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: 04/12/2022, 00:14:20.

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.


```
## Warning: package 'tictoc' was built under R version 4.2.2
```

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.

```
# 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.

```
# 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.")
```

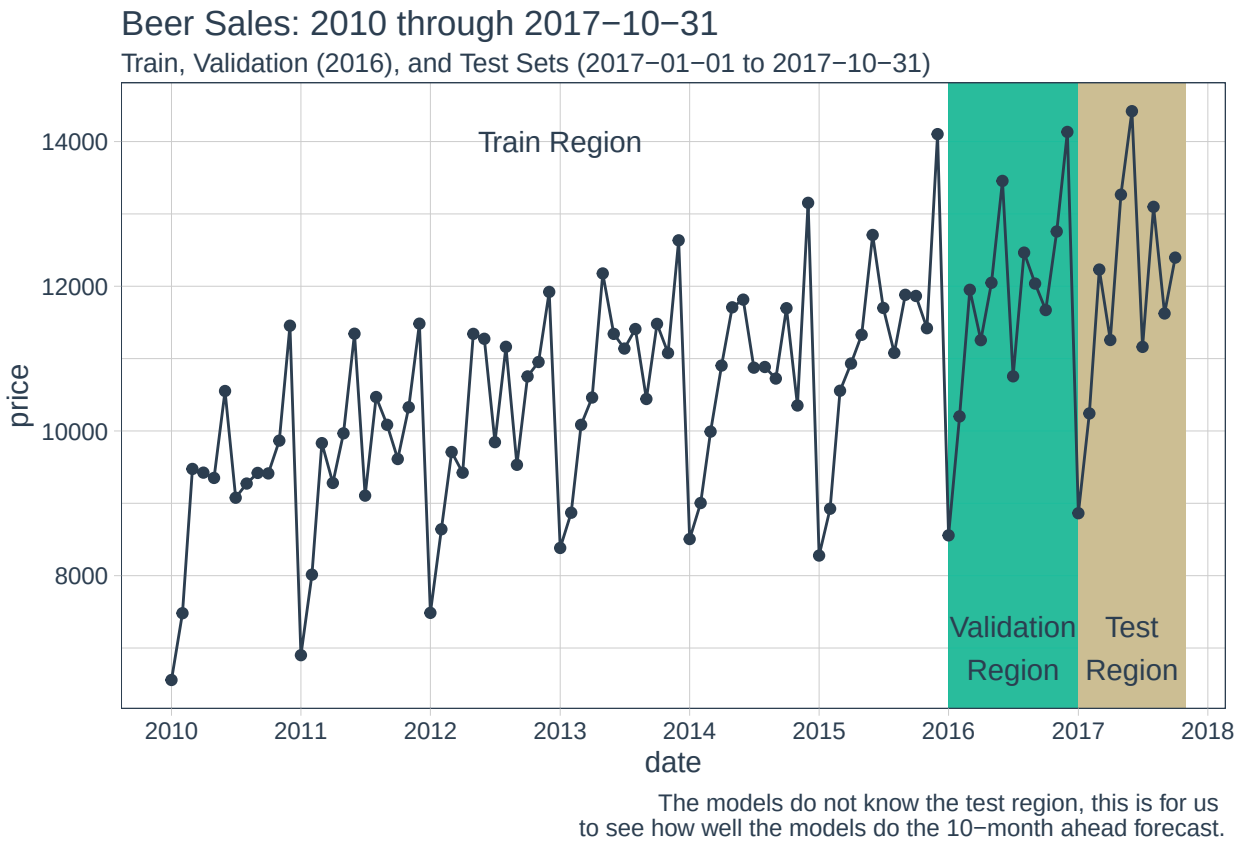

Figure 2.0.1: Beer, Wine, and Distilled Alcoholic Beverages Sales.

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.

```
# 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, 13266, 14420, 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\RtmpcPm7XQ\file27587636b6d/h2o_ML_started_from_r.o
##     C:\Users\ML\AppData\Local\Temp\RtmpcPm7XQ\file275849964e4e/h2o_ML_started_from_r.
##
##
## Starting H2O JVM and connecting: . Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         5 seconds 248 milliseconds
##     H2O cluster timezone:       America/Mexico_City
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.38.0.1
##     H2O cluster version age:    2 months and 14 days
##     H2O cluster name:           H2O_started_from_R_ML_tkw311
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.52 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
##     R Version:                  R version 4.2.1 (2022-06-23 ucrt)
```

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

```
# 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.

```
# 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.

```
# 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)
```

```
##
## 00:14:56.732: User specified a validation frame with cross-validation still enabled.
## 00:14:56.746: AutoML: XGBoost is not available; skipping it.
## 00:14:56.904: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:14:57.478: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:14:57.478: _min_rows param, The dataset size is too small to split for min_rows=10
## 00:14:57.484: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:14:58.328: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
```

```
## 00:14:59.166: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:14:59.922: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:15:00.575: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:15:01.284: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:15:01.766: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:15:02.421: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:15:02.929: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:15:03.293: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:15:13.183: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:15:13.795: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:15:14.310: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:15:14.667: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:15:15.142: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:15:15.991: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:15:16.245: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:15:31.833: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:15:32.202: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
```

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	rmse	mse	mae	rmsle	mean_residual_deviance
1	StackedEnsemble_BestOfFamily_6_AutoML_1_20221204_01456	664.22	441183.1	561.90	0.05	441183.1
2	StackedEnsemble_AllModels_4_AutoML_1_20221204_01456	695.00	483027.8	561.94	0.06	483027.8
3	GBM_grid_1_AutoML_1_20221204_01456_model_97	701.38	491936.6	577.80	0.06	491936.6
4	GBM_grid_1_AutoML_1_20221204_01456_model_81	762.86	581955.5	626.21	0.06	581955.5
5	GBM_grid_1_AutoML_1_20221204_01456_model_15	784.98	616193.8	576.39	0.06	616193.8
6	StackedEnsemble_BestOfFamily_5_AutoML_1_20221204_01456	807.55	652130.2	612.01	0.06	652130.2
7	GBM_grid_1_AutoML_1_20221204_01456_model_71	807.73	652433.7	636.73	0.07	652433.7
8	StackedEnsemble_BestOfFamily_4_AutoML_1_20221204_01456	814.59	663559.2	672.93	0.07	663559.2
9	GBM_grid_1_AutoML_1_20221204_01456_model_66	818.71	670286.2	653.33	0.07	670286.2
10	GBM_grid_1_AutoML_1_20221204_01456_model_12	842.30	709469.2	688.18	0.07	709469.2

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")
```

```
##
## 00:22:19.645: User specified a validation frame with cross-validation still enabled.
## 00:22:19.682: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
```

