

Stat E-139 Project Report

*Stephen Camera-Murray, Desiree Koh,
Jennifer Le Hégaré, Elizabeth Wilson-Milne*

December 20, 2015

Contents

Overview	2
The Data	2
Derived Data	2
Checking the Independence Assumption	3
Checking the Normality Assumption	4
Checking the Linearity Assumption	5
Imagining Interaction Terms	5
Creating the Model	5
Process	5
Baseline Model	6
Full Model	7
Summary	11
Particular Challenges	11
Conclusion	11
A Fun Aside: The Kaggle Process	11
Appendix	13
Variables	13
Coefficient Table, Full Model	14
Code	16

Overview

Rossmann is a German drugstore chain which operates over 3,000 drug stores in 7 European countries. In order to schedule staff effectively, store managers need to predict their daily sales for six weeks in advance. Rossman decided to create a Kaggle competition to find a satisfactory prediction model.

For our project, our team joined this contest and created a model based on the historical data provided by Rossman. We used R for data exploration and manipulation, to create linear regression models, and to validate our results.

MORE HERE - TELL THE CONCLUSION, THEN SHOW HOW WE GOT THERE

The Data

The Rossmann data consisted of two files which, together, provided information about 942 days of sales at 1,115 stores. The first file contained store-level data such as:

- Type of store (in terms of its “model” and its overall selection)
- Information about the closest competitor (distance and longevity)
- Information about that store’s particular promotion cycle

The second file contained daily data for each store for 2013, 2014, and the first part of 2015. This data consisted of items such as:

- date (also noting that date’s exact weekday, holiday status, and school holiday status)
- whether or not that particular store was open that particular day
- whether or not the store was running a promotion that day
- sales amount and number of customers

Our first step was to combine the two datasets into one flat file. We also converted certain variables to strings and factors to prevent R from treating them as numerics. Then, after using all of the data to derive all additional variables, we removed the rows of “closed day” data. After all, the stores consistently posted zero sales when closed, and our model is to predict sales when open.

Derived Data

While the Rossmann-provided data had a huge number of records, we saw the opportunity to add several interesting columns. For example:

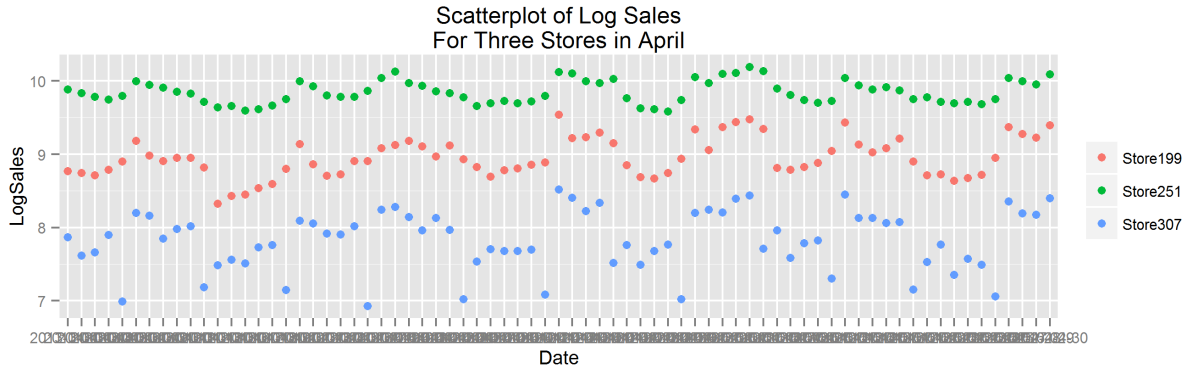
- We converted the rather esoterically-delivered promotion information to determine on exactly which days each store ran exactly how many promotions.
- We analyzed the distribution of distance between stores and their nearest competitors, and derived a *Close Competitor* variable for those with the 10% nearest Competitors.
- We analyzed which stores had summer sales that were at least 150% the level of winter sales, and hypothesized that these might be seasonal, coastally-located stores deserving a special indicator variable we called *Summer Boost*.
- We noticed that some stores were completely closed for long stretches of time within our dataset, and hypothesized that perhaps, after a lengthy renovation, sales would increase due to a “buzz”. However, we made our *Reopened* dummy variable even more subtle than that - it applied to a certain number of days after a store reopened, proportional to the amount of time a store had been closed. Therefore, whether it was due to a huge renovation or just a long weekend which forced loyal clients to wait to restock their homes, *Reopened* would capture both possible boosts to sales.

- For the years of data after the first year of data, we added a year-to-year *Prior Daily Average* variable that was a backwards look at each particular store’s average daily sales in the prior calendar year. However, as this then limited our model to run on two-thirds of our data, it was not very popular within the team.
- Most significantly, we analyzed the dates within our dataset and added several dummy and factor-level variables to distinguish some dates from others. We combined the school calendar information with basic knowledge of the seasons to create a rather specific *Season* variable - some periods of time fall under relatively generic terms such as “Spring”, or “Fall Break”, while other dates are assigned a factor-level as specific as “Christmas Eve”.

(Please see the appendix for the complete list of all variables, both original and derived.)

Checking the Independence Assumption

As previously mentioned, Grossman’s data observations cover 942 days of sales for 1,115 stores. As a result, many of the data points are clustered by store:

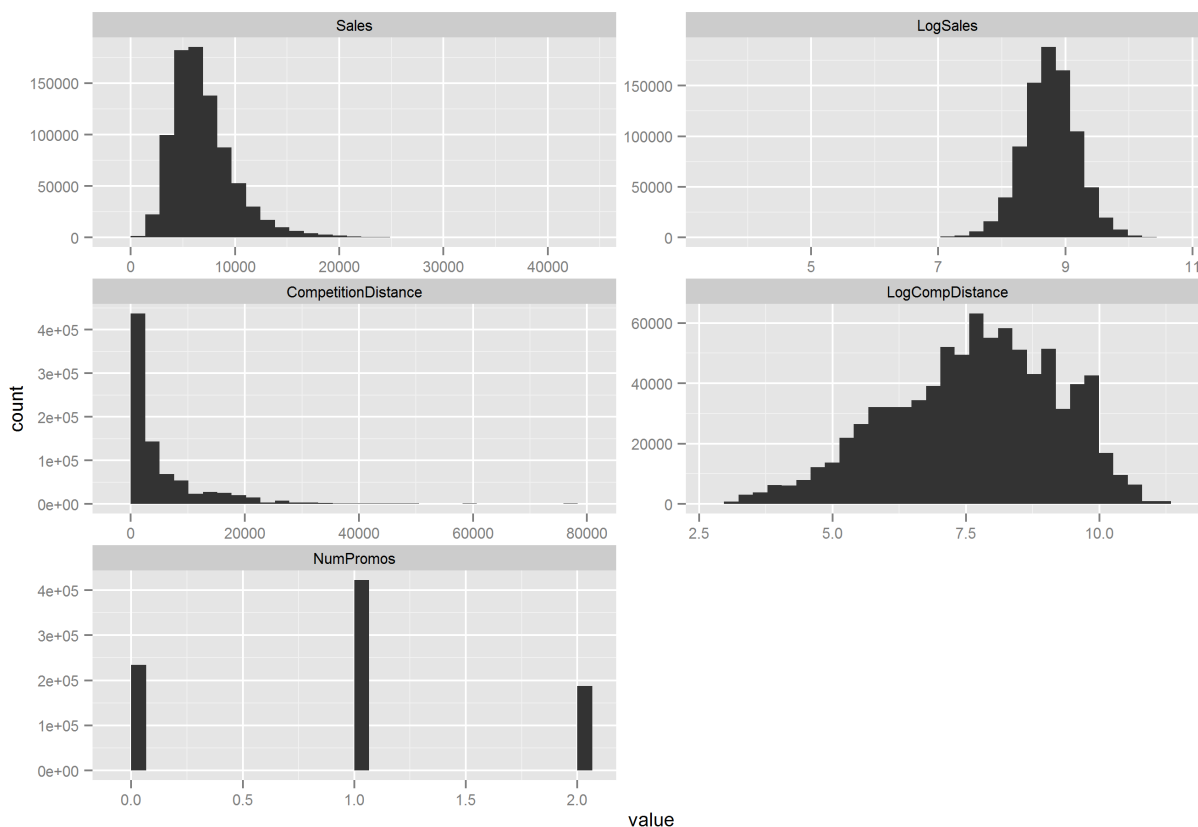


Any linear model not taking this into account would have residuals that were not independent and so be invalid. Therefore, we expanded our toolset to include a “random effects” model, as provided by the **lmer** command in R’s **lme4** package. Unlike R’s more basic **lm** approach, this model allows the predicted intercept for each store to vary, thereby “absorbing” the observed and unobserved differences between stores. In fact, it is even possible, with enough processing power, for a random effects model to allow the slopes of certain variables to vary by store as well.

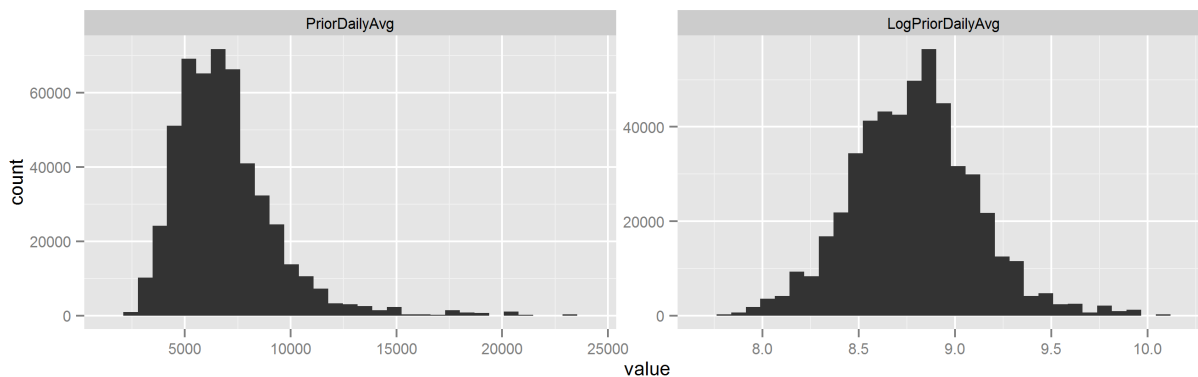
Unfortunately for us, a full, varied-slope random effects model required more processing power than any of us had at our disposal. However, we found it reasonable to hope that even the basic, intercept-only, random effects model could overcome this violation of independence within our dataset.

Checking the Normality Assumption

Our dataset contained remarkably few quantitative variables - most of the information was about various categorizations. However, those few quantitative variables still required some transformations so that our statistical tools could be accurate. *Sales*, the response variable, and *Distance from Competition* were both right-skewed until a log transformation was applied. However, the total *Number of Promotions* offered by a store at any one time was originally reasonably symmetric.

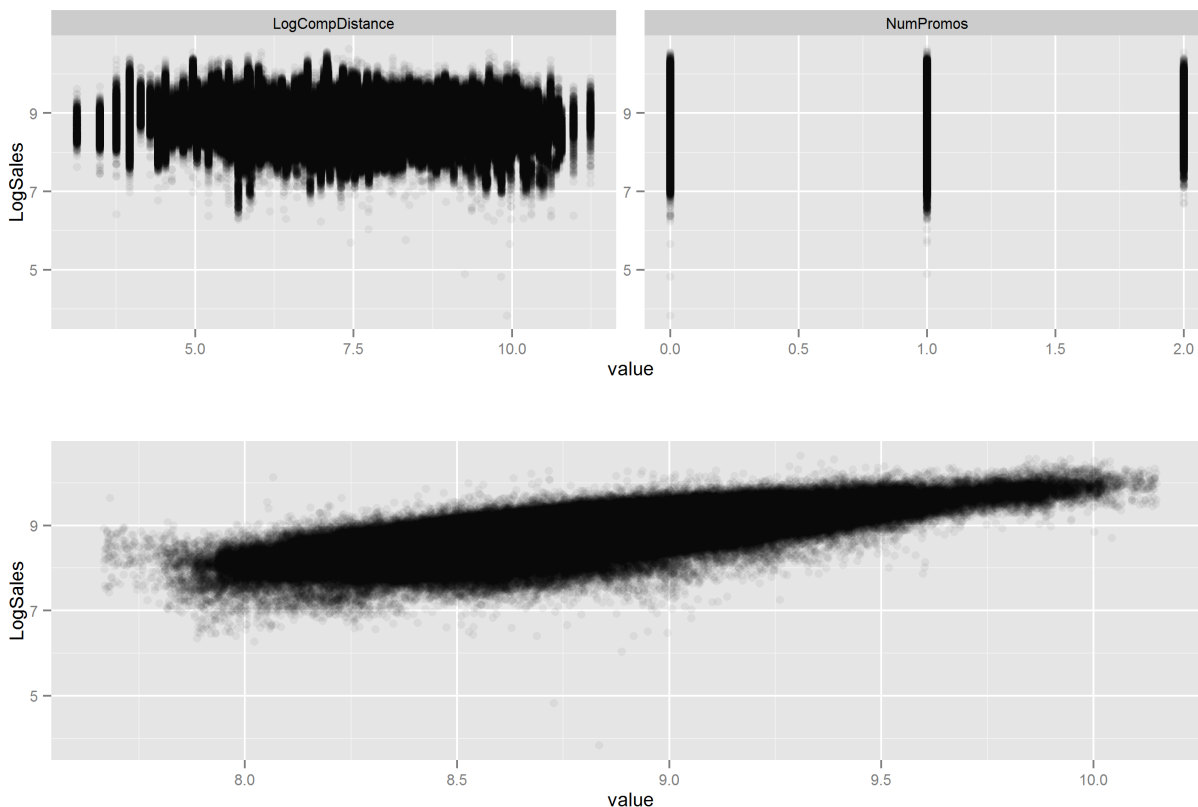


As mentioned before, our *PriorDailyAverage* variable applies only for the second year on of our data, as the first year's data becomes summary information. Therefore, we look at that subset of data to evaluate this variable's normality, and again decide to perform a log transformation:



Checking the Linearity Assumption

Next, we looked at scatterplots of our response variable, *LogSales*, against each of our quantitative predictor variables: the log of the distance to the closest competitor, the number of promotions at any one time, and the log of the prior year's average daily sales. Happily, we did not see any evidence of nonlinearity:



Imagining Interaction Terms

Even though we are not expert in the drugstore business, we knew we needed several interaction variables to allow for different slopes for certain variables under certain conditions. For example, we needed to allow the summer season to have a different magnitude of effect if a store seemed to be a seasonal summer destination, compared to all of the other stores in Germany. We needed to ensure that the distance from the closest competitor could only matter if that competitor was already open for business.

Although we imagined many more interaction terms as well, we unfortunately ran into the limits of our processing power rather quickly with these models. As a result, we made do with the two mentioned above.

Creating the Model

Process

Once we had all of our variables in place, we created the simplest **baseline model** we could imagine. Then, we created a **full model**, which was allowed to include all of our contributed variables and interaction terms as well. Lastly, we compared the two in terms of prediction and reliability.

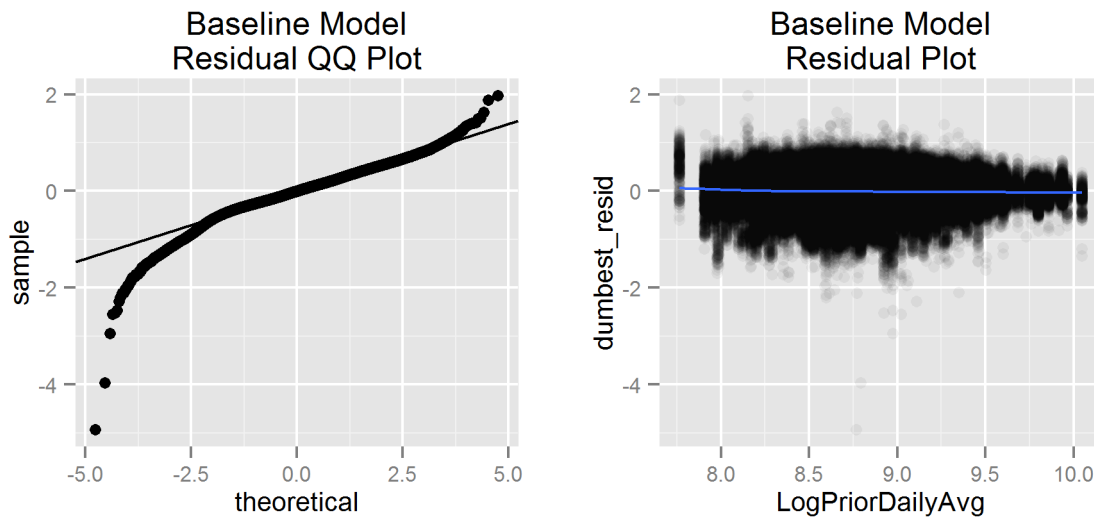
Baseline Model

The simplest baseline model we could imagine is that each store continues to have sales at the same rate as in the prior year, as based on the prior year's daily average. Given our data, of course, we deal with log values for these variables:

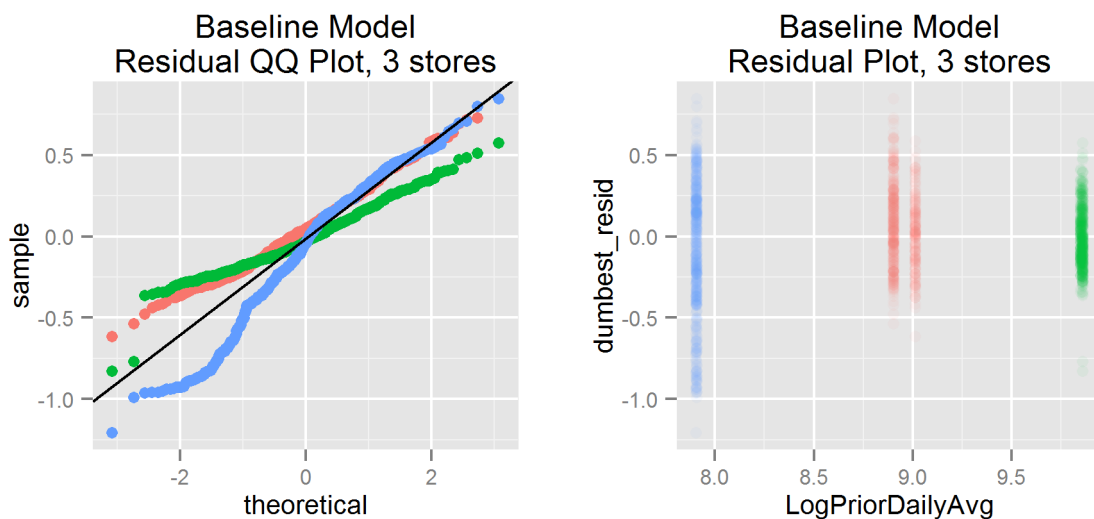
$\text{LogSales} = \beta_0 + \beta_1 \text{LogPriorDailyAverage}$, where $\beta_0 = 0$ and $\beta_1 = 1$.

When we run this model against our data for (only) the years 2014 and 2015, we find that the standard error of the residuals is 0.2789 and our residuals range from -4.9384 to 1.9728.

If our model is valid, then we should observe normality of the residuals, constant variance of the residuals, and linearity between the residuals and the fitted values, and these assumptions should hold true both overall and at the store-specific level.



The above plots show our overall residuals for the entire model. The below plots highlight store-specific residuals for three representative stores:



Overall, our residuals are somewhat right-skewed. This means that this simple model tends to underpredict daily sales at each store as based on last year's average sales – and perhaps this could be attributed to simple inflation. Looking at the residual plot, we see that our residuals do seem to show constant variance in a nicely even random cloud formation. As the cloud follows a perfectly flat line around zero, we see no evidence of non-linearity.

