Table of Contents

- 1_Read & interpret dataset
  - 1.1_JEPX_Intra_datasets (Target: "Close price")
    - 1.1.1_Basic characteristics
    - 1.1.2_Stationarity
    - 1.1.3_Seasonality
    - 1.1.4_Autocorrelation
    - 1.1.5_Trading vollume from 2014 to 2019
  - 1.2_JEPX_Spot_datasets
    - 1.2.1_The relation between spot and close price
    - 1.2.2_Bidding balance
    - 1.2.3_The correlation between Area price in different area and "Close" price.
  - 1.3_Hourly_TotalDemand_with_Subtotal_of_EachArea
  - 1.4_Hourly_TodalDemand_with_Generation_from_DifferentPower
  - 1.5_Demand peak for the next day
  - 1.6_Fit_forecast
  - 1.7_Weather data
  - 1.8_Other data
    - 1.8.1_LNG price
- 2_Make all_data
  - 2.1_Merge all input data
  - 2.2_Adjust all_data (Add, remove outlier and drop features that have high correlation)
  - 2.3_Save the all_data as a csv file

Read & interpret dataset

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

※ If the following codes for reading csv do not work well, please directly download required files from my Dropbox link and read them.

- Datasets on Dropbox
  https://www.dropbox.com/sh/bd78j9g3yc7dvif/AAA5YMDsEDGm6FYFa1aKtv9oa?dl=0
  (https://www.dropbox.com/sh/bd78j9g3yc7dvif/AAA5YMDsEDGm6FYFa1aKtv9oa?dl=0)

In [1]:

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

# 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")

(Source)

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

In [2]:

```
#Intra_dataset
df_intra_2016 = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/im_trade_summa
ry_2016.csv', sep=',', header=0, encoding='shift_jis')
df_intra_2017 = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/im_trade_summa
ry_2017.csv', sep=',', header=0, encoding='shift_jis')
df_intra_2018 = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/im_trade_summa
ry_2018.csv', sep=',', header=0, encoding='shift_jis')
df_intra_2019 = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/im_trade_summa
ry_2019.csv', sep=',', header=0, encoding='shift_jis')
df_intra_2020 = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/im_trade_summa
ry_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])

# 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)'})

# 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 [3]:

```
df_intra_price = df_intra[df_intra['Date'] <= '2020-12-31']
df_intra_price = df_intra_price[["Open", "High", "Low", "Close", "Average"]]
df_intra_price.describe().round(2)
```

Out[3]:

	Open	High	Low	Close	Average
count	83256.00	83256.00	83256.00	83256.00	83256.00
mean	9.94	13.28	6.36	8.64	8.77
std	19.37	20.45	3.15	4.67	4.83
min	0.01	1.00	0.01	0.01	0.01
25%	6.74	8.90	4.69	6.12	6.53
50%	8.68	11.05	5.98	7.94	7.95
75%	11.00	14.38	7.63	10.00	9.82
max	801.00	801.00	75.20	150.00	194.65

In [4]:

```
# 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)

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

Out[4]:

```
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 [5]:

```
# Make a table for QH_number
df_intra_HH = df_intra.reset_index()
HH_table = pd.DataFrame(df_intra_HH["HH"])
HH_table = HH_table.drop_duplicates()
HH_table['Time'] = pd.date_range('2020/01/01', periods=48,  freq='30min').strftime('%H:%M')
HH_table['Time'] = pd.to_datetime(HH_table['Time']).dt.time

HH_table.head()
```

Out[5]:

	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 [6]:

```
# 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"].astype(str) + " " + cl
ose_price_graph["Time"].astype(str), 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 [7]:

```
# Simple plot
fig, ax = plt.subplots(1, figsize=plt.figaspect(.25))
plt.title('Price transition for close price')

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 [8]:

```
# 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)

close_HH_table.head()
```

Out[8]:

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

In [9]:

```
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 [10]:

```
# 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");
```

The item for the timezone from 16:30 to 17:30 tend to easy to spike the price.

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.

