Gross_lag_7 +
                          XmasFlags*YearNum*This.Week.s.Gross_lag_1 +
                          I(TotalSeats^2) + I(Per^2) + I(Num.Shows^2), train.preds )

# Build a LASSO model
m = cv.glmnet( x=x, y=train.response, alpha=1, family="gaussian", type="mse" )

# If we've been told to print summary statistics about the model, do that now
if ( summarize ) {
  print( summary( m ) )
  print( coef( m ) )
  plot( m )
}

# Same as above, build a model matrix for the test set
test.preds = test
test.preds$This.Week.s.Gross = NULL
newx = model.matrix( ~ . +
                     (.)*This.Week.s.Gross_lag_1 +
                     (.)*This.Week.s.Gross_lag_2 +
                     (.)*This.Week.s.Gross_lag_3 +
                     (.)*This.Week.s.Gross_lag_4 +
                     (.)*This.Week.s.Gross_lag_5 +
                     (.)*This.Week.s.Gross_lag_6 +
                     (.)*This.Week.s.Gross_lag_7 +
                     XmasFlags*YearNum*This.Week.s.Gross_lag_1 +
                     I(TotalSeats^2) + I(Per^2) + I(Num.Shows^2),
                   test.preds )

# Predict on the test set, being careful to manage data types
preds = predict( m, newx=newx )
preds = as.vector( preds )
preds[is.na( preds )] = median( preds, na.rm=TRUE )

# "Denormalize" the variables, if we've normalized
scale.factor = 1
center.factor = 0

if ( normalize ) {
  scale.factor = attr( train$This.Week.s.Gross, "scaled:scale" )
  center.factor = attr( train$This.Week.s.Gross, "scaled:center" )
}

preds = ( preds * scale.factor ) + center.factor
test$This.Week.s.Gross = ( test$This.Week.s.Gross * scale.factor ) +
                         center.factor

# Calculate per-show RMSE and output it, if we're summarizing
show.rmse = sqrt( mean( ( preds - test$This.Week.s.Gross )^2 ) )
if ( summarize ) {
```

```
    cat( "Show RMSE = ", show.rmse, "\n", sep="" )
  }

  # Build the prediction set and calculate RMSE
  shows.preds = data.frame( Week=test$Week, This.Week.s.Gross=preds )
  preds = aggregate( shows.preds$This.Week.s.Gross, list( shows.preds$Week ),
                     sum )[, 2]
  actuals = aggregate( test$This.Week.s.Gross, list( test$Week ), sum )[, 2]

  rmse = sqrt( mean( ( preds - actuals )^2 ) )

  # Return either the predictions or the RMSE, depending on the call paramters
  if ( return.rmse ) return( c( rmse, show.rmse ) )
  else return( preds )
}

# Run the cross validation on the model and output some useful standardized
# charts about the fit
rmse = cross.validate.time.rmse( data.to.use, build.model,
                                 response=response, k=20, train.len=min( 5000,
                                             floor( nrow( data.to.use ) / 2 ) ) )

show.rmse = rmse$Show.RMSE
rmse = rmse$RMSE

mean.rmse = mean( rmse )
se.rmse = sd( rmse ) / sqrt( length( rmse ) )

mean.show.rmse = mean( show.rmse )
se.show.rmse = sd( show.rmse ) / sqrt( length( show.rmse ) )

train_frac = 0.8
train.data = data.to.use[1:( train_frac * nrow( data.to.use ) ),]
test.data = data.to.use[( train_frac * nrow( data.to.use ) + 1 ):
                          nrow( data.to.use ),]

preds = build.model( train.data, test.data, return.rmse=FALSE )

unscaled = test.data

actuals = aggregate( unscaled$This.Week.s.Gross,
                     list( unscaled$Week ), sum )
weeks = actuals[, 1]
actuals = actuals[, 2]

plot.fit( weeks, NULL, actuals - preds,
          main=paste( "Residuals (RMSE = ",
                      format( mean.rmse, digits=3, scientific=TRUE ), ", SE = ",
                      format( se.rmse, digits=3, scientific=TRUE ), ")",
                      sep="" ), xlab="Week", ylab=response )

plot.fit( weeks, actuals, preds,
          main=paste( "Model vs. test set (RMSE = ",
                      format( mean.rmse, digits=3, scientific=TRUE ), ", SE = ",
                      format( se.rmse, digits=3, scientific=TRUE ), ")",
                      sep="" ), xlab="Week", ylab=response )
```

```
build.model( data.to.use, data.to.use, summarize=TRUE )

cat( "Show RMSE = ", mean.show.rmse, ", SE = ", se.show.rmse, "\n" )
```

Listing 2: Utility R script

```
library( zoo )
library( timeDate )

# Runs a time-series cross validation with the given parameters and model
cross.validate.time = function( data, build.model, response="Gross",
                               k=NULL, train.len=NULL, test.len=NULL,
                               min.train.length=400 ) {
  if ( is.null( k ) ) k = floor( nrow( data ) / min.train.length )
  if ( is.null( train.len ) ) train.len = round( nrow( data ) / k )
  if ( is.null( test.len ) ) test.len = min( round( train.len * 0.5 ),
                                            nrow( data ) - train.len )

  rmse = rep( -1, k )
  show.rmse = rep( -1, k )
  for ( i in 1:k ) {
    train.start = sample( nrow( data ) - ( train.len + test.len ), 1,
                          replace=F )
    test.start = train.start + train.len + 1

    train = data[train.start:( train.start + train.len ),]
    test = data[test.start:( test.start + test.len ),]

    preds = build.model( train, test )

    if ( !is.null( attr( preds, "show.rmse" ) ) ) {
      show.rmse[i] = attr( preds, "show.rmse" )
    }

    rmse[i] = sqrt( mean( ( preds - test[, response] )^2 ) )
  }

  return( list( RMSE=rmse, Show.RMSE=show.rmse ) )
}

# Runs a time-series cross validation with the given parameters and model, but
# assumes build.model returns RMSE instead of predictions
cross.validate.time.rmse = function( data, build.model, response="Gross",
                                    k=NULL, train.len=NULL, test.len=NULL,
                                    min.train.length=400 ) {
  if ( is.null( k ) ) k = floor( nrow( data ) / min.train.length )
  if ( is.null( train.len ) ) train.len = round( nrow( data ) / k )
  if ( is.null( test.len ) ) test.len = min( round( train.len * 0.5 ),
                                            nrow( data ) - train.len )

  rmse = rep( -1, k )
  show.rmse = rep( -1, k )
  for ( i in 1:k ) {
    train.start = sample( nrow( data ) - ( train.len + test.len ), 1,
                          replace=F )
    test.start = train.start + train.len + 1
```

