A VERY quick and dirty PoC for the IWD Use Case

Andrew Lowe

9 February 2018

Business challenge

- Quantify the impact of various drivers on the share performance of a specific brand across a selected market.
- Predict the share performance of brands for future periods.

In the following, we will prove that we have achieved both objectives.

Configure setup and load the data

The easiest way to import the data into R is to first export it from Spotfire as an Excel spreadsheet. (Unfortunately, exporting the data as a tab-delimited text file results in weird encoding issues that cause errors like "embedded nul in string" when using `read.table` or `fread` to read the data in R.):

```
suppressPackageStartupMessages( # Tidyverse is verbose on startup
  require(tidyverse) # We'll need tools from this library
)
```

```
## Warning: package 'tidyverse' was built under R version 3.4.2

## Warning: package 'tibble' was built under R version 3.4.2

## Warning: package 'tidyr' was built under R version 3.4.2

## Warning: package 'purrr' was built under R version 3.4.2

## Warning: package 'dplyr' was built under R version 3.4.2

## Warning: package 'forcats' was built under R version 3.4.2
```

```
set.seed(42) # Set random number seed for the sake of reproducibility

require(readxl) # For reading Excel files
```

```
## Loading required package: readxl
```

```
dat.xls <- read_xls("Data Table.xls") # Read in data
names(dat.xls) # Print column names
```

```
##  [1] "Column 1"                   "TimeName"
##  [3] "PeriodType"                 "PeriodEndDate"
##  [5] "Region"                     "Country"
##  [7] "Area"                       "AreaHierarchy"
##  [9] "Category"                   "Company"
## [11] "Brand"                      "Form"
## [13] "Concentration"              "SecondBenefit"
## [15] "NumberOfJobs"               "BasicSize"
## [17] "Item"                       "ValueSalesMLC"
## [19] "VolumeSalesMSU"             "UnitSalesx1000"
## [21] "WeightedDistribution"       "WDDisplay"
## [23] "WDAnyPromo"                 "WDFeature"
```

```
## [25] "WDPriceCut"                 "NumSize"
## [27] "PPU"                        "PPSU"
## [29] "PricePerScoop"              "PromoPerDistribution (2)"
## [31] "ScoopsSold (2)"             "Fitted"
## [33] "Resid"                      "Filtered out at 5:38:00 PM"
## [35] "ValidPpuCount"              "Date"
## [37] "PromoPerDistribution"       "ScoopsSold"
## [39] "PriceCutPerDistribution"
```

Data cleaning

We subset the data to contain just the columns that we believe we need for buildig a predictive model:

```
# Select the columns we need (according to Tamas Sarkadi <Tamas_Sarkadi@epam.com>):
dat <- dat.xls[, names(dat.xls) %in% c(
  "Category",
  "Company",
  "Brand",
  "Form",
  "Concentration",
  "BasicSize",
  "SecondBenefit",
  "NumberOfJobs",
  "Item",
  "ValueSalesMLC", # Target
  "Date", # Timestamp
  "WeightedDistribution", # Feature
  "WDFeature", # Feature
  "WDDisplay", # Feature
  "WDPriceCut", # Feature
  "PPSU" # Feature
)]
names(dat)
```

```
##  [1] "Category"             "Company"              "Brand"
##  [4] "Form"                 "Concentration"        "SecondBenefit"
##  [7] "NumberOfJobs"         "BasicSize"            "Item"
## [10] "ValueSalesMLC"        "WeightedDistribution" "WDDisplay"
## [13] "WDFeature"            "WDPriceCut"           "PPSU"
## [16] "Date"
```

Some columns are single valued and therefore have no predictive value; they are removed:

```
strip.single.valued <- function(df) {
  only.one.value <- sapply(df, function(x) length(unique(x)) == 1) # Count unique values
  str(df[1, only.one.value]) # These columns only have one unique value! Print them
  df <- df[, !only.one.value] # Remove those columns
  return(df)
}

dat <- strip.single.valued(dat) # Do it
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1 obs. of  3 variables:
##  $ Category : chr "LAUNDRY DETERGENTS V2 CATEGORY"
##  $ WDDisplay: logi NA
```

```
##  $ WDFeature: logi NA
```

The date information (in *POSIXct* format) is transformed into *Date* format:

```r
require(lubridate)
```

```
## Loading required package: lubridate

## Warning: package 'lubridate' was built under R version 3.4.2

##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##     date
```

```r
dat$Date <- ymd(dat$Date) # Transform into Date object
```

We remove any duplicated columns, if present:

```r
# Remove columns with duplicate entries by fast comparison of hashes:
require(digest)
```

```
## Loading required package: digest
```

```r
duplicate.columns <- names(dat)[duplicated(lapply(dat, digest))]
if(length(duplicate.columns) == 0) {# Are there any duplicate columns?
  print("No duplicated columns")
} else {
  print(duplicate.columns)
}
```

```
## [1] "No duplicated columns"
```

```r
dat <- dat[, !names(dat) %in% duplicate.columns]
names(dat)
```

```
##  [1] "Company"               "Brand"                 "Form"
##  [4] "Concentration"         "SecondBenefit"         "NumberOfJobs"
##  [7] "BasicSize"             "Item"                  "ValueSalesMLC"
## [10] "WeightedDistribution" "WDPriceCut"            "PPSU"
## [13] "Date"
```

We transform character strings to categorical variables:

```r
dat <- dat %>% mutate_if(is.character, as.factor)
```

Now we do the drill-down to get the products/SKUs. How many do we have?

```r
dat %>% group_by(Company,
                 Brand,
                 Form,
                 Concentration,
                 SecondBenefit,
                 NumberOfJobs,
                 BasicSize,
                 Item) %>%
  mutate(SKU = paste( # Add SKU id
    Company,
    Brand,
    Form,
```

```
    Concentration,
    SecondBenefit,
    NumberOfJobs,
    BasicSize,
    Item,
    sep = " | ")
  ) %>%
  arrange(SKU, Date) %>% # Order by SKU then Date
  mutate(count = n()) -> products

products %>% count() %>% nrow() # How many products?
```

```
## [1] 691
```

```
length(unique(products$SKU)) # Check maths: wow many SKUs? Should be identical.
```

```
## [1] 691
```

Some products have data reported for many time points, while others have little data:

```
products %>% pull(count) %>% summary() # Print summary of counts
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0    78.0   113.0   125.7   181.0   258.0
```

What's the maximum size of the reporting period?

```
range(products$Date)
```

```
## [1] "2010-11-07" "2015-10-18"
```