The selected model by `h2o.automl` is:

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

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

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

```
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	rmse	mse	mae	rmsle	mean_residual_deviance
1	DeepLearning_grid_1_AutoML_2_20221204_02219_model_4	474.79	225423.3	337.36	0.04	225423.3
2	DeepLearning_grid_1_AutoML_2_20221204_02219_model_6	522.11	272594.6	418.20	0.04	272594.6
3	DeepLearning_grid_1_AutoML_2_20221204_02219_model_2	572.08	327271.7	369.97	0.05	327271.7
4	DeepLearning_grid_1_AutoML_2_20221204_02219_model_3	600.26	360306.7	376.52	0.05	360306.7
5	DeepLearning_grid_1_AutoML_2_20221204_02219_model_1	862.36	743667.1	725.31	0.07	743667.1
6	DeepLearning_1_AutoML_2_20221204_02219	1034.87	1070948.6	734.74	0.09	1070948.6
7	DeepLearning_grid_1_AutoML_2_20221204_02219_model_8	1320.06	1742559.5	1129.97	0.12	1742559.5
8	DeepLearning_grid_1_AutoML_2_20221204_02219_model_7	1667.46	2780415.3	1211.63	0.15	2780415.3
9	DeepLearning_grid_1_AutoML_2_20221204_02219_model_5	2002.73	4010908.7	1698.53	0.17	4010908.7

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")
```

```
##
## 00:25:26.948: User specified a validation frame with cross-validation still enabled.
## 00:25:26.950: AutoML: XGBoost is not available; skipping it.
## 00:25:26.960: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:25:27.72: _train param, Dropping bad and constant columns: [symbol, hour, am.pm, m
## 00:25:27.72: _min_rows param, The dataset size is too small to split for min_rows=100
```

```
## 00:25:27.73: _train param, Dropping bad and constant columns: [symbol, hour, am.pm, m
## 00:25:27.357: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:25:27.797: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:25:28.77: _train param, Dropping bad and constant columns: [symbol, hour, am.pm, m
## 00:25:28.413: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:25:28.825: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:25:29.181: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:25:29.594: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:25:29.928: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
## 00:25:30.66: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mday
## 00:25:30.439: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:26:19.38: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mday
## 00:26:19.402: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:26:25.753: _train param, Dropping unused columns: [symbol, hour, am.pm, mday7, mda
## 00:26:26.896: _train param, Dropping bad and constant columns: [symbol, hour, am.pm,
```

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	rmse	mse	mae	rmsle	mean_residual_deviance
1	DeepLearning_grid_1_AutoML_3_20221204_02526_model_3	504.84	254861.3	423.30	0.04	254861.3
2	StackedEnsemble_BestOfFamily_3_AutoML_3_20221204_02526	544.80	296802.3	441.17	0.04	296802.3
3	DeepLearning_grid_2_AutoML_3_20221204_02526_model_3	711.77	506610.9	582.41	0.06	506610.9
4	GBM_grid_1_AutoML_3_20221204_02526_model_23	734.00	538752.8	606.90	0.06	538752.8
5	GBM_grid_1_AutoML_3_20221204_02526_model_88	752.83	566756.6	566.54	0.06	566756.6
6	DeepLearning_grid_1_AutoML_3_20221204_02526_model_4	765.37	585791.2	677.46	0.07	585791.2
7	GBM_grid_1_AutoML_3_20221204_02526_model_12	842.26	709407.5	633.96	0.07	709407.5
8	GBM_grid_1_AutoML_3_20221204_02526_model_134	846.36	716327.7	703.82	0.07	716327.7
9	GBM_grid_1_AutoML_3_20221204_02526_model_70	851.59	725200.4	676.65	0.07	725200.4
10	GBM_grid_1_AutoML_3_20221204_02526_model_68	851.96	725840.9	747.57	0.07	725840.9

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
664.22	474.79	504.84

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	9281.016	-418.016	-0.047
2	S4248SM144NCEN	2017-02-01	10242	9910.008	331.992	0.032
3	S4248SM144NCEN	2017-03-01	12231	11885.538	345.462	0.028
4	S4248SM144NCEN	2017-04-01	11257	11911.312	-654.312	-0.058
5	S4248SM144NCEN	2017-05-01	13266	13107.128	158.872	0.012
6	S4248SM144NCEN	2017-06-01	14420	12942.466	1477.534	0.102
7	S4248SM144NCEN	2017-07-01	11162	11974.161	-812.161	-0.073
8	S4248SM144NCEN	2017-08-01	13098	12465.864	632.136	0.048
9	S4248SM144NCEN	2017-09-01	11624	12040.000	-416.000	-0.036
10	S4248SM144NCEN	2017-10-01	12396	12768.562	-372.562	-0.030