In [11]:

```
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=50, ax=ax1)

# # Partial Autocorrelation
# ax2 = fig.add_subplot(212)
# fig = sm.graphics.tsa.plot_pacf(close_price_graph, lags=50, ax=ax2)
```

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

In [12]:

```
# See the detail the data dividing into trends and seasonalities
seasonal_decompose_res = sm.tsa.seasonal_decompose(close_price_graph.values, freq=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 [13]:

```
# 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 v
alue)
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"]

# 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[13]:

	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 [14]:

```
# Price transition of Original close price
plt.plot(close_price_graph)
```

Out[14]:

[<matplotlib.lines.Line2D at 0x12b72f290>]

In [15]:

```
# 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 [16]:

```
# 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 [17]:

```
# 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.

Seasonality

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

In [18]:

```
# 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 [19]:

```
# 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[19]:

	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 [20]:

```
# 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', 'Da
y', '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_level_values(
'Month').astype(str) + "/" + close_price_graph_table_yearly.index.get_level_values('Day').astype(s
tr)
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", "Day", "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 [21]:

```
# Plot Average close price for each time slot based on Month
close_price_graph_table_monthly = pd.DataFrame(close_price_graph_table.pivot(index=['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_col, axis=1)
close_price_graph_table_monthly["Date"] = close_price_graph_table_monthly.index.get_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", "Time"], axis=1)
close_price_graph_table_monthly = close_price_graph_table_monthly.set_index("Date")
# close_price_graph_table_monthly = close_price_graph_table_monthly.drop(["Month"], 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 [22]:

```
# Plot Average close price for each time slot based on Weekday
close_price_graph_table_Weekly = pd.DataFrame(close_price_graph_table.pivot(index=['DayofWee
k', 'Time'], columns=['Year','Month','Week'], values='Close'))
close_price_graph_table_Weekly = close_price_graph_table_Weekly.groupby("DayofWeek").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_col, 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 [23]:

```
# 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'], colu
mns=['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 [24]:

```
# 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}")
```

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 [25]:

```
# 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])

# Plot
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[0:60]))
    rho.plot(label=label)

plt.title('The relationship between autocorrelation and the number of lag')
ax.set(ylim=(-1, 1), xlabel="Lags", ylabel="Autocorrelation (normalized)")
plt.legend(loc="upper left")
```

Out[25]:

```
<matplotlib.legend.Legend at 0x12ccdc4d0>
```

- 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 [26]:

```
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[26]:

	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 [27]:

```
# 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 [28]:

```
# 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 [29]:

```
# Calculate the monthly total volume
IntraVolume_daily = intra_volume.groupby("Date").agg({"Volume(MWh/h)": 'sum', "Volume(Tick co
unt)": '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[29]:

		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 [30]:

```
# 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 [31]:

```
# 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

(Source)

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

In [32]:

```
#Spot_dataset
df_spot_2016 = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/spot_2016.csv',
sep=',', header=0, encoding='shift_jis')
df_spot_2017 = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/spot_2017.csv',
sep=',', header=0, encoding='shift_jis')
df_spot_2018 = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/spot_2018.csv',
sep=',', header=0, encoding='shift_jis')
df_spot_2019 = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/spot_2019.csv',
sep=',', header=0, encoding='shift_jis')
df_spot_2020 = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/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])

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

# 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_volum
e(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/kW
h)': '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/kW
h)': '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/kW
h)': '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)'
                                 })

# # Apply to_datetime
df_spot["Date"] = pd.to_datetime(df_spot["Date"])

df_spot.head()
```

Out[32]:

	Date	HH	Sell_volume(kWh)	Buy_volume(kWh)	Total_volume(kWh)	System_price(Yen/kWh)
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 [33]:

```
# 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) + " " + spo
t_price_graph["Time"].astype(str), 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)

#  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.inde
x, 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 [34]:

```
# 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)


# 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 [35]:

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

spot_price_graph.head()
```

Out[35]:

DateTime	Sell_volume(kWh)	Buy_volume(kWh)	Total_volume(kWh)	System_price(Yen/kWh)	Pri
2016-04-01 00:00:00	5077000	2120500	675500	6.61	
2016-04-01 00:30:00	5621500	2239000	755500	6.34	
2016-04-01 01:00:00	5710500	2219000	713500	6.34	
2016-04-01 01:30:00	5536000	2202500	717500	6.25	
2016-04-01 02:00:00	5461000	2192000	870500	6.57	

