Prediction on test data

Mark Wang and Paola Aleman

9/30/2019

# Instructions

This markdown file is used to make predictions based on the whole existing dataset, as well as the four best-performing algorithms identified on the existing dataset.  
This markdown should be used along with the training markdown. Existing data shall be processed and models trained using the training markdown. Then this markdown file can be applied.  
Note the models are label by “c1k”. which stands for c -> colombia, 1 -> club 6101 and k -> kitchen trash bags.  
It is necessary to change “c1k” whenever you’re analyzing a different set of item and club. This way graphs and models are saved into folders with their specific name.  
For easier substitution ctrl+f. In the first box type the name you want to substitute, for example “c1k”. and on the second box include the prefix you want to substitute with, for example “c2k”. Then click the last All option.

# Installing packages

# Prepare data

knitr::opts\_knit$set(root.dir = normalizePath("C:/Users/mwang/Desktop/Forecast"))  
setwd("C:/Users/mwang/Desktop/Forecast")  
getwd()

## [1] "C:/Users/mwang/Desktop/Forecast"

len\_dec<-1 #The frequency in which buyers make purchase decisions   
len\_inv<-17 #The time span from purchase to selling of an item   
#Item, club and country we are analyzing   
item <- "Kitchen Trash Bag"  
club <- 6101  
country <- "Colombia"  
load(paste0("Output/Countries/Colombia/", club, ".RData"))  
c1k\_prediction<-data.frame(TransactionTime=(as.Date(c1k\_week$TransactionTime[nrow(c1k\_week)])+7\*(1:(len\_dec+len\_inv))), quantity\_avrg=NA)

# Identify seasons

The periodogram function does not detect high frequency seasons accurately. To solve this problem, one of the built-in functions need to be modified.  
Run the following line in r (which is included in this chunk of code): trace(spec.pgram, edit=TRUE) Then change line 9 to: N <- N0 <- nrow(x) \* 128 Then click “save” The function is hence temporarily changed.

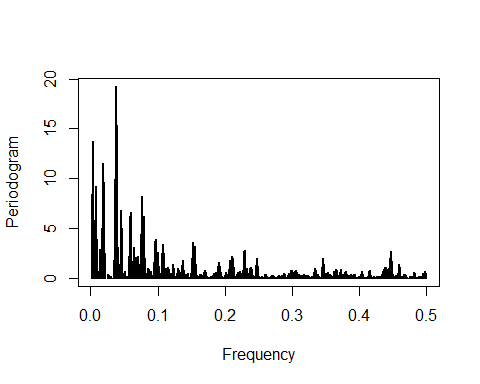
Similar to in the training markdown, the identification of seasons here is not completely automated. Personal judgement is still needed.  
Firstly, 52.14, the number of weeks per year, is always included.  
Secondly, only periods shorter than 52.14 should be included, and their length should be adjusted to its closest whole-month length. For example, if 13.02 is reported is a strong period by Fourier transformation, 12.53 (three months) should be the season used in analyses.  
Thirdly, none period should be multiple of the other (otherwise Fourier regressor does not work). Therefore, half-year period is changed from 26.07 to 25.07 weeks. We believe better ways to deal with this problem exist.

trace(spec.pgram, edit=TRUE)

## Tracing function "spec.pgram" in package "stats"

## [1] "spec.pgram"

p = periodogram(c1k\_week$quantity\_avrg)



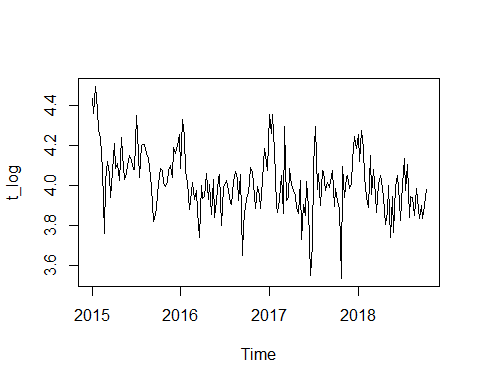
dd = data.frame(freq=p$freq, spec=p$spec)  
order = dd[order(-dd$spec),]  
  
order<-order[(1/order)<60,]  
(1/(order$freq))%>%head(10)

## [1] 26.40909 25.26087 52.81818 27.66667 48.41667 12.91111 22.34615  
## [8] 16.60000 17.08824 12.63043

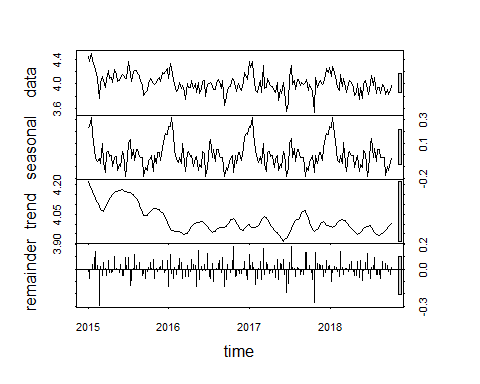
season1<-25.07  
season2<-52.14  
season3<-12.53

# Creating time series

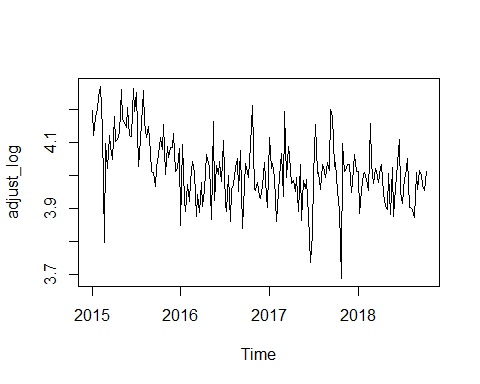
t\_log<-ts(log(c1k\_week$quantity\_avrg), frequency = 52.14, start = c(c1k\_week$week\_number\_year[1], c1k\_week$week\_number[1]))   
  
plot(t\_log)



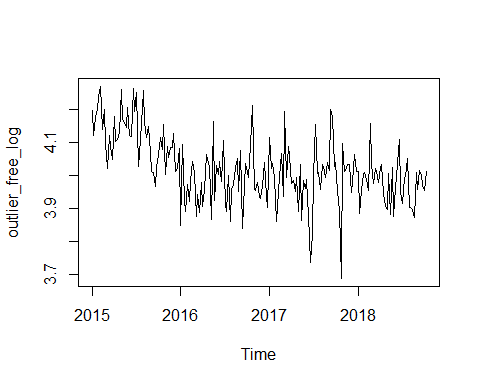
decompose\_log <- stl(t\_log, s.window = 13, t.window = 13)  
plot(decompose\_log)



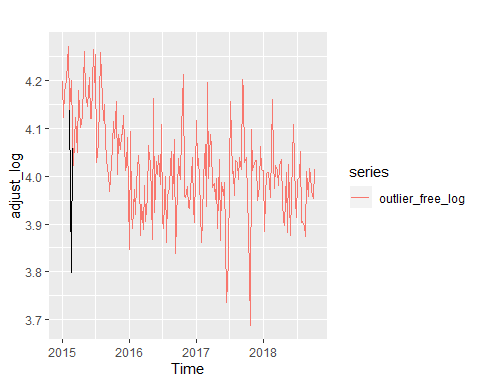
adjust\_log<- t\_log - decompose\_log$time.series[,1] # deseasonalize data   
  
outlier\_free\_log<- tsclean(adjust\_log)  
trend\_log<- decompose\_log$time.series[, 2]  
detrend\_ts\_log <- outlier\_free\_log-(trend\_log - trend\_log[1]) # corrected detrending part   
  
plot(adjust\_log)



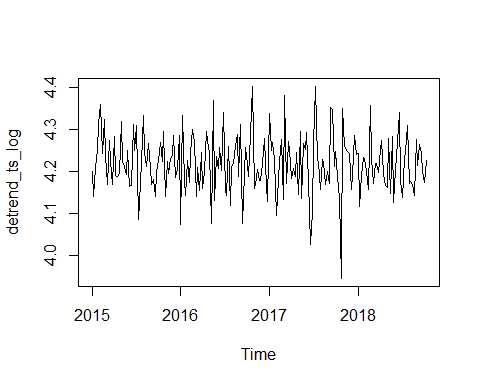
plot(outlier\_free\_log)



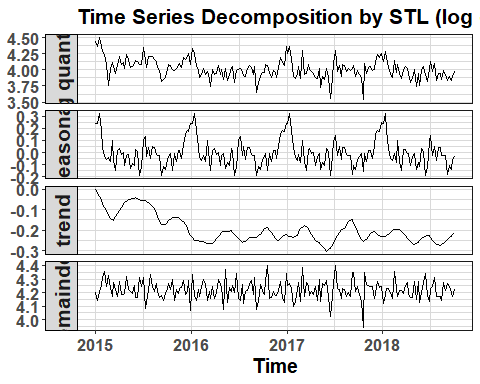
autoplot(adjust\_log)+autolayer(outlier\_free\_log)



plot(detrend\_ts\_log)

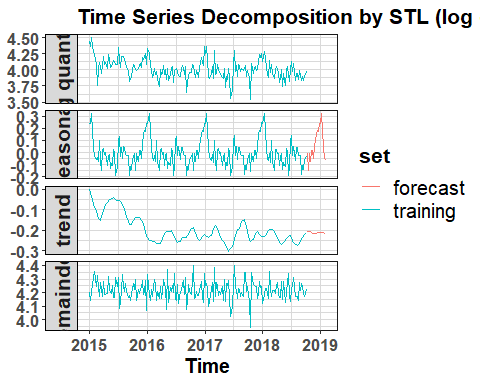


##Training Set: uni-variate methods  
   
training\_log<-ts(detrend\_ts\_log, frequency = 52.14, start = c(c1k\_week$week\_number\_year[1], c1k\_week$week\_number[1]))   
  
##future trend and seasonality, which will be added back:  
  
trend\_fit\_log <- auto.arima(trend\_log)  
trend\_for\_log <- forecast(trend\_fit\_log, len\_inv+len\_dec)$mean  
retrend\_log<-trend\_for\_log  
reseasonal\_log<-forecast(decompose\_log$time.series[,1], len\_inv+len\_dec)$mean  
  