```
    train = data[train.start:( train.start + train.len ),]
    test = data[test.start:( test.start + test.len ),]

    rmses = build.model( train, test )

    rmse[i] = rmses[1]
    show.rmse[i] = rmses[2]

    #rmse[i] = sqrt( mean( ( preds - test[, response] )^2 ) )
  }

  return( list( RMSE=rmse, Show.RMSE=show.rmse ) )
}

# Produces a basic plot of a time-series fit
plot.fit = function( xvals, yvals, preds, residuals=NULL,
                     plot.residuals=FALSE, ... ) {
  plot( range( xvals ), range( c( yvals, preds ) ), type='n', ... )
  lines( xvals, preds, col="red", lwd=2 )
  lines( xvals, yvals, col="green", lwd=2 )
}

# Aggregates raw data into total Broadway gross rows, for aggregate models
total.gross.by.week = function( data, lags=c( 1, 2 ) ) {
  Total.Gross = aggregate( data$This.Week.s.Gross, list( data$Week ), sum )
  Week = Total.Gross$Group.1
  Week.of.Year = as.factor( format( Week + 3, "%U" ) )

  Total.Gross = Total.Gross[,2]

  Total.Seats = aggregate( data$TotalSeats, list( data$Week ), sum )[,2]

  Avg.Top.Ticket = aggregate( data$Top.Ticket, list( data$Week ), mean )[,2]

  Avg.Ticket = Total.Gross / Total.Seats
  Avg.Ticket[Avg.Ticket == Inf] = 0

  Total.Performances = aggregate( data$Per, list( data$Week ), sum )[,2]

  Num.Shows = rep( 0, length( Week ) )
  for ( w in 1:length( Week ) ) {
    Num.Shows[w] = length( unique( data$Show.name[data$Week == Week[w]] ) )
  }

  Num.Musicals = rep( 0, length( Week ) )
  for ( w in 1:length( Week ) ) {
    Num.Musicals[w] = length( unique( data$Show.name[data$Week == Week[w] &
                                          data$Category == "musical"] ) )
  }

  Median.Weeks.Since.Open = aggregate( data$Weeks.since.open, list( data$Week ),
                                       median )[,2]
  Mean.Weeks.Since.Open = aggregate( data$Weeks.since.open, list( data$Week ),
                                     mean )[,2]
  Max.Weeks.Since.Open = aggregate( data$Weeks.since.open, list( data$Week ),
```

STANFORD GRADUATE SCHOOL OF BUSINESS

```
                                        max)$x
    Min.Weeks.Since.Open = aggregate( data$Weeks.since.open, list( data$Week ),
                                        min )[,2]

    data$DJIA = suppressWarnings( as.numeric( as.character( data$DJIA ) ) )
    data$DJIA[is.na( data$DJIA )] = 0

    DJIA = aggregate( data$DJIA, list( data$Week ), mean )[,2]
    PRCP = aggregate( data$PRCP, list( data$Week ), mean )[,2]
    SNWD = aggregate( data$SNWD, list( data$Week ), mean )[,2]
    TMIN = aggregate( data$TMIN, list( data$Week ), min )[,2]
    TMAX = aggregate( data$TMAX, list( data$Week ), max )[,2]

    All.Broadway = data.frame( cbind( Week, Week.of.Year, Total.Gross,
                                        Total.Seats, Total.Performances, Num.Shows,
                                        Avg.Top.Ticket, Median.Weeks.Since.Open,
                                        Mean.Weeks.Since.Open,
                                        Max.Weeks.Since.Open,
                                        Min.Weeks.Since.Open, Num.Musicals,
                                        Avg.Ticket, DJIA, PRCP, SNWD, TMIN, TMAX ) )

    All.Broadway$Week = Week

    All.Broadway = add.lags( All.Broadway, variable="Total.Gross", lags=lags )
    All.Broadway = add.lags( All.Broadway, variable="Avg.Ticket",
                              lags=c( 1, 2, 3 ) )

    return( All.Broadway )
  }

  # Adds lags for the given variable, grouping by group.by (if given)
  #
  # WARNING: Runs really slowly when given a group.by, likely because it's not
  # implemented particularly efficiently
  add.lags = function( data, variable, lags=c( 1 ), group.by=NULL,
                       time.field="Week", remove.NAs=TRUE ) {
    if (!is.null( group.by ) ) {
      data = data[order( data[, group.by], data[, time.field]),]
    }

    for ( l in lags ) {
      col.name = paste( variable, "lag", l, sep="_" )

      if ( is.null( group.by ) ) {
        data[, col.name] = rep( NA, nrow( data ) )
        data[, col.name][( l + 1 ):nrow( data )] =
          data[, variable][1:( nrow( data ) - l )]
      } else {
        data[, col.name] = rep( NA, nrow( data ) )

        cur.group = NA
        for ( i in 1:nrow( data ) ) {
          group = data[i, group.by]
          if ( is.na( cur.group ) || group != cur.group ) {
            cur.group = group
            lag.vals = rep( NA, length( lags ) )
```

```
            start.i = i
          } else if ( i - start.i > l) {
            data[i, col.name] = data[i - l, variable]
          }
        }
      }
    }

