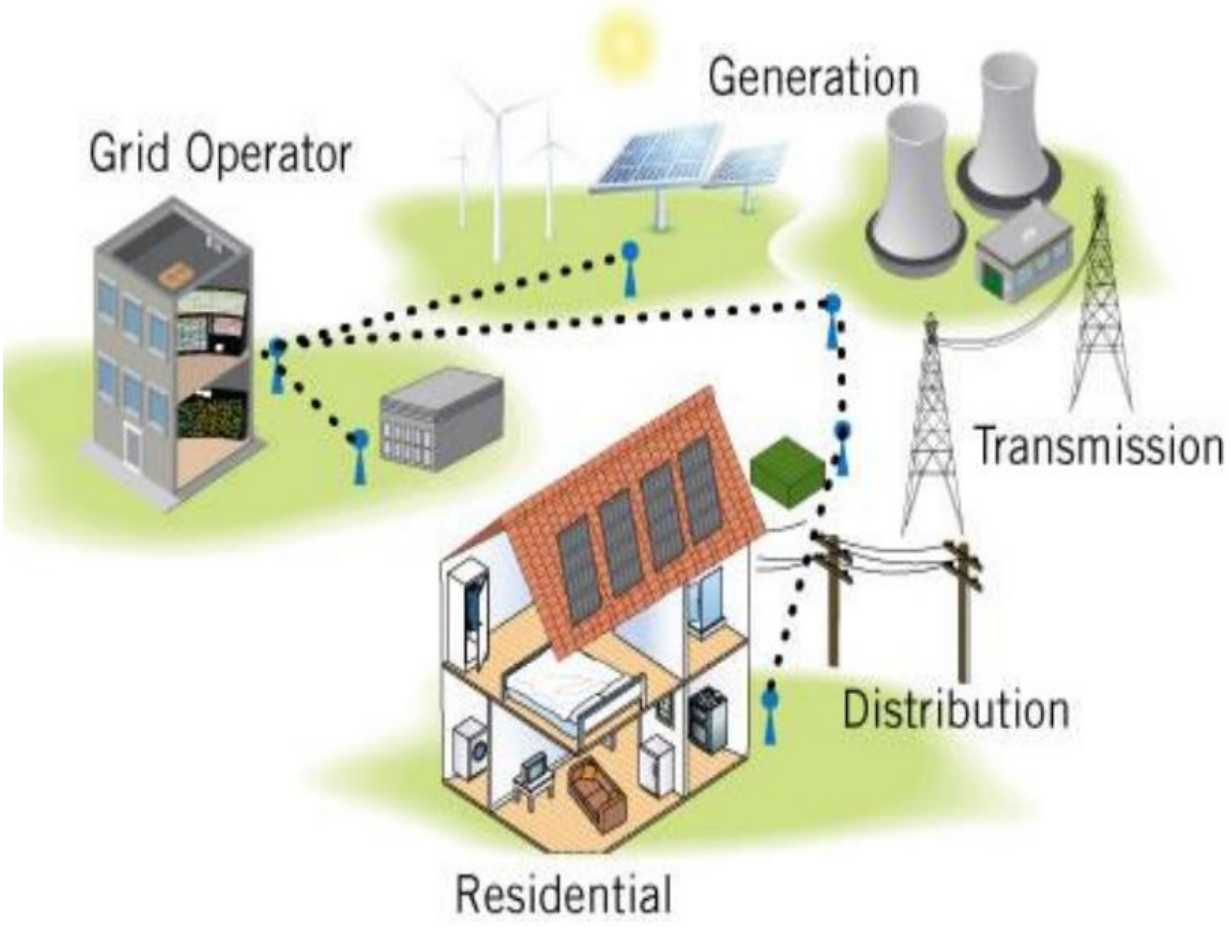
Business Intelligence Workspace Project 2

Forecasting Electricity Prices (Data Set: France)



Submitted by: *Hammad Ullah*

Submitted to: *Piotr Nyczka*

Date: *30-May-2022*

Background:

Electricity price forecasting, a branch of energy forecasting methods, focuses on the spot and forward prices in the wholesale electricity markets. It is an important tool that influences the energy companies' decision-making at a larger corporate level. The forecasts not only predict the prices for the coming days but it also helps us see the seasonality in the prices whether that is weekly, daily, monthly, or even yearly. The whole purpose of forecasting is mainly for the bidding on power at a wholesale market as the prices for a certain day is decided a day before through the bidding system as can be seen in Figure 1 below.

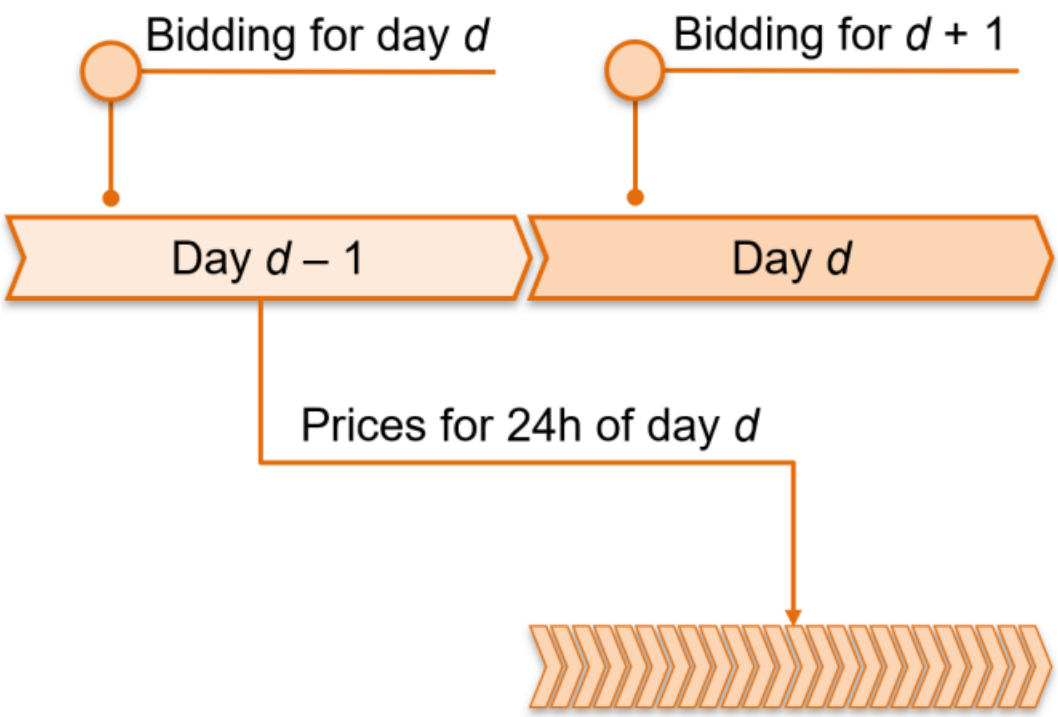


Figure 1 Day-Ahead Market for Electricity

Aim and Tools:

Data Preparation:

Our aim for this report is to download the electricity price data for France from the start of the year 2019 till 25th of October 2021. The data in these files include the day-ahead prices for all the days as well as the day-ahead load forecast. We need to treat the data for **daylight saving time** and correct the repetitions or any missing day's day. For the missing days in the data, we take the average of the neighboring days and take that value for the missing day. For the day's that are repeated, an average of the 2 days is taken and substituted as one entry.

Data Visualization:

After the data has been cleaned and organized, we need to prepare scatter plots using the data of the first 2 years i.e., 2019-2020. The following **scatter plots** of forecasted load versus the price shall be prepared:

- For all data
- For all hours on Saturdays
- For the hour 10am on all days of the week

Along with the scatter plots, we have to prepare **weekly and daily seasonality plots** for both the price and load data for the same span of time i.e., 2019-2020.

Data Forecasting:

After the scatter and seasonality plots, we need to compute forecasts for all the days in 2021 that we have the data for i.e., from 1st January to 25th October. We shall prepare forecasts for the electricity price using 4 different models.

The models are as follows:

[1] Naive 1:

$$\hat{P}_{d,h} = P_{d-7,h}$$

[2] Naive 2:

$$\begin{cases} \hat{P}_{d,h} = P_{d-7,h} & \text{for } d = \text{Mon, Sat, Sun} \\ \hat{P}_{d,h} = P_{d-1,h} & \text{otherwise} \end{cases}$$

[3] Autoregressive Model (ARX1):

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

where

- $P_{d,h}$ – price for hour h day d,
- $\hat{Z}_{d,h}$ – load forecast for day d and hour h,
- $D_{Mon}, D_{Sat}, D_{Sun}$ – dummies for Monday, Saturday and Sunday, e.g., $D_{Mon} = 1$ for $d =$ Monday and 0 otherwise.

For each hour h of the day, we shall compute forecasts of the ARX1 model for all days in 2021. We will calibrate the model only once – using a fixed two-year window (2019-2020).

[4] Autoregressive Model (ARX1)

We shall be using the same model i.e., ARX1, but this time we shall roll our calibration window. Therefore, for each hour h of the day, we will compute forecasts of the ARX1 model for all days in 2021. This time we will use a two-year rolling calibration window, i.e., for the 24h of 1.01.2021 use data from 1.01.2019-31.12.2020, for the 24h of 2.01.2021 use data from 2.01.2019-1.01.2021, etc.

However, it must be noted that there is multicollinearity present in our ARX1 model between P_{d-1} for the 24^{th} hour and the $P_{d-1,24}$ variable since both are not similar, but rather identical. This exists only for the 24^{th} hour so we shall make sure to remove that from our model for h = 24.

For all the 4 models, we will calculate the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) for each hour of the day separately and jointly for all hours.

We shall be using **MATLAB** for all of these tasks. The code is explained in the following sub-section of this report.

Neural Network (Bonus Task):

For each hour h of the day, we need to compute forecasts for all days in 2021 of a multilayer perceptron (MLP) with:

- Same inputs as the 24 hourly ARX1 models
- 2 hidden layers
- 24 outputs, i.e., Pd,1, Pd,2, ..., Pd,24
- sigmoid activation function for both hidden layers,
- A fixed two-year calibration window (2019-2020).

And we shall calculate MAE and RMSE and compare with the other four models. For this task, we shall be using **Python.**

Code and Methodology:

```
clc;
clear;
close all;
warning off
```

- This initial piece of code is just to make sure all the figures are closed, and the command window is cleared every time we run the code.

```
day_ahead_2yr = readtable('Day-ahead_prices_201901010000_202101012359.csv');
day_ahead_2yr_fr = day_ahead_2yr(:,[1,2,6]);

day_ahead_rest = readtable('Day-ahead_prices_202101020000_202110252359.csv');
day_ahead_rest_fr = day_ahead_rest(:,[1,2,6]);

day_ahead_price_all = [day_ahead_2yr_fr; day_ahead_rest_fr];
```

- Since, it was not possible to download the price data for all the 3 years at once from the website, two csv files were downloaded. Thus, here we read both files into a table using the readtable() function and then merge both the tables together by putting them on top of each other.

- The rows selected (1,2 and 6) correspond to date, hour of the day, and day-ahead prices for France.

```
Date = ["Mar 31, 2019";"Mar 29, 2020";"Mar 28, 2021";"Oct 27, 2019";"Oct 25, 2020"];
TimeOfDay = ["2:00 AM";"2:00 AM";"2:00 AM";"2:00 AM";"2:00 AM"];
France___MWh_ = [mean(day_ahead_price_all.France___MWh_(2138:2139));mean(day_ahead_price_all.France___MWh_(10874:10875));
    mean(day_ahead_price_all.France___MWh_(19610:19611));mean(day_ahead_price_all.France___MWh_(7178:7179));
    mean(day_ahead_price_all.France___MWh_(15914:15915))];
newRows = table(Date,TimeOfDay,France___MWh_);
```

- Since, we were aware of the row positions that needed to be fixed for daylight saving, a new table was created with the date, time and price entries. This table was named 'newRows'. The averages of the price entries were taken directly from the data loaded before at the corresponding rows.

```
final_data_day_ahead = [day_ahead_price_all(1:2138, :); newRows(1,:); day_ahead_price_all(2139:7177,:);newRows(4,:);
    day_ahead_price_all(7180:10874,:); newRows(2,:); day_ahead_price_all(10875:15913,:); newRows(5,:);
    day_ahead_price_all(15916:19610,:);newRows(3,:); day_ahead_price_all(19611:end,:) ];
```

- In the above code, we cut the data loaded at different points and insert rows from the table 'newRows' into the right positions.

```
final_day_ahead_demand_all = readtable('FRhourlyload.csv');
data = [final_day_ahead_demand_all final_data_day_ahead(:,"France___MWh_")];
data.Properties.VariableNames([1 2 3]) = {'Time' 'Day-ahead Load forecasts','Day-ahead prices'};
```

- The day ahead load data is also loaded into a table and both the price data and load data is merged into one table and the column names are changed in order to make sure the dataset looks organized. The data table can be seen below:

24696x5 table

	1 Time	2 Day-ahead Load forecasts	3 Day-ahead prices
1	01.01.2019 00:00:00	63450	51
2	01.01.2019 01:00:00	60800	46.2700
3	01.01.2019 02:00:00	59950	39.7800
4	01.01.2019 03:00:00	56400	27.8700
5	01.01.2019 04:00:00	53550	23.2100
6	01.01.2019 05:00:00	52200	22.6400
7	01.01.2019 06:00:00	52150	20.9000
8	01.01.2019 07:00:00	52200	26.5400
9	01.01.2019 08:00:00	52050	25
10	01.01.2019 09:00:00	52200	25
11	01.01.2019 10:00:00	54250	34.6400
12	01.01.2019 11:00:00	56350	41.0300
13	01.01.2019 12:00:00	58800	48.9700
14	01.01.2019 13:00:00	59400	41.0600
15	01.01.2019 14:00:00	56550	37.9500
16	01.01.2019 15:00:00	53900	33.4200
17	01.01.2019 16:00:00	53650	39.8000
18	01.01.2019 17:00:00	56650	44.2900
19	01.01.2019 18:00:00	61150	62.6500
20	01.01.2019 19:00:00	63200	65.5600
21	01.01.2019 20:00:00	62600	61.2700
22	01.01.2019 21:00:00	60450	57.4100

```
figure
plot(data{1:17544,"Day-ahead prices"},data{1:17544,"Day-ahead Load forecasts"},'.')
ylabel('Day-ahead Load forecasts');
xlabel('Day-ahead prices');
title('Year 2019-2020 Load vs Price Scatter Plot for all days')
```

- Here, the first scatter plot of forecasted load versus price for 2019-2020 has been plotted.

```
data.Time = datetime(data.Time,'InputFormat','dd-MM-uuuu''HH:mm:ss');
data.day = weekday(datetime(data.Time));
data_sat = data(weekday(data.Time) == 7, :);
figure
plot(data_sat{1:2496,"Day-ahead prices"},data_sat{1:2496,"Day-ahead Load forecasts"},'.')
ylabel('Day-ahead Load forecasts');
xlabel('Day-ahead prices');
title('Year 2019-2020 Load vs Price Scatter Plot for all Saturdays')
```

- The code for the second scatter plot can be seen above. All the days corresponding to a Saturday is put into a dataset called 'data_sat' and then all Saturdays from 2019-2029 has been plotted for Load vs Price. The resulting plot can be seen below.

```
data_hr10 = data(hour(data.Time) == 10, :);
figure
plot(data_hr10{1:731,"Day-ahead prices"},data_hr10{1:731,"Day-ahead Load forecasts"},'.')
ylabel('Day-ahead Load forecasts');
xlabel('Day-ahead prices');
title('Year 2019-2020 Load vs Price Scatter Plot for hour = 10:00 all days')
```

- The above code is for the third scatter plot, for which all the days corresponding to hour 10 has been extracted and plotted for load vs price.

```
data.hour = timeofday(data.Time);
figure
%Convert columns to arrays
elec_time = table2array(data(1:17544,'Time'));
elec_demand = table2array(data(1:17544,'Day-ahead Load forecasts'));
elec_price = table2array(data(1:17544,'Day-ahead prices'));
```

- Before we move on to the seasonality plots, the electricity time, demand and price are converted into arrays to help us plot the seasonality plots.

```
%Daily season for demand/load
subplot(2,2,1)
elec_day_demand = reshape(elec_demand,24,[]);
plot(timeofday(elec_time(1:24)),elec_day_demand)
xlabel('Time of day')
ylabel('Load/Demand')
title('Daily seasonality for demand/load')
```

- The above code reshapes the array of electricity demand into days i.e., reshapes it into rows of 24 values and then plots it against the time which is reshaped in a similar manner.

```
%Daily season for price
subplot(2,2,2)
elec_day_price = reshape(elec_price,24,[]);
plot(timeofday(elec_time(1:24)),elec_day_price)
xlabel('Time of day')
ylabel('Price')
title('Daily seasonality for price')
```

- The above code reshapes the array of electricity price into days i.e., reshapes it into rows of 24 values and then plots it against the time which is reshaped in a similar manner.

```
%Weekly season for demand/load
subplot(2,2,3)
elec_week_demand_ext = [elec_demand; NaN*ones(24*4,1)];
elec_week_demand = reshape(elec_week_demand_ext,24*7,[]);
plot(elec_time(1:24*7),elec_week_demand')
datetick('x','ddd')
xlabel('Weekday')
ylabel('Load/Demand')
title('Weekly season for demand/load')
```

- The above code reshapes the array of electricity demand into weeks i.e., reshapes it into rows of 24*7 values and then plots it against the time which is reshaped in a similar manner.

```
%Weekly season for price
subplot(2,2,4)
elec_week_price_ext = [elec_price; NaN*ones(24*4,1)];
elec_week_price = reshape(elec_week_price_ext,24*7,[]);
plot(elec_time(1:24*7),elec_week_price')
datetick('x','ddd')
xlabel('Weekday')
ylabel('Price')
title('Weekly season for price')
```

- The above code reshapes the array of electricity price into weeks i.e., reshapes it into rows of 24*7 values and then plots it against the time which is reshaped in a similar manner.

```
%Yearly Data
data19 = data(1:8760,:);
data20 = data(8761:17544,:);
data21 = data(17545:end,:);
```

- Before, we move on to the forecasting, I divided to the data into new variables of tables by year.

```
%naive#1 d-7
%total hrs in 2021 = 7152
%days = 7152/24 = 298
%2021 starts at 17545th row
T = 731; %cal length
test_len = 298;
npf7 = array2table(zeros(298*24,1));
npf7(1:end,1) = data(17544-167:end-168,"Day-ahead prices");
data21arrr  = table2array(data21(1:end,"Day-ahead prices"));
npf7arrr = table2array(npf7(1:end,1));

mae1 = mean(abs(data21arrr - npf7arrr));
rmse1 =(mean(data21arrr - npf7arrr).^2).^0.5;

to_display1 = ['Mean Absolute Error for the Naive#1 model: ', num2str(mae1)];
disp(to_display1)
to_display2 = ['Root Mean Squar Error for the Naive#1 model: ', num2str(rmse1)];
disp(to_display2)
```

- The above code is for our Naive#1 model. Since we have to go 7 days back for each hour for the model, the code is written for the respective commands and then the results for the mean-absolute-error and root-mean-square-error for all the hours combined are displayed. Basically, data from last 7 days of 2020 till the 8th last day of 2021 data we have represents our Naive#1 model.

```
%now lets do hour by hour mean error and rmse too for naive 7
tab = [];
for hour = 1:24
    p = table2array(data21(hour:24:end,"Day-ahead prices"));
    g = npf7arrr(hour:24:end);
    tab(hour,1:3) = [hour mean(abs(p - g)) (mean(p - g).^2).^0.5];
end
tab = array2table(tab);
tab.Properties.VariableNames([1 2 3]) = {'Hour' 'MAE','RMSE'};
```

- Then the results for the mean-absolute-error and root-mean-square-error for each hour separately were calculated and tabulated.

```
%naive17
npf17 = array2table(zeros(298*24,1));
z = 0;
for d = 1:7152
    d_for_hour = data21(d,:);
    if d_for_hour.('day') == 3 || d_for_hour.('day') == 4  || d_for_hour.('day') == 5  || d_for_hour.('day') == 6
        npf17(d,1) = data(17545-24+z,3);
        z = z + 1;
    else
        npf17(d,1) = data(17545-168+z,3);
        z = z + 1;
    end
end
data21arr  = table2array(data21(1:end,"Day-ahead prices"));
npf17arr = table2array(npf17(1:end,1));
mae2 = mean(abs(data21arr - npf17arr));
rmse2 =(mean(data21arr - npf17arr).^2).^0.5;
disp(' ')
to_display1 = ['Mean Absolute Error for the Naive#2 model: ', num2str(mae2)];
disp(to_display1)
to_display2 = ['Root Mean Squar Error for the Naive#2 model: ', num2str(rmse2)];
disp(to_display2)
```

- The above code is for our Naive#2 model. Since we have to go 7 days back for hours on some day and 1 day back for hours on other days for the model, the code is written for the respective command inside an if-statement and then the results for the mean-absolute-error and root-mean-square-error for all the hours combined are displayed. Basically, for Tuesday, Wednesday, Thursday and Friday, we take the data of one day back for each hour, but for Monday, Saturday, and Sunday, we take the data for price for 7 days back for each hour.

```
%now lets do hour by hour mean error and rmse too for naive2 model
tab2 = [];
for hour = 1:24
    p = table2array(data21(hour:24:end,"Day-ahead prices"));
    g = npf17arr(hour:24:end);
    tab2(hour,1:3) = [hour mean(abs(p - g)) (mean(p - g).^2).^0.5];
end
tab2 = array2table(tab2);
tab2.Properties.VariableNames([1 2 3]) = {'Hour' 'MAE','RMSE'};
```

- Then the results for the mean-absolute-error and root-mean-square-error for each hour separately were calculated and tabulated for the

```
%cal window parameters
mincol = zeros(size(data,1),1);
for i = 24:24:24696
    q = table2array(data(i-23:i,'Day-ahead prices'));
    s = min(q);
    t = ones(24,1);
    f = (s.*t);
    mincol(i+1:i+24) = f(1:24);
end
mincol = array2table(mincol);
mincol = mincol(25:24720,1);
```

- Before we could move on to the ARX1 model, we need to make sure our independent variables are ready. The above code represents the variable which basically represents the price for previous day's minimum taken as price for the whole next day.

```
hr_24_prev_day = zeros(size(data,1),1);
for i = 24:24:24696
    g = data(i,3);
    g = table2array(g);
    u = ones(24,1);
    h = (g.*u);
    hr_24_prev_day(i+1:i+24) = h(1:24);
end
hr_24_prev_day = array2table(hr_24_prev_day);
hr_24_prev_day = hr_24_prev_day(25:24720,1);
```

- The above code represents the independent variable which basically represents the price for previous day's 24^{th} hour value taken as price for the whole next day. Now we move on to the ARX1 model which involves fixed calibration length.

```
tab3 = [];
T = 731;
PF = zeros(size(data(:,3)));
for hour=1:24
    p = table2array(data(hour:24:end,"Day-ahead prices"));
    x = table2array(data(hour:24:end,"Day-ahead Load forecasts"));
    k = table2array(mincol(hour:24:end,1));
    z = table2array(hr_24_prev_day(hour:24:end,1));
    Dsat = table2array(data(hour:24:end, "day")) == 7;
    Dsun = table2array(data(hour:24:end, "day")) == 1;
    Dmon = table2array(data(hour:24:end, "day")) == 2;
    % AR(1) - estimation
    pcal = p(1:T);
    xcal = x(1:T);
    kcal = k(1:T);
    zcal = z(1:T);
    Dsatcal = Dsat(1:T);
    Dsuncal = Dsun(1:T);
    Dmoncal = Dmon(1:T);
    y = pcal(8:end); % for day d, d-1, d-7, load, min24hrbefore, dummies  ...
    if hour == 24
        X = [ones(T-7,1) pcal(7:end-1) pcal(1:end-7) xcal(8:end) kcal(7:end-1) Dsatcal(8:end) Dsuncal(8:end) Dmoncal(8:end)];
        X_fut = [ones(length(p)-T,1) p(T:end-1) p(T-6:end-7) x(T+1:end) k(T:end-1) Dsat(T+1:end) Dsun(T+1:end) Dmon(T+1:end)];
    else
        X = [ones(T-7,1) pcal(7:end-1) pcal(1:end-7) xcal(8:end) kcal(7:end-1) zcal(7:end-1) Dsatcal(8:end) Dsuncal(8:end) Dmoncal(8:end)]
        X_fut = [ones(length(p)-T,1) p(T:end-1) p(T-6:end-7) x(T+1:end) k(T:end-1) z(T:end-1) Dsat(T+1:end) Dsun(T+1:end) Dmon(T+1:end)];
    end
    % Regression, i.e., estimate betas
    beta = regress(y,X);
```

```
    beta = regress(y,X);
    % Make prediction
    pf1 = zeros(size(p));
    pf1(T+1:end,1) = X_fut*beta;
    PF(hour:24:end) = pf1;
    tab3(hour,1:3) = [hour mean(abs(p(T+1:end) - pf1(T+1:end))) (mean(p(T+1:end) - pf1(T+1:end)).^2).^0.5];
end
tab3 = array2table(tab3);
tab3.Properties.VariableNames([1 2 3]) = {'Hour' 'MAE','RMSE'};

mae3 = mean(abs(table2array(data(T*24+1:end,3)) - PF(T*24+1:end)));
rmse3 =(mean(table2array(data(T*24+1:end,3)) - PF(T*24+1:end)).^2).^0.5;
disp(' ')
to_display1 = ['Mean Absolute Error for the ARX1 fixed calibration model: ', num2str(mae3)];
disp(to_display1)
to_display2 = ['Root Mean Squar Error for the ARX1 fixed calibration model: ', num2str(rmse3)];
disp(to_display2)
```

- The above code is for ARX1 model based on fixed calibration window length of 731 days (2019-2020). The dummies for Monday, Saturday and Sunday are represented by Dmon, Dsat, and Dsun. First, we extract each independent variable's same hour data and then take only the first 731 values i.e., for 2019 and 2020. Then we use that to form our dependent variable and our independent variables matrix and their future values of matrix that we shall use later to make the predictions.

- The point to be noted here is that as mentioned before, in order to avoid multi-collinearity, we removed the $P_{d-1,24}$ from the ARX1 model for the 24^{th} hour. We did this by using an IF statement.

- After estimating the betas using regression, we made our predictions and put them inside a table that represents mean-

absolute-error and root-mean-square-error for each hour separately.

- Then we calculate the mean-absolute-error and root-mean-square-error for all the hours combined.

```
datafinal  = [data mincol hr_24_prev_day];
index_for_roll_cal = 0;
T = 731;
ind_d_entries = 0;
vec = zeros(7152,1);
index_for_data_entry = 1;
for day = 1:298
    real_train = datafinal(index_for_roll_cal*24+1:(731+index_for_roll_cal)*24,:);
    for hour=1:24
        p = table2array(real_train(hour:24:end,"Day-ahead prices"));
        x = table2array(real_train(hour:24:end,"Day-ahead Load forecasts"));
        k = table2array(real_train(hour:24:end,'mincol'));
        z = table2array(real_train(hour:24:end,'hr_24_prev_day'));
        Dsat = table2array(real_train(hour:24:end, "day")) == 7;
        Dsun = table2array(real_train(hour:24:end, "day")) == 1;
        Dmon = table2array(real_train(hour:24:end, "day")) == 2;
        xx = table2array(datafinal(hour:24:end,"Day-ahead Load forecasts"));
        DDsat = table2array(datafinal(hour:24:end, "day")) == 7;
        DDsun = table2array(datafinal(hour:24:end, "day")) == 1;
        DDmon = table2array(datafinal(hour:24:end, "day")) == 2;
        % AR(1) - estimation
        pcal = p(1:T);
        xcal = x(1:T);
        kcal = k(1:T);
        zcal = z(1:T);
        Dsatcal = Dsat(1:T);
        Dsuncal = Dsun(1:T);
        Dmoncal = Dmon(1:T);
        yr= pcal(8:end); % for day d, d-1, d-7, load, min24hrbefore, 24th hour from day before, dummies  ...
        if hour == 24
            Xr = [ones(T-7,1) pcal(7:end-1) pcal(1:end-7) xcal(8:end) kcal(7:end-1) Dsatcal(8:end) Dsuncal(8:end) Dmoncal(8:end)];
            X_futr = [1 p(T) p(T-6) xx(T+1+ind_d_entries) k(T) DDsat(T+1+ind_d_entries) DDsun(T+1+ind_d_entries) DDmon(T+1+ind_d_entries)];
        else
            Xr = [ones(T-7,1) pcal(7:end-1) pcal(1:end-7) xcal(8:end) kcal(7:end-1) zcal(7:end-1) Dsatcal(8:end) Dsuncal(8:end) Dmoncal(8:end)];
            X_futr = [1 p(T) p(T-6) xx(T+1+ind_d_entries) k(T) z(T) DDsat(T+1+ind_d_entries) DDsun(T+1+ind_d_entries) DDmon(T+1+ind_d_entries)];
        end
        % Regression, i.e., estimate betas
        beta = regress(yr,Xr);
        % Make prediction
        pf1r = X_futr*beta;
        vec(index_for_data_entry) = pf1r;
        index_for_data_entry = index_for_data_entry + 1;
    end
    index_for_roll_cal = index_for_roll_cal + 1;
    ind_d_entries = ind_d_entries + 1;
end

mae4 = mean(abs(table2array(data(T*24+1:end,3)) - vec));
rmse4 =(mean(table2array(data(T*24+1:end,3)) - vec).^2).^0.5;
disp(' ')
to_display1 = ['Mean Absolute Error for the ARX1 rolling calibration model: ', num2str(mae4)];
disp(to_display1)
to_display2 = ['Root Mean Squar Error for the ARX1 rolling calibration model: ', num2str(rmse4)];
disp(to_display2)
```

```
tab4 = [];
for hour = 1:24
    p = table2array(data21(hour:24:end,"Day-ahead prices"));
    g = vec(hour:24:end);
    tab4(hour,1:3) = [hour mean(abs(p - g)) (mean(p - g).^2).^0.5];
end
tab4 = array2table(tab2);
tab4.Properties.VariableNames([1 2 3]) = {'Hour' 'MAE','RMSE'};
```

- The above code is for ARX1 model based on rolling calibration window length of 731 days (2019-2020). We roll by each day. The dummies for Monday, Saturday and Sunday are represented by Dmon, Dsat, and Dsun. First, we chose our real training data that rolls 24 entries at every loop in order to make sure the length of the calibration window remains same but the data used for calibration rolls by one day. We extract each independent variable's same hour data and then take the 731 values i.e., for 2019 and 2020. We could have skipped this step as well since the length of the same hour data will remain 731 in total as we are using real_train data which comprises of 731 days in total. Then we use that to form our dependent variable and our independent variables matrix and their future values of matrix that we shall use later to make the predictions.

- The point to be noted here is that as mentioned before, in order to avoid multi-collinearity, we removed the $P_{d-1,24}$ from the ARX1 model for the 24^{th} hour. We did this by using an IF statement.

- After estimating the betas using regression, we made our predictions and put them inside a table that represents mean-absolute-error and root-mean-square-error for each hour separately.

- Then we calculate the mean-absolute-error and root-mean-square-error for all the hours combined.

BONUS Task Code and Methodology:

```
import numpy as np
import pandas as pd
import os
import datetime

os.getcwd()
```

- The libraries are imported

```
day_ahead_2yr = pd.read_csv('Day-ahead_prices_201901010000_202101012359.csv', delimiter=';', usecols=(0,1,5))
day_ahead_rest = pd.read_csv('Day-ahead_prices_202101020000_202110252359.csv', delimiter=';', usecols=(0,1,5))
day_ahead_price_all = pd.concat([day_ahead_2yr, day_ahead_rest],ignore_index=True)
day_ahead_price_all.columns = ['Date', 'TimeOfDay', 'France___MWh_']


newRows = {'Date': ["Mar 31, 2019","Mar 29, 2020","Mar 28, 2021","Oct 27, 2019","Oct 25, 2020"],
        'TimeOfDay': ["2:00 AM","2:00 AM","2:00 AM","2:00 AM","2:00 AM"],
       'France___MWh_':[day_ahead_price_all.France___MWh_[2137:2139].mean(),
                        day_ahead_price_all.France___MWh_[10873:10875].mean(),
                       day_ahead_price_all.France___MWh_[19609:19611].mean(),
                        day_ahead_price_all.France___MWh_[7177:7179].mean(),
                       day_ahead_price_all.France___MWh_[15913:15915].mean()]}
newRows = pd.DataFrame.from_dict(newRows,orient='columns')


final_data_day_ahead = pd.concat([day_ahead_price_all[0:2138],newRows[0:1],
                                  day_ahead_price_all[2138:7177],newRows[3:4],
                                  day_ahead_price_all[7179:10874],newRows[1:2],
                                  day_ahead_price_all[10874:15913],newRows[4:5],
                                  day_ahead_price_all[15915:19610],newRows[2:3],
                                  day_ahead_price_all[19610::]],ignore_index=True)


final_day_ahead_demand_all = pd.read_csv('FRhourlyload.csv', delimiter=',',parse_dates=['Time (CET/CEST)'])
final_day_ahead_demand_all.columns = ['Date', 'DayAheadDemand']
data = pd.concat([final_day_ahead_demand_all['Date'],
                  final_data_day_ahead['France___MWh_'],
                  final_day_ahead_demand_all['DayAheadDemand']],axis=1,ignore_index=False)

days = [1,2,3,4,5,6,0]*1029
Weekday = pd.DataFrame(np.zeros((24696, 1)))
idx = 0
for i in days:
    Weekday[idx:idx+24] = i
    idx = idx + 24
data['Weekday'] = Weekday

mincol = np.zeros(24720)
for i in range(24,24721,24):
    mincol[i:i+24] = data['France___MWh_'][i-24:i].min()
mincol = pd.DataFrame(mincol[24:24721])
data = pd.concat([data,mincol],axis=1,ignore_index=False)


data.columns = ['Date', 'DayAheadPrice', 'DayAheadDemand','Weekday','MinCol']
```

```
hr_24_prev_day = np.zeros(24720)
for i in range(24,24721,24):
    hr_24_prev_day[i:i+24] = data['DayAheadPrice'][i-1:i]
hr_24_prev_day = pd.DataFrame(hr_24_prev_day[24:24721])
data = pd.concat([data,hr_24_prev_day],axis=1,ignore_index=True)


data.columns = ['Date', 'DayAheadPrice', 'DayAheadDemand','Weekday','MinCol','24hrbefore']
data
```

```
dumsat = np.zeros(24696)
dumsun = np.zeros(24696)
dummon = np.zeros(24696)

for i in range(24696):
    if data['Weekday'][i] == 5:
        dumsat[i] = 1
    if data['Weekday'][i] == 6:
        dumsun[i] = 1
    if data['Weekday'][i] == 0:
        dummon[i] = 1

dumsat = pd.DataFrame(dumsat)
dumsun = pd.DataFrame(dumsun)
dummon = pd.DataFrame(dummon)

data = pd.concat([data,dumsat,dumsun,dummon],axis=1,ignore_index=False)


data.columns = ['Date', 'DayAheadPrice', 'DayAheadDemand','Weekday','MinCol','24hrbefore','DummSat','DummSun','DummMon']
```

- In the above, large portions of code, we have done all the data correction that we previously did in MATLAB in order to do the neural network on the same data in a different programming language.

	Date	DayAheadPrice	DayAheadDemand	Weekday	MinCol	24hrbefore	DummSat	DummSun	DummMon
0	2019-01-01 00:00:00	51.00	63450.0	1.0	20.90	54.91	0.0	0.0	0.0
1	2019-01-01 01:00:00	46.27	60800.0	1.0	20.90	54.91	0.0	0.0	0.0
2	2019-01-01 02:00:00	39.78	59950.0	1.0	20.90	54.91	0.0	0.0	0.0
3	2019-01-01 03:00:00	27.87	56400.0	1.0	20.90	54.91	0.0	0.0	0.0
4	2019-01-01 04:00:00	23.21	53550.0	1.0	20.90	54.91	0.0	0.0	0.0
5	2019-01-01 05:00:00	22.64	52200.0	1.0	20.90	54.91	0.0	0.0	0.0
6	2019-01-01 06:00:00	20.90	52150.0	1.0	20.90	54.91	0.0	0.0	0.0
7	2019-01-01 07:00:00	26.54	52200.0	1.0	20.90	54.91	0.0	0.0	0.0
8	2019-01-01 08:00:00	25.00	52050.0	1.0	20.90	54.91	0.0	0.0	0.0
9	2019-01-01 09:00:00	25.00	52200.0	1.0	20.90	54.91	0.0	0.0	0.0
10	2019-01-01 10:00:00	34.64	54250.0	1.0	20.90	54.91	0.0	0.0	0.0

- The resulting data looks as shown above.

```
data = data.to_numpy()
data = data[:, [1,2,3,4,5,6,7,8]]
cal_len = 731
test_len = data.shape[0] // 24 - cal_len
real = data[-298*24:, 0]
data = data.astype('float64')
data
```

- The data set is converted to numpy array and respected columns are chosen.

```
import tensorflow
from tensorflow import keras
import numpy as np
import optuna # !!!
import logging
import sys
cal_len = 731
test_len = data.shape[0] // 24 - cal_len
real = data[-298*24:, 0].reshape((298, 24))
price = data[:731*24, 0].reshape((731, 24))
load = data[:731*24, 1].reshape((731, 24))
#dumm = data[:731*24:, 2].reshape((731, 24))
minn = data[:731*24, 3].reshape((731, 24))
prev24 = data[:731*24,4].reshape((731, 24))
dumsat = data[:731*24, 5].reshape((731, 24))
dumsun = data[:731*24, 6].reshape((731, 24))
dummon = data[:731*24, 7].reshape((731, 24))


real_train = data[:731*24, 0]

y = real_train.reshape((731, 24))
y = y[7:]

x = np.stack([np.ones( (724,24) ), price[6:-1, :], price[:-7, :], load[7:, :], minn[6:-1, :],prev24[6:-1, :],
              dumsat[7:, :], dumsun[7:, :],dummon[7:, :]])

x = x.reshape((724, 24*9))

pricef = data[:, 0].reshape((1029, 24))
loadf = data[:, 1].reshape((1029, 24))
minnf = data[:, 3].reshape((1029, 24))
prev24f = data[:,4].reshape((1029, 24))
dumsatf = data[:, 5].reshape((1029, 24))
dumsunf = data[:, 6].reshape((1029, 24))
dummonf = data[:, 7].reshape((1029, 24))

xf = np.stack([np.ones( (298,24) ), pricef[730:-1, :], pricef[724:-7, :], loadf[731:, :],
                   minnf[730:-1, :], prev24f[731:, :],
                  dumsatf[731:, :], dumsunf[731:, :], dummonf[731:, :]])

xf = xf.reshape((298, 24*9))
```

- We import the necessary libraries and prepare the x matrix and future x matrix for the neural network.

```
permutation = np.random.permutation(list(range(724)))
percentage_val = 0.2
Nval = int(724 * percentage_val)
seen_x = x[permutation[:724-Nval], :]
seen_y = y[permutation[:724-Nval], :]
unseen_x = x[permutation[724-Nval:], :]
unseen_y = y[permutation[724-Nval:], :]
```

- The above code is used to carry out a trial if our neural network produces credible results

```
def objective(trial):
    inputs = keras.Input((24*9)) # Input layer of size equal to number of columns
    hidden = keras.layers.Dense(trial.suggest_int('hidden_neurons', 10, 150), activation='sigmoid')(inputs)
    hidden2 = keras.layers.Dense(trial.suggest_int('hidden_neurons2', 10, 150), activation = 'sigmoid')(hidden)
    output = keras.layers.Dense(24, activation='linear')(hidden2)
    model = keras.Model(inputs=inputs, outputs=output)
    model.compile(optimizer=keras.optimizers.Adam(), loss='mse')
    model.fit(seen_x, seen_y, epochs=50, verbose=0, batch_size=32)
    pred = model.predict(unseen_x)
    return (np.mean(np.abs(pred - unseen_y)))
```

- The above code implements the neural network on our trial code where the number of layers for the two hidden layers shall be chosen and optimized. For the output, we have use the linear activation as after many trials, I found that linear fits best to the output data.

```
optuna.logging.get_logger('optuna').addHandler(logging.StreamHandler(sys.stdout))
study_name = 'TEST'
storage_name = f'sqlite:///trials/{study_name}'
study = optuna.create_study(study_name=study_name, storage=storage_name, load_if_exists=True)
study.optimize(objective, n_trials=20, show_progress_bar=True)
best_params = study.best_params
print(best_params)
print(study.trials_dataframe())
```

- The optuna library runs all the trials on the trial data and stores the study in a file names TEST on the computer using the objective function we created earlier.

```
study.best_params
```

```
def evaluate(params):
    inputs = keras.Input((24*9))
    hidden = keras.layers.Dense(params['hidden_neurons'], activation="sigmoid")(inputs)
    hidden2 = keras.layers.Dense(params['hidden_neurons2'], activation="sigmoid")(hidden)
    output = keras.layers.Dense(24, activation='linear')(hidden2)
    model = keras.Model(inputs=inputs, outputs=output)
    model.compile(optimizer=keras.optimizers.Adam(), loss='mse')
    model.fit(x, y, epochs=50, verbose=0, batch_size=32)
    preds = model.predict(xf)
    return(np.mean(np.abs(preds - real)))
```

```
evaluate(study.best_params)
```

- Now we run the evaluate function where we use the best parameters from the optuna results and use it to get the mean absolute error.

Results and Analysis:

1. Scatter Plot of Load vs Price (All Data):

In Figure 2 below, we can see the correlation between the day-ahead load and day-ahead price of electricity for all days and all hours from start of 2019 to end of 2020. We can see that the correlation between the two is positive as the plot move upwards from left to right. As the demand for the electricity rises, so does the price which is a typical market trend in the modern world.

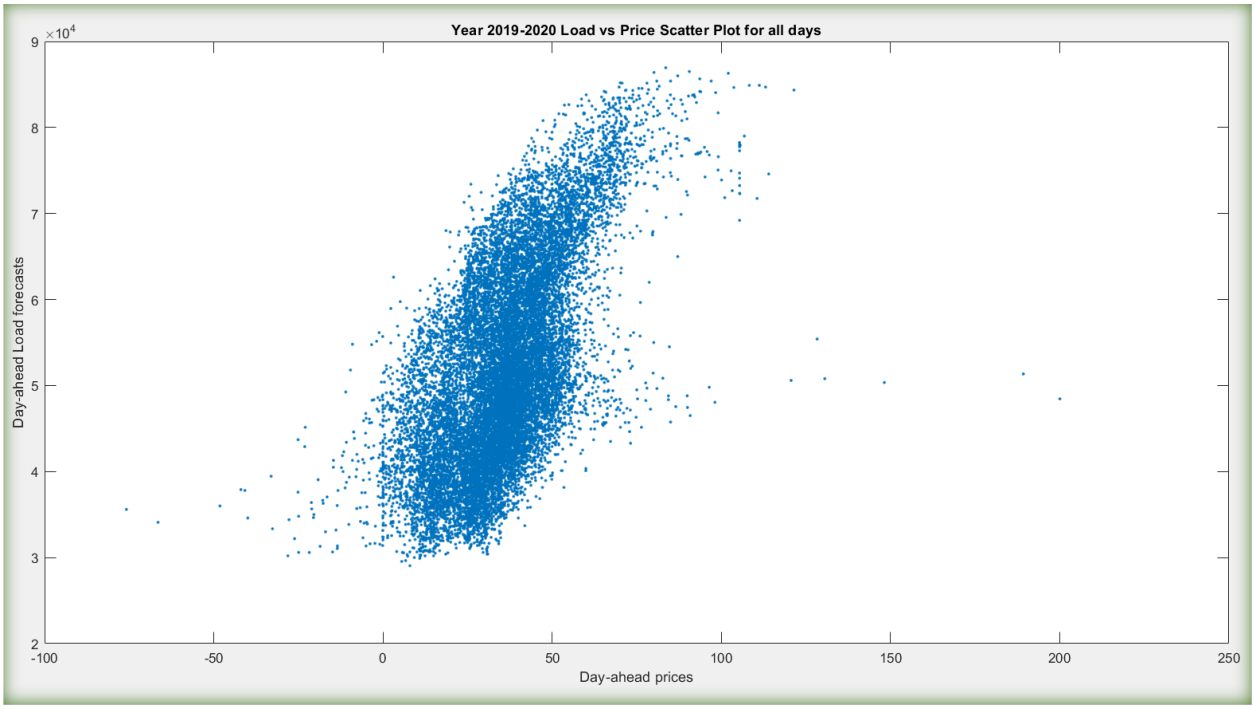


Figure 2 Load vs Price Scatter Plot (All days All Hours)

2. Scatter Plot of Load vs Price (All Hours on Saturdays):

In Figure 3 below, we can see the correlation between the day-ahead load and day-ahead price of electricity for all hours on Saturdays from start of 2019 to end of 2020. We can see that the correlation between the two is positive as the plot move upwards from left to right. As the demand for the electricity rises, so does the price which is a typical market trend in the modern world.

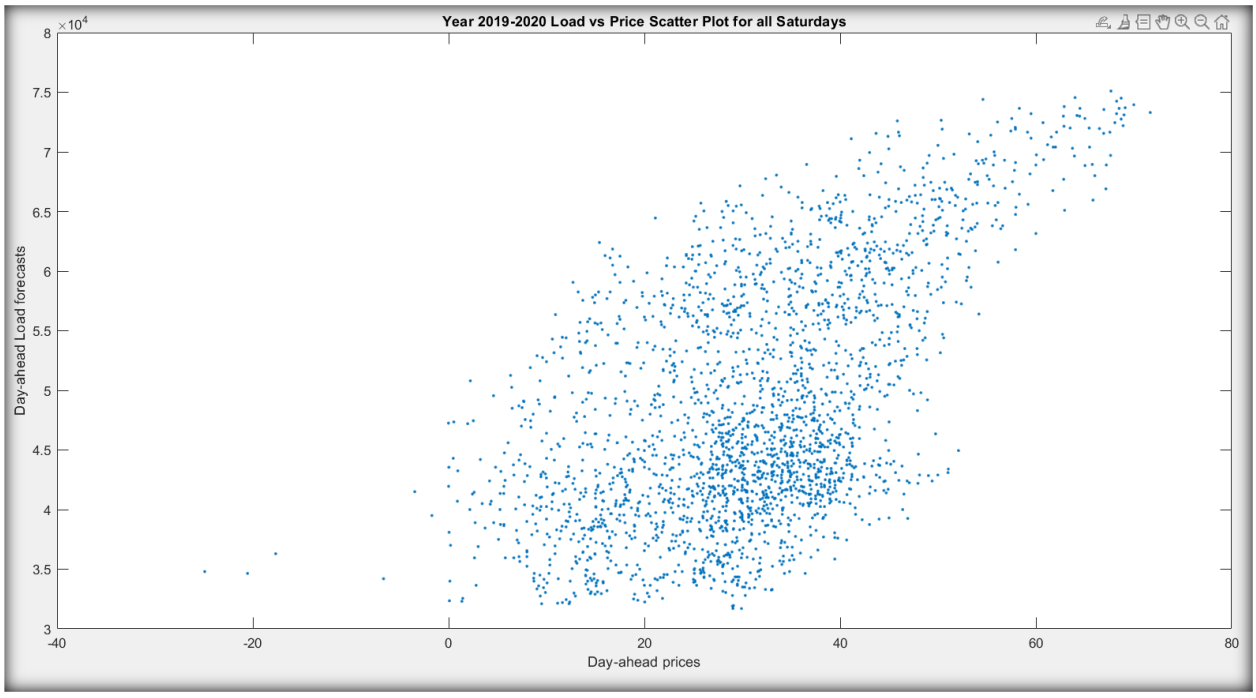


Figure 3 Load vs Price Scatter Plot (Saturdays All Hours)

3. Scatter Plot of Load vs Price (10am on all days):

In Figure 4 below, we can see the correlation between the day-ahead load and day-ahead price of electricity for hour = 10 am on all days of the week from start of 2019 to end of 2020. We can see that the correlation between the two is positive as the plot move upwards from left to right. As the demand for the electricity rises, so does the price which is a typical market trend in the modern world.

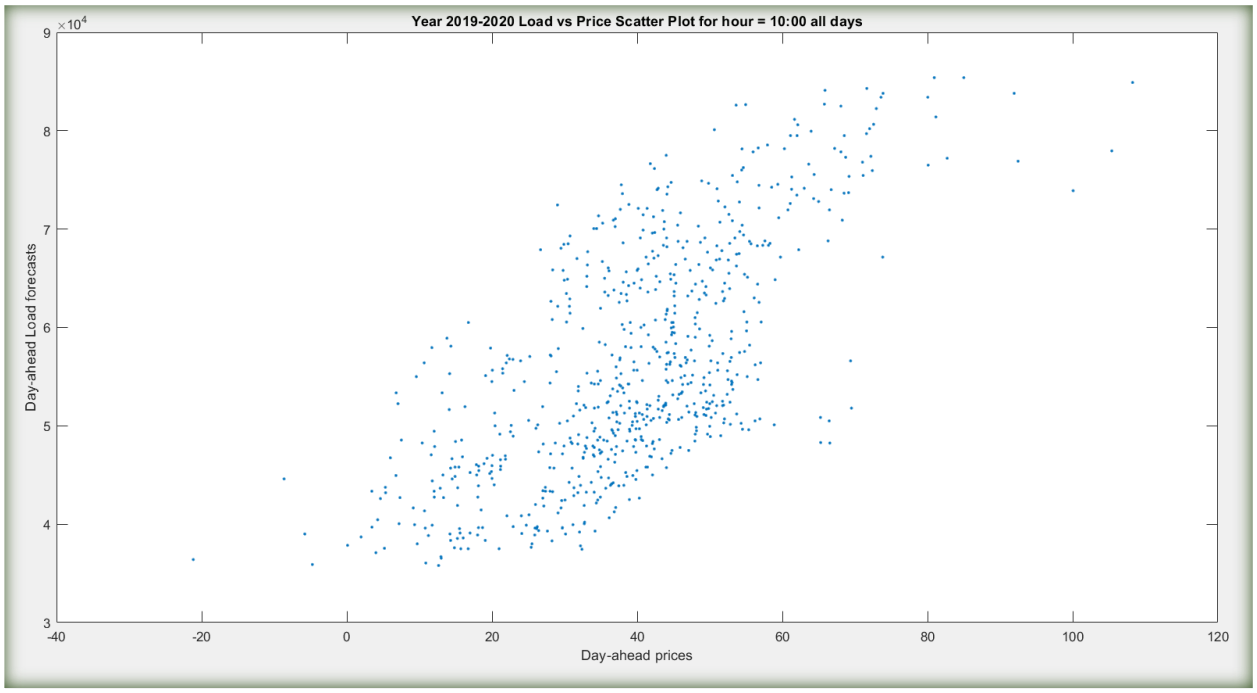


Figure 4 Load vs Price Scatter Plot (All days for Hour = 10)

4. Daily Seasonality Plot for Price:

In Figure 5 below, we can see the daily seasonality plot for electricity price from start of 2019 to end of 2020. We can see that the price for electricity rises for time around 8am and also around 6pm as assuming the demand during these hours rises since we saw in the scatter plots that when the demand rises, the price per MW rises as well. Along with the demand, the price is also high during evening hours because there is less production from solar sources or other renewable resources. Furthermore, the electricity demand decreases during late night and very early morning hours as people are not usually active during those hours. Thus, as the demand is less, we can see in the figure below that there is a dip in the prices for those hours.

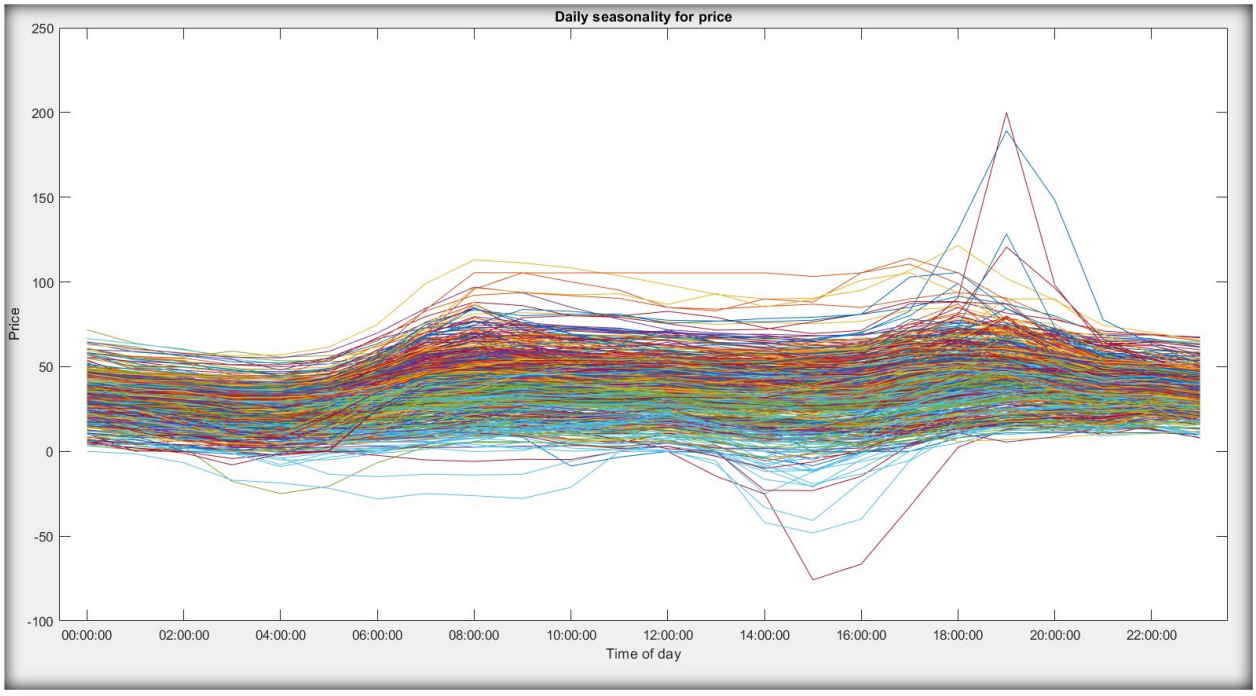


Figure 5 Daily Seasonality Plot for Price

5. Daily Seasonality Plot for Demand:

In Figure 6 below, we can see the daily seasonality plot for electricity demand/load from start of 2019 to end of 2020. We can see that the demand for electricity rises for time around 8am and also around 6pm. Furthermore, the electricity demand decreases during late night and very early morning hours as people are not usually active during those hours. Thus, as the demand is less, we can see in the Figure 5 that there is a dip in the prices for those hours.

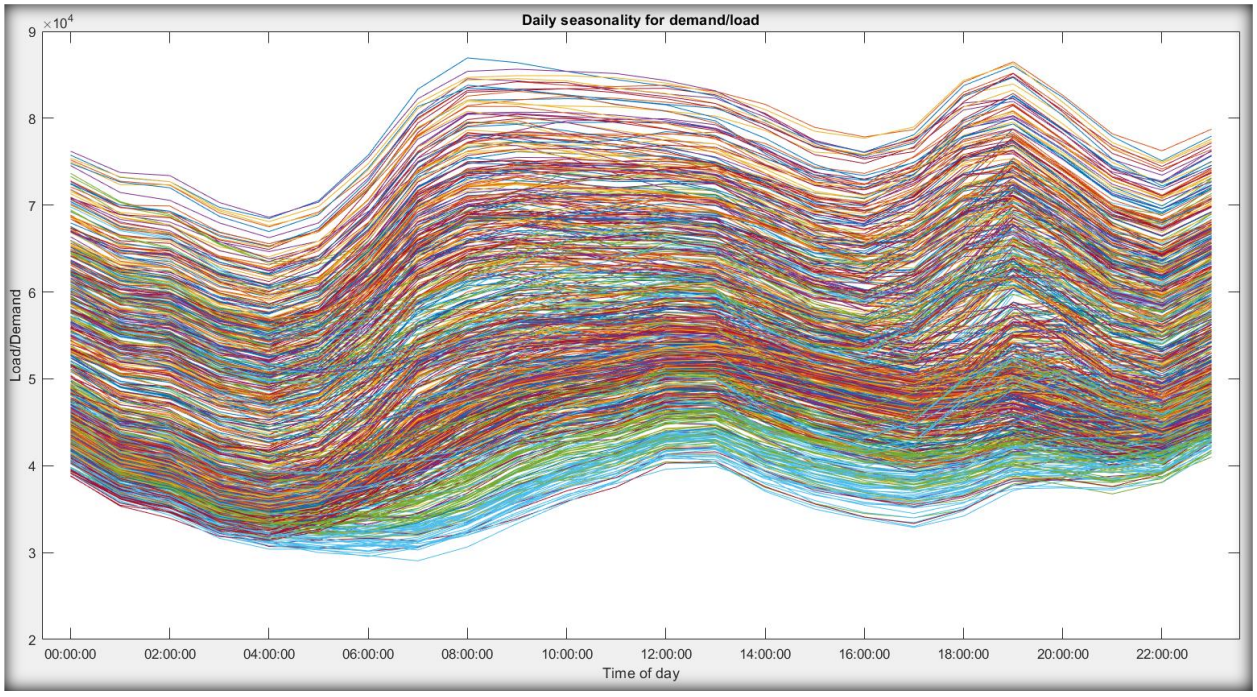


Figure 6 Daily Seasonality Plot for Demand

6. Weekly Seasonality Plot for Price:

In Figure 7 below, we can see the weekly seasonality plot for electricity price from start of 2019 to end of 2020. We can see that the price for electricity is high on Monday to Friday while low on the weekends. This is because there is less demand for electricity on the weekend than on the weekdays. Moreover, if we are careful in our observation, it is noticeable that Monday's price is a little bit lower than other weekdays. This is a great reason to use different complex linear regression models for forecasting the prices rather than just taking the simple naive model that takes the previous day's or previous 7 day's prices. This also explains why in our Naive#2 model, we used d-1 for Tuesday to Friday and d-7 for the rest of the days.

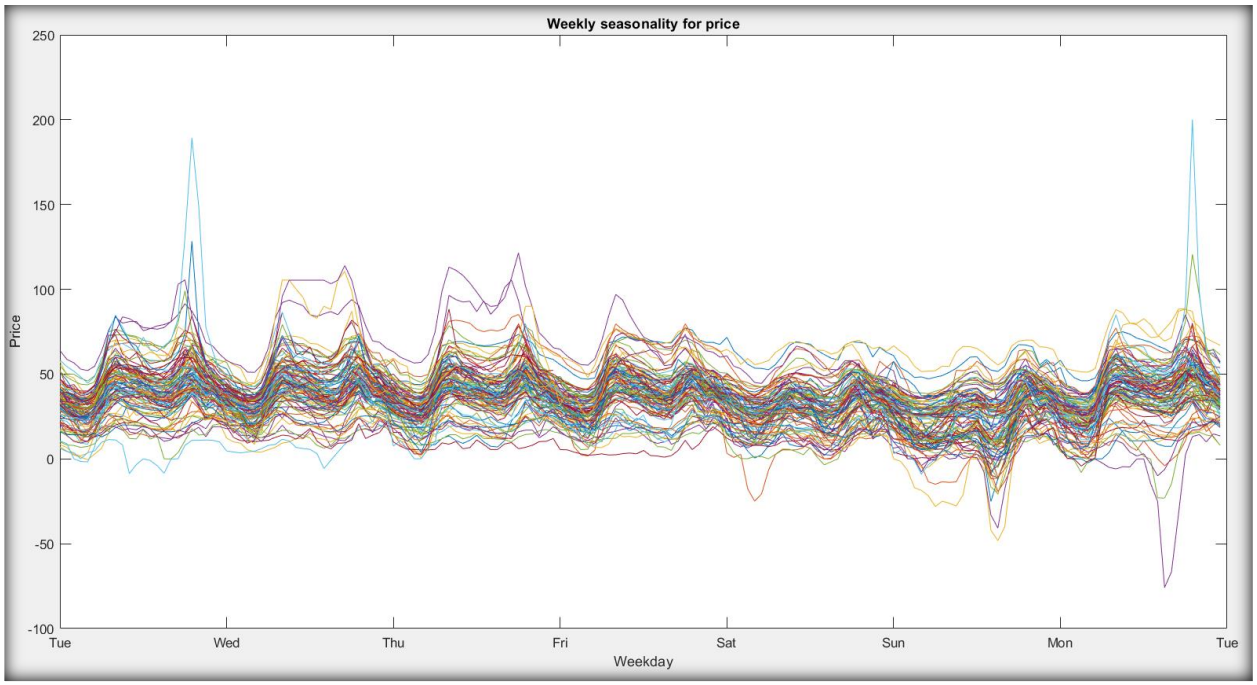


Figure 7 Weekly Seasonality for Price

7. Weekly Seasonality Plot for Demand:

In Figure 8 below, we can see the weekly seasonality plot for electricity demand from start of 2019 to end of 2020. We can see that the demand for electricity is high on Monday to Friday while low on the weekends. Moreover, if we are careful in our observation, it is noticeable that Monday's morning demand is a little bit lower than other weekdays. This is a great reason to use different complex linear regression models for forecasting the demand rather than just taking the simple naive model that takes the previous day's data.

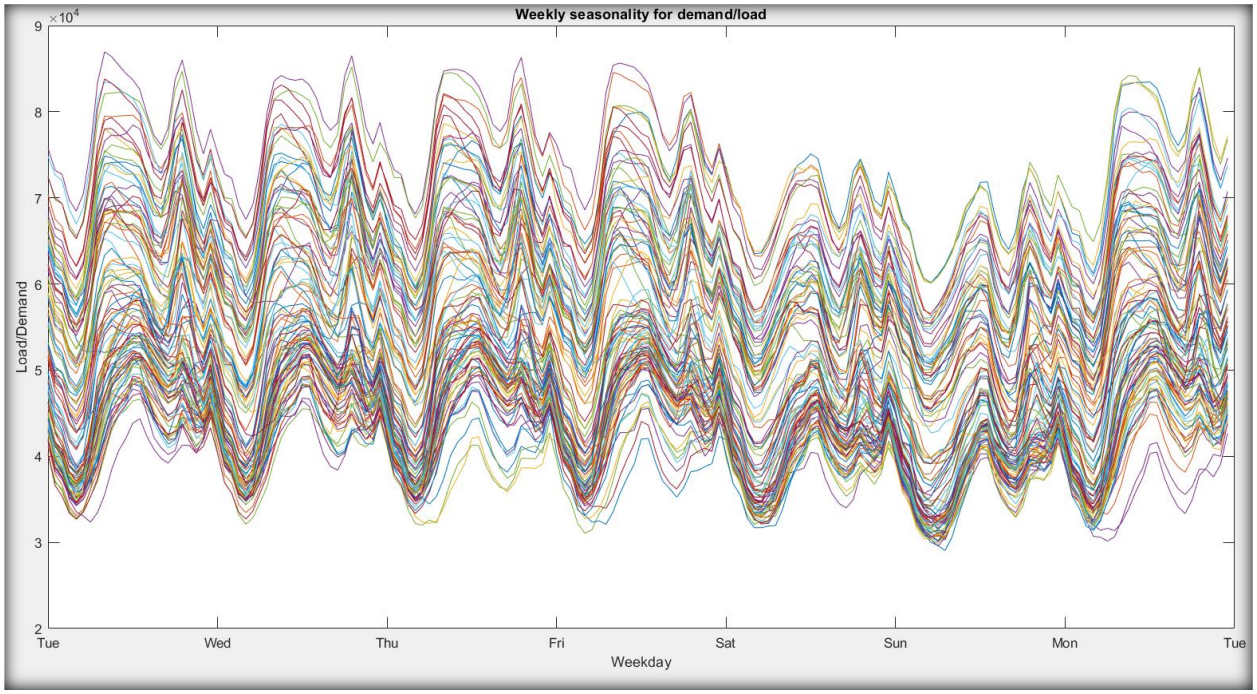


Figure 8 Weekly Seasonality for Demand

8. Naive#1 Model

The MAE and RMSE for Naive#2 model for all the hours combined can be seen below in the table. As we can see, the values are not really ideal especially for MAE

Mean Absolute Error	Root Mean Square Error
17.9594	**27.2855**

The MAE and RMSE for all the hours separately can be seen in Figure 9 below.

1 Hour	2 MAE	3 RMSE
1	16.6603	27.7017
2	17.1878	28.4443
3	18.2172	29.1602
4	19.4457	30.7769
5	20.8824	32.2431
6	19.0585	29.5070
7	19.4744	30.2306
8	18.6187	27.7836
9	18.7086	28.6779
10	17.5520	25.8542
11	17.5816	24.6396
12	18.1778	25.7365
13	16.7760	23.4900
14	18.4236	25.7689
15	20.3573	28.8311
16	21.3276	29.7902
17	21.2021	30.0051
18	20.3897	29.2227
19	18.2254	26.9290
20	16.9793	25.6042
21	14.9747	22.6083
22	14.2292	22.5460
23	13.0423	22.3907
24	13.5327	23.3919

Figure 9 Hourly MAE and RMSE for Naive#1 Model

9. Naive#2 Model

The MAE and RMSE for Naive#2 model for all the hours combined can be seen below in the table. As we can see, the values are not really ideal. Nevertheless, the values are better than Naive#1 model as we put into effect the weekly seasonality into our model whereas in Naive#1 model, only d-7 prices were taken.

Mean Absolute Error	Root Mean Square Error
14.7961	**24.3956**

The MAE and RMSE for all the hours separately can be seen in Figure 10 below.

1 Hour	2 MAE	3 RMSE
1	14.6673	26.9851
2	15.0881	27.3543
3	16.0318	27.9353
4	16.1624	26.8220
5	17.3594	28.1074
6	15.7855	27.5212
7	15.9332	26.8316
8	14.9019	24.2872
9	14.5004	23.4023
10	13.7466	21.2171
11	14.0694	21.2740
12	15.0524	23.1456
13	13.5626	20.5874
14	15.3532	23.1095
15	17.5380	26.5415
16	18.1934	27.2723
17	18.1940	27.0660
18	16.8726	25.4522
19	14.7974	22.6481
20	13.1714	20.7171
21	11.4887	19.5555
22	10.8583	20.1719
23	10.3134	20.8071
24	11.4655	22.6492

Figure 10 Hourly MAE and RMSE for Naive#2 Model

10. ARX1 Model with Fixed Calibration Window

The MAE and RMSE for ARX1 Model with fixed calibration window for all the hours combined can be seen below in the table. As we can see, the values are even better than Naive#2 model for both MAE and RMSE. This is because we put into effect not only the weekly seasonality into our model, but only other deciding factors such as minimum of previous day and the 24th hour value of the previous day. Such deciding factors really helped our case.

Mean Absolute Error	Root Mean Square Error
11.6457	**19.2895**

The MAE and RMSE for all the hours separately can be seen in Figure 10 below.

1 Hour	2 MAE	3 RMSE
1	7.1928	13.7035
2	7.8643	14.9523
3	8.6037	15.6821
4	11.3825	19.8972
5	12.5780	21.1115
6	10.3709	17.8541
7	11.7490	18.5584
8	12.8143	20.3097
9	12.8124	21.7627
10	12.3047	19.8862
11	12.0893	18.3961
12	12.1967	18.8857
13	11.1486	17.2832
14	12.8874	20.4469
15	14.0370	23.0524
16	14.3561	22.9599
17	13.5903	21.6074
18	13.1571	20.0235
19	12.5355	19.4243
20	12.0392	20.0916
21	10.6053	17.4267
22	10.4583	17.4061
23	10.4390	18.0424
24	12.2832	20.8568

Figure 11 ARX1 Model with Fixed Calibration Window

11. ARX1 Model with Rolling Calibration Window

The MAE and RMSE for ARX1 Model with rolling calibration window for all the hours combined can be seen below in the table. As we can see, the values are even better than the fixed calibration window model for both MAE and RMSE. This is because every day, we roll the calibration window which means the weekly and daily trends are up to date, thus, giving us better results.

Mean Absolute Error	Root Mean Square Error
10.9926	**18.015**

The MAE and RMSE for all the hours separately can be seen in Figure 10 below.

1 Hour	2 MAE	3 RMSE
1	7.5139	13.7636
2	8.0702	14.6804
3	8.6329	15.0905
4	10.1012	17.0709
5	11.4403	18.4570
6	10.1511	16.6258
7	11.6945	18.1911
8	11.6485	18.6670
9	12.1304	20.0799
10	11.7052	18.4972
11	11.5259	17.2170
12	11.9840	18.2065
13	10.8442	16.3594
14	12.4805	19.0240
15	13.9495	21.3140
16	14.1598	21.6276
17	14.0339	21.2732
18	13.2838	19.6000
19	11.4817	17.6136
20	10.6203	17.7146
21	9.1540	16.1073
22	8.7459	16.0631
23	8.5294	16.9525
24	9.9416	19.4421

Figure 12 ARX1 Model with Rolling Calibration Window

12. MLP Architecture

The MAE and RMSE for neural network model with fixed calibration window for all the hours combined and separate can be seen in the tables next page. As we can see, the values are not ideal and totally not expected. After days of trying with the code, my best conclusion on the worse results for the neural network can only be explained (if there is no error in my code) is that the predictor variables that we have chosen simply does not work well for a neural network. I came to this conclusion because if the neural network is run on the same data where we divide some part of the data into seen and unseen x and y and run the model, the results seem to make sense as the mean absolute error is between 9 and 11 as shown below which is even better than our rolling calibration model as one would expect.

```
{'hidden_neurons': 94, 'hidden_neurons2': 36}
```

	number	value	datetime_start	datetime_complete
0	0	9.882782	2022-06-05 16:29:33.382337	2022-06-05 16:29:43.386264
1	1	10.309068	2022-06-05 16:29:43.477365	2022-06-05 16:29:51.509202
2	2	10.292797	2022-06-05 16:29:51.602979	2022-06-05 16:30:00.108277
3	3	10.304391	2022-06-05 16:30:00.190400	2022-06-05 16:30:08.829101
4	4	10.098525	2022-06-05 16:30:08.936435	2022-06-05 16:30:17.636419
5	5	10.291790	2022-06-05 16:30:17.744298	2022-06-05 16:30:26.398060
6	6	9.977865	2022-06-05 16:30:26.673967	2022-06-05 16:30:35.881000
7	7	10.284046	2022-06-05 16:30:35.974396	2022-06-05 16:30:44.323029
8	8	9.822963	2022-06-05 16:30:44.435552	2022-06-05 16:30:54.663108
9	9	10.252710	2022-06-05 16:30:54.748513	2022-06-05 16:31:03.848038
10	10	9.763225	2022-06-05 16:31:03.944886	2022-06-05 16:31:13.062542
11	11	9.889735	2022-06-05 16:31:13.157807	2022-06-05 16:31:22.645575
12	12	10.328291	2022-06-05 16:31:22.738333	2022-06-05 16:31:32.369822
13	13	9.819725	2022-06-05 16:31:32.485560	2022-06-05 16:31:41.747005
14	14	10.056692	2022-06-05 16:31:41.857214	2022-06-05 16:31:50.603694
15	15	10.303985	2022-06-05 16:31:50.704542	2022-06-05 16:31:58.941796
16	16	9.778019	2022-06-05 16:31:59.034995	2022-06-05 16:32:07.405351
17	17	9.890912	2022-06-05 16:32:07.511930	2022-06-05 16:32:16.050848
18	18	10.239513	2022-06-05 16:32:16.121659	2022-06-05 16:32:23.857618
19	19	10.278742	2022-06-05 16:32:23.972311	2022-06-05 16:32:32.281904

	duration	params_hidden_neurons	params_hidden_neurons2	state
0	00:00:10.003927	67	78	COMPLETE
1	00:00:08.031837	22	56	COMPLETE
2	00:00:08.505298	15	59	COMPLETE
3	00:00:08.638701	11	99	COMPLETE
4	00:00:08.699984	62	68	COMPLETE
5	00:00:08.653762	13	33	COMPLETE
6	00:00:09.207033	69	62	COMPLETE
7	00:00:08.348633	13	24	COMPLETE
8	00:00:10.227556	89	81	COMPLETE
9	00:00:09.099525	65	97	COMPLETE
10	00:00:09.117656	94	36	COMPLETE
11	00:00:09.487768	100	39	COMPLETE
12	00:00:09.631489	99	10	COMPLETE
13	00:00:09.261445	86	81	COMPLETE
14	00:00:08.746480	83	44	COMPLETE
15	00:00:08.237254	42	82	COMPLETE
16	00:00:08.370356	81	46	COMPLETE
17	00:00:08.538918	77	47	COMPLETE
18	00:00:07.735959	48	26	COMPLETE
19	00:00:08.309593	75	10	COMPLETE

Nevertheless, the best parameters that evaluate function gave me we the following:

- Number of layers for 1st hidden layer = 94
- Number of layers for 2nd hidden layer = 36

Mean Absolute Error	Root Mean Square Error
47.4051	**64.0703**

The hourly MAE and RMSE are given below:

Hour	MAE	RMSE
1	46.902892	63.561578
2	45.978084	59.798660
3	46.026891	64.646082
4	45.053705	59.783355
5	39.470135	50.372494
6	47.327895	63.047795
7	54.631221	72.877175
8	55.210988	71.795968
9	52.706576	67.825885
10	48.803499	68.675612
11	38.553331	45.952885
12	37.111074	44.610859
14	43.849324	57.239754
15	45.397688	57.197549
16	48.937273	63.597238
17	58.713629	89.412324
18	50.289796	67.883029
19	50.550682	66.988597
20	46.269023	60.947832
21	46.804036	64.620583
22	43.650595	61.975198
22	44.755591	61.149679
23	50.948758	68.760360
24	49.107727	69.439993

Conclusion

We have analyzed the load and price seasonality daily and weekly, and noticed that on a daily basis, both of them are high around 8am and 6pm. Whereas, on a weekly basis, both are high generally on the weekdays and low on the weekends. Moving on to the 5 forecasting models that we prepared are analyzed, we can see the summary below**:**

	MEA	**RMSE**
Naive#1	**17.9594**	**27.2855**
Naive#2	**14.7961**	**24.3956**
ARX1 Fixed Window	**11.6457**	**19.2895**
ARX1 Rolling Window	**10.9926**	**18.015**
MLP	**47.3718**	**64.037**

As we can see the RMSE of the models decreases as we move down along the list of models, meaning that the points of the forecasts are more concentrated around the actual line indicating a better fit. The rolling calibration window model has the best value for both the MAE and RMSE and it is because of the many carefully thought independent variables in our model, the removal of multicollinearity, as well as the regular update of the calibration window daily which gives us better predictions in the long run. However, the neural network results are not satisfying for the reason I mentioned earlier that the predictor variables seem to be causing problems for the length of testing and training data we have because for the random data that I tested it on, the MAE error was near 10 which seems highly probable considering the ARX1 model.