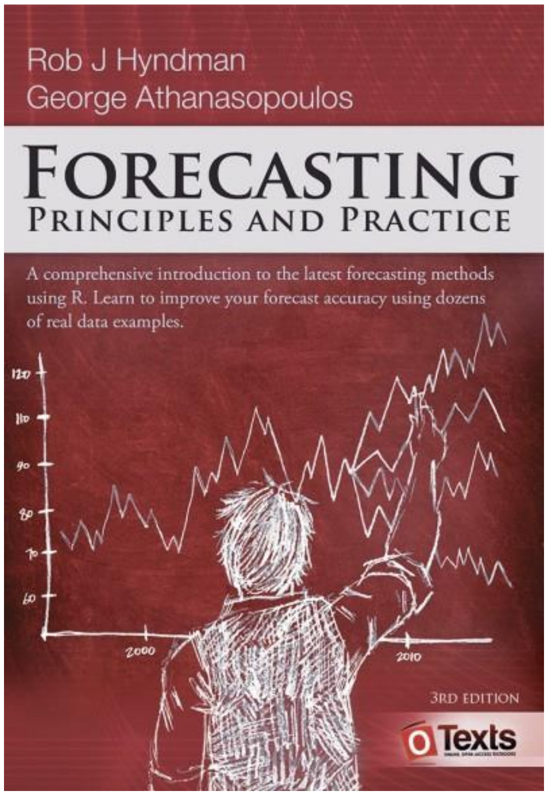
BI Workplace T03
Predictive Analytics

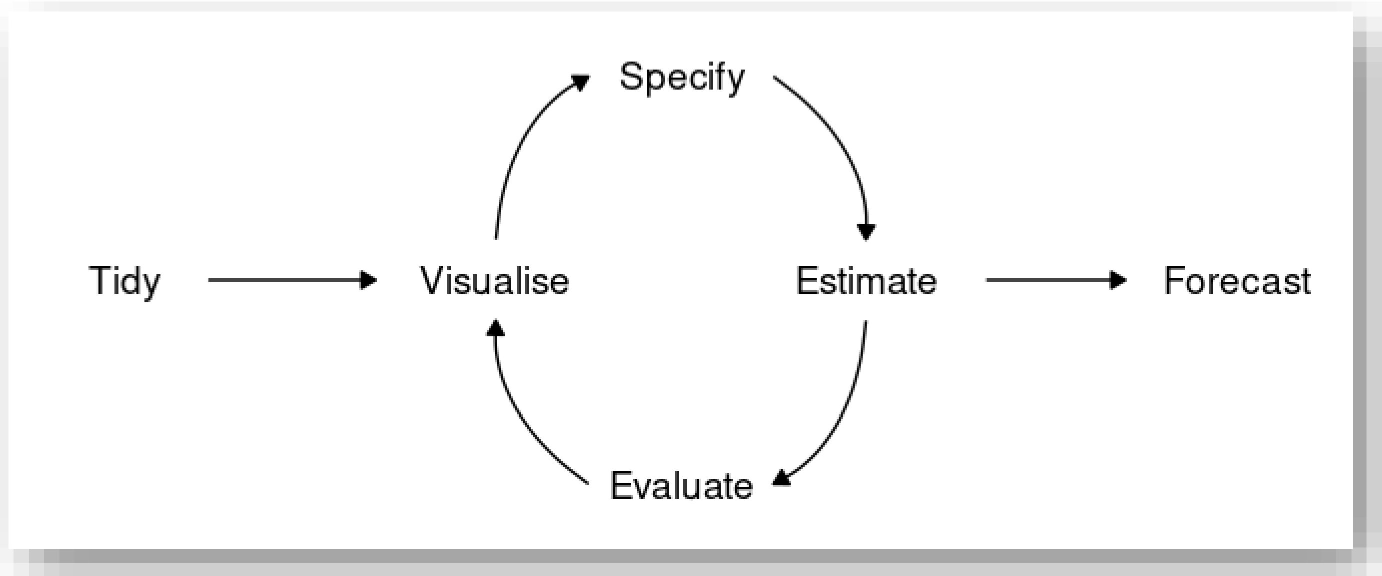
Sec. 5: The forecaster's toolbox



https://otexts.com/fpp3

Forecasting workflow

1. Prepare data
2. Visualize data
3. Specify the model
4. Estimation the model
5. Evaluate performance
6. Produce forecasts



#1 Tidy (Preparing data)

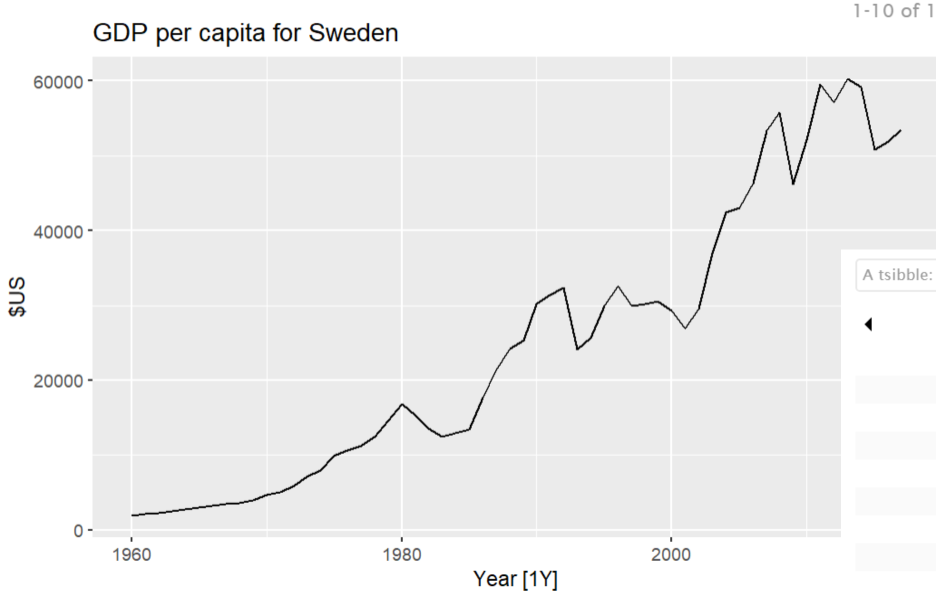
- May involve:
  - Loading the data
  - Identifying missing values
  - Filtering the time series
  - Transforming the variables

```
{r}
# Load GDP data and compute GDP per capita
gdppc <- global_economy %>%
  mutate(GDP_per_capita = GDP / Population)

gdppc

# write.csv(gdppc,"gdppc.csv", row.names = FALSE)
```

- Check your data before estimating models!

