Table of Contents

Read & interpret dataset

In [1]:

```
# Import modules
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
import pathlib
import glob
import math
import statsmodels.api as sm
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
import datetime
from numpy import sqrt
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Show all the rows and columns up to 200
pd.set_option('display.max_columns', 200)
pd.set_option('display.max_rows', 200)
```

In [2]:

```
all_data = pd.read_csv('all_data.csv', sep=',', header=0)
all_data.head()
```

Out[2]:

	Date	HH	Open	High	Low	Close	Average	Volume(MWh/h)	Volume(Tick count)	Time	Sys
0	2016-04-01	1	7.69	7.69	7.69	7.69	7.69	0.70	1	00:00:00	
1	2016-04-01	2	7.45	7.45	7.45	7.45	7.45	0.75	0	00:30:00	
2	2016-04-01	3	7.21	7.21	7.21	7.21	7.21	0.80	1	01:00:00	
3	2016-04-01	4	7.06	7.06	7.06	7.06	7.06	0.80	1	01:30:00	
4	2016-04-01	5	7.21	7.21	7.21	7.21	7.21	0.80	1	02:00:00	

Feature Engineering

Moving average/VWAP

In [3]:

```
# Moving average for System price (Do not need lag) 1month, 3month, 1year
all_data["Spot_MA25d"] = all_data["System_price(Yen/kWh)"].rolling(1200).mean().round(2)
all_data["Spot_MA75d"] = all_data["System_price(Yen/kWh)"].rolling(3600).mean().round(2)
all_data["Spot_MA200d"] = all_data["System_price(Yen/kWh)"].rolling(9600).mean().round(2)
```

In [4]:

```
# Adjust unit from MWh to kWh to make VWAP column
all_data['Volume_kWh'] = (all_data['Volume(MWh/h)']/2)*1000
all_data['Cum_Vol'] = all_data['Volume_kWh'].cumsum()
all_data['Cum_Vol_Price'] = (all_data['Volume_kWh'] * all_data['Average'] ).cumsum()
all_data['VWAP'] = all_data['Cum_Vol_Price'] / all_data['Cum_Vol']
all_data = all_data.drop(["Volume_kWh", "Cum_Vol", "Cum_Vol_Price"], axis=1)
all_data = all_data.fillna(0)
all_data.isnull().sum()
```

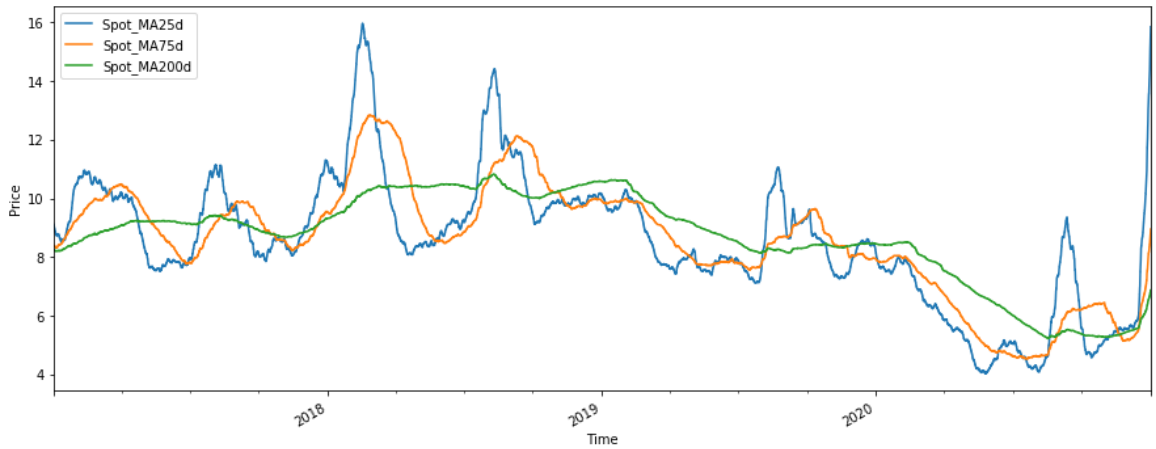
Out[4]:

```
Date                        0
HH                          0
Open                        0
High                        0
Low                         0
Close                       0
Average                     0
Volume(MWh/h)               0
Volume(Tick count)          0
Time                        0
System_price(Yen/kWh)       0
Price_Hokkaido(Yen/kWh)     0
Price_Tokyo(Yen/kWh)        0
Price_Kansai(Yen/kWh)       0
BidExceed_diff              0
HourlyDemand_Hokkaido       0
HourlyDemand_Tokyo          0
HourlyDemand_Kansai         0
HourlyDemand_Okinawa        0
Wind_TOH                    0
Geothermal_TOH              0
PumpedStorage_TOH           0
Interconnection_TOH         0
Geothermal_HOK              0
Wind_HOK                    0
Interconnection_HOK         0
Wind_TKO                    0
Interconnection_TKO         0
Wind_CHB                    0
Interconnection_CHB         0
Wind_HKU                    0
PumpedStorage_HKU           0
Interconnection_HKU         0
Nuclear_KAN                 0
Wind_KAN                    0
Interconnection_KAN         0
Thermal_CHG                 0
Water_CHG                   0
Wind_CHG                    0
Interconnection_CHG         0
Nuclear_SHI                 0
Wind_SHI                    0
Interconnection_SHI         0
Nuclear_KYU                 0
Geothermal_KYU              0
PV_KYU                      0
PVCurtailment_KYU           0
Wind_KYU                    0
WindCurtailment_KYU         0
Interconnection_KYU         0
Thermal_OKI                 0
Biomass_OKI                 0
Wind_OKI                    0
Allarea_TotalDemand         0
Allarea_Water               0
Allarea_Thermal             0
Allarea_Geothermal          0
Allarea_Biomass             0
Allarea_PumpedStorage       0
```

```
Allarea_Interconnection    0
Water_Ratio                0
Thermal_Ratio              0
Geothermal_Ratio           0
Planned_Min_flag           0
Planned_Max_flag           0
ForecastPV_Hokkaido        0
Forecast_PV_Chubu          0
Forecast_Wind_Chubu        0
ForecastPV_Hokuriku        0
ForecastWind_Kansai        0
Temp_Tokyo                 0
SunLight(Time)_Tokyo       0
WindSpeed(m/s)_Tokyo       0
WindDirection_Tokyo        0
SunLight(MJ/㎡)_Tokyo      0
Gas_Price                  0
Oil_Price                  0
month                      0
dayofweek                  0
holiday                    0
Spot_MA25d                 0
Spot_MA75d                 0
Spot_MA200d                0
VWAP                       0
dtype: int64
```

In [5]:

```
# plot the moving average
fig, ax = plt.subplots(1, figsize=(15, 6))

graph = all_data.copy()
graph["DateTime"] = pd.to_datetime(graph["Date"].astype(str) + " " + graph["Time"].astype(str))

# Set index
graph = graph.set_index("DateTime")
start = "2017-01-01 00:00:00"
end = "2020-12-31 23:30:00"

# Plot Close
graph.Spot_MA25d [graph.index > start].plot(ax=ax, label="Spot_MA25d ")
graph.Spot_MA75d[graph.index > start].plot(ax=ax, label="Spot_MA75d")
graph.Spot_MA200d[graph.index > start].plot(ax=ax, label="Spot_MA200d")

# x-axis
plt.gcf().autofmt_xdate()
ax.set(xlabel="Time", ylabel="Price")
plt.legend(loc="upper left");
```



Lag-features (Avoiding data leakage)

In [6]:

```
# Check the remaining memory on PC
import gc
gc.collect()
```

Out[6]:

In [7]:

```
# This is for generating lag
def generate_lag(train, lag_sizes, cols, lag_sizes_type):

    lag_sizes = np.array(lag_sizes)
    if lag_sizes_type == "HH":
        lag_sizes_adj = lag_sizes * 1 # nothing changes
    elif lag_sizes_type == "Hours":
        lag_sizes_adj = lag_sizes * 2
    elif lag_sizes_type == "Day":
        lag_sizes_adj = lag_sizes * 48

    for ix, lag_sizes_adj_ in enumerate(lag_sizes_adj):
        shifted_df = all_data[cols].shift(lag_sizes_adj_)
        shifted_df.columns = [f"{col_name}_lag_{lag_sizes[ix]}_{lag_sizes_type}" for col_name in shif
ted_df.columns]
        if ix == 0:
            shifted_df_return = shifted_df
        else:
            shifted_df_return = shifted_df_return.merge(shifted_df, how="left", left_index=True, right_i
ndex=True)
    return shifted_df_return
```

- Generage lagged features for spot market

In [8]:

```
# Pick up columns that are related to spot market
lag_columns = all_data[['System_price(Yen/kWh)',
      'Price_Hokkaido(Yen/kWh)', 'Price_Tokyo(Yen/kWh)', 'Price_Kansai(Yen/kWh)',
      'BidExceed_diff','Spot_MA25d', 'Spot_MA75d', 'Spot_MA200d', 'VWAP']].columns

# Generate lag features and drop original columns
# all_data_lagged_HH = generate_lag(all_data, [49, 97], lag_columns, "HH")
# all_data_lagged_Hour = generate_lag(all_data, [2], lag_columns, "Hours")
all_data_lagged_Day = generate_lag(all_data, [1, 2], lag_columns, "Day")

# join everything
# all_data_lagged = pd.merge(all_data, all_data_lagged_HH, how="left", left_index=True, right_ind
ex=True)
# all_data = pd.merge(all_data, all_data_lagged_Hour, how="left", left_index=True, right_index=Tru
e)
all_data_lagged_spot = pd.merge(all_data, all_data_lagged_Day, how="left", left_index=True, right
_index=True)

# Drop lag_columns which can be data leakage
all_data_lagged_spot = all_data_lagged_spot.drop(lag_columns, axis=1)

all_data_lagged_spot.columns
```