    # Clean NA's automatically, if we;ve been told to
    if ( remove.NAs ) {
      if ( is.null( group.by ) ) data = data[( max( lags ) + 1 ):nrow( data ),]
      else {
        clean.rows = rep( TRUE, nrow( data ) )
        for ( i in 1:nrow( data ) ) {
          for ( l in lags ) {
            col.name = paste( variable, "lag", l, sep="_" )
            if ( is.na( data[i, col.name] ) ) clean.rows[i] = FALSE
          }
        }

        data = data[clean.rows,]
      }
    }

    return( data[( max( lags ) + 1 ):nrow( data ),] )
  }

  # A generic "problem solver" function for cleaning data
  fix.probs = function( data, test.func, numeric.func=median,
                        factor.val="Unknown") {
    cleaned = data

    for ( i in 1:ncol( cleaned ) ) {
      type = sapply( cleaned, class )[i][1]

      if ( sum( test.func( cleaned[, i] ) ) == 0 ) next

      if ( type == "character") {
        cleaned[, i] = as.factor( cleaned[, i] )
        type = "factor"
      }

      if ( type == "factor" ){
        levels( cleaned[, i] )[length( levels( cleaned[, i] ) ) + 1] = factor.val
        cleaned[, i][test.func( cleaned[, i] )] = factor.val
      } else if ( type == "integer" | type == "numeric" ) {
        if ( is.numeric( numeric.func ) ) v = numeric.func
        else v = numeric.func( cleaned[, i], na.rm=TRUE )
        cleaned[, i][test.func( cleaned[, i] )] = v
      } else if ( type == "logical" ) {
        cleaned[, i][test.func( cleaned[, i] )] = FALSE
      }
    }

    return( cleaned )
  }
```

```
# Removes NA's in data
fix.na = function( data, numeric.func=median, factor.val="Unknown" ) {
  cleaned = data

  for ( i in 1:ncol( cleaned ) ) {
    type = sapply( cleaned, class )[i][1]

    if ( sum( is.na( cleaned[, i] ) ) == 0 ) next

    if ( type == "character") {
      cleaned[, i] = as.factor( cleaned[, i] )
      type = "factor"
    }

    if ( type == "factor" ){
      levels( cleaned[, i] )[length( levels( cleaned[, i] ) ) + 1] = factor.val
      cleaned[, i][is.na( cleaned[, i] )] = factor.val
    } else if ( type == "integer" | type == "numeric" ) {
      if ( is.numeric( numeric.func ) ) v = numeric.func
      else v = numeric.func( cleaned[, i], na.rm=TRUE )
      cleaned[, i][is.na( cleaned[, i] )] = v
    } else if ( type == "logical" ) {
      cleaned[, i][is.na( cleaned[, i] )] = FALSE
    }
  }

  return( cleaned )
}

# Aggregates and adds Google Trends data to data, as separate columns
add.trends = function( data ) {
  # Files containing semi-raw Trends data (from Python script)
  trends.files = list(
    name=read.csv( "trends-name.csv" ),
    tickets=read.csv( "trends-name-tickets.csv" ),
    broadway=read.csv( "trends-name-broadway.csv" ),
    name.rel=read.csv( "trends-name-relative.csv" ),
    tickets.rel=read.csv( "trends-name-tickets-relative.csv" ),
    broadway.rel=read.csv( "trends-name-broadway-relative.csv" ) )

  # First, clean up the data
  for ( f in names( trends.files ) ) {
    for ( c in 2:ncol( trends.files[[f]] ) ) {
      to.replace = sapply( trends.files[[f]][,c],
                           FUN=function( x ){ return( x == "" || x == "N/A") } )
      trends.files[[f]][,c] = replace( trends.files[[f]][,c], to.replace, 0 )
      trends.files[[f]][,c] = as.numeric( as.character(
                                          trends.files[[f]][,c] ) )
    }
  }

  # Compute the baselines
  baselines = list()
  for ( f in names( trends.files ) ) {
    l = list( one.mos.pre=rep( 1, nrow( as.data.frame(trends.files[f] ) ) ),
```

```
            three.mos.pre=rep( 1, nrow( as.data.frame(trends.files[f] ) ) ),
            six.mos.pre=rep( 1, nrow( as.data.frame(trends.files[f] ) ) ),
            one.yr.pre=rep( 1, nrow( as.data.frame(trends.files[f] ) ) ) )
  baselines[[f]] = 1
}

week.to.col = function( w ) { return( w + 104 + 2 ) }
range.mean = function( begin, end, data ) { }

for ( f in names( baselines ) ) {
  for ( i in 1:nrow( trends.files[[f]] ) ) {
    baselines[[f]]$one.mos.pre[i] = mean( as.numeric( trends.files[[f]][i,
                                    week.to.col(-5):week.to.col(-1)] ) )
    baselines[[f]]$three.mos.pre[i] = mean( as.numeric( trends.files[[f]][i,
                                    week.to.col(-13):week.to.col(-1)] ) )
    baselines[[f]]$six.mos.pre[i] = mean( as.numeric( trends.files[[f]][i,
                                    week.to.col(-25):week.to.col(-1)] ) )
    baselines[[f]]$one.yr.pre[i] = mean( as.numeric( trends.files[[f]][i,
                                    week.to.col(-53):week.to.col(-1)] ) )
  }
}

# Parameters for lag terms, denormalization denominators, and average terms
lags = c( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 )
terms = c( "name", "tickets", "broadway" )
denoms = c( "1-mo", "3-mo", "6-mo", "1-yr" )
averages = c( 2, 4, 6, 12, 26 )

# Create columns for all of these terms (not complexity of f*p*l)
for ( f in terms ) {
  for ( p in denoms ) {
    for ( l in lags ) {
      col.name = paste( f, "-", l, ":", p, sep="" )
      data[, col.name] = rep( 0, nrow( data ) )
    }
  }
}

# Add some additional columns
for ( f in terms ) {
  for ( d in denoms ) {
    e = "1-yr"
    if ( d == e ) next

    col.name = paste( f, "-", d, ":", e, sep="" )
    data[, col.name] = rep( 0, nrow( data ) )
  }
}

