M5 FORECASTING - ACCURACY

A variety of items competes inside the supermarket where a retailer places a different lease price on its shelves. Supermarket shelves store the items in the supermarket for sale. This shelf mainly impacts product visibility, saleability, and customer engagement. Product manufacturers and retailers put efforts into their products - marketing, sales, manufacturing, logistics, and cost management. These factors encompass to one single thing - the unit sales.

Imagine this conversation:

CEO: "How much unit sales did we make for January at Store 10 at this food product?"

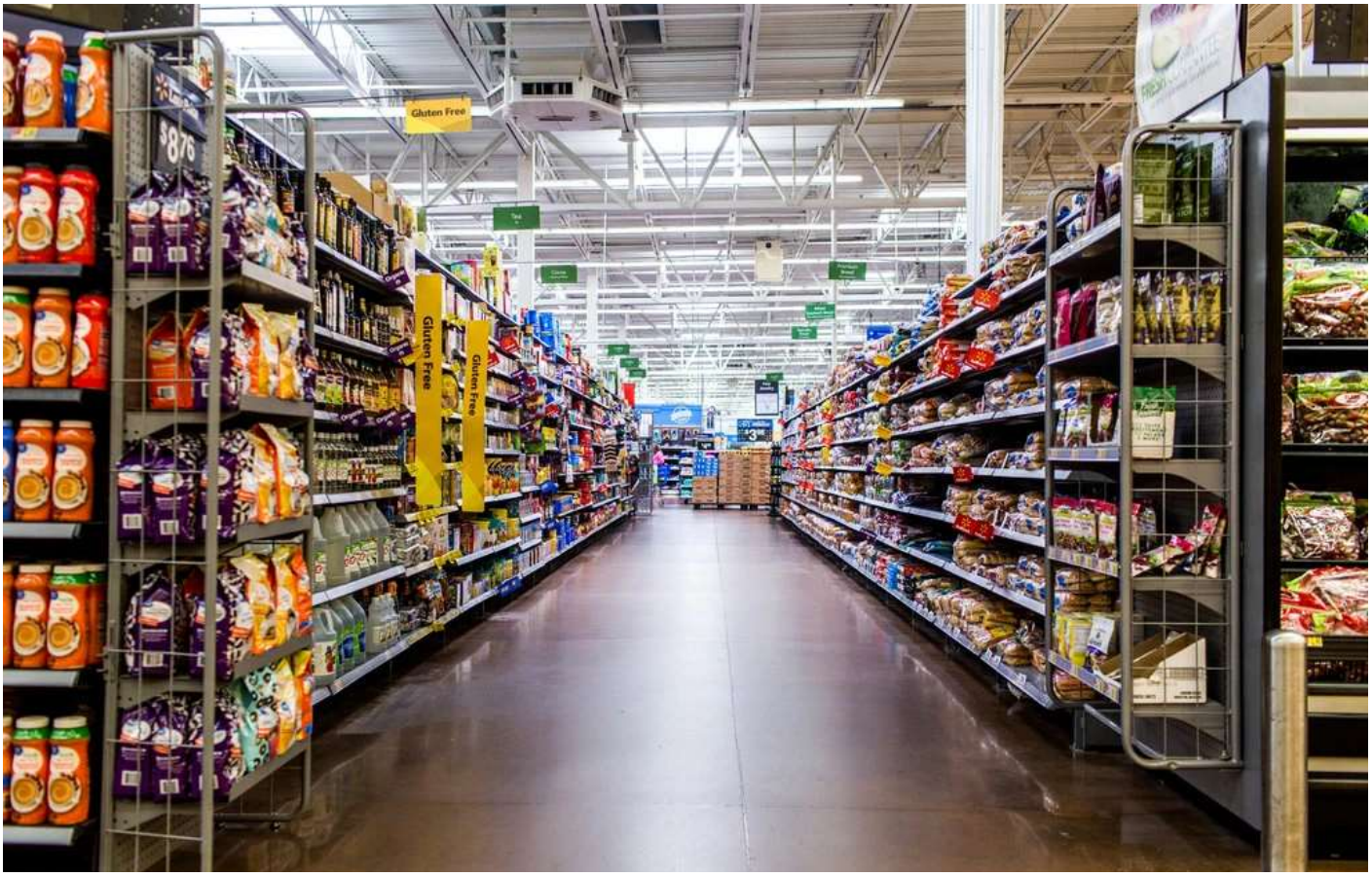
Salesperson: "Approximately 20,000 USD for 5,000 food products sold, Sir."

Accountant: "How many did we make and ship for Store 10 for January?"

Manufacturer: "We have manufactured and shipped 10,000 food products, Sir. More than what was sold."

Accountant: "Oh boy! Then, those food products have a very short expiration and the company might get a loss for those products when they're unsold after expiration."

See how the conversation went. Every product is unique and companies want to forecast sales to have a revenue target, plan product manufacturing and shipping and cut down waste and product loss. This all boils down to the objective of the M5 Forecasting case.



About the Case

The case is about the time-series sales data of Walmart. The Makridakis Open Forecasting Center (MOFC) at the University of Nicosia conducts cutting-edge forecasting research and provides business forecast training. It helps companies achieve accurate predictions, estimate the levels of uncertainty, avoiding costly mistakes, and apply best forecasting practices.

Objective

The aim of this case is to advance the theory and practice of forecasting by identifying the methods that provide the most accurate point forecasts using the retail data of Walmart.

Hypothesis

I believe that the unit sales in terms of the retail supermarket are affected by the following:

1. **Time -** Spending behavior is expected to be different for each day in a week due to different time availability of the people to go and do groceries.
2. **Event/Holidays -** Depending on the event or holidays, the people usually take time to spend such events or holidays through the feast, special dinner, reunions, and so on and because of this, the people spend more than their usual spending.
3. **Location -** The location greatly affects unit sales because each location has a unique group of people. A location may have residents who are not price-sensitive and choose quality regardless price. In another location, a different group of residents may be price sensitive.
4. **Product Pricing -** The price of the product greatly affect the spending behavior of the customer and likewise, sales of the product.
5. **Product Category -** Each item is categorized by product category. Where the item lies in each category affects the perception of the people towards it which consequently, affects their buying behavior.
6. **Brand -** The brand of the product creates a relationship between the product and the customer. Brand composed of quality, functionality, purpose, and benefits of the product given to the customer and as soon as it gains a good branding, the customer will have a constant buying behavior towards a product.

OSEMN Pipeline

I will be using the OSEMN Pipeline which I learned from Randy Lao at this link (https://medium.com/breathe-publication/life-of-data-data-science-is-osemn-f453e1febc10 (https://medium.com/breathe-publication/life-of-data-data-science-is-osemn-f453e1febc10)):

1. Obtaining the data
2. Scrubbing or cleaning the data
3. Exploring the data to identify significant behavior, trends, and insights.
4. Modelling the data for prediction.
5. Interpreting the data and results.

Obtaining the Data

About the data, there are three datasets that I will gather: train set, sell_prices, and calendar.

- **Calendar dataset** contains all information about time such as weekday, week, date, and year.
- **Sell Prices** contains information about the price of the product for each day in five years. The price is based on the combination of item - location - store - time - event.
- **Train** contains the sales data for each store in five years for each product.

In [1]:
```
## Loading the libraries
library(reshape2) ## for data cleaning
library(data.table) ## for quick and RAM efficient loading
library(tidyr) ## for data cleaning
library(tidyverse) ## for data cleaning and piping
library(lubridate) ## for date class transformation
library(splitstackshape) ## for stratified sampling
library(ggplot2) ## for data visualizations and exploration
library(ggpubr) ## for compiling and combining the plots
library(ggcorrplot) ## for correlation
library(xgboost) ## for modelling
library(Matrix) ## for converting data to sparse matrix
library(RcppRoll)
library(zoo)
library(SHAPforxgboost) ## SHAP for xgboost
library(writexl) ## for exporting in excel

## Loading the files
sell_prices <- fread("../input/m5-forecasting-accuracy/sell_prices.csv", sep =
",", na.strings = c("NA", ""), header = TRUE, stringsAsFactors = TRUE)

calendar <- fread("../input/m5-forecasting-accuracy/calendar.csv", sep = ",",n
a.strings = c("NA", ""), header = TRUE, stringsAsFactors = TRUE)

train <- fread("../input/m5-forecasting-accuracy/sales_train_validation.csv",
 sep = ",", na.strings = c("NA", ""), header = TRUE, stringsAsFactors = TRUE)
```

```
Attaching package: ‘data.table’


The following objects are masked from ‘package:reshape2’:

    dcast, melt


Attaching package: ‘tidyr’


The following object is masked from ‘package:reshape2’:

    smiths


── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0
──

✓ ggplot2 3.3.1.9000     ✓ dplyr   1.0.0
✓ tibble  3.0.1          ✓ stringr 1.4.0
✓ readr   1.3.1          ✓ forcats 0.5.0
✓ purrr   0.3.4

── Conflicts ────────────────────────────────────────── tidyverse_conflicts()
──
✕ dplyr::between()   masks data.table::between()
✕ dplyr::filter()    masks stats::filter()
✕ dplyr::first()     masks data.table::first()
✕ dplyr::lag()       masks stats::lag()
✕ dplyr::last()      masks data.table::last()
✕ purrr::transpose() masks data.table::transpose()


Attaching package: ‘lubridate’


The following objects are masked from ‘package:data.table’:

    hour, isoweek, mday, minute, month, quarter, second, wday, week,
    yday, year


The following objects are masked from ‘package:base’:

    date, intersect, setdiff, union


Attaching package: ‘xgboost’


The following object is masked from ‘package:dplyr’:

    slice
```

```
Attaching package: ‘Matrix’

The following objects are masked from ‘package:tidyr’:

    expand, pack, unpack


Attaching package: ‘zoo’

The following objects are masked from ‘package:base’:

    as.Date, as.Date.numeric
```

Cleaning the Data

After obtaining the data, we will take the following steps:

1. Check out the data structure;
2. Transform the data into the right structure;
3. Check out the NAs and missing values for each data;
4. Fill out the NA values and missing values;
5. Review the data and check for additional features that can be created and added;
6. Throw away features that will not be used in data exploration and modelling to reduce RAM usage.

In [2]:
```
## Checking out the dimensions of each data
data.frame(calendar = dim(calendar), sell_prices = dim(sell_prices), train = d
im(train), row.names = c("Rows", "Columns"))
```

A data.frame: 2 × 3

	calendar	sell_prices	train
	<int>	<int>	<int>
Rows	1969	6841121	30490
Columns	14	4	1919

Based on the calendar table, there are 1,969 days. In the train set, there are about 30,490 unique products which are sold across all of the time periods.

Data Transformation - Part 1

In this section, the training dataset is transformed by melting the day's unit sales columns into two columns: day (the day id) and Unit Sales (the day's unit sales).

The calendar data is combined with the train data using day as reference.

In [3]:
```
## Transforming the train set into the right structure by transferring the d c
olumns into rows
train <- train %>%
            melt(id.vars = c("id", "item_id", "dept_id", "cat_id", "store_id",
"state_id"),
                 variable.name = "day",
                 value.name = "Unit_Sales")

## Merging the calendar data into the train data
train <- left_join(train, calendar,
                   by = c("day" = "d"))
```

Checking Null Values

In this section, we will check for NA values in each columns for each table.

In this case, there are two kinds of null values:

1. **No Values** - means Missing Information
2. **Zero Values** - This is applicable for numeric or integer variables which are the item sales price and unit sales.

In [4]:
```
## Calculating the number of NA values for each column in each dataset
data.frame(calendar = colSums(is.na(calendar)))
data.frame(sell_prices = colSums(is.na(sell_prices)))
data.frame(train = colSums(is.na(train)))
```

A data.frame: 14 × 1

	calendar
	<dbl>
date	0
wm_yr_wk	0
weekday	0
wday	0
month	0
year	0
d	0
event_name_1	1807
event_type_1	1807
event_name_2	1964
event_type_2	1964
snap_CA	0
snap_TX	0
snap_WI	0

A data.frame: 4 × 1

	sell_prices
	<dbl>
store_id	0
item_id	0
wm_yr_wk	0
sell_price	0

A data.frame: 21 × 1

	train
	<dbl>
id	0
item_id	0
dept_id	0
cat_id	0
store_id	0
state_id	0
day	0
Unit_Sales	0
date	0
wm_yr_wk	0
weekday	0
wday	0
month	0
year	0
event_name_1	53631910
event_type_1	53631910
event_name_2	58205410
event_type_2	58205410
snap_CA	0
snap_TX	0
snap_WI	0

Null Values

There are about 1,807 and 1,964 nulls for event_1 and event_2, respectively. This is expected because only a few holidays and occasions are observed in a 360 calendar day.

In [5]:
```
## Calculating the missing values for sell_price column
data.frame(`Unit Sales`= sum(train$Unit_Sales == 0)/nrow(train)*100,
           `Sale Price`= sum(sell_prices$sell_price == 0)/nrow(sell_prices)*10
0,
           row.names = "% of zero values")
```

A data.frame: 1 × 2

	Unit.Sales	Sale.Price
	<dbl>	<dbl>
% of zero values	68.19628	0

Zero Values

Our table shows that there are absolutely no zero values for the Item Selling Price. However, Unit Sales show 68.20% zero unit sales which tells us that 68.20% of our dependent variable in our data have zero values.

This percentage demands further investigation. My hypotheses are:

> Zero unit sales may indicate that the product has not yet been launched in the branches in a specific time and branch.
>
> Zero unit sales may indicate seasonal sales or rare sales. For example, shark meats are sold during the shark's migration season which may last only for a very short period of time.

In the following section, we will be counting the number of zero unit sales across all years. Next, we will classify whether the product was available or unavailable by checking the number of counts. If the number of counts equal to the total number of days per year, then the item will be classified as **Product Unavailable**. Otherwise, **Product Available**.