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

In [36]:

```
# 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)

# 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}, ytic
klabels=cols.values, xticklabels=cols.values)
plt.show()
```

In [37]:

```
# Jouint plot
g = sns.jointplot(data=spot_intra, x="BidExceed_diff", y="Close", height=4)
g.plot_marginals(sns.rugplot, color="r", height=-.30, clip_on=False)
```

Out[37]:

<seaborn.axisgrid.JointGrid at 0x12b9cee90>

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 [38]:

```
# Try to see the national average of avoidable price
spot_intra["Avoidable_price_mean"] = round((spot_intra["Avoidable_price_HOK(Yen/kWh)"] + spo
t_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)

# 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}, ytic
klabels=cols.values, xticklabels=cols.values)
plt.show()
```

"Avoidable_price_mean" can be avoided as well.

In [39]:

```
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 [40]:

```
# 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)

df_spot.head()
```

Out[40]:

	Date	HH	System_price(Yen/kWh)	Price_Hokkaido(Yen/kWh)	Price_Tokyo(Yen/kWh)	Price_K
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

(Source)

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 [41]:

```
# Read dataset
df_HourlyDemand_All = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/Actual_Demand(Daily)_ALL.csv', sep=',', header=0, encoding='shift_jis')

# 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'})

# 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)

df_HourlyDemand_All.head()
```

Out[41]:

	Date	HourlyDemand_Hokkaido	HourlyDemand_Tohoku	HourlyDemand_Tokyo	HourlyDemar
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 [42]:

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

In [43]:

```
df_HourlyDemand_All.describe()
```

Out[43]:

	HourlyDemand_Hokkaido	HourlyDemand_Tohoku	HourlyDemand_Tokyo	HourlyDemand_(
count	42120.000000	42120.000000	42120.000000	42120.(
mean	3489.348884	9304.052042	32430.130722	15233.7
std	622.181993	1544.686428	6535.346217	3139.9
min	0.000000	5955.000000	18769.000000	8258.(
25%	2982.000000	8031.000000	27259.000000	12762.(
50%	3394.000000	9204.000000	31962.000000	14909.(
75%	3942.000000	10340.000000	36611.250000	17333.(
max	5422.000000	14796.000000	56532.000000	26243.(

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 [44]:

```
# 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 [45]:

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

In [46]:

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

Out[46]:

	Date	HourlyDemand_Hokkaido	HourlyDemand_Tohoku	HourlyDemand_Tokyo	HourlyD
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

(Source)

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 [47]:

```
# 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 [48]:

```
# File path
path = '/Users/kenotsu/Dropbox/Master_thesis_datasets/actual_generation_demand/'

# Tohoku as a base
df_Demand_AreaPlant = pd.read_csv(path + "/Demand_plant_Tohoku.csv", sep=',', header=0, enco
ding='shift_jis')
df_Demand_AreaPlant["DateTime"] =  pd.to_datetime(df_Demand_AreaPlant["DateTime"])

Areas = ["Hokkaido", "Tokyo", "Chubu", "Hokuriku", "Kansai", "Chugoku", "Shikoku", "Kyushu", "Oki
nawa"]

# Add other areas
for area in Areas:
    df = pd.read_csv(path + "/Demand_plant_" + area + ".csv", sep=',', header=0, encoding='shift_ji
s')
    df["DateTime"] = pd.to_datetime(df["DateTime"])
    df_Demand_AreaPlant = pd.merge(df_Demand_AreaPlant, df, how="left", on="DateTime")
df_Demand_AreaPlant.fillna(0, inplace=True)

# Drop columns that contain only 0
tmp=df_Demand_AreaPlant==0
col=df_Demand_AreaPlant.columns.values[tmp.sum(axis=0)==len(df_Demand_AreaPlant)]
df_Demand_AreaPlant = df_Demand_AreaPlant.drop(col, axis=1)