# And a couple more columns...
for ( f in c( "tickets", "broadway" ) ) {
  for ( l in lags ) {
    col.name = paste( f, "-", l, "-percent-of-tot", sep="" )
    data[, col.name] = rep( 0, nrow( data ) )
  }
```

```
      for ( a in averages ) {
        col.name = paste( f, "-", a, "-week-average", sep="" )
        data[, col.name] = rep( 0, nrow( data ) )
      }
    }

    # Now fill those columns
    max.week = 623 # Maximum week to search for, to speed things up a little
    for ( i in 1:nrow( data ) ) {
      cur.week = data[i, "Weeks.since.open"]
      show.i = which.max( as.character(trends.files$name.rel[,1]) ==
                          as.character(data[i, "Show.name"]) )

      # Add the fixed pre-open average columns
      for ( f in terms ) {
        for ( d in denoms ) {
          e = "1-yr"
          if ( d == e ) next

          col.name = paste( f, "-", d, ":", e, sep="" )

          denom = baselines[[f]]$one.yr.pre[show.i]
          numer = 0
          if ( d == "1-mo" ) numer = baselines[[f]]$one.mos.pre[show.i]
          if ( d == "3-mo" ) numer = baselines[[f]]$three.mos.pre[show.i]
          if ( d == "6-mo" ) numer = baselines[[f]]$six.mos.pre[show.i]

          v = numer / denom

          if ( !is.null( v ) && !is.na( v ) ) {
            if ( !is.nan( v ) && v != Inf ) {
              data[show.i, col.name] = v
            }
          }
        }
      }

      if ( cur.week > max.week ) next

      # Grab the relative query rows, we'll need them soon
      name.rel.row = trends.files$name.rel[show.i,]
      tickets.rel.row = trends.files$tickets.rel[show.i,]
      broadway.rel.row = trends.files$broadway.rel[show.i,]

      # Add the relatie query rows
      for ( l in lags ) {
        tickets.rel = as.numeric( tickets.rel.row[week.to.col(cur.week - l)] ) /
          as.numeric( name.rel.row[week.to.col(cur.week - l)] )
        broadway.rel = as.numeric( broadway.rel.row[week.to.col(cur.week - l)] ) /
          as.numeric( name.rel.row[week.to.col(cur.week - l)] )

        if ( !is.nan( tickets.rel ) && !is.na( tickets.rel ) &&
                tickets.rel != Inf ) {
          data[i, paste( "tickets-", l, "-percent-of-tot", sep="" )] =
            tickets.rel
        }
```

```
    if ( !is.nan( broadway.rel ) && !is.na( broadway.rel ) &&
           broadway.rel != Inf ) {
      data[i, paste( "broadway-", l, "-percent-of-tot", sep="" )] =
        broadway.rel
    }
  }

  # Now add some averaged relative terms
  for ( a in averages ) {
    tickets.rel = mean( as.numeric(
                        tickets.rel.row[week.to.col(cur.week - a - 1):
                                        week.to.col(cur.week - 1)] ) ) /
                  mean( as.numeric(
                        name.rel.row[week.to.col(cur.week - a - 1):
                                     week.to.col(cur.week - 1)] ) )

    broadway.rel = mean( as.numeric(
                         broadway.rel.row[week.to.col(cur.week - a - 1):
                                          week.to.col(cur.week - 1)] ) ) /
                   mean( as.numeric(
                         name.rel.row[week.to.col(cur.week - a - 1):
                                      week.to.col(cur.week - 1)] ) )

    if ( !is.nan( tickets.rel ) && !is.na( tickets.rel ) &&
           tickets.rel != Inf ) {
      data[i, paste( f, "-", a, "-week-average", sep="" )] =
        tickets.rel
    }

    if ( !is.nan( broadway.rel ) && !is.na( broadway.rel ) &&
           broadway.rel != Inf ) {
      data[i, paste( "broadway-", a, "-week-average", sep="" )] =
        broadway.rel
    }
  }

  # Finally, add the absolute terms, divided by all of the different
  # denominators we're trying
  for ( f in terms ) {
    for ( p in denoms ) {
      denom = NULL

      if ( p == "1-mo" ) denom = baselines[[f]]$one.mos.pre[show.i]
      if ( p == "3-mo" ) denom = baselines[[f]]$three.mos.pre[show.i]
      if ( p == "6-mo" ) denom = baselines[[f]]$six.mos.pre[show.i]
      if ( p == "1-yr" ) denom = baselines[[f]]$one.yr.pre[show.i]

      for ( l in lags ) {
        num = as.numeric( trends.files[[f]][show.i,
                                            week.to.col( cur.week - l )] )

        rel = as.numeric( num / denom )

        if ( !is.null( rel ) && !is.na( rel ) ) {
          if ( !is.nan( rel ) && rel != Inf ) {
```

```
                col.name = paste( f, "-", l, ":", p, sep="" )
                data[show.i, col.name] = rel
              }
            }
          }
        }
      }
    }

    return( data )
  }

# Process the raw data (takes a long time, so we usually cache this...)
read.data = function( lags=c( 1:20 ) ) {
  # Read the raw grosses data and do a little type cleaning
  all.data = read.csv( "grosses.csv" )
  all.data$Week = as.Date( all.data$Week, format="%m/%d/%y" )
  all.data = all.data[order( all.data$Show.name, all.data$Week ),]

  # Add lags of the response variable
  all.data = add.lags( all.data, "This.Week.s.Gross", lags=lags,
                       group.by="Show.name" )

  # Add some summary columns about how many shows are running right now
  Num.Shows = rep( 0, length( all.data$Week ) )
  Num.Musicals = rep( 0, length( all.data$Week ) )
  for ( w in 1:length( all.data$Week ) ) {
    Num.Shows[w] = length( unique( all.data$Show.name[all.data$Week ==
                                                     all.data$Week[w]] ) )
    Num.Musicals[w] = length( unique( all.data$Show.name[
      all.data$Week == all.data$Week[w] & all.data$Category == "musical"] ) )
  }