Out[8]:

```
Index(['Date', 'HH', 'Open', 'High', 'Low', 'Close', 'Average',
       'Volume(MWh/h)', 'Volume(Tick count)', 'Time', 'HourlyDemand_Hokkaido',
       'HourlyDemand_Tokyo', 'HourlyDemand_Kansai', 'HourlyDemand_Okinawa',
       'Wind_TOH', 'Geothermal_TOH', 'PumpedStorage_TOH',
       'Interconnection_TOH', 'Geothermal_HOK', 'Wind_HOK',
       'Interconnection_HOK', 'Wind_TKO', 'Interconnection_TKO', 'Wind_CHB',
       'Interconnection_CHB', 'Wind_HKU', 'PumpedStorage_HKU',
       'Interconnection_HKU', 'Nuclear_KAN', 'Wind_KAN', 'Interconnection_KAN',
       'Thermal_CHG', 'Water_CHG', 'Wind_CHG', 'Interconnection_CHG',
       'Nuclear_SHI', 'Wind_SHI', 'Interconnection_SHI', 'Nuclear_KYU',
       'Geothermal_KYU', 'PV_KYU', 'PVCurtailment_KYU', 'Wind_KYU',
       'WindCurtailment_KYU', 'Interconnection_KYU', 'Thermal_OKI',
       'Biomass_OKI', 'Wind_OKI', 'Allarea_TotalDemand', 'Allarea_Water',
       'Allarea_Thermal', 'Allarea_Geothermal', 'Allarea_Biomass',
       'Allarea_PumpedStorage', 'Allarea_Interconnection', 'Water_Ratio',
       'Thermal_Ratio', 'Geothermal_Ratio', 'Planned_Min_flag',
       'Planned_Max_flag', 'ForecastPV_Hokkaido', 'Forecast_PV_Chubu',
       'Forecast_Wind_Chubu', 'ForecastPV_Hokuriku', 'ForecastWind_Kansai',
       'Temp_Tokyo', 'SunLight(Time)_Tokyo', 'WindSpeed(m/s)_Tokyo',
       'WindDirection_Tokyo', 'SunLight(MJ/㎡)_Tokyo', 'Gas_Price', 'Oil_Price',
       'month', 'dayofweek', 'holiday', 'System_price(Yen/kWh)_lag_1_Day',
       'Price_Hokkaido(Yen/kWh)_lag_1_Day', 'Price_Tokyo(Yen/kWh)_lag_1_Day',
       'Price_Kansai(Yen/kWh)_lag_1_Day', 'BidExceed_diff_lag_1_Day',
       'Spot_MA25d_lag_1_Day', 'Spot_MA75d_lag_1_Day', 'Spot_MA200d_lag_1_D
ay',
       'VWAP_lag_1_Day', 'System_price(Yen/kWh)_lag_2_Day',
       'Price_Hokkaido(Yen/kWh)_lag_2_Day', 'Price_Tokyo(Yen/kWh)_lag_2_Day',
       'Price_Kansai(Yen/kWh)_lag_2_Day', 'BidExceed_diff_lag_2_Day',
       'Spot_MA25d_lag_2_Day', 'Spot_MA75d_lag_2_Day', 'Spot_MA200d_lag_2_D
ay',
       'VWAP_lag_2_Day'],
      dtype='object')
```

- Prediction point is at 10:00 on a day before the day-ahead market opens
- At least, daylag1 for Spot market informaiton, and daylag2 and 3 for Intra day market information are necessary
- Generage lagged features for any other information except for target and features that are not necessary.
  -->Drop Date, Time, HH, Min/Max_flag, date_block_num, month, dayofweek, holiday and lagged spot features.

In [9]:

```
# Drop columns that are not necessary to generate lag features
lag_columns = all_data_lagged_spot.columns.drop(['Date', 'Time', 'HH', 'Planned_Min_flag', 'Planned_Max_flag', 'month',
    'dayofweek', 'holiday', 'System_price(Yen/kWh)_lag_1_Day',
    'Price_Hokkaido(Yen/kWh)_lag_1_Day', 'Price_Tokyo(Yen/kWh)_lag_1_Day',
    'Price_Kansai(Yen/kWh)_lag_1_Day', 'BidExceed_diff_lag_1_Day',
    'Spot_MA25d_lag_1_Day', 'Spot_MA75d_lag_1_Day', 'Spot_MA200d_lag_1_Day',
    'VWAP_lag_1_Day', 'System_price(Yen/kWh)_lag_2_Day', 'Price_Hokkaido(Yen/kWh)_lag_2_Day',
    'Price_Tokyo(Yen/kWh)_lag_2_Day', 'Price_Kansai(Yen/kWh)_lag_2_Day',
    'BidExceed_diff_lag_2_Day', 'Spot_MA25d_lag_2_Day',
    'Spot_MA75d_lag_2_Day', 'Spot_MA200d_lag_2_Day', 'VWAP_lag_2_Day'])

# Generate lag features and drop original columns
all_data_lagged_Day = generate_lag(all_data, [2, 3], lag_columns, "Day")

# Join everything
all_data_lagged = pd.merge(all_data_lagged_spot, all_data_lagged_Day, how="left", left_index=True, right_index=True)

# Drop lag_columns which can be data leakage
all_data_lagged = all_data_lagged.drop(lag_columns, axis=1)

# Keep both the original and lagged "Close" --> Original: target / Lagged: feature
all_data_lagged["Close"] = all_data["Close"]

all_data_lagged.shape
```

Out[9]:

(83328, 161)

In [10]:

```
all_data_lagged.fillna(0, inplace=True)
print(all_data_lagged.isnull().sum())
```

Date 0
HH 0
Time 0
Planned_Min_flag 0
Planned_Max_flag 0
month 0
dayofweek 0
holiday 0
System_price(Yen/kWh)_lag_1_Day 0
Price_Hokkaido(Yen/kWh)_lag_1_Day 0
Price_Tokyo(Yen/kWh)_lag_1_Day 0
Price_Kansai(Yen/kWh)_lag_1_Day 0
BidExceed_diff_lag_1_Day 0
Spot_MA25d_lag_1_Day 0
Spot_MA75d_lag_1_Day 0
Spot_MA200d_lag_1_Day 0
VWAP_lag_1_Day 0
System_price(Yen/kWh)_lag_2_Day 0
Price_Hokkaido(Yen/kWh)_lag_2_Day 0
Price_Tokyo(Yen/kWh)_lag_2_Day 0
Price_Kansai(Yen/kWh)_lag_2_Day 0
BidExceed_diff_lag_2_Day 0
Spot_MA25d_lag_2_Day 0
Spot_MA75d_lag_2_Day 0
Spot_MA200d_lag_2_Day 0
VWAP_lag_2_Day 0
Open_lag_2_Day 0
High_lag_2_Day 0
Low_lag_2_Day 0
Close_lag_2_Day 0
Average_lag_2_Day 0
Volume(MWh/h)_lag_2_Day 0
Volume(Tick count)_lag_2_Day 0
HourlyDemand_Hokkaido_lag_2_Day 0
HourlyDemand_Tokyo_lag_2_Day 0
HourlyDemand_Kansai_lag_2_Day 0
HourlyDemand_Okinawa_lag_2_Day 0
Wind_TOH_lag_2_Day 0
Geothermal_TOH_lag_2_Day 0
PumpedStorage_TOH_lag_2_Day 0
Interconnection_TOH_lag_2_Day 0
Geothermal_HOK_lag_2_Day 0
Wind_HOK_lag_2_Day 0
Interconnection_HOK_lag_2_Day 0
Wind_TKO_lag_2_Day 0
Interconnection_TKO_lag_2_Day 0
Wind_CHB_lag_2_Day 0
Interconnection_CHB_lag_2_Day 0
Wind_HKU_lag_2_Day 0
PumpedStorage_HKU_lag_2_Day 0
Interconnection_HKU_lag_2_Day 0
Nuclear_KAN_lag_2_Day 0
Wind_KAN_lag_2_Day 0
Interconnection_KAN_lag_2_Day 0
Thermal_CHG_lag_2_Day 0
Water_CHG_lag_2_Day 0
Wind_CHG_lag_2_Day 0
Interconnection_CHG_lag_2_Day 0
Nuclear_SHI_lag_2_Day 0
Wind_SHI_lag_2_Day 0
Interconnection_SHI_lag_2_Day 0

