Predicting Store Sales for Efficient Workforce Scheduling

BC2406, Analytics 1: Visual & Predictive Techniques Final Report

Team 5

Shaun Lim Choon Hoh, U1521806L
Lee Chong Yan, U1721687G
Low Wei Rong, U1722274G
Er Jia Chin, U1722575L

Executive Summary

Business Problem

In the context of the retail business, the effective management of ground staff through staff scheduling has a direct effect on business performance. Understaffing results in reduced employee satisfaction, reputation, quality of service and brand loyalty. Overstaffing hinders a business by increasing operational costs [3]. Thus, this report focused on optimising the staff scheduling problem, by proposing solutions based on the sales forecast. This report was tailored for *Rossman*, Germany's second largest drogerie store chain. In the drug store industry, exploiting the changing traffic pattern by staff-scheduling saves cost of up to 5%[11]. Faced with intense competition by growing drugstore chains like Dr. Max and A&D Pharma, Rossmann has to keep her operations lean and maintain her service levels to remain competitive. Therefore, we aimed to**:**

I. *Accurately predict the sales volume of Rossmann on different days*
II. *Use the prediction model to streamline the staff scheduling process*
III. *Generate additional business insights for Rossman's general operations*

Methodology & Key Results

Our methods make use of an existing dataset collected on Rossman's store sales. After data cleaning, additional features were created to incorporate the effects of trends. To predict sales, 4 models were explored. These models are Linear Regression, CART (ANOVA), RandomForest and XGBoost. Normalised Root Mean Square Error (NRMSE) was used as the measure of success. XGBoost displayed the lowest NRMSE of 0.084 on the test set while CART showed the highest NRMSE of 0.179. In the final assembly, CART and RandomForest were left out and the other models were aggregated and tested to predict sales for Store #887.

Insights and Suggestions

With the creation of the assembled model, sales forecasts can be normalised on the mean daily sales of the individual stores. These normalised values can then be categorised, according to a legend, to fixed bands of "Light", "Average" or "Heavy" sales. Managers would then have an estimate of how much manpower is needed on the specific date. This provides the basis for automatic workforce scheduling in internal enterprise systems, once the matching of sales categories to manpower required is established. Further, a system that engages the use of freelancers and real-time workforce rebalancing forms a basis for Rossmann to take staff scheduling to the next level, enabling optimal service without strain on her workers.

Probing further into the data, examining the variable importance of each model has allowed us to derive key insights into the factors affecting Rossmann's sales. The distance of competition was found to be significant, where Rossmann seems to perform more poorly in areas of higher competition. This may signify a need for Rossmann to continually differentiate herself in order to win in the Retail Market. It was also seen that customers are extremely sensitive to promotions, since sales tend to drop drastically after each promotional period. Rossmann should plan its promotion intervals and duration wisely by gauging the net increase in sales due to promotion after accounting for the decrease in sales after promotion. Lastly, some stores tend to generally do better, in terms of mean sales, than other stores that cannot be sufficiently explained by the base set of attributes. It is recommended for further studies to concentrate on geographic, weather and demographic conditions to explain these differences.

Table of Contents

1. Business Understanding

1.1 Introduction to Workforce Planning

Workforce Planning is the continuous process of aligning the needs and priorities of the business with those of its workforce, in order to meet the regulatory, legislative, service and production requirements. Strategic workforce planning focuses on budgeting and staffing on long term business goals, while operational workforce planning focuses on fulfilling the short term demands on the business with respect to talent supply. [1]

The rise of business analytics & artificial intelligence has allowed for businesses to predict disruptions and minimise impact on business operations. Once the crucial step of data gathering is done, businesses are able to leverage on data driven decision making, and improve operational forecasts for efficient daily operations. [2]

1.2 Target Audience

ACE HR seeks to expand their HR IT services to include Workforce Planning Analytics services. We therefore want to deliver a Proof-Of-Concept to ACE HR by utilizing different statistical modelling methods to solve workforce planning problems in the identified case study.

1.3 Identification of Business Opportunity

Rossmann is Germany's second largest drogerie store chain, operating over 3,000 drug stores across Europe. Currently, store managers are required to create staff schedules based on sales forecasts up to 6 weeks ahead. Effective staff schedules is an important pillar of Workforce Planning as it is critical to employee productivity and motivation, driving long term sales growth and profitability for the organisation. [3] Studies in the industry have found that matching staffing levels with changing traffic patterns can result in a 6.15% savings in lost sales and a 5.74% improvement in profitability [11]. For Rossmann, this means that customers will be serviced at the required service levels without overworking her staff, and sales can be captured in the most optimal manner.

With thousands of managers spread over their stores, the accuracy of demand forecasting and the effectiveness of creating staff schedules can be improved. Fortunately, Rossmann has regularly gathered data, including number of customers, competition, sales, school and state holidays, from some of their stalls. This enables the possible use of modelling techniques to predict sales, thereby creating more effective staffing schedules.

1.4 Consequences of Understaffing

Understaffing has negative effects on both short-term as well as long-term profitability. In the short-term, understaffing leads to lower standards of service, potentially turning customers away. Over the long-term, customers tend to avoid understaffed shops and turn to competitors instead. Additionally, unhappy customers may also express their discontent online, such as Instagram and Twitter, propagating a harmful word of mouth effect that affects a retail chain's reputation [21]. Also, understaffing leads to the overworking of store associates. This negatively impacts staff satisfaction [22] and studies have shown that the decline in staff satisfaction is correlated with a decline in a store's financial performance [23].

1.5 Consequences of Overstaffing

Similar to understaffing, overstaffing has the potential of leading to financial losses in a retail chain. In the service line, most stores run retail costs between 30-35% of total sales. Overstaffing would thus significantly affect bottom line growth [24]. Other intangible costs include bored employees, and underpaid employees when staff shifts are terminated due to the lack of customers. [24]

1.6 Project Objective & Project Flow

Given the negative consequences of improper staffing, we aim to enable Rossmann store managers to produce effective staff schedules by predicting Rossmann store sales based on the available dataset.

The dataset will first be explored, cleaned and new features will be created to facilitate the prediction process. Next, various models such as linear regression, CART, random forest and XGBoost will be used as predicted store sales. These models will be assembled into a final model. Meaningful solutions can then draw on the predicted sales information for more accurate staffing to occur.

1.7 Measures of Success

Two most popular measures of success in measuring the accuracy of continuous variable predictions are the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual the observation where all individual differences have equal weight. RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It's the square root of the average of squared differences between prediction and actual observation.

Both MAE and RMSE are indifferent to the direction of error, and are negatively oriented scores -- lower values are better. However, since RMSE squares the errors before they are averaged, RMSE gives a relatively higher weight to larger errors. Thus, RMSE is more useful when large errors are especially unwanted. In the context of the business, large errors would lead to extreme understaffing or overstaffing.

Specifically for this study, we will be using Normalised RMSE, which is derived from dividing the RMSE by the range of the measured data. This is because we prepared the training sets for different models differently, hence the predicted values derived by the different models have to be normalised to their respective range of true values in order to draw a fair comparison among all the different models.

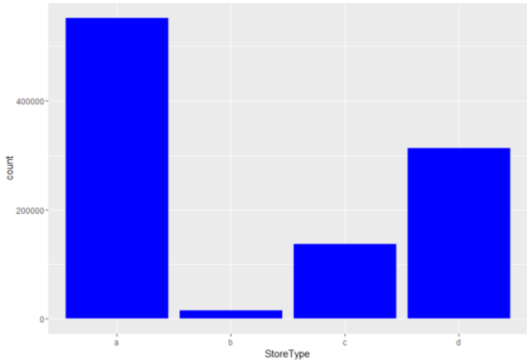
2. Data Preparation

2.1 Data Exploration

Our data source consists of 2 CSV files: *train.csv* and *store.csv*. The dataset *train.csv* and contain sales related information while *store.csv* contains the information specific to each store in the previous files. A breakdown of the data fields is illustrated in the Data Dictionary provided.

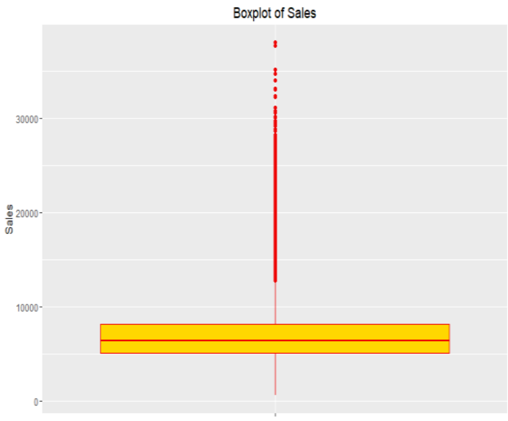
2.1.1 Store Type Distribution

The bar chart below illustrates the distribution of the store types before data trimming:



2.1.2 Checking of Negative Income

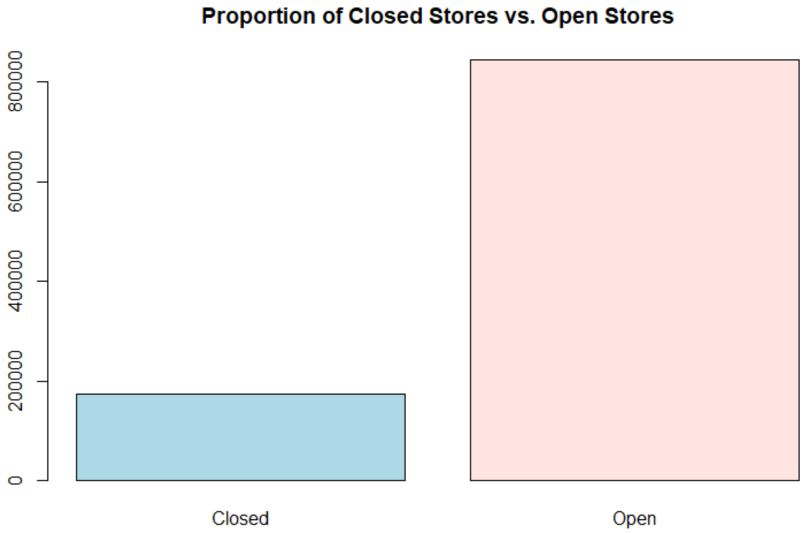
Plotting the boxplot for train.dt before cleaning and data trimming:



We confirm that there are no erroneous negative sales in the Sales column. After cleaning and data trimming, which will be shown in the later sections, we see from the boxplot below that our choosing of the 12 stores balances the data more in terms of the distribution of Sales, with less extreme values:

2.1.3 Checking of Proportion of Closed Stores

We see that the number of open stores is more than 4 times the number of closed stores before any data cleaning or trimming:



2.2 Data Cleaning

2.2.1 Merging of Data Sets

Before exploring the dataset, it was noticed that the Rossmann data set separated the store data from the sales data. Hence, we decided to merge the csv files in order to more efficiently explore and work on the data. We merged the data using the join function by the Store and used the setDT function to reset it to a data table, as the join function turns it into a dataframe.

2.2.2 Correcting Data Types

From the table, it can be seen that many data types are erroneously classified upon extraction. For example, data type of "*Promo"* should be a factor and not an integer, as it does not make sense for "*Promo*" to run between 0 and 1. Thus, the data type of certain columns was changed. Moreover, certain columns made more sense as another data type for comparison purposes. For instance, "*StoreType*" should be classified as a factor instead of a character.

2.2.3 Removal of Entries

It was found that for all data points where stores are closed, there are 0 sales. Thus, these points are removed as we are analysing to forecast sales when stores are opened.

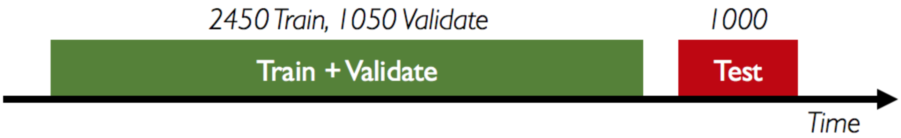
2.2.4 Data Trimming

It must first be acknowledged that there is a limitation in the number of data points set for the project at a total of 4500 lines. Initial exploration of the data reveals that there were exceedingly many (1017209) (Fig 2.2.4a) rows of data, which exceeded the limit. Hence, we decided to trim the data by only focusing on Store Type "D".

A summary of the data (Fig. 2.2.4b) revealed that even StoreType D had way too many rows (312912) and hence, we decided to further subset the data by recency (Fig. 2.2.4c) as well as storetype to ensure our predictions using the train data is up to date and relevant for the future.

Within Store Type "D", we were presented of a choice of either choosing more stores over a smaller date range or less stores over a larger date range. To achieve a balance, we have chosen to study 12 stores over the period of over a year -- this allows us to fit over a year of data into 4500 lines. To compensate for the lack of store variety, we picked the stores according to equal percentile distributions, and created new category features to be explained in Section 3.

2.3 Train-Validate-Test Split



With the limitation of 4500 lines, we first adhered to the Professor's regulation of 1000 test lines. Next, we split the train-validate set on a 70-30 ratio on a time scale.

3. Feature Creation

Upon the examination of the given data fields, it was noticed that the features provided did not best provide information that picks up trends. For example, *SchoolHoliday* indicates that there is or isn't a school holiday on that day, but it does not capture whether that day precedes a school holiday or not. This may leave out important relations to previous or future dates that also affect sales. Hence, feature engineering was carried out to create new data fields that can significantly improve and fit our models more accurately. These features are listed below:

i. Promosinceint

This feature keeps track of the number of days since the introduction of the first promotion.

```
train.dt[,Promo2sinceDay:=1,]
train.dt[Promo2==0,Promo2sinceDay:=NA,]
train.dt[,Promo2SinceDate:=paste(train.dt$Promo2sinceDay,(floor(train.dt$Promo2SinceWeek/4)+1),train.dt$Promo2SinceYear,sep="-"),]
train.dt$Promo2SinceDate<-dmy(train.dt$Promo2SinceDate)
train.dt[,Promosinceint:=(Date-Promo2SinceDate)]
```

In the initial dataset, only *Promo2SinceWeek* and *Promo2SinceYear* were available, hence we created another variable Promo2sinceDay, setting it to 1 and assume that the promotion starts on the first day of that month of that year by combining these date data into date format, *Promo2SinceDate*. The day difference can then be calculated. This feature checks if the store has a long track record of promotion. If it has a long track record of giving promotion, perhaps the effect of giving promotion now will not be very pronounced.

ii. Isweekend

This feature determines if the day falls on a weekend. If the DayOfWeek is 6 or 7, this variable will be 1 or else it will be 0.

```
train.dt[,isweekend:=ifelse(DayOfWeek==6|DayOfWeek==7,1,0),]
```

Sales can vary from store to store, depending on whether it falls on a weekend or not. Hence, it is worth picking up the sales trend if it is or is not a weekend.

iii. Isweekendyest

This feature determines if the day falls after a weekend, which is on Monday. If the DayOfWeek is 1, this indicates that the Day of Sale fell on Monday, and hence the day before it is a weekend.

```
train.dt[,isweekendyest:=ifelse(DayOfWeek==1,1,0),]
```

This variable concerns stores which will be closed on weekend, and we want to pick up the trend of people buying more as their demand is accumulated over the weekend.

iv. isweekendtmr

This feature determines if the day falls on before a weekend, which is on Friday.

```
train.dt[,isweekendtmr:=ifelse(DayOfWeek==5,1,0),]
```

If the DayOfWeek is 5, this indicates that the Day of Sale fell on Friday, and hence the day after it is a weekend.

