Forecast with automatic machine learning and other techniques.

Dr. Martín Lozano.

<martin.lozano@udem.edu>
https://sites.google.com/site/mlozanoqf/
https://github.com/mlozanoqf/

Last compiled on: 30/06/2021, 23:14:38.

Abstract

We tackle a single forecast problem using several techniques. This document is based on some freely available Business Science IO codes in R that explains how to implement machine learning workflow using H2O. Some mathematical background is skipped to emphasize the data analysis, model logic, discussion, graphical approach and R coding. As in the philosophy of Donald Knuth [Knuth, 1984], the objective of this document is to explain to human beings what we want a computer to do as literate programming. This is a work in progress and it is under revision.

Index.

1 Introduction.

This document relies on some freely available Business Science IO R codes in the web. The CEO of Business Science IO is Matt Dancho, he is the creator of `tidyquant` and `timetk`, and I truly believe we can learn a lot from their publicly available data science examples in general. This private firm declares a nice motivation that I fully support: *A gap exists between the data scientist's skillset and the business objectives. Organizations are investing heavily into data science hiring because they know that artificial intelligence, machine learning, and data science are the future. However, this investment takes time to pay off because data scientists need to learn the business and understand which problems are important to focus on. This is the gap. business science has developed methodologies, tools, and techniques to overcome the gap through our consulting program. Now, business science has opened these tools up to the public as a way to accelerate the growth of these powerful data scientists. It's this data scientist empowerment that motivates us.*

In this context, I think that this gap exists at the moment just as this firm declares. This gap is currently being addressed in two ways: data scientists (like engineers) are learning business, and business professionals are learning data science. I also think that this gap is far from being fully covered as researchers are producing more methods and theory, technology is putting the data science frontiers even further, there are more data than people who can analyze it, and business problems are becoming more complex. As a result, I consider we are obligated to understand business as good as we can because that is our main core, and at the same time learn data science as good as we can because that will allow us to propose innovative ways to tackle current and future business problems. This gap also represents a good reason to support constant professional training, and reveal the multidisciplinary requirements in the job market.

Machine learning is the study of computer algorithms that improve automatically through experience. There are many ways and many approaches to implement machine learning especially in time series forecasts purposes. This document heavily relies on `h2o` library. The `h2o` package is a product offered by H2O.ai that contains a number of cutting edge machine learning algorithms, performance metrics, and auxiliary functions to make machine learning both powerful and easy to implement.

One of the most important features of this package is the `h2o.automl` (Automatic Machine Learning). H2O's AutoML can be used for automating the machine learning workflow, which includes automatic training and tuning of many models within a user-specified time-limit. Stacked Ensembles – one based on all previously trained models, another one on the best model of each family – will be automatically trained on collections of individual models to produce highly predictive ensemble models which, in most cases, will be the top performing models in the AutoML Leaderboard. We can verify this in the example below.

This document has limited explanations about the applied machine learning techniques. The value of this document is to gather several examples that are originally presented separately in Business Science IO and r-bloggers.com sites and extend the analysis to elaborate further on the code logic and interpretation. It can also be useful to better understand how the R functions work, how results are produced, and it could help to replicate a different example with a new database for those who are new in the field.

You have to download and install H2O. Click here for full instructions. You are also expected to review the H2O webpage contents because they have important information that will allow you to better understand the value of this machine learning tool.

2 The forecast problem.

The problem is to forecast a time series. In particular, the time series is the *Beer, Wine, and Distilled Alcoholic Beverages Sales*. I did not pick this series by myself, this is taken from an existing example. I would rather prefer a milkshake or hot chocolate. The data is taken from FRED (Federal Reserve Economic Data). The data belongs to the non-durable goods category, it includes U.S. merchant wholesalers, except manufacturers' sales branches and offices sales. The monthly time series goes from 2010-01-01 to 2017-10-31. And the goal is to use 2017 data (10 months) as a test data to conduct the forecast.

For the full database details see: https://fred.stlouisfed.org/series/S4248SM144NCEN

Let's load the R packages.

```r
# Load libraries
library(h2o)        # Awesome ML Library.
library(timetk)     # Toolkit for working with time series in R.
library(tidyquant)  # Loads tidyverse, financial pkgs, used to get data.
library(dplyr)      # Database manipulation.
library(ggplot2)    # Nice plots.
library(tibble)     # Nice tables.
library(kableExtra) # Nicer tables.
library(knitr)      # I do not remember.
library(bit64)      # Useful in the machine learning workflow.
library(sweep)      # Broom-style tidiers for the forecast package.
library(forecast)   # Forecasting models and predictions package.
```

We can conveniently download the data directly from the FRED API in one line of code. Manually downloading the data and then importing into R is not considered *cool* anymore.

```
# Beer, Wine, Distilled Alcoholic Beverages, in Millions USD.
beer_sales_tbl <- tq_get("S4248SM144NCEN", get = "economic.data",
                         from = "2010-01-01", to = "2017-10-31")
```

Let's have a look of the data set. By default it says `price`, but these are basically sales figures in monetary terms. According to the main FRED reference, these are in millions of dollars, not seasonally adjusted.

```
# A quick look at the original data.
glimpse(beer_sales_tbl)
```

```
## Rows: 94
## Columns: 3
## $ symbol <chr> "S4248SM144NCEN", "S4248SM144NCEN", "S4248SM144NCEN", "S4248SM1~
## $ date   <date> 2010-01-01, 2010-02-01, 2010-03-01, 2010-04-01, 2010-05-01, 20~
## $ price  <int> 6558, 7481, 9475, 9424, 9351, 10552, 9077, 9273, 9420, 9413, 98~
```

Visualization is particularly important for time series analysis and forecasting. It's a good idea to identify spots where we will split the data into training, test and validation sets. This kind of split is consistent with most machine learning algorithms. The *training* dataset is the sample of data used to fit and train the model by learning from the data. The *validation* dataset is the sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The *test* dataset is the sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. The test dataset provides the gold standard used to evaluate the model. It is only used once a model is completely trained (using the train and validation sets). The test set is generally what is used to evaluate competing models.

It is also important to see the time series because normally the models will perform better if we can identify basic characteristics such as trend and seasonality. This data set clearly has a trend and a seasonality as people drink more alcohol in December. This time series is not very long, I would propose to expand the time length some more to unleash the H2O full potential.

```r
# Plot Beer Sales with train, validation, and test sets shown.
beer_sales_tbl %>%
  ggplot(aes(date, price)) +
  # Train Region:
  annotate("text", x = ymd("2013-01-01"), y = 14000,
           color = palette_light()[[1]], label = "Train Region") +
  # Validation Region:
  geom_rect(xmin = as.numeric(ymd("2016-01-01")),
            xmax = as.numeric(ymd("2016-12-31")), ymin = 0, ymax = Inf,
            alpha = 0.02, fill = palette_light()[[3]]) +
  annotate("text", x = ymd("2016-07-01"), y = 7000,
           color = palette_light()[[1]], label = "Validation\nRegion") +
  # Test Region:
  geom_rect(xmin = as.numeric(ymd("2017-01-01")),
            xmax = as.numeric(ymd("2017-10-31")), ymin = 0, ymax = Inf,
            alpha = 0.02, fill = palette_light()[[4]]) +
  annotate("text", x = ymd("2017-06-01"), y = 7000,
           color = palette_light()[[1]], label = "Test\nRegion") +
  # Data.
  geom_line(col = palette_light()[1]) +
  geom_point(col = palette_light()[1]) +
  # Aesthetics.
  theme_tq() +
  scale_x_date(date_breaks = "1 year", date_labels = "%Y") +
  labs(title = "Beer Sales: 2010 through 2017-10-31",
       subtitle =
  "Train, Validation (2016), and Test Sets (2017-01-01 to 2017-10-31)",
       caption = "The models do not know the test region, this is for us
       to see how well the models do the 10-month ahead forecast.")
```

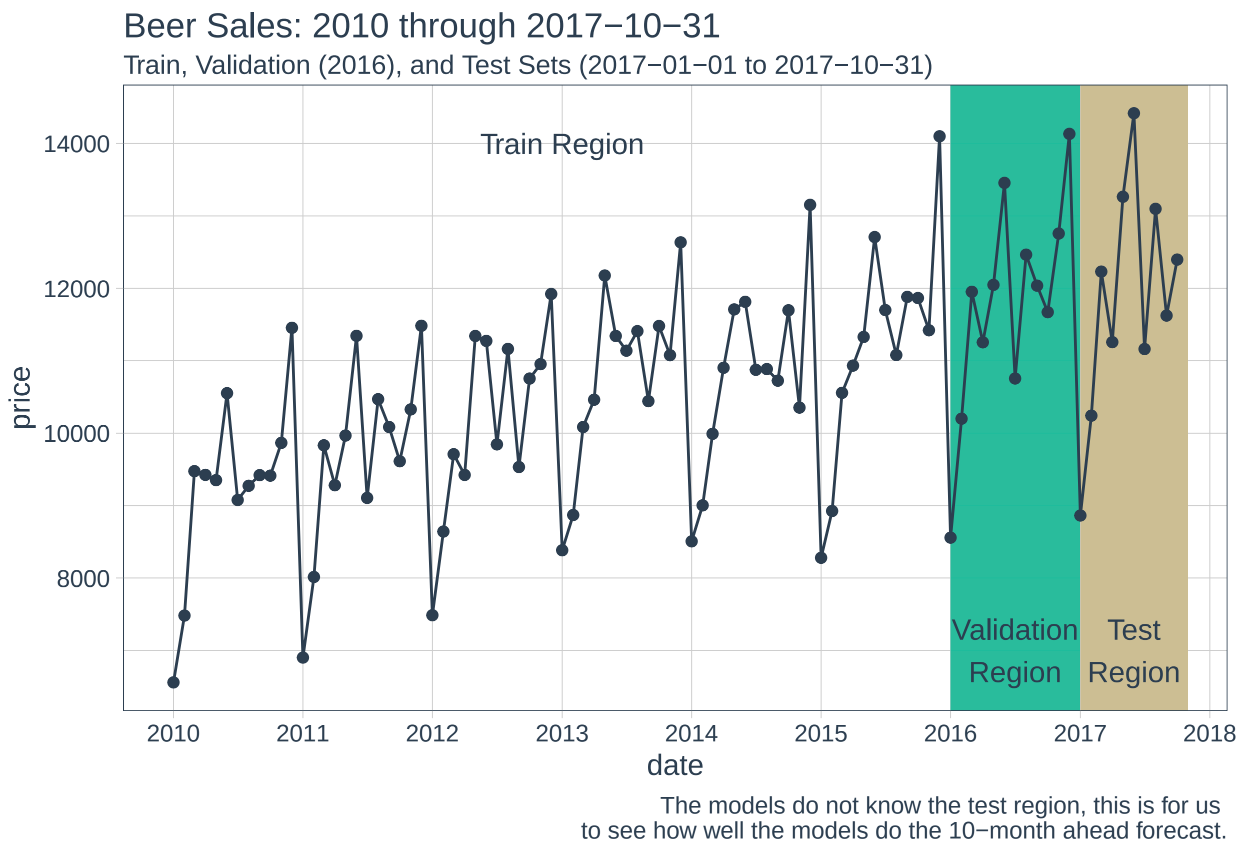

Figure 2.0.1: Beer, Wine, and Distilled Alcoholic Beverages Sales.

Then, the problem is to forecast the 10 months of the test region. This is, from January to October 2017.

We will do that by implementing a battery of forecasting techniques:

- H2O machine learning.
- Linear regression.
- ARIMA.

3 H2O machine learning.

The main objective here is to use `h2o` locally (in your own computer) to develop a high accuracy time series model on the `beer_sales_tbl` data set. This is a supervised machine learning regression problem. An interesting reference to learn the basics of supervised and unsupervised machine learning techniques applied to business is: Machine Learning in

Business: An Introduction to the World of Data Science (2019), by John C. Hull.

