Automated Model Selection for GDP Growth Forecasting

Introduction

What follows is a report of a project I embarked on to establish a more solid foundation on how to run a forecasting analysis in R. The end result of the project was simple to envision though, as it usually is the case, its implementation turned up a fair number of twists and turns.

The goals for this project were as follows:

- Given a set of quarterly time series data of a country's GDP, automatically identify the model with the best forecasting performance
- Use said model to forecast GDP values for the following 4 quarters, i.e., for the following year.

The data used was retrieved from the *GDP and main components (output, expenditure, and income)* database from Eurostat's website. It includes quarterly GDP data from 2005Q1 to 2016Q from 30 European countries.

Data Wrangling

```
#ORIGINAL DATA
countrydata <- read_csv("~/Dropbox/Data/GDP Forecasting/GDP Forecasting/namq_10_gdp_1_Data.csv")
```

As it stands, *countrydata* is a mess.

```
head(countrydata)
```

```
## # A tibble: 6 x 7
##   TIME   GEO        UNIT      S_ADJ     NA_ITEM     Value `Flag and Footn~
##   <chr>  <chr>      <chr>     <chr>     <chr>       <dbl> <chr>
## 1 2005Q1 Belgium    Chain li~ Seasonal~ Gross dom~ 8.45e4 <NA>
## 2 2005Q1 Bulgaria   Chain li~ Seasonal~ Gross dom~ 7.88e3 <NA>
## 3 2005Q1 Czech Rep~ Chain li~ Seasonal~ Gross dom~ 3.39e4 <NA>
## 4 2005Q1 Denmark    Chain li~ Seasonal~ Gross dom~ 5.92e4 <NA>
## 5 2005Q1 Germany (~ Chain li~ Seasonal~ Gross dom~ 6.00e5 <NA>
## 6 2005Q1 Estonia    Chain li~ Seasonal~ Gross dom~ 3.60e3 <NA>
```

Because the analysis we'll be doing is on a per country basis, it would be helpful if we could group all data relating to each country to its own particular data table which we would then feed to any function we might want to use that data with. Thankfully, that's not hard to do in R:

```
by_country  <- countrydata %>%
  select(GEO,TIME,Value) %>%
  group_by(GEO)%>%
  nest()
```

Now we have a single tibble composed of multiple tibbles that contain each country's data:

```
head(by_country)
```

```
## # A tibble: 6 x 2
##   GEO            data
##   <chr>          <list>
```

```
## 1 Belgium        <tibble [48 x 2]>
## 2 Bulgaria       <tibble [48 x 2]>
## 3 Czech Republic <tibble [48 x 2]>
## 4 Denmark        <tibble [48 x 2]>
## 5 Germany        <tibble [48 x 2]>
## 6 Estonia        <tibble [48 x 2]>
```

Each country's data can be called by simply doing:

```
by_country$data[1]
```

```
## [[1]]
## # A tibble: 48 x 2
##    TIME    Value
##    <chr>   <dbl>
##  1 2005Q1 84546.
##  2 2005Q2 84967.
##  3 2005Q3 85279.
##  4 2005Q4 85968.
##  5 2006Q1 86522.
##  6 2006Q2 86802.
##  7 2006Q3 87507.
##  8 2006Q4 88448.
##  9 2007Q1 89583.
## 10 2007Q2 89922.
## # ... with 38 more rows
```

The methods we will be using expect a time series object, *ts*.

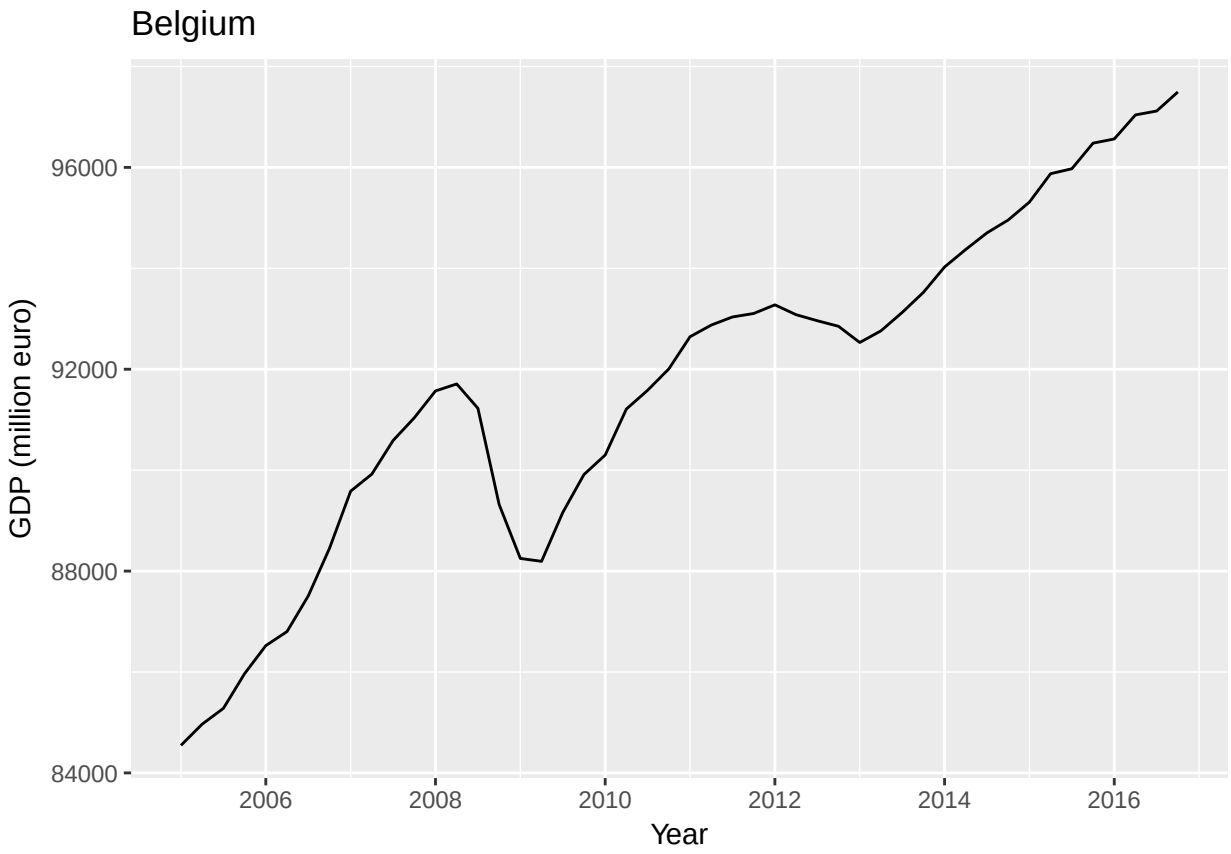
```
create_ts <- function(df){
  #creates ts object with proper start date based on length of list
  #ATTENTION THIS IS ONLY FOR THE INITIAL REMOVAL OF VALUES FROM THE CSV
  ts(as.data.frame(df$Value), start=c(2005,1),frequency=4)
}

create_ts_i <- function(df){
  #creates ts object with proper start date based on length of list
  df <- unlist(df)
  year <- 2016 - as.integer(length(df)/4)
  if((round(length(df)/4) - length(df)/4) == 0.25){
    quarter <- 2
  }
  else if((round(length(df)/4) - length(df)/4) == 0.5){
    quarter <- 3
  }
  else if((round(length(df)/4) - length(df)/4) == 0){
    quarter <- 1
  }
  else{quarter <- 4}
  return(ts(as.data.frame(df), start=c(year,quarter),frequency=4))
}
```

```
ts_country <- lapply(by_country$data,create_ts) #turn values into a time series
```

Let's plot one example just to see if it worked.

```r
autoplot(ts_country[[1]],ylab="GDP (million euro)",xlab="Year", main=by_country$GEO[1])
```



It is common to apply some sort of mathematical transformation to the data so that we are better able to analyze it. The log-transform is a common example of one. Another example is the Box-Cox transformation, commonly used for time series data and described in Rob Hyndman's Forecasting book, where the transformation depends on a variable λ as follows:

$$w_t = \begin{cases} \ln(y_t) & \text{if } \lambda = 0 \\ \frac{y_t^{\lambda}-1}{\lambda} & \text{otherwise} \end{cases}$$

The values for λ are chosen so that the seasonal variance of the time series is approximately constant across the whole series. Though we can fiddle around manually to find a good value for λ, the *forecast* package includes a way to determine λ automatically.

```r
boxcox_create_ts <- function(df){
  #box cox transformation
  BoxCox(df,BoxCox.lambda(df))
}

boxcox_lambda <- function(df){
  #retrieve lambda values
  BoxCox.lambda(df)
}
```

```r
inv_boxcox_ts <- function(x, lambda)
    #retrieve the original time series data by inversing the transformation
    if (lambda == 0) exp(x) else (lambda*x + 1)^(1/lambda)
```

Now we have a way to transform the data and retrieve it back untransformed.

```
#apply a box-cox transformation to all those series
boxcox_ts <- lapply(ts_country,boxcox_create_ts)

#so that latter can perform inverse transformation of the forecasted values
lambdas <- lapply(ts_country,boxcox_lambda)
```

Recall that our goal is to forecast growth values. We calculate each quarter's growth rate by doing

$$g = \frac{t_2 - t_1}{t_1} * 100$$

where t_i means GDP at time i and g is the growth rate in percentage points.

```
percentagechange <-function(timeseries){
  #growth rate
  (diff(timeseries,1)/timeseries[-c(1)])*100
}
```

```
#calculate a growth rate for that transformed series
growth_ts <- lapply(boxcox_ts,percentagechange)
corrected_ts <- growth_ts
```

Data Exploration

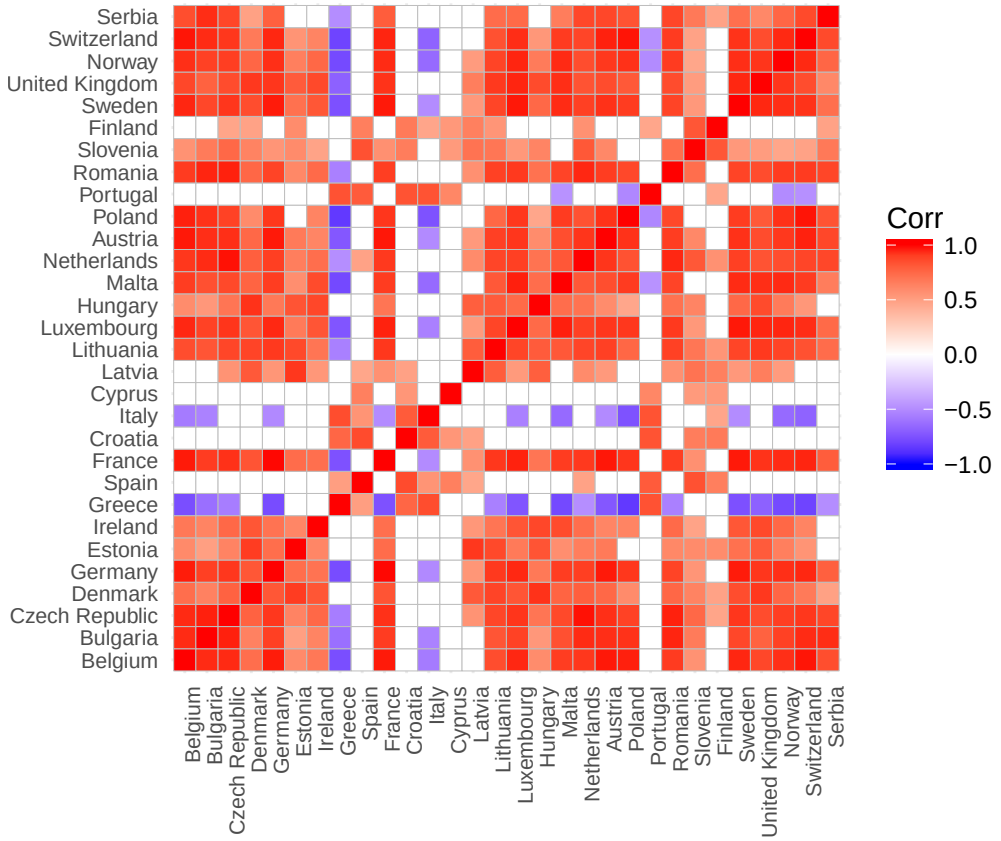
How does a country's GDP behave given the behavior of another country's GDP?

Since we already have our data loaded up and ready to go, we can answer that question with a correlation matrix of each country's GDP.

```
cor_table <- data.frame()
sigcor_table <- data.frame()
for(i in 1:30){
  i_cor <- data.frame()
  i_sigcor <- data.frame()
  for(p in 1:30){
    i_cor <- rbind(i_cor,cor(ts_country[[i]],ts_country[[p]]))
    i_sigcor <- rbind(i_sigcor,cor.test(ts_country[[i]],ts_country[[p]])$p.value)
  }
  cor_table <- rbind(cor_table,t(i_cor))
  sigcor_table <- rbind(sigcor_table,t(i_sigcor))
}
colnames(cor_table)<- by_country$GEO
rownames(cor_table)<- by_country$GEO

colnames(sigcor_table)<- by_country$GEO
rownames(sigcor_table)<- by_country$GEO
```

```
ggcorrplot(cor_table, method = "square", p.mat = sigcor_table, sig.level = 0.001,insig = "blank")+
  theme(axis.text.x = element_text(angle=90, hjust=1, size=8), axis.text.y = element_text(hjust=1, size=
```



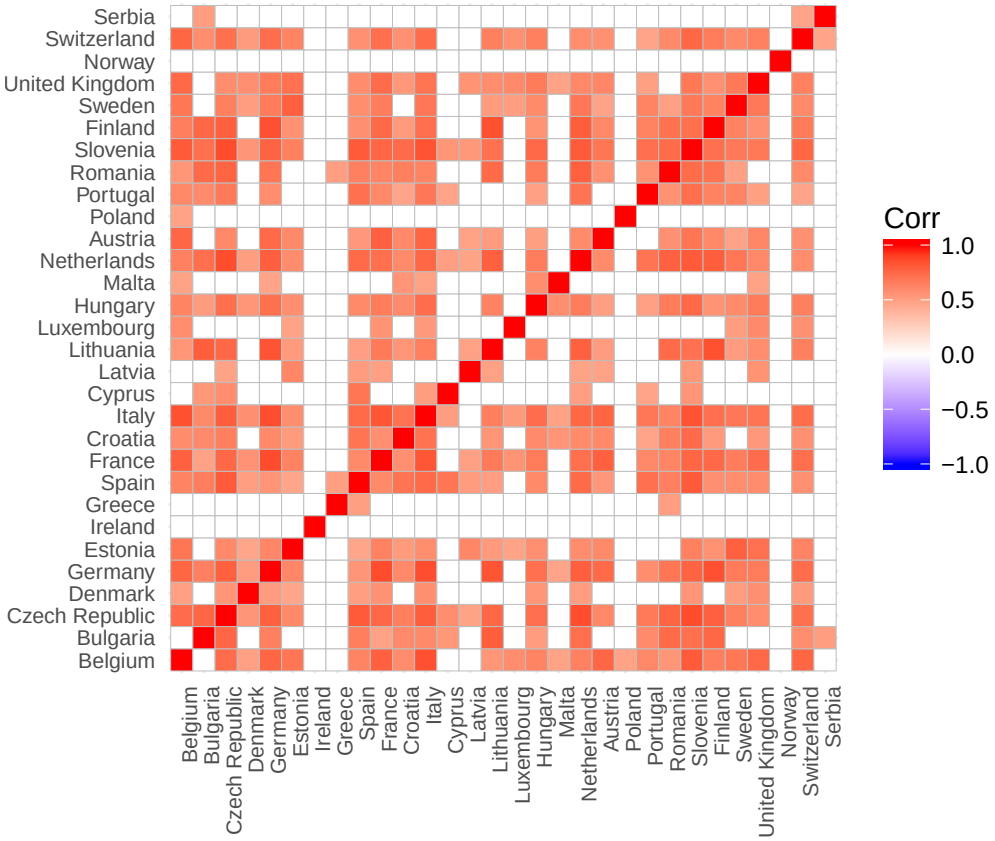

```
ggsave("GDP - Correlation Matrix.png", width = 30, height = 30, units = "cm")
```

We can do the same for growth values:

```
cor_table <- data.frame()
sigcor_table <- data.frame()
for(i in 1:30){
  i_cor <- data.frame()
  i_sigcor <- data.frame()
  for(p in 1:30){
    i_cor <- rbind(i_cor,cor(growth_ts[[i]],growth_ts[[p]]))
    i_sigcor <- rbind(i_sigcor,cor.test(growth_ts[[i]],growth_ts[[p]])$p.value)
  }
  cor_table <- rbind(cor_table,t(i_cor))
  sigcor_table <- rbind(sigcor_table,t(i_sigcor))
}
colnames(cor_table)<- by_country$GEO
rownames(cor_table)<- by_country$GEO

colnames(sigcor_table)<- by_country$GEO
rownames(sigcor_table)<- by_country$GEO
```

```
ggcorrplot(cor_table, method = "square", p.mat = sigcor_table, sig.level = 0.001,insig = "blank")+
  theme(axis.text.x = element_text(angle=90, hjust=1, size=8), axis.text.y = element_text(hjust=1, size=
```




```
ggsave("GDP Growth - Correlation Matrix.png", width = 30, height = 30, units = "cm")
```

```
#corrected_ts <- lapply(growth_ts,ts_breakpoints)
```

Forecasting Analysis

We are now in a good position start on our forecasting goal.

Usually I would use both exponential smoothing methods along with ARIMA models and from there select the one model with the best forecasting performance. However, I have previously done a similar analysis using only Greece's GDP data and I already know that ARIMA models have much better forecasting accuracy than exponential smoothing. We're going to cheat a bit here and skip having to code them, though If you have never seen or used exponential smoothing as a forecasting tool make sure to learn about it in Rob Hyndman's book. They're cheap and can be quite accurate, always a nice tool to have in your toolkit.

What is an ARIMA model?

If you've done time series analysis before you're likely familiar with autoregressive and moving average models.

In an autoregressive model, or AR model, the value of a time series at t_i depends linearly on its past values plus a stochastic term. If we say we are dealing with an AR(1) model what we are saying is that the time series at time t_i depends linearly on only the previous value, t_{i-1}, while if we are dealing with an AR(2) model each time series value depends not only on its previous value, but the one before that as well, i.e. t_{i-1} and t_{i-2}.

Generalizing, we cay think of an AR(p) model as defined by:

$$r_t = \beta_0 + \sum_{i=1}^{p} \beta_i r_{t-i} + \epsilon_t$$

In a moving average model, or MA model, we envision the time series as being dependent on a mean plus the linear effect of current and past values of a stochastic term, or shock.

$$r_t = \mu + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$$

Putting these two together gets us what is called an autoregressive moving average model, or ARMA model, where we envision the time series as being modelled by the sum of both an $AR(p)$ and $MA(q)$ model:

$$r_t = c + \sum_{i=1}^{p} \beta_i r_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-1} + \epsilon_t$$

We can further generalize the ARMA model by making it so that the data we're using has been differenced. That is, instead of using the original data series, we use $\Delta Y = Y_t - Y_{t-1}$ as the new time series we are to find a model for. Differencing is a useful tool to stationarize a time series, and the *forecast* package has multiple ways to determine how many times you need to difference a time series to get it to be stationary.

ARIMA models are characterized by three parameters, (p, d, q), where p is the order of the autoregressive model, d the number of times the data has been differenced, and p is the order of the moving average model. You can also include a seasonal component to an ARIMA model, which we denote as $ARIMA\,(p, d, q)\,(P, D, Q)_m$ where m is the number of periods in each season.

Procedure

We are trying to identify the best model out from the set of all models that can be characterized the 6 parameter values of a seasonal ARIMA, meaning that we have $p \times d \times q \times P \times D \times Q$ models out from which we want to select one.

How are we going to be making that decision?

The *forecast* package has an immensively useful function called *tsCV()* that allows us to compute the forecast errors of a model using cross validation. What it does is it uses a portion of the data, say from 2005Q1 to 2006Q1, to then fit a model frrom which it will produce a forecast for h periods ahead. With that forecast at hand, it then computes the error between it and the true value for that period. Having done that, it then expands the portion of the data that it used before and repeats the whole process until the data it is using is the whole time series data it started with.

The idea is then to compute the cross-validation errors of $p \times d \times q \times P \times D \times Q$ models and then select the one which has the lowest errors. Each country gets its own forecasting model, and the model with the lowest RMSE of whole set is the one we select for our subsequent forecasts.

```
model_selection <- function(df,nonseasonal,seasonal){
  arima.cverrors <- data.frame()
  for(p in 0:nonseasonal){
    for(d in 0:nonseasonal){
      for(q in 0:nonseasonal){
        for(P in 0:seasonal){
          for(D in 0:seasonal){
            for(Q in 0:seasonal){
              cvarima <- function(x,h){forecast(arima(x, order=c(p,d,q), seasonal=c(P,D,Q)),h=h)}
```

```
              cverror <- tsCV(create_ts_i(unlist(df)),cvarima,h=4)
              arima.cverrors <- rbind(arima.cverrors,c(sqrt(mean(cverror^2,na.rm=TRUE)),mean(abs(cverro
            }
          }
        }
      }
    }
  }
  colnames(arima.cverrors) <- c("RMSE","MAE","p","d","q","P","D","Q")
  arima.cverrors[which.min(arima.cverrors$RMSE),]
}
```

This is the most time consuming step of the whole project. Notice that since we're dealing with 6 parameter values and going through each possible combination of them, this means that we need to check $p \times d \times q \times P \times D \times Q$ models for 30 distinct time series. Any increase in the range of parameter values is a nonlinear increase in running time, and that means we need to set an interval for those values such that it we can get a reasonable level of accuracy in a reasonable amount of time. The parameter value range used here comes from me having had to run this multiple times and thus knowing more or less where that local optimum is.

An upgrade that I want to do in the future is to parallelize the procedure such that each physical CPU core is selecting the best model for a country. That should reduce the running time considerably.

```
models <- lapply(corrected_ts,model_selection,nonseasonal = 1,seasonal = 2)
```

We need to have a way to use the growth rates we've just forecasted to retrieve the forecasted GDP values.

```
forecast_values <- function(ts,df){
  lastvalue <- ts[length(ts)]
  q1 <- lastvalue*(1+df[1]/100)
  q2 <- q1*(1+df[2]/100)
  q3 <- q2*(1+df[3]/100)
  q4 <- q3*(1+df[4]/100)
  c(q1,q2,q3,q4)
}
```

We then need to transform those forecasts back to the data's original state, that is, we need not only to get the actual values from the forecasted growth rates but we also need to remove the Box-Cox transformation that we applied before.

```
forecast_2017_values <- data.frame()
forecast_2017_growth <- data.frame()
for(i in 1:30){

  timeseries <- create_ts_i(corrected_ts[i]) #growth time series data of country i
  model <- unlist(models[i]) #model that we determined provides the best forecast performance

  forecast <- forecast(arima(timeseries,
                  order=c(model[3],model[4],model[5]),
                  seasonal = c(model[6],model[7],model[8]),
                  method="ML"),
                  h=4)

  forecastvalues <- forecast_values(unlist(boxcox_ts[i]),forecast$mean) #Calculates the (transformed) f
  prediction <- ts(
                inv_boxcox_ts(forecastvalues,unlist(lambdas[i])),
                start=c(2017,1),frequency = 4) #Removes the transformation for the forecasted GDP value
```

```
  forecast_growth <- percentagechange(c(as.data.frame(by_country$data[i])$Value,prediction))

  forecast_2017_values <- rbind(forecast_2017_values, prediction) #Forecasted GDP values
  forecast_2017_growth <- rbind(forecast_2017_growth,forecast_growth[-c(1:47)])#Forecasted growth rates
}


colnames(forecast_2017_values) <- (c("Q1","Q2","Q3","Q4"))
colnames(forecast_2017_growth) <- (c("Q1","Q2","Q3","Q4"))
```

Now that we have our forecasts for each country's GDP in 2017, it makes sense, given we're already in 2018, to compare them with what actually ended up happening.

The 2017 data is from the same Eurostat database, using the same units.

```
GDP_2017<- read_csv("~/Dropbox/Data/GDP Forecasting/GDP Forecasting/2017_GDP_Data.csv")
by_country_2017  <- GDP_2017 %>%
  select(GEO,TIME,Value) %>%
  group_by(GEO)%>%
  nest()
```

```
values_2017_GDP <- data.frame()
growth_2017_GDP <- data.frame()
for(i in 1:30){
  values_2017_GDP <- rbind(values_2017_GDP, ts(as.data.frame(by_country_2017$data[i])$Value, start = c(
  growth_2017_GDP <- rbind(growth_2017_GDP, percentagechange(c(as.data.frame(by_country$data[i])$Value,
}
colnames(values_2017_GDP) <- (c("Q1","Q2","Q3","Q4"))
colnames(growth_2017_GDP) <- (c("Q1","Q2","Q3","Q4"))
```

The error measures we'll be using in our comparisons:

```
MAE <- function(forecast,reality){
  mean(unlist(abs(reality - forecast)))
}

RMSE <- function(forecast,reality){
  sqrt(mean(unlist((reality-forecast)^2)))
}

MAPE <- function(forecast,reality){
  mean(unlist(100*(reality-forecast)/reality))
}

sMAPE <- function(forecast,reality){
  mean(unlist(200*(abs(reality-forecast)/(forecast+reality))))
}
```

```
errors <- data.frame()
for(i in 1:30){
  errors <- rbind(errors,c(MAE(forecast_2017_values[i,],values_2017_GDP[i,]),
                           RMSE(forecast_2017_values[i,],values_2017_GDP[i,]),
                           MAPE(forecast_2017_values[i,],values_2017_GDP[i,]),
                           sMAPE(forecast_2017_values[i,],values_2017_GDP[i,])
                           ))
}
```

```
colnames(errors) <- (c("MAE","RMSE","MAPE","sMAPE"))
rownames(errors) <- by_country$GEO
```

```
for(i in 1:30){
  autoplot(ts(t(growth_2017_GDP[i,])),ylab="Growth Rate",main = by_country$GEO[i],series = "Real")+
    autolayer(ts(t(forecast_2017_growth[i,])),series = "Forecast")
#  ggsave(paste("autoplot_growthrate_",countrynames[i],".png",sep = ""))
}
```

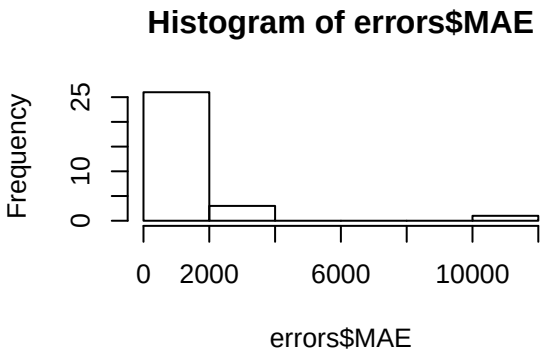
How did we do?

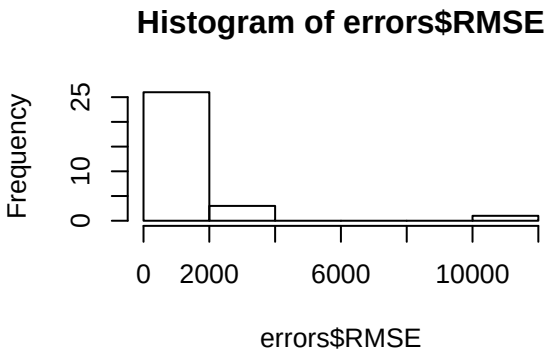
```
mean_errors <- as.data.frame(t(c(mean(errors$MAE),mean(errors$RMSE),mean(errors$MAPE),mean(errors$sMAPE)
colnames(mean_errors) <-c("MAE","RMSE","MAPE","sMAPE")
kable(mean_errors, caption = "Mean of error measures",digits = 2)
```

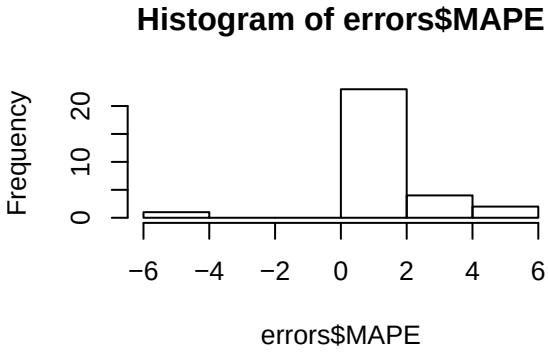
Table 1: Mean of error measures

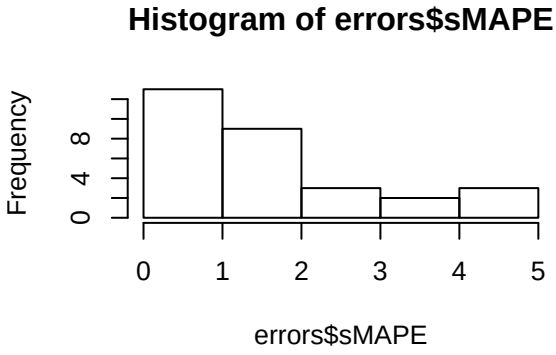
MAE	RMSE	MAPE	sMAPE
1265.35	1340.85	1.26	1.61

```
par(mfrow=c(2,2))
hist(errors$MAE)
hist(errors$RMSE)
hist(errors$MAPE)
hist(errors$sMAPE)
```









Which countries did we do the worse with?

```
higher_errors <- data.frame(by_country$GEO[order(errors$MAE)],
                            by_country$GEO[order(errors$RMSE)],
                            by_country$GEO[order(abs(errors$MAPE))],
                            by_country$GEO[order(errors$sMAPE)])

colnames(higher_errors) <- c("MAE","RMSE","MAPE","sMAPE")
kable(head(higher_errors))
```

MAE	RMSE	MAPE	sMAPE
Estonia	Estonia	Poland	Poland
Serbia	Serbia	Estonia	Italy
Lithuania	Lithuania	Serbia	Sweden
Cyprus	Cyprus	Italy	France
Croatia	Malta	Sweden	Estonia
Malta	Croatia	France	Belgium

Inversely, which countries did we do the best with?

```
lower_errors <- data.frame(by_country$GEO[order(errors$MAE,decreasing = TRUE)],
                            by_country$GEO[order(errors$RMSE,decreasing = TRUE)],
                            by_country$GEO[order(abs(errors$MAPE),decreasing = TRUE)],
                            by_country$GEO[order(errors$sMAPE,decreasing = TRUE)])

colnames(lower_errors) <- c("MAE","RMSE","MAPE","sMAPE")
kable(head(lower_errors))
```

MAE	RMSE	MAPE	sMAPE
Germany	Germany	Luxembourg	Ireland
United Kingdom	United Kingdom	Ireland	Luxembourg
Ireland	Ireland	Malta	Malta
France	France	Romania	Romania
Spain	Spain	Czech Republic	Czech Republic
Czech Republic	Czech Republic	Hungary	Hungary

Conclusion

The main strength of the procedure I followed here is that it is quite cheap in terms of its complexity and the data gathering efforts needed for the accuracy you end up getting. No central bank would dare to use it in their forecasts since, for their lofty purposes, the diminishing returns of more complex methods are more than worth the extra resources needed, but this procedure could still be used as a kind of sanity check to see if that extra investment is worth it if the difference in accuracy is insignificant from the more simple, and considerably cheaper, method.

There's also ways you could make this procedure cheaper in computational terms. I previously mentioned a multi-core upgrade that could be implememented (and which I hope to do soon), but one other issue with the procedure is that the cross-validation step takes into account the full dataset when it likely doesn't need to. The idea is that the model fitted with data ranging from 2010Q1 to 2016Q4 might have better forecasting performance for 2017 than the model fitted with data from 2005Q1 to 2016Q4. This extra data that we are dealing with not only might be hindering the forecasting accuracy we are getting - though that's an hypothesis, we would have to check, - but it also means that each cross-validation step needs longer to run.

That is, a shorter timeframe of the data used is also a consideration to keep in mind in the speed-accuracy tradeoff, not only the interval range of the parameters used.

All in all, I am quite satisfied with the project.