# Adjust Datatype from object to float for Tokyo and Kyushu area (Had to be pre-processed separat
ely)
object_cols = ["TotalDemand_TKO", "Thermal_TKO", "TotalSupply_TKO", "TotalDemand_KYU",
               "Nuclear_KYU", "Thermal_KYU", "PVCurtailment_KYU", "PumpedStorage_KYU", "Interconn
ection_KYU"]
for col in object_cols:
    df_Demand_AreaPlant[col] = df_Demand_AreaPlant[col].apply(lambda x: float(x.replace(",", "")))
    df_Demand_AreaPlant[col] = pd.to_numeric(df_Demand_AreaPlant[col].replace({',':'', '—':0}), do
wncast='integer')

# Adjust Datatype from object to float for Chugoku area (Had to be pre-processed separately)
object_cols = ["PVCurtailment_CHG", "WindCurtailment_CHG"]
for col in object_cols:
    df_Demand_AreaPlant[col] = pd.to_numeric(df_Demand_AreaPlant[col].replace({',':'', '—':0}), do
wncast='integer')
df_Demand_AreaPlant = get_group_stats(df_Demand_AreaPlant)

# Make Ratio columns
df_Demand_AreaPlant["Water_Ratio"] = round(df_Demand_AreaPlant["Allarea_Water"] / df_Dema
nd_AreaPlant["Allarea_TotalDemand"], 2)
df_Demand_AreaPlant["Thermal_Ratio"] = round(df_Demand_AreaPlant["Allarea_Thermal"] / df_D
emand_AreaPlant["Allarea_TotalDemand"], 2)
df_Demand_AreaPlant["Geothermal_Ratio"] = round(df_Demand_AreaPlant["Allarea_Geothermal"]
/ df_Demand_AreaPlant["Allarea_TotalDemand"], 2)

# 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", "WindCur
tailment_CHG",
                                                "TotalSupply_HOK", "TotalSupply_TKO", "TotalSupply_SHI", "TotalSuppl
y_OKI"],  axis=1)
```

In [49]:

```
# 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/Dropbox/Master_thesis_datasets/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/Dropbox/Master_thesis_datasets/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 [50]:

```
# 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)

# 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}, ytic
klabels=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 [51]:

```
# 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", "Total
Demand_SHI",
                                "TotalDemand_KYU", "TotalDemand_HKU", "TotalDemand_TKO","Total
Demand_KAN","TotalDemand_OKI", "TotalDemand_HOK",
                                "Thermal_CHB", "Thermal_TKO", "Thermal_KAN", "Thermal_TOH", "The
rmal_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_Wind
Curtailment", "Allarea_PV", "Allarea_PVCurtailment", "Biomass_CHG",
                                "Biomass_HKU", "Biomass_SHI", "Biomass_HOK", "Biomass_TKO", "Bio
mass_KYU", "Biomass_HKU", "Biomass_TOH", "PV_OKI",
                                "Allarea_Nuclear", "Allarea_Wind"
                                ], axis=1)

# 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}, ytic
klabels=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.

Demand peak for the next day

(Reference)

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 [52]:

```
# Daily demand peak
df_DemandPeak = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/DemandPeak_forecast.csv', sep=',', header=0, encoding='shift_jis')


# Rename the columns
df_DemandPeak = df_DemandPeak.rename(columns={'策定日': 'PlanDate', '対象日付': 'Date', '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_Min(MW)',
                                              '最大総需要予想時刻': 'Time_Max', '最大総需要予想(MW)': 'Planned_Max(MW)',
                                              '最大供給力予想(MW)': 'Planned_Max_Capa(MW)', '予想使用率': 'Planned_usege(%)', '予想予備率': 'Planned_margin(%)'})

# Add Date column
df_DemandPeak["Date"] = pd.to_datetime(df_DemandPeak["Date"])

# Order and index adjustment
df_DemandPeak = df_DemandPeak.set_index("Date")
df_DemandPeak = df_DemandPeak.sort_index(ascending=True)
df_DemandPeak = df_DemandPeak.reset_index()

# Pick up only rowa for the total of  10 areas
df_DemandPeak_All = df_DemandPeak[df_DemandPeak["Area"] == "10エリア計"]

df_DemandPeak_All.head()
```

