Walmart_Sales_Forecasting

Christina Gao

2/7/2022

Walmart Sales Forecasting

1. Data Preparation

1.1 Load Packages

```
# Load packages

# Data Preparation
library(dplyr)  #data manipulation
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(readr)    # read rectangular data like csv, tsv
library(skimr) # provide broad overview of dataframe
library(xts)   # handling time-based data
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
##
## Attaching package: 'xts'
```

```
## The following objects are masked from 'package:dplyr':
##
##     first, last
```

```
library(lubridate) # easier to work with dates & time
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(tidyr)  # tidy messy data


# Data Visualization
library(ggplot2)
library(plotly)  # produce interactive plots
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(ggcorrplot) # create corrlation matrix using ggplot2
library(ggpubr)  # easier to use functions working with ggplot2
library(viridis) # access to viridis palette
```

```
## Loading required package: viridisLite
```

```
library(gridExtra) # arrange multiple grid-based plots on a page
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(ggthemes) # create additional themes
library(corrplot) # create correlation plot
```

```
## corrplot 0.89 loaded
```

```
library(naniar) # display #s of missing values visually
```

```
##
## Attaching package: 'naniar'
```

```
## The following object is masked from 'package:skimr':
##
##     n_complete
```

```
# Data Wrangling
library(varhandle)  # to unfactor factor
library(Amelia)  # visualizing missing values
```

```
## Loading required package: Rcpp
```

```
## ##
## ## Amelia II: Multiple Imputation
## ## (Version 1.8.0, built: 2021-05-26)
## ## Copyright (C) 2005-2022 James Honaker, Gary King and Matthew Blackwell
## ## Refer to http://gking.harvard.edu/amelia/ for more information
## ##
```

```
library(VIM) # display missing proportion plot
```

```
## Loading required package: colorspace
```

```
## Loading required package: grid
```

```
## VIM is ready to use.
```

```
## Suggestions and bug-reports can be submitted at: https://github.com/statistikat/VIM/issues
```

```
##
## Attaching package: 'VIM'
```

```
## The following object is masked from 'package:datasets':
##
##     sleep
```

```
# Time Series
library(imputeTS)  # use if for imputating missing values for ts object
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
##
## Attaching package: 'imputeTS'
```

```
## The following object is masked from 'package:zoo':
##
##     na.locf
```

```
library(tictoc) # timing function
library(MetricsWeighted) # performance measures used in machine learning
library(forecast)
```

```
##
## Attaching package: 'forecast'
```

```
## The following object is masked from 'package:MetricsWeighted':
##
##     accuracy
```

```
## The following object is masked from 'package:ggpubr':
##
##     gghistogram
```

Summary

- loaded necessary packages for data preparation, visualization, features selection, wrangling, and building machine learning models

1.2 Clear Global Environment & Set Seed

```
# Clear the global env variables
rm(list=ls())

# Set seed so code can be reproduced
set.seed(2021)
```

1.3 Load Files

```
# Read the train dataset and create desired data types
train <- read_csv("C:/Users/chris/OneDrive/Documents/Projects/Project_5_Forecasting/walmart_sale
s/train_dataset.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Store = col_double(),
##   Dept = col_double(),
##   Date = col_date(format = ""),
##   Weekly_Sales = col_double(),
##   IsHoliday = col_logical()
## )
```

```
# View the first 10 observations
head(train, n = 10)
```

```
## # A tibble: 10 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
##  7     1     1 2010-03-19       22137. FALSE
##  8     1     1 2010-03-26       26229. FALSE
##  9     1     1 2010-04-02       57258. FALSE
## 10     1     1 2010-04-09       42961. FALSE
```

```
# Take a look at the structure of the train data set
# skim(train)
```

Summary:

Walmart provided us with 4 data sets. This is the train data set, which has 5 predictors. Within the data set, you will find the data set spans from 2012/02/05 to 2012/11/01. Some information about each predictor:

1. **store**: goes from 1 to 45
2. **dept:** department number, range from 1 to 99
3. **date:** every Friday of the week
4. **weekly_sales:** the sales for the given dept in the given store
5. **IsHoliday:** a boolean value containing T or F to indicate holiday week

```
# Read the stores data set and create desired data types
stores <- read_csv("C:/Users/chris/OneDrive/Documents/Projects/Project_5_Forecasting/walmart_sal
es/stores_dataset.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Store = col_double(),
##   Type = col_character(),
##   Size = col_double()
## )
```

```
# View the first 10 observations
head(stores, n = 10)
```

```
## # A tibble: 10 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
##  7     7 B      70713
##  8     8 A     155078
##  9     9 B     125833
## 10    10 B     126512
```

```
# Take a look at the structure of the stores data set
# skim(stores)
```

Summary:

In the stores data set, there are 3 predictors. This data set contains information on the Size and Type columns of about 45 stores. Some information about the 3 predictors:

1. **store**: 45 stores, labeled as 1 to 45
2. **size**: the size of a given store that is identified by the numbers of products available in the store ranging from 34k to 210k
3. **type:** 3 types(labeled as A, B, C)

```
# Read the features data set and create desired data types
features <- read_csv("C:/Users/chris/OneDrive/Documents/Projects/Project_5_Forecasting/walmart_s
ales/features_dataset.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Store = col_double(),
##   Date = col_date(format = ""),
##   Temperature = col_double(),
##   Fuel_Price = col_double(),
##   MarkDown1 = col_double(),
##   MarkDown2 = col_double(),
##   MarkDown3 = col_double(),
##   MarkDown4 = col_double(),
##   MarkDown5 = col_double(),
##   CPI = col_double(),
##   Unemployment = col_double(),
##   IsHoliday = col_logical()
## )
```

```
# View the first 10 observations
head(features, n = 10)
```

```
## # A tibble: 10 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
##  7     1 2010-03-19        54.6       2.72        NA        NA        NA
##  8     1 2010-03-26        51.4       2.73        NA        NA        NA
##  9     1 2010-04-02        62.3       2.72        NA        NA        NA
## 10     1 2010-04-09        65.9       2.77        NA        NA        NA
## # ... with 5 more variables: MarkDown4 <dbl>, MarkDown5 <dbl>, CPI <dbl>,
## #   Unemployment <dbl>, IsHoliday <lgl>
```

```
# Take a look at the structure of the features data set
# skim(features)
```

Summary:

In this data set, there are 12 predictors. Some new information on some of the predictors that aren't found in the stores & train data sets:

1. **temperature**: temperature of a region for a given week

2. **fuel_price**: fuel price of a region for a given week
3. **markdown1-5:** contains anonymized data related to promotional markdowns that Walmart was running. These 5 columns were only available after Nov 2011 and were not available for all stores, so they contain many missing values represented by “NA”. (NOTE: the markdowns are known to affect sales but they are difficult to estimate which dept will be affected)
4. **CPI**: consumer price index
5. **unemployment**: an unemployment rate of a given week in a region of a given store

```
# Read the test data set and create desired data types
test <- read_csv("C:/Users/chris/OneDrive/Documents/Projects/Project_5_Forecasting/walmart_sale
s/test_dataset.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Store = col_double(),
##   Dept = col_double(),
##   Date = col_date(format = ""),
##   IsHoliday = col_logical()
## )
```

```
# View the first 10 observations
head(test, n = 10)
```

```
## # A tibble: 10 x 4
##    Store  Dept Date       IsHoliday
##    <dbl> <dbl> <date>     <lgl>
##  1     1     1 2012-11-02 FALSE
##  2     1     1 2012-11-09 FALSE
##  3     1     1 2012-11-16 FALSE
##  4     1     1 2012-11-23 TRUE
##  5     1     1 2012-11-30 FALSE
##  6     1     1 2012-12-07 FALSE
##  7     1     1 2012-12-14 FALSE
##  8     1     1 2012-12-21 FALSE
##  9     1     1 2012-12-28 TRUE
## 10     1     1 2013-01-04 FALSE
```

```
# Take a look at the structure of the test data set
# skim(test)
```

Summary:

This is the test data set, which has 4 predictors. Within the data set, you will find the data set spans from 2012-11-02 to 2013-07-26. The test data set is pretty similar to the train data set, except it doesn’t has the response variable - the Weekly_Sales.

1.4 Merge Data sets

```
# Merge train & features data set and saved into newly created df
df <- merge(train, features)
# Merge stores data set with the df variable
train_df <- merge(df, stores, by = "Store")
# View the first 10 observations
head(train_df, n = 10)
```

```
##    Store       Date IsHoliday Dept Weekly_Sales Temperature Fuel_Price
## 1      1 2010-02-05     FALSE    1     24924.50       42.31      2.572
## 2      1 2010-02-05     FALSE   26     11737.12       42.31      2.572
## 3      1 2010-02-05     FALSE   17     13223.76       42.31      2.572
## 4      1 2010-02-05     FALSE   45        37.44       42.31      2.572
## 5      1 2010-02-05     FALSE   28      1085.29       42.31      2.572
## 6      1 2010-02-05     FALSE   79     46729.77       42.31      2.572
## 7      1 2010-02-05     FALSE   55     21249.31       42.31      2.572
## 8      1 2010-02-05     FALSE    5     32229.38       42.31      2.572
## 9      1 2010-02-05     FALSE   58      7659.97       42.31      2.572
## 10     1 2010-02-05     FALSE    7     21084.08       42.31      2.572
##    MarkDown1 MarkDown2 MarkDown3 MarkDown4 MarkDown5      CPI Unemployment Type
## 1         NA        NA        NA        NA        NA 211.0964        8.106    A
## 2         NA        NA        NA        NA        NA 211.0964        8.106    A
## 3         NA        NA        NA        NA        NA 211.0964        8.106    A
## 4         NA        NA        NA        NA        NA 211.0964        8.106    A
## 5         NA        NA        NA        NA        NA 211.0964        8.106    A
## 6         NA        NA        NA        NA        NA 211.0964        8.106    A
## 7         NA        NA        NA        NA        NA 211.0964        8.106    A
## 8         NA        NA        NA        NA        NA 211.0964        8.106    A
## 9         NA        NA        NA        NA        NA 211.0964        8.106    A
## 10        NA        NA        NA        NA        NA 211.0964        8.106    A
##      Size
## 1  151315
## 2  151315
## 3  151315
## 4  151315
## 5  151315
## 6  151315
## 7  151315
## 8  151315
## 9  151315
## 10 151315
```

Summary:

- merged stores & features data sets with the train data set

1.5 Split Date into Year, Month, Week, Day

```
# Split into year, month, week, day
train_df <- train_df %>%
  dplyr::mutate(Year = lubridate::year(train_df$Date),  # using mutate() function from dplyr pac
kage
                Month = lubridate::month(train_df$Date),
                Week = lubridate::week(train_df$Date),
                Day = lubridate::day(train_df$Date)
                )
head(train_df)
```

```
##   Store       Date IsHoliday Dept Weekly_Sales Temperature Fuel_Price MarkDown1
## 1     1 2010-02-05     FALSE    1     24924.50       42.31      2.572        NA
## 2     1 2010-02-05     FALSE   26     11737.12       42.31      2.572        NA
## 3     1 2010-02-05     FALSE   17     13223.76       42.31      2.572        NA
## 4     1 2010-02-05     FALSE   45        37.44       42.31      2.572        NA
## 5     1 2010-02-05     FALSE   28      1085.29       42.31      2.572        NA
## 6     1 2010-02-05     FALSE   79     46729.77       42.31      2.572        NA
##   MarkDown2 MarkDown3 MarkDown4 MarkDown5      CPI Unemployment Type   Size
## 1        NA        NA        NA        NA 211.0964        8.106    A 151315
## 2        NA        NA        NA        NA 211.0964        8.106    A 151315
## 3        NA        NA        NA        NA 211.0964        8.106    A 151315
## 4        NA        NA        NA        NA 211.0964        8.106    A 151315
## 5        NA        NA        NA        NA 211.0964        8.106    A 151315
## 6        NA        NA        NA        NA 211.0964        8.106    A 151315
##   Year Month Week Day
## 1 2010     2    6   5
## 2 2010     2    6   5
## 3 2010     2    6   5
## 4 2010     2    6   5
## 5 2010     2    6   5
## 6 2010     2    6   5
```

Summary:

- split the Date column into year, month, week, and day

1.6 Create Unique Identifier

```
# Create a function to combine store & dept and form an ID
unique_dept_store <- function(data){
  mutate(data, Store_Dept = paste0(Store, "_", Dept),
         .before = 1)
}

# Apply the function to both data sets - train & test
train_forecast <- unique_dept_store((train_df))
test_forecast <- unique_dept_store((test))

head(train_forecast)
```

```
##   Store_Dept Store       Date IsHoliday Dept Weekly_Sales Temperature
## 1        1_1     1 2010-02-05     FALSE    1     24924.50       42.31
## 2       1_26     1 2010-02-05     FALSE   26     11737.12       42.31
## 3       1_17     1 2010-02-05     FALSE   17     13223.76       42.31
## 4       1_45     1 2010-02-05     FALSE   45        37.44       42.31
## 5       1_28     1 2010-02-05     FALSE   28      1085.29       42.31
## 6       1_79     1 2010-02-05     FALSE   79     46729.77       42.31
##   Fuel_Price MarkDown1 MarkDown2 MarkDown3 MarkDown4 MarkDown5      CPI
## 1      2.572        NA        NA        NA        NA        NA 211.0964
## 2      2.572        NA        NA        NA        NA        NA 211.0964
## 3      2.572        NA        NA        NA        NA        NA 211.0964
## 4      2.572        NA        NA        NA        NA        NA 211.0964
## 5      2.572        NA        NA        NA        NA        NA 211.0964
## 6      2.572        NA        NA        NA        NA        NA 211.0964
##   Unemployment Type   Size Year Month Week Day
## 1        8.106    A 151315 2010     2    6   5
## 2        8.106    A 151315 2010     2    6   5
## 3        8.106    A 151315 2010     2    6   5
## 4        8.106    A 151315 2010     2    6   5
## 5        8.106    A 151315 2010     2    6   5
## 6        8.106    A 151315 2010     2    6   5
```

```
head(test_forecast)
```

```
## # A tibble: 6 x 5
##   Store_Dept Store  Dept Date       IsHoliday
##   <chr>      <dbl> <dbl> <date>     <lgl>
## 1 1_1            1     1 2012-11-02 FALSE
## 2 1_1            1     1 2012-11-09 FALSE
## 3 1_1            1     1 2012-11-16 FALSE
## 4 1_1            1     1 2012-11-23 TRUE
## 5 1_1            1     1 2012-11-30 FALSE
## 6 1_1            1     1 2012-12-07 FALSE
```

Summary:

- create a unique ID to identify Store&Dept

1.8 Re-organize the Data frame

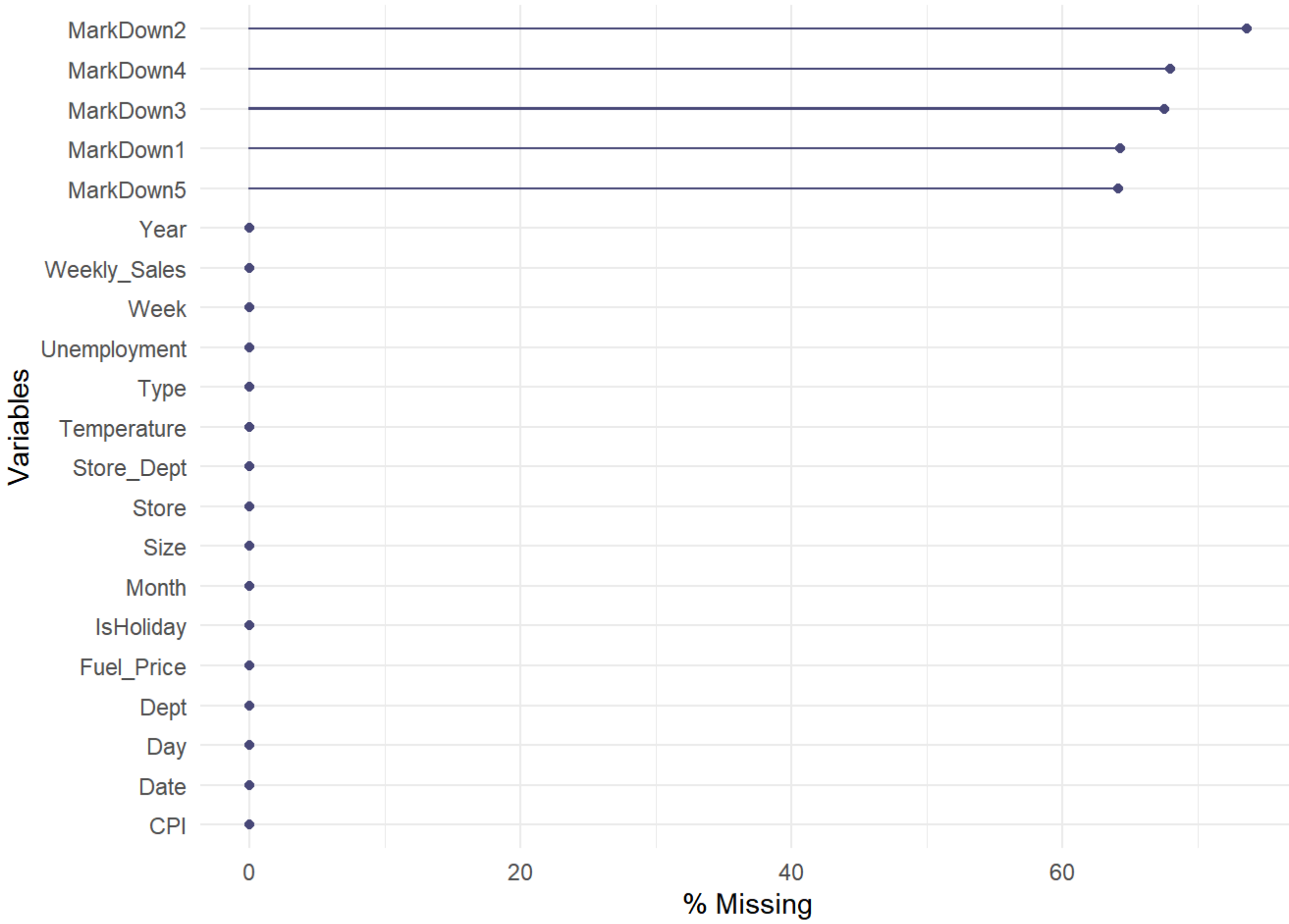
```
# re-order the dataset to have the response variable - weekly sales to the last column
col_df <- c("Date", "Year","Month", "Week", "Day", "Store_Dept", "Dept", "Store", "Type", "Size"
, "IsHoliday", "Temperature", "Fuel_Price", "CPI", "Unemployment", "MarkDown1", "MarkDown2", "Ma
rkDown3", "MarkDown4", "MarkDown5", "Weekly_Sales")

# Assign the order of the columns into the train_df
train_forecast <- train_forecast[, col_df]
head(train_forecast)
```

```
##         Date Year Month Week Day Store_Dept Dept Store Type   Size IsHoliday
## 1 2010-02-05 2010     2    6   5        1_1    1     1    A 151315     FALSE
## 2 2010-02-05 2010     2    6   5       1_26   26     1    A 151315     FALSE
## 3 2010-02-05 2010     2    6   5       1_17   17     1    A 151315     FALSE
## 4 2010-02-05 2010     2    6   5       1_45   45     1    A 151315     FALSE
## 5 2010-02-05 2010     2    6   5       1_28   28     1    A 151315     FALSE
## 6 2010-02-05 2010     2    6   5       1_79   79     1    A 151315     FALSE
##   Temperature Fuel_Price      CPI Unemployment MarkDown1 MarkDown2 MarkDown3
## 1       42.31      2.572 211.0964        8.106        NA        NA        NA
## 2       42.31      2.572 211.0964        8.106        NA        NA        NA
## 3       42.31      2.572 211.0964        8.106        NA        NA        NA
## 4       42.31      2.572 211.0964        8.106        NA        NA        NA
## 5       42.31      2.572 211.0964        8.106        NA        NA        NA
## 6       42.31      2.572 211.0964        8.106        NA        NA        NA
##   MarkDown4 MarkDown5 Weekly_Sales
## 1        NA        NA     24924.50
## 2        NA        NA     11737.12
## 3        NA        NA     13223.76
## 4        NA        NA        37.44
## 5        NA        NA      1085.29
## 6        NA        NA     46729.77
```

1.9 Examine Missing Values

```
# Display the %s of missing values for each variable in a plot
gg_miss_var(train_forecast, show_pct = TRUE)
```

```
## Warning: It is deprecated to specify `guide = FALSE` to remove a guide. Please
## use `guide = "none"` instead.
```



Summary:

- in the train data set, as displayed above, only columns MarkDown1 to 5 have missing values
- MarkDown1 to 5 is consists of more than 50% of missing values
- as mentioned earlier, they are anonymized columns and they correspond to the promotional activities being carried out at different stores

Let's perform further analysis into the data set to decide whether or not we should utilize data imputation methods to estimate the missing values.

