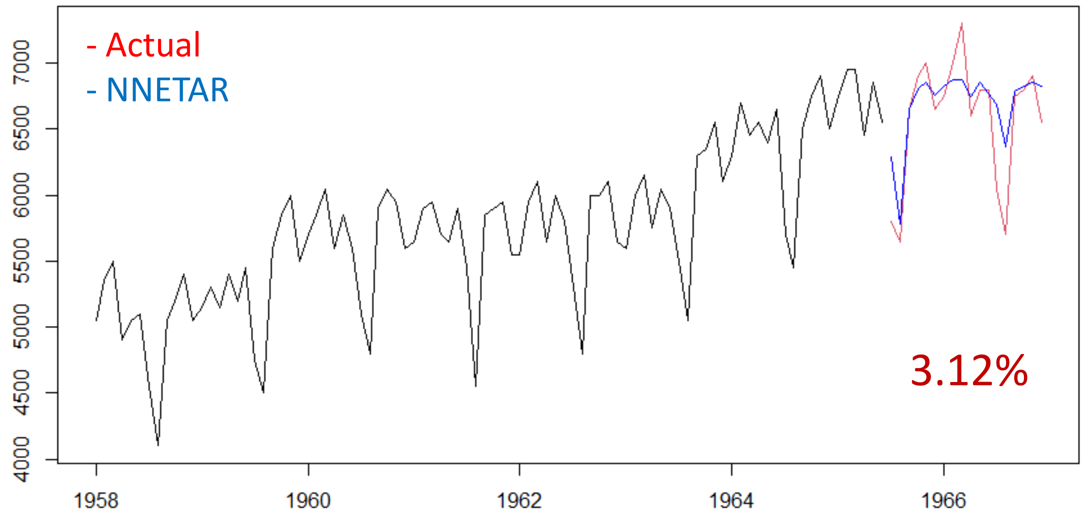
Advanced Machine Learning Methods

Week 4: Advanced ML Methods with Applications

UNIVERSITY *of* NICOSIA

Auto-regressive ML methods

```
library(Mcomp)
series <- subset(M3, 12)[[1110]]
insample <- series$x
outsample <- series$xx
plot(series)

set.seed(647)
frcs <- forecast(nnetar(insample), h=18)$mean
mean(abs(frcs-outsample)*100/outsample) #MAPE
plot(series)
lines(ts(frcs, frequency = frequency(outsample), start = start(outsample)), col="blue")
```



We have to define an accuracy measure as an evaluation criteria, such as MAPE

$$MAPE = \frac{1}{18} \sum_{i=n+1}^{n+18} \frac{|F_i - Y_i|}{Y_i} * 100\%$$

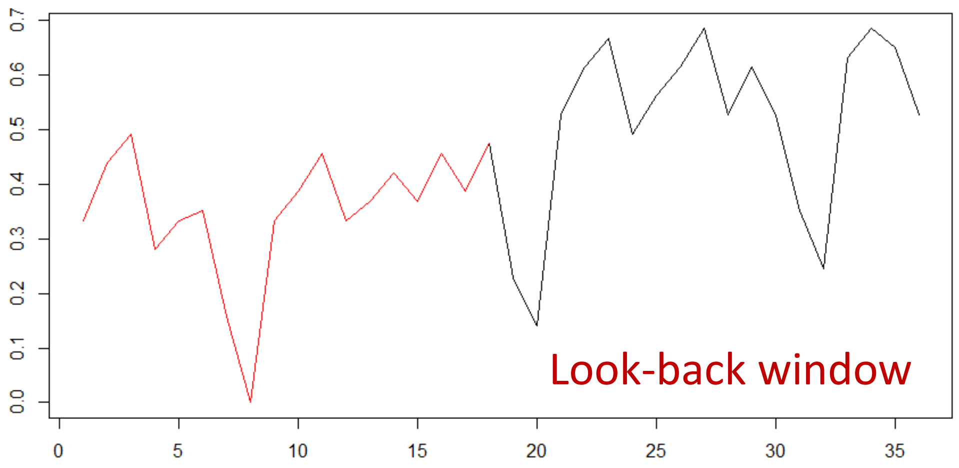
UNIC

MOFC

Auto-regressive ML methods

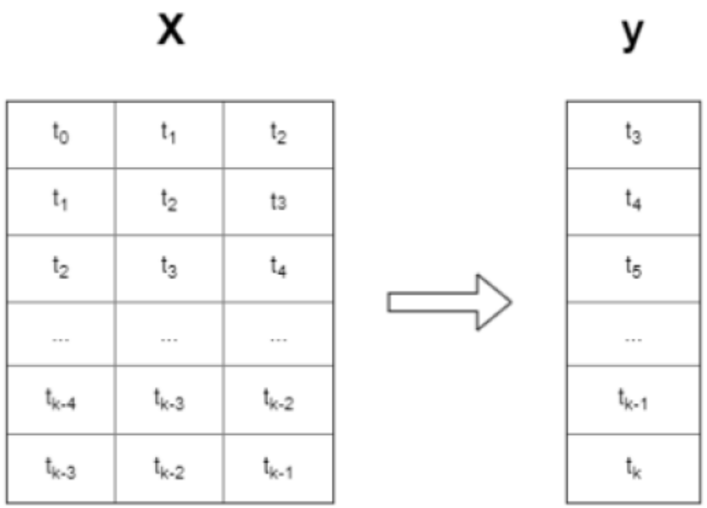
```
h <- 18
lag_length <- 18
train_x = train_y <- NULL
for (i in lag_length:(length(insample)-h)){
  tmp <- as.numeric(tail(insample[1:i], lag_length))
  train_x <- rbind(train_x, (tmp-min(tmp))/(max(tmp)-min(tmp)))
  tmp2 <- as.numeric(tail(insample[1:(i+h)], h))
  train_y <- rbind(train_y, (tmp2-min(tmp))/(max(tmp)-min(tmp)))
}

#Look-back window
plot(c(train_x[1,],train_y[1,]), type="l")
lines(c(train_x[1,],rep(NA, h)), col="red")
```

The look-back window is typically larger than the frequency of the series to capture seasonality



Auto-regressive ML methods

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
[1,]	0.2280702	0.1403509	0.5263158	0.6140351	0.6666667	0.4912281	0.5614035	0.6140351	0.6842105
[2,]	0.1403509	0.5263158	0.6140351	0.6666667	0.4912281	0.5614035	0.6140351	0.6842105	0.5263158
[3,]	0.5263158	0.6140351	0.6666667	0.4912281	0.5614035	0.6140351	0.6842105	0.5263158	0.6140351
[4,]	0.6140351	0.6666667	0.4912281	0.5614035	0.6140351	0.6842105	0.5263158	0.6140351	0.5263158
[5,]	0.6666667	0.4912281	0.5614035	0.6140351	0.6842105	0.5263158	0.6140351	0.5263158	0.3508772
[6,]	0.4912281	0.5614035	0.6140351	0.6842105	0.5263158	0.6140351	0.5263158	0.3508772	0.2456140



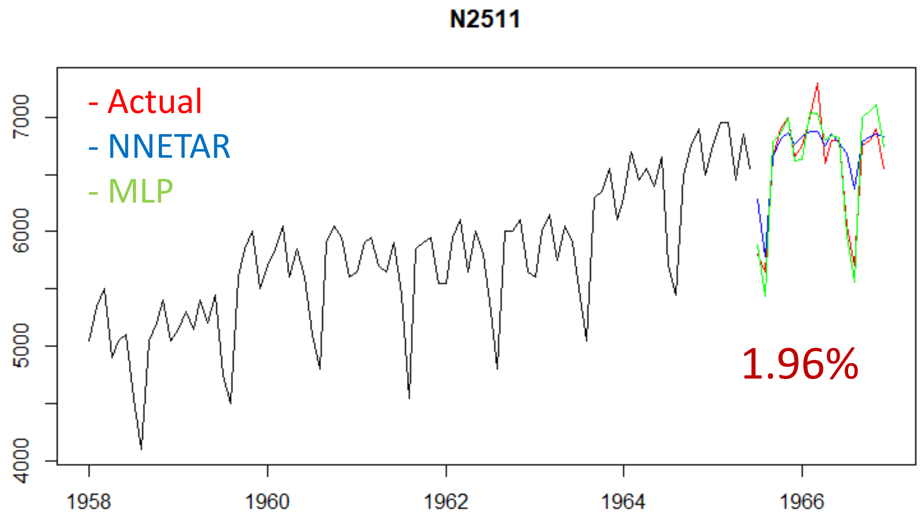
lagged inputs

UNIC

MOFC

Auto-regressive ML methods

```
tmp <- t(matrix(tail(insample, lag_length)))
test_x <- (tmp-min(tmp))/(max(tmp)-min(tmp))
test_y <- as.numeric(outsample)
train.set <- data.frame(train_x, train_y)

library(RSNNS)
set.seed(647)
mlp.fit = mlp(train_x, train_y, size = c(18*2), maxit = 250,
        initFunc = "Randomize_Weights", learnFunc = "Std_Backpropagation",
        hiddenActFunc = "Act_Logistic", linOut = TRUE)
frcs_mlp <- as.numeric(predict(mlp.fit, test_x))
frcs_mlp <- frcs_mlp*(max(tmp)-min(tmp)) + min(tmp) #Re-scale
mean(abs(frcs_mlp-outsample)*100/outsample) #MAPE
lines(ts(frcs_mlp, frequency = frequency(outsample), start = start(outsample)), col="green")
```



Auto-regressive ML methods - Recurrent

```
h <- 1
lag_length <- 18
train_x = train_y <- NULL
for (i in lag_length:(length(insample)-h)){
  tmp <- as.numeric(tail(insample[1:i], lag_length))
  train_x <- rbind(train_x, (tmp-min(tmp))/(max(tmp)-min(tmp)))
  tmp2 <- as.numeric(tail(insample[1:(i+h)], h))
  train_y <- rbind(train_y, (tmp2-min(tmp))/(max(tmp)-min(tmp)))
}
test_x <- t(matrix(tail(insample, lag_length)))
test_y <- as.numeric(outsample)

train.set <- data.frame(train_x, train_y)
```

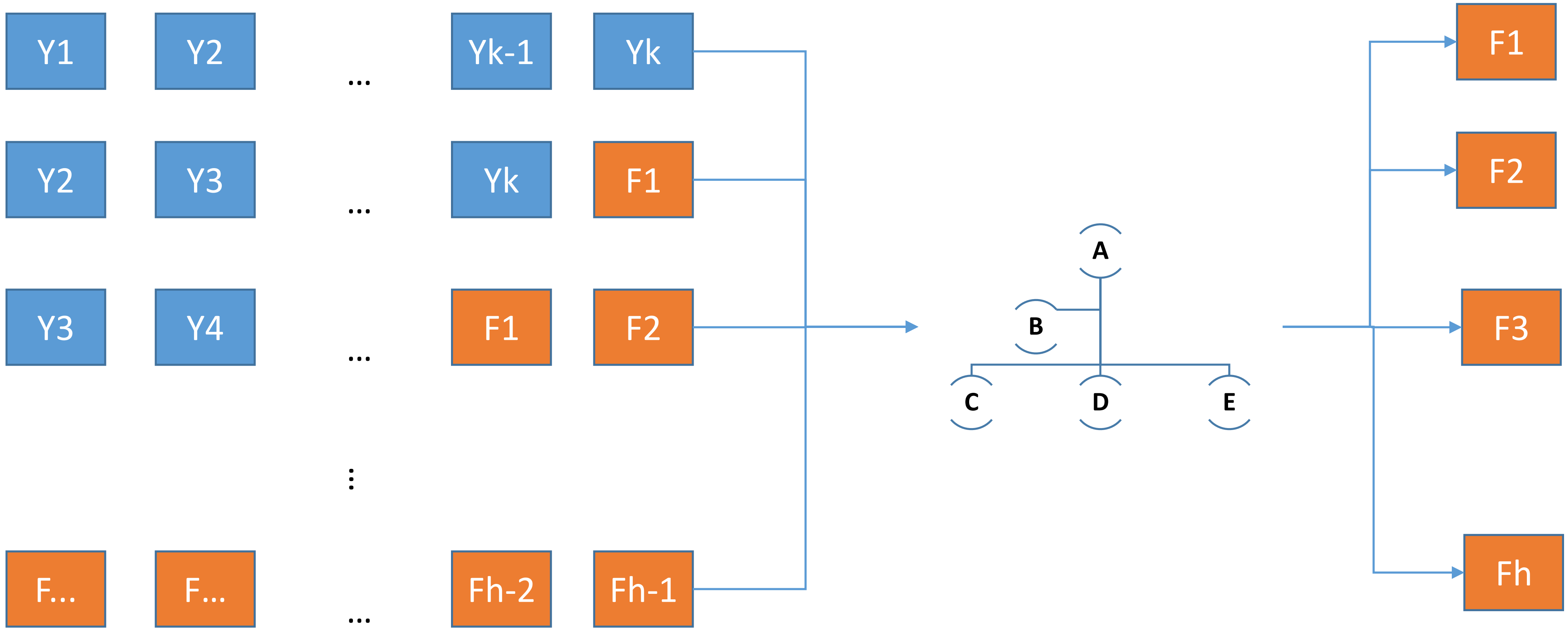
Recurrent forecasting

In contrast to NNs, Tree-based models have a single output so, *typically*, it is not possible for them to produce multi-step-ahead forecasts

