COMPUTER ENGINEERING DEPARTMENT

Applied Machine Learning Project Report

“SALE FORECASTING”

Enes Kayan

Ahmed H. Ibrahim

Hasibullah Mahmood

Mohamed Mahmoud Abbiyah

05/06/2020

Table of Contents

1. Project Overview

The data belongs to Walmart Inc. an American multinational retail corporation that operates a chain of hypermarkets. It covers sales from 2011-01-29 up to 2016-04-24, 1913 days, more than 5 years in three states of USA, California, Texas and Wisconsin. There are 10 stores, 4 in CA, 3 in TX and 3 in WI. Also, there are 3 categories of products Food, Hobbies and Household which in turn are divided into 3, 2, 2 departments respectively.

In this project, we are going to Forecast or predict future sales based on states.
The following steps will be done:

- Data description
- Data pre-processing
- Creating Time Series models
- Creating the Regression Model
- Models comparison

Data Pre-processing, modeling and plotting are done in R programming language.

Three informative files are loaded and described below. The data is downloaded from the Kaggle website.

2. Data Description

2.1 Loading data

```
sales.df <- read.csv("sales_train_validation.csv")
calendar.df <- read.csv("calendar.csv")
price.df <- read.csv("sell_prices.csv")
```

2.2 sales_train_validation.csv

Sample of data in the file:

	id	item_id	dept_id	cat_id	store_id	state_id	d_1	d_2	...d-1913
1	HOBBIES_1_001_CA_1_validation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0
2	HOBBIES_1_002_CA_1_validation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0
3	HOBBIES_1_003_CA_1_validation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0
4	HOBBIES_1_004_CA_1_validation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0
5	HOBBIES_1_005_CA_1_validation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0
6	HOBBIES_1_006_CA_1_validation	HOBBIES_1_006	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0
7	HOBBIES_1_007_CA_1_validation	HOBBIES_1_007	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0
8	HOBBIES_1_008_CA_1_validation	HOBBIES_1_008	HOBBIES_1	HOBBIES	CA_1	CA	12	15	0
9	HOBBIES_1_009_CA_1_validation	HOBBIES_1_009	HOBBIES_1	HOBBIES	CA_1	CA	2	0	7
10	HOBBIES_1_010_CA_1_validation	HOBBIES_1_010	HOBBIES_1	HOBBIES	CA_1	CA	0	0	1

Features descriptions and types

- id: unique id combination of item-id and store-id.
- item-id: non-unique id combination of dept-id.
- dept-id(department-id): There are 7 departments. Its type is nominal.
- cat-id(categorical-id): There are 3 categories, Food, Household and hobbies. Its type is nominal.
- store-id: There are 10 stores in 3 states of the USA. Its type is nominal.
- state-id: There are 3 states, CA, TX and WI. Its type is nominal.

- d-1 … d-1913: 1913 days are describing the number of sales per day. Its type is numeric.

2.3 calendar.csv

Sample of data in the file:

	date	wm_yr_wk	weekday	wday	month	year	d	event_name_1	event_type_1	event_name_2	event_type_2	snap_CA	snap_TX	snap_WI
1	2011-01-29	11101	Saturday	1	1	2011	d_1					0	0	0
2	2011-01-30	11101	Sunday	2	1	2011	d_2					0	0	0
3	2011-01-31	11101	Monday	3	1	2011	d_3					0	0	0
4	2011-02-01	11101	Tuesday	4	2	2011	d_4					1	1	0
5	2011-02-02	11101	Wednesday	5	2	2011	d_5					1	0	1
6	2011-02-03	11101	Thursday	6	2	2011	d_6					1	1	1
7	2011-02-04	11101	Friday	7	2	2011	d_7					1	0	0
8	2011-02-05	11102	Saturday	1	2	2011	d_8					1	1	1
9	2011-02-06	11102	Sunday	2	2	2011	d_9	SuperBowl	Sporting			1	1	1
10	2011-02-07	11102	Monday	3	2	2011	d_10					1	1	0
11	2011-02-08	11102	Tuesday	4	2	2011	d_11					1	0	1

Features descriptions and types

- date: dates for 1913 days described above.
- wm_yr_wk: Starting from left, the first digit stands for Walmart id, 2nd and 3rd digits stand for year and last two digits stand for week's number. Its type is numeric.
- weekday: Its type is ordinal.
- wday(weekday): an encoded form of weekday. Its type is ordinal.
- month: 12 months, its type is ordinal.
- year: 2011:2016, its type is ordinal.
- d(day): 1:1913, its type is ordinal.
- event_name_1: There are around 30 events like SuperBowl. Its type is nominal.
- event_type_1: There are around 5 event types like Sporting. Its type is nominal.
- event_name_2 & event_type_2: There may be two events on the same day.
- snap_CA, snap_TX & snap_WI: Supplemental Nutrition Assistance Program is a federal program that provides food-purchasing assistance for low and no-income people. Its type is binary.

2.4 **Sell_price**.csv

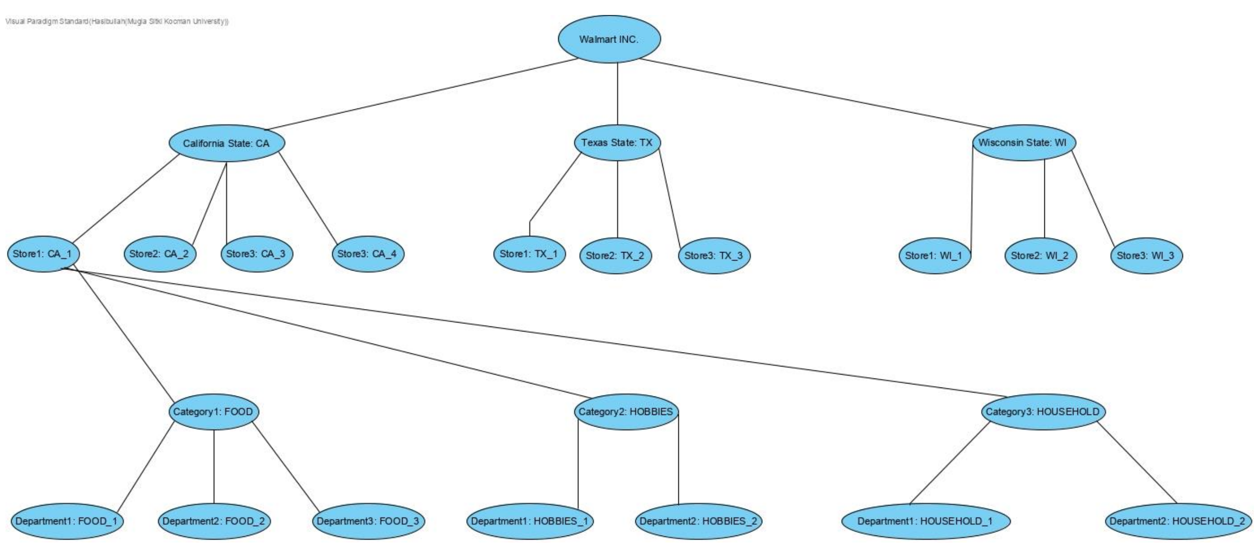
```
head(price.df)

##   store_id        item_id wm_yr_wk sell_price
## 1     CA_1 HOBBIES_1_001    11325       9.58
## 2     CA_1 HOBBIES_1_001    11326       9.58
## 3     CA_1 HOBBIES_1_001    11327       8.26
## 4     CA_1 HOBBIES_1_001    11328       8.26
## 5     CA_1 HOBBIES_1_001    11329       8.26
## 6     CA_1 HOBBIES_1_001    11330       8.26
```

Features descriptions and types

- sell_price: The price of product per store and week. Its type is numeric.

Note: The rest of the features are described above.

2.5 Hierarchy summary of data



3. Data Pre-processing

```
## Upload all required libraries

library(data.table) # For converting data into time series format
library(ggplot2)    # To plot various plots
library(fpp2)       # For examining seasonality graphically
library(forecast)   # For various functions related to Time series
library(stats)      # For applying tests like acf, Ljung-Box Tests
library(tseries)    # For applying Dickey Fuller test
library(xts)        # Extensible time series format
```

The Calendar dataset will be used for all models, so we will clean it first.

```
length(calendar.df[, 1])

## [1] 1969
```

There are 1969 rows, but the labeled data is only 1913 rows, so we get the subset of it with necessary columns.

```
# Get subset of data from row 1 up-to 1913
calendar.df <- calendar.df[1:1913, c(1, 4, 8:length(calendar.df))]


# Show 10 rows of data
head(calendar.df, 10)
```

	date	wday	event_name_1	event_type_1	event_name_2	event_type_2	snap_CA	snap_TX	snap_WI
1	2011-01-29	1					0	0	0
2	2011-01-30	2					0	0	0
3	2011-01-31	3					0	0	0
4	2011-02-01	4					1	1	0
5	2011-02-02	5					1	0	1
6	2011-02-03	6					1	1	1
7	2011-02-04	7					1	0	0
8	2011-02-05	1					1	1	1
9	2011-02-06	2	SuperBowl	Sporting			1	1	1
10	2011-02-07	3					1	1	0

```
# Factorize(encode) categorical columns.
event.name.1 <- unique(calendar.df$event_name_1)
event.type.1 <- unique(calendar.df$event_type_1)
event.name.2 <- unique(calendar.df$event_name_2)
event.type.2 <- unique(calendar.df$event_type_2)
# Show an example of factorized columns.
data.frame(event.name.1)

##           event.name.1
## 1
## 2            SuperBowl
## 3        ValentinesDay
## 4        PresidentsDay
## 5            LentStart
## 6            LentWeek2
## 7        StPatricksDay
## 8            Purim End
## 9       OrthodoxEaster
## 10          Pesach End
## 11        Cinco De Mayo
```

```
## 12        Mother's day
## 13         MemorialDay
## 14      NBAFinalsStart
## 15        NBAFinalsEnd
## 16        Father's day
## 17     IndependenceDay
## 18      Ramadan starts
## 19         Eid al-Fitr
## 20            LaborDay
## 21         ColumbusDay
## 22           Halloween
## 23           EidAlAdha
## 24         VeteransDay
## 25        Thanksgiving
## 26           Christmas
## 27        Chanukah End
## 28             NewYear
## 29   OrthodoxChristmas
## 30 MartinLutherKingDay
## 31              Easter

# Apply factorization on data.
calendar.df$event_name_1 <- factor(calendar.df$event_name_1,
                                   levels = event.name.1,
                                   labels = 1:length(event.name.1))

calendar.df$event_type_1 <- factor(calendar.df$event_type_1,
                                   levels = event.type.1,
                                   labels = 1:length(event.type.1))

calendar.df$event_name_2 <- factor(calendar.df$event_name_2,
                                   levels = event.name.2,
                                   labels = 1:length(event.name.2))

calendar.df$event_type_2 <- factor(calendar.df$event_type_2,
                                   levels = event.type.2,
                                   labels = 1:length(event.type.2))

# Remove unnecessary variables
rm(event.name.1, event.type.1, event.name.2, event.type.2)
```

```r
# Show 10 rows of data
head(calendar.df, 10)
```