#2 Visualize

```
{r}
# Plot the data (visualize)
gdppc %>%
  filter(Country == "Sweden") %>%
  autoplot(GDP_per_capita) +
  labs(y = "$US", title = "GDP per capita for Sweden")
```



A tsibble: 15,150 x 10 [1Y] Key: Country [263]

Country <fctr>	Code <fctr>	Year <dbl>	GDP <dbl>	Growth <dbl>	CPI <dbl>
Afghanistan	AFG	1960	5.377778e+...	NA	NA
Afghanistan	AFG	1961	5.488889e+...	NA	NA
Afghanistan	AFG	1962	5.466667e+...	NA	NA
Afghanistan	AFG	1963	7.511112e+...	NA	NA
Afghanistan	AFG	1964	8.000000e+...	NA	NA
Afghanistan	AFG	1965	1.006667e+...	NA	NA
Afghanistan	AFG	1966	1.400000e+...	NA	NA
Afghanistan	AFG	1967	1.673333e+...	NA	NA
Afghanistan	AFG	1968	1.373333e+...	NA	NA
Afghanistan	AFG	1969	1.408889e+...	NA	NA

1-10 of 15,150 rows | 1-6 of 10 columns   Previous 1 2 3 4 5 6 ... 100 Next

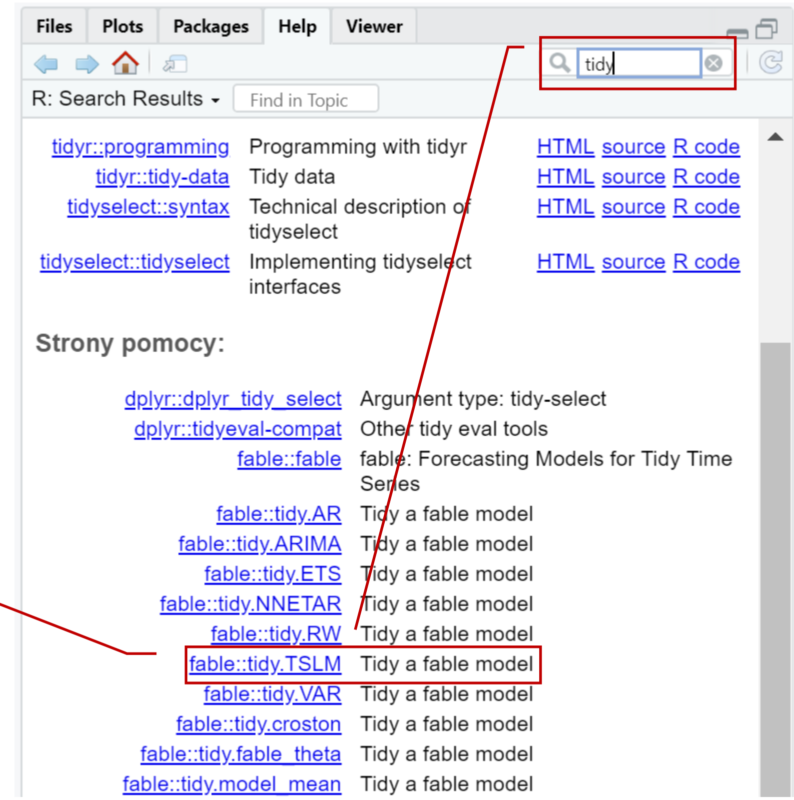
A tsibble: 15,150 x 10 [1Y] Key: Country [263]

CPI <dbl>	Imports <dbl>	Exports <dbl>	Population <dbl>	GDP_per_capita <dbl>
NA	7.024793	4.132233	8996351	59.77733
NA	8.097166	4.453443	9166764	59.87815
NA	9.349593	4.878051	9345868	58.49287
NA	16.863910	9.171601	9533954	78.78276
NA	18.055555	8.888893	9731361	82.20844
NA	21.412803	11.258279	9938414	101.29047
NA	18.571429	8.571429	10152331	137.89936
NA	14.209827	6.772908	10372630	161.32200
NA	15.210356	8.899677	10604346	129.50665
NA	14.984227	10.094637	10854428	129.79854

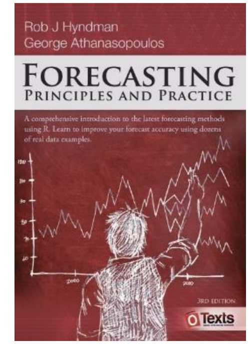
1-10 of 15,150 rows | 6-10 of 10 columns   Previous 1 2 3 4 5 6 ... 100 Next

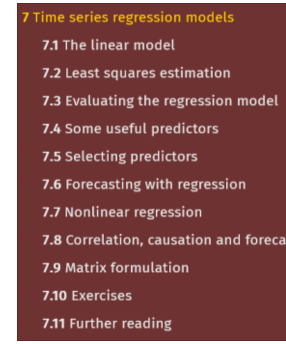
#3-4 Specify and train a model

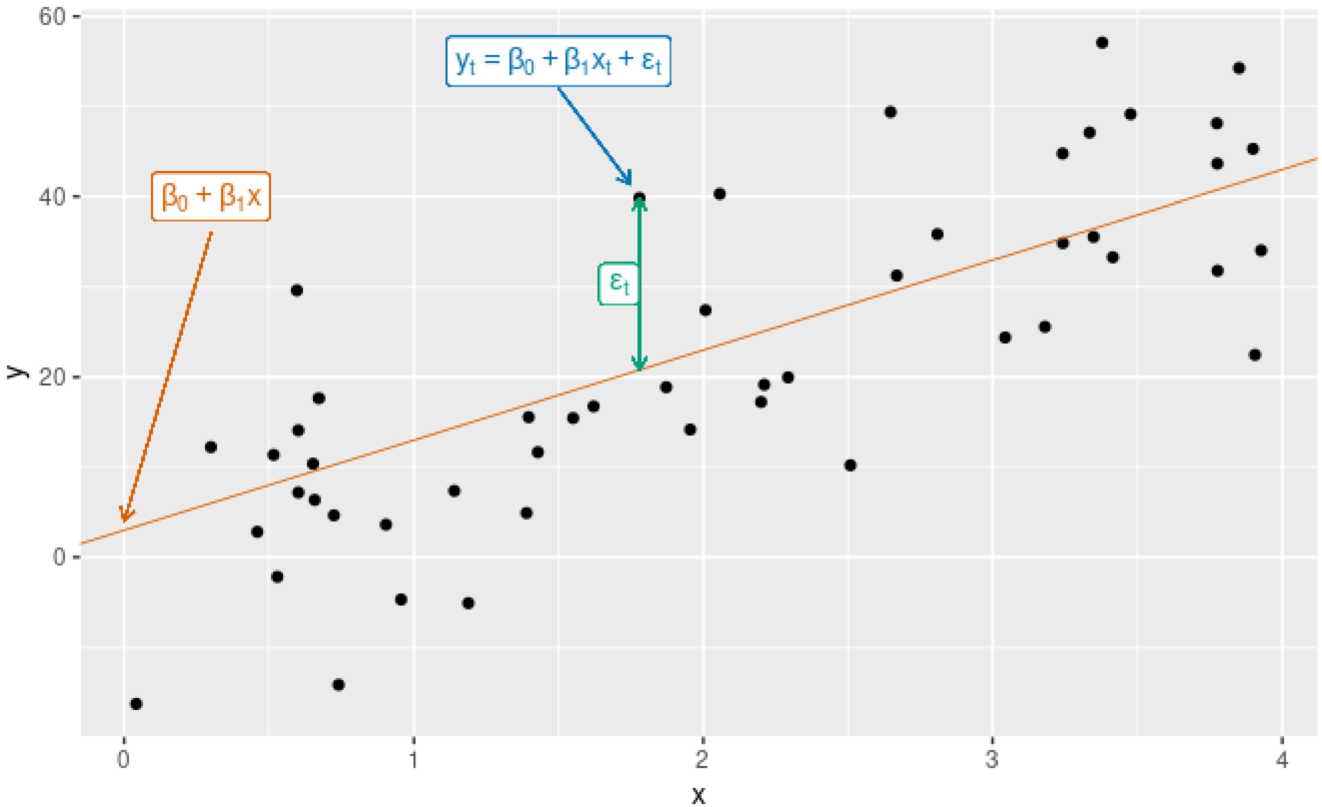
- Models in **fable** (‘forecast table’)
  - Use a formula interface: **y ~ x**
  - **y** – response variable(s)
  - **x** – model structure
- E.g., a linear trend model for GDP per capita:
  - TSLM(GDP_per_capita ~ trend())