2. Explanatory Data Analysis

```
# Create a new df for visualization only
forecasting_vis <- train_df
```

2.1 Numerical Variables Visualization

2.1.1 Relationship of Dept vs Weekly Sales

```
# Take the average weekly sales and arrange from largest to smallest
d1 <- forecasting_vis %>%
  group_by(Dept, Year) %>%
  summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
  arrange(desc(avg_weeklysales))
```

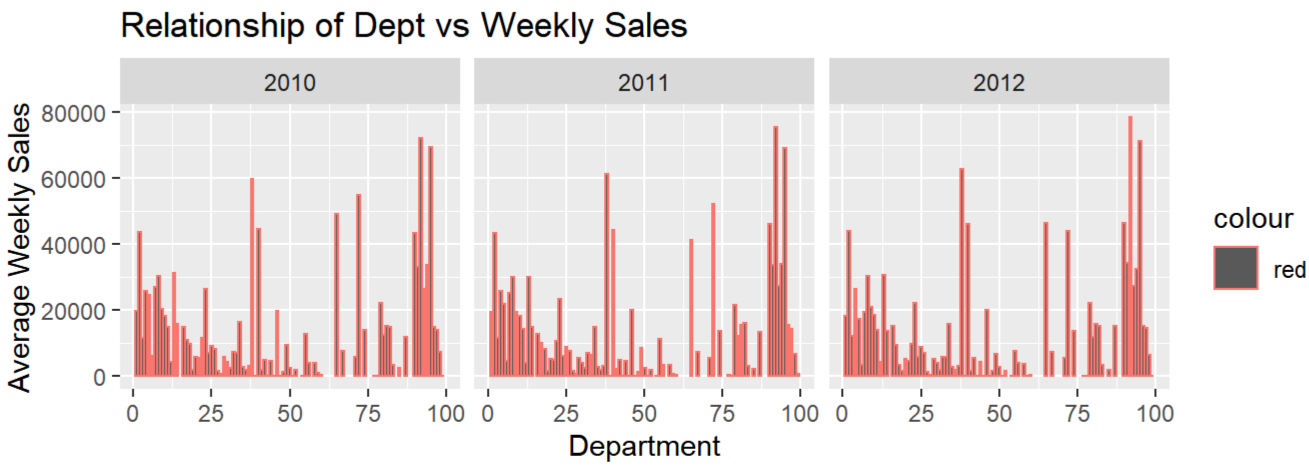
```
## `summarise()` has grouped output by 'Dept'. You can override using the `.groups` argument.
```

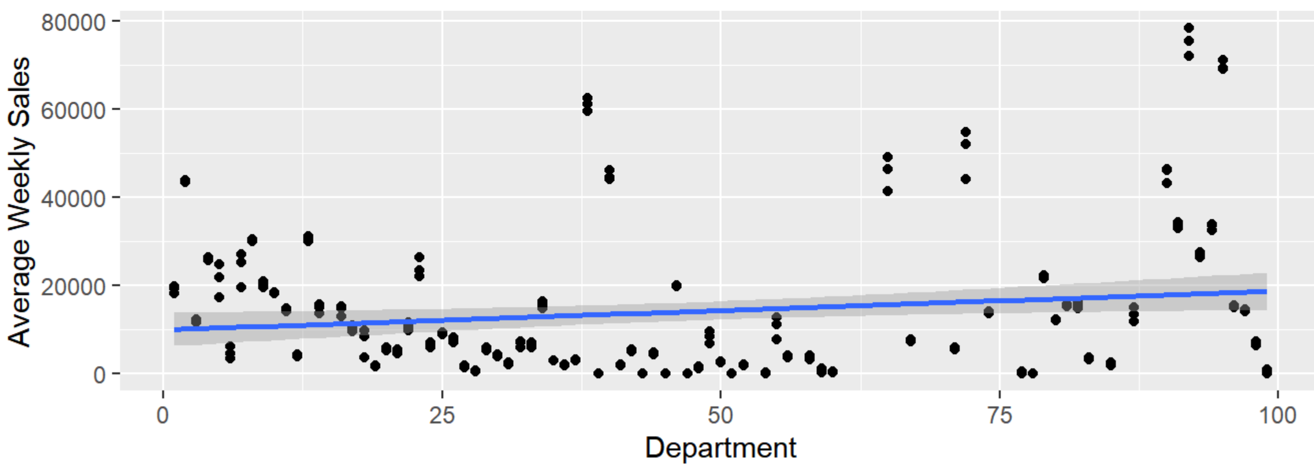
```
d1
```

```
## # A tibble: 243 x 3
## # Groups:   Dept [81]
##     Dept  Year avg_weeklysales
##    <dbl> <dbl>           <dbl>
##  1    92  2012          78361.
##  2    92  2011          75417.
##  3    92  2010          72147.
##  4    95  2012          71262.
##  5    95  2010          69379.
##  6    95  2011          69047.
##  7    38  2012          62533.
##  8    38  2011          61223.
##  9    38  2010          59655.
## 10    72  2010          54707.
## # ... with 233 more rows
```

```
# Create a bar plot for Dept vs Weekly Sales
b1 <- ggplot(data = d1, aes(x = Dept,y = avg_weeklysales, color = "red")) +
  geom_col() +
  facet_wrap(~Year) +
  labs(x = "Department", y = "Average Weekly Sales") +
        ggtitle("Relationship of Dept vs Weekly Sales")


# Create a scatter plot for Dept vs Weekly Sales
s1 <- ggplot(d1, aes(x=Dept, y=avg_weeklysales)) + geom_point() +
  scale_colour_hue(l=50) + # Use a slightly darker palette than normal
  geom_smooth(method=lm,   # Add linear regression lines
              se=TRUE,    # Don't add shaded confidence region
              fullrange=TRUE) + # Extend regression lines
              labs(x = "Department", y = "Average Weekly Sales")
# Organized the plots in one page
grid.arrange(b1, s1, nrow=2)
```

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






```
cor(forecasting_vis$Dept,forecasting_vis$Weekly_Sales)
```

```
## [1] 0.1480321
```

Summary:

- bar plot:department **#92** has the **highest** average weekly sales for 2010 - 2012 that are around 70k vs department **#47** in 2010 and 2011 with a **lowest(negative)** average sales
- scatter plot:this plot showcases the linearity of x-variable **Dept** vs y-variable **Avg Weekly Sales**, we can see the linear line is more straight which indicate it has **a bare minimal relationship**

2.1.1.2 Examine Top Ten Departments by Average Weekly Sales

```
# Take the average weekly sales by dept and arrange from desc to asce
# For 2010 Only
top_dept_2010 <- forecasting_vis %>%
  group_by(Dept) %>%
  filter(Year == "2010") %>%
  summarise(Top_Avg_Wklysales = mean(Weekly_Sales)) %>%
  # rename(Top_Avg_Wklysales = Freq) %>%
  arrange(desc(Top_Avg_Wklysales))
top_dept_2010
```

```
## # A tibble: 81 x 2
##     Dept Top_Avg_Wklysales
##    <dbl>             <dbl>
##  1    92            72147.
##  2    95            69379.
##  3    38            59655.
##  4    72            54707.
##  5    65            49096.
##  6    40            44637.
##  7     2            43543.
##  8    90            43243.
##  9    94            33717.
## 10    91            33067.
## # ... with 71 more rows
```

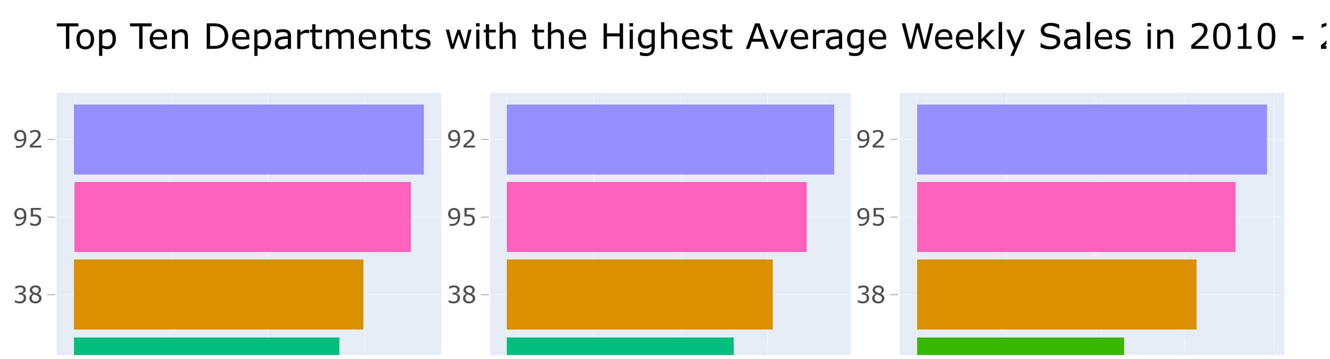
```
# Create a plot to display top ten highest weekly sales by dept - 2010
p_2010 <- head(top_dept_2010, n = 10) %>%
  ggplot(aes(x = reorder(as.factor(Dept),
                 Top_Avg_Wklysales),
                 y = Top_Avg_Wklysales,
                 fill=as.factor(Dept))) +
  geom_bar(stat = 'identity') +
  theme(legend.position = "none")+
  labs(y = "Total Weekly Sales", x = 'Departments', title = "Top Ten Departments with the Highes
t Average Weekly Sales - 2010") +
  coord_flip()


# For 2011 Only
top_dept_2011 <- forecasting_vis %>%
  group_by(Dept) %>%
  filter(Year == "2011") %>%
  summarise(Top_Avg_Wklysales = mean(Weekly_Sales)) %>%
  # rename(Top_Avg_Wklysales = Freq) %>%
  arrange(desc(Top_Avg_Wklysales))
top_dept_2011
```

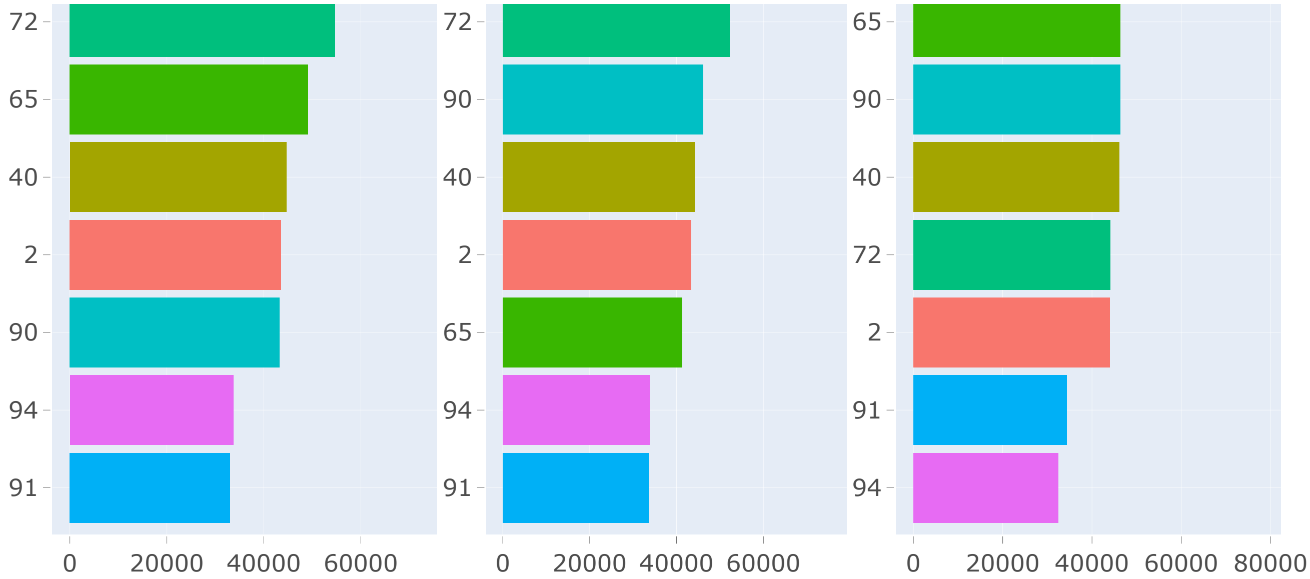
```
## # A tibble: 81 x 2
##     Dept Top_Avg_Wklysales
##    <dbl>             <dbl>
##  1    92            75417.
##  2    95            69047.
##  3    38            61223.
##  4    72            52184.
##  5    90            46132.
##  6    40            44155.
##  7     2            43378.
##  8    65            41301.
##  9    94            33959.
## 10    91            33704.
## # ... with 71 more rows
```

```
# Create a plot to display top ten highest weekly sales by dept - 2011
p_2011 <- head(top_dept_2011, n = 10) %>%
  ggplot(aes(x = reorder(as.factor(Dept),
                 Top_Avg_Wklysales),
                 y = Top_Avg_Wklysales,
                 fill=as.factor(Dept))) +
  geom_bar(stat = 'identity') +
  theme(legend.position = "none")+
  labs(y = "Total Weekly Sales", x = 'Departments', title = "Top Ten Departments with the Highes
t Average Weekly Sales - 2011") +
  coord_flip()


# For 2012 Only
top_dept_2012 <- forecasting_vis %>%
  group_by(Dept) %>%
  filter(Year == "2012") %>%
  summarise(Top_Avg_Wklysales = mean(Weekly_Sales)) %>%
  # rename(Top_Avg_Wklysales = Freq) %>%
  arrange(desc(Top_Avg_Wklysales))
top_dept_2012
```

```
## # A tibble: 81 x 2
##     Dept Top_Avg_Wklysales
##    <dbl>             <dbl>
##  1    92            78361.
##  2    95            71262.
##  3    38            62533.
##  4    65            46370.
##  5    90            46365.
##  6    40            46098.
##  7    72            44028.
##  8     2            43955.
##  9    91            34361.
## 10    94            32461.
## # ... with 71 more rows
```

```
# Create a plot to display top ten highest weekly sales by dept - 2011
p_2012 <- head(top_dept_2012, n = 10) %>%
  ggplot(aes(x = reorder(as.factor(Dept),
                Top_Avg_Wklysales),
                y = Top_Avg_Wklysales,
                fill=as.factor(Dept))) +
  geom_bar(stat = 'identity') +
  theme(legend.position = "none")+
  labs(y = "Total Weekly Sales", x = 'Departments', title = "Top Ten Departments with the Highes
t Average Weekly Sales - 2011") +
  coord_flip()

# Create a subplot
fig <- subplot(p_2010, p_2011, p_2012)%>%
  layout(title = list(text = "Top Ten Departments with the Highest Average Weekly Sales in 2010
 - 2012"),
        plot_bgcolor='#e5ecf6',
        xaxis = list(
          zerolinecolor = '#ffff',
          zerolinewidth = 2,
          gridcolor = 'ffff'),
        yaxis = list(
          zerolinecolor = '#ffff',
          zerolinewidth = 2,
          gridcolor = 'ffff'))
fig
```





Summary:

- the number one dept in the top ten depts rank for all three years is **#92**, compared to the bottom one in the top ten depts rank is **#91** in 2010-2011, and **#94** in 2012, **#91** moves up one rank in **2012**
- surprisingly, there are **no depts fall off the top ten depts rank** in these three years, and **no additional depts climbed up in the top ten depts rank**

2.1.2 Relationship of Temperature vs Weekly Sales

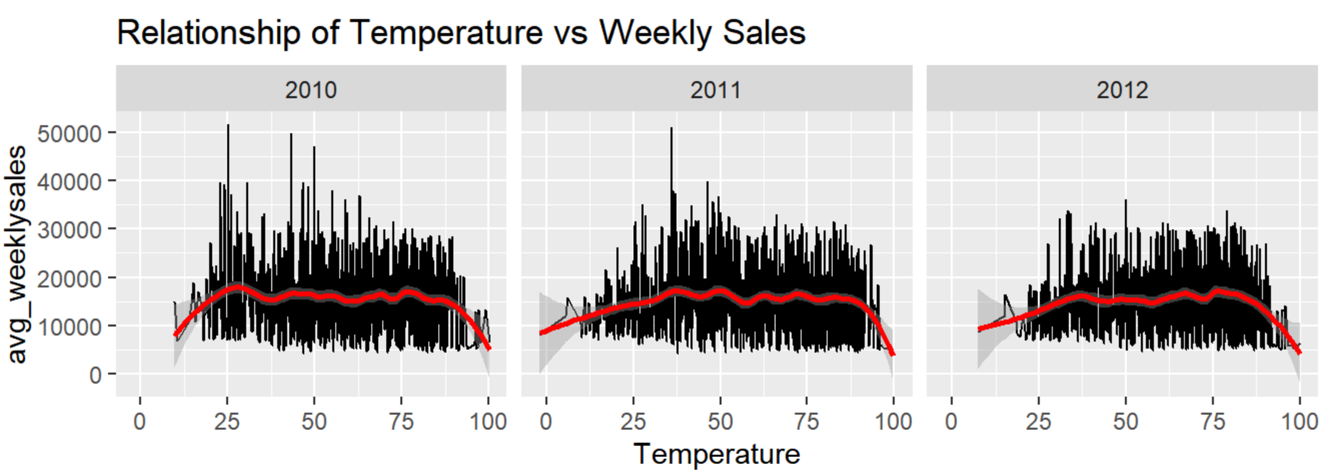
```
# Take the average weekly sales and arrange from largest to smallest
d1 <- forecasting_vis %>%
  group_by(Temperature, Year) %>%
  summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
  arrange(desc(avg_weeklysales))
```

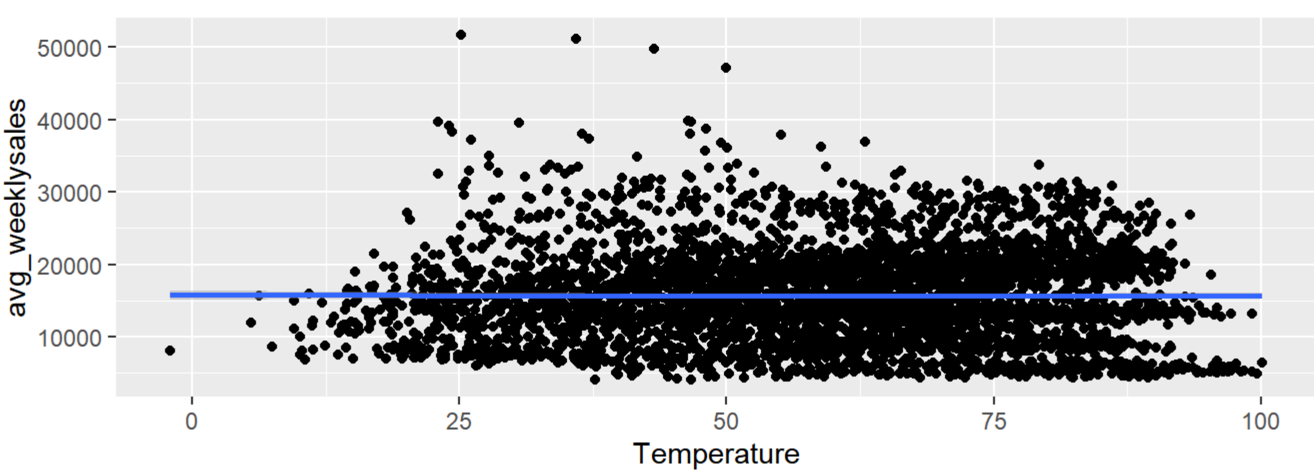
```
## `summarise()` has grouped output by 'Temperature'. You can override using the `.groups` argum
ent.
```

```
# View(d1)

# Create a line plot for Temperature vs Weekly Sales
b2 <- ggplot(data = d1, aes(x = Temperature,y = avg_weeklysales, color = "red")) +
        geom_line(color = "black") +
        geom_smooth(method = "loess", color = "red", span = 1/5) +
        ggtitle("Relationship of Temperature vs Weekly Sales") +
        facet_wrap(~Year)


# Create a scatter plot for Temperature vs Weekly Sales
s2 <- ggplot(d1, aes(x = Temperature, y = avg_weeklysales)) + geom_point() +
  scale_colour_hue(l=50) + # Use a slightly darker palette than normal
  geom_smooth(method = lm,    # Add linear regression lines
              se = TRUE,     # Don't add shaded confidence region
              fullrange = TRUE) # Extend regression lines

# Organized the plots in one page
grid.arrange(b2, s2, nrow = 2)
```

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

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





```
cor(forecasting_vis$Temperature,forecasting_vis$Weekly_Sales)
```

```
## [1] -0.002312447
```

Summary:

- line plot: the **temperature** at**25.17** in 2010 has the highest average weekly sales:**51k** vs the temperature at**37.74** in 2011 has the lowest average weekly sales:**4k**
- bar plot: shows the relationship of temperature vs average weekly sales, the line is flat which indicates there's barely any relationship which is proved by a correlation of**-0.0023**

2.1.3 Relationship of Fuel_Price vs Weekly Sales

```
# Take the average weekly sales and arrange from largest to smallest
d1 <- forecasting_vis %>%
  group_by(Fuel_Price, Year) %>%
  summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
  arrange(desc(avg_weeklysales))
```

```
## `summarise()` has grouped output by 'Fuel_Price'. You can override using the `.groups` argume
nt.
```

