Contents

1 Project Definition & Introduction

The Mall of America (MoA) is the largest shopping mall in the United States and is located in Bloomington, Minnesota. Every year, the mall produces $2 billion in economic activity. With more than 11,000 employees, it is a major contributor to Bloomington's job base. Bloomington has more jobs per capita than Minneapolis or St. Paul at a rate that is 52 percent higher than an average city[1].

One of its star attractions is the 7-acre indoor amusement park Nickelodeon Universe (NU), owned by MoA itself. The park is a significant contributor to MoA's revenue. Functions like staffing, maintenance, and budgeting for the park become crucial towards ensuring an enjoyable and hassle-free guest experience. These functions are all dependent primarily on how well the mall can predict the theme park's ridership for both the short-term and long-term.

Currently, the mall projects the ridership in the fall of every year, but manually. This process gives them results which are off by 20-30% on the worst-case days and consumes 2 weeks of a full-time employee's time annually. Therefore, the key challenge here is to design an automated modeling process that reduces both the baseline prediction error and current model turnaround time, and provides interpretable results.

2 Solution Overview

The final recommendation is a three-tier solution. Three models are provided that serve three distinct functions.

The annual model replaces the current manual process. It provides daily ridership predictions a year in advance. This model is used for the annual planning that occurs each fall and is used for costing purposes, operating budgets and annual staffing predictions. Much of the model has been automated. There are several features which were not able to be generated by the model. Calculating school calendars will continue to be a manual process going forward, and daily weather data will need to be captured by the same procedure that is currently in place. The current method of using the same model for both long and short-term staffing has limitations, so two additional models were created.

Short-term staffing needs will be better served by the second-tier solution: the addition of a fourteen-day model. The fourteen-day model is fine tuned to current weather and recent actual ridership numbers. It improves upon predictions and will reduce over and understaffing in the short term, with the actual staff scheduling process. Improving upon actual staffing will assist the MoA and NU in two ways. More accurate staffing will provide for a better guest experience if staff numbers are better matched to the number of guests on any given day. It will also improve staff morale to have the ideal number of employees at the park on any given day. Over or understaffing is difficult for employees. Too few and it can become stressful, too many and there is boredom or employees are potentially let go early or called off.

The third-tier is a one-day look ahead that will assist with short term adjustments like staffing alerts for last minute surges or lags in projected ridership for the next day. It will also assist with park set up. It will provide a quick and easy check on the number of cars to have set up and ready to go on any given day.

3 Technical Specifications

The technical environment used for the analysis is as follows:

Platform	x86_64-w64-mingw32
Arch	x86_64
OS	mingw32
System Status	x86_64, mingw32
Major	3
Minor	4.1
Year	2017
Month	06
Day	30
Svn Rev	72865
Language	R
Version.String	R version 3.4.1 (2017-06-30)
Nickname	Single Candle

Technical specifications can be verified simply by typing the command 'version' in the R Console.

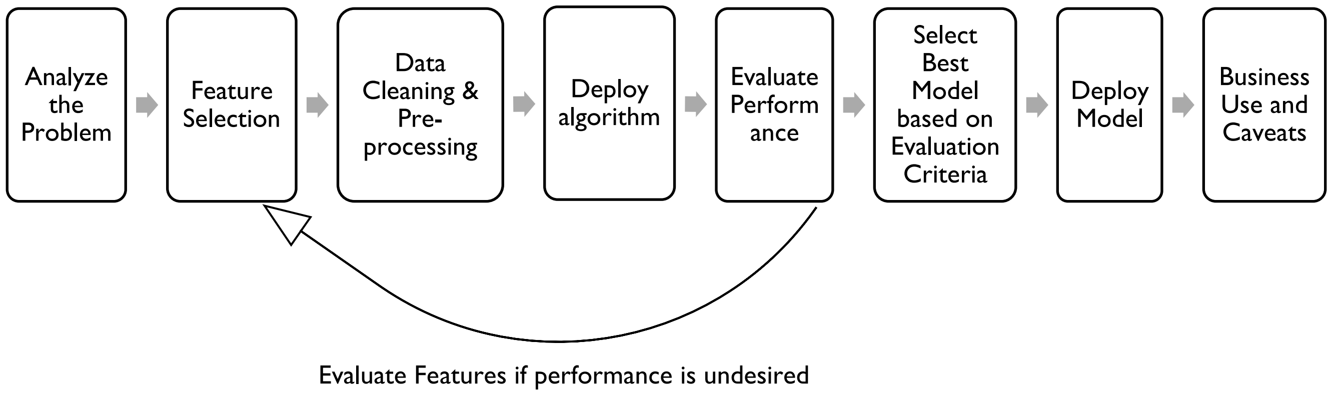
We have applied the following R libraries in data processing and modeling:

R packages		
caret	Metrics	randomForest
Cubist	mice	randomGLM
dplyr	MLmetrics	RColorBrewer
e1071	nnet	readxl
elasticnet	penalized	reshape2
ggplot2	plyr	rnn
lars	qrnn	RRF
LiblineaR	Quandl	xgboost
lubridate	quantregForest	xts

- The packages can be installed by using the command 'install.packages(<package-name>)' in the console.
- As shown above, the packages can be loaded by using the command 'library(<package-name>)' in the console.