Linear regression







#4-6 Train & predict

#3-4 Specify and train (estimate) a Time Series Linear (regression) Model

```{r}
fit <- gdppc %>%
  model(trend_model = TSLM(GDP_per_capita ~ trend()))

fit
```

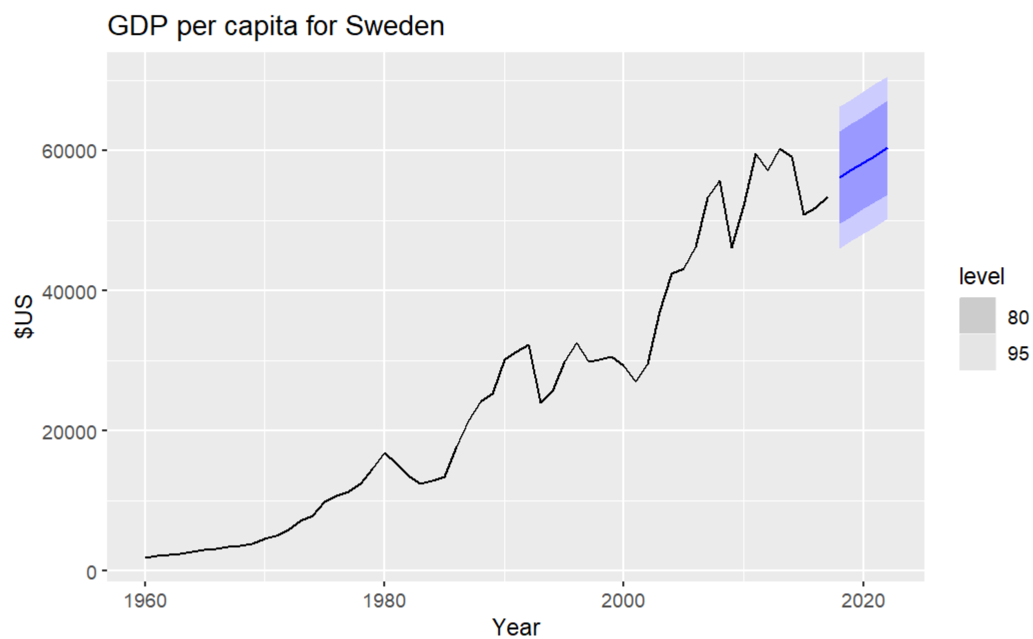
A mable: 263 x 2 Key: Country [263]

Country <fctr>	trend_model <S3: lst_mdl>
Afghanistan	<S3: lst_mdl>
Albania	<S3: lst_mdl>
Algeria	<S3: lst_mdl>
American Samoa	<S3: lst_mdl>
Andorra	<S3: lst_mdl>
Angola	<S3: lst_mdl>
Antigua and Barbuda	<S3: lst_mdl>
Arab World	<S3: lst_mdl>
Argentina	<S3: lst_mdl>
Armenia	<S3: lst_mdl>

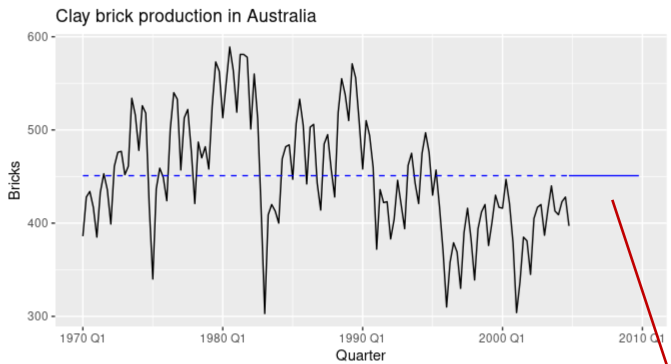
1-10 of 263 rows Previous 1 2

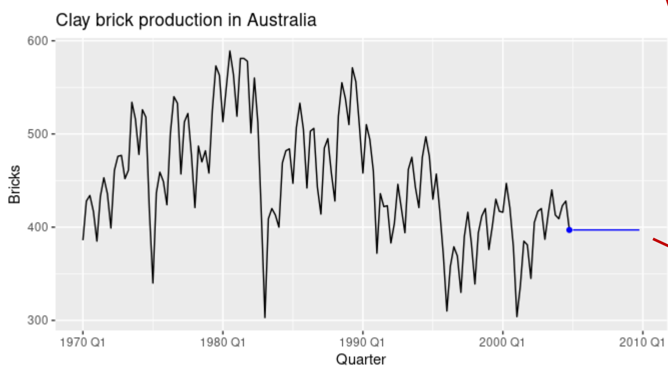
#6 Produce and visualize the forecasts

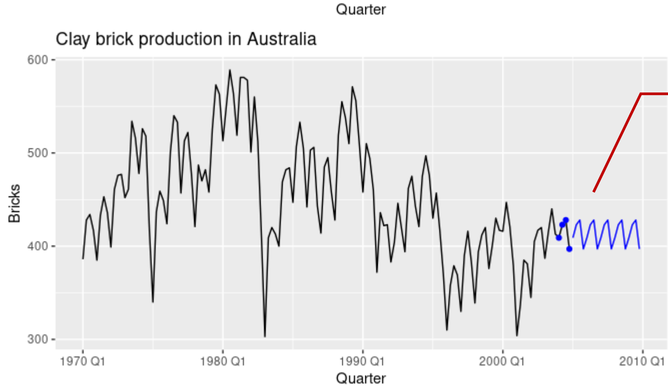
```{r}
fit %>%
  forecast(h = "5 years") %>%
  filter(Country == "Sweden") %>%
  autoplot(gdppc) +
  labs(y = "$US", title = "GDP per capita for Sweden")
```



Simple forecasting methods








```
> bricks <- aus_production %>%
    filter_index("1970 Q1" ~ "2004 Q4") %>%
    select(Bricks)
```

Mean forecast

> bricks %>% model(**MEAN**(Bricks))

Naïve forecast

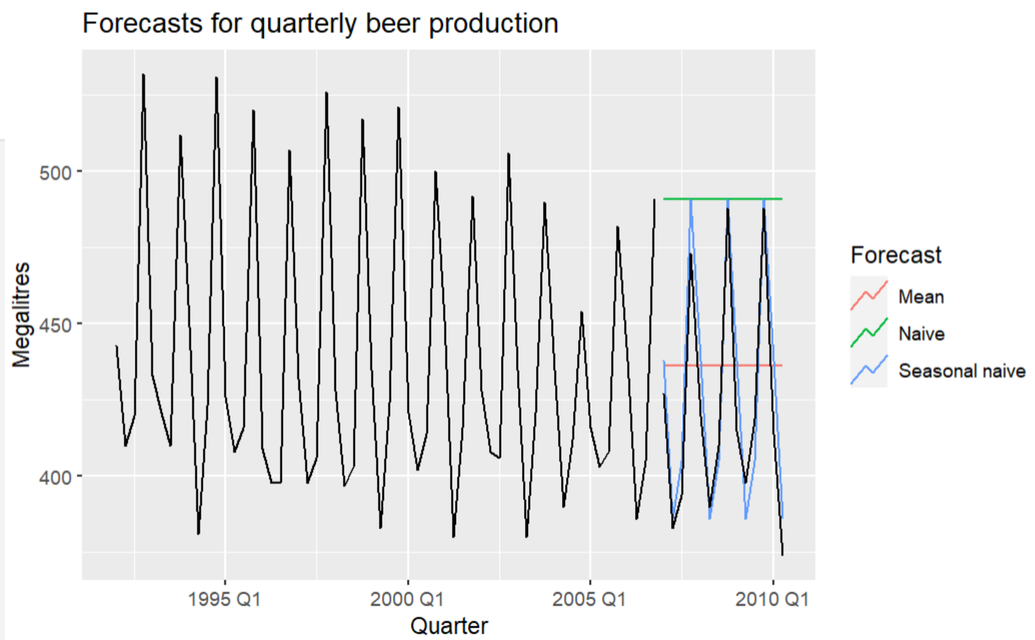
> bricks %>% model(**NAIVE**(Bricks))