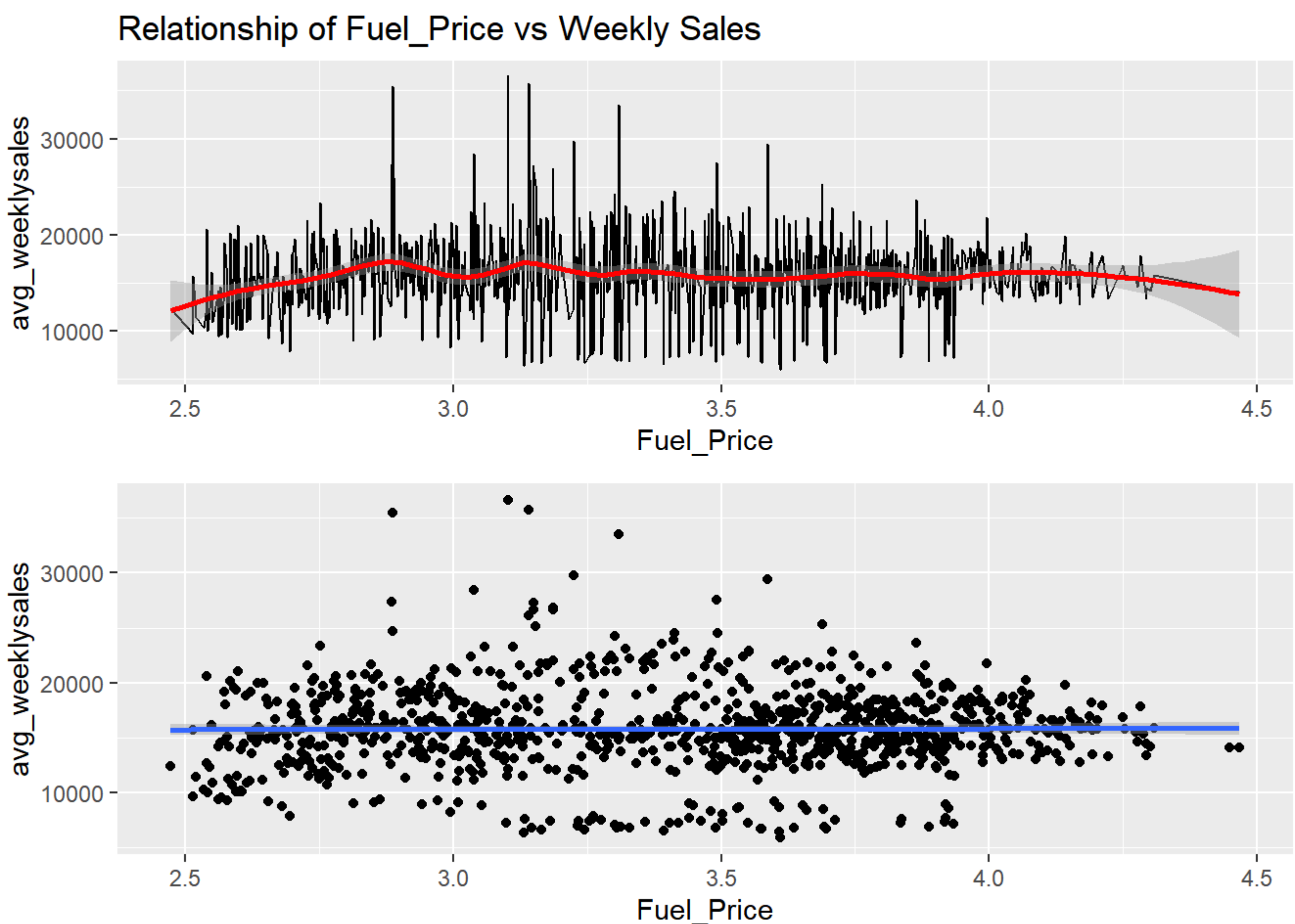
```
# view(d1)

# Create a line plot for Fuel_Price vs Weekly Sales
b3 <- ggplot(data = d1, aes(x = Fuel_Price,y = avg_weeklysales, color = "red")) +
        geom_line(color = "black") +
        geom_smooth(method = "loess", color = "red", span = 1/5) +
        ggtitle("Relationship of Fuel_Price vs Weekly Sales")
        facet_wrap(~Year)
```

```
## <ggproto object: Class FacetWrap, Facet, gg>
##     compute_layout: function
##     draw_back: function
##     draw_front: function
##     draw_labels: function
##     draw_panels: function
##     finish_data: function
##     init_scales: function
##     map_data: function
##     params: list
##     setup_data: function
##     setup_params: function
##     shrink: TRUE
##     train_scales: function
##     vars: function
##     super:  <ggproto object: Class FacetWrap, Facet, gg>
```

```
# Create a scatter plot for Fuel_Price vs Weekly Sales
s3 <- ggplot(d1, aes(x=Fuel_Price, y=avg_weeklysales)) + geom_point() +
  scale_colour_hue(l=50) + # Use a slightly darker palette than normal
  geom_smooth(method=lm,   # Add linear regression lines
              se=TRUE,    # Don't add shaded confidence region
              fullrange=TRUE) # Extend regression lines

# Organized the plots in one page
grid.arrange(b3, s3, nrow=2)
```

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

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




```
cor(forecasting_vis$Fuel_Price,forecasting_vis$Weekly_Sales)
```

```
## [1] -0.0001202955
```

Summary:

- line plot: the fuel price at**$3.10** in**2011** has the highest average weekly sales:**36k** vs the fuel price at**$3.61** in**2012** has the lowest average weekly sales: **5k**

- bar plot: shows the relationship of fuel price vs average weekly sales, the line is flat which indicates there's barely any relationship which is proved by a correlation of**-0.00012**

2.1.4 Relationship of MarkDown1-MarkDown5 vs Weekly Sales

```
# Create a bar plot for MarkDown1 vs Weekly Sales
b4 <- forecasting_vis %>%
    group_by(MarkDown1, Year) %>%
    summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
    arrange(desc(avg_weeklysales)) %>%
    ggplot(aes(x = MarkDown1,y = avg_weeklysales, color = "red")) +
      geom_col() +
      facet_wrap(~Year) +
      ggtitle("MarkDown1 vs Weekly Sales")
```

```
## `summarise()` has grouped output by 'MarkDown1'. You can override using the `.groups` argumen
t.
```

```
# Create a bar plot for MarkDown2 vs Weekly Sales
b5 <- forecasting_vis %>%
    group_by(MarkDown2, Year) %>%
    summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
    arrange(desc(avg_weeklysales)) %>%
    ggplot(aes(x = MarkDown2,y = avg_weeklysales, color = "red")) +
      geom_col() +
      facet_wrap(~Year) +
      ggtitle("MarkDown2 vs Weekly Sales")
```

```
## `summarise()` has grouped output by 'MarkDown2'. You can override using the `.groups` argumen
t.
```

```
# Create a bar plot for MarkDown3 vs Weekly Sales
b6 <- forecasting_vis %>%
      group_by(MarkDown3, Year) %>%
      summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
      arrange(desc(avg_weeklysales)) %>%
      ggplot(aes(x = MarkDown3,y = avg_weeklysales, color = "red")) +
        geom_col() +
        facet_wrap(~Year) +
        ggtitle("MarkDown3 vs Weekly Sales")
```

```
## `summarise()` has grouped output by 'MarkDown3'. You can override using the `.groups` argumen
t.
```

```
# Create a bar plot for MarkDown4 vs Weekly Sales
b7 <- forecasting_vis %>%
      group_by(MarkDown4, Year) %>%
      summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
      arrange(desc(avg_weeklysales)) %>%
      ggplot(aes(x = MarkDown4,y = avg_weeklysales, color = "red")) +
        geom_col() +
        facet_wrap(~Year) +
        ggtitle("MarkDown4 vs Weekly Sales")
```

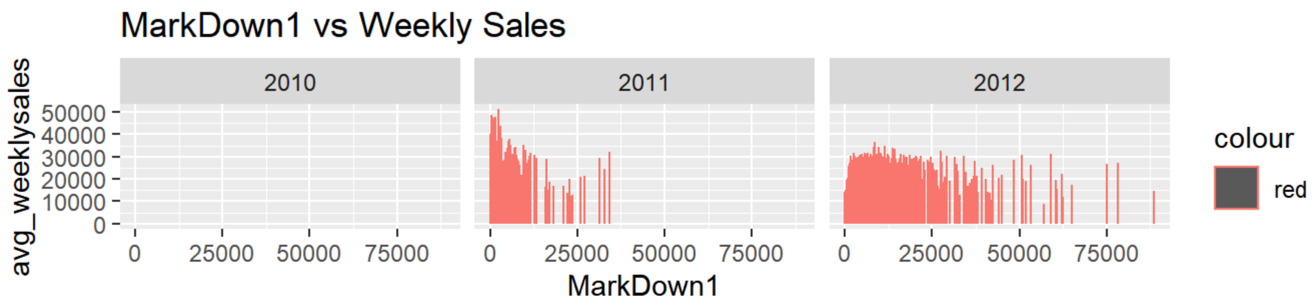
```
## `summarise()` has grouped output by 'MarkDown4'. You can override using the `.groups` argumen
t.
```

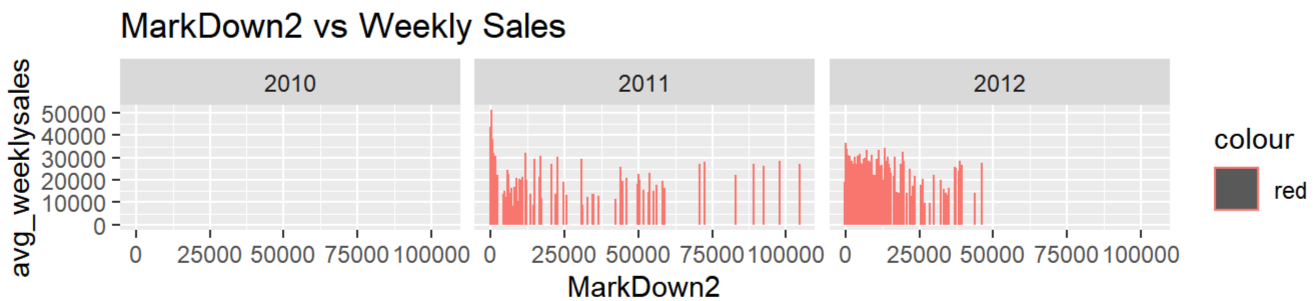
```
# Create a bar plot for MarkDown5 vs Weekly Sales
b8 <- forecasting_vis %>%
      group_by(MarkDown5, Year) %>%
      summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
      arrange(desc(avg_weeklysales)) %>%
      ggplot(aes(x = MarkDown5,y = avg_weeklysales, color = "red")) +
        geom_col() +
        facet_wrap(~Year) +
        ggtitle("MarkDown5 vs Weekly Sales")
```

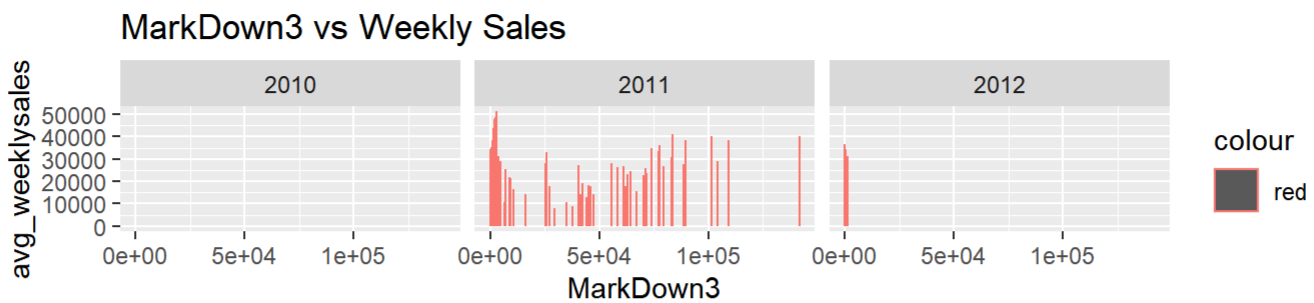
```
## `summarise()` has grouped output by 'MarkDown5'. You can override using the `.groups` argumen
t.
```

```
# Organized the plots in one page
grid.arrange(b4, b5, b6, nrow=3)
```

```
## Warning: Removed 3 rows containing missing values (position_stack).

## Warning: Removed 3 rows containing missing values (position_stack).

## Warning: Removed 3 rows containing missing values (position_stack).
```



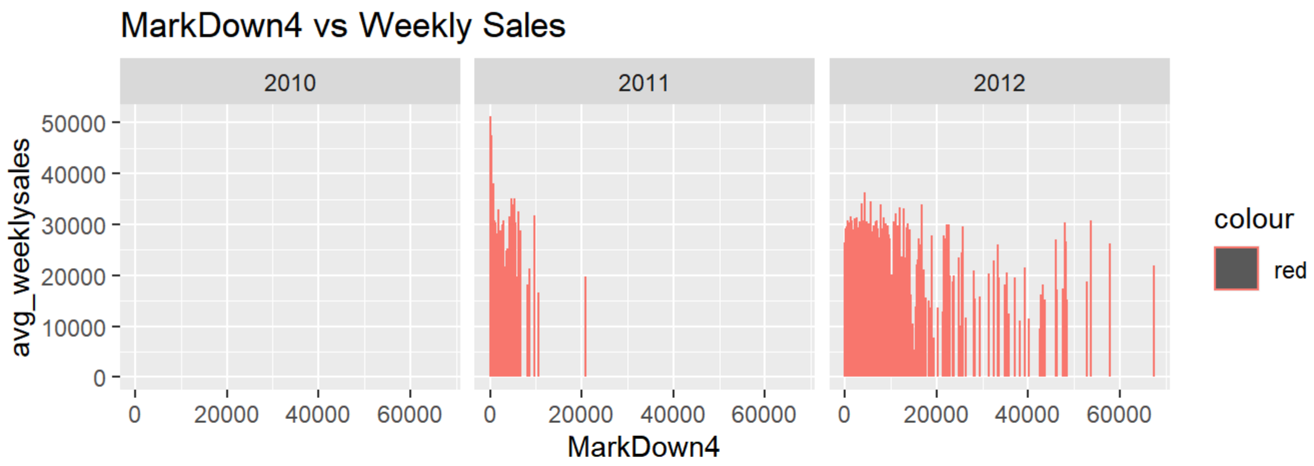


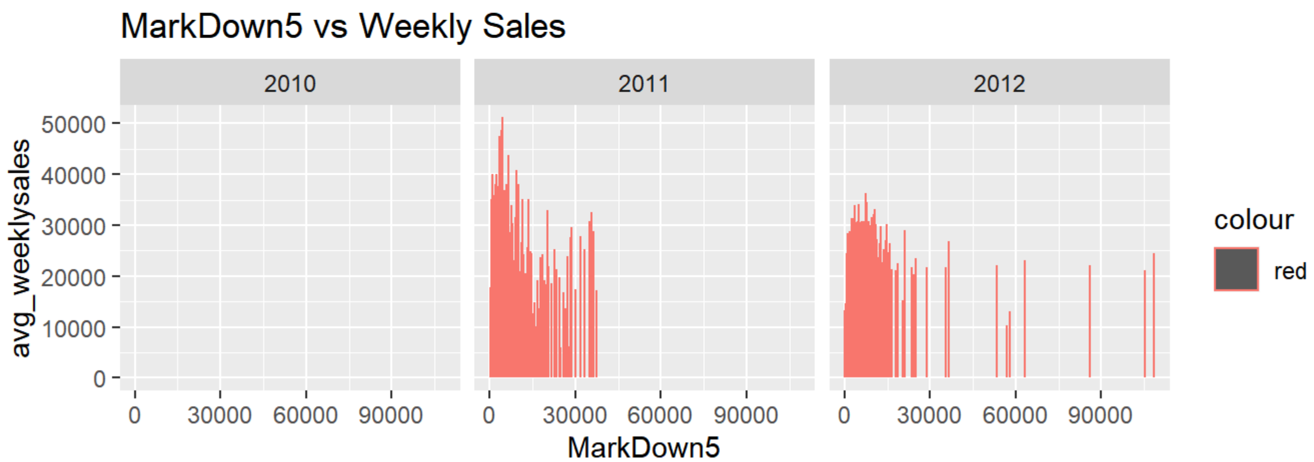



```
grid.arrange(b7, b8, nrow=2)
```

```
## Warning: Removed 3 rows containing missing values (position_stack).
```

```
## Warning: Removed 2 rows containing missing values (position_stack).
```





Summary:

- since MarkDown1 to MarkDown5 contain anonymized information, as we can see above there aren't much we can draw from these plots besides knowing the lowest and highest range
    - All MarkDowns 1 to 5 have missing values in 2010
    - MarkDown1 ranges from 0.27 to 88k
    - MarkDown2 ranges from a negative 265.8 to 104k
    - MarkDown3 ranges from a negative 29.10 to 141k
    - MarkDown4 ranges from 0.22 to 67k
    - MarkDown5 ranges from 135.2 to 108k

2.1.5 Relationship of CPI vs Weekly Sales

```
# Take the average weekly sales and arrange from largest to smallest
d1 <- forecasting_vis %>%
  group_by(CPI, Year) %>%
  summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
  arrange(desc(avg_weeklysales))
```

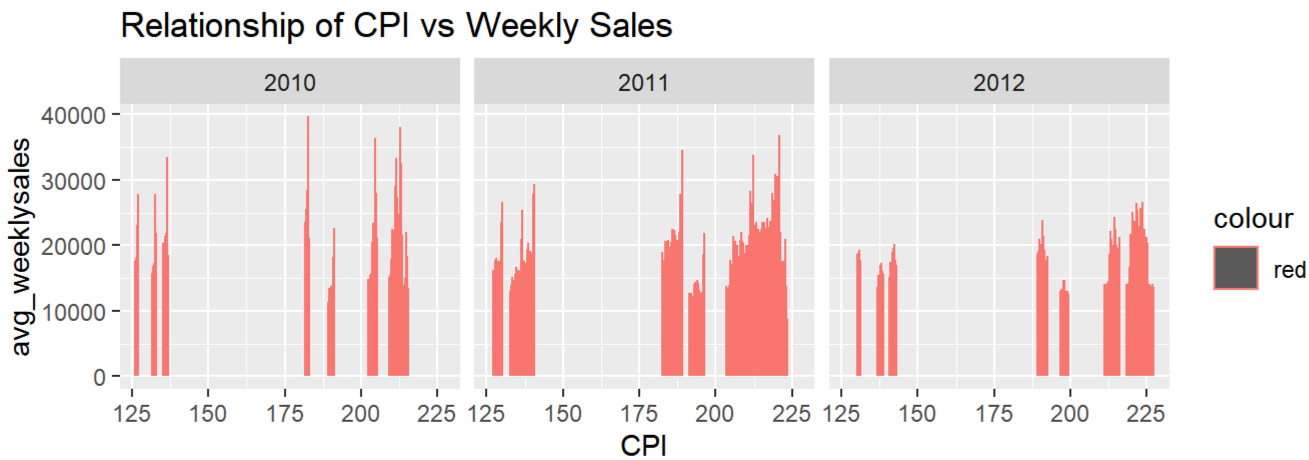
```
## `summarise()` has grouped output by 'CPI'. You can override using the `.groups` argument.
```

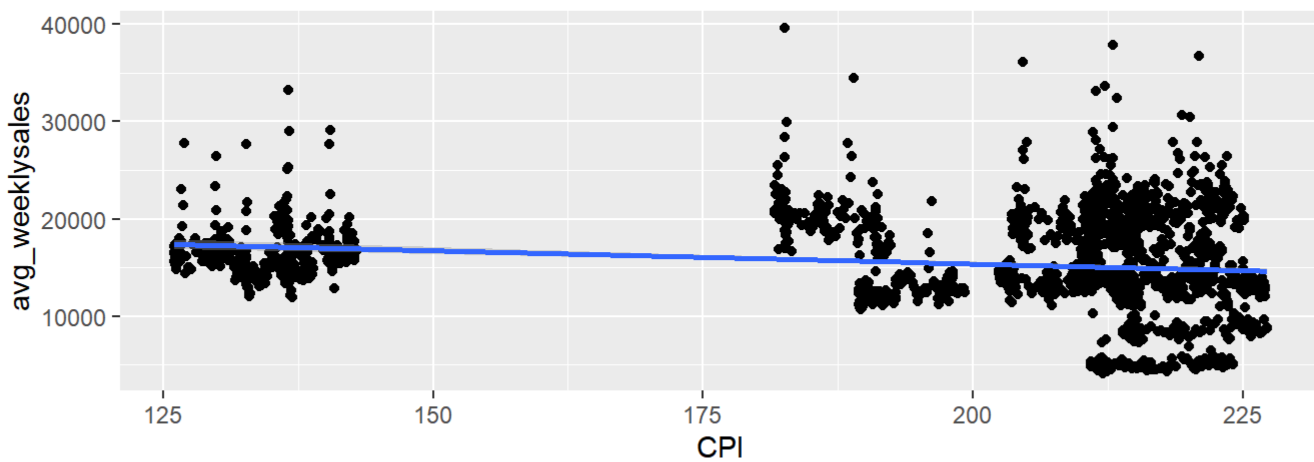
```
d1
```

```
## # A tibble: 2,145 x 3
## # Groups:   CPI [2,145]
##      CPI  Year avg_weeklysales
##    <dbl> <dbl>           <dbl>
##  1  183.  2010          39579.
##  2  213.  2010          37883.
##  3  221.  2011          36731.
##  4  205.  2010          36158.
##  5  189.  2011          34444.
##  6  212.  2011          33652.
##  7  137.  2010          33268.
##  8  211.  2010          33166.
##  9  213.  2010          32392.
## 10  219.  2011          30678.
## # ... with 2,135 more rows
```

```
# Create a bar plot for CPI vs Weekly Sales
b9 <- ggplot(data = d1, aes(x = CPI,y = avg_weeklysales, color = "red")) +
      geom_col() +
      ggtitle("Relationship of CPI vs Weekly Sales") +
      facet_wrap(~Year)

# Create a scatter plot for CPI vs Weekly Sales
s9 <- ggplot(d1, aes(x=CPI, y=avg_weeklysales)) + geom_point() +
  scale_colour_hue(l=50) + # Use a slightly darker palette than normal
  geom_smooth(method=lm,   # Add linear regression lines
              se=TRUE,    # Don't add shaded confidence region
              fullrange=TRUE) # Extend regression lines

# Organized the plots in one page
grid.arrange(b9, s9, nrow=2)
```