This variable concerns more for stores which will be closed on weekend, and we want to pick up the sales trend of people stocking up and buying more since the stores will be closed tomorrow.

v. month

This feature determines which month does this sale fall on.

```
train.dt[,month:=as.numeric(format(Date,format="%m"))]
```

We introduced this variable as there may be sales trend over the month, which we would like to incorporate in our model. For example, seasonal diseases may happen on certain months, resulting in a spike in demand for drugs.

vi. year

This feature determines which year does this sale fall on.

```
train.dt[,year:=as.numeric(format(Date,format="%Y"))]
```

We introduced this variable as we find that year measures the size of demand due to consumer awareness and loyalty for the store which are accumulated over the years.

vii. Isclosetmr

This feature determines if the store is going to be closed tomorrow.

```
train.dt<-train.dt[order(Store,-Date)]
```

Before we can create this feature, we have to group our data based on store, then date in descending order. This will be crucial for the lag and lead function which will be elaborated further.

```
train.dt[,isclosetmr:=ifelse(lag(Open)==0,1,0)]
```

After that, the lag function will return the next upper observation in the train.dt, which is grouped based on descending date, hence the next upper observation is tomorrow for that store, We then evaluate if the Open variable in the next upper observation is set to 0. If it is 0, the store is indeed closed tomorrow. This variable is similar to the one tracking whether it is weekend tomorrow, whereby we want to see if the sales will increase due to people stocking up their supplies.

viii. Iscloseyest

This feature determines if the store is closed yesterday.

```
train.dt<-train.dt[order(Store,-Date)]
```

We first need to group the train.dt.

```
train.dt[,iscloseyest:=ifelse(lead(Open)==0,1,0)]
```

After that, lead function will return the next lower observation in train.dt, which is grouped based on descending date, hence the next lower observation is yesterday for that store. We then evaluate if the Open variable in the next lower observation is set to 0. If it is, the store is indeed closed yesterday. This variable is similar to the one tracking whether it is weekend yesterday, whereby we want to see if sales increases due to accumulated demand when stores are closed.

ix. isschholtmr

This feature determines if it is a school holiday tomorrow.

```
train.dt<-train.dt[order(Store,-Date)]
```

We first need to group the train.dt.

```
train.dt[,isschholtmr:=ifelse(lag(SchoolHoliday)==1,1,0)]
```

After which, lag function will return the next higher observation in the train.dt, which is grouped based on descending date, which is effectively tomorrow's observation. Here, we will evaluate if the SchoolHoliday is set to 1 tomorrow. If it is set to 1, it means that it is school holiday tomorrow.

x. isschholyest

This feature determines if it is a school holiday yesterday.

```
train.dt<-train.dt[order(Store,-Date)]
```

We first need to group the train.dt.

```
train.dt[,isschholyest:=ifelse(lead(SchoolHoliday)==1,1,0)]
```

After which, the lead function will return the next lower observation in the train.dt, which is grouped based on descending date, effectively yesterday's observation. Here, we will evaluate if SchoolHoliday is set to 1 yesterday. If it is set to 1, it means that it is school holiday yesterday.

xi. SinceLastPromo2

This feature keeps track of how many months it has been since the last continuing promotion for those participating stores. We feel it could be meaningful since customers might buy lesser when the long-running promotion is nearing, in preparation to splurge when the promotion itself happens. Creation of this feature was straightforward albeit long, by setting SinceLastPromo2 to 0 for those months where the promotion is actively happening, while the rest of the values is calculated from the current month minus the month where the most recent long promotion occurred. For those stores not participating, the value will just be NA.

xii. DaysSincePromo

DaysSincePromo is meant to be a counter for how many days it has been since a particular store has last held a day-long promotion. This is an interesting feature, because from a logical point of view, it can be quite hard to determine how the feature affects sales. Will spacing out the interval between day-long promotions boost sales by inducing anticipation into customers, or will it negatively affect sales if customers go to other drug stores instead? One thing we definitely know is that this feature will be significantly tied to predicting sales reliably for the stores. Implementing this feature was complex and required us to create a function (Fig. 3a).

It works by using a variable called value, which is the count for DaysSincePromo for that particular row. If Promo == 1 for that row, DaysSincePromo is also 0 and value is reset to 0. Else if Promo == 0, then value and thus DaysSincePromo increments row by row until the next Promo == 1. The function iterates through all rows for that particular store no. in the datatable, and is aided by the offset, a static variable that keeps track of which rows have already been iterated through. We can do it in this manner because the stores are already ordered by store and then date in ascending order right before calling the function. We then call the unique(train.dt$Store) command to identify all the unique store nos., then proceed to call the function for each one of them respectively to generate the correct DaysSincePromo values (Fig 3b).

xiii. *cat*

Due to our limitation of 4500 lines, we have chosen to select 12 stores for the study.
To obtain the 12 stores, we categorised the data into 12 divisions based on mean sales.
This corresponds to 7.7, 15.4, 23.1, 30.8, 38.5, 46.2, 53.8, 61.5, 69.2, 76.9, 84.6, 92.3 percentiles -- categories are then created for them accordingly. Therefore, the category would be an indication of how busy the store usually is over the course of the past year.

4. Modelling & Evaluation

4.1 Linear Regression Model

4.1.1 Model Introduction

Linear Regression is a basic model used for continuous data analysis. It depicts the relationship between an outcome variable, sales in this case, and one or more explanatory variables, by indicating how significantly and in what magnitude these explanatory variables impact the outcome variable, in a linear fashion [16]. There are some considerations when building the model, such as whether the explanatory variables used are significant, and if multicollinearity exists between these variables [17].

4.1.2.1 Methodology

We start off by building a base linear model in R, using almost all the features listed above in our trainset as explanatory variables for Sales, except StateHoliday and Competitionsinceint. StateHoliday is removed because it is only a single-value field in the trainset (all 0), and Competitionsinceint is not used as it causes errors with regards to factor level contrasts. This is a slight regret, but is not too significant as we still have CompetitionDistance to indicate whether a store has a rival store operating nearby or not. With that, our base linear model is as below:

```
> mlinear <- lm(Sales~DayOfWeek+Promo+CompetitionDistance+SchoolHoliday+Assortment+PromoInterval+Promosinceint+isweeken
dyest+isweekendtmr+isweekend+month+year+isclosetmr+iscloseyest+isschholtmr+isschholyest+SinceLastPromo2+cat+DaysSincePr
omo,data=trainset)
```

Following that, we use stepwise AIC (Fig 4.1.2.1a) to extract the most pertinent features for predicting Sales, and our resulting model is below (summary statistics in Fig 4.1.2.1b):

```
lm(formula = Sales ~ DayOfWeek + Promo + CompetitionDistance +
    SchoolHoliday + Assortment + PromoInterval + Promosinceint +
    month + year + isclosetmr + iscloseyest + isschholyest +
    SinceLastPromo2 + cat + DaysSincePromo, data = trainset)
```

We see all variables are significant for the model, except for SinceLastPromo2 which has p-value of 0.056. We also run vif command to check for multicollinearity between variables (Fig 4.1.2.1c). We find out that some variables such as DayOfWeek and PromoInterval have high multicollinearity with the other variables. Removing all these insignificant and high vif variables, we end up with the following linear model, **mlinear2** (summary and vif statistics in Fig 4.1.2.1d):

```
> mlinear2 <- lm(Sales~Promo+CompetitionDistance+SchoolHoliday+Assortment+month+year+isclosetmr+iscloseyest+isschholyest+cat+DaysSinc
ePromo, data = trainset)
```

We see that all variables are significant, there are no multicollinearity issues, and the model passes the F-test, with a respectable Adjusted R-squared value of 0.72.

4.1.2.2 Correlation

In order to verify that the model initially generated by AIC and trimmed by us is the best model, we generate a separate linear model using another method for feature selection: correlation. Using rcorr command to find the correlation between each pair of variables with Sales respectively, then establishing a correlation matrix (Fig. 4.1.2.2a), we select the variables with the highest correlation (>0.3) to form the **mlinear_corr** model below:

```
> mlinear_corr <- lm(Sales~Promo+PromoInterval+cat+DaysSincePromo, data=trainset)
```

All variables are significant with no multicollinearity present, verified by the summary and vif commands. Model passes the F-test, with an Adjusted R-squared of 0.67 (Fig. 4.1.2.2b).

4.1.3 Results

With our metric being RMSE, we use our mlinear2 and mlinear_corr models to then predict Sales for the train set, validation set, and the test set, and compare the results:

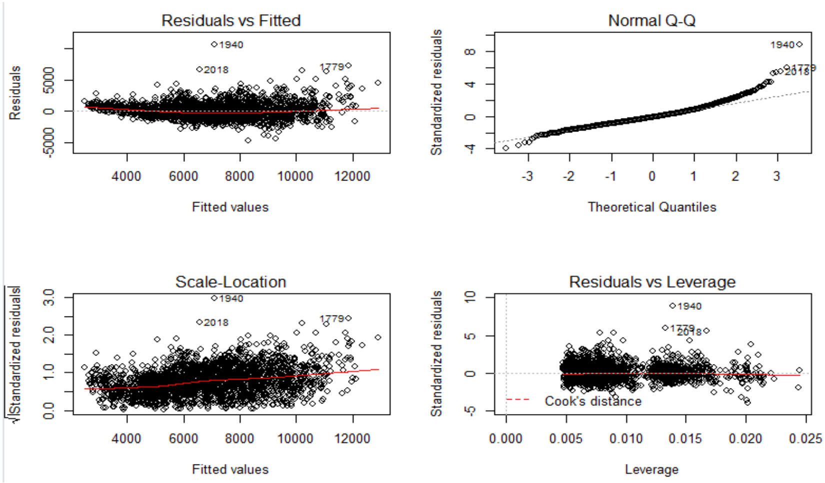
<u>mlinear2 results (Fig. 4.1.3a):</u>
Train set RMSE: 0.0741 = **7.4%** (Normalised)
Validation set RMSE: 0.0756 = **7.6%** (Normalised)
Test set RMSE: 0.1123 = **11.2%** (Normalised)

<u>mlinear_corr results (Fig. 4.1.3b):</u>
Train set RMSE: 0.0802 = **8.0%** (Normalised)
Validation set RMSE: 0.0836 = **8.4%** (Normalised)
Test set RMSE: 0.1318 = **13.2%** (Normalised)

All our results achieve a satisfactory RMSE value (< 20%), and the outcome of our test set having lower RMSE than our train and validation sets proves that both models are exemplary and not overfitted to the training set. However, as can be seen, the model generated by AIC (mlinear2) still outperforms the one generated by correlation (mlinear_corr) due to superior feature selection, so that is model we will be using whenever we refer to our linear model.

4.1.4 Discussion

Using plot command, we get the diagnostic plots for our linear regression model, shown below:



Looking at Residuals vs. Fitted (Graph 1), we can see that the points are evenly spread out across the horizontal red line without any distinct pattern, indicating that it is highly unlikely for there to be non-linear relationships [14]. This is acceptable.

For the Normal Q-Q (Graph 2), we see that while the points lie mostly on the straight line, it veers off at the very start and towards the end. This is concerning and indicates that data for our dependent variable Sales might not come from a truly Normal Distribution, as there are more extreme values than normally expected [15].

Scale-Location (Graph 3) shows residuals are mostly randomly spread, with a few outliers and slightly more spreading out towards the end. As a result, the red line is not completely horizontal but is passable, and our assumption of equal variance can still hold true for this model [14].

Residuals vs Leverage (Graph 4) has no issues as the Cook's distance lines are not visible at all, indicating our outlier cases are not as influential to the regression results. It is observed though that the outlier cases are the same in all 4 of the graphs [14].

Taking into account all of this, with the complication happening in Graph 2 holding major sway, we can conclude that while our Linear Model is reasonably successful at predicting Sales for Rossmann with a respectable Adj. R-Squared of 0.72 and RMSE values below 20%, there are definitely better models out there that can handle these extreme values better than linear regression alone.

For the variable importance, using the varImp command from caret package, we find out that the top 3 most important variables are cat, Promo1 and isclosetmr1 (Fig. 4.1.4).

4.2 CART

4.2.1 Model Introduction

CART uses a decision tree to split the dataset along certain features. Using this, we can split the data set into smaller and smaller portions and from there, derive a prediction for the value. CART accepts 2 parameters - CP (complexity parameter), which dictates the cost of growing the tree along that particular node, and depth, which is the extent to which the tree is grown. The complexity parameter in CART decides whether to prune the tree based on the complexity cost of growing the tree against the cost without.

4.2.2 Methodology

There are 4 methods we use to generate our trees.

1. Rpart with default specifications; we do not input any parameters into the formula and we use this model as a base to compare against our other models.
2. Rpart with important parameters; we use the results from (1) and using the varImp function in the caret package together with the default variable.importance name available in rpart, we select a few important variables to build our model.
3. Setting the CP to 0, we grow the tree out to its maximum and subsequently, prune using the CP that will give us the lowest xerror value.
4. Using the caret package, we train the rpart model based on its depth. We do not adjust for CP as we utilise the fact that CP value has already been tuned in model 3 to simply use that model as the base model for training.

The metric that we use to judge our model is RMSE, with the R-squared value taken into consideration in the event of extremely similar (or identical) values. Our reasoning for doing so is as discussed in Section 1. We select test RMSE over cross-validated RMSE as we are primarily interested in the predictions on the test set.

Another aspect of using rpart is the pre-processing of data (Fig. 4.2.2a). To ensure accurate comparisons across all models, we normalise all data (subtract mean and divide by SD). This also allows us to compare the different variables on the same scale and does not overweight each variable. The same is also done for the test set in order to ensure that the test and train set operate on the same scale of values. However, in the interest of space, we omit the picture.

There are missing values present in our data before any imputation or removal methods (Fig. 4.2.2b). The missing data points are primarily concentrated in 3 regions: Promosinceint, which describes the number of days since promotion started; Competitionsinceint, which describes the number of days since there was a competing store; and lastly, SinceLastPromo2, which describes the number of months since the last promotion of type 2.

We attempted K nearest neighbour (knn) imputation for the missing values (Fig. 4.2.2c). It works by extrapolating from the data points that are most similar to the missing data point[7] and from there, fills in the missing values of the data point through majority rule (for categorical values) or through the mean (for numerical values). We went with the default of k = 5 as our data set of n = 2450 is not large and we do not want to blur the local effects of certain missing data points by choosing a large k value (as locality allows us to tune our model to suit this specific data set). Although knn impute allows us to make an intelligent guess at the missing data points, a point to note here is that knn impute only works for missing at random data points.[18]

The missing data points in our data set are missing not at random - the NA values occur because there is no promotion, not because of missing data. Hence, knn impute fails as an appropriate substitute. As evidenced above, knnimpute fails to generate any values below -1.20266 (which is transformed from the original 0 in the dataset). However, this is erroneous as when there is no promotion, the level of the promotion is still greater than 1.20266 (which is 0 before normalisation). This implies that there is still a promotion and introduces bias into our model.

To address this issue, we change the SinceLastPromo2 variable into a factor and assign a level of -1 to it when there is no promotion. For the remaining 2, the same problem remains - their values should be negative when there is no competing store or promotion. In order to ensure that these missing values are weighted to reflect the fact there are no promotions/competing stores, we use the negative mean of their respective columns to replace the missing values.

The purpose is twofold - firstly, it allows us to assign a non-trivial weight to the missing value. This is especially important as the missing data points ought to be treated with equal importance as any existing data points (the lack of promotion/competition will affect the sales to a similar extent as the existence of said variable); secondly, it allows us to have a fuller data set to work with instead of merely omitting (and hence, ignoring) these data points. Ignoring these data points also leads us to a bias - that there is *always a* promotion or competitor.

4.2.2.1 Base Rpart

Using cleaned data set, we generate a rpart model and set it to be the base model which we compare our other models (Refer to Fig. 4.2.2.1a for Predict vs Actual Sales Plot for model 1).

```
#model 1 - rpart, using all variables
tree1 = rpart(Sales ~ ., data = train_tree)
predict_tree1 = predict(tree1, test_tree)
ggplot(aes(x = Sales, y = predict_tree1), data = test_tree) + geom_smooth()
summary(tree1)
```

The lowest xerror is 0.2715 and the number of splits is 11 (Fig. 4.2.2.1b). We use this as a base - ie, any changes that we make to our model has to improve it. Else, we are better off not making any adjustments to the model and sticking to the base one.

The r-squared value, which is given by 1-xerror or 1- relerror (depending on relative r-squared or from cross-validation) is also acceptable, peaking at 0.7459569 and 0.7284886227 for relative r-squared and relative r-squared from cross validation (Fig. 4.2.2.1c).

```
> rmse(predict_tree1, test.dt$Sales)/(max(test.dt$Sales) - min(test.dt$Sales))
[1] 0.1871368
```

Using our metric of RMSE, the value given by the model is 0.1871, which is already an acceptable value. Hence, going forward, our tuning has to enhance the model.

4.2.2.2 Rpart with important variables

Using the inbuilt functions in caret and rpart, we select the most important variables to keep and include in the next iteration of the model (Fig. 4.2.2.2a). From caret's documentation[4], varImp returns the sum of the reduction in the loss function at each split. From rpart's documentation[5], variable importance is a measure of its performance as a primary/surrogate splitter.

