Table of Contents

- 1 Purpose of the analysis on this notebook
- 2 Read & interpret dataset
  - 2.1 JEPX Intra datasets (Target: "Close price")
    - 2.1.1 Basic characteristics
    - 2.1.2 Stationarity
    - 2.1.3 Seasonality
    - 2.1.4 Autocorrelation
    - 2.1.5 Trading vollume from 2014 to 2019
  - 2.2 JEPX Spot datasets
    - 2.2.1 The relation between spot and close price
    - 2.2.2 Bidding balance
    - 2.2.3 The correlation between Area price in different area and "Close" price.
  - 2.3 Hourly TotalDemand with Subtotal of EachArea
  - 2.4 Hourly TodalDemand with Generation from DifferentPower
  - 2.5 Depand peak for the next day
  - 2.6 Fit actual predicted (Not completed yet) ※ Add only if the electricity companies updated the data
  - 2.7 Other data
    - 2.7.1 Actual generation(Tohoku area) ※Skip
    - 2.7.2 Weather data (Tohoku area) ※Skip
    - 2.7.3 LNG price
- 3 Make all data
  - 3.1 Merge all input data
  - 3.2 Adjust all data (Add, remove outlier and drop features that have high correlation)
  - 3.3 Save the all data as a csv file

Purpose of the analysis on this notebook

- Interpreting the dataset related to electricity trading in Japan.
- Seeking useful features for the prediction.
- Create a dataframe for all_data which is used for prediction.

Read & interpret dataset

- Loading some datasets
- Confirming and interpreting it
- Preprocessing missing values and outliers on the datasets

In [1766]:
```
# 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

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

JEPX_Intra_datasets (Target: "Close price")

http://www.jepx.org/market/index.html (http://www.jepx.org/market/index.html)

(Reference)

Time-series data analysis

- https://github.com/mloning/intro-to-ml-with-time-series-DSSGx-2020/blob/master/notebooks/02_exploratory_data_analysis.ipynb (https://github.com/mloning/intro-to-ml-with-time-series-DSSGx-2020/blob/master/notebooks/02_exploratory_data_analysis.ipynb)
- https://github.com/juanitorduz/btsa/blob/master/python/fundamentals/notebooks/eda_part_2_correlations.ip (https://github.com/juanitorduz/btsa/blob/master/python/fundamentals/notebooks/eda_part_2_correlations.ip

In [1768]:
```
# read the dataset (Here, these are re-organised after getting the latest dataset)

#Intra_dataset
df_intra_2016 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/im_trade_summary_2016.csv', sep=',', header=0, encoding='shift_jis')
df_intra_2017 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/im_trade_summary_2017.csv', sep=',', header=0, encoding='shift_jis')
df_intra_2018 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/im_trade_summary_2018.csv', sep=',', header=0, encoding='shift_jis')
df_intra_2019 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/im_trade_summary_2019.csv', sep=',', header=0, encoding='shift_jis')
df_intra_2020 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/im_trade_summary_2020.csv', sep=',', header=0, encoding='shift_jis')

# marge all the intra datasets
df_intra = pd.concat([df_intra_2016, df_intra_2017])
df_intra = pd.concat([df_intra, df_intra_2018])
df_intra = pd.concat([df_intra, df_intra_2019])
df_intra = pd.concat([df_intra, df_intra_2020])
```

In [1769]:
```
# Rename the columns in English
df_intra = df_intra.rename(columns={'年月日': 'Date',
                                    '時刻コード': 'HH',
                                    '始値(円/kWh)': 'Open',
                                    '高値(円/kWh)': 'High',
                                    '安値(円/kWh)': 'Low',
                                    '終値(円/kWh)': 'Close',
                                    '平均(円/kWh)': 'Average',
                                    '約定量合計(MWh/h)': 'Volume(MWh/h)',
                                    '約定件数': 'Volume(Tick count)'})
```

In [1770]:
```
# Apply to_datetime
df_intra["Date"] = pd.to_datetime(df_intra["Date"])
```

Basic characteristics

(Reference) Time series data analysis https://github.com/mloning/intro-to-ml-with-time-series-DSSGx-2020/blob/master/notebooks/02_exploratory_data_analysis.ipynb (https://github.com/mloning/intro-to-ml-with-time-series-DSSGx-2020/blob/master/notebooks/02_exploratory_data_analysis.ipynb)

Try to look at the over all transition of close price from 2016 to 2020

In [1498]:
```
# Select the data until 2020-12-31
close_price = df_intra[df_intra['Date'] <= '2020-12-31']
close_price = close_price[['Date', 'HH', 'Close']]

# Fill missing values with interpolate method
close_price.interpolate(method='linear', inplace=True)
```

In [1771]:
```
# See the descriptive statictics of Target "Close price"
close_price["Close"].describe()
```

Out[1771]:
```
count   83328.000000
mean        8.635154
std         4.666505
min         0.010000
25%         6.130000
50%         7.930000
75%        10.000000
max       150.000000
Name: Close, dtype: float64
```

Make Half Hourly time table which is useful for the following analysis

In [1772]:
```
# Pick up "HH" columns from df_intra
df_intra_HH = df_intra.reset_index().copy()
HH_table = pd.DataFrame(df_intra_HH["HH"])
HH_table = HH_table.drop_duplicates()

# Make "Time" column
HH_table['Time'] = pd.date_range('2020/01/01', periods=48,  freq='30min').strftime(
'%H:%M:%S')
HH_table['Time'] = HH_table['Time'].str[-8:]

HH_table.head()
```

Out[1772]:

	HH	Time
0	1	00:00:00
1	2	00:30:00
2	3	01:00:00
3	4	01:30:00
4	5	02:00:00

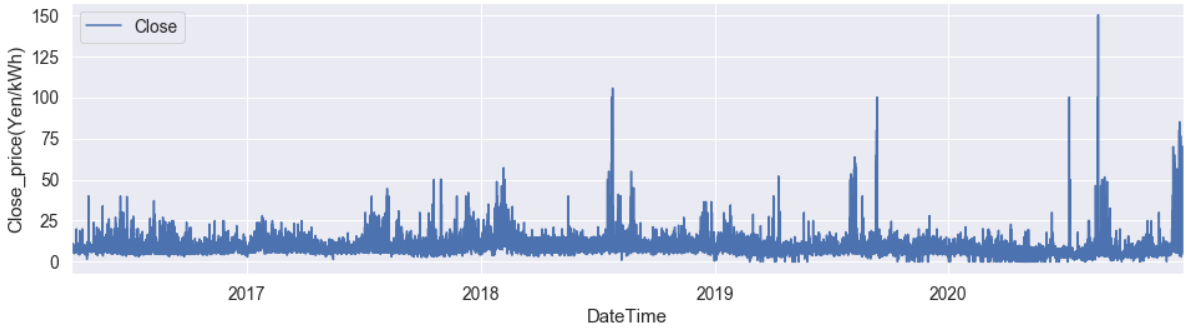
In [1838]:
```
# Combine "Date" and  "Time" to make "DateTime" column
close_price_graph = close_price.copy()
close_price_graph = pd.merge(close_price_graph, HH_table, how="left", on=['HH'])
close_price_graph["Date"] = close_price_graph["Date"].astype(str)
close_price_graph["DateTime"] = pd.to_datetime(close_price_graph["Date"] + " " + cl
ose_price_graph["Time"], format='%Y-%m-%d %H:%M')
close_price_graph = close_price_graph.drop(["Date", "HH", "Time"], axis=1)

# Set it as index
close_price_graph.set_index("DateTime", inplace=True)
```

Try to plot and interpret for the following 2 type of timeframes of the data.

- 30min transition for all items
- Daily transition for each items

[30min transition for all items]

In [1774]:
```
# Simple plot
fig, ax = plt.subplots(1, figsize=plt.figaspect(.25))

start = "2016-04-01"
end = "2020-12-31"

close_price_graph["Close"][start : end].plot(ax=ax)
ax.set(ylabel="Close_price(Yen/kWh)", xlabel="DateTime")
plt.legend(loc="upper left");
```



We can see;

- No trend
- Some seasonal patterns
- Price spike

[Daily transition for each item]

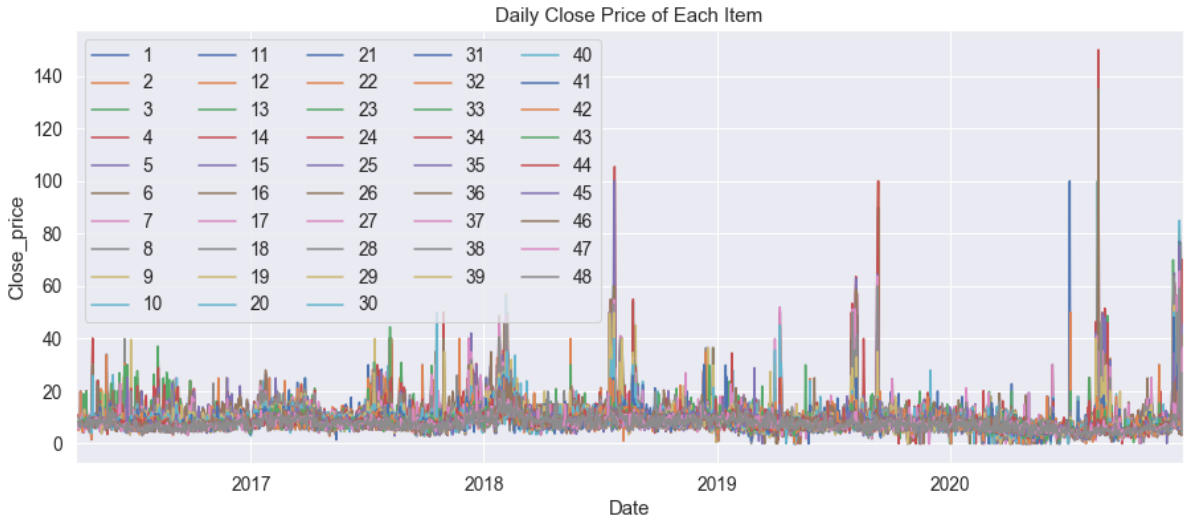
In [1775]:
```
# Pivot to make daily price for each item
close_HH_table = pd.DataFrame(close_price.pivot(index='Date', columns='HH', values=
'Close'))
col = close_HH_table.loc[: , 0:48]

# Add mean on the table
close_HH_table["mean"] = col.mean(axis=1)
```

In [1776]:
```
close_HH_table.head()
```

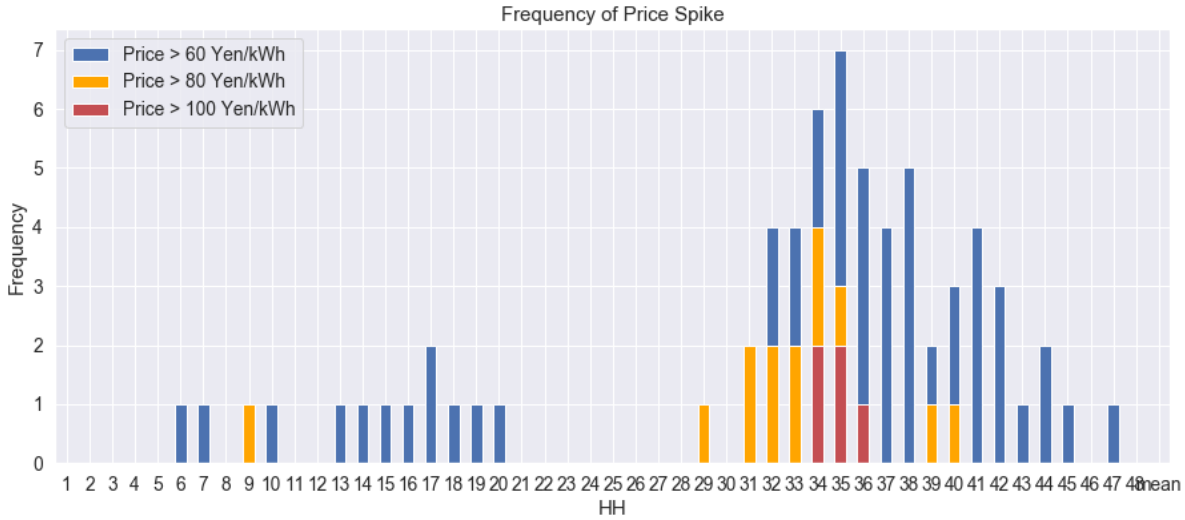
Out[1776]:

HH	1	2	3	4	5	6	7	8	9	10
Date										
2016-04-01	7.690000	7.450000	7.210000	7.060000	7.210000	7.210000	7.210000	7.210000	7.06	7.
2016-04-02	7.500000	6.780000	6.938571	7.097143	7.255714	7.414286	7.572857	7.731429	7.89	8.
2016-04-03	6.506667	6.613333	6.720000	6.826667	6.933333	7.040000	7.146667	7.253333	7.36	7.
2016-04-04	6.496250	6.487500	6.478750	6.470000	6.465000	6.460000	6.510000	6.520000	6.54	6.
2016-04-05	7.000000	7.000000	7.000000	7.000000	7.000000	6.430000	6.480000	6.520000	6.51	6.

In [1777]:
```
fig, ax = plt.subplots(1, figsize=(15, 6))
plt.title('Daily Close Price of Each Item')

start = "2016-04-01"
end = "2020-12-31"

# Pick up the items of each timeslot from 1st HH to 48th HH
for i in list(range(1,49)):
    # plot all items for a specific periods
    close_HH_table[i][start : end].plot(ax=ax)

ax.set(ylabel="Close_price", xlabel="Date")
plt.legend(loc="upper left", ncol=5);
```



In [1798]:
```
# Plot the number of price spice for each item
fig, ax = plt.subplots(1, figsize=(15, 6))
plt.title('Frequency of Price Spike')

# Count the price spike
PriceSpike_60 = (close_HH_table > 60).sum()
PriceSpike_80 = (close_HH_table > 80).sum()
PriceSpike_100 = (close_HH_table > 100).sum()


# Plot
PriceSpike_60.plot.bar(label="Price > 60 Yen/kWh")
PriceSpike_80.plot.bar(label="Price > 80 Yen/kWh", color="orange")
PriceSpike_100.plot.bar(label="Price > 100 Yen/kWh", color="r")

plt.xticks(rotation= 0)
plt.ylabel("Frequency", rotation=90)
ax.legend(loc="upper left");
```



In [1800]:
```
print(HH_table[HH_table["HH"]==34])
print(HH_table[HH_table["HH"]==36])
```

```
   HH     Time
33  34  16:30:00
   HH     Time
35  36  17:30:00
```

Each item has different move. Contretely, The items for the period from 16:30 to 17:30 tend to easy to spike the price.

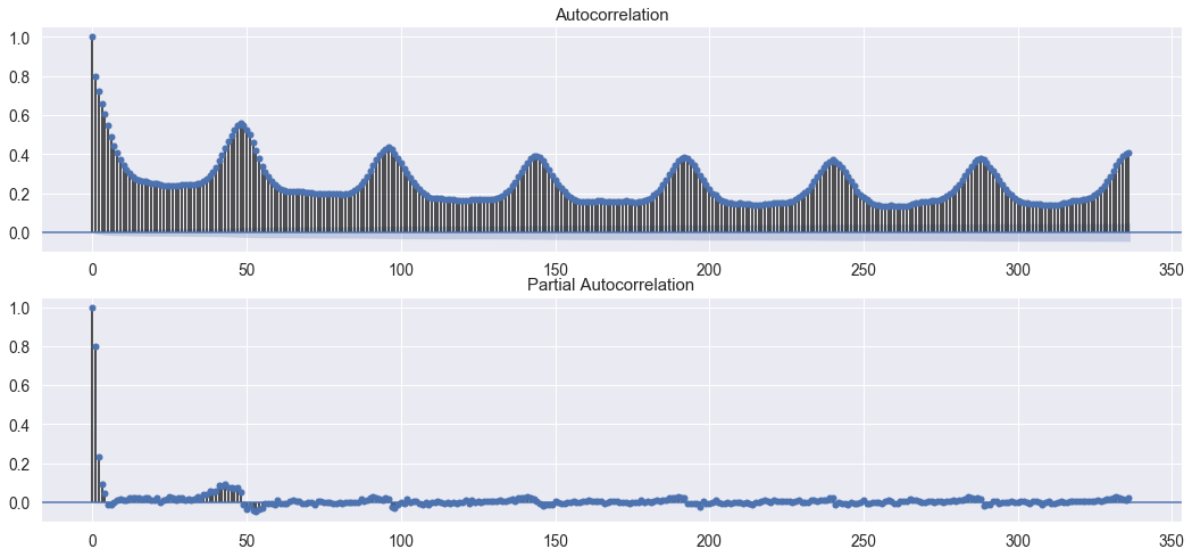
Basically, the 30min transition for all items is used for prediction as follows.

Stationarity

Time-series data analysis/prediction must be confirmed whether it is stationarity or not, because constructed models may not be able to apply for out-of-sample dataset if the target is not stationary. We can use the machine learning approach on the condition of stationarity.

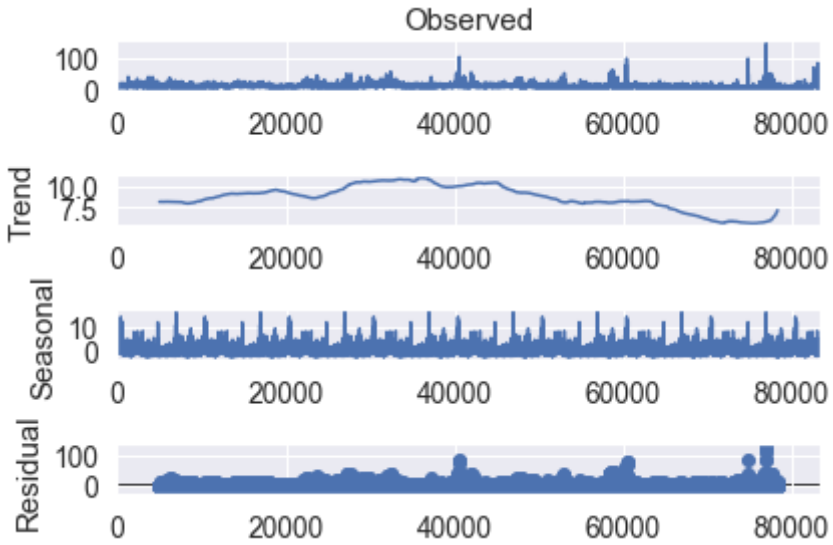
(Preference) in Japanese: [https://deepblue-ts.co.jp/%E7%B5%B1%E8%A8%88%E5%AD%A6/introduction-to-time-series-analysis/ (https://deepblue-ts.co.jp/%E7%B5%B1%E8%A8%88%E5%AD%A6/introduction-to-time-series-analysis/)](https://deepblue-ts.co.jp/%E7%B5%B1%E8%A8%88%E5%AD%A6/introduction-to-time-series-analysis/)

In [1280]:
```
fig = plt.figure(figsize=(18,8))
# Autocorrelation (1lag=30min --> 336 lags=1week)
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(close_price_graph, lags=336, ax=ax1)