	date	wday	event_name_1	event_type_1	event_name_2	event_type_2	snap_CA	snap_TX	snap_WI
1	2011-01-29	1	1	1	1	1	0	0	0
2	2011-01-30	2	1	1	1	1	0	0	0
3	2011-01-31	3	1	1	1	1	0	0	0
4	2011-02-01	4	1	1	1	1	1	1	0
5	2011-02-02	5	1	1	1	1	1	0	1
6	2011-02-03	6	1	1	1	1	1	1	1
7	2011-02-04	7	1	1	1	1	1	0	0
8	2011-02-05	1	1	1	1	1	1	1	1
9	2011-02-06	2	2	2	1	1	1	1	1
10	2011-02-07	3	1	1	1	1	1	1	0

```r
# Get the subset of data from column 6 up-to end
ca.df <- sales.df[, 6:ncol(sales.df)]

# Get the rows where state-id is equal to CA(California)
ca.df <- ca.df[ca.df$state_id=="CA",]

# sum the number of items sold per day
ca.df <- data.frame(colSums(ca.df[, 2:ncol(ca.df)]))

# Change the column name
colnames(ca.df) <- "nItemSold"

# Print sample
head(ca.df)

##     nItemSold
## d_1     14195
## d_2     13805
## d_3     10108
## d_4     11047
## d_5      9925
## d_6     11322

# Print summary
summary(ca.df)

##    nItemSold
##  Min.   :    5
##  1st Qu.:12834
##  Median :14678
##  Mean   :14990
##  3rd Qu.:16846
##  Max.   :25224
```

As shown above, the minimum number of item sold per day is only 5, mean and median are 14990 and 14678 items per day respectively. Also the maximum number of items sold per day is 25224 items.

4. Time Series Analysis

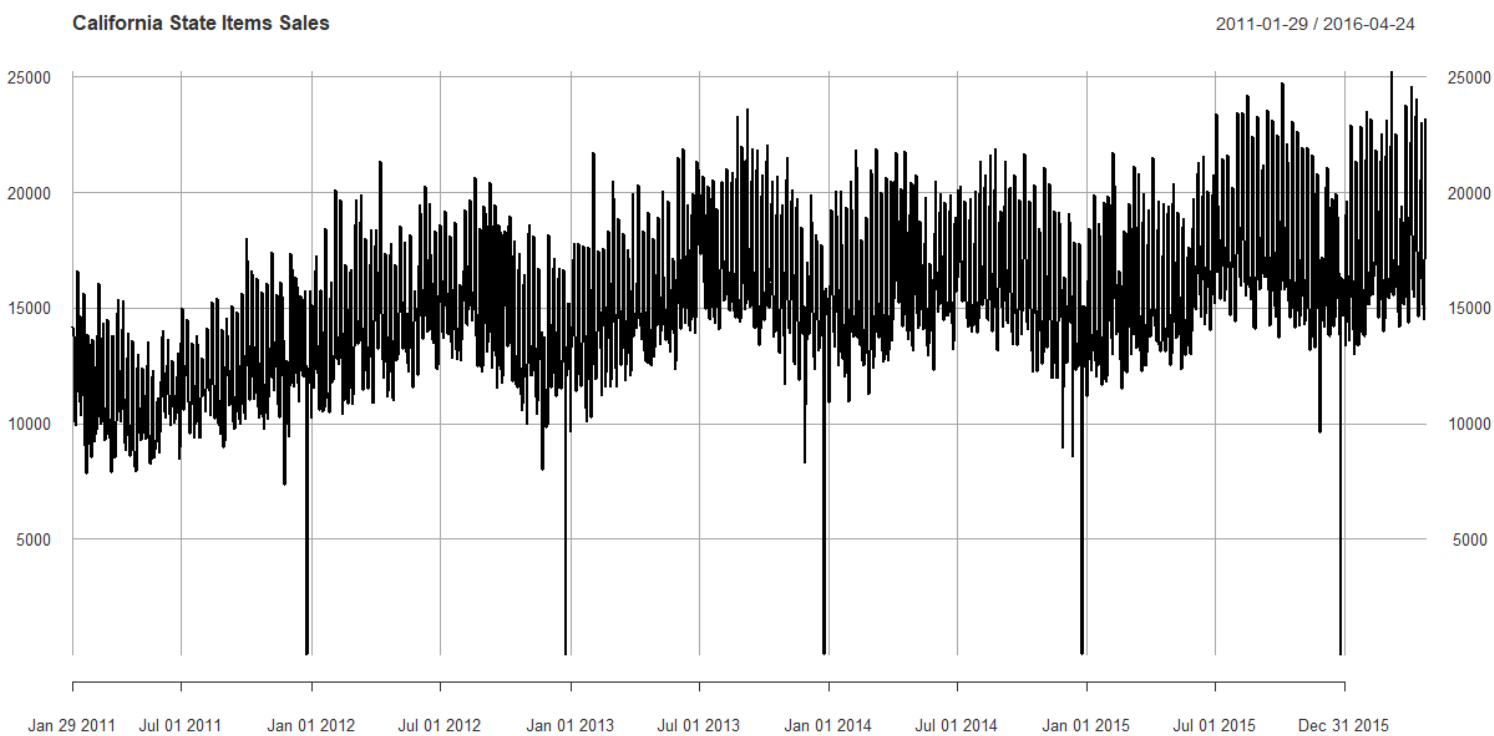
4.1 Data Decomposition

```
# Add date to dataset
ca.df$date <- calendar.df[, 1]
# order the columns
ca.df <- ca.df[, c(2, 1)]
# Convert character to date format
ca.df$date <- strptime(as.character(ca.df$date), "%m/%d/%Y")

summary(ca.df)

##       date                          nItemSold
##  Min.   :2011-01-29 00:00:00   Min.   :    5
##  1st Qu.:2012-05-21 00:00:00   1st Qu.:12834
##  Median :2013-09-11 00:00:00   Median :14678
##  Mean   :2013-09-11 00:25:22   Mean   :14990
##  3rd Qu.:2015-01-02 00:00:00   3rd Qu.:16846
##  Max.   :2016-04-24 00:00:00   Max.   :25224
```

As printed above, the date starts from 2011-01-29 up to 2016-04-24.

```
# Convert data into time seires formats
ca.xts <- xts(ca.df$nItemSold, ca.df$date)
ca.ts <- ts(ca.df[, 2], start = c(2011, 29), end = c(2016, 116), frequency =
365)
# Change column name
colnames(ca.xts) <- c("nItemSold")
length(ca.ts)

## [1] 1913
```

```r
## Plot the Time series data
plot(ca.xts, xlab="Date", ylab = "# of Items sold",
     main = "Items Sales data")
```