Nuclear_KYU_lag_2_Day 0
Geothermal_KYU_lag_2_Day 0
PV_KYU_lag_2_Day 0
PVCurtailment_KYU_lag_2_Day 0
Wind_KYU_lag_2_Day 0
WindCurtailment_KYU_lag_2_Day 0
Interconnection_KYU_lag_2_Day 0
Thermal_OKI_lag_2_Day 0
Biomass_OKI_lag_2_Day 0
Wind_OKI_lag_2_Day 0
Allarea_TotalDemand_lag_2_Day 0
Allarea_Water_lag_2_Day 0
Allarea_Thermal_lag_2_Day 0
Allarea_Geothermal_lag_2_Day 0
Allarea_Biomass_lag_2_Day 0
Allarea_PumpedStorage_lag_2_Day 0
Allarea_Interconnection_lag_2_Day 0
Water_Ratio_lag_2_Day 0
Thermal_Ratio_lag_2_Day 0
Geothermal_Ratio_lag_2_Day 0
ForecastPV_Hokkaido_lag_2_Day 0
Forecast_PV_Chubu_lag_2_Day 0
Forecast_Wind_Chubu_lag_2_Day 0
ForecastPV_Hokuriku_lag_2_Day 0
ForecastWind_Kansai_lag_2_Day 0
Temp_Tokyo_lag_2_Day 0
SunLight(Time)_Tokyo_lag_2_Day 0
WindSpeed(m/s)_Tokyo_lag_2_Day 0
WindDirection_Tokyo_lag_2_Day 0
SunLight(MJ/㎡)_Tokyo_lag_2_Day 0
Gas_Price_lag_2_Day 0
Oil_Price_lag_2_Day 0
Open_lag_3_Day 0
High_lag_3_Day 0
Low_lag_3_Day 0
Close_lag_3_Day 0
Average_lag_3_Day 0
Volume(MWh/h)_lag_3_Day 0
Volume(Tick count)_lag_3_Day 0
HourlyDemand_Hokkaido_lag_3_Day 0
HourlyDemand_Tokyo_lag_3_Day 0
HourlyDemand_Kansai_lag_3_Day 0
HourlyDemand_Okinawa_lag_3_Day 0
Wind_TOH_lag_3_Day 0
Geothermal_TOH_lag_3_Day 0
PumpedStorage_TOH_lag_3_Day 0
Interconnection_TOH_lag_3_Day 0
Geothermal_HOK_lag_3_Day 0
Wind_HOK_lag_3_Day 0
Interconnection_HOK_lag_3_Day 0
Wind_TKO_lag_3_Day 0
Interconnection_TKO_lag_3_Day 0
Wind_CHB_lag_3_Day 0
Interconnection_CHB_lag_3_Day 0
Wind_HKU_lag_3_Day 0
PumpedStorage_HKU_lag_3_Day 0
Interconnection_HKU_lag_3_Day 0
Nuclear_KAN_lag_3_Day 0
Wind_KAN_lag_3_Day 0
Interconnection_KAN_lag_3_Day 0
Thermal_CHG_lag_3_Day 0

```
Water_CHG_lag_3_Day                0
Wind_CHG_lag_3_Day                 0
Interconnection_CHG_lag_3_Day      0
Nuclear_SHI_lag_3_Day              0
Wind_SHI_lag_3_Day                 0
Interconnection_SHI_lag_3_Day      0
Nuclear_KYU_lag_3_Day              0
Geothermal_KYU_lag_3_Day           0
PV_KYU_lag_3_Day                   0
PVCurtailment_KYU_lag_3_Day        0
Wind_KYU_lag_3_Day                 0
WindCurtailment_KYU_lag_3_Day      0
Interconnection_KYU_lag_3_Day      0
Thermal_OKI_lag_3_Day              0
Biomass_OKI_lag_3_Day              0
Wind_OKI_lag_3_Day                 0
Allarea_TotalDemand_lag_3_Day      0
Allarea_Water_lag_3_Day            0
Allarea_Thermal_lag_3_Day          0
Allarea_Geothermal_lag_3_Day       0
Allarea_Biomass_lag_3_Day          0
Allarea_PumpedStorage_lag_3_Day    0
Allarea_Interconnection_lag_3_Day  0
Water_Ratio_lag_3_Day              0
Thermal_Ratio_lag_3_Day            0
Geothermal_Ratio_lag_3_Day         0
ForecastPV_Hokkaido_lag_3_Day      0
Forecast_PV_Chubu_lag_3_Day        0
Forecast_Wind_Chubu_lag_3_Day      0
ForecastPV_Hokuriku_lag_3_Day      0
ForecastWind_Kansai_lag_3_Day      0
Temp_Tokyo_lag_3_Day               0
SunLight(Time)_Tokyo_lag_3_Day     0
WindSpeed(m/s)_Tokyo_lag_3_Day     0
WindDirection_Tokyo_lag_3_Day      0
SunLight(MJ/㎡)_Tokyo_lag_3_Day    0
Gas_Price_lag_3_Day                0
Oil_Price_lag_3_Day                0
Close                              0
dtype: int64
```

In [11]:

```
all_data_lagged.tail()
```

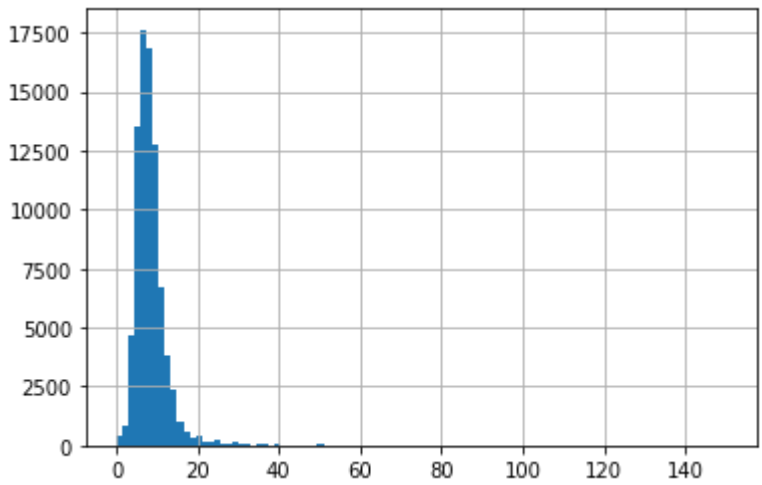
Out[11]:

	Date	HH	Time	Planned_Min_flag	Planned_Max_flag	month	dayofweek	holiday
83323	2020-12-31	44	21:30:00	0.0	0.0	12	3	0
83324	2020-12-31	45	22:00:00	0.0	0.0	12	3	0
83325	2020-12-31	46	22:30:00	0.0	0.0	12	3	0
83326	2020-12-31	47	23:00:00	0.0	0.0	12	3	0
83327	2020-12-31	48	23:30:00	0.0	0.0	12	3	0

Standardise with Log-features

In [12]:

```
all_data_lagged["Close"].hist(bins=100);
```



Features seem to be positive skew, and have different scale. --> Need to standardise

In [13]:

```
all_data_lagged.columns
```

Out[13]:

```
Index(['Date', 'HH', 'Time', 'Planned_Min_flag', 'Planned_Max_flag', 'month',
       'dayofweek', 'holiday', 'System_price(Yen/kWh)_lag_1_Day',
       'Price_Hokkaido(Yen/kWh)_lag_1_Day',
       ...
       'ForecastPV_Hokuriku_lag_3_Day', 'ForecastWind_Kansai_lag_3_Day',
       'Temp_Tokyo_lag_3_Day', 'SunLight(Time)_Tokyo_lag_3_Day',
       'WindSpeed(m/s)_Tokyo_lag_3_Day', 'WindDirection_Tokyo_lag_3_Day',
       'SunLight(MJ/㎡)_Tokyo_lag_3_Day', 'Gas_Price_lag_3_Day',
       'Oil_Price_lag_3_Day', 'Close'],
      dtype='object', length=161)
```

In [14]:

```
# Divide the data into two group: One is not necessary to be lagged; another is necessary to be lagge
d
all_data_lagged1 = all_data_lagged[['Date', 'Time', 'HH', 'month', 'dayofweek', 'holiday', 'Planned_M
in_flag', 'Planned_Max_flag']]
all_data_lagged2 = all_data_lagged.drop(['Date', 'Time', 'HH', 'month', 'dayofweek', 'holiday', 'Plann
ed_Min_flag', 'Planned_Max_flag'], axis=1)
```

In [15]:

```
# # logarithmic transformation for standardised
from sklearn.preprocessing import PowerTransformer
def trans_yeo_johnson(df, df2):
    pt = PowerTransformer() #default: Yeo-Johnson (this allow also 0-value to transform to logarithmi
c value)
    pt.fit_transform(df)
    return pt.transform(df2)

def inverse_trans_yeo_johnson(df, df2):
    pt = PowerTransformer() #default: Yeo-Johnson (this allow also 0-value to transform to logarithmi
c value)
    pt.fit_transform(df)
    return pt.inverse_transform(df2)
```

In [16]:

```
# Lag transform
all_data_log2_transformed = all_data_lagged2.copy()
cols = all_data_log2_transformed.columns

all_data_log2_transformed = trans_yeo_johnson(all_data_lagged2, all_data_log2_transformed)
all_data_log2_transformed = pd.DataFrame(all_data_log2_transformed)
all_data_log2_transformed.columns = cols

all_data_log = pd.concat([all_data_lagged1, all_data_log2_transformed], axis=1)

all_data_log.tail()
```

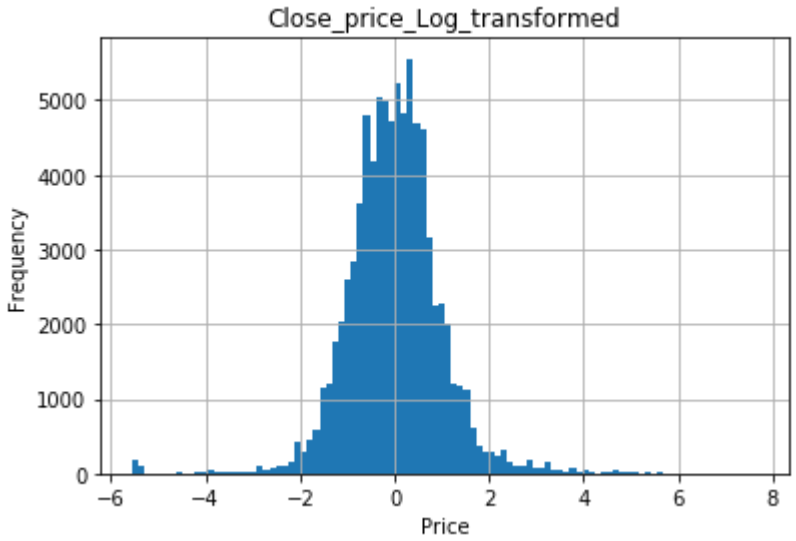
Out[16]:

	Date	Time	HH	month	dayofweek	holiday	Planned_Min_flag	Planned_Max_flag
83323	2020-12-31	21:30:00	44	12	3	0	0.0	0.0
83324	2020-12-31	22:00:00	45	12	3	0	0.0	0.0
83325	2020-12-31	22:30:00	46	12	3	0	0.0	0.0
83326	2020-12-31	23:00:00	47	12	3	0	0.0	0.0
83327	2020-12-31	23:30:00	48	12	3	0	0.0	0.0

In [17]:

```
# Plot the lagged target
fig, ax = plt.subplots(1, figsize=plt.figaspect(.65))

all_data_log["Close"].hist(bins=100);
print("Skew: %f" % all_data_log["Close"].skew())
print("Kurt: %f" % all_data_log["Close"].kurt())
print("Mean: %f" % all_data_log["Close"].mean())
print("Std: %f" % all_data_log["Close"].std())
ax.set(title="Close_price_Log_transformed", ylabel="Frequency", xlabel="Price");
```

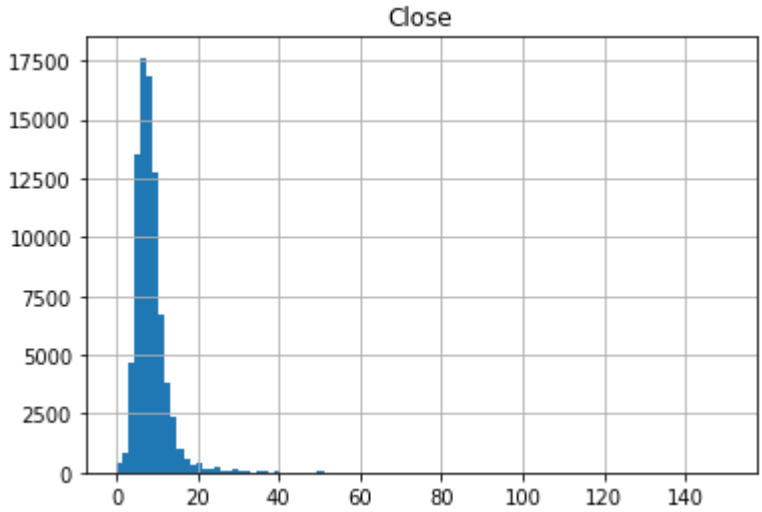
Skew: 0.016581
Kurt: 5.092394
Mean: -0.000000
Std: 1.000006



※ Confirm whether inverse transform works for Close price appropriately, or not

In [18]:

```
# Try to see whether inverse_trans_yeo_johnson works well (Only for target)
y_log_inversed = inverse_trans_yeo_johnson(all_data_lagged2["Close"].values.reshape(-1,1), all_d
ata_log["Close"].values.reshape(-1, 1))
y_log_inversed = pd.DataFrame(y_log_inversed)
y_log_inversed.columns = ["Close"]
y_log_inversed.hist(bins=100);
```

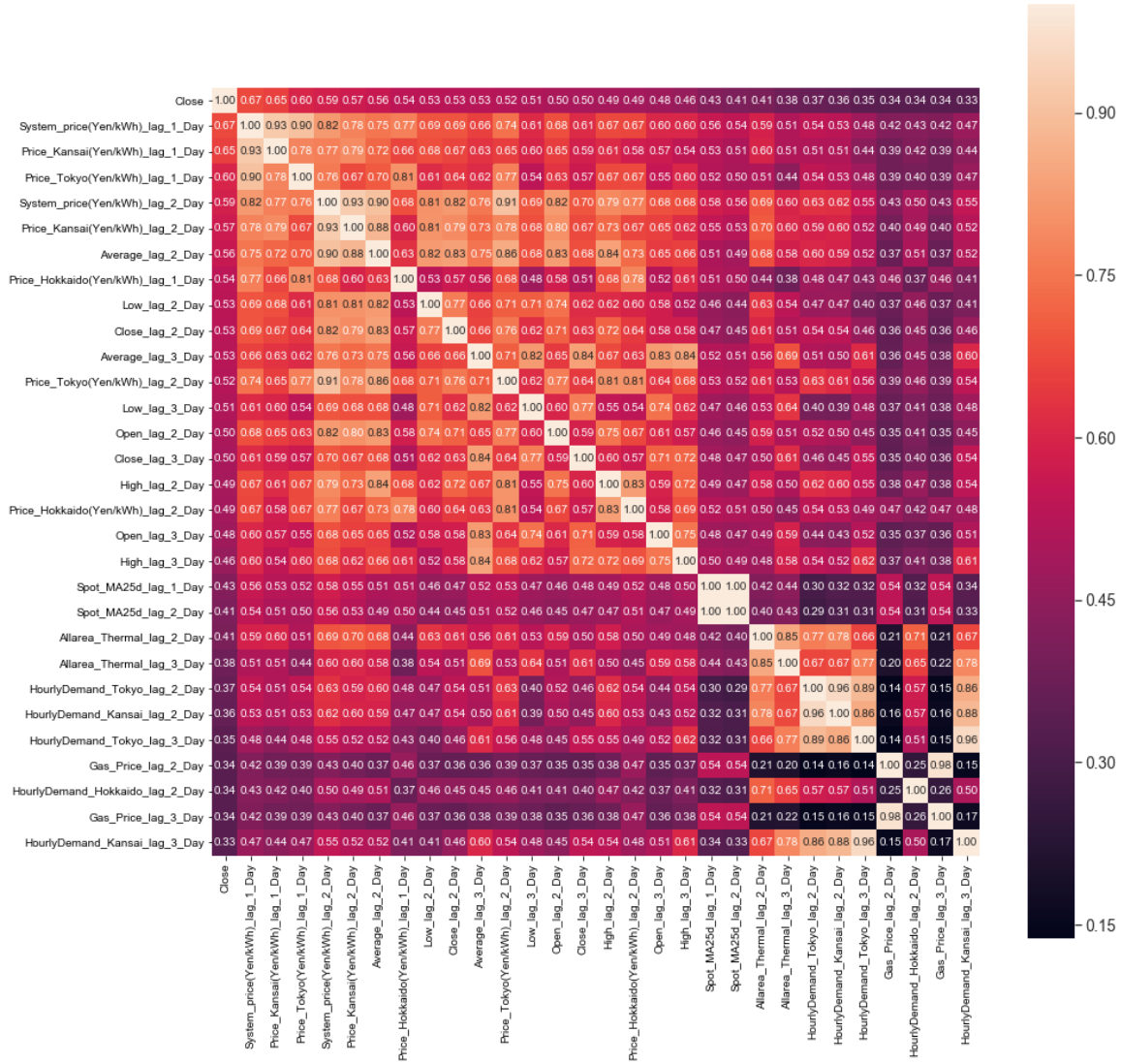


EDA (for all_data)

Check the correlation of features with the target

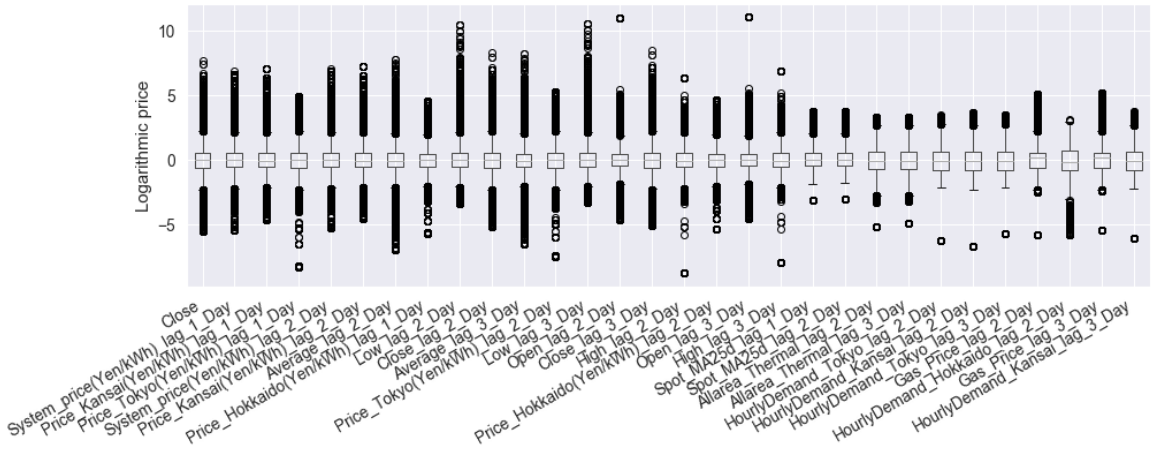
In [19]:

```
# correlation matrix
corrmat = all_data_log.corr()

# revenue correlation matrix
k = 30 # The number of variables on the heatmap
cols = corrmat.nlargest(k, 'Close')['Close'].index
cm = np.corrcoef(all_data_log[cols].values.T)
f, ax = plt.subplots(figsize=(15, 15))
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, ytic
klabels=cols.values, xticklabels=cols.values)
plt.show()
```



In [20]:

```
fig, ax = plt.subplots(1, figsize=(15, 5))
all_data_log[cols].boxplot()
plt.gcf().autofmt_xdate()
ax.set(ylabel="Logarithmic price");
plt.show()
```



In [21]:

```
all_data_log.tail()
```

Out[21]:

	Date	Time	HH	month	dayofweek	holiday	Planned_Min_flag	Planned_Max_flag
83323	2020-12-31	21:30:00	44	12	3	0	0.0	0.0
83324	2020-12-31	22:00:00	45	12	3	0	0.0	0.0
83325	2020-12-31	22:30:00	46	12	3	0	0.0	0.0
83326	2020-12-31	23:00:00	47	12	3	0	0.0	0.0
83327	2020-12-31	23:30:00	48	12	3	0	0.0	0.0

Training Models with "all_data"

Preparation

In [22]:

```
X = all_data_log.drop(['Close', 'Date', 'Time'], axis=1)
y = all_data_log.Closey = all_data_log.Close
```

Split all_data into train and valid for validation

In [23]:

```
# Rolling/Walk forward validation (Timeseries validation with the parameter of "max_train_size")
from sklearn.model_selection import TimeSeriesSplit

n_splits=10
train_ratio=8
test_ratio=2
max_train_size= 48*365
test_size=int(round((len(X)*test_ratio)/(train_ratio+test_ratio*n_splits), 0))

tscv = TimeSeriesSplit(n_splits=n_splits, test_size=test_size
                       , max_train_size=max_train_size
                      )
Min_valid_index = len(X) - (n_splits * test_size)

print(tscv)
print('Minimum of valid_index: %.0f' % Min_valid_index)

# Confirming the split logic
for train_index, valid_index in tscv.split(X):
    # Divide the train/valid set  into 10 folds and pick up it.
    X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
    y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]
    print("TRAIN:", train_index, "Valid:", valid_index)
```

```
TimeSeriesSplit(gap=0, max_train_size=17520, n_splits=10, test_size=5952)
Minimum of valid_index: 23808
TRAIN: [ 6288  6289  6290 ... 23805 23806 23807] Valid: [23808 23809 23810 ...
29757 29758 29759]
TRAIN: [12240 12241 12242 ... 29757 29758 29759] Valid: [29760 29761 29762
... 35709 35710 35711]
TRAIN: [18192 18193 18194 ... 35709 35710 35711] Valid: [35712 35713 35714
... 41661 41662 41663]
TRAIN: [24144 24145 24146 ... 41661 41662 41663] Valid: [41664 41665 41666
... 47613 47614 47615]
TRAIN: [30096 30097 30098 ... 47613 47614 47615] Valid: [47616 47617 47618
... 53565 53566 53567]
TRAIN: [36048 36049 36050 ... 53565 53566 53567] Valid: [53568 53569 53570
... 59517 59518 59519]
TRAIN: [42000 42001 42002 ... 59517 59518 59519] Valid: [59520 59521 59522
... 65469 65470 65471]
TRAIN: [47952 47953 47954 ... 65469 65470 65471] Valid: [65472 65473 65474
... 71421 71422 71423]
TRAIN: [53904 53905 53906 ... 71421 71422 71423] Valid: [71424 71425 71426
... 77373 77374 77375]
TRAIN: [59856 59857 59858 ... 77373 77374 77375] Valid: [77376 77377 77378
... 83325 83326 83327]
```

Make the tables for graph visualisation and for evaluation results

In [24]:

```
# DataTable for graph_log
graph_data_log = all_data_log[["Date", "Time", "Close"]]
graph_data_log["DateTime"] = pd.to_datetime(graph_data_log["Date"].astype(str) + " " + graph_
data_log["Time"].astype(str))
prediction_point = graph_data_log["DateTime"][graph_data_log.index==Min_valid_index].iat[-1]
graph_data_log = graph_data_log.drop(["Date", "Time"], axis=1)
print("Prediction_point: {}".format(prediction_point))

# DataTable for Evaluation results_log
Eval_table_log = pd.DataFrame()
Eval_table_log["EvalFunc"] = pd.Series(["RMSE_log", "MAE_log"])
Eval_table_log
```

Prediction_point: 2017-08-10 00:00:00

Out[24]:

	EvalFunc
0	RMSE_log
1	MAE_log

In [25]:

```
# DataTable for graph_original
graph_data_original = all_data_lagged[["Date", "Time", "Close"]]
graph_data_original["DateTime"] = pd.to_datetime(graph_data_original["Date"].astype(str) + " " +
graph_data_original["Time"].astype(str))
prediction_point = graph_data_original["DateTime"][graph_data_original.index==Min_valid_index].i
at[-1]
graph_data_original = graph_data_original.drop(["Date", "Time"], axis=1)
print("Prediction_point: {}".format(prediction_point))

# DataTable for Evaluation functions_original
Eval_table_original = pd.DataFrame()
Eval_table_original["EvalFunc"] = pd.Series(["RMSE_Yen/kWh", "MAE_Yen/kWh"])
Eval_table_original
```

Prediction_point: 2017-08-10 00:00:00

Out[25]:

	EvalFunc
0	RMSE_Yen/kWh
1	MAE_Yen/kWh

Lasso

In [26]:

```
from sklearn.linear_model import Lasso

for train_index, valid_index in tscv.split(X):
    # Divide the train/valid set  into 10 folds and pick up it.
    X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
    y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]

    # For the test of alpha
    for alpha in [0.02, 0.5, 1]: #0.02 is the best (0.02, 0.5, 1)
        modelLasso = Lasso(alpha=alpha).fit(X_train, y_train)
        print("\n alpha={}".format(str(alpha)))
        print("Train set score: {:.2f}".format(modelLasso.score(X_train, y_train)))
        print("Test set score: {:.2f}".format(modelLasso.score(X_valid, y_valid)))
        y_pred = modelLasso.predict(X_valid)
        print("RMSE: {:.2f}".format(sqrt(mean_squared_error(y_valid, y_pred))) )
        print("MAE: {:.2f}".format(mean_absolute_error(y_valid, y_pred)))
        print("Number of features used:{}".format(np.sum(modelLasso.coef_ != 0)))
```

alpha=0.02
Train set score: 0.51
Test set score: 0.32
RMSE: 0.65
MAE: 0.47
Number of features used:23

alpha=0.5
Train set score: 0.08
Test set score: 0.04
RMSE: 0.77
MAE: 0.58
Number of features used:2

alpha=1
Train set score: 0.05
Test set score: 0.03
RMSE: 0.78
MAE: 0.59
Number of features used:1

alpha=0.02
Train set score: 0.44
Test set score: 0.44
RMSE: 0.67
MAE: 0.50
Number of features used:25

alpha=0.5
Train set score: 0.05
Test set score: -0.37
RMSE: 1.05
MAE: 0.79
Number of features used:1

alpha=1
Train set score: 0.04
Test set score: -0.37
RMSE: 1.06
MAE: 0.79
Number of features used:1

alpha=0.02
Train set score: 0.53
Test set score: 0.44
RMSE: 0.65
MAE: 0.47
Number of features used:22

alpha=0.5
Train set score: 0.05
Test set score: 0.04
RMSE: 0.85
MAE: 0.59
Number of features used:2

alpha=1
Train set score: 0.04
Test set score: 0.04
RMSE: 0.85
MAE: 0.59

Number of features used:1

alpha=0.02
Train set score: 0.51
Test set score: 0.30
RMSE: 0.56
MAE: 0.44
Number of features used:28

alpha=0.5
Train set score: 0.06
Test set score: 0.05
RMSE: 0.65
MAE: 0.50
Number of features used:2

alpha=1
Train set score: 0.04
Test set score: 0.05
RMSE: 0.65
MAE: 0.50
Number of features used:1

alpha=0.02
Train set score: 0.49
Test set score: 0.10
RMSE: 0.77
MAE: 0.57
Number of features used:21

alpha=0.5
Train set score: 0.05
Test set score: -0.25
RMSE: 0.91
MAE: 0.67
Number of features used:1

alpha=1
Train set score: 0.04
Test set score: -0.25
RMSE: 0.91
MAE: 0.67
Number of features used:1

alpha=0.02
Train set score: 0.40
Test set score: 0.36
RMSE: 0.68
MAE: 0.49
Number of features used:22

alpha=0.5
Train set score: 0.05
Test set score: -0.23
RMSE: 0.94
MAE: 0.70
Number of features used:1

alpha=1
Train set score: 0.04
Test set score: -0.24

RMSE: 0.95
MAE: 0.70
Number of features used:1

alpha=0.02
Train set score: 0.41
Test set score: 0.27
RMSE: 0.71
MAE: 0.47
Number of features used:29

alpha=0.5
Train set score: 0.04
Test set score: -0.01
RMSE: 0.83
MAE: 0.55
Number of features used:1

alpha=1
Train set score: 0.04
Test set score: -0.01
RMSE: 0.83
MAE: 0.55
Number of features used:1

alpha=0.02
Train set score: 0.35
Test set score: 0.26
RMSE: 0.84
MAE: 0.58
Number of features used:22

alpha=0.5
Train set score: 0.04
Test set score: -0.30
RMSE: 1.11
MAE: 0.78
Number of features used:1

alpha=1
Train set score: 0.04
Test set score: -0.30
RMSE: 1.11
MAE: 0.78
Number of features used:1

alpha=0.02
Train set score: 0.40
Test set score: 0.34
RMSE: 0.97
MAE: 0.64
Number of features used:29

alpha=0.5
Train set score: 0.04
Test set score: -0.45
RMSE: 1.44
MAE: 1.07
Number of features used:2

alpha=1

Train set score: 0.03
Test set score: -0.46
RMSE: 1.45
MAE: 1.07
Number of features used:1

alpha=0.02
Train set score: 0.46
Test set score: 0.55
RMSE: 0.90
MAE: 0.62
Number of features used:30

alpha=0.5
Train set score: 0.19
Test set score: 0.23
RMSE: 1.18
MAE: 0.77
Number of features used:3

alpha=1
Train set score: 0.02
Test set score: 0.01
RMSE: 1.34
MAE: 0.87
Number of features used:1