3.1 Prepare the data.

The tk_augment_timeseries_signature function expands out the timestamp information column-wise into a machine learning feature set, adding columns of time series information to the original data frame. We'll again use glimpse for quick inspection of this expansion. See how there are now 31 features extracted from the original database. Not all will be important for the final and chosen models, but some will.

```
# See the full list of new variables to realize the expansion effect.
beer_sales_tbl_aug <- beer_sales_tbl %>%
  tk_augment_timeseries_signature() %>%
  glimpse()
```

```
## Rows: 94
## Columns: 31
## $ symbol    <chr> "S4248SM144NCEN", "S4248SM144NCEN", "S4248SM144NCEN", "S4248~
## $ date      <date> 2010-01-01, 2010-02-01, 2010-03-01, 2010-04-01, 2010-05-01,~
## $ price     <int> 6558, 7481, 9475, 9424, 9351, 10552, 9077, 9273, 9420, 9413,~
## $ index.num <dbl> 1262304000, 1264982400, 1267401600, 1270080000, 1272672000, ~
## $ diff      <dbl> NA, 2678400, 2419200, 2678400, 2592000, 2678400, 2592000, 26~
## $ year      <int> 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, ~
## $ year.iso  <int> 2009, 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, ~
## $ half      <int> 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, ~
## $ quarter   <int> 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 1, 1, 1, 2, 2, 2, 3, 3, ~
## $ month     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7, ~
## $ month.xts <int> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 1, 2, 3, 4, 5, 6, 7~
## $ month.lbl <ord> enero, febrero, marzo, abril, mayo, junio, julio, agosto, se~
## $ day       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ hour      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ minute    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ second    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ hour12    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ am.pm     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ wday      <int> 6, 2, 2, 5, 7, 3, 5, 1, 4, 6, 2, 4, 7, 3, 3, 6, 1, 4, 6, 2, ~
## $ wday.xts  <int> 5, 1, 1, 4, 6, 2, 4, 0, 3, 5, 1, 3, 6, 2, 2, 5, 0, 3, 5, 1, ~
## $ wday.lbl  <ord> viernes, lunes, lunes, jueves, sábado, martes, jueves, domin~
## $ mday      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ qday      <int> 1, 32, 60, 1, 31, 62, 1, 32, 63, 1, 32, 62, 1, 32, 60, 1, 31~
## $ yday      <int> 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 1, 32~
## $ mweek     <int> 5, 4, 4, 5, 5, 5, 5, 5, 5, 5, 4, 5, 5, 5, 5, 5, 4, 5, 5, 4, ~
## $ week      <int> 1, 5, 9, 13, 18, 22, 26, 31, 35, 40, 44, 48, 1, 5, 9, 13, 18~
## $ week.iso  <int> 53, 5, 9, 13, 17, 22, 26, 30, 35, 39, 44, 48, 52, 5, 9, 13, ~
## $ week2     <int> 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, ~
## $ week3     <int> 1, 2, 0, 1, 0, 1, 2, 1, 2, 1, 2, 0, 1, 2, 0, 1, 0, 1, 2, 1, ~
## $ week4     <int> 1, 1, 1, 1, 2, 2, 2, 3, 3, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 3, ~
## $ mday7     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
```

Note how we went from 3 columns in `beer_sales_tbl` to 31 columns in `beer_sales_tbl_aug`, almost *out of the blue*.

We need to prepare the data in a format for H2O. First, let's remove any unnecessary columns such as dates or those with missing values, and change the ordered classes to plain factors. We prefer `dplyr` operations for these steps. Sometimes we do not need to implement this step as the data is already clean (as in this case), but sometimes it is not. Thus, let's clean the data.

```
# See the full list of variables to realize the cleaning effect.
beer_sales_tbl_clean <- beer_sales_tbl_aug %>%
  select_if(~ !is.Date(.)) %>%
  select_if(~ !any(is.na(.))) %>%
  mutate_if(is.ordered, ~ as.character(.) %>% as.factor) %>%
  glimpse()
```

```
## Rows: 94
## Columns: 29
## $ symbol    <chr> "S4248SM144NCEN", "S4248SM144NCEN", "S4248SM144NCEN", "S4248~
## $ price     <int> 6558, 7481, 9475, 9424, 9351, 10552, 9077, 9273, 9420, 9413,~
## $ index.num <dbl> 1262304000, 1264982400, 1267401600, 1270080000, 1272672000, ~
## $ year      <int> 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, ~
## $ year.iso  <int> 2009, 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, ~
## $ half      <int> 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, ~
## $ quarter   <int> 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 1, 1, 1, 2, 2, 2, 3, 3, ~
## $ month     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7, ~
## $ month.xts <int> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 1, 2, 3, 4, 5, 6, 7~
## $ month.lbl <fct> enero, febrero, marzo, abril, mayo, junio, julio, agosto, se~
## $ day       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ hour      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ minute    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ second    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ hour12    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ am.pm     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ wday      <int> 6, 2, 2, 5, 7, 3, 5, 1, 4, 6, 2, 4, 7, 3, 3, 6, 1, 4, 6, 2, ~
## $ wday.xts  <int> 5, 1, 1, 4, 6, 2, 4, 0, 3, 5, 1, 3, 6, 2, 2, 5, 0, 3, 5, 1, ~
## $ wday.lbl  <fct> viernes, lunes, lunes, jueves, sábado, martes, jueves, domin~
## $ mday      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ qday      <int> 1, 32, 60, 1, 31, 62, 1, 32, 63, 1, 32, 62, 1, 32, 60, 1, 31~
## $ yday      <int> 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 1, 32~
## $ mweek     <int> 5, 4, 4, 5, 5, 5, 5, 5, 5, 5, 4, 5, 5, 5, 5, 5, 4, 5, 5, 4, ~
## $ week      <int> 1, 5, 9, 13, 18, 22, 26, 31, 35, 40, 44, 48, 1, 5, 9, 13, 18~
## $ week.iso  <int> 53, 5, 9, 13, 17, 22, 26, 30, 35, 39, 44, 48, 52, 5, 9, 13, ~
## $ week2     <int> 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, ~
## $ week3     <int> 1, 2, 0, 1, 0, 1, 2, 1, 2, 1, 2, 0, 1, 2, 0, 1, 0, 1, 2, 1, ~
## $ week4     <int> 1, 1, 1, 1, 2, 2, 2, 3, 3, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 3, ~
## $ mday7     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
```

The database did not change too much. Now we have 29 columns in `beer_sales_tbl_clean`. In the case of two variables, the structure ordered factors `<ord>` changed into factors `<fct>`, which is necessary for some H2O functions.

Let's split the database into a training, validation and test sets following the time ranges in the visualization above. These training sets are the way most machine learning algorithms can be implemented and evaluated. We normally take more observations for the training, and less observations for the validation and test. The test set (the most recent dates) is unknown in the learning process of the models, the test set will be useful for us to be able to compare forecasts versus what really happened. This is how we can measure out-of-sample estimation errors.

```r
# Split into training, validation and test sets.
train_tbl <- beer_sales_tbl_clean %>% filter(year < 2016)
valid_tbl <- beer_sales_tbl_clean %>% filter(year == 2016)
test_tbl  <- beer_sales_tbl_clean %>% filter(year == 2017)
```

```
glimpse(test_tbl)
```

```
## Rows: 10
## Columns: 29
## $ symbol    <chr> "S4248SM144NCEN", "S4248SM144NCEN", "S4248SM144NCEN", "S4248~
## $ price     <int> 8863, 10242, 12231, 11257, 13265, 14418, 11162, 13098, 11624~
## $ index.num <dbl> 1483228800, 1485907200, 1488326400, 1491004800, 1493596800, ~
## $ year      <int> 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017
## $ year.iso  <int> 2016, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017
## $ half      <int> 1, 1, 1, 1, 1, 1, 2, 2, 2, 2
## $ quarter   <int> 1, 1, 1, 2, 2, 2, 3, 3, 3, 4
## $ month     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
## $ month.xts <int> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
## $ month.lbl <fct> enero, febrero, marzo, abril, mayo, junio, julio, agosto, se~
## $ day       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
## $ hour      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
## $ minute    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
## $ second    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
## $ hour12    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
## $ am.pm     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
## $ wday      <int> 1, 4, 4, 7, 2, 5, 7, 3, 6, 1
## $ wday.xts  <int> 0, 3, 3, 6, 1, 4, 6, 2, 5, 0
## $ wday.lbl  <fct> domingo, miércoles, miércoles, sábado, lunes, jueves, sábado~
## $ mday      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
## $ qday      <int> 1, 32, 60, 1, 31, 62, 1, 32, 63, 1
## $ yday      <int> 1, 32, 60, 91, 121, 152, 182, 213, 244, 274
## $ mweek     <int> 5, 5, 5, 5, 4, 5, 5, 5, 5, 4
## $ week      <int> 1, 5, 9, 13, 18, 22, 26, 31, 35, 40
## $ week.iso  <int> 52, 5, 9, 13, 18, 22, 26, 31, 35, 39
## $ week2     <int> 1, 1, 1, 1, 0, 0, 0, 1, 1, 0
## $ week3     <int> 1, 2, 0, 1, 0, 1, 2, 1, 2, 1
## $ week4     <int> 1, 1, 1, 1, 2, 2, 2, 3, 3, 0
## $ mday7     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

Remember our goal is to forecast the first 10 months of 2017.

3.2 Prepare for H2O.

First, fire up H2O. This will initialize the Java Virtual Machine (JVM) that H2O uses locally. In simple terms, here your local computer will remotely connect to a high-power clusters to do the H2O machine learning job. This is not only amazing, it is also free.

```
h2o.init() # Fire up h2o.
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     C:\Users\ML\AppData\Local\Temp\RtmpQtt0u5\file105027513a92/h2o_ML_started_from_r.
##     C:\Users\ML\AppData\Local\Temp\RtmpQtt0u5\file105018a13133/h2o_ML_started_from_r.
##
##
## Starting H2O JVM and connecting: . Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         6 seconds 365 milliseconds
##     H2O cluster timezone:       America/Mexico_City
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.32.1.3
##     H2O cluster version age:    1 month and 11 days
##     H2O cluster name:           H2O_started_from_R_ML_snx817
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.54 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Cor
##     R Version:                  R version 4.1.0 (2021-05-18)
```

We need the data sets in a format that can be readable by H2O. This is an easy step.

```r
# Convert to H2OFrame objects.
h2o.no_progress() # We do not need a progress bar here.
train_h2o <- as.h2o(train_tbl)
valid_h2o <- as.h2o(valid_tbl)
test_h2o  <- as.h2o(test_tbl)
```

Let's list the names of the variables.

```r
# Set names for h2o.
y <- "price"
x <- setdiff(names(train_h2o), y) # Adds price to the names list.
kable(matrix(x, 7, 4), caption = "Summary of variable names.") %>%
kable_styling(latex_options = "HOLD_position")
```

Table 3.2.1: Summary of variable names.

symbol	month.xts	am.pm	mweek
index.num	month.lbl	wday	week
year	day	wday.xts	week.iso
year.iso	hour	wday.lbl	week2
half	minute	mday	week3
quarter	second	qday	week4
month	hour12	yday	mday7

The `h2o.automl` is a function in H2O that automates the process of building a large number of models, with the goal of finding the *best* model without any prior knowledge or effort by the data scientist. The alternative of using `h2o.automl` is to pick some models according to the database characteristics, implement the models, and pick the one with the best performance according to some evaluation criterion. This alternative is time consuming and it could use an intensive computational memory and power, this is why H2O is valuable. If H2O was already amazing, this function makes it even more powerful.

The available algorithms that `h2o.automl` currently run and compare are (click on each one to see a full description):

- Distributed Random Forest (DRF).
- Generalized Linear Model (GLM).
- XGBoost.

- Gradient Boosting Machine (GBM).
- Deep Learning (Neural Networks).
- Stacked Ensembles.

It is a good time to define how we are going to use some concepts at least in this document. Here, we call forecasting *techniques* to the three techniques implemented in this document: machine learning using H2O, linear regression, and ARIMA. When we implement `h2o.automl` function, H2O test for the six *algorithms* listed above. Each algorithm includes many other *models* that belongs to these algorithms in the machine learning process. The result of `h2o.automl` is one model that belongs to one algorithm. This is the difference between forecasting techniques, algorithms, and models.

3.3 Implement `h2o.automl`.

Here, we implement the `h2o.automl` in three different ways because of reproducibility issues. Reproducibility means obtaining consistent computational results using the same input data, computational steps, methods, code, and conditions of analysis. It turns out that Deep Learning cannot be reproducible by construction. Then, we first apply `h2o.automl` without Deep Learning. Second, we apply `h2o.automl` with only Deep Learning (here the results will be different each time we run the code). And third, including all available algorithms in `h2o.automl` (again, the results might change every time we run the code). The first is the only one which can be reproducible and the other two are expected to change every time we run the R code.

Please note that in the code below we set `exclude_algos` to exclude Deep Learning, and `seed = 236` to make sure every time we run the code we can get the same results.