  # Bin the category variable
  all.data$Is.Play = all.data$Category == "play"
  all.data$Is.Musical = all.data$Category == "musical"

  # Aggregate total grosses
  all.data$Num.Shows = Num.Shows
  all.data$Num.Musicals = Num.Musicals
  all.data$Total.Gross = rep( 0, nrow( all.data ) )
  for ( i in 1:nrow( all.data ) ) {
    all.data$Total.Gross[i] = sum( all.data$This.Week.s.Gross[
      all.data$Week == all.data$Week[i]] )
  }
  all.data = all.data[order( all.data$Week ),]

  # Add trends and seasonality predictors
  all.data = add.trends( all.data )
  all.data = add.seasonality( all.data )

  return( all.data )
}

# Reads and adds seasonality predictors
```

```
add.seasonality = function( data ) {
  data$Month = format(data$Week, "%b" ) # can also use as.yearmon
  data$MonthNUM = as.numeric(format(data$Week, "%m" ))
  data$QuarterFull <- as.yearqtr(as.yearmon(data$Week, "%m/%d/%Y") + 1/12)
  data$QuarterNum <- factor(format(data$QuarterFull, "%q"), levels = 1:4)
  data$Season <- factor(format(data$QuarterFull, "%q"), levels = 1:4,
                        labels = c("winter", "spring", "summer", "fall"))
  data$Year = format(data$Week, "%Y" )
  data$YearNum = as.numeric(format(data$Week, "%Y" ))-1983


  # create column per date tagging whether that week is within 5 days of a
  # public holiday all holidays (G-7 --> likely meaningless)
  allHolidays = as.data.frame(holiday(1984:2014,listHolidays()))
  colnames(allHolidays)[1]="Date"

  n = length(data$Week)
  m = length(allHolidays$Date)

  # create flags for NYSE holidays
  NYSEHolidays = as.data.frame(holidayNYSE(1984:2014))
  colnames(NYSEHolidays)[1]="Date"

  o = length(NYSEHolidays$Date)

  data$NYSEHolidayFlags = 0

  for(j in (1: o)) {
    for (i in (1 : n)) {
      if(    abs(data$Week[i]-as.Date(NYSEHolidays$Date[j])) < 6) {
        data$NYSEHolidayFlags[i] = 1
      }
    }
  }


  # create flags for Christmas weeks
  Xmas = as.data.frame(holiday(1984:2014,"ChristmasDay"))
  colnames(Xmas)[1]="Date"

  p = length(Xmas$Date)

  data$XmasFlags = 0

  for(j in (1: p)) {
    for (i in (1 : n)) {
      if(    abs(data$Week[i]-as.Date(Xmas$Date[j])) < 6) {
        data$XmasFlags[i] = 1
      }
    }
  }


  # create flags for Thanksgiving weeks
  Thanksgiv = as.data.frame(holiday(1984:2014,"USThanksgivingDay"))
  colnames(Thanksgiv)[1]="Date"
```

```
  q = length(Thanksgiv$Date)

  data$ThanksgivFlags = 0

  for(j in (1: q)) {
    for (i in (1 : n)) {
      if(   abs(data$Week[i]-as.Date(Thanksgiv$Date[j])) < 6) {
        data$ThanksgivFlags[i] = 1
      }
    }
  }

  return( data )
}

# Add weather data
add.weather = function( data, file="newGrosses.csv" ) {
  weather = read.csv( file )

  weather$Week = as.Date( weather$Week, "%m/%d/%y" )

  weather$DJIA = suppressWarnings( as.numeric( as.character( weather$DJIA ) ) )
  weather$DJIA[is.na( weather$DJIA )] = 0

  DJIA = aggregate( weather$DJIA, list( weather$Week ), mean )
  PRCP = aggregate( weather$PRCP, list( weather$Week ), mean )
  SNWD = aggregate( weather$SNWD, list( weather$Week ), mean )
  TMIN = aggregate( weather$TMIN, list( weather$Week ), min )
  TMAX = aggregate( weather$TMAX, list( weather$Week ), max )

  data$DJIA = rep( -1, nrow( data ) )
  data$PRCP = rep( -1, nrow( data ) )
  data$SNWD = rep( -1, nrow( data ) )
  data$TMAX = rep( -1, nrow( data ) )
  data$TMIN = rep( -1, nrow( data ) )

  for ( i in 1:nrow( data ) ) {
    w = data[i, "Week"]
    data[i, "DJIA"] = DJIA[which.min( DJIA[,1] == w ), 2]
    data[i, "PRCP"] = PRCP[which.min( PRCP[,1] == w ), 2]
    data[i, "SNWD"] = SNWD[which.min( SNWD[,1] == w ), 2]
    data[i, "TMAX"] = TMAX[which.min( TMAX[,1] == w ), 2]
    data[i, "TMIN"] = TMIN[which.min( TMIN[,1] == w ), 2]
  }

  return( data )
}
```

D Data Collection Code

Listing 3: Python script for retreiving and processing Broadway World and Google Trends data

```
#!/usr/bin/python
import sys
import re
import os.path
import requests
import StringIO
import csv
import datetime
import time
import string
import xlsxwriter
import unicodedata
from bs4 import BeautifulSoup
from openpyxl import Workbook
from openpyxl import load_workbook
from nltk.corpus import wordnet
from uncapped import uncapped_words
from cookies2 import google_cookies

#### CONSTANTS ####
header_rows = 1 # Number of header rows

suffixes = ["", " tickets", " broadway"] # Trends suffixes
request_timeout = 30 # Seconds to wait before the next request
cache_trends = 1 # 1 to cache trends in local files (to avoid throttling)
trends_begin = datetime.date(2004, 01, 01)
get_trends = 1
min_weeks_for_trends = 0 # To allow excluding shorter shows

grosses_path = "Grosses" # Where to cache grosses
trends_path = "Trends" # Where to cache trends data
###################

def read_grosses_date(s):
    return datetime.datetime.strptime(s, "%m/%d/%Y").date()

