# Walmart - Store Sales Forecasting

Kacey Ries

5/11/2022

## Introduction

Walmart is an American multinational retail corporation that operates a chain of hypermarkets and grocery stores in the United States, headquartered in Bentonville, Arkansas. Through Kaggle, Walmart opened a recruiting competition where job-seekers are provided with historical sales data for 45 Walmart stores across the country each containing many departments. Additionally, Walmart runs promotional markdown events throughout the year. These markdowns precede holidays, the four largest of which are the Super Bowl, Labor Day, Thanksgiving, and Christmas. The objective of this paper is to predict the sales for each department in each store (~3,000 Forecasts) using Multiple Linear Regression, Random Forest and Gradient Boost.

This document is structured in four primary sections:

- Introduction: Description of the project, goal, and the primary steps/techniques.
- Methods & Analysis: In-depth description of process and techniques, as well as insights gained through exploratory analysis and finally the modeling approach.
- Results: Review model results and performance.
- Conclusion: Summary of the report and forward-looking thoughts.

### Walmart Dataset

#### Original Data Sets:

(https://www.kaggle.com/competitions/walmart-recruiting-store-sales-forecasting/data)

Through Kaggle, we are provided three different data sets with historical sales data for 45 Walmart stores in various regions. Each store contains a number of departments, and the objective is to predict the department-wide sales for each store. The historical data spans 2010-02-05 to 2012-11-01 and the features in each data set are:

- train.csv: The historical training data covers 2010-02-05 to 2012-11-01 with five features.
  - Store: Store number.
  - Dept: Department number.
  - Date: The week.
  - Weekly_Sales: Sales numbers for the given department in a given store.
  - IsHoliday: If the week is a special holiday week.
- stores.csv: Contains anonymized information about the 45 stores.
  - Store: Store numbers ranging from 1 to 45.

  - Type: Types of stores labeled as 'A', 'B', or 'C'.
  - Size: Size of a store by the number of products available ranging from 34,000 to 210,000.

- features.csv: Contains additional data related to the store, department, and regional activity for the given dates. It contains eight features.
  - Store: Store Number.
  - Date: the week.
  - Temperature: Average temperature in the region.
  - Fuel_Price: Cost of fuel in the region.
  - MarkDown1-5: Anonymized data related to promotional markdowns that Walmart is running.
  - CPI: Consumer Price Index.
  - Unemployment: Unemployment Rate.
  - IsHoliday: Whether the week is a special holiday week.

In typical Machine Learning tasks, train, validation, and test splits are done randomly as there is little to no dependence from one observation to the other. However, our data-set is a series of data points indexed in time order; sequenced at successive equally spaced points in time. This type of data is called Time Series Data and we therefore cannot split our data randomly.

Our approach will be an out-of-time validation where we validate and test our model on a later dataset than the one we fit our model on. This approach is ideal to our dataset as the objective is to predict department-wide sales for each store and this approach is a simple method to ensure that all department and stores are in all datasets.

### Model Evaluation

The model will be evaluated through a performance metric provided by Walmart. This metric is, Weighted Mean Absolute Error (WMAE) mathematically represented as:

$$WMAE = \frac{1}{\Sigma_i w_i} \Sigma_i w_i |y_i - \hat{y}|$$

Breaking this formula down:

- $n$ is the number of rows.
- $\hat{y}_i$ is the predicted sales
- $y_i$ is the actual sales
- $w_i$ are the weights.

The model is evaluated through a WMAE because weeks that include holidays are weighted five times higher in the evaluation than non-holiday weeks. During holidays $w$ is equal to 5, else $w$ is equal to 1.

## Methods & Analysis

### Data Preparation

We will now load the dataset and libraries for analysis.

```r
# Load Packages
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra", repos = "http://cran.us.r-project.org")
if(!require(timetk)) install.packages("timetk", repos = "http://cran.us.r-project.org")
if(!require(reshape2)) install.packages("reshape2", repos = "https://cran.us.r-project.org")
if(!require(Boruta)) install.packages("Boruta", repos = "https://cran.us.r-project.org")
if(!require(imputeTS)) install.packages("imputeTS", repos = "https://cran.us.r-project.org")
if(!require(modeltime)) install.packages("modeltime", repos = "https://cran.us.r-project.org")
if(!require(parsnip)) install.packages("parsnip", repos = "https://cran.us.r-project.org")
if(!require(recipes)) install.packages("recipes", repos = "https://cran.us.r-project.org")
if(!require(workflows)) install.packages("workflows", repos = "https://cran.us.r-project.org")
# Load in the raw Walmart data
train_df <- read_csv("data/train.csv")
stores_df <- read_csv("data/stores.csv")
features_df <- read_csv("data/features.csv")
```

Below are the first 6 rows of the train_df dataset. There are five features represented by the columns and each row represents an observation. We can confirm that this dataset is in tidy format. However, we also observe that the "Store" and "Dept" are being read as "dbl" meaning that they are being read as numeric values with decimal points. We will want to convert these to Factors to represent categorical data.

```r
head(train_df)
```

```
# A tibble: 6 x 5
  Store  Dept Date       Weekly_Sales IsHoliday
  <dbl> <dbl> <date>            <dbl> <lgl>
1     1     1 2010-02-05       24924. FALSE
2     1     1 2010-02-12       46039. TRUE
3     1     1 2010-02-19       41596. FALSE
4     1     1 2010-02-26       19404. FALSE
5     1     1 2010-03-05       21828. FALSE
6     1     1 2010-03-12       21043. FALSE
```

```r
train_df <- train_df %>%
    mutate(
    Store = as_factor(Store),
    Dept = as_factor(Dept)
)
```

The first 6 rows of the stores_df dataset are generated below. There are three features represented by the columns and each row represents an observation. We can confirm that this dataset is in a tidy format. As with train_df, we also observe that the "Store" is being read as "dbl" and the Type variable is being read as "chr" a character. We will want to convert these to Factors to represent categorical data.

```r
head(stores_df)
```

```
# A tibble: 6 x 3
  Store Type    Size
  <dbl> <chr>  <dbl>
1     1 A     151315
2     2 A     202307
```

```
3     3 B      37392
4     4 A     205863
5     5 B      34875
6     6 A     202505
```

```
stores_df <- stores_df %>%
    mutate(
        Store = as_factor(Store),
        Type = as_factor(Type)
    )
```

The first 6 rows of the final dataset, features_df, are listed below. There are 12 features represented by the columns and each row represents an observation. We can confirm that this dataset is in tidy format. As with the previous datasets, we convert "Store" to a Factor. We also observe the MarkDown columns have NA observations. We will replace NA with 0.

```
head(features_df)
```

```
# A tibble: 6 x 12
  Store Date       Temperature Fuel_Price MarkDown1 MarkDown2 MarkDown3
  <dbl> <date>           <dbl>      <dbl>     <dbl>     <dbl>     <dbl>
1     1 2010-02-05        42.3       2.57        NA        NA        NA
2     1 2010-02-12        38.5       2.55        NA        NA        NA
3     1 2010-02-19        39.9       2.51        NA        NA        NA
4     1 2010-02-26        46.6       2.56        NA        NA        NA
5     1 2010-03-05        46.5       2.62        NA        NA        NA
6     1 2010-03-12        57.8       2.67        NA        NA        NA
# ... with 5 more variables: MarkDown4 <dbl>, MarkDown5 <dbl>, CPI <dbl>,
#   Unemployment <dbl>, IsHoliday <lgl>
```

```
features_df <- features_df %>%
    mutate(
        Store = as_factor(Store)
    )
features_df[is.na(features_df)] <- 0
```