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)))))
```

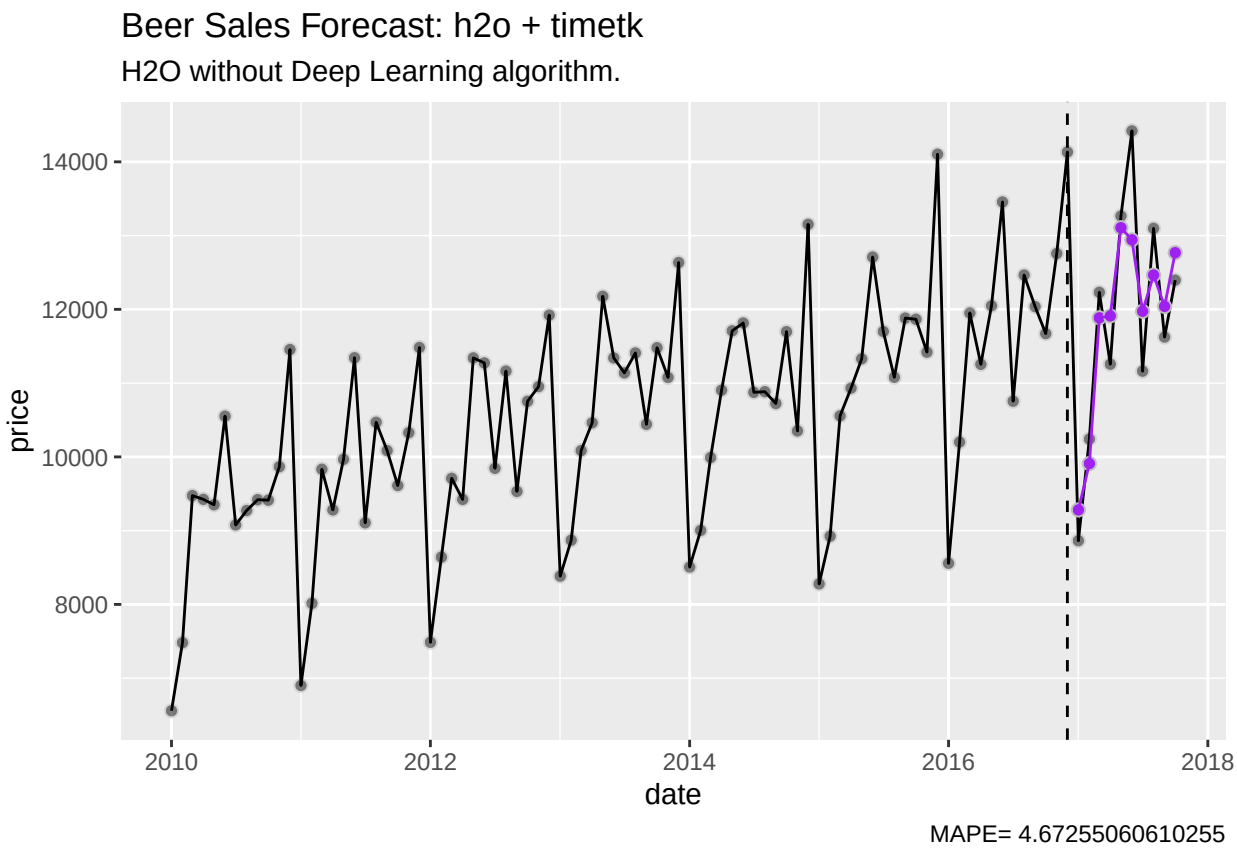

Figure 3.4.1: Forecast: H2O without Deep Learning algorithm.

This is an additional performance summary.

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

```
## H2ORegressionMetrics: stackedensemble
##
## MSE:  441183.1
## RMSE:  664.2162
## MAE:  561.9047
## RMSLE:  0.05342514
## Mean Residual Deviance :  441183.1
```

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	9435.365	-572.365	-0.065
2	S4248SM144NCEN	2017-02-01	10242	10334.468	-92.468	-0.009
3	S4248SM144NCEN	2017-03-01	12231	12119.272	111.728	0.009
4	S4248SM144NCEN	2017-04-01	11257	10839.661	417.339	0.037
5	S4248SM144NCEN	2017-05-01	13266	12918.161	347.839	0.026
6	S4248SM144NCEN	2017-06-01	14420	13195.743	1224.257	0.085
7	S4248SM144NCEN	2017-07-01	11162	11347.715	-185.715	-0.017
8	S4248SM144NCEN	2017-08-01	13098	12849.893	248.107	0.019
9	S4248SM144NCEN	2017-09-01	11624	11726.396	-102.396	-0.009
10	S4248SM144NCEN	2017-10-01	12396	12324.613	71.387	0.006

The same information in a plot.