##data visualization  
theme\_ts <- theme(panel.border = element\_rect(fill = NA,   
 colour = "grey10"),  
 panel.background = element\_blank(),  
 panel.grid.minor = element\_line(colour = "grey85"),  
 panel.grid.major = element\_line(colour = "grey85"),  
 panel.grid.major.x = element\_line(colour = "grey85"),  
 axis.text = element\_text(size = 13, face = "bold"),  
 axis.title = element\_text(size = 15, face = "bold"),  
 plot.title = element\_text(size = 16, face = "bold"),  
 strip.text = element\_text(size = 16, face = "bold"),  
 strip.background = element\_rect(colour = "black"),  
 legend.text = element\_text(size = 15),  
 legend.title = element\_text(size = 16, face = "bold"),  
 legend.background = element\_rect(fill = "white"),  
 legend.key = element\_rect(fill = "white"))  
  
decomp\_stl\_log<- data.table(Quant = c(t\_log, decompose\_log$time.series[, 1], decompose\_log$time.series[, 2]-decompose\_log$time.series[, 2][1], detrend\_ts\_log),  
 Date = rep(c1k\_week$TransactionTime, ncol(decompose\_log$time.series)+1),  
 Type = factor(rep(c("log quantity", colnames(decompose\_log$time.series)),  
 each = nrow(decompose\_log$time.series)),  
 levels = c("log quantity", colnames(decompose\_log$time.series))))  
  
ggplot(decomp\_stl\_log, aes(x = Date, y = Quant)) +  
 geom\_line() +   
 facet\_grid(Type ~ ., scales = "free\_y", switch = "y") +  
 labs(x = "Time", y = NULL,  
 title = "Time Series Decomposition by STL (log quantity)") +  
 theme\_ts



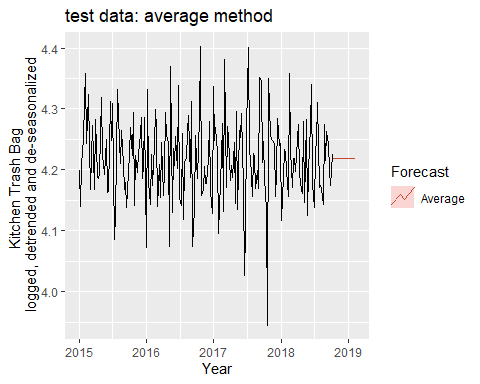
decomp\_stl\_training\_log<- decomp\_stl\_log  
decomp\_stl\_training\_log$set<-"training"  
  
decomp\_stl\_forecast\_log<-data.table(Quant=c(rep(NA, len\_dec+len\_inv), reseasonal\_log, retrend\_log-trend\_log[1], rep(NA, len\_dec+len\_inv)),   
 Date = rep(c1k\_prediction$TransactionTime, ncol(decompose\_log$time.series)+1),   
 Type = factor(rep(c("log quantity", colnames(decompose\_log$time.series)),  
 each = len\_dec+len\_inv),  
 levels = c("log quantity", colnames(decompose\_log$time.series))))  
decomp\_stl\_forecast\_log$set<-"forecast"  
decomp\_stl\_combined\_log<-rbind(decomp\_stl\_training\_log, decomp\_stl\_forecast\_log)  
  
ggplot(decomp\_stl\_combined\_log, aes(x = Date, y = Quant, col=set)) +  
 geom\_line() +   
 facet\_grid(Type ~ ., scales = "free\_y", switch = "y") +  
 labs(x = "Time", y = NULL,  
 title = "Time Series Decomposition by STL (log quantity)") +  
 theme\_ts

## Warning: Removed 36 rows containing missing values (geom\_path).



# Simple model: Average method

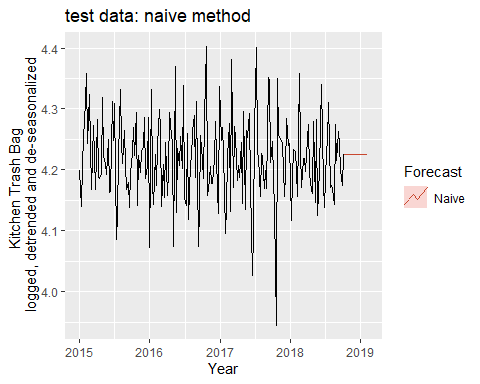
Average\_Method<-meanf(training\_log, h = len\_dec+len\_inv)  
autoplot(training\_log) +  
 autolayer(Average\_Method,  
 series = "Average", PI = FALSE)+  
 ggtitle("test data: average method") +  
 xlab("Year") + ylab( paste(item, "\n logged, detrended and de-seasonalized"))+  
 guides(colour=guide\_legend(title="Forecast"))



c1k\_prediction$Average\_Method<-exp(Average\_Method$mean+(retrend\_log-trend\_log[1])+reseasonal\_log)

# Simple model: Naive method

Naive\_Method<-naive(training\_log, h = len\_dec+len\_inv)  
autoplot(training\_log) +  
 autolayer(Naive\_Method,  
 series = "Naive", PI = FALSE)+  
 ggtitle("test data: naive method") +  
 xlab("Year") + ylab( paste(item, "\n logged, detrended and de-seasonalized"))+  
 guides(colour=guide\_legend(title="Forecast"))



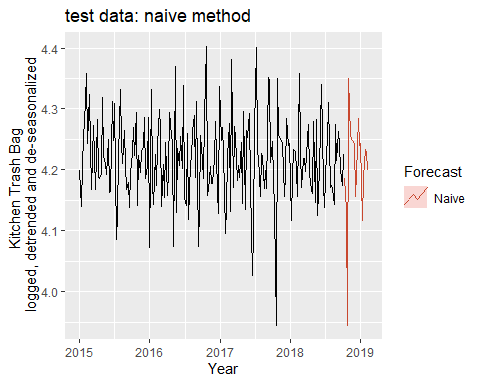
c1k\_prediction$Naive\_Method<-exp(Naive\_Method$mean+(retrend\_log-trend\_log[1])+reseasonal\_log)

# Simple model: Seasonal Naive method

Seasonal\_Naive\_Method<-snaive(training\_log, h = len\_dec+len\_inv)

## Warning in lag.default(y, -lag): 'k' is not an integer

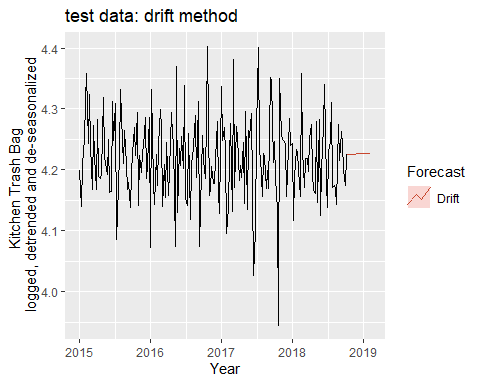
autoplot(training\_log) +  
 autolayer(Seasonal\_Naive\_Method,  
 series = "Naive", PI = FALSE)+  
 ggtitle("test data: naive method") +  
 xlab("Year") + ylab( paste(item, "\n logged, detrended and de-seasonalized"))+  
 guides(colour=guide\_legend(title="Forecast"))



c1k\_prediction$Seasonal\_Naive\_Method<-exp(Seasonal\_Naive\_Method$mean+(retrend\_log-trend\_log[1])+reseasonal\_log)

# Simple modeol: Drift method

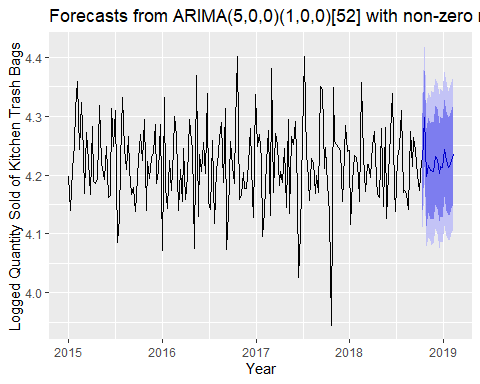
# Drift method  
Drift\_Method<-rwf(training\_log, h = len\_dec+len\_inv, drift = TRUE)  
  
autoplot(training\_log) +  
 autolayer(Drift\_Method,  
 series = "Drift", PI = FALSE)+  
 ggtitle("test data: drift method") +  
 xlab("Year") + ylab( paste(item, "\n logged, detrended and de-seasonalized"))+  
 guides(colour=guide\_legend(title="Forecast"))



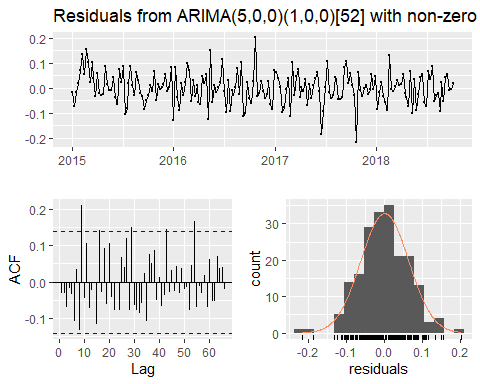
c1k\_prediction$Drift\_Method<-exp(Drift\_Method$mean+(retrend\_log-trend\_log[1])+reseasonal\_log)

# Arima: auto.arima

Simple\_Arima<- auto.arima(training\_log)  
fc\_Simple\_Arima\_1<- forecast(Simple\_Arima, len\_dec+len\_inv, bootstrap = TRUE)  
  
autoplot(fc\_Simple\_Arima\_1) +  
 xlab("Year") + ylab("Logged Quantity Sold of Kitchen Trash Bags")+  
 guides(colour=guide\_legend(title="Validation Set"))



cr\_Simple\_Arima\_1<-checkresiduals(fc\_Simple\_Arima\_1)



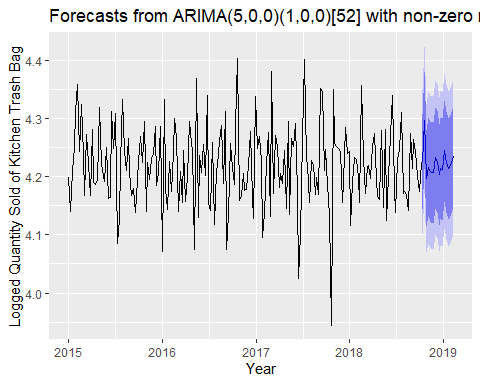
##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(5,0,0)(1,0,0)[52] with non-zero mean  
## Q\* = 60.37, df = 32, p-value = 0.001766  
##   
## Model df: 7. Total lags used: 39

c1k\_prediction$Simple\_Arima\_1<-exp(fc\_Simple\_Arima\_1$mean+(retrend\_log-trend\_log[1])+reseasonal\_log)

# ARIMA double season

This section makes Arima predictions based on seasons 1 and 2

#this arima model uses fourier and the three seasonality periods obtained above.  
  