In [6]:
```
## Calculating the number of zero unit sales for each item
Rank_Zero_Sales <- train %>%
                        group_by(year, month, store_id, cat_id, item_id) %>%
                        summarize(Zero_Sales_Days = sum(Unit_Sales == 0)) %>%
                        as.data.frame()

Rank_Zero_Sales_2011 <- Rank_Zero_Sales %>%
                            filter(year == "2011") %>%
                            spread(month, Zero_Sales_Days) %>%
                            mutate(Total_Unsold_Days = rowSums(select(., 5:16
)),
                            Classification = ifelse(Total_Unsold_Days == 337,
"Product Unavailable", "Product Available"))

Rank_Zero_Sales_2012 <- Rank_Zero_Sales %>%
                            filter(year == "2012") %>%
                            spread(month, Zero_Sales_Days) %>%
                            mutate(Total_Unsold_Days = rowSums(select(., 5:16
)),
                            Classification = ifelse(Total_Unsold_Days == 366,
"Product Unavailable", "Product Available"))

Rank_Zero_Sales_2013 <- Rank_Zero_Sales %>%
                            filter(year == "2013") %>%
                            spread(month, Zero_Sales_Days) %>%
                            mutate(Total_Unsold_Days = rowSums(select(., 5:16
)),
                            Classification = ifelse(Total_Unsold_Days == 365,
"Product Unavailable", "Product Available"))

Rank_Zero_Sales_2014 <- Rank_Zero_Sales %>%
                            filter(year == "2014") %>%
                            spread(month, Zero_Sales_Days) %>%
                            mutate(Total_Unsold_Days = rowSums(select(., 5:16
)),
                            Classification = ifelse(Total_Unsold_Days == 365,
"Product Unavailable", "Product Available"))

Rank_Zero_Sales_2015 <- Rank_Zero_Sales %>%
                            filter(year == "2015") %>%
                            spread(month, Zero_Sales_Days) %>%
                            mutate(Total_Unsold_Days = rowSums(select(., 5:16
)),
                            Classification = ifelse(Total_Unsold_Days == 365,
"Product Unavailable", "Product Available"))

Rank_Zero_Sales_2016 <- Rank_Zero_Sales %>%
                            filter(year == "2016") %>%
                            spread(month, Zero_Sales_Days) %>%
                            mutate(`5` = "NA",
                                   `6` = "NA",
                                   `7` = "NA",
                                   `8` = "NA",
                                   `9` = "NA",
                                   `10` = "NA",
                                   `11` = "NA",
```

```
                                    `12` = "NA") %>%
                                as.data.frame() %>%
                                mutate(Total_Unsold_Days = rowSums(select(., 5:8
)),
                                       Classification = ifelse(Total_Unsold_Days =
= 115, "Product Unavailable", "Product Available"))

## Combining the yearly classifications
Product_Avail_Unavail <- rbind(Rank_Zero_Sales_2011,
                               Rank_Zero_Sales_2012,
                               Rank_Zero_Sales_2013,
                               Rank_Zero_Sales_2014,
                               Rank_Zero_Sales_2015,
                               Rank_Zero_Sales_2016)

## Converting the classification into a factor class
Product_Avail_Unavail$Classification <- as.factor(Product_Avail_Unavail$Classi
fication)

## Cleaning the memory for RAM
gc()
```

```
`summarise()` regrouping output by 'year', 'month', 'store_id', 'cat_id' (ove
rride with `.groups` argument)
```

A matrix: 2 × 6 of type dbl

	used	(Mb)	gc trigger	(Mb)	max used	(Mb)
Ncells	2314343	123.6	12590604	672.5	12708920	678.8
Vcells	650327643	4961.7	1791804960	13670.4	2052651054	15660.5

Finally, the table below shows us the number of products which are available for sale and unavailable per year.

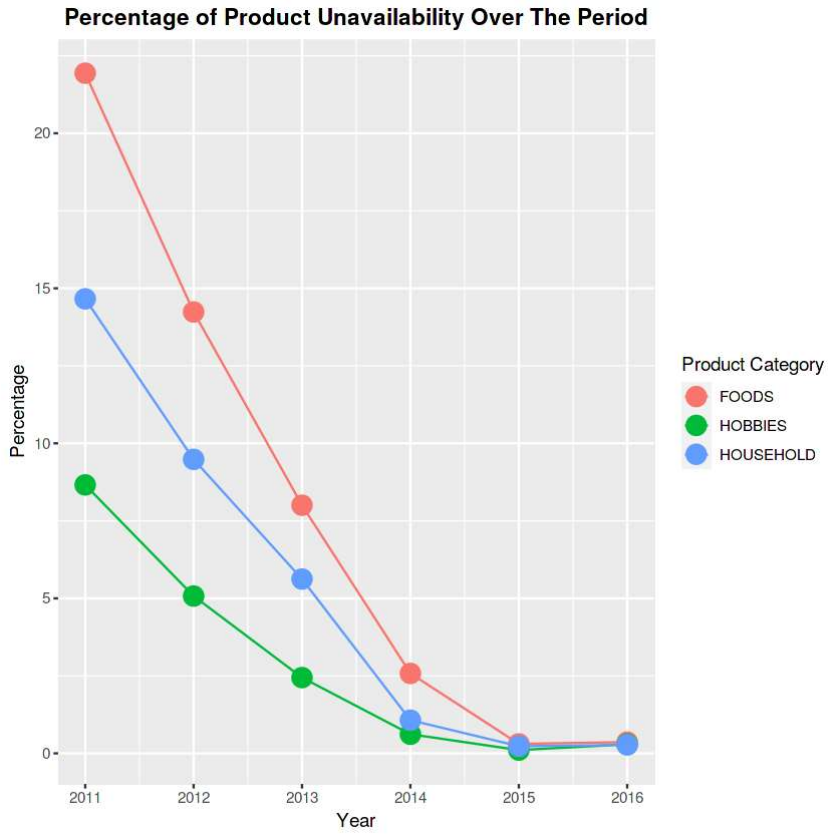
Items under the FOOD and HOUSEHOLD categories have the highest percentage of unavailability at 2011. After 2011, all items are getting available in the market as shown how the % of unavailability and % of availability change.

In [7]:
```
## Calculating the percentage of products available and unavailable across the
five year period
Product_Avail_Comparison <- Product_Avail_Unavail %>%
                                group_by(year, cat_id) %>%
                                summarize(`Product Unavailable` = sum(Classifi
cation == "Product Unavailable"),
                                          `Product Available` = sum(Classifica
tion == "Product Available"),
                                          `% of Unavailability` = sum(Classifi
cation == "Product Unavailable")/30490*100,
                                          `% of Availability` = sum(Classifica
tion == "Product Available")/30490*100) %>%
                                arrange(cat_id)

Product_Avail_Comparison

## Plotting the product unavailability
ggplot(data = Product_Avail_Comparison, aes(x = year,y = `% of Unavailability`
, label = `% of Unavailability`)) +
    geom_line(aes(y = `% of Unavailability`, color = cat_id)) +
    geom_point(aes(color = cat_id), size = 5, stat = "identity") +
    scale_colour_discrete("Product Category") +
    xlab("Year") +
    ylab("Percentage") +
    ggtitle("Percentage of Product Unavailability Over The Period") +
    theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

```
`summarise()` regrouping output by 'year' (override with `.groups` argument)
```

A grouped_df: 18 × 6

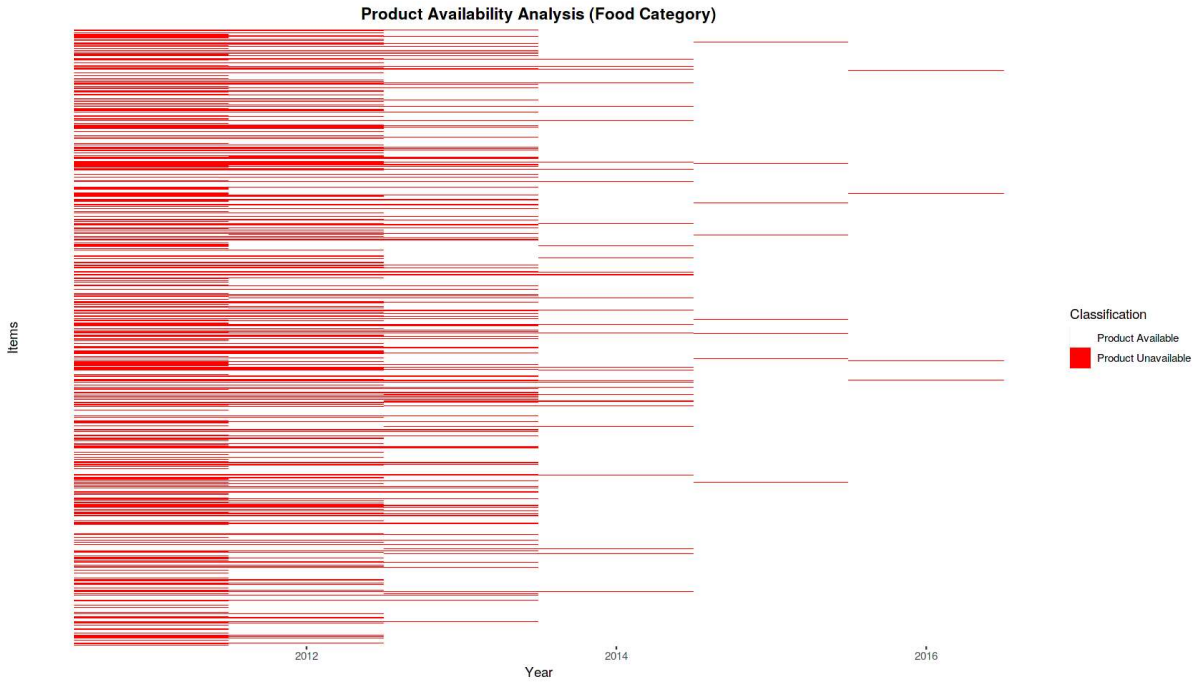
year	cat_id	Product Unavailable	Product Available	% of Unavailability	% of Availability
<int>	<fct>	<int>	<int>	<dbl>	<dbl>
2011	FOODS	6689	7681	21.9383404	25.191866
2012	FOODS	4341	10029	14.2374549	32.892752
2013	FOODS	2440	11930	8.0026238	39.127583
2014	FOODS	786	13584	2.5778944	44.552312
2015	FOODS	93	14277	0.3050180	46.825189
2016	FOODS	111	14259	0.3640538	46.766153
2011	HOBBIES	2640	3010	8.6585766	9.872089
2012	HOBBIES	1547	4103	5.0737947	13.456871
2013	HOBBIES	745	4905	2.4434241	16.087242
2014	HOBBIES	188	5462	0.6165956	17.914070
2015	HOBBIES	32	5618	0.1049524	18.425713
2016	HOBBIES	90	5560	0.2951787	18.235487
2011	HOUSEHOLD	4470	6000	14.6605444	19.678583
2012	HOUSEHOLD	2891	7579	9.4817973	24.857330
2013	HOUSEHOLD	1714	8756	5.6215153	28.717612
2014	HOUSEHOLD	326	10144	1.0692030	33.269925
2015	HOUSEHOLD	70	10400	0.2295835	34.109544
2016	HOUSEHOLD	83	10387	0.2722204	34.066907



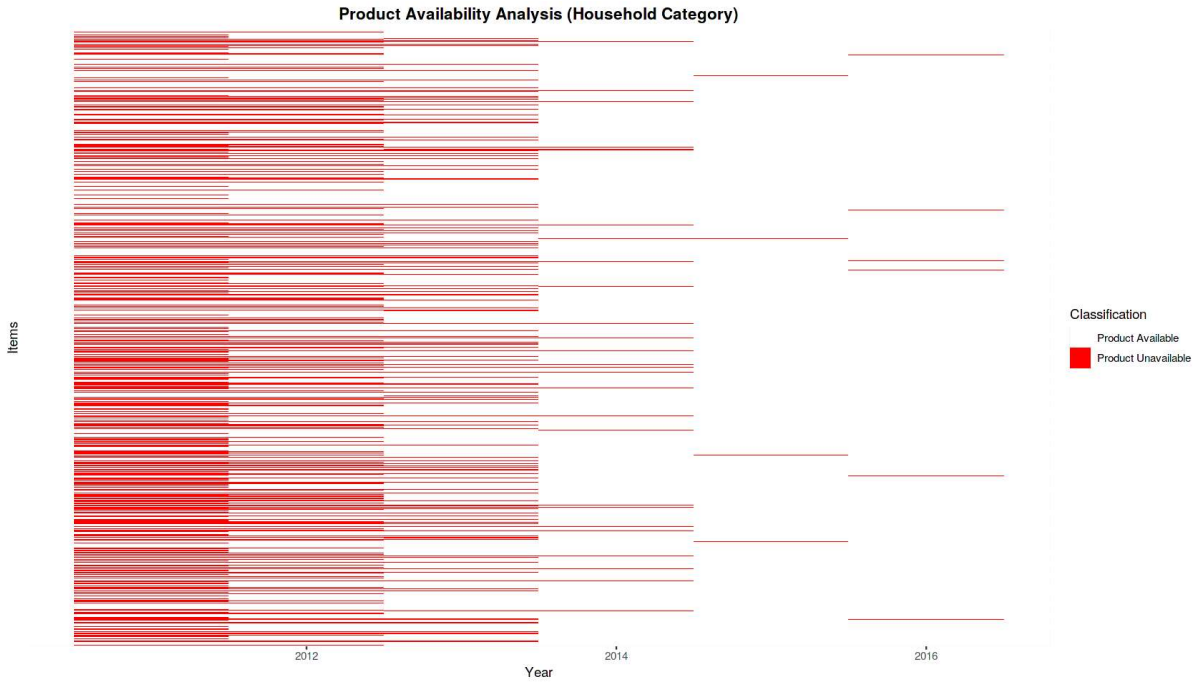
Product Availability Analysis

Heatmap is plotted to visualize the product's availability over the five year period.

In [8]:
```
## Enlarging the graph
options(repr.plot.width = 14, repr.plot.height = 8)

## Heatmap of Food Items
ggplot(data = Product_Avail_Unavail %>%
               filter(cat_id == "FOODS") %>%
               select(year, cat_id, store_id, item_id, Classification) %>%
               arrange(item_id), aes(x = year, y = item_id, fill = Classifica
tion)) +
          geom_tile() +
          scale_fill_manual(values = c("white","red")) +
          xlab(label = "Year") +
          ylab(label = "Items") +
          ggtitle("Product Availability Analysis (Food Category)") +
          theme(plot.title = element_text(hjust = 0.5, face = "bold"),
                axis.text.y = element_blank(),
                axis.ticks.y = element_blank())

## Heatmap of Hobbies Items
ggplot(data = Product_Avail_Unavail %>%
               filter(cat_id == "HOBBIES") %>%
               select(year, cat_id, store_id, item_id, Classification) %>%
               arrange(item_id), aes(x = year, y = item_id, fill = Classifica
tion)) +
          geom_tile() +
          scale_fill_manual(values = c("white","red")) +
          xlab(label = "Year") +
          ylab(label = "Items") +
          ggtitle("Product Availability Analysis (Hobbies Category)") +
          theme(plot.title = element_text(hjust = 0.5, face = "bold"),
                axis.text.y = element_blank(),
                axis.ticks.y = element_blank())

## Heatmap of Household Items
ggplot(data = Product_Avail_Unavail %>%
               filter(cat_id == "HOUSEHOLD") %>%
               select(year, cat_id, store_id, item_id, Classification) %>%
               arrange(item_id), aes(x = year, y = item_id, fill = Classifica
tion)) +
          geom_tile() +
          scale_fill_manual(values = c("white","red")) +
          xlab(label = "Year") +
          ylab(label = "Items") +
          ggtitle("Product Availability Analysis (Household Category)") +
          theme(plot.title = element_text(hjust = 0.5, face = "bold"),
                axis.text.y = element_blank(),
                axis.ticks.y = element_blank())
```


Product Availability Analysis (Food Category)
Items
2012
2014
2016
Year
Classification
Product Available
Product Unavailable


Product Availability Analysis (Hobbies Category)
Items
2012
2014
2016
Year
Classification
Product Available
Product Unavailable



In this heatmap, I have removed the item codes in the y axis in order for you to focus more on the plot itself. In the plot, the red color means that the product is unavailable in that year. If the product is available in the market, it is represented by a white color.

In the plot, these are some key observations:

1. A product with consecutive years of no unit sales since 2011 means that the product is not yet launched and unavailable in the market.
2. Once the product becomes available in the market after years of unavailability, most of the products never become unavailable for a whole year.
3. Products which were available in the earlier years but became unavailable in the later years mean that the product was hold for sale in the market.

Final Analysis and Action for Zero Unit Sales

Products available in the market that display a high number of days with zero sales might mean that the product is sold during a particular season. Take for example, fruits. Some of the fruits like strawberries, mangoes, and avocadoes have high unit sales during on-season and moderate or low sales during off-season. Another example is inflattable tubes for children. These inflattable tubes are usually sold during the summer season where children are off for swimming.

After careful analysis, the items with zero unit sales for the entire year will be removed from the training dataset except for 2016. The reasons why 2016 is an exception are:

- The data from year 2016 spans only from January 1 until April 24.
- Zero unit sales for these period cannot be concluded that the product will be unavailable for the next succeeding periods.
- Some products are subject to seasonality and they may not be currently selling during the first quarter of 2016 but may be selling in the next quarters. (http://)

Data Transformation - Part 2

In this section, the following data transformation is performed:

1. Removal of observations where the product is entirely unavailable for a year.
2. Sampling a data via stratification. The strata are: item_id and month. The advantage of using stratified sampling is that we can get data from each strata that can represent the entire population.
3. Merging the selling_prices data with the train data

In [9]:
```
## Removing of Product Unavailable