Out[52]:

	Date	PlanDate	Area	Time_Min	Planned_Min(MW)	Time_Max	Planned_Max(MW)	Plan
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	

In [53]:

```
# Set flags
df_DemandPeak_All["Planned_Min_flag"] = 1
df_DemandPeak_All["Planned_Max_flag"] = 9

# 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[53]:

	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 [54]:

```
# 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[54]:

	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_forecast

(Source)

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&t

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 [55]:

```
# File path
path = "/Users/kenotsu/Dropbox/Master_thesis_datasets/Fit_actual_prediction/"

Areas = ["Hokkaido", "Tohoku", "Chubu", "Hokuriku", "Kansai", "Chugoku", "Shikoku", "Kyushu", "Ok
inawa"]

# Read Hokkkaido first as a base
df_Fit_forecast = pd.read_csv(path + "/FITforecast_" + "Tokyo" + ".csv", sep=',', header=0, encodi
ng='shift_jis')

# Add other areas
for area in Areas:
    df = pd.read_csv(path + "/FITforecast_" + area + ".csv", sep=',', header=0, encoding='shift_jis')
    df_Fit_forecast = pd.merge(df_Fit_forecast, df, how="left", on=["Date", "Time"])

df_Fit_forecast["Date"] = pd.to_datetime(df_Fit_forecast["Date"])
df_Fit_forecast = df_Fit_forecast.rename(columns={'Time': 'Time_forFIT'})

df_Fit_forecast.head()
```

Out[55]:

	Date	Time_forFIT	ForecastPV_Tokyo	ForecastWind_Tokyo	ForecastPV_Hokkaido	Forecast
0	2016-04-01	00:00:00	0	10,660	NaN	
1	2016-04-01	00:30:00	0	10,888	NaN	
2	2016-04-01	01:00:00	0	9,305	NaN	
3	2016-04-01	01:30:00	0	11,027	NaN	
4	2016-04-01	02:00:00	0	11,705	NaN	

In [56]:

```
df_Fit_forecast.isnull().sum()
```

Out[56]:

```
Date                    0
Time_forFIT             0
ForecastPV_Tokyo        0
ForecastWind_Tokyo      0
ForecastPV_Hokkaido     65760
ForecastWind_Hokkaido   65760
ForecastPV_Tohoku       0
ForecastWind_Tohoku     0
Forecast_PV_Chubu       0
Forecast_Wind_Chubu     0
ForecastPV_Hokuriku     35040
ForecastWind_Hokuriku   35040
ForecastPV_Kansai       0
ForecastWind_Kansai     0
ForecastPV_Chugoku      30720
ForecastWind_Chugoku    30720
ForecastPV_Shikoku      0
ForecastWind_Shikoku    0
Forecast_PV_Kyushu      0
Forecast_Wind_Kyushu    0
ForecastPV_Okinawa      0
ForecastWind_Okinawa    0
dtype: int64
```

In [57]:

```
# Fill the missing values with 0
df_Fit_forecast.fillna(0, inplace=True)

df_Fit_forecast.isnull().sum()
```

Out[57]:

```
Date                    0
Time_forFIT             0
ForecastPV_Tokyo        0
ForecastWind_Tokyo      0
ForecastPV_Hokkaido     0
ForecastWind_Hokkaido   0
ForecastPV_Tohoku       0
ForecastWind_Tohoku     0
Forecast_PV_Chubu       0
Forecast_Wind_Chubu     0
ForecastPV_Hokuriku     0
ForecastWind_Hokuriku   0
ForecastPV_Kansai       0
ForecastWind_Kansai     0
ForecastPV_Chugoku      0
ForecastWind_Chugoku    0
ForecastPV_Shikoku      0
ForecastWind_Shikoku    0
Forecast_PV_Kyushu      0
Forecast_Wind_Kyushu    0
ForecastPV_Okinawa      0
ForecastWind_Okinawa    0
dtype: int64
```

Dataset of Hokkaido, Hokuriku and Chugoku has no data before a certain point. --> 0