Arima\_AIC <- auto.arima(training\_log)  
bestfit <- list(aicc=Arima\_AIC$aicc, i=0, j=0, Arima\_Seasons1\_2=Arima\_AIC)  
  
fc\_ARIMA\_fourier<- forecast(Arima\_AIC, h = len\_dec+len\_inv)  
autoplot(fc\_ARIMA\_fourier) +  
 xlab("Year") + ylab(paste("Logged Quantity Sold of", item))+  
 guides(colour=guide\_legend(title="Validation Set"))



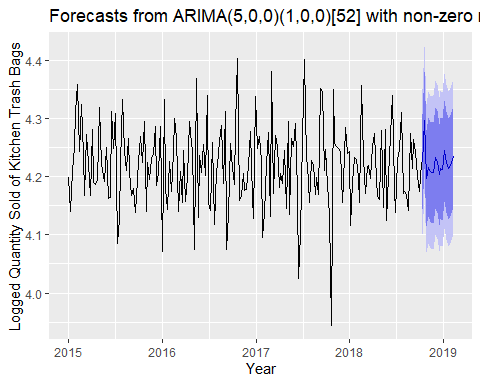
for(i in 1:3) {  
 for (j in 1:3){  
 z1 <- fourier(ts(training\_log, frequency= season1), K=i)  
 z2 <- fourier(ts(training\_log, frequency= season2), K=j)  
 Arima\_Seasons1\_2 <- auto.arima(training\_log, xreg=cbind(z1, z2), seasonal=F)  
 if(Arima\_Seasons1\_2$aicc < bestfit$aicc) {  
 bestfit <- list(aicc=Arima\_Seasons1\_2$aicc, i=i, j=j, Arima\_Seasons1\_2=Arima\_Seasons1\_2)  
 }  
 }  
}  
bestfit

## $aicc  
## [1] -513.2707  
##   
## $i  
## [1] 0  
##   
## $j  
## [1] 0  
##   
## $Arima\_Seasons1\_2  
## Series: training\_log   
## ARIMA(5,0,0)(1,0,0)[52] with non-zero mean   
##   
## Coefficients:  
## ar1 ar2 ar3 ar4 ar5 sar1 mean  
## -0.174 -0.1434 -0.2679 -0.2151 -0.1803 -0.2635 4.2184  
## s.e. 0.070 0.0705 0.0674 0.0696 0.0701 0.0770 0.0019  
##   
## sigma^2 estimated as 0.004031: log likelihood=265.02  
## AIC=-514.04 AICc=-513.27 BIC=-487.77

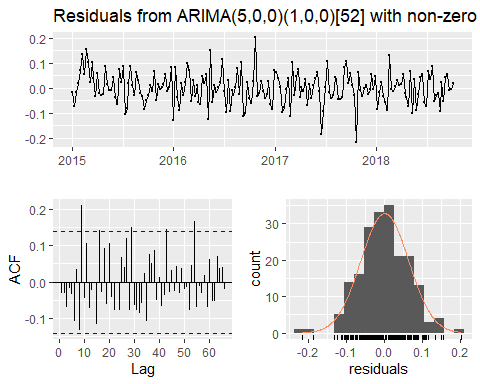
fc\_Arima\_Seasons1\_2 <- forecast(bestfit$Arima\_Seasons1\_2, h=len\_dec+len\_inv,  
 xreg=cbind(  
 fourier(ts(training\_log, frequency=season1), K=bestfit$i, h=len\_dec+len\_inv),  
 fourier(ts(training\_log, frequency=season2), K=bestfit$j, h=len\_dec+len\_inv)))

## Warning in forecast.Arima(bestfit$Arima\_Seasons1\_2, h = len\_dec +  
## len\_inv, : xreg not required by this model, ignoring the provided  
## regressors

autoplot(fc\_Arima\_Seasons1\_2) +  
 xlab("Year") + ylab("Logged Quantity Sold of Kitchen Trash Bags")+  
 guides(colour=guide\_legend(title="Validation Set"))



cr\_ARIMA\_seasons1\_2<-checkresiduals(fc\_Arima\_Seasons1\_2)



##   
## Ljung-Box test  
##   
## data: Residuals from ARIMA(5,0,0)(1,0,0)[52] with non-zero mean  
## Q\* = 60.37, df = 32, p-value = 0.001766  
##   
## Model df: 7. Total lags used: 39

c1k\_prediction$Arima\_Seasons1\_2<-exp(fc\_Arima\_Seasons1\_2$mean+(retrend\_log-trend\_log[1])+reseasonal\_log)

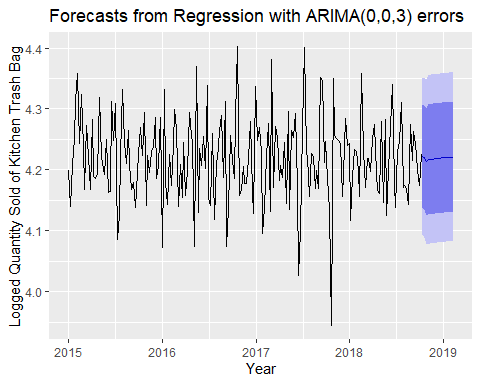
# ARIMA single season

This section uses grid to tune the K parameter to get the Fourier regressor

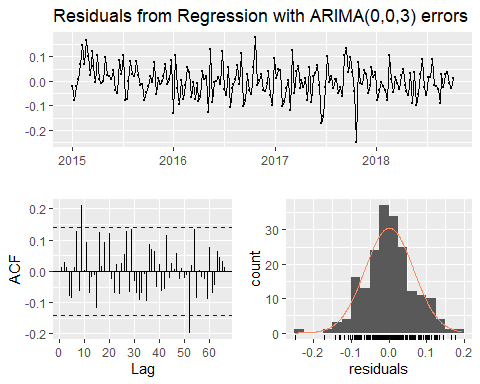
Arima\_Fourier\_AIC<-list(aicc=Inf)  
for(K in seq(25)) {  
 fit <- auto.arima(training\_log, xreg=fourier(training\_log, K=K),  
 seasonal=FALSE)  
 if(fit[["aicc"]] < Arima\_Fourier\_AIC[["aicc"]]) {  
 Arima\_Fourier\_AIC <- fit  
 bestK <- K  
 }  
}  
Arima\_Fourier\_AIC

## Series: training\_log   
## Regression with ARIMA(0,0,3) errors   
##   
## Coefficients:  
## ma1 ma2 ma3 intercept S1-52 C1-52  
## -0.2219 -0.2341 -0.2569 4.2180 0.0028 0.0013  
## s.e. 0.0699 0.0695 0.0639 0.0014 0.0023 0.0024  
##   
## sigma^2 estimated as 0.004264: log likelihood=260.78  
## AIC=-507.56 AICc=-506.97 BIC=-484.58

fc\_Arima\_Fourier\_AIC <- forecast(Arima\_Fourier\_AIC,xreg=fourier(training\_log, K=bestK, h=len\_dec+len\_inv))  
autoplot(fc\_Arima\_Fourier\_AIC)+  
 xlab("Year") + ylab(paste("Logged Quantity Sold of", item))+  
 guides(colour=guide\_legend(title="Validation Set"))



cr\_ARIMA\_Fourier\_AIC<-checkresiduals(fc\_Arima\_Fourier\_AIC)

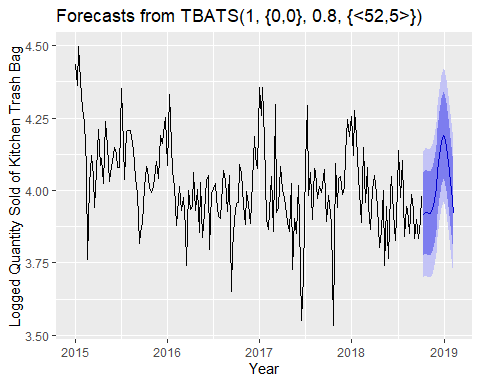


##   
## Ljung-Box test  
##   
## data: Residuals from Regression with ARIMA(0,0,3) errors  
## Q\* = 54.934, df = 33, p-value = 0.009633  
##   
## Model df: 6. Total lags used: 39

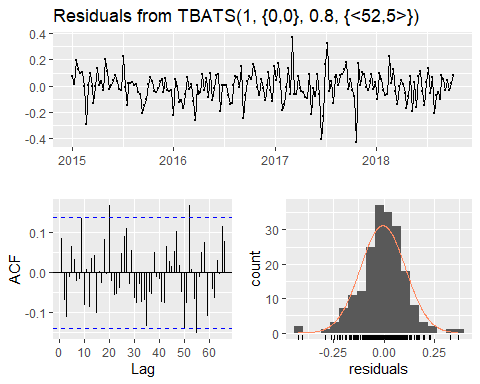
c1k\_prediction$Arima\_Fourier\_AIC<-exp(fc\_Arima\_Fourier\_AIC$mean+(retrend\_log-trend\_log[1])+reseasonal\_log)

# TBATS

# Uses a combination of Fourier terms with an exponential smoothing state space model and a Box-Cox transformation. Seasonality is allowed to change slowly over time.  
# Raw\_1  
# Feed Raw Data to this model [with seasonality, trend and outliers]  
fit\_TBATS\_Raw\_1<- tbats(t\_log, use.box.cox = NULL, use.trend = TRUE, use.damped.trend = NULL, seasonal.periods = 52, use.arma.errors = TRUE, biasadj = TRUE)  
  
  
fc\_TBATS\_Raw\_1<- forecast(fit\_TBATS\_Raw\_1, h=len\_dec+len\_inv, bootstrap = TRUE)   
  
autoplot(fc\_TBATS\_Raw\_1) +  
 xlab("Year") + ylab(paste("Logged Quantity Sold of", item))+  
 guides(colour=guide\_legend(title="Validation Set"))

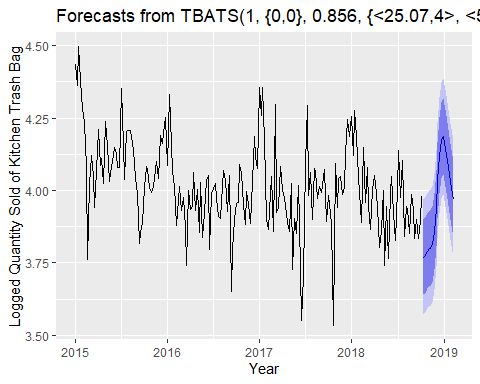


checkresiduals(fc\_TBATS\_Raw\_1)