            ## Getting the relevant columns only
                Product_Avail_Unavail <- Product_Avail_Unavail %>%
                                                mutate(Classification = replace(Cl
assification, year == "2016", "Product Available")) %>%
                                                select(year, store_id, cat_id, ite
m_id, Classification)

            ## Combining with the train set
                train <- train %>%
                            left_join(Product_Avail_Unavail,
                                    by = c("year" = "year",
                                           "store_id" = "store_id",
                                           "cat_id" = "cat_id",
                                           "item_id" = "item_id"))

            ## Removing the product unavailable
                train <- train %>%
                            filter(Classification == "Product Available")

## Converting the event_type_1 into character_variable
train$event_type_1 <- as.character(train$event_type_1)

## Replacing null values of event_type_1 into "NONE"
train <- train %>%
            mutate(event_type_1 = replace(event_type_1, is.na(event_type_1),
"None"))

## Reverting class of the event_type_1 into factor
train$event_type_1 <- as.factor(train$event_type_1)
```

In [10]:
```
## Making room for RAM
rm(Product_Avail_Unavail)
gc()
```

A matrix: 2 × 6 of type dbl

	used	(Mb)	gc trigger	(Mb)	max used	(Mb)
Ncells	2418797	129.2	8057988	430.4	12708920	678.8
Vcells	565088671	4311.3	1791804960	13670.4	2052651054	15660.5

In [11]:
```
## Sampling
set.seed(100)
train <- stratified(train, c("item_id", "month", "year"), .06)
```

In [12]:
```
## Merging the selling_prices data with train data
train <- train %>%
            left_join(sell_prices,
                      by = c("store_id" = "store_id",
                             "item_id" = "item_id",
                             "wm_yr_wk" = "wm_yr_wk"))
```

In [13]:
```
## Cleaning up tables
rm(sell_prices)
rm(calendar)
```

Exploring the Data and Feature Engineering

Now that we have processed the data, let's explore what our sample data tells us.

The Exploratory Data Analysis will be conducted as follows:

1. Time Series Analysis of Unit Sales
2. Sales Performance: Year, Month, and Weekday
3. Sales Performance: State, Store Branch, Product Category, and Product.
4. Price Sensitivity Analysis
5. Effect of Events and Holidays

Time Series Analysis of Unit Sales

By visualizing the unit sales over time, we can view the movement of unit sales.

The time series graph shown below displays the movement of unit sales over time for each state.

In [14]:
```
## Enlarging the graph
options(repr.plot.width = 14, repr.plot.height = 8)

## Time Series of Overall Unit Sales
train$date <- as.Date(train$date)
ggplot(data = train %>%
                    group_by(state_id, cat_id, date) %>%
                    summarise(Total_Sales = sum(Unit_Sales))) +
       geom_line(aes(x = date, y = Total_Sales, color = cat_id)) +
       scale_colour_discrete("Product Category") +
       facet_grid(state_id~.) +
       xlab("Time") +
       ylab("Total Unit Sales") +
       ggtitle("Unit Sales Time Series Per State") +
       theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

```
`summarise()` regrouping output by 'state_id', 'cat_id' (override with `.grou
ps` argument)
```

In the time series, it shows a seasonal movement which is parallel across California, Wisconsin, and Texas.

The unit sales under the FOOD category show the highest number of unit sales, followed by household and hobbies. Obviously, the items under the Food category are essential and necessity for people. The high unit sales tell us that food items are consumed easily and demand remain high and constant. Once the bought items are consumed, people will habitually buy again. For households, these are items which have enough to last for a certain period of time. Example: deodorant, detergent, soap, and other non-edible essential materials. Finally, hobbies are the least. This tells us that the demand for hobbies at Walmart is tremendously low.

Furthermore, the unit sales presented in the time series is affected by demographic factors such as population, economy, median income, and GDP.

Here are some few demographic differences among three States:

California

1. **Area**: 163,695 square miles.
2. **College Educated**: 41%
3. **GDP**: USD 2,797.601 Billion
4. **Population**: 39,536,653
5. **Median Income**: USD 35,046
6. The most populous state in the United States.
7. Home to Hollywood's stars, Silicon Valley's technology.
8. California's shipping, agriculture, construction and transportation industries boomed as the state became a land of economic opportunity for settlers.

Wisconsin

1. **Area**: 65,496 square miles.
2. **College Educated**: 41%
3. **GDP**: USD 321.373 Billion
4. **Population**: 5,795,483
5. **Median Income**: USD 31,998
6. Dairy is a major driver of Wisconsin's economy, generating more than USD 20 billion a year. Cheese-making, cranberries, snap beans and corn for silage are major agricultural products in the state.

Texas

1. **Area**: 268,596 square miles.
2. **College Educated**: 37%
3. **GDP**: USD 1,645.136 Billion
4. **Population**: 28,304,596
5. **Median Income**: USD 29,525
6. Traditionally, agriculture has been among the state’s largest industries, and it produces the most livestock and livestock product in the country. The state also is a leader in export revenues, according to the U.S. Census Bureau. Other industries driving growth include business, education and health, hospitality and manufacturing.

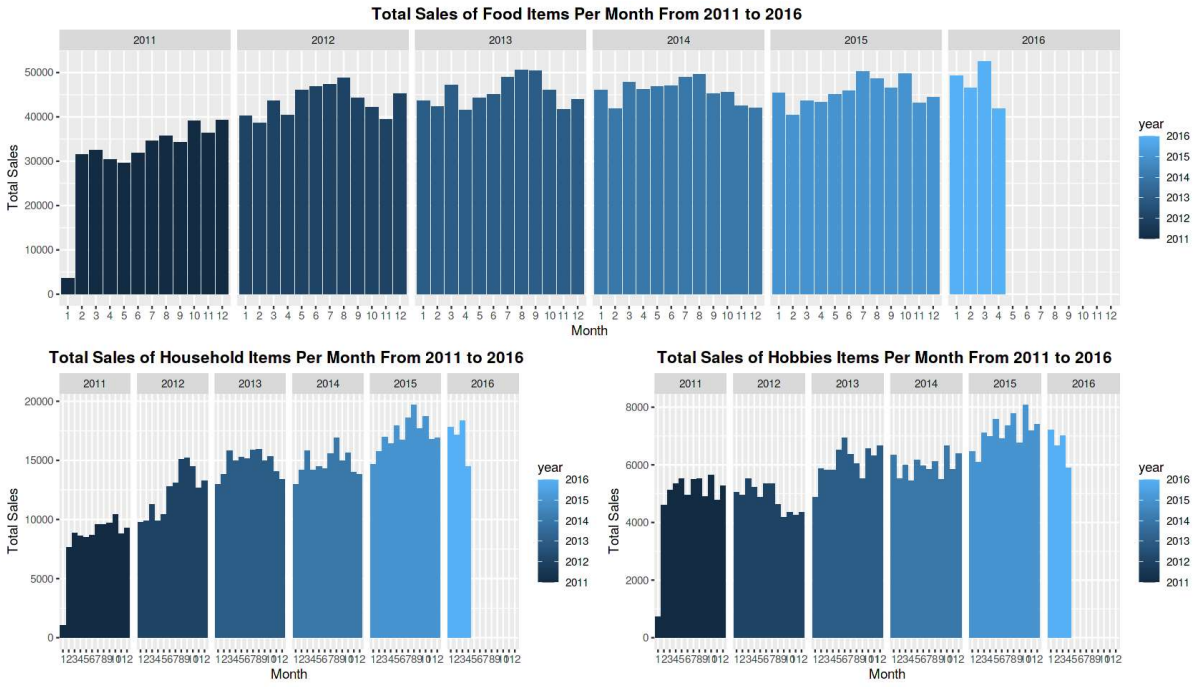
Unit Sales Analysis - Time Specific

In this section, we will delve into the unit sales more specific in time such as:

1. Which month usually has the consistent high unit sales and low unit sales for ea ch category?
2. Which day in a week usually show a high unit sales and low unit sales?
3. Which day in a month show a high unit sales and low unit sales?
4. How does events or holidays affect the unit sales?

In [15]:
```
## Enlarging the graph
options(repr.plot.width = 14, repr.plot.height = 8)

## Setting up the factor for months
levels <- as.character(c(1:12))
train$month <- factor(train$month, levels = levels)

## Plotting the unit sales by months over the years for FOOD Category
g <- ggplot(data = train %>%
                    group_by(state_id, cat_id, month, year) %>%
                    summarise(Total_Sales = sum(Unit_Sales)) %>%
                    filter(cat_id == "FOODS")) +
            geom_bar(aes(y = Total_Sales, x = month, fill = year), stat = "ide
ntity") + facet_grid(.~year) +
            xlab("Month") +
            scale_colour_discrete("Year") +
            ylab("Total Sales") +
            ggtitle("Total Sales of Food Items Per Month From 2011 to 2016 ")
+
            theme(plot.title = element_text(hjust = 0.5, face = "bold"))

## Plotting the unit sales by months over the years for HOUSEHOLD Category
h <- ggplot(data = train %>%
                    group_by(state_id, cat_id, month, year) %>%
                    summarise(Total_Sales = sum(Unit_Sales)) %>%
                    filter(cat_id == "HOUSEHOLD")) +
            geom_bar(aes(y = Total_Sales, x = month, fill = year), stat = "ide
ntity") + facet_grid(.~year) +
            xlab("Month") +
            ylab("Total Sales") +
            scale_colour_discrete("Year") +
            ggtitle("Total Sales of Household Items Per Month From 2011 to 201
6 ") +
            theme(plot.title = element_text(hjust = 0.5, face = "bold"))

## Plotting the unit sales by months over the years for HOBBIES Category
i <- ggplot(data = train %>%
                    group_by(state_id, cat_id, month, year) %>%
                    summarise(Total_Sales = sum(Unit_Sales)) %>%
                    filter(cat_id == "HOBBIES")) +
            geom_bar(aes(y = Total_Sales, x = month, fill = year), stat = "ide
ntity") + facet_grid(.~year) +
            xlab("Month") +
            ylab("Total Sales") +
            scale_colour_discrete("Year") +
            ggtitle("Total Sales of Hobbies Items Per Month From 2011 to 2016
 ") +
            theme(plot.title = element_text(hjust = 0.5, face = "bold"))

## Combining all the plots
ggarrange(g,
            ggarrange(h, i, ncol = 2),
          nrow = 2
         )
```

```
`summarise()` regrouping output by 'state_id', 'cat_id', 'month' (override wi
th `.groups` argument)

`summarise()` regrouping output by 'state_id', 'cat_id', 'month' (override wi
th `.groups` argument)

`summarise()` regrouping output by 'state_id', 'cat_id', 'month' (override wi
th `.groups` argument)
```



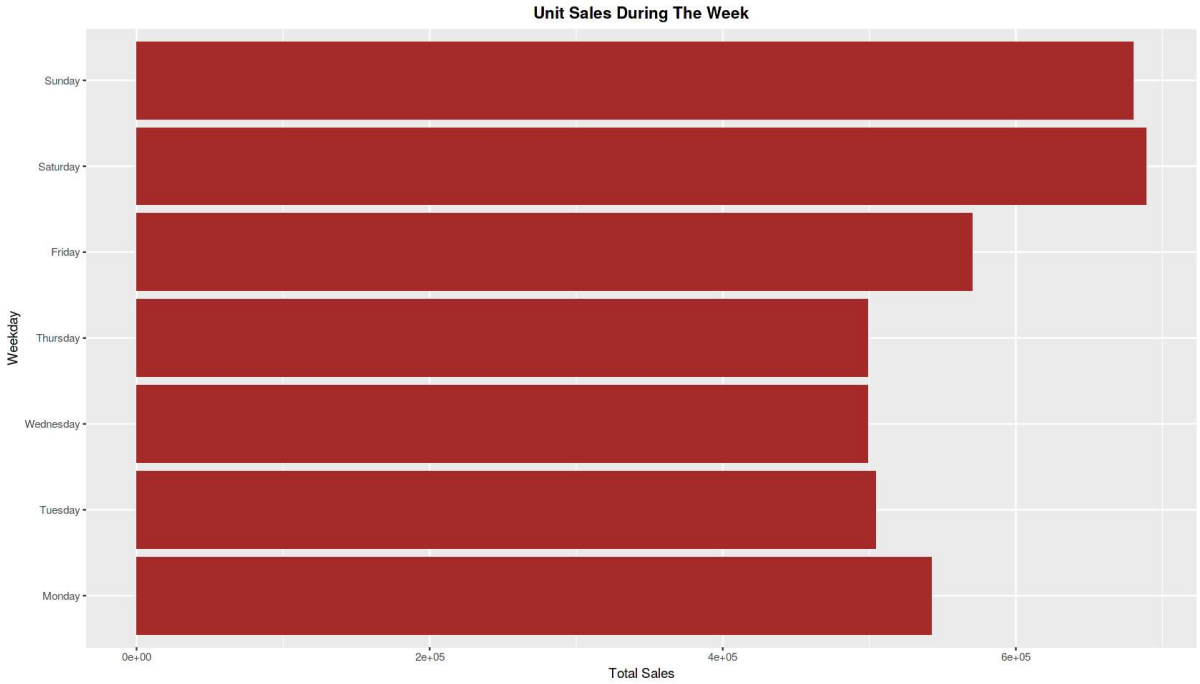
Note: The first month is really low because the start date is January 29, 2011.

In the figure above, the total sales for each month are showing an increasing trend every year. This trend is due to the introduction of new products every year at Walmart. At 2011, the food items had 21% missing products, 8% for the Hobbies, and 14% of the Household items. Gradually, this decreases year after year and seeing the increases over the month is caused by new products launched every year.

The unit sales tend to be at the highest during the fourth quarter where many holidays and events are taking place.