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






```
cor(forecasting_vis$CPI,forecasting_vis$Weekly_Sales)
```

```
## [1] -0.02092134
```

Summary:

- bar plot: CPI **182.55~** in **2010** has the highest average weekly sales: **39k** compared to CPI **132.11~** in 2010 has the lowest average weekly sales: **15k**
- scatter plot: this plot showcases the linearity of x-variable **CPI** vs y-variable **Avg Weekly Sales**, we can see the linear line is more straight, which indicate it has **a bare minimal relationship** and it is confirmed through a correlation of **-0.021**

2.1.6 Relationship of Unemployment vs Weekly Sales

```
# Take the average weekly sales and arrange from largest to smallest
d1 <- forecasting_vis %>%
  group_by(Unemployment, Year) %>%
  summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
  arrange(desc(avg_weeklysales))
```

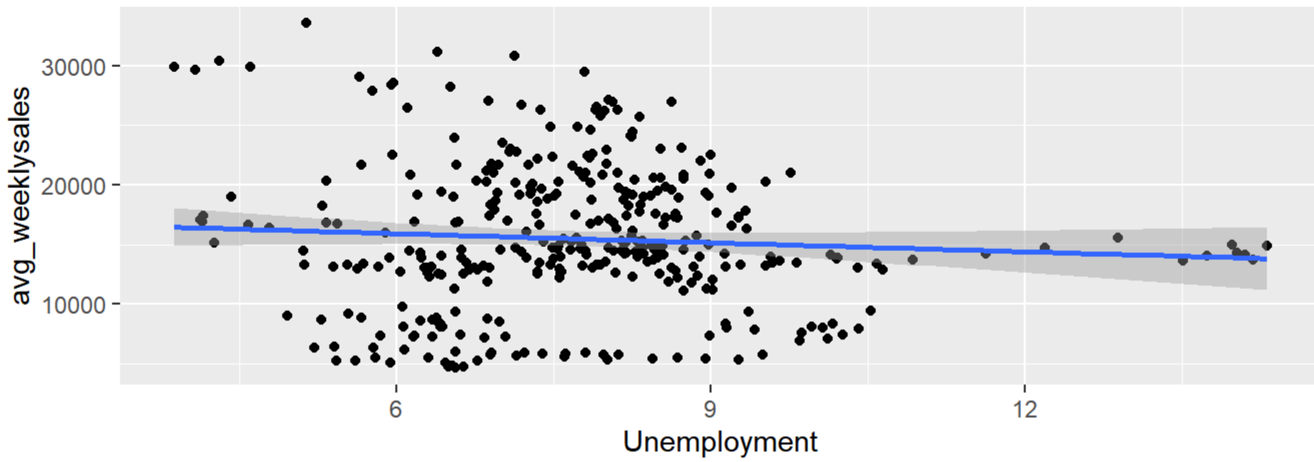
```
## `summarise()` has grouped output by 'Unemployment'. You can override using the `.groups` argu
ment.
```

```
d1
```

```
## # A tibble: 354 x 3
## # Groups:   Unemployment [349]
##    Unemployment  Year avg_weeklysales
##           <dbl> <dbl>           <dbl>
##  1         5.14  2011          33559.
##  2         6.39  2011          31117.
##  3         7.13  2010          30828.
##  4         4.31  2012          30432.
##  5         3.88  2012          29929.
##  6         4.61  2012          29872.
##  7         4.08  2012          29634.
##  8         7.80  2010          29482.
##  9         5.64  2011          29092.
## 10         5.96  2012          28551.
## # ... with 344 more rows
```

```
# Create a bar plot for Unemployment vs Weekly Sales
b10 <- ggplot(data = d1, aes(x = Unemployment,y = avg_weeklysales, color = "red")) +
      geom_col() +
      ggtitle("Relationship of Unemployment vs Weekly Sales") +
      facet_wrap(~Year)

# Create a scatter plot for Unemployment vs Weekly Sales
s10 <- ggplot(d1, aes(x=Unemployment, y=avg_weeklysales)) + geom_point() +
  scale_colour_hue(l=50) + # Use a slightly darker palette than normal
  geom_smooth(method=lm,   # Add linear regression lines
              se=TRUE,    # Don't add shaded confidence region
              fullrange=TRUE) # Extend regression lines

# Organized the plots in one page
grid.arrange(b10, s10, nrow=2)
```

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




```
cor(forecasting_vis$Unemployment,forecasting_vis$Weekly_Sales)
```

```
## [1] -0.02586372
```

Summary:

- bar plot: unemployment at 5.14% in 2011 has the highest average weekly sales: 33k compared to unemployment at 6.57~ in 2010 has the lowest average weekly sales: 4k
- scatter plot: this plot showcases the linearity of x-variable **unemployment** vs y-variable **Avg Weekly Sales**, we can see the linear line is more straight, which indicate it has **a bare minimal relationship** and it is confirmed through a correlation of **-0.025**

2.1.7 Relationship of Size vs Weekly Sales

```
# Take the average weekly sales and arrange from largest to smallest
d1 <- forecasting_vis %>%
  group_by(Size, Year) %>%
  summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
  arrange(desc(avg_weeklysales))
```

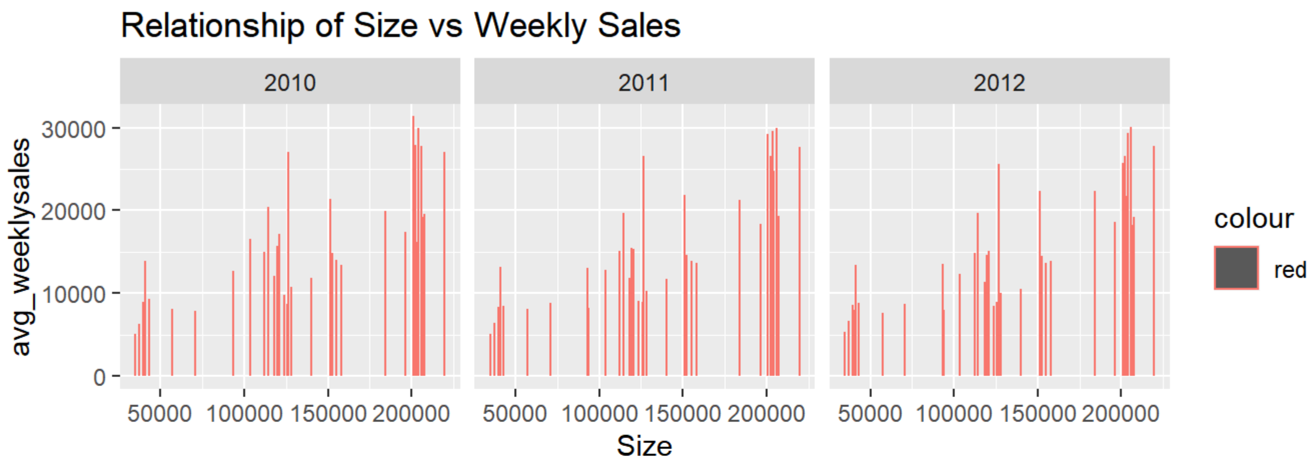
```
## `summarise()` has grouped output by 'Size'. You can override using the `.groups` argument.
```

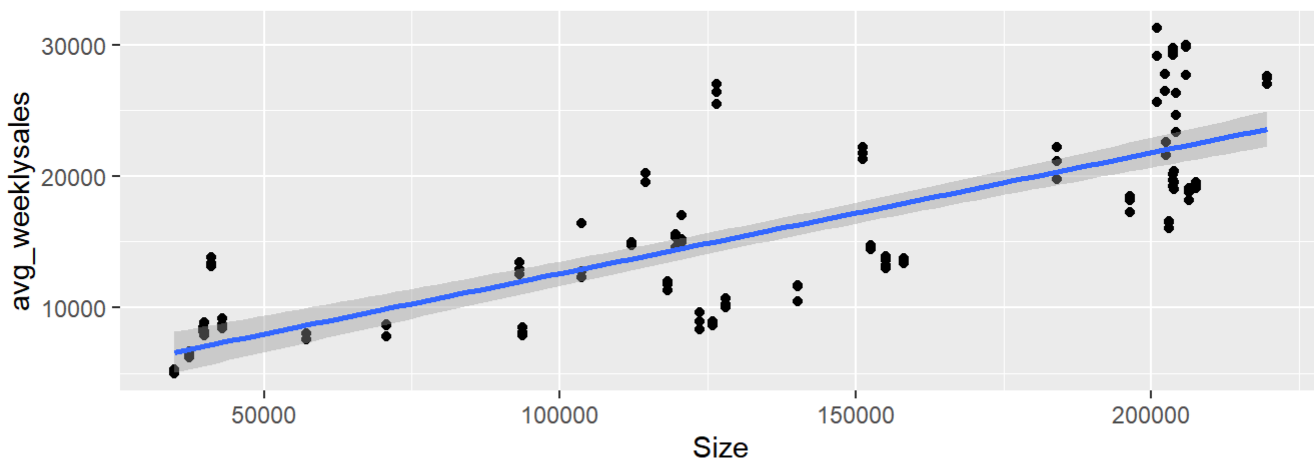
```
d1
```

```
## # A tibble: 120 x 3
## # Groups:   Size [40]
##      Size  Year avg_weeklysales
##     <dbl> <dbl>           <dbl>
##  1 200898  2010          31257.
##  2 205863  2012          29975.
##  3 205863  2011          29831.
##  4 203742  2010          29790.
##  5 203742  2011          29463.
##  6 203742  2012          29250.
##  7 200898  2011          29115.
##  8 202307  2010          27794.
##  9 205863  2010          27709.
## 10 219622  2012          27631.
## # ... with 110 more rows
```

```
# Create a bar plot for Size vs Weekly Sales
b11 <- ggplot(data = d1, aes(x = Size,y = avg_weeklysales, color = "red")) +
      geom_col() +
      ggtitle("Relationship of Size vs Weekly Sales") +
      facet_wrap(~Year)

# Create a scatter plot for Size vs Weekly Sales
s11 <- ggplot(d1, aes(x=Size, y=avg_weeklysales)) + geom_point() +
  scale_colour_hue(l=50) + # Use a slightly darker palette than normal
  geom_smooth(method=lm,   # Add linear regression lines
              se=TRUE,    # Don't add shaded confidence region
              fullrange=TRUE) # Extend regression lines

# Organized the plots in one page
grid.arrange(b11, s11, nrow=2)
```

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






```
cor(forecasting_vis$Size,forecasting_vis$Weekly_Sales)
```

```
## [1] 0.243828
```

Summary:

- bar plot: the store size at 200k in 2010 has the highest average weekly sales: 31k compared to store size at 34k in 2010 has the lowest average weekly sales: 4k
- scatter plot: this plot showcases the linearity of x-variable **store size** vs y-variable **Avg Weekly Sales**, we can see the linear line is going upward, which indicate it has **a positive linearity relationship** and it is confirmed through a correlation of **+0.24**

2.1.8 Relationship of Store vs Weekly Sales

2.1.8.1 Examine Top 10 Stores by Average Weekly Sales for each Year

```
# Calculate the average of weekly sales for each week for 2010
avg_sales_2010 <- forecasting_vis %>%
  group_by(Store) %>%
  filter(Year == "2010") %>%
  summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
  arrange(desc(avg_weeklysales)) %>%
  mutate(across(where(is.numeric), ~ round(., 2)))  # round to 2 dec. places
avg_sales_2010
```

```
## # A tibble: 45 x 2
##    Store avg_weeklysales
##    <dbl>           <dbl>
##  1    14          31257.
##  2    20          29790.
##  3     2          27794.
##  4     4          27709.
##  5    10          26984.
##  6    13          26982.
##  7    27          26320.
##  8     6          22555.
##  9     1          21283.
## 10    19          21253.
## # ... with 35 more rows
```

```
# Create a plot to display top ten highest weekly sales by stores - 2010
s_2010 <- head(avg_sales_2010, n = 10) %>%
  ggplot(aes(x = reorder(as.factor(Store),
                avg_weeklysales),
                y = avg_weeklysales,
                fill=as.factor(Store))) +
  geom_bar(stat = 'identity') +
  theme(legend.position = "none")+
  labs(y = "Total Average Sales", x = 'Stores', title = "Top Ten Stores with the Highest Weekly
 Sales - 2010") +
  coord_flip()

# Calculate the average of weekly sales for each week for 2011
avg_sales_2011 <- forecasting_vis %>%
  group_by(Year, Store) %>%
  filter(Year == "2011") %>%
  summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
  arrange(desc(avg_weeklysales)) %>%
  mutate(across(where(is.numeric), ~ round(., 2)))  # round to 2 dec. places
```

```
## `summarise()` has grouped output by 'Year'. You can override using the `.groups` argument.
```

```
avg_sales_2011
```

```
## # A tibble: 45 x 3
## # Groups:   Year [1]
##     Year Store avg_weeklysales
##    <dbl> <dbl>           <dbl>
##  1  2011     4          29831.
##  2  2011    20          29463.
##  3  2011    14          29115.
##  4  2011    13          27474.
##  5  2011     2          26444.
##  6  2011    10          26399.
##  7  2011    27          24657.
##  8  2011     1          21718.
##  9  2011     6          21607.
## 10  2011    39          21102.
## # ... with 35 more rows
```

```
# Create a plot to display top ten highest weekly sales by stores - 2011
s_2011 <- head(avg_sales_2011, n = 10) %>%
  ggplot(aes(x = reorder(as.factor(Store),
                 avg_weeklysales),
                 y = avg_weeklysales,
                 fill=as.factor(Store))) +
  geom_bar(stat = 'identity') +
  theme(legend.position = "none")+
  labs(y = "Total Average Sales", x = 'Stores', title = "Top Ten Stores with the Highest Weekly
 Sales - 2011") +
  coord_flip()


# Calculate the average of weekly sales for each week for 2010
avg_sales_2012 <- forecasting_vis %>%
  group_by(Year, Store) %>%
  filter(Year == "2012") %>%
  summarise(avg_weeklysales = mean(Weekly_Sales)) %>%
  arrange(desc(avg_weeklysales)) %>%
  mutate(across(where(is.numeric), ~ round(., 2)))  # round to 2 dec. places
```

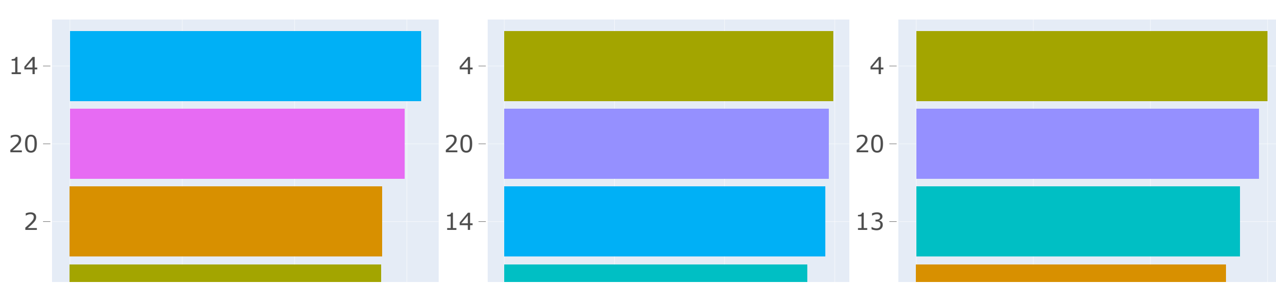
```
## `summarise()` has grouped output by 'Year'. You can override using the `.groups` argument.
```

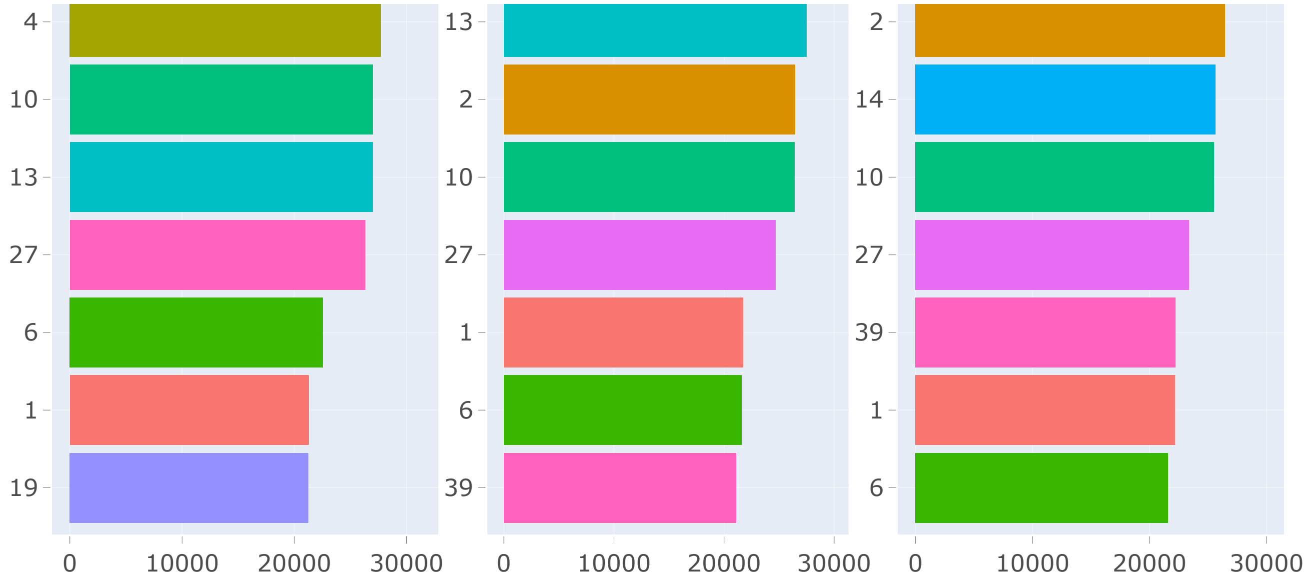
```
avg_sales_2012
```

```
## # A tibble: 45 x 3
## # Groups:   Year [1]
##     Year Store avg_weeklysales
##    <dbl> <dbl>           <dbl>
##  1  2012     4          29975.
##  2  2012    20          29250.
##  3  2012    13          27631.
##  4  2012     2          26451.
##  5  2012    14          25626.
##  6  2012    10          25507.
##  7  2012    27          23373.
##  8  2012    39          22229.
##  9  2012     1          22180.
## 10  2012     6          21573.
## # ... with 35 more rows
```

```
# Create a plot to display top ten highest weekly sales by stores - 2011
s_2012 <- head(avg_sales_2012, n = 10) %>%
  ggplot(aes(x = reorder(as.factor(Store),
                avg_weeklysales),
                y = avg_weeklysales,
                fill=as.factor(Store))) +
  geom_bar(stat = 'identity') +
  theme(legend.position = "none")+
  labs(y = "Total Average Weekly Sales", x = 'Stores', title = "Top Ten Stores with the Highest
 Average Weekly Sales - 2012") +
  coord_flip()

fig <- subplot(s_2010, s_2011, s_2012)%>%
  layout(title = list(text = "Top Ten Stores with the Highest Average Weekly Sales in 2010 - 201
2"),
         plot_bgcolor='#e5ecf6',
         xaxis = list(
           zerolinecolor = '#ffff',
           zerolinewidth = 2,
           gridcolor = 'ffff'),
         yaxis = list(
           zerolinecolor = '#ffff',
           zerolinewidth = 2,
           gridcolor = 'ffff'))
fig
```

Top Ten Stores with the Highest Average Weekly Sales in 2010 - 2012





Summary:

- the number one store in the top ten stores rank **#14** in 2010 is, **#4** in 20111-2012, as compared to the bottom one in the top ten stores rank is **#19** in 2010, **#39** in 2011, and **#6** in 2012
- surprisingly, **#19 store drops in the top ten stores rank** in 2011 to 2012, and **#39 climbed up the top ten stores rank**