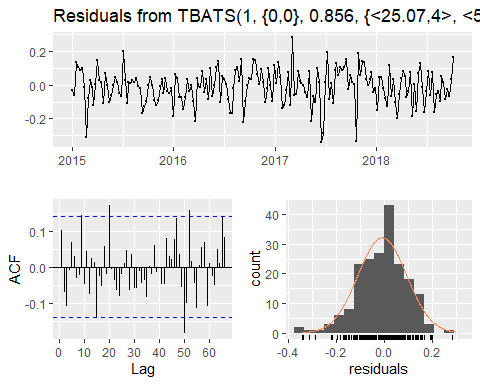


##   
## Ljung-Box test  
##   
## data: Residuals from TBATS(1, {0,0}, 0.8, {<52,5>})  
## Q\* = 43.777, df = 22, p-value = 0.003781  
##   
## Model df: 17. Total lags used: 39

c1k\_prediction$fc\_TBATS\_Raw\_1<-exp(fc\_TBATS\_Raw\_1$mean)  
  
# TBATS Model with top 3 seasonal periods   
  
## Seasons 1 and 2  
fc\_TBATS\_Season1\_2 <- forecast(tbats(t\_log, use.box.cox = NULL, use.trend = NULL, use.damped.trend = NULL, seasonal.periods = c(season2,season1), use.arma.errors = TRUE, biasadj = TRUE), h=len\_dec+len\_inv, bootstrap = TRUE)  
autoplot(fc\_TBATS\_Season1\_2)+  
 xlab("Year") + ylab(paste("Logged Quantity Sold of", item))+  
 guides(colour=guide\_legend(title="Validation Set"))

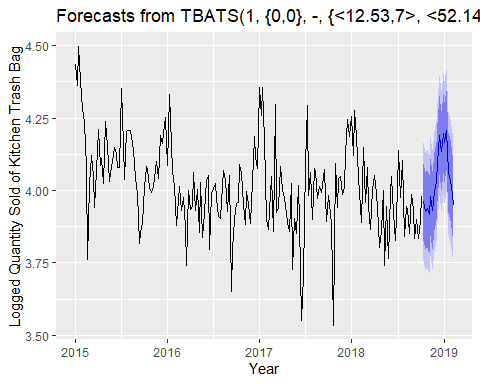


checkresiduals(fc\_TBATS\_Season1\_2)

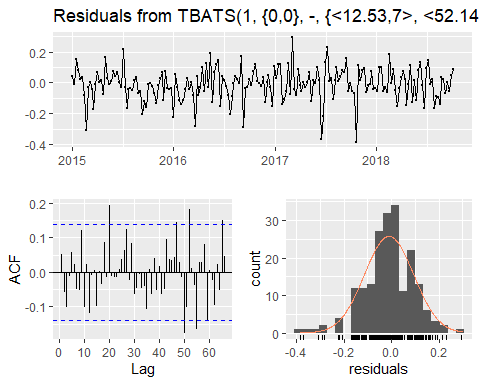


##   
## Ljung-Box test  
##   
## data: Residuals from TBATS(1, {0,0}, 0.856, {<25.07,4>, <52.14,6>})  
## Q\* = 35.242, df = 10, p-value = 0.0001135  
##   
## Model df: 29. Total lags used: 39

c1k\_prediction$fc\_TBATS\_Season1\_2<-exp(fc\_TBATS\_Season1\_2$mean)  
  
## Seasons 2 and 3  
fc\_TBATS\_Season2\_3<- forecast(tbats(t\_log, use.box.cox = NULL, use.trend = NULL, use.damped.trend = NULL, seasonal.periods = c(season2,season3), use.arma.errors = TRUE, biasadj = TRUE),h=len\_dec+len\_inv)   
autoplot(fc\_TBATS\_Season2\_3)+ xlab("Year") + ylab(paste("Logged Quantity Sold of", item))+  
 guides(colour=guide\_legend(title="Validation Set"))

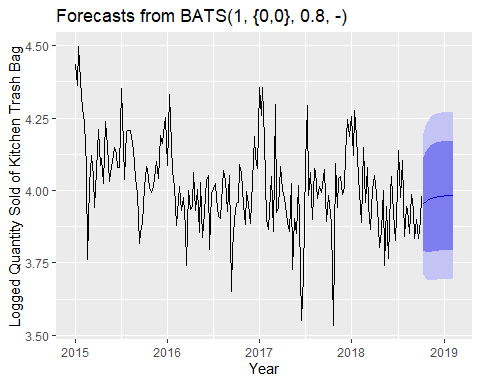


checkresiduals(fc\_TBATS\_Season2\_3)

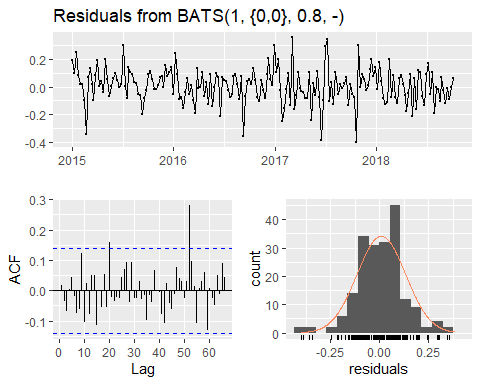


##   
## Ljung-Box test  
##   
## data: Residuals from TBATS(1, {0,0}, -, {<12.53,7>, <52.14,5>})  
## Q\* = 40.55, df = 9, p-value = 6.038e-06  
##   
## Model df: 30. Total lags used: 39

c1k\_prediction$fc\_TBATS\_Season2\_3<-exp(fc\_TBATS\_Season2\_3$mean)  
  
## Seasons 1 and 3  
fc\_TBATS\_Season1\_3 <- forecast(tbats(t\_log, use.box.cox = NULL, use.trend = NULL, use.damped.trend = NULL, seasonal.periods = c(season1,season3), use.arma.errors = TRUE, biasadj = TRUE), h=len\_dec+len\_inv, bootstrap = TRUE)  
autoplot(fc\_TBATS\_Season1\_3)+  
 xlab("Year") + ylab(paste("Logged Quantity Sold of", item))+  
 guides(colour=guide\_legend(title="Validation Set"))

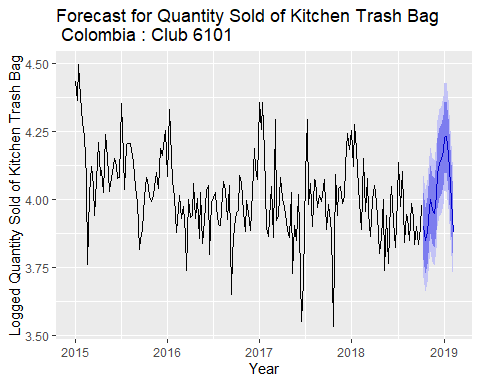


checkresiduals(fc\_TBATS\_Season1\_3)

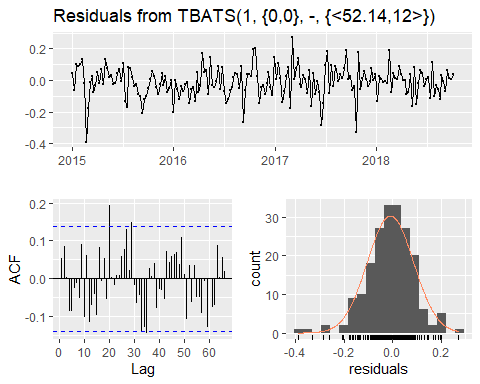


##   
## Ljung-Box test  
##   
## data: Residuals from BATS(1, {0,0}, 0.8, -)  
## Q\* = 33.088, df = 34, p-value = 0.5122  
##   
## Model df: 5. Total lags used: 39

c1k\_prediction$fc\_TBATS\_Season1\_3<-exp(fc\_TBATS\_Season1\_3$mean)  
  
# really simple tbats  
fit\_TBATS\_NoSeason<- tbats(t\_log)  
fc\_TBATS\_NoSeason<- forecast(fit\_TBATS\_NoSeason, h=len\_dec+len\_inv)  
  
autoplot(fc\_TBATS\_NoSeason)+  
 xlab("Year") + ylab(paste("Logged Quantity Sold of", item))+  
 ggtitle(paste("Forecast for Quantity Sold of", item, "\n", country, ": Club", club)) +  
 guides(colour=guide\_legend(title="Validation Set"))



checkresiduals(fc\_TBATS\_NoSeason)

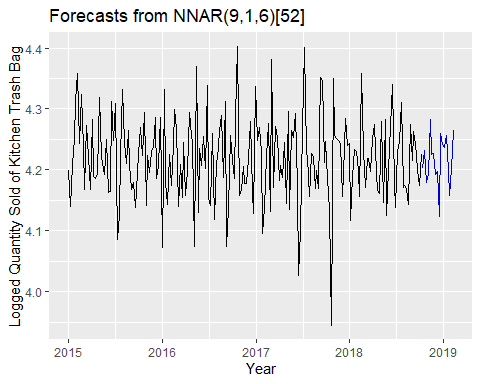


##   
## Ljung-Box test  
##   
## data: Residuals from TBATS(1, {0,0}, -, {<52.14,12>})  
## Q\* = 57.086, df = 11, p-value = 3.209e-08  
##   
## Model df: 28. Total lags used: 39

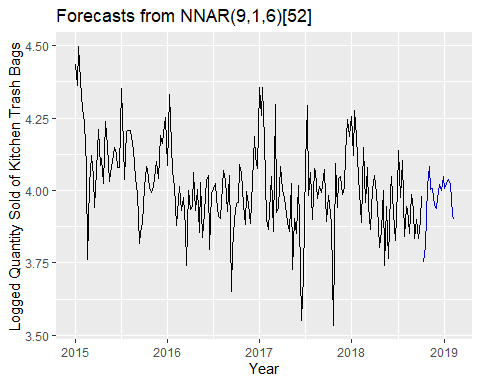
c1k\_prediction$fc\_TBATS\_NoSeason<-exp(fc\_TBATS\_NoSeason$mean)

# Neural network

#Neural Networks with detrneded and de-seasonalized data  
#NNAR(p,P,k)m -> p = lagged inputs, P = equivalent to ARIMA(p,0,0)(P,0,0)m, k = nods in the single hidden layer   
fit\_NN\_1<- nnetar(training\_log, lambda = "auto")  
fc\_NN\_1<- forecast(fit\_NN\_1, h=len\_dec+len\_inv)  
autoplot(fc\_NN\_1) + xlab("Year") + ylab(paste("Logged Quantity Sold of", item))



c1k\_week\_quantity\_models<-performance\_index\_dtds(df\_Actual = c1k\_week\_quantity\_models, pred\_name = "fc\_NN\_1", pred\_value = fc\_NN\_1$mean)  
c1k\_prediction$fc\_NN\_1<-exp(fc\_NN\_1$mean+(retrend\_log-trend\_log[1])+reseasonal\_log)  
  