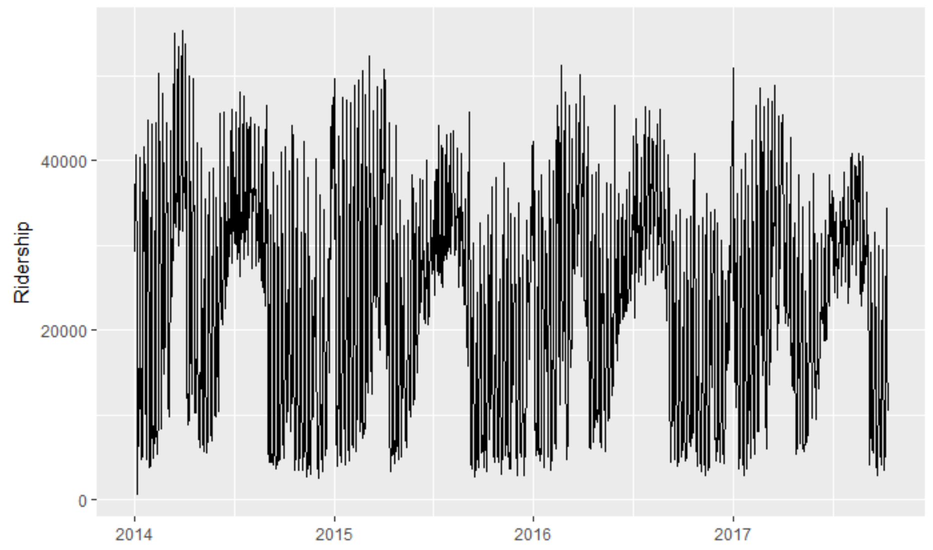
4 Methodology

The first step was to analyze the problem and understand the situation, complication, and key question. In any modeling process, the data that goes into a model plays a big role in ensuring accurate results. Therefore, next, the relevant features that helped achieve the objective were defined and an initial feature set was selected. Following this, any noisy or missing data was removed. (There were very few instances of this, i.e. Christmas. Some field's values were imputed - For example those of extreme temperature and Percentage of students out of school) After the data engineering, the core modeling process was started. Different algorithms were implemented on the feature set, along with cyclical addition and removal of features depending on performance and complexity of the features and the model used. Features that had obvious correlations to each other were not used together (For example, we considered using unemployment rate in Minnesota and Consumer Confidence as two measures of discretionary income, however the two are correlated and don't provide too much unique information. Therefore, we chose Unemployment because it was available at a state level and much easier to obtain, unlike Consumer Confidence which was involved a tedious extraction process). Next, the model with the best performance was chosen based on the evaluation criteria. Finally, the model was tested, and its possible business applications and caveats were reported.


Analyze the Problem
Feature Selection
Data Cleaning & Pre-processing
Deploy algorithm
Evaluate Perform ance
Select Best Model based on Evaluation Criteria
Deploy Model
Business Use and Caveats
Evaluate Features if performance is undesired

5 Analysis

Two projections (models) are proposed to Mall of America – a static model and a dynamic model. A static model would predict the daily ridership of the park for 2018 where as a dynamic model would be useful for predicting the ridership for a specified time (for example, if MOA wants to look at projected ridership for the next 2 weeks or the next day, it can do so). The first thing we did was to plot Ridership to see trends and seasonality and observe any abnormal time-periods.



After the missing value and outlier treatment is carried out, the dataset is split into a training set on which the model is trained on, and a validation set on which the model will be tested. 10-fold cross validation is used here, so the model is trained and test on all parts of the data to get the best model for that algorithm (not including tuning). The model is then tested on the validation set to avoid overfitting results. This rationale for this is explained in Section **5.2.1.**

5.1 Modeling

This is essentially a regression problem as the objective is to predict a discrete numerical value of ridership. First, a static model is used to project ridership. The first step in the **core** modeling process (after data preprocessing) is feature selection.

5.1.1 Feature Selection

The models and associated information are provided in R markdowns. Preprocessing code is included that will convert MoA ridership and additional feature data. The code is based on the feature formats and naming conventions received from MoA. The transformations include, for example, breaking the dates down into more granular features like day of the week and month of the year. Additional transformations include the addition of historic averages of high and low temperatures and, holiday affects (Christmas.x). In addition, code is provided to calculate lags as required for the dynamic model.

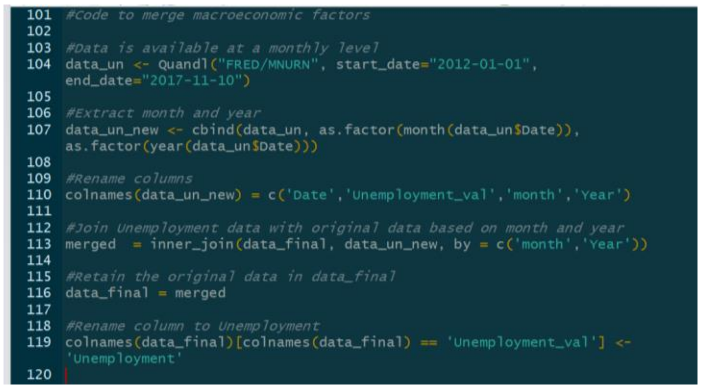
From the files provided by the Mall of America, the following features were determined to be significant and used in the annual model. Outside data was sourced as well, as marked below. Note that feature selection is a cyclical process – some features work better with some models. However, there were no algorithms that provided a major increase in performance with the addition of any specific feature from this dataset. In other words, there was no algorithm-feature combination that provided significantly improved results as compared to the final algorithm that was used, for any one feature in particular. The following features were used below with different algorithms (explained in the following section)