UNIC

MOFC

Auto-regressive ML methods - Recurrent

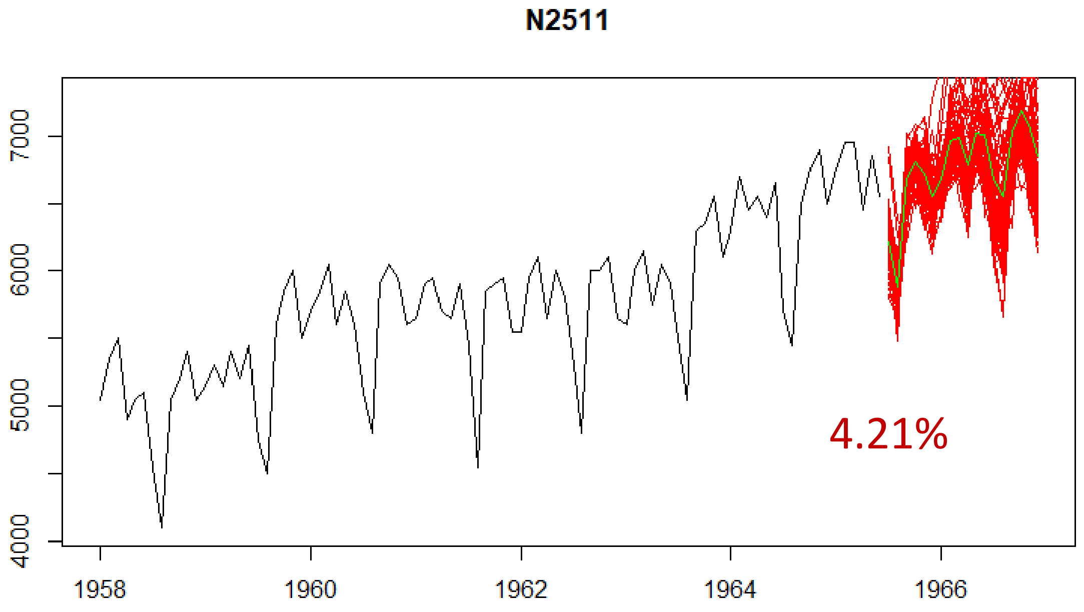


UNIC

MOFC

Auto-regressive ML methods - Recurrent

```
#Train multiple models
frc_list <- NULL
plot(series)
for (i in 1:100){
 set.seed(i)
 xgb.fit = gbm(train_y~., train.set,
              distribution = "laplace", n.trees = 500,
              interaction.depth = 3, shrinkage = runif(1,0.1,0.9))
 test.set <- data.frame(test_x)
 frcs <- c()
 for (fh in 1:18){
   tmp_x <- (test.set-min(test.set))/(max(test.set)-min(test.set))
   frcs <- c(frcs, predict(xgb.fit,newdata=tmp_x, n.trees = 500)*(max(test.set)-
min(test.set))+min(test.set))
   test.set[,1:(ncol(test.set)-1)] <- as.numeric(test.set[,2:ncol(test.set)])
   test.set[,ncol(test.set)] <- tail(frcs,1)
 }
 lines(ts(frcs, frequency = frequency(outsample), start = start(outsample)), col="red")
 frc_list <- rbind(frc_list, frcs)
}
frc_ens <- colMeans(frc_list)
mean(abs(frc_ens-outsample)*100/outsample) #MAPE
lines(ts(frc_ens, frequency = frequency(outsample), start = start(outsample)), col="green")
```



UNIC

MOFC

LightGBM

- M5 is the first competition where all top performing methods were both ``pure'' ML ones and significantly better than all statistical benchmarks and their combinations

- LightGBM proved that it can be used effectively to process numerous, correlated series and exogenous/explanatory variables and reduce forecast error

✓ Fast to compute
✓ Easy to implement and experiment with
✓ Accurate
✓ Capable of handling complex, non-linear dependencies

LightGBM

```
#LightGBM
# library(remotes)
# library(devtools)
# PKG_URL <- "https://github.com/microsoft/LightGBM/releases/download/v3.0.0/lightgbm-3.0.0-r-cran.tar.gz"
# remotes::install_url(PKG_URL)

library(lightgbm)
library(methods)
library(Matrix)
library(lubridate)
dataset_in <- read.csv("path/Walmart data.csv", stringsAsFactors = F)
dataset_in$year <- as.factor(year(dataset_in$date))
dataset_in$month <- as.factor(month(dataset_in$date))
dataset_in$weekdays <- as.factor(wday(dataset_in$date))
dataset_in[is.na(dataset_in$event_type),]$event_type <- "NoEvent"
```

Extract useful features to capture seasonality

LightGBM

```
Sales <- dataset_in
Sales$TX = Sales$WI = Sales$Total <- NULL
Sales$snap_TX = Sales$snap_WI <- NULL
Sales$date <- NULL
colnames(Sales)[1] <- "sales"

Sales$Lag30 = Sales$Lag60 <- NA
for (i in 61:nrow(Sales)){
  Sales$Lag30[i] <- Sales$sales[i-30]
  Sales$Lag60[i] <- Sales$sales[i-60]
}
Sales <- na.omit(Sales)
rownames(Sales) <- NULL
data_train_test <- sparse.model.matrix(sales ~ .-1, data = Sales)
```

Lagged features – *Recurrent features also applicable*

```
x_train <- head(data_train_test,1342)
x_test <- tail(data_train_test,28)
y_train = head(Sales,1342)$sales
y_test = tail(Sales,28)$sales

x_train@Dimnames
```

Both categorical and lagged variables are used

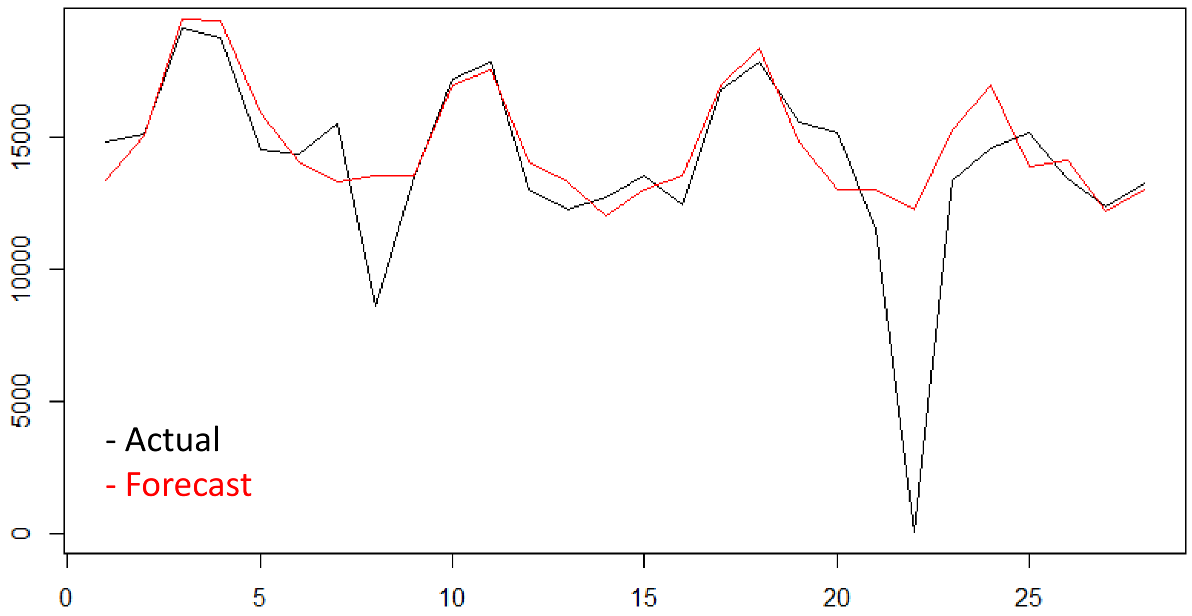
```
[[2]]
 [1] "event_typeCultural"  "event_typeNational"  "event_typeNoEvent"   "event_typeReligious" "event_typeSporting"
 [6] "snap_CA"             "year2012"            "year2013"            "year2014"            "month2"
[11] "month3"              "month4"              "month5"              "month6"              "month7"
[16] "month8"              "month9"              "month10"             "month11"             "month12"
[21] "weekdays2"           "weekdays3"           "weekdays4"           "weekdays5"           "weekdays6"
[26] "weekdays7"           "Lag60"               "Lag30"
```

UNIC

MOFC

LightGBM

```
bst <- lightgbm(
  data = x_train
  , label = y_train
  , learning_rate = 0.1
  , nrounds = 50L
  , objective = "regression"
  , verbose = 0
  , max_bin = 5
)

pred <- (predict(bst, x_test))
plot(y_test, type="l")
lines(pred, col="red")
mean(abs(pred-y_test)*200/(abs(y_test)+abs(pred)))
```



The method can be heavily parameterized, such as

- learning_rate: shrinkage rate
- nrounds: Number of training rounds
- objective: L2 loss or other optimization criteria
- max_bin: max number of bins that feature values will be bucketed in

https://lightgbm.readthedocs.io/en/latest/Parameters.html

UNIC

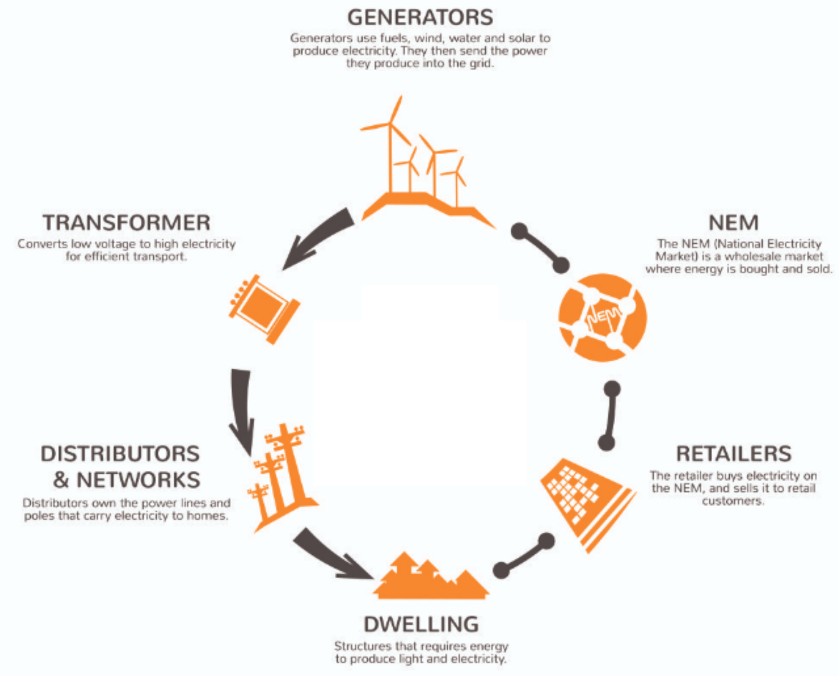
MOFC

Case-study

Week 4: Advanced ML Methods with Applications

Electricity price

- Electricity markets have been deregularizing, thus becoming more competitive when compared to their government-controlled ancestors
- Electricity price typically displays notable volatility
- Electricity cannot be stored at a large scale and production is not always optimally balanced to the consumption,



Implications for the operation and management of energy companies:

- Over- or under-contracting leads to high costs that may result in notable financial losses and deteriorate the profitability of the companies.
- Accurately predicting the price of electricity for the following hours or days, introduces vast opportunities for the energy companies, enhancing their competitive advantage and allowing for better short- and mid-term planning

Weron, R. (2014). Electricity price forecasting: A review of the state-of-the-art with a look into the future. International Journal of Forecasting, 30 (4), 1030-1081.

Electricity price forecasting

- A challenging task as it involves predicting series that are **influenced by numerous variables**, such as weather conditions, electricity consumption, and seasonal factors

- Accurate forecasts are necessary for supporting operational management and short- to mid-term planning of energy companies

- Various forecasting approaches have been proposed in the literature to perform this task, including statistical and Machine Learning (ML) methods

- Studies comparing their performance have been inconclusive with regards to the superiority of the one type of technique over the other in the task of forecasting for energy markets

Lago, J. , De Ridder, F. & De Schutter, B. (2018). Forecasting spot electricity prices: Deep learning approaches and empirical comparison of traditional algorithms. Applied Energy, 221, 386-405

Lago. J., De Ridder, F., Vrancx, P. & De Schutter, B. (2018). Forecasting day-ahead electricity prices in Europe: The importance of considering market integration. Applied Energy, 211, 890-903

UNIC

MOFC

Variables affecting electricity price

- ✓ **Demand vs. Production**, especially from Renewable Energy Sources (RES)

- ✓ **Seasonality**
  - Weekday (Monday-Sunday)
  - Time
  - Month or Season
  - Holidays

- ✓ **Events, recent changes and cross-correlations**
  - Special Events
  - Weather conditions
  - Trends in demand and supply
  - Fuel prices
  - Market integration

- ✓ Other **non-observable** variables

UNIC

MOFC

Required libraries

- library(lubridate) #For handling timestamps
- library(DT) #Helps us work with data.frames
- library(forecast) #Time series forecasting methods
- library(plyr) #Aggregating data for various key variables
- library(ggplot2) #Data visualization
- library(nnet) #Packages for applying Neural Networks
- library(neuralnet)
- library(RSNNS)

Data visualization (1/4)

#Read data
train <- read.csv("PATH/TrainSet.csv",stringsAsFactors = F, sep=",")
#Observe data
datatable(tail(train,100))

Target Variable
What we need to forecast

Show 10 entries Search:

	datetime_utc	Generation_BE	Generation_FR	Prices.BE	holidaysBE
59709	2016-10-26 20:00:00	9591.295	55930	92.23	0
59710	2016-10-26 21:00:00	10653.47	54446	72.39	0
59711	2016-10-26 22:00:00	10669.6325	52152	62.58	0
59712	2016-10-26 23:00:00	10679.87	52635	57.05	0
59713	2016-10-27 00:00:00	9607.925	51286	52.58	0
59714	2016-10-27 01:00:00	9606.805	49003	44.86	0
59715	2016-10-27 02:00:00	9603.085	49149	42.31	0
59716	2016-10-27 03:00:00	9599.0675	49161	39.66	0
59717	2016-10-27 04:00:00	9598.2325	47910	38.98	0
59718	2016-10-27 05:00:00	9594.8025	48307	42.31	0

From the timestamp we can extract useful information about seasonality

Binary variable indicating whether there is a public holiday (1) or not (0)

Forecasts about the amount of energy that we expect to be generated in Belgium and France