# neural network with raw data  
fit\_NN\_Raw<- nnetar(t\_log, lambda = "auto")  
fc\_NN\_Raw<- forecast(fit\_NN\_Raw, h=len\_dec+len\_inv)  
autoplot(fc\_NN\_Raw) +  
 xlab("Year") + ylab("Logged Quantity Sold of Kitchen Trash Bags")



c1k\_prediction$fc\_NN\_Raw<-exp(fc\_NN\_Raw$mean)

# Machine learning setup

# feature engineering  
c1k\_week$month<-month(c1k\_week$TransactionTime)  
c1k\_week$month<-as.factor(c1k\_week$month)  
c1k\_week$log1p\_transaction\_avrg<-log1p(c1k\_week$transaction\_avrg) #  
c1k\_week$log1p\_members\_avrg<-log1p(c1k\_week$members\_avrg) #  
c1k\_week$log1p\_sales\_local\_avrg<-log1p(c1k\_week$sales\_local\_avrg) #  
c1k\_week$log1p\_sales\_usd\_avrg<-log1p(c1k\_week$sales\_usd\_avrg) #  
c1k\_week$log1p\_category\_sales\_local\_avrg<-log1p(c1k\_week$category\_sales\_local\_avrg) #  
c1k\_week$log1p\_quantity\_avrg<-log1p(c1k\_week$quantity\_avrg) #  
c1k\_week$log1p\_category\_sales\_local\_avrg<- log1p(c1k\_week$category\_sales\_local\_avrg)  
c1k\_week$log1p\_category\_sales\_usd\_avrg<-log1p(c1k\_week$category\_sales\_usd\_avrg)  
c1k\_week$log1p\_category\_quantity\_avrg<- log1p(c1k\_week$category\_quantity\_avrg)  
c1k\_week$log1p\_salePrice\_local\_avrg<- log1p(c1k\_week$salePrice\_local\_avrg)  
c1k\_week$log1p\_salePrice\_usd\_avrg<- log1p(c1k\_week$salePrice\_usd\_avrg)  
c1k\_week\_prediction<-data.frame(matrix(nrow=len\_dec+len\_inv, ncol = ncol(c1k\_week)))  
colnames(c1k\_week\_prediction)<-colnames(c1k\_week)  
c1k\_week\_prediction$TransactionTime<-c1k\_prediction$TransactionTime  
c1k\_week\_prediction$week\_number<-c1k\_week\_prediction$TransactionTime%>%date2ISOweek()%>%substr(7,8)  
c1k\_week\_prediction$week\_number\_year<-c1k\_week\_prediction$TransactionTime%>%date2ISOweek()%>%substr(1,4)  
c1k\_week\_prediction$month<-month(c1k\_week\_prediction$TransactionTime)  
  
t\_v\_c1k<-rbind(as.data.frame(c1k\_week), c1k\_week\_prediction)  
t\_v\_c1k<- t\_v\_c1k[order(t\_v\_c1k$TransactionTime),]%>%ungroup()  
t\_v\_c1k<- feature\_engineering(t\_v\_c1k, c("quantity\_avrg", "transaction\_avrg", "members\_avrg", "sales\_local\_avrg", "exchange\_rate\_avrg", "sales\_usd\_avrg", "category\_sales\_local\_avrg", "category\_sales\_usd\_avrg", "category\_quantity\_avrg","salePrice\_local\_avrg","salePrice\_usd\_avrg", "log1p\_quantity\_avrg","log1p\_category\_sales\_local\_avrg","log1p\_salePrice\_local\_avrg","log1p\_salePrice\_usd\_avrg"))  
# creating training matrix  
t\_v\_c1k<- fastDummies::dummy\_cols(t\_v\_c1k, select\_columns = c("week\_number\_year", "week\_number", "month"))  
t\_v\_c1k$month<-NULL  
t\_v\_c1k$week\_number\_year<-NULL  
t\_v\_c1k$week\_number<-NULL  
  
train\_c1k<-t\_v\_c1k[1:(nrow(t\_v\_c1k)-len\_dec-len\_inv),]  
test\_c1k<-t\_v\_c1k[(nrow(t\_v\_c1k)-len\_dec-len\_inv+1):nrow(t\_v\_c1k),]

# XGB model

trainc1k\_XGB<-train\_c1k%>%select(-c("transaction\_avrg", "members\_avrg", "sales\_local\_avrg", "exchange\_rate\_avrg", "sales\_usd\_avrg", "category\_sales\_local\_avrg", "category\_sales\_usd\_avrg", "category\_quantity\_avrg","salePrice\_local\_avrg","salePrice\_usd\_avrg", "log1p\_quantity\_avrg", "log1p\_transaction\_avrg", "log1p\_members\_avrg", "log1p\_sales\_local\_avrg", "log1p\_sales\_usd\_avrg", "log1p\_category\_sales\_local\_avrg","log1p\_category\_sales\_usd\_avrg", "log1p\_category\_quantity\_avrg","log1p\_salePrice\_local\_avrg","log1p\_salePrice\_usd\_avrg","TransactionTime"))  
testc1k\_XGB<-test\_c1k%>%select(-c("transaction\_avrg", "members\_avrg", "sales\_local\_avrg", "exchange\_rate\_avrg", "sales\_usd\_avrg", "category\_sales\_local\_avrg", "category\_sales\_usd\_avrg", "category\_quantity\_avrg","salePrice\_local\_avrg","salePrice\_usd\_avrg", "log1p\_quantity\_avrg", "log1p\_transaction\_avrg", "log1p\_members\_avrg", "log1p\_sales\_local\_avrg", "log1p\_sales\_usd\_avrg", "log1p\_category\_sales\_local\_avrg","log1p\_category\_sales\_usd\_avrg", "log1p\_category\_quantity\_avrg","log1p\_salePrice\_local\_avrg","log1p\_salePrice\_usd\_avrg","TransactionTime"))  
testc1k\_XGB$quantity\_avrg<-0  
  
trainTask <- makeRegrTask(data = trainc1k\_XGB, target = "quantity\_avrg")  
testTask <- makeRegrTask(data = testc1k\_XGB, target = "quantity\_avrg")  
  
xgb\_learner <- makeLearner(  
 "regr.xgboost",  
 predict.type = "response",  
 par.vals = list(  
 objective = "reg:squarederror",  
 eval\_metric = "rmse",  
 nrounds = 200  
 )  
)

## Warning in makeParam(id = id, type = "numeric", learner.param = TRUE, lower = lower, : NA used as a default value for learner parameter missing.  
## ParamHelpers uses NA as a special value for dependent parameters.

# Create a model  
xgb\_model <- mlr::train(xgb\_learner, task = trainTask)  
  
xgb\_params <- makeParamSet(  
 # The number of trees in the model (each one built sequentially)  
 makeIntegerParam("nrounds", lower = 100, upper = 500),  
 # number of splits in each tree  
 makeIntegerParam("max\_depth", lower = 1, upper = 10),  
 # "shrinkage" - prevents overfitting  
 makeNumericParam("eta", lower = .1, upper = .5),  
 # L2 regularization - prevents overfitting  
 makeNumericParam("lambda", lower = -1, upper = 0, trafo = function(x) 10^x)  
)  
control <- makeTuneControlRandom(maxit = 1)  
resample\_desc <- makeResampleDesc("CV", iters = 10)  
tuned\_params <- tuneParams(  
 learner = xgb\_learner,  
 task = trainTask,  
 resampling = resample\_desc,  
 par.set = xgb\_params,  
 control = control  
)

## [Tune] Started tuning learner regr.xgboost for parameter set:

## Type len Def Constr Req Tunable Trafo  
## nrounds integer - - 100 to 500 - TRUE -  
## max\_depth integer - - 1 to 10 - TRUE -  
## eta numeric - - 0.1 to 0.5 - TRUE -  
## lambda numeric - - -1 to 0 - TRUE Y

## With control class: TuneControlRandom

## Imputation value: Inf

## [Tune-x] 1: nrounds=224; max\_depth=3; eta=0.404; lambda=0.248

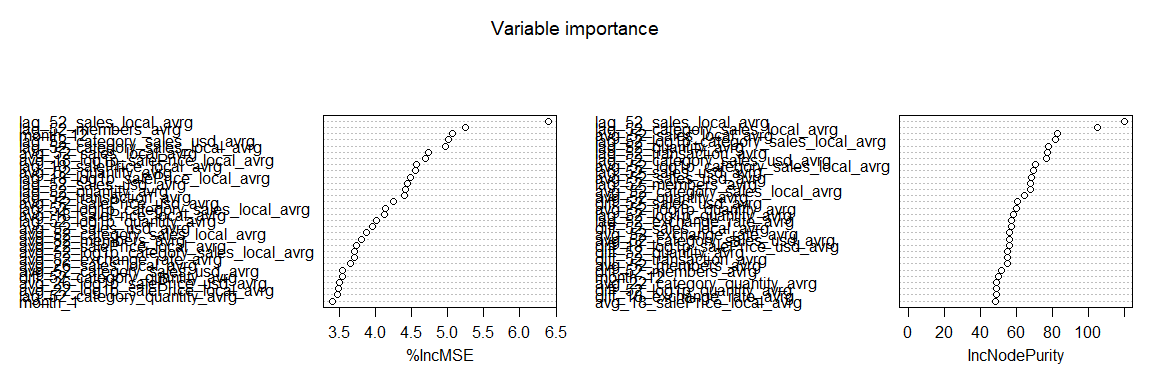
## [Tune-y] 1: mse.test.mean=45.4276661; time: 0.2 min

## [Tune] Result: nrounds=224; max\_depth=3; eta=0.404; lambda=0.248 : mse.test.mean=45.4276661

xgb\_tuned\_learner <- setHyperPars(  
 learner = xgb\_learner,  
 par.vals = tuned\_params$x  
)  
xgb\_model <- mlr::train(xgb\_tuned\_learner, trainTask)  
XGBoost\_pred <- predict(xgb\_model ,testTask)  
  
  
c1k\_week\_quantity\_models<-performance\_index\_raw(df\_Actual = c1k\_week\_quantity\_models, pred\_name = "XGBoost", pred\_value = log(XGBoost\_pred$data$response))  
c1k\_prediction$XGBoost<-log(XGBoost\_pred$data$response)