0.02 is the optimal

In [27]:

```
from sklearn import linear_model
import itertools

training_accuracy = []
valid_accuracy = []
rmse = []
mae = []
prediction_Lasso = []

for train_index, valid_index in tscv.split(X):
    # Divide the train/valid set  into 10 folds and pick up it.
    X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
    y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]

    #Fit train set to the model (Chooose one model)
    modelLasso = linear_model.Lasso(alpha=0.02).fit(X_train, y_train)

    # Generate prediction results
    y_pred = modelLasso.predict(X_valid)
    true_values = y_valid.values
    # Save prediction results
    prediction_Lasso.append(y_pred)
    # Save evaluation results for each 10 validation and get mean
    training_accuracy.append(modelLasso.score(X_train, y_train))
    valid_accuracy.append(modelLasso.score(X_valid, y_valid))
    rmse.append(sqrt(mean_squared_error(true_values, y_pred)))
    mae.append(mean_absolute_error(true_values, y_pred))

# print("Training_accuracy: {}".format(training_accuracy))
print("Training_accuracy: {}".format(np.mean(training_accuracy)))
# print("Valid_accuracy: {}".format(valid_accuracy))
print("Valid_accuracy: {}".format(np.mean(valid_accuracy)))

# print("RMSE: {}".format(rmse))
print("RMSE: {}".format(np.mean(rmse)))
# print("MAE: {}".format(mae))
print("MAE: {}".format(np.mean(mae)))

# Convert prediction results with valid data from 2D list to 1D list
prediction_Lasso = list(itertools.chain.from_iterable(prediction_Lasso))
#  Get prediction with train data
modelLasso = linear_model.Lasso(alpha=0.02).fit(X, y)
y_pred_train = list(modelLasso.predict(X)[:Min_valid_index])
# Store the prediction into the "graph data" table
graph_data_log["Close_pred_Lasso"] = pd.Series(y_pred_train + prediction_Lasso)
# Store the result of evaluation into the "Eval_table"
Eval_table_log["Lasso"] = pd.Series([np.mean(rmse), np.mean(mae)])
rmse_10fold_Lasso = pd.Series(rmse)
mae_10fold_Lasso = pd.Series(mae)
```

```
Training_accuracy: 0.45044087579550823
Valid_accuracy: 0.3372356817348764
RMSE: 0.741565799978881
MAE: 0.5246821259945508
```

Interpretation of the linear model

https://scikit-learn.org/stable/auto_examples/inspection/plot_linear_model_coefficient_interpretation.html (https://scikit-learn.org/stable/auto_examples/inspection/plot_linear_model_coefficient_interpretation.html)

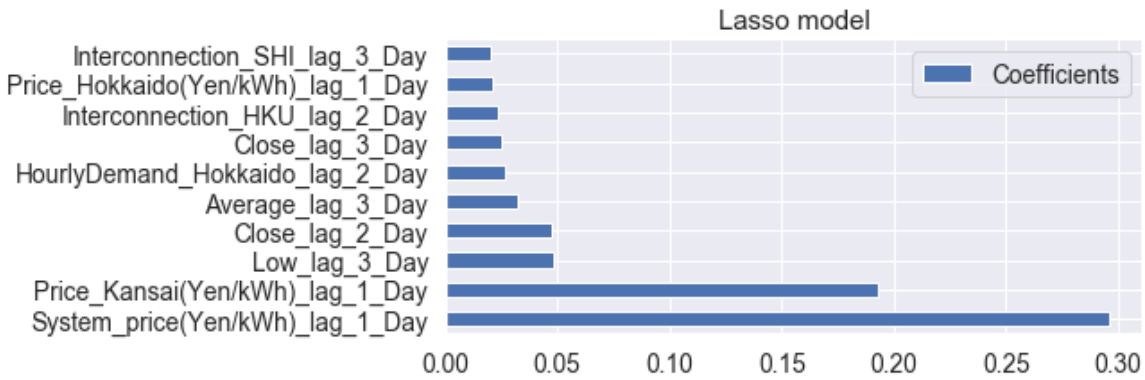
In [28]:

```
print("Intercept: {}".format(modelLasso.intercept_))

feature_names = X_train.columns
coefs = pd.DataFrame(modelLasso.coef_, columns=['Coefficients'], index=feature_names
).sort_values('Coefficients', ascending=False)

# Absolute value of coefficients
coef_abs = coefs.abs()
print(coef_abs.head(10))
```

```
Intercept: 0.23724161184507364
                                  Coefficients
System_price(Yen/kWh)_lag_1_Day       0.296036
Price_Kansai(Yen/kWh)_lag_1_Day       0.192718
Low_lag_3_Day                         0.048186
Close_lag_2_Day                       0.047103
Average_lag_3_Day                     0.032154
HourlyDemand_Hokkaido_lag_2_Day       0.026912
Close_lag_3_Day                       0.024976
Interconnection_HKU_lag_2_Day         0.023100
Price_Hokkaido(Yen/kWh)_lag_1_Day     0.020751
Interconnection_SHI_lag_3_Day         0.020183
```

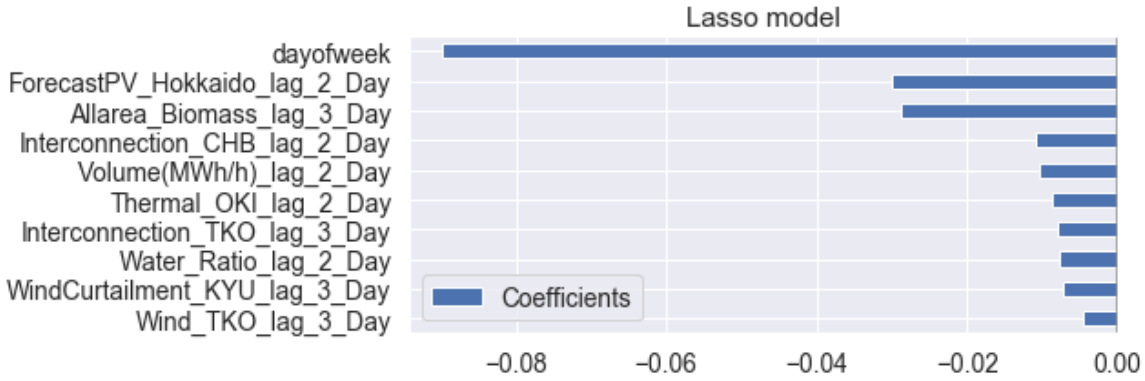
In [29]:

```
# Positive coeficients
coefs[:10].plot(kind='barh', figsize=(9, 3))
plt.title('Lasso model')
plt.axvline(x=0, color='.5')
plt.subplots_adjust(left=.3)
```



In [30]:

```
# Negative coeficients
end = len(coefs)
start = end - 10
coefs[start:end].plot(kind='barh', figsize=(9, 3))
plt.title('Lasso model')
plt.axvline(x=0, color='.5')
plt.subplots_adjust(left=.3)
```



In [31]:

```
coef_abs
```

Out[31]:

	Coefficients
System_price(Yen/kWh)_lag_1_Day	0.296036
Price_Kansai(Yen/kWh)_lag_1_Day	0.192718
Low_lag_3_Day	0.048186
Close_lag_2_Day	0.047103
Average_lag_3_Day	0.032154
HourlyDemand_Hokkaido_lag_2_Day	0.026912
Close_lag_3_Day	0.024976
Interconnection_HKU_lag_2_Day	0.023100
Price_Hokkaido(Yen/kWh)_lag_1_Day	0.020751
Interconnection_SHI_lag_3_Day	0.020183
Oil_Price_lag_2_Day	0.019116
Low_lag_2_Day	0.017608
SunLight(Time)_Tokyo_lag_2_Day	0.010257
Average_lag_2_Day	0.009995
BidExceed_diff_lag_2_Day	0.008493
Planned_Max_flag	0.002978
HH	0.001628
PumpedStorage_TOH_lag_3_Day	0.000000
Interconnection_CHB_lag_3_Day	0.000000
PumpedStorage_HKU_lag_3_Day	0.000000
Wind_CHB_lag_3_Day	0.000000
Interconnection_HOK_lag_3_Day	0.000000
Wind_HOK_lag_3_Day	0.000000
Geothermal_HOK_lag_3_Day	0.000000
Interconnection_TOH_lag_3_Day	0.000000
Interconnection_HKU_lag_3_Day	0.000000
Nuclear_KAN_lag_3_Day	0.000000
Wind_KAN_lag_3_Day	0.000000
Wind_HKU_lag_3_Day	0.000000
HourlyDemand_Kansai_lag_3_Day	0.000000
Geothermal_TOH_lag_3_Day	0.000000
Wind_TOH_lag_3_Day	0.000000
HourlyDemand_Okinawa_lag_3_Day	0.000000
Thermal_CHG_lag_3_Day	0.000000
HourlyDemand_Tokyo_lag_3_Day	0.000000
HourlyDemand_Hokkaido_lag_3_Day	0.000000

