

# Time Series Analysis Methods and Applications

*by Igor Baranov, Michael Parravani, Ariana Biagi, Hui Fang Cai*

**Abstract** The goal of this project is to discover Time Series analysis. We start with the data loading and creation of different types of object including `ts`, `xts` and `zoo` and proceed with data manipulation, conversion and visualization. Then we discover different algorithms of time series analysis like decomposition, forecasting and clustering. At the end we develop a practical example and build a Shiny application.

## Introduction

The goal of this project is to discover Time Series analysis. We start with the data loading and creation of different types of object including `ts`, `xts` and `zoo` and proceed with data manipulation, conversion and visualization. Then we discover different algorithms of time series analysis like decomposition, forecasting and clustering. At the end we develop a practical example and build a Shiny application.

## Background

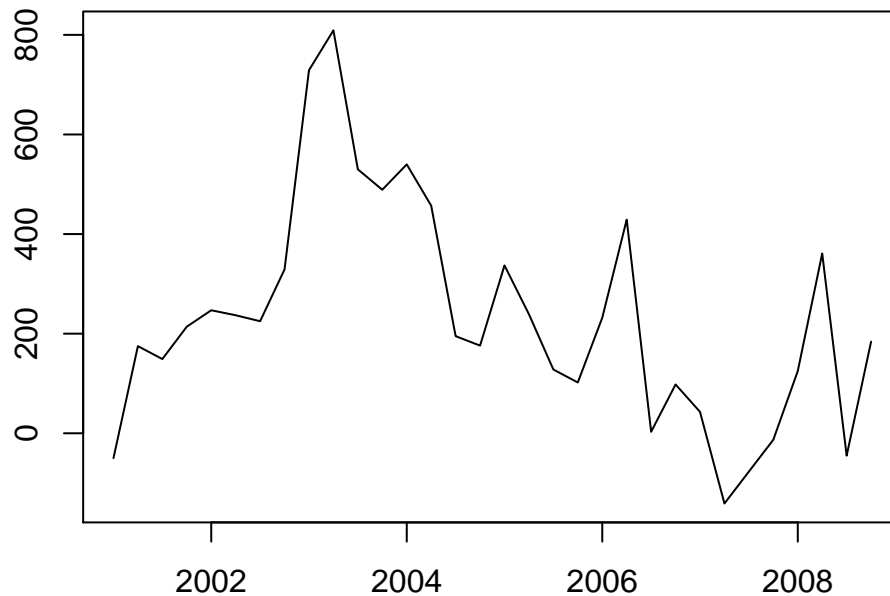
The function `ts` for the core package `stats` (R Core Team, 2012) is used to create time-series objects. These are vector or matrices with class of “`ts`” (and additional attributes) which represent data which has been sampled at equispaced points in time. In the matrix case, each column of the matrix data is assumed to contain a single (univariate) time series. Time series must have at least one observation, and although they need not be numeric there is very limited support for non-numeric series.

An `xts` object from package `xts` (Ryan and Ulrich, 2018) extends the S3 class `zoo` from the package of the same name (Zeileis and Grothendieck, 2005). Package `zoo` is the creator for an S3 class of indexed totally ordered observations which includes irregular time series.

Similar to `zoo` objects, `xts` objects must have an ordered index. While `zoo` indexes cannot contain duplicate values, `xts` objects have optionally supported duplicate index elements since version 0.5-0. The `xts` class has one additional requirement, the index must be a time-based class. Currently supported classes include: ‘Date’, ‘POSIXct’, ‘timeDate’, as well as ‘yearmon’ and ‘yearqtr’ where the index values remain unique.

## Ethical Consideration for the Time Series ML Framework

As the goal of this report is only to research the time series methods, many aspects of the ethical ML framework do not directly apply. The time series data used here is open source and we can assume was collected in transparent ways. That being said, there is likely a large segment of the populace that can’t take advantage of analysis presented in this report - we assume a low income population with limited access to internet and not STEM educated. If the outcome of this system were to be of more social impact, this would need to be use more appropriate datasets, analysis of which could be more advantageous to that population.