Hence, the top few variables present in both variable importance function calls are those that reduce the rmse significantly as well as appear multiple times to split the data set along important trend lines. We then use the top few variables to construct a new model, so that we filter out the less important/unimportant variables and hopefully, reduce the noise present in the data[6].

The resulting xerror of 0.2646 is lower than that of our first model and hence, it performs better as a model (Fig. 4.2.2.2b). From the graph of predicted vs actual value (Fig. 4.2.2.2c), the line has a better fit (indicating better predictions), especially towards the higher end of actual sales as compared to the baseline model. However, even though we attempted to reduce the dimensionality of the data, we did so through human selection, which might be flawed. As there is a cost associated with each variable (the complexity parameter), we could instead use that cost to make an informed selection on which feature to keep and which feature to reject.

For tree2, the RMSE and r-squared values are better than tree1. This suggests to us that the important variables explain the predicted variable sufficiently and without overfitting, especially when considered to the initial tree.

In order to further tweak the model, we tweak its hyper-parameters (CP and depth) in an attempt to make the model more robust and reduce its RMSE values.

```
> rmse(predict_tree2, test.dt$Sales)/(max(test.dt$Sales) - min(test.dt$Sales))
[1] 0.1338821
```

4.2.2.3 Rpart with pruning at min cost CP

Utilizing the ability of rpart to grow the tree to its maximum (through setting CP to 0), we create a tree with all of its branches and subsequently prune it using the CP value that produces the lowest xerror (Fig. 4.2.2.3a).

The lowest xerror is 0.194 (Fig. 4.2.2.3b), which is extremely good, especially when compared to our base model's xerror. However, there is a risk of overfitting on the training data, especially as the tree grown is extremely large - 104 splits.

Moreover, the performance of the model appears to be similar to that of tree2. This suggests that despite using CP to prune the tree, there still exists the problem of overfitting on the data set as evidenced from the number of splits which is 104, and the depth of the tree, which is 10. (Refer to Fig. 4.2.2.3c for the Predicted vs Actual Sales Plot for model 4).

After our tuning, the model appears to be performing better on the training data set, as evidenced by the apparent r-squared value (Fig. 4.2.2.3d). However, a reason to doubt its success is the huge difference between apparent r-squared and the r-squared obtained from cross-validation of the model.

This suggests that our model has overfitted on the training data, obtaining an extremely large apparent r-squared value at the cost of its actual performance. This is evidenced by the cross-validated r-square, which is only ahead of the base model by a small margin. Similarly, the normalised rmse is also only slightly better than that of the base model.

```
> rmse(predict_tree4, test.dt$Sales)/(max(test.dt$Sales) - min(test.dt$Sales))
[1] 0.1629958
```

From the plot of the tree (Fig. 4.2.2.3e), it is evident there might be an overfitting issue with the tree and the huge number of splits and depth on a small data set further suggests this. In order to counteract this issue, we attempt to use an external package to train the tree and optimise it.

4.2.2.4 Rpart with caret training, Final Results

We utilize an external package, caret, to train the model based on maxdepth. In doing so, we aim to prevent the model from overfitting on the training data set and hopefully, perform better on the test set. We fit the control parameters with method = repeatedcv (repeated cross validation) with 10 folds and repeated 10 times.

```
#model 5 - training our tree - we exploit the fact that CP is already tuned to adjust depth.
fitControl.rpart <- trainControl(method = "repeatedcv", number = 10, repeats = 10)
```

The algorithm works as such: it partitions the data set into 10 sections, with 1 section used as the validation data set. The model is then trained on the 9 sections and evaluated on the validation set. This is then repeated 10 times and the best performing model is returned. In the context of a small data set such as ours, we do not need 10 folds and in fact, 5 folds would suffice[8].

However, in order to prevent bias and variance, especially for a predictor such as decision tree that is susceptible to over-fitting on the data, we choose a large value. The model returned is at a max depth (the number of splits of the model) of 15 (Fig. 4.2.2.4a), which is significantly lesser than the 104 splits given in the previous model.

We exploit the fact that CP is already tuned in the tree before this to ignore CP tuning and instead, tune the tree for its depth. In this case, the final model returned is reasonably small and hence, not overfitted on the data set (Refer to Fig. 4.2.2.4b for final tree and depth).

The RMSE value returned here is lower than that of the base model. Furthermore, as the model has been cross-validated, we select this model as the best model to use for CART.

```
> rmse(predict_tree5, test.dt$Sales)/(max(test.dt$Sales) - min(test.dt$Sales))
[1] 0.1789976
```

The variable importance is returned to see which factors affect the tree the most (Fig 4.2.2.4c)

4.3 Random Forest

4.3.1 Model Introduction

Random forest is an ensemble technique based around the premise that a combination of weak learners can become a strong learner. A random forest is essentially a collection of highly uncorrelated trees and combined together to generate prediction values. In doing so, it reduces the probability of overfitting.

The random forest model accepts 1 tuning parameter - mtry. Mtry is the number of variables selected for tuning at each node of the individual decision tree. To illustrate this, consider an ordinary decision tree - along each node, when we consider what criteria to use to split the tree, we consider *all* the variables available to us. However, for each individual tree in the forest, we only consider *mtry* number of variables at each split.

The aim of tuning mtry is to reduce the correlation of each individual tree to every other tree. As the number of trees generated is 500 in the default case, if we were to use all the variables available to us at each individual split, the trees generated would be highly correlated to one another, leading to the forest being weaker as a predictor as a result.

4.3.2 Methodology

There are 4 methods which we use to generate our trees

1. Base randomforest, with every variable used
2. Randomforest with a random parameter search to optimise mtry value
3. Randomforest with a gridsearch of 20 values to optimise mtry value
4. Randomforest with gridsearch of 20 values but instead of k-folds cross validation, we use out of bag testing to select the validation set
5. Randomforest with tuneRF to select the optimised mtry value

Moreover, we reuse the same pre-processed data set, train.tree (Fig. 4.3.2) for our randomforest as the requirements for the data set are the same for both CART and randomforests.

4.3.2.1 Base RandomForest

This model is generated as a base model with all the variables included. The creation of this model without any prior changes implies that any subsequent changes made to the model should make the model better. Else, we should just stick with the base model.

```
#model 1 - simple randomForest
forest1 = randomForest(Sales~ ., data = train_forest, na.action = na.omit)
predict_forest1 = predict(forest1, testset)
plot(forest1)
ggplot(aes(x = Sales, y = predict_forest1), data = testset) + geom_smooth()
forest1$rsq
```

The metrics by which we judge the model are rmse and r-squared. The mse is 0.13608; the cross-validated rmse is 0.369; the actual RMSE is 0.1298 (Fig. 4.3.2.1); the r-squared with the full 500 trees is 86.39% (% variance explained).

4.3.2.2 randomforest with random parameter search

Even though randomforests are ensembles of trees, the individual trees may exhibit correlation between each other, due to the feature selection of each individual tree. As such, a random selection of features to generate each tree might serve to reduce the correlation and make the overall randomforest a better predictor[9]

```
#model 2 - using a random search to select parameters for rf
fitControl.rf <- trainControl(method="repeatedcv", number=5, repeats=3, search="random")
forest2 <- train(Sales ~ ., data = train_forest, method = "rf", tunelength = 10,
                 trControl = fitControl.rf, na.action = na.omit)
print(forest2) #takes forever to run
forest2
```

Here we utilise the ability of caret to set the parameter search as random through the search = “random” portion. This causes caret to randomly select parameter values for randomforest’s mtry value[10] (the number of parameters considered for splitting for the tree) and hopefully generate a less correlated tree. The tuneLength parameter instructs caret to select 10 default parameter values for tuning.

The optimal mtry value is 20 (Fig. 4.3.2.2a) which is significantly higher than the given mtry in the default model. However, the rmse and r-squared value are both worse than that of the default model. A possible explanation for this could be that the random selection of mtry value is not very efficient for a small data set (n = 2450), where variables could be correlated to each other, reducing the effectiveness of the random selection of mtry value. Another plausible reason might be due to noise in the data and it is possible that this would not occur in the test set.

A point to note here is that though the mtry value is randomly generated, the variable selected within each tree for splitting is still dependent on the CP value and this might generate correlated trees unintentionally.

4.3.2.3 randomforest with gridsearch to optimise mtry values, Final Results

Instead of using a random parameter search for us to optimise our randomforest model, we use a tuning grid to search through mtry values and select the one with the lowest rmse.

```
#model 3 - using a tuning grid to select parameters for rf
fitControl.rf2 <- trainControl(method="repeatedcv", number=5, repeats=3, search="grid")
tuneGrid.rf <- expand.grid(mtry = c(2,4,6,9,13,17,20))
forest3 <- train(Sales ~ ., data = train_forest, method = "rf", tuneGrid = tuneGrid.rf,
                 trainControl = fitControl.rf2, na.action = na.omit)
forest3
```

The resulting rmse value is optimised at mtry = 9, with cross-validated rmse, rmse and r-square value at 0.3790, 0.145 and 0.854 respectively (Fig. 4.3.2.3). It is important to note that these values are worse than the default randomforest implementation. However, when mtry value = 6 (same as default), the rmse and r-squared are also worse off than that of the model. A possible explanation for this could be that the base model does not use k-fold cross-validation, which is done here. Hence, although this model appears to be worse off than the base model, it could be due to the cross-validation and subsequent selection on the model.

The RMSE for this model is 0.145. This is higher than the base model's value and implies that the

```
> rmse(predict_forest3, test.dt$Sales)/(max(test.dt$Sales) - min(test.dt$Sales))
[1] 0.1452326
```

Hence, even though the model has an increased rmse on the testset, we postulate that this could be due to the rigorous cross-validation that the model is trained on.

4.3.2.4 Note: randomforest with out of bag testing

Decision trees work by taking bootstrapped sample from the training data set (ie, taking repeated samples from the data set and averaging it to get a new data sample). Hence, for each decision tree, there is some data point that is not used in the bootstrapped sample. By using the population of data points that are unused for each subsample of the random forest, we can estimate a test data set.[20]

A key point to note here is that often, the out of bag sampling will result in a lower RMSE than that of cross-validation. The reason for this is because we are testing the decision trees on data points that were not used in their construction, which simulates a test set. However, due to its extensive running time, we are not implementing it for this project.

4.3.2.5 Randomforest with tuneRF for optimisation

An alternative way to perform tuning for a random forest is to use the base tuneRF function provided in the package.

```
#model 5 - using tuneRF to adjust mtry parameter
temp_sales <- test_forest$Sales
test_forest$Sales <- NULL #done to prevent error
forest5 <- tuneRF(Sales ~ ., test_forest, mtryStart = 2, ntreeTry = 500, stepFactor = 1.5,
                  improve = 0.001, doBest = T, data = train_forest, plot = T)
str(forest5)
colnames(train_forest)
colnames(testset)
match(max(forest5$rsq), forest5$rsq) #max occurs at 478 trees
summary(forest5)
forest5$forest
#we know that the best mtry is at 6 (according to the tuning); we have to build a forest as tuneRF
#does not return a forest object (checked above from forest)
forest5 <- randomForest(Sales ~., data = train_forest, mtry = 6)
```

We first remove the sales column from our train set and store it in a temporary variable due to the constraints of the tuneRF function. We later append it back before trying to predict using the model.

We set the starting mtry value to be 2 and specify the number of trees to be 500, the same default value that we have used for our previous random forest models. We specify the mtry value to increase by 1.5 times at each tuning step and if the improvement to the out of bag error is less than 0.001 (0.1%), we will stop the tuning.

Fig 4.3.2.5a illustrates the difference in out of bag (OOB) error rate with respect to mtry variable. Evidently, the OOB error is lowest at mtry = 6 and any subsequent increase in mtry does not decrease the error but rather, increase it.

As the tuneRF function returns us a randomforest object with the randomforest$forest portion default, we pass the optimal mtry value into the randomforest function.

```
[491] 0.8623934 0.8623957 0.8624030 0.8624332 0.8624557 0.8624335 0.8624456 0.8624075 0.8624011 0.8624004
> rmse(predict_forest5, test.dt$Sales)/(max(test.dt$Sales) - min(test.dt$Sales))
[1] 0.1298228
```

The r-squared value of the forest is 0.8624 and the rmse is 0.1298228. This forest performs better than the base forest based on the out of bag error criteria but suffers on the r-squared criteria. We suspect that this might be because of the metric that the tuneRF uses to judge the best tree is out of bag error, which more closely corresponds to the actual test set rmse.

Hence, even though this model might have a lower r-squared value, we select it for its better performance on out of bag errors. From the plot of the variable importance (Fig 4.3.2.5b), we can see that the most important variables are cat, DaysSincePromo and promointerval since they have the biggest impact on node purity.

4.4 XGBoost

4.4.1 Model Introduction

XGBoost is a gradient boosting algorithm. Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It uses a gradient descent algorithm to minimize loss when adding new models.

4.4.2 Methodology

We first convert the trainset and testset into sparseMatrix in order to convert these dataset into xgb.DMatrix form (Fig. 4.4.2a).

Parameter	**Explanation**
Eta	The default value is set to 0.3. You need to specify step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative.
Gamma	The default value is set to 0. You need to specify minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm will be.
min_child_weight	The default value is set to 1. You need to specify the minimum sum of instance weight(hessian) needed in a child. The larger, the more conservative the algorithm will be.
Subsample	The default value is set to 1. You need to specify the subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees and this will prevent overfitting.
colsample_bytree	The default value is set to 1. You need to specify the subsample ratio of columns when constructing each tree.

By running 100 iterations (Fig. 4.4.2b), we randomly sample different combinations of the parameters and run xgb.cv for all of them to find the parameter combination (Fig. 4.4.2c) that gives us the smallest rmse. We then run another round of xgb.cv to find the optimal nrounds which gives us the smallest rmse (Fig. 4.4.2d). After that, we proceed in building the model based on nrounds=79.

4.4.3 Results

We normalise the rmse to the sales. The normalised rmse is 0.07281979 which is below 0.2, hence the model is acceptable (Fig. 4.4.3a). We then assess the variable importance in building this model (Fig. 4.4.3b). From here, we can see that cat, CompetitionDistance, Promo and DaysSincePromo are great predictors for sales. For cat, it is apparent that it will determine the most for the outcome given that cat is used to stratify the sales into various ranges. As for the rest of the predictors, they will determine the variations within their cat bands.

We then use this model to predict for the more recent new test data, and evaluate the rmse. The normalised rmse is still below 0.2, hence we are satisfied with this model.

```
rmse(test.dt_xgb$Sales,xgbpred2)/(max(test.dt_xgb$Sales)-min(test.dt$Sales))
1] 0.08477629
```

5. Model Selection & Assembly

5.1 Summary of Results

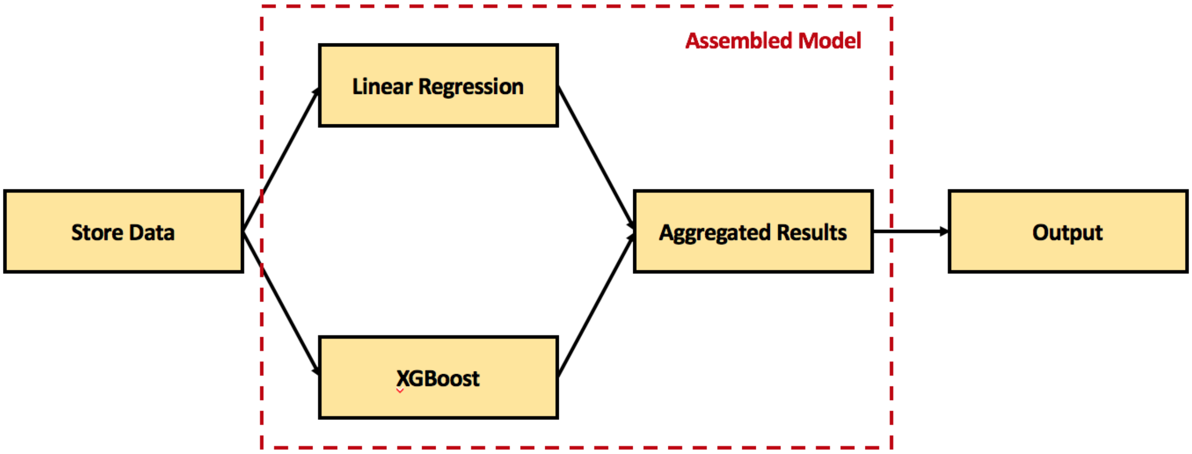
The results from section 4 of all 4 models are summarised in the table below:

Algorithm	NRMSE (Train)	NRMSE (Test)
Linear Regression (AIC)	0.07407643	0.1122801
CART	0.07835757	0.1789976
Random Forest	0.05208647	0.1298228
XGBoost	0.07281979	0.08477629

Upon experimenting with all 4 models, we set out to assemble these models to prevent overfitting of any specific model. All models satisfy our criteria of NRMSE < 0.2, however, upon closer examination, we note that both CART & RandomForest showed a sharp increase in NRMSE between the train and test sets. Test set errors are more than double that of Train set errors, indicating overfitting. Hence, only Linear Regression and XGBoost models will be used in the final model.