Through observing each dataset we notice that each set has the feature "Store". We can use this feature to join all datasets into a single master dataset for ease of Analysis. Additionally, we will create two unique identifiers; an ID labeled as ID combining Store, Dept and Date. And an ID labeled as StoreDept combining Store and Dept.

```
# Join datasets into master_df
master_df <- train_df %>% left_join(stores_df, by = "Store")
master_df <- master_df %>%
  select(-IsHoliday) %>%
  left_join(features_df, by = c("Store", "Date"))

# Create Unique Identifier across Store, Dept and Date
master_df <- master_df %>% unite(ID, Dept, Date, sep = "_", remove = FALSE) %>%
  unite(ID, Store, ID, sep = "_", remove = FALSE)

# Create Unique Identifier across Store and Dept
```

```
master_df <- master_df  %>%
  unite(StoreDept, Store, Dept, sep = "_", remove = FALSE)
head(master_df)
```

```
# A tibble: 6 x 18
  ID      StoreDept Store Dept  Date       Weekly_Sales Type    Size Temperature
  <chr>   <chr>     <fct> <fct> <date>            <dbl> <fct>  <dbl>       <dbl>
1 1_1_20~ 1_1       1     1     2010-02-05       24924. A     151315        42.3
2 1_1_20~ 1_1       1     1     2010-02-12       46039. A     151315        38.5
3 1_1_20~ 1_1       1     1     2010-02-19       41596. A     151315        39.9
4 1_1_20~ 1_1       1     1     2010-02-26       19404. A     151315        46.6
5 1_1_20~ 1_1       1     1     2010-03-05       21828. A     151315        46.5
6 1_1_20~ 1_1       1     1     2010-03-12       21043. A     151315        57.8
# ... with 9 more variables: Fuel_Price <dbl>, MarkDown1 <dbl>,
#   MarkDown2 <dbl>, MarkDown3 <dbl>, MarkDown4 <dbl>, MarkDown5 <dbl>,
#   CPI <dbl>, Unemployment <dbl>, IsHoliday <lgl>
```

As previously mentioned, master_df will be split by time for out-of-time validation. The first split will be into two datasets; Exploratory and Test Set. The full dataset contains the years 2010, 2011, 2012, spanning from 2010-02-05 to 2012-10-26. The Exploratory Set will have the first two years of the dataset and the remaining months will be reserved for the Test Set. The second split will be when we fit our Machine Learning Algorithms. The Exploratory Set will be split in half to train the Algorithms and to validate the algorithm, ensuring that both training and validation is done with an entire years worth of data.

The Exploratory Set will be used to explore our dataset and also to develop our model. The test set will be reserved for the final evaluation.

```
# Keep dates which are less than or equal to 2012-02-10
exp_df <- master_df %>%
  filter(Date <= c("2012-02-10"))
# Keep dates which are greater than 2012-02-10
test_df <- master_df %>%
  filter(Date > c("2012-02-10"))

test_df <- test_df %>% select(ID, StoreDept, Weekly_Sales, IsHoliday, Size)
remove(features_df, stores_df, train_df)
```

## Data Exploration

### Time Series Decomposition

Prior to building out the Machine Learning Algorithms, it is important to conduct exploratory analysis to understand the distributions of our features and the dependencies they may or may not have with each other.

As we are dealing with Time Series Analysis, we will first deconstruct the Time Series into three fundamental components by Store Type. This task of deconstructing a Time Series is called decomposition. Decomposition of Time Series looks at the following core components:

- Seasonality (S): Does the time series have patterns that repeat with a fixed period of time?
- Trend (T): Does the time series represent an upward or downward slope?
- Remainder (R): Also called “Random” or “Noise”. What are the outliers that are not consistent with the rest dataset?

**Decomposition Models**

There are two types of Decomposition methods, Additive and Multiplicative.

- Additive: Argues that time series data is a function of the sum of its components.

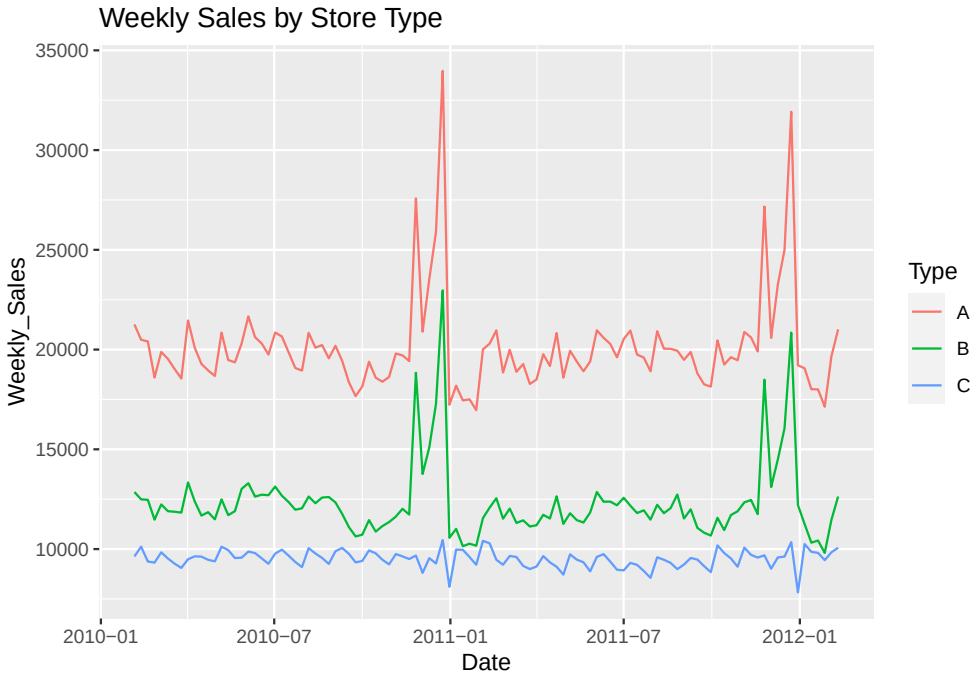
$$Y = S + T + R$$

- Multiplicative: Argues that time series data is a function of the product of its components.

$$Y = S * T * R$$

One method to distinguish between Additive and Multiplicative seasonal components is simply through visualization. If the seasonal components increase with time, we should use multiplicative decomposition.

```
exp_df %>%
    group_by(Type, Date) %>%
    summarise(Weekly_Sales = mean(Weekly_Sales)) %>%
    ggplot(aes(Date, Weekly_Sales, color = Type)) +
    geom_line() +
    ggtitle("Weekly Sales by Store Type")
```