**Figure 1:** Time Series ts class Example Plot

## Time Series Data Manipulating and Visualizing

### Constructing TS object

In the following example (Fig 1) we construct and plot a simple TS class:

```
simpleTS <- c(-50, 175, 149, 214, 247, 237, 225, 329, 729, 809,
             530, 489, 540, 457, 195, 176, 337, 239, 128, 102, 232, 429, 3,
             98, 43, -141, -77, -13, 125, 361, -45, 184)
simpleTS <- ts(simpleTS, start = c(2001, 1), end = c(2008, 4), frequency = 4)
print(simpleTS)

#>      Qtr1 Qtr2 Qtr3 Qtr4
#> 2001  -50  175  149  214
#> 2002   247   237   225   329
#> 2003   729   809   530   489
#> 2004   540   457   195   176
#> 2005   337   239   128   102
#> 2006   232   429     3    98
#> 2007    43  -141   -77   -13
#> 2008   125   361   -45   184

plot(simpleTS, ylab="", xlab="")
```

### Loading Stock Data

There are many ways to load times data, but the fastest is to use [zoo](#) that has many convenient utilities to manipulate times series data. This is especially convenient when dealing with complex stock data.

This is an example of loading and presenting of stock data using [Historical stock data for all current S&P 500 companies](#). Stock market data can be interesting to analyze and as a further incentive, strong predictive models can have large financial payoff. Data has the following columns:

- Date - in format: yy-mm-dd
- Open - price of the stock at market open (this is NYSE data so all in USD)
- High - Highest price reached in the day
- Low Close - Lowest price reached in the day
- Volume - Number of shares traded
- Name - the stock's ticker name

First we load the whole dataset as data frame:

```
stocks5 <- read_csv(file="../../data/all_stocks_5yr.csv.zip", col_names = TRUE)
stocks5$Name <- as.factor(stocks5$Name)
head(stocks5$Name)
```

```
#> [1] AAL AAL AAL AAL AAL AAL
#> 505 Levels: A AAL AAP AAPL ABBV ABC ABT ACN ADBE ADI ADM ADP ADS ... ZTS
```

As the name suggests we should have 500 (505 to be correct) stock names in the dataset. The code below extracts stock data of AAL (American Airlines Group Inc), converts it to **zoo** object and plots it as multi-variate time series (Fig 2).

```
stocks5_aal <- stocks5[stocks5$Name=="AAL", c(1:6)]
stocks5_aal <- zoo(stocks5_aal[, 2:6], stocks5_aal$date)
print(paste("Start date: ", start(stocks5_aal)))

#> [1] "Start date: 2013-02-08"

print(paste("Last date: ", end(stocks5_aal)))

#> [1] "Last date: 2018-02-07"

str(stocks5_aal)

#> 'zoo' series from 2013-02-08 to 2018-02-07
#> Data: num [1:1259, 1:5] 15.1 14.9 14.4 14.3 14.9 ...
#> - attr(*, "dimnames")=List of 2
#> ..$ : NULL
#> ..$ : chr [1:5] "open" "high" "low" "close" ...
#> Index: Date[1:1259], format: "2013-02-08" "2013-02-11" "2013-02-12" "2013-02-13" "2013-02-14" ...

plot(stocks5_aal, xlab = "", nc = 1, main = "")
```

## Additional time series data sets

### Yahoo Science labeled time series

Yahoo Science labeled time series is a big Synthetic and real time-series with labeled anomalies, it should be downloaded and used locally. [Yahoo Science labeled time series](#)

### NAB Data Corpus

[NAB Data Corpus](#) is a large dataset of ordered, timestamped, single-valued metrics. All data files contain anomalies, unless otherwise noted. This data better be loaded from github directly to R script.