Seasonal naïve

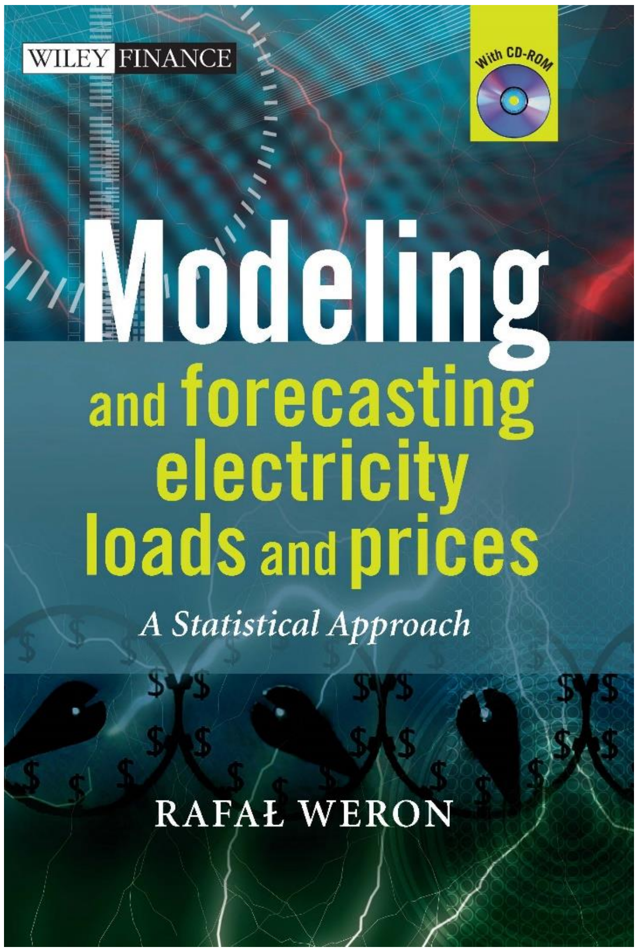
> bricks %>% model(**SNAIVE**(Bricks ~ lag("year")))

#5 Evaluate performance

```r
{r}
# Set training data from 1992 to 2006
train <- aus_production %>%
  filter_index("1992 Q1" ~ "2006 Q4")
# Fit the models
beer_fit <- train %>%
  model(
    Mean = MEAN(Beer),
    `Naïve` = NAIVE(Beer),
    `Seasonal naïve` = SNAIVE(Beer)
  )
# Generate forecasts for 14 quarters
beer_fc <- beer_fit %>% forecast(h = 14)
# Plot forecasts against actual values
beer_fc %>%
  autoplot(train, level = NULL) +
  autolayer(
    filter_index(aus_production, "2007 Q1" ~ .),
    colour = "black"
  ) +
  labs(
    y = "Megalitres",
    title = "Forecasts for quarterly beer production"
  ) +
  guides(colour = guide_legend(title = "Forecast"))
```



Intro to project P02




International Journal of Forecasting 30 (2014) 1030–1081

Contents lists available at ScienceDirect

International Journal of Forecasting

journal homepage: www.elsevier.com/locate/ijforecast

Review

Electricity price forecasting: A review of the state-of-the-art with a look into the future

Rafał Weron

Institute of Organization and Management, Wrocław University of Technology, Wrocław, Poland

ARTICLE INFO

Keywords:
Electricity price forecasting
Day-ahead market
Seasonality
Autoregression
Neural network
Factor model
Forecast combination
Probabilistic forecast

ABSTRACT

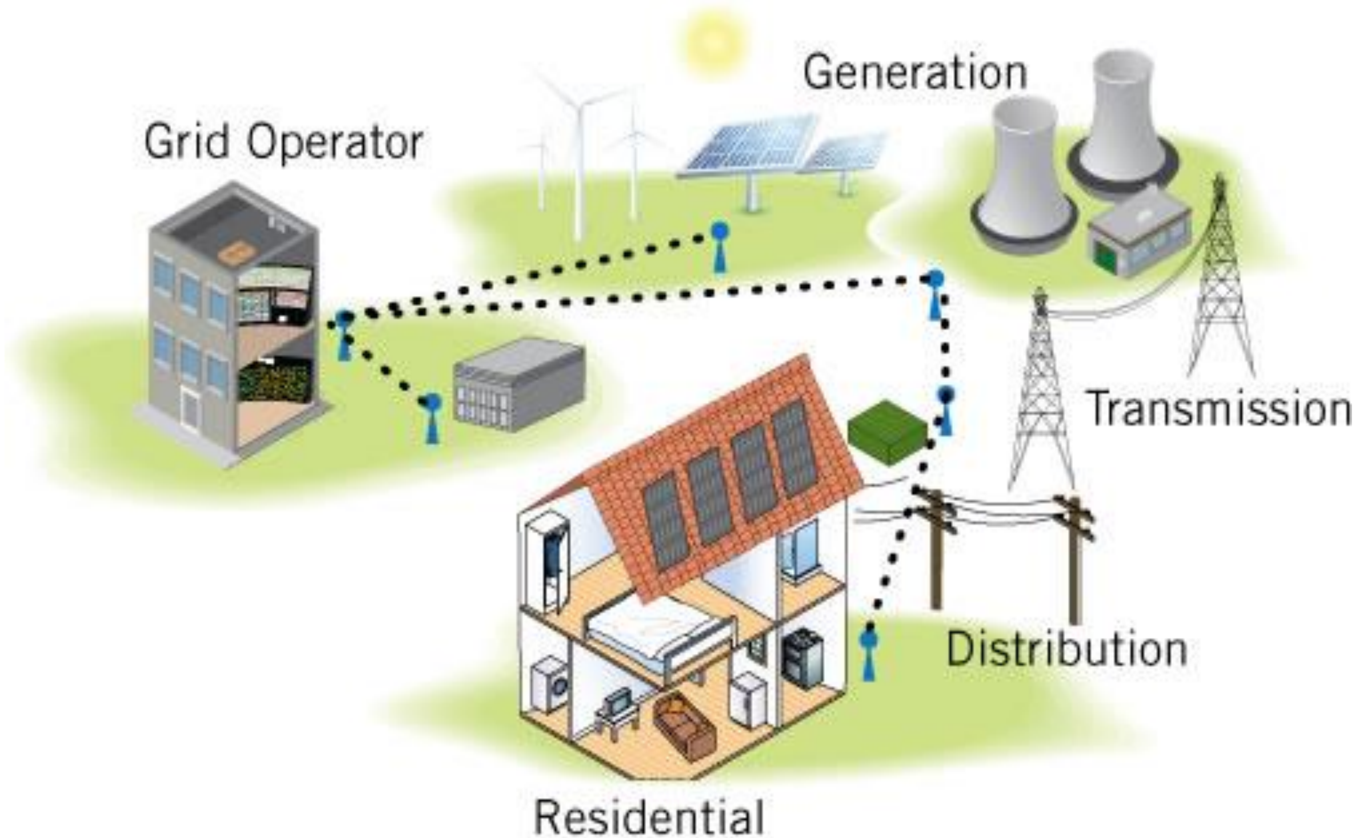
A variety of methods and ideas have been tried for *electricity price forecasting* (EPF) over the last 15 years, with varying degrees of success. This review article aims to explain the complexity of available solutions, their strengths and weaknesses, and the opportunities and threats that the forecasting tools offer or that may be encountered. The paper also looks ahead and speculates on the directions EPF will or should take in the next decade or so. In particular, it postulates the need for objective comparative EPF studies involving (i) the same datasets, (ii) the same robust error evaluation procedures, and (iii) statistical testing of the significance of one model's outperformance of another.