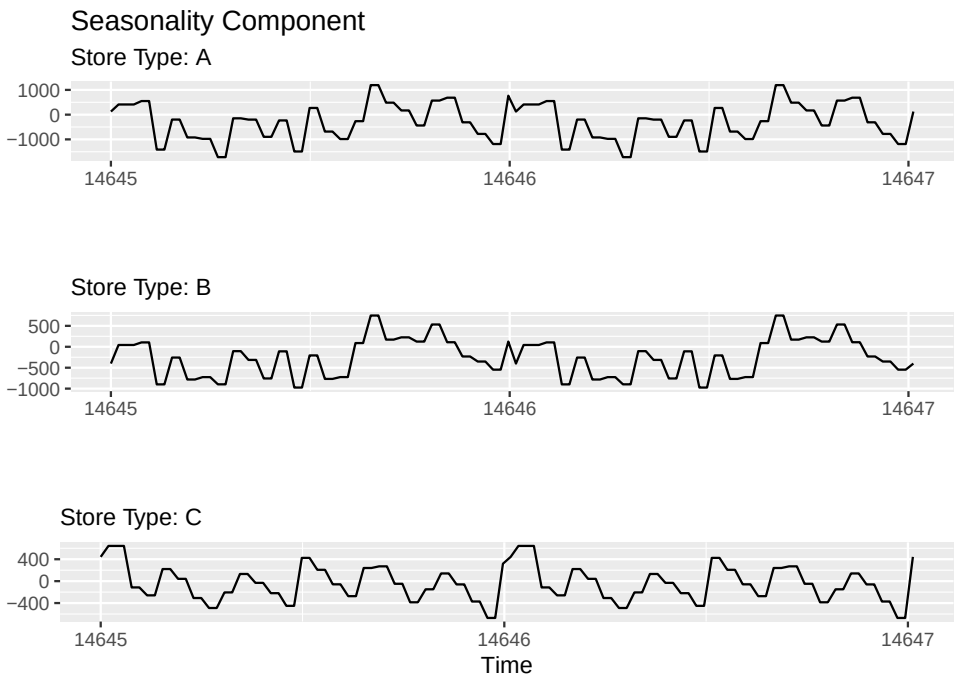

Observing the Weekly Sales plot above, we see that Store type A is by far more popular than store B and C. We also observe that the seasonal components do not increase with time. Therefore, our Decomposition approach will be Additive. A technique available to us for Additive Decomposition is, Seasonal and Trend decomposition using Loess (STL). The STL approach has a few advantages; it can handle any type of seasonality (Weekly, Monthly, Quarterly etc.) and it is robust to outliers. We will take this approach as there is variability across Store Types.

```r
# Average Weekly Sales by Store Type
exp_sales <- exp_df %>%
    group_by(Type, Date) %>% # Group by Type and Date
    summarise(Weekly_Sales = mean(Weekly_Sales)) %>% # Mean of Weekly Sales
    ungroup() %>%
    spread(key = Type, value = Weekly_Sales) # Convert to wide data for lapply
# Convert exp_sales dataframe to timeseries with weekly frequency
weekly_ts <- lapply(exp_sales, function(t) ts(t, start = c(exp_sales[1, 1]), frequency = 365.25/7))
# Apply STL Function
stl_weekly_ts <- lapply(weekly_ts, function(t) stl(t, "periodic"))
```

**Seasonality**

By removing trend from the time series, what is left is a time series that exposes seasonality. We observe that there is slight variability in seasonality across the stores, however, there is a distinct seasonal effect in November and December.

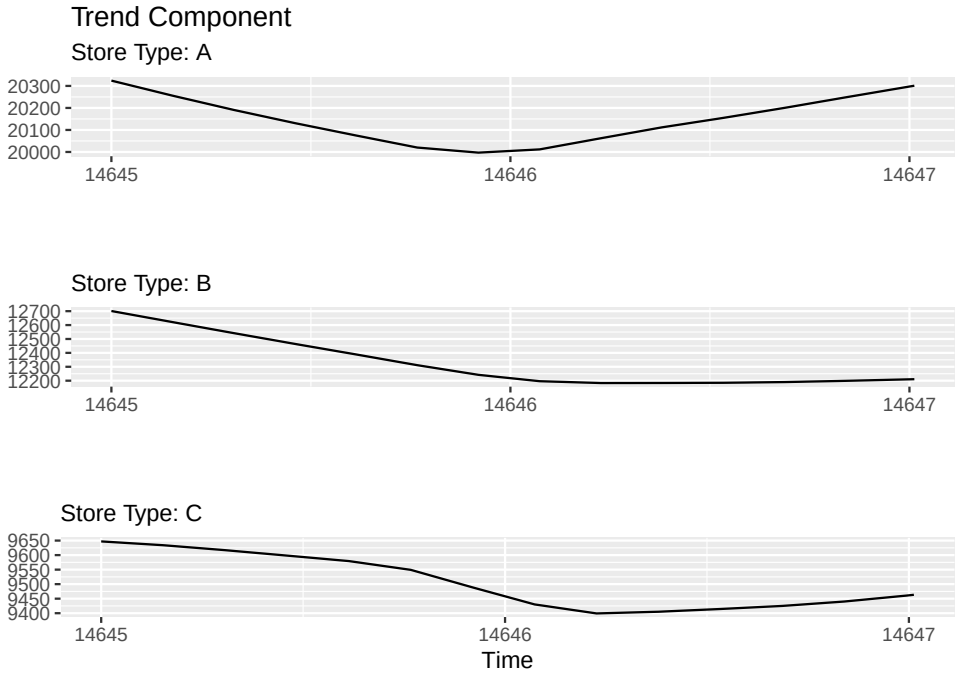
```r
grid.arrange(
    autoplot(stl_weekly_ts$A$time.series[,1]) + labs(title = "Seasonality Component", subtitle = "Store
    autoplot(stl_weekly_ts$B$time.series[,1]) + labs(subtitle = "Store Type: B", x = "", y = ""),
    autoplot(stl_weekly_ts$C$time.series[,1]) + labs(subtitle = "Store Type: C", y = "")
)
```




**Trend**

Using a moving average window, we can detect the underlying trend which has a smoothing effect on the time series. To perform this decomposition, it is important to use a moving window that represents the size of the seasonality. As our time series is weekly, our moving average window is ~52. We observe that all stores follow a trend, with an inflection plot towards the end of the year.

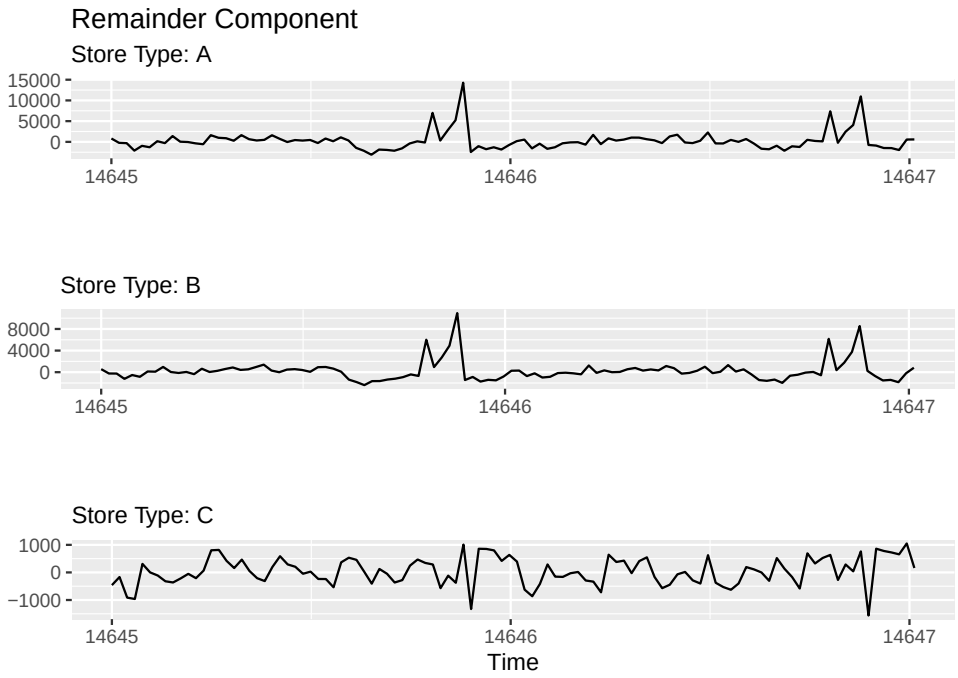
```
grid.arrange(
    autoplot(stl_weekly_ts$A$time.series[,2]) + labs(title = "Trend Component", subtitle = "Store Type:
    autoplot(stl_weekly_ts$B$time.series[,2]) + labs(subtitle = "Store Type: B", x = "", y = ""),
    autoplot(stl_weekly_ts$C$time.series[,2]) + labs(subtitle = "Store Type: C", y = "")
)
```