```r
# This might take some time to run.
automl_models_h2o <- h2o.automl(x = x, y = y, training_frame = train_h2o,
    validation_frame = valid_h2o, leaderboard_frame = test_h2o,
    exclude_algos = c("DeepLearning"), # without Deep Learning.
    #max_models = 10, # We can adjust this to save time.
    max_runtime_secs = 60, stopping_metric = "deviance", seed = 236)
```

```
##
## 23:15:09.779: User specified a validation frame with cross-validation still enabled.
## 23:15:09.800: AutoML: XGBoost is not available; skipping it.
## 23:15:16.258: Skipping training of model GBM_5_AutoML_20210630_231509 due to exceptio
```

The selected model by `h2o.automl` is:

```
# Extract leader model.
automl_leader <- automl_models_h2o@leader
automl_leader@algorithm
```

```
## [1] "stackedensemble"
```

See why Gradient Boosting Machine (GBM) was the chosen one:

```
# Show the first 10.
kable(head(automl_models_h2o@leaderboard, 10),
caption = "Model rankings: h2o.automl without Deep Learning
algorithm.", digits = 2, row.names = TRUE) %>%
kable_styling(font_size = 7, latex_options = "HOLD_position")
```

Table 3.3.1: Model rankings: h2o.automl without Deep Learning algorithm.

	model_id	mean_residual_deviance	rmse	mse	mae	rmsle
1	StackedEnsemble_BestOfFamily_AutoML_20210630_231509	602105.3	775.95	602105.3	639.91	0.07
2	GBM_grid__1_AutoML_20210630_231509_model_16	612170.1	782.41	612170.1	631.42	0.07
3	GBM_grid__1_AutoML_20210630_231509_model_27	684793.7	827.52	684793.7	759.00	0.08
4	GBM_grid__1_AutoML_20210630_231509_model_10	694709.6	833.49	694709.6	667.91	0.07
5	GBM_grid__1_AutoML_20210630_231509_model_45	732765.4	856.02	732765.4	712.03	0.07
6	GBM_grid__1_AutoML_20210630_231509_model_12	768529.2	876.66	768529.2	717.52	0.08
7	GBM_grid__1_AutoML_20210630_231509_model_19	791223.7	889.51	791223.7	710.53	0.07
8	GBM_grid__1_AutoML_20210630_231509_model_22	800591.7	894.76	800591.7	782.32	0.07
9	GBM_grid__1_AutoML_20210630_231509_model_55	802136.2	895.62	802136.2	731.96	0.07
10	GBM_grid__1_AutoML_20210630_231509_model_50	804518.2	896.95	804518.2	753.34	0.07

The `model_id` column list the top 10 models with the lowest errors. The value of `h2o.automl` is that we can take the best model and use it to conduct our forecast. Remember we proposed to run `h2o.automl` three times. Now let's consider the second alternative (only Deep Learning). There are several ways to implement Deep Learning, this is why it makes sense to use only this family into the `h2o.automl` function. Deep Learning cannot be reproducible by construction so adding a seed in this case would be useless.

```
# This might take some time to run.
DL <- h2o.automl(x = x, y = y, training_frame = train_h2o,
    validation_frame = valid_h2o, leaderboard_frame = test_h2o,
    include_algos = c("DeepLearning"), max_runtime_secs = 60,
    stopping_metric = "deviance")
```

```
##
```

```
## 23:16:01.436: User specified a validation frame with cross-validation still enabled.
```

The selected model by `h2o.automl` is:

```r
# Extract leader model
automl_DL <- DL@leader
automl_DL@algorithm
```

```
## [1] "deeplearning"
```

See why this specific Deep Learning model was the chosen one:

```r
kable(DL@leaderboard,
  caption = "Model rankings: h2o.automl with only Deep Learning algorithm.",
  digits = 2, row.names = TRUE) %>%
  kable_styling(font_size = 7, latex_options = "HOLD_position")
```

Table 3.3.2: Model rankings: h2o.automl with only Deep Learning algorithm.

	model_id	mean_residual_deviance	rmse	mse	mae	rmsle
1	DeepLearning_grid__2_AutoML_20210630_231601_model_4	275370.8	524.76	275370.8	358.87	0.04
2	DeepLearning_grid__1_AutoML_20210630_231601_model_2	613708.2	783.40	613708.2	572.82	0.06
3	DeepLearning_1_AutoML_20210630_231601	744424.6	862.80	744424.6	699.94	0.07
4	DeepLearning_grid__1_AutoML_20210630_231601_model_5	987800.9	993.88	987800.9	818.87	0.08
5	DeepLearning_grid__1_AutoML_20210630_231601_model_4	1005842.4	1002.92	1005842.4	811.50	0.08
6	DeepLearning_grid__2_AutoML_20210630_231601_model_2	1135396.2	1065.55	1135396.2	822.00	0.09
7	DeepLearning_grid__3_AutoML_20210630_231601_model_2	1387547.1	1177.94	1387547.1	896.37	0.10
8	DeepLearning_grid__1_AutoML_20210630_231601_model_1	1399466.2	1182.99	1399466.2	949.85	0.10
9	DeepLearning_grid__2_AutoML_20210630_231601_model_3	1550736.2	1245.29	1550736.2	1006.43	0.10
10	DeepLearning_grid__3_AutoML_20210630_231601_model_3	1904045.8	1379.87	1904045.8	1157.04	0.12
11	DeepLearning_grid__2_AutoML_20210630_231601_model_1	2333735.6	1527.66	2333735.6	1312.98	0.13
12	DeepLearning_grid__3_AutoML_20210630_231601_model_1	3193936.4	1787.16	3193936.4	1513.29	0.15
13	DeepLearning_grid__1_AutoML_20210630_231601_model_3	5040375.7	2245.08	5040375.7	1848.89	0.20

All models belong to the same algorithm, but we clearly choose the first one of the list. The machine learning workflow estimate a number of models using the train region and evaluate them using the validation region. The estimated model parameters then change as they learn from their mistakes. This process is repeated until a specific restriction meets, in this case `max_runtime_secs` is set to 60 seconds. At the end, we select the best ranked model.

Now let's consider the third alternative. This is, run `h2o.automl` with no restrictions at all. Here, it would be interesting to see if this led to the best alternative. In principle, we cannot anticipate which one of these three runs will be the best. This is because the Deep Learning algorithm has a random component which might lead to better results, and remember the second round was exclusive for Deep Learning and the third includes Deep Learning. Then, every time I compile this document or run this R code we should expect different results in the second and third alternative.

```
# This might take some time to run.
automl_models_h2o_all <- h2o.automl(x = x, y = y,
    training_frame = train_h2o, validation_frame = valid_h2o,
    leaderboard_frame = test_h2o, max_runtime_secs = 60,
    stopping_metric = "deviance")
```

```
##
## 23:17:16.55: User specified a validation frame with cross-validation still enabled. P
## 23:17:16.56: AutoML: XGBoost is not available; skipping it.
## 23:17:23.426: Skipping training of model GBM_5_AutoML_20210630_231716 due to exceptio
```

The selected model by `h2o.automl` is:

```
# Extract leader model
automl_leader_all <- automl_models_h2o_all@leader
automl_leader_all@algorithm
```

```
## [1] "deeplearning"
```

See why deeplearning model was the chosen one in this specific and unique code compilation:

```
# Let's show the first 10 of the list.
kable(head(automl_models_h2o_all@leaderboard, 10),
    caption = "Model rankings: h2o.automl with all available algorithms.",
    digits = 2, row.names = TRUE) %>%
  kable_styling(font_size = 7, latex_options = "HOLD_position")
```

Table 3.3.3: Model rankings: h2o.automl with all available algorithms.

	model_id	mean_residual_deviance	rmse	mse	mae	rmsle
1	DeepLearning_grid__1_AutoML_20210630_231716_model_2	397184.1	630.23	397184.1	462.00	0.05
2	DeepLearning_grid__1_AutoML_20210630_231716_model_5	570671.0	755.43	570671.0	604.18	0.06
3	GBM_grid__1_AutoML_20210630_231716_model_22	690874.0	831.19	690874.0	645.89	0.07
4	GBM_grid__1_AutoML_20210630_231716_model_15	757492.2	870.34	757492.2	698.13	0.07
5	GBM_grid__1_AutoML_20210630_231716_model_7	779513.9	882.90	779513.9	729.57	0.07
6	DeepLearning_grid__3_AutoML_20210630_231716_model_4	791789.9	889.83	791789.9	747.52	0.08
7	GBM_grid__1_AutoML_20210630_231716_model_38	813949.1	902.19	813949.1	757.64	0.07
8	GBM_grid__1_AutoML_20210630_231716_model_18	828229.0	910.07	828229.0	745.26	0.07
9	GBM_grid__1_AutoML_20210630_231716_model_30	837968.2	915.41	837968.2	757.11	0.07
10	GBM_grid__1_AutoML_20210630_231716_model_1	885476.1	941.00	885476.1	730.79	0.08

Let's summarize the results according to the mean residual deviance as this was the criterion in stopping_metric. The table shows the best ranked model according to our three different runs of h2o.automl.

```
# Collect model names and the mean residual deviance.
without_DL <- c(automl_leader@algorithm,
                round(automl_models_h2o@leaderboard[1, 2], 2))
only_DL <- c(automl_DL@algorithm,
             round(DL@leaderboard[1,2], 2))
all <- c(automl_leader_all@algorithm,
         round(automl_models_h2o_all@leaderboard[1, 2], 2))
# Three different runs of h2o.automl.
automl_three <- data.frame(without_DL, only_DL, all)
colnames(automl_three) <- c("Without Deep Learning", "Only Deep Learning",
                            "All algorithms")
kable(automl_three,
caption = "Top ranked models: h2o.automl mean residual deviance.") %>%
kable_styling(latex_options = "HOLD_position")
```

Table 3.3.4: Top ranked models: h2o.automl mean residual deviance.

Without Deep Learning	Only Deep Learning	All algorithms
stackedensemble	deeplearning	deeplearning
602105.35	275370.81	397184.12

This is interesting because this suggest that it makes sense to run the H2O more than one time. It would be good to test for a different stopping_metric, max_runtime_secs and max_models.

3.4 Predict.

Here are how the forecasts are calculated.

```
# The h2o.predict function do the job.
pred_h2o <- h2o.predict(automl_leader, newdata = test_h2o)
pred_h2o_DL <- h2o.predict(automl_DL, newdata = test_h2o)
pred_h2o_all <- h2o.predict(automl_leader_all, newdata = test_h2o)
```

Let's show the results in a table. First, the case without Deep Learning.

```
# 10-period forecast error: h2o.automl without Deep Learning.
error_tbl <- beer_sales_tbl %>%
    filter(lubridate::year(date) == 2017) %>%
    add_column(pred = pred_h2o %>% as_tibble() %>% pull(predict)) %>%
    rename(actual = price) %>%
    mutate(error = actual - pred, error_pct = error / actual)
kable(error_tbl,
caption = "Detailed performance: h2o.automl without Deep Learning algorithm.",
digits = 3, row.names = TRUE) %>%
kable_styling(latex_options = "HOLD_position")
```

Table 3.4.1: Detailed performance: h2o.automl without Deep Learning algorithm.

	symbol	date	actual	pred	error	error_pct
1	S4248SM144NCEN	2017-01-01	8863	8096.278	766.722	0.087
2	S4248SM144NCEN	2017-02-01	10242	8980.955	1261.045	0.123
3	S4248SM144NCEN	2017-03-01	12231	11627.455	603.545	0.049
4	S4248SM144NCEN	2017-04-01	11257	11364.748	-107.748	-0.010
5	S4248SM144NCEN	2017-05-01	13265	12721.607	543.393	0.041
6	S4248SM144NCEN	2017-06-01	14418	12974.305	1443.695	0.100
7	S4248SM144NCEN	2017-07-01	11162	11801.853	-639.853	-0.057
8	S4248SM144NCEN	2017-08-01	13098	12295.149	802.851	0.061
9	S4248SM144NCEN	2017-09-01	11624	11798.532	-174.532	-0.015
10	S4248SM144NCEN	2017-10-01	12397	12452.718	-55.718	-0.004

The forecast looks good. Note that in some cases it over-estimate and in others under-estimate the real values, but in general these differences are small. Now, let's look at the same information in a plot.