# Partial Autocorrelation
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(close_price_graph, lags=336, ax=ax2)
```



- Trend: It looks no trend
- Seasonality: 48 cycle, which mean strong correlation with 1day lag price

In [1782]:
```
# See the detail the data dividing into trends and seasonalities
seasonal_decompose_res = sm.tsa.seasonal_decompose(close_price_graph.values, fre
q=10000).plot()
```



※Trend and Seasonality can be seen only if the premeter of "freq" is sufficiently large. (1year?)

- Augmented Dickey-Fuller unit root test (ADF test)
  The Augmented Dickey-Fuller test can be used to test for a unit root in a univariate process in the presence of serial correlation.

Test stationality to the close price data using ADF test
(https://www.statsmodels.org/stable/generated/statsmodels.tsa.stattools.adfuller.html (https://www.statsmodels.org/stable/generated/statsmodels.tsa.stattools.adfuller.html))

It sets the null hypothesis that there is a unit root, and provide the results of p-value. --> If null hypothesis can be rejected (p-value < 5%)

In [1282]:
```
# logarithmic transformation using Yeo-Johnson
from sklearn.preprocessing import PowerTransformer
import itertools

pt = PowerTransformer() #default: Yeo-Johnson (this allow also 0-value to transform to
 logarithmic value)
pt.fit(close_price_graph)
logClose = close_price_graph.copy()
index = logClose.index
logClose = pt.transform(logClose)
# ndarray make flat as a list
logClose = list(itertools.chain.from_iterable(logClose))
logClose = pd.DataFrame(logClose)
logClose.index = index
logClose.columns = ["Close_log"]
# logClose = diff_logClose.dropna()
```

In [1783]:
```
# Apply ADF test for original close price and logarithmic price
pvalue_results_30min = pd.DataFrame()
data_list = [close_price_graph, logClose]
data_num = [1, 2]
for data, i in zip(data_list, data_num):
    results = []
    # “ctt” : constant, and linear and quadratic trend.
    ctt = sm.tsa.stattools.adfuller(data, regression="ctt")
    results.append(round(ctt[1], 4))
    # "ct” : constant and trend.
    ct = sm.tsa.stattools.adfuller(data, regression="ct")
    results.append(round(ct[1], 4))
    # “c” : constant only (default)
    c = sm.tsa.stattools.adfuller(data, regression="c")
    results.append(round(c[1], 4))
    # “nc” : no constant, no trend.
    nc = sm.tsa.stattools.adfuller(data, regression="nc")
    results.append(round(nc[1], 4))

    pvalue_results_30min[i] = pd.Series(results)

pvalue_results_30min.columns = ["p-value (Original)", "p-value(logClose)"]
pvalue_results_30min.index = ["constant/linear and quadratic trend", "constant/trend"
, "constant", "no constant/trend"]
pvalue_results_30min
```

Out[1783]:

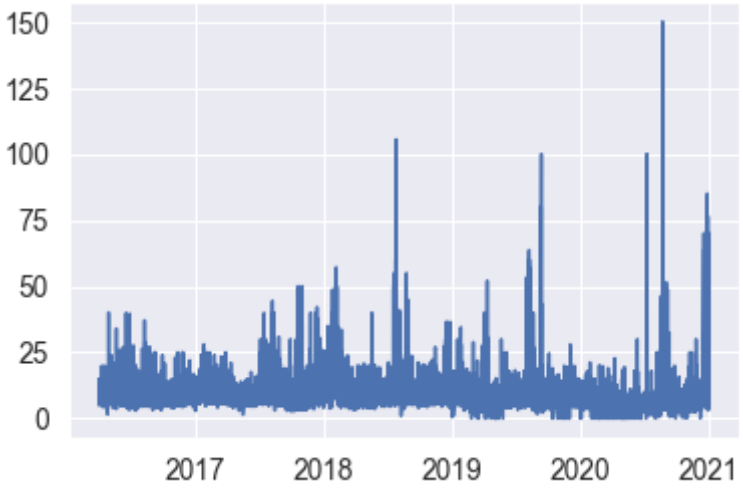
	p-value (Original)	p-value(logClose)
constant/linear and quadratic trend	0.0000	0.0
constant/trend	0.0000	0.0
constant	0.0000	0.0
no constant/trend	0.0003	0.0

Surprisingly, the null hypothesis can be rejected both for Original/logClose.(Succeeded: p-value < significance level 5% or 0%)

Let's try to compare histgram of original close price with that of logarithmic price

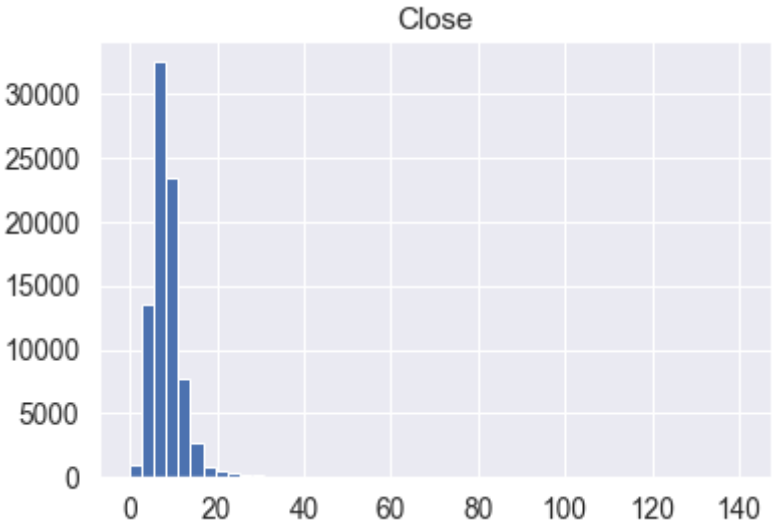
In [1286]:
```
# Price transition of Original close price
plt.plot(close_price_graph)
```

Out[1286]: [<matplotlib.lines.Line2D at 0x13512a050>]

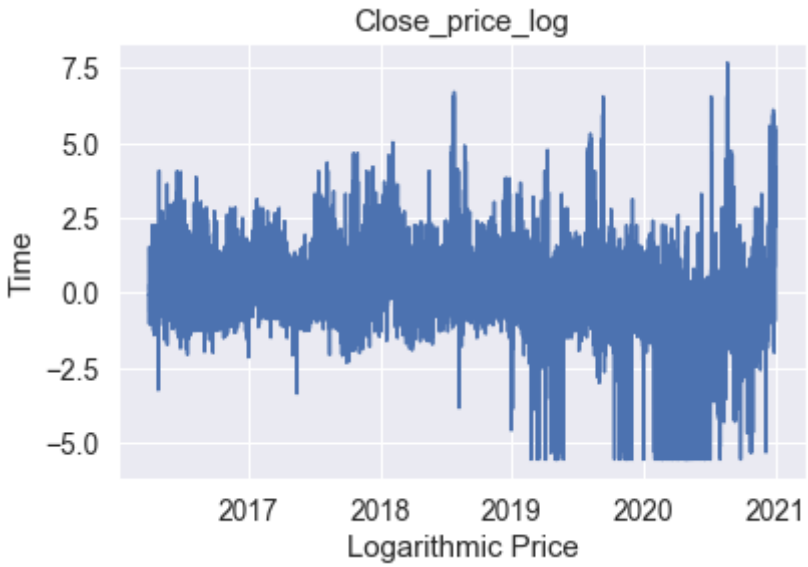


In [1784]:
```
# Histgram of Original close price
close_price_graph.hist(bins=50, range=(0, 140), rwidth=1.0);

print("Mean: %f" % close_price_graph.mean())
print("Std: %f" % close_price_graph.std())
```

Mean: 8.635154
Std: 4.666505

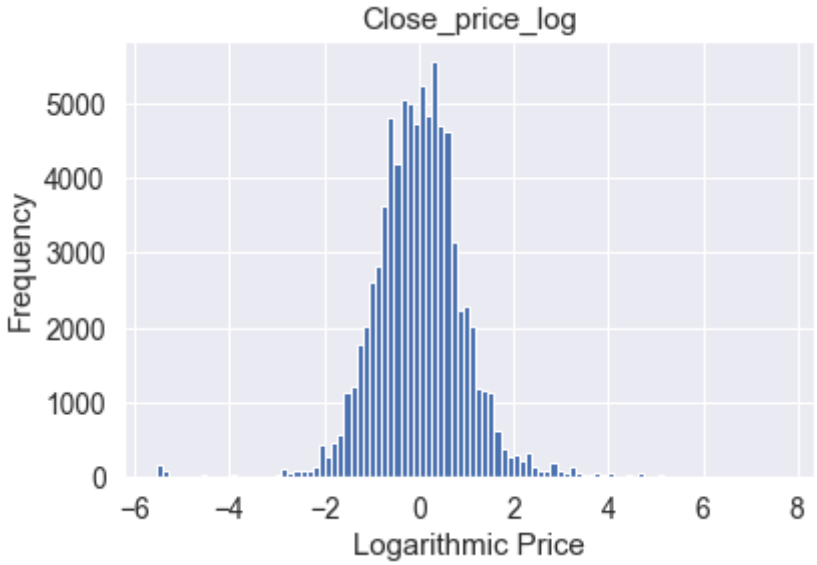


In [1673]:
```
# Price transition of log-price
fig, ax = plt.subplots(1, figsize=plt.figaspect(.65))

plt.plot(logClose)
ax.set(title="Close_price_log", ylabel="Time", xlabel="Logarithmic Price");
```



In [1785]:
```
# Histgram of log-price
fig, ax = plt.subplots(1, figsize=plt.figaspect(.65))

logClose.hist(bins=100, rwidth=1.0, ax=ax)
ax.set(title="Close_price_log", ylabel="Frequency", xlabel="Logarithmic Price");

print("Mean: %f" % logClose.mean())
print("Std: %f" % logClose.std())
```

```
Mean: -0.000000
Std: 1.000006
```

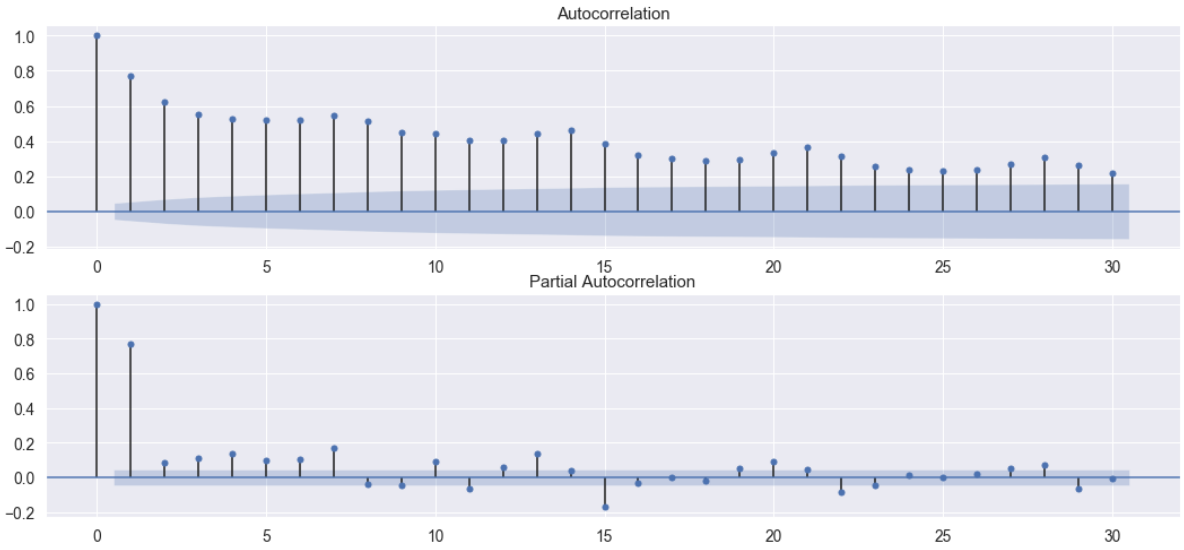


The distribusion of close price is positive skew while that of the log-price is no skew. In addition, log-price has z-distribution.

-->Basically, log price shuold be used for prediction.

~~※ Daily transition for each item (Try to see just in case)~~

In [1290]:
```
# fig = plt.figure(figsize=(18,8))
# # Autocorrelation (1lag=1day)
# ax1 = fig.add_subplot(211)
# fig = sm.graphics.tsa.plot_acf(close_HH_table["mean"], lags=30, ax=ax1)

# # Partial Autocorrelation
# ax2 = fig.add_subplot(212)
# fig = sm.graphics.tsa.plot_pacf(close_HH_table["mean"], lags=30, ax=ax2)
```

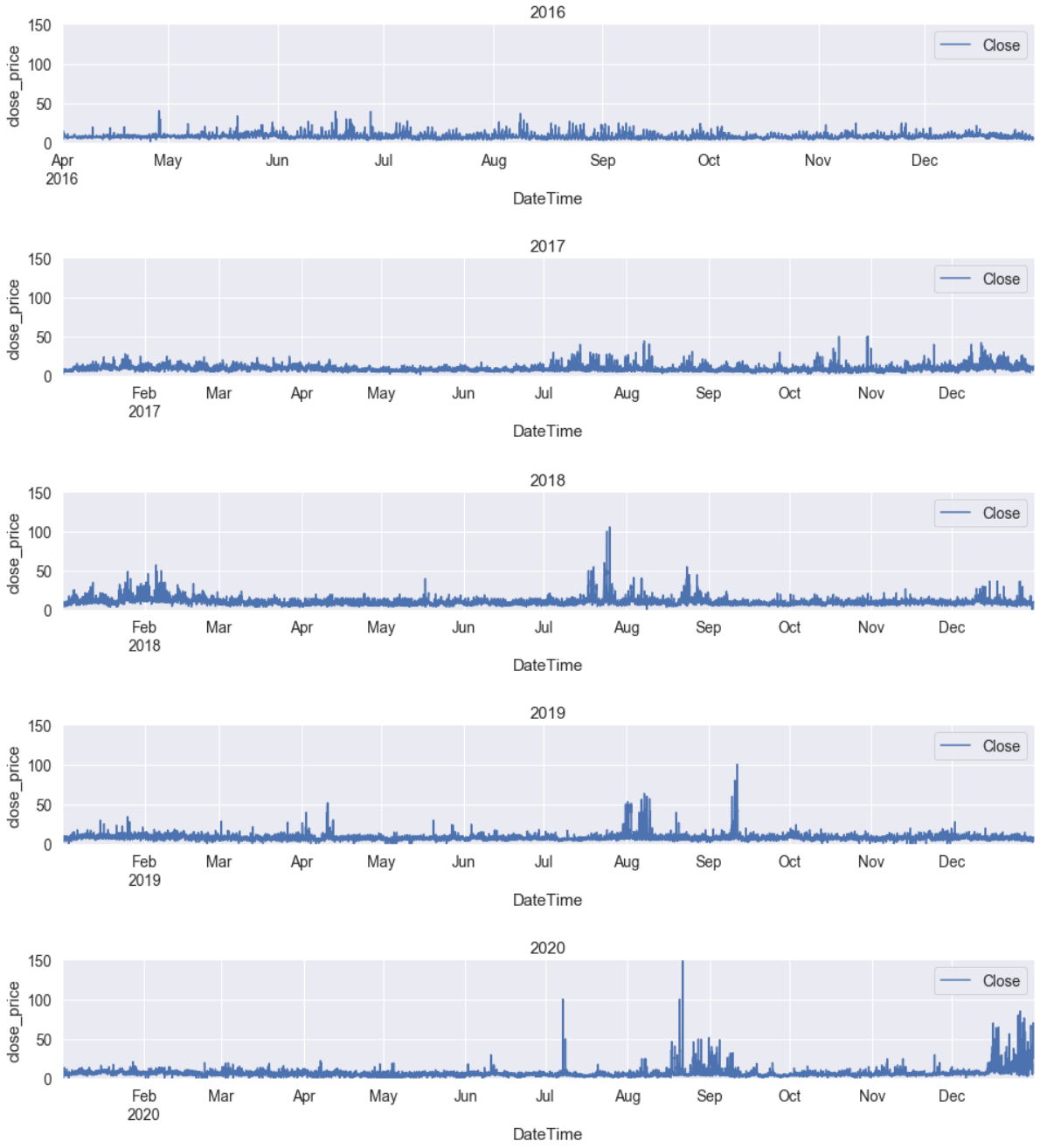


~~Correlation can be seen every 7days~~

Seasonality

From here, 30min slot data is used to see the seasonality.

In [1294]:
```
# Plot close price for each year

years = [2016, 2017, 2018, 2019, 2020]

for year in years:
    start = str(year) + "-01-01 00:00:00"
    end = str(year) + "-12-31 23:30:00"
    close_price_graph_y = close_price_graph[np.logical_and(start < close_price_graph.index, close_price_graph.index <= end)]

    fig, ax = plt.subplots(1, figsize=plt.figaspect(.05))
    close_price_graph_y.plot(ax=ax)
    ax.set_ylim(0, 150)
    ax.set(title=year, ylabel="close_price", xlabel="DateTime");
```



Can obsearve price spices which tend to be happend from Augast to September, especially since 2018

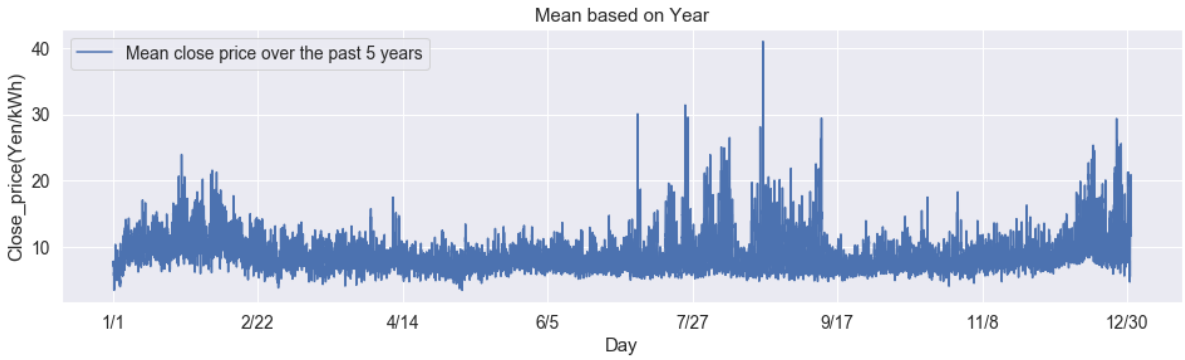
Let's try to see one more detailed trends to confirm yearly, weekly and daily seasonality.

In [1295]:
```
# Make the table for pivot
close_price_graph_table = close_price_graph.copy()
close_price_graph_table["Year"] = close_price_graph_table.index.year
close_price_graph_table["Month"] = close_price_graph_table.index.month
close_price_graph_table["Week"] = close_price_graph_table.index.isocalendar().week
close_price_graph_table["DayofWeek"] = close_price_graph_table.index.dayofweek
close_price_graph_table["Day"] = close_price_graph_table.index.day
close_price_graph_table["Time"] = close_price_graph_table.index.time
close_price_graph_table = close_price_graph_table.reset_index()
close_price_graph_table = close_price_graph_table.drop("DateTime", axis=1)
close_price_graph_table.head()
```

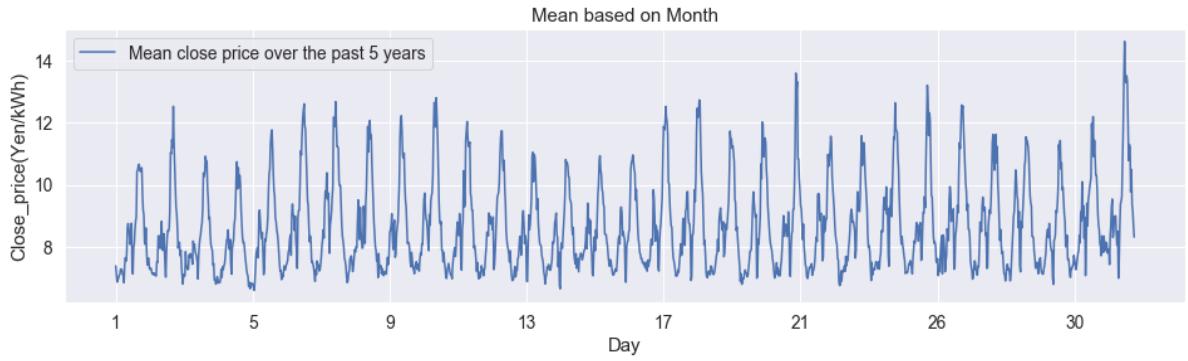
Out[1295]:

	Close	Year	Month	Week	DayofWeek	Day	Time
0	7.69	2016	4	13	4	1	00:00:00
1	7.45	2016	4	13	4	1	00:30:00
2	7.21	2016	4	13	4	1	01:00:00
3	7.06	2016	4	13	4	1	01:30:00
4	7.21	2016	4	13	4	1	02:00:00

In [1649]:
```
# Plot Average close price for each time slot based on Year
close_price_graph_table_yearly = pd.DataFrame(close_price_graph_table.pivot(index=
['Month', 'Day', 'Time'], columns='Year', values='Close'))
year_col = close_price_graph_table_yearly.loc[: , "2016":"2020"]
close_price_graph_table_yearly["Mean"] = year_col.mean(axis=1)
close_price_graph_table_yearly["Date"] = close_price_graph_table_yearly.index.get_l
evel_values('Month').astype(str) + "/" + close_price_graph_table_yearly.index.get_lev
el_values('Day').astype(str)
close_price_graph_table_yearly = close_price_graph_table_yearly.drop(years, axis=1)
close_price_graph_table_yearly = close_price_graph_table_yearly.reset_index()
close_price_graph_table_yearly = close_price_graph_table_yearly.drop(["Month", "Da
y", "Time"], axis=1)
close_price_graph_table_yearly = close_price_graph_table_yearly.set_index("Date")


fig, ax = plt.subplots(1, figsize=plt.figaspect(.25))
close_price_graph_table_yearly.plot(ax=ax)
ax.legend(["Mean close price over the past 5 years"], loc="upper left")
ax.set(title="Mean based on Year", ylabel="Close_price(Yen/kWh)", xlabel="Day");
```



In [1650]:
```
# Plot Average close price for each time slot based on Month
close_price_graph_table_monthly = pd.DataFrame(close_price_graph_table.pivot(inde
x=['Day', 'Time'], columns=['Year', 'Month'], values='Close'))
month_col = close_price_graph_table_monthly.loc[: , :]
close_price_graph_table_monthly["Mean"] = month_col.mean(axis=1)
month_col = month_col.drop("Mean", axis=1)
close_price_graph_table_monthly = close_price_graph_table_monthly.drop(month_co
l, axis=1)
close_price_graph_table_monthly["Date"] = close_price_graph_table_monthly.index.g
et_level_values('Day').astype(str)
close_price_graph_table_monthly = close_price_graph_table_monthly.reset_index()
close_price_graph_table_monthly = close_price_graph_table_monthly.drop(["Day", "T
ime"], axis=1)
close_price_graph_table_monthly = close_price_graph_table_monthly.set_index("Dat
e")
# close_price_graph_table_monthly = close_price_graph_table_monthly.drop(["Mont
h"], axis=1)