**Remainder**

Without trend or seasonality, what is left behind is a time series with random noise. Random noise is the fluctuations that trend or seasonality cannot explain. We observe that stores A and B have similar Remainder Components. Store C has much more "Noise" evenly distributed throughout the year.

```
grid.arrange(
    autoplot(stl_weekly_ts$A$time.series[,3]) + labs(title = "Remainder Component", subtitle = "Store Ty
    autoplot(stl_weekly_ts$B$time.series[,3]) + labs(subtitle = "Store Type: B", x = "", y = ""),
    autoplot(stl_weekly_ts$C$time.series[,3]) + labs(subtitle = "Store Type: C", y = "")
)
```




```
remove(stl_weekly_ts, weekly_ts, exp_sales)
```

**Feature Selection**

Now that we have performed Time Series Decomposition and seen the effects that Trend, Seasonality and Remainders have on our time series, we will dive into our features and determine through Feature Selection which features have the greatest influence on Weekly Sales, our target variable.

Feature Selection is the process of selecting a subset of relevant features which influence our target variable for use in a machine learning model. There are primarily three benefits of performing feature selection:

- Reduce Overfitting & Improve Accuracy: The less redundant data we have, the less likely we are to make decisions based on "Noise".
- Reduce Training Time: Fewer data points reduces the model complexity and results in an algorithm which will train more quickly.
- Occam's Razor: The principle that the most likely solution is the simplest one. This principle also lends to improving the "explainability" of our model due to having less features.

In this exercise, we will use the following Feature Selection Methods:

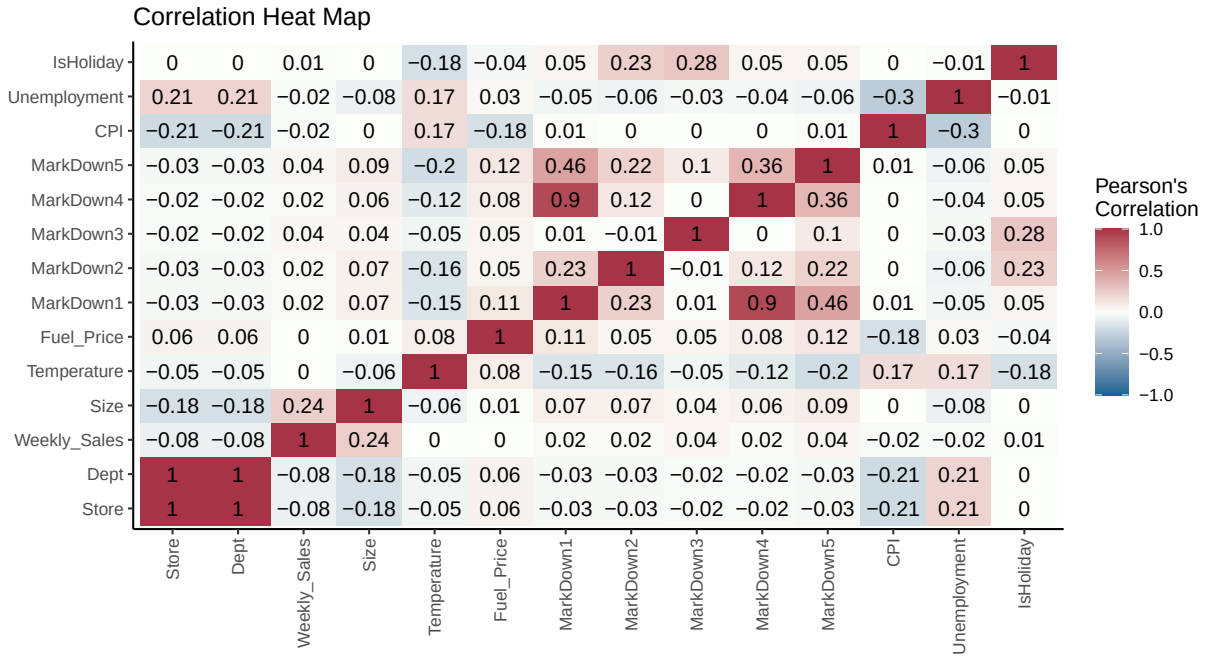
1. Correlation Matrix with Heatmap
2. Feature Importance with Boruta Algorithm

**Correlation Matrix with Heatmap**

A Correlation Heat map will show us how each feature moves in association to the other. Although we are working with time series data, the concept of correlation is the same with non-time series data.

As a refresher, correlation measures the strength of the linear relationship between two variables. The closer the correlation coefficient is to 1, the stronger the positive linear relationship. Conversely, the closer the correlation coefficient is to -1, the stronger the negative linear relationship. Finally, the closer the correlation coefficient is to 0, the weaker the linear relationship.

```
Cor_Df <- exp_df %>% select(-Date, -Type, -ID, -StoreDept) %>% mutate(
  Store = as.numeric(Store),
  Dept = as.numeric(Store),
  IsHoliday = as.numeric(IsHoliday)
) %>%
  cor() %>%
  round(2)
Cor_Df %>%
  melt() %>%
  ggplot(aes(Var1, Var2, fill = value)) +
  geom_tile() +
  geom_text(aes(Var1, Var2, label = value)) +
  labs(x = "", y = "", fill = "Pearson's \nCorrelation", title = "Correlation Heat Map") +
  scale_fill_gradient2(mid="#FBFEF9",low="#0C6291",high="#A63446", limits=c(-1,1)) +
  theme_classic() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```




As the purpose of this project is to project Weekly Sales, we will focus on the correlation of features with our target variable. By correlation, the most important feature is, "Size" representing a significant correlation coefficient of 0.24.