In [16]:
```
## Setting up the factor for weekdays
levels <- as.character(c("Monday", "Tuesday", "Wednesday", "Thursday", "Frida
y", "Saturday", "Sunday"))
train$weekday <- factor(train$weekday, levels = levels)

## Plotting the unit sales by weekdays
ggplot(data = train %>%
                    group_by(state_id, cat_id, month, weekday) %>%
                    summarise(Total_Sales = sum(Unit_Sales))) +
geom_bar(aes(x = Total_Sales, y = weekday), fill = "brown", stat = "identity")
+
xlab("Total Sales") +
ylab("Weekday") +
ggtitle("Unit Sales During The Week") +
theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

```
`summarise()` regrouping output by 'state_id', 'cat_id', 'month' (override wi
th `.groups` argument)
```



The graph shows that Saturday and Sunday are the peak days of the unit sales. This is obvious as most of the people - children and adults have weekends as their days off from school and work. Next to it is Friday and Monday which may mean that some people prefer to do groceries and shopping at these days to make their weekends free for some activities.

Event and Holiday Analysis

The events and holidays in a day involves celebration and greater spending than usual because of feast, family gatherings, and celebrations. In this section, we aim to see the magnitude of the holidays and events in unit sales.

In [17]:
```
## Selecting the relevant variables
event_holiday <- train %>%
                    select(year, month, date, event_name_1, event_type_1, even
t_name_2, event_type_2, Unit_Sales) %>%
                    mutate(day = day(date)) %>%
                    group_by(year, month, day, date, event_name_1, event_type_
1, event_name_2, event_type_2) %>%
                    summarise(Total_Sales = sum(Unit_Sales))
```

```
`summarise()` regrouping output by 'year', 'month', 'day', 'date', 'event_nam
e_1', 'event_type_1', 'event_name_2' (override with `.groups` argument)
```

In [18]:
```
## January
January <- ggplot(data = event_holiday %>%
                    filter(month == 1)) +
                  geom_line(aes(x = day, y = Total_Sales)) +
                  geom_point(aes(x = day, y = Total_Sales)) +
                  facet_grid(year~.) +
                  geom_point(data = event_holiday %>%
                                       filter(!is.na(event_name_1), month == 1
),
                              aes(x = day, y = Total_Sales, col = event_type_1),
 size = 4) +
                  geom_text(data = event_holiday %>%
                                       filter(!is.na(event_name_1), month == 1
), aes(x= day, y = Total_Sales, label = event_name_1),
                             size=3) +
                  xlab("Day") +

                  ylab("Total Sales") +
                  ggtitle("Event and Holiday Analysis for January") +
                  theme(plot.title = element_text(hjust = 0.5, face = "bold"))

## February
February <- ggplot(data = event_holiday %>%
                     filter(month == 2)) +
                   geom_line(aes(x = day, y = Total_Sales)) +
                   geom_point(aes(x = day, y = Total_Sales)) +
                   facet_grid(year~.) +
                   geom_point(data = event_holiday %>%
                                        filter(!is.na(event_name_1), month == 2
),
                               aes(x = day, y = Total_Sales, col = event_type_1),
 size = 4) +
                   geom_text(data = event_holiday %>%
                                        filter(!is.na(event_name_1), month == 2
), aes(x= day, y = Total_Sales, label = event_name_1),
                              size=3) +
                   xlab("Day") +

                   ylab("Total Sales") +
                   ggtitle("Event and Holiday Analysis for February") +
                   theme(plot.title = element_text(hjust = 0.5, face = "bold"))

## March
March <- ggplot(data = event_holiday %>%
                    filter(month == 3)) +
                  geom_line(aes(x = day, y = Total_Sales)) +
                  geom_point(aes(x = day, y = Total_Sales)) +
                  facet_grid(year~.) +
                  geom_point(data = event_holiday %>%
                                       filter(!is.na(event_name_1), month == 3
),
                              aes(x = day, y = Total_Sales, col = event_type_1),
 size = 4) +
                  geom_text(data = event_holiday %>%
                                       filter(!is.na(event_name_1), month == 3
), aes(x= day, y = Total_Sales, label = event_name_1),
```

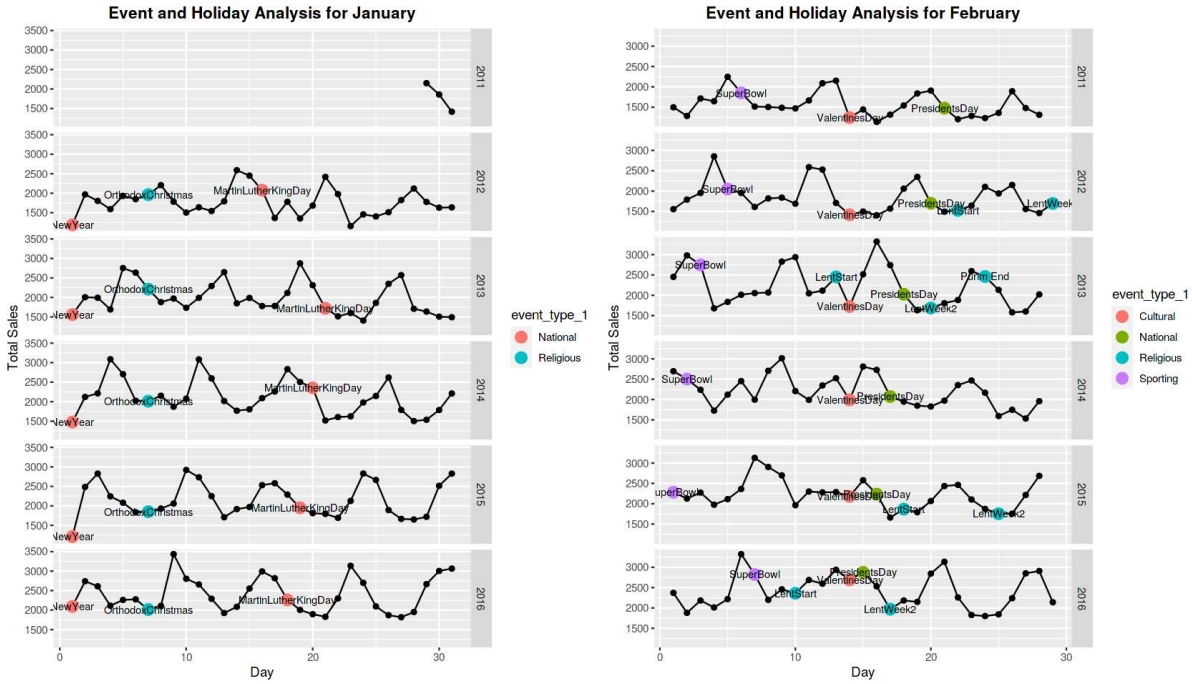
```
                          size=3) +
                xlab("Day") +

                ylab("Total Sales") +
                ggtitle("Event and Holiday Analysis for March") +
                theme(plot.title = element_text(hjust = 0.5, face = "bold"))


## April
April <- ggplot(data = event_holiday %>%
                  filter(month == 4)) +
                geom_line(aes(x = day, y = Total_Sales)) +
                geom_point(aes(x = day, y = Total_Sales)) +
                facet_grid(year~.) +
                geom_point(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 4
),
                           aes(x = day, y = Total_Sales, col = event_type_1),
 size = 4) +
                geom_text(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 4
), aes(x= day, y = Total_Sales, label = event_name_1),
                          size=3) +
                geom_point(data = event_holiday %>%
                                    filter(!is.na(event_name_2), month == 4
),
                           aes(x = day), y = 30000, col = "slategray", size =
4) +
                geom_text(data = event_holiday %>%
                                    filter(!is.na(event_name_2), month == 4
), aes(x= day, label = event_name_2), y = 30000,
                          size=3) +
                xlab("Day") +
                ylab("Total Sales") +
                ggtitle("Event and Holiday Analysis for April") +
                theme(plot.title = element_text(hjust = 0.5, face = "bold"))


## May
May <- ggplot(data = event_holiday %>%
                  filter(month == 5)) +
                geom_line(aes(x = day, y = Total_Sales)) +
                geom_point(aes(x = day, y = Total_Sales)) +
                facet_grid(year~.) +
                geom_point(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 5
),
                           aes(x = day, y = Total_Sales, col = event_type_1),
 size = 4) +
                geom_text(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 5
), aes(x= day, y = Total_Sales, label = event_name_1),
                          size=3) +
                geom_point(data = event_holiday %>%
                                    filter(!is.na(event_name_2), month == 5
),
                           aes(x = day), y = 30000, col = "slategray", size =
```

```
4) +
                geom_text(data = event_holiday %>%
                                    filter(!is.na(event_name_2), month == 5
), aes(x= day, label = event_name_2), y = 30000,
                          size=3) +
                xlab("Day") +
                ylab("Total Sales") +
                ggtitle("Event and Holiday Analysis for May") +
                theme(plot.title = element_text(hjust = 0.5, face = "bold"))


## June
June <- ggplot(data = event_holiday %>%
                  filter(month == 6)) +
                geom_line(aes(x = day, y = Total_Sales)) +
                geom_point(aes(x = day, y = Total_Sales)) +
                facet_grid(year~.) +
                geom_point(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 6
),
                           aes(x = day, y = Total_Sales, col = event_type_1),
 size = 4) +
                geom_text(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 6
), aes(x= day, y = Total_Sales, label = event_name_1),
                          size=3) +
                geom_point(data = event_holiday %>%
                                    filter(!is.na(event_name_2), month == 6
),
                           aes(x = day), y = 30000, col = "slategray", size =
4) +
                geom_text(data = event_holiday %>%
                                    filter(!is.na(event_name_2), month == 6
), aes(x= day, label = event_name_2), y = 30000,
                          size=3) +
                xlab("Day") +
                ylab("Total Sales") +
                ggtitle("Event and Holiday Analysis for June") +
                theme(plot.title = element_text(hjust = 0.5, face = "bold"))

## July
July <- ggplot(data = event_holiday %>%
                  filter(month == 7)) +
                geom_line(aes(x = day, y = Total_Sales)) +
                geom_point(aes(x = day, y = Total_Sales)) +
                facet_grid(year~.) +
                geom_point(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 7
),
                           aes(x = day, y = Total_Sales, col = event_type_1),
 size = 4) +
                geom_text(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 7
), aes(x= day, y = Total_Sales, label = event_name_1),
                          size=3) +
                xlab("Day") +
```

```
                ylab("Total Sales") +
                ggtitle("Event and Holiday Analysis for July") +
                theme(plot.title = element_text(hjust = 0.5, face = "bold"))

## August
August <- ggplot(data = event_holiday %>%
                   filter(month == 8)) +
                geom_line(aes(x = day, y = Total_Sales)) +
                geom_point(aes(x = day, y = Total_Sales)) +
                facet_grid(year~.) +
                geom_point(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 8
),
                           aes(x = day, y = Total_Sales, col = event_type_1),
 size = 4) +
                geom_text(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 8
), aes(x= day, y = Total_Sales, label = event_name_1),
                          size=3) +
                xlab("Day") +

                ylab("Total Sales") +
                ggtitle("Event and Holiday Analysis for August") +
                theme(plot.title = element_text(hjust = 0.5, face = "bold"))

## September
September <- ggplot(data = event_holiday %>%
                   filter(month == 9)) +
                geom_line(aes(x = day, y = Total_Sales)) +
                geom_point(aes(x = day, y = Total_Sales)) +
                facet_grid(year~.) +
                geom_point(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 9
),
                           aes(x = day, y = Total_Sales, col = event_type_1),
 size = 4) +
                geom_text(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 9
), aes(x= day, y = Total_Sales, label = event_name_1),
                          size=3) +
                xlab("Day") +

                ylab("Total Sales") +
                ggtitle("Event and Holiday Analysis for September") +
                theme(plot.title = element_text(hjust = 0.5, face = "bold"))


## October
October <- ggplot(data = event_holiday %>%
                   filter(month == 10)) +
                geom_line(aes(x = day, y = Total_Sales)) +
                geom_point(aes(x = day, y = Total_Sales)) +
                facet_grid(year~.) +
                geom_point(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 10
),
                           aes(x = day, y = Total_Sales, col = event_type_1),
```

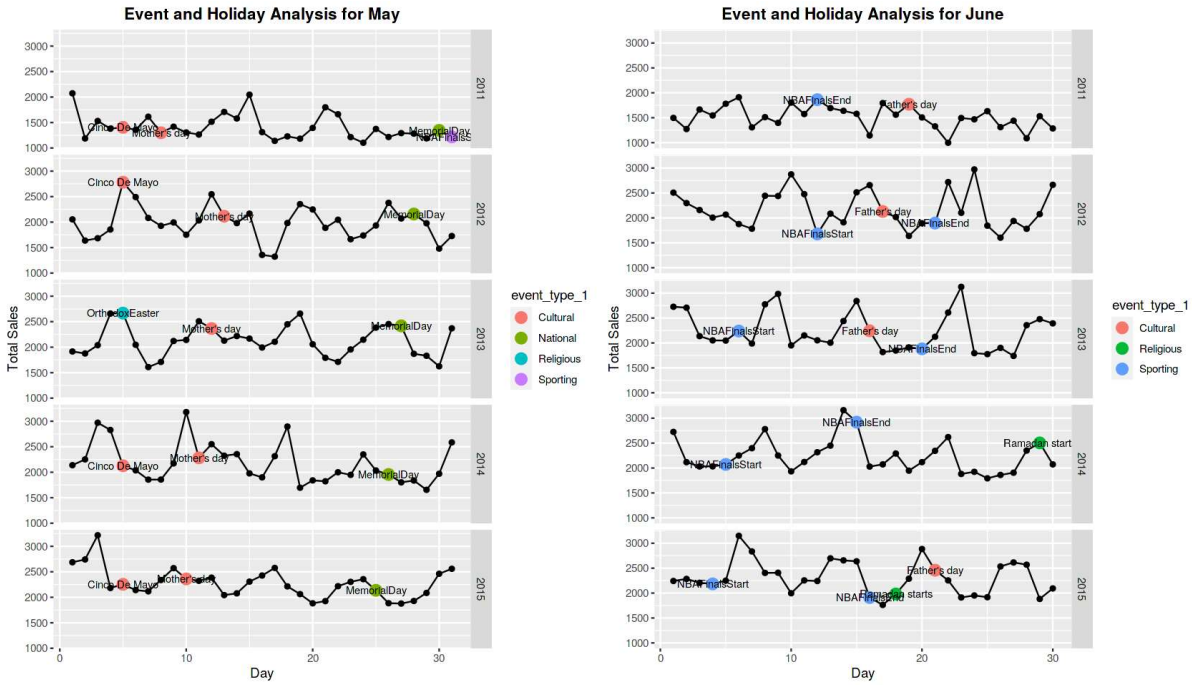
```
 size = 4) +
                geom_text(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 10
), aes(x= day, y = Total_Sales, label = event_name_1),
                          size=3) +
                xlab("Day") +

                ylab("Total Sales") +
                ggtitle("Event and Holiday Analysis for October") +
                theme(plot.title = element_text(hjust = 0.5, face = "bold"))

## November
November <- ggplot(data = event_holiday %>%
                  filter(month == 11)) +
                geom_line(aes(x = day, y = Total_Sales)) +
                geom_point(aes(x = day, y = Total_Sales)) +
                facet_grid(year~.) +
                geom_point(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 11
),
                           aes(x = day, y = Total_Sales, col = event_type_1),
 size = 4) +
                geom_text(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 11
), aes(x= day, y = Total_Sales, label = event_name_1),
                          size=3) +
                xlab("Day") +

                ylab("Total Sales") +
                ggtitle("Event and Holiday Analysis for November") +
                theme(plot.title = element_text(hjust = 0.5, face = "bold"))


## December
December <- ggplot(data = event_holiday %>%
                  filter(month == 12)) +
                geom_line(aes(x = day, y = Total_Sales)) +
                geom_point(aes(x = day, y = Total_Sales)) +
                facet_grid(year~.) +
                geom_point(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 12
),
                           aes(x = day, y = Total_Sales, col = event_type_1),
 size = 4) +
                geom_text(data = event_holiday %>%
                                    filter(!is.na(event_name_1), month == 12
), aes(x= day, y = Total_Sales, label = event_name_1),
                          size=3) +
                xlab("Day") +

                ylab("Total Sales") +
                ggtitle("Event and Holiday Analysis for December") +
                theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

In [19]:
```
## Enlarging the graph
options(repr.plot.width = 14, repr.plot.height = 8)

## Combining January and February Plots
ggarrange(January, February, ncol = 2)
```