# Grabs and parses the grosses file from Broadway World
def download_grosses(path):
    print "Downloading grosses"
    shows = fetch_show_list()

    headers = { "Host" : "www.broadwayworld.com",
                "User-Agent" : "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9;" +\
                               "rv:35.0) Gecko/20100101 Firefox/35.0",
                "Accept" : "text/html,application/xhtml+xml,application/xml;" +\
                           "q=0.9,*/*;q=0.8",
                "Accept-Language" : "en-US,en;q=0.5",
                "Accept-Encoding" : "gzip, deflate",
                "Connection" : "keep-alive" }
    cookies = { "CFID" : "1140731393",
                 "CFTOKEN" : "9bec5cd3391afedc-9E3CFFDA-E588-8F4F-" +\
                             "FDD65315D403E598" }
```

```
    i = 1
    for s in shows.keys():
        sys.stdout.write("\r\033[K")
        sys.stdout.write("\r.Fetching grosses...show " + str(i) + " of " +\
                         str(len(shows)) + ", '" + s + "'")
        sys.stdout.flush()


        show_p = os.path.join(path, "".join(c for c in s if c.isalnum() or
                                              c in string.whitespace) +\
                                      " Grosses.xlsx")

        if (os.path.isfile(show_p)):
            continue

        r = requests.get(shows[s], headers=headers, cookies=cookies)
        cleaned = r.content.replace("</a>", "")
        soup = BeautifulSoup(cleaned)

        wb = xlsxwriter.Workbook(show_p)
        ws = wb.add_worksheet()

        rows = soup.find_all("tr")

        r_i = 0
        for r in rows:
            if (r_i == 0):
                ws.write(r_i, 0, "Show name")
            else:
                ws.write(r_i, 0, s)
            cols = r.find_all("td")

            c_i = 1
            for c in cols:
                ws.write(r_i, c_i, c.text)
                c_i += 1
            r_i += 1


        wb.close()

        time.sleep(1) # To play nice with the BW servers

        i += 1

    sys.stdout.write("\r\033[K")
    sys.stdout.write("\r.Fetching grosses...done! Fetched " + str(len(shows)) +\
                     " grosses to " + str(path) + "\n")


# Grabs a list of shows from the Broadway World index (no caching)
def fetch_show_list():

    show_url_prefix = "http://www.broadwayworld.com/grossesshowexcel.cfm?show="
    show_url_suffix = "&all=on"
```

```
    url_stem = "http://www.broadwayworld.com/grossesbyshow.cfm?letter="
    urls = list((url_stem + c for c in string.ascii_lowercase + "1"))

    shows = {}

    i = 1
    for url in urls:
        sys.stdout.write("\r\033[K")
        sys.stdout.write("\r.Fetching show list...url " + str(i) + " of " +\
                         str(len(urls)) + " (" + url + ")")
        sys.stdout.flush()

        r = requests.get(url)
        page = r.text

        show_url_stem = "http://www.broadwayworld.com/grosses/"

        show_p = re.compile(re.escape("<a href=\"" + show_url_stem) + \
                            "([\w\-]+)" + \
                            re.escape("\">") +
                            r"((?:[^\s\<]\s*)+)" +
                            re.escape("</a></td>"))
        m = show_p.findall(page)

        for s in m:
            shows[s[1]] = show_url_prefix + s[0] + show_url_suffix

        time.sleep(1)

        i += 1
        #if (i > 1):
        #    break


    #print(shows)
    sys.stdout.write("\r\033[K")
    sys.stdout.write("\r.Fetching show list...done\n")
    sys.stdout.flush()
    return shows



# Parses the show name from the filename (assumed to be the show name followed
# by 'Grosses'. Then we attempt to split the show name into tokens by the
# following rules: 1) Capital letters denote a new token and 2) If a token
# contains something on the MLA's list of words not to capitalize in a title and
# (a) the token is not in the NLTK corpus of English words but (b) the token
# without the uncapitalized word (i.e. the stem) is in the corpus, we split that
# token into its stem and the uncapitalized word
#
# Then we rejoin the title with spaces
def parse_show_name(fn, i):
    p = re.compile("(.+)\s*Grosses")
    m = p.search(fn)

    if (m != None and m.group(1) != None):
```

```
            flat_name = m.group(1)
        else:
            print "WARNING: Couldn't find name for show in file '" + fn + \
                  "' defaulting to 'Unknown show " + str(i) + "'"
            return "Unknown show " + str(i)

        # If name already has spaces, then just assume it's the show name, convert
        # it to title case and return it
        if (len(flat_name.split()) < len(flat_name)):
            pass

        tokens = re.findall('[A-Z][^A-Z]*', flat_name)
        words = []

        # Check every token in the title to see if it might have an uncapitalized
        # word lurking on its end
        for t in tokens:
            w = t # Assume the token is a word until proven otherwise
            for u in uncapped_words:
                if (t.endswith(u)): # We've found a potential lurker...
                    stem = t.split(u)[0]

                    # We only split the token if the stem is a valid word and the
                    # original token is not
                    if (wordnet.synsets(stem) and not wordnet.synsets(t)):
                        words.append(stem)
                        w = u
                        break # Just take the first match

            words.append(w) # Reconstrust the list of words

        return " ".join(words)

    def cached_trends_path(terms):
        if (not isinstance(terms, basestring)):
            terms = ",".join(terms)

        fname = terms.replace(r"/", "-")
        fname = fname.replace(r"\\", "-")

        # If we're caching and the file exists, then return true
        return os.path.join(trends_path, fname + "_trends.csv")


    def is_cached(terms):
        # If we're caching and the file exists, then return true
        return cache_trends and os.path.isfile(cached_trends_path(terms))


    # Grab all the Google trends data for a show
    def fetch_google_trends(name):
        # Normalize the name before we go, to strip out weird characters
        name = unicodedata.normalize('NFKD', name).encode('ascii','ignore')
        sys.stdout.write("..Beginning Google trends request for '" + name + "'\n")
        terms = list((name + s for s in suffixes))
        terms.append(terms[:])
```