	Coefficients
Volume(Tick count)_lag_3_Day	0.000000
Volume(MWh/h)_lag_3_Day	0.000000
High_lag_3_Day	0.000000
Open_lag_3_Day	0.000000
Gas_Price_lag_2_Day	0.000000
SunLight(MJ/㎡)_Tokyo_lag_2_Day	0.000000
WindDirection_Tokyo_lag_2_Day	0.000000
WindSpeed(m/s)_Tokyo_lag_2_Day	0.000000
Temp_Tokyo_lag_2_Day	0.000000
Interconnection_KAN_lag_3_Day	0.000000
Wind_SHI_lag_3_Day	0.000000
Water_CHG_lag_3_Day	0.000000
ForecastPV_Hokuriku_lag_3_Day	0.000000
Water_Ratio_lag_3_Day	0.000000
Thermal_Ratio_lag_3_Day	0.000000
Geothermal_Ratio_lag_3_Day	0.000000
ForecastPV_Hokkaido_lag_3_Day	0.000000
Forecast_PV_Chubu_lag_3_Day	0.000000
Forecast_Wind_Chubu_lag_3_Day	0.000000
ForecastWind_Kansai_lag_3_Day	0.000000
Allarea_PumpedStorage_lag_3_Day	0.000000
Temp_Tokyo_lag_3_Day	0.000000
SunLight(Time)_Tokyo_lag_3_Day	0.000000
WindSpeed(m/s)_Tokyo_lag_3_Day	0.000000
WindDirection_Tokyo_lag_3_Day	0.000000
SunLight(MJ/㎡)_Tokyo_lag_3_Day	0.000000
Gas_Price_lag_3_Day	0.000000
Allarea_Interconnection_lag_3_Day	0.000000
Allarea_Geothermal_lag_3_Day	0.000000
Wind_CHG_lag_3_Day	0.000000
PVCurtailment_KYU_lag_3_Day	0.000000
Interconnection_CHG_lag_3_Day	0.000000
Nuclear_SHI_lag_3_Day	0.000000
ForecastPV_Hokuriku_lag_2_Day	0.000000
Nuclear_KYU_lag_3_Day	0.000000
Geothermal_KYU_lag_3_Day	0.000000
PV_KYU_lag_3_Day	0.000000

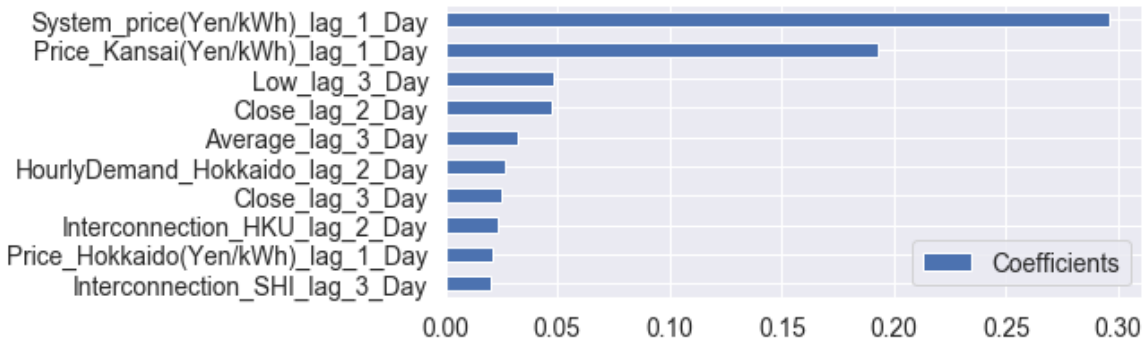
	Coefficients
Wind_KYU_lag_3_Day	0.000000
Allarea_Thermal_lag_3_Day	0.000000
Interconnection_KYU_lag_3_Day	0.000000
Thermal_OKI_lag_3_Day	0.000000
Biomass_OKI_lag_3_Day	0.000000
Wind_OKI_lag_3_Day	0.000000
Allarea_TotalDemand_lag_3_Day	0.000000
Allarea_Water_lag_3_Day	0.000000
ForecastWind_Kansai_lag_2_Day	0.000000
Oil_Price_lag_3_Day	0.000000
Forecast_Wind_Chubu_lag_2_Day	0.000000
Interconnection_TOH_lag_2_Day	0.000000
HourlyDemand_Tokyo_lag_2_Day	0.000000
HourlyDemand_Kansai_lag_2_Day	0.000000
HourlyDemand_Okinawa_lag_2_Day	0.000000
Wind_TOH_lag_2_Day	0.000000
Geothermal_TOH_lag_2_Day	0.000000
PumpedStorage_TOH_lag_2_Day	0.000000
Geothermal_HOK_lag_2_Day	0.000000
Planned_Min_flag	0.000000
Wind_HOK_lag_2_Day	0.000000
Interconnection_HOK_lag_2_Day	0.000000
Wind_TKO_lag_2_Day	0.000000
Interconnection_TKO_lag_2_Day	0.000000
Wind_CHB_lag_2_Day	0.000000
Forecast_PV_Chubu_lag_2_Day	0.000000
Volume(Tick count)_lag_2_Day	0.000000
High_lag_2_Day	0.000000
Open_lag_2_Day	0.000000
VWAP_lag_2_Day	0.000000
holiday	0.000000
Spot_MA200d_lag_2_Day	0.000000
Spot_MA75d_lag_2_Day	0.000000
Spot_MA25d_lag_2_Day	0.000000
Price_Kansai(Yen/kWh)_lag_2_Day	0.000000
Price_Tokyo(Yen/kWh)_lag_2_Day	0.000000
Price_Hokkaido(Yen/kWh)_lag_2_Day	0.000000

	Coefficients
System_price(Yen/kWh)_lag_2_Day	0.000000
VWAP_lag_1_Day	0.000000
Spot_MA200d_lag_1_Day	0.000000
Spot_MA75d_lag_1_Day	0.000000
Spot_MA25d_lag_1_Day	0.000000
BidExceed_diff_lag_1_Day	0.000000
Wind_HKU_lag_2_Day	0.000000
PumpedStorage_HKU_lag_2_Day	0.000000
Nuclear_KAN_lag_2_Day	0.000000
PVCurtailment_KYU_lag_2_Day	0.000000
Thermal_Ratio_lag_2_Day	0.000000
Allarea_Interconnection_lag_2_Day	0.000000
Allarea_PumpedStorage_lag_2_Day	0.000000
Allarea_Biomass_lag_2_Day	0.000000
Allarea_Thermal_lag_2_Day	0.000000
Allarea_Water_lag_2_Day	0.000000
Allarea_TotalDemand_lag_2_Day	0.000000
Wind_OKI_lag_2_Day	0.000000
Biomass_OKI_lag_2_Day	0.000000
Interconnection_KYU_lag_2_Day	0.000000
WindCurtailment_KYU_lag_2_Day	0.000000
Wind_KAN_lag_2_Day	0.000000
Wind_KYU_lag_2_Day	0.000000
Price_Tokyo(Yen/kWh)_lag_1_Day	0.000000
PV_KYU_lag_2_Day	0.000000
Nuclear_SHI_lag_2_Day	0.000000
Interconnection_KAN_lag_2_Day	0.000000
Thermal_CHG_lag_2_Day	0.000000
Water_CHG_lag_2_Day	0.000000
Wind_CHG_lag_2_Day	0.000000
Geothermal_KYU_lag_2_Day	0.000000
Interconnection_CHG_lag_2_Day	0.000000
Wind_SHI_lag_2_Day	0.000000
Interconnection_SHI_lag_2_Day	0.000000
Nuclear_KYU_lag_2_Day	0.000000
Allarea_Geothermal_lag_2_Day	0.000713
month	0.001267

	Coefficients
Geothermal_Ratio_lag_2_Day	0.004144
Wind_TKO_lag_3_Day	0.004522
WindCurtailment_KYU_lag_3_Day	0.007118
Water_Ratio_lag_2_Day	0.007697
Interconnection_TKO_lag_3_Day	0.008019
Thermal_OKI_lag_2_Day	0.008691
Volume(MWh/h)_lag_2_Day	0.010262
Interconnection_CHB_lag_2_Day	0.010881
Allarea_Biomass_lag_3_Day	0.028812
ForecastPV_Hokkaido_lag_2_Day	0.029951
dayofweek	0.089712

In [32]:

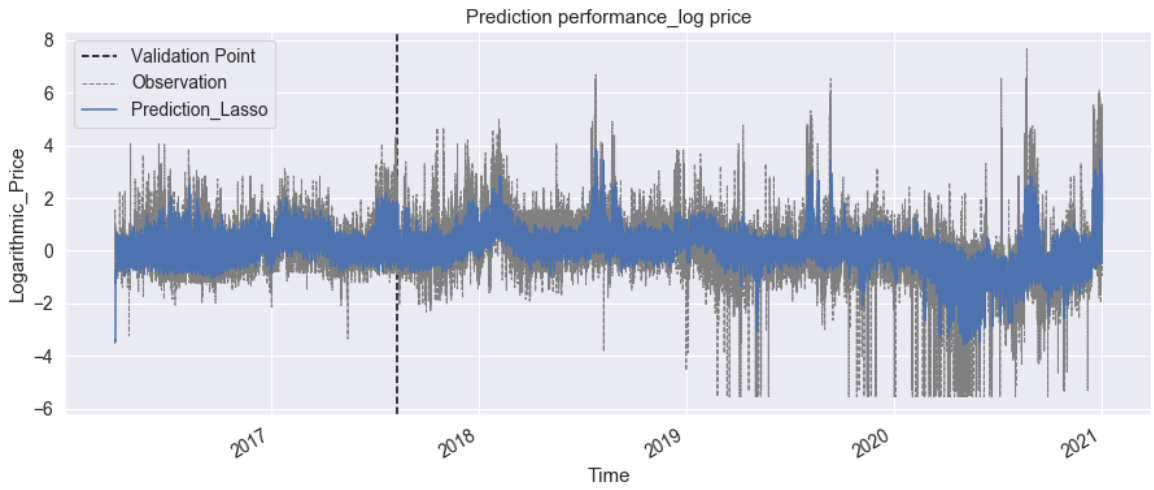
```
# Absolute coeficients
coef_abs[:10].sort_values("Coefficients", ascending=True).plot(kind='barh', figsize=(9, 3))
# plt.title('Lasso model')
plt.axvline(x=0, color='.5')
plt.subplots_adjust(left=.3)
```