```
# H2O without Deep Learning algorithm.
beer_sales_tbl %>%
    ggplot(aes(x = date, y = price)) +
    # Data.
    geom_point(size = 2, color = "grey", alpha = 0.5,
               shape = 21, fill = "black") +
    geom_line(color = "black", size = 0.5) +
    # Predictions.
    geom_point(aes(y = pred), size = 2,
               color = "gray", alpha = 1, shape = 21,
               fill = "purple", data = error_tbl) +
geom_line(aes(y = pred), color = "purple", size = 0.5, data = error_tbl) +
geom_vline(xintercept = as.numeric(as.Date("2016-12-01")), linetype=2) +
    # Aesthetics.
    labs(title = "Beer Sales Forecast: h2o + timetk",
    subtitle = "H2O without Deep Learning algorithm.",
    caption = c(paste("MAPE=",((mean(abs(error_tbl$error_pct))*100)))))
```

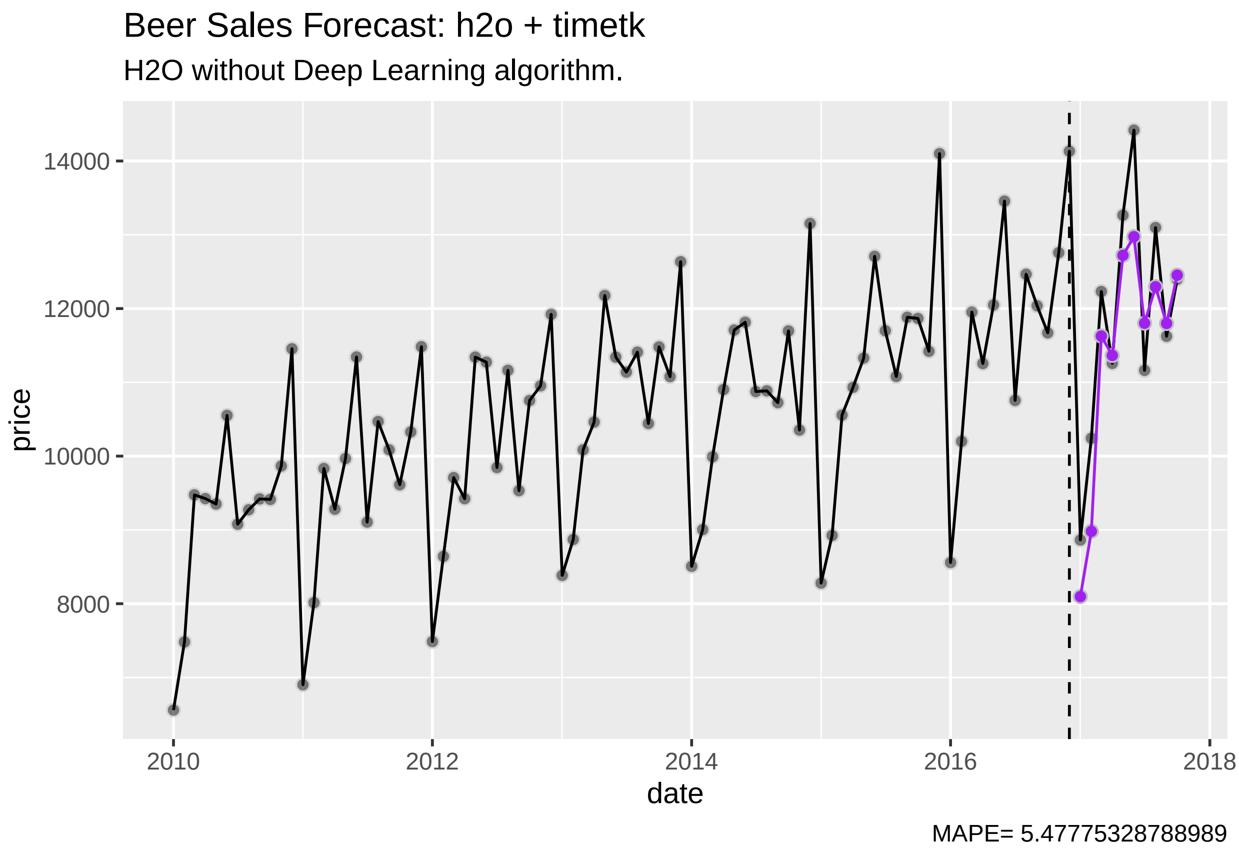

Figure 3.4.1: Forecast: H2O without Deep Learning algorithm.

This is an additional performance summary.

```
# Without Deep Learning.
h2o.performance(automl_leader, newdata = test_h2o)
```

```
## H2ORegressionMetrics: stackedensemble
##
## MSE:  602105.4
## RMSE:  775.9545
## MAE:  639.9103
## RMSLE:  0.06951576
## Mean Residual Deviance :  602105.4
```

Now, the case of only Deep Learning. The detailed forecast is in the following table.

```
# 10-period forecast error: h2o.automl only Deep Learning.
error_tbl_DL <- beer_sales_tbl %>%
    filter(lubridate::year(date) == 2017) %>%
    add_column(pred = pred_h2o_DL %>% as_tibble() %>% pull(predict)) %>%
    rename(actual = price) %>%
    mutate(error = actual - pred, error_pct = error / actual)
kable(error_tbl_DL,
caption = "Detailed performance: h2o.automl only Deep Learning algoritm.",
      digits = 3, row.names = TRUE) %>%
kable_styling(latex_options = "HOLD_position")
```

Table 3.4.2: Detailed performance: h2o.automl only Deep Learning algoritm.

	symbol	date	actual	pred	error	error_pct
1	S4248SM144NCEN	2017-01-01	8863	9644.515	-781.515	-0.088
2	S4248SM144NCEN	2017-02-01	10242	10413.011	-171.011	-0.017
3	S4248SM144NCEN	2017-03-01	12231	12314.663	-83.663	-0.007
4	S4248SM144NCEN	2017-04-01	11257	11097.040	159.960	0.014
5	S4248SM144NCEN	2017-05-01	13265	13230.929	34.071	0.003
6	S4248SM144NCEN	2017-06-01	14418	13275.766	1142.234	0.079
7	S4248SM144NCEN	2017-07-01	11162	11435.802	-273.802	-0.025
8	S4248SM144NCEN	2017-08-01	13098	12268.815	829.185	0.063
9	S4248SM144NCEN	2017-09-01	11624	11624.441	-0.441	0.000
10	S4248SM144NCEN	2017-10-01	12397	12284.146	112.854	0.009

The same information in a plot.

```r
# H2O including only Deep Learning algorithm.
beer_sales_tbl %>%
    ggplot(aes(x = date, y = price)) +
    # Data.
    geom_point(size = 2, color = "gray", alpha = 0.5,
               shape = 21, fill = "black") +
    geom_line(color = "black", size = 0.5) +
    # Predictions.
    geom_point(aes(y = pred), size = 2,
               color = "gray", alpha = 1, shape = 21,
               fill = "purple", data = error_tbl_DL) +
geom_line(aes(y = pred), color = "purple", size = 0.5,
          data = error_tbl_DL) +
    geom_vline(xintercept = as.numeric(as.Date("2016-12-01")), linetype=2) +
    # Aesthetics.
    labs(title = "Beer Sales Forecast: h2o + timetk",
    subtitle = "H2O including only Deep Learning algorithm.",
    caption = c(paste("MAPE=",((mean(abs(error_tbl_DL$error_pct))*100)))))
```

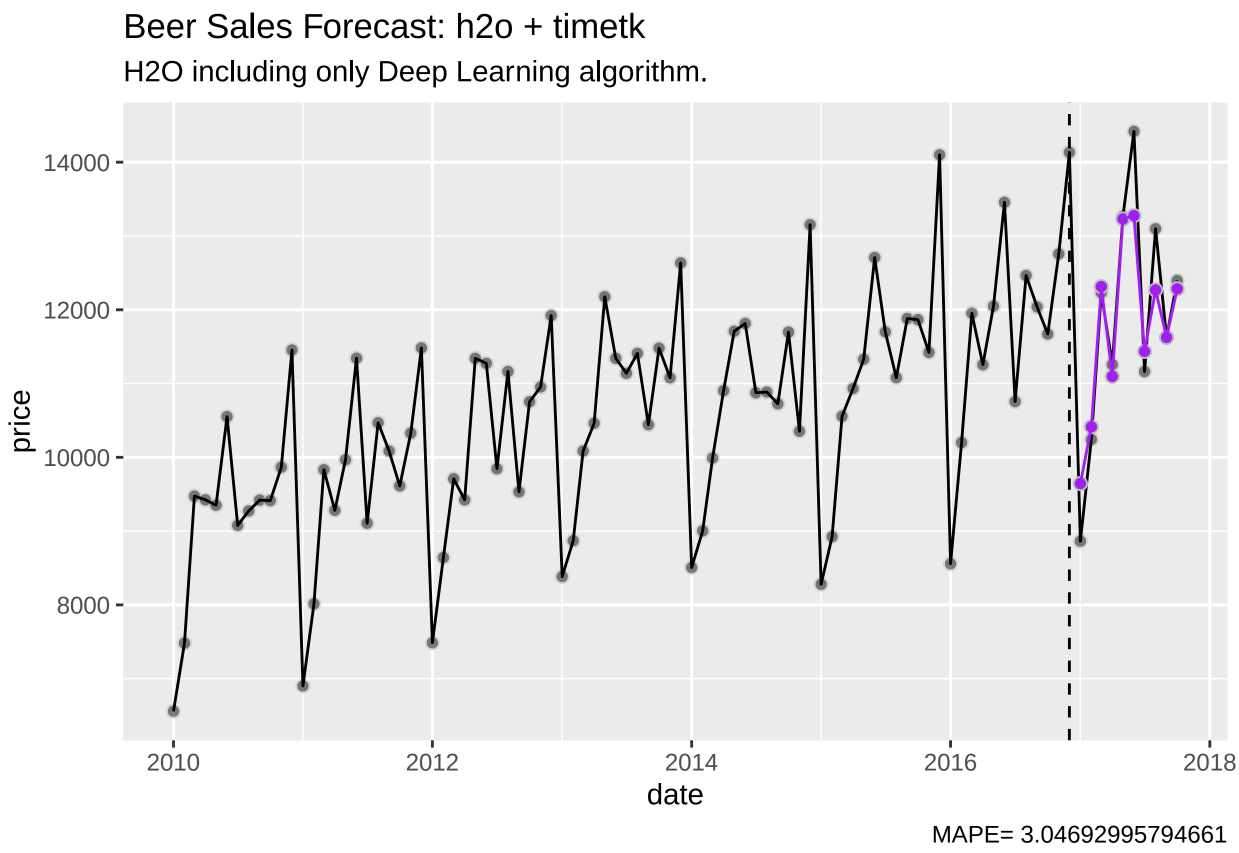

Figure 3.4.2: Forecast: H2O including only Deep Learning algorithm.

Additional performance indicators.

```
# Only Deep Learning.
h2o.performance(automl_DL, newdata = test_h2o)
```

```
## H2ORegressionMetrics: deeplearning
##
## MSE:  275370.8
## RMSE:  524.7579
## MAE:  358.8736
## RMSLE:  0.0440794
## Mean Residual Deviance :  275370.8
```

This is the H2O case with no restrictions, considering all available algorithms.