## Time Series Methods Showcase

### Time series decomposition

Time series decomposition is to decompose a time series into trend, seasonal, cyclical and irregular components. Frequency represents data which has been sampled at equispaced points in time:

- frequency=7: a weekly series
- frequency=12: a monthly series
- frequency=4: a quarterly series

To decompose a time series into components:

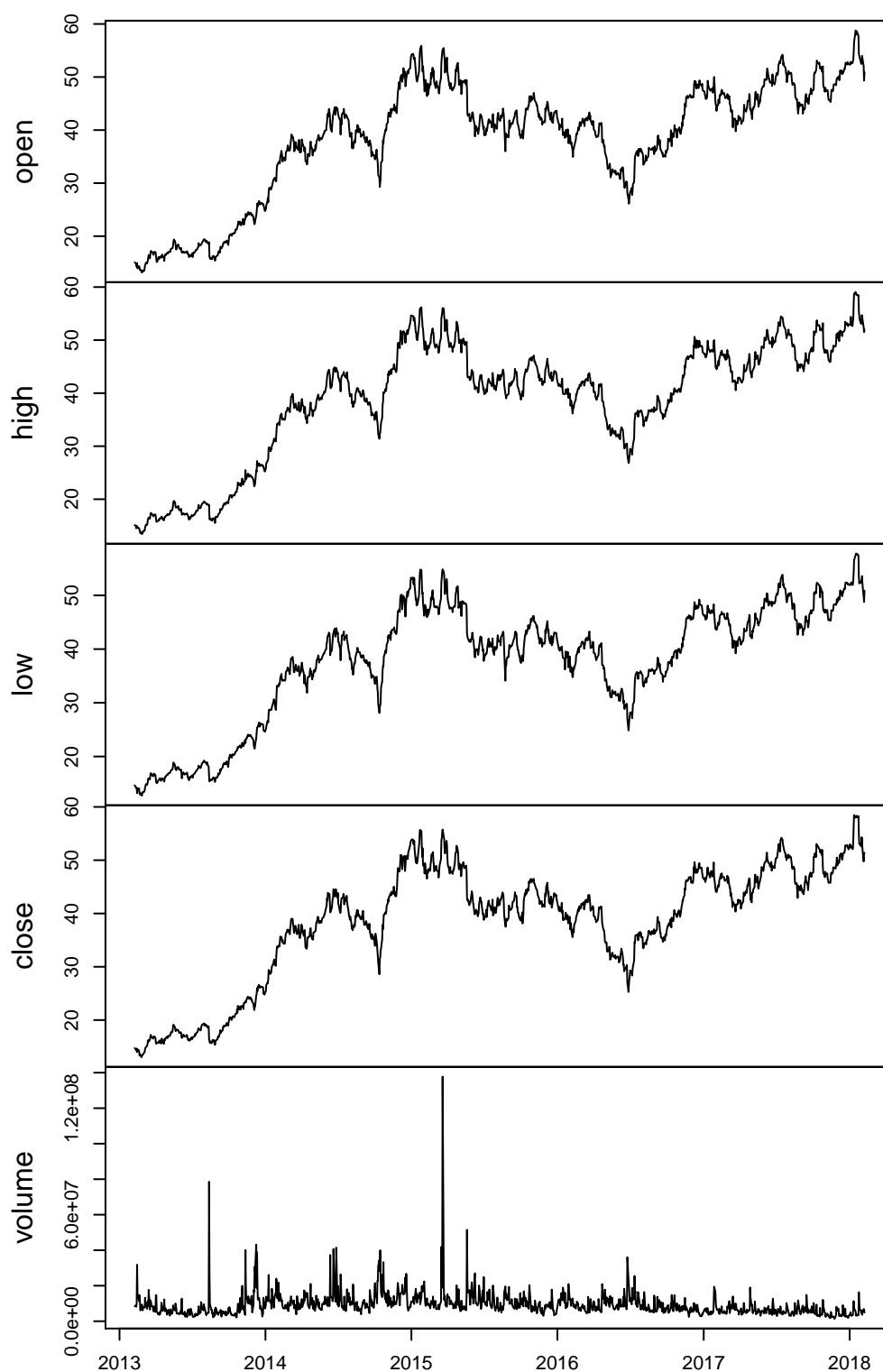
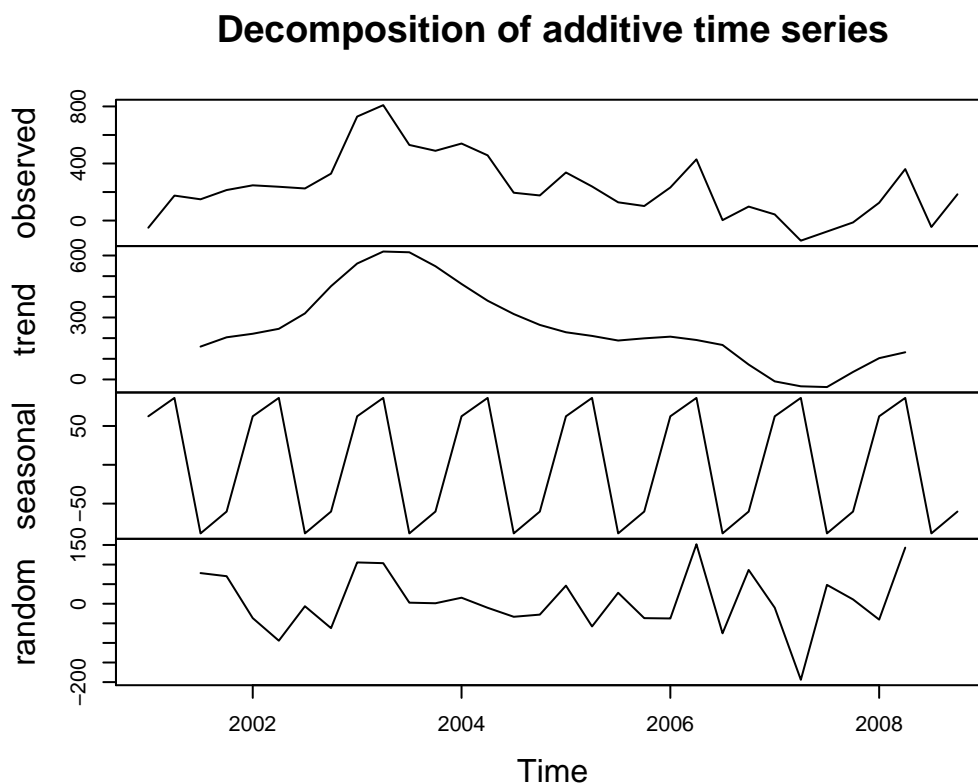


Figure 2: Multi-variate time series visualization of AAL stock



**Figure 3:** Decomposition of cyclic object example

- Trend component: long term trend
- Seasonal component: seasonal variation
- Cyclical component: repeated but non-periodic fluctuations
- Irregular component: the residuals

A **simpleTS** time series object was constructed in section [Constructing TS object](#). It is used below as an example to demonstrate time series decomposition (Fig 3). It was constructed to have quarterly data and will be decomposed with frequency 4.

```
m <- decompose(simpleTS)
plot(m)
```

A more complex example of time series manipulation and decomposition presented below. We will use 'open' series of the object `stocks5_aal` created in section [Loading Stock Data](#). The full daily chart of this series presented on (Fig. 4). To decompose the series, code below calculates yearly cycles of the last 4 years, data aggregated monthly (Fig 5).

```
last4 <- window(stocks5_aal$open, start=as.Date("2014-01-01"), end=as.Date("2017-12-31"))
plot(last4)
```

```
require(xts)
last4_mo <- last4[endpoints(last4, "month")]
m <- decompose(ts(last4_mo, frequency = 12))
plot(m, xlab="", xaxt="n")
axis(1, at=1:5, labels=c(2014,2015,2016,2017,2018), pos = -3.9)
```

### Time series forecasting

Time series forecasting is to forecast future events based on known past data. To forecast future events based on known past data For example, to predict the price of a stock based on its past performance. Popular models are:

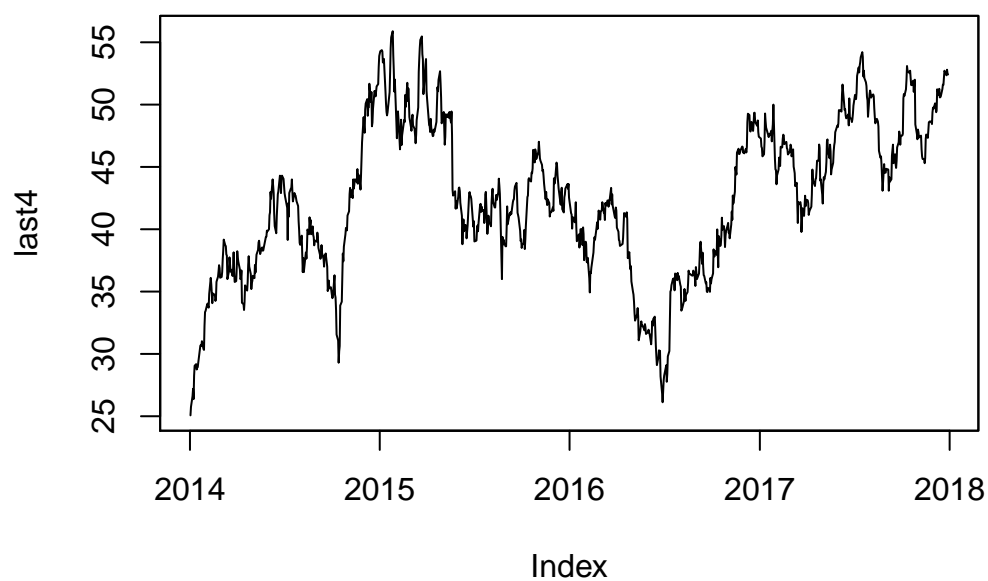


Figure 4: 4 years AAL 'open' series

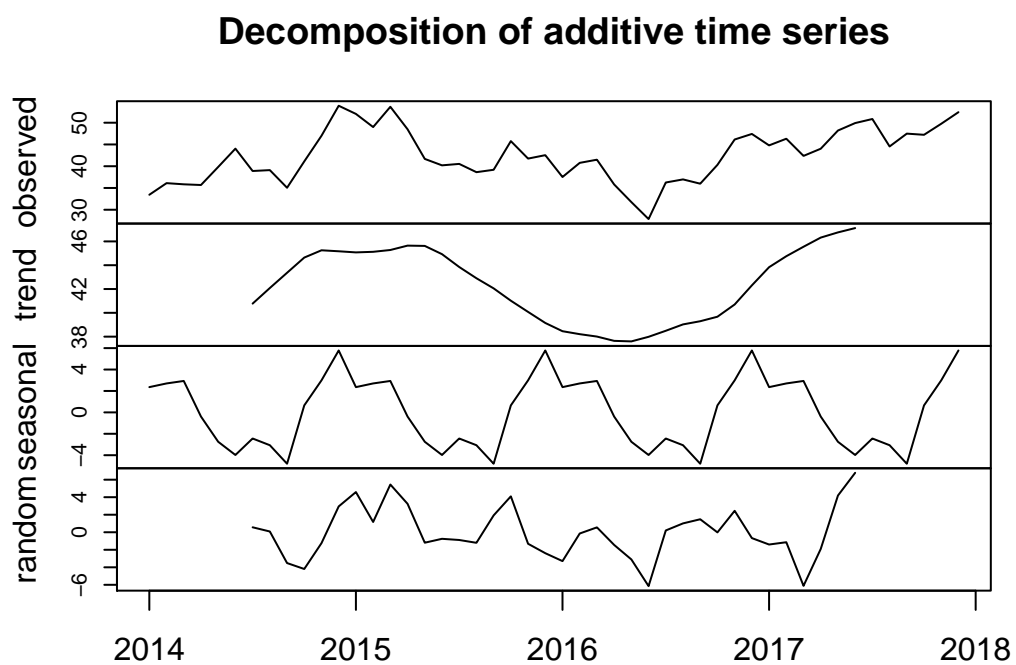
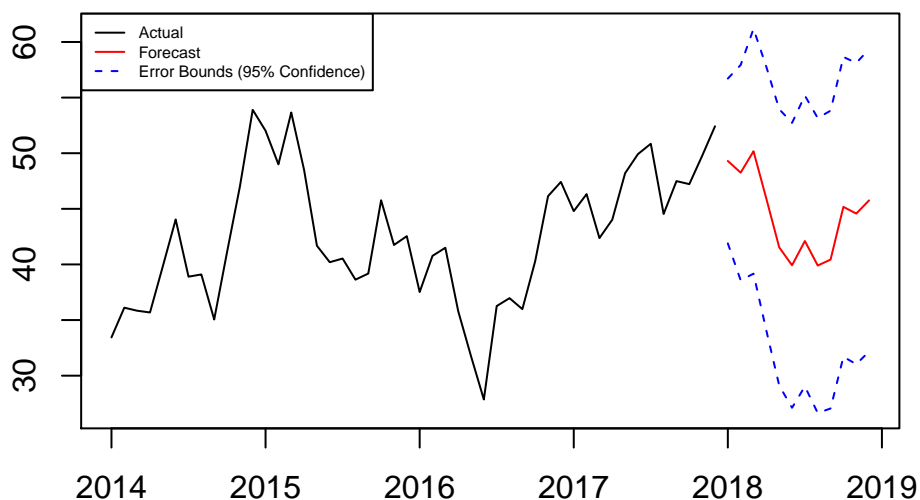