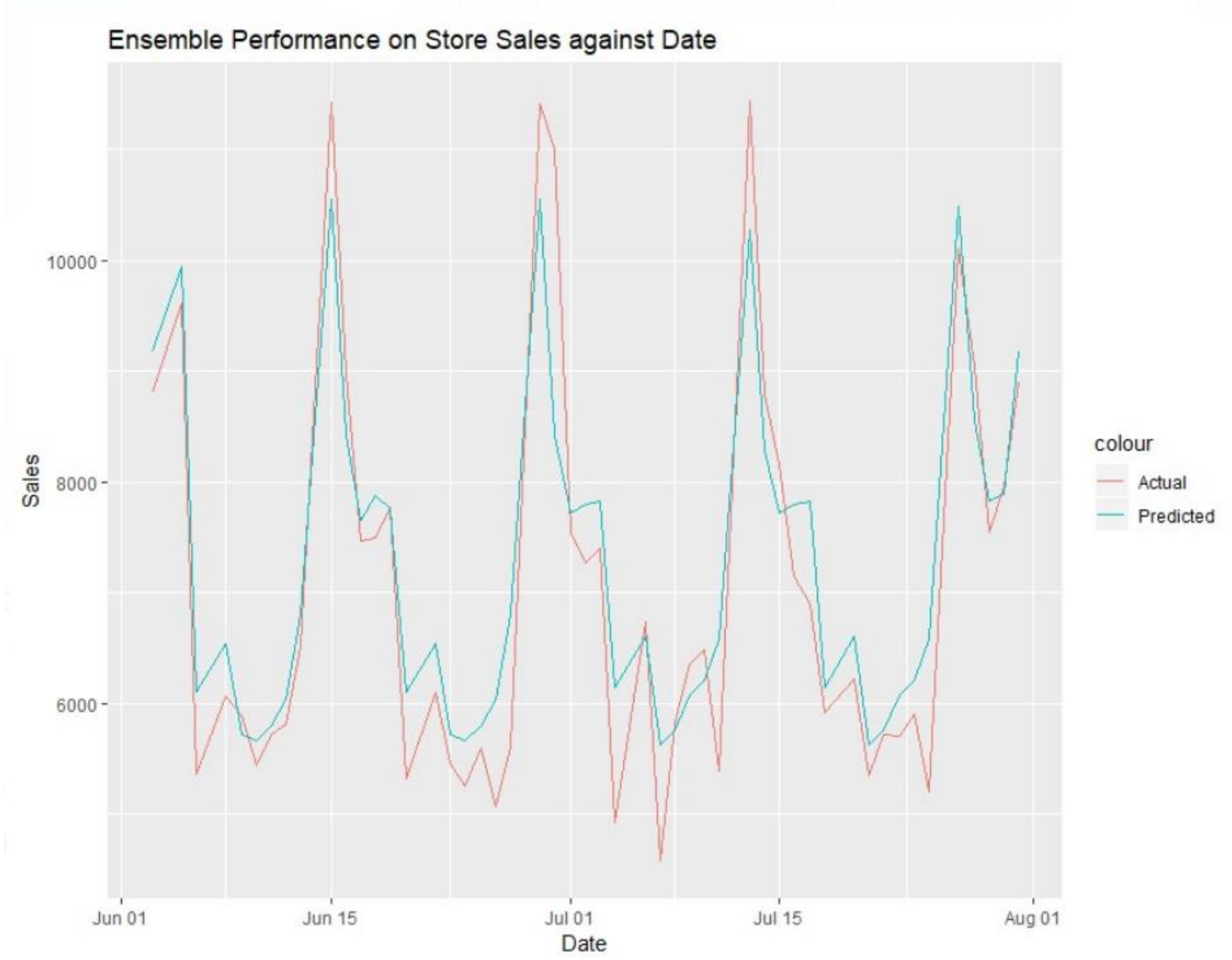
5.2 Model Assembly

For the final model assembly, store data will be fed into our final model to produce a simple average of the predictions. This is shown in the process flow diagram below:



5.3 Visualisation of Predictions

Using the model in 5.2, we generated the forecast for store #887 to better visualize the effectiveness of our assembled model.



6. Key Insights & Solutions

6.1 Significant Variables

Algorithm	Linear Regression	CART	Random Forest	XGBoost
Significant Variables	1. **cat** 2. Promo1 3. isclosetmr1	1. **DaysSincePromo** 2. Promo1 3. CompetitionDistance	1. cat 2. DaysSincePromo 3. PromoInterval	1. cat 2. DaysSincePromo 3. **CompetitionDistance**

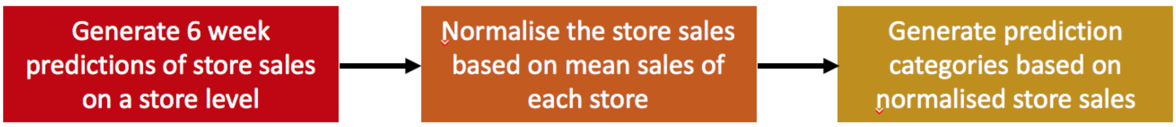
For all the models, we derive the variables which are most significant in predicting the sales. We shall look into cat, DaysSincePromo and CompetitionDistance in this report.

Variables	Business Insights
cat	The sample was stratified into categories based on their popularity. This serves as a catch-all variable, which covers the factors other than the variables available in our dataset. For example, locations and usual clientele are factors beyond the scope and not captured in our dataset. Hence, the business can look into other variables, such as customer satisfaction points on different stores and locations, etc when determining its sales. This will also be crucial in boosting their sales, as this variable shows that there are still variations in sales not explained by the existing variables.
DaysSincePromo	This shows that the number of days after the promotion ends play a significant part in determining sales. Looking at Figure 6.1.a, we see that after the promotion ends, the sales dropped immediately, which may be attributed to reduced purchase as customers brought forward their purchase to benefit from the promotion and sales only started picking up after 1 week. This shows that customers are sensitive to promotion. Hence, Rossmann should plan its promotion intervals and duration wisely by gauging the net increase in sales due to promotion after accounting for the decrease in sales after promotion. For example, having a long promotion will result in excessive brought-forward purchase, thus reducing its future income flow. Besides cost factors, companies should consider the long term detriments of having promotion too frequently, such as loss in product confidence and inuredment to promotion [26].
Competition Distance	From Fig 6.1.b, we can see that sales increases linearly with competitiondistance past a certain breakpoint. This suggests that with decreased competitiondistance, consumers have more choice of which store to shop at and this results in decreased sales for the Rossmann stores. As competitiondistance is an important predictor of sales, we suggest that Rossmann differentiate themselves to generate sales. Store differentiation allows a store to be uniquely different from its competitor and it also boosts consumer loyalty. This allows Rossmann's stores to be unsubstitutable by competition, making the distance less of a factor. Hence, we suggest that Rossmann takes steps to make its store unique and different from other competitors. A possible way to do this would be through creating a unique selling point, such as excellent customer service (which has been adopted by HaiDiLao)

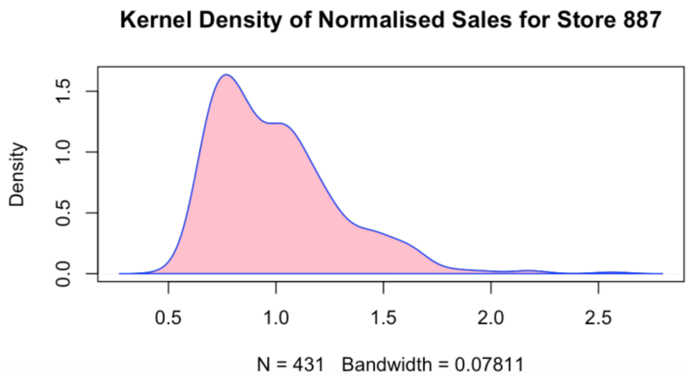
6.2 Normalisation of Sales for Staff Scheduling

The final assembled model can be used readily as a predictor of sales, providing managers an exact number of predicted sales across the weeks where staff schedules can be drafted upon.

However, in real life, it would be more feasible to classify these predictions as categorical instead of continuous. Just like how development stages of countries (first world, second world, etc), student grades (A, B, C) and road traffic predictions (Mild, Average, Heavy) are classified, the predicted sales can be normalised by the mean sales at a store level, and assigned categorical bands for store managers to plan their staff. The process can be summarised in the following flowchart:



This provides an advantage of being easy to interpret -- it is much simpler for managers to plan for more staff knowing that sales that week will be "heavy" instead of sales being forecasted at a fixed number. Based on the manager's expertise and input, a legend can then be created. An example legend of sales prediction categories is seen for store 887 below:



Sales Forecast Category	**Normalised Sales for Store #887** *Median: 6970, Min: 3534, Max: 17862*	**Number of Workers** *Based on Store Manager's Input*
Very Light	<0.60	4
Light	0.60 -- 0.80	6
Average	0.80 -- 1.20	8
Heavy	1.20 -- 1.40	10
Very Heavy	>1.40	12

6.3 Automatic Scheduling

Automated functions are rapidly becoming as qualified when it comes to logic-based tasks. In this respect, it is possible to remove the burden of creating staff schedules from the Store Manager entirely. Following the legend set out by sample Store #887 in Section 6.1, it is possible for automated systems to be set up, creating a rolling forecast based on predicted sales. Therefore, a manager would only need to manage the names of the staff required to fill that specific slot. An example of a snippet of such a schedule can be seen below:

Date	Sales Forecast	Manpower Required
12th November, Monday	**Light**	6
13th November, Tuesday	**Average**	8
14th November, Wednesday	**Store Closed**	-
15th November, Thursday	**Heavy**	10
16th November, Friday	**Average**	8

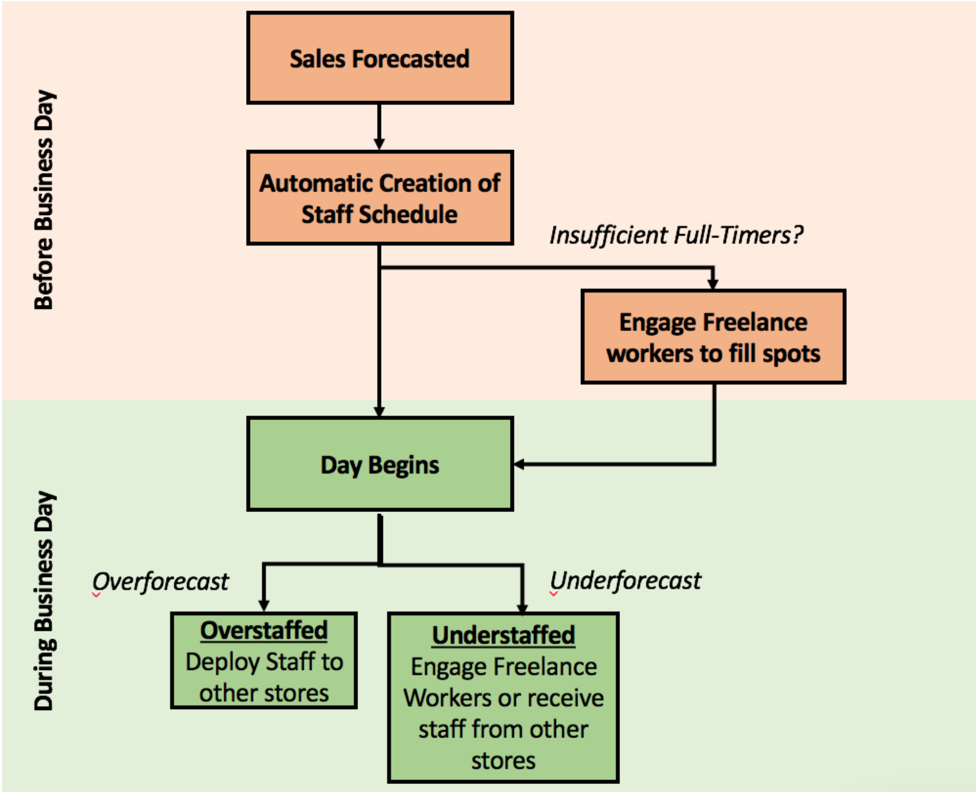
6.4 Time Sharing of Workers

While the automatic scheduling system established above would reduce the workload of the managers, it may not be entirely accurate, thus still leaving some stores overstaffed or understaffed.

One feature of recent business trends is the rise of Time Sharing features -- examples include, the use of OFO bikes for shared biking transport, and BlueSG pioneering the sharing of electric cars. Another recent trend is the rising popularity of freelance work, with the likes of Grab Drivers and neighbourhood delivery services. Building on the ideas of 6.2, it is thus possible for Rossmann to establish a system that enables the time-sharing of workers to further reduce the incorrect staffing issue.

In this integrated solution, the system would first recommend the required manpower based off the sales predictions. The store manager will then plan to fill this business need with the full-time employee base. Based on real time needs, any gaps or excesses in manpower can then be resolved by (i) re-deploying existing manpower and (ii) engaging freelancers. The redeployment of existing staff can be done through internal systems, while the engagement of freelancers can be done through popular external freelance applications, such as *FreeLancer* and *Fiverr.*

This process is illustrated in the following diagram:



6.4 Study Limitations

There are several key limitations of the study, noted below:

1) Due to limitations in processing power, only a total of 4,500 data points were used for the train test split. Therefore, a tradeoff had to be made -- we could either sample more stores over a shorter period of time, or less stores for a longer period of time. To capture important seasonal effects, we have chosen to select 12 stores to study data over an entire year. However, this has caused us to miss out a large number of stores. With a higher processing capability, we will be better able to train our models based on information on all stores, across the entire timeline. This would allow us to dive deeper in investigating the effects of each variable on each store.
2) While we attempted to create features to complement the base set, the presence of other geographical or demographic information may prove helpful. For example, it would be interesting to investigate the effect of weather conditions on store sales, thus aiding in the staff scheduling process.
3) The discussion on categorical Y was limited. In the future, studies can be done to investigate the use of logistic regression and decision trees for the various categories of normalised sales. This may yield more accurate results.
4) During model assembly, a simple average of all models are taken as the final result to prevent overfitting. This decision was taken due to our lack of expertise in this area. In fact, we have taken a further step to check out the predictions on store #887 purely with XGBoost, which seemed to yield better results in Figure 6.4a. While this is just a small subsample, it is an indication that more work can be done in this area. Further studies can be done to include more complex integration algorithms, such as stacked generalisation, bootstrap aggregating, bucket of models and boosting.

7. Conclusion

In conclusion, of all the models tested, XGBoost returned the lowest NRMSE. To prevent overfitting, the models were assembled and used to predicted 4 weeks of data of Store #887. With this, we have produced a scalable proof of concept for Rossmann's sales forecast in her retail stores. With regards to the main business problem, staff scheduling, the predicted sales forecast can be translated into categorical bands, which can then be used for workforce scheduling. We finally recommend further solutions of automatic scheduling and time sharing as part of Rossmann's long term workforce planning solution, and proposed points of further study to address the limitations of this report.

8.References

[1] Sloan, Julie. *The Workforce Planning Imperative* JSM, 2010. ISBN 978-1-921037-37-5 (pbk.)