fig, ax = plt.subplots(1, figsize=plt.figaspect(.25))
close_price_graph_table_monthly.plot(ax=ax)
ax.legend(["Mean close price over the past 5 years"], loc="upper left")
ax.set(title="Mean based on Month", ylabel="Close_price(Yen/kWh)", xlabel="Day");
```

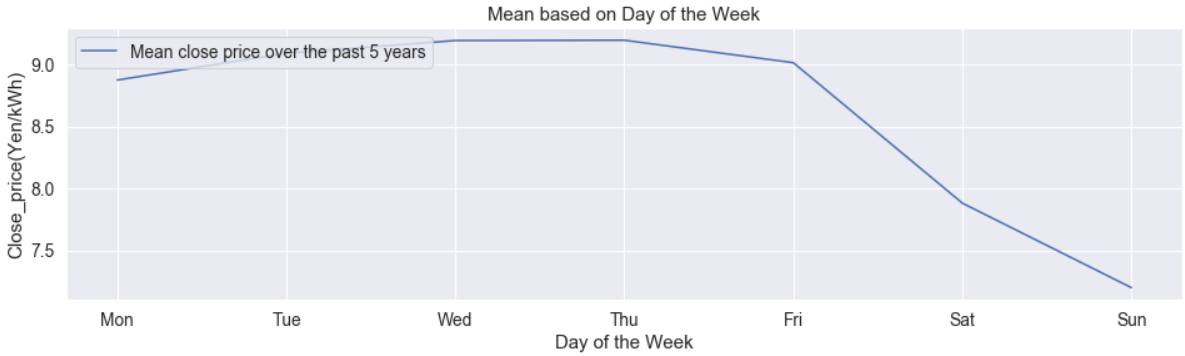


In [1298]:
```
close_price_graph_table_monthly.head()
```

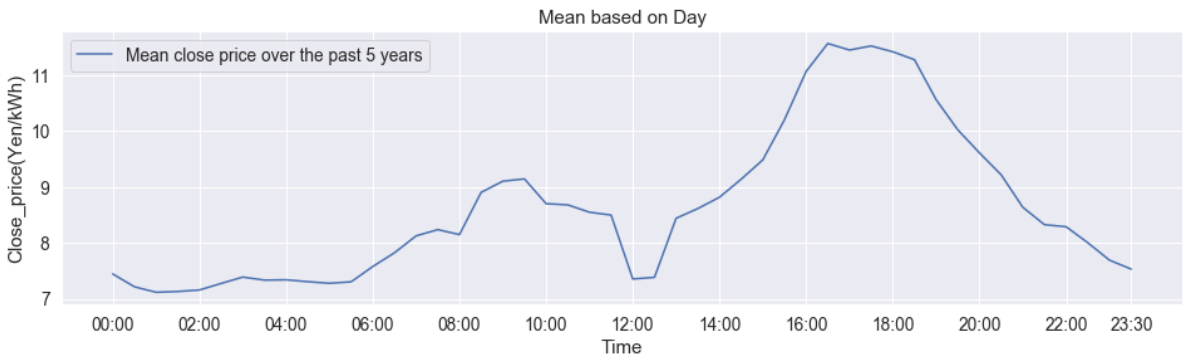
Out[1298]:

	Year	2016									2017				
	Month	4	5	6	7	8	9	10	11	12	1	2	3	4	5
Day	Time														
1	00:00:00	7.69	9.0	8.00	5.50	5.99	5.97	6.00	7.27	5.66	8.36	9.49	9.00	11.79	6.13
	00:30:00	7.45	9.0	7.00	5.50	5.99	5.97	8.39	7.27	7.95	8.08	9.49	9.00	10.67	6.07
	01:00:00	7.21	9.0	8.00	5.50	4.67	4.60	5.14	6.06	7.74	6.71	9.49	9.00	10.67	6.07
	01:30:00	7.06	9.0	6.82	5.84	4.67	4.60	5.14	6.06	7.80	7.22	9.49	9.06	10.71	8.12
	02:00:00	7.21	9.0	7.00	5.84	4.67	5.97	5.29	7.27	7.74	6.70	9.49	9.00	9.93	6.53

In [1651]:
```
# Plot Average close price for each time slot based on Weekday
close_price_graph_table_Weekly = pd.DataFrame(close_price_graph_table.pivot(index
=['DayofWeek', 'Time'], columns=['Year','Month','Week'], values='Close'))
close_price_graph_table_Weekly = close_price_graph_table_Weekly.groupby("DayofW
eek").mean()

weekday_col = close_price_graph_table_Weekly.loc[: , :]
close_price_graph_table_Weekly["Mean"] = weekday_col.mean(axis=1)
weekday_col = weekday_col.drop("Mean", axis=1)
close_price_graph_table_Weekly = close_price_graph_table_Weekly.drop(weekday_co
l, axis=1)
close_price_graph_table_Weekly["index"] = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
"Sun"]
close_price_graph_table_Weekly = close_price_graph_table_Weekly.set_index("index"
)

fig, ax = plt.subplots(1, figsize=plt.figaspect(.25))
close_price_graph_table_Weekly.plot(ax=ax)
ax.legend(["Mean close price over the past 5 years"], loc="upper left")
ax.set(title="Mean based on Day of the Week", ylabel="Close_price(Yen/kWh)", xlabel=
"Day of the Week");
```



In [1653]:
```
# Plot Average close price for each time slot based on Days(Looking at Daily seasonality)
import matplotlib.ticker as plticker
import matplotlib as mpl
import matplotlib.dates as mdates

close_price_graph_table_daily = pd.DataFrame(close_price_graph_table.pivot(index=['Time'], columns=['Year', 'Month', 'Day'], values='Close'))
day_col = close_price_graph_table_daily.loc[: , :]
close_price_graph_table_daily["Mean"] = day_col.mean(axis=1)
day_col = day_col.drop("Mean", axis=1)
close_price_graph_table_daily = close_price_graph_table_daily.drop(month_col, axis=1)
close_price_graph_table_daily = close_price_graph_table_daily.reset_index()
close_price_graph_table_daily = close_price_graph_table_daily.set_index("Time")


fig, ax = plt.subplots(1, figsize=plt.figaspect(.25))
close_price_graph_table_daily.plot(ax=ax)
ax.legend(["Mean close price over the past 5 years"], loc="upper left")
plt.xticks(["00:00:00", "02:00:00", "04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00",
        "14:00:00", "16:00:00", "18:00:00", "20:00:00", "22:00:00", "23:30:00"])

ax.set(title="Mean based on Day", ylabel="Close_price(Yen/kWh)", xlabel="Time");
```



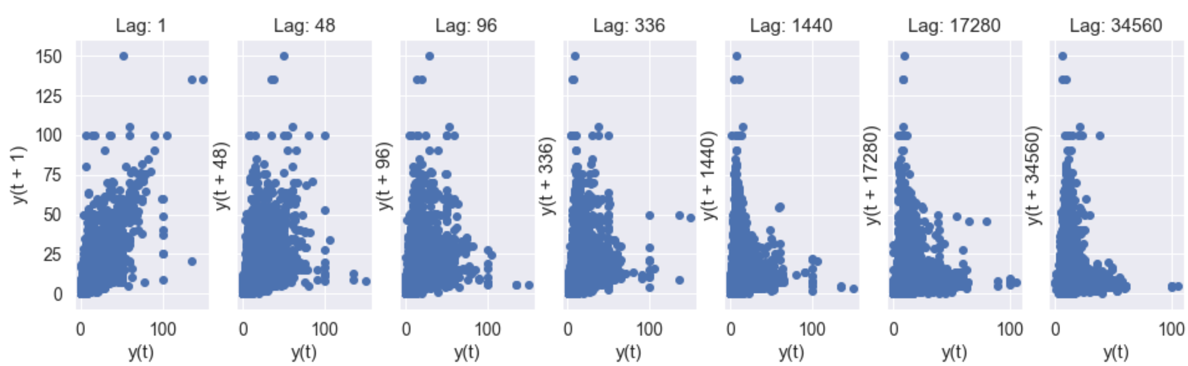
Regarding daily seasonality, the close price about from 16:00 to 18:30 tend to be high.

Autocorrelation

See correlatioin between original price and lagged price

In [1786]:
```
# plot the relation betweeen original/lagged price
from pandas.plotting import lag_plot
lags = [1, 48, 96, 336, 1440, 17280, 34560]
fig, axs = plt.subplots(ncols=7, figsize=plt.figaspect(.25), sharey=True)
for k, lag in enumerate(lags):
    ax = axs[k]
    lag_plot(close_price_graph.Close, lag=lag, ax=ax)
    ax.set_title(f"Lag: {lag}")
```

```
WARNING:matplotlib.axes._axes:'c' argument looks like a single numeric RGB or RGBA se
quence, which should be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to spe
cify the same RGB or RGBA value for all points.
WARNING:matplotlib.axes._axes:'c' argument looks like a single numeric RGB or RGBA se
quence, which should be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to spe
cify the same RGB or RGBA value for all points.
WARNING:matplotlib.axes._axes:'c' argument looks like a single numeric RGB or RGBA se
quence, which should be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to spe
cify the same RGB or RGBA value for all points.
WARNING:matplotlib.axes._axes:'c' argument looks like a single numeric RGB or RGBA se
quence, which should be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to spe
cify the same RGB or RGBA value for all points.
WARNING:matplotlib.axes._axes:'c' argument looks like a single numeric RGB or RGBA se
quence, which should be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to spe
cify the same RGB or RGBA value for all points.
WARNING:matplotlib.axes._axes:'c' argument looks like a single numeric RGB or RGBA se
quence, which should be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to spe
cify the same RGB or RGBA value for all points.
WARNING:matplotlib.axes._axes:'c' argument looks like a single numeric RGB or RGBA se
quence, which should be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to spe
cify the same RGB or RGBA value for all points.
```



1HH --> 30min

48HH --> 1day

96HH --> 2day

336HH --> 1week

1440HH --> 1month

17520HH --> 1year

25040HH --> 2years

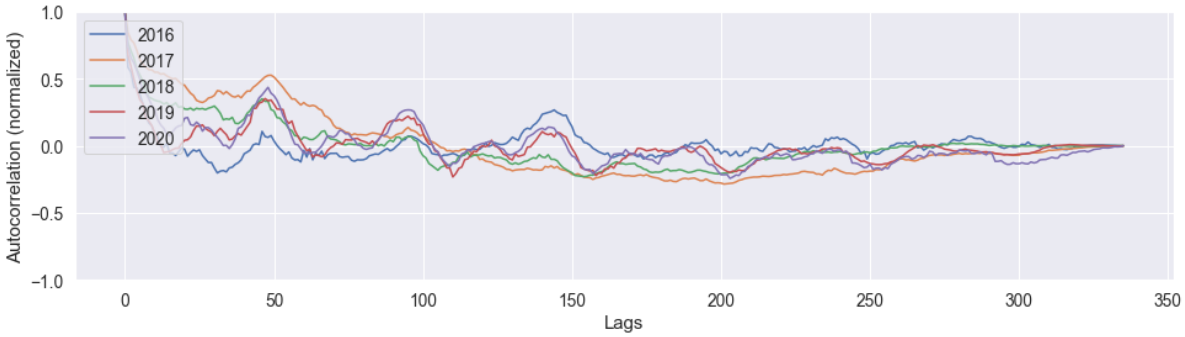
Corelation desappear as distance between original price and lagged price get farther although only a little difference can be seen.

Let's see which point is the point where the correlation decrease.

In [1306]:
```
# Function for computing autocorrelation between original value and lagged value
def compute_autocorr(x, lags=None):
    """
    Estimate autocorrelation for time series x and given lags

    References
    ----------
    ..[1] Time-Series Analysis: Forecasting and Control, 4th ed. (Box, Jenkins, Reinsel), p. 31
    """

    x = np.asarray(x)
    n_timepoints = x.shape[0]

    mu = x.mean()
    c_0 = np.var(x)  # np.std(x) * np.std(x)

    def corrcoef(k):
        """
        Estimate autocorrelation at lag k (sample autocorrelation function).
        """
        return np.sum((x[:-k] - mu) * (x[k:] - mu)) / n_timepoints / c_0

    lags = np.arange(1, n_timepoints)

    return np.array([1] + [corrcoef(lag) for lag in lags])
```

In [1307]:
```
fig, ax = plt.subplots(1, figsize=plt.figaspect(.25))

labels = ["2016", "2017", "2018", "2019", "2020"]
starts = [0, 13201, 30721, 48241, 65761] # index for each year
ends = [13200, 30720, 48240, 65760, 83328] # index for each year
for start, end, label in zip(starts, ends, labels):
    start =start
    end = end
    selected_period = close_price_graph.Close[start:end]
    rho = pd.Series(compute_autocorr(selected_period[0:336]))
    rho.plot(label=label)

ax.set(ylim=(-1, 1), xlabel="Lags", ylabel="Autocorrelation (normalized)")
plt.legend(loc="upper left")
```

Out[1307]: <matplotlib.legend.Legend at 0x134f859d0>



- Basically, corelation tend to be disappeared with few lagged value, but the speed depends on the year.
- Interestingly, negative correlation can be seen very often.
- This observation may be very important for rejecting EMH. It will be tested more in detail on the 2nd phase.

Trading vollume from 2014 to 2019

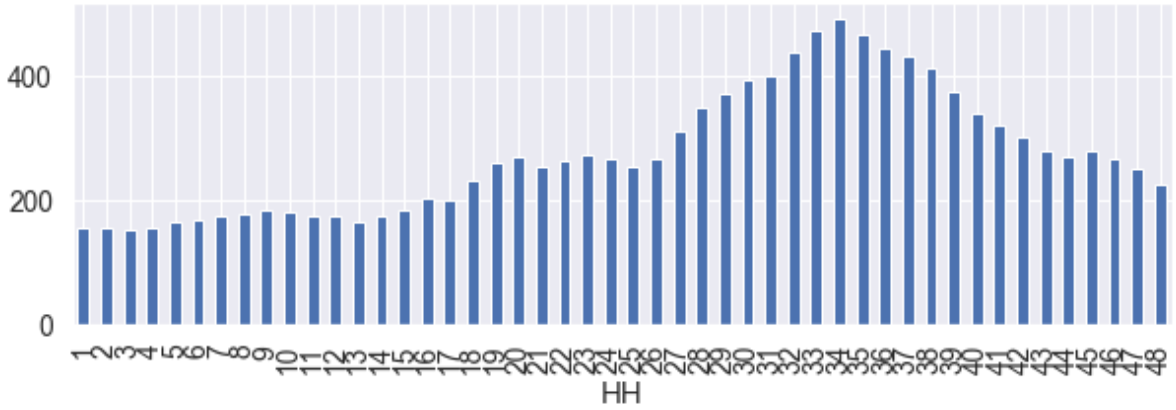
In [1308]:
```
intra_volume = df_intra[df_intra['Date'] <= '2020-12-31']
intra_volume = intra_volume[['Date', 'HH', 'Volume(MWh/h)', 'Volume(Tick count)']]
intra_volume.fillna(0, inplace=True)
intra_volume.head()
```

Out[1308]:

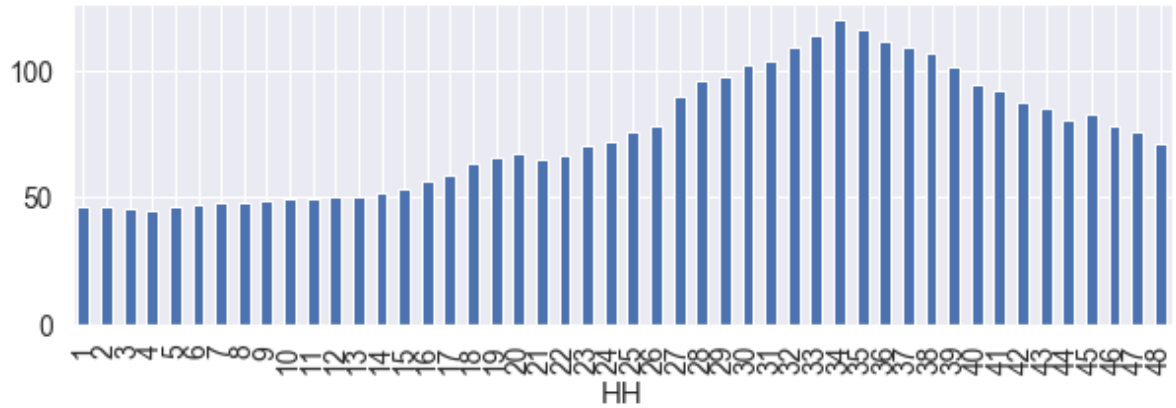
	Date	HH	Volume(MWh/h)	Volume(Tick count)
0	2016-04-01	1	0.7	1
1	2016-04-01	2	0.0	0
2	2016-04-01	3	0.8	1
3	2016-04-01	4	0.8	1
4	2016-04-01	5	0.8	1

Mean of the volume on each item

In [1787]:
```
# Plot the average trading volume (MWh/h) of each time slot.
IntraVolume_HH_mean =  intra_volume.groupby("HH").agg({"Volume(MWh/h)": 'mean'
, "Volume(Tick count)": 'mean'}).reset_index()
IntraVolume_HH_mean = IntraVolume_HH_mean.set_index("HH")
IntraVolume_HH_mean["Volume(MWh/h)"].plot.bar(figsize=(10, 3));
```



In [1788]:
```
# Plot the average trading volume (Tick count) of each time slot.
IntraVolume_HH_mean["Volume(Tick count)"].plot.bar(figsize=(10, 3));
```



The items of 34,35,36 is the peak.

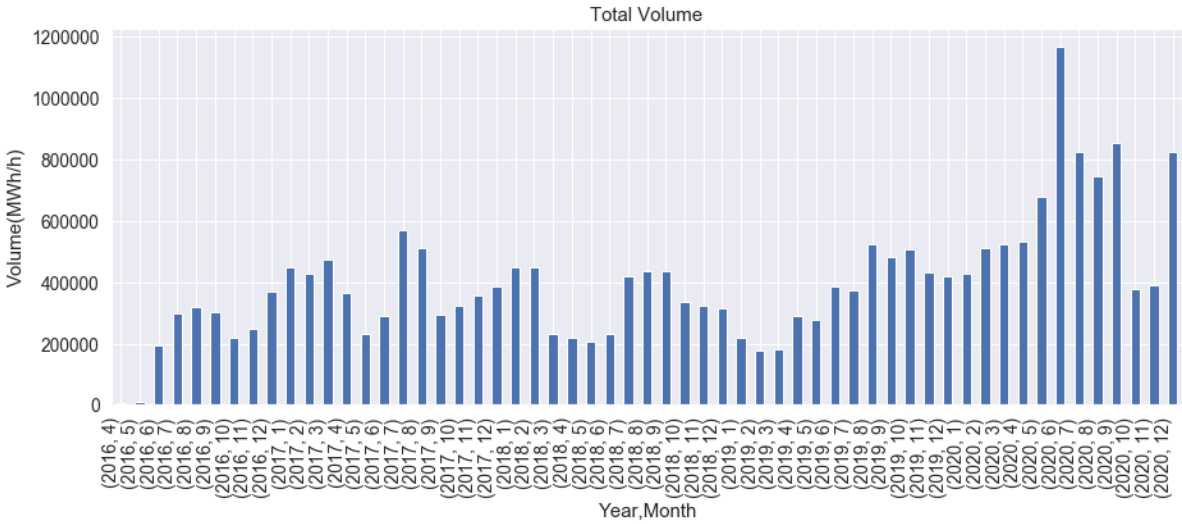
This shape is similar to daily basis avarage close price

In [1758]:
```
# Calculate the monthly total volume
IntraVolume_daily = intra_volume.groupby("Date").agg({"Volume(MWh/h)": 'sum', "Volume(Tick count)": 'sum'}).reset_index()
IntraVolume_daily["Year"] =  IntraVolume_daily["Date"].dt.year
IntraVolume_daily["Month"] =  IntraVolume_daily["Date"].dt.month
IntraVolume_daily = IntraVolume_daily.groupby(["Year", "Month"]).agg({"Volume(MWh/h)": 'sum', "Volume(Tick count)": 'sum'})

IntraVolume_daily.tail()
```

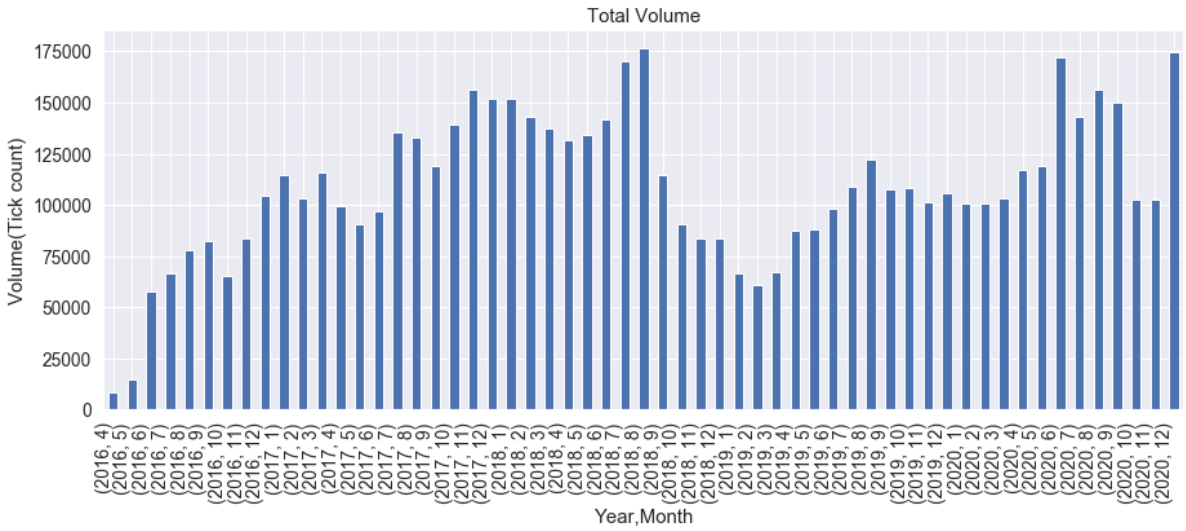
Out[1758]:

		Volume(MWh/h)	Volume(Tick count)
Year	**Month**		
2020	**8**	746479.5	156301
	9	852027.8	150067
	10	375764.1	102548
	11	390348.1	102376
	12	823287.3	174684

In [1764]:
```
# Plot the monthly total volume(MWh/h) from 2016 to 2020.
fig, ax = plt.subplots(1, figsize=(15, 6))

# # x axis
plt.gcf().autofmt_xdate()
# y axis
plt.ylabel("Volume(MWh/h)", rotation=90)
# plot
IntraVolume_daily['Volume(MWh/h)'].plot.bar()
plt.title('Total Volume')

plt.show();
```



In [1765]:
```
# Plot the monthly total volume(Tick count) from 2016 to 2020.
fig, ax = plt.subplots(1, figsize=(15, 6))

# # x axis
plt.gcf().autofmt_xdate()
# y axis
plt.ylabel("Volume(Tick count)", rotation=90)
# plot
IntraVolume_daily["Volume(Tick count)"].plot.bar();
plt.title('Total Volume')

plt.show();
```



The volume have not been increasing so much so far.

JEPX_Spot_datasets

http://www.jepx.org/market/index.html (http://www.jepx.org/market/index.html)

In [1818]:
```
# read the dataset (Here, these are re-organised after getting the latest dataset)