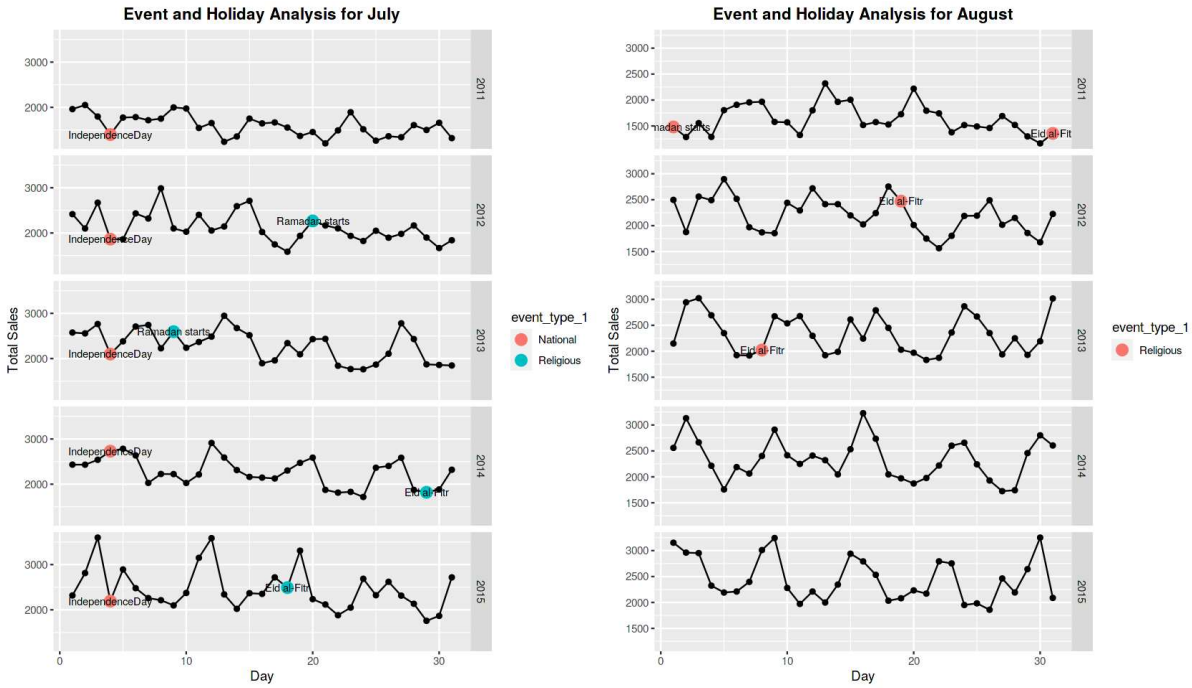


In [20]:
```
## Enlarging the graph
options(repr.plot.width = 14, repr.plot.height = 8)

## Combining March and April Plots
ggarrange(March, April, ncol = 2)
```

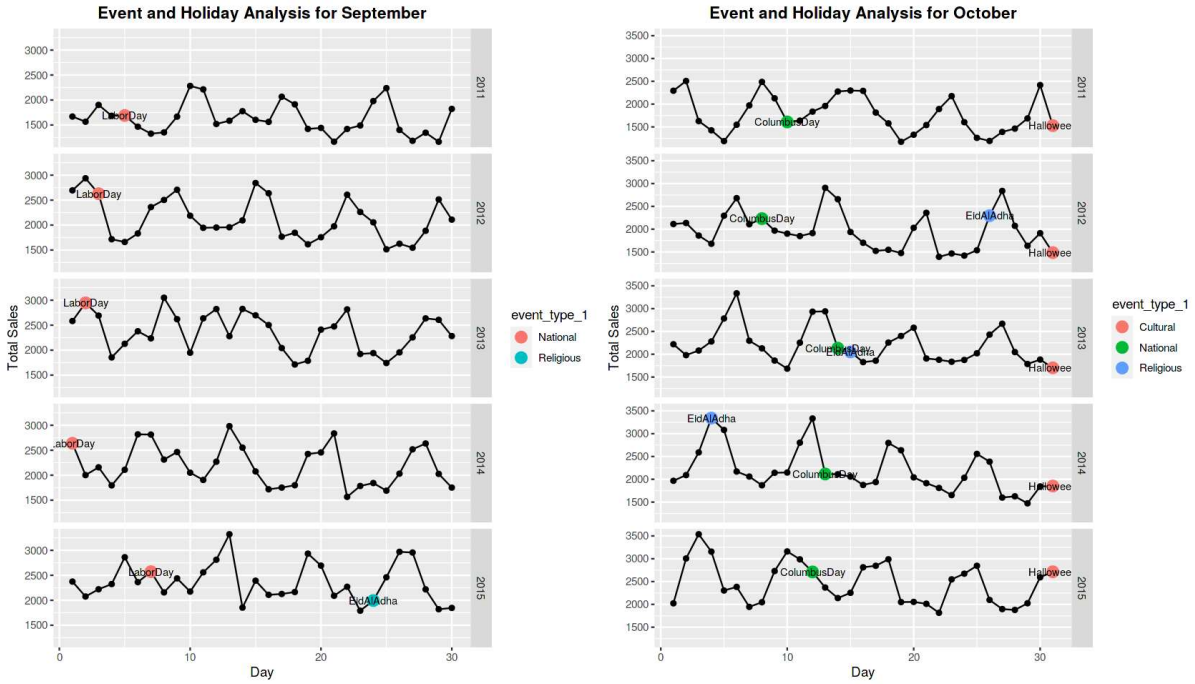
In [21]:
```
## Enlarging the graph
options(repr.plot.width = 14, repr.plot.height = 8)

## Combining May and June Plots
ggarrange(May, June, ncol = 2)
```



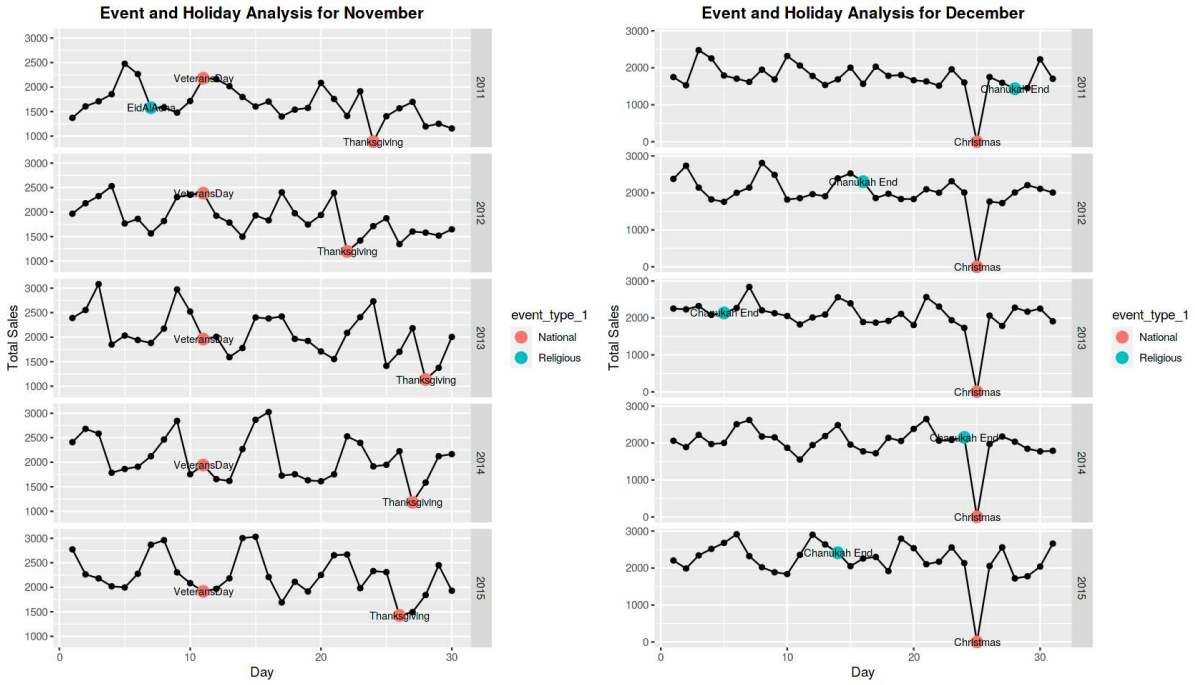
In [22]:
```
## Enlarging the graph
options(repr.plot.width = 14, repr.plot.height = 8)

## Combining July and August Plots
ggarrange(July, August, ncol = 2)
```



In [23]:
```
## Enlarging the graph
options(repr.plot.width = 14, repr.plot.height = 8)

## Combining September and October Plots
ggarrange(September, October, ncol = 2)
```



In [24]:
```
## Enlarging the graph
options(repr.plot.width = 14, repr.plot.height = 8)

## Combining November and December Plots
ggarrange(November, December, ncol = 2)
```



In [25]:
```
## Clearing the ggplot variables to make room for RAM
rm(January, February, March, April, May, June, July, August, September, Octobe
r, November, December)
gc()
```

A matrix: 2 × 6 of type dbl

	used	(Mb)	gc trigger	(Mb)	max used	(Mb)
Ncells	2517293	134.5	7740141	413.4	12708920	678.8
Vcells	55270999	421.7	917404140	6999.3	2052651054	15660.5

To summarise this section,

1. Every year starts at a very low sale due to reduced store hour during New Year's Eve and Americans spend the day at their homes after a long holiday season.
2. Sporting events such as Superbowl and NBA finals show an interesting insight. The day before the event, the unit sales are very high then suddenly drop on these events. Sport is part of the American culture and a lot of people go to stadiums to watch the events on live or stay at their homes to watch it via pay-per-view.
3. Religious events show little to no sharp increase in unit sales. Based on 2017 data, the religion of the majority is Protestantism (48.5%), followed by Catholicism (22.7%). 21.3% do not have religion. The remaining are minority. Muslim religious events in America do not any effect on unit sales because Muslims in America are minority.
4. National holidays tend to have a similar effect with Sporting events. The unit sales spike before the day and on the day itself, the unit sales dropped. Obviously, the people celebrates these days at their homes or restaurants.
5. Likewise, Cultural holidays such as Mother's Day and Father's Day behave National holidays. People spend their time on these days celebrating and feasting so unit sales in groceries are not that high.

Overall, the graph shows a seasonal trend because most of the Americans tend to have groceries during weekends.

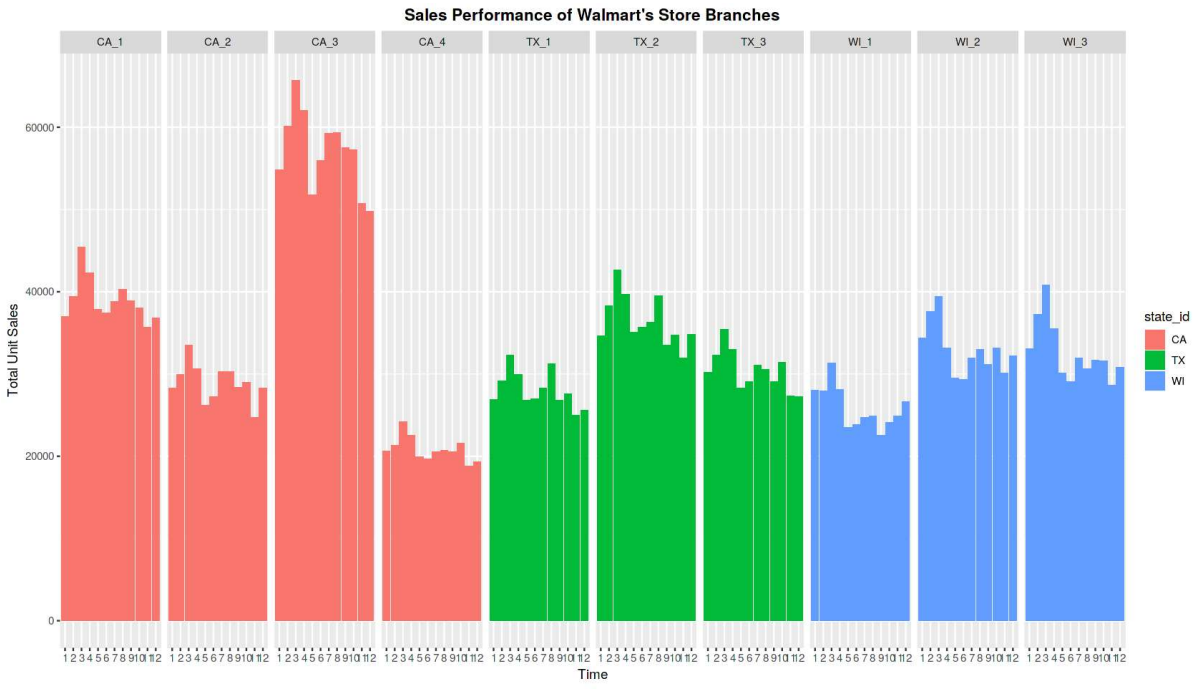
Branch Sales Performance

In this section, we are going to analyze the performance of the store branches. We aim to answer the following:

1. Which of the following branches are the most profitable? least profitable?
2. How are the store branches performing?

In [26]:
```
## Enlarging the graph
options(repr.plot.width = 14, repr.plot.height = 8)

## Plotting the Sales Performance
ggplot(data = train %>%
                    group_by(state_id, store_id, month, year) %>%
                    summarise(Total_Sales = sum(Unit_Sales))) +
                geom_bar(aes(x = month, y = Total_Sales, fill = state_id), sta
t = "identity") +
                facet_grid(.~store_id) +
                xlab("Time") +
                ylab("Total Unit Sales") +
                scale_colour_discrete("State") +
                ggtitle("Sales Performance of Walmart's Store Branches") +
                theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

```
`summarise()` regrouping output by 'state_id', 'store_id', 'month' (override
with `.groups` argument)
```



This graph displays clearly the sales performances of each branches.

1. Store branch, CA_3 is the highest performing branch in terms of unit sales.
2. Store branch, CA_4 is the least performing branch. This may be inferred that this is a newly established branch where it is located in a developing area.
3. The rest of the branches are performing nearly as well as the other branches.

Department Sales Performance

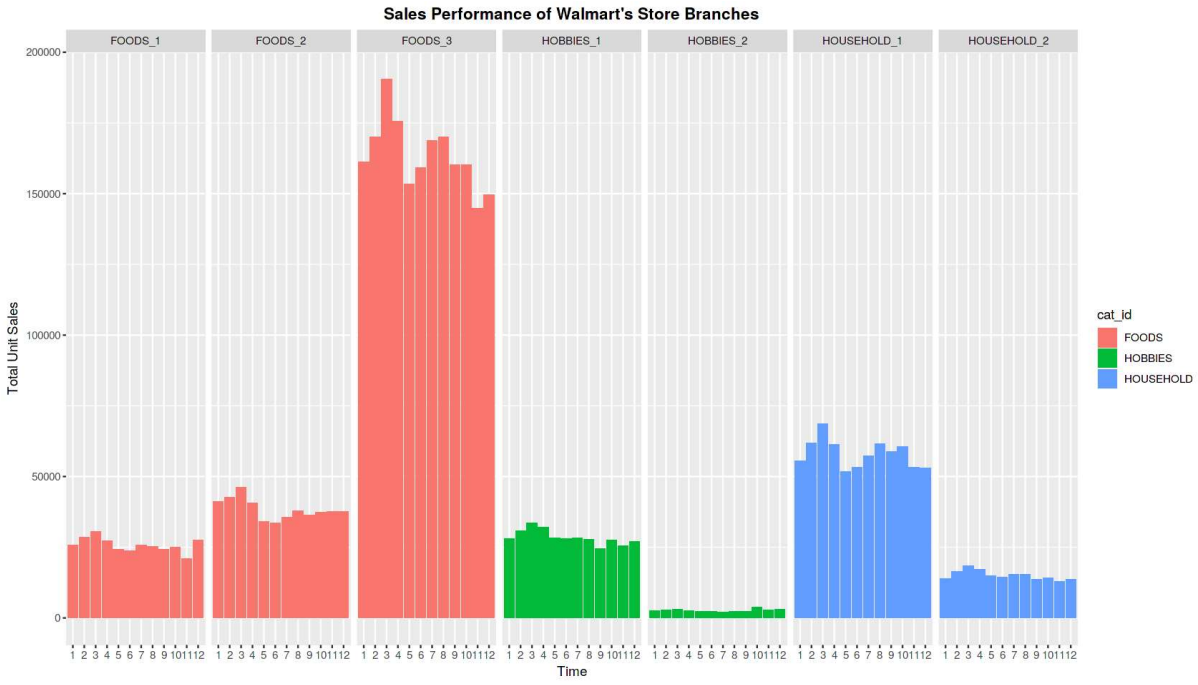
In this section, we will examine the sales performance of each product department. Product department is the food section you usually see in the groceries. For example, Food Department 1 is the Milk and Dairy Products; Food Department 2 is the Poultry, Beef, and Pork Products.

We aim to answer the following:

1. Which of the following departments are the best performing for each product category? least performing?
2. How is the performance of each product departments?

In [27]:
```
## Enlarging the graph
options(repr.plot.width = 14, repr.plot.height = 8)