# Random Forest 1

# RF1  
trainc1k\_RF<-train\_c1k%>%select(-c("transaction\_avrg", "members\_avrg", "sales\_local\_avrg", "exchange\_rate\_avrg", "sales\_usd\_avrg", "category\_sales\_local\_avrg", "category\_sales\_usd\_avrg", "category\_quantity\_avrg","salePrice\_local\_avrg","salePrice\_usd\_avrg", "log1p\_quantity\_avrg", "log1p\_transaction\_avrg", "log1p\_members\_avrg", "log1p\_sales\_local\_avrg", "log1p\_sales\_usd\_avrg", "log1p\_category\_sales\_local\_avrg","log1p\_category\_sales\_usd\_avrg", "log1p\_category\_quantity\_avrg","log1p\_salePrice\_local\_avrg","log1p\_salePrice\_usd\_avrg","TransactionTime"))  
testc1k\_RF<-test\_c1k%>%select(-c("transaction\_avrg", "members\_avrg", "sales\_local\_avrg", "exchange\_rate\_avrg", "sales\_usd\_avrg", "category\_sales\_local\_avrg", "category\_sales\_usd\_avrg", "category\_quantity\_avrg","salePrice\_local\_avrg","salePrice\_usd\_avrg", "log1p\_quantity\_avrg", "log1p\_transaction\_avrg", "log1p\_members\_avrg", "log1p\_sales\_local\_avrg", "log1p\_sales\_usd\_avrg", "log1p\_category\_sales\_local\_avrg","log1p\_category\_sales\_usd\_avrg", "log1p\_category\_quantity\_avrg","log1p\_salePrice\_local\_avrg","log1p\_salePrice\_usd\_avrg","TransactionTime"))  
  
RF\_1 <- randomForest(quantity\_avrg ~. , data = trainc1k\_RF,  
 ntree = 1000, mtry = 3, nodesize = 5, importance = TRUE, na.action = na.omit)  
  
varImpPlot(RF\_1, main = "Variable importance")



RF1\_pred<-predict(RF\_1, testc1k\_RF)  
c1k\_prediction$RF\_1<-RF1\_pred

# Random Forest 2

# RF2  
#Defining the Control  
trControl<- trainControl(method = "cv", number = 10, search = "grid")  
metric <- "RMSE"  
seed<- set.seed(156230)

## Step 1 Run a Default model

rf\_default<- caret::train(quantity\_avrg~ .  
 , data = trainc1k\_RF  
 , method = "rf", metric = "RMSE", trControl = trControl, na.action=na.exclude)  
rf\_default

## Random Forest   
##   
## 197 samples  
## 256 predictors  
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 129, 127, 127, 128, 127, 127, ...   
## Resampling results across tuning parameters:  
##   
## mtry RMSE Rsquared MAE   
## 2 6.412854 0.2878796 4.816868  
## 129 6.117802 0.2855863 4.636245  
## 256 6.234882 0.2653072 4.725933  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was mtry = 129.

#RMSE was used to select the optimal model using the smallest value.

## Step 2 Search best mtry

#mtry is the number of variables available for splitting at each tree node  
tuneGrid<- expand.grid(.mtry = seq(1, 60, by=5))  
rf\_mtry<- caret::train(quantity\_avrg ~ .,   
 data = trainc1k\_RF,   
 method = "rf", metric = "RMSE", tuneGrid = tuneGrid, trControl = trControl, importance = TRUE, na.action=na.exclude)  
rf\_mtry

## Random Forest   
##   
## 197 samples  
## 256 predictors  
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 128, 126, 129, 129, 129, 126, ...   
## Resampling results across tuning parameters:  
##   
## mtry RMSE Rsquared MAE   
## 1 6.573968 0.3285595 4.952693  
## 6 6.019663 0.3727813 4.581343  
## 11 6.038405 0.3465681 4.619910  
## 16 5.975858 0.3572465 4.579463  
## 21 5.943166 0.3646824 4.553805  
## 26 5.988262 0.3517577 4.589666  
## 31 5.978099 0.3503908 4.615982  
## 36 5.931263 0.3653116 4.586727  
## 41 5.958761 0.3577521 4.611710  
## 46 5.955462 0.3612368 4.567985  
## 51 5.992122 0.3459249 4.651455  
## 56 5.973968 0.3547086 4.608025  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was mtry = 36.

#RMSE was used to select the optimal model using the smallest value.  
best\_mtry<- rf\_mtry$bestTune$mtry #store the best value for mtry  
min(rf\_mtry$results$RMSE)

## [1] 5.931263

## Step 3 Search Best Maxnodes

store\_maxnode<- list() # create a list to find the optimal max of nodes  
tuneGrid<- expand.grid(.mtry= best\_mtry)  
for (maxnodes in c(1, 2, 3, 4, 5, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70)){  
 set.seed(156230)  
 rf\_maxnode<- caret::train(quantity\_avrg ~ .,   
 data = trainc1k\_RF,   
 method = "rf", metric = "RMSE", tuneGrid = tuneGrid, trControl = trControl, importance = TRUE, maxnodes = maxnodes, nodesize = 4, na.action = na.exclude)  
 current\_iteration<- toString(maxnodes)  
 store\_maxnode[[current\_iteration]]<- rf\_maxnode  
}

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =  
## trainInfo, : There were missing values in resampled performance measures.

results\_node<- resamples(store\_maxnode)  
results\_node<-summary(results\_node)  
nnode\_optimal<-results\_node$models[results\_node$statistics$RMSE%>%as.data.frame()%>%select("Mean")%>%as.matrix()%>%as.numeric()%>%which.min()]%>%as.numeric()

## Step 4 Search the best ntrees

store\_maxtrees <- list()  
for (ntree in c(10, 20, 30, 40, 50, 100 , 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700)) {  
 set.seed(156230)  
 rf\_maxtrees <- caret::train(quantity\_avrg ~ .,   
 data = trainc1k\_RF,  
 method = "rf",  
 metric = "RMSE",  
 tuneGrid = tuneGrid,  
 trControl = trControl,  
 importance = TRUE,  
 maxnodes = nnode\_optimal,  
 ntree = ntree,  
 na.action = na.exclude)  
 key <- toString(ntree)  
 store\_maxtrees[[key]] <- rf\_maxtrees  
}  
results\_tree <- resamples(store\_maxtrees)  
results\_tree<-summary(results\_tree)  
ntrees\_optimal<-results\_tree$models[results\_tree$statistics$RMSE%>%as.data.frame()%>%select("Mean")%>%as.matrix()%>%as.numeric()%>%which.min()]%>%as.numeric()

## Step 5 Run model with the best specifications found above

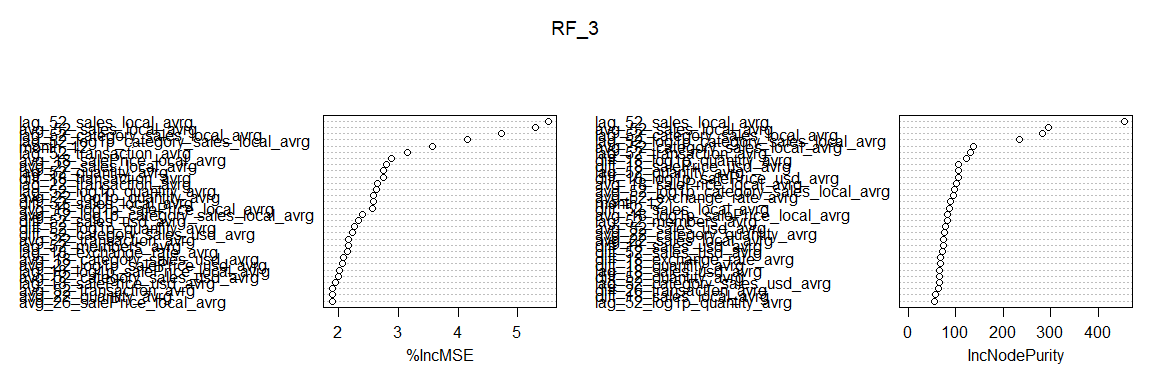
fit\_rf<- caret::train(quantity\_avrg ~ .,   
 data = trainc1k\_RF,  
 method = "rf",  
 metric = "RMSE",  
 tuneGrid = tuneGrid,  
 trControl = trControl,  
 importance = TRUE,  
 maxnodes = nnode\_optimal,  
 ntree = ntrees\_optimal,  
 na.action = na.exclude)

## Step 6 Evaluate Model

pred\_rf2<- predict(fit\_rf,test\_c1k, predict.all = TRUE)  
c1k\_prediction$RF\_2<-c(pred\_rf2, rep(NA, len\_dec+len\_inv-length(pred\_rf2)))

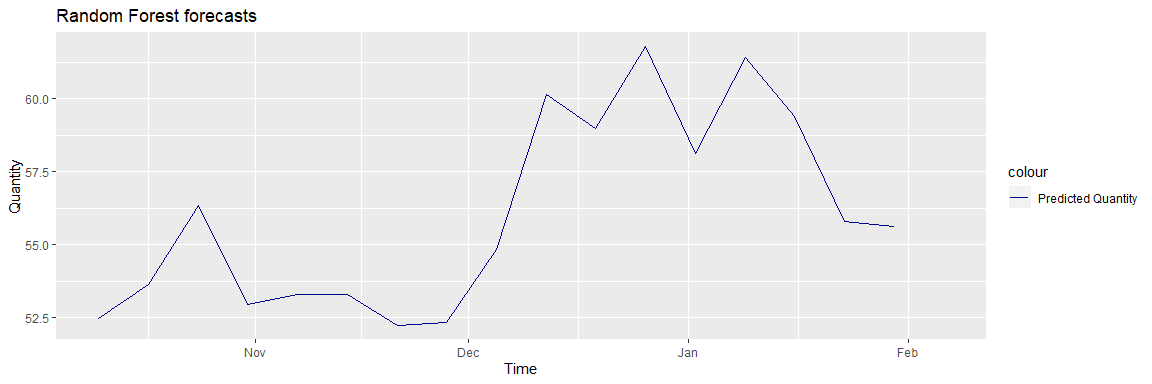
# Random Forest 3

RF\_3<- randomForest(quantity\_avrg ~ .,   
 data = trainc1k\_RF,  
 mtry = best\_mtry,  
 importance = TRUE,  
 maxnodes = nnode\_optimal,  
 ntree = ntrees\_optimal,  
 na.action = na.exclude)  
   
varImpPlot(RF\_3)



pred\_rf3 <- predict(RF\_3, test\_c1k)  
pred\_rf3<- data.table(Predicted\_Quantity = pred\_rf3, TransactionTime = test\_c1k$TransactionTime)  
  
ggplot() +  
 geom\_line(pred\_rf3, mapping = aes(TransactionTime, Predicted\_Quantity,color = "Predicted Quantity"))+  
 labs(x = "Time", y = "Quantity", title = "Random Forest forecasts") +  
 scale\_color\_manual(values = c("Predicted Quantity" = 'darkblue', "Actual Quantity" = 'red'))