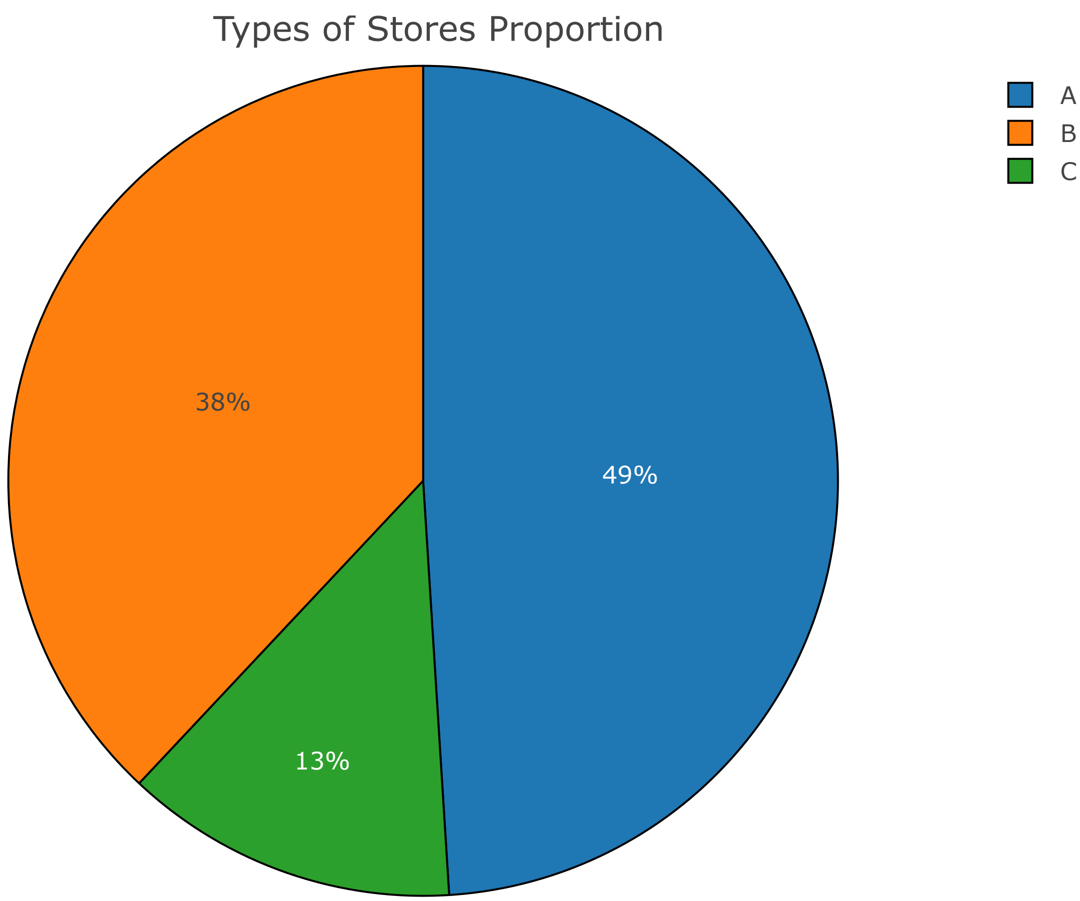
2.2 Categorical Variables Visualization

2.2.1 Composition Between Type vs Weekly Sales

```
# Group and calc. the percentage of the number of proportion in types A to C
stores_pct <- stores %>%
  group_by(Type) %>%
  summarize(counts = n(),
            percentage = n()/nrow(stores)) %>%
  mutate(across(where(is.numeric), ~ round(., 2))) # round to 2 dec. places
stores_pct
```

```
## # A tibble: 3 x 3
##   Type  counts percentage
##   <chr>  <dbl>      <dbl>
## 1 A         22       0.49
## 2 B         17       0.38
## 3 C          6       0.13
```

```
# Create a pie chart with plotly
store_pie <- plot_ly(data = stores_pct, labels = ~Type, values = ~percentage,
             type = 'pie',
             sort= FALSE,
             marker= list(colors = colors, line = list(color="black", width = 1))) %>%
            layout(title = "Types of Stores Proportion ")

store_pie
```



Summary:

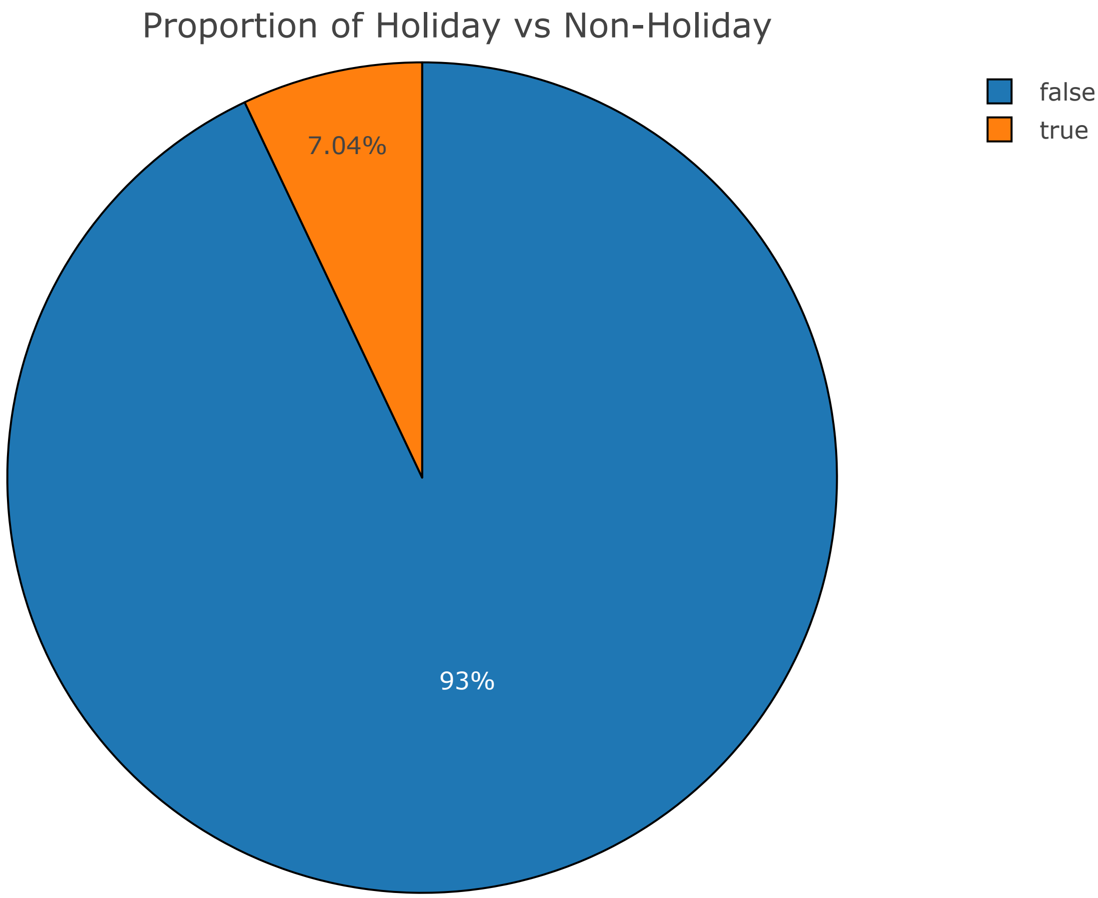
- **Type A** store contributes **most** weekly sales, followed by Type B, and Type C being the last

2.2.2 Composition Between IsHoliday vs Weekly Sales

```
# Group and calc. the percentage of the number of proportion in types A to C
isholiday_pct <- train %>%
  group_by(IsHoliday) %>%
  summarize(counts = n())
isholiday_pct
```

```
## # A tibble: 2 x 2
##   IsHoliday counts
##   <lgl>      <int>
## 1 FALSE     391909
## 2 TRUE       29661
```

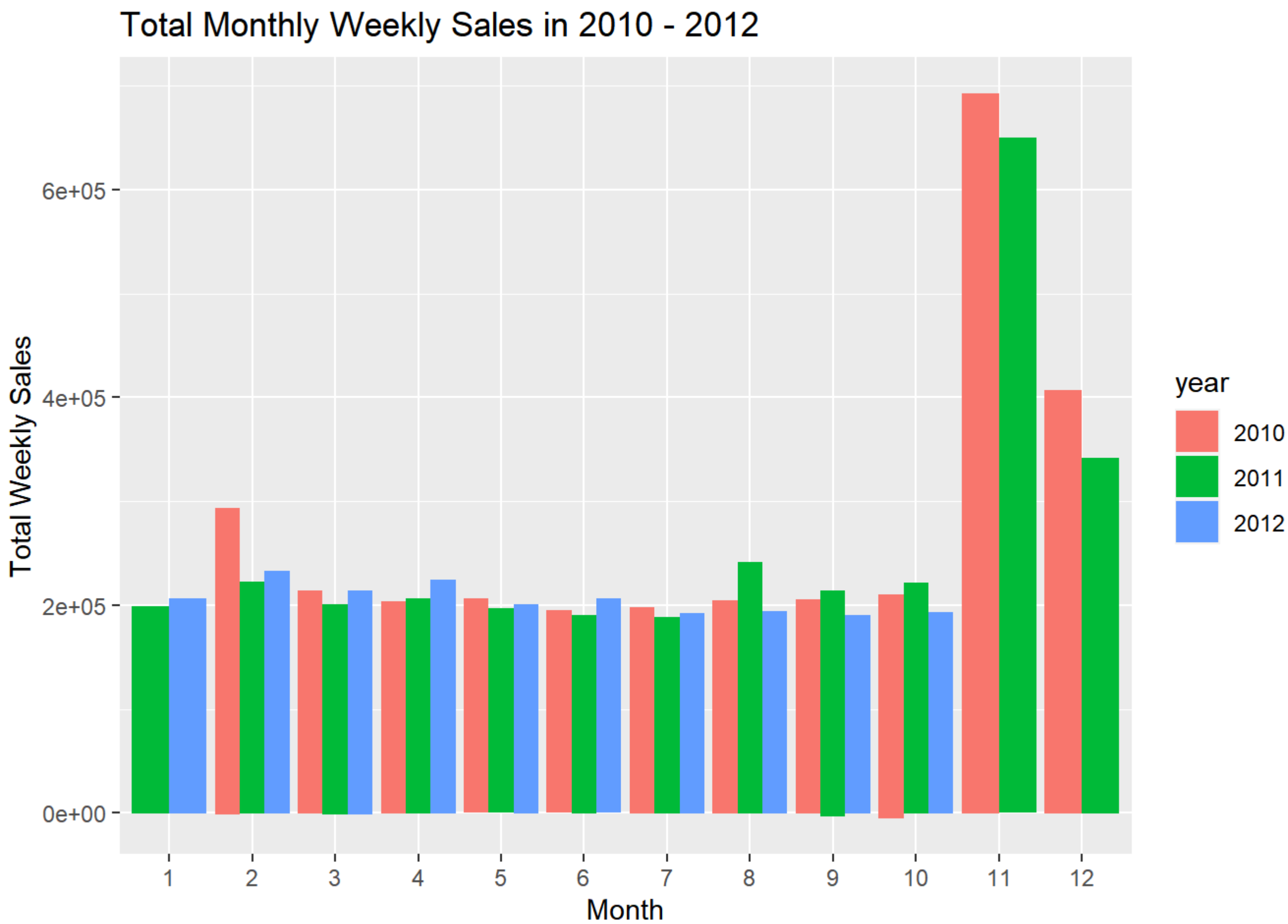
```
# Create a pie chart with plotly
isholiday_pie <- plot_ly(data = isholiday_pct, labels = ~IsHoliday, values = ~counts,
                          type = 'pie',
                          sort= FALSE,
                          marker= list(colors = colors,
                                       line = list(color="black", width = 1))) %>%
                          layout(title = "Proportion of Holiday vs Non-Holiday")

isholiday_pie
```



2.3 Times Series Visualization

2.3.1 Total Monthly Weekly Sales for 2010-2012

```
# Create a bar graph of Yearly & Monthly Total Weekly Sales for 2010 to 2012
df1 <- forecasting_vis %>%
  mutate(month = factor(Month)) %>%
  mutate(year = factor(Year)) %>%
  mutate(wkly_sales = Weekly_Sales)

p1 <- ggplot(df1, aes(x = month, y = wkly_sales, fill = year)) +
              geom_col(position = "dodge") +
              labs(x = "Month", y = "Total Weekly Sales", title = "Total Monthly Weekly Sales in 
2010 - 2012")
p1
```



Summary:

- in **November** for 2010 to 2011, it has the highest weekly sales, followed by December

2.4 Correlation Matrix

```
# Drop gender & geography as factors cant be calculated to get its correlation
corr_matrix <- forecasting_vis %>%
  dplyr:: select(where(is.numeric)) %>%
  dplyr::select(-c("MarkDown1", "MarkDown2", "MarkDown3", "MarkDown4", "MarkDown5"))

# Create a correlation matrix
corr <- round(cor(corr_matrix), 2)
corrplot(corr, method = "color", cl.pos = 'n', rect.col = "black",  tl.col = "indianred4", addCo
ef.col = "black", number.digits = 2, number.cex = 0.60, tl.cex = 0.7, cl.cex = 1, col = colorRam
pPalette(c("green4","white","red"))(100))
```

Summary:

- **dept, size, month, and week** all have a perfect positive relationship with the response variable - Weekly Sales
- **unemployment, store, CPI** all have a perfect negative relationship with the response variable - Weekly Sales
- Temperature, fuel price has no relationship with the response variable - Weekly Sales

3. Data Wrangling for Model Building

```
# Create a new train df for modeling
# train_forecast <- train_df
#
# # Create a new test df for modeling
# test_forecast <- test_df
```

3.1 Examine Numbers of Unique Observations

```
# Check to see how many unique observations there are in train data set & test data set
length(unique(train_forecast$Store_Dept))
```

```
## [1] 3331
```

```
length(unique(test_forecast$Store_Dept))
```

```
## [1] 3169
```

```
# Filter out the 11 observations that are in the test data set
# train_df <- filter(train_df, storeDept %in% unique(test_df$storeDept))
train_obs <- filter(train_forecast, Store_Dept %in% unique(test_forecast$Store_Dept))

# Subtract out the 11 more observations found in the test data set
# length(unique(test_forecast$Store_Dept)) - length(unique(train_forecast$Store_Dept))
length(unique(test_forecast$Store_Dept)) - length(unique(train_forecast$Store_Dept))
```

```
## [1] -162
```

```
# Find the 11 numbers of Dept_Store that are not found in the train data set
eleven_dept_store_nums <-
  test_forecast %>%
  filter(!Store_Dept %in% unique(train_obs$Store_Dept)) %>%
  .$Store_Dept %>%
  unique()
eleven_dept_store_nums
```

```
##  [1] "5_99"  "9_99"  "10_99" "18_43" "24_43" "25_99" "34_39" "36_30" "37_29"
## [10] "42_30" "45_39"
```

Summary:

- there are 11 more observations per each unique ID in the train data set vs test data set
- in other words, these are the unique IDs that are not present in the test data set: 5_99, 9_99, 10_99, 18_43, 24_43, 25_99, 34_39, 36_30, 37_29, 42_30, and 45_39

3.2 Irregular Time Series

```
# Check if the data has irregular time series (missing gaps between observations)
# Add 1 because the first week is not accounted for in the difference

# Check to see the beginning & ending of dates in the train data set
begin_train <- min(train_forecast$Date)
end_train <- max(train_forecast$Date)

# Check to see the beginning & ending of dates in the test data set
begin_test <- min(test_forecast$Date)
end_test <- max(test_forecast$Date)

# Calculate the #s of weeks between the ending & beginning dates for train & test data sets
nums_wks_train <- difftime(end_train, begin_train, units = "weeks") + 1
nums_wks_train
```

```
## Time difference of 143 weeks
```

```
nums_wks_test <- difftime(end_test, begin_test, units = "weeks") + 1
nums_wks_test
```

```
## Time difference of 39 weeks
```

```
# Retrieve numbers of observations in each store_dept in the train data set
nums_obs_dept_store <-
  train_forecast %>%
  count(Store_Dept) %>%
  arrange(n) %>%
  rename(Num_Obs = n)

unique(nums_obs_dept_store$Num_Obs)
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  48  49  50  51  52  53  54  56
##  [55]  57  58  59  60  61  62  64  65  66  67  68  69  70  71  74  75  76  77
##  [73]  78  79  81  82  83  84  85  86  87  88  89  90  91  92  93  95  96  97
##  [91]  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115
## [109] 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133
## [127] 134 135 136 137 138 139 140 141 142 143
```

```
# Identify the dept_store that has 143 obs
numObs_vs_weeklySales <- train_forecast %>%
  merge(nums_obs_dept_store, by = "Store_Dept") %>%
  dplyr::select(Date, Store_Dept, Weekly_Sales, Num_Obs)
```

Summary:

- there are any missing gaps among the train & test data sets

    - the train data set ranges from 2010-02-05 to 2012-10-26; while in the test data set, it ranges from 2012-11-02 to 2013-07-26
    - there are 143 weeks and only 39 weeks in the test data set
- there are 143 observations for each unique ID

3.3 Convert Irregular Time Series to Regular Time Series

```
# Filter only holiday weeks
holiday_wks <-
  train_forecast %>%
  filter(IsHoliday == TRUE) %>%
  .$Date %>%
  unique()
holiday_wks
```

```
##  [1] "2010-02-12" "2010-09-10" "2010-11-26" "2010-12-31" "2011-02-11"
##  [6] "2011-09-09" "2011-11-25" "2011-12-30" "2012-02-10" "2012-09-07"
```

```
# Calculate the weeks before the identified holiday weeks
before_holiday_wks <- holiday_wks - 7
before_holiday_wks
```

```
##  [1] "2010-02-05" "2010-09-03" "2010-11-19" "2010-12-24" "2011-02-04"
##  [6] "2011-09-02" "2011-11-18" "2011-12-23" "2012-02-03" "2012-08-31"
```

```
# Convert irregular ts to regular ts
# Create a tibble and then add in the missing gaps of the ts through outer joining with the date
s of the 143 wks
train_dates <- tibble("Date" = seq(begin_train, end_train, by = 7))

convert_regular_ts <- function(data){
  Store_Dept <- unique(data$Store_Dept)
  Store <- unique(data$Store)
  Dept <- unique(data$Dept)
  merge(data, train_dates, by = "Date", all = T) %>%
  replace_na(list(Store_Dept = Store_Dept,
                  Store = Store,
                  Dept = Dept #,
                 ))
}

store_dept_df <-
  train_forecast %>%
  dplyr::select(Store_Dept, Store, Dept, Date, Weekly_Sales) %>%
  group_by(Store_Dept) %>%
  do(convert_regular_ts(.)) %>%
  ungroup() %>%
  arrange(Store, Dept)

head(store_dept_df)
```

```
## # A tibble: 6 x 5
##   Date       Store_Dept Store  Dept Weekly_Sales
##   <date>     <chr>      <dbl> <dbl>        <dbl>
## 1 2010-02-05 1_1            1     1       24924.
## 2 2010-02-12 1_1            1     1       46039.
## 3 2010-02-19 1_1            1     1       41596.
## 4 2010-02-26 1_1            1     1       19404.
## 5 2010-03-05 1_1            1     1       21828.
## 6 2010-03-12 1_1            1     1       21043.
```

Summary:

- these are the dates of holidays: “2010-02-12”, “2010-09-10”, “2010-11-26” , “2010-12-31”, “2011-02-11” , “2011-09-09” , “2011-11-25”

I also calculated the weeks before each holiday date then, I created a tibble to add in the missing gaps of the time series by joining the dates of the 143 weeks

3.4 Convert into Multiple Time Series Object

```
# Convert into multiple time series object and use pivot_wider to spread the mts into separate c
olumns
store_dept_mts<-
  store_dept_df %>%
  dplyr::select(-Store, -Dept) %>%
  pivot_wider(names_from = Store_Dept, values_from = Weekly_Sales) %>%
  dplyr::select(-Date) %>%
  ts(start = decimal_date(begin_train), frequency = 52)

store_dept_mts[, 1]
```

```
## Time Series:
## Start = 2010.09589041096
## End = 2012.82665964173
## Frequency = 52
##   [1] 24924.50 46039.49 41595.55 19403.54 21827.90 21043.39 22136.64 26229.21
##   [9] 57258.43 42960.91 17596.96 16145.35 16555.11 17413.94 18926.74 14773.04
##  [17] 15580.43 17558.09 16637.62 16216.27 16328.72 16333.14 17688.76 17150.84
##  [25] 15360.45 15381.82 17508.41 15536.40 15740.13 15793.87 16241.78 18194.74
##  [33] 19354.23 18122.52 20094.19 23388.03 26978.34 25543.04 38640.93 34238.88
##  [41] 19549.39 19552.84 18820.29 22517.56 31497.65 44912.86 55931.23 19124.58
##  [49] 15984.24 17359.70 17341.47 18461.18 21665.76 37887.17 46845.87 19363.83
##  [57] 20327.61 21280.40 20334.23 20881.10 20398.09 23873.79 28762.37 50510.31
##  [65] 41512.39 20138.19 17235.15 15136.78 15741.60 16434.15 15883.52 14978.09
##  [73] 15682.81 15363.50 16148.87 15654.85 15766.60 15922.41 15295.55 14539.79
##  [81] 14689.24 14537.37 15277.27 17746.68 18535.48 17859.30 18337.68 20797.58
##  [89] 23077.55 23351.80 31579.90 39886.06 18689.54 19050.66 20911.25 25293.49
##  [97] 33305.92 45773.03 46788.75 23350.88 16567.69 16894.40 18365.10 18378.16
## [105] 23510.49 36988.49 54060.10 20124.22 20113.03 21140.07 22366.88 22107.70
## [113] 28952.86 57592.12 34684.21 16976.19 16347.60 17147.44 18164.20 18517.79
## [121] 16963.55 16065.49 17666.00 17558.82 16633.41 15722.82 17823.37 16566.18
## [129] 16348.06 15731.18 16628.31 16119.92 17330.70 16286.40 16680.24 18322.37
## [137] 19616.22 19251.50 18947.81 21904.47 22764.01 24185.27 27390.81
```

Summary:

- since the train data set is multivariate, we need to convert them into multivariate time series objects before we perform various time series forecasting methods

3.5 Interpolation

```r
# Perform interpolation to fill in missing values
impute <- function(current_ts){
 if(sum(!is.na(ts)) >= 3){
    na_seadec(current_ts)
 } else if(sum(!is.na(ts)) == 2){
   na_interpolation(current_ts)
 } else{
   na_locf(current_ts)
 }
}
for(i in 1:ncol(store_dept_mts)){
  store_dept_mts[, i] <- impute(store_dept_mts[, i])
}

sum(is.na(store_dept_mts))
```

```
## [1] 0
```

Summary:

- according to investopedia.com, “Interpolation is achieved by using other established values that are located in sequence with the unknown value,” so I created a interpolation function that which will be used later to generate forecasts for each forecasting method that I will be using

3.6 Split Data into Train and Validate

```r
# Determine which are holiday vs non-holiday
holiday_non_holiday <- train_forecast %>%
  dplyr::select(Date, IsHoliday) %>%
  unique() %>%
  .$IsHoliday

# Count the #s of rows in the mts train
total_data <- nrow(store_dept_mts)

# Split the data into 80% to train & 20% to validate
train_set <- round(0.80 * total_data)
validate_set <- total_data - train_set


validate_weights <- holiday_non_holiday[(total_data - validate_set + 1):total_data]
train_data <- store_dept_mts %>% subset(end = train_set)
validate_data <- store_dept_mts %>% subset(start = train_set + 1)
```

Summary:

- split the train data set into 80% into train and 20% into validate data set
- the validate data set will be used to calculate the WMAE

NOTE: the test data set will be untouched until I arrive at a final model based on the lowest WMAE score

3.7 WMAE Function

```
# Define a function to calculate the WMAE
WMAE <- function(fc){
  # rep() to replicate weights for each storeDept
  weights <- as.vector(rep(validate_weights, ncol(fc)))

  # as.vector() collapse all columns into one
  MetricsWeighted::mae(as.vector(validate_data), as.vector(fc), weights)
}
```

Summary:

- WMAE stands for weighted mean absolute error
- the lower the WMAE is, the better the model is at lowering the average of absolute errors for the prediction of observation vs true value of observation as “a measurement of the magnitude of errors for the entire group” (clarity.io)

NOTE: This is also the metric that Walmart specifically points out in the guideline to use.