Visualisation of the model performance

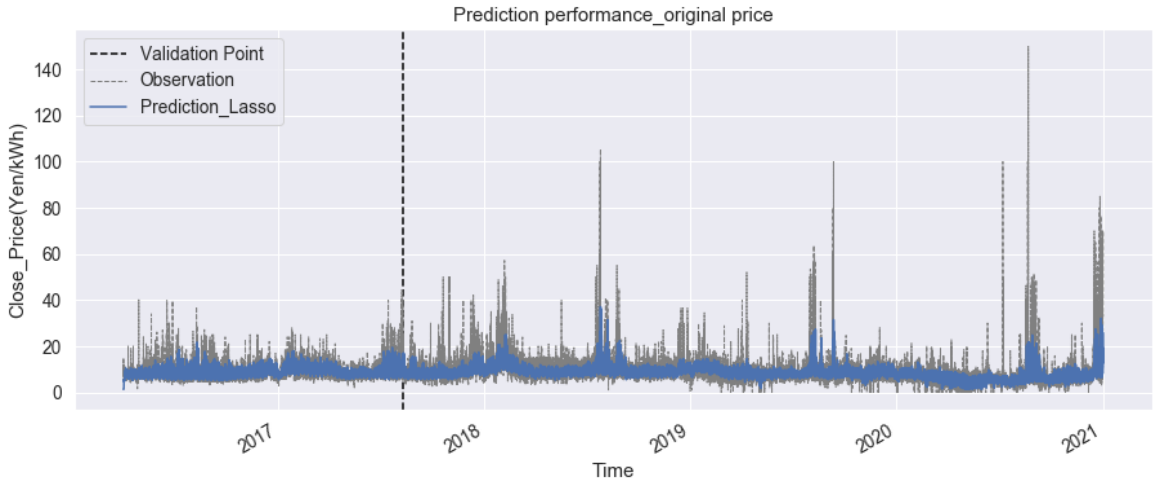
In [33]:

```
graph_data_log = graph_data_log.reset_index()

#対数化されたターゲットと予測値の比較

Models = ["Lasso"]

fig, ax = plt.subplots(1, figsize=(15, 6))
plt.title('Prediction performance_log price')

# Set index
graph_data_log = graph_data_log.set_index("DateTime")
start = "2016-04-01 00:00:00"
end = "2020-12-31 23:30:00"

# Vertical line (need to convert the date type from timestamp to datetime.datetime as x-axis)
plt.axvline(prediction_point.to_pydatetime(), label="Validation Point", linestyle="dashed", color="bl
ack")

# Plot Close
graph_data_log.Close[graph_data_log.index > start].plot(ax=ax, label="Observation", linestyle="da
shed", color="gray", linewidth=1)

for model in Models:
    graph_data_log["Close_pred_" + model][graph_data_log.index > start].plot(ax=ax, label="Predic
tion_" + model)

# x-axis
plt.gcf().autofmt_xdate()
ax.set(xlabel="Time", ylabel="Logarithmic_Price")
plt.legend(loc="upper left");
```



In [34]:

```
graph_data_original = graph_data_original.reset_index()

Models = ["Lasso"]

for model in Models:
    # inverse for Prediction
    y_pred_original = inverse_trans_yeo_johnson(all_data_lagged["Close"].values.reshape(-1,1), gra
ph_data_log["Close_pred_" + model].values.reshape(-1, 1))
    y_pred_original =  pd.DataFrame(y_pred_original)
    # Add the data on "graph_data"
    graph_data_original["Close_pred_" + model] = y_pred_original

    # Validation setに対してYen/kWhでの評価
    rmse = sqrt(mean_squared_error(graph_data_original.Close[Min_valid_index:], graph_data_origin
al["Close_pred_" + model][Min_valid_index:]))
    mae = mean_absolute_error(graph_data_original.Close[Min_valid_index:], graph_data_original["C
lose_pred_" + model][Min_valid_index:])
    # Store the result of evaluation into the "Eval_table"
    Eval_table_original[model] = pd.Series([np.mean(rmse), np.mean(mae)])

graph_data_original = graph_data_original.set_index("DateTime")

# Plot the original close price and  predicted price
fig, ax = plt.subplots(1, figsize=(15, 6))
# Vertical line (need to convert the date type from timestamp to datetime.datetime as x-axis)
plt.axvline(prediction_point.to_pydatetime(), label="Validation Point", linestyle="dashed", color="bl
ack")
# Plot Close
graph_data_original.Close[graph_data_original.index > start].plot(ax=ax, label="Observation", linest
yle="dashed", color="gray", linewidth=1)

# Plot the predicted price with each model
for model in Models:
    graph_data_original["Close_pred_" + model][graph_data_original.index > start].plot(ax=ax, label
="Prediction_" + model)

# x-axis
plt.title('Prediction performance_original price')
plt.gcf().autofmt_xdate()
ax.set(xlabel="Time", ylabel="Close_Price(Yen/kWh)")
plt.legend(loc="upper left");
```



In [35]:

```
# Evalation for the prediction of validation data based on log
Eval_table_log
```

Out[35]:

	EvalFunc	Lasso
0	RMSE_log	0.741566
1	MAE_log	0.524682

0.741566

In [36]:

```
# Evalation for the prediction of validation data based on original
Eval_table_original = Eval_table_original[["EvalFunc", "Lasso"]]
Eval_table_original
```

Out[36]:

	EvalFunc	Lasso
0	RMSE_Yen/kWh	3.942812
1	MAE_Yen/kWh	1.984035

3.942812

- Benchmark

In [37]:

```
# Diff between 1day-lagged spot price and observed close price
spot_1daylag = all_data["System_price(Yen/kWh)"].shift(1)[Min_valid_index:]
close = all_data["Close"][Min_valid_index:]

print("RMSE_Close/Spot: {}".format(round(sqrt(mean_squared_error(close, spot_1daylag)), 4)))
print("MAE_Close/Spot: {}".format(round(mean_absolute_error(close, spot_1daylag), 4)))
```

RMSE_Close/Spot: 3.1436
MAE_Close/Spot: 1.6334

In [38]:

```
# Diff between 1day-lagged close price and observed close price
close_1daylag = all_data["Close"].shift(1)[Min_valid_index:]
close = all_data["Close"][Min_valid_index:]

print("RMSE_Close/Closelag: {}".format(round(sqrt(mean_squared_error(close, close_1daylag)), 4
)))
print("MAE_Close/Closelag: {}".format(round(mean_absolute_error(close, close_1daylag), 4)))
```

RMSE_Close/Closelag: 3.216
MAE_Close/Closelag: 1.4776

Benchmark: Diff between 1day-lagged spot price and observed close price

RMSE_Spot/Close: 3.1436

MAE_Spot/Close: 1.6334

Making combined price data with all the tradable price

In [39]:

```
# Finalise dataset and make the csv data for trading phase

model = "Lasso"

# Pick up the requied data for the next stage
df_prediction_trader = all_data[['Date', 'HH', 'Open', 'High', 'Low', 'Close', 'Price_Tokyo(Yen/kWh)'
]]

# Add 1 day lagged price of DA area price in Tokyo
df_prediction_trader["Price_Tokyo(Yen/kWh)_1daylag"] = all_data["Price_Tokyo(Yen/kWh)"].shift
(48).fillna(0)
df_prediction_trader["DateTime"] = graph_data_original.reset_index()["DateTime"]
df_prediction_trader = df_prediction_trader.rename(columns={'Price_Tokyo(Yen/kWh)': 'Spot',
                                        'Price_Tokyo(Yen/kWh)_1daylag': 'Spot_1daylag'})
graph_data_original = graph_data_original.reset_index()
df_prediction_trader['Close_pred'] = graph_data_original["Close_pred_" + model].round(2)
df_prediction_trader = df_prediction_trader[df_prediction_trader.index >= Min_valid_index]

# CSV file
df_prediction_trader.to_csv('df_prediction_trader.csv', index=False)

# Check the data
df_prediction_trader
```

Out[39]:

	Date	HH	Open	High	Low	Close	Spot	Spot_1daylag	DateTime	Close_pred
23808	2017-08-10	1	10.70	12.98	5.94	9.40	8.19	8.10	2017-08-10 00:00:00	7.83
23809	2017-08-10	2	6.89	12.67	5.63	9.40	8.04	8.00	2017-08-10 00:30:00	7.58
23810	2017-08-10	3	6.42	11.86	5.58	9.40	7.93	8.05	2017-08-10 01:00:00	7.55
23811	2017-08-10	4	10.30	11.58	4.91	9.40	7.73	7.80	2017-08-10 01:30:00	7.27
23812	2017-08-10	5	10.50	11.62	4.88	9.40	7.93	7.94	2017-08-10 02:00:00	7.35
...	...	...	...	...	...	...	...	...	...	...
83323	2020-12-31	44	35.00	70.00	33.00	70.00	35.00	50.00	2020-12-31 21:30:00	18.81
83324	2020-12-31	45	42.00	70.00	41.01	45.48	40.00	40.00	2020-12-31 22:00:00	19.55
83325	2020-12-31	46	42.00	70.00	35.00	41.33	40.00	40.00	2020-12-31 22:30:00	17.98
83326	2020-12-31	47	37.00	70.00	33.93	36.66	35.00	33.21	2020-12-31 23:00:00	14.78
83327	2020-12-31	48	27.00	37.50	23.93	26.46	25.00	25.00	2020-12-31 23:30:00	12.57

59520 rows × 10 columns

In [ ]: