Timeseries Forecast Case-Study

Martin Hanewald

1 August 2018

1 Abstract

This analysis shows a simple ARIMA time-series forecast with captured seasonality. An automatic ARIMA fitting is conducted to find the best variant.

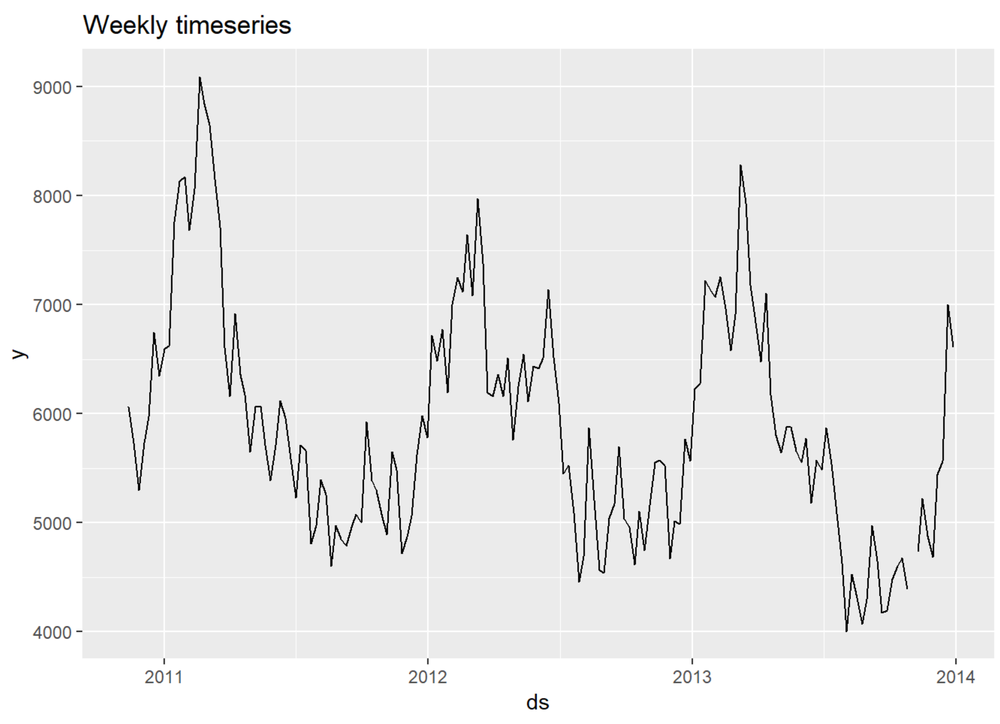
2 Configuration and libraries

```
library(tidyverse)
library(forecast)
library(lubridate)
library(DT)
library(zoo)
library(knitr)
```

3 Loading data

```
dat_ts <- read_csv2('data/timeseries2.csv') %>%
    group_by(time) %>%
    summarise(value = sum(value)) %>%
    rename(ds = time, y = value) %>%
    mutate(week = week(ds))
```

```
dat_ts %>%
    ggplot(aes(ds, y)) + geom_line() + labs(title='Weekly timeseries')
```



4 Preprocessing

4.1 Imputation

As a first step we need to convert the data frame into an R timeseries object with the function *ts()*.

```
timeseries_temp <- ts(dat_ts$y, start = c(2010,45), frequency = 52)
```

```
## Time Series:
## Start = c(2010, 45)
## End = c(2014, 1)
## Frequency = 52
##   [1]   NA 6062 5730 5296 5732 5974 6742 6346 6594 6624 7760 8130 8174 7686 8062 9092 8838 8646 8152 7712 6604
##  [22] 6160 6916 6358 6170 5652 6060 6068 5712 5384 5708 6120 5962 5584 5228 5710 5660 4806 4972 5396 5258 4600
##  [43] 4976 4850 4786 4958 5078 5004 5926 5386 5294 5076 4890 5646 5484 4720 4868 5062 5620 5978 5780 6718 6480
##  [64] 6774 6194 7002 7246 7120 7642 7082 7974 7382 6196 6156 6360 6158 6514 5756 6248 6548 6114 6432 6414 6510
##  [85] 7138 6516 6096 5448 5528 5090 4458 4690 5870 5210 4564 4536 5042 5176 5698 5036 4962 4616 5102 4748 5158
## [106] 5554 5574 5522 4672 5016 4984 5768 5568 6226 6276 7218 7128 7068 7258 6964 6578 6912 8280 7930 7180 6834
## [127] 6474 7106 6172 5802 5640 5878 5876 5654 5552 5772 5182 5574 5486 5866 5536 5082 4626 4004 4528 4310 4070
## [148] 4298 4972 4650 4172 4196 4486 4600 4676 4388   NA 4730 5222 4878 4682 5442 5566 7002 6608
```