Measurements appear to be taken weekly:

```
head(sort(unique(products$Date)))
```

```
## [1] "2010-11-07" "2010-11-14" "2010-11-21" "2010-11-28" "2010-12-05"
## [6] "2010-12-12"
```

We plot a quick glimpse of the data for the SKUs with more than the mean number of measurements.

```
products %>% filter(count > 125) %>%
  ggplot(aes(x = Date, y = ValueSalesMLC, group = SKU)) +
  facet_wrap(~SKU, scales = "free_y", ncol = 6) +
  geom_line() +
  theme_bw() +
  theme(
    strip.background = element_blank(),
    strip.text.x = element_blank(),
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    axis.ticks.x = element_blank(),
    axis.ticks.y = element_blank(),
    axis.text.x = element_blank(),
    axis.text.y = element_blank()
  )
```

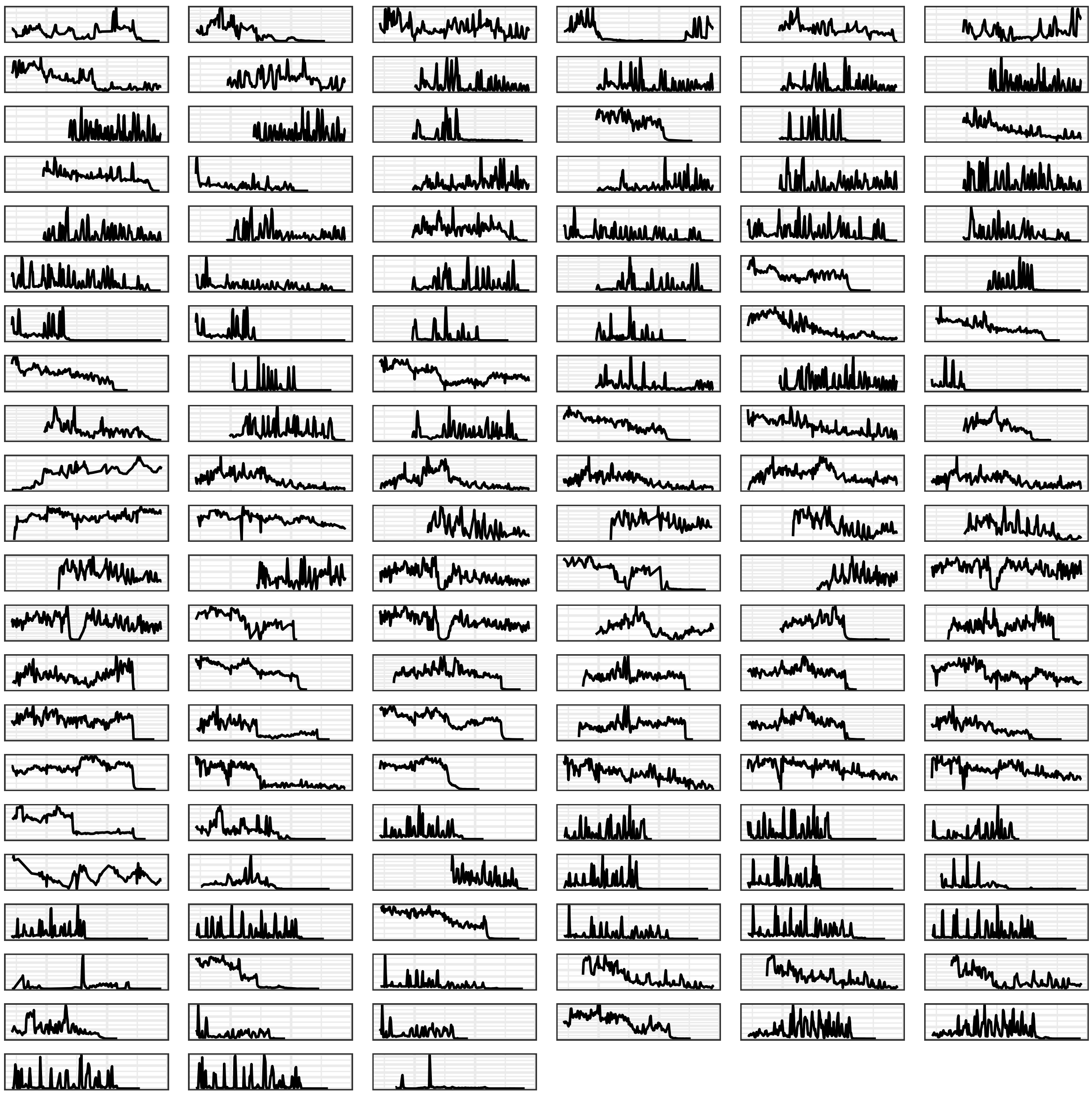


Figure 1: Data for a selection of SKUs.

Forecasting

We'll limit forecasts for SKUs for which we have at least 52 measurements:

```
products %>% filter(count >= 52) -> products
```

Make a list of SKUs:

```
sort(unique(products$SKU)) -> SKUs
```

Select an SKU in list:

```
products %>% filter(SKU == SKUs[2]) -> product
```

There is missing data that is invisibly missing; the rows themselves are missing. Therefore we left-join to a column of dates covering the period such that the original data is padded with missing values:

```
begin <- range(product$Date)[1]
end <- range(product$Date)[2]
dates <- data.frame(Date = seq(from = begin, to = end, by = "week"))

left_join(dates, product, by = "Date") %>%
  mutate( # Expand-out data information
    year = as.numeric(format(Date, format = "%Y")),
    week = week(Date) # Week number
  ) -> product
```

Partition the data into training and test data:

```
require(caret)
```

```
## Loading required package: caret

## Warning: package 'caret' was built under R version 3.4.2

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
split.percent <- 0.85 # 15% holdout
p <- floor(split.percent * nrow(product))
q <- nrow(product) - p
print(c(p+q, p, q)) # Total number of data points and number in each partition
```

```
## [1] 203 172  31
```

```
in.train <- createTimeSlices(1:nrow(product), p, q)

df <- as.data.frame(product)
train <- df[unlist(in.train$train),]
test <- df[unlist(in.train$test),]
```

Start of reporting periods (both partitions):

```
start.train <- c(train$year[1], train$week[1])
start.test <- c(test$year[1], test$week[1])
```