3.8 Forecast Function

```
# Define a function to generates forecast for each time series forecasting method
model_forecasts <- function(train_data, h, model, ...){

  tic()

  # Initialize forecasts with zeroes
  full_forecast <- matrix(0, h, ncol(train_data))

  # Iterate through all storeDept to perform forecasting
  for(i in 1:ncol(train_data)){
    current_ts <- train_data[, i]
    fc <- model(current_ts, h, ...)
    full_forecast[, i] <- fc
  }

  toc()

  # Return forecasts
  full_forecast
}
```

Summary:

- created a forecast function to generates forecasts that will be use later for each time series forecast method

4. Time Series Forecasting

4.1 Plot Function

```
# change index for different storeDept
basic_ts <- store_dept_mts[, 111]
basic_train_mod <- basic_ts %>% subset(end = 107)


# Create a plot function to autoplot each time series forecasting method
forecast_plots <- function(ref, fc_list, model_names){
  plt <- autoplot(ref)
  for(i in 1:length(fc_list)){
    plt <- plt + autolayer(fc_list[[i]], series = model_names[i], PI = F)
  }
  plt <- plt +
    ylab("Weekly_Sales") +
    guides(color = guide_legend(title = "Time Series Forecasting Method:"))
  plt
}
```

Summary:

- created a plot function to perform “autoplot” for each ts forecast method

4.2 Simple Exponential Smoothing Forecast

```
# Using the forecast function to generate weekly sales projection
simple_es <- function(current_ts, h){
  ses(current_ts, h = h)$mean
}
# Produce the summary of the simple ES
simple_es_mod <- ses(basic_train_mod, h = 36)
summary(simple_es_mod)
```

```
##
## Forecast method: Simple exponential smoothing
##
## Model Information:
## Simple exponential smoothing
##
## Call:
##  ses(y = basic_train_mod, h = 36)
##
##   Smoothing parameters:
##     alpha = 0.5233
##
##   Initial states:
##     l = 3722.5512
##
##   sigma:  757.2014
##
##      AIC     AICc      BIC
## 1922.714 1922.947 1930.733
##
## Error measures:
##                    ME     RMSE      MAE       MPE     MAPE      MASE       ACF1
## Training set 13.64275 750.0913 559.8444 -2.596425 15.21913 0.8879636 0.08546371
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80      Lo 95    Hi 95
## 2012.154       4486.434 3516.041 5456.827 3002.34662 5970.521
## 2012.173       4486.434 3391.209 5581.659 2811.43202 6161.436
## 2012.192       4486.434 3279.217 5693.651 2640.15452 6332.714
## 2012.211       4486.434 3176.766 5796.102 2483.47015 6489.398
## 2012.231       4486.434 3081.768 5891.100 2338.18343 6634.685
## 2012.250       4486.434 2992.800 5980.068 2202.11861 6770.749
## 2012.269       4486.434 2908.842 6064.026 2073.71498 6899.153
## 2012.288       4486.434 2829.131 6143.737 1951.80792 7021.060
## 2012.307       4486.434 2753.082 6219.786 1835.50102 7137.367
## 2012.327       4486.434 2680.232 6292.636 1724.08682 7248.781
## 2012.346       4486.434 2610.209 6362.659 1616.99534 7355.873
## 2012.365       4486.434 2542.707 6430.162 1513.75936 7459.109
## 2012.384       4486.434 2477.471 6495.397 1413.99021 7558.878
## 2012.404       4486.434 2414.288 6558.580 1317.36046 7655.508
## 2012.423       4486.434 2352.976 6619.892 1223.59116 7749.277
## 2012.442       4486.434 2293.377 6679.491 1132.44240 7840.426
## 2012.461       4486.434 2235.355 6737.513 1043.70602 7929.162
## 2012.481       4486.434 2178.792 6794.076  957.20005 8015.668
## 2012.500       4486.434 2123.582 6849.286  872.76432 8100.104
## 2012.519       4486.434 2069.634 6903.234  790.25694 8182.611
## 2012.538       4486.434 2016.863 6956.005  709.55154 8263.317
## 2012.557       4486.434 1965.197 7007.671  630.53496 8342.333
## 2012.577       4486.434 1914.569 7058.299  553.10542 8419.763
## 2012.596       4486.434 1864.918 7107.950  477.17097 8495.697
## 2012.615       4486.434 1816.190 7156.678  402.64822 8570.220
## 2012.634       4486.434 1768.336 7204.532  329.46123 8643.407
```

```
## 2012.654       4486.434 1721.309 7251.559  257.54066 8715.327
## 2012.673       4486.434 1675.069 7297.799  186.82295 8786.045
## 2012.692       4486.434 1629.578 7343.290  117.24970 8855.618
## 2012.711       4486.434 1584.800 7388.068   48.76707 8924.101
## 2012.731       4486.434 1540.702 7432.166  -18.67466 8991.543
## 2012.750       4486.434 1497.254 7475.614  -85.12157 9057.990
## 2012.769       4486.434 1454.430 7518.438 -150.61643 9123.484
## 2012.788       4486.434 1412.201 7560.667 -215.19901 9188.067
## 2012.807       4486.434 1370.545 7602.323 -278.90641 9251.774
## 2012.827       4486.434 1329.439 7643.429 -341.77328 9314.641
```

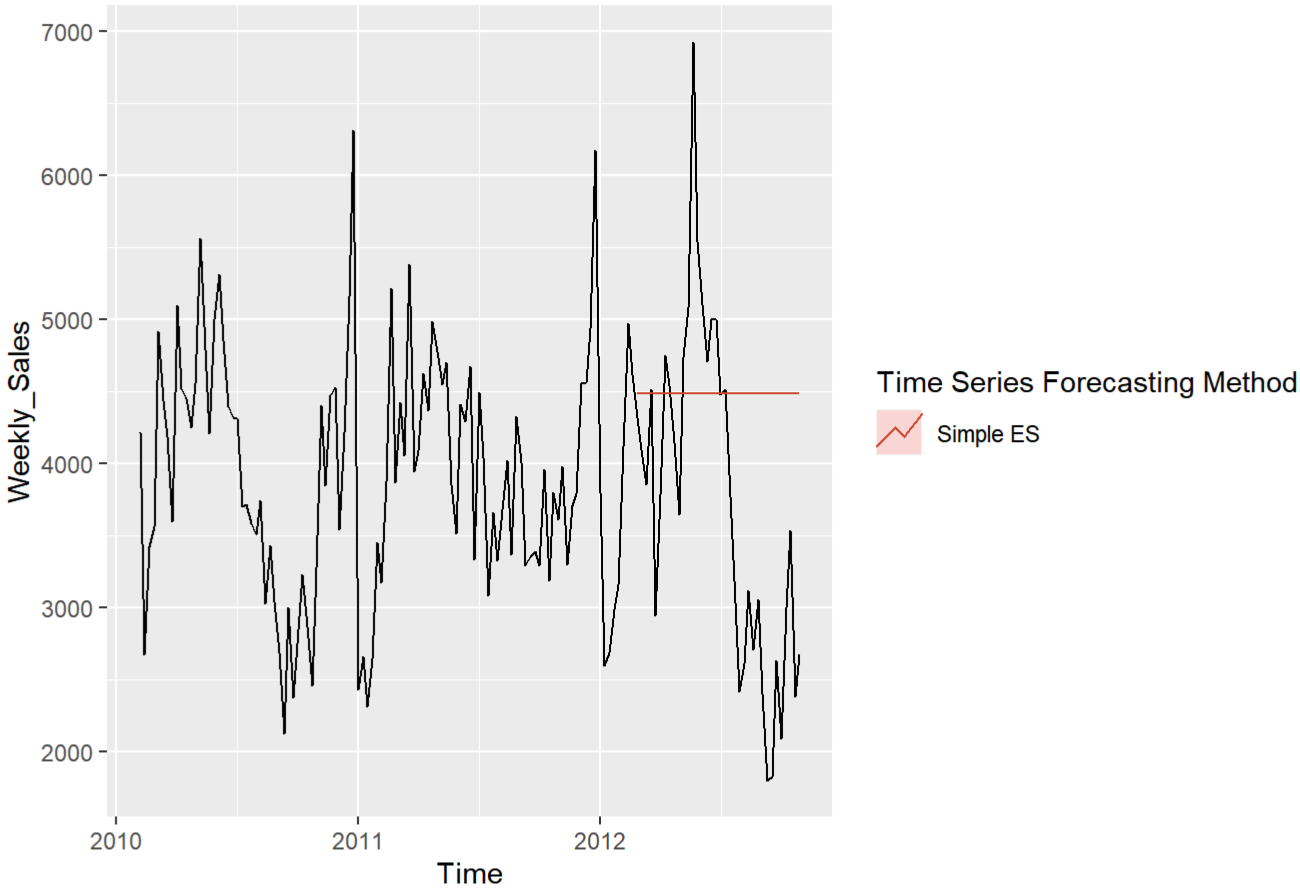
```
# Make prediction on the validate data for simple ES
simple_es_pred <- model_forecasts(train_data, validate_set, simple_es)
```

```
## 15.53 sec elapsed
```

```
# Calculate the WMAE on the simple ES validate data
WMAE(simple_es_pred)
```

```
## [1] 3114.852
```

```
# Create a plot to display simple ES
forecast_plots(basic_ts,
               list(
                    simple_es_mod
                    ),
               c(
                 "Simple ES"
                )
)
```



Summary:

- simple exponential smoothing is a method used for an univariate data without a trend or seasonality
- since our data is multivariate, without a surprise, the WMAE is: 3470.621 which is quite high simple ES also predicted in the next 36 months, the weekly sales will be constant which doesn't look right

4.3 Holt's Trend Forecast

```
# Using the forecast function to generate weekly sales projection
holt_trend <- function(current_ts, h){
  holt(current_ts, h = h)$mean
}

# Produce the summary of the Holt's Trend
holt_trend_mod <- holt(basic_train_mod, h = 36, seasonal = "multiplicative")
summary(holt_trend_mod)
```

```
##
## Forecast method: Holt's method
##
## Model Information:
## Holt's method
##
## Call:
##  holt(y = basic_train_mod, h = 36, seasonal = "multiplicative")
##
##   Smoothing parameters:
##     alpha = 0.5228
##     beta  = 1e-04
##
##   Initial states:
##     l = 3664.0851
##     b = 7.5552
##
##   sigma:  764.4764
##
##      AIC     AICc      BIC
## 1926.703 1927.297 1940.067
##
## Error measures:
##                     ME    RMSE      MAE       MPE     MAPE      MASE       ACF1
## Training set 0.4167589 750.051 560.0883 -2.949682 15.26321 0.8883504 0.08485551
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80       Lo 95    Hi 95
## 2012.154       4500.210 3520.494 5479.926 3001.863749 5998.556
## 2012.173       4507.770 3402.215 5613.325 2816.969365 6198.570
## 2012.192       4515.329 3296.822 5733.837 2651.783078 6378.876
## 2012.211       4522.889 3201.008 5844.770 2501.246478 6544.531
## 2012.231       4530.449 3112.676 5948.221 2362.152740 6698.744
## 2012.250       4538.008 3030.398 6045.619 2232.317350 6843.699
## 2012.269       4545.568 2953.149 6137.987 2110.172841 6980.963
## 2012.288       4553.128 2880.163 6226.092 1994.549296 7111.706
## 2012.307       4560.687 2810.853 6310.522 1884.546071 7236.828
## 2012.327       4568.247 2744.752 6391.742 1779.452287 7357.041
## 2012.346       4575.807 2681.487 6470.126 1678.695171 7472.918
## 2012.365       4583.366 2620.751 6545.981 1581.805192 7584.927
## 2012.384       4590.926 2562.288 6619.564 1488.391744 7693.460
## 2012.404       4598.485 2505.883 6691.088 1398.125744 7798.845
## 2012.423       4606.045 2451.352 6760.738 1310.726866 7901.363
## 2012.442       4613.605 2398.539 6828.671 1225.953991 8001.255
## 2012.461       4621.164 2347.306 6895.023 1143.597939 8098.731
## 2012.481       4628.724 2297.534 6959.915 1063.475841 8193.972
## 2012.500       4636.284 2249.117 7023.451  985.426731 8287.141
## 2012.519       4643.843 2201.962 7085.725  909.308039 8378.379
## 2012.538       4651.403 2155.986 7146.819  834.992778 8467.813
## 2012.557       4658.963 2111.116 7206.809  762.367262 8555.558
## 2012.577       4666.522 2067.283 7265.761  691.329228 8641.715
## 2012.596       4674.082 2024.428 7323.736  621.786300 8726.378
```

```
## 2012.615       4681.642 1982.496 7380.787  553.654693 8809.628
## 2012.634       4689.201 1941.437 7436.966  486.858141 8891.544
## 2012.654       4696.761 1901.205 7492.317  421.326983 8972.195
## 2012.673       4704.321 1861.759 7546.882  356.997388 9051.644
## 2012.692       4711.880 1823.060 7600.701  293.810696 9129.950
## 2012.711       4719.440 1785.073 7653.807  231.712851 9207.167
## 2012.731       4726.999 1747.765 7706.234  170.653908 9283.345
## 2012.750       4734.559 1711.107 7758.012  110.587612 9358.531
## 2012.769       4742.119 1675.069 7809.168   51.471024 9432.767
## 2012.788       4749.678 1639.626 7859.731   -6.735802 9506.093
## 2012.807       4757.238 1604.754 7909.722  -64.070098 9578.546
## 2012.827       4764.798 1570.430 7959.166 -120.566641 9650.162
```

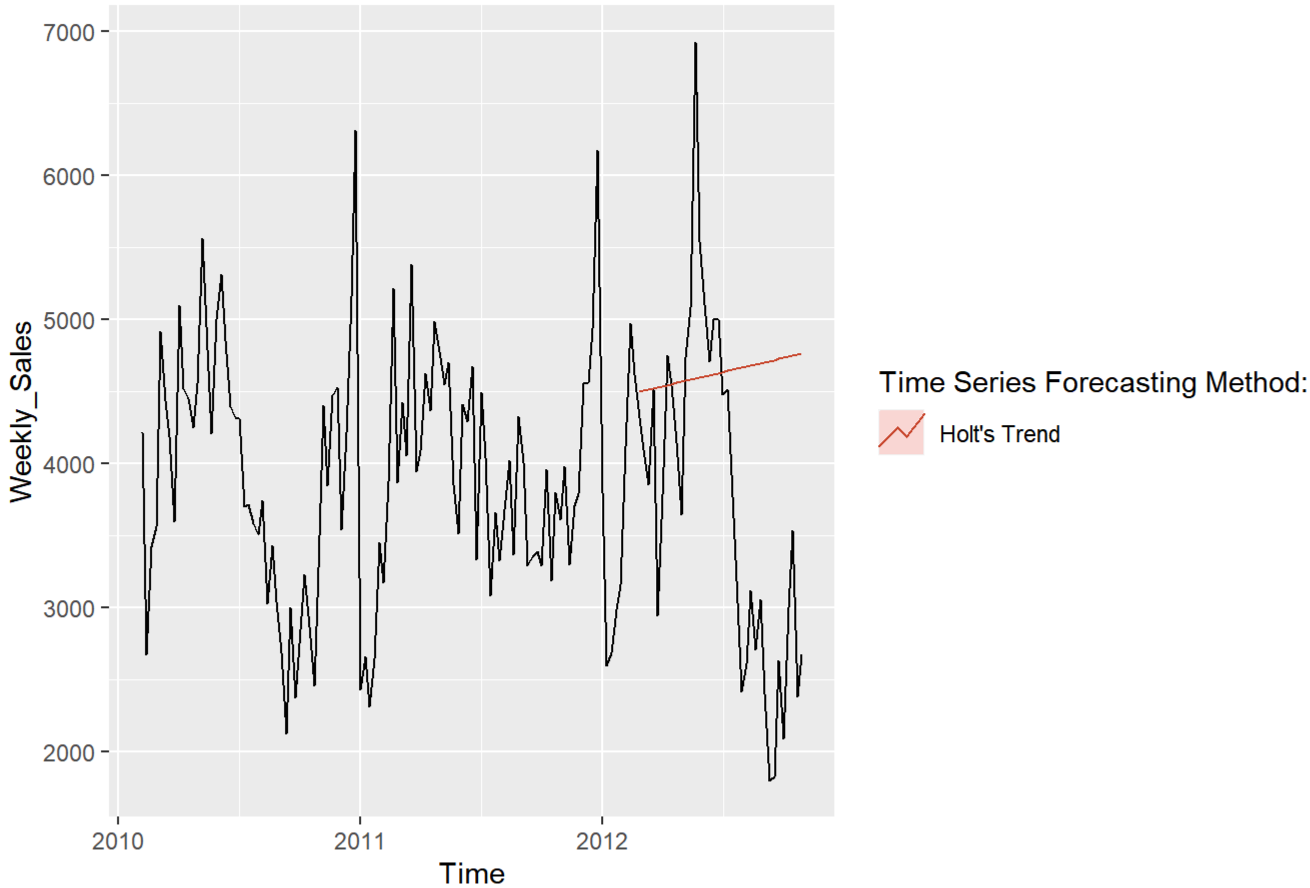
```
# Make prediction on the validate data for Holt's Trend
holt_trend_pred <- model_forecasts(train_data, validate_set, holt_trend)
```

```
## 22.92 sec elapsed
```

```
# Calculate the WMAE on the Holt's Trend validate data
WMAE(holt_trend_pred)
```

```
## [1] 4913.859
```

```
# Create a plot to display Holt's Trend
forecast_plots(basic_ts,
               list(
                    holt_trend_mod
                    ),
               c(
                 "Holt's Trend"
                )
)
```



Summary:

- holt's trend is also known as the linear exponential smoothing, which is used to forecast data with trend
- the WAME is 4914.25, which is the highest among all the other time series forecasting methods that I had used in this project
- compared to simple ES, holt's trend predicted the weekly sales will drop

4.4 Seasonal Naive Forecast

```
# Using the forecast function to generate weekly sales projection
seasonal_snaive <- function(current_ts, h){
  snaive(current_ts, h = h)$mean
}
# Produce the summary of the seasonal naive
snaive_mod <- snaive(basic_train_mod, 36)
summary(snaive_mod)
```

```
##
## Forecast method: Seasonal naive method
##
## Model Information:
## Call: snaive(y = basic_train_mod, h = 36)
##
## Residual sd: 770.7283
##
## Error measures:
##                    ME     RMSE      MAE      MPE    MAPE MASE     ACF1
## Training set 153.2638 770.7283 630.4813 3.367486 15.9803    1 0.227437
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80   Lo 95   Hi 95
## 2012.154        3868.82 2881.092 4856.548 2358.22 5379.42
## 2012.173        4425.76 3438.032 5413.488 2915.16 5936.36
## 2012.192        4054.91 3067.182 5042.638 2544.31 5565.51
## 2012.211        5383.70 4395.972 6371.428 3873.10 6894.30
## 2012.231        3946.91 2959.182 4934.638 2436.31 5457.51
## 2012.250        4092.91 3105.182 5080.638 2582.31 5603.51
## 2012.269        4625.38 3637.652 5613.108 3114.78 6135.98
## 2012.288        4371.79 3384.062 5359.518 2861.19 5882.39
## 2012.307        4984.82 3997.092 5972.548 3474.22 6495.42
## 2012.327        4780.88 3793.152 5768.608 3270.28 6291.48
## 2012.346        4549.64 3561.912 5537.368 3039.04 6060.24
## 2012.365        4703.73 3716.002 5691.458 3193.13 6214.33
## 2012.384        3854.91 2867.182 4842.638 2344.31 5365.51
## 2012.404        3515.91 2528.182 4503.638 2005.31 5026.51
## 2012.423        4412.82 3425.092 5400.548 2902.22 5923.42
## 2012.442        4293.91 3306.182 5281.638 2783.31 5804.51
## 2012.461        4675.58 3687.852 5663.308 3164.98 6186.18
## 2012.481        3332.88 2345.152 4320.608 1822.28 4843.48
## 2012.500        4494.38 3506.652 5482.108 2983.78 6004.98
## 2012.519        4041.10 3053.372 5028.828 2530.50 5551.70
## 2012.538        3082.50 2094.772 4070.228 1571.90 4593.10
## 2012.557        3656.93 2669.202 4644.658 2146.33 5167.53
## 2012.577        3327.76 2340.032 4315.488 1817.16 4838.36
## 2012.596        3684.17 2696.442 4671.898 2173.57 5194.77
## 2012.615        4019.59 3031.862 5007.318 2508.99 5530.19
## 2012.634        3368.93 2381.202 4356.658 1858.33 4879.53
## 2012.654        4324.61 3336.882 5312.338 2814.01 5835.21
## 2012.673        4007.97 3020.242 4995.698 2497.37 5518.57
## 2012.692        3289.04 2301.312 4276.768 1778.44 4799.64
## 2012.711        3355.03 2367.302 4342.758 1844.43 4865.63
## 2012.731        3391.13 2403.402 4378.858 1880.53 4901.73
## 2012.750        3294.28 2306.552 4282.008 1783.68 4804.88
## 2012.769        3958.71 2970.982 4946.438 2448.11 5469.31
## 2012.788        3185.88 2198.152 4173.608 1675.28 4696.48
## 2012.807        3797.81 2810.082 4785.538 2287.21 5308.41
## 2012.827        3611.89 2624.162 4599.618 2101.29 5122.49
```