Now it comes to our attentation, that there is a missing value (**NA**). This needs to be filled in order for any timeseries model to be fitted, since they cannot handle missing values.

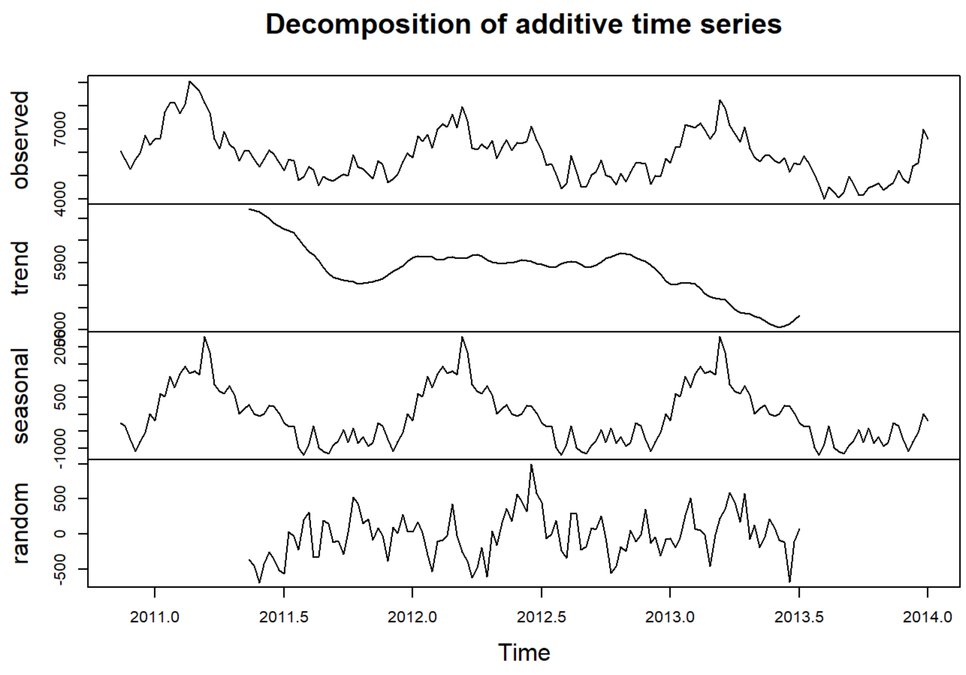
We use the automatic imputation function *na.approx()*, which just linearly interpolates between adjacent values.

```
timeseries <- timeseries_temp %>% na.approx()
timeseries
```

```
## Time Series:
## Start = c(2010, 46)
## End = c(2014, 1)
## Frequency = 52
##   [1] 6062 5730 5296 5732 5974 6742 6346 6594 6624 7760 8130 8174 7686 8062 9092 8838 8646 8152 7712 6604 6160
##  [22] 6916 6358 6170 5652 6060 6068 5712 5384 5708 6120 5962 5584 5228 5710 5660 4806 4972 5396 5258 4600 4976
##  [43] 4850 4786 4958 5078 5004 5926 5386 5294 5076 4890 5646 5484 4720 4868 5062 5620 5978 5780 6718 6480 6774
##  [64] 6194 7002 7246 7120 7642 7082 7974 7382 6196 6156 6360 6158 6514 5756 6248 6548 6114 6432 6414 6510 7138
##  [85] 6516 6096 5448 5528 5090 4458 4690 5870 5210 4564 4536 5042 5176 5698 5036 4962 4616 5102 4748 5158 5554
## [106] 5574 5522 4672 5016 4984 5768 5568 6226 6276 7218 7128 7068 7258 6964 6578 6912 8280 7930 7180 6834 6474
## [127] 7106 6172 5802 5640 5878 5876 5654 5552 5772 5182 5574 5486 5866 5536 5082 4626 4004 4528 4310 4070 4298
## [148] 4972 4650 4172 4196 4486 4600 4676 4388 4559 4730 5222 4878 4682 5442 5566 7002 6608
```