Auto-fit an ARIMA model for an SKU, and interpolate missing values:

```r
require(forecast)
```

```
## Loading required package: forecast
```

```
## Warning: package 'forecast' was built under R version 3.4.3
```

```r
train.ts <- na.interp(
  ts(train$ValueSalesMLC, frequency = 52, start = start.train)
)
train.WeightedDistribution <- na.interp(
  ts(train$WeightedDistribution, frequency = 52, start = start.train)
)
train.PPSU <- na.interp(
  ts(train$PPSU, frequency = 52, start = start.train)
)
train.WDPriceCut <- na.interp(
  ts(train$WDPriceCut, frequency = 52, start = start.train)
)
xregs <- cbind(train.WeightedDistribution, train.PPSU, train.WDPriceCut)

arima.fit <- auto.arima(train.ts, trace = TRUE, xreg = xregs)
```

```
##
##  Fitting models using approximations to speed things up...
##
##  Regression with ARIMA(2,1,2)(1,0,1)[52] errors : Inf
##  Regression with ARIMA(0,1,0)            errors : 284.8334
##  Regression with ARIMA(1,1,0)(1,0,0)[52] errors : 35.43219
##  Regression with ARIMA(0,1,1)(0,0,1)[52] errors : 267.5977
##  ARIMA(0,1,0)                                   : 282.7907
##  Regression with ARIMA(1,1,0)            errors : 280.3644
##  Regression with ARIMA(1,1,0)(1,0,1)[52] errors : Inf
##  Regression with ARIMA(0,1,0)(1,0,0)[52] errors : 48.25668
##  Regression with ARIMA(2,1,0)(1,0,0)[52] errors : 30.85524
##  Regression with ARIMA(2,1,1)(1,0,0)[52] errors : 30.22139
##  Regression with ARIMA(3,1,2)(1,0,0)[52] errors : 27.65443
##  ARIMA(3,1,2)(1,0,0)[52]                        : 26.66312
##  ARIMA(3,1,2)                                   : 248.8354
##  ARIMA(3,1,2)(1,0,1)[52]                        : Inf
##  ARIMA(2,1,2)(1,0,0)[52]                        : Inf
##  ARIMA(4,1,2)(1,0,0)[52]                        : 26.29973
##  ARIMA(4,1,1)(1,0,0)[52]                        : 30.72992
##  ARIMA(4,1,3)(1,0,0)[52]                        : Inf
##  ARIMA(3,1,1)(1,0,0)[52]                        : 27.65614
##  ARIMA(5,1,3)(1,0,0)[52]                        : 30.02482
##  Regression with ARIMA(4,1,2)(1,0,0)[52] errors : 26.47301
##  ARIMA(4,1,2)                                   : 251.4496
##  ARIMA(4,1,2)(1,0,1)[52]                        : Inf
##  ARIMA(5,1,2)(1,0,0)[52]                        : 34.14361
##
##  Now re-fitting the best model(s) without approximations...
##
##  ARIMA(4,1,2)(1,0,0)[52]                        : Inf
##  ARIMA(4,1,2)(1,0,0)[52] with drift             : Inf
```

```
##  ARIMA(3,1,2)(1,0,0)[52]                    : 243.6902
##
##  Best model: Regression with ARIMA(3,1,2)(1,0,0)[52] errors
```

```
summary(arima.fit)
```

```
## Series: train.ts
## Regression with ARIMA(3,1,2)(1,0,0)[52] errors
##
## Coefficients:
##           ar1      ar2      ar3     ma1      ma2    sar1
##       -0.5235  -0.0541  -0.2565  0.1593  -0.5349  0.3540
## s.e.   0.4693   0.1988   0.1477  0.4663   0.3237  0.1035
##       train.WeightedDistribution  train.PPSU  train.WDPriceCut
##                           0.1511     11.3662            0.7948
## s.e.                      0.0253      2.3237            0.1368
##
## sigma^2 estimated as 0.2169:  log likelihood=-111.16
## AIC=242.32   AICc=243.69   BIC=273.73
##
## Training set error measures:
##                        ME      RMSE     MAE       MPE     MAPE      MASE
## Training set -0.0204126 0.4519815 0.30161 -1.014428 9.109642 0.1724312
##                       ACF1
## Training set -0.01421554
```

Forecast ahead for the next 12 weeks:

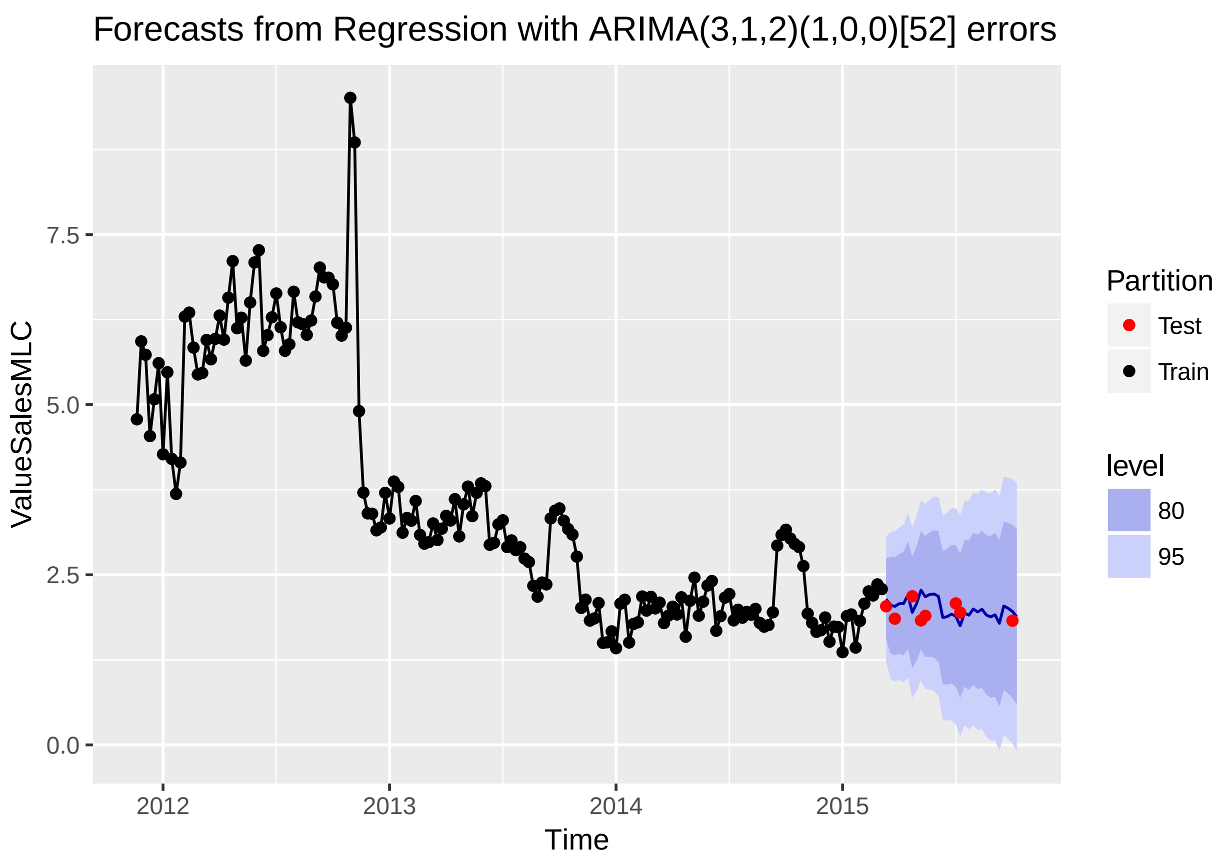
```
test.ts <- ts(test$ValueSalesMLC, frequency = 52, start = start.test)

test.WeightedDistribution <- na.interp(
  ts(test$WeightedDistribution, frequency = 52, start = start.test)
)
test.PPSU <- na.interp(
  ts(test$PPSU, frequency = 52, start = start.test)
)
test.WDPriceCut <- na.interp(
  ts(test$WDPriceCut, frequency = 52, start = start.test)
)
xregs <- cbind(test.WeightedDistribution, test.PPSU, test.WDPriceCut)

arima.forecast <- forecast(arima.fit, h = q, xreg = xregs)
```