UNIC

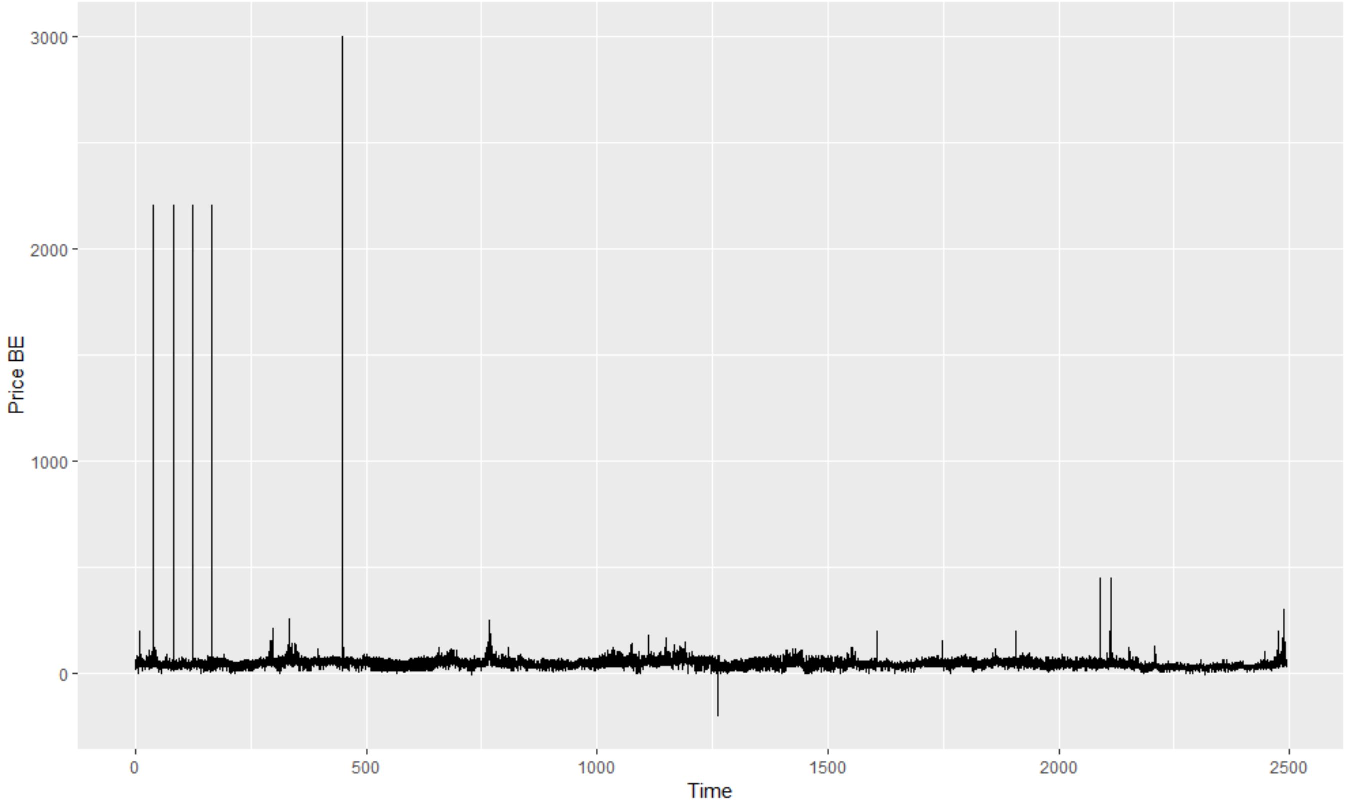
MOFC

Data visualization (2/4)

#Visualize data

```
plot(ts(train$Prices.BE,frequency = 24), ylab="Price BE")
```

This is a time-series object of frequency 24 hours



UNIC

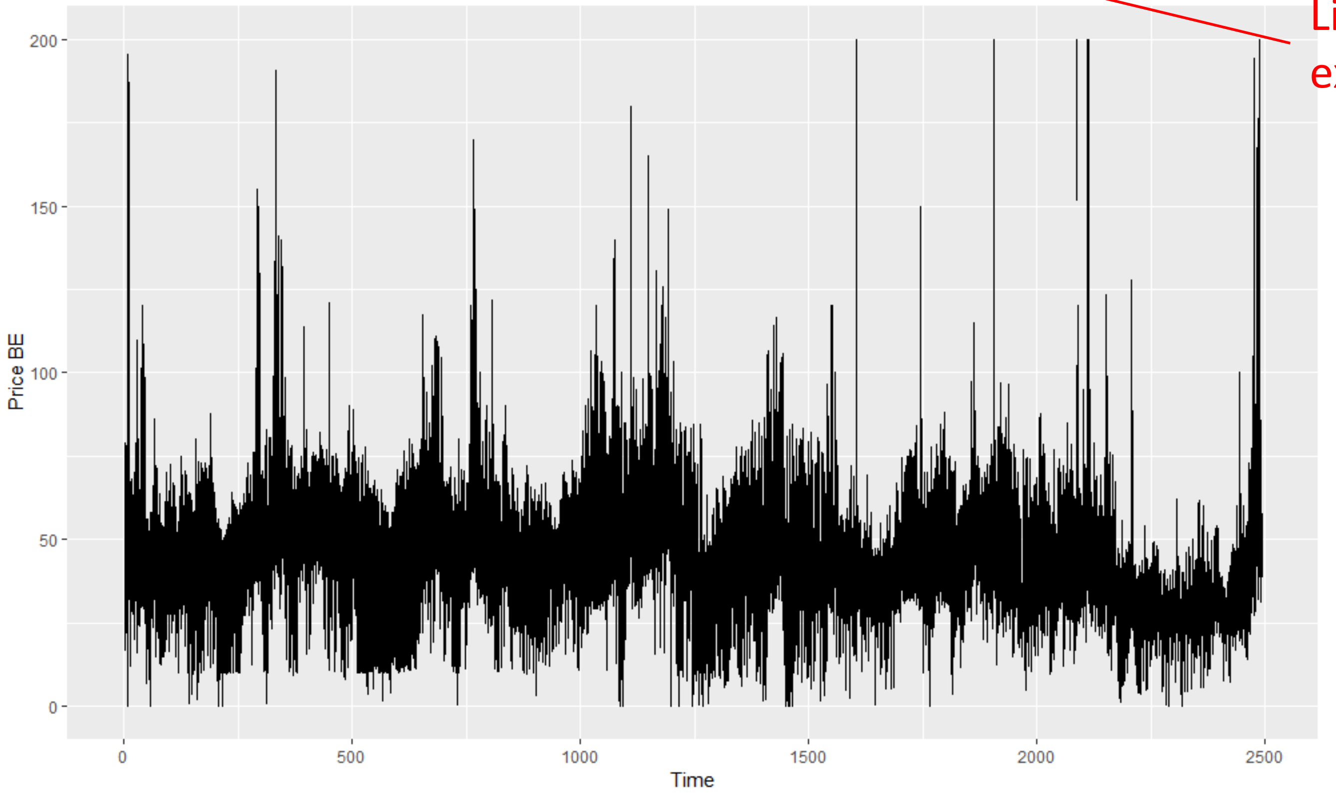
MOFC

Data visualization (3/4)

#See data - reasonable scale

```
plot(ts(train$Prices.BE,frequency = 24), ylab="Price BE", ylim=c(0,200))
```

Limit to 200 to exclude outliers

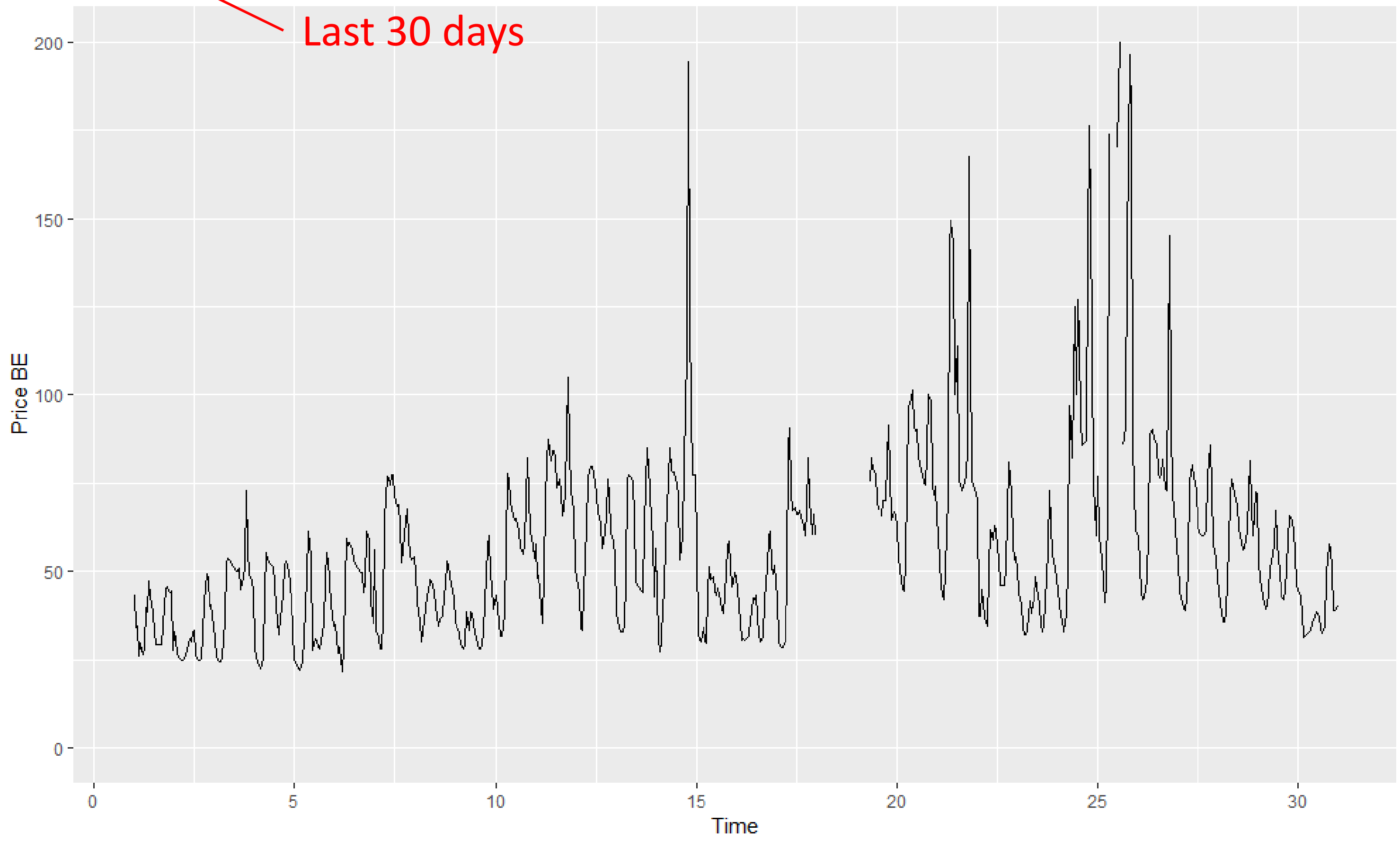


Hourly data from
2010-01-04
to
2016-10-30

Data visualization (4/4)

#See data - last month

```
autoplot(ts(tail(train$Prices.BE,30*24),frequency = 24), ylab="Price BE", ylim=c(0,200))
```



Missing values (1/5)

#Identify missing values

```
missing <- train[is.na(train$Prices.BE)==T,]
datatable(missing)
nrow(missing)
```

Find NA values

Show 100 entries

Search:

	datetime_utc	Generation_BE	Generation_FR	Prices.BE	holidaysBE
59497	2016-10-18 00:00:00	9200.3375	48167		0
59498	2016-10-18 01:00:00	9201.82	46300		0
59499	2016-10-18 02:00:00	9211.855	46482		0
59500	2016-10-18 03:00:00	9227.1825	45650		0
59501	2016-10-18 04:00:00	9257.54	45059		0
59502	2016-10-18 05:00:00	9313.8375	45142		0
59503	2016-10-18 06:00:00	9397.695	48633		0
59504	2016-10-18 07:00:00	9490.1775	51597		0
59505	2016-10-18 08:00:00	10004.775	52651		0
59506	2016-10-18 09:00:00	12458.7	52529		0
59507	2016-10-18 10:00:00	14430.6375	52537		0
59508	2016-10-18 11:00:00	15114.165	52105		0

29 missing (NA) values on days 18 and 19/11/2016 that have to be filled

UNIC

MOFC

Missing values (2/5)

```
#Fix missing values
train$Date <- as.Date(train$datetime_utc)
train$DateType <- wday(train$Date)
train$Hour <- hour(train$datetime_utc)
train1 = train2 = train3 = train4 <- train
```

Define day of week and hour (*or also month, year, start/mid/end of month…*)

```
#Create profiles
profiles <- ddply(na.omit(train[,-1]), .(DateType,Hour), colwise(mean))
```

Data is aggregated (averaged) for each weekday and hour

Missing values (3/5)

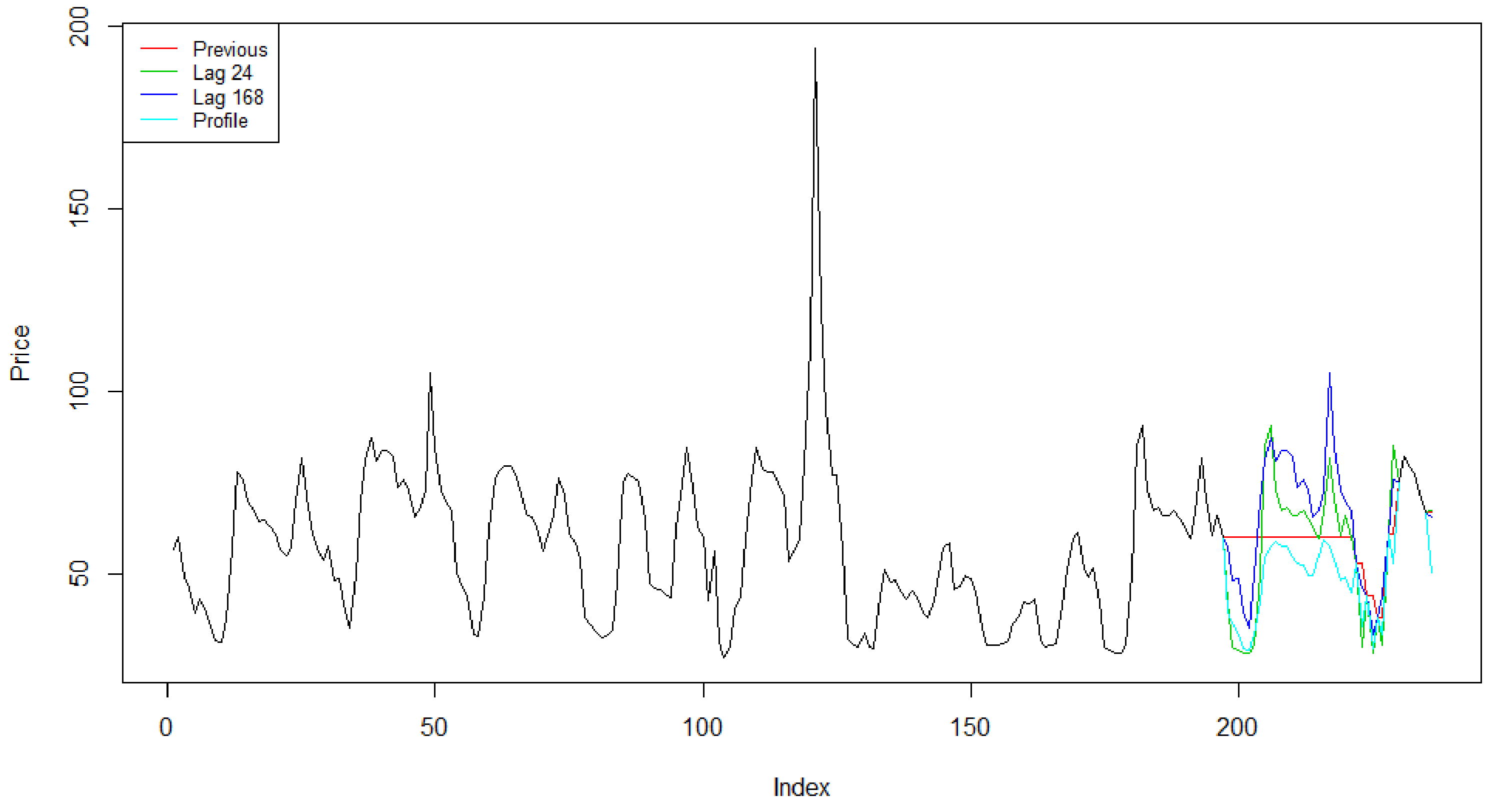
```
#Fill NAs
for (i in 59300:59535){
  if (is.na(train$Prices.BE[i])==T){
    train1$Prices.BE[i] <- train1$Prices.BE[i-1]
    train2$Prices.BE[i] <- train2$Prices.BE[i-24]
    train3$Prices.BE[i] <- train3$Prices.BE[i-168]
    train4$Prices.BE[i] <- profiles[(profiles$Hour==train4$Hour[i])&(profiles$DateType==train4$DateType[i]),]$Prices.BE
  }
}
#Plot solutions
plot(train1$Prices.BE[59300:59535],type="l", ylab = "Price",col=2)
lines(train2$Prices.BE[59300:59535],type="l", ylab = "Price",col=3)
lines(train3$Prices.BE[59300:59535],type="l", ylab = "Price",col=4)
lines(train4$Prices.BE[59300:59535],type="l", ylab = "Price",col=5)
lines(train$Prices.BE[59300:59535],type="l")
legend("topleft", legend=c("Previous", "Lag 24", "Lag 168", "Profile"),col=c(2, 3, 4, 5), lty=1, cex=0.8)
```