Contents

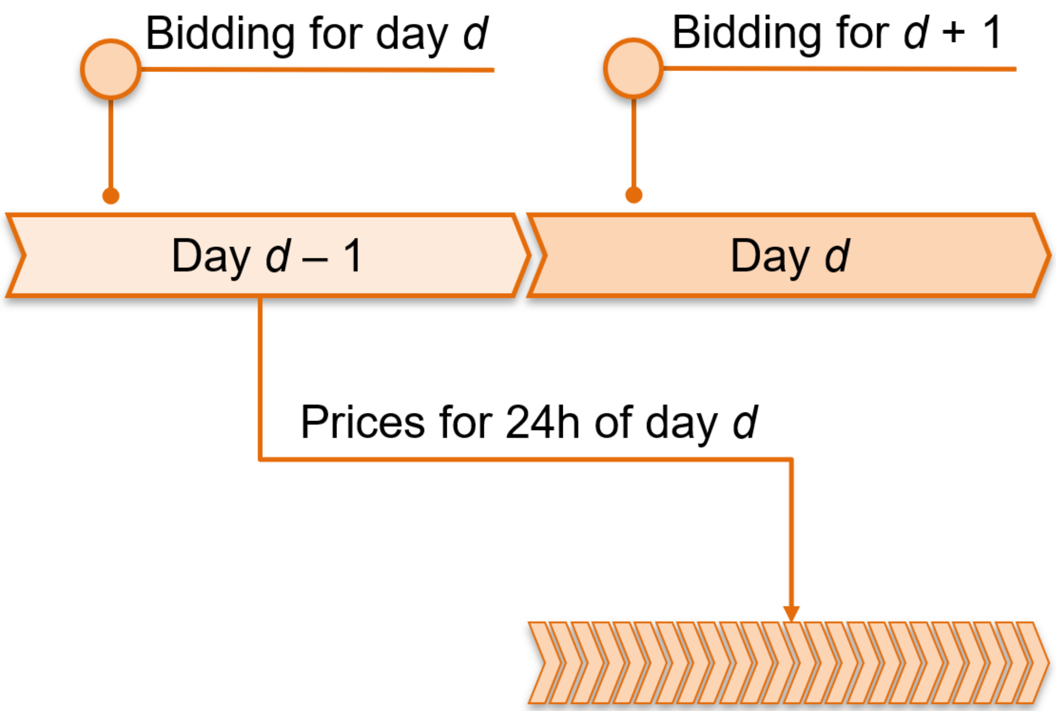
E-mail address: rafal.weron@pwr.wroc.pl.