[2] Clapon, P. and Clapon, P. (2018). *The Role Of HR Analytics In Workforce Planning | Hppy*. [online] Hppy. Available at: https://gethppy.com/talent-management/hr-analytics-workforce-planning

[3] Terpstra, D. E. and Rozell, E. J. (1993), The Relationship Of Staffing Practices To Organizational Level Measures Of Performance. Personnel Psychology, 46: 27-48. doi:10.1111/j.1744-6570.1993.tb00866.x

[4] Kuhn, M. (2018). *The caret Package*. [online] Topepo.github.io. Available at: https://topepo.github.io/caret/variable-importance.html

[5] Terry. M. Therneau. (2018 February). *An Introduction to Recursive Partitioning Using RPart Routines.* Available: https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

[6] Lei Yu, (2013 December). *Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution*. Available: http://www.aaai.org/Papers/ICML/2003/ICML03-111.pdf

[7] E. Acuna. (2002 December). *The Treatment of Missing Values and its Effect in the Classifier Accuracy*. Available: https://pdfs.semanticscholar.org/4172/f558219b94f850c6567f93fa60dee7e65139.pdf

[8] T. Fushiki. (2009 September). *Estimation of prediction error by using K-fold cross-validation*. Available: https://link-springer-com.ezlibproxy1.ntu.edu.sg/content/pdf/10.1007%2Fs11222-009-9153-8.pdf

[9] L. Breinman. (2001 January) *Random Forests*. Available: https://link-springer-com.ezlibproxy1.ntu.edu.sg/content/pdf/10.1023%2FA%3A1010933404324.pdf

[10] Github. (2017 Jan). *Random Hyperparameter Search.* Available: https://topepo.github.io/caret/random-hyperparameter-search.html

[11] V. Mani. (2010 May). *Estimating the impact of understaffing on sales and profitability in retail stores.* Available: https://public.kenan-flagler.unc.edu/faculty/kesavans/understaffing.pdf

[12] The Minitab Blog. (2013 May). *Regression Analysis: How Do I Interpret R-squared and Assess the Goodness-of-Fit?.*
Available: http://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit

[13] Human in a Machine World. (2016 May). *MAE and RMSE — Which Metric is Better? Available:* https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d

[14] University of Virginia Library. (2007 January). *Understanding Diagnostic Plots for Linear Regression Analysis.* Available:
https://data.library.virginia.edu/diagnostic-plots/

[15] University of Virginia Library. (2007 January). *Understanding Q-Q Plots.* Available: https://data.library.virginia.edu/understanding-q-q-plots/

[16] Statistics Solutions. (2017 April). *Linear Regression.* Available: https://www.statisticssolutions.com/what-is-linear-regression/

[17] The Minitab Blog. (2013 May). *Handling Multicollinearity in Regression Analysis.*
https://blog.minitab.com/blog/understanding-statistics/handling-multicollinearity-in-regression-analysis

[18] Columbia University. (2009 January). *Missing-data imputation.* Available: http://www.stat.columbia.edu/~gelman/arm/missing.pdf

[19] M. T. Grotenheius. (2008 July). *Dummy variables and their interactions in regression analysis: examples from research on body mass index.* Available: https://arxiv.org/ftp/arxiv/papers/1511/1511.05728.pdf

[20] Standford Univeristy. (2005 June). *Tree-based Methods.* Available: https://lagunita.stanford.edu/c4x/HumanitiesScience/StatLearning/asset/trees.pdf

[21] Park, Y, C.H. Park, V. Gaur. 2010. Consumer Learning, Word of Mouth, and Quality Competition. Working Paper, The Johnson School, Cornell University.

[22] Heskett, J.L., T.O. Jones, G.W. Loveman, W.E. Jr Sasser, L.A. Schlesinger. 1994. Putting the service profit chain to work. Harvard Business Review. 72(2), 164-174.
[23] Maxham, J.G. III, R.G. Netemeyer, D.R. Lichenstein. 2008. The Retail Value Chain: Linking Employee Perceptions to Employee Performance, Customer Evaluations, and Store Performance. Marketing Science. 27(2), 147 – 167.
[24] J. Day. (2017 March). *The Hidden Consequences of Under or Over Staffing.* Available: https://ctuit.compeat.com/the-hidden-consequences-of-under-or-over-staffing/
[25] CIRP. (2014 June). *Dimensional Statistics.* Available: (https://cirpwiki.info/wiki/Statistics#Normalization
[26] L. Ray. (2018 January). *The Long-Term Effects of Customer-Oriented Sales Promotions.* Available: https://smallbusiness.chron.com/longterm-effects-customeroriented-sales-promotions-21921.html

9. Appendix

Figure 2.2.1

```
#Merge Data
train.dt <- left_join(train.dt,store.dt,by="Store")
setDT(train.dt)
```

Figure 2.2.2

```
#type conversion to make the data easier to work with
train.dt$Promo = as.logical(train.dt$Promo)
train.dt$Open = as.logical(train.dt$Open)
train.dt$DayOfWeek = as.factor(train.dt$DayOfWeek)
train.dt$Store = as.integer(train.dt$Store)

store.dt$Promo2 = as.factor(store.dt$Promo2)
store.dt$StoreType = as.factor(store.dt$StoreType)
store.dt$Assortment = as.factor(store.dt$Assortment)
```

Figure 2.2.4a

```
> train.dt[,.N]
[1] 1017209
```

Figure 2.2.4b

```
StoreType
a:551627
b: 15830
c:136840
d:312912
```

Figure 2.2.4c

```
#extract first 3500 rows with store type == b
train.dt = train.dt[StoreType == "b" & rev(order(Date))]
train.dt = train.dt[1:3500]

#subset into train/test via date
test.dt = train.dt[2501:3500]
train.dt = train.dt[1:2500]
```

Figure 3a

```
train.dt<-train.dt[order(Store,Date)]

f <- function(datatable,storeno,rowoffset){
  value <- 0
  counter <- datatable[Store == storeno, .N]
  for(i in 1:counter){
    if(train.dt[i+rowoffset,Promo]==1){
      value <- 0
    }
    else{
      train.dt[i+rowoffset, DaysSincePromo:=value+1]
      value <- value + 1
    }
    offset <<- offset + 1
  }
}
```

Figure 3b

```
> unique(train.dt$Store)
 [1] 185 202 298 334 495 547 584 620 637 739 887 922
> offset = 0
> f(train.dt,185,0)
> f(train.dt,202,offset)
> f(train.dt,298,offset)
> f(train.dt,334,offset)
> f(train.dt,495,offset)
> f(train.dt,547,offset)
> f(train.dt,584,offset)
> f(train.dt,620,offset)
> f(train.dt,637,offset)
> f(train.dt,739,offset)
> f(train.dt,887,offset)
> f(train.dt,922,offset)
```

Figure 4.1.2.1a

```
> mlinear <- step(mlinear)
Step:  AIC=27973.63
Sales ~ DayOfWeek + Promo + CompetitionDistance + SchoolHoliday +
    Assortment + PromoInterval + Promosinceint + month + year +
    isclosetmr + iscloseyest + isschholyest + SinceLastPromo2 +
    cat + DaysSincePromo

                      Df  Sum of Sq        RSS   AIC
<none>                               2143844833 27974
- Assortment           1    2616127 2146460960 27974
- SinceLastPromo2      1    3973848 2147818681 27975
- year                 1    5927476 2149772310 27977
- isschholyest         1   12834404 2156679237 27984
- SchoolHoliday        1   14053841 2157898674 27985
- DaysSincePromo       1   16700538 2160545372 27987
- DayOfWeek            5   95293484 2239138318 28051
- isclosetmr           1   89884858 2233729692 28054
- Promosinceint        1  105139965 2248984798 28068
- CompetitionDistance  1  134315316 2278160149 28094
- month               11  184551128 2328395961 28118
- iscloseyest          1  171237678 2315082511 28126
- PromoInterval        2  259593044 2403437878 28200
- Promo                1  949750736 3093595570 28709
- cat                  1 1343125249 3486970083 28950
```

Figure 4.1.2.1b

```
> summary(mlinear)

Call:
lm(formula = Sales ~ DayOfWeek + Promo + CompetitionDistance +
    SchoolHoliday + Assortment + PromoInterval + Promosinceint +
    month + year + isclosetmr + iscloseyest + isschholyest +
    SinceLastPromo2 + cat + DaysSincePromo, data = trainset)

Residuals:
    Min      1Q  Median      3Q     Max
-4552.8  -595.3   -96.3   518.6 10190.0

Coefficients:
                                Estimate Std. Error t value Pr(>|t|)
(Intercept)                    8.778e+03  3.142e+02  27.936  < 2e-16 ***
DayOfWeek2                     5.103e+02  1.440e+02   3.545 0.000402 ***
DayOfWeek3                     1.944e+02  1.495e+02   1.300 0.193827
DayOfWeek4                     4.160e+02  1.504e+02   2.767 0.005712 **
DayOfWeek5                     8.519e+02  1.402e+02   6.074 1.49e-09 ***
DayOfWeek6                     3.019e+02  1.984e+02   1.522 0.128132
Promo1                         2.491e+03  8.411e+01  29.617  < 2e-16 ***
CompetitionDistance           -5.562e-02  4.994e-03 -11.138  < 2e-16 ***
SchoolHoliday1                 3.852e+02  1.069e+02   3.603 0.000323 ***
Assortmentc                    1.562e+02  1.005e+02   1.554 0.120247
PromoIntervalJan,Apr,Jul,Oct  -1.676e+03  1.215e+02 -13.788  < 2e-16 ***
PromoIntervalMar,Jun,Sept,Dec -1.589e+03  1.083e+02 -14.670  < 2e-16 ***
Promosinceint                  6.045e-01  6.134e-02   9.854  < 2e-16 ***
month2                         1.136e+02  1.174e+02   0.967 0.333530
month3                         2.025e+02  1.135e+02   1.785 0.074453 .
month4                         6.276e+02  1.134e+02   5.534 3.54e-08 ***
month5                         5.092e+02  1.156e+02   4.405 1.12e-05 ***
month6                         5.853e+02  1.136e+02   5.151 2.86e-07 ***
month7                         6.037e+02  1.149e+02   5.255 1.64e-07 ***
month8                         3.505e+02  1.754e+02   1.999 0.045756 *
month9                         2.845e+02  1.658e+02   1.716 0.086322 .
month10                        2.212e+02  1.609e+02   1.375 0.169212
month11                        5.408e+02  1.690e+02   3.201 0.001394 **
month12                        1.656e+03  1.644e+02  10.075  < 2e-16 ***
year2015                       1.387e+02  5.926e+01   2.340 0.019395 *
isclosetmr1                    1.237e+03  1.357e+02   9.111  < 2e-16 ***
iscloseyest1                   1.580e+03  1.256e+02  12.576  < 2e-16 ***
isschholyest1                 -3.636e+02  1.056e+02  -3.443 0.000587 ***
SinceLastPromo2                5.843e+01  3.050e+01   1.916 0.055539 .
cat                           -4.937e+02  1.402e+01 -35.220  < 2e-16 ***
DaysSincePromo                 6.620e+01  1.686e+01   3.927 8.88e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1041 on 1980 degrees of freedom
  (439 observations deleted due to missingness)
Multiple R-squared:  0.8127,	Adjusted R-squared:  0.8098
F-statistic: 286.4 on 30 and 1980 DF,  p-value: < 2.2e-16
```

Figure 4.1.2.1c

```
> vif(mlinear)
                         GVIF Df GVIF^(1/(2*Df))
DayOfWeek           29.784694  5        1.404104
Promo                3.262961  1        1.806367
CompetitionDistance  1.521504  1        1.233493
SchoolHoliday        3.099604  1        1.760569
Assortment           4.649124  1        2.156183
PromoInterval        6.107577  2        1.572053
Promosinceint        2.895976  1        1.701757
month                2.504026 11        1.042605
year                 1.628874  1        1.276273
isclosetmr           5.494318  1        2.343996
iscloseyest          4.634824  1        2.152864
isschholyest         3.123752  1        1.767414
SinceLastPromo2      1.169163  1        1.081278
cat                  4.561778  1        2.135832
DaysSincePromo       3.127433  1        1.768455
```

Figure 4.1.2.1d

```
> summary(mlinear2)

Call:
lm(formula = Sales ~ Promo + CompetitionDistance + SchoolHoliday +
    Assortment + month + year + isclosetmr + iscloseyest + isschholyest +
    cat + DaysSincePromo, data = trainset)

Residuals:
    Min      1Q  Median      3Q     Max
-4759.1  -769.2  -108.0   603.0 10746.7

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)          6.891e+03  1.556e+02  44.292  < 2e-16 ***
Promo1               2.514e+03  8.458e+01  29.721  < 2e-16 ***
CompetitionDistance  3.275e-03  4.208e-03   0.778 0.436563
SchoolHoliday1       4.456e+02  1.105e+02   4.033 5.67e-05 ***
Assortmentc          5.992e+02  5.295e+01  11.316  < 2e-16 ***
month2               1.757e+02  1.208e+02   1.455 0.145759
month3               2.361e+02  1.165e+02   2.026 0.042901 *
month4               6.633e+02  1.182e+02   5.612 2.23e-08 ***
month5               6.752e+02  1.185e+02   5.697 1.36e-08 ***
month6               7.194e+02  1.170e+02   6.149 9.10e-10 ***
month7               5.868e+02  1.193e+02   4.918 9.35e-07 ***
month8               2.217e+02  1.751e+02   1.266 0.205612
month9               1.740e+02  1.718e+02   1.013 0.311129
month10              1.424e+02  1.676e+02   0.850 0.395403
month11              6.186e+02  1.741e+02   3.553 0.000388 ***
month12              1.651e+03  1.689e+02   9.775  < 2e-16 ***
year2015             3.155e+02  5.848e+01   5.395 7.50e-08 ***
isclosetmr1          1.292e+03  6.810e+01  18.969  < 2e-16 ***
iscloseyest1         1.229e+03  6.642e+01  18.504  < 2e-16 ***
isschholyest1       -3.945e+02  1.095e+02  -3.602 0.000322 ***
cat                 -3.893e+02  7.209e+00 -54.000  < 2e-16 ***
DaysSincePromo       8.492e+01  1.696e+01   5.008 5.89e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1216 on 2428 degrees of freedom
Multiple R-squared:  0.7241,	Adjusted R-squared:  0.7217
F-statistic: 303.4 on 21 and 2428 DF,  p-value: < 2.2e-16

> vif(mlinear2)
                        GVIF Df GVIF^(1/(2*Df))
Promo               2.942029  1        1.715234
CompetitionDistance 1.133489  1        1.064655
SchoolHoliday       2.902108  1        1.703557
Assortment          1.150267  1        1.072505
month               1.924259 11        1.030199
year                1.417473  1        1.190577
isclosetmr          1.241490  1        1.114222
iscloseyest         1.156079  1        1.075211
isschholyest        2.975979  1        1.725103
cat                 1.154646  1        1.074544
DaysSincePromo      2.830285  1        1.682345
```

Figure 4.1.2.2a

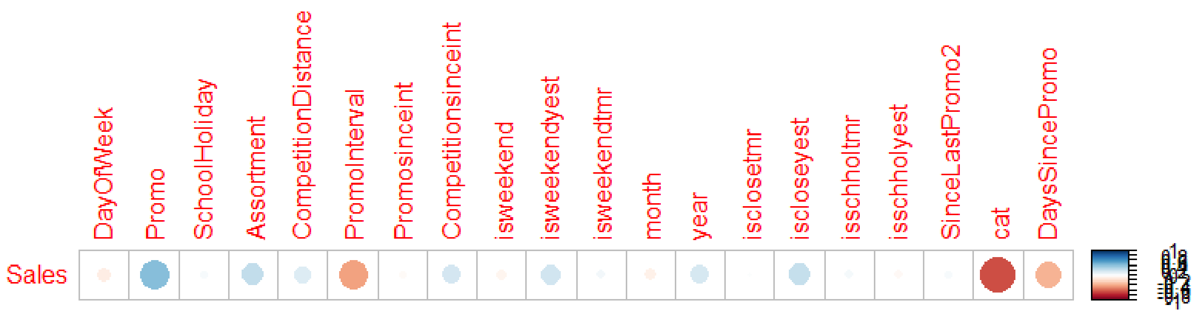


Figure 4.1.2.2b

```
> summary(mlinear_corr)

Call:
lm(formula = Sales ~ Promo + PromoInterval + cat + DaysSincePromo,
    data = trainset)

Residuals:
    Min      1Q  Median      3Q     Max
-3640.2  -858.2  -170.3   634.7 11457.9

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                 8473.348    100.962  83.926  < 2e-16 ***
Promo1                      1942.509     86.013  22.584  < 2e-16 ***
PromoIntervalFeb,May,Aug,Nov 2620.843   118.206  22.172  < 2e-16 ***
PromoIntervalJan,Apr,Jul,Oct  611.515    79.780   7.665 2.56e-14 ***
PromoIntervalMar,Jun,Sept,Dec 653.976   107.116   6.105 1.19e-09 ***
cat                         -441.386     10.409 -42.406  < 2e-16 ***
DaysSincePromo                -4.057     17.582  -0.231    0.818
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1312 on 2443 degrees of freedom
Multiple R-squared:  0.6764,    Adjusted R-squared:  0.6756
F-statistic: 851.1 on 6 and 2443 DF,  p-value: < 2.2e-16
> vif(mlinear_corr)
                   GVIF Df GVIF^(1/(2*Df))
Promo          2.610523  1        1.615711
PromoInterval  2.066074  3        1.128559
cat            2.065034  1        1.437022
DaysSincePromo 2.610551  1        1.615720
```