Fill missing values for each scenario (*last hour, same hour - last day, same hour - last week, average profile of weekday and hour*)

UNIC

MOFC

Missing values (4/5)



UNIC

MOFC

Missing values (5/5)

```
#Decide which method to use and apply changes
for (i in 59300:59535){
  if (is.na(train$Prices.BE[i])==T){
    train$Prices.BE[i] <- train$Prices.BE[i-168]
  }
}
missing <- train[is.na(train$Prices.BE)==T,]
nrow(missing)
```

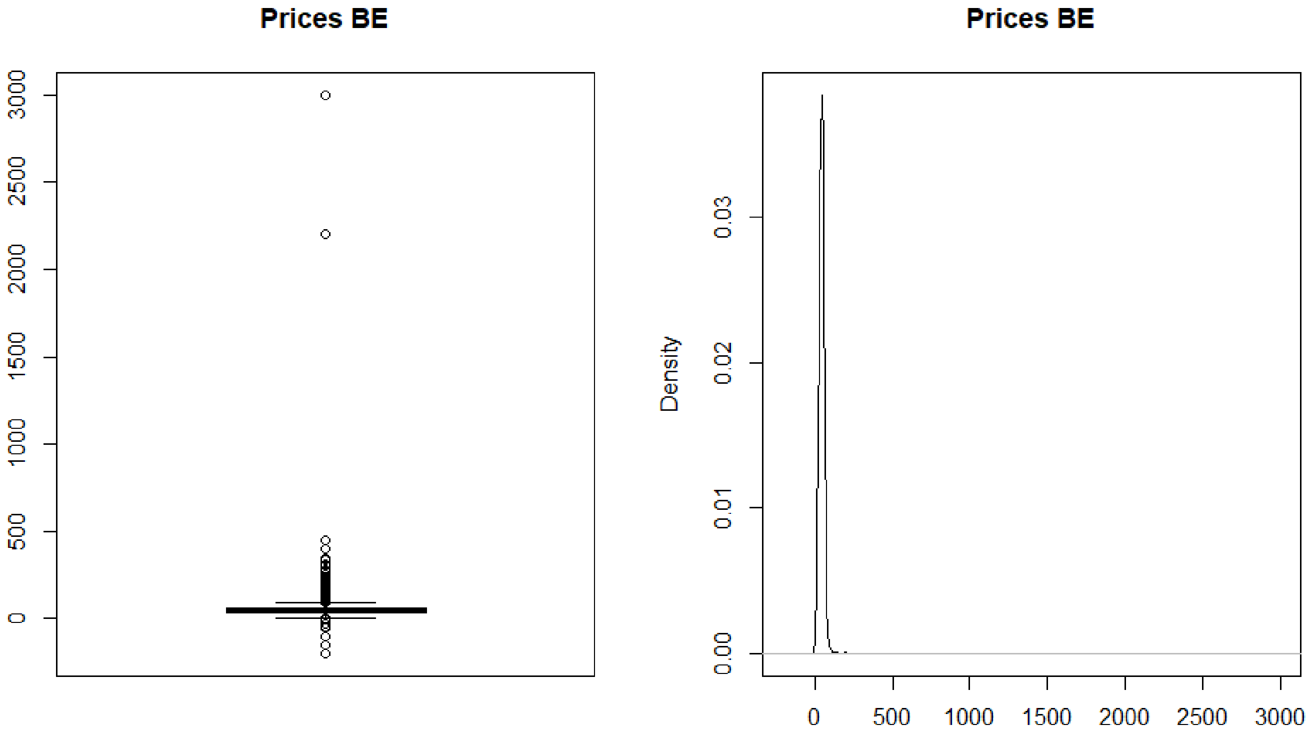
- The best way to see which approach works best, it to artificially create some missing values yourself and test, based on an accuracy measure (e.g. MSE) which would provide the best results
- Note that filling missing values is a prerequisite **ONLY** for time series forecasting methods and not for regression ones. In the latter case the NA observations can be simply removed from the train set

Extreme values (1/4)

#Deal with extreme values

```
par(mfrow=c(1,2))
boxplot(train$Prices.BE,main="Prices BE") ;
plot(density(train$Prices.BE),main="Prices BE")
```

Distribution of Price



UNIC

MOFC

Extreme values (2/4)

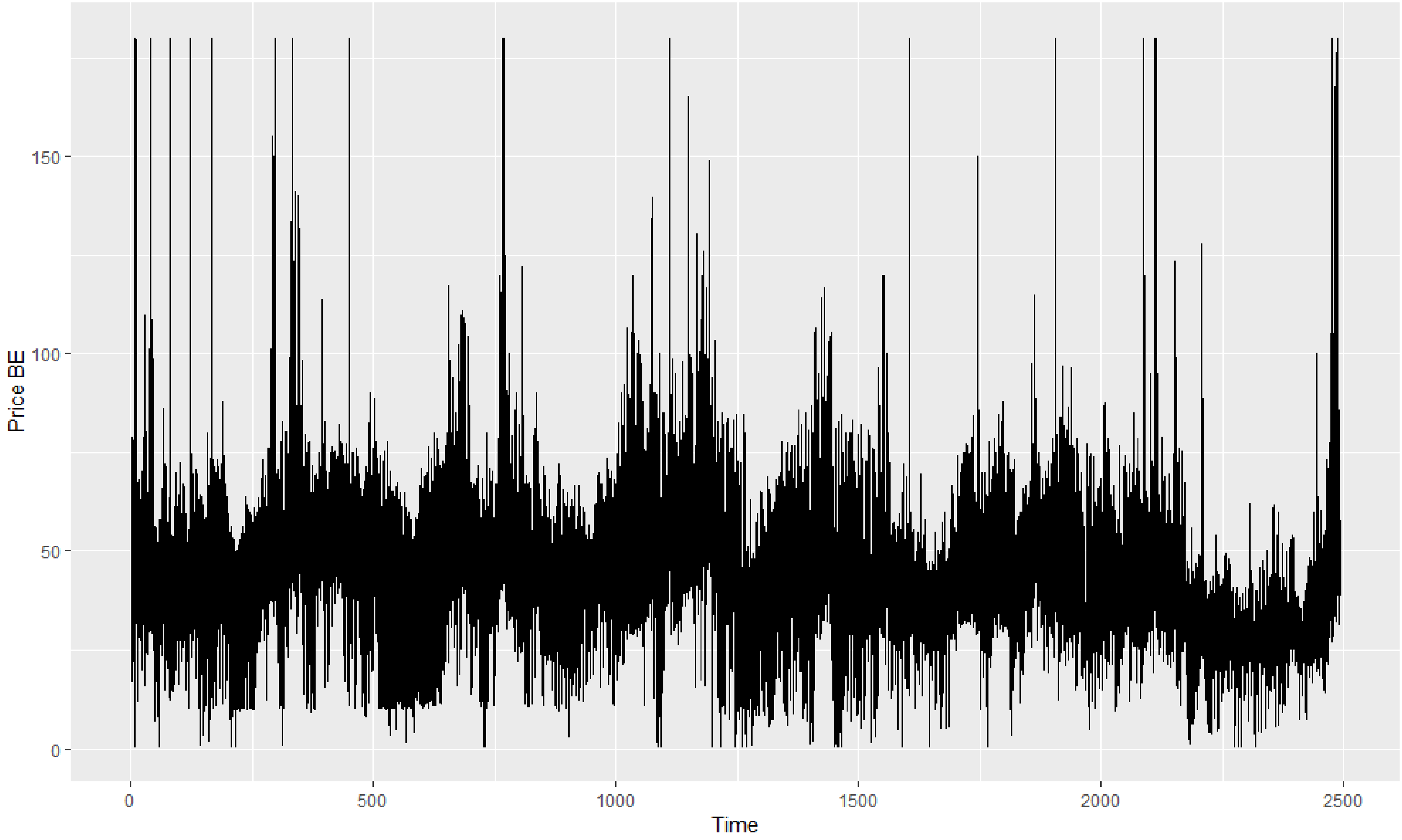
```
#Fix upper and lower bounds
LimitUp <- quantile(train$Prices.BE,0.999)
LimitDown <- quantile(train$Prices.BE,0.001)
train[train$Prices.BE>LimitUp,]$Prices.BE <- LimitUp
train[train$Prices.BE<LimitDown,]$Prices.BE <- LimitDown
#See data again
autoplot(ts(train$Prices.BE,frequency = 24), ylab="Price BE")
```

Find lowest/highest 0.1% values and use that as limit

UNIC

MOFC

Extreme values (3/4)

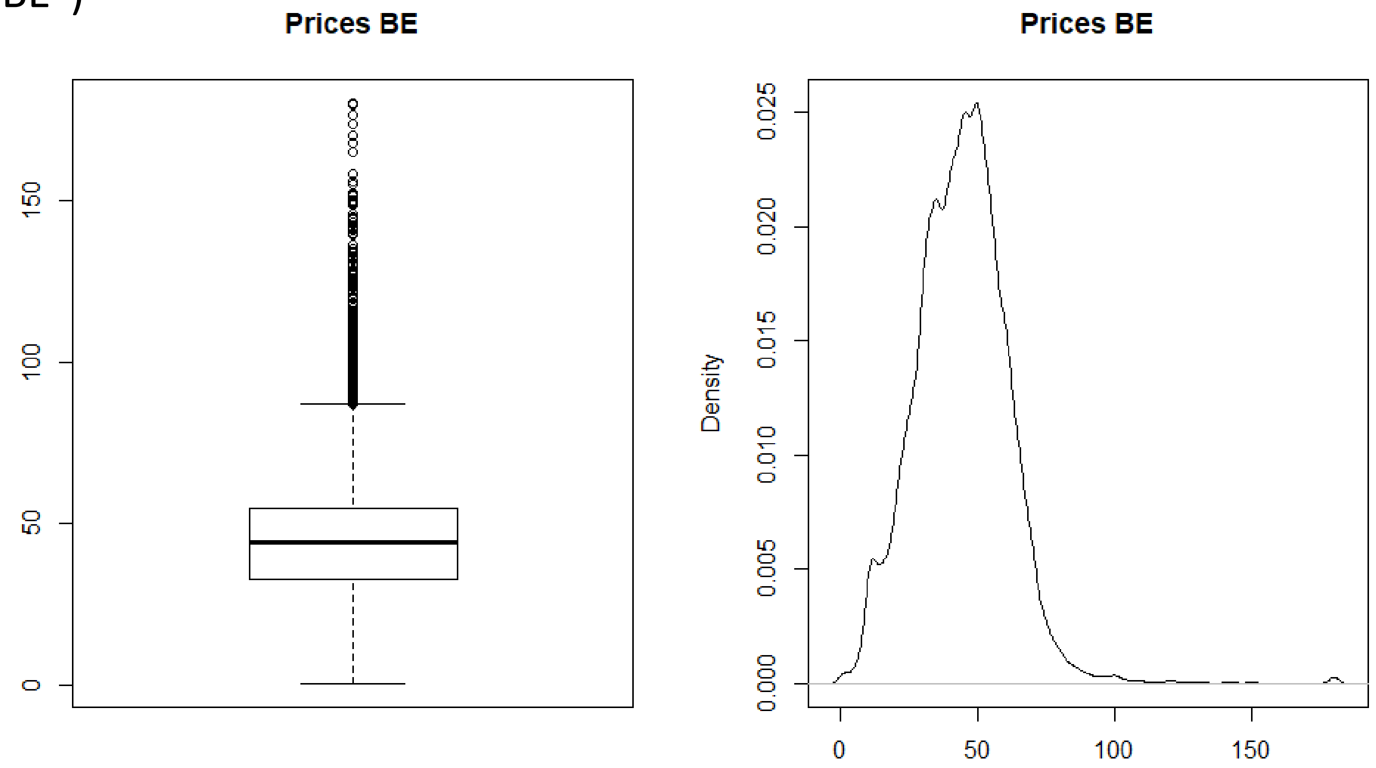


A threshold of 99,9% is used to normalize the data, i.e., remove extreme values

Extreme values (4/4)

```
#Re-inspect values
par(mfrow=c(1,2))
boxplot(train$Prices.BE,main="Prices BE") ;
plot(density(train$Prices.BE),main="Prices BE")
```

As an alternative, extreme values can be fixed using the same approach selected for the case of the missing values