While the QQ Plot of the small subset of data for three specific stores shows greater departures from normality per-store, each store is different and still reasonably close to the line. The Residual Plot is interesting: each store does seem to have its own level of variance around the mean, and one store seems to tend to be underpredicted while another store tends to be overpredicted.

These assumption checks did not have perfect results. Instead, they confirmed that we should apply a store-specific random effects model to predict sales at the various Rossman stores across Germany.

Full Model

To create our full random effects model, we first identified the subset of variables which had weight in explaining sales by running a simple **lm** model on the average prior daily average (as used in our baseline model), plus many others:

- factor variables for the year, the day of the week, and our extensively-specific *Season* variable,
- factor variables about the type of store and the type of selection of goods it offers,
- indicator variables for summer seasonality, stores that were regularly open on Sundays, stores that had recently reopened after a relatively lengthy period of time,
- indicator and quantitative variables about how many promotions the store ran each day, and the type of promotion, and
- an interaction term about the existence of competition combined with the distance from that competition, and an indicator term about whether or not this store was “extra close” to its competition.

The results of our basic model helped us to weed out a few promotion-oriented variables as well as *Reopened*. The code for our final model was thus

```
final <- lmer ( LogSales ~ YearFactor + Season + DayOfWeek_Named + Interact_SummerBoost +  
                SundayOpen + StoreType + Assortment + Promo + Interact_Competitor +  
                CloseCompetitor + LogPriorDailyAvg + (1|StoreFactor), data = full )
```

which results in the following equation:

$\text{Log Sales} = 5.02 + .43 \text{ Log Prior Avg Sales} - .10 \text{ Open Sunday} + .05 \text{ Close Competitor} - .01 \text{ Competitor Distance} + .43 \text{ Summer Boost} + .34 \text{ Promo} + .01 \text{ Year}_{2015} + \dots$

<i>Store Type</i>	<i>Assortment</i>	<i>Day of Week</i>	<i>Season</i>
a: 0.00	a: 0.00	Monday: 0.11	Easter Holiday: 0.28
b: 0.44	b: -0.09	Tuesday: -0.01	Easter Break: -0.07
c: 0.01	c: 0.09	Wednesday: -0.06	Spring: -0.15
d: 0.02		Thursday: -0.06	Summer: -0.14
		Friday: 0.00	Summer Break: -0.17
		Saturday: -0.05	Fall: -0.15
		Sunday: -0.11	Fall Break: -0.20
			Holiday Shopping: 0.05
			Christmas Break: 0.00
			Last Minute Holiday Shopping: 0.54
			Christmas Eve: -0.27
			New Year's Eve: -0.45
			Public Holiday: -0.21

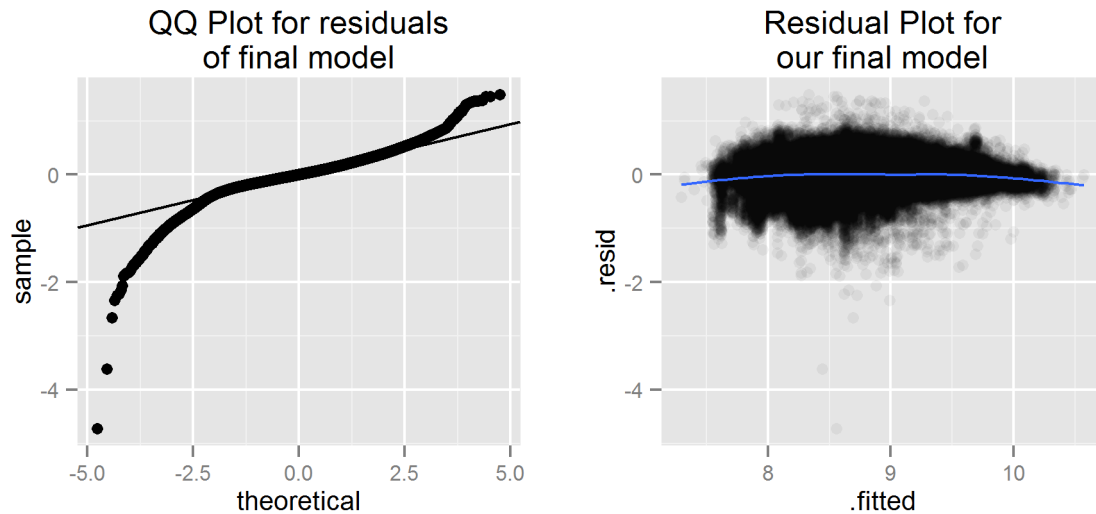
Satisfyingly, these coefficients all make intuitive sense. Some specific observations:

- It is interesting to see what a huge magnitude of an effect merely the day of the week and the season have on a store's sales - competition, the assortment of goods the store offers, and the type of store it is (most of the time) hardly matter in comparison.
- December and the resulting school vacation, apart from New Year's Eve, are extremely huge drivers of the entire year's sales. Other than that, only actual observed Easter holiday (and not the school's Easter-time break) manages to tax the store's capacity.
- Shoppers do respond to the store's primary promotion!
- Being open on Sundays does not end up benefiting the bottom line by much - if a store is open on Sundays, all of its other days are slightly depressed as an effect, and, on top of that, the Sunday itself is still relatively quiet.
- Our hypothesis that close competition might actually boost sales is held up - we had posited that clusters of similar stores can attract more shoppers, and we see that having a close competitor does indeed boost sales. The coefficient on *Competitor Distance* is slightly ambiguous - do sales go down the farther away the competitor is, or is that that sales went down a bit because this store does actually have a competitor?

(Please see the appendix for the complete coefficients table.)

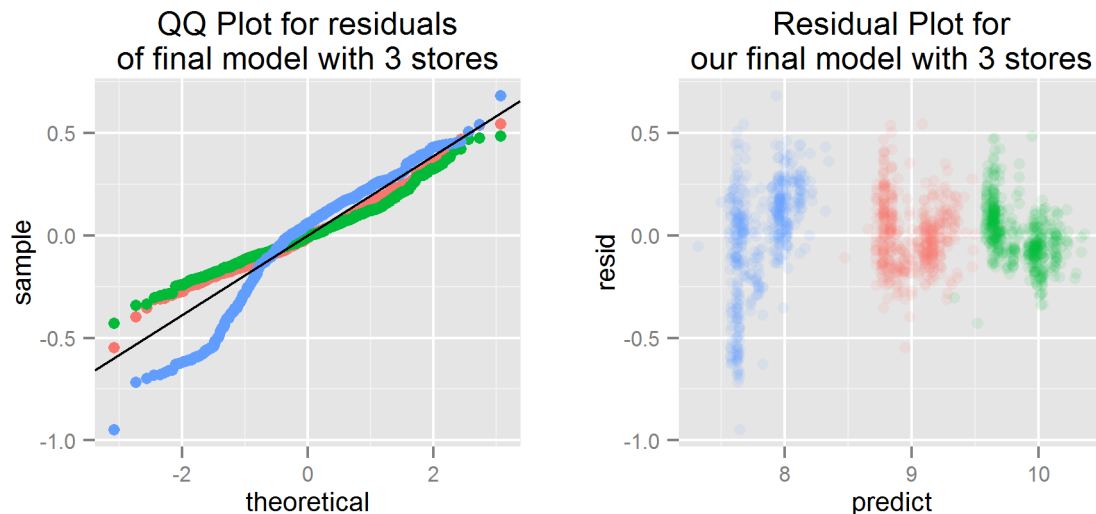
Assumptions Check

First, we will check the overall residuals of the model as a whole:



Unfortunately, in the QQ Plot, we see that our residuals are still right-skewed rather than normal. Now that we have the year factor in our model, we cannot write this off as a failure to account for inflation year-to-year. We also see a suspicious curve within the Residual Plot - while the overall shape is a nice cloud, and the line is quite close to zero, there is a slight arc to it that indicates nonlinearity. We are not confident that we can confirm our assumptions.

Checking the same three stores we had used before as a representative sample, we see a suspicious pattern:

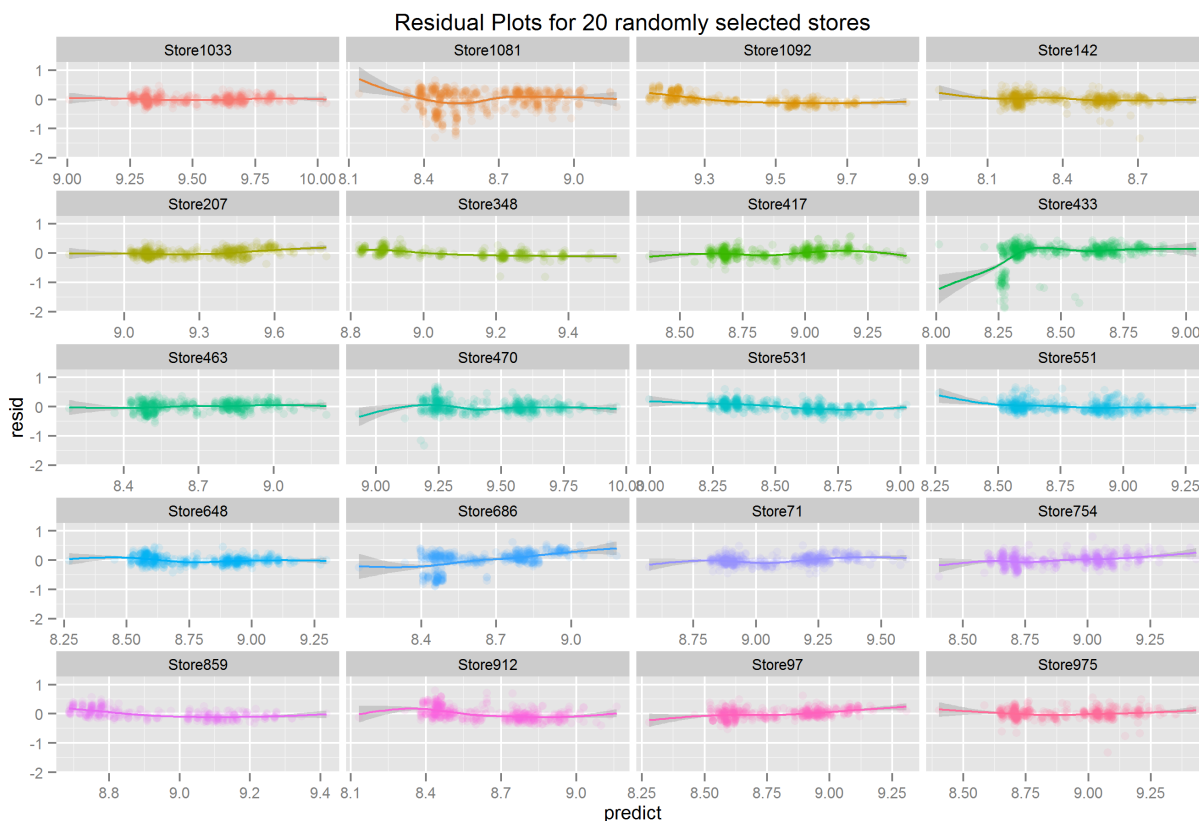


While the residuals seem to have somewhat improved normality, as compared to the baseline model, they seem highly nonlinear, as they have a per-store “U-Shape” rather than a cloud.

What lies behind this? Remembering our regression model, there are many indicator values contained within it. Sales vary per store quite dramatically by day and by season. Perhaps this is why the predicted values per store tend to group at different discrete amounts, rather than forming an even cloud. What about the

variance? For these three stores, it seems that the variance of the lower predictions is usually greater than the variance of higher predictions. Is this true for all stores?

We randomly sampled twenty more stores to see:



Happily, we see that every store is indeed different when it comes to the distribution of its residuals and that, overall, the stores' residuals are linear and with a constant variance around the zero line.

Taking the two views together, we conclude that the assumptions of linear regression seem still to hold more than not. However, we also see that a fuller random effects model could really be truly useful - some of the stores' density lines do have slight slopes up or down, and so different store-specific slopes for at least some of the variables in our model might be appropriate.

Performance as Compared to Baseline

We compared our final model to our baseline in a number of ways:

- Residual Standar Error
- R-squared
- Adjusted R-Squared
- AIC
- SSE

To evaluate the final model we compare the residual standard error (\hat{s}) of the baseline model to the final model. From the R summary of the baseline model we find $\sigma_{baseline} = XXX$ and for the final model $\sigma_{final} = XXX$. The final model has a lower σ^2 , benefiting slightly from the additional variables and interaction.

Summary

XXX

Particular Challenges

Outliers(or other appropriate name)

The baseline model has a few extreme outliers that no doubt impact our model and potentially could improve the some of the concerning patterns in our by store residual plots. Due to the size of the dataset, we were unable to investigate all outliers thoroughly.

Discussion of how we did find the Christmas Eve effect.

In fact, the large data set and number of predictors proved to be an issue for most of our inquiry. from trying to set up prediction intervals to attempting to identify which stores are being modeled well and which might need further attention, we were unable to apply many of the techniques that might work well on a smaller scale model. Simply setting up the data for use took a significant portion of the project. [other issues you all had?]

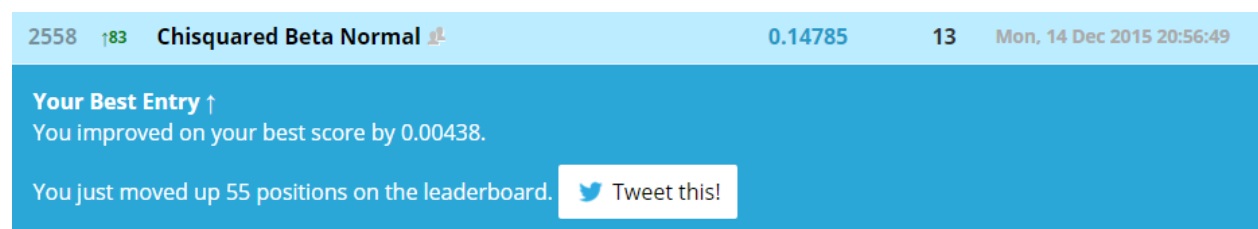
Conclusion

While we were able to explore many aspects of model building, the final model cannot actually be used to make economically-sound decisions for all Rossmann stores. One clear improvement to the model might be to let the impact of our fixed effects vary by store. For example, it is likely that the average increase in sales observed on Mondays is not the same across all stores.

In addition, assumptions violations persist in our model.

the shortcomings of the current project provide many opportunities for further inquiry.

A Fun Aside: The Kaggle Process



A screenshot of a Kaggle competition leaderboard. The top bar is light blue and contains the text '2558' with a green upward arrow and '83', 'Chisquared Beta Normal' with a small icon, '0.14785', '13', and 'Mon, 14 Dec 2015 20:56:49'. Below this is a darker blue section with the text 'Your Best Entry ↑' and 'You improved on your best score by 0.00438.' At the bottom of this section, it says 'You just moved up 55 positions on the leaderboard.' followed by a white button with a blue Twitter icon and the text 'Tweet this!'.

The Rossmann Store Sales Kaggle competition closed on Dec. 14 with around 3300 teams worldwide. Our team entered the competition when it was around two thirds done. At this point, the top teams had their models complete and were making slight adjustments to improve their predictions. Scoring is based on root mean square percentage error (RMSPE) on an unknown portion of a test dataset. And the upper half was fairly crowded with a RMSPRE range of around 0.13 to 0.09.

Before we learned about linear mixed models, our first submission was a straightforward multiple linear regression with transformed quantitative variables and scored below the Kaggle Median DayOfWeek Benchmark. At this point we knew something was wrong and learned about the linear mixed model which boosted our ranking dramatically. After adding several derived variables and interaction terms we were able to our final submission to attain an RMSPRE of 0.14785, moving our ranking up hundreds of positions to around the upper 75%:

We noted that most of the top 25% ranked teams with publicly available scripts used random forest models. With more time, we would consider using other modeling techniques to improve accuracy.

Appendix

Variables

WILL TRY AGAIN AT COPY / PASTE TO MAKE CONVERSION TO R MARKDOWN EASIER

Coefficient Table, Full Model

```
## Linear mixed model fit by REML ['lmerMod']
## Formula:
## LogSales ~ YearFactor + Season + DayOfWeek_Named + Interact_SummerBoost +
##   SundayOpen + StoreType + Assortment + Promo + Interact_Competitor +
##   CloseCompetitor + LogPriorDailyAvg + (1 | StoreFactor)
## Data: full
##
## REML criterion at convergence: -245134
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -25.064  -0.560  -0.020   0.565   7.890
##
## Random effects:
##   Groups      Name      Variance Std.Dev.
##   StoreFactor (Intercept) 0.0322   0.180
##   Residual              0.0356   0.189
## Number of obs: 506414, groups: StoreFactor, 1115
##
## Fixed effects:
##              Estimate Std. Error t value
## (Intercept)      5.022424   0.062890     80
## YearFactor2015      0.013807   0.000642     21
## SeasonChristmas_Eve -0.268753   0.006503    -41
## SeasonEaster_Break  -0.070351   0.002366    -30
## SeasonEaster_Holiday  0.277070   0.094675     3
## SeasonFall        -0.153724   0.002079    -74
## SeasonFall_Break   -0.203615   0.002809    -72
## SeasonHoliday_Shopping  0.054750   0.002400     23
## SeasonHolidayShopping_LastMin 0.535134   0.006491     82
## SeasonNYE         -0.450309   0.006503    -69
## SeasonPublic_Holiday -0.210110   0.009762    -22
## SeasonSpring       -0.147198   0.001986    -74
## SeasonSummer       -0.142050   0.002021    -70
## SeasonSummer_Break  -0.169496   0.002127    -80
## SeasonWinter       -0.200597   0.002011   -100
## DayOfWeek_NamedMon   0.106931   0.000931    115
## DayOfWeek_NamedSat  -0.050362   0.000979    -51
## DayOfWeek_NamedSun  -0.114430   0.004365    -26
## DayOfWeek_NamedThu  -0.060355   0.000935    -65
## DayOfWeek_NamedTue  -0.013627   0.000922    -15
## DayOfWeek_NamedWed  -0.056351   0.000926    -61
## Interact_SummerBoost  0.430664   0.009035     48
## SundayOpenTRUE     -0.095195   0.045775     -2
## StoreTypeb         0.441600   0.078483     6
## StoreTypec         0.008443   0.016580     1
## StoreTyped         0.017464   0.012702     1
## Assortmentb       -0.087628   0.087406    -1
## Assortmentc        0.089002   0.011308     8
## Promo             0.335284   0.000593    566
## Interact_Competitor -0.008331   0.000272   -31
## CloseCompetitorTRUE  0.052187   0.018490     3
```

```
## LogPriorDailyAvg          0.425436  0.007141    60
```

```
##
```

```
## Correlation matrix not shown by default, as p = 32 > 20.
```

```
## Use print(x, correlation=TRUE) or
```

```
##   vcov(x)      if you need it
```

Code

The following module contains the code we used to transform our data:

```
#####

# This module provides the functions we need to
#   obtain our data,
#   transform some variables
#   add derived data
#   and create a new data source file

getFullData <- function(pwd)
{
  # Read the files
  print("Reading files")
  dataStores <- read.csv(paste(pwd, "store.csv", sep=""), header=T)
  dataSales <- read.csv(paste(pwd, "train.csv", sep=""), header=T)

  # Mark which dates are reopening dates before we delete the zero data
  print("Adding reopened data")
  dataSales$Reopened <- addReopenedFlag(dataSales)

  # Only keep rows for open stores with positive sales
  dataSales <- dataSales [ dataSales$Sales > 0 & dataSales$Open != 0, ]

  # Add store-level derived data
  print("Adding store-level derived data")
  dataStores <- addStoreDerived(dataStores, dataSales)

  # Merge the data
  print("Merging the datasets")
  dataTraining <- merge ( dataSales, dataStores, by = "Store" )

  # add date-level derived data
  print("Adding date-level derived data")
  dataTraining <- addDateDerived(dataTraining, pwd)
  dataTraining$LogSales <- log(dataTraining$Sales) # we already removed 0 values

  # and more
  dataTraining$ID <- paste(dataTraining$Date, dataTraining$StoreFactor, sep="") # for graphs
  dataTraining$PriorDailyAvg <- NA
  dataTraining$PriorDailyAvg[dataTraining$YearFactor == 2014] <- dataTraining$DailyAvg_2013[dataTrain
  dataTraining$PriorDailyAvg[dataTraining$YearFactor == 2015] <- dataTraining$DailyAvg_2014[dataTrain
  dataTraining$LogPriorDailyAvg <- NA
  dataTraining$LogPriorDailyAvg[dataTraining$YearFactor == 2014] <- dataTraining$LogDailyAvg_2013[dat
  dataTraining$LogPriorDailyAvg[dataTraining$YearFactor == 2015] <- dataTraining$LogDailyAvg_2014[dat

  # export new files by given names

  print("Exporting full Stores data")
  write.csv(dataStores, paste(pwd, "dataStores_full.csv", sep=""))

  print("Exporting full Training data")
```



```

write.csv(dataTraining, paste(pwd, "dataTraining_full.csv", sep=""))

#print("Exporting pruned data")
#Pruned <- dataTraining[, c("StoreFactor", "LogSales", "Date", "YearFactor", "Month", "Season", "D
#write.csv(Pruned, paste(pwd, "dataModel_pruned_w2013.csv", sep=""))
#write.csv(Pruned[Pruned$YearFactor != 2013, ], paste(pwd, "dataModel_pruned.csv", sep=""))
print("Done!")
}

addStoreDerived <- function(dataStores, dataSales)
{
  # fix Store factor
  dataStores$StoreFactor <- paste("Store", as.factor(dataStores$Store), sep="")

  # A little data supplementation, to override NA values of competitor distance with the average
  dataStores$CompetitionDistance[dataStores$Store %in% c(291, 622, 879)] <- 5458

  # Mark if the store is open on Sundays
  dataSales$OpenSundayCheck <- dataSales$Open == 1 & dataSales$DayOfWeek == 7
  dataStores$SundayOpen <- dataStores$Store %in% unique(dataSales$Store[dataSales$OpenSundayCheck == 1])
  dataStores$SundayClosed <- abs(dataStores$SundayOpen - 1)

  # Mark which stores are in the closest 10% to their competitors
  dataStores$CloseCompetitor <- dataStores$CompetitionDistance < quantile(dataStores$CompetitionDistance, 0.9)

  # Average sales by store
  dataStores$DailyAvg_2013 = tapply ( dataSales$Sales[year(as.Date(dataSales$Date)) == 2013], dataStores$Store, FUN = function(x) {
    log(mean(x))
  })
  dataStores$LogDailyAvg_2013 <- log( dataStores$DailyAvg_2013 + 3)
  dataStores$DailyAvg_2014 = tapply ( dataSales$Sales[year(as.Date(dataSales$Date)) == 2014], dataStores$Store, FUN = function(x) {
    log(mean(x))
  })
  dataStores$LogDailyAvg_2014 <- log( dataStores$DailyAvg_2014 + 3)

  # Determine which stores are seasonal stores (high summer sales)

  # Average summer/winter sales by store
  dataSales$month = month(as.Date(dataSales$Date))
  dataSales$SummerMonthDummy <- (dataSales$month==6)|(dataSales$month==7)|(dataSales$month==8)
  dataSales$WinterMonthDummy <- (dataSales$month==12)|(dataSales$month==1)|(dataSales$month==2)
  dataStores$AvgSummerSales = tapply ( dataSales$Sales[dataSales$SummerMonthDummy==1], dataStores$Store, FUN = function(x) {
    log(mean(x))
  })
  dataStores$AvgWinterSales = tapply ( dataSales$Sales[dataSales$WinterMonthDummy==1], dataStores$Store, FUN = function(x) {
    log(mean(x))
  })
  dataStores$SeasonSalesRatio <- dataStores$AvgSummerSales/dataStores$AvgWinterSales
  dataStores$SummerBoost <- (dataStores$SeasonSalesRatio > 1.5)

  return(dataStores)
}

addDateDerived <- function(dataTraining, pwd)
{
  print("Adding quick calculations")
  dataTraining$Month = months(as.Date(dataTraining$Date))
  dataTraining$YearFactor <- as.factor(format(as.Date(dataTraining$Date), '%Y'))
  dataTraining$DayOfWeekDummy <- as.character(dataTraining$DayOfWeek)
  dataTraining$DayOfWeek_Named <- format(as.Date(dataTraining$Date), "%a")
  dataTraining$MonFri <- dataTraining$DayOfWeek_Named == "Mon" | dataTraining$DayOfWeek_Named == "Fri"
}

```

```

dataTraining$StateHolidayDummy <- dataTraining$StateHoliday != 0

print("Figuring out prior sales")
dataTraining <- addPriorSales(dataTraining, pwd)

print("Adding competition info")
dataTraining <- addCompetitionFlag(dataTraining) ##CompetitionOpen
dataTraining$CompetitionNONE <- abs(dataTraining$CompetitionOpen - 1)
dataTraining$CompetitionDistance <- dataTraining$CompetitionDistance + 3
dataTraining$LogCompDistance <- log(dataTraining$CompetitionDistance)

print("Adding promotion info")
dataTraining <- addPromo2Flag(dataTraining) ##Promo2
dataTraining$NumPromos <- dataTraining$Promo + dataTraining$Promo2

print("Adding season data")
dataTraining <- addSeason(dataTraining) # $Season

return(dataTraining)
}

addCompetitionFlag <- function(dataTraining)
{
  dataTraining$CompetitionOpen <- as.integer ( as.Date ( paste ( dataTraining$CompetitionOpenSinceYear,
    dataTraining$CompetitionOpen [ is.na ( dataTraining$CompetitionOpen ) ] <- 0

  return(dataTraining)
}

addPromo2Flag <- function(dataTraining)
{
  # Reorder factors for math to determine if we're in the promo2 month
  # Note: the factor order may be different in the test dataset

  dataTraining$PromoInterval <- factor ( dataTraining$PromoInterval, levels ( dataTraining$PromoInterval ) )
  # Flag to determine eligibility for promo 2
  dataTraining$Promo2Active <- as.integer (
    ( ( as.integer ( format ( as.Date ( dataTraining$Date ), "%Y%U" ) ) ) > ( ( dataTraining$Promo2SinceYear,
      & ( ( as.integer ( format ( as.Date ( dataTraining$Date ), "%m" ) ) ) %% 3 ) == ( as.integer (
    )
  )
  return(dataTraining)
}

addSeason <- function(dataTraining)
{
  dataTraining$Season <- ""

  # specific holiday times
  dataTraining$Season[dataTraining$SchoolHoliday == 1 & (month(as.Date(dataTraining$Date)) == 12 | month(as.Date(dataTraining$Date)) == 1) == 1
  dataTraining$Season[dataTraining$SchoolHoliday == 1 & (month(as.Date(dataTraining$Date)) == 3 | month(as.Date(dataTraining$Date)) == 4) == 1
  dataTraining$Season[dataTraining$SchoolHoliday == 1 & (month(as.Date(dataTraining$Date)) == 7 | month(as.Date(dataTraining$Date)) == 8) == 1
  dataTraining$Season[dataTraining$SchoolHoliday == 1 & month(as.Date(dataTraining$Date)) == 10 ] <- 1

  # super-specific dates