Fig. 4.1.3a

```
> RMSE.trainset
[1] 0.07407643
> RMSE.validationset
[1] 0.07563738
> RMSE.testset
[1] 0.1122801
```

Fig. 4.1.3b

```
> RMSE.trainset
[1] 0.08022156
> RMSE.validationset
[1] 0.08356124
> RMSE.testset
[1] 0.1317644
```

Fig. 4.1.4

```
> varImp(mlinear2, scale = FALSE)
                      Overall
Promo1             29.7212478
CompetitionDistance 0.7781394
SchoolHoliday1      4.0332382
Assortmentc        11.3160772
month2              1.4551444
month3              2.0257349
month4              5.6116425
month5              5.6974114
month6              6.1487562
month7              4.9176322
month8              1.2660625
month9              1.0130668
month10             0.8500107
month11             3.5532627
month12             9.7750222
year2015            5.3953442
isclosetmr1        18.9687238
iscloseyest1       18.5039041
isschholyest1       3.6022487
cat                53.9998716
DaysSincePromo      5.0082469
```

Fig 4.2.2a

```
#missing values so we attempt to use caret to predict
sum(is.na(train.tree))
train.preprocess <- preProcess(train.tree, method = c("center", "scale", 'knnImpute'))
train.tree <- predict(train.preprocess, newdata = train.tree)
sum(is.na(train.tree))
```

Fig 4.2.2b

```
> colSums(is.na(train.dt)) == 0
         DayOfWeek              Sales              Promo      SchoolHoliday         Assortment
              TRUE               TRUE               TRUE               TRUE               TRUE
CompetitionDistance      PromoInterval      Promosinceint Competitionsinceint      isweekendyest
              TRUE               TRUE              FALSE              FALSE               TRUE
      isweekendtmr          isweekend              month               year         isclosetmr
              TRUE               TRUE               TRUE               TRUE               TRUE
       iscloseyest        isschholtmr       isschholyest     SinceLastPromo2                cat
              TRUE               TRUE               TRUE              FALSE               TRUE
    DaysSincePromo
              TRUE
```

Fig. 4.2.2c

```
> sum(is.na(train.tree))
[1] 2188
> train.preprocess <- preProcess(train.tree, method = c("center", "scale", 'knnImpute'))
Warning in preProcess.default(train.tree, method = c("center", "scale",  :
  These variables have zero variances: DayOfWeek.7, Assortment.b, cat.5, cat.8
> train.tree <- predict(train.preprocess, newdata = train.tree)
> sum(is.na(train.tree))
[1] 0
> levels(train_tree$SinceLastPromo2)
 [1] "-1.20266580509953"  "-0.959594354440718" "-0.716522903781909" "-0.4734514531231"   "-0.230380002464291"
 [6] "0.0126914481945175" "0.0126914481945176" "0.255762898853326"  "0.498834349512135"  "0.741905800170944"
[11] "0.984977250829753"  "1.22804870148856"
```

Fig. 4.2.2.1a

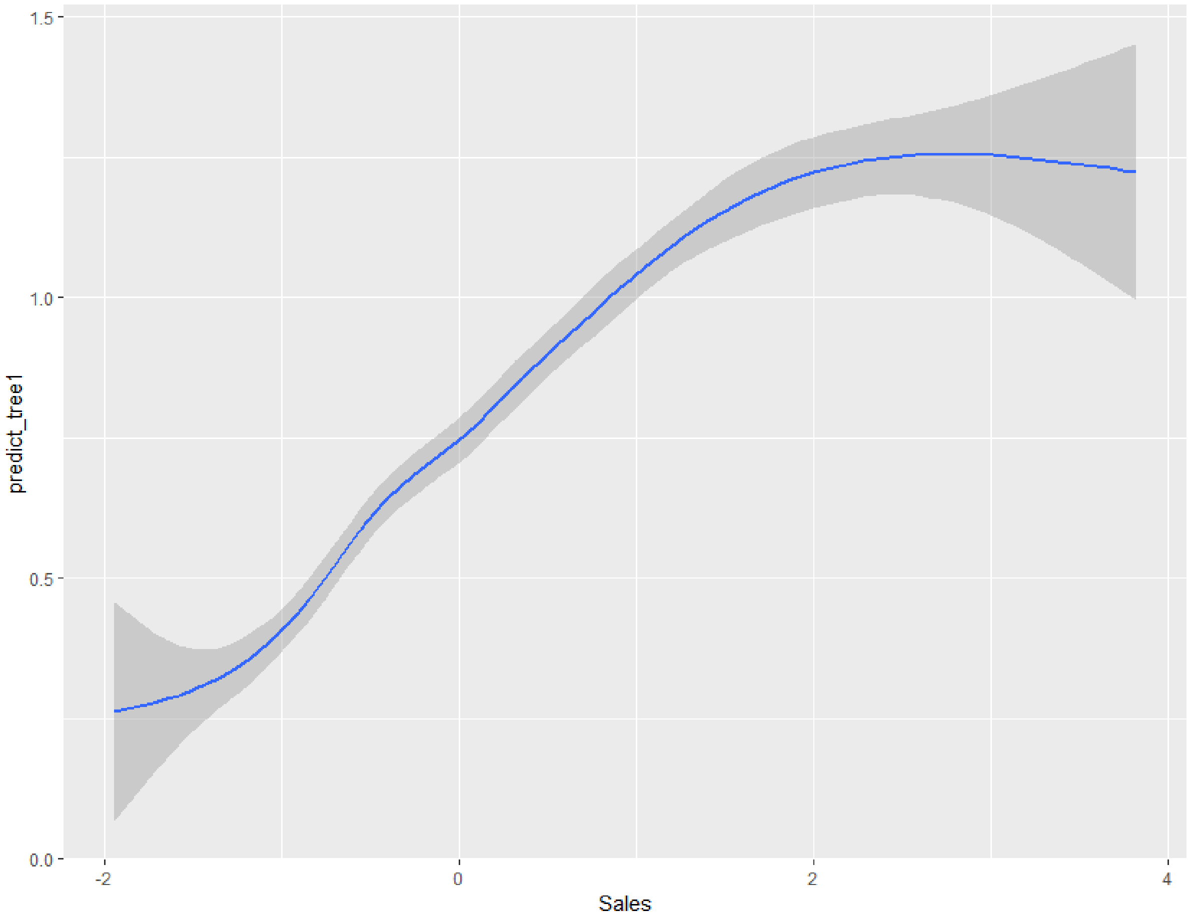
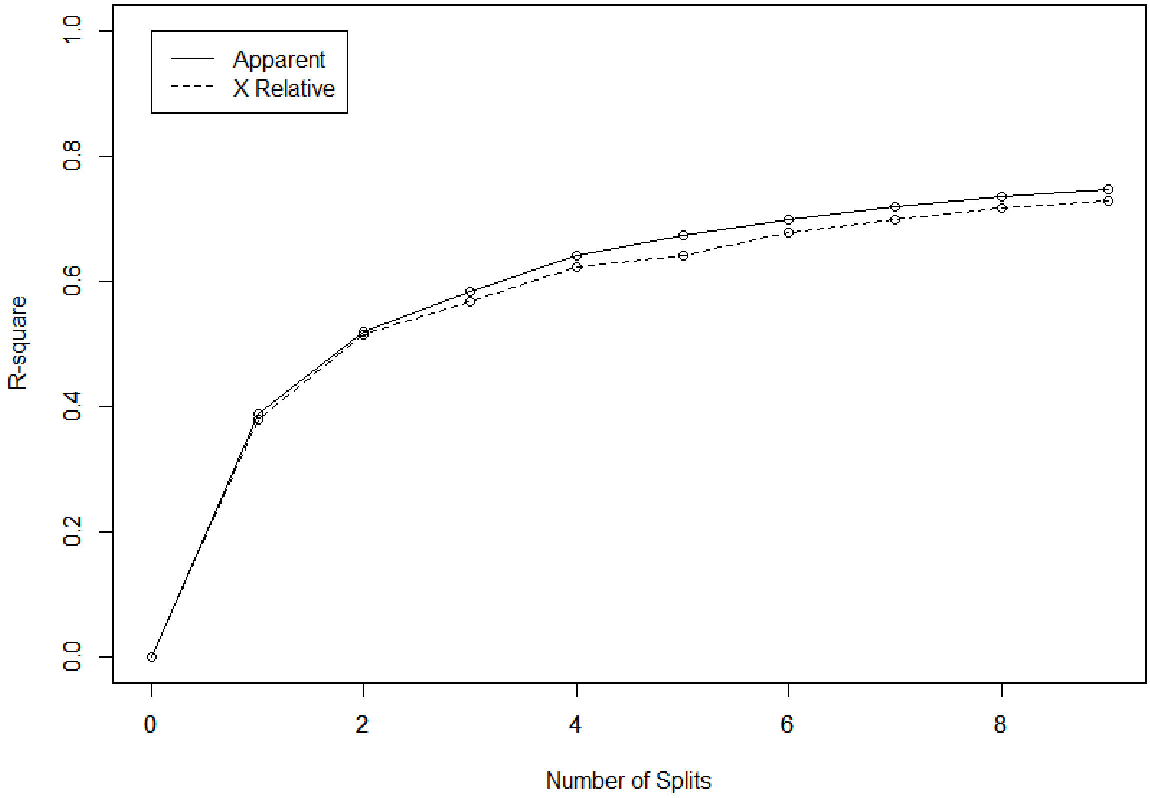


Fig. 4.2.2.1b

```
> summary(tree1)
Call:
rpart(formula = Sales ~ ., data = train_tree)
  n= 2450

           CP nsplit rel error    xerror       xstd
1  0.38845451      0 1.0000000 1.0002086 0.03682139
2  0.13010907      1 0.6115455 0.6208311 0.02773562
3  0.06506641      2 0.4814364 0.4847466 0.02357332
4  0.05836083      3 0.4163700 0.4319333 0.01977156
5  0.03153159      4 0.3580092 0.3766383 0.01927114
6  0.02526696      5 0.3264776 0.3592605 0.01892893
7  0.02126662      6 0.3012106 0.3233679 0.01853128
8  0.01464550      7 0.2799440 0.3013479 0.01717041
9  0.01125538      8 0.2652985 0.2833602 0.01568124
10 0.01000000      9 0.2540431 0.2715114 0.01554293
```

Fig. 4.2.2.1c



Apparent
X Relative
R-square
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
Number of Splits

Fig. 4.2.2.2a

```
> varImp(tree1)
                       Overall
Assortment           0.6421546
cat                  1.3958367
CompetitionDistance  0.3324542
Competitionsinceint  0.1058056
DayOfWeek            1.2371147
DaysSincePromo       0.9814901
iscloseyest          1.3694189
isweekendyest        1.0794684
month                0.3452058
Promo                0.9218843
PromoInterval        0.3472872
Promosinceint        0.1058056
SchoolHoliday        0.0000000
isweekendtmr         0.0000000
isweekend            0.0000000
year                 0.0000000
isclosetmr           0.0000000
isschholtmr          0.0000000
isschholyest         0.0000000
SinceLastPromo2      0.0000000
```

```
> tree1$variable.importance
                cat       PromoInterval      DaysSincePromo CompetitionDistance       Promosinceint 
        1208.418360          726.915848          461.562791          404.271290          400.810418 
              Promo     SinceLastPromo2 Competitionsinceint         iscloseyest           DayOfWeek 
         247.949706          209.628271          191.468601          162.439403          153.685510 
      isweekendyest          Assortment           isweekend          isclosetmr               month 
         140.312348           75.229149           24.796071           22.009996           10.874987 
               year 
           8.326231 
```

Fig. 4.2.2.2b

```
> summary(tree2)
Call:
rpart(formula = Sales ~ cat + DayOfWeek + iscloseyest + PromoInterval + 
    DaysSincePromo, data = train_tree)
  n= 2450 

           CP nsplit rel error    xerror       xstd
1  0.38845451      0 1.0000000 1.0009662 0.03684716
2  0.13010907      1 0.6115455 0.6275087 0.02849457
3  0.06506641      2 0.4814364 0.4836153 0.02429274
4  0.05836083      3 0.4163700 0.4494263 0.02211699
5  0.03153159      4 0.3580092 0.3820658 0.02023740
6  0.02526696      5 0.3264776 0.3497682 0.01905586
7  0.02126662      6 0.3012106 0.3234457 0.01889180
8  0.01464550      7 0.2799440 0.3000557 0.01670908
9  0.01125538      8 0.2652985 0.2769453 0.01589532
10 0.01000000      9 0.2540431 0.2646040 0.01550255
```

Fig. 4.2.2.2c

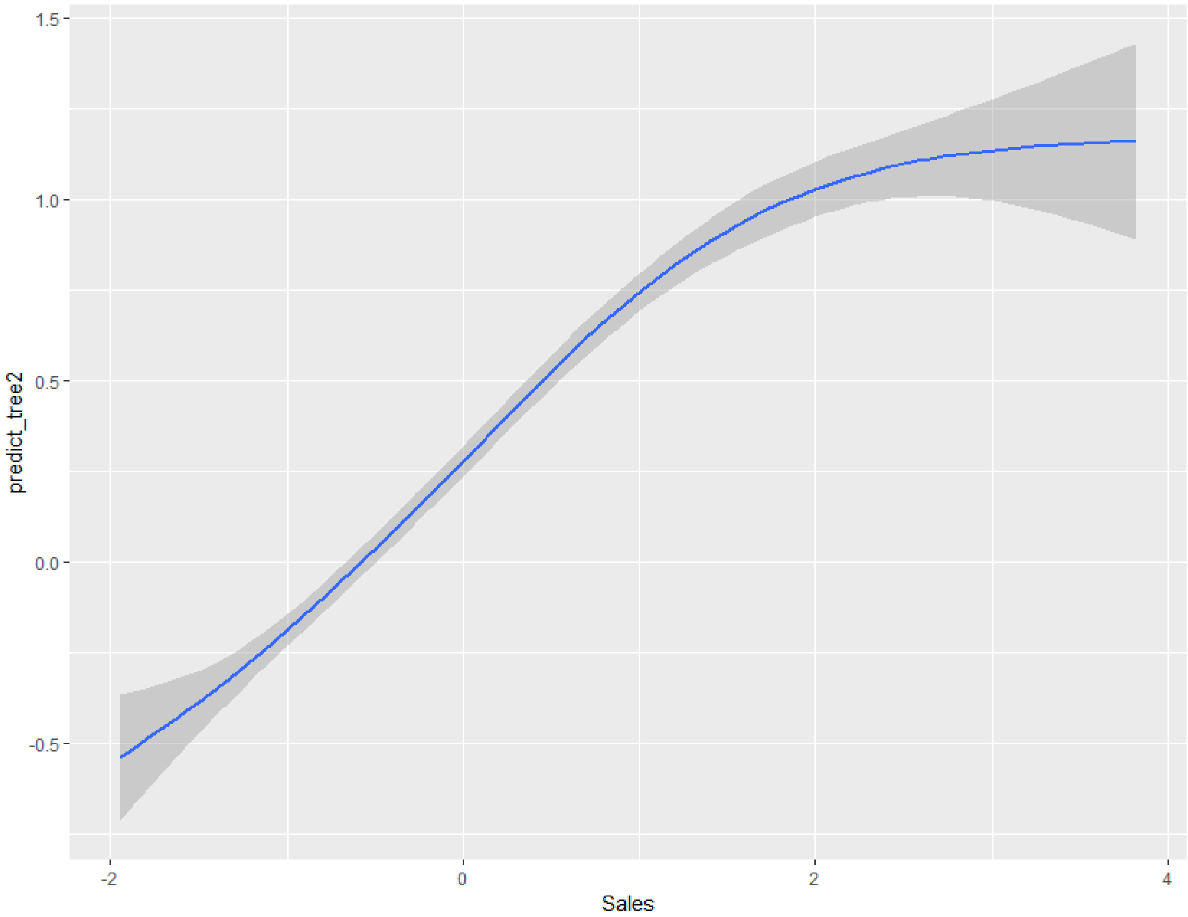


Fig. 4.2.2.3a

```
#model 3 - rpart, growing the tree to the maximum
tree3 <- rpart(Sales ~., data = train_tree, control = rpart.control(minsplit = 2, cp = 0))
printcp(tree3)

#model 4 - pruning the tree
cp.opt <- tree3$cptable[which.min(tree3$cptable[,"xerror"]),"CP"]
tree4 <- prune(tree3, cp.opt)
prp(tree4) #tree plotted out is still extremely big - possibility of overfitting
summary(tree4)
nodes <- as.numeric(rownames(tree4$frame))
max(rpart:::tree.depth(nodes)) #depth of 10 - risk of overfitting
printcp(tree4)
```