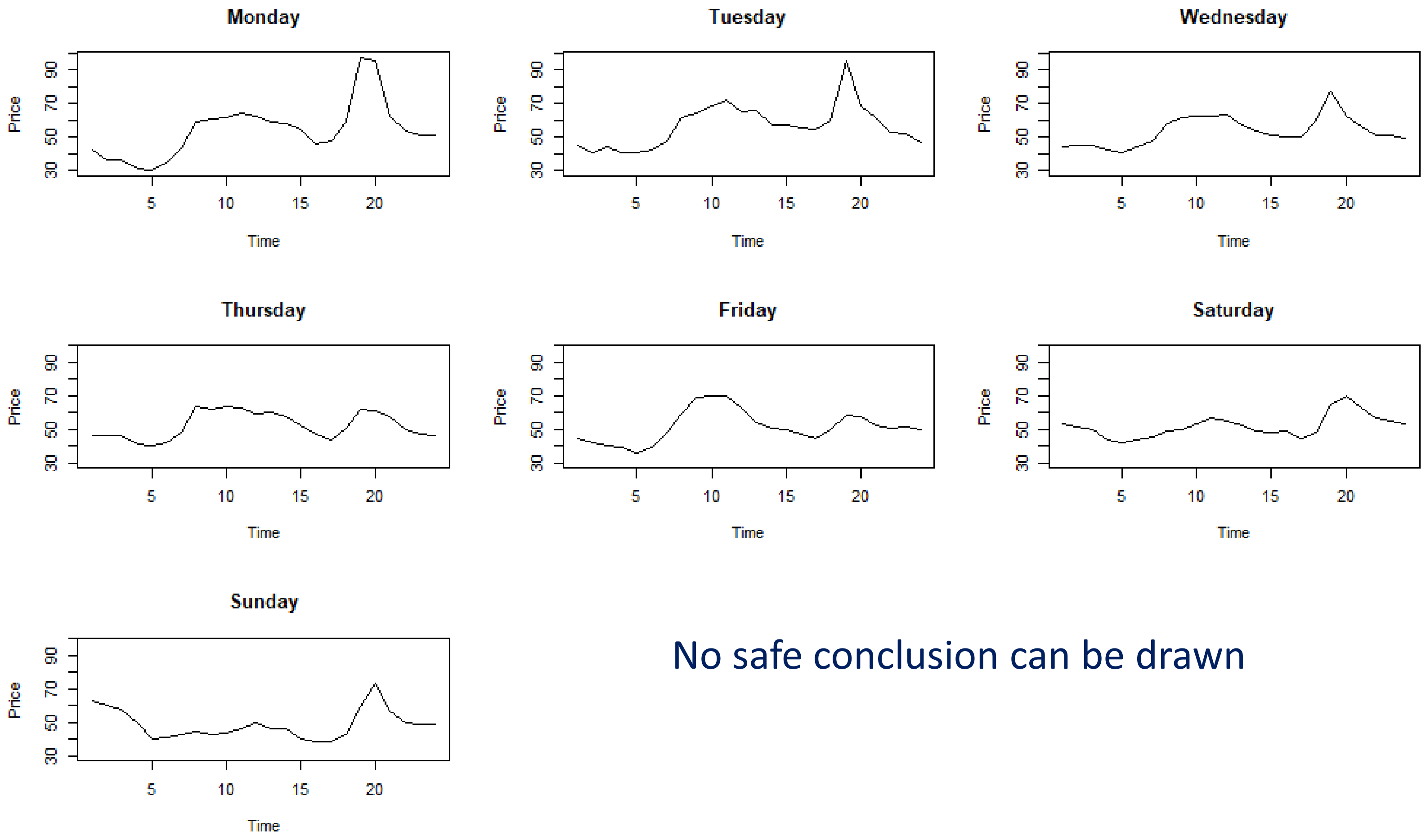
UNIC

MOFC

Exploratory Data Analysis - EDA (1/3)

```
#Start analysis to proceed with forecasting
par(mfrow=c(3,3))
maxi <- max(train[(train$Date>="2015-02-02")&(train$Date<="2015-02-08"),]$Prices.BE)
mini <- min(train[(train$Date>="2015-02-02")&(train$Date<="2015-02-08"),]$Prices.BE)
plot(train[train$Date=="2015-02-02",]$Prices.BE,main="Monday",type="l",ylab ="Price",xlab="Time",ylim = c(mini,maxi))
plot(train[train$Date=="2015-02-03",]$Prices.BE,main="Tuesday",type="l",ylab ="Price",xlab="Time",ylim = c(mini,maxi))
plot(train[train$Date=="2015-02-04",]$Prices.BE,main="Wednesday",type="l",ylab ="Price",xlab="Time",ylim = c(mini,maxi))
plot(train[train$Date=="2015-02-05",]$Prices.BE,main="Thursday",type="l",ylab ="Price",xlab="Time",ylim = c(mini,maxi))
plot(train[train$Date=="2015-02-06",]$Prices.BE,main="Friday",type="l",ylab ="Price",xlab="Time",ylim = c(mini,maxi))
plot(train[train$Date=="2015-02-07",]$Prices.BE,main="Saturday",type="l",ylab ="Price",xlab="Time",ylim = c(mini,maxi))
plot(train[train$Date=="2015-02-08",]$Prices.BE,main="Sunday",type="l",ylab ="Price",xlab="Time",ylim = c(mini,maxi))
```

Exploratory Data Analysis - EDA (2/3)



No safe conclusion can be drawn

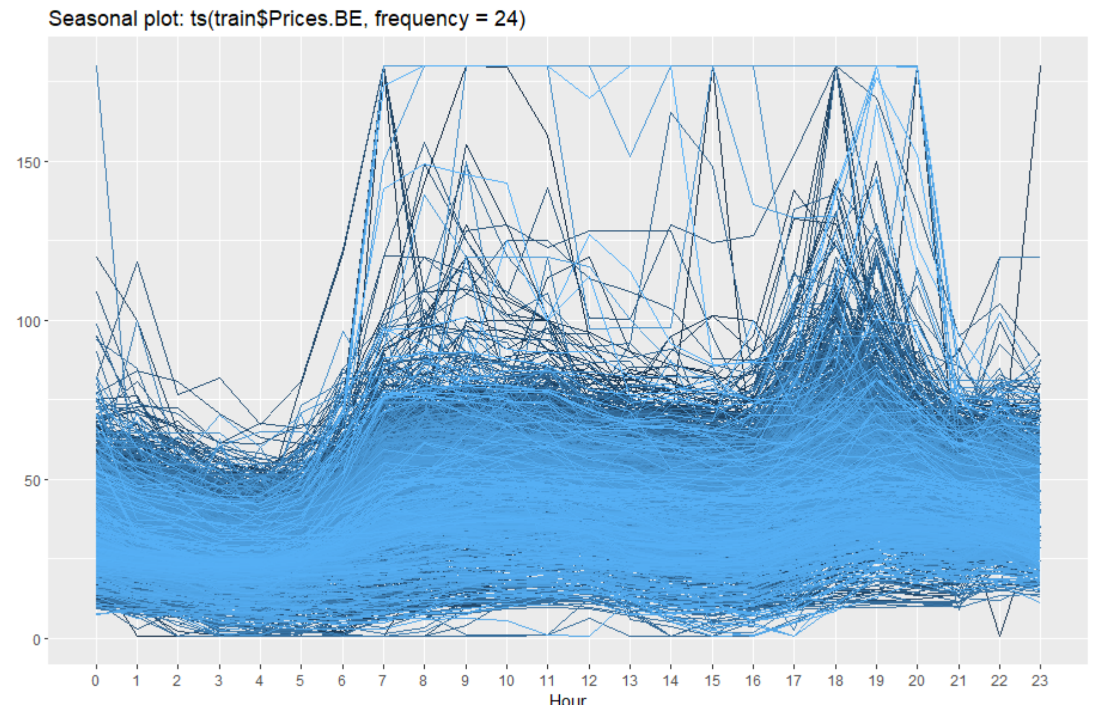
UNIC

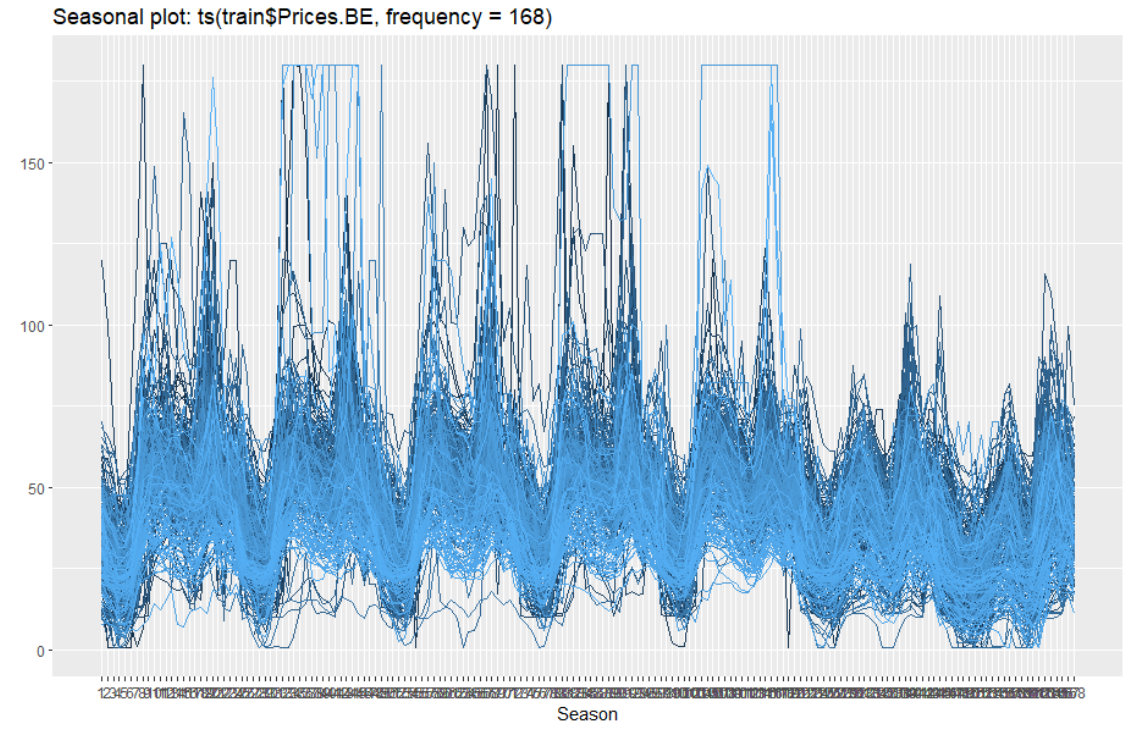
MOFC

Exploratory Data Analysis - EDA (3/3)

#Inspect Seasonplots

```
ggseasonplot(ts(train$Prices.BE, frequency=24),continuous=TRUE)
ggseasonplot(ts(train$Prices.BE, frequency=168),continuous=TRUE)
```





UNIC

MOFC

Validation of forecasting alternatives (1/15)

```
#Examine possible scenarios for producing forecasts
timeseries <- ts(train$Prices.BE, frequency=168)
fh <- 168
insample <- head(timeseries,length(timeseries)-fh)
outsample <- tail(timeseries,fh)
```

Set the forecasting horizon to 168

Create train and test sample

We have to define an accuracy measure as an evaluation criteria, such as sMAPE

$$sMAPE = \frac{1}{168}\sum_{i=n+1}^{n+168}\frac{|F_i - Y_i|}{|F_i| + |Y_i|} * 200\%$$

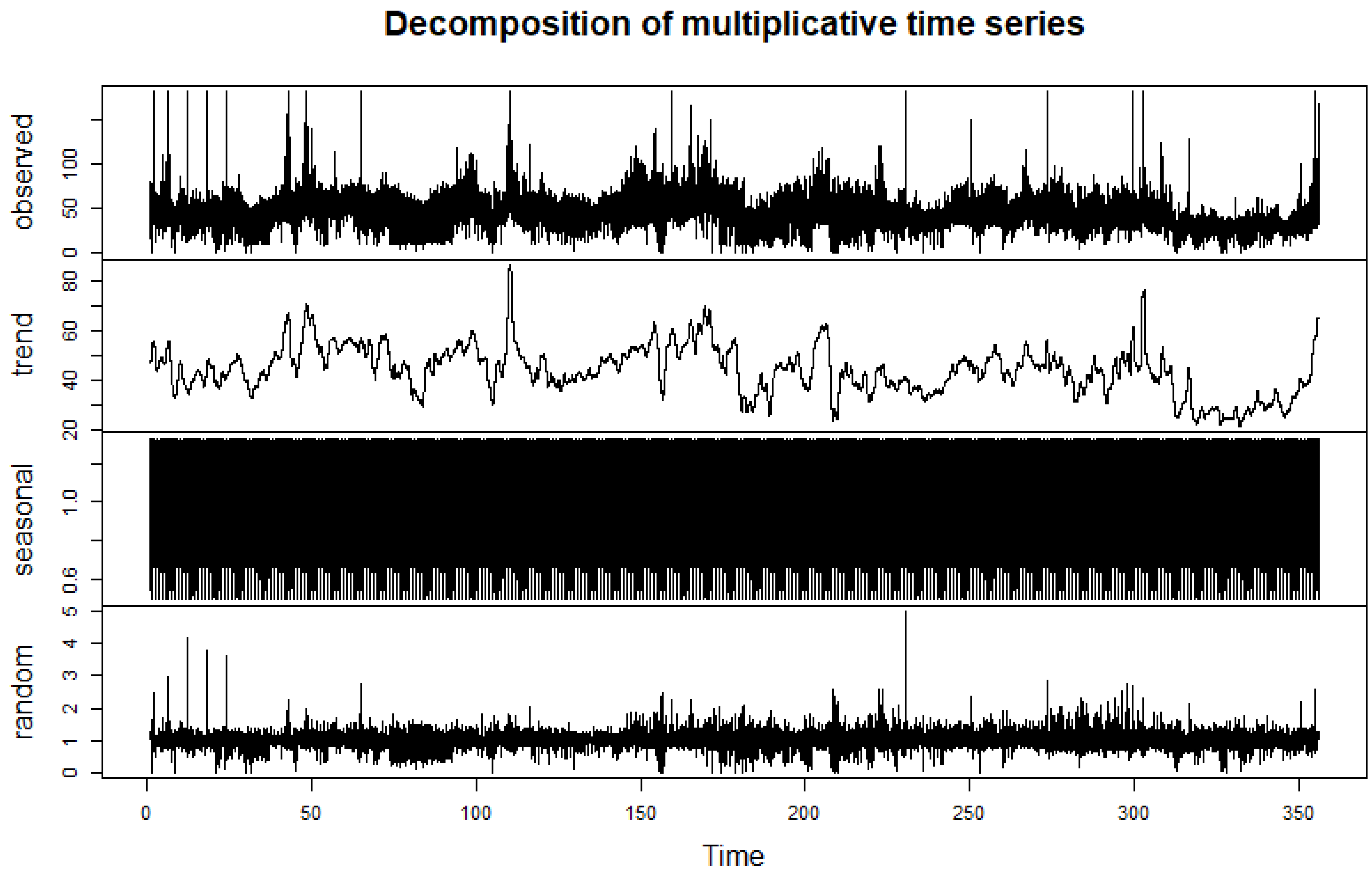
Validation of forecasting alternatives (2/15)

#Decomposition

```
dec <- decompose(insample, type = "multiplicative")
autoplot(head(dec$seasonal,168),ylab="Multiplicative Seasonality")
plot(dec)
```

The data is seasonal. But which is exactly the seasonal, hourly pattern?
Are seasonal adjustments more effective than seasonal models?