```
    trends = {}

    for t in terms:
        sys.stdout.write("...Requesting '" + str(t) + "' - ")
        cached = is_cached(t) # Need to check before we get the request

        if (request_timeout > 0 and not cached):
            sys.stdout.write("now sleeping " +\
                             str(request_timeout) + " " +\
                             "seconds before next request...")
            sys.stdout.flush()
            time.sleep(request_timeout)
        elif (cached):
            sys.stdout.write("cached - ")

        report = google_trends_request(t)

        if (not isinstance(t, basestring)):
            q_suff = " relative"
        else:
            q_suff = ""

        data = parse_trends_csv(report, q_suff=q_suff)

        for k in data.keys():
            if (k in trends):
                trends[k] = dict(trends[k].items() + data[k].items())
            else:
                trends[k] = dict(data[k].items())

        sys.stdout.write("done!\n")
        sys.stdout.flush()

    # Fill in gaps in the data with the last seen value.
    # Also keeps track of max value seen so far to do de-normalization, but this
    # feature is unused, since we do something similar in R instead
    max_val = {}
    all_terms = {} # In theory, we already know this, but...
    all_dates = trends.keys()
    all_dates.sort()
    for d in all_dates:
        for t in trends[d].keys():
            all_terms[t] = 1

    last_seen = dict(zip(all_terms.keys(), [0 for t in all_terms.keys()]))

    for d in all_dates:
        for t in trends[d].keys():
            last_seen[t] = trends[d][t]

            if (not t in max_val or max_val[t] < trends[d][t]):
                max_val[t] = trends[d][t]

        # Fill in the gaps with the last-seen value
        for t in all_terms.keys():
```

```
            if (not t in trends[d]):
                trends[d][t] = last_seen[t]



    return trends

# Issue a Trends query and cache it (or return the cached value if it's already
# cached)
def google_trends_request(terms):
    url = "http://www.google.com/trends/trendsReport"

    if (not isinstance(terms, basestring)):
        terms = ",".join(terms)

    # If we're caching and the file exists, then use the local copy
    if (cache_trends and is_cached(terms)):
        return open(cached_trends_path(terms)).read()

    params = { "hl"      : "en-US",
               "q"       : terms,
               "tz"      : "",
               "content" : 1,
               "export"  : 1 }

    r = requests.get(url, params=params, cookies=google_cookies)

    if (cache_trends):
        f = open(cached_trends_path(terms), "w")
        f.write(unicodedata.normalize('NFKD', r.text).encode('ascii','ignore'))

    return r.text

# Takes the Trends CSV from the Google query and parses it into a dict
def parse_trends_csv(report, q_suff=""):
    f = StringIO.StringIO(report)
    reader = csv.reader(f)

    in_block = 0

    data = {}

    week_str = "\d{4}\-\d{2}\-\d{2}"
    week_p = re.compile(week_str + "\s+\-\s+(" + week_str + ")")
    month_str = "(\d{4}\-\d{02})"
    month_p = re.compile(month_str)

    for row in reader:
        if (row == None or not row):
            if (in_block):
                break
            else:
                continue
        if (not in_block and row[0] != "Interest over time"):
            continue
        elif (not in_block and row[0] == "Interest over time"):
```

```
                in_block = 1
                continue
            elif (in_block and row[0] == "Week"):
                queries = [r.lower() + q_suff for r in row[1:]]
                date_p = week_p
                continue
            elif (in_block and row[0] == "Month"):
                queries = [r.lower() + q_suff for r in row[1:]]
                date_p = month_p
                continue
            elif (in_block and row[0] == ""):
                in_block = 0
                break

            # Now we're in the block
            m = date_p.search(row[0])
            if (m == None or m.group(1) == None):
                print "Couldn't find date in cell '" + row[0] + "'"
                sys.exit()
            else:
                s = m.group(1)

                # Convert monthly to weekly starting at first day of the month,
                # which we will eventually transform to the end of the month
                if (not re.compile(week_str).match(s)):
                    s += "-01"

                end_date = datetime.datetime.strptime(s, "%Y-%m-%d").date()

                # Now push forward one month and pull back, if it's monthly
                if (not re.compile(week_str).match(s)):
                    end_date = end_date + datetime.timedelta(months=1) -\
                               datetime.timedelta(days=1)

            if (end_date in data):
                print "Date " + str(end_date) + " already seen!"
                sys.exit()

            out = []
            for v in row[1:]:
                try:
                    out.append(int(v))
                except ValueError:
                    out.append(v)

            data[end_date] = dict(zip(queries, out))

        return data

    # Parse args
    if (len(sys.argv) < 2):
        print "usage: " + sys.argv[0] + " <output file> <input files>"
        sys.exit()
    elif(len(sys.argv) == 2):
        out_f = sys.argv[1]
```

```
        in_f = [os.path.join(grosses_path, f) for f in os.listdir(grosses_path)
                    if os.path.isfile(os.path.join(grosses_path, f)) and
                       os.path.splitext(f)[1] == ".xlsx"]
    else:
        out_f = sys.argv[1]
        in_f = sys.argv[2:]

    # Either open the output workbook or a create a new one, depending on whether
    # we're appending
    append_mode = os.path.isfile(out_f)

    if (append_mode):
        print "File '" + out_f + "' already exists -- appending"
        out_wb = load_workbook(out_f)
        out_ws = out_wb[out_wb.get_sheet_names()[0]] # Assume first sheet
    else:
        out_wb = Workbook()
        out_ws = out_wb.active
        out_ws.title = "Grosses"

    shows = {}
    trends_by_show = {}

    print "Iterating grosses files"
    # Copy each input file to the output workbook
    for i in range(len(in_f)):
        sys.stdout.write(".Reading file " + str(i + 1) + " of " + str(len(in_f)) +\
                         ", '" + os.path.split(in_f[i])[1] + "'\n")

        in_wb = load_workbook(in_f[i])
        in_ws = in_wb[in_wb.get_sheet_names()[0]] # Assume first sheet

        if (in_ws.cell(row=1, column=1).value != "Show name"):
            show_name = parse_show_name(in_f[i], i)
        else:
            show_name = unicode(in_ws.cell(row=2, column=1).value)