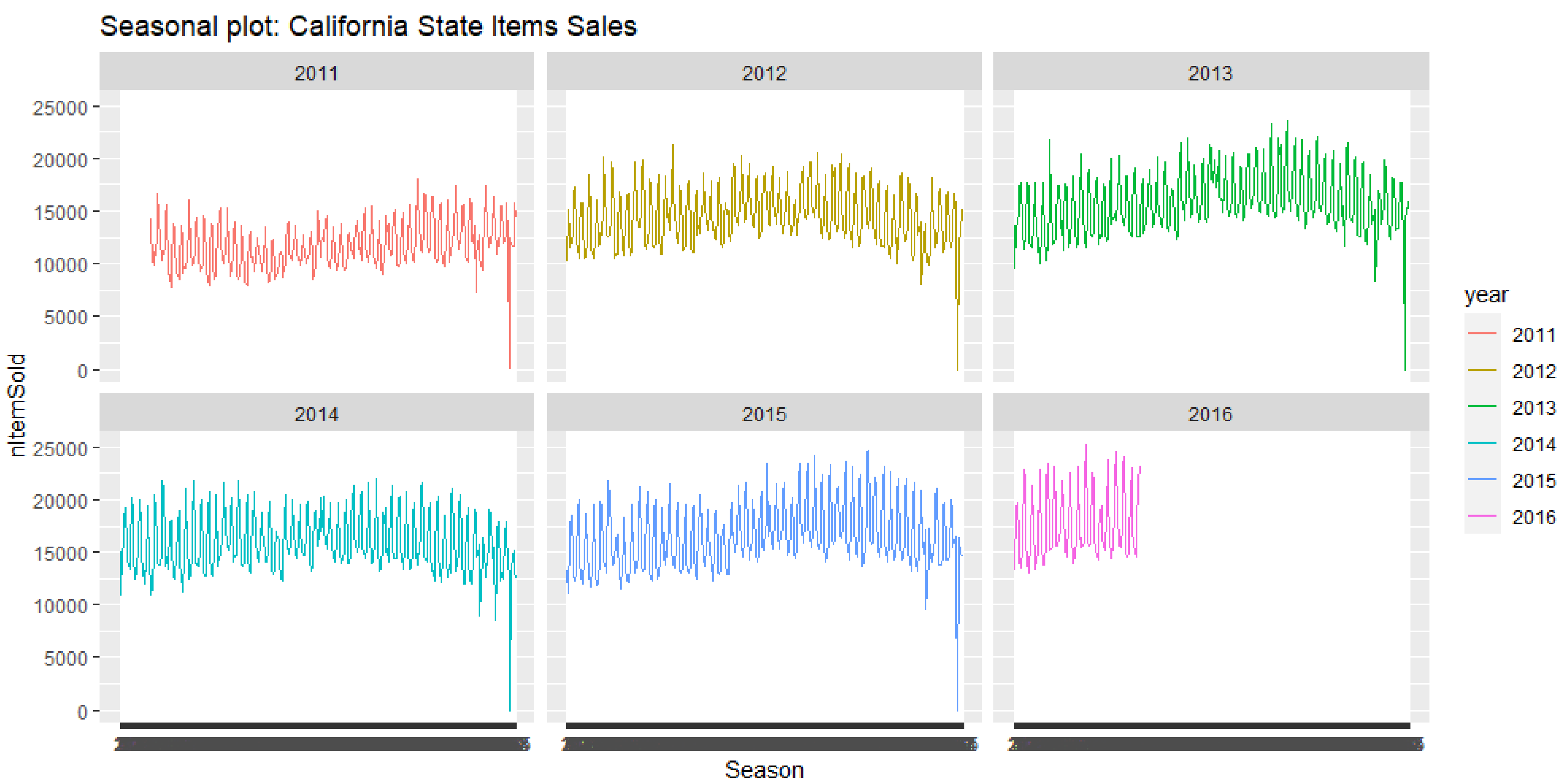


Following are the observations obtained from the above Figure:

- Data values are stored in the correct time order and no data is missing.
- The sales are increasing in numbers, implying the presence of a trend component.
- Intra-year stable fluctuations are indicative of a seasonal component.
- As the trend increases, fluctuations are also increasing. This is indicative of multiplicative seasonality.

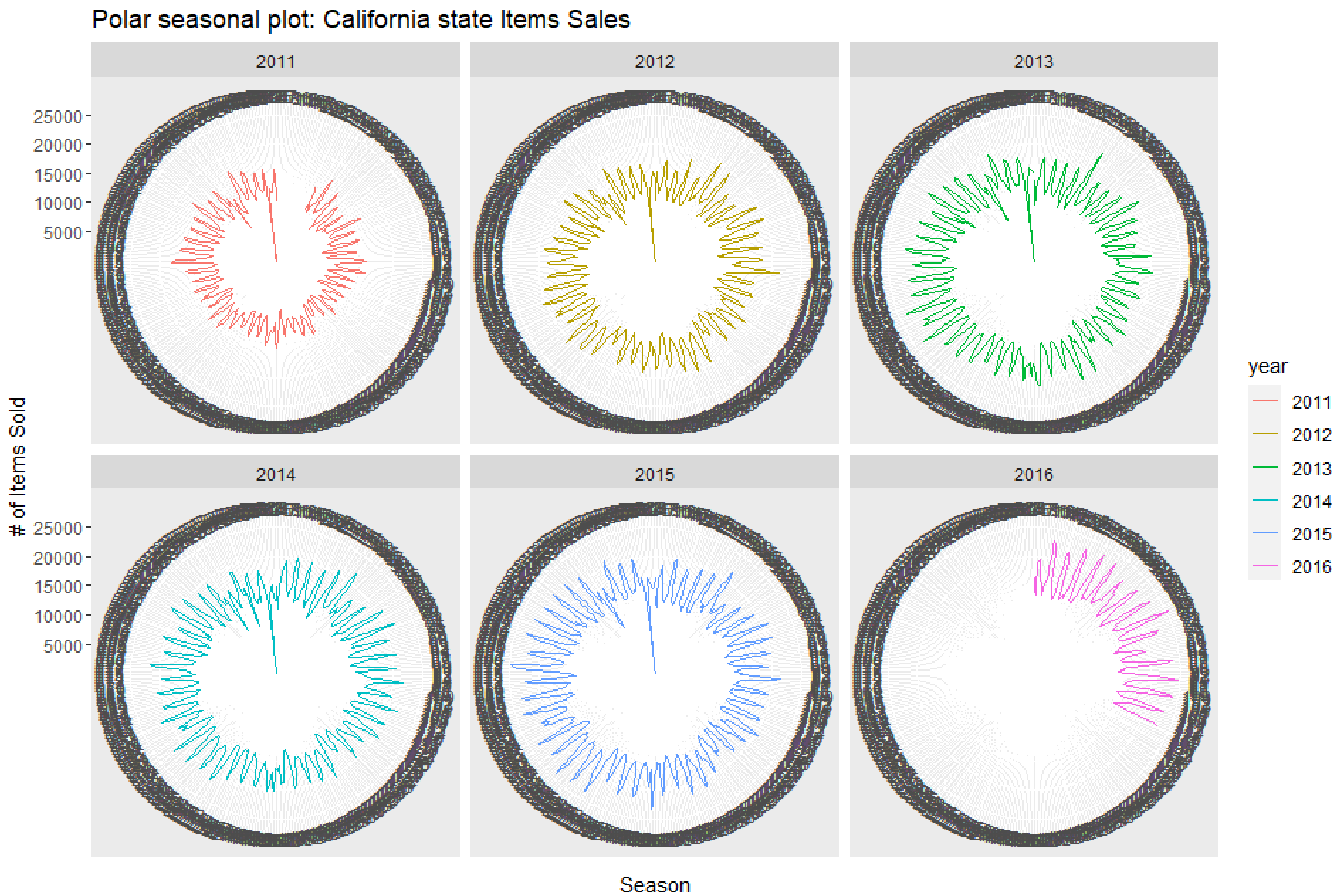
Following plots help to identify the seasonality fluctuations better.

```r
# Existence of seasonality can be observed using various plots
# Plot 1: Seasonal plot Year-wise (using ggseasonalplot())
ggseasonplot(x = ca.ts) +
  ylab("nItemSold") +
  ggtitle("Seasonal plot: California State Items Sales")  +
  facet_wrap(~year)
```



The above seasonal plot indicates, as the years go by, the sale increases. Also, there is a common seasonality pattern. The patterns are similar but not identical which means that there are trend and seasonality but there is also something else that is particular to given year and season.

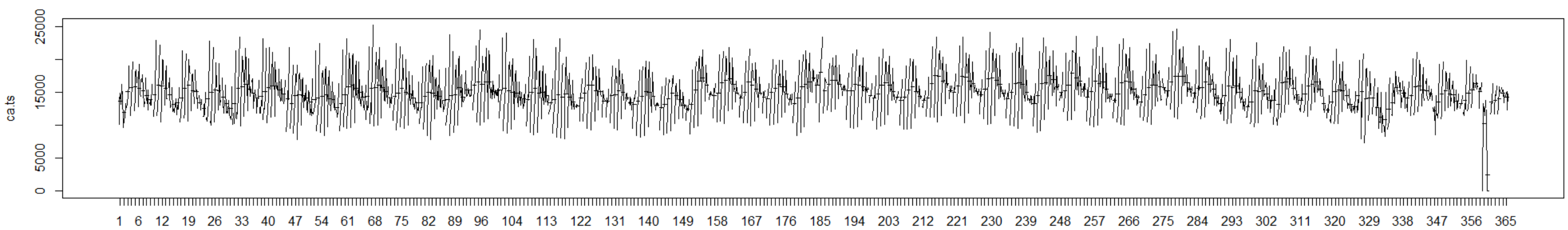
```r
## Plot 2: Polar Seasonal plot Year-wise (using ggseasonplot())
ggseasonplot(ca.ts, polar=TRUE) +
  ylab("# of Items Sold") +
  ggtitle("Polar seasonal plot: California state Items Sales") +
  facet_wrap(~year)
```



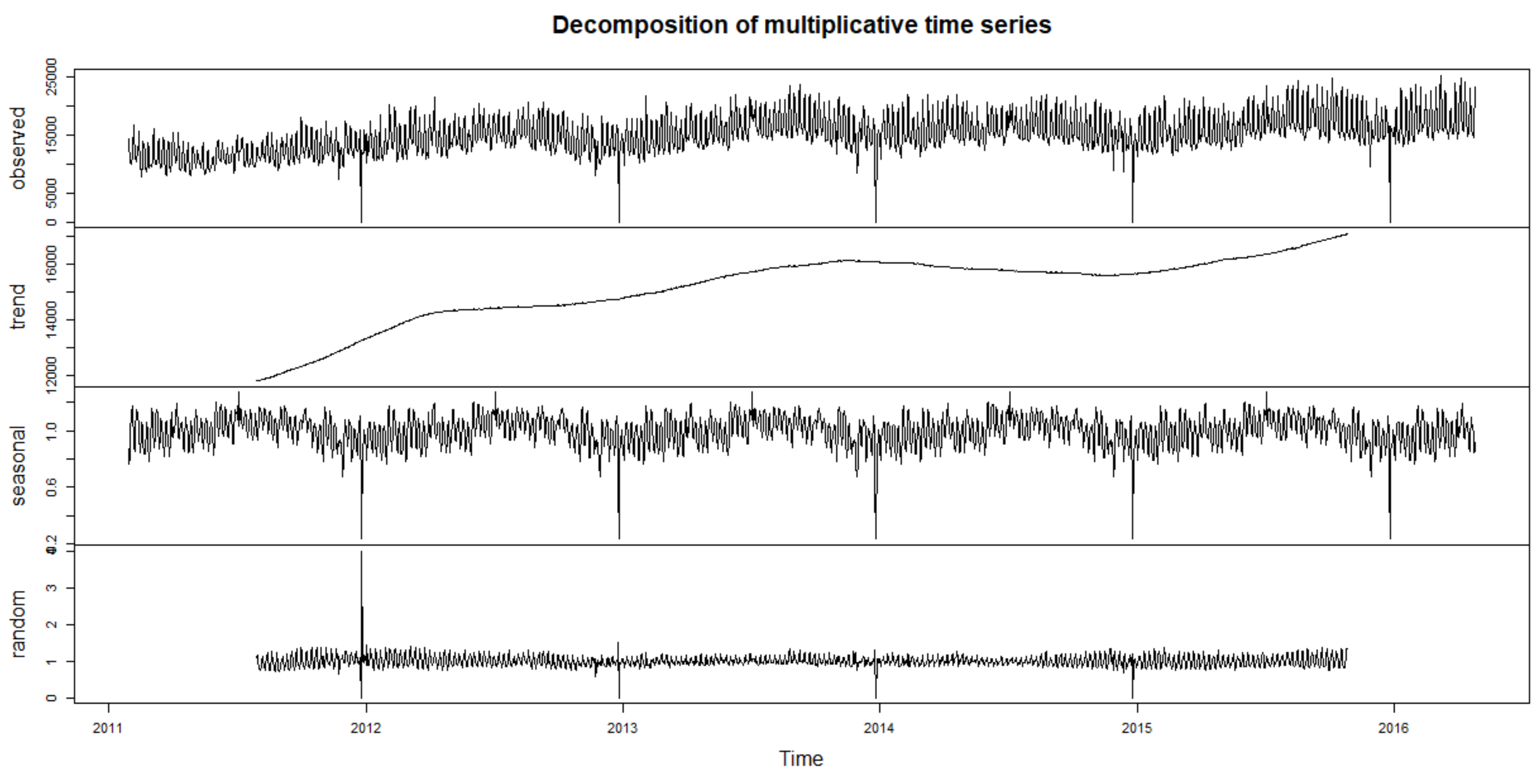
Similarly, if we focus to above figure we can identify each year the circle is getting bigger and bigger meaning of increases in sale.

```
## Plot 3: Seasonal plot Month-wise (using monthplot())
monthplot(ca.ts)
```