## Warning: Removed 1 rows containing missing values (geom\_path).



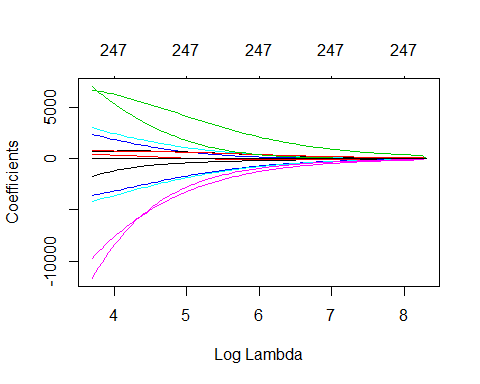
c1k\_prediction$RF\_3<-c(pred\_rf3$Predicted\_Quantity, rep(NA, len\_dec+len\_inv-length(pred\_rf3$Predicted\_Quantity)))

# Linear regression: Ridge, Random Forest and elastic

train\_c1k\_GLMNET<-drop\_na(train\_c1k%>%select(-c("transaction\_avrg", "members\_avrg", "sales\_local\_avrg", "exchange\_rate\_avrg", "sales\_usd\_avrg", "category\_sales\_local\_avrg", "category\_sales\_usd\_avrg", "category\_quantity\_avrg", "salePrice\_local\_avrg","salePrice\_usd\_avrg", "log1p\_quantity\_avrg", "log1p\_transaction\_avrg", "log1p\_members\_avrg", "log1p\_sales\_local\_avrg", "log1p\_sales\_usd\_avrg", "log1p\_category\_sales\_local\_avrg","log1p\_category\_sales\_usd\_avrg", "log1p\_category\_quantity\_avrg", "log1p\_salePrice\_local\_avrg","log1p\_salePrice\_usd\_avrg", "TransactionTime")))  
test\_c1k\_GLMNET<-drop\_na(test\_c1k%>%select(-c("quantity\_avrg", "transaction\_avrg", "members\_avrg", "sales\_local\_avrg", "exchange\_rate\_avrg", "sales\_usd\_avrg", "category\_sales\_local\_avrg", "category\_sales\_usd\_avrg", "category\_quantity\_avrg", "salePrice\_local\_avrg","salePrice\_usd\_avrg", "log1p\_quantity\_avrg", "log1p\_transaction\_avrg", "log1p\_members\_avrg", "log1p\_sales\_local\_avrg", "log1p\_sales\_usd\_avrg", "log1p\_category\_sales\_local\_avrg","log1p\_category\_sales\_usd\_avrg", "log1p\_category\_quantity\_avrg","log1p\_salePrice\_local\_avrg","log1p\_salePrice\_usd\_avrg", "TransactionTime")))  
  
y\_train<- data.matrix(train\_c1k\_GLMNET["quantity\_avrg"])  
x\_train<-data.matrix(subset(train\_c1k\_GLMNET, select=-c(quantity\_avrg)))  
  
x\_test<-data.matrix(test\_c1k\_GLMNET)

## Ridge

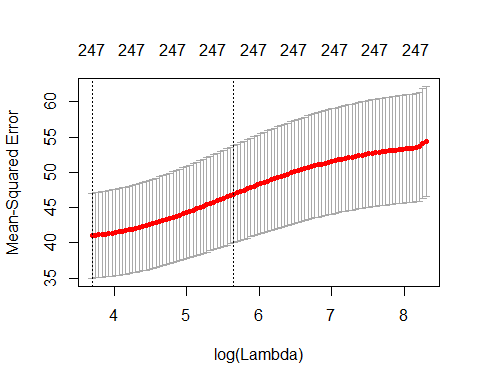
ridge <- glmnet(x\_train,y\_train,alpha = 0)  
  
plot(ridge, xvar = "lambda")



ridge$lambda %>% head()

## [1] 4027.958 3844.881 3670.125 3503.312 3344.081 3192.088

#Tuning to find the right value for lamda   
ridge\_cv <- cv.glmnet(x\_train,y\_train,alpha = 0)  
plot(ridge\_cv)



# as we constrain our coefficients with log ( λ ) ≥ 0 penalty, the MSE rises considerably. The numbers at the top of the plot (38) just refer to the number of variables in the model. Ridge regression does not force any variables to exactly zero so all features will remain in the model   
  
#The first and second vertical dashed lines represent the λ value with the minimum MSE and the largest λ value within one standard error of the minimum MSE.   
#extract our minimum and one standard error MSE and λ values  
min(ridge\_cv$cvm) #minimum MSE

## [1] 40.97907

ridge\_cv$lambda.min #lambda for this minimum MSE

## [1] 40.27958

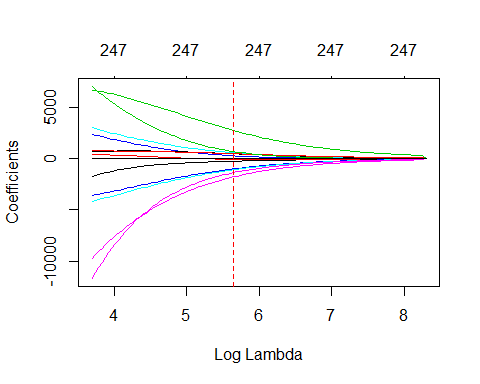
ridge\_cv$cvm[ridge\_cv$lambda == ridge\_cv$lambda.1se] # 1 st.error of min MSE

## [1] 46.89637

ridge\_cv$lambda.1se # lambda for this MSE

## [1] 284.1645

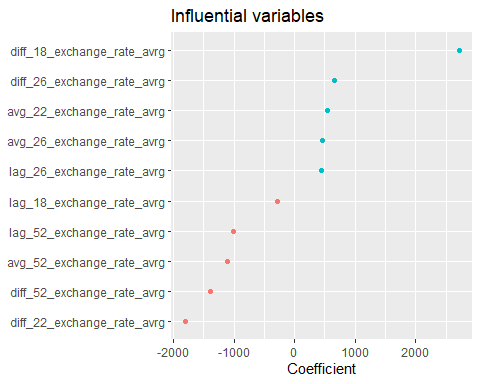
ridge\_min <- glmnet(x\_train,y\_train, alpha = 0)  
plot(ridge\_min, xvar = "lambda")  
abline(v = log(ridge\_cv$lambda.1se), col = "red", lty = "dashed")



#Most Influential Feautures to predict accuracy  
coef(ridge\_cv, s = "lambda.1se") %>%  
 tidy() %>%  
 filter(row != "(Intercept)") %>%  
 top\_n(10, wt = abs(value)) %>%  
 ggplot(aes(value, reorder(row, value), color = value > 0)) +  
 geom\_point(show.legend = FALSE) +  
 ggtitle("Influential variables") +  
 xlab("Coefficient") +  
 ylab(NULL)

## Warning: 'tidy.dgCMatrix' is deprecated.  
## See help("Deprecated")

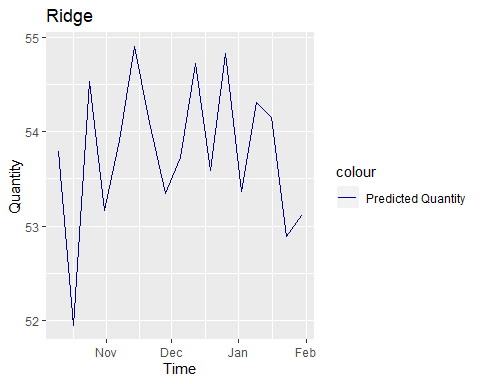
## Warning: 'tidy.dgTMatrix' is deprecated.  
## See help("Deprecated")



min(ridge\_cv$cvm)

## [1] 40.97907

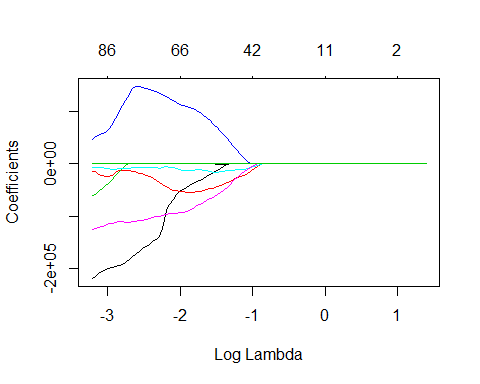
#Ridge model will retian all variables. Therefore, a ridge model is good only if we beleve that we need to retain all features in the model yet reduce the noise that less influential variable smay create and minimize collinearity. Ridge doesn't perform feature selection   
##PREDICTING  
ridge\_pred<- predict(ridge\_cv, s=ridge\_cv$lambda.1se, x\_test, type = "response")  
ridge\_pred\_train<-predict(ridge\_cv, s=ridge\_cv$lambda.1se, x\_train, type = "response")  
ridge\_pred\_train<-merge(data.frame(week=1:nrow(train\_c1k)), data.frame(week=ridge\_pred\_train%>%rownames()%>%as.numeric(), pred=ridge\_pred\_train), all.x = TRUE)  
colnames(ridge\_pred\_train)<-c("week", "pred")  
#Graph  
pred\_ridge<- data.frame(Predicted\_Quantity = ridge\_pred, TransactionTime = test\_c1k$TransactionTime[1:length(ridge\_pred)])  
colnames(pred\_ridge)<-c("Predicted\_Quantity", "TransactionTime")  
ggplot() +  
 geom\_line(pred\_ridge, mapping = aes(TransactionTime, Predicted\_Quantity, color = "Predicted Quantity"))+  
 labs(x = "Time", y = "Quantity", title = "Ridge") + scale\_color\_manual(values = c("Predicted Quantity" = 'darkblue', "Actual Quantity" = 'red'))



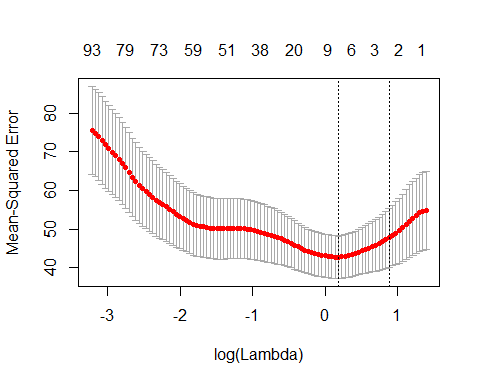
c1k\_prediction$Ridge<-c(pred\_ridge$Predicted\_Quantity, rep(NA, len\_dec+len\_inv-length(pred\_ridge$Predicted\_Quantity)))

## Lasso

lasso <- glmnet(x\_train,y\_train,alpha = 1)  
  
plot(lasso, xvar = "lambda")