To get a first feel for the trend and seasonality components of the time series we can apply a quick decomposition with the function *decompose()*. This fits an exponential model to the timeseries and the components can be plotted.

```
timeseries %>% decompose() %>% plot()
```



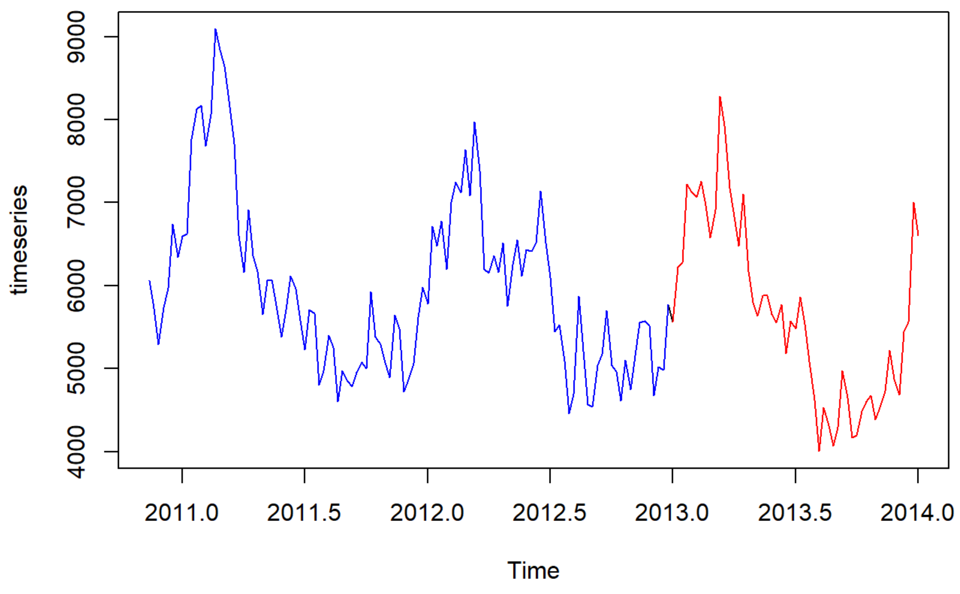
4.2 Splitting

When splitting a timeseries it is important that we cannot apply random sampling, since we need to have an unbroken series for timeseries models to be applied.

Therefore we choose the last year in the data set as a test set and all prior data as training set.

```
training <- timeseries %>% window(end=c(2012,52))
testing <- timeseries %>% window(start=c(2013,1))
```

```
plot(timeseries)
lines(training, col='blue')
lines(testing, col='red')
```



```
training
```

```
## Time Series:
## Start = c(2010, 46)
## End = c(2012, 52)
## Frequency = 52
##   [1] 6062 5730 5296 5732 5974 6742 6346 6594 6624 7760 8130 8174 7686 8062 9092 8838 8646 8152 7712 6604 6160
##  [22] 6916 6358 6170 5652 6060 6068 5712 5384 5708 6120 5962 5584 5228 5710 5660 4806 4972 5396 5258 4600 4976
##  [43] 4850 4786 4958 5078 5004 5926 5386 5294 5076 4890 5646 5484 4720 4868 5062 5620 5978 5780 6718 6480 6774
##  [64] 6194 7002 7246 7120 7642 7082 7974 7382 6196 6156 6360 6158 6514 5756 6248 6548 6114 6432 6414 6510 7138
##  [85] 6516 6096 5448 5528 5090 4458 4690 5870 5210 4564 4536 5042 5176 5698 5036 4962 4616 5102 4748 5158 5554
## [106] 5574 5522 4672 5016 4984 5768
```

```
testing
```

```
## Time Series:
## Start = c(2013, 1)
## End = c(2014, 1)
## Frequency = 52
##  [1] 5568 6226 6276 7218 7128 7068 7258 6964 6578 6912 8280 7930 7180 6834 6474 7106 6172 5802 5640 5878 5876
## [22] 5654 5552 5772 5182 5574 5486 5866 5536 5082 4626 4004 4528 4310 4070 4298 4972 4650 4172 4196 4486 4600
## [43] 4676 4388 4559 4730 5222 4878 4682 5442 5566 7002 6608
```