Plot the forecast:

```
train.df <- as.data.frame(time(train.ts))
names(train.df) <- "x"
train.df$y <- as.vector(train.ts)
train.df$Partition <- "Train"

test.df <- as.data.frame(time(test.ts))
names(test.df) <- "x"
test.df$y <- as.vector(test.ts)
test.df$Partition <- "Test"

points.dat <- suppressWarnings(bind_rows(test.df, train.df))
```

```r
autoplot(arima.forecast) +
  ylab("ValueSalesMLC") +
  geom_point(data = points.dat, aes(x = x, y = y, colour = Partition)) +
  scale_color_manual(values = c("Train" = 'Black','Test' = 'Red'))
```

```
## Warning: Removed 23 rows containing missing values (geom_point).
```




```r
accuracy(arima.forecast)
```

```
##                      ME      RMSE     MAE       MPE     MAPE      MASE
## Training set -0.0204126 0.4519815 0.30161 -1.014428 9.109642 0.1724312
##                     ACF1
## Training set -0.01421554
```

We perform a statistical siginificant test of the coefficients of the fitted model; if the p-value is less than 5% we take this as evidence that the coefficient is statistically significant. The size of the coefficient provides a measure of the corresponding variable's importance in the model. **This addresses the challenge of determining the performance drivers for a specific product:**

```r
require(lmtest)
```

```
## Loading required package: lmtest

## Warning: package 'lmtest' was built under R version 3.4.3

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 3.4.2
```

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

```
coeftest(arima.fit)
```

```
##
## z test of coefficients:
##
##                           Estimate Std. Error z value  Pr(>|z|)
## ar1                      -0.523537   0.469280 -1.1156 0.2645868
## ar2                      -0.054075   0.198848 -0.2719 0.7856671
## ar3                      -0.256493   0.147702 -1.7366 0.0824651 .
## ma1                       0.159265   0.466303  0.3415 0.7326901
## ma2                      -0.534879   0.323742 -1.6522 0.0984983 .
## sar1                      0.354045   0.103538  3.4195 0.0006274 ***
## train.WeightedDistribution  0.151135   0.025329  5.9669 2.418e-09 ***
## train.PPSU               11.366212   2.323672  4.8915 1.001e-06 ***
## train.WDPriceCut          0.794781   0.136822  5.8089 6.289e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
print(arima.fit$aic) # Print Akaike information criterion (AIC)
```

```
## [1] 242.3152
```

Wrap up everything we've just done for one SKU in a *crystal ball* function that we can run for any SKU:

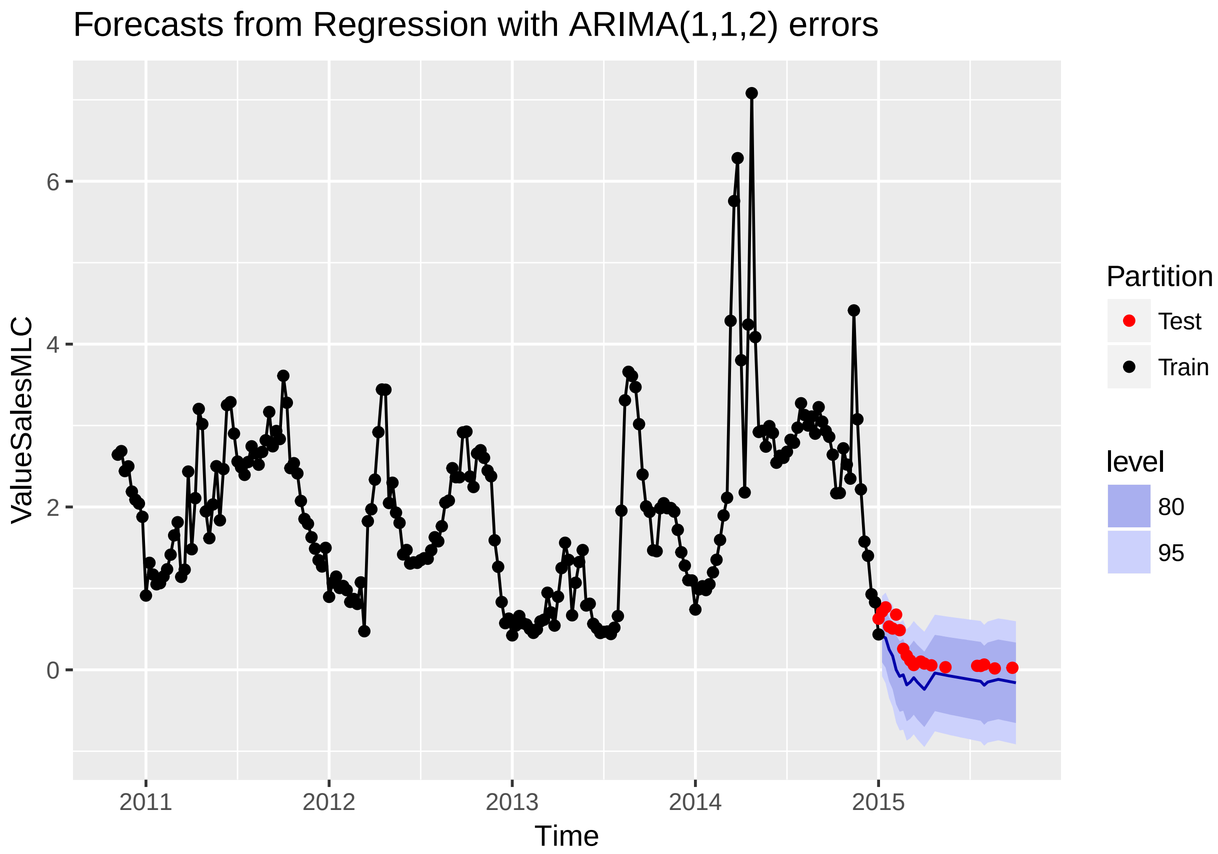
```
crystal.ball <- function(SKU.num, split.percent, verbose = FALSE) {
  products %>% filter(SKU == SKUs[SKU.num]) -> product

  begin <- range(product$Date)[1]
  end <- range(product$Date)[2]
  dates <- data.frame(Date = seq(from = begin, to = end, by = "week"))

  left_join(dates, product, by = "Date") %>%
    mutate( # Expand-out data information
      year = as.numeric(format(Date, format = "%Y")),
      week = week(Date) # Week number
    ) -> product

  p <- floor(split.percent * nrow(product))
  q <- nrow(product) - p

  in.train <- createTimeSlices(1:nrow(product), p, q)

  df <- as.data.frame(product)
  train <- df[unlist(in.train$train),]
  test <- df[unlist(in.train$test),]

  start.train <- c(train$year[1], train$week[1])
  start.test <- c(test$year[1], test$week[1])

  train.ts <- na.interp(
```

```
    ts(train$ValueSalesMLC, frequency = 52, start = start.train)
  )
  train.WeightedDistribution <- na.interp(
    ts(train$WeightedDistribution, frequency = 52, start = start.train)
  )
  train.PPSU <- na.interp(
    ts(train$PPSU, frequency = 52, start = start.train)
  )
  train.WDPriceCut <- na.interp(
    ts(train$WDPriceCut, frequency = 52, start = start.train)
  )
  train.xregs <- cbind(train.WeightedDistribution, train.PPSU, train.WDPriceCut)

  test.ts <- ts(test$ValueSalesMLC, frequency = 52, start = start.test)

  test.WeightedDistribution <- na.interp(
    ts(test$WeightedDistribution, frequency = 52, start = start.test)
  )
  test.PPSU <- na.interp(
    ts(test$PPSU, frequency = 52, start = start.test)
  )
  test.WDPriceCut <- na.interp(
    ts(test$WDPriceCut, frequency = 52, start = start.test)
  )
  test.xregs <- cbind(test.WeightedDistribution, test.PPSU, test.WDPriceCut)

  arima.fit <- auto.arima(train.ts, trace = FALSE, xreg = train.xregs)

  if(verbose) {
    print(coeftest(arima.fit))
    print(arima.fit$aic) # Print Akaike information criterion (AIC)
  }

  arima.forecast <- forecast(arima.fit, h = q, xreg = test.xregs)

  train.df <- as.data.frame(time(train.ts))
  names(train.df) <- "x"
  train.df$y <- as.vector(train.ts)
  train.df$Partition <- "Train"

  test.df <- as.data.frame(time(test.ts))
  names(test.df) <- "x"
  test.df$y <- as.vector(test.ts)
  test.df$Partition <- "Test"

  points.dat <- suppressWarnings(bind_rows(test.df, train.df))

  autoplot(arima.forecast) +
    ylab("ValueSalesMLC") +
    geom_point(data = points.dat, aes(x = x, y = y, colour = Partition)) +
    scale_color_manual(values = c("Train" = 'Black','Test' = 'Red'))
}
```

```r
crystal.ball(5, 0.85, verbose = TRUE)
```

```
##
## z test of coefficients:
##
##                              Estimate Std. Error  z value  Pr(>|z|)
## ar1                         0.8137498  0.0746154  10.9059 < 2.2e-16 ***
## ma1                        -1.2867231  0.1119447 -11.4943 < 2.2e-16 ***
## ma2                         0.3049328  0.1043563   2.9220  0.003478 **
## train.WeightedDistribution  0.0720436  0.0041545  17.3412 < 2.2e-16 ***
## train.PPSU                  0.0145742  0.0113041   1.2893  0.197302
## train.WDPriceCut            0.0148697  0.0033346   4.4592 8.227e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## [1] 24.98226
```

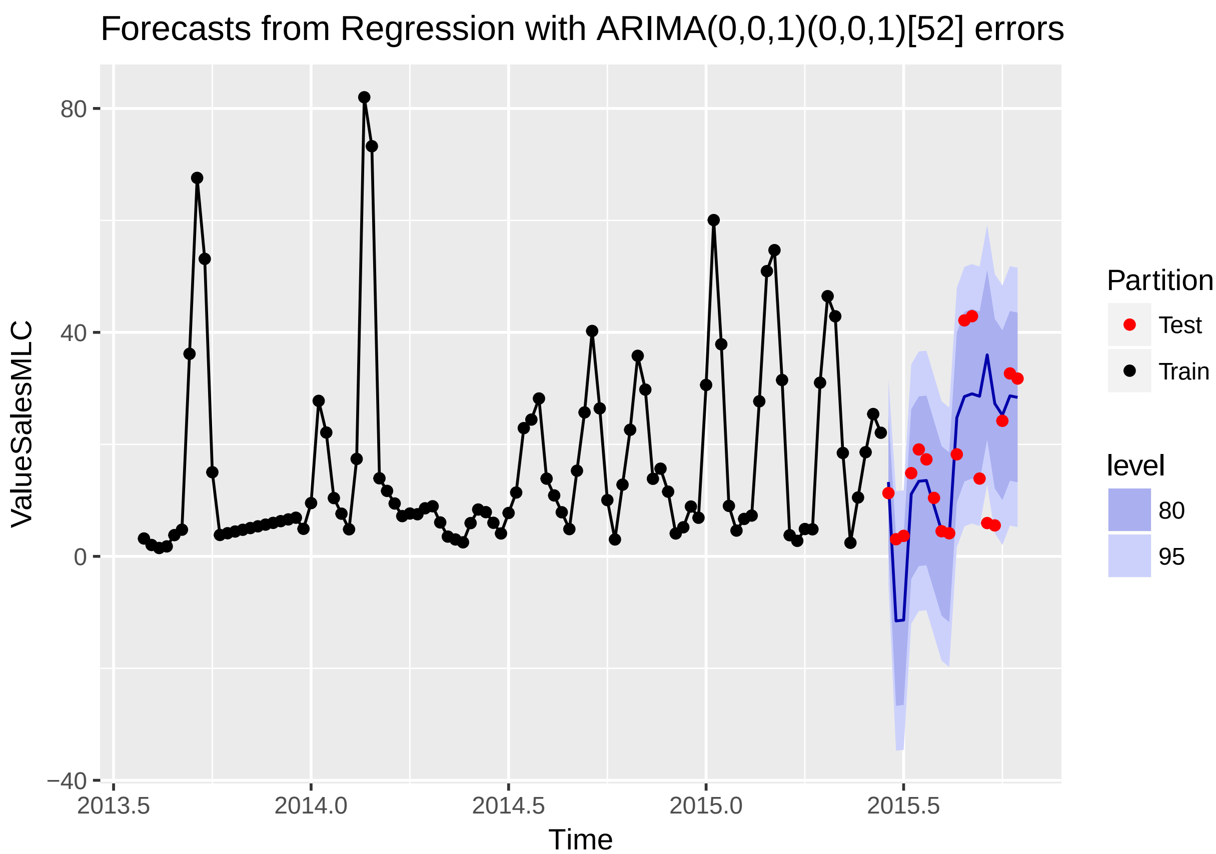
```
## Warning: Removed 19 rows containing missing values (geom_point).
```