```r
# 10-period forecast error: h2o.automl all algorithms.
error_tbl_all <- beer_sales_tbl %>%
    filter(lubridate::year(date) == 2017) %>%
    add_column(pred = pred_h2o_all %>% as_tibble() %>% pull(predict)) %>%
    rename(actual = price) %>%
    mutate(error = actual - pred, error_pct = error / actual)
kable(error_tbl_all,
      caption = "Detailed performance: h2o.automl all algorithms.",
      digits = 3, row.names = TRUE) %>%
kable_styling(latex_options = "HOLD_position")
```

Table 3.4.3: Detailed performance: h2o.automl all algorithms.

	symbol	date	actual	pred	error	error_pct
1	S4248SM144NCEN	2017-01-01	8863	9367.894	-504.894	-0.057
2	S4248SM144NCEN	2017-02-01	10242	10153.247	88.753	0.009
3	S4248SM144NCEN	2017-03-01	12231	11855.850	375.150	0.031
4	S4248SM144NCEN	2017-04-01	11257	10566.913	690.087	0.061
5	S4248SM144NCEN	2017-05-01	13265	13053.242	211.758	0.016
6	S4248SM144NCEN	2017-06-01	14418	12826.635	1591.365	0.110
7	S4248SM144NCEN	2017-07-01	11162	11196.999	-34.999	-0.003
8	S4248SM144NCEN	2017-08-01	13098	12774.825	323.175	0.025
9	S4248SM144NCEN	2017-09-01	11624	11812.626	-188.626	-0.016
10	S4248SM144NCEN	2017-10-01	12397	13008.194	-611.194	-0.049

The visual representation.

```r
# H2O all available algorithms.
beer_sales_tbl %>%
    ggplot(aes(x = date, y = price)) +
    # Data.
    geom_point(size = 2, color = "grey", alpha = 0.5,
               shape = 21, fill = "black") +
    geom_line(color = "black", size = 0.5) +
    # Predictions.
    geom_point(aes(y = pred), size = 2,
               color = "gray", alpha = 1, shape = 21,
               fill = "purple", data = error_tbl_all) +
geom_line(aes(y = pred), color = "purple", size = 0.5,
          data = error_tbl_all) +
geom_vline(xintercept = as.numeric(as.Date("2016-12-01")), linetype=2) +
    # Aesthetics.
    labs(title = "Beer Sales Forecast: h2o + timetk",
      subtitle = "H2O all available algorithms.",
    caption = c(paste("MAPE=",((mean(abs(error_tbl_all$error_pct))*100)))))
```

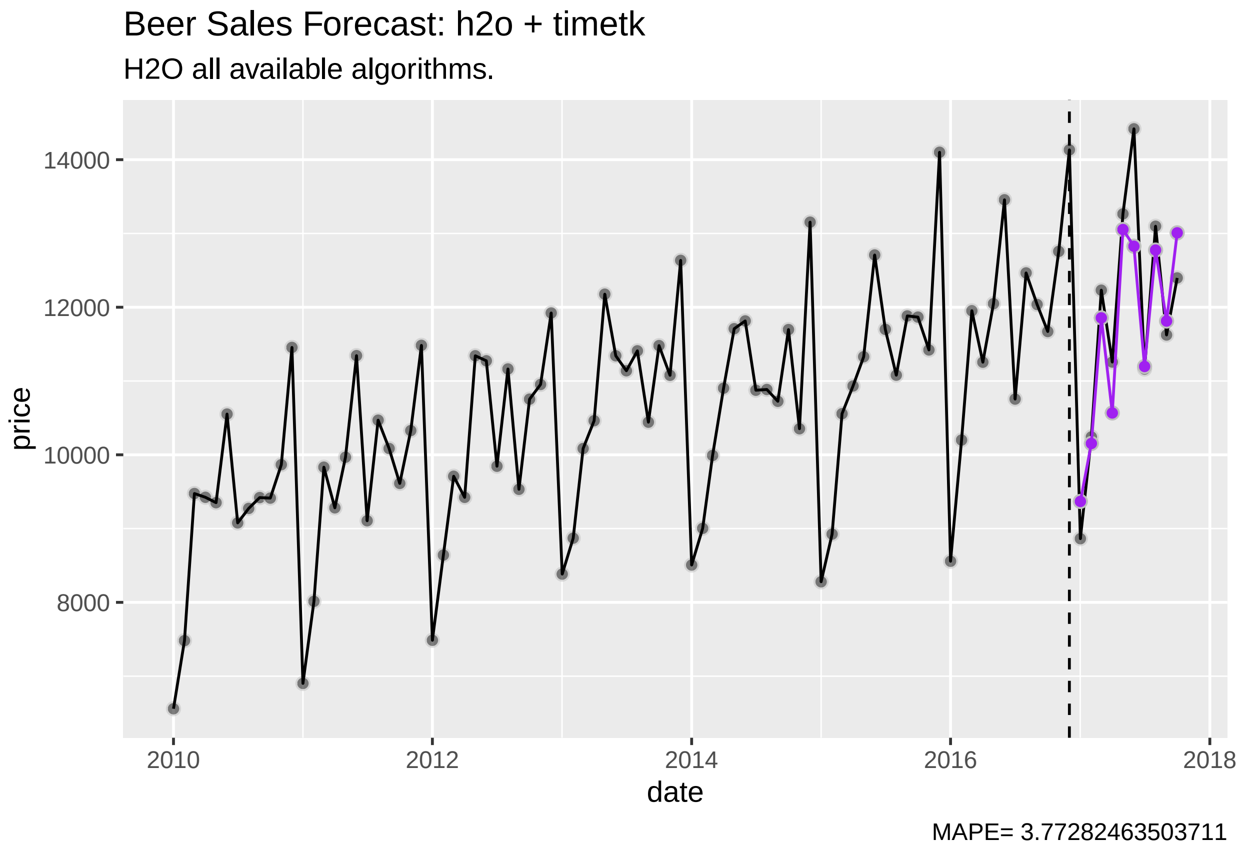

Figure 3.4.3: Forecast: H2O including all available algorithms.

Additional performance metrics.

```
h2o.performance(automl_leader_all, newdata = test_h2o)
```

```
## H2ORegressionMetrics: deeplearning
```

```
##
## MSE:  397184.1
## RMSE:  630.2255
## MAE:  462.0001
## RMSLE:  0.05025921
## Mean Residual Deviance :  397184.1
```

These plots show the power of modern forecasting techniques. In finance we care about the future and these techniques can be used as a tool to reduce the uncertainty about the future. Obviously, we cannot predict without errors, but the objective is to achieve the lowest forecasting errors possible.

3.5 Summary performance.

It is useful to see the performance results for the three different H2O runs above. First, the performance for the overall 10-period forecast.

```r
# There might be a more compact way to create this table.
error_tbl_summ <- error_tbl %>%
    summarise(model = automl_leader@algorithm,
      me = mean(error), rmse = mean(error^2)^0.5,
              mae  = mean(abs(error)), mape = 100 * mean(abs(error_pct)),
              mpe  = 100 * mean(error_pct))
error_tbl_DL_summ <- error_tbl_DL %>%
    summarise(model = automl_DL@algorithm,
      me = mean(error), rmse = mean(error^2)^0.5,
              mae = mean(abs(error)), mape = 100 * mean(abs(error_pct)),
              mpe = 100 * mean(error_pct))
error_tbl_all_summ <- error_tbl_all %>%
    summarise(model = automl_leader_all@algorithm,
      me = mean(error), rmse = mean(error^2)^0.5,
              mae  = mean(abs(error)), mape = 100 * mean(abs(error_pct)),
              mpe  = 100 * mean(error_pct))
error_automl_summ <- rbind(error_tbl_summ, error_tbl_DL_summ,
                                error_tbl_all_summ) %>%
  as.data.frame()
row.names(error_automl_summ) <- c("Without Deep Learning",
                                  "Only Deep Learning", "All algorithms")
```

```
kable(error_automl_summ,
caption = "Top ranked models: h2o.automl summary forecasting errors.",
digits = 2) %>%
kable_styling(latex_options = "HOLD_position")
```

Table 3.5.1: Top ranked models: h2o.automl summary forecasting errors.

	model	me	rmse	mae	mape	mpe
Without Deep Learning	stackedensemble	444.34	775.95	639.91	5.48	3.75
Only Deep Learning	deeplearning	96.79	524.76	358.87	3.05	0.32
All algorithms	deeplearning	194.06	630.23	462.00	3.77	1.26

As you can see, there are several ways in which we can measure the forecast errors. We can specify which one is the evaluation criterion to rank the models. And we can also determine which error measure: me (mean error), rmse (root mean squared error), mae (mean absolute error), mape (mean absolute percentage error), or mpe (mean percentage error) will be the one to choose between these three alternatives. In my experience, the rmse and the mape are the most popular ones, but the others might be useful in specific circumstances.

We can also show the best point forecast for the three `h2o.automl` runs.

```
point_forecast_1 <- data.frame(
  model = automl_leader@algorithm,
  error_tbl[which.min(abs(error_tbl$error_pct)), 2],
  error = error_tbl[which.min(abs(error_tbl$error_pct)), 6])
point_forecast_2 <- data.frame(
  model = automl_DL@algorithm,
  error_tbl_DL[which.min(abs(error_tbl_DL$error_pct)), 2],
  error = error_tbl_DL[which.min(abs(error_tbl_DL$error_pct)), 6])
point_forecast_3 <- data.frame(
  model = automl_leader_all@algorithm,
  error_tbl_all[which.min(abs(error_tbl_all$error_pct)), 2],
  error = error_tbl_all[which.min(abs(error_tbl_all$error_pct)), 6])
point_forecast <- rbind.data.frame(point_forecast_1, point_forecast_2,
                                   point_forecast_3)
row.names(point_forecast) <- c("Without Deep Learning",
                                    "Only Deep Learning", "All algorithms")
kable(point_forecast,
caption = "Top ranked models: Lowest point forecast percentage errors.",
digits = 6) %>%
kable_styling(latex_options = "HOLD_position")
```

Table 3.5.2: Top ranked models: Lowest point forecast percentage errors.

	model	date	error_pct
Without Deep Learning	stackedensemble	2017-10-01	-0.004494
Only Deep Learning	deeplearning	2017-09-01	-0.000038
All algorithms	deeplearning	2017-07-01	-0.003136

We normally do not choose a model according to one specific point forecast. However, it is interesting to see which alternative and which specific date has been forecasted with the highest accuracy.

4 Linear regression.

Let's implement a simple but powerful approach using the `lm` function.

4.1 Implement the model.

This is the simplest choice, and still has a very high R^2. The independent variables are all `beer_sales_tbl_aug` variables except for `date`, `diff`, and `symbol`.