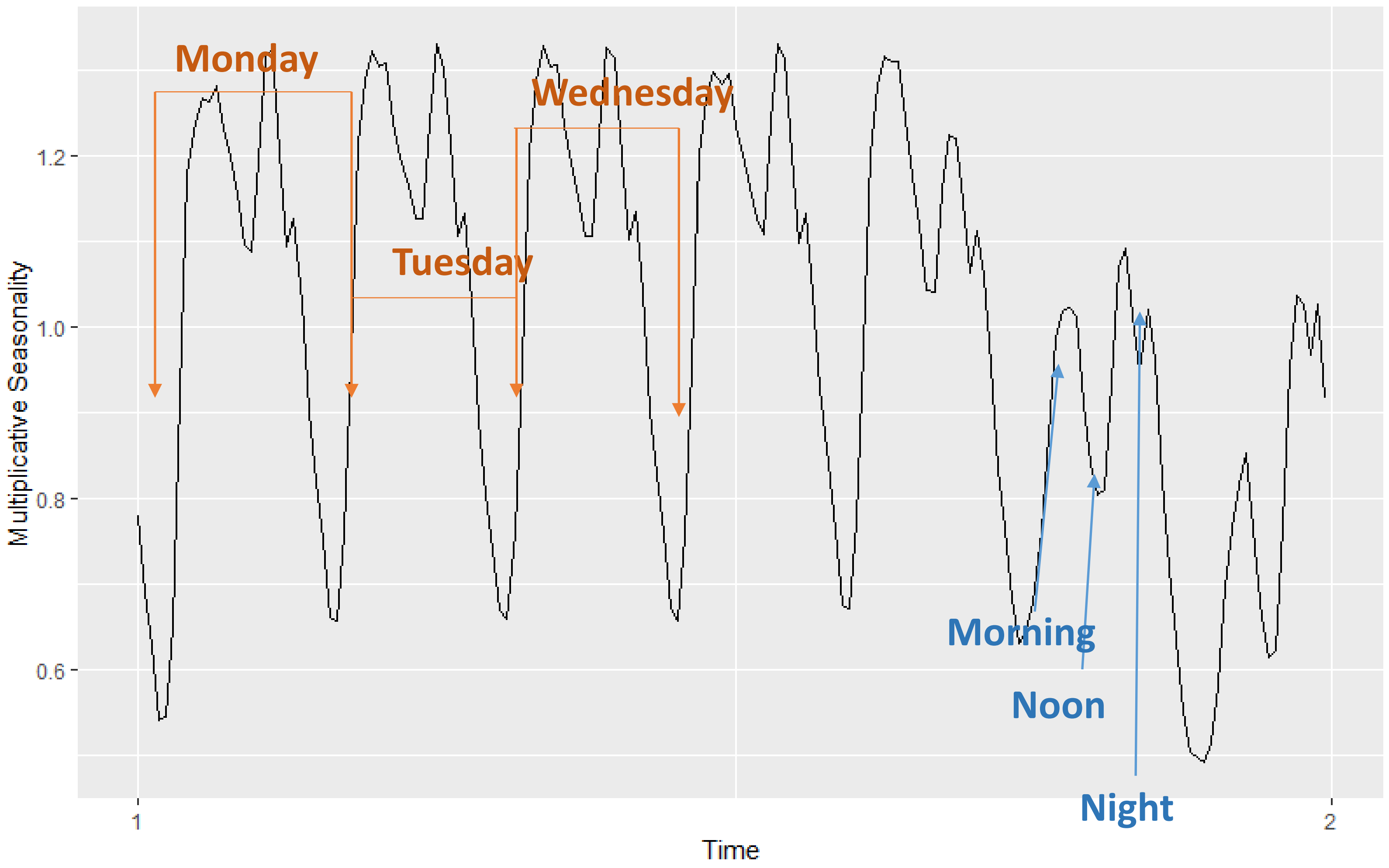
Validation of forecasting alternatives (3/15)



UNIC

MOFC EPU N T U A FORECASTING & STRATEGY UNIT

Validation of forecasting alternatives (4/15)



Validation of forecasting alternatives (5/15)

```
#Test various Forecasting Methods
Evaluation <- data.frame(matrix(NA, ncol = 1, nrow = 7))
row.names(Evaluation) <- c("Naive","SES","sNaive","SES_Mul","MLR","NN","Comb")
colnames(Evaluation) <- c("sMAPE")

#Naive
frc1 <- naive(insample,h=fh)$mean
Evaluation$sMAPE[1] <- mean(200*abs(outsample-frc1)/(abs(outsample)+abs(frc1)))

#SES - no decomposition
frc2 <- ses(insample,h=fh)$mean
Evaluation$sMAPE[2] <- mean(200*abs(outsample-frc2)/(abs(outsample)+abs(frc2)))

#Seasonal Naïve
frc3 <- as.numeric(tail(insample,fh)) + outsample - outsample
Evaluation$sMAPE[3] <- mean(200*abs(outsample-frc3)/(abs(outsample)+abs(frc3)))

#SES - with decomposition
Indexes_in <- decompose(insample, type = "multiplicative")$seasonal
Indexes_out <- as.numeric(tail(Indexes_in,fh))
frc4 <- ses(insample/Indexes_in,h=fh)$mean*Indexes_out
Evaluation$sMAPE[4] <- mean(200*abs(outsample-frc4)/(abs(outsample)+abs(frc4)))
```

Table of errors (sMAPE) per forecasting method

$mean extracts the out-of-sample forecasts
$fitted extracts the in-sample forecasts

How about ets() and auto.arima()?

Validation of forecasting alternatives (6/15)

#Inspect results

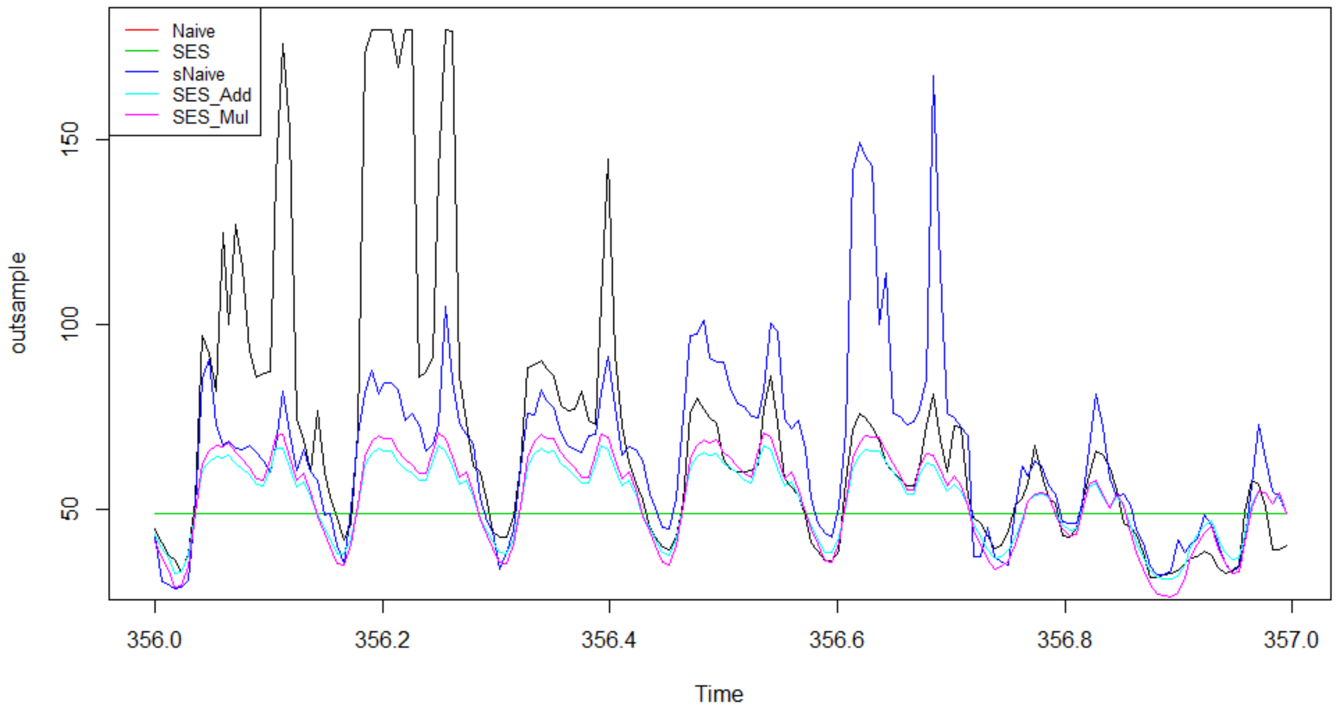
plot(outsample)

lines(frc1, col=2) ; lines(frc2, col=3)

lines(frc3, col=4) ; lines(frc4, col=5)

legend("topleft", legend=c("Naive","SES","sNaive","SES_Mul"),col=c(2:5), lty=1, cex=0.8)

Evaluation



	sMAPE
Naive	36.98777
SES	36.98746
sNaive	23.50712
SES_Mul	22.00467
MLR	NA
NN	NA
Comb	NA

UNIC

MOFC

Validation of forecasting alternatives (7/15)

```
#MLR
Data_ml <- train
Data_ml$Year <- factor(year(Data_ml$datetime_utc)) #Define Year
Data_ml$Month <- factor(month(Data_ml$datetime_utc)) #Define Month
Data_ml$DateType <- factor(Data_ml$DateType)
Data_ml$Hour <- factor(Data_ml$Hour)
Data_ml$Weekday <- 1
Data_ml[(Data_ml$DateType==1)|(Data_ml$DateType==7),]$Weekday <- 0
Data_ml$Weekday <- factor(Data_ml$Weekday)
Data_ml$Lag168 = Data_ml$Lag336 <- NA  #Define Level
Data_ml$Lag168 <- head(c(rep(NA,168), head(Data_ml,nrow(Data_ml)-168)$Prices.BE),nrow(Data_ml))
Data_ml$Lag336 <- head(c(rep(NA,336), head(Data_ml,nrow(Data_ml)-336)$Prices.BE),nrow(Data_ml))
Data_ml <- na.omit(Data_ml) #Delete NAs
insample_ml <- head(Data_ml,nrow(Data_ml)-fh) #in-sample for training
outsample_ml <- tail(Data_ml,fh) #out-of-sample for testing

#Inspect Correlations
library(corrplot)
corrplot(cor(insample_ml[,-c(1,6,7,8,9,10,11)]), method="color")
```

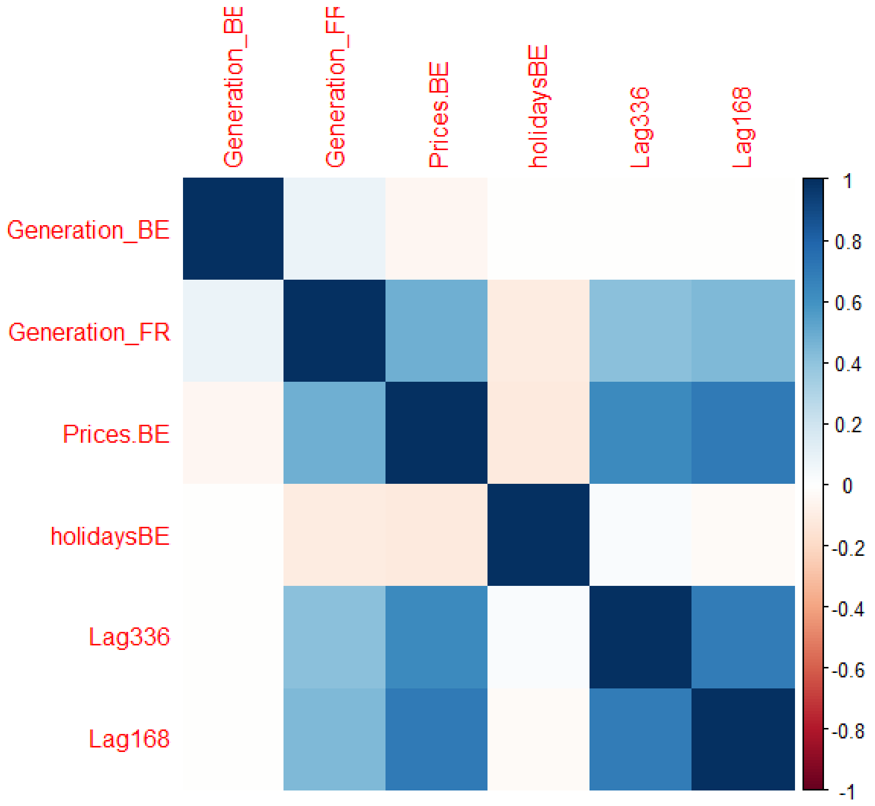
All categorical variables are transformed into factors so that they can be easily handled as so and not as numbers (*e.g. energy prices of month 12 are not 12 time bigger than those of month 1*)

Include explanatory variables such as

- Year
- Month
- Weekday or Weekend
- Lags (same case previous day, week etc.)