```
# 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)))))
```

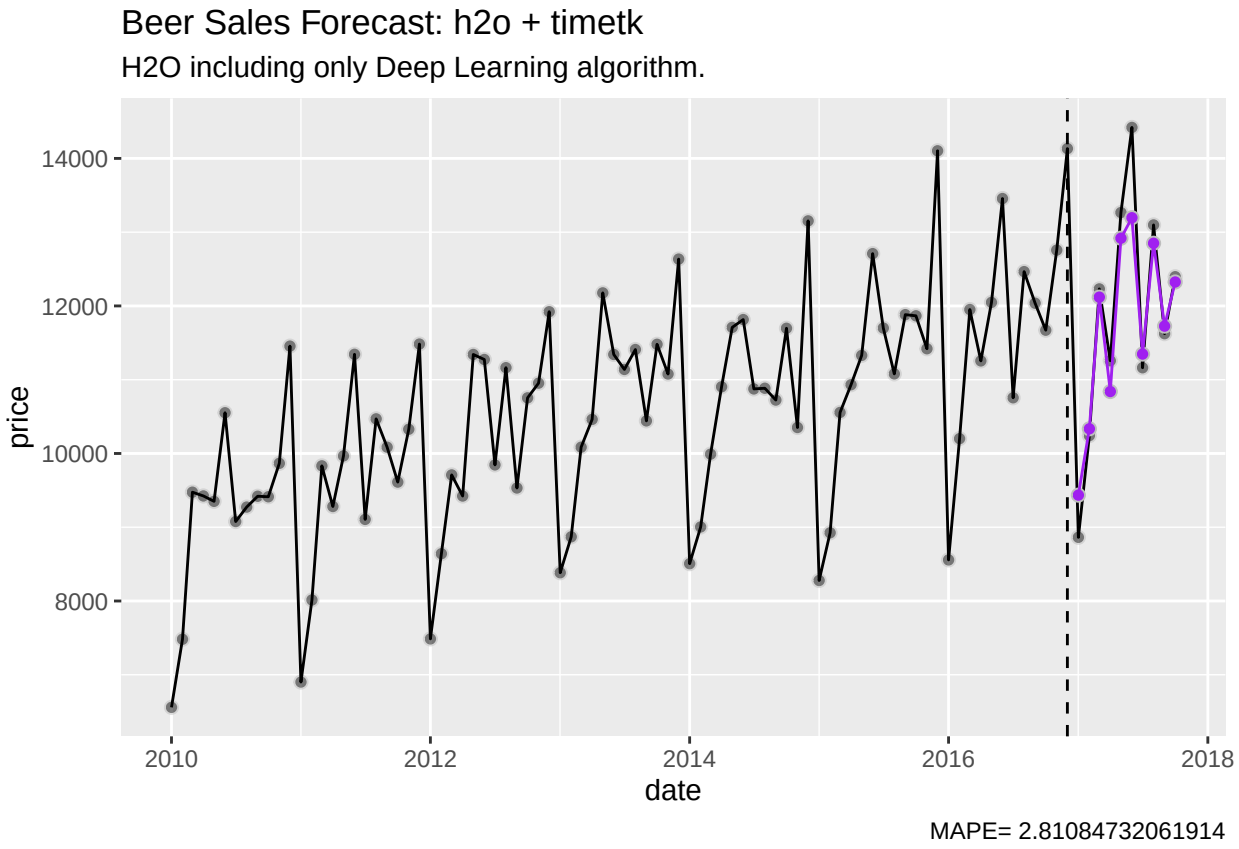

Figure 3.4.2: Forecast: H2O including only Deep Learning algorithm.

Additional performance indicators.

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

```
## H2ORegressionMetrics: deeplearning
##
## MSE:  225423.3
```

```
## RMSE:  474.7877
## MAE:  337.3602
## RMSLE:  0.03851199
## Mean Residual Deviance :  225423.3
```

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

```
# 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	9627.255	-764.255	-0.086
2	S4248SM144NCEN	2017-02-01	10242	10521.443	-279.443	-0.027
3	S4248SM144NCEN	2017-03-01	12231	11917.591	313.409	0.026
4	S4248SM144NCEN	2017-04-01	11257	10666.073	590.927	0.052
5	S4248SM144NCEN	2017-05-01	13266	13064.234	201.766	0.015
6	S4248SM144NCEN	2017-06-01	14420	13409.299	1010.701	0.070
7	S4248SM144NCEN	2017-07-01	11162	11426.245	-264.245	-0.024
8	S4248SM144NCEN	2017-08-01	13098	13124.522	-26.522	-0.002
9	S4248SM144NCEN	2017-09-01	11624	12033.564	-409.564	-0.035
10	S4248SM144NCEN	2017-10-01	12396	12023.812	372.188	0.030

The visual representation.

```
# 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)))))
```