#Spot_dataset
df_spot_2016 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/spot_2016.csv', sep=',', header=0, encoding='shift_jis')
df_spot_2017 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/spot_2017.csv', sep=',', header=0, encoding='shift_jis')
df_spot_2018 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/spot_2018.csv', sep=',', header=0, encoding='shift_jis')
df_spot_2019 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/spot_2019.csv', sep=',', header=0, encoding='shift_jis')
df_spot_2020 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/spot_2020.csv', sep=',', header=0, encoding='shift_jis')

# marge all the spot datasets
df_spot = pd.concat([df_spot_2016, df_spot_2017])
df_spot = pd.concat([df_spot, df_spot_2018])
df_spot = pd.concat([df_spot, df_spot_2019])
df_spot = pd.concat([df_spot, df_spot_2020])
```

In [1820]:
```
# Drop the columns that has no data
df_spot = df_spot.drop(['Unnamed: 15', 'Unnamed: 21'], axis=1)
```

In [1822]:
```
# Rename the columns
df_spot = df_spot.rename(columns={'年月日': 'Date', '時刻コード': 'HH', '売り入札量(kWh)': 'Sell_volume(kWh)',
                                  '買い入札量(kWh)': 'Buy_volume(kWh)', '約定総量(kWh)': 'Total_volume(kWh)', 'システムプライス(円/kWh)': 'System_price(Yen/kWh)',
                                  'エリアプライス北海道(円/kWh)': 'Price_Hokkaido(Yen/kWh)', 'エリアプライス東北(円/kWh)': 'Price_Tohoku(Yen/kWh)',
                                  'エリアプライス東京(円/kWh)': 'Price_Tokyo(Yen/kWh)', 'エリアプライス中部(円/kWh)': 'Price_Chubu(Yen/kWh)',
                                  'エリアプライス北陸(円/kWh)': 'Price_Hokuriku(Yen/kWh)','エリアプライス関西(円/kWh)': 'Price_Kansai(Yen/kWh)',
                                  'エリアプライス中国(円/kWh)': 'Price_Chugoku(Yen/kWh)', 'エリアプライス四国(円/kWh)': 'Price_Shikoku(Yen/kWh)',
                                  'エリアプライス九州(円/kWh)': 'Price_Kyushu(Yen/kWh)', 'スポット・時間前平均価格(円/kWh)': 'SpotIntraMean',
                                  'α上限値×スポット・時間前平均価格(円/kWh)': 'αUpper_SpotIntraMean', 'α下限値×スポット・時間前平均価格(円/kWh)': 'αLower_SpotIntraMean',
                                  'α速報値×スポット・時間前平均価格(円/kWh)': 'αPrompt_SpotIntraMean', 'α確報値×スポット・時間前平均価格(円/kWh)': 'αComfirmed_SpotIntraMean',
                                  '回避可能原価全国値(円/kWh)': 'Avoidable_price(Yen/kWh)', '回避可能原価北海道(円/kWh)': 'Avoidable_price_HOK(Yen/kWh)',
                                  '回避可能原価東北(円/kWh)': 'Avoidable_price_TOH(Yen/kWh)', '回避可能原価東京(円/kWh)': 'Avoidable_price_TKO(Yen/kWh)',
                                  '回避可能原価中部(円/kWh)': 'Avoidable_price_CHU(Yen/kWh)', '回避可能原価北陸(円/kWh)': 'Avoidable_price_HKU(Yen/kWh)',
                                  '回避可能原価関西(円/kWh)': 'Avoidable_price_KAN(Yen/kWh)', '回避可能原価中国(円/kWh)': 'Avoidable_price_CHG(Yen/kWh)',
                                  '回避可能原価四国(円/kWh)': 'Avoidable_price_SHI(Yen/kWh)', '回避可能原価九州(円/kWh)': 'Avoidable_price_KYU(Yen/kWh)'
                                 })
```

In [1823]:
```
## Apply to_datetime
df_spot["Date"] = pd.to_datetime(df_spot["Date"])
```

In [1824]:
```
df_spot.head()
```

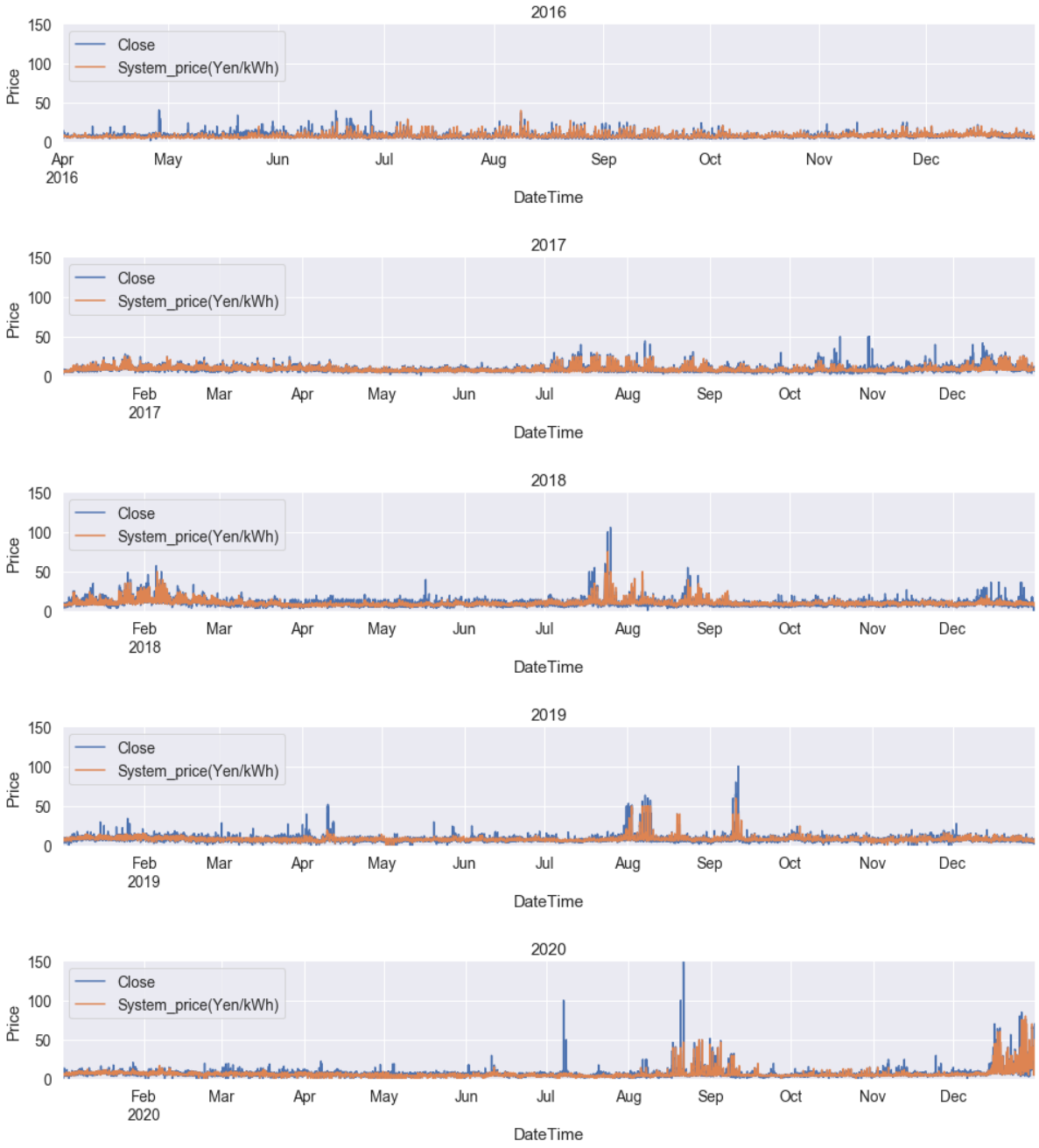
Out[1824]:

	Date	HH	Sell_volume(kWh)	Buy_volume(kWh)	Total_volume(kWh)	System_price(Yen/kWh)	P
0	2016-04-01	1	5077000	2120500	675500	6.61	
1	2016-04-01	2	5621500	2239000	755500	6.34	
2	2016-04-01	3	5710500	2219000	713500	6.34	
3	2016-04-01	4	5536000	2202500	717500	6.25	
4	2016-04-01	5	5461000	2192000	870500	6.57	

The relation between spot and close price

In [1839]:
```
# Make another DataFrame of df_spot for graph
spot_price_graph = df_spot.copy()
spot_price_graph = pd.merge(spot_price_graph, HH_table, how="left", on="HH")
spot_price_graph["DateTime"] = pd.to_datetime(spot_price_graph["Date"].astype(str
) + " " + spot_price_graph["Time"], format='%Y-%m-%d %H:%M')
spot_price_graph = spot_price_graph.drop(["Date", "HH", "Time"], axis=1)
spot_price_graph.set_index("DateTime", inplace=True)
```

In [1841]:
```
# Plot close price for a year

years = [2016, 2017, 2018, 2019, 2020]

for year in years:
    start = str(year) + "-01-01 00:00:00"
    end = str(year) + "-12-31 23:30:00"
    #Pick up the periods for Close price
    close_price_graph_y = close_price_graph["Close"][np.logical_and(start < close_price_graph.index, close_price_graph.index <= end)]
    #Pick up the periods for Spot price
    spot_price_graph_y = spot_price_graph["System_price(Yen/kWh)"][np.logical_and(start < spot_price_graph.index, spot_price_graph.index <= end)]

    fig, ax = plt.subplots(1, figsize=plt.figaspect(.05))

    #Plot Close price
    close_price_graph_y.plot(ax=ax)
    #Plot Spot price
    spot_price_graph_y.plot(ax=ax)

    ax.set_ylim(0, 150)
    ax.set(title=year, ylabel="Price", xlabel="DateTime")
    plt.legend(loc="upper left");
```



System price used to be linked with close price on Intra market. However, recently, the relationship is changing.

Price spike seems to be affected by seasonal factors...? [Spot]

- 2018-02(50 Yen)
- 2018-07 (70 Yen) -->Irrational bidding mentioned by the following news, Max temprature 39°C on 23th (Tokyo)
- 2018-09 (40 Yen) -->Hokkaido blackout due to earthquake
- 2019-07 (50 Yen)
- 2019-09 (60 Yen) -->36°C on 9th(Tokyo)、35°C on 10th(Tokyo)
- 2020-08 (50 Yen)
- 2020-09 (50 Yen)
- 2020-12 (80 Yen) -->LNG shock

[Close (Only or more than Spot price)]

- 2018-07 (100 Yen) -->Irrational bidding mentioned by the following news, Max temprature 39°C on 23th (Tokyo)
- 2019-09 (100 Yen) -->36°C on 9th(Tokyo)、35°C on 10th(Tokyo)
- 2020-07 (100 Yen) --> Only close price was spiked
- 2020-08 (150 Yen) --> Only close price was spiked

(Reference) Web news and a minute of meating among officials regarding price spike

- Nikkei cross tech (2018) in Japanese 「市場からの調達を確実なものにするため、相場感を無視した高値の買い入札が見られた」: https://xtech.nikkei.com/dm/atcl/feature/15/031400070/073100071/?P=2 (https://xtech.nikkei.com/dm/atcl/feature/15/031400070/073100071/?P=2)
- Meti (3P.) in Japanese 「2019年の100円スパイクはリテラシーの低い参加者によるインバランス回避のための不合理な入札」: https://www.emsc.meti.go.jp/activity/emsc_system/pdf/036_04_00.pdf (https://www.emsc.meti.go.jp/activity/emsc_system/pdf/036_04_00.pdf)

Possible reasons of price spike:

- Irrational bidding behaviour
- Extreme temprature (Especially in summer season)
- Lack of LNG gas due to Covid-19 (From the end of 2020 to the beginning of 2021)

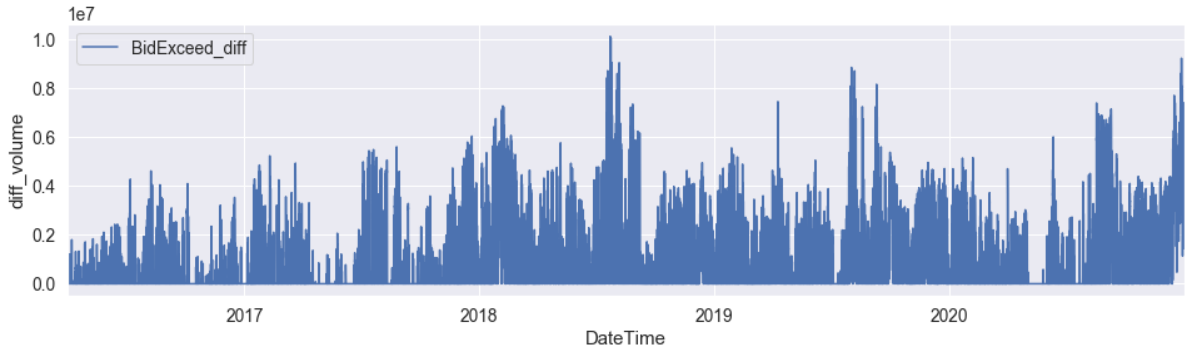
Try to make and add features of:

- Exceed bidding volume which can create with DA market information
- Weather forecast
- Demand peak forecast

Bidding balance

In [1842]:
```
# Make a columns for Bid balance
BidExceed_diff = []
Buy_volume_list = list(spot_price_graph["Buy_volume(kWh)"])
Sell_volume_list = list(spot_price_graph["Sell_volume(kWh)"])

for Buy_volume, Sell_volume in zip(Buy_volume_list, Sell_volume_list):
    if  Buy_volume > Sell_volume:
        BidExceed_diff.append(int(Buy_volume - Sell_volume))
    else:
        # Only exceeding volume is added
        BidExceed_diff.append(0)

index = spot_price_graph.index
spot_price_graph['BidExceed_diff'] = pd.Series(BidExceed_diff, index=index)
```

In [1844]:
```
# Simple plot of Sell/buy volume
fig, ax = plt.subplots(1, figsize=plt.figaspect(.25))

start = "2016-04-01"
end = "2020-12-31"

spot_price_graph['BidExceed_diff'][start : end].plot()

ax.set(ylabel="diff_volume", xlabel="DateTime")
plt.legend(loc="upper left");
```

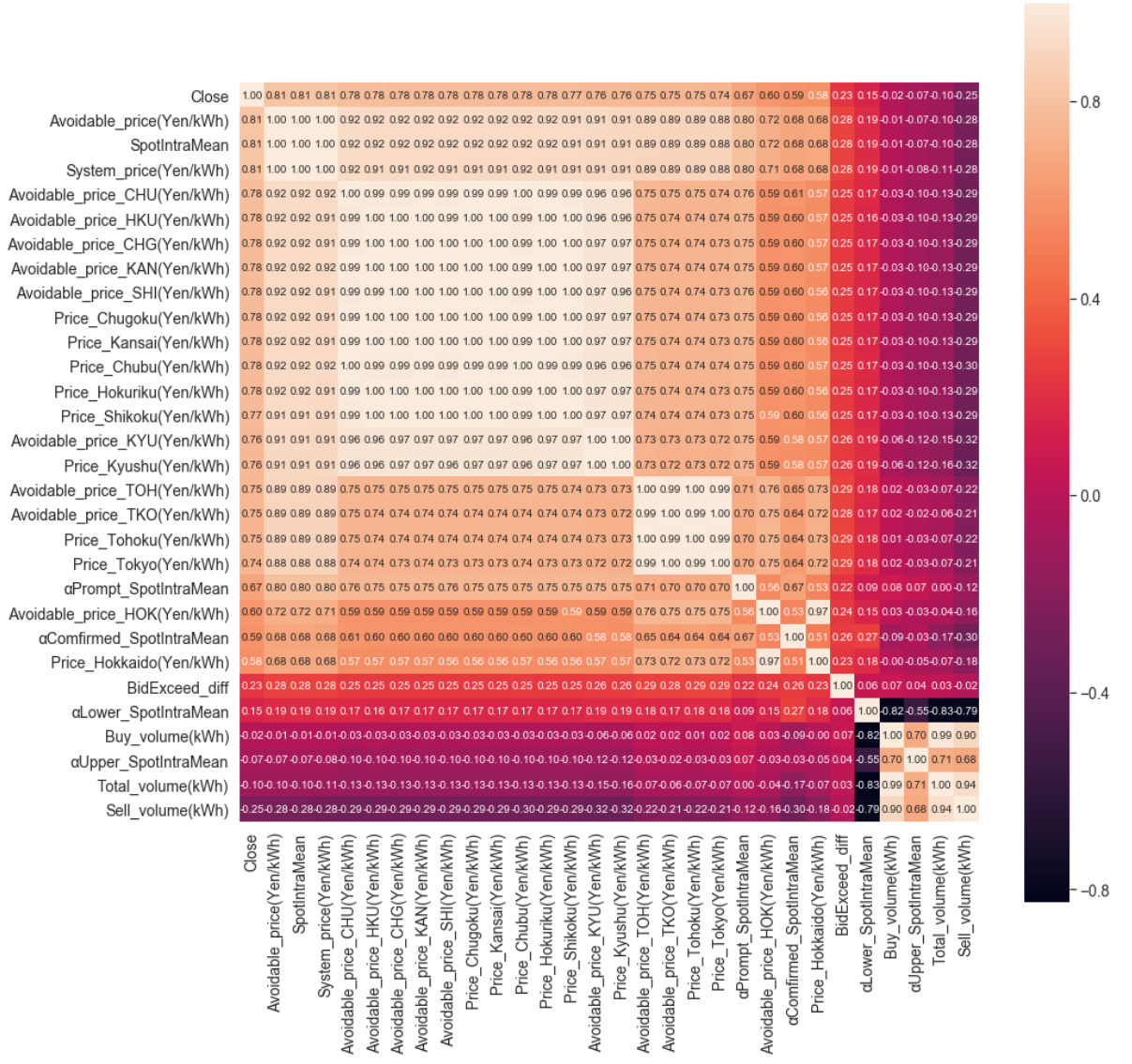


In [1845]:
```
# Add the columns on df_spot
df_spot['BidExceed_diff'] = pd.Series(BidExceed_diff)
```

The correlation between Area price in different area and "Close" price.

In [1865]:
```
# Pickup close price
intra_close = df_intra[['Date', 'HH', 'Close']]
intra_close["Date"] = pd.to_datetime(intra_close["Date"])

#Marge the tables
spot_intra = pd.merge(df_spot, intra_close,  how='left', on=['Date', 'HH'])

#Adjust the marged table
spot_intra = spot_intra.drop("HH", axis=1)
spot_intra = spot_intra[spot_intra["Date"] <= "2020/12/31"]
spot_intra.fillna(0, inplace=True)
```

In [1866]:
```
# Close price correlation matrix
corrmat = spot_intra.corr()

k = 50 # The number of variables on the heatmap
cols = corrmat.nlargest(k, 'Close')['Close'].index
cm = np.corrcoef(spot_intra[cols].values.T)
sns.set(font_scale=1.25)
f, ax = plt.subplots(figsize=(15,15))
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```

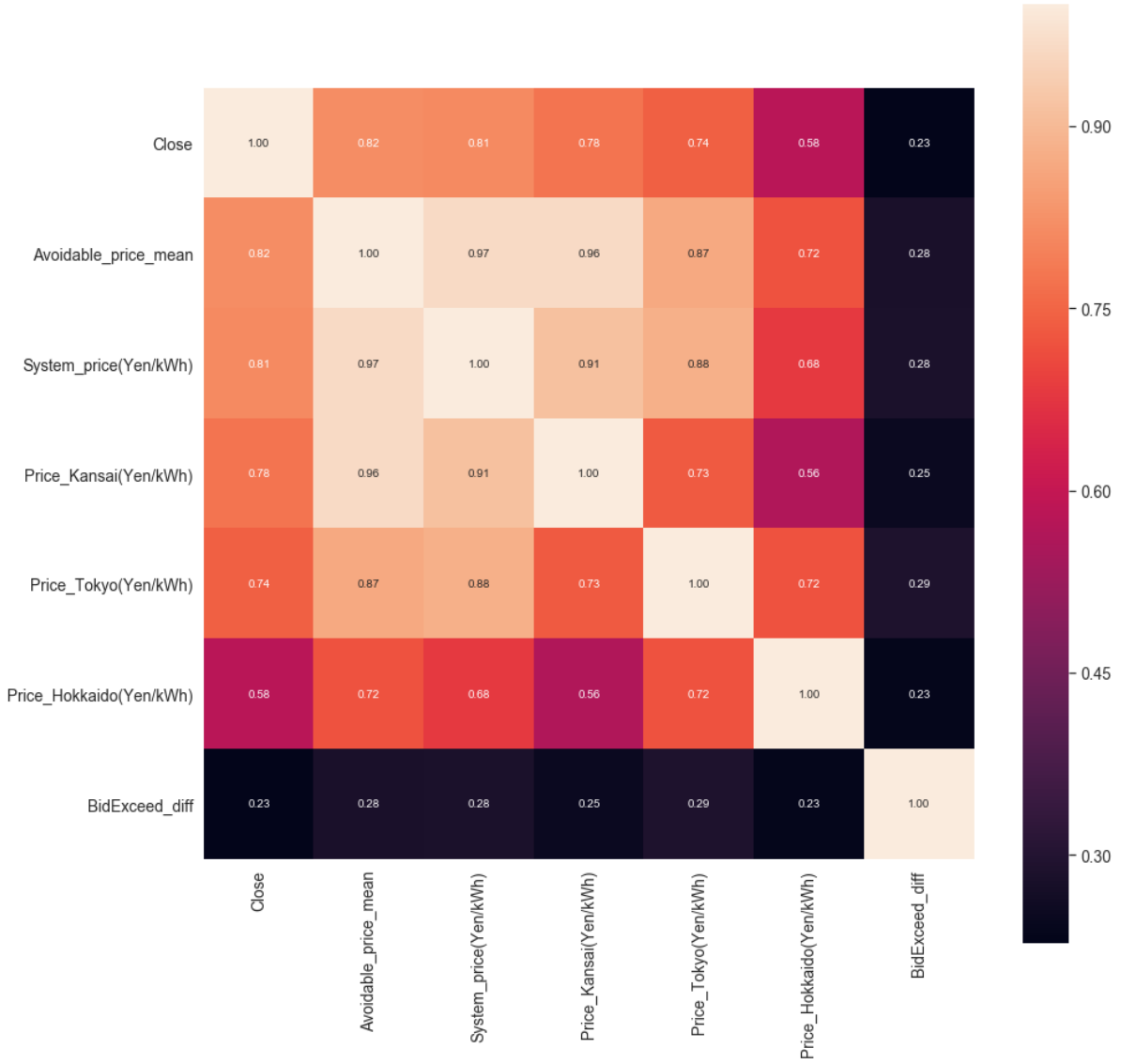


The following columns may be able to dropped.