In this figure, each vertical line represents a day in a year and shows minimum, maximum, and mean of item sales in more than 5 years.




```
## Decomposition of TS using decompose()
TSDecmpose<-decompose(ca.ts, type = "multiplicative")
plot(TSDecmpose)
```

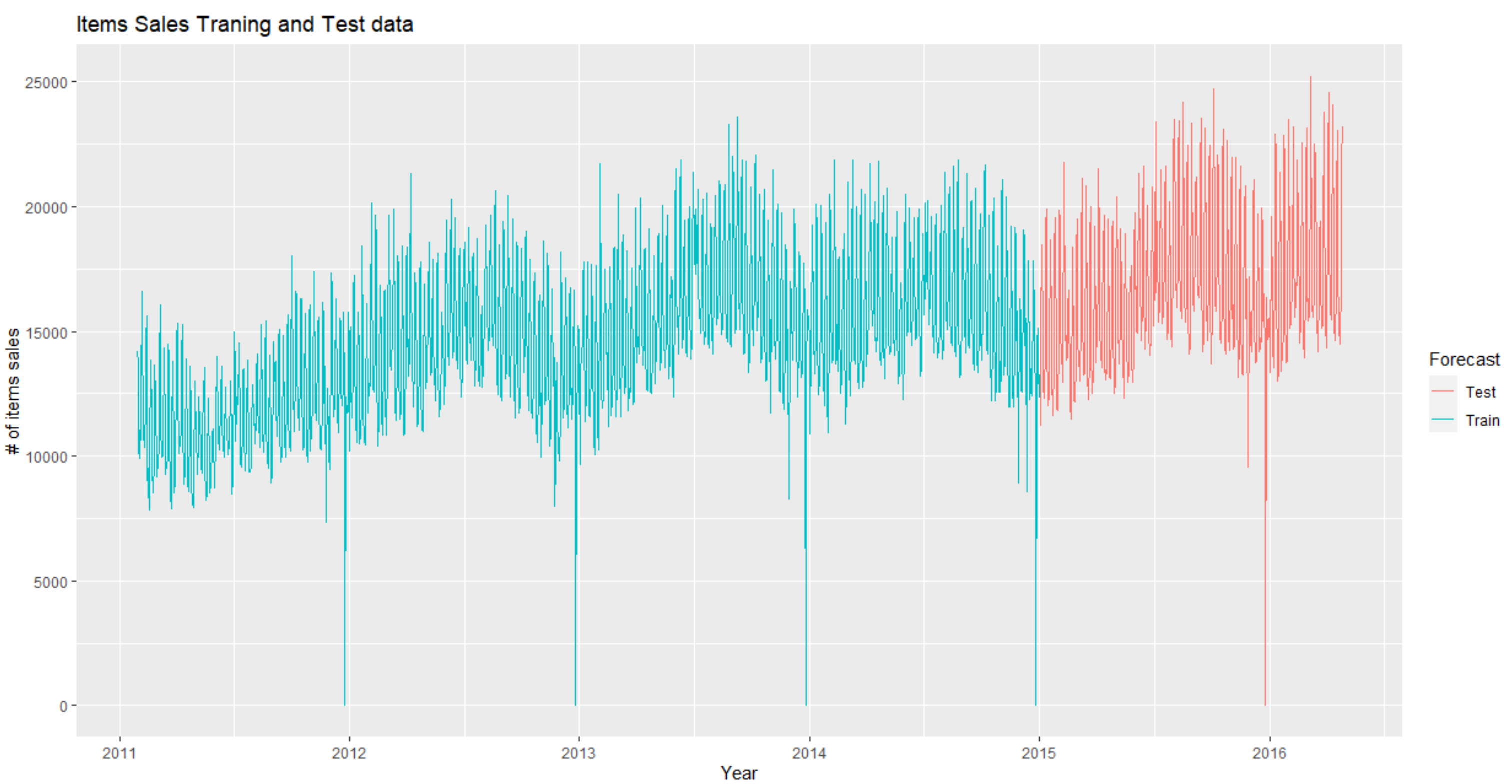


As we can see, the observed-plot shows the original data plot which is equal to the multiplication of trend, seasonality, and random error results. As the years go by, the trend increases linearly. Also, the seasonality in the graph is the same for all years because randomness or irregularity is taken out of it and shown in the random plot.

4.2 Splitting data

We are going to split data into 75% for training and 25% for testing. Here we can't split it randomly because the data for time series analysis needs to be ordered by time.

```
## Spliting data into training and test data sets
TS_Train <- window(ca.ts, start=c(2011,29), end=c(2014, 366), freq=365)
length(TS_Train)

## [1] 1433

1433 days as training set.

TS_Test <- window(ca.ts, start=c(2015,2), freq=365)
length(TS_Test)

## [1] 480

480 days as test set.

autoplot(TS_Train, series="Train") +
  autolayer(TS_Test, series="Test") +
  ggtitle("Items Sales Traning and Test data") +
  xlab("Year") + ylab("# of items sales") +
  guides(colour=guide_legend(title="Forecast"))
```



4.3 Random Walk with Drift Model

4.3.1 Model Creation

There are some limitation with $decompose()$ function so, $stl()$ function is used instead. But it doesn't admit seasonal multiplication. We will use log transformation to convert multiplicative seasonality to additive seasonality.

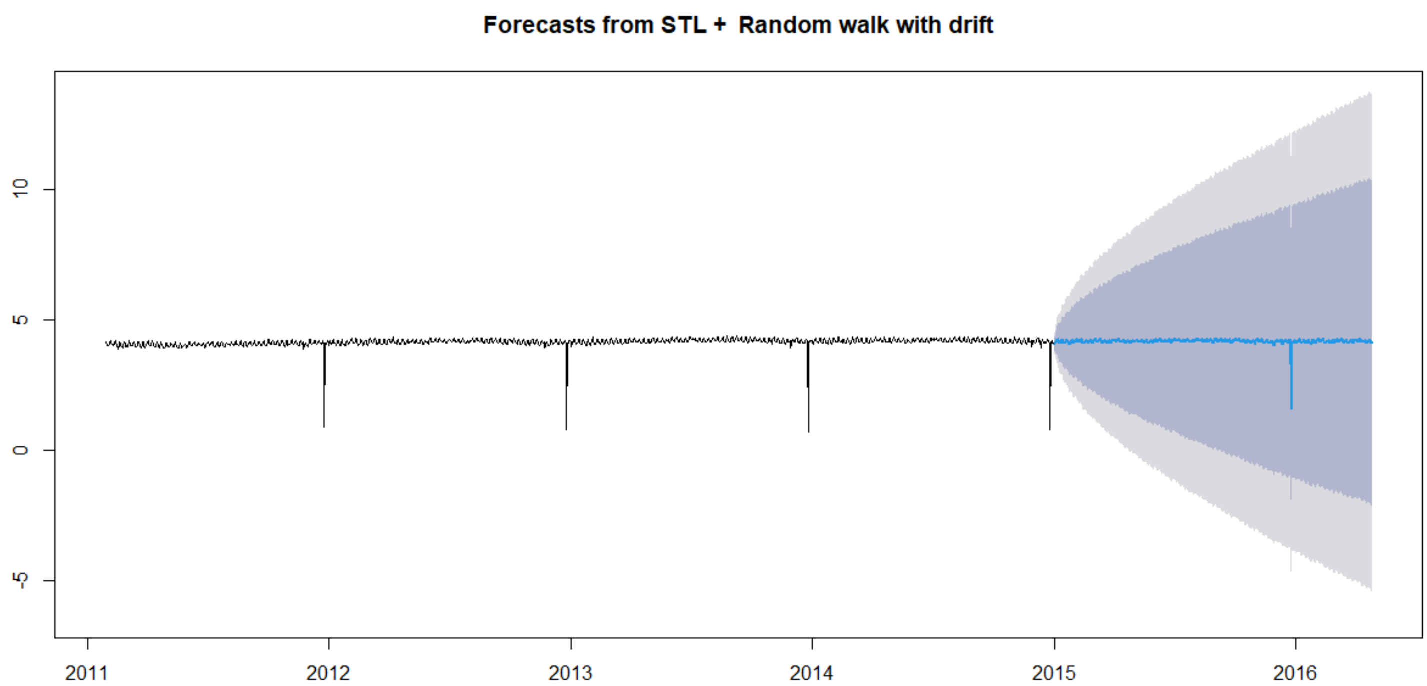
```
TSDecmpose_train_Log<-stl(log10(TS_Train), s.window='p')
head(TSDecmpose_train_Log)
```

```
## $time.series
## Time Series:
## Start = c(2011, 29)
## End = c(2015, 1)
## Frequency = 365
##                 seasonal    trend     remainder
## 2011.077  0.0090579105 4.002155  1.409222e-01
## 2011.079 -0.0461698696 4.002437  1.837690e-01
## 2011.082 -0.0819906316 4.002719  8.393666e-02
## 2011.085 -0.0214806431 4.003001  6.172383e-02
## 2011.088  0.0318581280 4.003283 -3.841075e-02
## 2011.090  0.0930531335 4.003565 -4.269508e-02
## 2011.093  0.0731269965 4.003847  1.119748e-02
## 2011.096  0.0521886493 4.004129  1.640520e-01
## 2011.099  0.0078492653 4.004411  1.549389e-01
## 2011.101 -0.0234300027 4.004693  9.142801e-02
## 2011.104 -0.0233897229 4.004975  5.715416e-02
## 2011.107  0.0286229529 4.005257 -1.893948e-02
## 2011.110  0.0707726008 4.005539 -2.693834e-02
## 2011.112  0.0728063382 4.005821 -2.347825e-02
```