```
# linear regression model used, but can use any model
fit_lm <- lm(price ~ ., data =
                select(beer_sales_tbl_aug, -c(date, diff, symbol)))
summary(fit_lm)
```

```
##
## Call:
## lm(formula = price ~ ., data = select(beer_sales_tbl_aug, -c(date,
##     diff, symbol)))
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -518.46 -165.14  -15.02  163.61  685.06
##
## Coefficients: (16 not defined because of singularities)
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.021e+08  1.127e+08   2.680 0.009262 **
## index.num     4.872e-03  1.813e-03   2.687 0.009092 **
## year         -1.571e+05  5.835e+04  -2.693 0.008940 **
## year.iso      3.778e+03  5.845e+03   0.646 0.520236
## half         -2.464e+03  6.317e+02  -3.900 0.000225 ***
## quarter      -2.138e+04  2.374e+04  -0.900 0.371194
## month        -2.595e+03  7.943e+03  -0.327 0.744851
## month.xts            NA         NA      NA       NA
## month.lbl.L          NA         NA      NA       NA
## month.lbl.Q  -1.774e+03  2.218e+02  -7.999 2.41e-11 ***
## month.lbl.C   6.588e+02  5.448e+02   1.209 0.230842
## month.lbl^4   7.313e+02  1.436e+02   5.093 3.07e-06 ***
## month.lbl^5   8.756e+02  4.434e+02   1.975 0.052413 .
## month.lbl^6   3.174e+02  1.719e+02   1.847 0.069190 .
## month.lbl^7  -4.171e+02  2.017e+02  -2.068 0.042546 *
## month.lbl^8   3.491e+02  3.447e+02   1.013 0.314917
## month.lbl^9          NA         NA      NA       NA
## month.lbl^10  6.014e+02  2.381e+02   2.526 0.013900 *
## month.lbl^11         NA         NA      NA       NA
## day                  NA         NA      NA       NA
## hour                 NA         NA      NA       NA
## minute               NA         NA      NA       NA
## second               NA         NA      NA       NA
## hour12               NA         NA      NA       NA
## am.pm                NA         NA      NA       NA
## wday         -5.959e+01  2.185e+01  -2.727 0.008148 **
## wday.xts             NA         NA      NA       NA
## wday.lbl.L           NA         NA      NA       NA
## wday.lbl.Q   -8.371e+02  1.089e+02  -7.688 8.79e-11 ***
## wday.lbl.C    1.516e+02  9.557e+01   1.587 0.117333
## wday.lbl^4    1.160e+02  1.135e+02   1.023 0.310200
## wday.lbl^5    6.354e+01  9.732e+01   0.653 0.516057
## wday.lbl^6    1.077e+02  8.671e+01   1.242 0.218576
## mday                 NA         NA      NA       NA
## qday         -2.354e+02  2.621e+02  -0.898 0.372251
## yday         -7.483e+01  1.187e+02  -0.630 0.530669
## mweek        -1.046e+02  1.535e+02  -0.681 0.497995
## week         -1.300e+02  1.994e+02  -0.652 0.516835
## week.iso      8.076e+01  1.129e+02   0.715 0.476845
## week2        -1.122e+02  1.675e+02  -0.670 0.505081
## week3                NA         NA      NA       NA
## week4                NA         NA      NA       NA
## mday7                NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 282.4 on 67 degrees of freedom
## Multiple R-squared:  0.977,  Adjusted R-squared:  0.9681
## F-statistic: 109.5 on 26 and 67 DF,  p-value: < 2.2e-16
```

At first sight, the model looks promising.

4.2 Predict.

Prediction is easy in R.

```
# Make predictions
pred <- predict(fit_lm, newdata = test_tbl)
future_idx <- tail(beer_sales_tbl$date, 10) # The 10-months forecast period.
predictions_tbl <- tibble(date  = future_idx, value = pred)
predictions_tbl
```

```
## # A tibble: 10 x 2
##    date        value
##    <date>      <dbl>
##  1 2017-01-01  9349.
##  2 2017-02-01 10742.
##  3 2017-03-01 12220.
##  4 2017-04-01 11267.
##  5 2017-05-01 13092.
##  6 2017-06-01 13733.
##  7 2017-07-01 11400.
##  8 2017-08-01 13059.
##  9 2017-09-01 11795.
## 10 2017-10-01 12338.
```

We can investigate the error on our test set (actuals vs predictions).

```
# Investigate test error
actuals_tbl <- tail(beer_sales_tbl[-1], 10)
error_tbl_lm <- left_join(actuals_tbl, predictions_tbl) %>%
    rename(actual = price, pred = value) %>%
    mutate(error = actual - pred, error_pct = error / actual)
error_tbl_lm
```

```
## # A tibble: 10 x 5
##    date       actual   pred  error error_pct
##    <date>      <int>  <dbl>  <dbl>     <dbl>
##  1 2017-01-01   8863  9349. -486.  -0.0548
##  2 2017-02-01  10242 10742. -500.  -0.0488
##  3 2017-03-01  12231 12220.   11.0  0.000901
##  4 2017-04-01  11257 11267.  -10.0 -0.000892
##  5 2017-05-01  13265 13092.  173.   0.0130
##  6 2017-06-01  14418 13733.  685.   0.0475
##  7 2017-07-01  11162 11400. -238.  -0.0213
##  8 2017-08-01  13098 13059.   39.1  0.00299
##  9 2017-09-01  11624 11795. -171.  -0.0147
## 10 2017-10-01  12397 12338.   59.2  0.00477
```

And we can calculate a few residuals metrics. A more complex algorithm could produce more accurate results.

```
# Calculating test error metrics
test_residuals_lm <- error_tbl_lm$error
test_error_pct_lm <- error_tbl_lm$error_pct * 100 # Percentage error.
me   <- mean(test_residuals_lm, na.rm = TRUE)
rmse <- mean(test_residuals_lm^2, na.rm = TRUE)^0.5
mae  <- mean(abs(test_residuals_lm), na.rm = TRUE)
```

```
mape <- mean(abs(test_error_pct_lm), na.rm = TRUE)
mpe  <- mean(test_error_pct_lm, na.rm = TRUE)
tibble(me, rmse, mae, mape, mpe) %>%
  glimpse()
```

```
## Rows: 1
## Columns: 5
## $ me   <dbl> -43.80576
## $ rmse <dbl> 328.0476
## $ mae  <dbl> 237.1904
## $ mape <dbl> 2.09748
## $ mpe  <dbl> -0.713821
```

Visualize our forecast.

```
# Plot Beer Sales Forecast
beer_sales_tbl %>%
    ggplot(aes(x = date, y = price)) +
    # Training data.
    geom_line(color = palette_light()[[1]]) +
    geom_point(color = palette_light()[[1]]) +
    # Predictions.
    geom_line(aes(y = value),
              color = palette_light()[[2]], data = predictions_tbl) +
    geom_point(aes(y = value),
               color = palette_light()[[2]], data = predictions_tbl) +
    # Actuals
    geom_line(color = palette_light()[[1]], data = actuals_tbl) +
    geom_point(color = palette_light()[[1]], data = actuals_tbl) +
geom_vline(xintercept = as.numeric(as.Date("2016-12-01")), linetype = 2) +
    # Aesthetics
    theme_tq() +
    labs(title = "Beer Sales Forecast.",
subtitle = "Multivariate linear regression can yield accurate results.",
        caption = c(paste("MAPE=",((mean(abs(test_error_pct_lm)))))))
```

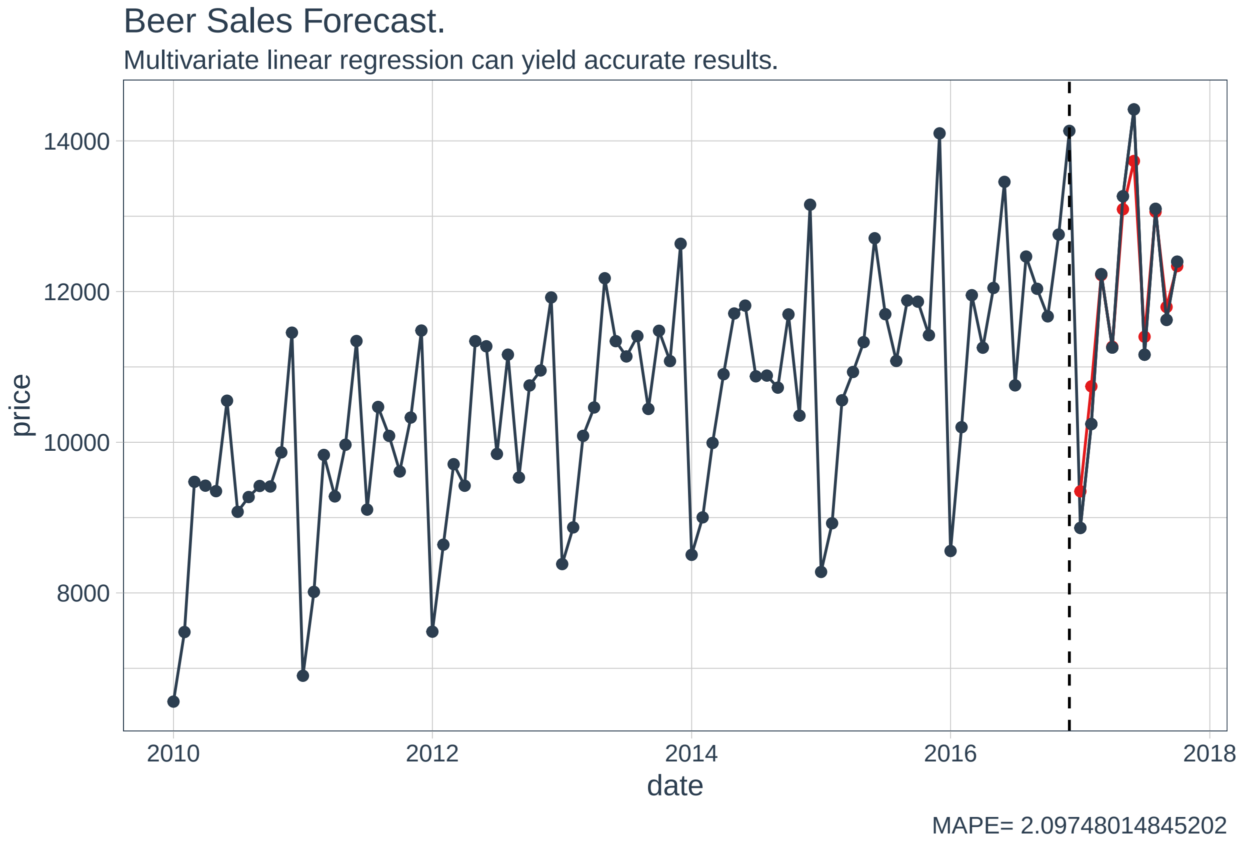

Figure 4.2.1: Forecast: multivariate linear regression.

This is clearly a good alternative. The H2O machine learning could lead to better results if we consider a longer time-series because in that way the possibilities to learn increases.

5 ARIMA.

Here, `sweep` is used for tidying the `forecast` package workflow. We'll work through an ARIMA analysis to forecast the next 10 months of time series data. In this way we can compare our previous results.

5.1 Prepare the data.

The `tk_ts` coerce time series objects and tibbles with date/date-time columns to ts (time-series).

```r
# Convert from tbl to ts.
beer_sales_ts <- tk_ts(beer_sales_tbl[1:84,], start = 2010, freq = 12)
beer_sales_ts
```

```
##        Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
## 2010  6558  7481  9475  9424  9351 10552  9077  9273  9420  9413  9866 11455
## 2011  6901  8014  9832  9281  9967 11344  9106 10469 10085  9612 10328 11483
## 2012  7486  8641  9709  9423 11342 11274  9845 11163  9532 10754 10953 11922
## 2013  8383  8870 10085 10462 12177 11342 11139 11409 10442 11480 11077 12635
## 2014  8506  9003  9991 10903 11709 11814 10875 10885 10725 11697 10353 13153
## 2015  8279  8925 10557 10933 11329 12708 11700 11079 11882 11866 11421 14100
## 2016  8557 10200 11952 11255 12048 13456 10755 12465 12037 11671 12757 14133
```

Just verify `tk_ts` worked.

```r
# Check that ts-object has a timetk index.
has_timetk_idx(beer_sales_ts)
```

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

Great. This will be important when we use `sw_sweep` later. Next, we'll model using ARIMA.

5.2 Implement the model.

We can use the `auto.arima` function from the forecast package to model the time series. By doing that, we do not have to impose a specific ARIMA model, the function can test the best specification for us.

```r
# Model using auto.arima.
set.seed(13)
fit_arima <- auto.arima(beer_sales_ts)
fit_arima
```

```
## Series: beer_sales_ts
## ARIMA(3,0,0)(0,1,1)[12] with drift
##
## Coefficients:
##           ar1     ar2     ar3     sma1    drift
##       -0.2678  0.0954  0.6022  -0.6769  33.5623
## s.e.   0.0956  0.1015  0.1038   0.5679   3.1678
```

```
##
## sigma^2 estimated as 153144:  log likelihood=-533.75
## AIC=1079.49   AICc=1080.79   BIC=1093.15
```

The `sw_tidy` function returns the model coefficients in a tibble (tidy data frame). This might be useful in some circumstances.

```
# sw_tidy - Get model coefficients.
sw_tidy(fit_arima)
```

```
## # A tibble: 5 x 2
##   term  estimate
##   <chr>    <dbl>
## 1 ar1    -0.268
## 2 ar2     0.0954
## 3 ar3     0.602
## 4 sma1   -0.677
## 5 drift  33.6
```

The `sw_glance` function returns the training set accuracy measures in a tibble (tidy data frame). We use glimpse to aid in quickly reviewing the model metrics.