- Avoidable price, SpotIntraMean --> They are the same as system price.
- Avoidable price in each area --> Take average and confirm the correlation with others --> confirm.
- Area price --> Choose representitative area which has relatively higher demand (System price, Tokyo, Kansai, Hokkaido) ※ Refer to 3.3 "Hourly_Total_Demand"
- Buy/Sell/Total volume
- The information related to SpotIntraMean --> They are not updated firsthand.

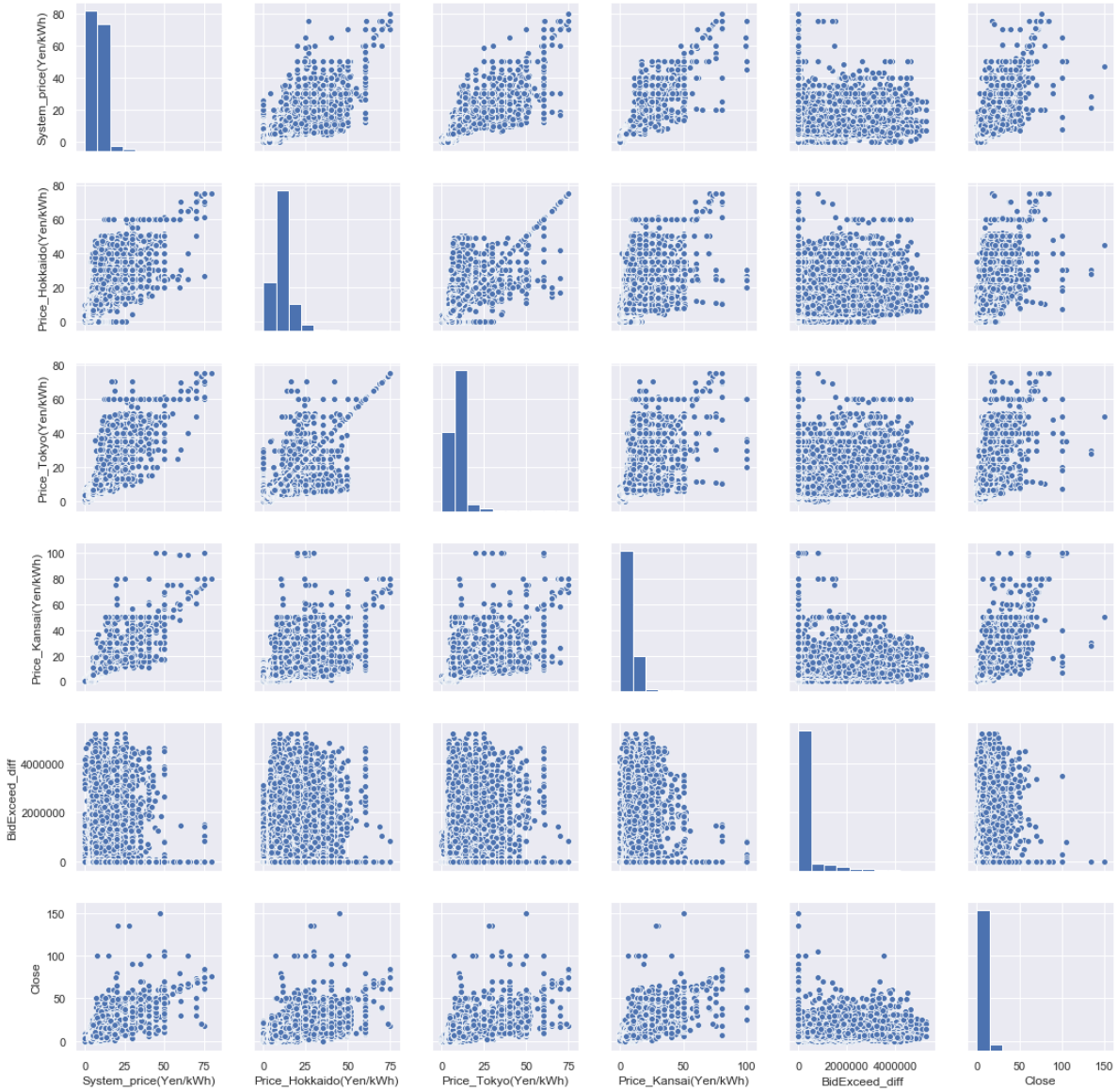
In [1869]:
```
# Try to see the national average of avoidable price
spot_intra["Avoidable_price_mean"] = round((spot_intra["Avoidable_price_HOK(Yen/kWh)"] + spot_intra["Avoidable_price_TOH(Yen/kWh)"]
                                + spot_intra["Avoidable_price_TKO(Yen/kWh)"] + spot_intra["Avoidable_price_CHU(Yen/kWh)"]
                                + spot_intra["Avoidable_price_HKU(Yen/kWh)"] + spot_intra["Avoidable_price_KAN(Yen/kWh)"]
                                + spot_intra["Avoidable_price_CHG(Yen/kWh)"] + spot_intra["Avoidable_price_SHI(Yen/kWh)"]
                                + spot_intra["Avoidable_price_KYU(Yen/kWh)"])/9, 2)
spot_intra = spot_intra.drop([
    'Sell_volume(kWh)', 'Buy_volume(kWh)', 'Total_volume(kWh)',
    'Price_Tohoku(Yen/kWh)', 'Price_Chubu(Yen/kWh)',
    'Price_Hokuriku(Yen/kWh)', 'Price_Chugoku(Yen/kWh)', 'Price_Shikoku(Yen/kWh)',
    'Price_Kyushu(Yen/kWh)', 'SpotIntraMean', 'αUpper_SpotIntraMean',
    'αLower_SpotIntraMean', 'αPrompt_SpotIntraMean',
    'αComfirmed_SpotIntraMean', 'Avoidable_price(Yen/kWh)',
    'Avoidable_price_HOK(Yen/kWh)', 'Avoidable_price_TOH(Yen/kWh)',
    'Avoidable_price_TKO(Yen/kWh)', 'Avoidable_price_CHU(Yen/kWh)',
    'Avoidable_price_HKU(Yen/kWh)', 'Avoidable_price_KAN(Yen/kWh)',
    'Avoidable_price_CHG(Yen/kWh)', 'Avoidable_price_SHI(Yen/kWh)',
    'Avoidable_price_KYU(Yen/kWh)'], axis=1)
```

In [1870]:
```
# Close price correlation matrix
corrmat = spot_intra.corr()

k = 50 # The number of variables on the heatmap
cols = corrmat.nlargest(k, 'Close')['Close'].index
cm = np.corrcoef(spot_intra[cols].values.T)
sns.set(font_scale=1.25)
f, ax = plt.subplots(figsize=(15,15))
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```



"Avoidable_price_mean" can be avoided as well.

In [1877]:
```
spot_intra = spot_intra.drop("Avoidable_price_mean", axis=1)

# scatterplot
sns.set()
cols = spot_intra.columns
sns.pairplot(spot_intra[cols], size=2.5)
plt.show()
```



Some outliers can be seen. They will be treated after all the datasets are merged.

In [1872]:
```
# Drop the above columns from df_spot
df_spot = df_spot.drop([
    'Sell_volume(kWh)', 'Buy_volume(kWh)', 'Total_volume(kWh)',
    'Price_Tohoku(Yen/kWh)', 'Price_Chubu(Yen/kWh)',
    'Price_Hokuriku(Yen/kWh)', 'Price_Chugoku(Yen/kWh)', 'Price_Shikoku(Yen/kWh)',
    'Price_Kyushu(Yen/kWh)', 'SpotIntraMean', 'αUpper_SpotIntraMean',
    'αLower_SpotIntraMean', 'αPrompt_SpotIntraMean',
    'αComfirmed_SpotIntraMean', 'Avoidable_price(Yen/kWh)',
    'Avoidable_price_HOK(Yen/kWh)', 'Avoidable_price_TOH(Yen/kWh)',
    'Avoidable_price_TKO(Yen/kWh)', 'Avoidable_price_CHU(Yen/kWh)',
    'Avoidable_price_HKU(Yen/kWh)', 'Avoidable_price_KAN(Yen/kWh)',
    'Avoidable_price_CHG(Yen/kWh)', 'Avoidable_price_SHI(Yen/kWh)',
    'Avoidable_price_KYU(Yen/kWh)'
     ], axis=1)
```

In [1873]:
```
df_spot.head()
```

Out[1873]:

	Date	HH	System_price(Yen/kWh)	Price_Hokkaido(Yen/kWh)	Price_Tokyo(Yen/kWh)	Price_Kar
0	2016-04-01	1	6.61	8.66	6.69	
1	2016-04-01	2	6.34	8.66	6.34	
2	2016-04-01	3	6.34	7.20	6.34	
3	2016-04-01	4	6.25	7.10	6.03	
4	2016-04-01	5	6.57	7.10	6.57	

Hourly_TotalDemand_with_Subtotal_of_EachArea

http://occtonet.occto.or.jp/public/dfw/RP11/OCCTO/SD/LOGIN_login#
(http://occtonet.occto.or.jp/public/dfw/RP11/OCCTO/SD/LOGIN_login#)

In [1876]:
```
# Read dataset
df_HourlyDemand_All = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/需要実績(日別)_ALL.csv', sep=',', header=0, encoding='shift_jis')
```

In [1878]:
```
# Rename the columns
df_HourlyDemand_All = df_HourlyDemand_All.rename(columns={'年月日': 'Date', '時間帯': 'Period',
                                                          '北海道': 'HourlyDemand_Hokkaido', '東北': 'HourlyDemand_Tohoku', '東京': 'HourlyDemand_Tokyo',
                                                          '中部': 'HourlyDemand_Chubu', '北陸': 'HourlyDemand_Hokuriku', '関西': 'HourlyDemand_Kansai',
                                                          '中国': 'HourlyDemand_Chugoku', '四国': 'HourlyDemand_Shikoku', '九州': 'HourlyDemand_Kyushu',
                                                          '沖縄': 'HourlyDemand_Okinawa', '10エリア計': 'HourlyDemand_Total'
})
```

In [1879]:
```
# Convert text on the Period column to NaN
df_HourlyDemand_All['Period'] = df_HourlyDemand_All['Period'].replace({'日電力量(MWh)': np.nan, '日最大電力(MW)': np.nan})

# Remove the rows that include NaN
df_HourlyDemand_All = df_HourlyDemand_All[~df_HourlyDemand_All.Period.str.contains("NaN", na=True)]

# Adjust for making "Date" and "Time" columns
df_HourlyDemand_All["DateTime"] = pd.to_datetime(df_HourlyDemand_All["Date"] + " " + df_HourlyDemand_All["Period"].str[:5])
df_HourlyDemand_All["Time"] = pd.to_datetime(df_HourlyDemand_All["DateTime"]).dt.time
df_HourlyDemand_All["Date"] = pd.to_datetime(df_HourlyDemand_All["DateTime"]).dt.date
df_HourlyDemand_All["Date"] = pd.to_datetime(df_HourlyDemand_All["Date"])

# Total_Demand should be dropped because it is also included in another dataset
df_HourlyDemand_All = df_HourlyDemand_All.drop(["Period", "DateTime", "HourlyDemand_Total"], axis=1)

print(df_HourlyDemand_All["Time"].value_counts())
```

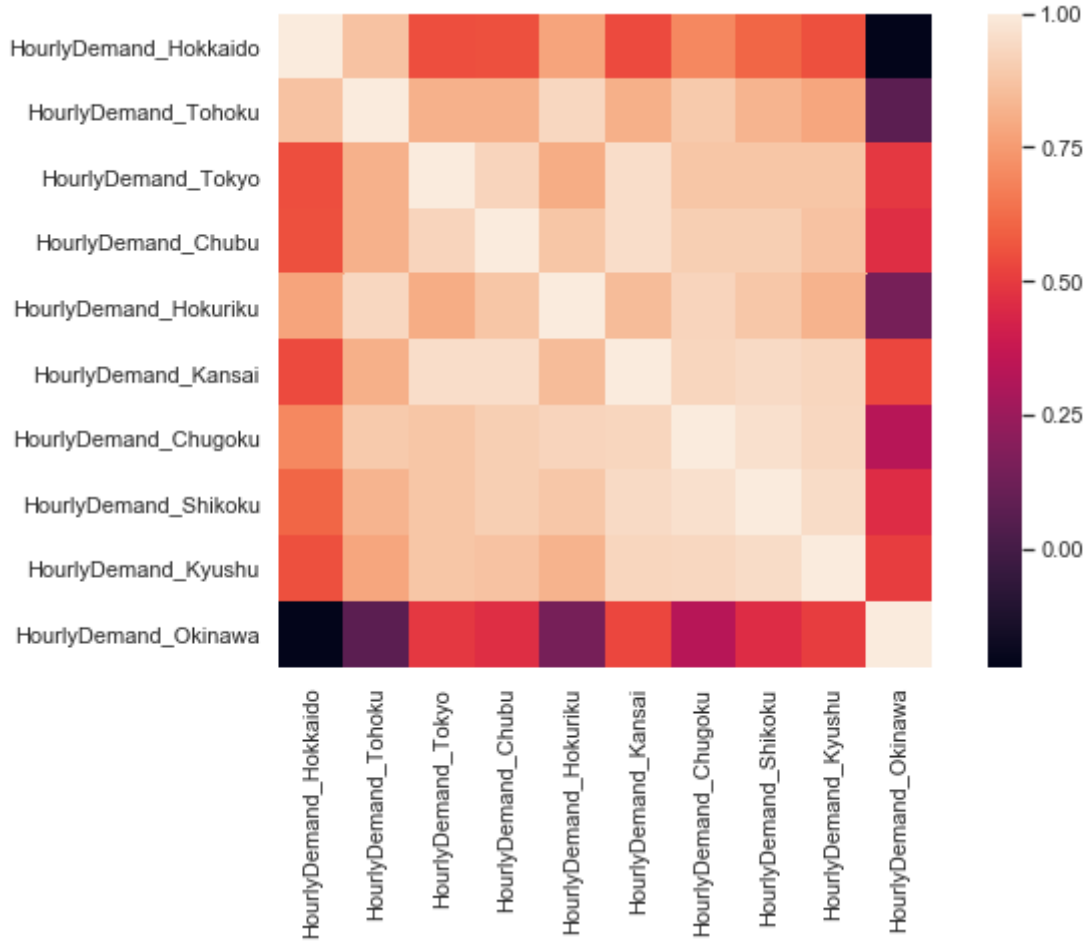
```
20:00:00    1755
21:00:00    1755
02:00:00    1755
12:00:00    1755
11:00:00    1755
00:00:00    1755
05:00:00    1755
10:00:00    1755
04:00:00    1755
16:00:00    1755
13:00:00    1755
06:00:00    1755
01:00:00    1755
14:00:00    1755
15:00:00    1755
23:00:00    1755
17:00:00    1755
19:00:00    1755
22:00:00    1755
08:00:00    1755
07:00:00    1755
03:00:00    1755
18:00:00    1755
09:00:00    1755
Name: Time, dtype: int64
```

In [1880]:
```
df_HourlyDemand_All.head()
```

Out[1880]:

	Date	HourlyDemand_Hokkaido	HourlyDemand_Tohoku	HourlyDemand_Tokyo	HourlyDemand_
0	2016-04-01	3166	7887	25547	
1	2016-04-01	3282	8158	24334	
2	2016-04-01	3387	8420	23934	
3	2016-04-01	3468	8584	23753	
4	2016-04-01	3598	8806	23897	

In [1881]:
```
# correlation matrix
corrmat = df_HourlyDemand_All.corr()
f, ax = plt.subplots(figsize=(10,6))
sns.heatmap(corrmat, square=True);
```



In [1885]:
```
df_HourlyDemand_All.describe()
```

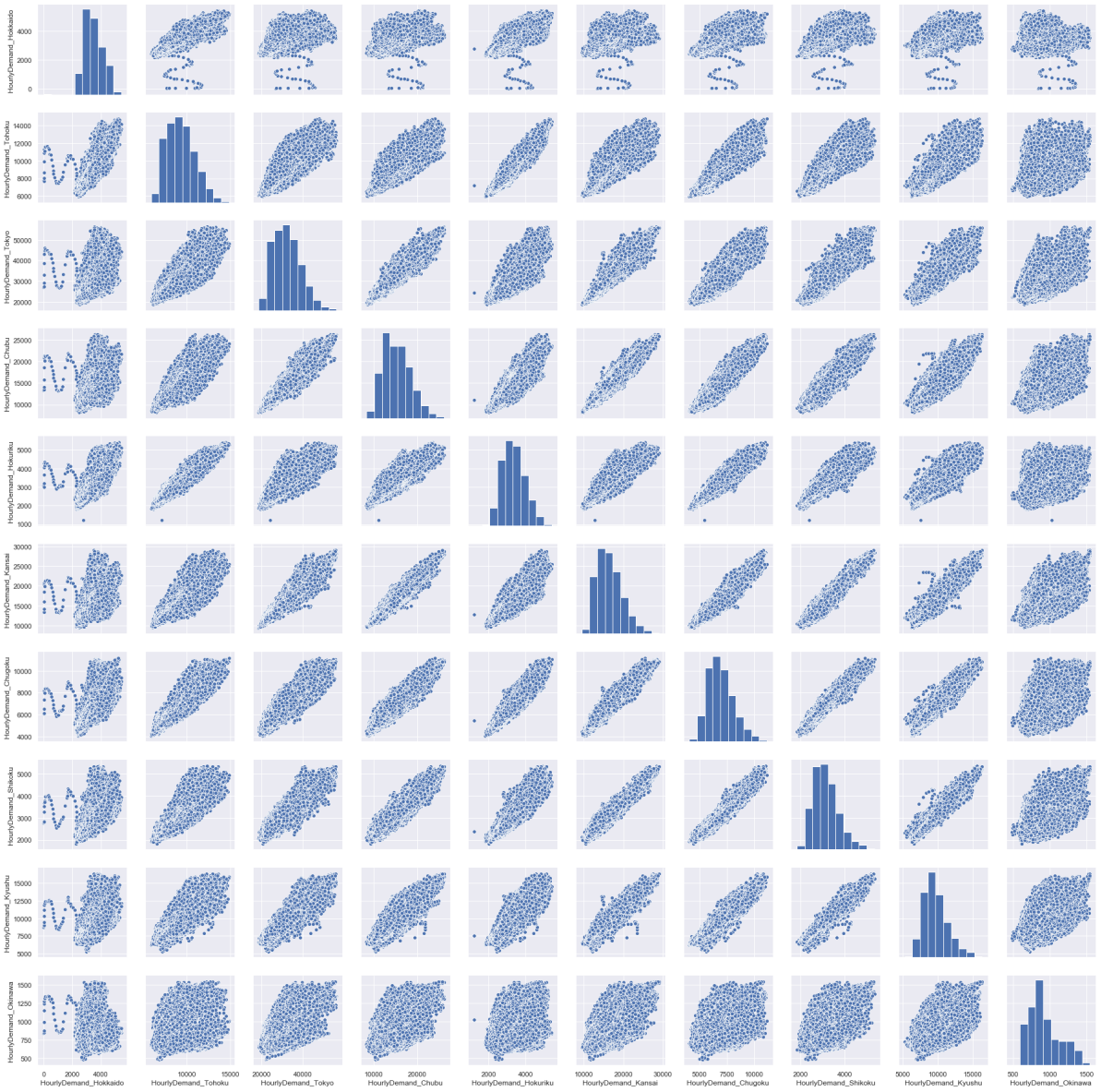
Out[1885]:

	HourlyDemand_Hokkaido	HourlyDemand_Tohoku	HourlyDemand_Tokyo	HourlyDemand_Ch
count	42120.000000	42120.000000	42120.000000	42120.000
mean	3489.348884	9304.052042	32430.130722	15233.754
std	622.181993	1544.686428	6535.346217	3139.964
min	0.000000	5955.000000	18769.000000	8258.000
25%	2982.000000	8031.000000	27259.000000	12762.000
50%	3394.000000	9204.000000	31962.000000	14909.000
75%	3942.000000	10340.000000	36611.250000	17333.000
max	5422.000000	14796.000000	56532.000000	26243.000

As the same as the spot price, keep the following information and drop others (Refer to 3.2.3)

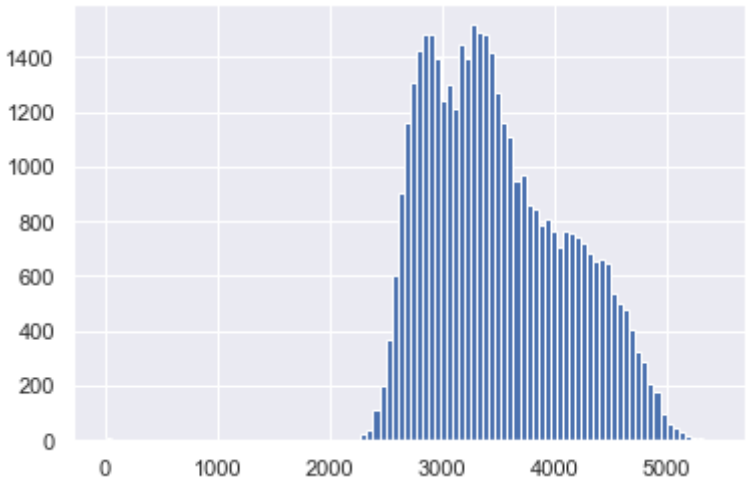
- System price
- Tokyo
- Kansai
- Hokkaido
- Okinawa

In [1882]:
```
# scatterplot
sns.set()
cols = df_HourlyDemand_All.columns
sns.pairplot(df_HourlyDemand_All[cols], size=2.5)
plt.show()
```



Try to see the irregular patten of the demand on Hokkaido area.

In [1883]:
```
df_HourlyDemand_All["HourlyDemand_Hokkaido"].hist(bins=100, rwidth=1.0);
```



In [1884]:
```
df_HourlyDemand_All[df_HourlyDemand_All["HourlyDemand_Hokkaido"] < 2300].sort_values('HourlyDemand_Hokkaido', ascending=False).head()
```

Out[1884]:

	Date	HourlyDemand_Hokkaido	HourlyDemand_Tohoku	HourlyDemand_Tokyo	HourlyDem
29362	2019-05-05	2299	6853	22408	
23194	2018-09-10	2298	6560	25218	
23136	2018-09-07	2295	8311	34998	
23135	2018-09-07	2291	8560	36381	
38468	2020-04-19	2288	7416	23157	

The cause of the outlier of "HourlyDemand_Hokkaido" was the blackout due to the earthquake. -->Remove after all the datasets are merged.

Hourly_TodalDemand_with_Generation_from_DifferentPower

Hokkaido：
https://www.hepco.co.jp/network/renewable_energy/fixedprice_purchase/supply_demand_results.html (https://www.hepco.co.jp/network/renewable_energy/fixedprice_purchase/supply_demand_results.html)

Tohoku: https://setsuden.nw.tohoku-epco.co.jp/download.html (https://setsuden.nw.tohoku-epco.co.jp/download.html)

Tokyo: https://www.tepco.co.jp/forecast/html/area_data-j.html (https://www.tepco.co.jp/forecast/html/area_data-j.html)

Chubu: https://powergrid.chuden.co.jp/denkiyoho/ (https://powergrid.chuden.co.jp/denkiyoho/)

Hokuriku: http://www.rikuden.co.jp/nw_jyukyudata/area_jisseki.html (http://www.rikuden.co.jp/nw_jyukyudata/area_jisseki.html)

Kansai: https://www.kansai-td.co.jp/denkiyoho/area-performance.html (https://www.kansai-td.co.jp/denkiyoho/area-performance.html)

Chugoku: https://www.energia.co.jp/nw/service/retailer/data/area/ (https://www.energia.co.jp/nw/service/retailer/data/area/)

Shikoku: https://www.yonden.co.jp/nw/renewable_energy/data/supply_demand.html (https://www.yonden.co.jp/nw/renewable_energy/data/supply_demand.html)

Kyushu: https://www.kyuden.co.jp/td_service_wheeling_rule-document_disclosure (https://www.kyuden.co.jp/td_service_wheeling_rule-document_disclosure)

Okinawa: https://www.okiden.co.jp/business-support/service/supply-and-demand/ (https://www.okiden.co.jp/business-support/service/supply-and-demand/)

In [1886]:
```
# File path
path = '/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/actual_generation_demand/'
```

In [1887]:
```
# Tohoku as a base
df_Demand_AreaPlant = pd.read_csv(path + "/Demand_plant_Tohoku.csv", sep=',', he
ader=0, encoding='shift_jis')
df_Demand_AreaPlant["DateTime"] =  pd.to_datetime(df_Demand_AreaPlant["DateTi
me"])