Looking to above output, we can see there are increase and decrease in seasonal variable and an increase in trend which are predictable. However, remainder values are random and unpredictable.

Using stl object and random walk with drift method, 480 days are forecasted and plotted below:

```
TS_Train_stl<-forecast(TSDecmpose_train_Log, method="rwdrift", h=480)
plot(TS_Train_stl)
```

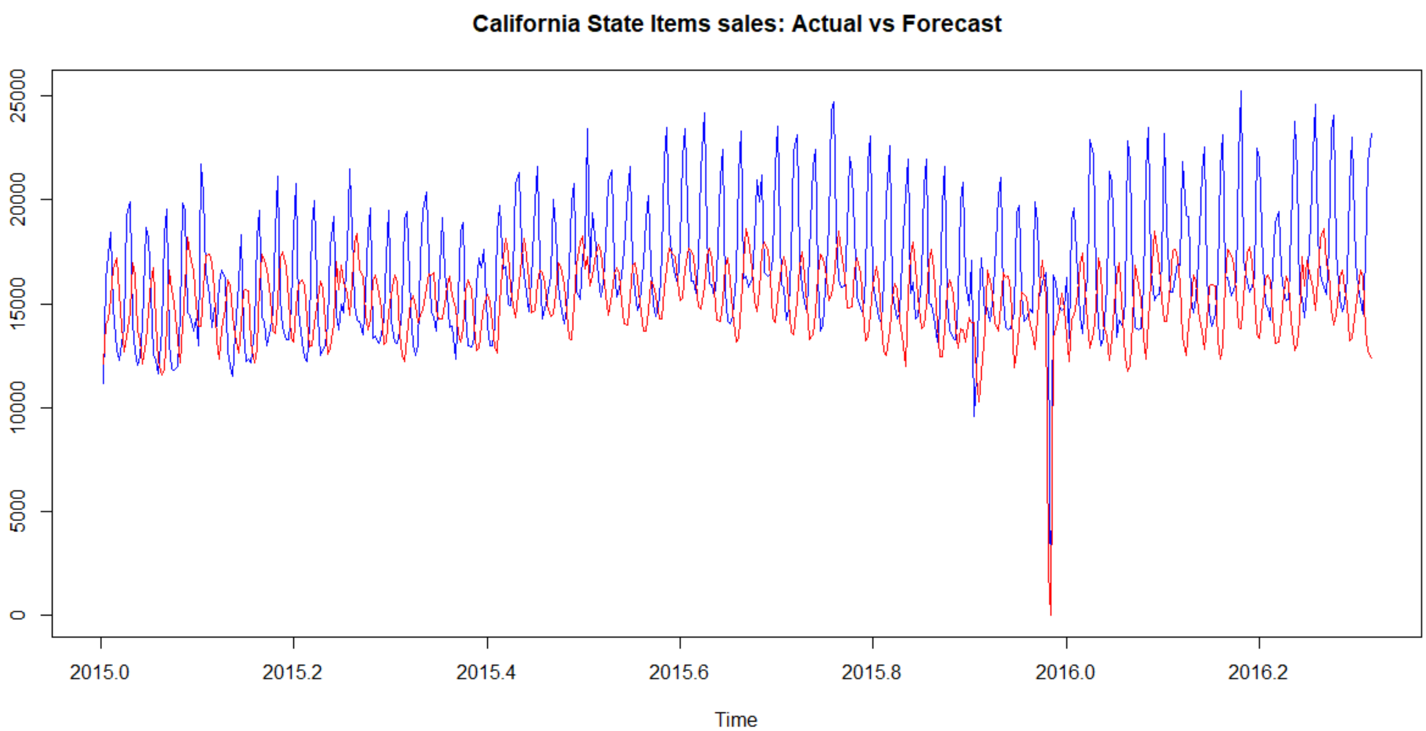



```
result <- data.frame(date=tail(ca.df$date, 480),
        original=tail(ca.df$nItemSold, 480),
        forecasted=10^TS_Train_stl$mean,
        lower80=10^TS_Train_stl$lower[,1], upper80=10^TS_Train_stl$upper[,1],
        lower95=10^TS_Train_stl$lower[,2], upper95=10^TS_Train_stl$upper[,2])


head(result)

##          date original forecasted  lower80  upper80  lower95   upper95
## 1 2015-01-01    11169   12082.02 6851.880 21304.40 5074.704  28765.26
## 2 2015-01-02    16179   14012.46 6281.014 31260.70 4107.270  47805.21
## 3 2015-01-03    17410   14249.73 5331.548 38085.55 3168.384  64087.85
## 4 2015-01-04    18465   15257.45 4901.248 47496.02 2686.801  86641.99
## 5 2015-01-05    14833   16706.32 4691.396 59492.15 2395.046 116532.71
## 6 2015-01-06    12796   17191.23 4274.371 69141.93 2045.992 144447.44
```

Date, original sale and forecasted sale are all listed above with 80% and 95% confidence interval. Greater confidence cause greater interval.

4.3.2 Accuracy measures: RMSE and MAPE

```
Vec2<- 10^(cbind(log10(TS_Test) ,as.data.frame(forecast(TSDecmpose_train_Log,
                                             method="rwdrift", h=480))[,1]))


ts.plot(Vec2, col=c("blue", "red"), main="California State Items sales: Actua
l vs Forecast")
```



Above figure shows a systematic under estimation.

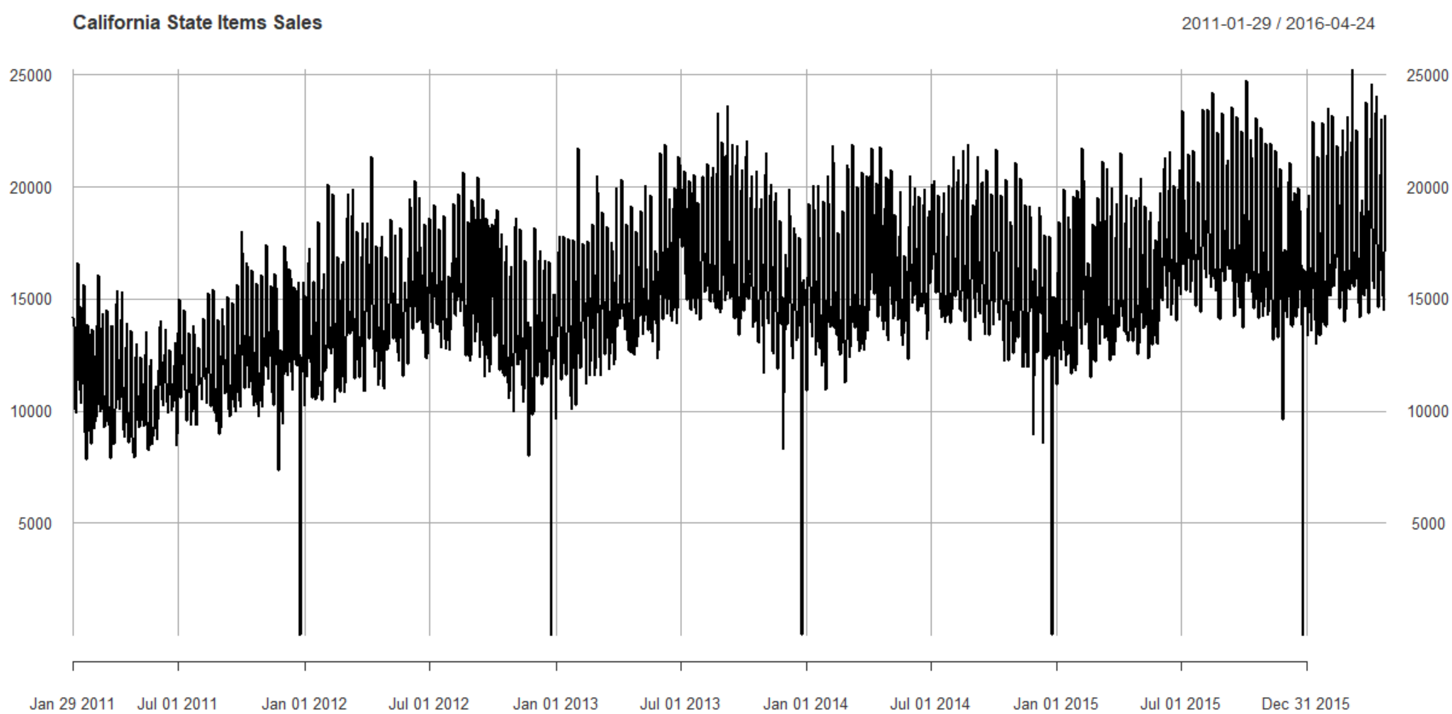
```
RMSE2 <- round(sqrt(sum(((Vec2[,1]-Vec2[,2])^2)/length(Vec2[,1]))),4)
MAPE2 <- round(mean(abs(Vec2[,1]-Vec2[,2])/Vec2[,1]),4)
paste("Accuracy Measures: RMSE:", RMSE2, "and MAPE:", MAPE2)

## [1] "Accuracy Measures: RMSE: 4083.1071 and MAPE: 0.1882"
```