In [58]:

```
# Adjust HH table ※Made from df_actual_generation_Tohoku
HH_table_forFIT = pd.DataFrame(df_intra_HH["HH"])
HH_table_forFIT = HH_table_forFIT.drop_duplicates()
time = df_Fit_forecast["Time_forFIT"]
time = pd.DataFrame(time)
time = time.drop_duplicates()
HH_table_forFIT["Time_forFIT"] = time

HH_table_forFIT.head()
```

Out[58]:

	HH	Time_forFIT
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 [59]:

```
# Data table for heatmap
intra_close = df_intra[["Date", "HH", "Close"]]

df_Fit_forecast_withClose = df_Fit_forecast.copy().set_index("Date")
df_Fit_forecast_withClose = pd.merge(df_Fit_forecast, HH_table_forFIT, how='left', on=["Time_for
FIT"])
df_Fit_forecast_withClose = pd.merge(df_Fit_forecast_withClose, intra_close, how='left', on=['Dat
e', 'HH'])
df_Fit_forecast_withClose = df_Fit_forecast_withClose.drop("HH", axis=1)
df_Fit_forecast_withClose["Close"] = df_Fit_forecast_withClose["Close"].fillna(0)

# Filter the data
start = "2020-01-01"
df_Fit_forecast_withClose = df_Fit_forecast_withClose[df_Fit_forecast_withClose["Date"] >= star
t]

df_Fit_forecast_withClose.head()
```

Out[59]:

	Date	Time_forFIT	ForecastPV_Tokyo	ForecastWind_Tokyo	ForecastPV_Hokkaido	For
65760	2020-01-01	00:00:00	0	46571	0.0	
65761	2020-01-01	00:30:00	0	50650	0.0	
65762	2020-01-01	01:00:00	0	50168	0.0	
65763	2020-01-01	01:30:00	0	49981	0.0	
65764	2020-01-01	02:00:00	0	46914	0.0	

In [60]:

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

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

In [61]:

```
# Pick up the columns that have the absolute correlation more than 1.0
corrmat_abs = corrmat["Close"].abs()
High_corrmat_abs = corrmat_abs[corrmat_abs >= 0.1]
High_corrmat_abs
```

Out[61]:

```
ForecastPV_Hokkaido     0.146000
Forecast_PV_Chubu       0.141428
Forecast_Wind_Chubu     0.101875
ForecastPV_Hokuriku     0.145310
ForecastPV_Kansai       0.132510
ForecastWind_Kansai     0.190072
ForecastPV_Chugoku      0.131413
ForecastPV_Shikoku      0.131827
Forecast_PV_Kyushu      0.134642
ForecastWind_Okinawa    0.111284
Close                   1.000000
Name: Close, dtype: float64
```

[Add to features]

- ForecastPV_Hokkaido 0.146000
- Forecast_PV_Chubu 0.141428
- Forecast_Wind_Chubu 0.101875
- ForecastPV_Hokuriku 0.145310
- ForecastWind_Kansai 0.190072

[Not: high correlated with PV_Chubu]

- ForecastWind_Okinawa 0.111284
- ForecastPV_Kansai 0.132510
- ForecastPV_Chugoku 0.131413
- ForecastPV_Shikoku 0.131827
- Forecast_PV_Kyushu 0.134642

In [62]:

```
# Adjust the columns
df_Fit_forecast = df_Fit_forecast[["Date", "Time_forFIT", "ForecastPV_Hokkaido", "Forecast_PV_Chubu",
                                   "Forecast_Wind_Chubu", "ForecastPV_Hokuriku", "ForecastWind_Kansai"]]
df_Fit_forecast.head()
```

Out[62]:

	Date	Time_forFIT	ForecastPV_Hokkaido	Forecast_PV_Chubu	Forecast_Wind_Chubu	Fore
0	2016-04-01	00:00:00	0.0	0.0	3274.0	
1	2016-04-01	00:30:00	0.0	0.0	3026.0	
2	2016-04-01	01:00:00	0.0	0.0	3437.0	
3	2016-04-01	01:30:00	0.0	0.0	4297.0	
4	2016-04-01	02:00:00	0.0	0.0	5507.0	