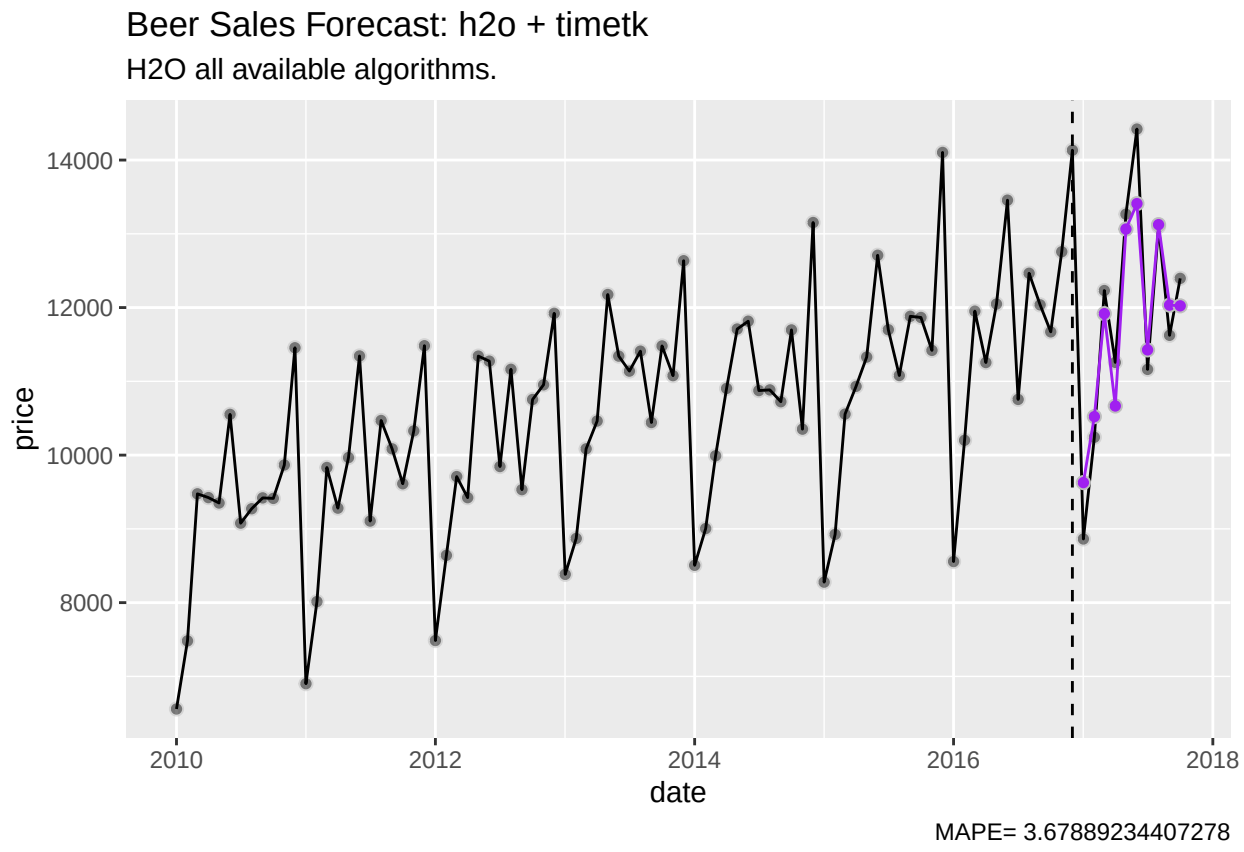

Figure 3.4.3: Forecast: H2O including all available algorithms.

Additional performance metrics.

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

```
## H2ORegressionMetrics: deeplearning
##
## MSE:  254861.3
## RMSE:  504.8379
## MAE:  423.3018
## RMSLE:  0.04397654
## Mean Residual Deviance :  254861.3
```

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.

```
# 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	27.29	664.22	561.90	4.67	-0.21
Only Deep Learning	deeplearning	146.77	474.79	337.36	2.81	0.83
All algorithms	deeplearning	74.50	504.84	423.30	3.68	0.19

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	2017-05-01	0.011976
Only Deep Learning	deeplearning	2017-10-01	0.005759
All algorithms	deeplearning	2017-08-01	-0.002025

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
## -519.21 -165.00  -15.29  163.61  686.28
##
## Coefficients: (16 not defined because of singularities)
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.022e+08  1.128e+08   2.678 0.009303 **
## index.num     4.873e-03  1.815e-03   2.685 0.009132 **
## year         -1.572e+05  5.840e+04  -2.691 0.008982 **
## year.iso      3.777e+03  5.850e+03   0.646 0.520766
## half         -2.465e+03  6.322e+02  -3.899 0.000226 ***
## quarter      -2.143e+04  2.376e+04  -0.902 0.370449
## month        -2.577e+03  7.949e+03  -0.324 0.746795
## month.xts            NA         NA      NA       NA
## month.lbl.L          NA         NA      NA       NA
## month.lbl.Q  -1.774e+03  2.219e+02  -7.994 2.47e-11 ***
## month.lbl.C   6.582e+02  5.453e+02   1.207 0.231672
## month.lbl^4   7.318e+02  1.437e+02   5.093 3.08e-06 ***
## month.lbl^5   8.769e+02  4.437e+02   1.976 0.052245 .
## month.lbl^6   3.169e+02  1.720e+02   1.842 0.069845 .
## month.lbl^7  -4.167e+02  2.019e+02  -2.064 0.042892 *
## month.lbl^8   3.483e+02  3.450e+02   1.010 0.316320
## month.lbl^9          NA         NA      NA       NA
## month.lbl^10  6.006e+02  2.383e+02   2.521 0.014088 *
## 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         -5.961e+01  2.187e+01  -2.726 0.008174 **
## wday.xts             NA         NA      NA       NA
```

```
## wday.lbl.L           NA         NA      NA       NA
## wday.lbl.Q   -8.372e+02  1.090e+02  -7.683 8.96e-11 ***
## wday.lbl.C    1.519e+02  9.565e+01   1.588 0.116892
## wday.lbl^4    1.158e+02  1.136e+02   1.020 0.311541
## wday.lbl^5    6.367e+01  9.740e+01   0.654 0.515538
## wday.lbl^6    1.080e+02  8.678e+01   1.244 0.217827
## mday                 NA         NA      NA       NA
## qday         -2.360e+02  2.623e+02  -0.900 0.371500
## yday         -7.499e+01  1.188e+02  -0.631 0.530100
## mweek        -1.044e+02  1.537e+02  -0.679 0.499297
## week         -1.296e+02  1.996e+02  -0.649 0.518244
## week.iso      8.073e+01  1.130e+02   0.715 0.477326
## week2        -1.122e+02  1.676e+02  -0.669 0.505648
## 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: 282.6 on 67 degrees of freedom
## Multiple R-squared:  0.977,  Adjusted R-squared:  0.968
## F-statistic: 109.3 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 13093.
##  6 2017-06-01 13734.
##  7 2017-07-01 11400.
##  8 2017-08-01 13059.
##  9 2017-09-01 11795.
## 10 2017-10-01 12337.
```

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.   11.1  0.000909
##  4 2017-04-01  11257 11267.  -10.1 -0.000897
##  5 2017-05-01  13266 13093.  173.   0.0131
##  6 2017-06-01  14420 13734.  686.   0.0476
##  7 2017-07-01  11162 11400. -238.  -0.0213
##  8 2017-08-01  13098 13059.   38.8  0.00297
##  9 2017-09-01  11624 11795. -171.  -0.0147
## 10 2017-10-01  12396 12337.   58.5  0.00472
```

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> -43.73911
## $ rmse <dbl> 328.3597
## $ mae  <dbl> 237.3146
## $ mape <dbl> 2.098322
## $ mpe  <dbl> -0.7135763
```