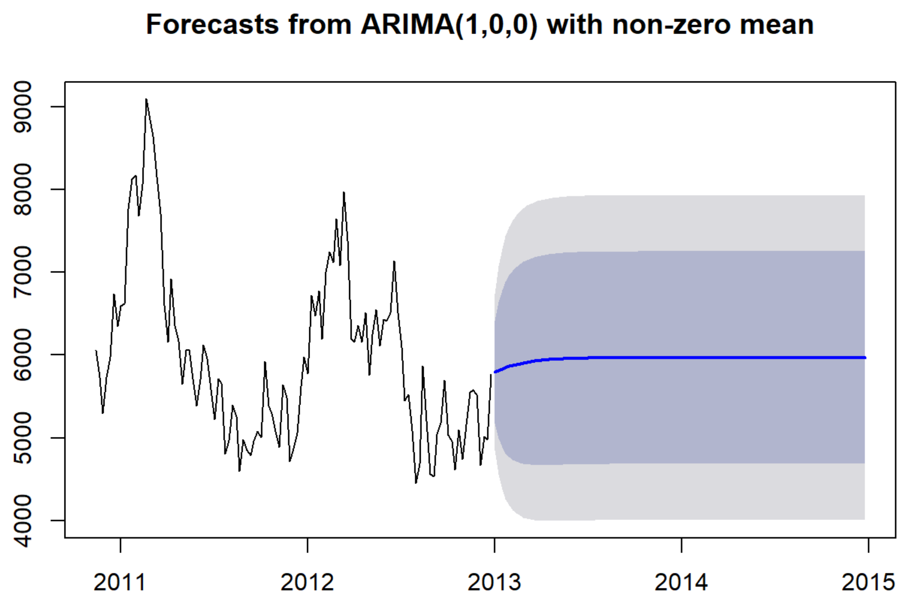
5 Model training

Now we can fit a timeseries model to the training set.

5.1 ARIMA (1,0,0)

First we just try out a random ARIMA configuration (a simple AR1 model).

```
arima(training, c(1,0,0)) %>% forecast() %>% plot()
```



The results are not very promising, since the AR1 model is not able to capture any seasonality and basically just extends the last known value.

5.2 Automatic ARIMA fitting

The function *auto.arima()* automatically fits the best ARIMA model. As a metric for model comparison the **Akaike Information Criterion (AIC)** is used.

```
fit <- auto.arima(training)
```

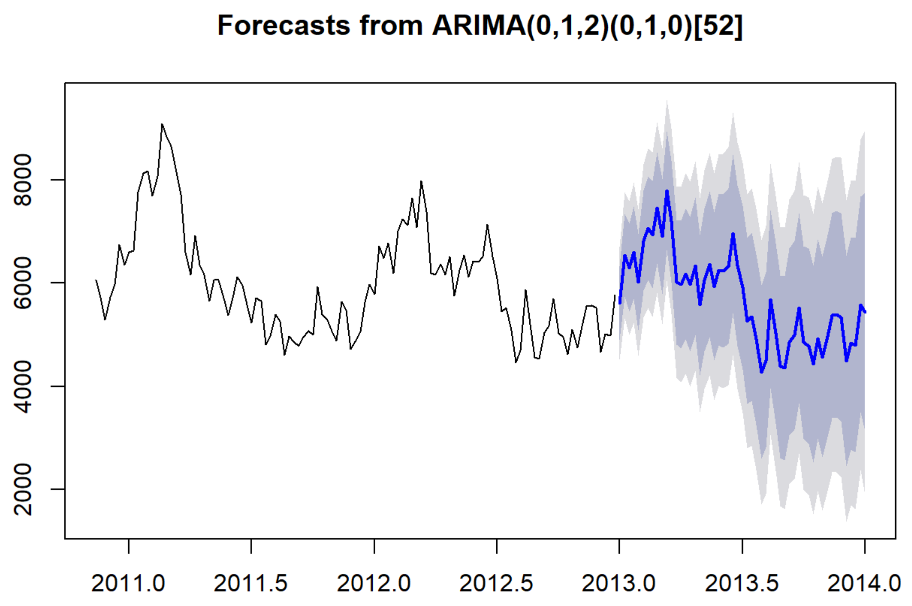
```
## Warning in value[[3L]](cond): The chosen test encountered an error, so no seasonal differencing is selected.
## Check the time series data.
## Series: training
## ARIMA(0,1,2)(0,1,0)[52]
##
## Coefficients:
##           ma1      ma2
##       -0.4909  -0.1260
## s.e.   0.1288   0.1272
##
## sigma^2 estimated as 309782:  log likelihood=-448.15
## AIC=902.3   AICc=902.74   BIC=908.48
```

6 Model evaluation

6.1 Conducting the forecast

In order to evaluate the model, we rely heavily on visual inspection. First the forecast covering the testset period is computed.

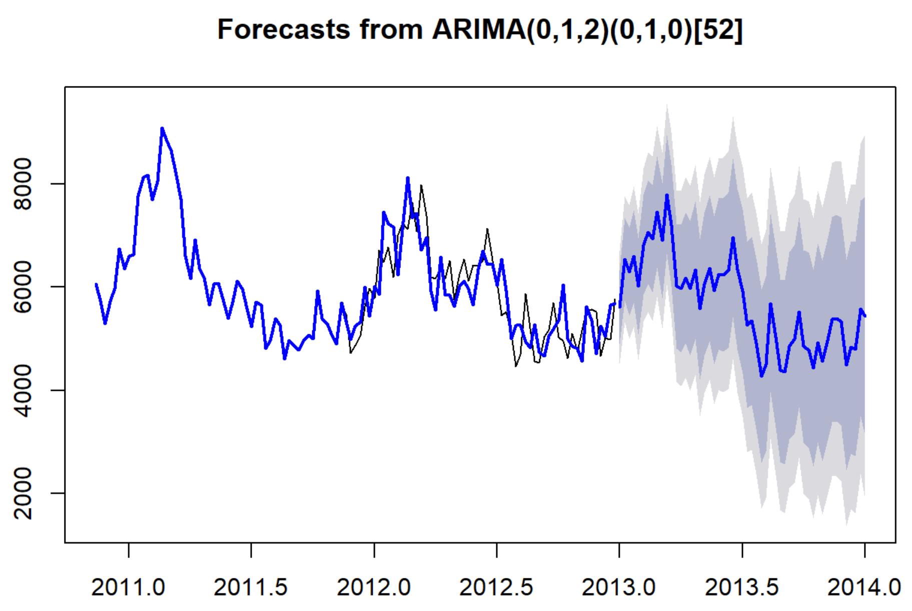
```
fc <- forecast(fit, h = length(testing))
plot(fc)
```



The forecast not only computes the most likely evolution of the timeseries (Point Forecast), but also the 80 and 95 % confidence intervals. It is important to note, that the confidence interval usually gets wider, the further the forecast runs into the future since the stochastic behaviour is compounded every timestep.