```

```

dataTraining[dataTraining$Date=="2013-12-23",]$Season="HolidayShopping_LastMin"
dataTraining[dataTraining$Date=="2014-12-23",]$Season="HolidayShopping_LastMin"
dataTraining[dataTraining$Date=="2013-12-24",]$Season="Christmas_Eve"
dataTraining[dataTraining$Date=="2014-12-24",]$Season="Christmas_Eve"
dataTraining[dataTraining$Date=="2013-12-31",]$Season="NYE"
dataTraining[dataTraining$Date=="2014-12-31",]$Season="NYE"

# state holidays
dataTraining$Season[dataTraining$Season == "" & dataTraining$StateHoliday == "a"] <- "Public_Holiday"
dataTraining$Season[dataTraining$Season == "" & dataTraining$StateHoliday == "b"] <- "Easter_Holiday"

# filling in what's left
dataTraining$Season[dataTraining$Season == "" & month(as.Date(dataTraining$Date)) == 12] <- "Holiday"
dataTraining$Season[dataTraining$Season == "" & (month(as.Date(dataTraining$Date)) == 1 | month(as.Date(dataTraining$Date)) == 2)] <- "Winter"
dataTraining$Season[dataTraining$Season == "" & (month(as.Date(dataTraining$Date)) == 3 | month(as.Date(dataTraining$Date)) == 4)] <- "Spring"
dataTraining$Season[dataTraining$Season == "" & (month(as.Date(dataTraining$Date)) == 6 | month(as.Date(dataTraining$Date)) == 7)] <- "Summer"
dataTraining$Season[dataTraining$Season == "" & (month(as.Date(dataTraining$Date)) == 9 | month(as.Date(dataTraining$Date)) == 10)] <- "Autumn"

dataTraining$SummerMonthDummy <- 0
dataTraining$SummerMonthDummy[ dataTraining$Season == "Summer" | dataTraining$Season == "Summer_Break"] <- 1
dataTraining$WinterMonthDummy <- 0
dataTraining$WinterMonthDummy[ dataTraining$Season == "Winter" | dataTraining$Season == "Christmas_Last"] <- 1

return(dataTraining)
}

addReopenedFlag <- function(dset)
{
  #print("Sorting table")
  dset <- dset[order(dset$Store, dset$Date),]

  # the vector to return
  Reopened <- rep(0, length(dset$Open))

  # make a vector of true/false according to if there is a change or not
  diffs <- dset$Open[-1L] != dset$Open[-length(dset$Open)]; #diffs

  # make a vector of the last indexes of each particular run
  idx <- c(which(diffs), length(diffs)); #idx

  # this vector counts how long each run lasted
  runs <- diff(c(1, idx)); #runs

  # find index values of the last day of each long run and length of each run
  # NOTE: IF HAVE PROBLEMS, MAYBE JUST FLAG 14 DAYS AND UP HERE RATHER THAN 2
  poss <- idx[runs > 2]; #poss
  lengths <- runs[runs > 2]; #lengths

  # check each long run to see if it is a run of closures
  numRuns <- length(poss); #numRuns
  #print("Checking runs")
  for(i in 1:numRuns)

```

```

{
  if(dset$Open[poss[i]] == 0) # this run was of closed days
  {
    # mark the first days of REOPENING as "bump"
    currStore <- dset$Store[poss[i]]

    #print(paste("Found one for Store #", currStore, "at index # ", poss[i]))

    if(lengths[i] < 5) # long weekend, maybe just one day's bump
    {
      cap <- 1
    }
    else if(lengths[i] < 31) # a month or less closed gives it several days
    {
      cap <- 5
    }
    else # after longer than a month, give the excitement two weeks
    {
      cap <- 14
    }
    # don't go past our dataset, though
    cap <- min(cap, length(dset$Open) - poss[i])
    #print(paste("For a run of", lengths[i], "cap is", cap))
    j <- 1
    while(j <= cap & dset$Store[poss[i] + j] == currStore)
    {
      #print(paste("Put a 1 in", (poss[i] + j)))
      Reopened[poss[i] + j] <- 1
      j <- j + 1
    }
  }
}
#print(paste("done. Reopened has values of"))
#print(unique(Reopened))
return(Reopened)
}

addPriorSales <- function(dset, pwd)
{
  #dset <- dataTraining

  # When necessary, we make a file of 2013 and 2014 by store, by month numbers
  if(!file.exists(paste(pwd, "data_PriorSales.csv", sep="")))
  {
    print("Making data file")
    makeSalesFile(pwd)
  }
  print("Reading in data file")
  Prior <- read.csv(paste(pwd, "data_PriorSales.csv", sep=""), header=T)
  Prior <- Prior[, -1] # drop column of row numbers

  # Then, for every month and store in the dataset we are given, we find the prior year's sales information
  dset$PriorSalesKey <- paste(dset$Store, months(as.Date(dset$Date)), (year(as.Date(dset$Date)) - 1),

```

```

dset <- merge ( Prior, dset, by = "PriorSalesKey", all.y = TRUE )
dset$PriorSales[is.na(dset$PriorSales) | is.nan(dset$PriorSales)] <- 0
dset$PriorSales <- dset$PriorSales + 1
dset$LogPriorSales <- log(dset$PriorSales)
return(dset)
}

makeSalesFile <- function(pwd)
{
  Sales <- read.csv(paste(pwd, "train.csv", sep=""), header=T)
  Sales <- Sales[year(as.Date(Sales$Date)) < 2015,]
  Sales$MonthYear <- paste(months(as.Date(Sales$Date)), year(as.Date(Sales$Date)), sep="")
  Sales <- Sales[order(Sales$Store, Sales$MonthYear),]
  SumData <- tapply(Sales$Sales, list(Sales$Store, Sales$MonthYear), sum)

  Melted <- melt(SumData)
  colnames(Melted) <- c("Store", "MonthYear", "PriorSales")
  Melted$PriorSalesKey <- paste(Melted$Store, Melted$MonthYear, sep="")

  write.csv(Melted[,c("PriorSalesKey", "PriorSales")], paste(pwd, "data_PriorSales.csv", sep=""))
}

```