```
# sw_glance - Get model description and training set accuracy measures.
sw_glance(fit_arima) %>%
    glimpse()
```

```
## Rows: 1
## Columns: 12
## $ model.desc <chr> "ARIMA(3,0,0)(0,1,1)[12] with drift"
## $ sigma      <dbl> 391.3357
## $ logLik     <dbl> -533.747
## $ AIC        <dbl> 1079.494
## $ BIC        <dbl> 1093.154
## $ ME         <dbl> 9.076361
## $ RMSE       <dbl> 349.5
## $ MAE        <dbl> 248.4856
## $ MPE        <dbl> -0.04041324
## $ MAPE       <dbl> 2.327402
## $ MASE       <dbl> 0.4527409
## $ ACF1        <dbl> -0.003806875
```

This looks good.

5.3 Predict.

The `sw_augument` function helps with model evaluation. We get the ".actual", ".fitted" and ".resid" columns, which are useful in evaluating the model against the training data. Note that we can pass `timetk_idx = TRUE` to return the original date index.

```
# sw_augment - get model residuals
sw_augment(fit_arima, timetk_idx = TRUE)
```

```
## # A tibble: 84 x 4
##    index      .actual .fitted .resid
##    <date>       <dbl>   <dbl>  <dbl>
##  1 2010-01-01    6558   6551.   6.52
##  2 2010-02-01    7481   7474.   7.41
##  3 2010-03-01    9475   9466.   9.37
##  4 2010-04-01    9424   9415.   9.29
##  5 2010-05-01    9351   9342.   9.18
##  6 2010-06-01   10552  10542.  10.4
##  7 2010-07-01    9077   9068.   8.84
##  8 2010-08-01    9273   9264.   9.00
##  9 2010-09-01    9420   9411.   9.12
## 10 2010-10-01    9413   9404.   9.08
## # ... with 74 more rows
```

We can visualize the residual diagnostics for the training data to make sure there is no pattern leftover. This looks homoscedastic.

```
# Plotting residuals
sw_augment(fit_arima, timetk_idx = TRUE) %>%
    ggplot(aes(x = index, y = .resid)) +
    geom_point() +
    geom_hline(yintercept = 0, color = "red") +
    labs(title = "Residual diagnostic") +
    scale_x_date(date_breaks = "1 year", date_labels = "%Y") +
    theme_tq()
```

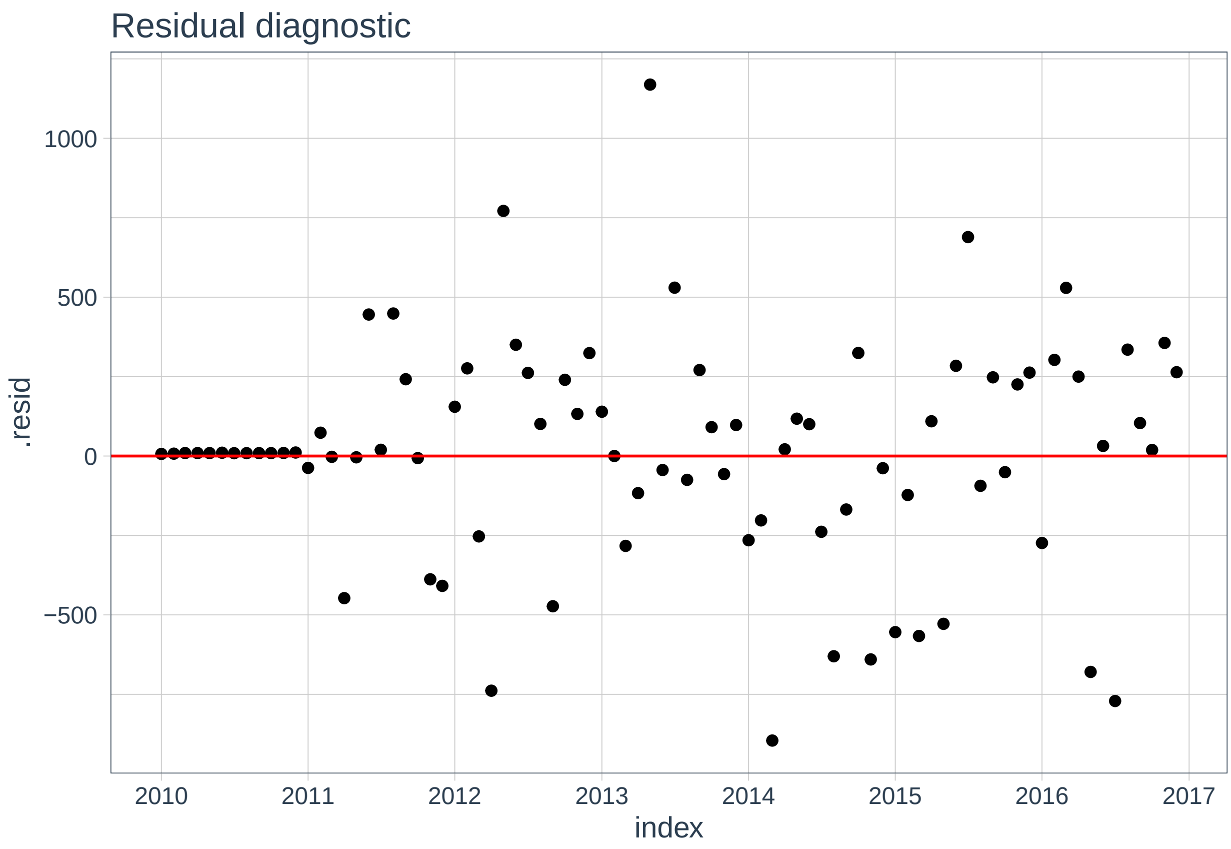

Figure 5.3.1: Forecast: ARIMA residual diagnosis.

Make a forecast using the `forecast` function. This function also delivers some convenient error bounds.

```
# Forecast next 10 months
fcast_arima <- forecast(fit_arima, h = 10)
fcast_arima
```

```
##          Point Forecast     Lo 80     Hi 80     Lo 95     Hi 95
## Jan 2017       8972.436  8470.056  9474.815  8204.113  9740.759
## Feb 2017      10921.213 10401.156 11441.269 10125.854 11716.571
## Mar 2017      11690.876 11164.106 12217.646 10885.251 12496.501
## Apr 2017      11704.825 11114.343 12295.307 10801.760 12607.889
## May 2017      13025.344 12417.478 13633.209 12095.693 13954.994
## Jun 2017      13288.042 12669.454 13906.629 12341.993 14234.090
## Jul 2017      11914.454 11285.058 12543.850 10951.875 12877.032
## Aug 2017      12730.505 12092.092 13368.918 11754.137 13706.873
## Sep 2017      12134.896 11487.285 12782.508 11144.460 13125.333
## Oct 2017      12585.176 11936.883 13233.468 11593.697 13576.654
```

One problem is the forecast output is not "tidy". We need it in a data frame if we want to work with it using the tidyverse functionality. The class is "forecast", which is a ts-based-object (its contents are ts-objects).

```
class(fcast_arima)
```

```
## [1] "forecast"
```

We can use `sw_sweep` to tidy the forecast output. As an added benefit, if the forecast-object has a `timetk` index, we can use it to return a date/datetime index as opposed to regular index from the ts-based-object.

First, let's check if the forecast-object has a timetk index.

```
# Check if object has timetk index
has_timetk_idx(fcast_arima)
```

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

Great. Now, use `sw_sweep` to tidy the forecast output.

```
# sw_sweep - tidies forecast output
fcast_tbl <- sw_sweep(fcast_arima, timetk_idx = TRUE)
fcast_tbl
```

```
## # A tibble: 94 x 7
##    index      key    price lo.80 lo.95 hi.80 hi.95
##    <date>     <chr>  <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 2010-01-01 actual  6558    NA    NA    NA    NA
##  2 2010-02-01 actual  7481    NA    NA    NA    NA
##  3 2010-03-01 actual  9475    NA    NA    NA    NA
##  4 2010-04-01 actual  9424    NA    NA    NA    NA
##  5 2010-05-01 actual  9351    NA    NA    NA    NA
##  6 2010-06-01 actual 10552    NA    NA    NA    NA
##  7 2010-07-01 actual  9077    NA    NA    NA    NA
##  8 2010-08-01 actual  9273    NA    NA    NA    NA
##  9 2010-09-01 actual  9420    NA    NA    NA    NA
## 10 2010-10-01 actual  9413    NA    NA    NA    NA
## # ... with 84 more rows
```

We can investigate the error on our test set (actuals vs predictions).

```
# Investigate test error
error_tbl_arima <- left_join(actuals_tbl, fcast_tbl,
                             by = c("date" = "index")) %>%
    rename(actual = price.x, pred = price.y) %>%
    select(date, actual, pred) %>%
    mutate(error = actual - pred, error_pct = error / actual)
error_tbl_arima
```

```
## # A tibble: 10 x 5
##    date       actual   pred error error_pct
##    <date>      <int>  <dbl> <dbl>     <dbl>
##  1 2017-01-01   8863  8972. -109.   -0.0123
##  2 2017-02-01  10242 10921. -679.   -0.0663
##  3 2017-03-01  12231 11691.  540.    0.0442
##  4 2017-04-01  11257 11705. -448.   -0.0398
##  5 2017-05-01  13265 13025.  240.    0.0181
##  6 2017-06-01  14418 13288. 1130.    0.0784
##  7 2017-07-01  11162 11914. -752.   -0.0674
##  8 2017-08-01  13098 12731.  367.    0.0281
##  9 2017-09-01  11624 12135. -511.   -0.0440
## 10 2017-10-01  12397 12585. -188.   -0.0152
```

And we can calculate a few residuals metrics.

```
# Calculate test error metrics
test_residuals_arima <- error_tbl_arima$error
test_error_pct_arima <- error_tbl_arima$error_pct * 100 # Percentage error
me   <- mean(test_residuals_arima, na.rm=TRUE)
rmse <- mean(test_residuals_arima^2, na.rm=TRUE)^0.5
mae  <- mean(abs(test_residuals_arima), na.rm=TRUE)
mape <- mean(abs(test_error_pct_arima), na.rm=TRUE)
mpe  <- mean(test_error_pct_arima, na.rm=TRUE)
tibble(me, rmse, mae, mape, mpe) %>%
  glimpse()
```

```
## Rows: 1
## Columns: 5
## $ me   <dbl> -41.07646
## $ rmse <dbl> 574.238
## $ mae  <dbl> 496.5233
## $ mape <dbl> 4.136446
## $ mpe  <dbl> -0.7633304
```

Notice that we have the entire forecast in a tibble. We can now more easily visualize the forecast.