Let's test our crystal ball function on some randomly-selected SKUs:

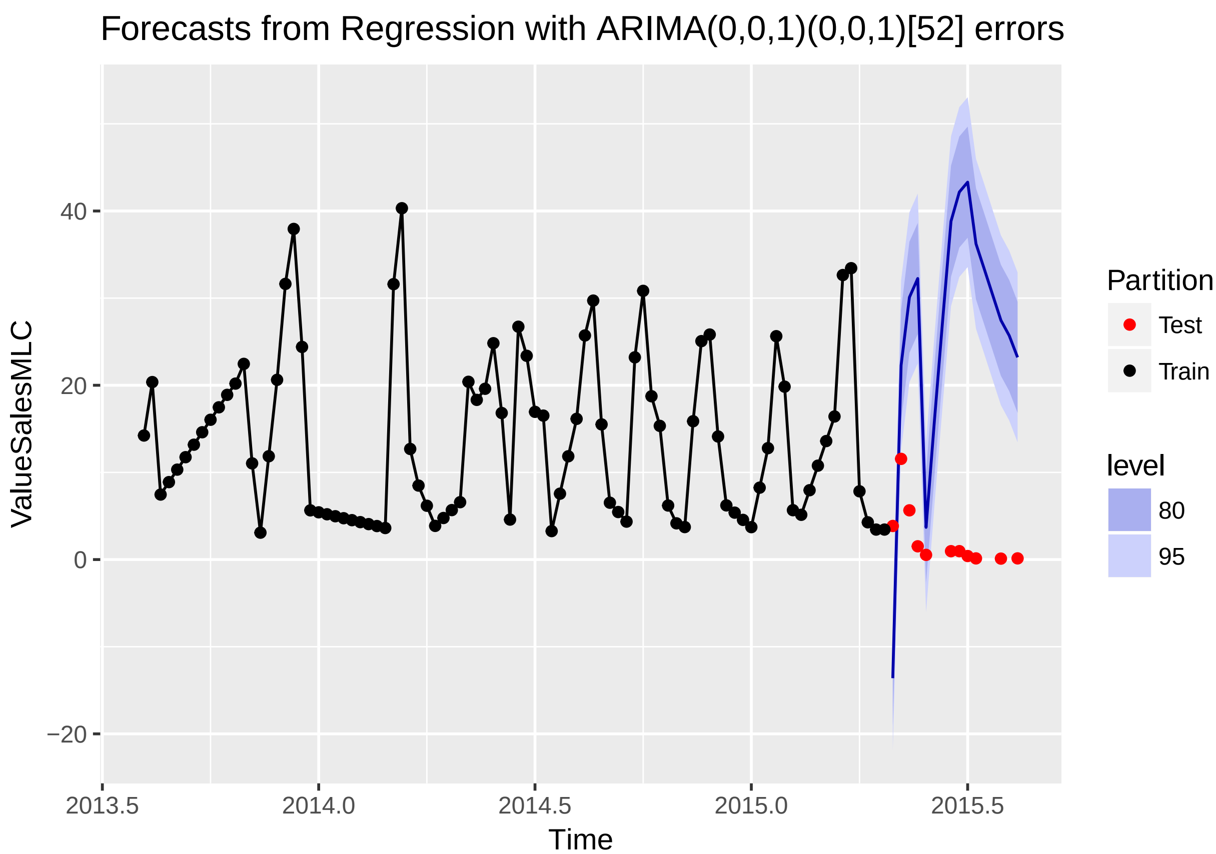
```r
random.SKUs <- sample(length(SKUs), size = 6, replace = FALSE)
lapply(random.SKUs, function(N) crystal.ball(N, 0.85))
```