http://dx.doi.org/10.1016/j.ijforecast.2014.08.008

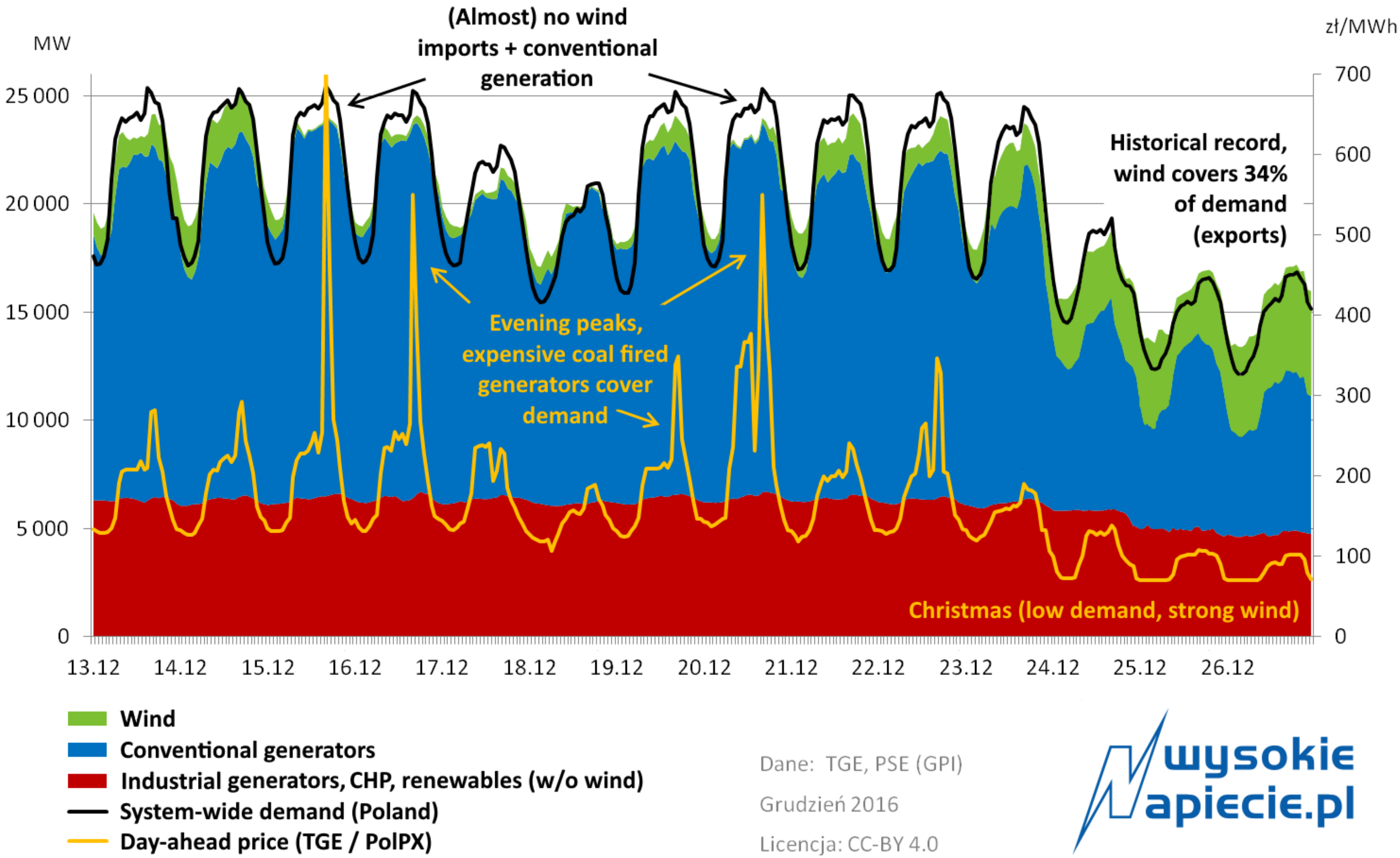
The power system



The day-ahead market for electricity



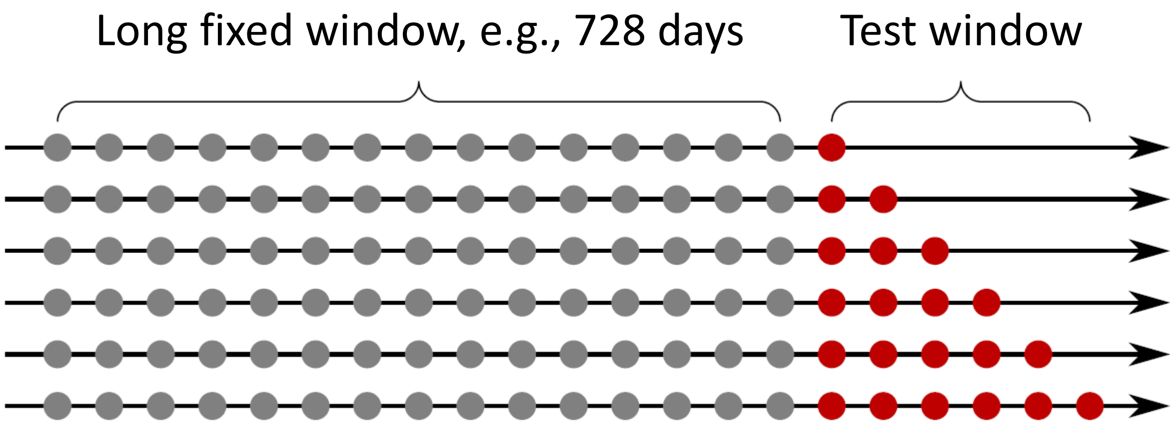
Wholesale electricity prices

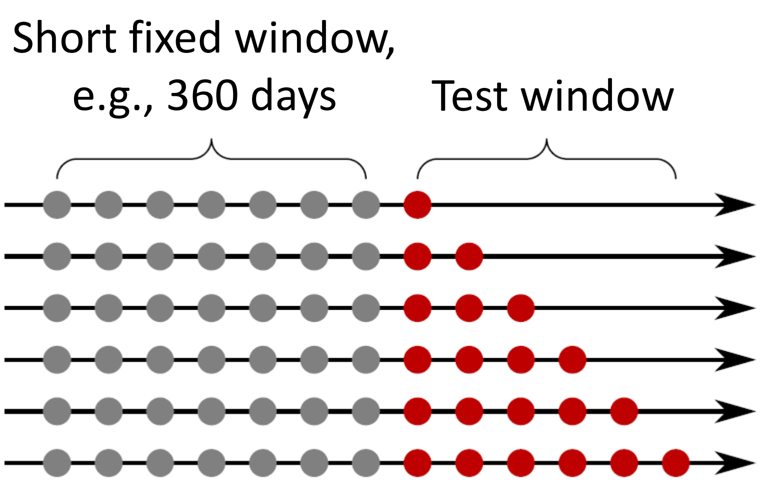


Seasonal naïve forecast

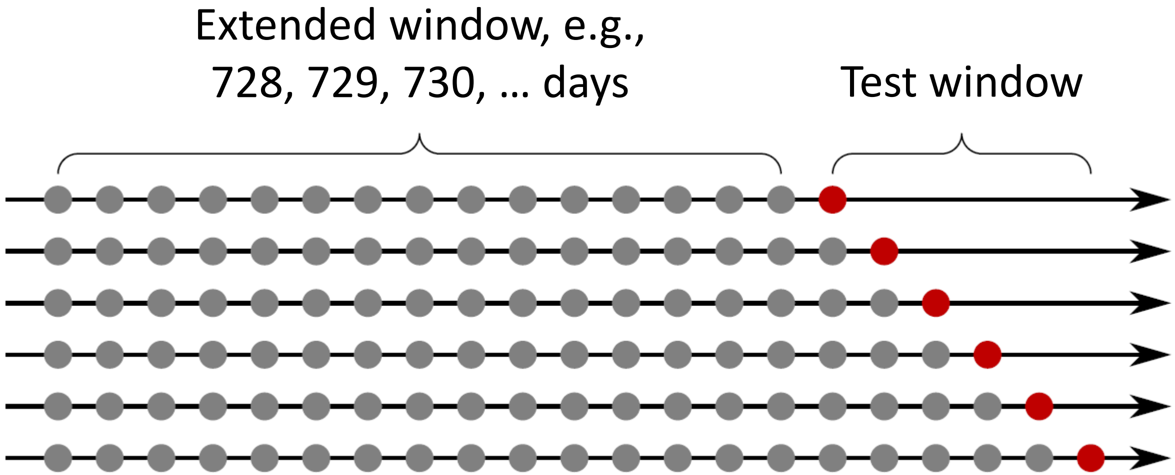
```
% Load GEFCom2014 data
% (6 kolumn: YYYYMMDD, HH, zonal price, system load, zonal load, day-of-the-week)
d = load('GEFCOM.txt');

% Select one hour for analysis ...
hour = 9; p = d(hour:24:end,3);
% ... or take the daily average (uncomment)
% p = mean( reshape(d(:,3),24,length(p)/24) )';

% Last day of the calibration period
T = 360;

% Day-of-the-week
dow = d(1:24:end,6);

% Naive method - point forecasts
pf = zeros(size(p));
for j=8:length(p)
  switch dow(j)
    case {1, 6, 7}
      pf(j) = p(j-7);
    otherwise
      pf(j) = p(j-1);
  end
end

disp('    MAE')
disp(mean(abs(p(T+1:end) - pf(T+1:end))))
```