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))))))
```

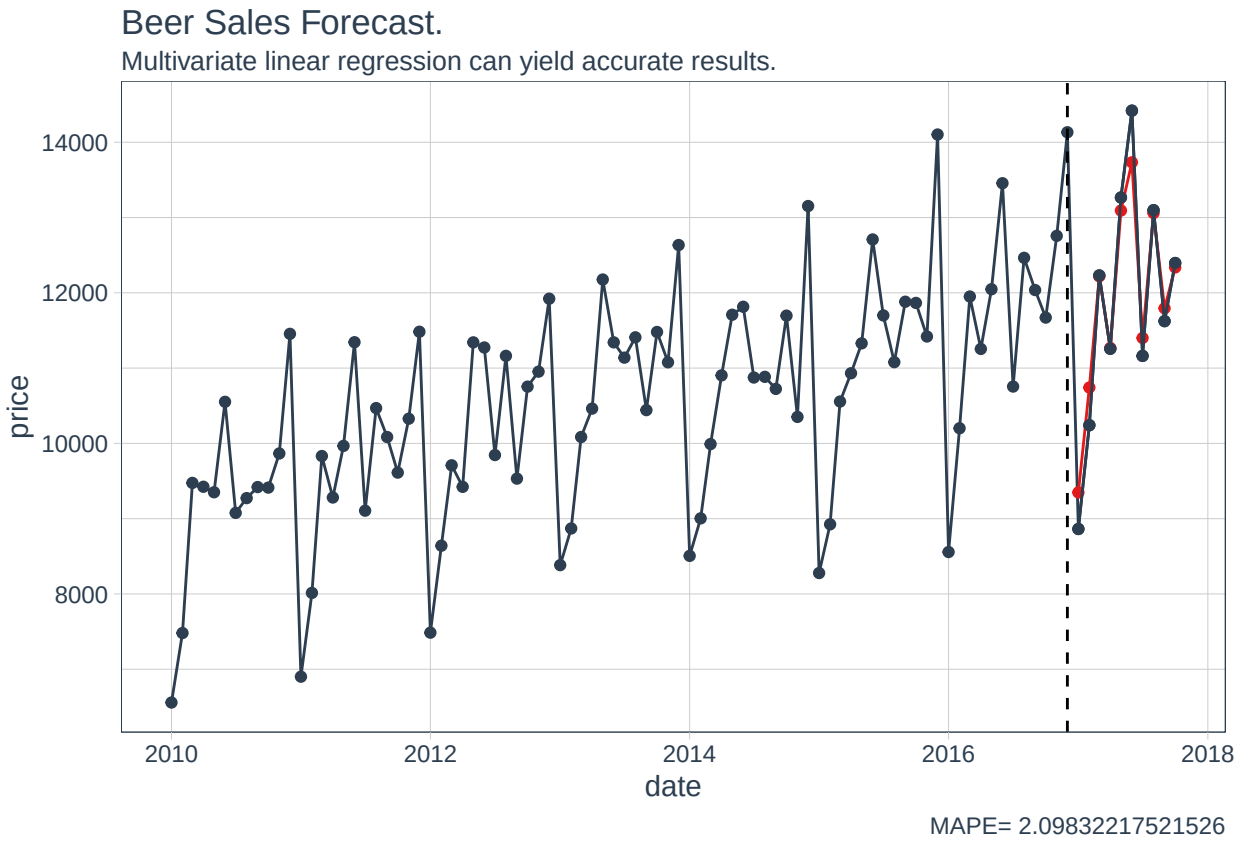

Figure 4.2.1: Forecast: multivariate linear regression.

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).

```
# 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  9004  9991 10904 11709 11815 10875 10884 10724 11697 10352 13153
## 2015  8278  8925 10556 10932 11329 12709 11700 11079 11882 11866 11420 14103
## 2016  8557 10201 11951 11255 12048 13456 10755 12465 12037 11671 12757 14133
```

Just verify `tk_ts` worked.

```
# 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.

```
# 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
##       -0.2675  0.0959  0.6028  -0.6722  33.5698
## s.e.   0.0956  0.1015  0.1036   0.5620   3.1931
##
## sigma^2 = 153501:  log likelihood = -533.76
## AIC=1079.53   AICc=1080.82   BIC=1093.19
```

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     0.0959
## 3 ar3     0.603
## 4 sma1   -0.672
## 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> 391.792
## $ logLik     <dbl> -533.7643
## $ AIC        <dbl> 1079.529
## $ BIC        <dbl> 1093.189
## $ ME         <dbl> 8.985371
## $ RMSE       <dbl> 349.9076
```

```
## $ MAE         <dbl> 248.7835
## $ MPE         <dbl> -0.04120242
## $ MAPE        <dbl> 2.330129
## $ MASE        <dbl> 0.453192
## $ ACF1        <dbl> -0.003762408
```

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()
```