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



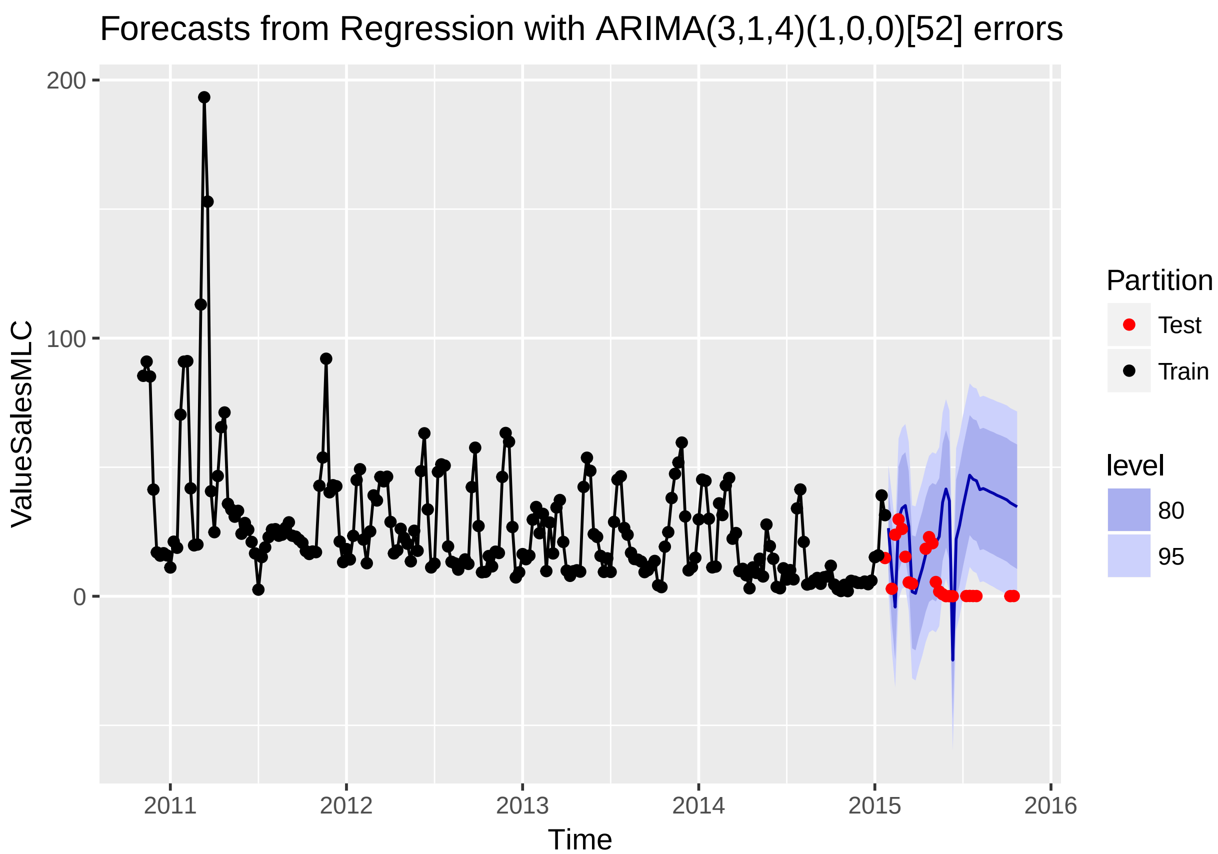

```
##
## [[2]]
```

```
## Warning: Removed 5 rows containing missing values (geom_point).
```



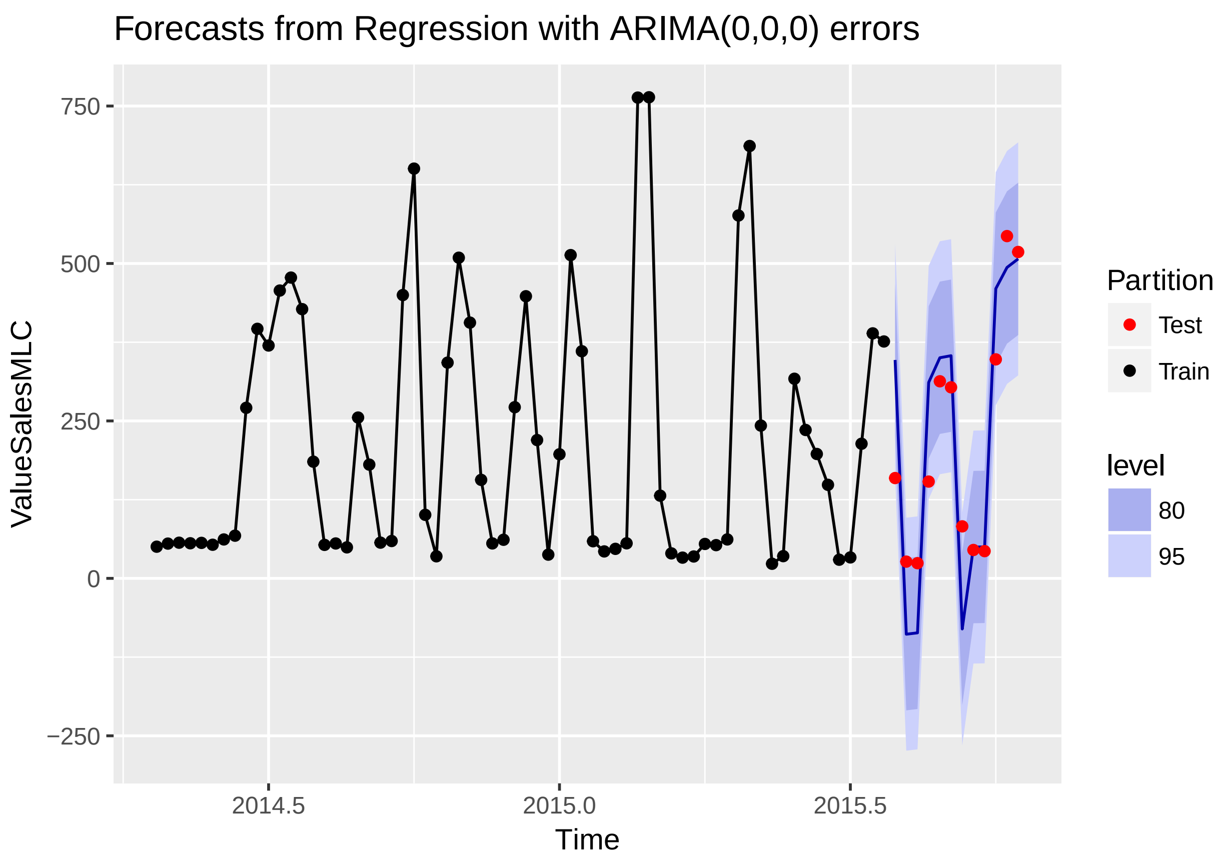

```
##
## [[3]]
```

```
## Warning: Removed 16 rows containing missing values (geom_point).
```