- **Month of the Year** - wide variation based on time of the year
- **Day of the Week** - wider variation based on the day of the week. Days later in the week and on weekends tended to have higher ridership.
- **Week of the Month** - there was an element of variation based on the time of the month. This could be due to pay cycles, fluctuating guest discretionary income throughout the month due to fixed monthly income sources (like government assistance, retirement income, or social security benefit payments) or fixed monthly expenses (like rent or utilities being due at specific points in a month).
- **Holiday Status** - whether a day for which we are predicting ridership is a holiday or not.
- **Percent of Area Students Out of School** - Using the data from MOA for students who are out of school on specific dates, we tabulated all the students in the population and all the students out of school. We divided the number of students out of school by the total population of students in the area on any given date.
- **Average Prior Ridership** - We wanted to use some element of prior daily ridership in our prediction, but wanted to do so in a way that would allow our model to predict future ridership with a degree of flexibility. To achieve this, we divided our dates into two groups - holidays and normal days.
    - For holidays we looked at the ridership on the exact same holiday in the prior two years. We took these two prior ridership amounts and averaged them. For holidays for which we were making a prediction, we used this value as our "Average Prior Ridership" figure. *Note that "holiday" is meant to convey traditional (religious and/or recognized by the government) holidays and on occasion days immediately prior to or following traditional holidays. Additionally,*

traditional holidays that did not have a noticeable effect on ridership were excluded as "holidays" for this purpose. Traditional holidays which were removed were Easter, Fourth of July, and Christmas Eve.
    - For normal days we took historical ridership on non-holidays and averaged ridership by year, month, and weekday. When we then went to predict future ridership on a non-holiday, we used this average ridership value for the prior year and two years prior. We then averaged these figures to make one predictor called "Average Prior Ridership".
- **AvgHighF** - Average high temperature for the prior decade's weather data- Pulled from an external source.[2]
- **AvgLowF** - Average low temperature for the prior decade's weather data – Pulled from an external source.[2]
- **Christmas.x** - We noticed that the two days prior to Christmas Eve, and the three days following Christmas experienced a particularly strong increase in ridership phenomenon. We created a binary variable specifically for these days such that 12/22, 12/23, 12/26, 12/27, and 12/28 received a 1 for this field and all other days received a zero. We posit that these days have such increased ridership due to exceptionally high holiday traffic on these days. We believe that Christmas Eve (12/24) does not experience this same effect as most guests would traditionally spend Christmas Eve with their families.
- **Unemployment** - This is the Minnesota unemployment percentage on the day of the prediction. This economic indicator proved to be useful likely because it provides rough information about the percentage of the population with discretionary income. From a technical standpoint, this was an external feature that we pulled using the Quandl package in R. (shown in code)

Additional features used in the 14-day model include all the features used in the annual prediction plus the features listed below:

- **Ridership** - The Ridership 14 and 28 days prior to the day being predicted.
- **ExtremeTemp** - whether the weather was "extreme" defined as being above 85F or below 0F on the day of prediction and 14 days prior. This ended up being a useful predictor as guest behavior was negatively impacted by extreme temperatures. This field constructed from the weather file provided by the Mall of America.
- **Percent of Area Students Out of School** - percent of area students out of school on 14 days prior and 28 days prior.
- **High and Low Temperatures** - the high and low temperatures on the day for which we are predicting ridership. These values would be taken from weather predictions for future predictions. This field from the weather file provided by the Mall of America.
- **Snow Total:** The snow in inches (to the 100ths of an inch) on the day to be predicted. This value would be approximated from weather forecasts at the time of the prediction. This field from the weather file provided by the Mall of America.
- **Unemployment** - This is the Minnesota unemployment percentage on the day of the prediction. This economic indicator proved to be useful likely because it provides rough information about the percentage of the population with discretionary income. From a technical standpoint, this was an external feature that we pulled using the Quandl package in R. (shown below in code)



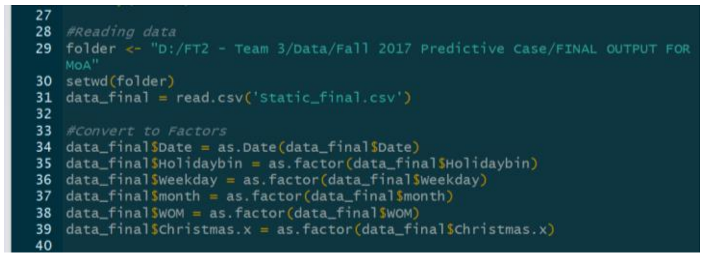
```r
#Code to merge macroeconomic factors

#Data is available at a monthly level
data_un <- Quandl("FRED/MNURN", start_date="2012-01-01",
end_date="2017-11-10")

#Extract month and year
data_un_new <- cbind(data_un, as.factor(month(data_un$Date)),
as.factor(year(data_un$Date)))

#Rename columns
colnames(data_un_new) = c('Date','Unemployment_val','month','Year')

#Join Unemployment data with original data based on month and year
merged  = inner_join(data_final, data_un_new, by = c('month','Year'))

#Retain the original data in data_final
data_final = merged

#Rename column to Unemployment
colnames(data_final)[colnames(data_final) == 'Unemployment_val'] <-
'Unemployment'

```

Additional features used in the 1-day model include all the features used in the annual prediction plus the features listed below:

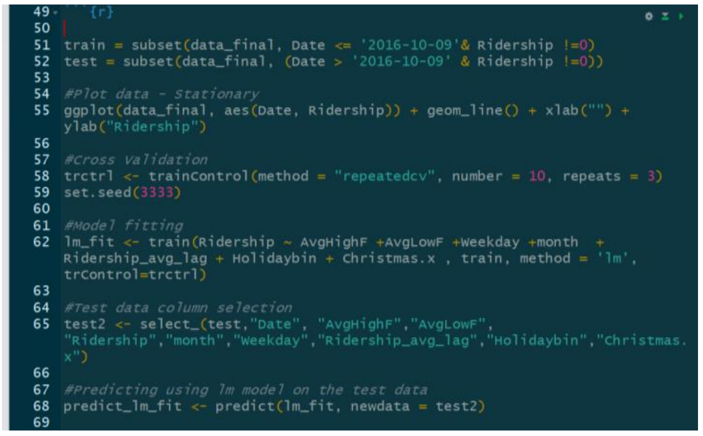
- **Ridership** - previous day's and one week prior
- **Percent of Area Students Out of School** - percent of area students out of school for the prior day and for one week prior.
- **High and Low Temperatures** - the high and low temperatures on the day for which we are predicting ridership. These values would be taken from weather predictions for future predictions. This field constructed from the weather fle provided by the Mall of America.
- **Unemployment** - This is the Minnesota unemployment percentage on the day of the prediction. This economic indicator proved to be useful likely because it provides rough information about the percentage of the population with discretionary income.
- **ExtremeTemp** - Whether the weather was "extreme" on the day for which we are predicting ridership. "Extreme" is defined as having a high temperature above 85F or a low temperature below 0F. This ended up being a useful predictor as guest behavior was negatively impacted by extreme temperatures.

5.1.2 Algorithm Selection

For algorithm selection, we started with the simplest model. We determined this to be a linear regression model, and ran it on the entire validation set (which was a 365-day period post 2016-10-09) and measuring performance. Code snippets are shown below:



```r

#Reading data
folder <- "D:/FT2 - Team 3/Data/Fall 2017 Predictive Case/FINAL OUTPUT FOR
MoA"
setwd(folder)
data_final = read.csv('Static_final.csv')

#Convert to Factors
data_final$Date = as.Date(data_final$Date)
data_final$Holidaybin = as.factor(data_final$Holidaybin)
data_final$Weekday = as.factor(data_final$Weekday)
data_final$month = as.factor(data_final$month)
data_final$WOM = as.factor(data_final$WOM)
data_final$Christmas.x = as.factor(data_final$Christmas.x)

```



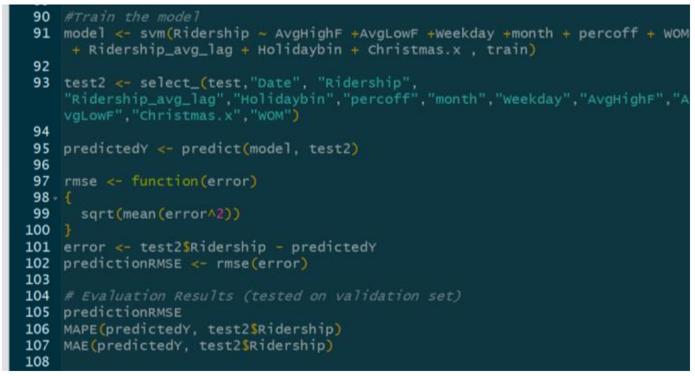
~~~r
```{r}

train = subset(data_final, Date <= '2016-10-09'& Ridership !=0)
test = subset(data_final, (Date > '2016-10-09' & Ridership !=0))

#Plot data - Stationary
ggplot(data_final, aes(Date, Ridership)) + geom_line() + xlab("") +
ylab("Ridership")

#Cross Validation
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
set.seed(3333)

#Model fitting
lm_fit <- train(Ridership ~ AvgHighF +AvgLowF +Weekday +month  +
Ridership_avg_lag + Holidaybin + Christmas.x , train, method = 'lm',
trControl=trctrl)

#Test data column selection
test2 <- select_(test,"Date", "AvgHighF","AvgLowF",
"Ridership","month","Weekday","Ridership_avg_lag","Holidaybin","Christmas.
x")

#Predicting using lm model on the test data
predict_lm_fit <- predict(lm_fit, newdata = test2)

~~~

For the entire code, refer to file 'Static_Model_LM.RMD' [Section 9]

Comparing the results to MOA's baseline model, the 365-day model performed worse than the baseline.

After adding additional features like Week of month and a numerical value of % students out of school, we observed a slight improvement (performance measures detailed in Section **5.2.2**), but deduced that a more complex algorithm may perform better.

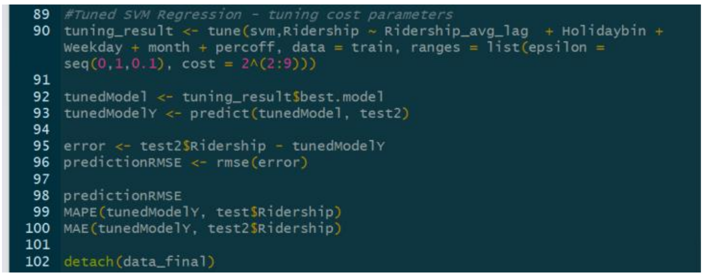
Our next iteration was **a support vector machine** with a tuned cost function, this gave us slightly improved performance.