Figure 5: Decomposition of 4 years AAL 'open' series



**Figure 6:** Cyclic Time Series Forecasting with ARIMA model

- Autoregressive moving average (ARMA)
- Autoregressive integrated moving average (ARIMA)

Example on Fig. 6 shows forecasting using ARIMA model.

```
ff <- ts(last4_mo)
fit <- arima(ff, order=c(1,0,0), list(order=c(2,1,0), period=12))
fore <- predict(fit, n.ahead=12)

# error bounds at 95% confidence level
U <- fore$pred + 2*fore$se
L <- fore$pred - 2*fore$se

ts.plot(ff, fore$pred, U, L, col=c(1,2,4,4), lty = c(1,1,2,2),
        xlab="", gpars = list(xaxt="n"))
axis(1, at=seq(from=1, to=72, by=12), labels=c(2014,2015,2016,2017,2018,2019))
legend("topleft", c("Actual", "Forecast", "Error Bounds (95% Confidence)"),
      col=c(1,2,4), lty=c(1,1,2), cex=0.5)
```

### Time series forecasting with forecast package

It is common practice to split the data into two parts, training and test data, where the training data is used to estimate any parameters of a forecasting method and the test data is used to evaluate its accuracy. To perform this task and also to test several additional forecasting methods we used package **forecast** (Hyndman and Khandakar, 2008).

Code below creates train and test data from the the previously obtained 'open' stock data for the last 4 years aggregated monthly. Then it fits the train data using several models and plots the fitting results in the Fig.7. It also plots the test portion in red for easy comparison of prediction and real data.

```
library(forecast)

# test data
test_x <- window(last4_mo, start=as.Date("2016-12-20"))
```

```

test_x <- ts(test_x, frequency = 12)
str(test_x)

#> Time-Series [1:13] from 1 to 2: 47.4 44.8 46.3 42.4 44 ...
#> - attr(*, "index")= Date[1:13], format: "2016-12-30" "2017-01-31" ...

#train data
x <- window(last4_mo, end=as.Date("2016-12-31"))
x <- ts(x, frequency = 12)
str(x)

#> Time-Series [1:36] from 1 to 3.92: 33.4 36.1 35.8 35.7 39.8 ...
#> - attr(*, "index")= Date[1:36], format: "2014-01-31" "2014-02-28" ...

models <- list(
  mod_arima = auto.arima(x, ic='aicc', stepwise=FALSE),
  mod_exp = ets(x, ic='aicc', restrict=FALSE),
  mod_neural = nnetar(x, p=12, size=25),
  mod_t bats = tbats(x, ic='aicc', seasonal.periods=12),
  mod_bats = bats(x, ic='aicc', seasonal.periods=12),
  mod_stl = stlm(x, s.window=12, ic='aicc', robust=TRUE, method='ets'),
  mod_sts = StructTS(x)
)

forecasts <- lapply(models, forecast, 12)
forecasts$naive <- naive(x, 12)
par(mfrow=c(4, 2))
for(f in forecasts){
  plot(f, xlab="", xaxt="n")
  lines(y=test_x[1:13], x=seq(from=(4-1/12), to=5, by=1/12)[1:13], col='red')
  axis(1, at=1:5, labels=c(2014,2015,2016,2017,2018))
}

```

### Evaluating forecast accuracy

Because the test data is not used in determining the forecasts, it should provide a reliable indication of how well the model is likely to forecast on new data. The metrics used below are described in (Hyndman and Athanasopoulos, 2014), also available online [here](#). The metrics are:

- ME - Mean absolute error
- RMSE - Root mean squared error
- MAE - mean absolute error
- MPE - mean percentage error
- MAPE - Mean absolute percentage error
- MASE - mean absolute scaled error
- Theil's U - Uncertainty coefficient

Code below (adaptation of [Timeseries analysis procedure and methods using R](#)) calculates common forecast errors of the predictions made in the previous section.