Weather data

(Source)

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 [63]:

```
path = '/Users/kenotsu/Dropbox/Master_thesis_datasets/Weather'

# Read
df_weather_Tokyo = pd.read_csv(path + '/Weather_Tokyo.csv', sep=',', header=0, encoding='cp93
2')
df_weather_Hokkaido = pd.read_csv(path + '/Weather_Hokkaido.csv', sep=',', header=0, encoding=
'cp932')
df_weather_Okinawa = pd.read_csv(path + '/Weather_Okinawa.csv', sep=',', header=0, encoding='c
p932')

# Merge the datasets
df_weather_all = pd.merge(df_weather_Tokyo, df_weather_Hokkaido, how='left', on=['DateTime'])
df_weather_all = pd.merge(df_weather_all, df_weather_Okinawa, how='left', on=['DateTime'])

#Drop the columns that are not necessary
drop_cols = [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=drop_cols)

# 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'})

# Adjust time information
df_weather_all["Time"] = pd.to_datetime(df_weather_all["DateTime"]).dt.time
df_weather_all["Date"] = pd.to_datetime(df_weather_all["DateTime"]).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)
df_weather_all = df_weather_all.drop("DateTime", axis=1)
```

In [64]:

```
# Make a table for HH_number
df_intra_HH = df_intra.reset_index()
HH_table = pd.DataFrame(df_intra_HH["HH"])
HH_table = HH_table.drop_duplicates()
HH_table['Time'] = pd.date_range('2020/01/01', periods=48,  freq='30min').strftime('%H:%M')
HH_table['Time'] = pd.to_datetime(HH_table['Time']).dt.time

HH_table.head()
```

Out[64]:

	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 [65]:

```
# Data table for heatmap
intra_close = close_price[["Date", "HH", "Close"]]

# df_weather_all_withClose = df_weather_all.copy().set_index("Date")

df_weather_all_withClose = pd.merge(df_weather_all, HH_table, how='left', on=["Time"])
df_weather_all_withClose = pd.merge(df_weather_all_withClose, intra_close, how='left', on=['Date'
, 'HH'])
df_weather_all_withClose = df_weather_all_withClose.drop("HH", axis=1)
df_weather_all_withClose = df_weather_all_withClose.fillna(0)

df_weather_all_withClose.isnull().sum()
```

Out[65]:

```
Temp_Tokyo              0
SunLight(Time)_Tokyo      0
WindSpeed(m/s)_Tokyo      0
WindDirection_Tokyo       0
SunLight(MJ/㎡)_Tokyo      0
Temp_Hokkaido             0
SunLight(Time)_Hokkaido    0
WindSpeed(m/s)_Hokkaido    0
WindDirection_Hokkaido     0
SunLight(MJ/㎡)_Hokkaido    0
Temp_Okinawa              0
SunLight(Time)_Okinawa     0
WindSpeed(m/s)_Okinawa     0
WindDirection_Okinawa      0
SunLight(MJ/㎡)_Okinawa     0
Time                      0
Date                      0
Close                     0
dtype: int64
```

In [66]:

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

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

In [67]:

```
# Jointplot
g = sns.jointplot(data=df_weather_all_withClose, x="Temp_Tokyo", y="Close", height=8)
g.plot_marginals(sns.rugplot, color="r", height=-.15, clip_on=False)
```

Out[67]:

<seaborn.axisgrid.JointGrid at 0x12b090a90>

Temprature in each area has high correlation. --> Pick up the information of Tokyo.