```
Top_Features <- Cor_Df["Weekly_Sales",] %>% as.matrix() %>% abs()
Top_Features[order(Top_Features[,1], decreasing = T),]
```

```
Weekly_Sales         Size        Store         Dept    MarkDown3    MarkDown5
        1.00         0.24         0.08         0.08         0.04         0.04
   MarkDown1    MarkDown2    MarkDown4          CPI Unemployment    IsHoliday
        0.02         0.02         0.02         0.02         0.02         0.01
 Temperature   Fuel_Price
        0.00         0.00
```

```
remove(Top_Features, Cor_Df)
```

### Feature Importance with Boruta Algorithm

We will build a model from the Boruta Algorithm which is based on a Random Forest Algorithm. Simply put, Random Forest Algorithms can be loosely thought of as a bunch of recursive if-then rules. What exactly a Random Forest Algorithm is, will be explained in greater depth in the following section.

Boruta Works through the following steps:

1. Creates Shadow Features which adds randomness to a data set by creating shuffled copies of all the features.
2. Trains a Random Forest Classifier and applies a feature importance measure. The default is Mean Decrease Accuracy where features with higher means are more important.
3. Through iteration, compares a real feature with its best shadow feature and removes features which are deemed as unimportant.
4. Concludes once the specified iterations are satisfied or when all features are confirmed or rejected.

A key feature of the Boruta Algorithm is that it is capable of capturing non-linear relationships through Random Forest Classifiers. This is ideal to us as we previously captured the linear relationships between features through Pearson's Correlation. This will give us a balanced perspective.

As outputs, Boruta classifies features to the following: Important, Tentative, Unimportant.

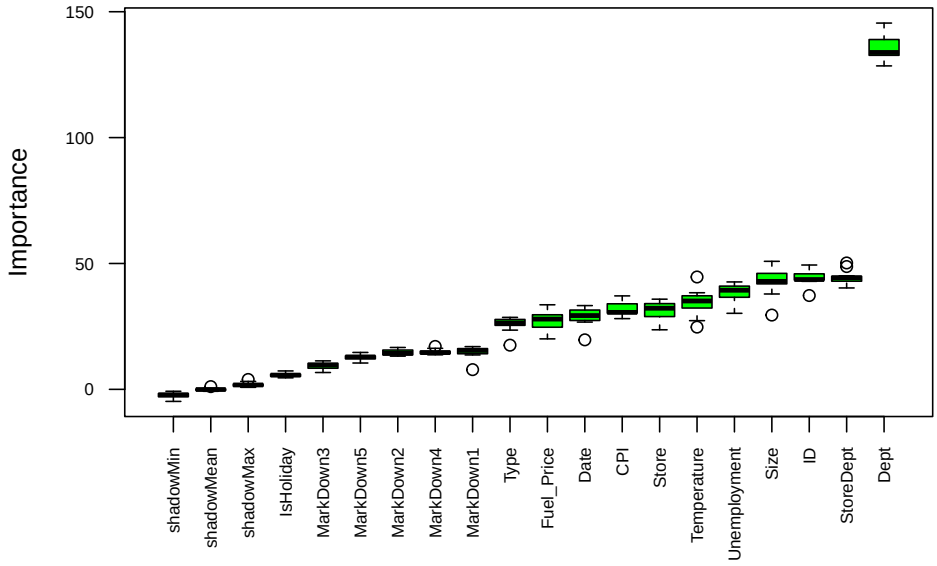
```
print(boruta)
```

```
Boruta performed 11 iterations in 1.200755 hours.
 17 attributes confirmed important: CPI, Date, Dept, Fuel_Price, ID and
12 more;
 No attributes deemed unimportant.
```

The following is a chart which represents the feature importance produced by Boruta. Each feature is given a color which corresponds to the classifications assigned by the algorithm. The features are also plotted with a box plot which corresponds to their minimal, average and maximum Z score. In a single statement, a Z score provides insight into how far from the mean a data point is.

- Blue: Shadow Attribute.
- Red: Rejected Feature.
- Yellow: Tentative Feature.
- Green: Confirmed Feature.

```
plot(boruta, xlab = "", las = 2, cex.axis = 0.7)
```




Through Boruta, we can observe that the most important features are "Dept", "StoreDept", "ID", and "Size". Interestingly, the "Size" feature was also considered an important feature through Pearson's Correlation. As features "Dept", "StoreDept", and "ID"will be used as classifiers in Machine Learning Models, we will use "Size" as our primary feature in making predictions.

```
boruta_stats <- as.vector(attStats(boruta))
boruta_stats[order(boruta_stats$medianImp, decreasing = TRUE),]
```

```
               meanImp  medianImp     minImp     maxImp normHits  decision
Dept         136.042699 133.812830 128.484080 145.489432        1 Confirmed
StoreDept     44.551698  44.308375  40.278547  50.207991        1 Confirmed
ID            44.287295  43.615235  37.266601  49.390419        1 Confirmed
Size          42.877847  42.884879  29.504071  50.856726        1 Confirmed
Unemployment  38.138933  39.368154  30.204593  42.669250        1 Confirmed
Temperature   34.287385  35.100272  24.729654  44.629785        1 Confirmed
Store         31.140866  32.219320  23.659380  35.820698        1 Confirmed
CPI           31.882429  30.562162  28.089870  37.124970        1 Confirmed
Date          28.757253  29.301884  19.664328  33.264997        1 Confirmed
Fuel_Price    27.233465  27.911433  20.050730  33.588706        1 Confirmed
Type          25.683901  26.389432  17.546749  28.591363        1 Confirmed
MarkDown1     14.781908  15.458321   7.825489  16.968872        1 Confirmed
MarkDown4     14.828566  14.601538  13.719998  17.040779        1 Confirmed
MarkDown2     14.634304  14.288300  13.189698  16.630567        1 Confirmed
MarkDown5     12.825604  12.887929  10.458467  14.656119        1 Confirmed
MarkDown3      9.353051   9.646833   6.683031  11.311214        1 Confirmed
IsHoliday      5.773572   5.660351   4.578393   7.312796        1 Confirmed
```