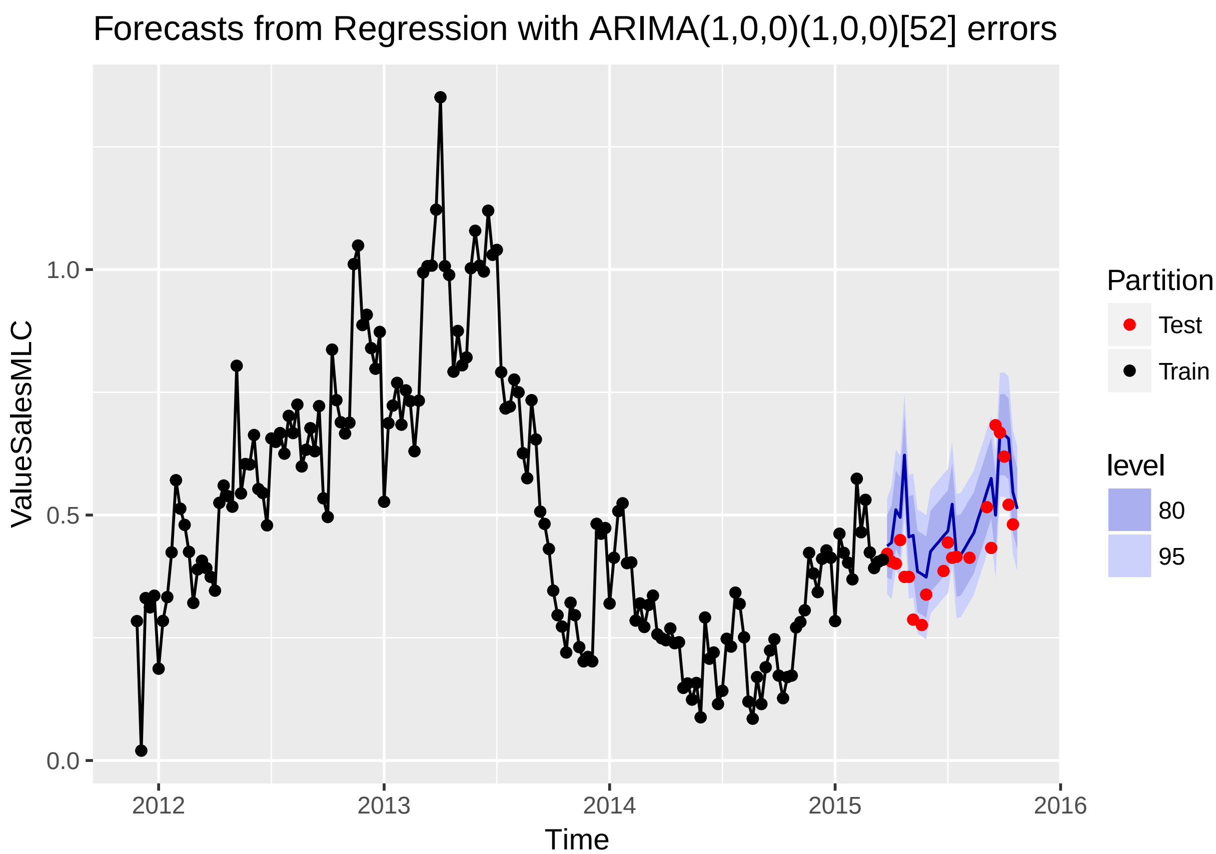
Forecasts from Regression with ARIMA(3,1,4)(1,0,0)[52] errors
200
100
0
ValueSalesMLC
2011
2012
2013
2014
2015
2016
Time
Partition
Test
Train
level
80
95


```
##
## [[4]]
```



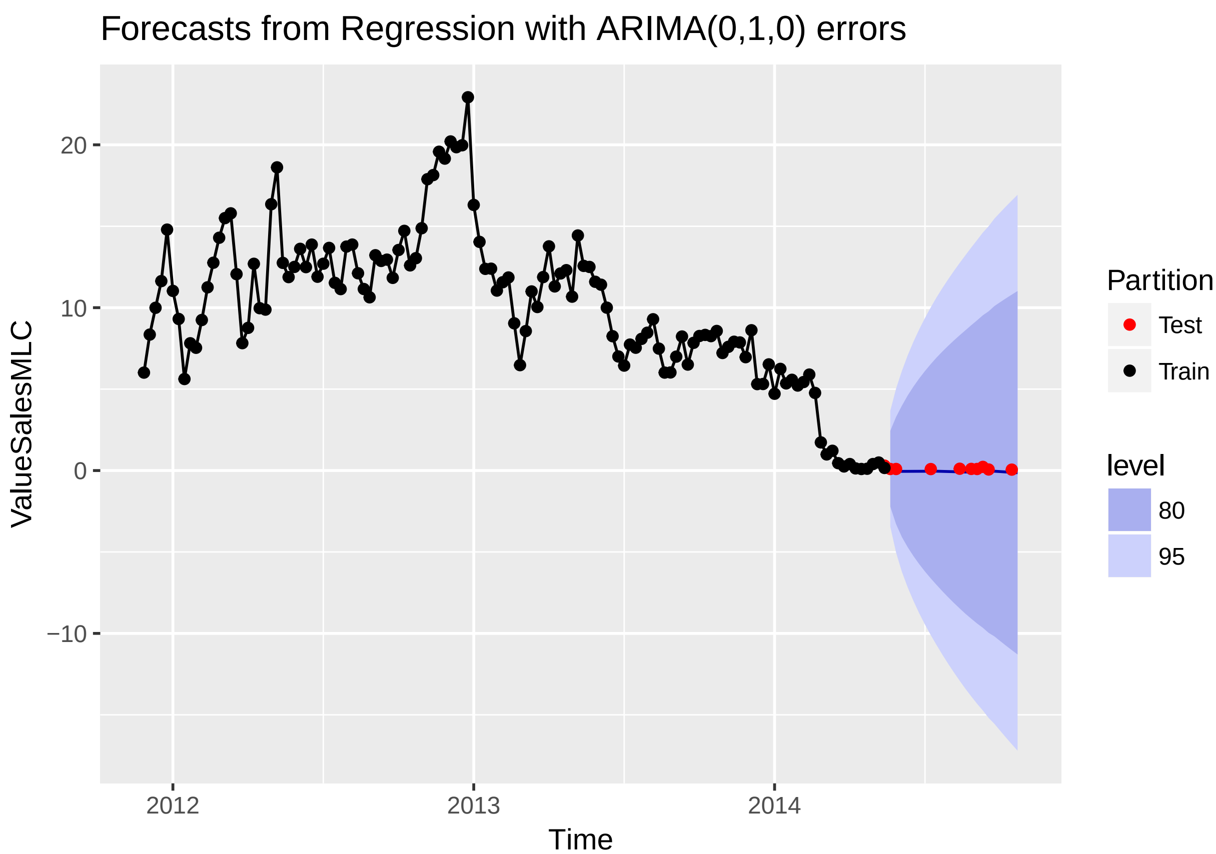

```
##
## [[5]]
```

```
## Warning: Removed 10 rows containing missing values (geom_point).
```




```
##
## [[6]]
```

```
## Warning: Removed 13 rows containing missing values (geom_point).
```



What could have been done better

We could introduce lagged versions of the explanatory variables into the model, and we could select the best model using AIC – we didn't do this here, but it would be reasonably trivial to implement with more time.