Mean Absolute Percentage Error (MAPE) = **18.82%**
Root Mean Square Error (RMSE) = **4083.11**

4.4 Auto-Regressive Integrated Moving Average (ARIMA) Model

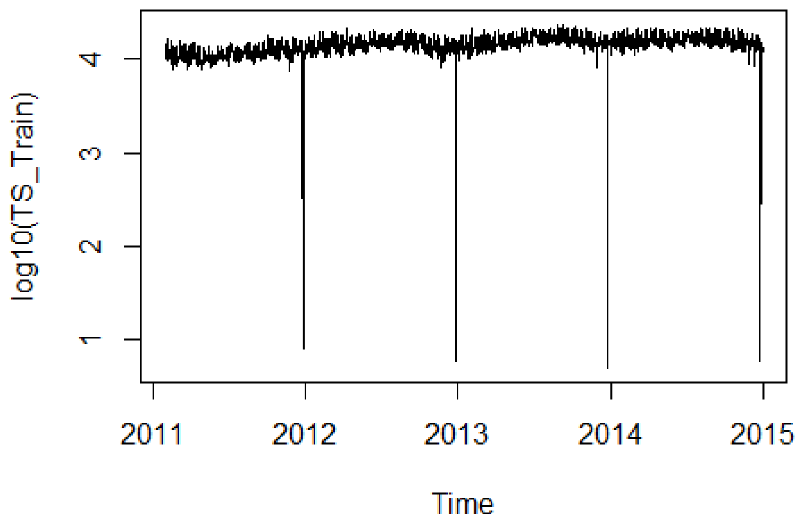
ARIMA model can only be applied to stationary time series data. First, we need to check whether our data is stationary or not. If the data is not stationary we need to transform it into stationary.

```
# Plot items sales
plot(ca.ts, xlab="Years", ylab = "# of Items sold",
     main = "California State Items Sales")
```

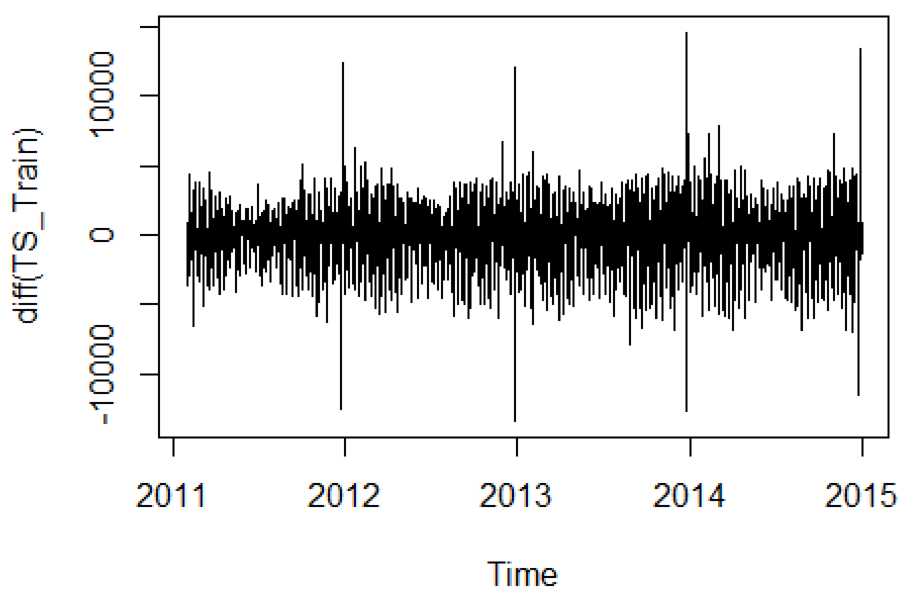


As we can see, the mean is changing, the variance is not constant and a seasonality pattern exists over time in the above figure. These all violate stationary assumptions so, we need to stationarize the data. To stationarize the data, we are going to use $diff()$ and $log10()$ functions. $Diff()$ function will be used for making mean equal over time and $log10()$ function will be used for making the variance equal over time.

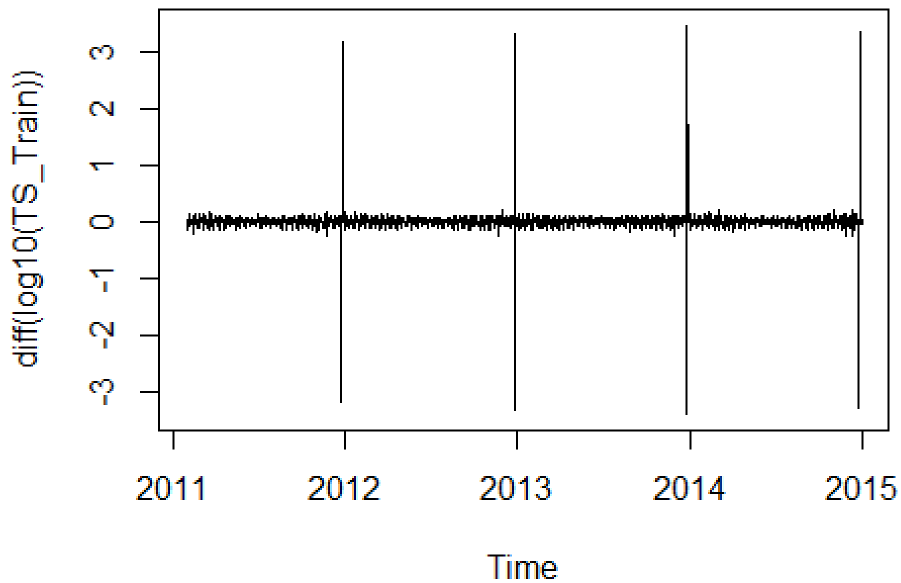
```r
# Make the variance equal
plot(log10(TS_Train))
```




```r
# Make the means equal
plot(diff(TS_Train))
```



```r
# Combination of both functions
plot(diff(log10(TS_Train)))
```



There is a drop in the sale at the end of every year because of Christmas day and we need to keep it for future sales forecasting.

We can use the ADF test to check whether the data is stationary or not.

```r
# The adf test before stationarization
adf.test(TS_Train, k=125)

##
##  Augmented Dickey-Fuller Test
##
## data:  TS_Train
## Dickey-Fuller = -1.7545, Lag order = 125, p-value = 0.6823
## alternative hypothesis: stationary
```

As p-value > 0.05, we can’t reject null hypothesis, so the data is not stationary.

```r
# The adf test after stationarization
adf.test(diff(log10(TS_Train)), k=125)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  diff(log10(TS_Train))
## Dickey-Fuller = -3.6921, Lag order = 125, p-value = 0.02439
## alternative hypothesis: stationary
```

As p-value < 0.05, now we can reject null hypothesis, so the data is stationary.

ACF and PACF used together to identify the order of the ARMA. Also, seasonal ACF and PACF examines correlations for seasonal data.

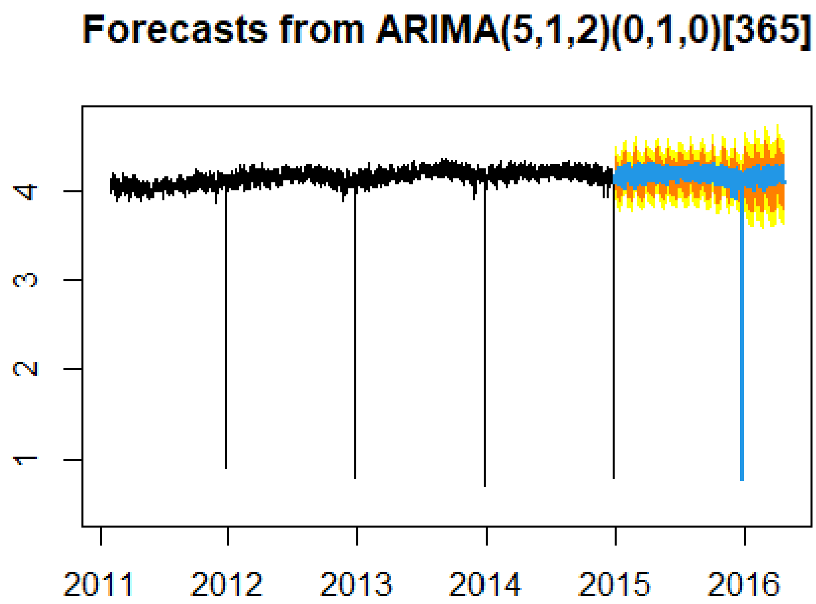
4.4.1 Model Creation

$auto.arima()$ returns best ARIMA model according to either AIC, AICc or BIC value. The function conducts a search over possible model within the order constraints provided.

```r
TS_Train_log = log10(TS_Train)

# Run auto arima
TS_AutoARIMA <- auto.arima(TS_Train_log, seasonal = TRUE)
TS_AutoARIMA
```