Fixed calibration windows



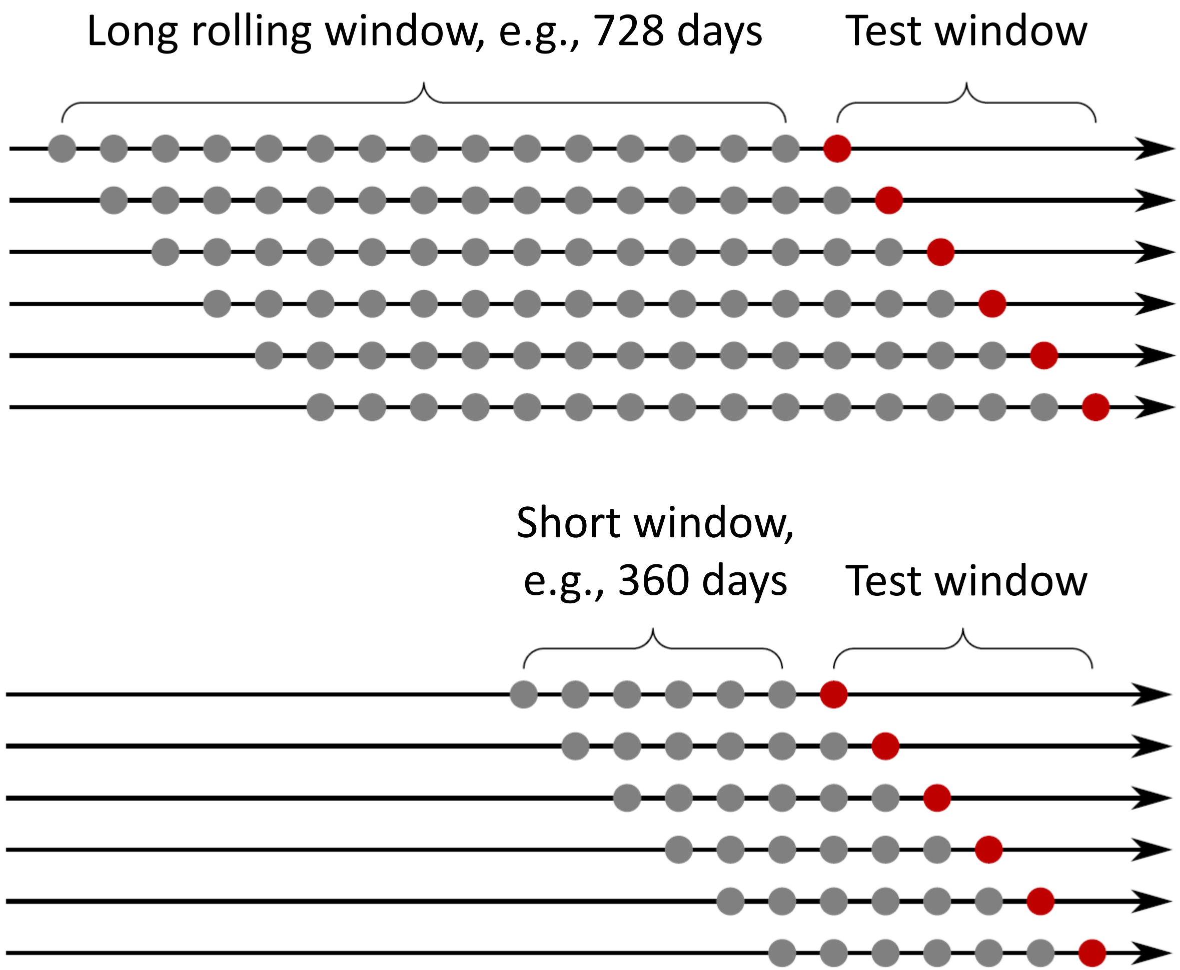


Extended calibration windows



- Not recommended
- Statistical properties change over time

Rolling calibration windows



Autoregressive (AR) models

- AR(1)

$$P_{d,h} = \beta_{0,h} + \beta_{1,h} P_{d-1,h} + \varepsilon_{d,h}$$

- Sparse AR(7)

$$P_{d,h} = \beta_{0,h} + \beta_{1,h} P_{d-1,h} + \beta_{2,h} P_{d-2,h} + \beta_{3,h} P_{d-7,h} + \varepsilon_{d,h}$$

- Sparse ARX(7)

$$P_{d,h} = \beta_{0,h} + \beta_{1,h} P_{d-1,h} + \beta_{2,h} P_{d-2,h} + \beta_{3,h} P_{d-7,h} + \beta_{4,h} \hat{Z}_{d,h} + \varepsilon_{d,h}$$

AR(1) vs. sparse AR(7)

```
% Load GEFCom2014 data
d = load('GEFCOM.txt');
% Select one hour for analysis ...
hour = 9; p = d(hour:24:end,3);

% AR(1), i.e., lag 1
pcal = p(1:T);
y = pcal(8:end); % for day d, d-1, ...
X = [ones(T-7,1) pcal(7:end-1)];

X_fut = [ones(length(p)-T,1) p(T:end-1)];

% Regression, i.e., estimate betas
beta = regress(y,X);
% Make prediction
pf1 = zeros(size(p));
pf1(T+1:end,1) = X_fut*beta;
```

```
% Sparse AR(7), i.e., lags 1,2,7
pcal = p(1:T);
y = pcal(8:end); % for day d, d-1, ...
X = [ones(T-7,1) pcal(7:end-1) ...
  pcal(6:end-2) pcal(1:end-7)];
X_fut = [ones(length(p)-T,1) p(T:end-1) ...
  p(T-1:end-2) p(T-6:end-7)];
% Regression, i.e., estimate betas
beta = regress(y,X);
% Make prediction
pf7 = zeros(size(p));
pf7(T+1:end,1) = X_fut*beta;
```

Sparse AR(7) vs. sparse ARX(7)

```
% Load GEFCom2014 data
d = load('GEFCOM.txt');
% Select one hour for analysis ...
hour = 9; p = d(hour:24:end,3);
```

```
% Sparse AR(7), i.e., lags 1,2,7
pcal = p(1:T);
y = pcal(8:end); % for day d, d-1, ...
X = [ones(T-7,1) pcal(7:end-1) ...
  pcal(6:end-2) pcal(1:end-7)];
X_fut = [ones(length(p)-T,1) p(T:end-1) ...
  p(T-1:end-2) p(T-6:end-7)];
% Regression, i.e., estimate betas
beta = regress(y,X);
% Make prediction
pf7 = zeros(size(p));
pf7(T+1:end,1) = X_fut*beta;
```

```
% Sparse ARX(7), i.e., lags 1,2,7 + X
pcal = p(1:T); xcal = x(1:T);
y = pcal(8:end); % for day d, d-1, ...
X = [ones(T-7,1) pcal(7:end-1) ...
  pcal(6:end-2) pcal(1:end-7) xcal(8:end)];
X_fut = [ones(length(p)-T,1) p(T:end-1) ...
  (T-1:end-2) p(T-6:end-7) x(T+1:end)];
% Regression, i.e., estimate betas
beta = regress(y,X);
% Make prediction
pf7x = zeros(size(p));
pf7x(T+1:end,1) = X_fut*beta;
```

MAE and RMSE errors for AR-type and naive forecasts

	Naive	Naive168	Naive24
MAE	6.7798	11.5186	7.4377
RMSE	13.3265	24.5567	12.8539

	AR(1)	Sparse AR(7)	Sparse ARX(7)
MAE	7.8072	7.0290	6.7882
RMSE	12.6462	12.5198	12.1237

AR(1): Various calibration windows

AR1_variants.ipynb

Code

```
[1]: import numpy as np
     data = np.loadtxt('GEFCOM.txt', delimiter='\t', usecols=list(range(6)))
```

```
[2]: # calibration window: days 1-728 (as in the "L07+L08 recording" by Grzegorz Marcjasz)
     real = data[-354*24:, 2]
     real_train = data[:728*24, 2]
     ar1 = np.zeros((354, 24))
     # only intercept and price of the same hour, d-1
     for hour in range(24):
         # y - labels for training (dependent variable)
         # x - inputs for training (independent variables)
         # xf - inputs for the test
         y = real_train[hour::24]
         x = np.stack([np.ones( (727,) ), y[:-1]])
         xf = np.stack([np.ones( (354,) ), data[hour::24, 2][727:-1]])
         y = y[1:]
         betas = np.linalg.lstsq(x.T, y, rcond=None)[0]
         pred = np.dot(betas, xf)
         ar1[:, hour] = pred
         print([hour, np.mean(np.abs(pred - real[hour::24])), betas]) # added row
     ar1 = np.reshape(ar1, (ar1.shape[0] * ar1.shape[1], ))
     print(['All hours', np.mean(np.abs(ar1 - real))])
```