Areas = ["Hokkaido", "Tokyo", "Chubu", "Hokuriku", "Kansai", "Chugoku", "Shikoku", "K
yushu", "Okinawa"]

# Add other areas
for area in Areas:
    df = pd.read_csv(path + "/Demand_plant_" + area + ".csv", sep=',', header=0, encod
ing='shift_jis')
    df["DateTime"] = pd.to_datetime(df["DateTime"])
    df_Demand_AreaPlant = pd.merge(df_Demand_AreaPlant, df, how="left", on="DateT
ime")

df_Demand_AreaPlant.fillna(0, inplace=True)
```

In [1888]:
```
# Drop columns that contain only 0
tmp=df_Demand_AreaPlant==0
col=df_Demand_AreaPlant.columns.values[tmp.sum(axis=0)==len(df_Demand_AreaPla
nt)]
df_Demand_AreaPlant = df_Demand_AreaPlant.drop(col, axis=1)
```

In [1889]:
```
# Adjust Datatype from object to float for Tokyo and Kyushu area (Had to be pre-proces
sed separately)
object_cols = ["TotalDemand_TKO", "Thermal_TKO", "TotalSupply_TKO", "TotalDeman
d_KYU",
               "Nuclear_KYU", "Thermal_KYU", "PVCurtailment_KYU", "PumpedStorage_KYU"
, "Interconnection_KYU"]

for col in object_cols:
    df_Demand_AreaPlant[col] = df_Demand_AreaPlant[col].apply(lambda x: float(x.repl
ace(",", "")))
    df_Demand_AreaPlant[col] = pd.to_numeric(df_Demand_AreaPlant[col].replace({',':
'', '—':0}), downcast='integer')

# Adjust Datatype from object to float for Chugoku area (Had to be pre-processed separ
ately)
object_cols = ["PVCurtailment_CHG", "WindCurtailment_CHG"]

for col in object_cols:
    df_Demand_AreaPlant[col] = pd.to_numeric(df_Demand_AreaPlant[col].replace({',':
'', '—':0}), downcast='integer')
```

In [1890]:
```
# Calculate statistical results for each area
def get_group_stats(df):
    col_names = ["TotalDemand", "Water", "Thermal", "Nuclear", "PV", "PVCurtailment", "Wind",
                 "WindCurtailment", "Geothermal", "Biomass", "PumpedStorage", "Interconnection"]
    for group in col_names:
        cols = [col for col in df.columns if group in col]
        df[f"Allarea_{group}"] = df[cols].sum(axis=1)

    return df
```

In [1891]:
```
df_Demand_AreaPlant = get_group_stats(df_Demand_AreaPlant)
```

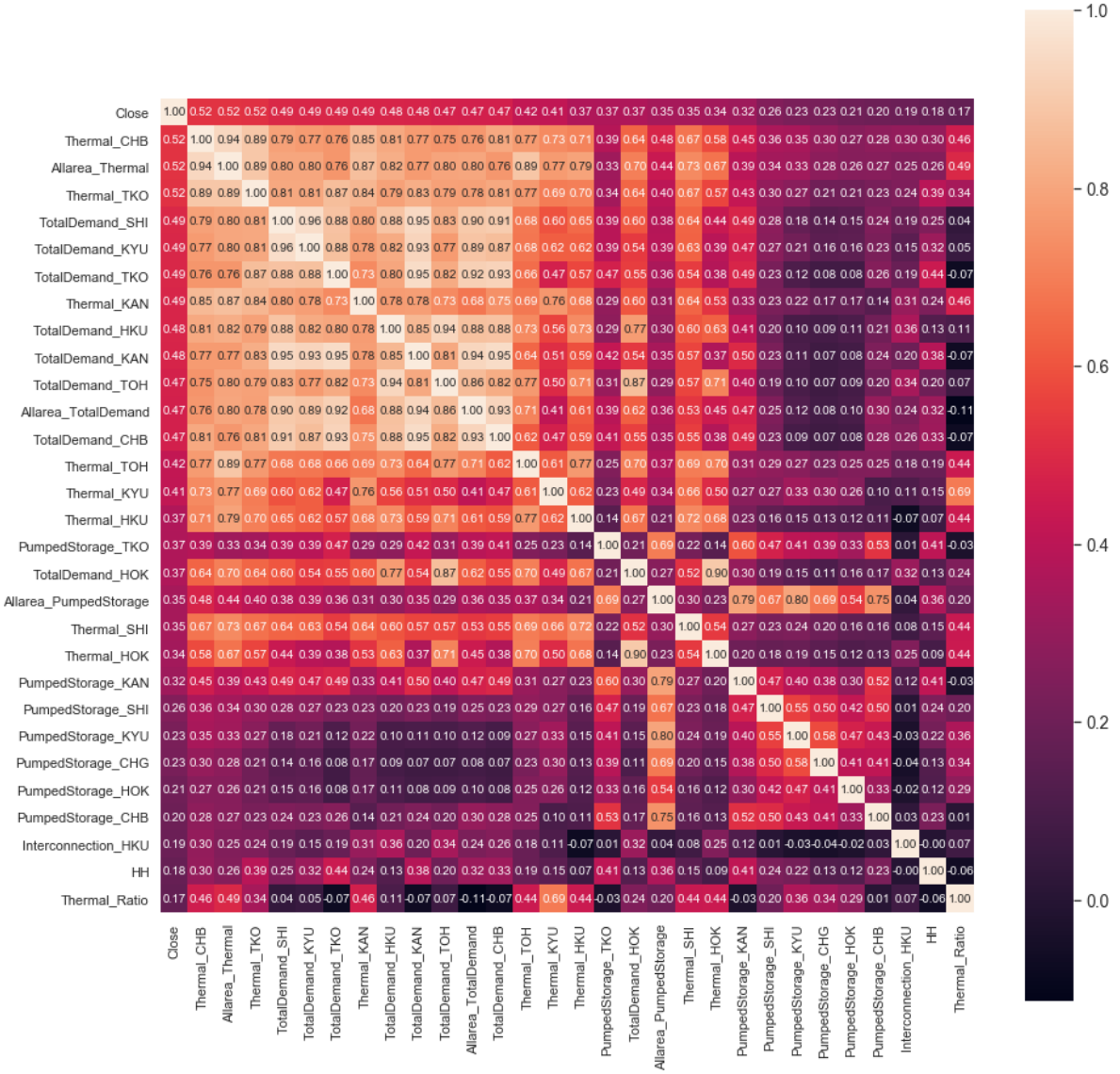
In [1892]:
```
df_Demand_AreaPlant["Water_Ratio"] = round(df_Demand_AreaPlant["Allarea_Water"] / df_Demand_AreaPlant["Allarea_TotalDemand"], 2)
df_Demand_AreaPlant["Thermal_Ratio"] = round(df_Demand_AreaPlant["Allarea_Thermal"] / df_Demand_AreaPlant["Allarea_TotalDemand"], 2)
df_Demand_AreaPlant["Geothermal_Ratio"] = round(df_Demand_AreaPlant["Allarea_Geothermal"] / df_Demand_AreaPlant["Allarea_TotalDemand"], 2)
```

In [1893]:
```
# Make the columns for "Time" and "Date"
df_Demand_AreaPlant["Time"] = pd.to_datetime(df_Demand_AreaPlant["DateTime"]).dt.time
df_Demand_AreaPlant["Date"] = pd.to_datetime(df_Demand_AreaPlant["DateTime"]).dt.date
df_Demand_AreaPlant["Date"] = pd.to_datetime(df_Demand_AreaPlant["Date"])

# Drop the columns that have no value and Total_supply that is the same as Total_demand
df_Demand_AreaPlant = df_Demand_AreaPlant.drop(["DateTime", "PVCurtailment_CHG", "WindCurtailment_CHG",
                                                "TotalSupply_HOK", "TotalSupply_TKO", "TotalSupply_SHI", "TotalSupply_OKI"],  axis=1)
```

In [1894]:
```
# This container is only for getting time table data
#Actual generation_dataset from 2016-08-01
df_actual_generation_1 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/使うかわからない軍/Actual_generation_5min/Actual_generation1.csv', sep=',', header=0, encoding='shift_jis')
#Actual generation_dataset from 2020-04-14
df_actual_generation_2 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/使うかわからない軍/Actual_generation_5min/Actual_generation2.csv', sep=',', header=0, encoding='shift_jis')

df_actual_generation_Tohoku = pd.concat([df_actual_generation_1, df_actual_generation_2])
df_actual_generation_Tohoku = df_actual_generation_Tohoku.rename(columns={'DATE': 'Date', 'TIME': 'Time',
   '太陽光発電実績(5分間隔値)(万kW)': 'gen_Solar(mkW)', '当日実績(5分間隔値)(万kW)': 'gen_all(mkW)', '風力発電実績(5分間隔値)(万kW)': 'gen_Wind(mkW)'
})

df_actual_generation_Tohoku["DateTime"] = pd.to_datetime(df_actual_generation_Tohoku["Date"] + " " + df_actual_generation_Tohoku["Time"])
df_actual_generation_Tohoku = df_actual_generation_Tohoku.groupby(pd.Grouper(key="DateTime", freq='30min')).sum()

df_actual_generation_Tohoku.reset_index(inplace = True)

df_actual_generation_Tohoku["Time"] = pd.to_datetime(df_actual_generation_Tohoku["DateTime"]).dt.time
df_actual_generation_Tohoku["Date"] = pd.to_datetime(df_actual_generation_Tohoku["DateTime"]).dt.date
df_actual_generation_Tohoku["Date"] = pd.to_datetime(df_actual_generation_Tohoku["Date"])

df_actual_generation_Tohoku = df_actual_generation_Tohoku.drop("DateTime", axis=1)

# Adjust HH table ※原因はわからないが1から作るとワークしないので df_actual_generation_Tohokuから作成
HH_table = pd.DataFrame(df_intra_HH["HH"])
HH_table = HH_table.drop_duplicates()
time = df_actual_generation_Tohoku["Time"]
time = pd.DataFrame(time)
time = time.drop_duplicates()
HH_table["Time"] = time
```

In [1895]:
```
# data table for heatmap
intra_close = df_intra[["Date", "HH", "Close"]]
Demand_AreaPlant_withClose = pd.merge(df_Demand_AreaPlant, HH_table, how='left', on=["Time"])
Demand_AreaPlant_withClose = pd.merge(Demand_AreaPlant_withClose, intra_close, how='left', on=['Date', 'HH'])
Demand_AreaPlant_withClose["Close"] = Demand_AreaPlant_withClose["Close"].fillna(0)
```

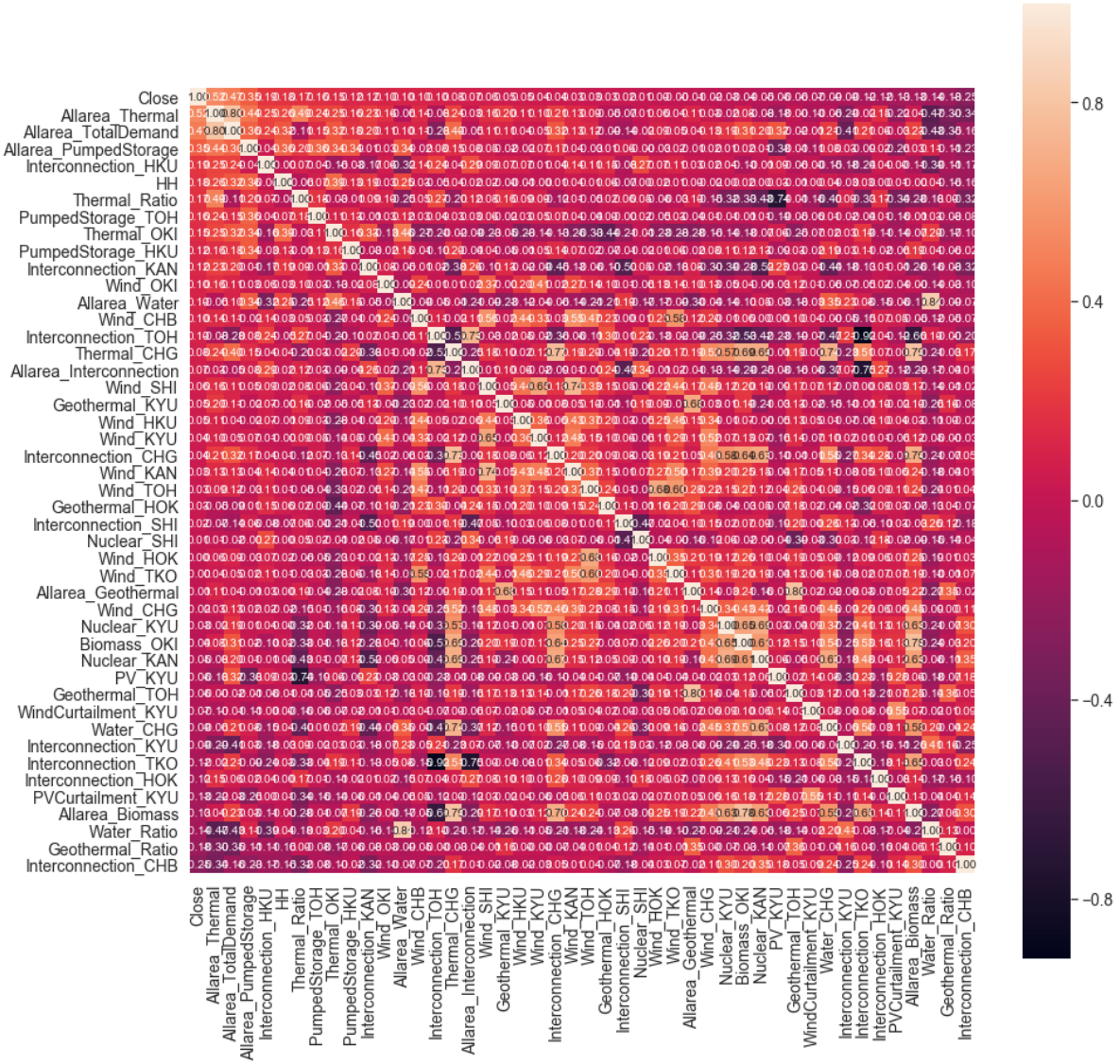
In [1896]:
```
# correlation matrix
corrmat = Demand_AreaPlant_withClose.corr()

# revenue correlation matrix
k = 30 # The number of variables on the heatmap
cols = corrmat.nlargest(k, 'Close')['Close'].index
cm = np.corrcoef(Demand_AreaPlant_withClose[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}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```



- Keep the features of thermal plant in all the area, and total information
- Remove the features that has high correlation with each others

In [1897]:
```
# Drop features that have high correlation with each other as much as possible
Demand_AreaPlant_withClose = Demand_AreaPlant_withClose.drop([
                                "TotalDemand_TOH", "TotalDemand_CHB", "TotalDemand_
CHG", "TotalDemand_SHI",
                                "TotalDemand_KYU", "TotalDemand_HKU", "TotalDemand_
TKO","TotalDemand_KAN","TotalDemand_OKI", "TotalDemand_HOK",
                                "Thermal_CHB", "Thermal_TKO", "Thermal_KAN", "Thermal
_TOH", "Thermal_KYU", "Thermal_HKU", "Thermal_SHI", "Thermal_HOK",
                                "PumpedStorage_TKO", "PumpedStorage_KAN", "PumpedS
torage_SHI", "PumpedStorage_KYU", "PumpedStorage_CHG",
                                "PumpedStorage_HOK", "PumpedStorage_CHB", "Water_K
AN", "Water_CHB", "Water_HOK", "Water_OKI", "Water_TKO",
                                "Water_HOK", "Water_KYU", "Water_TOH", "Water_SHI",
"Water_HKU", "PV_TKO", "PV_SHI", "PV_KAN", "PV_CHB",
                                "PV_TKO", "PV_HKU", "PV_HOK", "PV_TOH", "PV_CHG",
"Allarea_WindCurtailment", "Allarea_PV", "Allarea_PVCurtailment", "Biomass_CHG",
                                "Biomass_HKU", "Biomass_SHI", "Biomass_HOK", "Biomass
_TKO", "Biomass_KYU", "Biomass_HKU", "Biomass_TOH", "PV_OKI",
                                "Allarea_Nuclear", "Allarea_Wind"
                               ], axis=1)
```