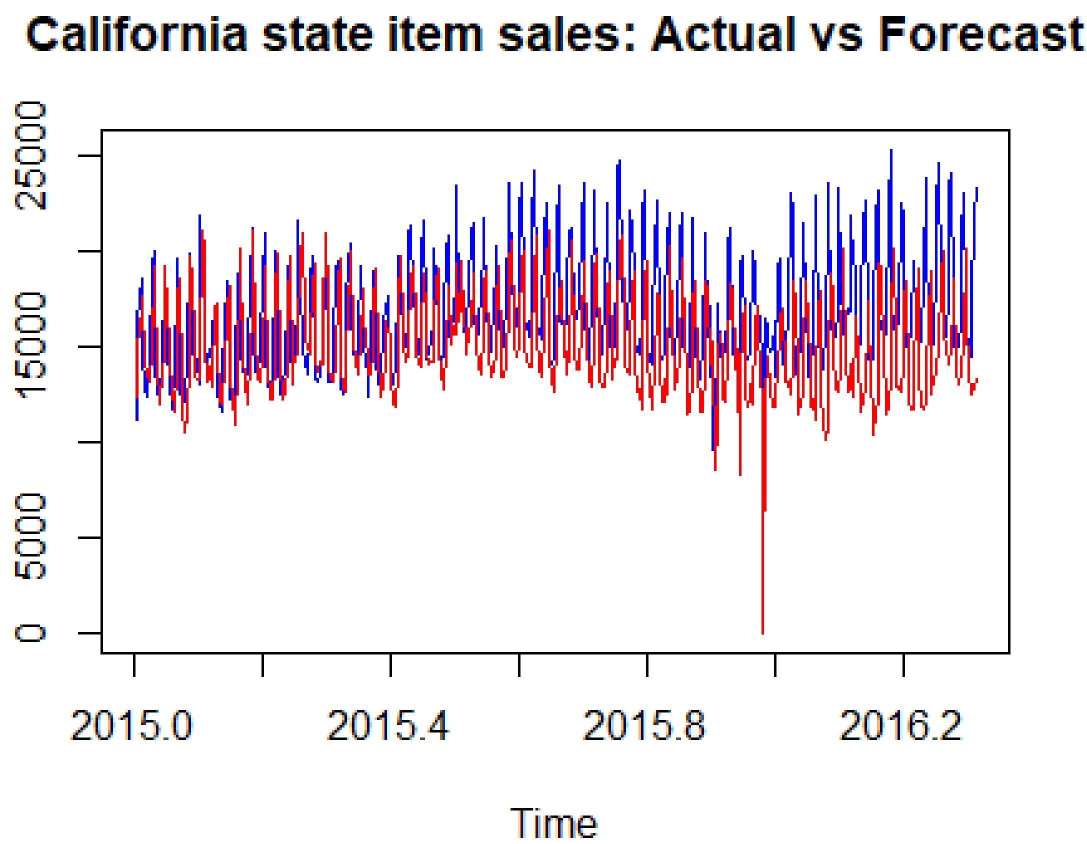
```
## Series: TS_Train_log
## ARIMA(5,1,2)(0,1,0)[365]
##
## Coefficients:
##           ar1      ar2      ar3      ar4      ar5      ma1     ma2
##       -0.1757  -0.2416  -0.2676  -0.2587  -0.2080  -1.3462  0.4038
## s.e.   0.0643   0.0416   0.0362   0.0356   0.0362   0.0609  0.0578
##
## sigma^2 estimated as 0.01862:  log likelihood=608.86
## AIC=-1201.73   AICc=-1201.59   BIC=-1161.95
```

```r
# Creating model
model <- arima(TS_Train_log, c(5, 1, 2), seasonal = list(order = c(0, 1, 0),
period = 365))

## Forecast sales
SalesForecasts <- forecast(model, h=480)
plot(SalesForecasts, shadecols = "oldstyle")
```



4.4.2 Accuracy measures: RMSE and MAPE

```
## Accuracy measures: RMSE and MAPE using ARIMA
Vec1<- 10^(cbind(log10(TS_Test) ,as.data.frame(SalesForecasts)[,1]))
ts.plot(Vec1, col=c("blue", "red"), main="California state item sales: Actual
vs Forecast")
```




```
RMSE1 <- round(sqrt(sum(((Vec1[,1]-Vec1[,2])^2)/length(Vec1[,1]))),4)
MAPE1 <- round(mean(abs(Vec1[,1]-Vec1[,2])/Vec1[,1]),4)
paste("Accuracy Measures: RMSE:", RMSE1, "and MAPE:", MAPE1)

## [1] "Accuracy Measures: RMSE: 3738.4256 and MAPE: 0.1598"
```

Mean Absolute Percentage Error (MAPE) = **15.98%**
Root Mean Square Error (RMSE) = **3738.43**

5. Multi Linear Regression

5.1 Data pre-processing

```
# Joining calendar with nItemSold
ca.df <- cbind(calendar.df, ca.df$nItemSold)
# Renaming the column
colnames(ca.df)[length(ca.df)] <- "nItemSold"
# Remove unnecessary columns
ca.df <- ca.df[, c(-1, -8, -9)]

ca.df$wday <- as.factor(ca.df$wday)
ca.df$snap_CA <- as.factor(ca.df$snap_CA)
```

	wday	event_name_1	event_type_1	event_name_2	event_type_2	snap_CA	nItemSold
1	1	1	1	1	1	0	14195
2	2	1	1	1	1	0	13805
3	3	1	1	1	1	0	10108
4	4	1	1	1	1	1	11047
5	5	1	1	1	1	1	9925
6	6	1	1	1	1	1	11322
7	7	1	1	1	1	1	12251
8	1	1	1	1	1	1	16610
9	2	2	2	1	1	1	14696
10	3	1	1	1	1	1	11822

5.2 Splitting data

We are going to split data into 75% for training and 25% for testing.

R language will take care of dummy variables, so we don't need to encode them.

```
# Split the dataset into training-set and test-set
training.set <- ca.df[1:1433,]
test.set <- ca.df[1434:length(ca.df$nItemSold),]
```

5.3 Model Creation

```
# Fit multiple linear regression to the training-set
regressor <- lm(formula = nItemSold ~ ., data = training.set)
# show summary
summary(regressor)

##
## Call:
```

```
## lm(formula = nItemSold ~ ., data = training.set)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6385.3 -1422.2   311.2  1552.7  6678.3
##
## Coefficients: (6 not defined because of singularities)
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     17122.90     151.56 112.975  < 2e-16 ***
## wday2             377.36     212.94   1.772 0.076589 .
## wday3           -3636.16     212.13 -17.141  < 2e-16 ***
## wday4           -4665.31     207.55 -22.478  < 2e-16 ***
## wday5           -4969.39     209.56 -23.714  < 2e-16 ***
## wday6           -4856.08     209.08 -23.226  < 2e-16 ***
## wday7           -3214.57     207.34 -15.504  < 2e-16 ***
## event_name_12   -2055.68    1059.68  -1.940 0.052592 .
## event_name_13   -2017.37    1048.68  -1.924 0.054593 .
## event_name_14     982.01    1057.25   0.929 0.353133
## event_name_15    -909.71    1056.09  -0.861 0.389169
## event_name_16    -352.76    1056.63  -0.334 0.738543
## event_name_17     -81.43    1048.73  -0.078 0.938121
## event_name_18      11.87    1052.39   0.011 0.991002
## event_name_19    -839.26    2096.50  -0.400 0.688986
## event_name_110   -436.65    1051.78  -0.415 0.678095
## event_name_111  -1588.73    1212.76  -1.310 0.190409
## event_name_112  -2952.99    1056.64  -2.795 0.005266 **
## event_name_113   1285.51    1057.25   1.216 0.224227
## event_name_114    193.09    1050.73   0.184 0.854224
## event_name_115   -153.97    1212.84  -0.127 0.899001
## event_name_116  -1919.93    1217.43  -1.577 0.115015
## event_name_117    565.99    1051.08   0.538 0.590329
## event_name_118    624.56    1048.18   0.596 0.551371
## event_name_119    968.98    1048.02   0.925 0.355346
## event_name_120   3691.35    1059.69   3.483 0.000510 ***
## event_name_121    810.31    1056.75   0.767 0.443336
## event_name_122  -1723.60    1048.69  -1.644 0.100489
## event_name_123    182.40    1048.11   0.174 0.861868
## event_name_124   1082.52    1048.71   1.032 0.302141
## event_name_125  -4119.32    1056.64  -3.899 0.000101 ***
## event_name_126 -13588.29    1048.74 -12.957  < 2e-16 ***
## event_name_127   -232.25    1049.37  -0.221 0.824872
## event_name_128  -4972.70    1212.78  -4.100 4.37e-05 ***
## event_name_129   -659.33    1212.77  -0.544 0.586769
## event_name_130    951.26    1217.34   0.781 0.434684
## event_name_131  -2848.72    1486.36  -1.917 0.055496 .
## event_type_12         NA         NA      NA       NA
## event_type_13         NA         NA      NA       NA
## event_type_14         NA         NA      NA       NA
## event_type_15         NA         NA      NA       NA
## event_name_22   -4940.00    2956.28  -1.671 0.094943 .
```

```
## event_name_23    1083.09    2958.74   0.366 0.714374
## event_name_24    3701.46    2560.92   1.445 0.148582
## event_name_25     150.70    2418.00   0.062 0.950313
## event_type_22         NA         NA      NA       NA
## event_type_23         NA         NA      NA       NA
## snap_CA1         1182.91     120.72   9.799  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2090 on 1391 degrees of freedom
## Multiple R-squared:  0.5627, Adjusted R-squared:  0.5498
## F-statistic: 43.66 on 41 and 1391 DF,  p-value: < 2.2e-16
```

As we can see, the Coefficient of determination or R-square value for multi linear regression is 56.27% and the Adjusted R-squared is 54.98%.