Lags are selected so that they capture both daily and weekly seasonal patterns. Moreover, they may depend on the lead-time or not (forecasts are computed recurrently)

Validation of forecasting alternatives (8/15)



Power generation of France and Lags
are highly correlated to energy prices

Validation of forecasting alternatives (9/15)

#Only Month

```
ml_model <- lm(Prices.BE~Month,data=insample_ml)
frc5_1 <- predict(ml_model,outsample_ml)
mean(200*abs(outsample_ml$Prices.BE-
frc5_1)/(abs(outsample_ml$Prices.BE)+abs(frc5_1)))
```

35.51%

#Only Month and Weekday

```
ml_model <- lm(Prices.BE~Month+DateType,data=insample_ml)
frc5_2 <- predict(ml_model,outsample_ml)
mean(200*abs(outsample_ml$Prices.BE-
frc5_2)/(abs(outsample_ml$Prices.BE)+abs(frc5_2)))
```

31.59%

#Only Month, Weekday and Hour

```
ml_model <- lm(Prices.BE~Month+Hour+DateType,data=insample_ml)
frc5_3 <- predict(ml_model,outsample_ml)
mean(200*abs(outsample_ml$Prices.BE-
frc5_3)/(abs(outsample_ml$Prices.BE)+abs(frc5_3)))
```

25.86%

#Only Month, Weekday, Hour and Holidays

```
ml_model <- lm(Prices.BE~Month+Hour+DateType+holidaysBE,data=insample_ml)
frc5_4 <- predict(ml_model,outsample_ml)
mean(200*abs(outsample_ml$Prices.BE-
frc5_4)/(abs(outsample_ml$Prices.BE)+abs(frc5_4)))
```

25.89%

#Only Month, Weekday, Hour and Generation

```
ml_model <- lm(Prices.BE~Month+Hour+DateType+Generation_BE,
data=insample_ml)
frc5_5_1 <- predict(ml_model,outsample_ml)
mean(200*abs(outsample_ml$Prices.BE-
frc5_5_1)/(abs(outsample_ml$Prices.BE)+abs(frc5_5_1)))
```

23.87%

```
ml_model <- lm(Prices.BE~Month+Hour+DateType+Generation_FR,
data=insample_ml)
frc5_5_2 <- predict(ml_model,outsample_ml)
mean(200*abs(outsample_ml$Prices.BE-
frc5_5_2)/(abs(outsample_ml$Prices.BE)+abs(frc5_5_2)))
```

31.67%

```
ml_model <- lm(Prices.BE~Month+Hour+DateType+Generation_BE+Generation_FR,
data=insample_ml)
frc5_5_3 <- predict(ml_model,outsample_ml)
mean(200*abs(outsample_ml$Prices.BE-
frc5_5_3)/(abs(outsample_ml$Prices.BE)+abs(frc5_5_3)))
```

28.64%

```
frc5_5 <- frc5_5_1
```

#Only Month, Weekday, Hour, BE Generation and Lags

```
ml_model <- lm(Prices.BE~Month+Hour+DateType+
Generation_BE+Lag168+Lag336,data=insample_ml)
frc5_6 <- predict(ml_model,outsample_ml)
mean(200*abs(outsample_ml$Prices.BE-
frc5_6)/(abs(outsample_ml$Prices.BE)+abs(frc5_6)))
```

19.82%

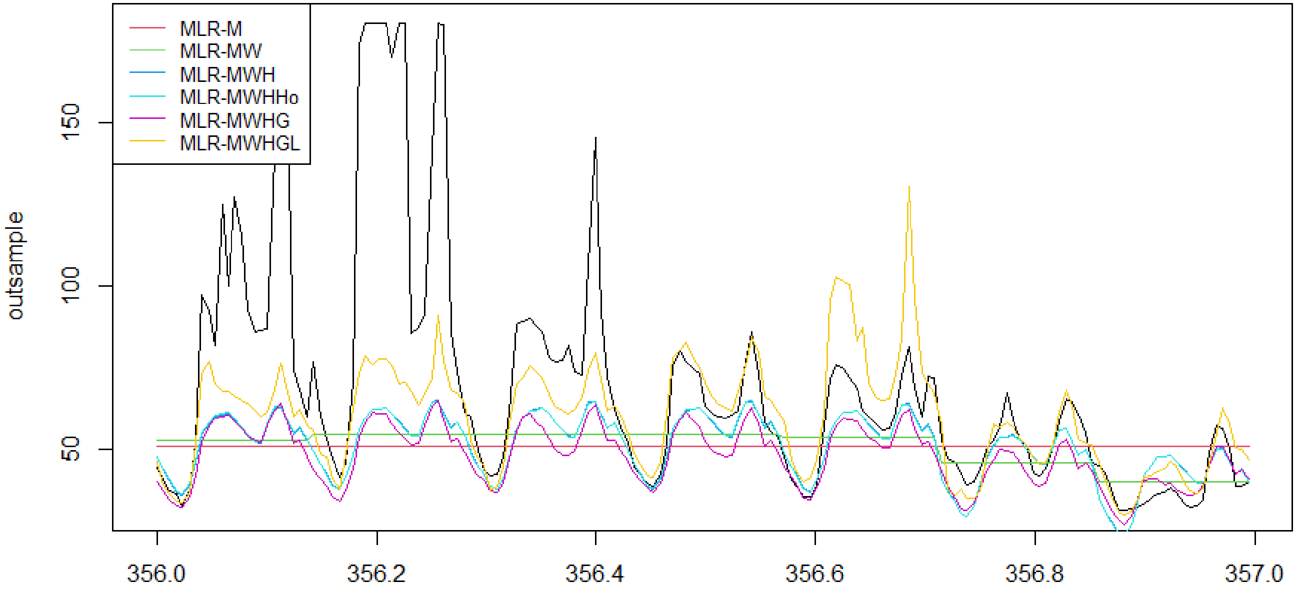
UNIC

MOFC

Validation of forecasting alternatives (10/15)

```
#Define final MLR
frc5 <- frc5_6
Evaluation$sMAPE[5] <- mean(200*abs(outsample-frc5)/(abs(outsample)+abs(frc5)))

#Inspect MLR
plot(outsample)
lines(frc5_1+outsample-outsample, col=2)
lines(frc5_2+outsample-outsample, col=3)
lines(frc5_3+outsample-outsample, col=4)
lines(frc5_4+outsample-outsample, col=5)
lines(frc5_5+outsample-outsample, col=6)
lines(frc5_6+outsample-outsample, col=7)
legend("topleft", legend=c("MLR-M","MLR-MW","MLR-MWH“,
"MLR-MWHHo","MLR-MWHG","MLR-MWHGL"),
col=c(2:7), lty=1, cex=0.8)
Evaluation
```

	sMAPE
Naive	36.98777
SES	36.98746
sNaive	23.50712
SES_Mul	22.00467
MLR	19.81739
NN	NA
Comb	NA



Validation of forecasting alternatives (11/15)

```
#NN
ForScaling <- rbind(insample_ml,outsample_ml)[,c("Generation_BE","Lag168","Lag336",
                                   "Month","Hour","DateType")]
ForScaling$Generation_BE <- ((ForScaling$Generation_BE - min(insample_ml$Generation_BE)) / (max(insample_ml$Generation_BE) - min(insample_ml$Generation_BE)))
ForScaling$Lag168 <- ((ForScaling$Lag168 - min(insample_ml$Lag168)) / (max(insample_ml$Lag168) - min(insample_ml$Lag168)))
ForScaling$Lag336 <- ((ForScaling$Lag336 - min(insample_ml$Lag336)) / (max(insample_ml$Lag336) - min(insample_ml$Lag336)))
```

Scale data from 0 to 1

```
#Create dummy variables
library(caret)
dummy <- dummyVars(" ~ .", data=ForScaling[,c("Month","Hour","DateType")])
dummy <- data.frame(predict(dummy, newdata = ForScaling[,c("Month","Hour","DateType")]))
dummy$Month.1= dummy$Hour.0 = dummy$DateType.1 <- NULL
ForScaling$Month = ForScaling$Hour = ForScaling$DateType <- NULL
ForScaling <- cbind(ForScaling, dummy)
```

Create dummy variables and drop the ones that lead to multi-linearities

```
trainNN_x <- head(ForScaling, nrow(ForScaling)-fh)
trainNN_y <- ((insample_ml$Prices.BE - min(insample_ml$Prices.BE)) / (max(insample_ml$Prices.BE) - min(insample_ml$Prices.BE)))
testNN_x <- tail(ForScaling, fh)
testNN_y <- outsample_ml$Prices.BE
```

Data used for training and testing

Validation of forecasting alternatives (12/15)

```
#Single layer
set.seed(101)
model1<-mlp(trainNN_x, trainNN_y,
        size = 20, maxit = 100,initFunc = "Randomize_Weights",
        learnFunc = "BackpropWeightDecay", hiddenActFunc = "Act_Logistic",
        shufflePatterns = FALSE, linOut = FALSE)
frc6_1 <- as.numeric(predict(model1,testNN_x))*(max(insample_ml$Prices.BE) - min(insample_ml$Prices.BE)) + min(insample_ml$Prices.BE)
mean(200*abs(testNN_y-frc6_1)/(abs(testNN_y)+abs(frc6_1)))
```

19.95%

```
set.seed(101)
model2<-mlp(trainNN_x, trainNN_y,
        size = c(20,15,10,5), maxit = 100,initFunc = "Randomize_Weights",
        learnFunc = "BackpropWeightDecay", hiddenActFunc = "Act_Logistic",
        shufflePatterns = FALSE, linOut = FALSE)
frc6_2 <- as.numeric(predict(model2,testNN_x))*(max(insample_ml$Prices.BE) - min(insample_ml$Prices.BE)) + min(insample_ml$Prices.BE)
mean(200*abs(testNN_y-frc6_2)/(abs(testNN_y)+abs(frc6_2)))
```

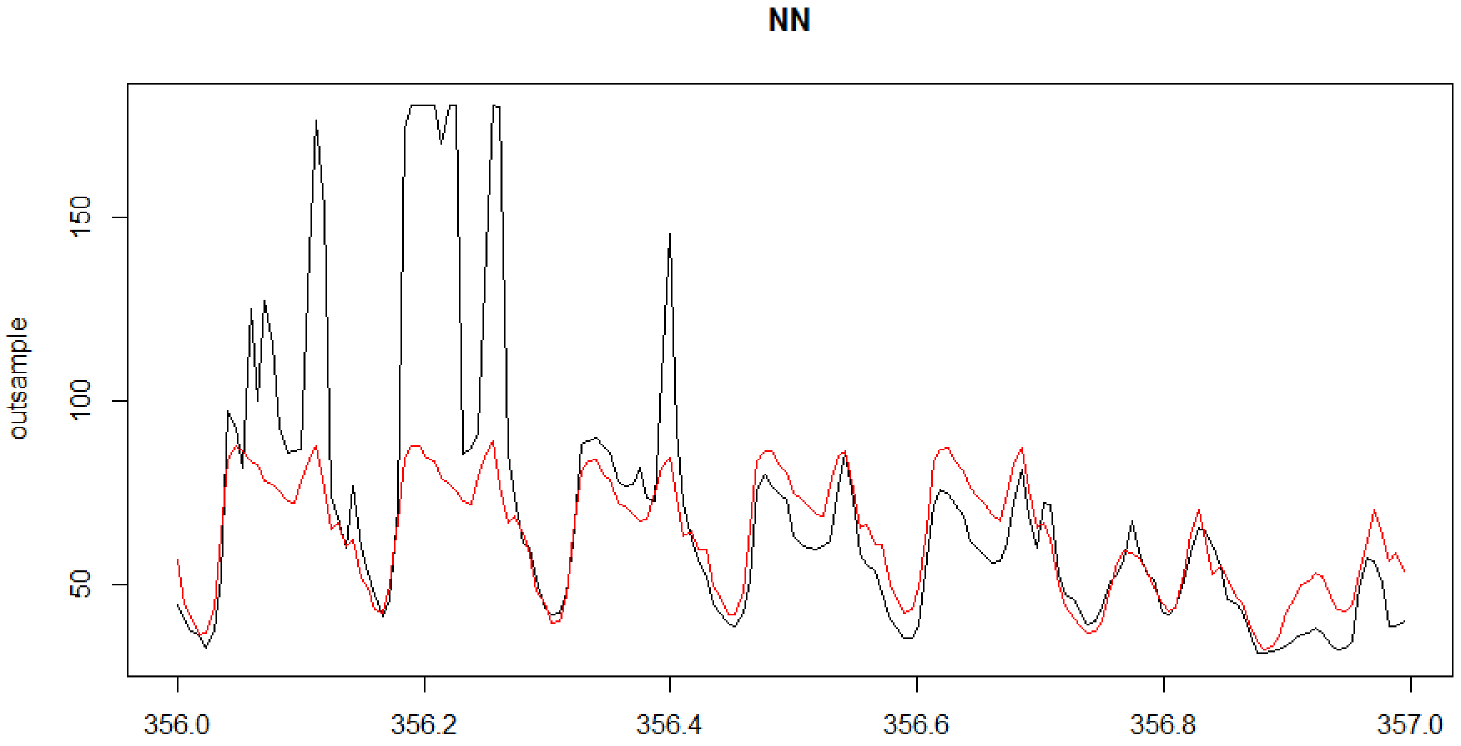
18.13%

```
frc6 <- frc6_2

plot(outsample,type="l", main="NN")
lines(frc6+outsample-outsample, col="red",type="l")
Evaluation$sMAPE[6] <- mean(200*abs(outsample-frc6)/(abs(outsample)+abs(frc6)))
```

UNIC

MOFC

Validation of forecasting alternatives (13/15)



UNIC

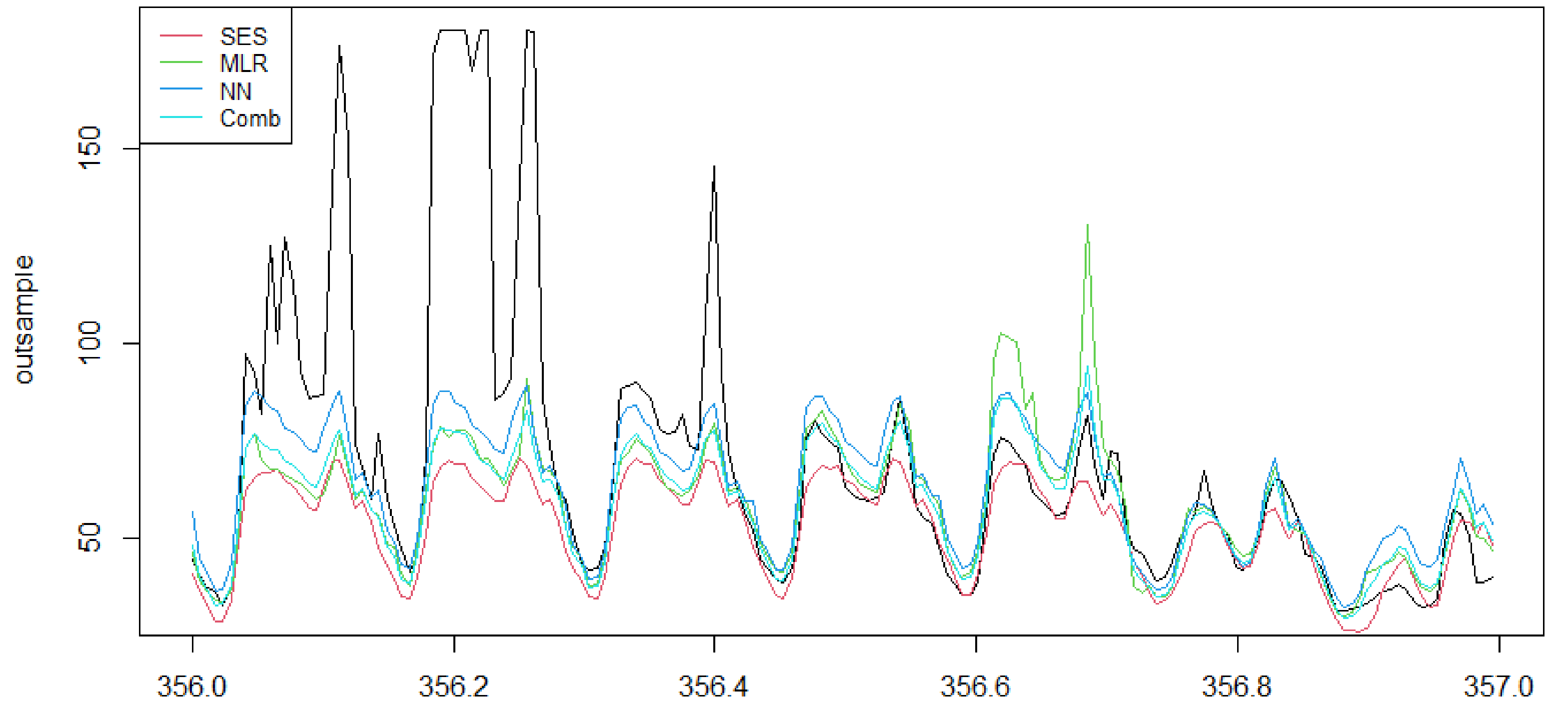
Validation of forecasting alternatives (14/15)

```
frc7 <- (as.numeric(frc4)+frc5+frc6)/3
Evaluation$sMAPE[7] <- mean(200*abs(outsample-frc7)/(abs(outsample)+abs(frc7)))
plot(outsample,type="l")
lines(frc4+outsample-outsample, col=2,type="l")
lines(frc5+outsample-outsample, col=3,type="l")
lines(frc6+outsample-outsample, col=4,type="l")
lines(frc7+outsample-outsample, col=5,type="l")
legend("topleft", legend=c("SES","MLR","NN","Comb"),col=c(2:5), lty=1, cex=0.8)
```