```
remove(boruta_stats, boruta)
```

## Modeling

### Linear Regression

The first model we will use for our Time Series forecast is Linear Regression. Fundamentally, regressions determine the relationship between a dependent variable and a set of independent variables. This can be represented in a simple linear equation:

$$Y = a + bX$$

- Y : Dependent Variable/Value.
- a : Intercept Coefficient.
- b : Slope of the Line.
- X : Independent Variable/Value.

For this project, we will use a Multiple Linear Regression model.

As the name would imply, a Multiple Linear Regression model uses several variables to predict the dependent variable. This is illustrated in the following equation:

$$Y = a + B_1X_1 + B_2X_2 + B_3X_3$$

In our Multiple Linear Regression model, we will use the Feature determined from Pearson's Correlation Coefficient and the Boruta Algorithm, "Size". It is important to note that adding too many features can result in overfitting; meaning that our model will fit our training set very well but will not perform well on our test set. Therefore, we will proceed with a single predictor.

### Random Forest Regressor

Simply put, Random Forest is an ensemble (combined predictions from two or more models) of Decision Tree algorithms.

A Decision Tree algorithm works by creating a "tree" of all the possible solutions to a decision based on defined conditions. It operates very much like recursive if-then rules. A collection of trees is a "forest" hence Random Forest.

Random Forest is a bagging technique. It combines the results of all decision trees, selecting observations with replacement randomly, to get a generalized result.

### Gradient Boosting Regressor

Gradient Boosting, also known as Extreme Gradient Boosting (XGBoost) is an ensemble of decision tree algorithms where new trees fix the errors of trees that are already part of the model. Trees are continuously added until improvements can no longer be made.

Gradient Boosting consists of two sub-terms, which are gradient and boosting:

- Boosting: Fundamentally, boosting algorithms seek to improve predictions by training a sequence of weak models, with each model compensating the weakness of its predecessors. The definition of weakness varies across algorithms. For Gradient Boosting, weakness is a numerical optimization problem with an objective to minimize a loss function (loss of a single training set) using gradient descent.
- Gradient Descent: Is an optimization algorithm used to find the values of coefficients (parameters) of a function which minimizes a cost function (the average loss over the entire training dataset).

## Data Manipulation for Modeling

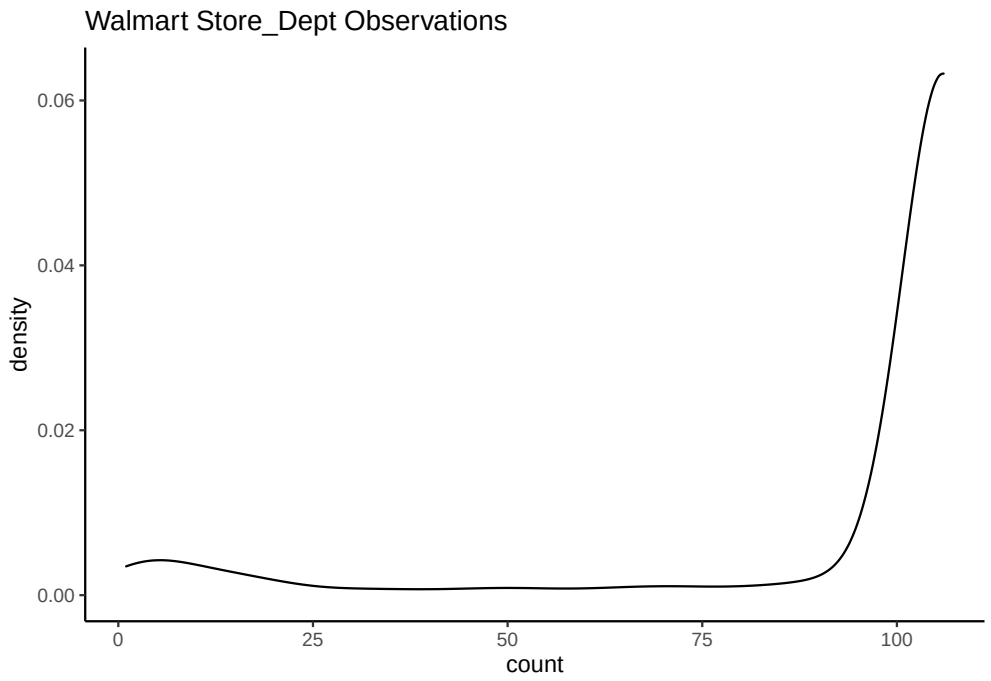
### Interpolation

Shown in the chart below, we observe that the distribution of observations for each unique Department per Walmart Store is not even. While the vast majority of stores have 106 observations, which is the total number of weeks in the period assigned to our exploratory dataset, we cannot ignore the stores that have missing values. A solution to this problem is Interpolation and Imputation, however for this exercise we will use two variations of Interpolation.

Interpolation is the process of determining unknown values by using linear polynomials to construct data points within a range of known data points.

Our first approach will be using Loess Decomposition & Interpolation. This algorithm removes the seasonal component, performs Interpolation and finally adds the seasonal component back again. Our second approach will be regular Interpolation. Where either of these methodologies cannot fill unknown values, we will carry forward the last known observation.

```
exp_df %>%
  group_by(StoreDept) %>%
  summarise(count = n()) %>%
  ggplot(aes(count)) +
  geom_density() +
  theme_classic() +
  ggtitle("Walmart Store_Dept Observations")
```



# Results

## Function for Model Evaluation

We start off by creating our loss function, Weighted Mean Absolute Error (WMAE), where holiday weeks are weighted five times as much as non-holiday weeks.

```
WMAE <- function(pred, actual) {
  weights <- if_else(actual$IsHoliday, 5,1)
  (sum(weights * abs(pred$Weekly_Sales - actual$Weekly_Sales)) / sum(weights))
}
```

## Interpolation

Prior to building our models, we perform Interpolation to fill missing sales observations. To perform Interpolation we first convert our data to wide format, creating columns for each StoreDept. By having a column for each StoreDept, missing observations can be found through NAs. After filling observations through Seasonally Decomposed Interpolation, Linear Interpolation, and finally carrying over last observations, we convert our dataframe back to Tidy Format and add features which will act as predictors.

```
## Convert Data to wide format, spreading by StoreDept
wide_exp_df <- exp_df %>%
    group_by(StoreDept, Date) %>%
    summarise(Weekly_Sales = sum(Weekly_Sales))%>%
    ungroup() %>%
    spread(key = StoreDept, value = Weekly_Sales)
sum(is.na(wide_exp_df))
```

```
[1] 38794
```

```
## Fill values with Seasonally Decomposed Interpolation
wide_exp_df <- na_seadec(wide_exp_df, algorithm = "interpolation")
sum(is.na(wide_exp_df))
```

```
[1] 7317
```

```
## Fill values with Linear Interpolation
wide_exp_df <- na_interpolation(wide_exp_df, option = "linear")
sum(is.na(wide_exp_df))
```

```
[1] 3885
```

```
## Fill remaining values with Last Observation Carried Forward
wide_exp_df <- na_locf(wide_exp_df)
sum(is.na(wide_exp_df))
```

```
[1] 0
```

```
## Convert Back to Tidy Format
tidy_df <- wide_exp_df %>%
  pivot_longer(!Date, names_to = "StoreDept", values_to = "Weekly_Sales") %>%
  arrange(StoreDept)
remove(wide_exp_df)

## Add Features Back tidy_df
tidy_df <- tidy_df %>%
  separate(StoreDept, c("Store", "Dept"), "_", remove = FALSE) %>%
  mutate(Store = as.numeric(Store),
         Dept = as.numeric(Dept),
         StoreDept = as_factor(StoreDept))

stores_df <- read_csv("data/stores.csv")
features_df <- read_csv("data/features.csv")

tidy_df <- left_join(tidy_df, stores_df, by = "Store")
tidy_df <- left_join(tidy_df, features_df, by = c("Store", "Date"))
tidy_df <- tidy_df %>%
  select(Date, StoreDept, Store, Dept, Weekly_Sales, Size, IsHoliday)

remove(stores_df, features_df, exp_df)
```

## Multiple Linear Regression

Our first Model is the simplest, Multiple Linear Regression, with Size as our predictor.

```
W_LM <- lm(Weekly_Sales ~ Size, data = tidy_df)
W_LM_Model <- predict(W_LM, test_df[,c("Size")])
W_LM_Yhat <- data.frame(ID = test_df$ID,
                        Weekly_Sales = W_LM_Model)

results <- data_frame(
  Model = "Multiple Linear Regression",
  WMAE = WMAE(W_LM_Yhat, test_df)
)

results %>% knitr::kable()
```

| Model | WMAE |
|---|---|
| Multiple Linear Regression | 14014.95 |

```
remove(W_LM, W_LM_Model, W_LM_Yhat)
```

## Model Building

### Data Preparation

In building our Machine Learning Algorithms, we will use the help of a library Modeltime. Modeltime allows the user to develop multiple Machine Learning Algorithms for Time Series Forecasting. The first

steps in using Modeltime is data preparation. Here, we first extend our time series to as far as we wish to predict. Next, we identify our ID column which acts as our "grouper" which is StoreDept and again restate our extension. Finally, as we stated in the beginning of this project, we split our data into a Train and Validation set. We split by equal parts, 53 weeks, ensuring that we have a full year to fit our models.