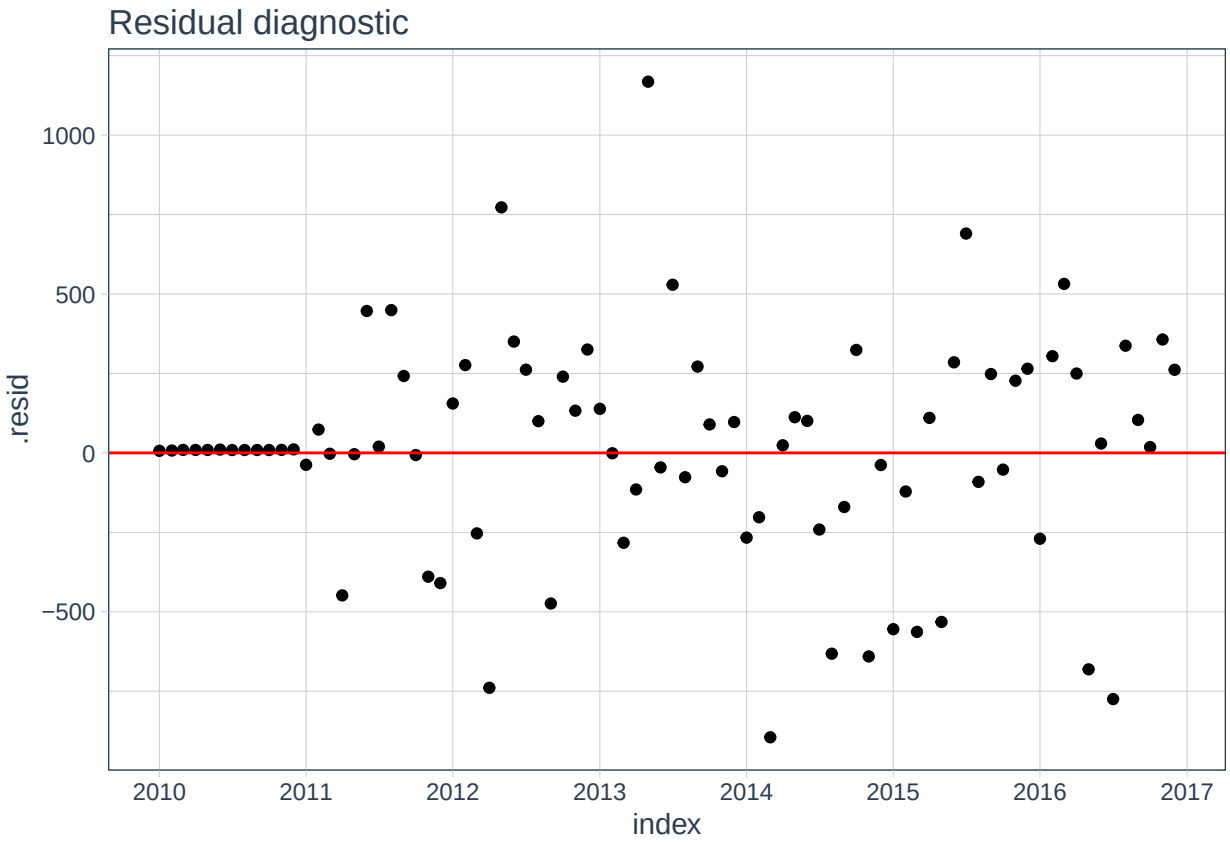

Figure 5.3.1: Forecast: ARIMA residual diagnosis.

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       8969.785  8466.884  9472.685  8200.665  9738.904
## Feb 2017      10923.899 10403.333 11444.466 10127.762 11720.037
## Mar 2017      11688.045 11160.728 12215.362 10881.583 12494.506
```

```
## Apr 2017      11705.253 11114.063 12296.442 10801.106 12609.399
## May 2017      13023.871 12415.296 13632.445 12093.136 13954.605
## Jun 2017      13288.918 12669.565 13908.272 12341.699 14236.138
## Jul 2017      11913.425 11283.224 12543.627 10949.615 12877.236
## Aug 2017      12730.228 12091.005 13369.452 11752.621 13707.836
## Sep 2017      12136.012 11487.539 12784.485 11144.258 13127.766
## Oct 2017      12585.078 11935.917 13234.239 11592.272 13577.884
```

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  8970. -107.   -0.0120
##  2 2017-02-01  10242 10924. -682.   -0.0666
##  3 2017-03-01  12231 11688.  543.    0.0444
##  4 2017-04-01  11257 11705. -448.   -0.0398
##  5 2017-05-01  13266 13024.  242.    0.0183
##  6 2017-06-01  14420 13289. 1131.    0.0784
##  7 2017-07-01  11162 11913. -751.   -0.0673
##  8 2017-08-01  13098 12730.  368.    0.0281
##  9 2017-09-01  11624 12136. -512.   -0.0440
## 10 2017-10-01  12396 12585. -189.   -0.0153
```

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> -40.55136
## $ rmse <dbl> 575.1431
## $ mae  <dbl> 497.339
## $ mape <dbl> 4.142284
## $ mpe  <dbl> -0.7590743
```

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()
```

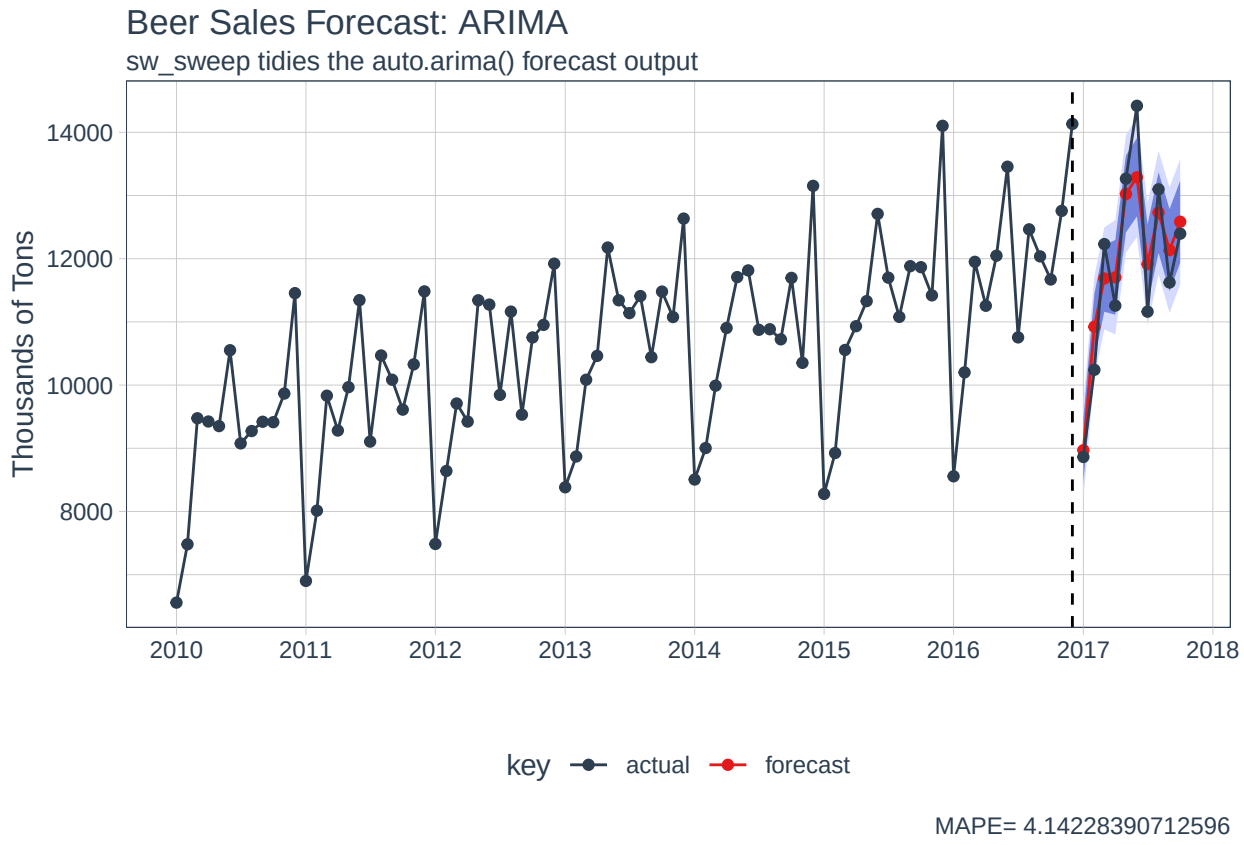

Figure 5.3.2: Forecast: ARIMA.

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  9332. -469.  -0.0530
##  2 2017-02-01  10242 10486. -244.  -0.0239
##  3 2017-03-01  12231 11966.  265.   0.0217
##  4 2017-04-01  11257 11278.  -20.9 -0.00185
##  5 2017-05-01  13266 13041.  225.   0.0169
##  6 2017-06-01  14420 13314. 1106.   0.0767
##  7 2017-07-01  11162 11612. -450.  -0.0403
##  8 2017-08-01  13098 12846.  252.   0.0192
##  9 2017-09-01  11624 11946. -322.  -0.0277
## 10 2017-10-01  12396 12408.  -11.9 -0.000961
```

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> 32.85431
## $ rmse <dbl> 446.6237
## $ mae  <dbl> 336.6895
## $ mape <dbl> 0.02822534
## $ mpe  <dbl> -0.001316507
```

Visualize the average 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_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)))))))
```

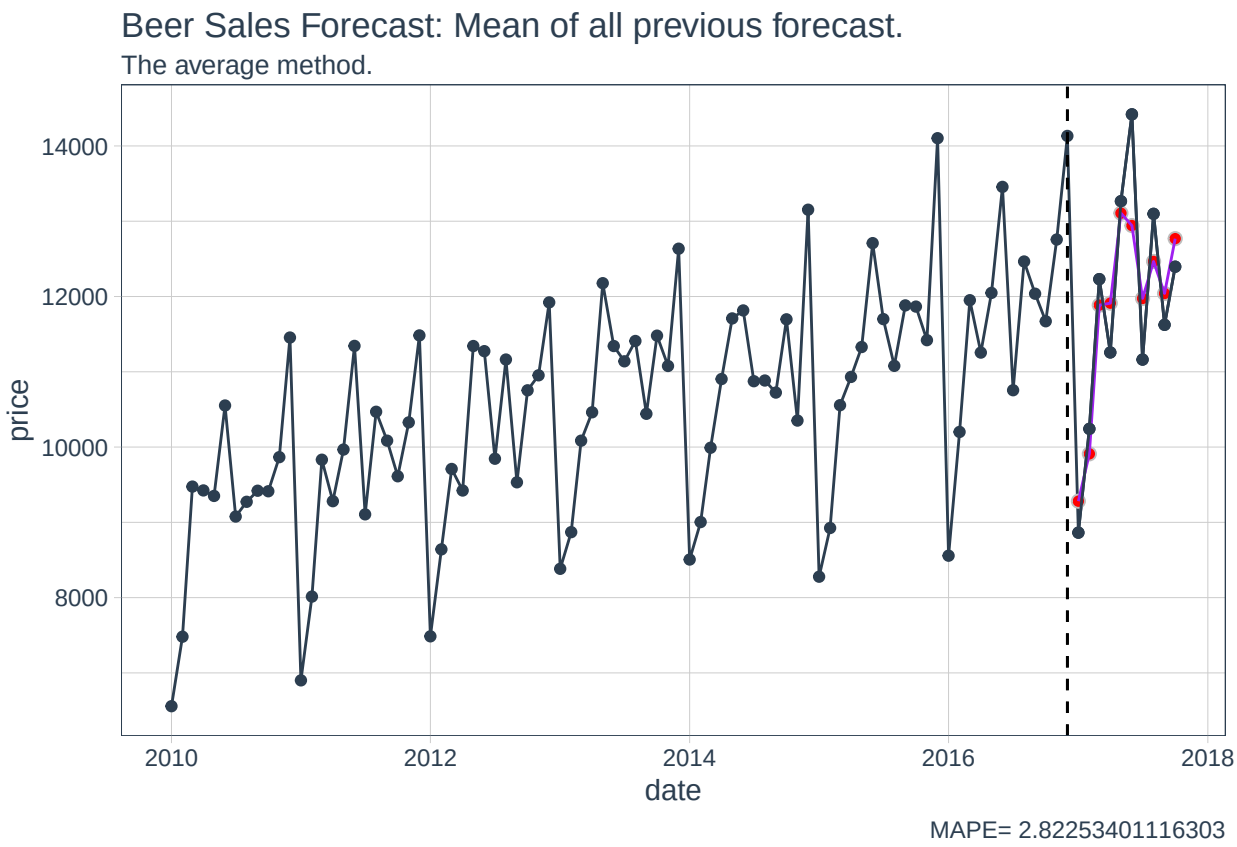

Figure 6.0.1: Forecast: average forecast.

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	4.672551
H2O including only Deep Learning algorithm	2.810847
H2O all available algorithms	3.678892
Multivariate linear regression	2.098322
ARIMA	4.142284
Average	2.822534

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()
```

```
## 1215.75 sec elapsed
```

This document took 1215.75 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.