```r
#Train the model
model <- svm(Ridership ~ AvgHighF +AvgLowF +Weekday +month + percoff + WOM
 + Ridership_avg_lag + Holidaybin + Christmas.x , train)

test2 <- select_(test,"Date", "Ridership",
"Ridership_avg_lag","Holidaybin","percoff","month","Weekday","AvgHighF","A
vgLowF","Christmas.x","WOM")

predictedY <- predict(model, test2)

rmse <- function(error)
{
  sqrt(mean(error^2))
}
error <- test2$Ridership - predictedY
predictionRMSE <- rmse(error)

# Evaluation Results (tested on validation set)
predictionRMSE
MAPE(predictedY, test2$Ridership)
MAE(predictedY, test2$Ridership)

```

For the entire code, refer to file 'Static_Model_SVM.RMD' [Section 9]

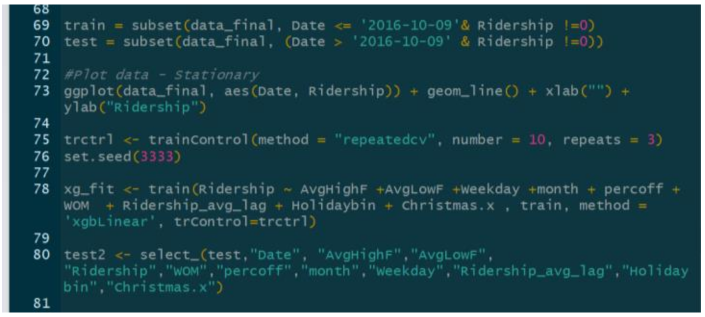
After tuning the SVM regression, we noticed significant improvement. Code snippet is shown below:



```r
#Tuned SVM Regression - tuning cost parameters
tuning_result <- tune(svm,Ridership ~ Ridership_avg_lag  + Holidaybin +
Weekday + month + percoff, data = train, ranges = list(epsilon =
seq(0,1,0.1), cost = 2^(2:9)))

tunedModel <- tuning_result$best.model
tunedModelY <- predict(tunedModel, test2)

error <- test2$Ridership - tunedModelY
predictionRMSE <- rmse(error)

predictionRMSE
MAPE(tunedModelY, test$Ridership)
MAE(tunedModelY, test2$Ridership)

detach(data_final)
```

For the entire code, refer to file 'Static_Model_SVM.RMD' [Section 9]

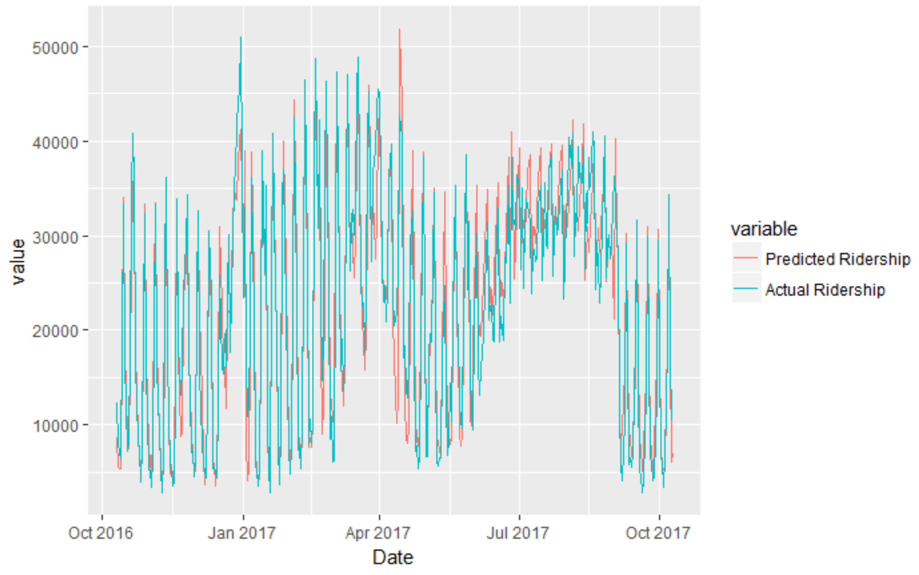
To attempt to improve performance further, we moved on to ensemble models. We tried using boosted versions of models including xGboost and random forests, but they did not yield a decrease in RMSE either (compared to the best 12-period linear model). Code snippet is shown below:



```
train = subset(data_final, Date <= '2016-10-09'& Ridership !=0)
test = subset(data_final, (Date > '2016-10-09' & Ridership !=0))

#Plot data - Stationary
ggplot(data_final, aes(Date, Ridership)) + geom_line() + xlab("") +
ylab("Ridership")

trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
set.seed(3333)

xg_fit <- train(Ridership ~ AvgHighF +AvgLowF +Weekday +month + percoff +
WOM  + Ridership_avg_lag + Holidaybin + Christmas.x , train, method =
'xgbLinear', trControl=trctrl)