# when log(λ)=− 3 all 8 variables are in the model, when log(λ)=−1 2 variables are retained  
   
#Tuning to find the right value for lamda   
lasso\_cv <- cv.glmnet(x\_train,y\_train,alpha = 1)  
plot(lasso\_cv)



#extract our minimum and one standard error MSE and λ values  
min(lasso\_cv$cvm) #minimum MSE

## [1] 42.74379

lasso\_cv$lambda.min #lambda for this minimum MSE

## [1] 1.2018

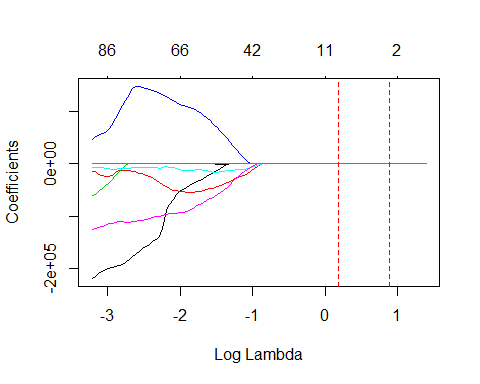
lasso\_cv$cvm[lasso\_cv$lambda == lasso\_cv$lambda.1se] # 1 st.error of min MSE

## [1] 47.75308

lasso\_cv$lambda.1se # lambda for this MSE

## [1] 2.414697

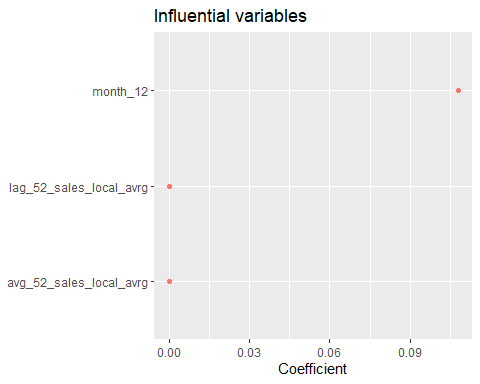
lasso\_min <- glmnet(x\_train,y\_train, alpha = 1)  
plot(lasso\_min, xvar = "lambda")  
abline(v = log(lasso\_cv$lambda.min), col = "red", lty = "dashed")  
abline(v = log(lasso\_cv$lambda.1se), col = "red", lty = "dashed")



#Most Influential Feautures to predict accuracy  
coef(lasso\_cv, s = "lambda.1se") %>%  
 tidy() %>%  
 filter(row != "(Intercept)") %>%  
 ggplot(aes(value, reorder(row, value), color = value > 0)) +  
 geom\_point(show.legend = FALSE) +  
 ggtitle("Influential variables") +  
 xlab("Coefficient") +  
 ylab(NULL)

## Warning: 'tidy.dgCMatrix' is deprecated.  
## See help("Deprecated")

## Warning: 'tidy.dgTMatrix' is deprecated.  
## See help("Deprecated")



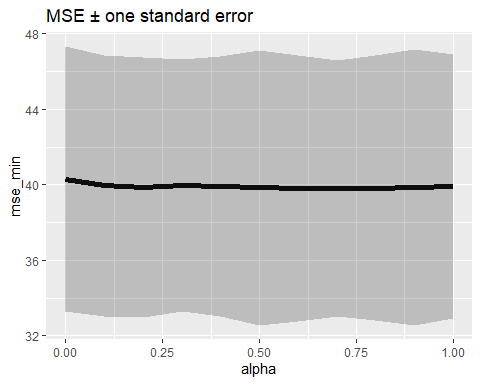
##Predicting  
lasso\_pred<- predict(lasso\_cv, s=lasso\_cv$lambda.min, x\_test, type = "response")  
c1k\_prediction$Lasso<-c(lasso\_pred, rep(NA, len\_dec+len\_inv-length(lasso\_pred)))

## Elstic Net

#Tune λ and the alpha parameters.  
# maintain the same folds across all models  
fold\_id <- sample(1:10, size = length(y\_train), replace=TRUE)  
  
# search across a range of alphas  
tuning\_grid <- tibble::tibble(  
 alpha = seq(0, 1, by = .1),  
 mse\_min = NA,  
 mse\_1se = NA,  
 lambda\_min = NA,  
 lambda\_1se = NA  
)  
  
#Now we can iterate over each alpha value, apply a CV elastic net, and extract the minimum and one standard error MSE values and their respective λ values.  
for(i in seq\_along(tuning\_grid$alpha)) {  
   
 # fit CV model for each alpha value  
 fit <- cv.glmnet(x\_train, y\_train, alpha = tuning\_grid$alpha[i], foldid = fold\_id)  
   
 # extract MSE and lambda values  
 tuning\_grid$mse\_min[i] <- fit$cvm[fit$lambda == fit$lambda.min]  
 tuning\_grid$mse\_1se[i] <- fit$cvm[fit$lambda == fit$lambda.1se]  
 tuning\_grid$lambda\_min[i] <- fit$lambda.min  
 tuning\_grid$lambda\_1se[i] <- fit$lambda.1se  
}  
  
tuning\_grid

## # A tibble: 11 x 5  
## alpha mse\_min mse\_1se lambda\_min lambda\_1se  
## <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0 40.3 47.3 40.3 312.   
## 2 0.1 39.9 46.9 7.55 20.0   
## 3 0.2 39.9 46.7 4.14 11.0   
## 4 0.3 40.0 46.6 3.03 7.68  
## 5 0.4 39.9 46.8 2.74 6.04  
## 6 0.5 39.8 47.1 2.29 5.06  
## 7 0.6 39.8 46.8 1.91 4.22  
## 8 0.7 39.8 46.6 1.64 3.61  
## 9 0.8 39.8 46.8 1.43 3.31  
## 10 0.9 39.9 47.2 1.27 3.08  
## 11 1 39.9 46.9 1.15 2.78

#plot the MSE   
elastic\_tuning<-tuning\_grid %>%mutate(se = mse\_1se - mse\_min)  
  
elastic\_tuning%>%  
 ggplot(aes(alpha, mse\_min)) +  
 geom\_line(size = 2) +  
 geom\_ribbon(aes(ymax = mse\_min + se, ymin = mse\_min - se), alpha = .25) +  
 ggtitle("MSE ± one standard error")



alpha\_optimal<-(elastic\_tuning%>%as.data.frame()%>%select("alpha"))[which.min(elastic\_tuning%>%as.data.frame()%>%select("mse\_1se")%>%unlist()%>%as.numeric()),1]  
#advantage of the elastic net model is that it enables effective regularization via the ridge penalty with the feature selection characteristics of the lasso penalty  
  
elastic\_cv<-cv.glmnet(x\_train,y\_train,alpha = alpha\_optimal)  
  
elastic\_pred<- predict(elastic\_cv, s=elastic\_cv$lambda.1se, x\_test, type = "response")  
c1k\_prediction$elastic<-c(elastic\_pred, rep(NA, len\_dec+len\_inv-length(lasso\_pred)))

# Final prediction

c1k\_prediction$quantity\_avrg<-rowMeans(c1k\_prediction%>%select(best\_two\_total\_ml[1],  
 best\_two\_total\_ml[2],  
 best\_two\_total[1],  
 best\_two\_total[2]))  
c1k\_prediction$week\_begin<-c1k\_prediction$TransactionTime-2  
c1k\_prediction$week\_end<-c1k\_prediction$TransactionTime+4

# Benchmark model 1: HoltWinter Smoothing

t\_log\_HW<- ts(log(c1k\_week$quantity\_avrg), frequency = 52)  
expsmo<-HoltWinters(t\_log\_HW, seasonal = "additive")  
pred\_HW<-predict(expsmo, n.ahead = len\_dec+len\_inv)  
c1k\_prediction$HW<-exp(c(pred\_HW%>%as.numeric(), rep(NA, len\_dec+len\_inv-length(pred\_HW)))+(retrend\_log-trend\_log[1])+reseasonal\_log)

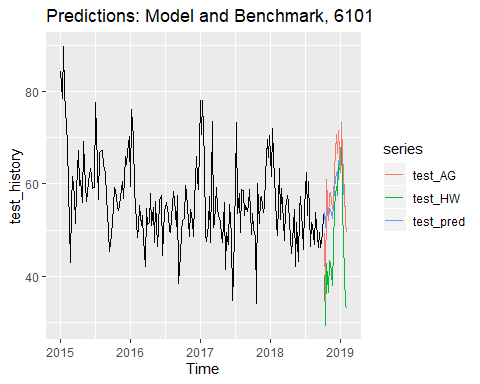
# Benchmark model 2: Annual Growth

annual\_growth\_last<-t\_log\_HW[length(t\_log\_HW)]/lag(t\_log\_HW%>%as.numeric(), n = 52)[length(t\_log\_HW)]  
pred\_AG<-c(c1k\_week$quantity\_avrg, rep(NA, len\_dec+len\_inv))  
pred\_AG\_lag52<-lag(pred\_AG%>%as.numeric(), n = 52)%>%tail(len\_dec+len\_inv)  
c1k\_prediction$AG<-pred\_AG\_lag52\*annual\_growth\_last

# Output

# Spreadsheet  
write.csv(c1k\_prediction%>%select("quantity\_avrg", "week\_begin", "week\_end", "HW", "AG"), "Output/Countries/Colombia/pred\_c1k.csv")  
# Grphs  
test\_history<-ts(c1k\_week$quantity\_avrg, frequency = 52.14, start = c(c1k\_week$week\_number\_year[1], c1k\_week$week\_number[1]))   
c1k\_prediction$week\_number<-c1k\_prediction$TransactionTime%>%date2ISOweek()%>%substr(7,8)%>%as.numeric()  
c1k\_prediction$week\_number\_year<-c1k\_prediction$TransactionTime%>%date2ISOweek()%>%substr(1,4)%>%as.numeric()  
test\_pred<-ts(c1k\_prediction$quantity\_avrg, frequency = 52.14, start = c(c1k\_prediction$week\_number\_year[1], c1k\_prediction$week\_number[1]))   
test\_HW<-ts(c1k\_prediction$HW, frequency = 52.14, start = c(c1k\_prediction$week\_number\_year[1], c1k\_prediction$week\_number[1]))   
test\_AG<-ts(c1k\_prediction$AG, frequency = 52.14, start = c(c1k\_prediction$week\_number\_year[1], c1k\_prediction$week\_number[1]))   
autoplot(test\_history)+autolayer(test\_pred)+autolayer(test\_HW)+autolayer(test\_AG)+ggtitle("Predictions: Model and Benchmark, 6101")

## Warning: Removed 1 rows containing missing values (geom\_path).



save.image(paste0("Output/Countries/Colombia/", club, "test.RData"))