Fig. 4.2.2.3b

```
86 0.00055443     104   0.095267 0.19411 0.015013
```

Fig. 4.2.2.3c

```
> max(rpart:::tree.depth(nodes)) #depth of 12 - risk of overfitting
[1] 10
```

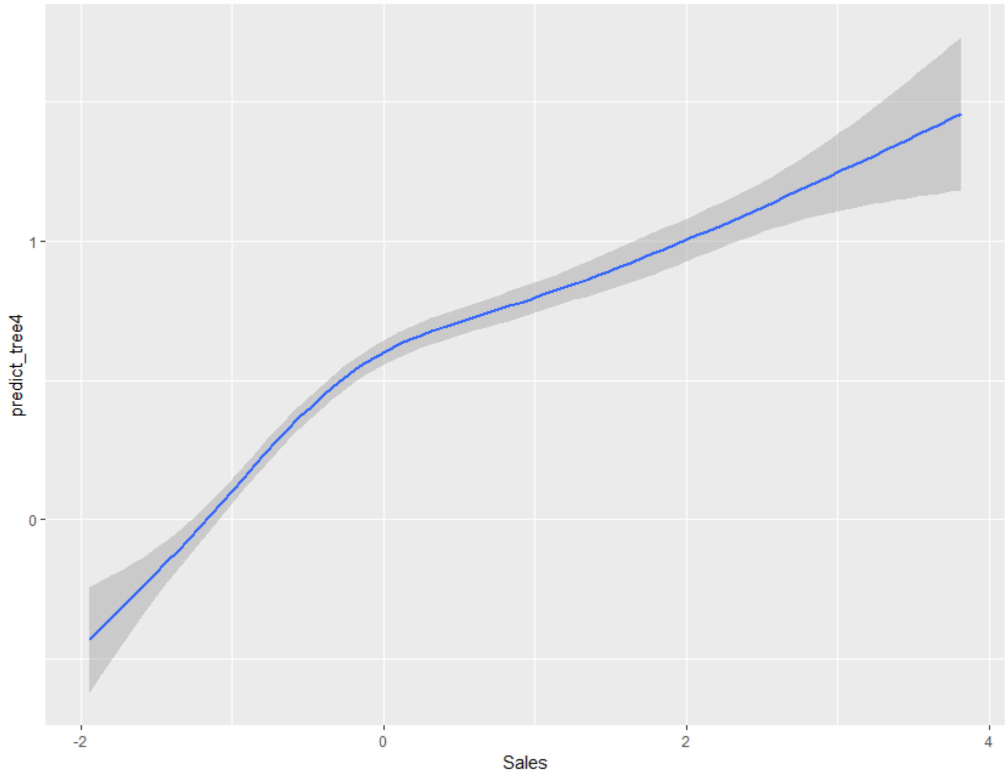


Fig. 4.2.2.3d

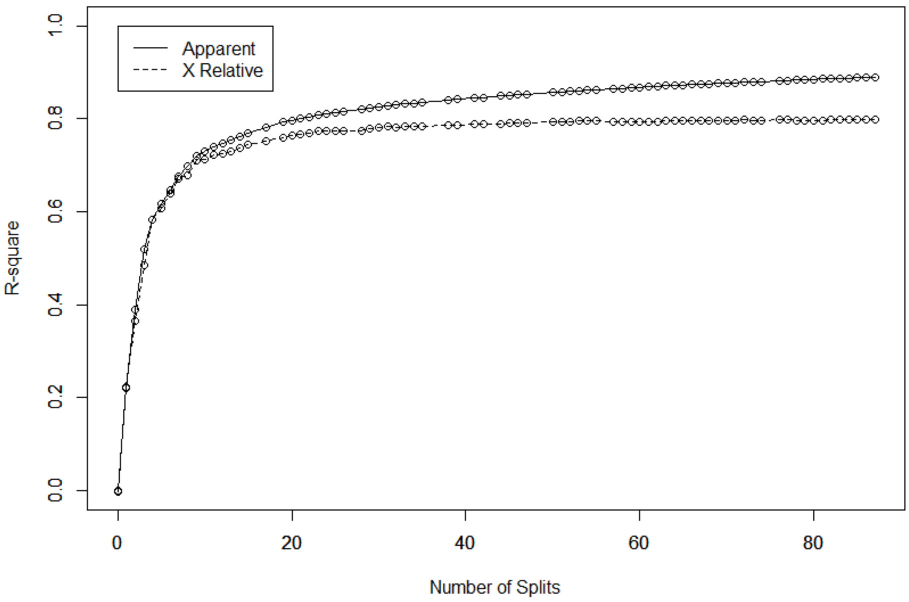


Fig. 4.2.2.3e

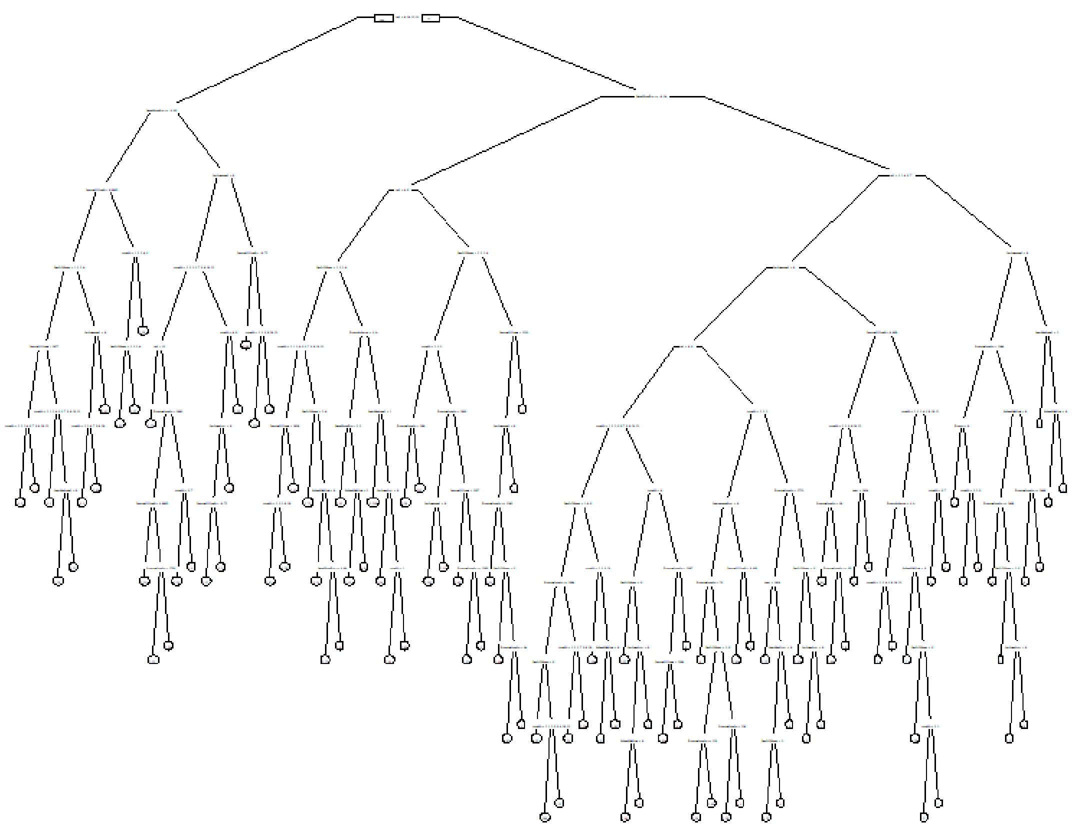
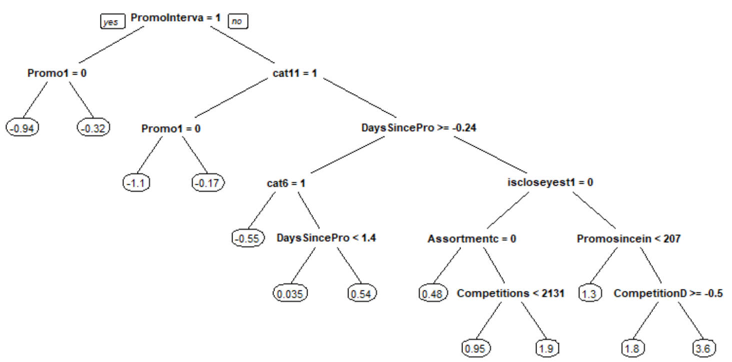


Fig. 4.2.2.4a

```
> tree5
CART

2450 samples
  20 predictor

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 2450, 2450, 2450, 2450, 2450, 2450, ...
Resampling results across tuning parameters:

  maxdepth  RMSE       Rsquared   MAE
   1        0.8754867  0.2266122  0.6646790
   2        0.7843333  0.3799204  0.5989796
   3        0.7078433  0.4944428  0.5475273
   4        0.6711196  0.5457214  0.5196423
   5        0.6472788  0.5774776  0.4993962
   6        0.6265310  0.6042996  0.4788834
   7        0.6016896  0.6352918  0.4550571
   8        0.5761687  0.6653243  0.4318195
   9        0.5589462  0.6849883  0.4183782
  11        0.5288958  0.7176617  0.3985258
  12        0.5233757  0.7234466  0.3936535
  13        0.5202636  0.7266621  0.3908224
  14        0.5199548  0.7269725  0.3906312
  15        0.5198464  0.7270837  0.3905606
  17        0.5198464  0.7270837  0.3905606
  19        0.5198464  0.7270837  0.3905606
  20        0.5198464  0.7270837  0.3905606
  22        0.5198464  0.7270837  0.3905606
  23        0.5198464  0.7270837  0.3905606
  24        0.5198464  0.7270837  0.3905606

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was maxdepth = 15.
```

Fig. 4.2.2.4b




```
> max(rpart:::tree.depth(nodes.model5)) #finding depth of tree
[1] 6
```

Fig. 4.2.2.4c

```
> varImp(tree5)
rpart2 variable importance

  only 20 most important variables shown (out of 51)

                              Overall
DaysSincePromo                100.000
Promo1                         81.905
CompetitionDistance            72.569
Assortmentc                    68.427
Promosinceint                  65.137
Competitionsinceint            55.196
cat6                           39.003
iscloseyest1                   22.974
cat11                          12.206
isclosetmr1                    11.585
isweekendyest1                 11.321
PromoIntervalMar,Jun,Sept,Dec  11.154
cat3                           10.241
DayOfWeek6                      8.656
cat4                            8.496
isweekend1                      7.119
cat12                           5.578
cat9                            5.062
year2015                        0.000
```

Fig. 4.3.2

```
#using randomforest
library(randomForest)
train_forest = train.tree
```

Fig. 4.3.2.1

```
> forest1

Call:
 randomForest(formula = Sales ~ ., data = train_forest)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 6

          Mean of squared residuals: 0.1360833
                    % Var explained: 86.39
>
> rmse(predict_forest1, test.dt$Sales)/(max(test.dt$Sales) - min(test.dt$Sales))
[1] 0.1298064
```

Fig. 4.3.2.2a

```
> forest2
Random Forest

2450 samples
  20 predictor

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 1958, 1962, 1960, 1959, 1961, 1959, ...
Resampling results across tuning parameters:

  mtry  RMSE       Rsquared   MAE
   4    0.4309087  0.8378191  0.3046848
   5    0.4042968  0.8477562  0.2832206
  13    0.3765650  0.8588633  0.2612406
  15    0.3768998  0.8584630  0.2608351
  17    0.3763731  0.8587989  0.2606825
  20    0.3757165  0.8592292  0.2603655
  22    0.3768899  0.8583362  0.2608292
  29    0.3760629  0.8589789  0.2601216
  47    0.3813428  0.8551478  0.2621385

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 20.
```

Fig. 4.3.2.2b

```
> rmse(predict_forest2, test.dt$Sales)/(max(test.dt$Sales) - min(test.dt$Sales))
[1] 0.1477568
```

Fig. 4.3.2.3

```
> forest3
Random Forest

2450 samples
  20 predictor

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 2450, 2450, 2450, 2450, 2450, 2450, ...
Resampling results across tuning parameters:

  mtry  RMSE       Rsquared   MAE
   2    0.6092647  0.7874844  0.4590768
   4    0.4231992  0.8418887  0.3057549
   6    0.3843772  0.8555990  0.2746472
   9    0.3750795  0.8583064  0.2682619
  13    0.3754769  0.8570276  0.2682832
  17    0.3771255  0.8555755  0.2689772
  20    0.3789558  0.8541239  0.2697084

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 9.
```

Fig. 4.3.2.5a

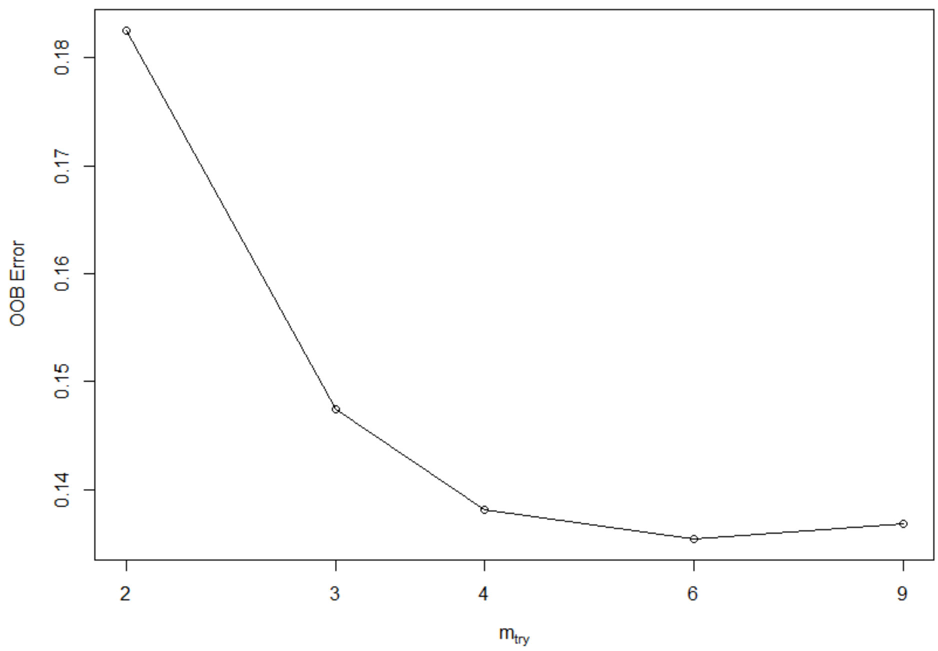


Fig. 4.3.2.5b

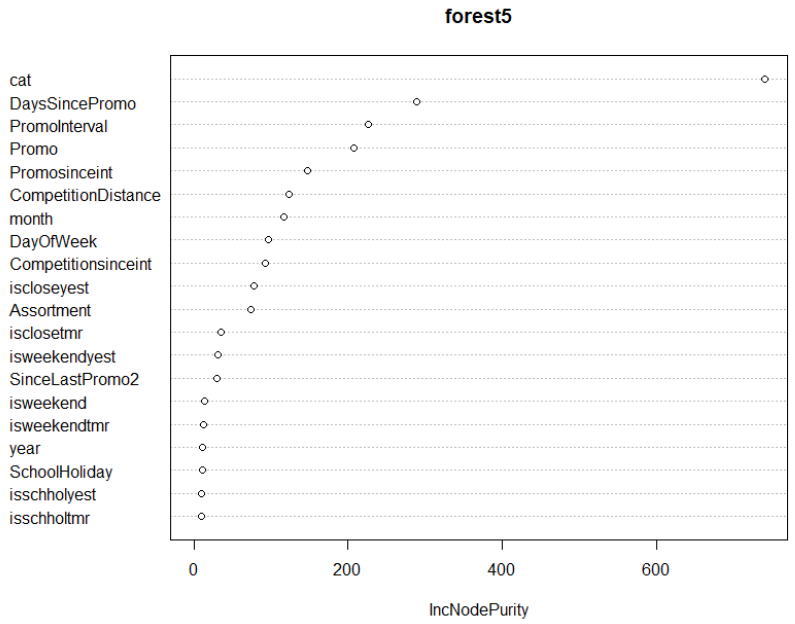


Fig. 4.4.2a

```
trainm <-as.matrix(trainset[,-c("Sales")])
testm <-as.matrix(testset[,-c("Sales")])

trainm<-as(trainm,"sparseMatrix")
testm<-as(testm,"sparseMatrix")

train_label<-trainset$Sales
train_label <- as.numeric(train_label)
test_label<-testset$Sales
test_label <- as.numeric(test_label)
train_matrix <- xgb.DMatrix(data=trainm,label=train_label)
test_matrix <- xgb.DMatrix(data=testm,label=test_label)
```