```
## Data Preparation
nested_table <- tidy_df %>%
  extend_timeseries(
    .id_var = StoreDept,
    .date_var =  Date,
    .length_future = 37 # Extend forecast by 37 weeks
  ) %>%
  nest_timeseries( # create a nested tibble for each StoreDept with past and future
    .id_var = StoreDept,
    .length_future = 37,
    .length_actual = 106
  ) %>%
  split_nested_timeseries( # Creates training and testing indicies
    .length_test = 53
  )
```

### Model Pre-processing

Here we create our "recipe" which is a description of the steps to prepare a data set for modelling. A recipe is written similarly to a regression formula starting with our dependent variable (Weekly_Sales), followed by our independent variable (Date) and predictor (Size).

We also perform some Feature Engineering using the Date variable. Feature Engineering is a technique to generate features that can be used in supervised learning. We break apart the Date variable into various pieces as demonstrated in the table below. These pieces can improve the results of the algorithm by detecting seasonal trends.

```
# Recipe for future prediction
model_recipe <- recipe(Weekly_Sales ~ Date + Size, data = extract_nested_train_split(nested_table)) %>%
  step_timeseries_signature(Date) %>% # Feature Engineering
  step_rm(matches("(hour)|(minute)|(second)|(am.pm)|(xts)|(iso)|(lbl)"), Date) %>%
  step_zv(all_predictors()) %>%
  step_dummy(all_nominal())

## Visualize Recipe
model_recipe %>% prep() %>% juice()
```

```
# A tibble: 53 x 16
   Weekly_Sales Date_index.num Date_year Date_half Date_quarter Date_month
          <dbl>          <dbl>     <int>     <int>        <int>      <int>
 1       24924.     1265328000      2010         1            1          2
 2       46039.     1265932800      2010         1            1          2
 3       41596.     1266537600      2010         1            1          2
 4       19404.     1267142400      2010         1            1          2
 5       21828.     1267747200      2010         1            1          3
 6       21043.     1268352000      2010         1            1          3
 7       22137.     1268956800      2010         1            1          3
 8       26229.     1269561600      2010         1            1          3
```

```
 9       57258.     1270166400      2010         1            2          4
10       42961.     1270771200      2010         1            2          4
# ... with 43 more rows, and 10 more variables: Date_day <int>,
#   Date_mday <int>, Date_qday <int>, Date_yday <int>, Date_mweek <int>,
#   Date_week <int>, Date_week2 <int>, Date_week3 <int>, Date_week4 <int>,
#   Date_mday7 <int>
```

### Regressors

Here we create workflows which will fit Random Forest and Gradient Boosting models to our data so we can generate forecasts. Note that "ranger" is just a fast implementation of Random Forest.

```
## Random Forest Regressor
rfr <- workflow() %>%
  add_model(rand_forest("regression") %>% set_engine("ranger")) %>%
  add_recipe(model_recipe)

## Gradient Boosting Regressor
xgb <- workflow() %>%
  add_model(boost_tree("regression") %>% set_engine("xgboost")) %>%
  add_recipe(model_recipe )

remove(model_recipe)
```

### Fit Model on Training Set

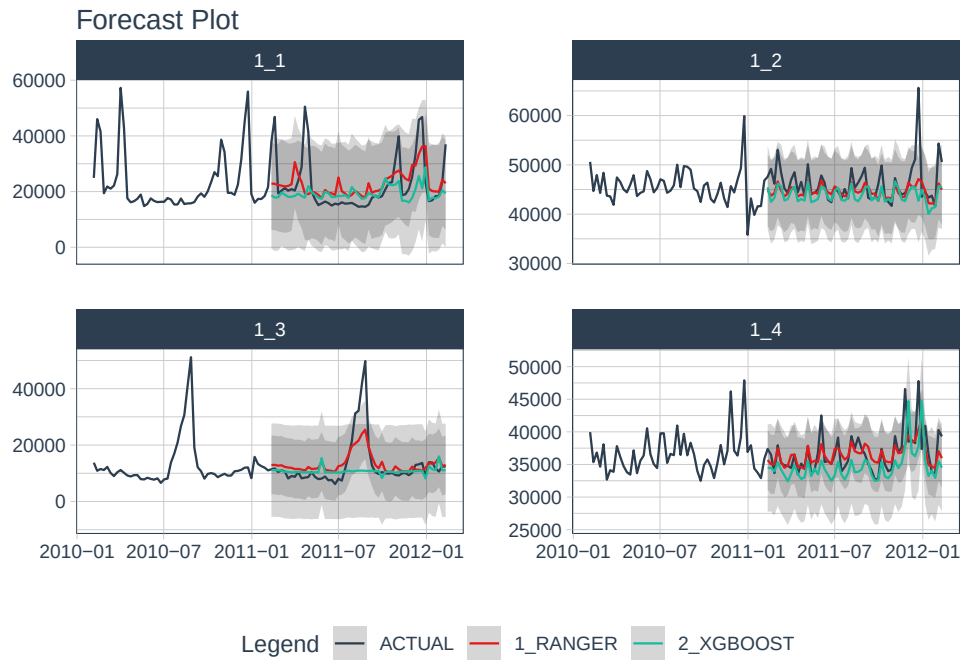
```
set.seed(1)
nested_models <- modeltime_nested_fit(
  nested_data = nested_table,
  # Add Models
  rfr,
  xgb)

remove(rfr, xgb)
```

### Extract Training Set

Here we visualize how our model fitted our Training Set. In this visualization we look at Departments 1-4 in Store 1.

```
nested_models %>%
  extract_nested_test_forecast() %>%
  group_by(StoreDept) %>%
  filter(StoreDept %in% c("1_1", "1_2", "1_3", "1_4")) %>%
  plot_modeltime_forecast(
    .facet_ncol  = 2,
    .interactive = FALSE
  )
```




## Random Forest Result on Training Set

The first results are in. On the training set, we see a major improvement from our Multiple Linear Regression model.

```
F_RandomForest <- filtered_fit_models %>%
  select(ID, RANGER)

F_RandomForest <- inner_join(F_RandomForest, tidy_df, by = "ID") %>%
  select(ID, RANGER)

test_df_RF <- inner_join(F_RandomForest, tidy_df, by = "ID") %>%
  select(ID, Weekly_Sales, IsHoliday)

colnames(F_RandomForest)[2] <- "Weekly_Sales"

results <- bind_rows(results,
                     data_frame(
                       Model = "Random Forest - Training Set",
                       WMAE = WMAE(F_RandomForest, test_df_RF)
                     ))

results %>% knitr::kable()
```

| Model | WMAE |
|---|---|
| Multiple Linear Regression | 14014.948 |
| Random Forest - Training Set | 2248.181 |

## Gradient Boosting Results on Test Set

With Gradient Boost, we also see a major improvement from Multiple Linear Regression, however, our results are not as good as those from Random Forest.