```
# Visualize the forecast with ggplot
fcast_tbl %>%
    ggplot(aes(x = index, y = price, color = key)) +
    # 95% CI
    geom_ribbon(aes(ymin = lo.95, ymax = hi.95),
                fill = "#D5DBFF", color = NA, size = 0) +
    # 80% CI
    geom_ribbon(aes(ymin = lo.80, ymax = hi.80, fill = key),
                fill = "#596DD5", color = NA, size = 0, alpha = 0.8) +
    # Prediction
    geom_line() +
    geom_point() +
    # Actuals
    geom_line(aes(x = date, y = price), color = palette_light()[[1]],
              data = actuals_tbl) +
    geom_point(aes(x = date, y = price), color = palette_light()[[1]],
               data = actuals_tbl) +
        geom_vline(xintercept = as.numeric(as.Date("2016-12-01")),
                   linetype=2) +
    # Aesthetics
labs(title = "Beer Sales Forecast: ARIMA", x = "", y = "Thousands of Tons",
         subtitle = "sw_sweep tidies the auto.arima() forecast output",
         caption = c(paste("MAPE=",((mean(abs(test_error_pct_arima))))))) +
    scale_x_date(date_breaks = "1 year", date_labels = "%Y") +
    scale_color_tq() +
    scale_fill_tq() +
    theme_tq()
```

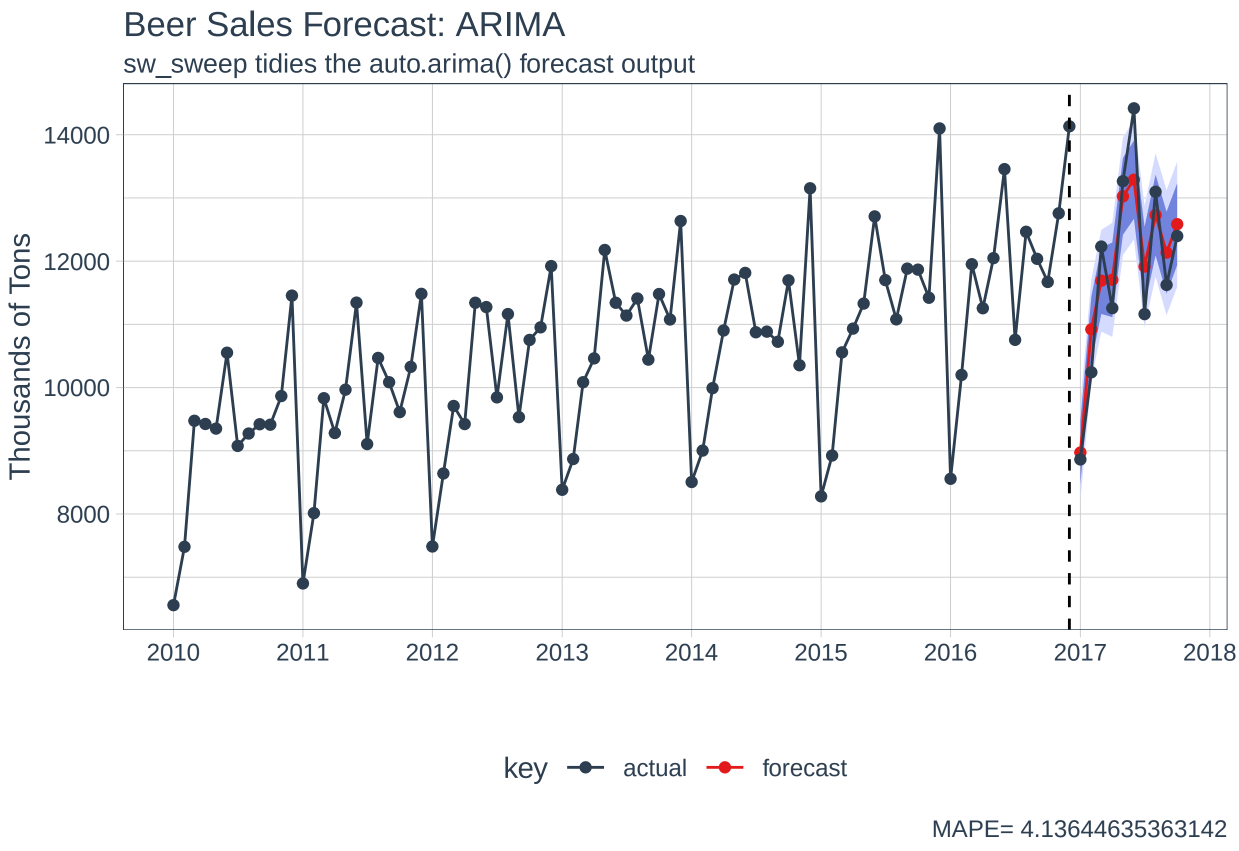

Figure 5.3.2: Forecast: ARIMA.

This is a decent forecast.

6 The average forecast.

An interesting question is: *What happens to the accuracy when you average the predictions of all different methods?* This question makes sense because the decision of using one technique or another is not trivial. Taking the average could be useful to avoid extreme results but at the same time it could be hard to interpret as the forecast comes from different techniques. In any case, it is interesting to see how it works.

The forecast mean is calculated as:

```
m <- data.frame(as.data.frame(pred_h2o),as.data.frame(pred_h2o_DL),
          as.data.frame(pred_h2o_all),as.data.frame(predictions_tbl$value),
        as.data.frame(fcast_tbl$price[(nrow(fcast_tbl)-9):nrow(fcast_tbl)]))
pred_mean <- rowMeans(m)
pred_mean <- as.tibble(pred_mean)
```

Now let's see actual versus predicted.

```
error_tbl_mean <-as_tibble(c(as.data.frame(actuals_tbl),
                        as.data.frame(pred_mean$value))) %>%
    rename(actual = price, pred = `pred_mean$value`) %>%
    select(date,actual, pred) %>%
    mutate(error = actual - pred, error_pct = error / actual)
error_tbl_mean
```

```
## # A tibble: 10 x 5
##    date       actual   pred     error   error_pct
##    <date>      <int>  <dbl>     <dbl>       <dbl>
##  1 2017-01-01   8863  9086. -223.     -0.0252
##  2 2017-02-01  10242 10242.   -0.0530 -0.00000518
##  3 2017-03-01  12231 11942.  289.      0.0236
##  4 2017-04-01  11257 11200.   56.9     0.00505
##  5 2017-05-01  13265 13025.  240.      0.0181
##  6 2017-06-01  14418 13220. 1198.      0.0831
##  7 2017-07-01  11162 11550. -388.     -0.0347
##  8 2017-08-01  13098 12626.  472.      0.0361
##  9 2017-09-01  11624 11833. -209.     -0.0180
## 10 2017-10-01  12397 12534. -137.     -0.0110
```

Summarize the individual point forecast errors.

```
error_tbl_mean %>%
    summarise(me = mean(error), rmse = mean(error^2)^0.5,
        mae  = mean(abs(error)), mape = mean(abs(error_pct)),
        mpe  = mean(error_pct)) %>%
glimpse()
```

```
## Rows: 1
## Columns: 5
## $ me   <dbl> 130.0605
## $ rmse <dbl> 454.6017
## $ mae  <dbl> 321.3871
## $ mape <dbl> 0.02549247
## $ mpe  <dbl> 0.00770797
```

Visualize the average forecast.

```r
# Plot Beer Sales Forecast
beer_sales_tbl %>%
    ggplot(aes(x = date, y = price)) +
    # Training data
    geom_line(color = palette_light()[[1]]) +
    geom_point(color = palette_light()[[1]]) +
    # Predictions
    geom_point(aes(y = pred), size = 2,
               color = "gray", alpha = 1, shape = 21,
               fill = "red", data = error_tbl) +
geom_line(aes(y = pred), color = "purple", size = 0.5, data = error_tbl) +
geom_vline(xintercept = as.numeric(as.Date("2016-12-01")), linetype = 2) +
      # Actuals
    geom_line(color = palette_light()[[1]], data = actuals_tbl) +
    geom_point(color = palette_light()[[1]], data = actuals_tbl) +
geom_vline(xintercept = as.numeric(as.Date("2016-12-01")), linetype = 2) +
    # Aesthetics
    theme_tq() +
    labs(title = "Beer Sales Forecast: Mean of all previous forecast.",
subtitle = "The average method.",
caption = c(paste("MAPE=",((100*mean(abs(error_tbl_mean$error_pct)))))))
```

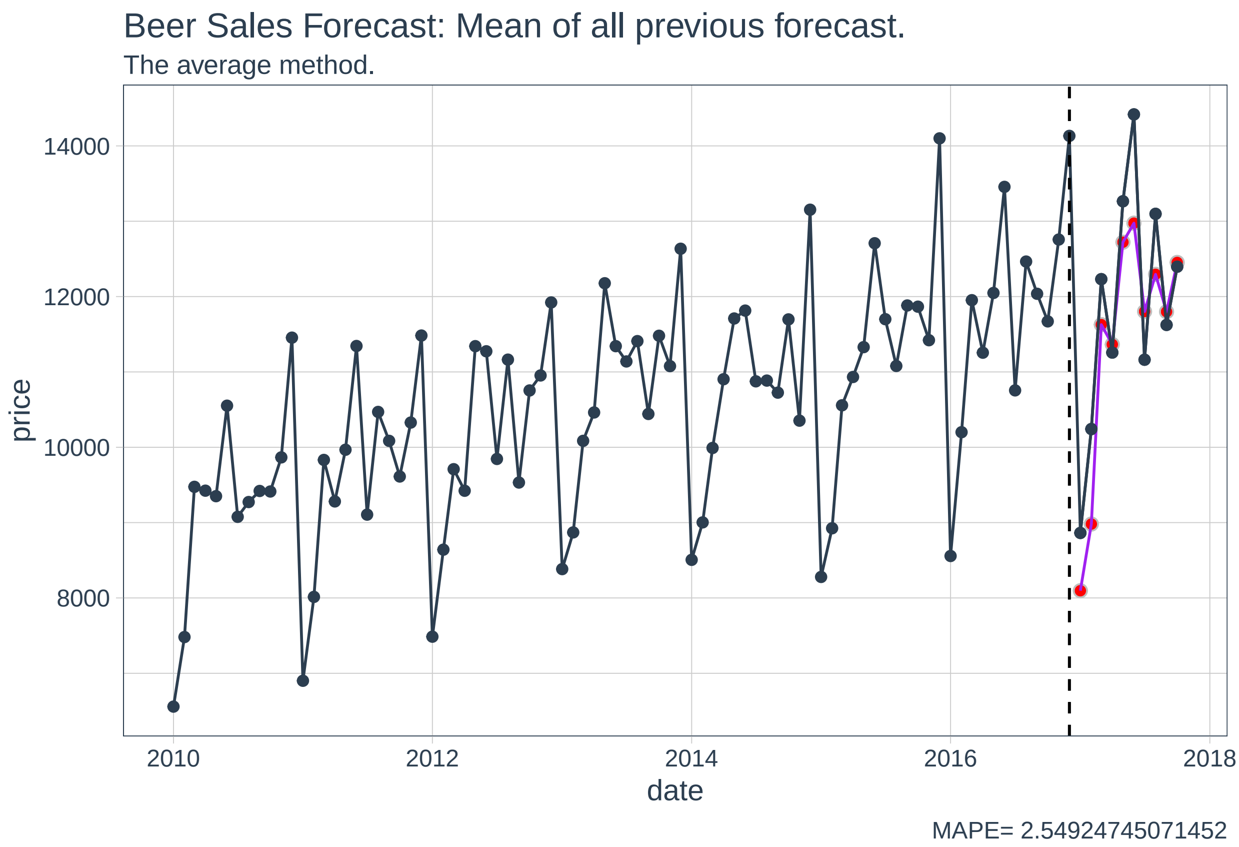

Figure 6.0.1: Forecast: average forecast.

Not bad, as expected.

7 Summary of all results.

Let's see all results at once: H2O, linear regression, ARIMA and the average forecast.

```
summary_techniques <- c("H2O without Deep Learning algorithm",
             "H2O including only Deep Learning algorithm",
             "H2O all available algorithms",
             "Multivariate linear regression",
             "ARIMA",
             "Average")
summary_mape <- c(mean(abs(error_tbl$error_pct))*100,
             mean(abs(error_tbl_DL$error_pct))*100,
             mean(abs(error_tbl_all$error_pct))*100,
             mean(abs(test_error_pct_lm)),
             mean(abs(test_error_pct_arima)),
             100*mean(abs(error_tbl_mean$error_pct)))
sum <- data.frame(summary_techniques, summary_mape)
kable(sum, caption = "Summary of results.") %>%
kable_styling(latex_options = "HOLD_position")
```

Table 7.0.1: Summary of results.

summary_techniques	summary_mape
H2O without Deep Learning algorithm	5.477753
H2O including only Deep Learning algorithm	3.046930
H2O all available algorithms	3.772825
Multivariate linear regression	2.097480
ARIMA	4.136446
Average	2.549247

Nice.

```
h2o.shutdown(prompt = TRUE) # yes (Y) instead of TRUE?
```

```
## Are you sure you want to shutdown the H2O instance running at http://localhost:54321/
```

```
a <- toc()
```

```
## 245.06 sec elapsed
```

This document took 245.06 seconds to compile in Rmarkdown.

8 Unsorted references.

The main web references of this document are (these are web links):

- Time Series Machine Learning with h2o and timetk.
- Time Series Machine Learning with timetk.
- Tidy Forecasting with sweep
- H2O Tutorials
- Machine Learning to Reduce Employee Attrition

References.

Donald Ervin Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.