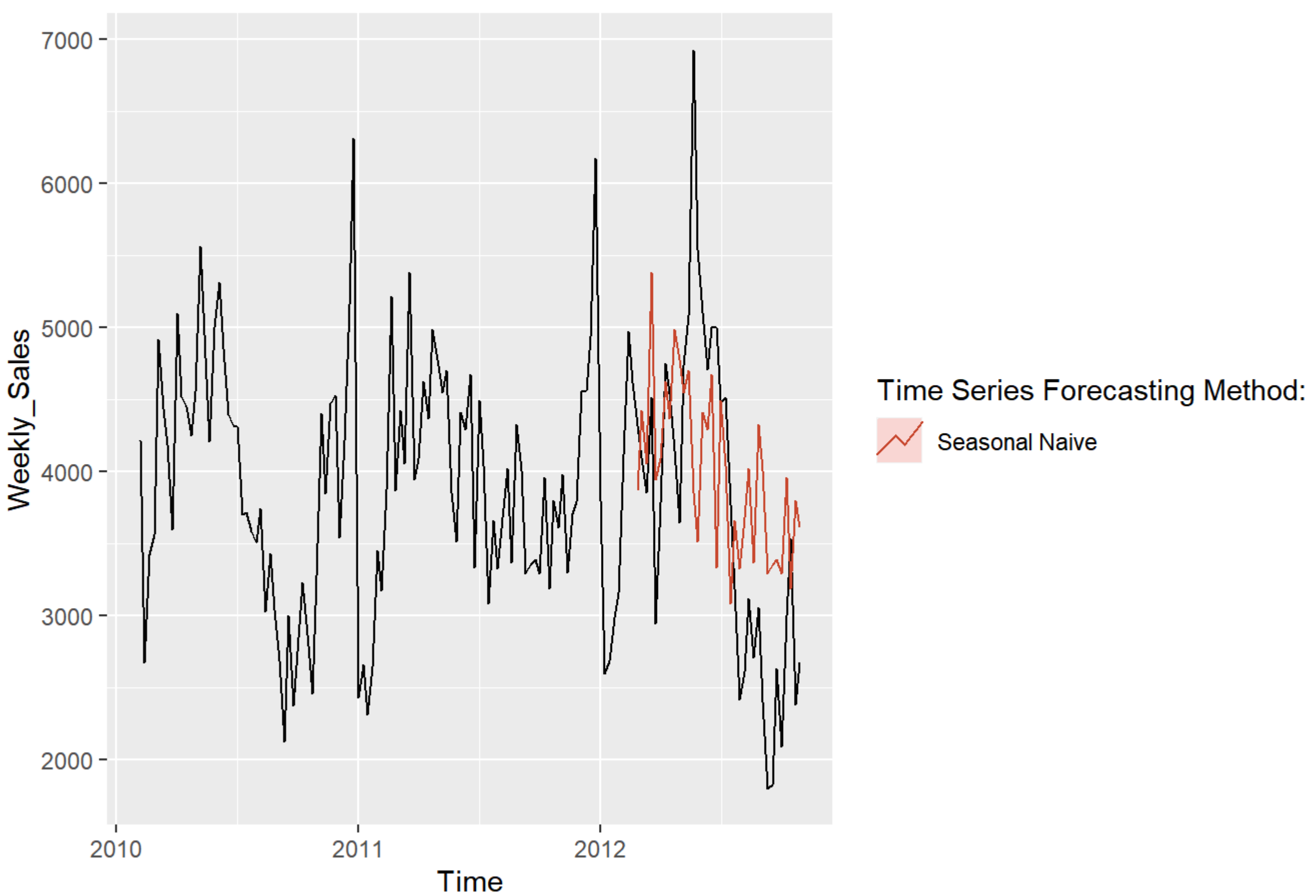
```
# Make prediction on the validate data for SNAIVE
snaive_pred <- model_forecasts(train_data, validate_set, seasonal_snaive)
```

```
## 6.52 sec elapsed
```

```
# Calculate the WMAE on the SNAIVE validate data
WMAE(snaive_pred)
```

```
## [1] 1603.568
```

```
# Create a plot to display snaive
forecast_plots(basic_ts,
               list(
                    snaive_mod
                    ),
               c(
                 "Seasonal Naive"
                )
)
```



Summary:

- the seasonal naive forecast is similar to the naive forecast method, except it is mainly used to forecast highly seasonal data
- in this forecast model, we set each forecast to equal to the last observed value from the same season(e.g. same month of the previous year)
- WMAE is 1686.329, which is significantly low as compared to the other two methods
- the predicted weekly sales also makes more sense as compared to simple ES and holt’s trend

4.5 Linear Model with Time Series Components(trend and seasonality)

```
# Using the forecast function to generate weekly sales projection
tslm_forecast <- function(current_ts, h){
  tslm(current_ts ~ trend + season) %>%
    forecast( h = h) %>%
    .$mean
}

# Produce the summary of the tslm
tslm_mod <- tslm(basic_train_mod ~ trend + season) %>% forecast(h = 36)
summary(tslm_mod)
```

```
##
## Forecast method: Linear regression model
##
## Model Information:
##
## Call:
## tslm(formula = basic_train_mod ~ trend + season)
##
## Coefficients:
## (Intercept)        trend      season2      season3      season4      season5
##    2837.911        3.730     -489.425     -620.535     -310.324      176.666
##     season6      season7      season8      season9     season10     season11
##     800.054      802.874     1378.051      767.610     1719.346     1288.191
##    season12     season13     season14     season15     season16     season17
##    1825.356      808.731     1627.001     1602.007     1434.732     1637.267
##    season18     season19     season20     season21     season22     season23
##    1685.567     2069.967     1758.033     1035.893     1261.663     1862.888
##    season24     season25     season26     season27     season28     season29
##    1568.204     1526.809      814.729     1388.249      851.379      373.350
##    season30     season31     season32     season33     season34     season35
##     593.835      386.770      676.995      484.976      358.416      637.526
##    season36     season37     season38     season39     season40     season41
##     296.476     -348.719      121.297     -178.633        6.712      524.697
##    season42     season43     season44     season45     season46     season47
##     -46.447       51.788      364.598     1105.963      485.848     1002.449
##    season48     season49     season50     season51     season52
##    1067.219      951.529     1283.869     1942.170     3136.860
##
##
## Error measures:
##                        ME    RMSE      MAE       MPE     MAPE      MASE
## Training set -3.82731e-14 387.511 306.1423 -1.102105 8.192261 0.4855692
##                   ACF1
## Training set 0.2318816
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2012.154       4008.338 3120.502 4896.173 2636.389 5380.287
## 2012.173       4963.803 4075.967 5851.638 3591.854 6335.752
## 2012.192       4536.378 3648.542 5424.213 3164.429 5908.327
## 2012.211       5077.273 4189.437 5965.108 3705.324 6449.222
## 2012.231       4064.378 3176.542 4952.213 2692.429 5436.327
## 2012.250       4886.378 3998.542 5774.213 3514.429 6258.327
## 2012.269       4865.113 3977.277 5752.948 3493.164 6237.062
## 2012.288       4701.568 3813.732 5589.403 3329.619 6073.517
## 2012.307       4907.833 4019.997 5795.668 3535.884 6279.782
## 2012.327       4959.863 4072.027 5847.698 3587.914 6331.812
## 2012.346       5347.993 4460.157 6235.828 3976.044 6719.942
## 2012.365       5039.788 4151.952 5927.623 3667.839 6411.737
## 2012.384       4321.378 3433.542 5209.213 2949.429 5693.327
## 2012.404       4550.878 3663.042 5438.713 3178.929 5922.827
## 2012.423       5155.833 4267.997 6043.668 3783.884 6527.782
```

```
## 2012.442       4864.878 3977.042 5752.713 3492.929 6236.827
## 2012.461       4827.213 3939.377 5715.048 3455.264 6199.162
## 2012.481       4118.863 3231.027 5006.698 2746.914 5490.812
## 2012.500       4696.113 3808.277 5583.948 3324.164 6068.062
## 2012.519       4162.973 3275.137 5050.808 2791.024 5534.922
## 2012.538       3688.673 2800.837 4576.508 2316.724 5060.622
## 2012.557       3912.888 3025.052 4800.723 2540.939 5284.837
## 2012.577       3709.553 2821.717 4597.388 2337.604 5081.502
## 2012.596       4003.508 3115.672 4891.343 2631.559 5375.457
## 2012.615       3815.218 2927.382 4703.053 2443.269 5187.167
## 2012.634       3692.388 2804.552 4580.223 2320.439 5064.337
## 2012.654       3975.228 3087.392 4863.063 2603.279 5347.177
## 2012.673       3637.908 2750.072 4525.743 2265.959 5009.857
## 2012.692       2996.443 2108.607 3884.278 1624.494 4368.392
## 2012.711       3470.188 2582.352 4358.023 2098.239 4842.137
## 2012.731       3173.988 2286.152 4061.823 1802.039 4545.937
## 2012.750       3363.063 2475.227 4250.898 1991.114 4735.012
## 2012.769       3884.778 2996.942 4772.613 2512.829 5256.727
## 2012.788       3317.363 2429.527 4205.198 1945.414 4689.312
## 2012.807       3419.328 2531.492 4307.163 2047.379 4791.277
## 2012.827       3735.868 2848.032 4623.703 2363.919 5107.817
```

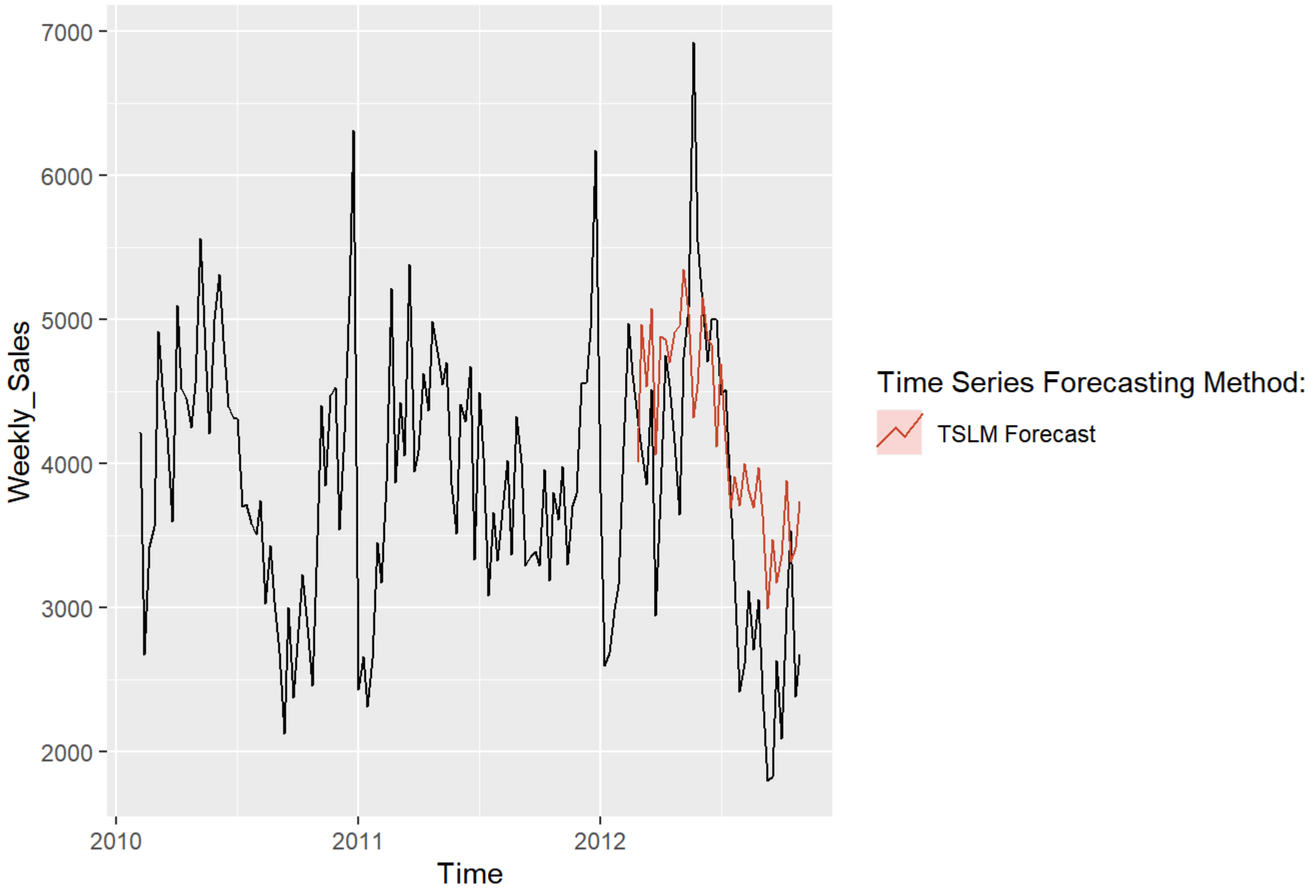
```
# Make prediction on the validate data for tslm
tslm_pred <- model_forecasts(train_data, validate_set, tslm_forecast)
```

```
## 34.81 sec elapsed
```

```
# Calculate the WMAE on the tslm validate data
WMAE(tslm_pred)
```

```
## [1] 1482.701
```

```
# Create a plot to display linear model with ts components
forecast_plots(basic_ts,
               list(
                    tslm_mod
                    ),
               c(
                 "TSLM Forecast"
                )
)
```



Summary:

- TSLM model is a linear model that we can use to fit with time series that has both trend and seasonality components
- the WMAE is 1674.749, which is slightly lower than the seasonal naive forecast
- as shown in the summary table, TSLM predicted that the 3rd month has the highest weekly sales while the rest of the months' weekly sales are relatively similar to what seasonal naive forecast predicted

4.6 TBATS Forecast

```
# Using the forecast function to generate weekly sales projection
tbats_fc <- function(current_ts, h){
  forecast::forecast(current_ts, h = h)$mean
}

# Produce the summary of the TBATS
tbats_mod <- forecast::forecast(basic_train_mod, h = 36)
summary(tbats_mod)
```

```
##
## Forecast method: STL +  ETS(A,N,N)
##
## Model Information:
## ETS(A,N,N)
##
## Call:
##  ets(y = na.interp(x), model = etsmodel, allow.multiplicative.trend = allow.multiplicative.tr
end)
##
##   Smoothing parameters:
##     alpha = 0.2068
##
##   Initial states:
##     l = 3663.313
##
##   sigma:  388.7651
##
##      AIC     AICc      BIC
## 1780.050 1780.283 1788.069
##
## Error measures:
##                    ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 26.69818 385.1146 305.4423 -0.2866351 8.043471 0.4844589
##                    ACF1
## Training set 0.06771511
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2012.154       4127.258 3629.036 4625.481 3365.293 4889.224
## 2012.173       5075.838 4567.073 5584.603 4297.750 5853.927
## 2012.192       4647.011 4127.918 5166.104 3853.126 5440.895
## 2012.211       5195.314 4666.094 5724.534 4385.942 6004.686
## 2012.231       4175.228 3636.072 4714.385 3350.659 4999.797
## 2012.250       4987.000 4438.087 5535.913 4147.510 5826.491
## 2012.269       4969.558 4411.059 5528.058 4115.407 5823.710
## 2012.288       4802.081 4234.156 5370.005 3933.515 5670.646
## 2012.307       5010.285 4433.090 5587.480 4127.542 5893.029
## 2012.327       5056.271 4469.952 5642.590 4159.573 5952.968
## 2012.346       5433.978 4838.674 6029.281 4523.540 6344.416
## 2012.365       5128.382 4524.227 5732.536 4204.408 6052.356
## 2012.384       4405.441 3792.564 5018.318 3468.127 5342.756
## 2012.404       4625.133 4003.656 5246.611 3674.665 5575.601
## 2012.423       5230.646 4600.686 5860.607 4267.205 6194.088
## 2012.442       4938.805 4300.474 5577.136 3962.562 5915.048
## 2012.461       4903.234 4256.641 5549.827 3914.356 5892.113
## 2012.481       4184.284 3529.534 4839.035 3182.929 5185.639
## 2012.500       4765.415 4102.607 5428.223 3751.738 5779.093
## 2012.519       4230.080 3559.311 4900.848 3204.227 5255.932
## 2012.538       3746.787 3068.151 4425.423 2708.903 4784.671
## 2012.557       3972.063 3285.651 4658.476 2922.285 5021.842
## 2012.577       3764.055 3069.952 4458.157 2702.516 4825.593
```

```
## 2012.596       4055.604 3353.896 4757.313 2982.434 5128.775
## 2012.615       3870.466 3161.233 4579.698 2785.788 4955.143
## 2012.634       3738.119 3021.441 4454.796 2642.055 4834.183
## 2012.654       4025.748 3301.702 4749.794 2918.415 5133.081
## 2012.673       3685.360 2954.020 4416.701 2566.871 4803.849
## 2012.692       3039.877 2301.314 3778.440 1910.342 4169.411
## 2012.711       3505.624 2759.909 4251.339 2365.151 4646.097
## 2012.731       3210.342 2457.542 3963.141 2059.034 4361.649
## 2012.750       3392.886 2633.068 4152.704 2230.845 4554.928
## 2012.769       3913.235 3146.463 4680.007 2740.558 5085.912
## 2012.788       3340.260 2566.596 4113.923 2157.043 4523.476
## 2012.807       3445.283 2664.789 4225.777 2251.620 4638.946
## 2012.827       3752.684 2965.418 4539.950 2548.665 4956.703
```

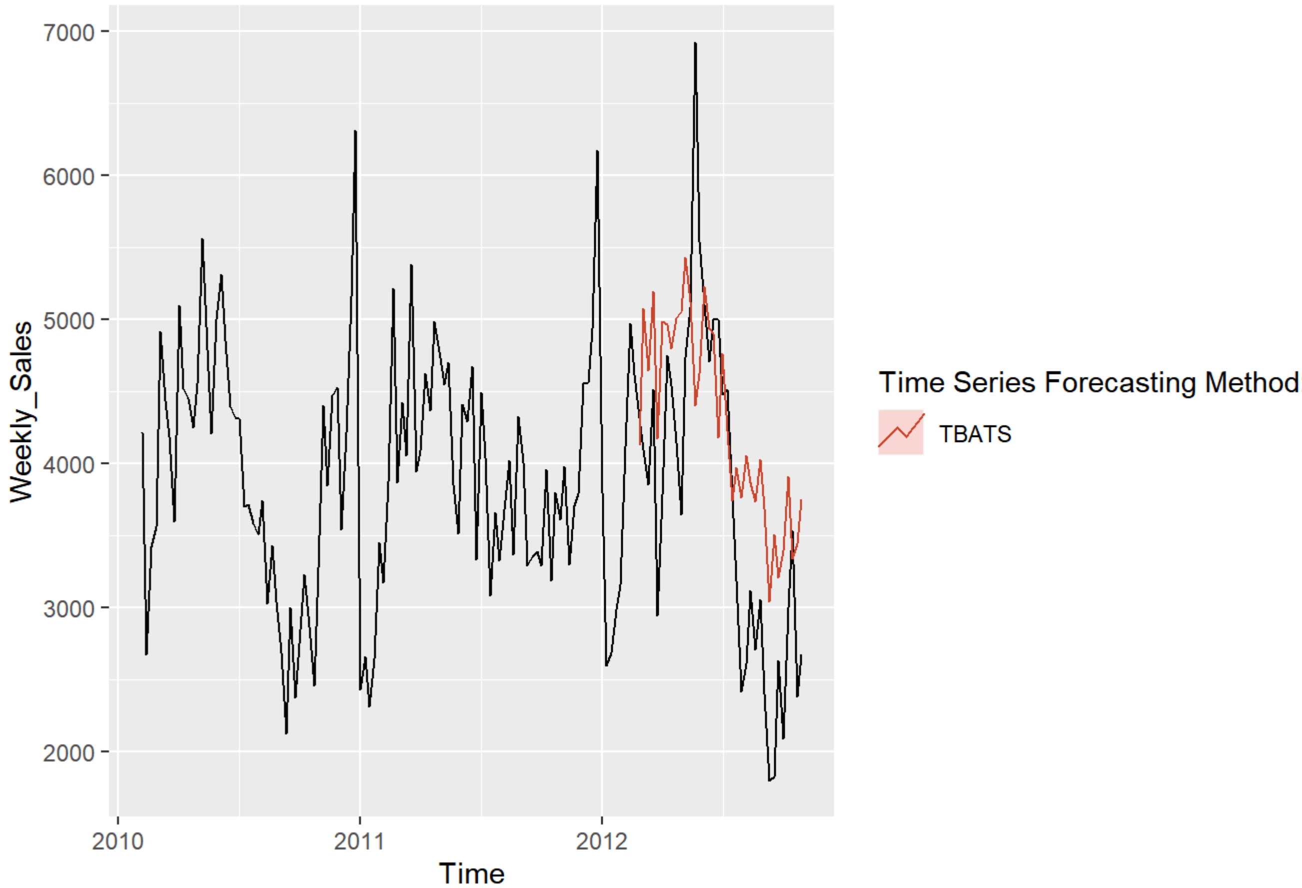
```
# Make prediction on the validate data for TBATS
tbats_pred <- model_forecasts(train_data, validate_set, tbats_fc)
```

```
## 117.92 sec elapsed
```

```
# Calculate the WMAE on the TBATS validate data
WMAE(tbats_pred)
```

```
## [1] 1354.773
```

```
# Create a plot to display ES state space model with Box-Cox transformation, ARMA errors, Trend
 and Seasonal components
forecast_plots(basic_ts,
               list(
                    tbats_mod
                    ),
               c(
                 "TBATS"
                )
)
```