        # For the first sheet of the file, need to copy header rows
        if (i == 0 and not append_mode):
            for j in range(len(in_ws.columns)):
                h = (in_ws.cell(row=r + 1, column=j + 1).value
                     for r in range(header_rows))

                out_ws.cell(row=1, column=j + 1).value = \
                    " ".join(c for c in h if c != None)

            j = len(in_ws.columns) + 1
            out_ws.cell(row=1, column=j).value = "Weeks since open"

            out_r = 2
        elif (i == 0 and append_mode):
            out_r = len(out_ws.rows) + 1 # 1 for 1 indexing + 1 for next row

        sys.stdout.flush()

        shows[show_name] = read_grosses_date(in_ws.cell(row=2, column=2).value)
```

```
        # Find the last week in the gross data, to determine Trends availability
        max_week = read_grosses_date(in_ws.cell(row=len(in_ws.rows),
                                       column=2).value) - datetime.timedelta(days=7)
        trends_avail = max_week > trends_begin

        # Fetch trends if available and store them for later
        if (get_trends and trends_avail and len(in_ws.rows) > min_weeks_for_trends):
            trends = fetch_google_trends(show_name)
        else:
            trends = None

        trends_by_show[show_name] = trends


        # Copy cell by cell -- there may be a more efficient way to do this...
        for r in range(3, len(in_ws.rows) + 1):
            for c in range(1, len(in_ws.columns) + 1):
                v = in_ws.cell(row=r, column=c).value
                if (c > 2 and v.find("N/A") == -1):
                    v = string.replace(v, "$", "")
                    v = string.replace(v, ",", "")

                    if (v.find("%") > -1):
                        v = string.replace(v, "%", "")
                        w = float(v) / 100
                    else:
                        if (v.find(".") > -1):
                            w = float(v)
                        else:
                            w = int(v)
                else:
                    w = v.strip()

                out_ws.cell(row=out_r, column=c).value = w

            out_ws.cell(row=out_r, column=len(in_ws.columns) + 1).value = r - 3

            out_r += 1


    terms = list(("Name" + s for s in suffixes))
    rels = list((t + " relative" for t in terms))
    terms.extend(rels)
    trends_ws = [out_wb.create_sheet() for t in terms]

    # Create the Trends sheets -- one for each query
    for i in range(len(trends_ws)):
        trends_ws[i].title = "Trends - " + terms[i]

    # Pull Trends (if available) from 2 years before open to 12 years after
    date_range = range(-2 * 52, 12 * 52)

    # Print headers
    for ws in trends_ws:
        ws.cell(row=1, column=1).value = "Show name"
```

```
    c = 2
    for d in date_range:
        ws.cell(row=1, column=c).value = d
        c += 1


# Pull the Trends data from the list and put it in the appropriate sheet, show
# by show
r = 2
for s in shows.keys():
    trends_terms = list((s.lower() + suff for suff in suffixes))
    rels = list((t + " relative" for t in trends_terms))
    trends_terms.extend(rels)

    t = 0
    opening = shows[s]
    trends = trends_by_show[s]

    for ws in trends_ws:
        c = 1
        ws.cell(row=r, column=c).value = s

        c += 1

        if (trends == None):
            continue

        for d in date_range:
            week = opening + datetime.timedelta(days=7) * d

            trends_weeks = list((k for k in trends.keys() if k <= week))

            if (not trends_weeks):
                v = "N/A"
            else:
                latest_week = max((k for k in trends.keys() if k <= week))
                trends_data = trends[latest_week]
                if (trends_terms[t] in trends_data):
                    v = trends_data[trends_terms[t]]
                else:
                    v = "N/A"

            ws.cell(row=r, column=c).value = v
            c += 1

        t += 1

    r += 1

out_wb.save(out_f) # Save the concatenated file
```

Listing 4: Python script for retreiving show categories

```
#!/usr/bin/python
import mechanize
import sys
import csv
```

```python
import time

# Parse the response from IBDB, returning the category with the most "votes"
# (i.e. occurrences)
def read_response(resp):
    categories = ["musical", "play", "special"]

    resp = resp.lower()

    cat_i = [resp.count(c) for c in categories]
    cat_i[0] -= 1 # Net out base rate occurrences in the page
    cat_i[1] -= 8

    sys.stderr.write(str(cat_i) + "\n")

    if (max(cat_i) > 0):
        cat = categories[cat_i.index(max(cat_i))]
    else:
        cat = "NA"

    return cat

in_fn = sys.argv[1] # List of shows
out_fn = sys.argv[2] # Output file

# Initialize a browser that looks like Firefox
br = mechanize.Browser()
br.addheaders = [('User-agent', 'Mozilla/5.0 (X11; U; Linux i686; en-US;' +\
                  'rv:1.9.0.1) Gecko/2008071615 Fedora/3.0.1-1.fc9' +\
                  'Firefox/3.0.1')]

# Read in the list of shows
shows = {}
show_data = {}
f = open(in_fn, "rU")
reader = csv.reader(f)

for r in reader:
    s = r[0]
    shows[s] = 1

show = shows.keys()

# Query each show
i = 0
for s in shows:
    sys.stderr.write(s + ": ")

    # Search for that show's name, and then parse the result
    search_page = br.open("http://www.ibdb.com/advSearchShows.php")
    br.select_form(nr=0)
    br.form["ShowProperName"] = s
    br.submit()
    show_data[s] = read_response(br.response().read())

    i += 1
```

```
        time.sleep(1) # To avoid overloading the IBDB server


# Write the output
f = open(out_fn, "wb")
writer = csv.writer(f)

writer.writerow(["Show", "Category"])
for s in show_data.keys():
    writer.writerow([s, show_data[s]])
    print ",".join([s, show_data[s]])
    sys.std.out.flush()

f.close()
```