test2 <- select_(test,"Date", "AvgHighF","AvgLowF",
"Ridership","WOM","percoff","month","Weekday","Ridership_avg_lag","Holiday
bin","Christmas.x")
```

For the entire code, refer to file 'Static_Model_XGB.RMD' [Section 9]

However, it was another model that yielded best results, known as **Cubist.** The cubist algorithm, which is an enhanced version of a regression tree, proved to have the greatest predictive performance based on MAPE, RMSE, and MAE of all the models that we built.



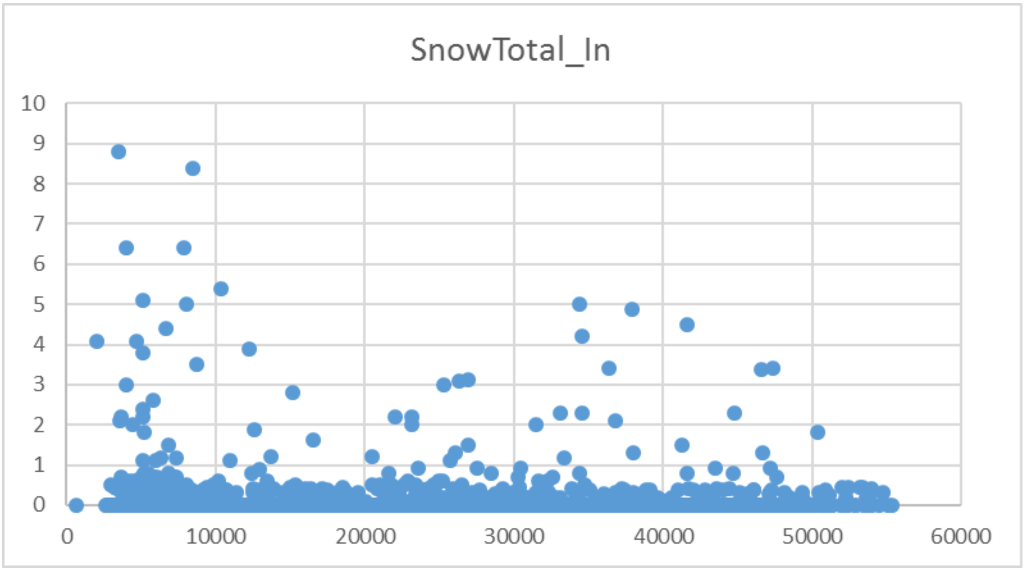
The file 'export_static_cubist.csv' [Section 9] has the actual vs projected values for the test data.

We also tested the same algorithms above for the **dynamic model**, and Cubist once again turned out to give the best performance. In addition to the above algorithms, we tried to implement a recurrent neural network model for the dynamic version, but did not proceed because of the difficulties in obtaining the right format for input into the algorithm, as well as for interpretability reasons. Since Cubist gave us such successful results, we explain it in detail below.

Cubist

Cubist is a regression-tree method for generating rule-based predictive models from data[3]. The goal of cubist is to estimate the target variable's values (in this case Ridership) based on other relevant attributes. In this algorithm, multiple linear models are built, based on certain rules or conditions being satisfied. This occurs at every node of the regression, and every child node uses a part of its parent's coefficients to fine tune the model. Essentially, it is based on the simple logic (going down a tree) as follows: If a condition is met, then apply linear regression.

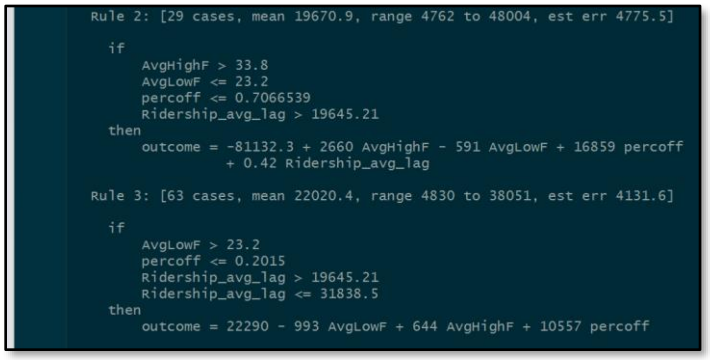
One of the main reasons why we hypothesize that this algorithm is appropriate in the context of ridership projections using our suggested features, is because the model is piecewise linear, implementing segmented regression. This means that independent variables exhibit different relationships (magnitude and direction) with the target variable in different buckets of their range of values. For example, the relationship between the ridership and amount of snow is stronger when snow is between 0-1 inch, and less definitive otherwise (While most records are between 0-1, there are also some ridership buckets that vary for different values). There is a certainly a case to be made that this emulates the kind of relationships that **actually exist** between Ridership and our suggested features, as one would expect lesser prospects of snow to increase the chances of a guest going to the park.



***Plotted in MS Excel from file 'dynam_lags_FINAL.csv' [Section 9]**

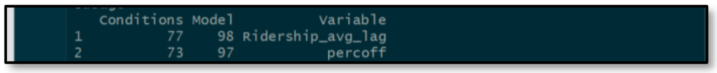
Snippets of the model output of the tree is as follows.

Note: *These snippets are the output of the cubist.default trained model. We built our models using the caret package, but there is a separate Cubist package as well. While Cubist package provides a way to look at every decision rule and regression equation in the tree, we chose its caret implementation for ease of use and the fact that it automatically chooses the optimal parameters. However, as a future step, implementing the algorithm in cubist may yield even more interpretative results. The code to get these outputs, along with the main code, is available in file 'Static_Model_Cubist.RMD'[***Section 9***] for the static model, and file 'Dynamic_Model_Cubist'[***Section 9***] for the dynamic model.*



```
Rule 2: [29 cases, mean 19670.9, range 4762 to 48004, est err 4775.5]

  if
      AvgHighF > 33.8
      AvgLowF <= 23.2
      percoff <= 0.7066539
      Ridership_avg_lag > 19645.21
  then
      outcome = -81132.3 + 2660 AvgHighF - 591 AvgLowF + 16859 percoff
                + 0.42 Ridership_avg_lag