```
F_XGBOOST<- filtered_fit_models %>%
  select(ID, XGBOOST)

F_XGBOOST <- inner_join(F_XGBOOST, tidy_df, by = "ID") %>%
  select(ID, XGBOOST)

test_df_XG <- inner_join(F_XGBOOST, tidy_df, by = "ID") %>%
  select(ID, Weekly_Sales, IsHoliday)

colnames(F_XGBOOST)[2] <- "Weekly_Sales"

results <- bind_rows(results,
                     data_frame(
                       Model = "Gradient Boosting - Training Set",
                       WMAE = WMAE(F_XGBOOST, test_df_XG)
                     ))

results %>% knitr::kable()
```

| Model | WMAE |
|---|---|
| Multiple Linear Regression | 14014.948 |
| Random Forest - Training Set | 2248.181 |
| Gradient Boosting - Training Set | 2779.179 |

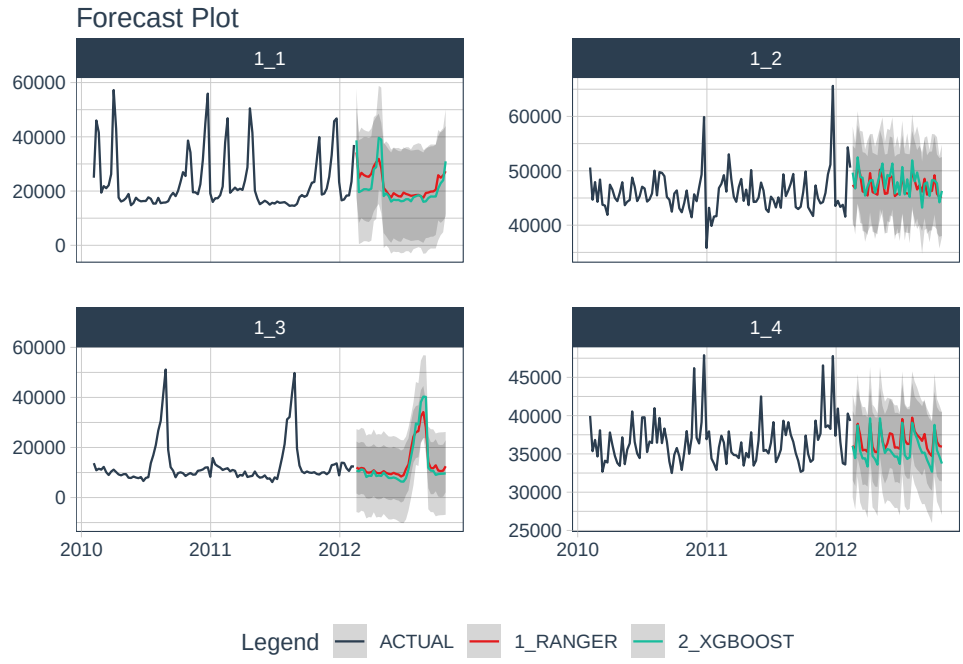
### Refit Model for Future Forecast

```
set.seed(1)
refit_models <- nested_models %>%
  modeltime_nested_refit(
    control = control_nested_refit()
  )
```

### Extract Future Forecast

With refitting on our entire Exploratory Data Set done, we can now visualize our future forecast for Departments 1-4 in Store 1.

```
refit_models %>%
  extract_nested_future_forecast() %>%
  group_by(StoreDept) %>%
  filter(StoreDept %in% c("1_1", "1_2", "1_3", "1_4")) %>%
  plot_modeltime_forecast(
    .interactive = FALSE,
    .facet_ncol = 2
  )
```




## Random Forest Results on Test Set

We see a massive improvement in our results from Multiple Linear Regression and the training set results for Random Forest.

```
F_RandomForest<- filtered_refit_models %>%
  select(ID, RANGER)

F_RandomForest <- inner_join(F_RandomForest, test_df, by = "ID") %>%
  select(ID, RANGER)

test_df_RF <- inner_join(F_RandomForest, test_df, by = "ID") %>%
  select(ID, Weekly_Sales, IsHoliday)

colnames(F_RandomForest)[2] <- "Weekly_Sales"

results <- bind_rows(results,
                     data_frame(
                       Model = "Random Forest - Test Set",
                       WMAE = WMAE(F_RandomForest, test_df_RF)
                     ))

results %>% knitr::kable()
```

| Model | WMAE |
|---|---|
| Multiple Linear Regression | 14014.948 |
| Random Forest - Training Set | 2248.181 |
| Gradient Boosting - Training Set | 2779.179 |

| Model | WMAE |
|---|---|
| Random Forest - Test Set | 1677.724 |

### Gradient Boosting Results on Test Set

The final results are in. We see a slight decrease in accuracy from Random Forest as we did from the training sets, but overall, both models outperformed our Linear Regression Model.

```
F_XGBOOST<- filtered_refit_models %>%
  select(ID, XGBOOST)

F_XGBOOST <- inner_join(F_XGBOOST, test_df, by = "ID") %>%
  select(ID, XGBOOST)

test_df_XG <- inner_join(F_XGBOOST, test_df, by = "ID") %>%
  select(ID, Weekly_Sales, IsHoliday)

colnames(F_XGBOOST)[2] <- "Weekly_Sales"

results <- bind_rows(results,
                     data_frame(
                       Model = "Gradient Boosting - Test Set",
                       WMAE = WMAE(F_XGBOOST, test_df_XG)
                     ))

results %>% knitr::kable()
```

| Model | WMAE |
|---|---|
| Multiple Linear Regression | 14014.948 |
| Random Forest - Training Set | 2248.181 |
| Gradient Boosting - Training Set | 2779.179 |
| Random Forest - Test Set | 1677.724 |
| Gradient Boosting - Test Set | 1703.672 |

## Conclusion

For this challenge we constructed over 3,000 forecasts for three machine learning algorithms to predict the Weekly Sales for Departments in 45 different Walmart Stores.

We started off by splitting our dataset into an exploratory and test set through an out-of-time method by keeping the first two years of the dataset in the exploratory set. We proceeded to Data Exploration by performing Time Series Decomposition where we observed that our time series has Additive Seasonal Components. Moving on to Feature Selection, we performed two variations: Correlation Matrix and Boruta Algorithm. A distinct take away through performing Feature Selection was that “Size” was a strong predictor of Weekly Sales. Prior to to building models for making predictions, we observed missing Weekly Sales values in our dataset. We filled these missing values through two variations of Interpolation. These variations were Loess Decomposition & Interpolation and Regular Interpolation. Now having a complete dataset, we proceeded to building Multiple Linear Regression, Random Forest and Gradient Boosting Algorithms. Random Forest was our best performing algorithm with a WMAE of 1677.7243374

Looking forward, there are many methodologies we could implement to improve our forecasts. To name a few:

- Differencing: Making our data stationary which means that the mean & variance does not change over time. As we observed through Time Series Decomposition, our dataset is not stationary and does have a trend and strong seasonal components. The advantage is that we are removing the effect of time and can thus approach forecasting as a standard probability distribution function.
- Simpler Models: In this specific challenge we wanted to explore forecasting using Machine Learning Algorithms. However, simpler statistical models such as ARIMA and Prophet models can be far more efficient and deliver accurate results.