There is a magical function called step() which automatically removes insignificant variables from the model. It is applied as following:

```
reduced.regressor <-step(regressor)

## Start:  AIC=21952.27
## nItemSold ~ wday + event_name_1 + event_type_1 + event_name_2 +
##     event_type_2 + snap_CA
##
##
## Step:  AIC=21952.27
## nItemSold ~ wday + event_name_1 + event_type_1 + event_name_2 +
##     snap_CA
##
##
## Step:  AIC=21952.27
## nItemSold ~ wday + event_name_1 + event_name_2 + snap_CA
##
##                Df  Sum of Sq        RSS   AIC
## - event_name_2  4   29764620 6.1081e+09 21951
## <none>                       6.0784e+09 21952
## - snap_CA       1  419547364 6.4979e+09 22046
## - event_name_1 30 1057615962 7.1360e+09 22122
## - wday          6 6029770280 1.2108e+10 22928
##
## Step:  AIC=21951.27
## nItemSold ~ wday + event_name_1 + snap_CA
##
##                Df  Sum of Sq        RSS   AIC
## <none>                       6.1081e+09 21951
## - snap_CA       1  423403948 6.5315e+09 22045
```

```
## - event_name_1 30 1059735160 7.1679e+09 22121
## - wday          6 6039725659 1.2148e+10 22925
```

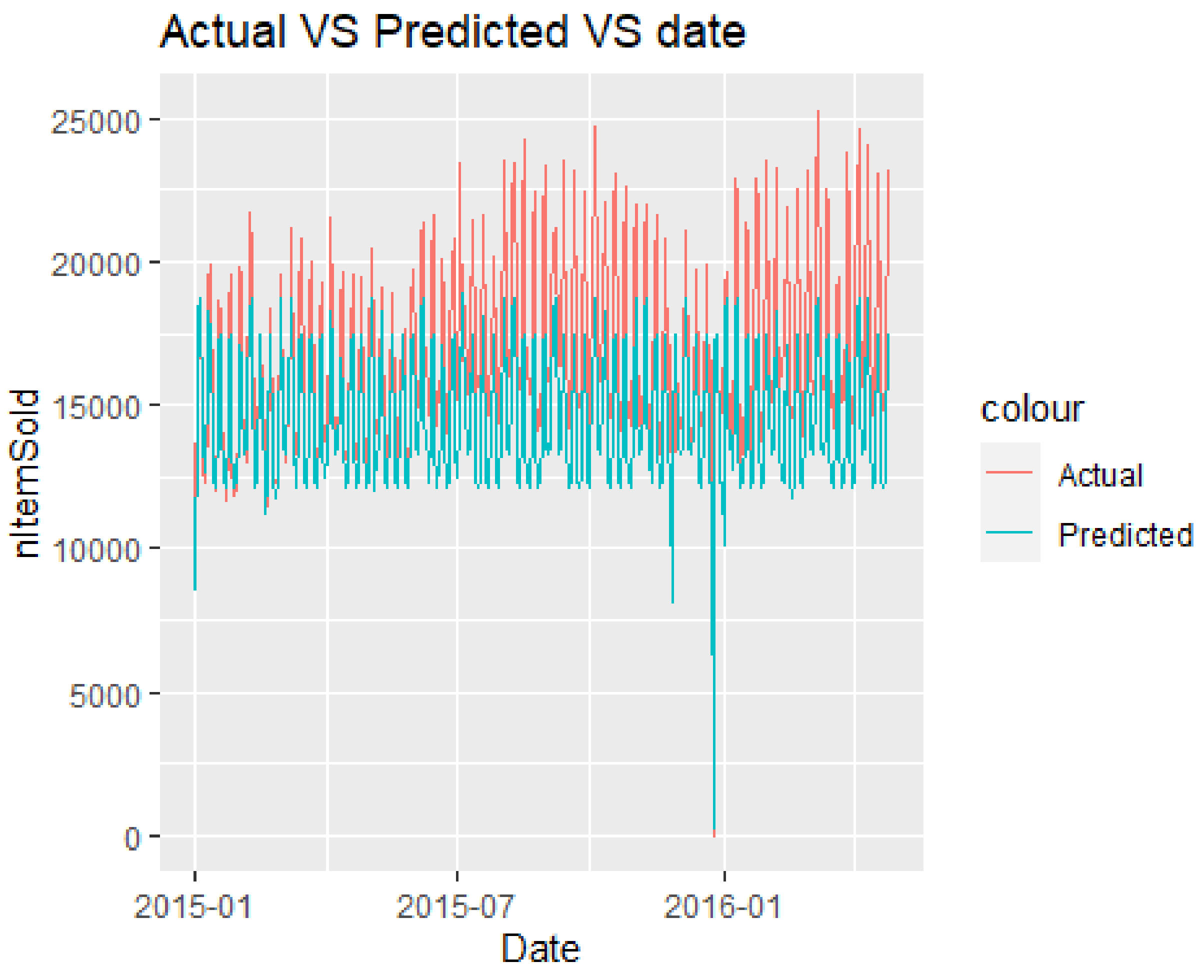
```r
summary(reduced.regressor)
```

```
##
## Call:
## lm(formula = nItemSold ~ wday + event_name_1 + snap_CA, data = training.se
t)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6384.4 -1444.2   313.6  1562.6  6676.1
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     17121.69     151.70 112.864  < 2e-16 ***
## wday2             377.73     213.06   1.773 0.076457 .
## wday3           -3636.17     212.34 -17.124  < 2e-16 ***
## wday4           -4665.33     207.76 -22.456  < 2e-16 ***
## wday5           -4969.42     209.77 -23.690  < 2e-16 ***
## wday6           -4856.52     209.21 -23.214  < 2e-16 ***
## wday7           -3214.58     207.55 -15.488  < 2e-16 ***
## event_name_12   -2058.61    1060.71  -1.941 0.052486 .
## event_name_13   -2016.04    1049.73  -1.921 0.054996 .
## event_name_14     983.23    1058.31   0.929 0.353020
## event_name_15    -910.36    1057.16  -0.861 0.389306
## event_name_16    -351.52    1057.69  -0.332 0.739679
## event_name_17     -80.20    1049.79  -0.076 0.939117
## event_name_18      11.97    1053.44   0.011 0.990934
## event_name_19   -2125.32    1217.98  -1.745 0.081214 .
## event_name_110   -436.36    1052.84  -0.414 0.678602
## event_name_111  -1591.15    1213.98  -1.311 0.190178
## event_name_112  -2953.09    1057.68  -2.792 0.005309 **
## event_name_113   1286.73    1058.31   1.216 0.224253
## event_name_114    192.65    1051.78   0.183 0.854697
## event_name_115   -115.04    1052.63  -0.109 0.912989
## event_name_116  -1919.09    1218.64  -1.575 0.115533
## event_name_117    563.54    1052.13   0.536 0.592304
## event_name_118    623.80    1049.24   0.595 0.552253
## event_name_119    969.28    1049.08   0.924 0.355681
## event_name_120   3688.79    1060.75   3.478 0.000522 ***
## event_name_121    809.63    1057.81   0.765 0.444171
## event_name_122  -1722.26    1049.74  -1.641 0.101094
## event_name_123    181.74    1049.17   0.173 0.862503
## event_name_124   1083.65    1049.77   1.032 0.302121
## event_name_125  -4117.66    1057.69  -3.893 0.000104 ***
## event_name_126 -13587.05    1049.79 -12.943  < 2e-16 ***
## event_name_127   -231.95    1050.43  -0.221 0.825267
## event_name_128  -4975.37    1214.00  -4.098  4.4e-05 ***
```

```
## event_name_129   -661.88    1213.99  -0.545 0.585697
## event_name_130    952.48    1218.56   0.782 0.434558
## event_name_131  -1615.32    1217.98  -1.326 0.184982
## snap_CA1         1186.69     120.68   9.834  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2093 on 1395 degrees of freedom
## Multiple R-squared:  0.5606, Adjusted R-squared:  0.5489
## F-statistic:  48.1 on 37 and 1395 DF,  p-value: < 2.2e-16
```

Similarly, after implying *step*() function Coefficient of determination or R-square value for multi linear regression is 56.06% and Adjusted R-squared is 54.89%.

```
# Predicting the test-set
pred <- predict(reduced.regressor, newdata = test.set)

# preparing data for plotting
data.for.plotting <- data.frame("date" = seq(as.Date("2015/01/01"), as.Date("
2016/04/24"), "day"))
data.for.plotting$actual <- ca.df[1434:1913, length(ca.df)]
data.for.plotting$predicted <- pred

# Show 10 row
head(data.for.plotting, 10)

##          date actual predicted
## 1  2015-01-01  11169  8476.481
## 2  2015-01-02  16179 15093.799
## 3  2015-01-03  17410 18308.377
## 4  2015-01-04  18465 18686.111
## 5  2015-01-05  14833 14672.211
## 6  2015-01-06  12796 13643.046
## 7  2015-01-07  12301 12677.082
## 8  2015-01-08  13088 13451.853
## 9  2015-01-09  13865 15093.799
## 10 2015-01-10  19244 18308.377

ggplot(data.for.plotting, aes(data.for.plotting, x = date)) +
  geom_line(aes(y = actual, colour = "Actual")) +
  geom_line(aes(y = predicted, colour = "Predicted")) +
  labs(title = "Actual VS Predicted VS date", x = "Date", y = "nItemSold")
```



According to above graph, there is a systematic under estimation.

5.4 Accuracy measures: RMSE and MAPE

In order to compare Time Series Models with Multi Linear Regression Model, RMSE and MAPE values are also calculated for Regression model.

```
## Accuracy measures: RMSE and MAPE
RMSE1 <- round(sqrt(sum(((data.for.plotting[,2]-data.for.plotting[,3])^2)/len
gth(data.for.plotting[,2]))),4)
MAPE1 <- round(mean(abs(data.for.plotting[,2]-data.for.plotting[,3])/data.for
.plotting[,2]),4)
paste("Accuracy Measures: RMSE:", RMSE1, "and MAPE:", MAPE1)

## [1] "Accuracy Measures: RMSE: 2705.0211 and MAPE: 0.2392"
```

Mean Absolute Percentage Error (MAPE) = **23.92%**
Root Mean Square Error (RMSE) = **2705.02**

6. Models Comparison

Comparison table

Model	RMSE	MAPE
Random Walk with Drift	4083.11	18.82%
ARIMA	3738.43	15.98%
Multi-linear Regression	2705.02	23.92%

Now, using the above comparison table, we are comparing results obtained from Random walk with drift, ARIMA, and Multi-Linear Regression models. It is concluded that the Autoregressive integrated moving average (ARIMA) method predicts well as compared to other models. As the value of MAPE measure is minimum though the value of RMSE value has higher.