## Plotting the Department Performance
ggplot(data = train %>%
                    group_by(cat_id, dept_id, month, year) %>%
                    summarise(Total_Sales = sum(Unit_Sales))) +
                geom_bar(aes(x = month, y = Total_Sales, fill = cat_id), stat
= "identity") +
                facet_grid(.~dept_id) +
                xlab("Time") +
                ylab("Total Unit Sales") +
                scale_colour_discrete("State") +
                ggtitle("Sales Performance of Walmart's Store Branches") +
                theme(plot.title = element_text(hjust = 0.5, face = "bold"))
```

```
`summarise()` regrouping output by 'cat_id', 'dept_id', 'month' (override wit
h `.groups` argument)
```



In this graph, the performance of each product department are vastly different.

- Food_3 is the highest performing. It may mean that this product department has all the essential food items.
- Hobbies_2 is the least performing.
- Surprisingly, Food_1 and Food_2 are comparatively very low.
- Househhold_1 is second highest performing. This may indicate that this product department holds the everyday essential items such as detergent, soaps, cleaning equipment, and daily household items.

Obviously, if the product is an essential and for daily/habitual use, then the unit sales of that product is expected to be very high.

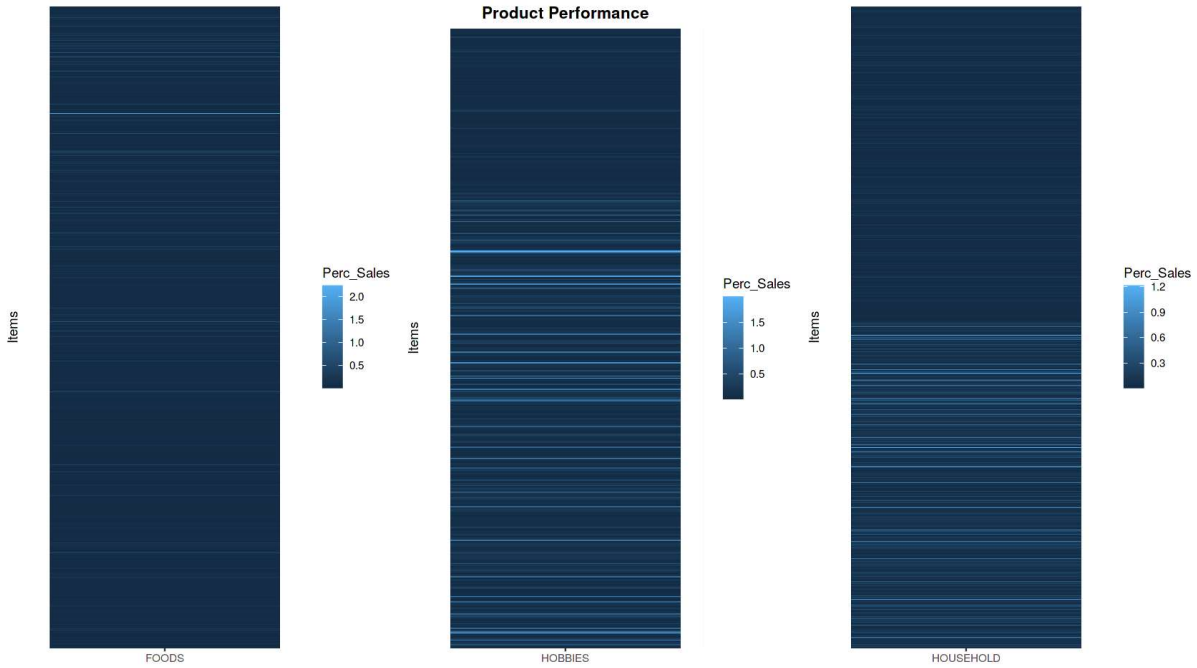
Item Sales Performance

In this section, we are going to analyze how are the items performing. If the item is selling consistently, then we expect that the item is marketed well by its brand and marketers, trusted by the consumers, and will get sold again.

In [28]:
```
## Calculating the performance of each item by product category
FOODS_item_performance <- train %>%
                                  select(cat_id, item_id, sell_price, Unit_Sales
) %>%
                                  filter(cat_id == "FOODS") %>%
                                  mutate(revenue = sell_price*Unit_Sales) %>%
                                  group_by(cat_id, item_id) %>%
                                  summarise(Total_Sales = sum(Unit_Sales),
                                            Total_Revenue = sum(revenue)) %>%
                                  mutate(Perc_Sales = Total_Sales/sum(Total_Sale
s)*100,
                                         Perc_Revenue = Total_Revenue/sum(Total_
Revenue, na.rm = TRUE)*100)

HOBBIES_item_performance <- train %>%
                                  select(cat_id, item_id, sell_price, Unit_Sales
) %>%
                                  filter(cat_id == "HOBBIES") %>%
                                  mutate(revenue = sell_price*Unit_Sales) %>%
                                  group_by(cat_id, item_id) %>%
                                  summarise(Total_Sales = sum(Unit_Sales),
                                            Total_Revenue = sum(revenue)) %>%
                                  mutate(Perc_Sales = Total_Sales/sum(Total_Sale
s)*100,
                                         Perc_Revenue = Total_Revenue/sum(Total_
Revenue, na.rm = TRUE)*100)

HOUSEHOLD_item_performance <- train %>%
                                    select(cat_id, item_id, sell_price, Unit_S
ales) %>%
                                    filter(cat_id == "HOUSEHOLD") %>%
                                    mutate(revenue = sell_price*Unit_Sales) %
>%
                                    group_by(cat_id, item_id) %>%
                                    summarise(Total_Sales = sum(Unit_Sales),
                                              Total_Revenue = sum(revenue)) %
>%
                                    mutate(Perc_Sales = Total_Sales/sum(Total_
Sales)*100,
                                           Perc_Revenue = Total_Revenue/sum(To
tal_Revenue, na.rm = TRUE)*100)

## Combining all item performance tables
item_performance <- rbind(FOODS_item_performance,
                          HOBBIES_item_performance,
                          HOUSEHOLD_item_performance)
```

```
`summarise()` regrouping output by 'cat_id' (override with `.groups` argumen
t)

`summarise()` regrouping output by 'cat_id' (override with `.groups` argumen
t)

`summarise()` regrouping output by 'cat_id' (override with `.groups` argumen
t)
```

In [29]:
```
## Product Performance Heatmap
g <- ggplot(data = FOODS_item_performance, aes(x = cat_id, y = item_id, fill =
Perc_Sales)) +
                  geom_tile() +
                  xlab(label = "") +
                  ylab(label = "Items") +
                  theme(plot.title = element_text(hjust = 0.5, face = "bold"),
                        axis.text.y = element_blank(),
                        axis.ticks.y = element_blank())

h <- ggplot(data = HOBBIES_item_performance, aes(x = cat_id, y = item_id, fill
= Perc_Sales)) +
                  geom_tile() +
                  xlab(label = "") +
                  ylab(label = "Items") +
                  ggtitle("Product Performance") +
                  theme(plot.title = element_text(hjust = 0.5, face = "bold"),
                        axis.text.y = element_blank(),
                        axis.ticks.y = element_blank())

i <- ggplot(data = HOUSEHOLD_item_performance, aes(x = cat_id, y = item_id, fi
ll = Perc_Sales)) +
                  geom_tile() +
                  xlab(label = "") +
                  ylab(label = "Items") +
                  theme(plot.title = element_text(hjust = 0.5, face = "bold"),
                        axis.text.y = element_blank(),
                        axis.ticks.y = element_blank())

## Combining all plots
ggarrange(g, h, i, ncol = 3, nrow = 1)
```



In [30]:
```
## Putting the item performance in the table
item_performance <- item_performance %>%
                        select(cat_id, item_id, Perc_Sales, Perc_Revenue)

train <- train %>%
           left_join(item_performance, by = c("cat_id" = "cat_id",
                                              "item_id" = "item_id"))
```

In [31]:
```
## Making room for RAM
rm(item_performance)
gc()
```

A matrix: 2 × 6 of type dbl

	used	(Mb)	gc trigger	(Mb)	max used	(Mb)
Ncells	2512335	134.2	7740141	413.4	12708920	678.8
Vcells	61151882	466.6	733923312	5599.4	2052651054	15660.5

Our heatmap shows how much is the unit sales of each item compared to overall items. There are 1437 items under the FOOD category, 565 HOBBIES category, and 1047 HOUSEHOLD category so the distribution of percentage is relatively very low due to the high volume of products.

The performance of the products based on percentage of sales shows us:

1. Stiff competition.
2. Higher unit sales of the product can translate an established reputation to the market.

Time-Varying Effect

I am going to include time lags and rolling average for our variables. These lags would consist of the following: one day before, one week before, and one month before. These time variables may add some value in predicting the unit sales.

In [32]:
```
## Calculating the time-varying effect
train <- train %>%
            group_by(cat_id, item_id, date) %>%
            mutate(lag_day = lag(Unit_Sales, k = 1),
                   lag_week = lag(Unit_Sales, k = 7),
                   lag_month = lag(Unit_Sales, k = 30))
```

Summary of EDA

After exploring the entire data, we have found a lot of interesting insights which I would like to summarize:

1. A lot of products were not introduced at the start of the year. Gradually, it became available in the market.
2. Our timeseries show a seasonal and increasing trend due to new introduction of products in the market.
3. Demographic differences have an effect on unit sales: California displays a higher sales compared to Wisconsin and Texas.
4. Looking at the monthly sales, the fourth quarter of the year shows a relatively higher unit sales compared to other months of the year.
5. Looking at the day sales, weekends (Saturday and Sunday) consistently have the highest unit sales.
6. Sporting events and national and cultural holidays show a drop in unit sales because Americans spend these days to watch sports and celebrate holidays. The unit sales tend to have an increase before holidays.
7. The unit sales is also affected by the store branches. The sales performance of store branches will depend on its location, product availability, size, and other demographic features.
8. The product category of the items has a significant effect with the unit sales. Based on our findings, all items under the FOOD_3 have the highest unit sales combined compared to other categories.
9. The low percentage of unit sales of the products shows that there is a stiff competition. If the product has a high unit sales, then the product is competitive and established in the market.
10. The time-varifying effect variables (lag variables - days, weeks, and months) have been added.

Modelling

The EDA has helped us understood the insights and information of what the data has. With this insights, we can now proceed to craft our prediction model. This model will help us achieve our objective, *to advance the theory and practice of forecasting by identifying the methods that provide the most accurate point forecasts using the retail data of Walmart.*

Extreme Gradient Boosting

The model that I am going to use is Extreme Gradient Boosting. It is a machine learning model for regression and classification that uses an ensemble of decision trees to make prediction and continually improves the model based on the calculated errors.

The steps in performing these modelling are as follows:

1. **Data Partitioning** - The dataset will be divided as follows: 70% as training set and 30% as validation set.
2. **One-hot Encoding** - Convert the categorical variables into a numeric form.
3. **Converting the data into matrix form**
4. **Setting up the parameters**
5. **Modelling and Performance Assessment**

Data Partitioning

In this section, I have separately gathered the id and date which will be used later on for the submission. I converted the year variable into factor class and gathered the features to be used for the model.

In [33]:
```
## Getting the item
id <- train %>%
            ungroup() %>%
            select(id, date)

## Selecting only the relevant features
train <- train %>%
            ungroup() %>%
            select(Unit_Sales, sell_price, lag_day, lag_week, lag_month,event_
type_1, cat_id, dept_id, store_id, state_id,
                   Perc_Sales, Perc_Revenue, month, year, weekday)

## Converting year into a factor variables
train$year <- as.factor(train$year)
```

After gathering the features, I separated the dependent variable, Unit Sales from the independent variables. Going forward, I divided the dataset into two - 70% will be the training data and 30 % will be the test data.

In [34]:
```
## Separating the Unit Sales from the predictors
data <- select(train, -Unit_Sales)
labels <- select(train, Unit_Sales)

## Dividing the number of dataset into training and test sets
        ## training data
        train_data <- data[1:(round(nrow(train)*0.70)), ]
        train_labels <- labels[1:(round(nrow(train)*0.70)), ]

        ## test data
        test_data <- data[(round(nrow(train)*0.70)+1):nrow(train), ]
        test_labels <- labels[(round(nrow(train)*0.70)+1):nrow(train), ]
```

Since xgboost model accepts data in numeric, we need to convert the categorical variables into numeric (i.e. One hot encoding).

In [35]:
```
## One-hot encoding for train data
trainMatrix <- model.matrix(~ event_type_1 + cat_id + dept_id + store_id + sta
te_id + month + year + weekday - 1,
                                    data = train_data)

## One-hot encoding for validation data
testMatrix <- model.matrix(~ event_type_1 + cat_id + dept_id + store_id + stat
e_id + month + year + weekday - 1,
                                    data = test_data)
```

After one-hot encoding, we will combine all of the variables and transform the data frame into dmatrix. DMatrix is needed for the xgboost model.

In [36]:
```
## Storing numeric variables into a separate variable
trainNum <- train_data %>%
                select(sell_price, lag_day, lag_week, lag_month, Perc_Sales, P
erc_Revenue)

testNum <- test_data %>%
                select(sell_price, lag_day, lag_week, lag_month, Perc_Sales, P
erc_Revenue)


## Combining the numeric variables with One-Hot Encoded variables
trainMatrix <- cbind(trainNum, trainMatrix)
testMatrix <- cbind(testNum, testMatrix)
```

In [37]:
```
## Converting the cleaned data frame into a dmatrix - train set
trainDMatrix <- xgb.DMatrix(data = data.matrix(trainMatrix), label = data.matr
ix(train_labels))

## Converting the cleaned data frame into a dmatrix - test set
testDMatrix <- xgb.DMatrix(data = data.matrix(testMatrix), label = data.matrix
(test_labels))
```

Setting up the model parameters

In [38]:
```
#Set parameters of model
parameters <- list(booster = "gbtree",
                   objective = "reg:linear",
                   gamma = 3, ## regularization (or prevents overfitting).
                   eta = 0.3, ## the learning rate, i.e., the rate at which ou
r model learns patterns in data
                   max_depth = 5, ## the depth of the tree.
                   sub_sample = 0.7, ## the number of samples (observations) s
upplied to a tree.
                   nfold = 4
                   )

#Cross-validation
xgb.tab <- xgb.cv(data = trainDMatrix,
                  param = parameters,
                  nrounds = 200, ## the maximum number of iterations required
 for gradient descent to converge.
                  eval_metric = "rmse",
                  early_stopping_rounds = 10, ## the number of iterations the
 model will continue to try to improve.
                  nfold = 4
                  )
```

```
[11:54:47] WARNING: amalgamation/../src/objective/regression_obj.cu:170: reg:
linear is now deprecated in favor of reg:squarederror.
[11:54:52] WARNING: amalgamation/../src/objective/regression_obj.cu:170: reg:
linear is now deprecated in favor of reg:squarederror.
[11:54:57] WARNING: amalgamation/../src/objective/regression_obj.cu:170: reg:
linear is now deprecated in favor of reg:squarederror.
[11:55:02] WARNING: amalgamation/../src/objective/regression_obj.cu:170: reg:
linear is now deprecated in favor of reg:squarederror.
[1]     train-rmse:4.051144+0.014117    test-rmse:4.059685+0.048691
Multiple eval metrics are present. Will use test_rmse for early stopping.
Will train until test_rmse hasn't improved in 10 rounds.