In [1899]:
```
# correlation matrix
corrmat = Demand_AreaPlant_withClose.corr()

# revenue correlation matrix
k = 50 # The number of variables on the heatmap
cols = corrmat.nlargest(k, 'Close')['Close'].index
cm = np.corrcoef(Demand_AreaPlant_withClose[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}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```



Drop the above data at the end of "4 Make all_data" because some of them will be used to remove the outliers.

Depand peak for the next day

http://occtonet.occto.or.jp/public/dfw/RP11/OCCTO/SD/LOGIN_login#
(http://occtonet.occto.or.jp/public/dfw/RP11/OCCTO/SD/LOGIN_login#)

In [1900]:
```
# Daily demand peak
df_DemandPeak = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/
Master_thesis/DemandPeak_forecast.csv', sep=',', header=0, encoding='shift_jis')
```

In [1901]:
```
df_DemandPeak = df_DemandPeak.rename(columns={'策定日': 'PlanDate', '対象日付': 'D
ate', '対象エリア': 'Area',
                          '最小総需要予想時刻': 'Time_Min', '最小総需要予想(MW)': 'Planned_Mi
n(MW)',
                          '最大総需要予想時刻': 'Time_Max', '最大総需要予想(MW)': 'Planned_M
ax(MW)',
                          '最大供給力予想(MW)': 'Planned_Max_Capa(MW)', '予想使用率': 'Plan
ned_usege(%)', '予想予備率': 'Planned_margin(%)'})

df_DemandPeak["Date"] = pd.to_datetime(df_DemandPeak["Date"])

df_DemandPeak = df_DemandPeak.set_index("Date")
df_DemandPeak = df_DemandPeak.sort_index(ascending=True)
df_DemandPeak = df_DemandPeak.reset_index()
```

Make peak flags based on the information of the total of the 10 areas.

In [1902]:
```
df_DemandPeak_All = df_DemandPeak[df_DemandPeak["Area"] == "10エリア計"]
```

In [1903]:
```
df_DemandPeak_All.head()
```

Out[1903]:

	Date	PlanDate	Area	Time_Min	Planned_Min(MW)	Time_Max	Planned_Max(MW)	Planne
0	2016-04-01	2016/03/31	10エリア計	02:00	81777	19:00	109605	
23	2016-04-02	2016/04/01	10エリア計	24:00	82868	19:00	99047	
35	2016-04-03	2016/04/02	10エリア計	07:00	74764	20:00	94163	
47	2016-04-04	2016/04/03	10エリア計	02:00	74641	19:00	107828	
59	2016-04-05	2016/04/04	10エリア計	02:00	75602	19:00	107112	

Make max/min flag

In [1904]:
```
df_DemandPeak_All["Planned_Min_flag"] = 1
df_DemandPeak_All["Planned_Max_flag"] = 9
```

In [1906]:
```
# Make the min flag
df_DemandPeakMin_All = df_DemandPeak_All[["Date", "Time_Min", "Planned_Min_flag"]]
df_DemandPeakMin_All = df_DemandPeakMin_All.rename(columns={"Time_Min": "Time"})

df_DemandPeakMin_All["Time"] = df_DemandPeakMin_All["Time"].str.replace('24:00', '23:00')
df_DemandPeakMin_All["Date"] = pd.to_datetime(df_DemandPeakMin_All["Date"])
df_DemandPeakMin_All["Time"] = pd.to_datetime(df_DemandPeakMin_All["Time"]).dt.time

df_DemandPeakMin_All.head()
```

Out[1906]:

	Date	Time	Planned_Min_flag
0	2016-04-01	02:00:00	1
23	2016-04-02	23:00:00	1
35	2016-04-03	07:00:00	1
47	2016-04-04	02:00:00	1
59	2016-04-05	02:00:00	1

In [1907]:
```
# Make the max flag
df_DemandPeakMax_All = df_DemandPeak_All[["Date", "Time_Max", "Planned_Max_flag"]]
df_DemandPeakMax_All = df_DemandPeakMax_All.rename(columns={"Time_Max": "Time"})

df_DemandPeakMax_All["Date"] = pd.to_datetime(df_DemandPeakMax_All["Date"])
df_DemandPeakMax_All["Time"] = pd.to_datetime(df_DemandPeakMax_All["Time"]).dt.time

df_DemandPeakMax_All.head()
```

Out[1907]:

	Date	Time	Planned_Max_flag
0	2016-04-01	19:00:00	9
23	2016-04-02	19:00:00	9
35	2016-04-03	20:00:00	9
47	2016-04-04	19:00:00	9
59	2016-04-05	19:00:00	9

Done

Fit_actual_predicted (Not completed yet) ※ Add only if the electricity companies updated the data

Hokkaido: https://www.hepco.co.jp/network/con_service/supply_overview/genecapacity/ (https://www.hepco.co.jp/network/con_service/supply_overview/genecapacity/)

-->Only after 2020

Tohoku: https://nw.tohoku-epco.co.jp/consignment/fit/ (https://nw.tohoku-epco.co.jp/consignment/fit/)

--> They will update the data in the end of March 2021

Tokyo: https://www.tepco.co.jp/forecast/html/fit_data-j.html (https://www.tepco.co.jp/forecast/html/fit_data-j.html)

Chubu: https://powergrid.chuden.co.jp/takuso_service/hatsuden_kouri/takuso_kyokyu/rule/tokureihatsuden/ (https://powergrid.chuden.co.jp/takuso_service/hatsuden_kouri/takuso_kyokyu/rule/tokureihatsuden/)

Hokuriku: http://www.rikuden.co.jp/nw_jyukyudata/tokurei1_jisseki.html (http://www.rikuden.co.jp/nw_jyukyudata/tokurei1_jisseki.html)

Kansai(Can find only from google search): https://search.kansai-td.co.jp/? (https://search.kansai-td.co.jp/?)ie=u&page=1&kw=FIT+%E5%AE%9F%E7%B8%BE&ref=https%3A%2F%2Fsearch.kansai-td.co.jp%2F%3Fkw%3DFIT%25E7%2589%25B9%25E4%25BE%258B%25E2%2591%25A0%26ie%3Du&temp

Chugoku: https://www.energia.co.jp/nw/service/retailer/data/fit/ (https://www.energia.co.jp/nw/service/retailer/data/fit/)

Shikoku: https://www.yonden.co.jp/nw/renewable_energy/data/value.html (https://www.yonden.co.jp/nw/renewable_energy/data/value.html)

Kyushu: https://www.kyuden.co.jp/td_service_wheeling_outline_index.html (https://www.kyuden.co.jp/td_service_wheeling_outline_index.html)

Okinawa: https://www.okiden.co.jp/business-support/service/consignment/imbalance-achievement/index.html (https://www.okiden.co.jp/business-support/service/consignment/imbalance-achievement/index.html)

- Prediction data is published at 6 am on a day before the delivery date --> Can be used without lag https://www.occto.or.jp/oshirase/sonotaoshirase/2019/files/191220_FIT1unyohenko.pdf.pdf (https://www.occto.or.jp/oshirase/sonotaoshirase/2019/files/191220_FIT1unyohenko.pdf.pdf)
- The timing that the data is published was changed from April 2020 (two days before --> a day before) https://www.occto.or.jp/oshirase/sonotaoshirase/2019/files/191220_FIT1unyohenko.pdf.pdf (https://www.occto.or.jp/oshirase/sonotaoshirase/2019/files/191220_FIT1unyohenko.pdf.pdf)

In [ ]:

In [ ]:

Other data

- The following data are avoided as features since they seems very low correlation based on "8. EDA(for all_data)"
- If I have more time and the performance of the model should be improved more, they are considered.

Actual_generation(Tohoku_area) ※Skip

https://setsuden.nw.tohoku-epco.co.jp/download.html (https://setsuden.nw.tohoku-epco.co.jp/download.html)

In [1908]:
```
#Actual generation_dataset from 2016-08-01
df_actual_generation_1 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/使うかわからない軍/Actual_generation_5min/Actual_generation1.csv', sep=',', header=0, encoding='shift_jis')

#Actual generation_dataset from 2020-04-14
df_actual_generation_2 = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/使うかわからない軍/Actual_generation_5min/Actual_generation2.csv', sep=',', header=0, encoding='shift_jis')
```

In [1909]:
```
df_actual_generation_Tohoku = pd.concat([df_actual_generation_1, df_actual_generation_2])
df_actual_generation_Tohoku = df_actual_generation_Tohoku.rename(columns={'DATE': 'Date', 'TIME': 'Time',
    '太陽光発電実績(5分間隔値)(万kW)': 'gen_Solar(mkW)', '当日実績(5分間隔値)(万kW)': 'gen_all(mkW)', '風力発電実績(5分間隔値)(万kW)': 'gen_Wind(mkW)'
})

df_actual_generation_Tohoku["gen_Solar(mkW)"].fillna(0, inplace=True)
df_actual_generation_Tohoku["gen_all(mkW)"].fillna(0, inplace=True)
df_actual_generation_Tohoku["gen_Wind(mkW)"].fillna(0, inplace=True)

df_actual_generation_Tohoku["DateTime"] = pd.to_datetime(df_actual_generation_Tohoku["Date"] + " " + df_actual_generation_Tohoku["Time"])
df_actual_generation_Tohoku = df_actual_generation_Tohoku.groupby(pd.Grouper(key="DateTime", freq='30min')).sum()

df_actual_generation_Tohoku.reset_index(inplace = True)

df_actual_generation_Tohoku["Time"] = pd.to_datetime(df_actual_generation_Tohoku["DateTime"]).dt.time
df_actual_generation_Tohoku["Date"] = pd.to_datetime(df_actual_generation_Tohoku["DateTime"]).dt.date
df_actual_generation_Tohoku["Date"] = pd.to_datetime(df_actual_generation_Tohoku["Date"])

df_actual_generation_Tohoku = df_actual_generation_Tohoku.drop("DateTime", axis=1)
```

Weather data (Tohoku area) ※Skip

https://www.data.jma.go.jp/gmd/risk/obsdl/index.php (https://www.data.jma.go.jp/gmd/risk/obsdl/index.php)

- Weather data in areas where the electricity companies exist will be added if the prediction performance is not improved sufficiently.
- Please note: The correlation weather data and intraday close price seems to be very low.

In [1562]:
```
# #Read all the weather data in Tohoku area

# df_weather_Aomori = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/Weather/Weather_Aomori.csv', sep=',', header=0, encoding='cp932')
# df_weather_Akita = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/Weather/Weather_Akita.csv', sep=',', header=0, encoding='cp932')
# df_weather_Morioka = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/Weather/Weather_Morioka.csv', sep=',', header=0, encoding='cp932')
# df_weather_Yamagata = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/Weather/Weather_Yamagata.csv', sep=',', header=0, encoding='cp932')
# df_weather_Sendai = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/Weather/Weather_Sendai.csv', sep=',', header=0, encoding='cp932')
# df_weather_Fukushima = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/Weather/Weather_Fukushima.csv', sep=',', header=0, encoding='cp932')
# df_weather_Niigata = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/Weather/Weather_Niigata.csv', sep=',', header=0, encoding='cp932')
```

In [1563]:
```
# # marge all the spot datasets
# df_weather_all = pd.merge(df_weather_Aomori, df_weather_Akita, how='left', on=['Date'])
# df_weather_all = pd.merge(df_weather_all, df_weather_Morioka, how='left', on=['Date'])
# df_weather_all = pd.merge(df_weather_all, df_weather_Yamagata, how='left', on=['Date'])
# df_weather_all = pd.merge(df_weather_all, df_weather_Sendai, how='left', on=['Date'])
# df_weather_all = pd.merge(df_weather_all, df_weather_Fukushima, how='left', on=['Date'])
# df_weather_all = pd.merge(df_weather_all, df_weather_Niigata, how='left', on=['Date'])
# print(df_weather_all.shape)
```

In [1564]:
```
# #Drop the columns that are not necessary
# a = [item for item in df_weather_all.columns if item.find('Qual') != -1 or item.find('Num') != -1or item.find('None') != -1]
# df_weather_all = df_weather_all.drop(columns=a)

# # Replace/adjust the name of direction
# df_weather_all = df_weather_all.replace({'北西)': '北西', '南東)': '南東', '東南東)': '東南東', '南)': '南'})

# # Categolize direction
# df_weather_all = df_weather_all.replace({'静穏': '0', '北': '1', '北北東': '2', '北東': '3',
#                                          '東北東': '4', '東': '5', '東南東': '6', '南東': '7', '南南東': '8',
#                                          '南': '9', '南南西': '10', '南西': '11', '西南西': '12',
#                                          '西': '13', '西北西': '14', '北西': '15', '北北西': '16'})

# df_weather_all["Time"] = pd.to_datetime(df_weather_all["Date"]).dt.time
# df_weather_all["Date"] = pd.to_datetime(df_weather_all["Date"]).dt.date
# df_weather_all["Date"] = pd.to_datetime(df_weather_all["Date"])

# cols = [item for item in df_weather_all.columns if item.find('WindDirection') != -1]
# df_weather_all[cols] = df_weather_all[cols].apply(pd.to_numeric)
```

In [1565]:
```
# df_Temp = df_weather_all.iloc[:, df_weather_all.columns.str.contains("Temp")]
# df_SunLight_Time = df_weather_all.iloc[:, df_weather_all.columns.str.contains("SunLight" and "Time")].drop("Time", axis=1)
# df_SunLight_Volume = df_weather_all.iloc[:, df_weather_all.columns.str.contains("SunLight" and "MJ/㎡")]
# df_WindSpeed = df_weather_all.iloc[:, df_weather_all.columns.str.contains("WindSpeed")]
# df_WindDirection = df_weather_all.iloc[:, df_weather_all.columns.str.contains("WindDirection")]
```

In [1566]:
```
# # correlation matrix
# corrmat = df_SunLight_Time.corr()
# f, ax = plt.subplots(figsize=(12,9))
# sns.heatmap(corrmat, square=True);
```

LNG price

Only daily data of LNG price exists.

In [1910]:
```
LNGPrice = pd.read_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/LNG_historical-data.csv', sep=',', header=0)
LNGPrice.columns = ["Year", "Month", "Status", "ContractPrice(USD/MMBtu)", "Arrival Price(USD/MMBtu)", "Unnamed: 5"]
LNGPrice["Date"] = pd.to_datetime(LNGPrice["Year"].astype(str) + "-" + LNGPrice["Month"].astype(str), format='%Y-%m')
LNGPrice = LNGPrice.drop(["Year", "Month", "Unnamed: 5"], axis=1)
LNGPrice = LNGPrice.replace({'確報': 'Confirmed', '速報': 'Quick', '×': '0'})
LNGPrice["ContractPrice(USD/MMBtu)"] = LNGPrice["ContractPrice(USD/MMBtu)"].astype(float)
LNGPrice["Arrival Price(USD/MMBtu)"] = LNGPrice["Arrival Price(USD/MMBtu)"].astype(float)
LNGPrice.tail()
```

Out[1910]:

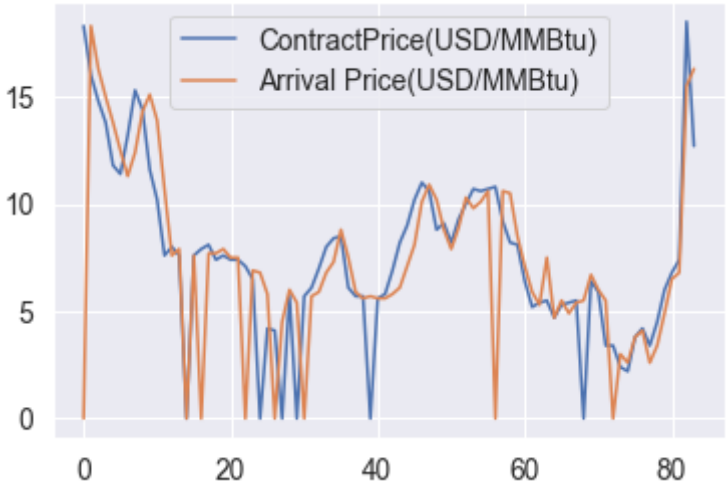
	Status	ContractPrice(USD/MMBtu)	Arrival Price(USD/MMBtu)	Date
79	Confirmed	6.0	4.9	2020-10-01
80	Confirmed	6.8	6.5	2020-11-01
81	Confirmed	7.4	6.8	2020-12-01
82	Confirmed	18.5	15.5	2021-01-01
83	Quick	12.7	16.3	2021-02-01

In [1919]:
```
close = df_intra[["Date", "Close"]]
close = pd.merge(close, LNGPrice, how="left", on="Date")
close = close.drop(["Status"], axis=1)
close = close.fillna(0)
close.head()
```

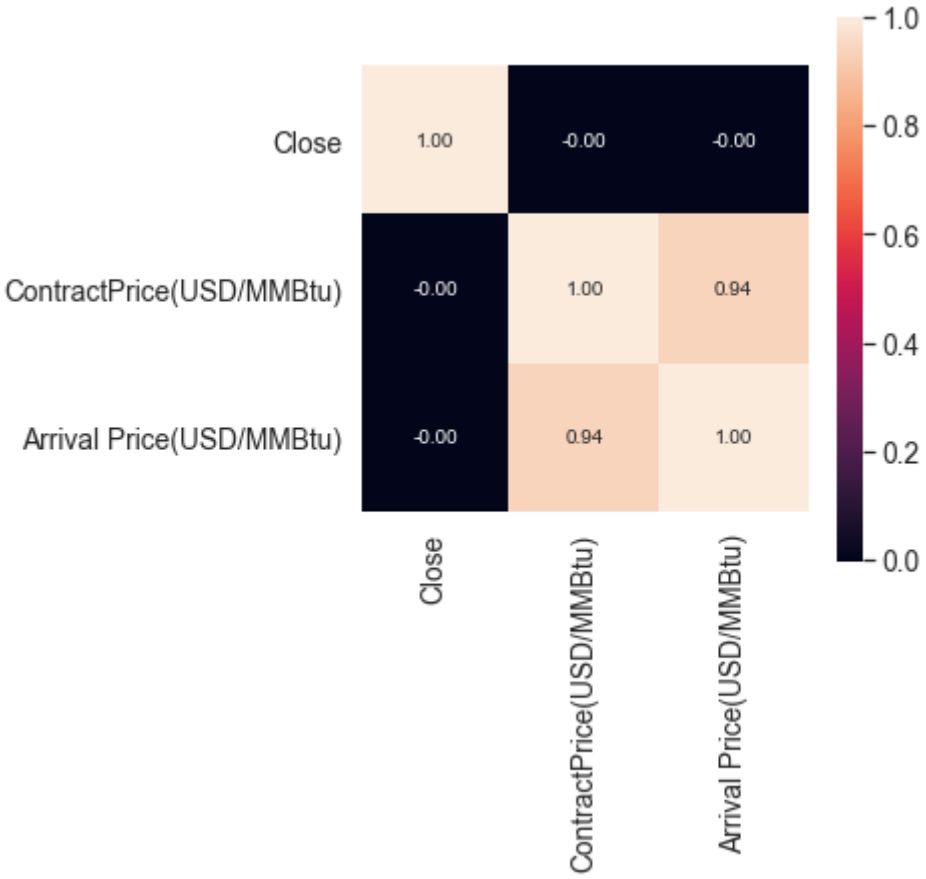
Out[1919]:

	Date	Close	ContractPrice(USD/MMBtu)	Arrival Price(USD/MMBtu)
0	2016-04-01	7.69	4.2	5.8
1	2016-04-01	0.00	4.2	5.8
2	2016-04-01	7.21	4.2	5.8
3	2016-04-01	7.06	4.2	5.8
4	2016-04-01	7.21	4.2	5.8

In [1913]:
```
#LNGPrice = LNGPrice.set_index("Date")
LNGPrice["ContractPrice(USD/MMBtu)"].plot()
LNGPrice["Arrival Price(USD/MMBtu)"].plot()
plt.legend();
```



In [1921]:
```
# Close price correlation matrix
corrmat = close.corr()

k = 3 # The number of variables on the heatmap
cols = corrmat.nlargest(k, 'Close')['Close'].index
cm = np.corrcoef(close[cols].values.T)
sns.set(font_scale=1.25)
f, ax = plt.subplots(figsize=(5,5))
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```



These looks not meaningful to be added.

Make all_data

Merge all input data

In [1922]:
```
# Adjust HH table ※ Make from df_actual_generation_Tohoku
HH_table = pd.DataFrame(df_intra_HH["HH"])
HH_table = HH_table.drop_duplicates()
time = df_actual_generation_Tohoku["Time"]
time = pd.DataFrame(time)
time = time.drop_duplicates()
HH_table["Time"] = time
```

In [1923]:
```
all_data = df_intra.copy()

# HH_table
all_data = pd.merge(all_data, HH_table, how="left", on=['HH'])

# df_spot
all_data = pd.merge(all_data, df_spot, how="left", on=['Date', 'HH'])

#df_HourlyDemand_All
all_data = pd.merge(all_data, df_HourlyDemand_All, how="left", on=['Date', 'Time'])

# df_Demand_AreaPlant
all_data = pd.merge(all_data, df_Demand_AreaPlant, how="left", on=['Date', 'Time'])

#df_DemandPeakMin_All
all_data = pd.merge(all_data, df_DemandPeakMin_All, how="left", on=['Date', 'Time'])

#df_DemandPeakMax_All
all_data = pd.merge(all_data, df_DemandPeakMax_All, how="left", on=['Date', 'Time'])

# #df_weather_all
# all_data = pd.merge(all_data, df_weather_all, how="left", on=['Date','Time'])

# #df_actual_generation_Tohoku
# all_data = pd.merge(all_data, df_actual_generation_Tohoku, how="left", on=['Date',
 'Time'])

all_data.head()
```

Out[1923]:

	Date	HH	Open	High	Low	Close	Average	Volume(MWh/h)	Volume(Tick count)	Time	Syster
0	2016-04-01	1	7.69	7.69	7.69	7.69	7.69	0.7	1	00:00:00	
1	2016-04-01	2	NaN	NaN	NaN	NaN	NaN	NaN	0	00:30:00	
2	2016-04-01	3	7.21	7.21	7.21	7.21	7.21	0.8	1	01:00:00	
3	2016-04-01	4	7.06	7.06	7.06	7.06	7.06	0.8	1	01:30:00	
4	2016-04-01	5	7.21	7.21	7.21	7.21	7.21	0.8	1	02:00:00	

In [1924]:
```
all_data.shape
```

Out[1924]: (84281, 126)

In [1925]:
```
all_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 84281 entries, 0 to 84280
Columns: 126 entries, Date to Planned_Max_flag
dtypes: datetime64[ns](1), float64(121), int64(3), object(1)
memory usage: 81.7+ MB
```

In [1926]:
```
print(all_data.isnull().sum())
```

Date	0
HH	0
Open	72
High	72
Low	72
Close	72
Average	72
Volume(MWh/h)	72
Volume(Tick count)	0
Time	0
System_price(Yen/kWh)	0
Price_Hokkaido(Yen/kWh)	960
Price_Tokyo(Yen/kWh)	0
Price_Kansai(Yen/kWh)	0
BidExceed_diff	0
HourlyDemand_Hokkaido	42161
HourlyDemand_Tohoku	42161
HourlyDemand_Tokyo	42161
HourlyDemand_Chubu	42161
HourlyDemand_Hokuriku	42161
HourlyDemand_Kansai	42161
HourlyDemand_Chugoku	42161
HourlyDemand_Shikoku	42161
HourlyDemand_Kyushu	42161
HourlyDemand_Okinawa	42161
TotalDemand_TOH	42617
Water_TOH	42617
Thermal_TOH	42617
PV_TOH	42617
Wind_TOH	42617
Geothermal_TOH	42617
Biomass_TOH	42617
PumpedStorage_TOH	42617
Interconnection_TOH	42617
TotalDemand_HOK	42617
Thermal_HOK	42617
Water_HOK	42617
Geothermal_HOK	42617
Biomass_HOK	42617
PV_HOK	42617
Wind_HOK	42617
PumpedStorage_HOK	42617
Interconnection_HOK	42617
TotalDemand_TKO	42617
Thermal_TKO	42617
Water_TKO	42617
Biomass_TKO	42617
PV_TKO	42617
Wind_TKO	42617
PumpedStorage_TKO	42617
Interconnection_TKO	42617
TotalDemand_CHB	42617
Thermal_CHB	42617
Water_CHB	42617
PV_CHB	42617
Wind_CHB	42617
PumpedStorage_CHB	42617