Rule 3: [63 cases, mean 22020.4, range 4830 to 38051, est err 4131.6]

  if
      AvgLowF > 23.2
      percoff <= 0.2015
      Ridership_avg_lag > 19645.21
      Ridership_avg_lag <= 31838.5
  then
      outcome = 22290 - 993 AvgLowF + 644 AvgHighF + 10557 percoff
```

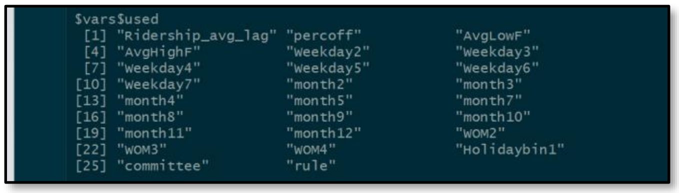
The picture above is an example of a snippet of a rule-linear model combination generated by the model.



```
   Conditions Model          Variable
1          77    98 Ridership_avg_lag
2          73    97           percoff
```

(The picture above implies that Ridership lag is used in the condition part of rules that cover 77% of the cases and in the formulas of rules that cover 98% of the cases)

The final set of features used in the static model are:



```
$vars$used
 [1] "Ridership_avg_lag" "percoff"           "AvgLowF"
 [4] "AvgHighF"          "Weekday2"          "Weekday3"
 [7] "Weekday4"          "Weekday5"          "Weekday6"
[10] "Weekday7"          "month2"            "month3"
[13] "month4"            "month5"            "month7"
[16] "month8"            "month9"            "month10"
[19] "month11"           "month12"           "WOM2"
[22] "WOM3"              "WOM4"              "Holidaybin1"
[25] "committee"         "rule"
```

We also note that:

- The caret package in R tunes the parameters in the model. We also use a parameter called committees. (the optimal number of committees = 10). A committee based model is essentially comprised of multiple rule-based models. The first committee generates a rule-based model. The second committee is designed to improve the prediction of the first one, the third of the second, and so on. Caret automatically provides the optimal number of committees.
- Rules can overlap.

The primary advantages of this algorithm are its interpretability and speed as compared to other models (both ensemble and non-ensemble). It also automatically deals with missing values.

5.2 Model Evaluation

5.2.1 Why Training, Testing and Visualization?

If we were to build AND test a model on the same dataset, then we are bound to commit overfitting. Overfitting is a problem that can cover up the deficiencies of a model; therefore, it is important that we split the dataset into a training set on which we train the model, and a validation set on which we validate it. This split is carried out first. In our analysis, we have also carried out the process of 10-fold cross-validation to give us the best performing model **WITHIN** the algorithm we have chosen. Essentially, cross validation here splits the data into 10 pieces – 9 train, 1 test and repeats this in a way such that every fold is given a chance to be the test dataset. This therefore considers all the data for modeling (within the train set) and tests it on every fold- removing bias. Following this, the trained model is applied on the validation set which we had kept aside earlier, and the following evaluation metrics are used to evaluate the performance on the validation set:

5.2.2 Performance Evaluation

The main performance metrics to judge the accuracy of our model are mean absolute error (MAE), mean average percentage error (MAPE), and root mean square error (RMSE). Ultimately MOA is interested in getting a close estimate of ridership, so a balance of good performance as measured by each of these metrics may prove to be the most valuable.

A comparison of the major models considered for evaluation, and their performance, is shown below*:

	Metric		
	RMSE	MAPE	MAE
Model			
MOA Manual Prediction	4462	20.16%	3465
Linear Regression (OLS)	5228	26.5%	3902
Support Vector Machine	4495	19.11%	3294
Ensemble xGBoost Tree	4385	21.95%	3395
Tuned Support Vector Machine	3945	17.5%	3016
Regression Tree (Cubist- caret)	3780	17.35%	2851

*These metrics are for a long-term static model.

5.2.3 Static Model (Cubist)

RMSE	3780
MAPE	17.35%
MAE	2851

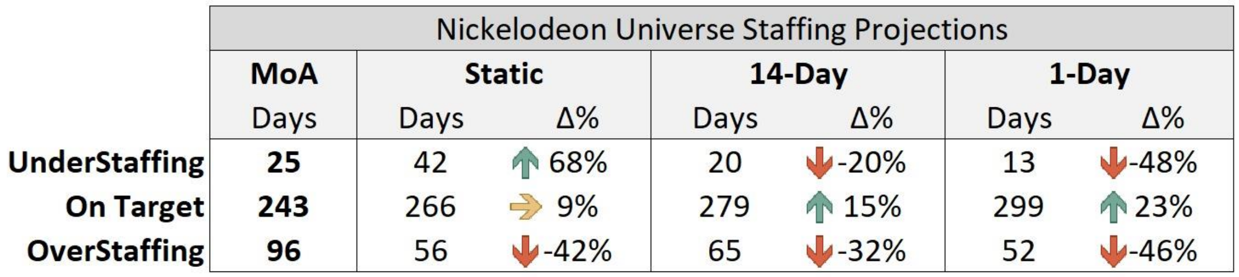
5.2.4 Dynamic Model (Cubist)

	14-day	1-day
RMSE	3546	3077
MAPE	16.1%	13.2%
MAE	2693	2266

6 Business Insights

The final three-tier solution (annual, fourteen-day & one-day models) will provide a close estimate of ridership that is more accurate and easier to implement than the current forecasting approach. The three-tier solution will save MoA weeks of manual labor over what is currently expended during annual forecasting. Additionally, value will be gained by allowing NU to more precisely staff their park according to the shorter-term model's day of operation predictions. It will also better anticipate revenue forecasts. The short-term models will allow MoA to save money when park ridership is low by operating with a skeleton crew, or by increase their staff on high ridership days.

The annual model provides a 19% decrease in staffing prediction errors. The fourteen-day model reduces overstaffing by 32% and understaffing by 20%. The complete breakdown is shown in the table below:



	Nickelodeon Universe Staffing Projections						
	MoA	**Static**		**14-Day**		**1-Day**	
	Days	Days	Δ%	Days	Δ%	Days	Δ%