Fig. 4.4.2b

```
best_param = list()
best_seednumber = 1234
best_rmse = Inf

for (iter in 1:100) {
  param <- list(objective = "reg:linear",
                eval_metric = "rmse",
                max_depth = sample(6:10, 1),
                eta = runif(1, .01, .3),
                gamma = runif(1, 0.0, 0.2),
                subsample = runif(1, .6, .9),
                colsample_bytree = runif(1, .5, .8),
                min_child_weight = sample(1:40, 1),
                max_delta_step = sample(1:10, 1)
  )
  cv.nround = 1000
  cv.nfold = 5
  seed.number = sample.int(10000, 1)[[1]]
  set.seed(seed.number)
  mdcv <- xgb.cv(data=train_matrix, params = param, nthread=6,
                 nfold=cv.nfold, nrounds=cv.nround,
                 verbose = F, early_stopping_rounds=8, maximize=FALSE)

  min_rmse = min(mdcv$evaluation_log[, test_rmse_mean])

  if (min_rmse < best_rmse) {
    best_seednumber = seed.number
    best_param = param
  }
}
```

Fig. 4.4.2c

best_param	list [9]	List of length 9
objective	character [1]	'reg:linear'
eval_metric	character [1]	'rmse'
max_depth	integer [1]	10
eta	double [1]	0.2146533
gamma	double [1]	0.0282414
subsample	double [1]	0.8611443
colsample_bytree	double [1]	0.7506555
min_child_weight	integer [1]	26
max_delta_step	integer [1]	7

Fig. 4.4.2d

```
[1]     train-rmse:5779.970410+41.693709        test-rmse:5786.588281+159.260153
Multiple eval metrics are present. Will use test_rmse for early stopping.
Will train until test_rmse hasn't improved in 40 rounds.

[11]    train-rmse:1028.811535+6.894969 test-rmse:1099.387280+102.299317
[21]    train-rmse:730.106519+12.378571 test-rmse:853.989697+79.389475
[31]    train-rmse:665.118042+12.163274 test-rmse:822.135181+72.716939
[41]    train-rmse:628.226416+10.868932 test-rmse:806.229602+69.718820
[51]    train-rmse:600.311902+12.049614 test-rmse:801.066516+69.879493
[61]    train-rmse:578.985632+11.120469 test-rmse:798.172559+68.476575
[71]    train-rmse:561.670386+11.619217 test-rmse:797.825928+70.160338
[81]    train-rmse:547.820374+10.869931 test-rmse:798.884522+70.454405
[91]    train-rmse:534.530261+11.065036 test-rmse:800.070850+68.722420
[101]   train-rmse:523.432239+10.616550 test-rmse:801.219324+69.862177
[111]   train-rmse:513.506140+10.516272 test-rmse:802.924805+67.956601
Stopping. Best iteration:
[79]    train-rmse:550.220288+11.139018 test-rmse:797.426953+70.721011
```

Fig. 4.4.3a

```
 rmse(testset$Sales,xgbpred)/(max(test.dt$Sales)-min(test.dt$Sales))
1] 0.07281979
```

Fig. 4.4.3b

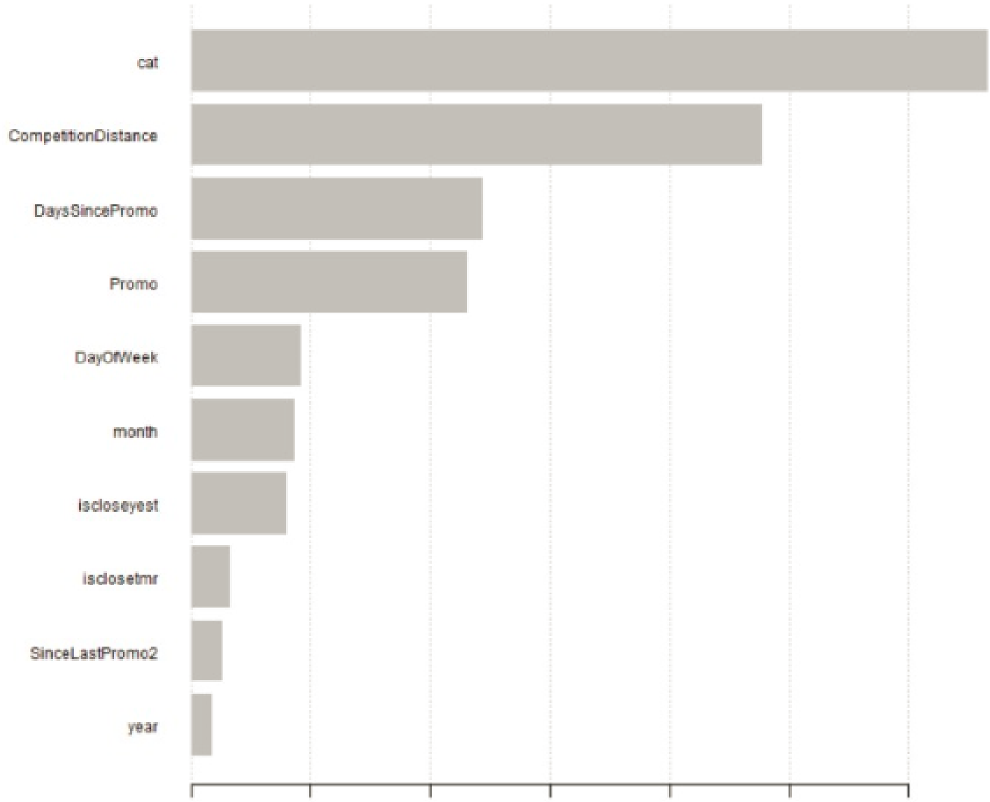


Fig. 6.1.a

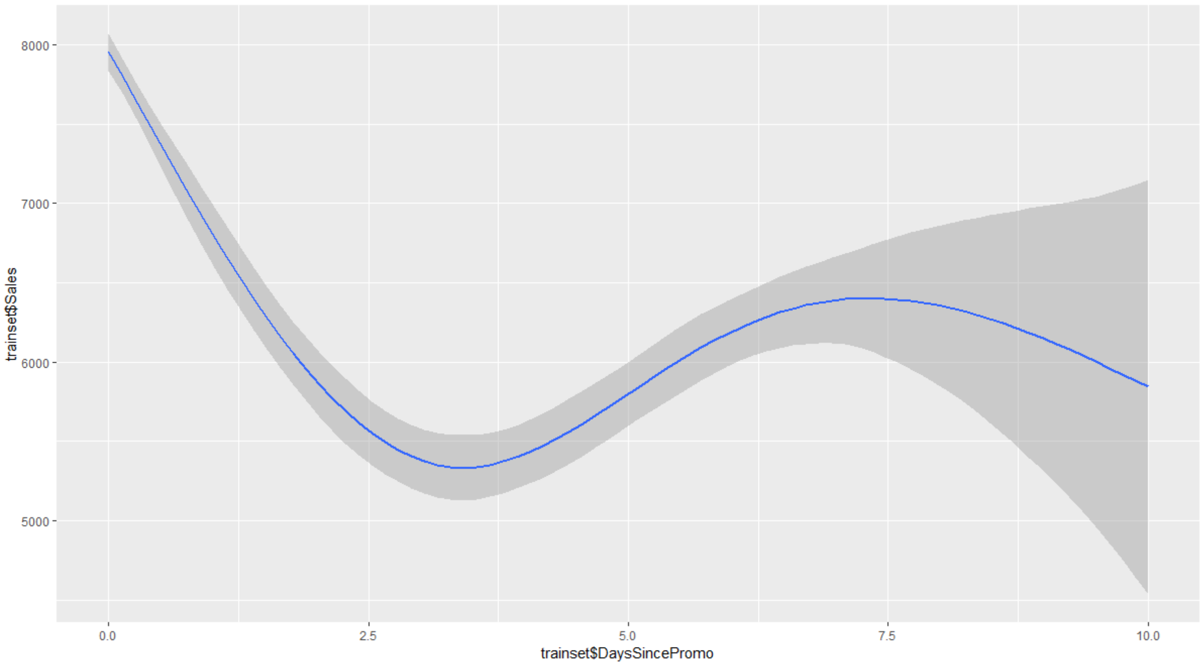


Fig. 6.1.b

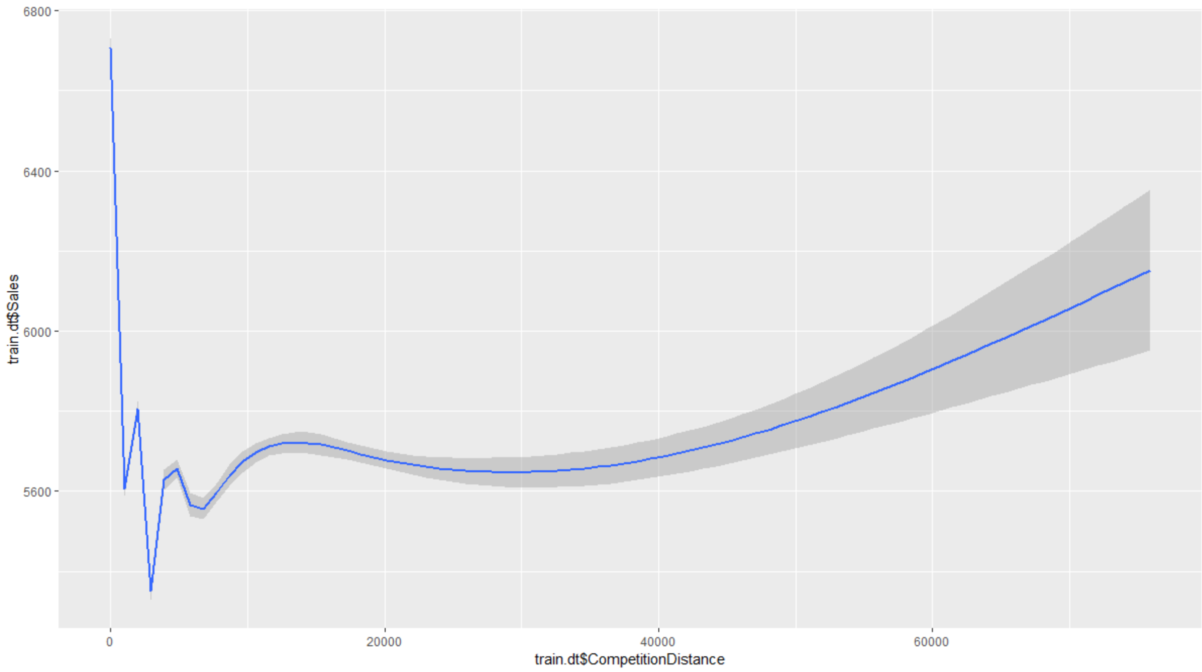
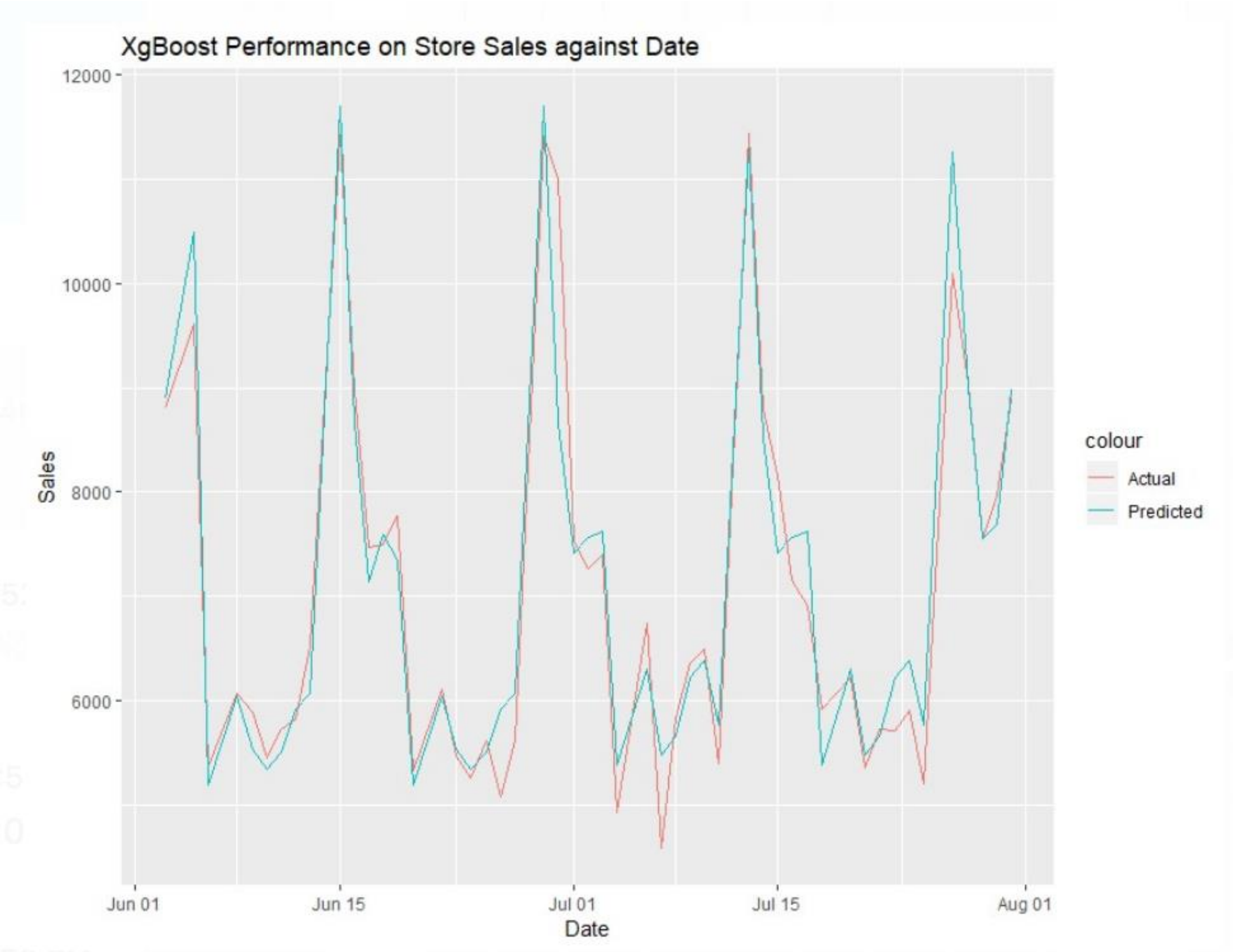


Fig. 6.4a



XgBoost Performance on Store Sales against Date
12000
10000
8000
6000
Sales
Jun 01
Jun 15
Jul 01
Jul 15
Aug 01
Date
colour
Actual
Predicted