AR(1): Various calibration windows

Matlab vs Python: various calibration windows

```
% Last day of the calibration period
T = 728

disp('Fit AR(1) on all available data for AR(1)')
disp('MAE   hour     AR(1)   beta(0)   beta(1)')
PF = zeros(size(d(:,3)));
for hour=1:24
    p = d(hour:24:end,3);
    % AR(1) - estimation
    pcal = p(1:T);
    y = pcal(2:end); % for day d, d-1, ...
    X = [ones(T-1,1) pcal(1:end-1)];
    X_fut = [ones(length(p)-T,1) p(T:end-1)];
    % Regression, i.e., estimate betas
    beta = regress(y,X);
    % Make prediction
    pf1 = zeros(size(p));
    pf1(T+1:end,1) = X_fut*beta;
    PF(hour:24:end) = pf1;
    disp([hour, mean(abs(p(T+1:end) - pf1(T+1:end))), beta'])
end
disp('MAE - All hours')
disp(mean(abs(d(T*24+1:end,3) - PF(T*24+1:end))))
```

Sample “expert” ARX-type model

- Let $p_{d,h} = \log(P_{d,h})$ and $z_{d,h} = log(\hat{Z}_{d,h})$
- Consider the following “expert” model:

$$\begin{aligned} p_{d,h} = \beta_{0,h} &+ \beta_{1,h} p_{d-1,h} + \beta_{2,h} p_{d-2,h} + \beta_{3,h} p_{d-7,h} \\ &+ \beta_{4,h} \min_{k=1,\dots,24} p_{d-1,k} + \beta_{5,h} \hat{z}_{d,h} \\ &+ \beta_{6,h} D_{Sat} + \beta_{7,h} D_{Sun} + \beta_{8,h} D_{Mon} + \varepsilon_{d,h} \end{aligned}$$

- Then $\hat{P}_{d,h} = \exp(\hat{p}_{d,h})$

“Expert” model: Main function

```
% Preliminaries
data = load('GEFCOM.txt');
startd = 1;                       % first day of the calibration window
endd = 360;                       % last day of the calibration window
Ndays = 722;                      % user provided number of days to be predicted
                                  % (max is 722 for GEFCom with endd=360)

% Estimate and compute forecasts of the ARX model
res_ARX = epf_arx(data(:,1:4),Ndays,startd,endd);

% Compute naive forecasts, function startnaive.m can be found here:
% https://ideas.repec.org/c/wuu/hscode/zip16002.html
res_naive = startnaive(data,Ndays,startd,endd*24);

% Compute and display MAE
TT = 1:Ndays*24;
disp(['MAE for days ' num2str(endd+1) ' to ' num2str(length(data)/24) ' across all hours'])
disp(['(length of the calibration window for point forecasts = ' num2str(endd) ' days)'])
disp(['ARX     ' num2str(mean(abs(res_ARX(TT,3)-res_ARX(TT,4))))])
disp(['Naive   ' num2str(mean(abs(res_naive(TT,3)-res_naive(TT,4))))])

% Plot prices and forecasts
plot([res_ARX(:,3:4) res_naive(:,4)])
legend('Price','ARX forecast','Naive forecast')
```

epf_arx.m

```
function [result] = epf_arx(data,Ndays,startd,endd)
%   DATA:   4-column matrix (date, hour, price, load forecast)
%   RESULT: 4-column matrix (date, hour, price, forecasted price)

i = weekday(datenum(num2str(data(1,1)),'yyyymmdd'))-1;  % Weekday of starting day
N = length(data);
data = [data zeros(N,4)];          % Append 'data' matrix with daily dummies & p_min
for j=1:24:N
  switch mod(i,7)
    case 6
      data(j:j+23,5) = 1;          % Saturday dummy in column 5
    case 0
      data(j:j+23,6) = 1;          % Sunday dummy in column 6
    case 1
      data(j:j+23,7) = 1;          % Monday dummy in column 7
  end;
  i=i+1;
  data(j:j+23,8) = min(data(j:j+23,3)); % p_min in column 8
end;

result = zeros(Ndays*24,4);        % Initialize 'result' matrix
result(:,1:3) = data(endd*24+1:(endd+Ndays)*24,1:3);
for j = 1:Ndays                    % For all days ...
  for hour = 1:24                  % ... compute 1-day ahead forecasts for each hour
    data_h = data(hour:24:end,:);
    % Change forecast_arx to forecast_narx for NARX ... much, much slower !!!
    result((j-1)*24+hour,4) = forecast_arx(data_h(startd+(j-1):endd+j,:));
  end
end
```

forecast_arx.m (variant 1)

```
function prediction=forecast_arx(DATA)
%  DATA: 8-column matrix (date,hour(fixed),price,load forecast,Sat,Sun,Mon dummy,p_min)

% Select data to be used
price = DATA(1:end-1,3);                                  % For day d (d-1, ...)
price_min = DATA(1:end-1,8);                              % For day d
Dummies = DATA(2:end,5:7);                                % Dummies for day d+1
loadr = DATA(2:end,4);                                    % Load for day d+1

% Take logarithms
price = log(price);
mc = mean(price); price = price - mc;                     % Remove mean(price)
price_min = log(price_min);
price_min = price_min - mean(price_min);                  % Remove mean(price_min)
loadr = log(loadr);

% Calibrate the ARX model (WITHOUT intercept)
y = price(8:end);                                         % For day d, d-1, ...
% Define explanatory variables for calibration
X = [price(7:end-1) price(6:end-2) price(1:end-7)...
price_min(7:end-1) loadr(7:end-1) Dummies(7:end-1,1:3)];
% Define explanatory variables for day d+1
X_fut = [price(end) price(end-1) price(end-6)...
price_min(end) loadr(end) Dummies(end,1:3)];

beta = regress(y,X);                                      % Estimate the ARX model
prog = X_fut*beta;                                        % Compute a step-ahead forecast
prediction = exp(prog+mc);                                % Convert to price level
```

forecast_arx.m (variant 2)

```
function prediction=forecast_arx(DATA)
%  DATA: 8-column matrix (date,hour(fixed),price,load forecast,Sat,Sun,Mon dummy,p_min)

% Select data to be used
price = DATA(1:end-1,3);                                   % For day d (d-1, ...)
price_min = DATA(1:end-1,8);                               % For day d
Dummies = DATA(2:end,5:7);                                 % Dummies for day d+1
loadr = DATA(2:end,4);                                     % Load for day d+1

% Take logarithms
price = log(price);                                        % No need to remove mean(price)
price_min = log(price_min);                                % No need to remove mean(price_min)
loadr = log(loadr);

% Calibrate the ARX model (WITH intercept)
y = price(8:end);                                          % For day d, d-1, ...
% Define explanatory variables for calibration
X = [ones(size(y)) price(7:end-1) price(6:end-2) price(1:end-7)...
price_min(7:end-1) loadr(7:end-1) Dummies(7:end-1,1:3)];
% Define explanatory variables for day d+1
X_fut = [1 price(end) price(end-1) price(end-6)...
price_min(end) loadr(end) Dummies(end,1:3)];

beta = regress(y,X);                                       % Estimate the ARX model
prog = X_fut*beta;                                         % Compute a step-ahead forecast
prediction = exp(prog);                                    % Convert to price level
```

Results

Laptop with i7-1065G7

NARX: Elapsed time is 2512.997536 seconds (ca. 41.88 minutes)
ARX: Elapsed time is 2.673782 seconds
Naive: Elapsed time is 0.298362 seconds

MAE for days 361 to 1082 across all hours
(length of the calibration window for point forecasts = 360 days)

NARX	6.6169
ARX	5.8718
Naive	7.6340