Summary:

- TBATS model is a forecasting method used to forecast complex seasonal time series data using exponential smoothing
- the WMAE is 1515.041, which is a lot smaller than the other four forecast methods that we have seen so far
- as shown in the summary table, TSLM also predicted that the 3rd month has the highest weekly sales while the rest of the months' weekly sales are relatively similar to what the seasonal naive forecast predicted

4.7 Seasonal and Trend decomposition using Loess Forecasting Model - ETS

```
# Using the forecast function to generate weekly sales projection
stl_ets <- function(current_ts, h){
  stlf(current_ts, method = "ets", opt.crit = 'mae', h = h)$mean
}

# Produce the summary of the TBATS
stl_ets_mod <- stlf(basic_train_mod, method = "ets", 36)
summary(stl_ets_mod)
```

```
##
## Forecast method: STL +  ETS(A,N,N)
##
## Model Information:
## ETS(A,N,N)
##
## Call:
##  ets(y = na.interp(x), model = etsmodel, allow.multiplicative.trend = allow.multiplicative.tr
end)
##
##   Smoothing parameters:
##     alpha = 0.2068
##
##   Initial states:
##     l = 3663.313
##
##   sigma:  388.7651
##
##      AIC     AICc      BIC
## 1780.050 1780.283 1788.069
##
## Error measures:
##                    ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 26.69818 385.1146 305.4423 -0.2866351 8.043471 0.4844589
##                    ACF1
## Training set 0.06771511
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2012.154       4127.258 3629.036 4625.481 3365.293 4889.224
## 2012.173       5075.838 4567.073 5584.603 4297.750 5853.927
## 2012.192       4647.011 4127.918 5166.104 3853.126 5440.895
## 2012.211       5195.314 4666.094 5724.534 4385.942 6004.686
## 2012.231       4175.228 3636.072 4714.385 3350.659 4999.797
## 2012.250       4987.000 4438.087 5535.913 4147.510 5826.491
## 2012.269       4969.558 4411.059 5528.058 4115.407 5823.710
## 2012.288       4802.081 4234.156 5370.005 3933.515 5670.646
## 2012.307       5010.285 4433.090 5587.480 4127.542 5893.029
## 2012.327       5056.271 4469.952 5642.590 4159.573 5952.968
## 2012.346       5433.978 4838.674 6029.281 4523.540 6344.416
## 2012.365       5128.382 4524.227 5732.536 4204.408 6052.356
## 2012.384       4405.441 3792.564 5018.318 3468.127 5342.756
## 2012.404       4625.133 4003.656 5246.611 3674.665 5575.601
## 2012.423       5230.646 4600.686 5860.607 4267.205 6194.088
## 2012.442       4938.805 4300.474 5577.136 3962.562 5915.048
## 2012.461       4903.234 4256.641 5549.827 3914.356 5892.113
## 2012.481       4184.284 3529.534 4839.035 3182.929 5185.639
## 2012.500       4765.415 4102.607 5428.223 3751.738 5779.093
## 2012.519       4230.080 3559.311 4900.848 3204.227 5255.932
## 2012.538       3746.787 3068.151 4425.423 2708.903 4784.671
## 2012.557       3972.063 3285.651 4658.476 2922.285 5021.842
## 2012.577       3764.055 3069.952 4458.157 2702.516 4825.593
```

```
## 2012.596       4055.604 3353.896 4757.313 2982.434 5128.775
## 2012.615       3870.466 3161.233 4579.698 2785.788 4955.143
## 2012.634       3738.119 3021.441 4454.796 2642.055 4834.183
## 2012.654       4025.748 3301.702 4749.794 2918.415 5133.081
## 2012.673       3685.360 2954.020 4416.701 2566.871 4803.849
## 2012.692       3039.877 2301.314 3778.440 1910.342 4169.411
## 2012.711       3505.624 2759.909 4251.339 2365.151 4646.097
## 2012.731       3210.342 2457.542 3963.141 2059.034 4361.649
## 2012.750       3392.886 2633.068 4152.704 2230.845 4554.928
## 2012.769       3913.235 3146.463 4680.007 2740.558 5085.912
## 2012.788       3340.260 2566.596 4113.923 2157.043 4523.476
## 2012.807       3445.283 2664.789 4225.777 2251.620 4638.946
## 2012.827       3752.684 2965.418 4539.950 2548.665 4956.703
```

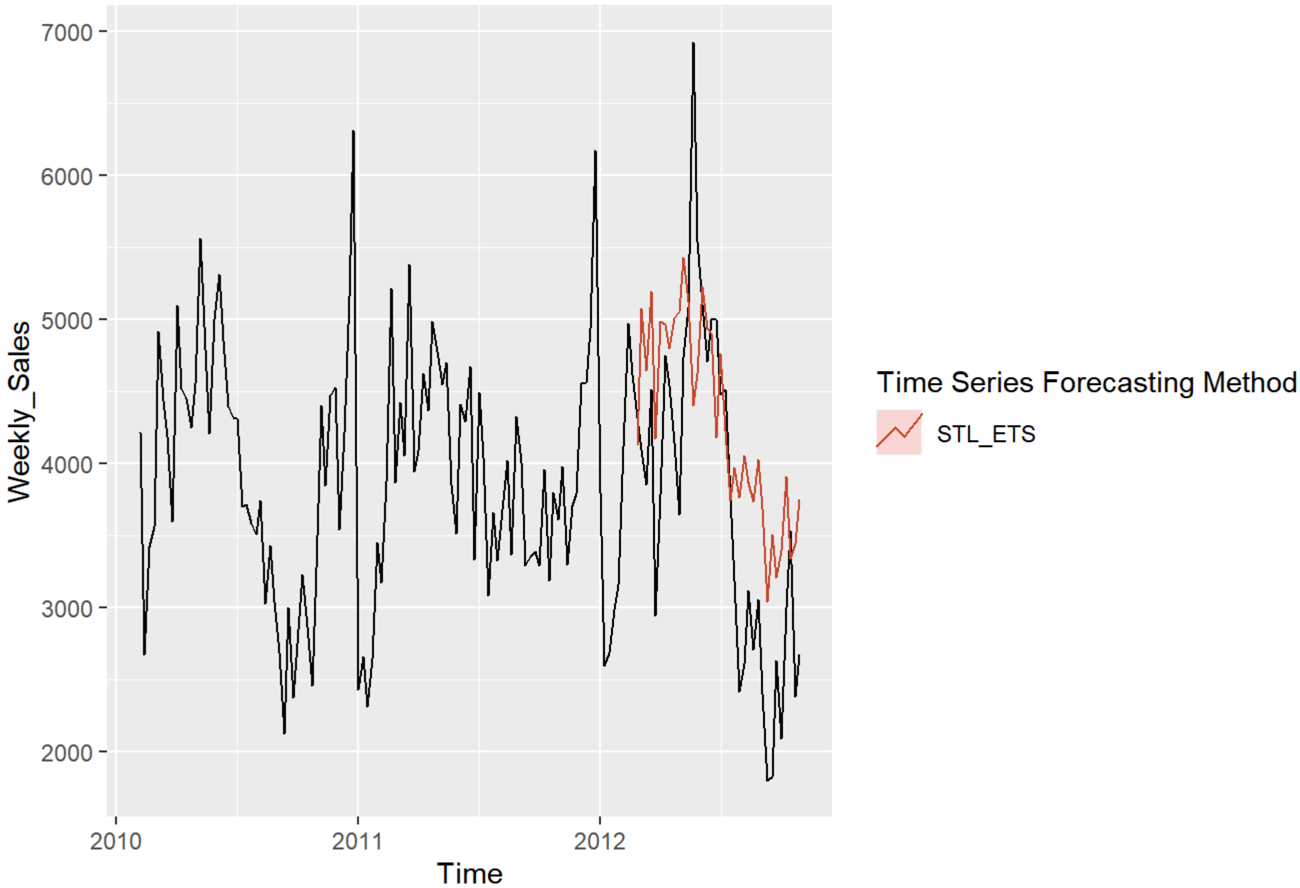
```
# Make prediction on the validate data for TBATS
stl_ets_pred <- model_forecasts(train_data, validate_set, stl_ets)
```

```
## 143.97 sec elapsed
```

```
# Calculate the WMAE on the TBATS validate data
WMAE(stl_ets_pred)
```

```
## [1] 1362.378
```

```
# Create a plot to display ES state space model with Box-Cox transformation, ARMA errors, Trend
 and Seasonal components
forecast_plots(basic_ts,
               list(
                    stl_ets_mod
                    ),
               c(
                 "STL_ETS"
                )
)
```



Summary:

- STL-ETS model is a method to decompose time series, it stands for “Seasonal and Trend decomposition using Loess”
  - it can be used to estimate nonlinear relationships
- the WMAE is 1477.846, which is a lot smaller than the other five forecast methods that we have seen so far
- as shown in the summary table, STL-ETS, it also predicted that the 3rd month has the highest weekly sales while the rest of the months’ weekly sales are relatively similar to what the seasonal naive forecast predicted

4.8 Seasonal and Trend decomposition using Loess Forecasting Model - ARIMA

```
# Using the forecast function to generate weekly sales projection
stl_arima <- function(current_ts, h){
  stlf(current_ts, method = "arima", h = h)$mean
}

# Produce the summary of the TBATS
stl_arima_mod <- stlf(basic_train_mod, method = "ets", 36)
summary(stl_arima_mod)
```

```
##
## Forecast method: STL +  ETS(A,N,N)
##
## Model Information:
## ETS(A,N,N)
##
## Call:
##  ets(y = na.interp(x), model = etsmodel, allow.multiplicative.trend = allow.multiplicative.tr
end)
##
##   Smoothing parameters:
##     alpha = 0.2068
##
##   Initial states:
##     l = 3663.313
##
##   sigma:  388.7651
##
##      AIC     AICc      BIC
## 1780.050 1780.283 1788.069
##
## Error measures:
##                    ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 26.69818 385.1146 305.4423 -0.2866351 8.043471 0.4844589
##                    ACF1
## Training set 0.06771511
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2012.154       4127.258 3629.036 4625.481 3365.293 4889.224
## 2012.173       5075.838 4567.073 5584.603 4297.750 5853.927
## 2012.192       4647.011 4127.918 5166.104 3853.126 5440.895
## 2012.211       5195.314 4666.094 5724.534 4385.942 6004.686
## 2012.231       4175.228 3636.072 4714.385 3350.659 4999.797
## 2012.250       4987.000 4438.087 5535.913 4147.510 5826.491
## 2012.269       4969.558 4411.059 5528.058 4115.407 5823.710
## 2012.288       4802.081 4234.156 5370.005 3933.515 5670.646
## 2012.307       5010.285 4433.090 5587.480 4127.542 5893.029
## 2012.327       5056.271 4469.952 5642.590 4159.573 5952.968
## 2012.346       5433.978 4838.674 6029.281 4523.540 6344.416
## 2012.365       5128.382 4524.227 5732.536 4204.408 6052.356
## 2012.384       4405.441 3792.564 5018.318 3468.127 5342.756
## 2012.404       4625.133 4003.656 5246.611 3674.665 5575.601
## 2012.423       5230.646 4600.686 5860.607 4267.205 6194.088
## 2012.442       4938.805 4300.474 5577.136 3962.562 5915.048
## 2012.461       4903.234 4256.641 5549.827 3914.356 5892.113
## 2012.481       4184.284 3529.534 4839.035 3182.929 5185.639
## 2012.500       4765.415 4102.607 5428.223 3751.738 5779.093
## 2012.519       4230.080 3559.311 4900.848 3204.227 5255.932
## 2012.538       3746.787 3068.151 4425.423 2708.903 4784.671
## 2012.557       3972.063 3285.651 4658.476 2922.285 5021.842
## 2012.577       3764.055 3069.952 4458.157 2702.516 4825.593
```

```
## 2012.596       4055.604 3353.896 4757.313 2982.434 5128.775
## 2012.615       3870.466 3161.233 4579.698 2785.788 4955.143
## 2012.634       3738.119 3021.441 4454.796 2642.055 4834.183
## 2012.654       4025.748 3301.702 4749.794 2918.415 5133.081
## 2012.673       3685.360 2954.020 4416.701 2566.871 4803.849
## 2012.692       3039.877 2301.314 3778.440 1910.342 4169.411
## 2012.711       3505.624 2759.909 4251.339 2365.151 4646.097
## 2012.731       3210.342 2457.542 3963.141 2059.034 4361.649
## 2012.750       3392.886 2633.068 4152.704 2230.845 4554.928
## 2012.769       3913.235 3146.463 4680.007 2740.558 5085.912
## 2012.788       3340.260 2566.596 4113.923 2157.043 4523.476
## 2012.807       3445.283 2664.789 4225.777 2251.620 4638.946
## 2012.827       3752.684 2965.418 4539.950 2548.665 4956.703
```

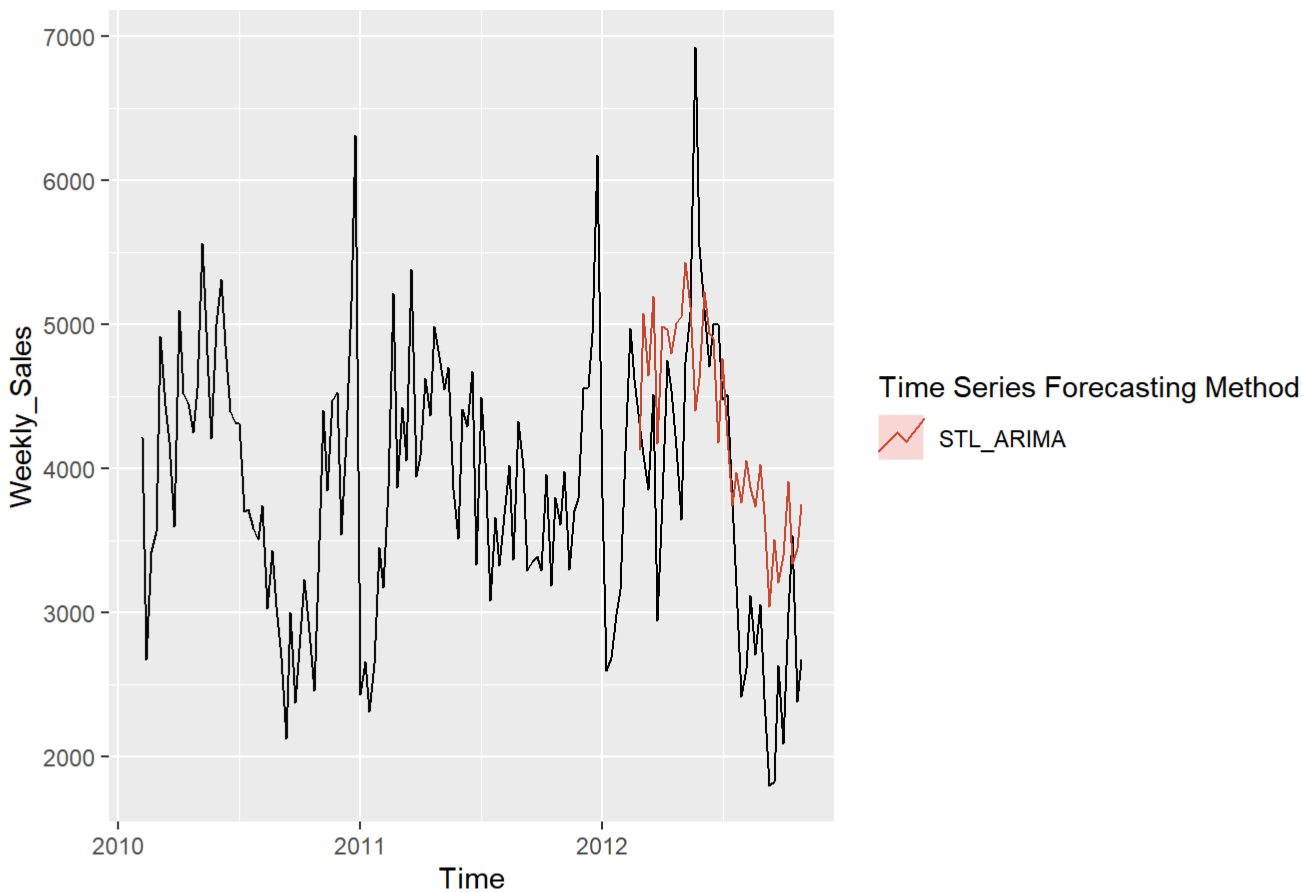
```
# Make prediction on the validate data for TBATS
stl_arima_pred <- model_forecasts(train_data, validate_set, stl_arima)
```

```
## 355.15 sec elapsed
```

```
# Calculate the WMAE on the TBATS validate data
WMAE(stl_arima_pred)
```

```
## [1] 1324.513
```

```
# Create a plot to display ES state space model with Box-Cox transformation, ARMA errors, Trend
 and Seasonal components
forecast_plots(basic_ts,
               list(
                    stl_arima_mod
                    ),
               c(
                 "STL_ARIMA"
                )
)
```



Summary:

- STL-ARIMA is similar to STL-ETS, except it used ARIMA components to decompose the time series
- the WMAE is 1454.388, which is a lot smaller than the other six forecast methods that we have seen so far
- as shown in the summary table, STL-ARIMA, it also predicted that the 3rd month has the highest weekly sales while the rest of the months' weekly sales are relatively similar to what the seasonal naive forecast predicted

4.9 Final Model

```
# Define a variable to save the final forecast model based on the lowest WMAE
final_stl_arima_mod <- model_forecasts(store_dept_mts, nums_wks_test, stl_arima)
```

```
## 367.72 sec elapsed
```

4.10 Holidays Adjustments

```
# Shift the holidays
adjust_holidays <- function(full_forecast){
  adjustment <- function(fc){
  if(2 * fc[9] < fc[8]){
    adj <- fc[8] * (2.5 / 7)
    fc[9] <- fc[9] + adj
    fc[8] <- fc[8] - adj
    }
  fc
  }
  apply(full_forecast, 2, adjustment)
}

final_forecast <- adjust_holidays(final_stl_arima_mod)

# # Make the observations 0 for stores that don't have historical observations
store_dept_names <- colnames(store_dept_mts)
colnames(final_forecast) <- store_dept_names

# Create a tibble
test_dates <- tibble("Date" = seq(begin_test, end_test, by = 7))
final <-
  cbind(test_dates, final_forecast) %>%
  pivot_longer(!Date, names_to = "Store_Dept", values_to = "Weekly_Sales")
```

Summary:

- created a post-forecast adjustment to account for any given department whose average sales for example in weeks 49, 50,51 are at least 10% higher than for weeks 48 and 52, then the code will shift a fraction of the sales from weeks 48 and 52 into the next week
- this function will apply to the final model - STL_ARIMA

4.11 Final Weekly Sales Projection

```
# Write the projected weekly sales to csv
final_wkly_sales <-
  test_forecast %>%
  left_join(final, by = c("Store_Dept", "Date")) %>%
  replace_na(list(Weekly_Sales = 0)) %>%
  mutate(Id = paste0(Store_Dept, "_", Date)) %>%
  dplyr::select(Id, Weekly_Sales)
final_wkly_sales
```

```
## # A tibble: 115,064 x 2
##    Id             Weekly_Sales
##    <chr>                 <dbl>
##  1 1_1_2012-11-02       34210.
##  2 1_1_2012-11-09       19124.
##  3 1_1_2012-11-16       19305.
##  4 1_1_2012-11-23       19881.
##  5 1_1_2012-11-30       23921.
##  6 1_1_2012-12-07       32408.
##  7 1_1_2012-12-14       45340.
##  8 1_1_2012-12-21       32974.
##  9 1_1_2012-12-28       39568.
## 10 1_1_2013-01-04       16261.
## # ... with 115,054 more rows
```

```
write_csv(final_wkly_sales, "D:\\Projects\\walmart_weekly_sales_projection.csv")
```