UNIC

MOFC

Validation of forecasting alternatives (15/15)

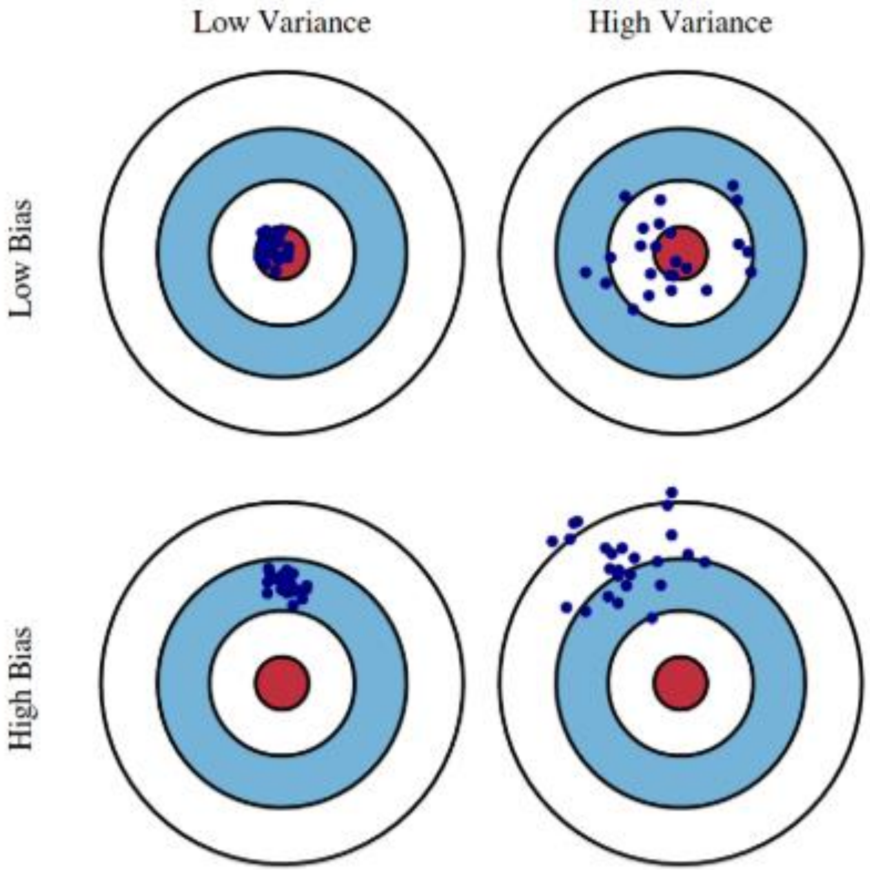


	sMAPE
Naive	36.98777
SES	36.98746
sNaive	23.50712
SES_Mul	22.00467
MLR	19.81739
NN	18.13236
Comb	18.12837

UNIC

MOFC

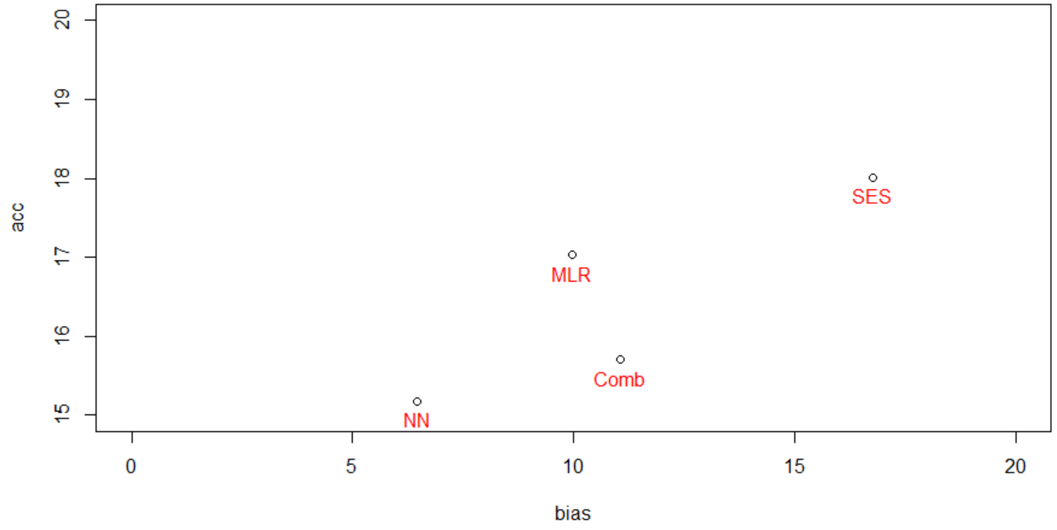
Experimentation



Experimentation

#Explain the effect of comb

```
ME1 <- outsample-frc4
ME2 <- outsample-frc5
ME3 <- outsample-frc6
ME4 <- outsample-frc7
bias <- c(mean(ME1),mean(ME2),mean(ME3),mean(ME4))
acc <- c(mean(abs(ME1)),mean(abs(ME2)),mean(abs(ME3)),mean(abs(ME4)))
plot(bias,acc, ylim=c(15,20), xlim=c(0,20))
text(bias, acc, c("SES","MLR","NN","Comb"), col="red",pos=1)
```



Further experimentation: Models

Models:
- Support Vector Machines
- Decision trees
- Random Forests
- Gradient boosting trees

Different models make different assumptions about the data and help us diversify our final forecasts

Optimize:
- Hyper-parameters
- Architecture
- Inputs (select or create additional ones)

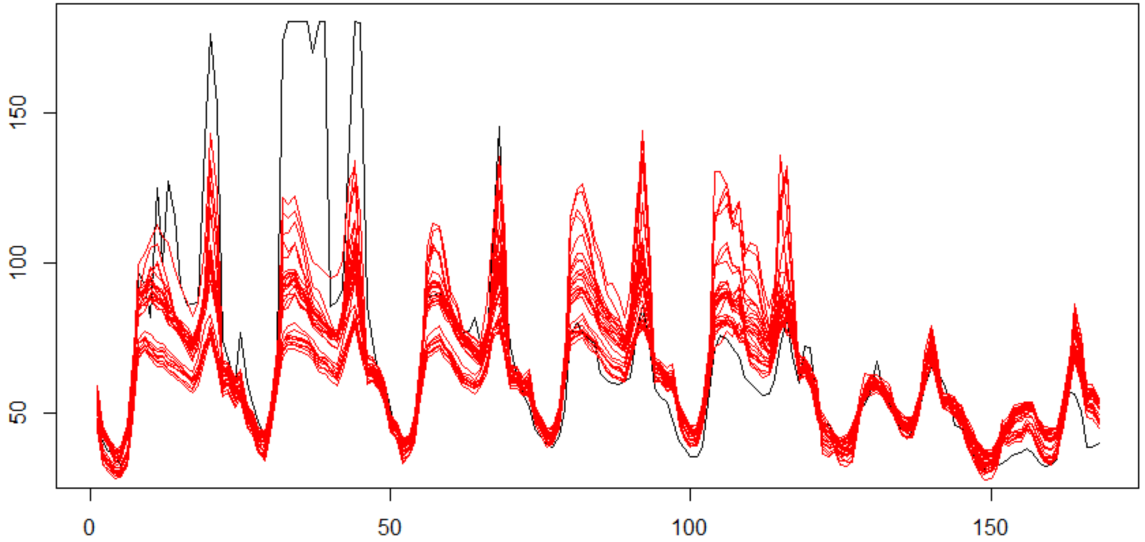
Depending on the data, different "versions" of the models may be more appropriate – Feature engineering and selection is also important

Ensembles of multiple models and variations of them

Most ML methods are strongly affected by their random initializations, so suing ensembles helps us stabilize their performance

Further experimentation: Models

```
#Ensembles
frc_ens <- NULL
plot(as.numeric(outsample),type="l")
for (i in 1:30){
  set.seed(i)
  model2<-mlp(trainNN_x, trainNN_y,
              size = c(20,15,10,5), maxit = round(runif(1,100,300)),initFunc = "Randomize_Weights",
              learnFunc = "BackpropWeightDecay", hiddenActFunc = "Act_Logistic",
              shufflePatterns = round(runif(1,0,1)), linOut = round(runif(1,0,1)))
  frc_t <- as.numeric(predict(model2,testNN_x))*(max(insample_ml$Prices.BE) - min(insample_ml$Prices.BE)) + min(insample_ml$Prices.BE)
  lines(frc_t, col="red")
  frc_ens <- rbind(frc_ens, frc_t)
}

library(robustbase)
frc_ens_m <- colMeans(frc_ens)
frc_ens_md <- colMedians(frc_ens)
mean(200*abs(testNN_y-frc_ens_m)/(abs(testNN_y)+abs(frc_ens_m)))
mean(200*abs(testNN_y-frc_ens_md)/(abs(testNN_y)+abs(frc_ens_md)))
```



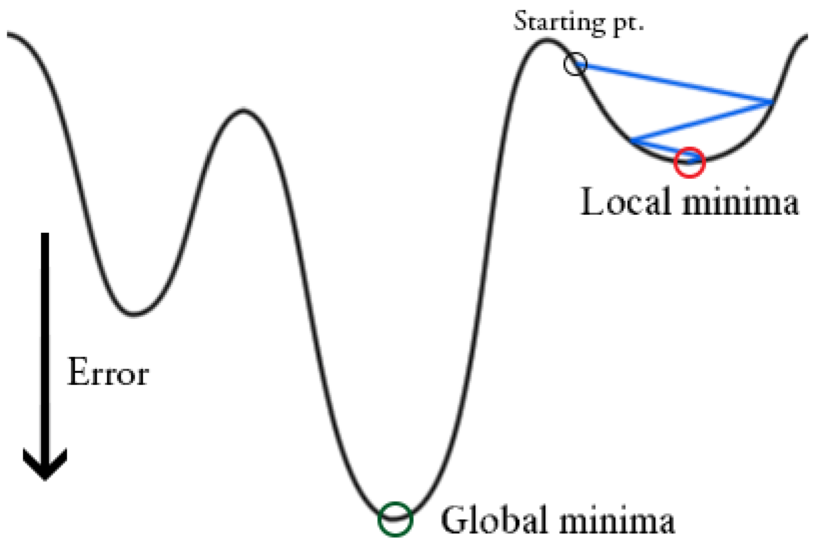
18.39% vs. 17.87%

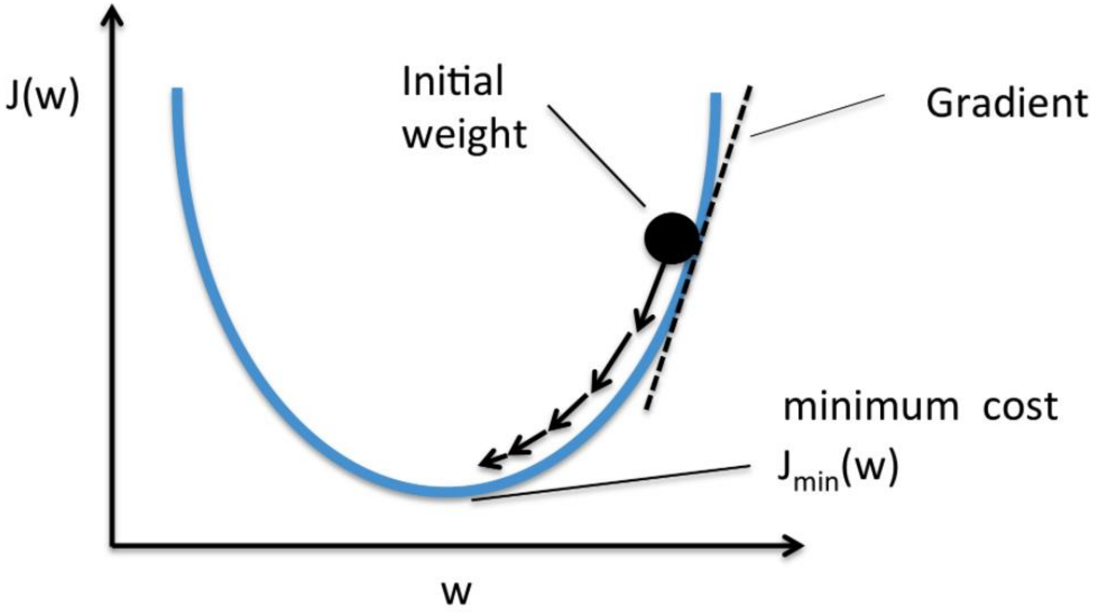
UNIC

MOFC

Further experimentation: Cross-Validation

Further experimentation: Multiple models and trainings





UNIVERSITY *of* NICOSIA