[2]     train-rmse:3.758027+0.013800    test-rmse:3.767863+0.044194
[3]     train-rmse:3.581309+0.009786    test-rmse:3.593288+0.046827
[4]     train-rmse:3.469111+0.008072    test-rmse:3.490946+0.039076
[5]     train-rmse:3.403122+0.008189    test-rmse:3.429323+0.038322
[6]     train-rmse:3.358451+0.007917    test-rmse:3.387141+0.039542
[7]     train-rmse:3.321560+0.010203    test-rmse:3.358755+0.044823
[8]     train-rmse:3.293316+0.007388    test-rmse:3.328421+0.047710
[9]     train-rmse:3.269469+0.011058    test-rmse:3.304861+0.044578
[10]    train-rmse:3.254023+0.011692    test-rmse:3.292096+0.043014
[11]    train-rmse:3.241389+0.015951    test-rmse:3.282869+0.041749
[12]    train-rmse:3.222561+0.014563    test-rmse:3.270858+0.046249
[13]    train-rmse:3.211216+0.014059    test-rmse:3.262198+0.048316
[14]    train-rmse:3.199192+0.014921    test-rmse:3.255212+0.049357
[15]    train-rmse:3.189669+0.014508    test-rmse:3.247622+0.047726
[16]    train-rmse:3.180971+0.016509    test-rmse:3.241750+0.046463
[17]    train-rmse:3.172572+0.020917    test-rmse:3.235802+0.042675
[18]    train-rmse:3.164269+0.019892    test-rmse:3.229044+0.043869
[19]    train-rmse:3.151388+0.016018    test-rmse:3.221845+0.045748
[20]    train-rmse:3.143241+0.017473    test-rmse:3.213082+0.045865
[21]    train-rmse:3.136091+0.014915    test-rmse:3.209521+0.046689
[22]    train-rmse:3.129148+0.016765    test-rmse:3.205577+0.047323
[23]    train-rmse:3.124345+0.017123    test-rmse:3.202209+0.047155
[24]    train-rmse:3.120041+0.017812    test-rmse:3.199352+0.048191
[25]    train-rmse:3.113627+0.018343    test-rmse:3.194600+0.048233
[26]    train-rmse:3.107338+0.020458    test-rmse:3.189020+0.045846
[27]    train-rmse:3.103074+0.019994    test-rmse:3.184629+0.046488
[28]    train-rmse:3.096229+0.015726    test-rmse:3.180463+0.050991
[29]    train-rmse:3.085944+0.017122    test-rmse:3.171736+0.053311
[30]    train-rmse:3.082741+0.016833    test-rmse:3.169391+0.054107
[31]    train-rmse:3.078122+0.015764    test-rmse:3.166629+0.055351
[32]    train-rmse:3.075041+0.015780    test-rmse:3.164557+0.054917
[33]    train-rmse:3.072197+0.015207    test-rmse:3.162838+0.055561
[34]    train-rmse:3.068891+0.015972    test-rmse:3.159772+0.054553
[35]    train-rmse:3.065751+0.016633    test-rmse:3.157660+0.054598
[36]    train-rmse:3.062961+0.016697    test-rmse:3.155814+0.054598
[37]    train-rmse:3.057288+0.015161    test-rmse:3.151084+0.054487
[38]    train-rmse:3.054035+0.013866    test-rmse:3.149072+0.054585
[39]    train-rmse:3.049956+0.012591    test-rmse:3.145211+0.055500
[40]    train-rmse:3.046913+0.015295    test-rmse:3.143238+0.053711
[41]    train-rmse:3.044036+0.016605    test-rmse:3.142187+0.052871
[42]    train-rmse:3.041994+0.016343    test-rmse:3.141412+0.053596
[43]    train-rmse:3.038323+0.015997    test-rmse:3.138764+0.054226
[44]    train-rmse:3.034162+0.017646    test-rmse:3.135629+0.053683
[45]    train-rmse:3.031791+0.018687    test-rmse:3.133754+0.053531
[46]    train-rmse:3.028418+0.018135    test-rmse:3.131009+0.052578
```

```
[47]    train-rmse:3.024382+0.018647    test-rmse:3.127879+0.052239
[48]    train-rmse:3.020869+0.017496    test-rmse:3.125628+0.053335
[49]    train-rmse:3.019020+0.017030    test-rmse:3.123219+0.054655
[50]    train-rmse:3.016831+0.017360    test-rmse:3.121173+0.055078
[51]    train-rmse:3.013803+0.017574    test-rmse:3.118673+0.055484
[52]    train-rmse:3.011441+0.017667    test-rmse:3.116523+0.055279
[53]    train-rmse:3.009670+0.019306    test-rmse:3.115330+0.053389
[54]    train-rmse:3.003658+0.019494    test-rmse:3.111340+0.052401
[55]    train-rmse:3.001439+0.018104    test-rmse:3.109805+0.053457
[56]    train-rmse:2.997539+0.019206    test-rmse:3.107053+0.053539
[57]    train-rmse:2.994468+0.018466    test-rmse:3.105258+0.054260
[58]    train-rmse:2.991862+0.019027    test-rmse:3.102944+0.054203
[59]    train-rmse:2.989645+0.020236    test-rmse:3.101488+0.052821
[60]    train-rmse:2.987141+0.019665    test-rmse:3.099731+0.053199
[61]    train-rmse:2.984309+0.018918    test-rmse:3.098783+0.054485
[62]    train-rmse:2.982427+0.018864    test-rmse:3.097710+0.054111
[63]    train-rmse:2.977904+0.018752    test-rmse:3.097386+0.054149
[64]    train-rmse:2.977004+0.018789    test-rmse:3.097011+0.054272
[65]    train-rmse:2.974722+0.018229    test-rmse:3.095220+0.054687
[66]    train-rmse:2.973205+0.018328    test-rmse:3.094493+0.054474
[67]    train-rmse:2.970634+0.019018    test-rmse:3.093118+0.053829
[68]    train-rmse:2.965917+0.018118    test-rmse:3.089652+0.055166
[69]    train-rmse:2.963682+0.017311    test-rmse:3.088532+0.055821
[70]    train-rmse:2.960884+0.018403    test-rmse:3.085892+0.055760
[71]    train-rmse:2.958690+0.017771    test-rmse:3.083844+0.055760
[72]    train-rmse:2.956394+0.016864    test-rmse:3.082474+0.056895
[73]    train-rmse:2.955310+0.016285    test-rmse:3.081930+0.057406
[74]    train-rmse:2.952694+0.015814    test-rmse:3.079916+0.057675
[75]    train-rmse:2.948823+0.017257    test-rmse:3.077141+0.057228
[76]    train-rmse:2.947169+0.017577    test-rmse:3.075897+0.056753
[77]    train-rmse:2.944696+0.017155    test-rmse:3.074608+0.057057
[78]    train-rmse:2.943869+0.017666    test-rmse:3.074051+0.056684
[79]    train-rmse:2.942004+0.017690    test-rmse:3.072677+0.056908
[80]    train-rmse:2.939278+0.018077    test-rmse:3.071331+0.056822
[81]    train-rmse:2.937849+0.017594    test-rmse:3.070892+0.056810
[82]    train-rmse:2.935011+0.017838    test-rmse:3.068423+0.056845
[83]    train-rmse:2.931080+0.016503    test-rmse:3.067304+0.057344
[84]    train-rmse:2.929378+0.016799    test-rmse:3.066365+0.057596
[85]    train-rmse:2.927978+0.016577    test-rmse:3.065639+0.057592
[86]    train-rmse:2.927105+0.017025    test-rmse:3.065503+0.057762
[87]    train-rmse:2.926043+0.017213    test-rmse:3.065355+0.057406
[88]    train-rmse:2.924980+0.017616    test-rmse:3.064503+0.057089
[89]    train-rmse:2.923148+0.016775    test-rmse:3.063539+0.057801
[90]    train-rmse:2.920540+0.018353    test-rmse:3.062075+0.058106
[91]    train-rmse:2.917880+0.017911    test-rmse:3.060494+0.059123
[92]    train-rmse:2.916963+0.017869    test-rmse:3.060056+0.059344
[93]    train-rmse:2.915437+0.018158    test-rmse:3.059329+0.059779
[94]    train-rmse:2.911580+0.017407    test-rmse:3.055979+0.061590
[95]    train-rmse:2.910356+0.017799    test-rmse:3.055167+0.061448
[96]    train-rmse:2.908520+0.017750    test-rmse:3.053755+0.061266
[97]    train-rmse:2.906537+0.018767    test-rmse:3.052374+0.061166
[98]    train-rmse:2.905084+0.018113    test-rmse:3.051457+0.062088
[99]    train-rmse:2.904393+0.018387    test-rmse:3.051040+0.061792
[100]   train-rmse:2.902025+0.018843    test-rmse:3.049611+0.061696
[101]   train-rmse:2.900523+0.019132    test-rmse:3.049113+0.062040
[102]   train-rmse:2.898697+0.017845    test-rmse:3.048606+0.062763
[103]   train-rmse:2.897574+0.017154    test-rmse:3.047592+0.062886
```

```
[104]   train-rmse:2.895540+0.017862    test-rmse:3.046060+0.061516
[105]   train-rmse:2.895075+0.017648    test-rmse:3.045649+0.061741
[106]   train-rmse:2.893890+0.017475    test-rmse:3.045278+0.061767
[107]   train-rmse:2.892900+0.017178    test-rmse:3.044575+0.062080
[108]   train-rmse:2.891994+0.016890    test-rmse:3.044233+0.062462
[109]   train-rmse:2.890086+0.016490    test-rmse:3.043169+0.062744
[110]   train-rmse:2.888343+0.016908    test-rmse:3.041997+0.062028
[111]   train-rmse:2.887034+0.017467    test-rmse:3.041137+0.061447
[112]   train-rmse:2.886226+0.018273    test-rmse:3.041021+0.061146
[113]   train-rmse:2.885413+0.018294    test-rmse:3.040930+0.061325
[114]   train-rmse:2.882980+0.018679    test-rmse:3.039325+0.061062
[115]   train-rmse:2.881185+0.019812    test-rmse:3.037985+0.060630
[116]   train-rmse:2.880019+0.020845    test-rmse:3.037363+0.060383
[117]   train-rmse:2.878867+0.020643    test-rmse:3.036965+0.060407
[118]   train-rmse:2.877550+0.021592    test-rmse:3.036612+0.059677
[119]   train-rmse:2.876395+0.021195    test-rmse:3.035725+0.060622
[120]   train-rmse:2.875556+0.021386    test-rmse:3.034966+0.060417
[121]   train-rmse:2.874831+0.021323    test-rmse:3.034587+0.060430
[122]   train-rmse:2.873552+0.021292    test-rmse:3.034305+0.060612
[123]   train-rmse:2.872429+0.020899    test-rmse:3.033674+0.060541
[124]   train-rmse:2.870914+0.020782    test-rmse:3.032803+0.060164
[125]   train-rmse:2.869933+0.021156    test-rmse:3.032129+0.060043
[126]   train-rmse:2.868403+0.021743    test-rmse:3.031080+0.059544
[127]   train-rmse:2.866804+0.021966    test-rmse:3.029901+0.059743
[128]   train-rmse:2.864930+0.022171    test-rmse:3.028873+0.059698
[129]   train-rmse:2.863841+0.021813    test-rmse:3.028193+0.059742
[130]   train-rmse:2.861879+0.021931    test-rmse:3.027637+0.060285
[131]   train-rmse:2.860533+0.021795    test-rmse:3.026715+0.060316
[132]   train-rmse:2.859107+0.021310    test-rmse:3.026032+0.060523
[133]   train-rmse:2.857778+0.021539    test-rmse:3.025011+0.060101
[134]   train-rmse:2.857108+0.021307    test-rmse:3.024749+0.060306
[135]   train-rmse:2.856717+0.021262    test-rmse:3.024437+0.060348
[136]   train-rmse:2.854715+0.021439    test-rmse:3.023130+0.060601
[137]   train-rmse:2.853616+0.021039    test-rmse:3.022449+0.060285
[138]   train-rmse:2.853198+0.020906    test-rmse:3.022349+0.060350
[139]   train-rmse:2.851294+0.020162    test-rmse:3.021467+0.061532
[140]   train-rmse:2.850423+0.019720    test-rmse:3.020875+0.061971
[141]   train-rmse:2.849541+0.019653    test-rmse:3.020520+0.062087
[142]   train-rmse:2.848181+0.019640    test-rmse:3.019725+0.062325
[143]   train-rmse:2.846388+0.020787    test-rmse:3.018610+0.061627
[144]   train-rmse:2.845418+0.020508    test-rmse:3.017790+0.062207
[145]   train-rmse:2.844753+0.020531    test-rmse:3.017323+0.062195
[146]   train-rmse:2.842906+0.020525    test-rmse:3.016834+0.061948
[147]   train-rmse:2.842166+0.020278    test-rmse:3.016397+0.062267
[148]   train-rmse:2.841552+0.020071    test-rmse:3.016087+0.062451
[149]   train-rmse:2.840067+0.019689    test-rmse:3.015404+0.062296
[150]   train-rmse:2.838532+0.018776    test-rmse:3.014682+0.062825
[151]   train-rmse:2.837851+0.018863    test-rmse:3.014145+0.062682
[152]   train-rmse:2.836502+0.018418    test-rmse:3.013607+0.062822
[153]   train-rmse:2.835605+0.018969    test-rmse:3.013184+0.062607
[154]   train-rmse:2.834677+0.019128    test-rmse:3.012900+0.063074
[155]   train-rmse:2.833435+0.019384    test-rmse:3.012010+0.063041
[156]   train-rmse:2.832270+0.020010    test-rmse:3.011378+0.062318
[157]   train-rmse:2.831003+0.019900    test-rmse:3.010468+0.062508
[158]   train-rmse:2.830240+0.020073    test-rmse:3.009947+0.062355
[159]   train-rmse:2.829392+0.020084    test-rmse:3.009421+0.062383
[160]   train-rmse:2.827729+0.020974    test-rmse:3.008479+0.062427
```