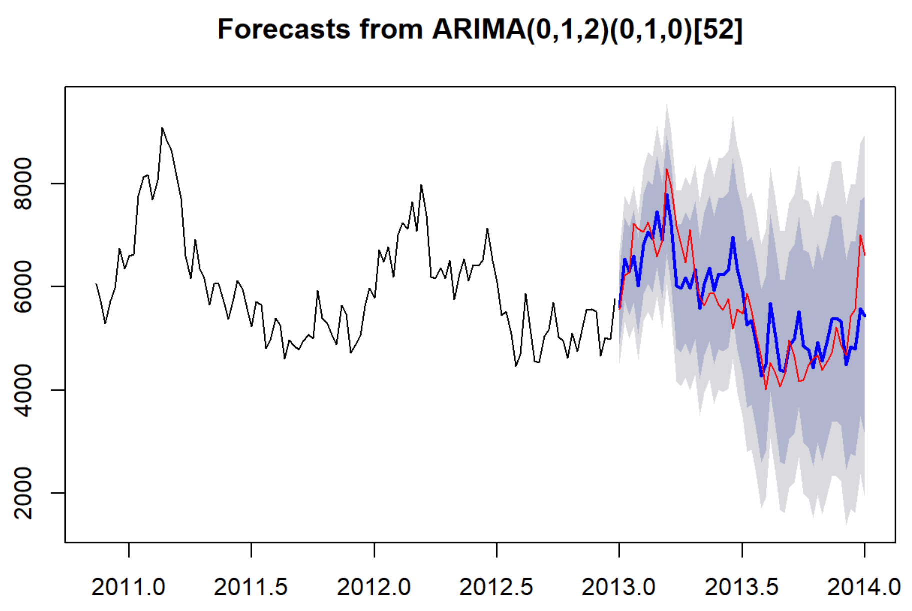
As a first visual check, we can check the fit of the model to the training period.

```
plot(fc)
lines(fit$fitted, col='blue', lwd=2)
```



Next we compare the fit on the test period.

```
plot(fc)
lines(testing, col='red')
```



The performance is also measured by performance metrics on both training and testing period.

```
accuracy(fc, testing) %>% kable()
```

	ME	RMSE	MAE	MPE	MAPE	MASE	ACF1	Theil's U
Training set	5.298197	395.3310	234.8047	-0.2100068	4.054983	0.4057844	-0.0071824	NA
Test set	-35.516712	657.9317	520.0071	-1.9191244	9.331962	0.8986648	0.6014512	1.394399

Especially the MAPE (Mean Absolute Percentage Error) has a very popular interpretation, since describes the mean deviation in percent.

In this instance we a remaining mean deviation of **9.3 %** on the testing period.

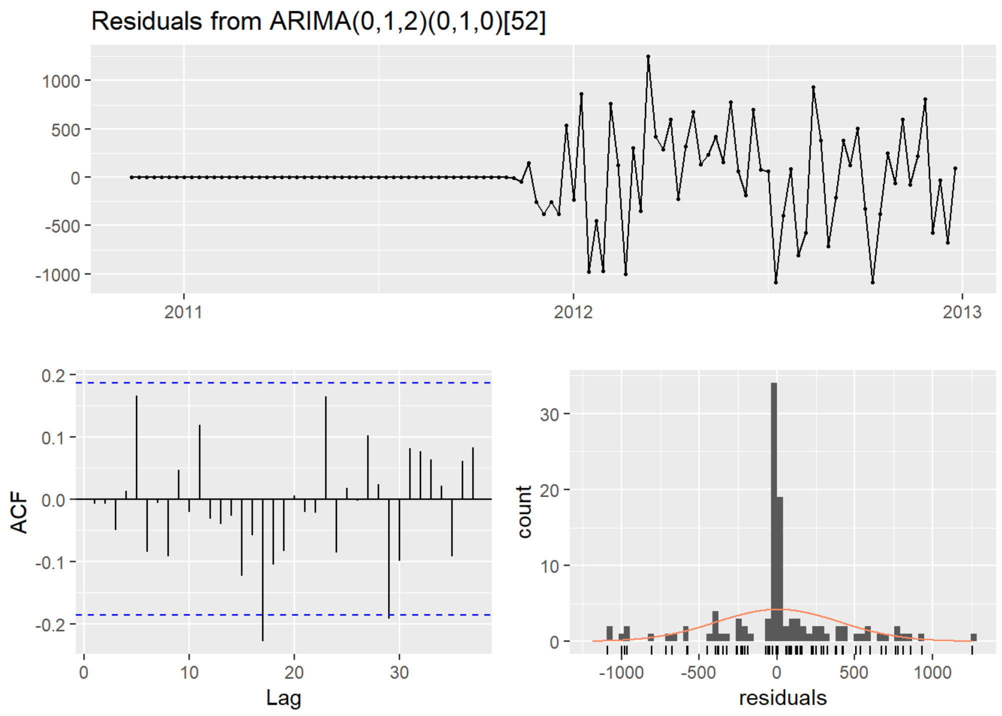
6.2 Residual checking

A common sanity check for the conducted model is to inspect the residuals, meaning the difference between forecast and actuals.

They should be

1. Stationary, i.e. not show time dependend behaviour or autocorrelation
2. Close to normally distributed

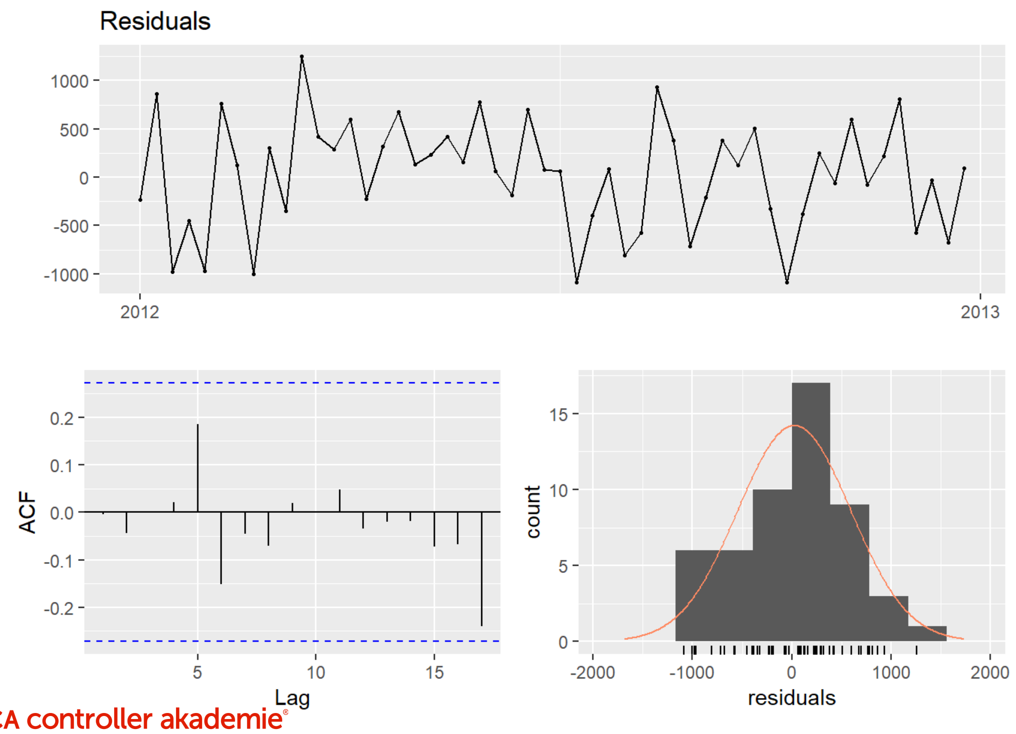
```
checkresiduals(fit)
```



```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,2)(0,1,0)[52]
## Q* = 19.957, df = 20.2, p-value = 0.4734
##
## Model df: 2.   Total lags used: 22.2
```

Since we fitted a seasonal model it is common practive to only review the residuals after the first full period (year), since we have a 100% fit on the first period.

```
checkresiduals(fit$residuals %>% window(start=c(2012,1)))
```



We can see that the residuals look almost flawless.