UnderStaffing	**25**	42	↑ 68%	20	↓ -20%	13	↓ -48%
On Target	**243**	266	→ 9%	279	↑ 15%	299	↑ 23%
OverStaffing	**96**	56	↓ -42%	65	↓ -32%	52	↓ -46%

7 Future Steps

Risks associated with proposed plan/Mitigation Strategy include the variability and changing nature of some of the features.

- The variability in school schedules will still require a manual process to collect and collate values for each school.
- Global changing weather patterns will necessitate continuing support for accurate ridership predictions. Historic averages should be monitored.
- The models are based on past ridership over two years in the longest term, if there are sudden or sharp changes due to unforeseen factors predictions may be affected.
- From a technical standpoint, implementing the algorithm using the cubist.default method in R, in conjunction with the caret method, may yield more informative results.

8 Challenges Faced

Challenges fell into several categories. The two biggest challenges were picking the best model and secondly, picking the most important features.

Finding the most important features and breaking them down to the best granularity required testing many different combinations of features, at different levels. It was an iterative process that eventually was fruitful. Another challenge was separating the features to find the most important. It was challenging due to overlap and collinearity that occurs between the different variables. For example, percent of school children off is directly related to holidays, and ridership numbers are directly affected by various weather attributes. Finding the right combination of features that would provide the best solution required many iterations and a variety of transformations.

Thorough testing of a variety of models yielded a strong result. The result is a model that performs well and will serve the Nickelodeon Universe and their guests well.

9 File Locations, Guidelines and Troubleshooting

The location of every file referenced in the analysis, that is required to run the code/models is given below (directory specified at the top):

D:\FT2 - Team 3\Data\Fall 2017 Predictive Case\FINAL OUTPUT FOR MoA		
S.No	**File**	**Description**
1	2012Ridership.xlsx	MoA Provided 2012 Ridership Numbers
2	Calendar_final.RMD	Feature Transformation Code

3	Calendar_holiday_rev_final.csv	Intermediary Ridership, Holiday, and Percentage off students
4	holiday_JAH.csv	Intermediary Named Holidays and Dates
5	Ridership_2012_agg.csv	Aggregated Ridership from 2012Ridership.xlsx across Rides
6	School.csv	MoA Provided School Information
7	Summer breaks.csv	Aggregated School Information
8	Usage.csv	MoA Provided Ridership Numbers 2013- 2017
9	Weather.csv	MoA Provided High/Low/Precip/Snow Information 2013-2017
10	weather-averages.csv	Historical Averages by month and day (High/Low/Precip/Snow)
11	Static_final.csv	Input File for Static Models
12	export_static_cubist.csv	Actual vs Projected Ridership for Test Dataset – Static
13	dynam_lags_FINAL.csv	Input File for Dynamic Models
14	export_dynamic_14d.csv	Actual vs Projected Ridership for Test Dataset – Dynamic 14d
15	export_dynamic_1d.csv	Actual vs Projected Ridership for Test Dataset – Dynamic 1d
16	Static_Model_LM.RMD	Code to run Static Linear Regression Model
17	Static_Model_SVM.RMD	Code to run Static SVM Model
18	Static_Model_XGB.RMD	Code to run Static xGBoost Model
19	Static_Model_Cubist.RMD	Code to run Static Cubist Model
20	Dynamic_Model_Cubist.RMD	Code to run Dynamic Cubist Models (14d & 1d)
21	Staffing.RMD	Code to estimate Staffing Numbers for Static & Dynamic Models

The code has been written in R Markdown files (.RMD). It contains several chunks of embedded code. To run each chunk, click on the green arrow in the top right corner. When the **Knit** button is clicked a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

In the event there are errors in the code (appears in the output at the bottom of code chunks along with line number of error) that cannot be debugged easily, the following people can be contacted:

- Kaustubh Lavakare, Haoyang Cai and Gowthami Peri for the model analysis and algorithms used
- Justin Hagstrom, Suzanne Kaminski and Ramnath Kamakoti for feature engineering

A few checks before starting are as follows:

- Ensure that the working directory in R/R Studio is the same as the location of all the files required to run the code. (This can be verified by the command getwd())
- Ensure that all packages in the code are installed before calling their libraries.
- Ensure that the 'Date' field in the data frames used is indeed in the Date format (can be converted to Date using as.Date() function if not). The code is written such that all dates are in Date format for the analysis but on occasion some systems internally change this format, leading to errors.

10 References

[1] http://www.startribune.com/some-mall-of-america-fast-facts-for-its-25th-anniversary/439745043/

[2] https://www.usclimatedata.com/climate/minneapolis/minnesota/united-states/usmn0503/2007/1

[3] https://www.rulequest.com/cubist-win.html