```
[161]   train-rmse:2.826368+0.021165    test-rmse:3.007882+0.062899
[162]   train-rmse:2.825213+0.020839    test-rmse:3.007304+0.063008
[163]   train-rmse:2.824385+0.020723    test-rmse:3.005552+0.062978
[164]   train-rmse:2.824107+0.020698    test-rmse:3.005641+0.062846
[165]   train-rmse:2.822990+0.022164    test-rmse:3.005743+0.063164
[166]   train-rmse:2.822361+0.022107    test-rmse:3.005204+0.063459
[167]   train-rmse:2.821815+0.021918    test-rmse:3.004822+0.063677
[168]   train-rmse:2.821171+0.021819    test-rmse:3.004529+0.063731
[169]   train-rmse:2.820579+0.021753    test-rmse:3.004085+0.063788
[170]   train-rmse:2.819774+0.021617    test-rmse:3.003554+0.064213
[171]   train-rmse:2.818804+0.021748    test-rmse:3.003036+0.064146
[172]   train-rmse:2.818125+0.021505    test-rmse:3.002731+0.064236
[173]   train-rmse:2.817212+0.021647    test-rmse:3.002581+0.064403
[174]   train-rmse:2.816201+0.021381    test-rmse:3.002738+0.064587
[175]   train-rmse:2.815556+0.021871    test-rmse:3.002486+0.064687
[176]   train-rmse:2.814834+0.022814    test-rmse:3.002826+0.064865
[177]   train-rmse:2.814277+0.022668    test-rmse:3.002388+0.065265
[178]   train-rmse:2.813785+0.022887    test-rmse:3.002139+0.064952
[179]   train-rmse:2.812743+0.023040    test-rmse:3.001669+0.064772
[180]   train-rmse:2.811850+0.022912    test-rmse:2.999504+0.064629
[181]   train-rmse:2.811056+0.023436    test-rmse:2.999473+0.064623
[182]   train-rmse:2.810195+0.023619    test-rmse:2.998951+0.064675
[183]   train-rmse:2.809406+0.023865    test-rmse:2.998496+0.064585
[184]   train-rmse:2.808338+0.023711    test-rmse:2.998031+0.064733
[185]   train-rmse:2.807571+0.023761    test-rmse:2.997532+0.064788
[186]   train-rmse:2.807041+0.023578    test-rmse:2.997326+0.064859
[187]   train-rmse:2.806482+0.023274    test-rmse:2.997020+0.065016
[188]   train-rmse:2.804106+0.024079    test-rmse:2.994993+0.064693
[189]   train-rmse:2.803535+0.024072    test-rmse:2.994713+0.064871
[190]   train-rmse:2.803111+0.023998    test-rmse:2.994755+0.064711
[191]   train-rmse:2.801788+0.024042    test-rmse:2.994103+0.064762
[192]   train-rmse:2.801135+0.024460    test-rmse:2.993821+0.064560
[193]   train-rmse:2.800660+0.024387    test-rmse:2.993332+0.064973
[194]   train-rmse:2.799654+0.024556    test-rmse:2.992957+0.064436
[195]   train-rmse:2.798862+0.024582    test-rmse:2.993128+0.064396
[196]   train-rmse:2.798281+0.024844    test-rmse:2.993244+0.064431
[197]   train-rmse:2.797732+0.024633    test-rmse:2.992964+0.064525
[198]   train-rmse:2.797090+0.024400    test-rmse:2.992696+0.064752
[199]   train-rmse:2.795856+0.024395    test-rmse:2.990725+0.064842
[200]   train-rmse:2.794998+0.023963    test-rmse:2.990040+0.065538
```

In [95]:
```
## Getting the best iteration
best_iteration <- xgb.tab$best_iteration
```

In [40]:
```
## Creating the model
model <- xgb.train(data = trainDMatrix,
                   param = parameters,
                   maximize = FALSE,
                   nrounds = best_iteration
                   )
```

```
[12:25:33] WARNING: amalgamation/../src/objective/regression_obj.cu:170: reg:
linear is now deprecated in favor of reg:squarederror.
[12:25:33] WARNING: amalgamation/../src/learner.cc:480:
Parameters: { nfold, sub_sample } might not be used.

  This may not be accurate due to some parameters are only used in language b
indings but
  passed down to XGBoost core.  Or some parameters are not used but slip thro
ugh this
  verification. Please open an issue if you find above cases.
```

In [41]:
```
## Getting the id of the test data
id <- id[(round(nrow(train)*0.70)+1):nrow(train), ]
```

In [42]:
```
## Reducing the RAM usage
rm(train)
gc()
```

A matrix: 2 × 6 of type dbl

	used	(Mb)	gc trigger	(Mb)	max used	(Mb)
Ncells	4572381	244.2	15276517	815.9	23869557	1274.8
Vcells	213838552	1631.5	435769374	3324.7	2052651054	15660.5

Results

Now that the model is finished, we will evaluate the result of the predictions by calculating the RMSE, R-Squared, and SHAP.

Prediction

In [73]:
```
## Predicting the Unit Sales based on the test data
predict <- data.frame(Unit_Sales = predict(model, testDMatrix))

## Combining the predictions and the id
predictions <- cbind(id, predict)

## Creating the date id
calendar <- predictions %>%
                select(date) %>%
                arrange(desc(date)) %>%
                distinct(date) %>%
                as.data.frame() %>%
                mutate(d = paste0("F", seq(from = 1, to = nrow(.), by = 1)))
```

In [58]:
```
## Transforming the data based on the format prescribed for submission
predictions <- predictions %>%
                    left_join(calendar, by = c("date" = "date")) %>%
                    select(-date) %>%
                    group_by(id) %>%
                    spread(key = "d", value = "Unit_Sales") %>%
                    select(id, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F
12, F13, F14, F15, F16, F17, F18, F19, F20, F21,
                                F22, F23, F24, F25, F26, F27, F28)

## Exporting the predictions
write_xlsx(predictions, "predictions.xlsx")
```

In [75]:
```
## Creating a new vector for the analysis

    ## Renaming the column
    test_labels <- rename(test_labels, Actual_Sales = Unit_Sales)

    ## Combining the predictions and test tables
    predict <- cbind(predictions,test_labels)

    ## Calculating the residual errors
    predict <- predict %>%
                    mutate(errors = abs(Actual_Sales - Unit_Sales))
```

A data.frame: 6 × 5

	id	date	Unit_Sales	Actual_Sales	errors
	<fct>	<date>	<dbl>	<int>	<dbl>
1	HOBBIES_1_381_CA_2_validation	2015-01-10	2.8120561	3	0.1879439
2	HOBBIES_1_381_WI_2_validation	2015-01-16	1.5616575	0	1.5616575
3	HOBBIES_1_382_CA_3_validation	2015-01-08	0.8236625	0	0.8236625
4	HOBBIES_1_382_WI_3_validation	2015-01-06	0.5150372	0	0.5150372
5	HOBBIES_1_382_CA_1_validation	2015-01-15	0.7788112	0	0.7788112
6	HOBBIES_1_382_WI_2_validation	2015-01-02	0.6182321	0	0.6182321

Root Mean Square Error (RMSE)

RMSE is the standard deviation of the residuals (i.e. the difference between the actual values and the predicted values). It is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

In [82]:
```
## RMSE
RMSE <- round(sqrt(mean(predict$errors^2)),4)
cat("The root mean square error of the test data is", RMSE)
```

```
The root mean square error of the test data is 2.7448
```

Previously, during cross validation, our best iteration provides an test RMSE of approx. 2.99. In our calculation, the RMSE is 2.7448 which is lower than the CV. Since RMSE decreases based on our predictions, it means that our model has generalized well our training data and that the predictions prove to be favorable.

R-Squared

R-squared is a metric that measures the percentage of variations of dependent variable explained by the variations of independent variables. Higher r-squared means the variations of dependent variables are explained well by the independent variables.

In [84]:
```
## Calculating the R-Squared manually
    ## Mean of actual sales of test data
    y_mean <- mean(predict$Actual_Sales)

    ## Total Sum of Squares
    TSS <- sum((predict$Actual_Sales - y_mean)^2)

    ## Residual Sum of Squares
    RSS <- sum(predict$errors^2)

    ## Calculate R-squared
    RSQ  =  round((1 - (RSS/TSS)), 4)

cat('The R-square of the test data is ', RSQ)
```

```
The R-square of the test data is  0.4153
```

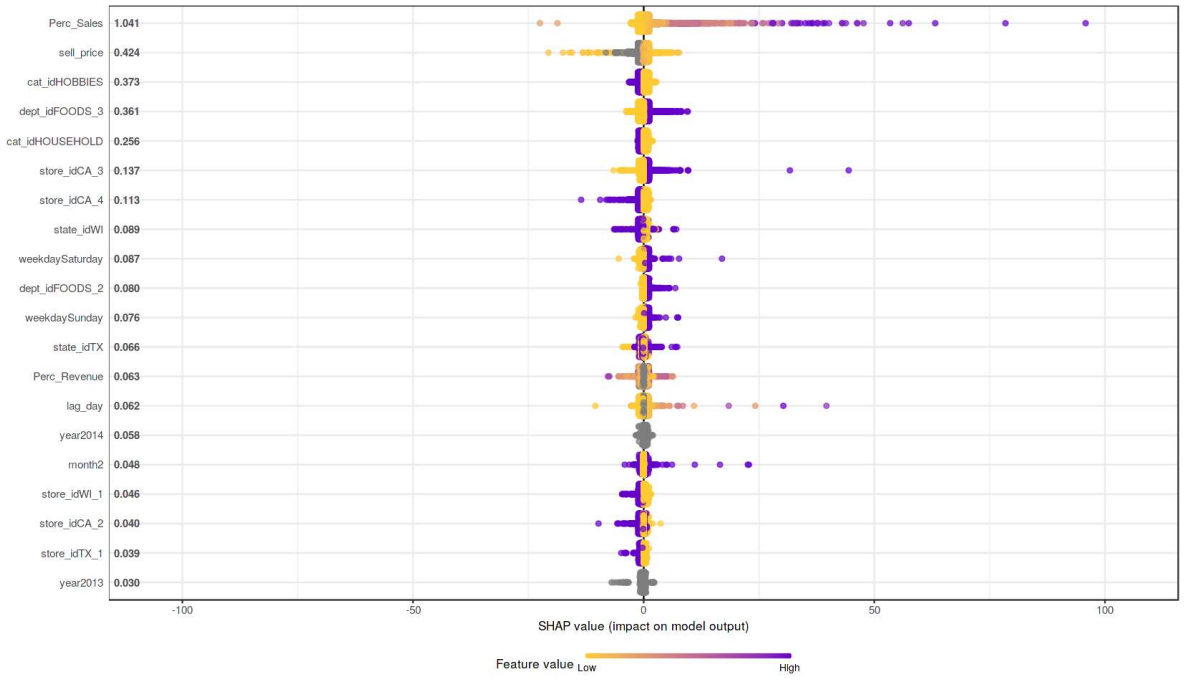
Our r-squared is only 41.53%. This value is low. It may indicate that our variables are not able to explain fully the Unit Sales. Thus, there might be other variables that can better explain the Unit Sales.

SHAP (SHapley Additive exPlanations)

The SHAP measures the impact of explanatory variables (i.e. independent variables) taking into account the interaction with the other variables. It calculates the importance of a feature by comparing what a model predicts with and without the feature.

Feature importance also measures the impact of explanatory variables, but it suffers from impact when interaction of variables take place. For example, Percentage of Revenue may have a higher feature importance when the sell_price variable is excluded. If sell_price variable were to be included, then the feature importance of Percentage of Revenue goes down.

In [43]:
```
## Calculating and Plotting the SHAP
shap.plot.summary.wrap1(model, X = data.matrix(trainMatrix[1:50000,]), top_n =
20)
```

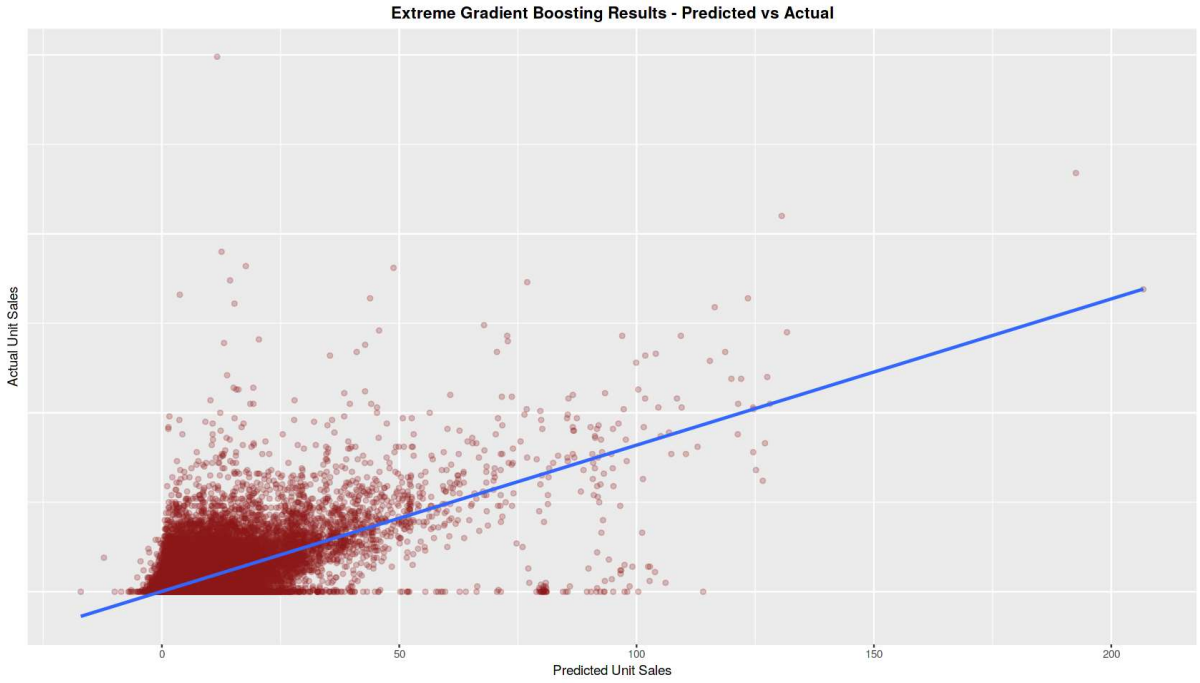


Based on the graph, we can see that the Percentage of Sales variable has the highest SHAP value, but the importance is relatively low. Selling price is surprisingly not important based on the result.

On the other hand, stores, states, departments, and weekdays variables have a higher importance despite of their low SHAP value. Collectively speaking, these variables are based on demographics and time. Hence, the predictive power for this case may perform well on demographics and time-specific variables.

In [93]:
```
## Plotting the actual_sales and predicted sales

ggplot(data = predict, aes(x = Unit_Sales, y = Actual_Sales)) +
    geom_point(color = "firebrick4", alpha = 1/4) +
    geom_smooth(method = "lm") +
    xlab(label = "Predicted Unit Sales") +
    ylab(label = "Actual Unit Sales") +
    ggtitle("Extreme Gradient Boosting Results - Predicted vs Actual") +
    theme(plot.title = element_text(hjust = 0.5, face = "bold"),
          axis.text.y = element_blank(),
          axis.ticks.y = element_blank())
```

```
`geom_smooth()` using formula 'y ~ x'
```



A bar plot is provided to visualize the interaction of actual unit sales and predicted unit sales. The blue line shows the regression trend. As you can see, the points are widely spread (i.e. the predictions were not able to approximate the unit sales). This graph alone is sufficient enough to inform us that the model must be reviewed and improved in order to reduce the variance and to better approximate the target variable.

Notes

Thank you for reading my attempt to develop a forecasting model. I hope you learn something from this notebook. Based on my self-assessment, I needed to improve more on feature engineering and incorporate other models such as intermittent demand and ARIMA that may improve the RMSE even further.

I am very much happy to take criticisms and points for improvements. Thanks!