Interconnection_CHB 42617
TotalDemand_HKU 42617
Thermal_HKU 42617
Water_HKU 42617
Biomass_HKU 42617
PV_HKU 42617
Wind_HKU 42617
PumpedStorage_HKU 42617
Interconnection_HKU 42617
TotalDemand_KAN 42617
Nuclear_KAN 42617
Thermal_KAN 42617
Water_KAN 42617
PV_KAN 42617
Wind_KAN 42617
PumpedStorage_KAN 42617
Interconnection_KAN 42617
TotalDemand_CHG 42617
Thermal_CHG 42617
Water_CHG 42617
Biomass_CHG 42617
PV_CHG 42617
Wind_CHG 42617
PumpedStorage_CHG 42617
Interconnection_CHG 42617
TotalDemand_SHI 42617
Nuclear_SHI 42617
Thermal_SHI 42617
Water_SHI 42617
Biomass_SHI 42617
PV_SHI 42617
Wind_SHI 42617
PumpedStorage_SHI 42617
Interconnection_SHI 42617
TotalDemand_KYU 42617
Nuclear_KYU 42617
Thermal_KYU 42617
Water_KYU 42617
Geothermal_KYU 42617
Biomass_KYU 42617
PV_KYU 42617
PVCurtailment_KYU 42617
Wind_KYU 42617
WindCurtailment_KYU 42617
PumpedStorage_KYU 42617
Interconnection_KYU 42617
TotalDemand_OKI 42617
Thermal_OKI 42617
Water_OKI 42617
Biomass_OKI 42617
PV_OKI 42617
Wind_OKI 42617
Allarea_TotalDemand 42617
Allarea_Water 42617
Allarea_Thermal 42617
Allarea_Nuclear 42617
Allarea_PV 42617

```
Allarea_PVCurtailment     42617
Allarea_Wind              42617
Allarea_WindCurtailment   42617
Allarea_Geothermal        42617
Allarea_Biomass           42617
Allarea_PumpedStorage     42617
Allarea_Interconnection   42617
Water_Ratio               42617
Thermal_Ratio             42617
Geothermal_Ratio          42617
Planned_Min_flag          82525
Planned_Max_flag          82525
dtype: int64
```

In [1927]:
```
# Fillna with 0 and flag
all_data["Planned_Min_flag"].fillna(0, inplace=True)
all_data["Planned_Max_flag"].fillna(0, inplace=True)
```

In [1928]:
```
# Fillna with the average between the previous and later slots for Price, Demand
all_data.interpolate(method='linear', inplace=True)
```

In [1929]:
```
print(all_data.isnull().sum())
```

```
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_Tohoku         0
HourlyDemand_Tokyo          0
HourlyDemand_Chubu          0
HourlyDemand_Hokuriku       0
HourlyDemand_Kansai         0
HourlyDemand_Chugoku        0
HourlyDemand_Shikoku        0
HourlyDemand_Kyushu         0
HourlyDemand_Okinawa        0
TotalDemand_TOH             0
Water_TOH                   0
Thermal_TOH                 0
PV_TOH                      0
Wind_TOH                    0
Geothermal_TOH              0
Biomass_TOH                 0
PumpedStorage_TOH           0
Interconnection_TOH         0
TotalDemand_HOK             0
Thermal_HOK                 0
Water_HOK                   0
Geothermal_HOK              0
Biomass_HOK                 0
PV_HOK                      0
Wind_HOK                    0
PumpedStorage_HOK           0
Interconnection_HOK         0
TotalDemand_TKO             0
Thermal_TKO                 0
Water_TKO                   0
Biomass_TKO                 0
PV_TKO                      0
Wind_TKO                    0
PumpedStorage_TKO           0
Interconnection_TKO         0
TotalDemand_CHB             0
Thermal_CHB                 0
Water_CHB                   0
PV_CHB                      0
Wind_CHB                    0
PumpedStorage_CHB           0
```

```
Interconnection_CHB       0
TotalDemand_HKU           0
Thermal_HKU               0
Water_HKU                 0
Biomass_HKU               0
PV_HKU                    0
Wind_HKU                  0
PumpedStorage_HKU         0
Interconnection_HKU       0
TotalDemand_KAN           0
Nuclear_KAN               0
Thermal_KAN               0
Water_KAN                 0
PV_KAN                    0
Wind_KAN                  0
PumpedStorage_KAN         0
Interconnection_KAN       0
TotalDemand_CHG           0
Thermal_CHG               0
Water_CHG                 0
Biomass_CHG               0
PV_CHG                    0
Wind_CHG                  0
PumpedStorage_CHG         0
Interconnection_CHG       0
TotalDemand_SHI           0
Nuclear_SHI               0
Thermal_SHI               0
Water_SHI                 0
Biomass_SHI               0
PV_SHI                    0
Wind_SHI                  0
PumpedStorage_SHI         0
Interconnection_SHI       0
TotalDemand_KYU           0
Nuclear_KYU               0
Thermal_KYU               0
Water_KYU                 0
Geothermal_KYU            0
Biomass_KYU               0
PV_KYU                    0
PVCurtailment_KYU         0
Wind_KYU                  0
WindCurtailment_KYU       0
PumpedStorage_KYU         0
Interconnection_KYU       0
TotalDemand_OKI           0
Thermal_OKI               0
Water_OKI                 0
Biomass_OKI               0
PV_OKI                    0
Wind_OKI                  0
Allarea_TotalDemand       0
Allarea_Water             0
Allarea_Thermal           0
Allarea_Nuclear           0
Allarea_PV                0
```

```
Allarea_PVCurtailment      0
Allarea_Wind               0
Allarea_WindCurtailment    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
dtype: int64
```

Adjust all_data (Add, remove outlier and drop features that have high correlation)

- Filter all_data from 2016-04-01 to 2020-12-31

In [1930]:
```
all_data = all_data[all_data["Date"] <= "2020-12-31"]
```

- Add date information

In [1931]:
```
# Create a column for "date block num"
all_data['Date'] = pd.to_datetime(all_data['Date'])
# all_data["date_block_num"] = np.trunc(all_data['Date'].map(pd.Timestamp.timesta
mp).astype(int) / 86400 - 16891)
```

In [1932]:
```
all_data['month'] = pd.to_datetime(all_data["Date"]).dt.month
```

In [1933]:
```
# From Monday:0 to Sunday:6
all_data['dayofweek'] = pd.to_datetime(all_data["Date"]).dt.dayofweek
```

In [1934]:
```
# Check the national holiday in Japan
len(jpholiday.between(datetime.date(2016, 4, 1), datetime.date(2020, 12, 31)))
```

Out[1934]: 89

In [1935]:
```
# Add the national holidays in all_data
import jpholiday

Date_list = list(all_data["Date"])
DayofWeek_list = list(all_data["dayofweek"])
holiday_judge = []

for d, dw in zip(Date_list, DayofWeek_list):
    # For holiday
    if jpholiday.is_holiday(datetime.date(d.year, d.month, d.day)) == "True":
        holiday_judge.append(1)
    # For Sunday and Satarday
    elif dw >= 5:
        holiday_judge.append(1)
    else:
        holiday_judge.append(0)

all_data["holiday"] = pd.Series(holiday_judge)
```

- Treat the outlier

In [1936]:
```
all_data[all_data["Close"] > 120]
```

Out[1936]:

	Date	HH	Open	High	Low	Close	Average	Volume(MWh/h)	Volume(Tick count)	Time
76977	2020-08-21	34	53.96	160.0	45.01	150.0	93.48	561.4	106	16:30:00
76978	2020-08-21	35	32.07	135.0	27.89	135.0	74.70	449.2	92	17:00:00
76979	2020-08-21	36	31.97	135.0	27.00	135.0	64.57	480.0	86	17:30:00

In [1937]:
```
all_data[all_data["TotalDemand_HOK"] < 2200]
```

Out[1937]:

	Date	HH	Open	High	Low	Close	Average	Volume(MWh/h)	Volume(Tick count)	Time
42629	2018-09-06	6	10.37	16.10	2.80	6.16	10.71	147.4	86	02:30:00
42630	2018-09-06	7	7.20	16.30	2.90	9.68	10.88	144.4	77	03:00:00
42631	2018-09-06	8	10.35	16.30	2.90	7.08	10.67	151.1	80	03:30:00
42632	2018-09-06	9	10.30	12.65	3.10	6.52	8.80	257.2	81	04:00:00
42633	2018-09-06	10	10.33	12.73	3.10	11.22	8.09	335.3	79	04:30:00
42634	2018-09-06	11	10.83	13.16	3.20	8.68	8.08	334.0	73	05:00:00
42635	2018-09-06	12	10.35	13.16	3.40	8.80	10.91	142.8	78	05:30:00
42636	2018-09-06	13	10.95	13.20	3.40	8.75	10.50	160.2	77	06:00:00
42637	2018-09-06	14	10.30	13.16	4.50	9.83	10.88	136.9	73	06:30:00
42638	2018-09-06	15	10.09	12.59	4.50	8.30	10.53	132.9	84	07:00:00
42639	2018-09-06	16	10.39	11.78	4.50	8.65	8.20	36.8	78	07:30:00
42640	2018-09-06	17	7.12	20.01	5.32	7.32	7.77	41.4	75	08:00:00
42641	2018-09-06	18	11.97	15.50	6.43	6.51	7.89	220.7	90	08:30:00
42642	2018-09-06	19	10.49	20.00	6.51	10.55	8.67	189.5	101	09:00:00
42643	2018-09-06	20	10.84	15.50	6.18	10.94	9.86	286.4	110	09:30:00
42644	2018-09-06	21	10.80	13.58	6.51	8.90	8.11	286.2	110	10:00:00
42645	2018-09-06	22	11.00	13.46	6.22	12.50	7.99	357.3	96	10:30:00
42646	2018-09-06	23	11.59	13.46	6.22	13.23	8.68	518.0	123	11:00:00
42647	2018-09-06	24	11.30	14.28	6.24	14.28	8.88	422.6	133	11:30:00
42648	2018-09-06	25	10.90	13.68	3.50	13.68	8.65	263.6	154	12:00:00
42649	2018-09-06	26	10.88	13.79	6.25	7.75	8.66	197.0	126	12:30:00
42650	2018-09-06	27	12.67	17.13	6.80	11.40	9.36	504.5	169	13:00:00
42651	2018-09-06	28	13.66	17.93	6.51	12.51	10.11	441.7	167	13:30:00

	Date	HH	Open	High	Low	Close	Average	Volume(MWh/h)	Volume(Tick count)	Time
42652	2018-09-06	29	12.62	19.70	7.62	12.62	12.79	634.1	177	14:00:00
42653	2018-09-06	30	18.00	25.10	8.15	15.38	14.39	694.8	196	14:30:00
42654	2018-09-06	31	19.07	20.00	8.15	12.37	13.40	294.7	137	15:00:00
42655	2018-09-06	32	15.00	23.04	10.10	13.00	17.71	484.7	183	15:30:00
42656	2018-09-06	33	20.00	32.88	8.69	18.00	19.80	1301.9	266	16:00:00
42657	2018-09-06	34	30.00	32.88	11.72	11.72	21.34	1132.0	260	16:30:00
42658	2018-09-06	35	24.00	24.24	9.77	16.56	17.12	626.4	195	17:00:00
42659	2018-09-06	36	20.00	23.04	8.17	15.00	14.96	498.9	168	17:30:00
42660	2018-09-06	37	22.00	23.04	10.10	16.56	15.74	540.8	169	18:00:00
42661	2018-09-06	38	21.02	21.02	9.16	13.82	14.43	303.4	154	18:30:00
42662	2018-09-06	39	15.21	20.01	8.97	13.21	13.74	155.8	126	19:00:00
42663	2018-09-06	40	14.90	20.01	10.63	11.07	12.70	117.0	123	19:30:00
42664	2018-09-06	41	14.50	20.01	9.41	10.91	10.73	360.9	142	20:00:00
42665	2018-09-06	42	15.87	20.00	8.47	12.75	10.27	337.7	140	20:30:00
42666	2018-09-06	43	11.55	20.01	6.52	8.16	9.34	603.1	118	21:00:00
42667	2018-09-06	44	11.23	20.00	6.52	6.74	8.28	514.3	134	21:30:00
42668	2018-09-06	45	13.47	20.00	6.27	6.95	8.85	375.8	130	22:00:00
42669	2018-09-06	46	12.67	20.00	6.52	6.88	8.88	332.8	116	22:30:00
42670	2018-09-06	47	11.23	20.00	6.36	6.80	8.52	423.9	129	23:00:00
42671	2018-09-06	48	10.82	17.00	4.40	7.02	7.03	427.7	118	23:30:00
42672	2018-09-07	1	10.73	10.73	3.50	9.21	6.47	334.7	77	00:00:00
42673	2018-09-07	2	7.11	10.45	3.10	6.15	6.70	282.4	86	00:30:00
42674	2018-09-07	3	10.39	10.39	3.10	8.05	6.38	310.8	83	01:00:00

	Date	HH	Open	High	Low	Close	Average	Volume(MWh/h)	Volume(Tick count)	Time
42675	2018-09-07	4	10.29	10.29	3.00	8.03	6.39	321.3	82	01:30:00
42676	2018-09-07	5	10.26	10.26	3.00	8.03	6.37	269.0	79	02:00:00
42677	2018-09-07	6	6.94	10.17	3.00	6.18	6.35	301.5	77	02:30:00
42678	2018-09-07	7	7.11	9.74	3.10	8.44	7.02	412.8	85	03:00:00
42679	2018-09-07	8	7.13	9.60	3.20	8.12	6.95	413.0	82	03:30:00
42680	2018-09-07	9	7.11	9.63	3.10	8.34	7.02	481.3	89	04:00:00
42681	2018-09-07	10	7.12	9.74	3.10	8.72	6.34	321.6	75	04:30:00
42682	2018-09-07	11	7.18	9.67	3.20	8.97	6.37	360.6	69	05:00:00
42683	2018-09-07	12	3.20	9.85	3.20	8.64	6.39	354.4	75	05:30:00
42684	2018-09-07	13	7.50	9.73	3.50	8.82	6.39	418.5	86	06:00:00
42685	2018-09-07	14	13.53	13.53	3.80	6.19	6.30	535.2	94	06:30:00
42686	2018-09-07	15	10.14	10.14	3.70	8.69	6.47	386.8	95	07:00:00
42687	2018-09-07	16	10.72	10.72	4.50	6.17	6.84	246.9	96	07:30:00
42688	2018-09-07	17	10.94	10.94	4.50	8.46	6.89	638.0	115	08:00:00
42689	2018-09-07	18	15.21	15.21	9.42	11.43	11.07	306.1	102	08:30:00
42690	2018-09-07	19	17.28	18.28	8.47	12.32	12.10	481.9	124	09:00:00
42691	2018-09-07	20	18.58	18.58	6.52	13.22	11.42	699.0	134	09:30:00
42692	2018-09-07	21	18.04	18.04	8.28	16.14	11.98	658.3	138	10:00:00
42693	2018-09-07	22	18.27	18.27	8.15	13.82	12.17	985.7	152	10:30:00
42694	2018-09-07	23	19.63	19.63	8.47	13.99	12.98	772.7	135	11:00:00
42695	2018-09-07	24	18.92	19.42	8.47	16.51	13.11	867.8	162	11:30:00
42696	2018-09-07	25	17.80	18.37	7.72	14.71	10.99	750.3	160	12:00:00
42697	2018-09-07	26	11.82	18.37	6.52	15.21	10.56	768.3	159	12:30:00

	Date	HH	Open	High	Low	Close	Average	Volume(MWh/h)	Volume(Tick count)	Time
42698	2018-09-07	27	18.97	18.97	8.19	17.45	12.46	865.5	163	13:00:00
42699	2018-09-07	28	19.72	19.72	8.47	14.03	13.06	556.8	221	13:30:00
42700	2018-09-07	29	23.38	23.38	9.08	15.10	16.68	501.1	159	14:00:00
42701	2018-09-07	30	23.76	23.76	10.35	23.76	17.27	468.1	157	14:30:00
42702	2018-09-07	31	23.92	23.92	8.90	22.49	15.16	583.3	176	15:00:00
42703	2018-09-07	32	25.39	25.39	6.18	13.82	16.42	743.4	172	15:30:00
42704	2018-09-07	33	27.73	27.80	7.53	14.94	18.14	1301.6	237	16:00:00
42705	2018-09-07	34	30.08	30.15	7.31	15.00	19.20	1022.2	215	16:30:00
42706	2018-09-07	35	28.91	28.91	9.10	11.22	13.95	843.0	167	17:00:00
42707	2018-09-07	36	11.91	15.21	8.53	8.53	10.88	814.0	154	17:30:00
42708	2018-09-07	37	11.91	25.10	6.60	20.31	12.74	627.7	144	18:00:00
42709	2018-09-07	38	26.40	26.40	8.28	11.70	11.31	550.4	129	18:30:00
42710	2018-09-07	39	14.56	18.36	6.51	12.19	9.02	428.1	118	19:00:00
42768	2018-09-09	1	6.98	10.26	6.19	6.19	7.96	44.9	64	00:00:00
42769	2018-09-09	2	10.11	10.11	2.70	6.15	6.84	93.1	63	00:30:00
42770	2018-09-09	3	9.13	9.72	4.97	7.53	6.46	161.9	59	01:00:00
42816	2018-09-10	1	2.10	8.39	2.10	8.29	6.64	76.9	68	00:00:00
42817	2018-09-10	2	1.60	8.29	1.60	8.29	6.21	157.0	62	00:30:00
42818	2018-09-10	3	1.10	8.19	1.10	7.02	6.50	135.7	78	01:00:00

- These price spikes cannot be reasonable based on the features prepared. -->Remove the rows
- The black out in Hokkaido area on Sep 2018 -->Remove the rows
  -->(Reference) https://www.denkishimbun.com/sp/33180 (https://www.denkishimbun.com/sp/33180)

In [1938]:
```
# Remove the price spikes
all_data = all_data[all_data["Close"] < 120]
# Remove the black out in Hokkaido area
all_data = all_data[all_data["TotalDemand_HOK"] > 2200]
all_data = all_data.reset_index(drop=True)
```

- Remove the features that have high correlation with each other

In [1939]:
```
# from df_HourlyDemand_All  HourlyDemand
all_data = all_data.drop(['HourlyDemand_Tohoku',  'HourlyDemand_Chubu', 'HourlyDemand_Chugoku',
                          'HourlyDemand_Shikoku', 'HourlyDemand_Kyushu', 'HourlyDemand_Hokuriku'], axis=1)
```

In [1940]:
```
# Drop features that have high correlation with each other as much as possible
all_data = all_data.drop([
                          "TotalDemand_TOH", "TotalDemand_CHB", "TotalDemand_CHG", "TotalDemand_SHI",
                          "TotalDemand_KYU", "TotalDemand_HKU", "TotalDemand_TKO","TotalDemand_KAN","TotalDemand_OKI", "TotalDemand_HOK",
                          "Thermal_CHB", "Thermal_TKO", "Thermal_KAN", "Thermal_TOH", "Thermal_KYU", "Thermal_HKU", "Thermal_SHI", "Thermal_HOK",
                          "PumpedStorage_TKO", "PumpedStorage_KAN", "PumpedStorage_SHI", "PumpedStorage_KYU", "PumpedStorage_CHG",
                          "PumpedStorage_HOK", "PumpedStorage_CHB", "Water_KAN", "Water_CHB", "Water_HOK", "Water_OKI", "Water_TKO",
                          "Water_HOK", "Water_KYU", "Water_TOH", "Water_SHI", "Water_HKU", "PV_TKO", "PV_SHI", "PV_KAN", "PV_CHB",
                          "PV_TKO", "PV_HKU", "PV_HOK", "PV_TOH", "PV_CHG", "Allarea_WindCurtailment", "Allarea_PV", "Allarea_PVCurtailment", "Biomass_CHG",
                          "Biomass_HKU", "Biomass_SHI", "Biomass_HOK", "Biomass_TKO", "Biomass_KYU", "Biomass_HKU", "Biomass_TOH", "PV_OKI",
                          "Allarea_Nuclear", "Allarea_Wind"
                          ], axis=1)
```

In [1945]:
```
all_data.tail()
```

Out[1945]:

	Date	HH	Open	High	Low	Close	Average	Volume(MWh/h)	Volume(Tick count)	Time	S
83232	2020-12-31	44	35.0	70.0	33.00	70.00	44.71	328.7	110	21:30:00	
83233	2020-12-31	45	42.0	70.0	41.01	45.48	50.19	409.9	104	22:00:00	
83234	2020-12-31	46	42.0	70.0	35.00	41.33	42.20	383.2	112	22:30:00	
83235	2020-12-31	47	37.0	70.0	33.93	36.66	44.09	345.9	60	23:00:00	
83236	2020-12-31	48	27.0	37.5	23.93	26.46	28.42	233.3	66	23:30:00	

In [1942]:
```
all_data.shape
```

Out[1942]: (83237, 68)

In [1943]:
```
all_data.describe()
```

Out[1943]:

	HH	Open	High	Low	Close	Average	Vol
count	83237.000000	83237.000000	83237.000000	83237.000000	83237.000000	83237.000000	
mean	24.502060	9.934688	13.269915	6.363957	8.627951	8.769484	
std	13.854119	19.374277	20.435942	3.140658	4.599437	4.810060	
min	1.000000	0.010000	1.000000	0.010000	0.010000	0.010000	
25%	13.000000	6.740000	8.890000	4.700000	6.120000	6.530000	
50%	25.000000	8.670000	11.040000	5.980000	7.930000	7.950000	
75%	37.000000	11.000000	14.370000	7.630000	10.000000	9.810000	
max	48.000000	801.000000	801.000000	75.200000	105.500000	194.650000	

Save the all_data as a csv file

In [1944]:
```
all_data.to_csv('/Users/kenotsu/Documents/master_thesis/Datasets/Master_thesis/all_data.csv', index=False)
```