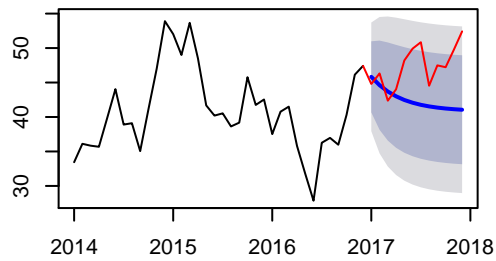
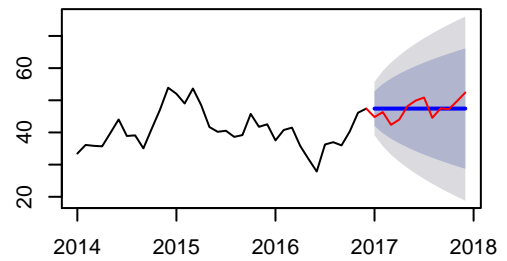
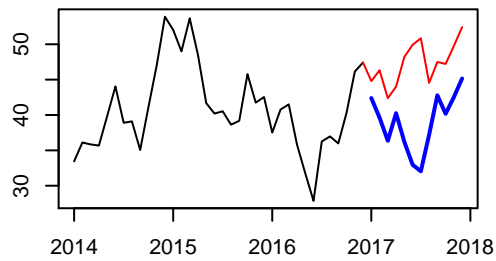
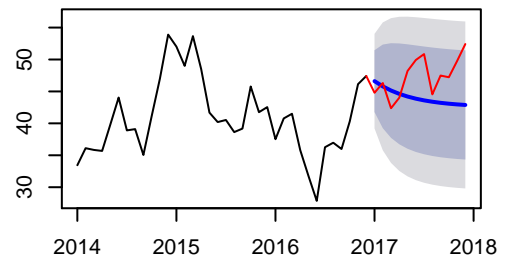
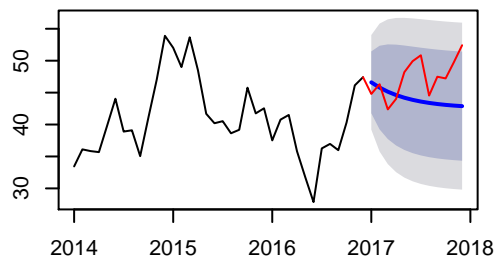
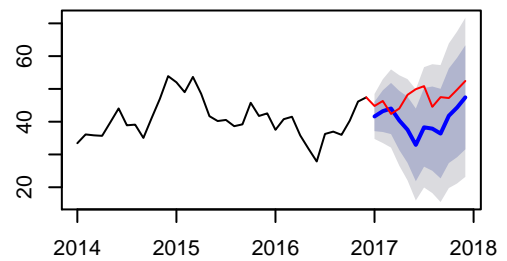
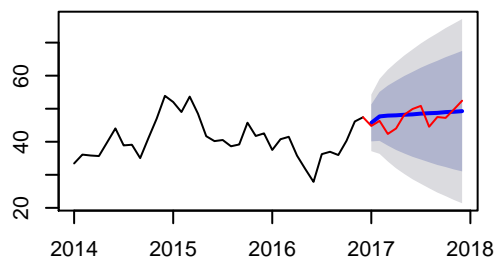
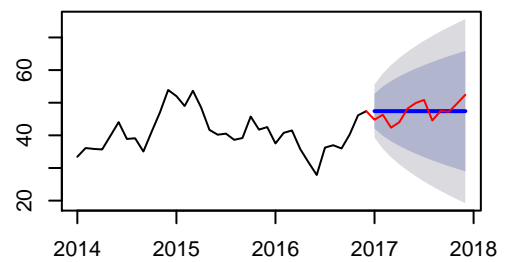
```

test_xx <- ts(test_x[2:13], start=c(4,1), frequency = 12)
acc <- lapply(forecasts, function(f){
  accuracy(f, test_xx)[2,,drop=FALSE]
})
acc <- Reduce(rbind, acc)
row.names(acc) <- names(forecasts)
acc <- acc[order(acc[, 'MASE']),]
round(acc, 2)

#>           ME RMSE  MAE   MPE  MAPE  MASE  ACF1  Theil's U
#> mod_sts    -0.92  2.73  2.23 -2.28  4.83  0.28  0.25    0.93
#> mod_exp    -0.09  2.92  2.45 -0.58  5.21  0.31  0.33    0.96
#> naive      -0.10  2.92  2.45 -0.58  5.21  0.31  0.33    0.96
#> mod_t bats   3.21  4.92  4.08  6.34  8.33  0.52  0.48    1.64
#> mod_bats     3.21  4.92  4.08  6.34  8.33  0.52  0.48    1.64

```



**Forecasts from ARIMA(1,0,0) with non-zero  $m$** **Forecasts from ETS(A,N,N)****Forecasts from NNAR(12,1,25)[12]****Forecasts from BATS(1, {0,0}, 0.8, -)****Forecasts from BATS(1, {0,0}, 0.8, -)****Forecasts from STL + ETS(M,N,N)****Forecasts from Basic structural model****Forecasts from Naive method****Figure 7: AAL stock prediction vs reality using different forecasting methods**

```
#> mod_arma    4.85 6.26 5.24  9.78 10.69 0.67 0.50      2.10
#> mod_stl     6.85 8.41 7.12 14.12 14.77 0.90 0.54      2.85
#> mod_neural  8.35 9.65 8.35 17.34 17.34 1.06 0.46      3.24
```

## Example of Time-Series Analysis Practical Application

### Prepare Shiny App for Deployment

#### Saving recommender data objects

```
#saveRDS(rcmnd_ub, file = "jokeRecommender.Rds")
#saveRDS(jokes, file = "jokes.Rds")
```

### Deployment Discussion

This model is currently not of much use given its accuracy but it will serve as a proof of concept. This model could be used to help writers of movies/tv shows write jokes appropriate for a specific or large audience.

More data should be collected from this userbase to fill a training dataset. The dataset in its current state is quite sparse. The data would need to be updated every 3-5 years as people's taste changes and people within certain age groups mature. The Shiny app developed would be a deployment method to collect more data.

Further analysis could be done (with the appropriate data) to see how similar taste in humor is related to age.

The model developed in this project was used to create Shiny application currently deployed at [ivbsoftware.shinyapps.io/JokeRecommender/](http://ivbsoftware.shinyapps.io/JokeRecommender/). Code of the application could be found in [Github](#).

## Bibliography

- R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, S.L., print edition edition, 2014. ISBN 9780987507105. OCLC: 935493912. [p8]
- R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, 26(3):1–22, 2008. URL <http://www.jstatsoft.org/article/view/v027i03>. [p7]
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL <http://www.R-project.org/>. ISBN 3-900051-07-0. [p1]
- J. A. Ryan and J. M. Ulrich. *xts: eXtensible Time Series*, 2018. URL <https://CRAN.R-project.org/package=xts>. R package version 0.11-2. [p1]
- A. Zeileis and G. Grothendieck. zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, 14(6):1–27, 2005. doi: 10.18637/jss.v014.i06. [p1]

## Note from the Authors

This file was generated using [The R Journal style article template](#), additional information on how to prepare articles for submission is here - [Instructions for Authors](#). The article itself is an executable R Markdown file that could be [downloaded from Github](#) with all the necessary artifacts.

*Igor Baranov*

*York University School of Continuing Studies*

<https://learn.continue.yorku.ca/user/profile.php?id=21219>

*Michael Parravani*

*York University School of Continuing Studies*

*Ariana Biagi*

*York University School of Continuing Studies*

*Hui Fang Cai*

*York University School of Continuing Studies*