In [68]:

```
# Joint plot
g = sns.jointplot(data=df_weather_all_withClose, x="SunLight(MJ/㎡)_Tokyo", y="Close", height=8
)
g.plot_marginals(sns.rugplot, color="r", height=-.15, clip_on=False)
```

Out[68]:

<seaborn.axisgrid.JointGrid at 0x133a4f810>

In [69]:

```
# Joint plot
g = sns.jointplot(data=df_weather_all_withClose, x="WindSpeed(m/s)_Tokyo", y="Close", height=4
)
g.plot_marginals(sns.rugplot, color="r", height=-.15, clip_on=False)
```

Out[69]:

<seaborn.axisgrid.JointGrid at 0x12c8fe810>

In [70]:

```
# Joint plot
g = sns.jointplot(data=df_weather_all_withClose, x="WindSpeed(m/s)_Hokkaido", y="Close", heigh
t=4)
g.plot_marginals(sns.rugplot, color="r", height=-.15, clip_on=False)
```

Out[70]:

<seaborn.axisgrid.JointGrid at 0x13a6174d0>

In [71]:

```
# Joint plot
g = sns.jointplot(data=df_weather_all_withClose, x="WindSpeed(m/s)_Okinawa", y="Close", height
=4)
g.plot_marginals(sns.rugplot, color="r", height=-.15, clip_on=False)
```

Out[71]:

<seaborn.axisgrid.JointGrid at 0x13fe33690>

The importance of wind speed in each area is also the similar. --> Pick up the information of Tokyo.

In [72]:

```
# Joint plot
g = sns.jointplot(data=df_weather_all_withClose, x="SunLight(Time)_Tokyo", y="Close", height=4)
g.plot_marginals(sns.rugplot, color="r", height=-.15, clip_on=False)
```

Out[72]:

<seaborn.axisgrid.JointGrid at 0x142b9a950>

In [73]:

```
# Joint plot
g = sns.jointplot(data=df_weather_all_withClose, x="WindDirection_Tokyo", y="Close", height=4)
g.plot_marginals(sns.rugplot, color="r", height=-.15, clip_on=False)
```

Out[73]:

<seaborn.axisgrid.JointGrid at 0x1458a7450>

Sunlight(Time) and WindDirection seems not to be so useful.

In [74]:

```
# Adjust the weather dataset that is added to all_data

# Read data (Choose Tokyo)
df_weather = pd.read_csv(path + '/Weather_Tokyo.csv', sep=',', header=0, encoding='cp932')

#Drop the columns that are not necessary
drop_cols = [item for item in df_weather.columns if item.find('Qual') != -1 or item.find('Num') != -1o
r item.find('None') != -1]
df_weather = df_weather.drop(columns=drop_cols)

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

# Categolize direction
df_weather = df_weather.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["Time"] = pd.to_datetime(df_weather["DateTime"]).dt.time
df_weather["Date"] = pd.to_datetime(df_weather["DateTime"]).dt.date
df_weather["Date"] = pd.to_datetime(df_weather["Date"])

cols = [item for item in df_weather.columns if item.find('WindDirection') != -1]
df_weather[cols] = df_weather[cols].apply(pd.to_numeric)
df_weather = df_weather.drop("DateTime", axis=1)
df_weather.head()
```

Out[74]:

	Temp_Tokyo	SunLight(Time)_Tokyo	WindSpeed(m/s)_Tokyo	WindDirection_Tokyo	SunLight(M m²)_Tok
0	13.3	0.0	1.4	11.0	
1	13.8	0.0	2.6	15.0	
2	13.0	0.0	3.1	15.0	
3	12.2	0.0	2.4	16.0	
4	11.2	0.0	1.5	16.0	

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.

(Source)

- Natural gas price https://www.macrotrends.net/2478/natural-gas-prices-historical-chart (https://www.macrotrends.net/2478/natural-gas-prices-historical-chart)
- Brent crude oil price https://www.macrotrends.net/2480/brent-crude-oil-prices-10-year-daily-chart (https://www.macrotrends.net/2480/brent-crude-oil-prices-10-year-daily-chart)
- WTI crude oil price https://www.macrotrends.net/2516/wti-crude-oil-prices-10-year-daily-chart (https://www.macrotrends.net/2516/wti-crude-oil-prices-10-year-daily-chart)

LNG price

Only daily data of LNG price exists. Let's see the correlation with Close price

In [75]:

```
# Read datasets
Gas_Price = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/natural gas prices-10-year-daily-chart.csv', sep=',', skiprows=list(range(0,15)), header=0)
BrentOil_Price = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/brent-crude-oil-prices-10-year-daily-chart.csv', sep=',', skiprows=list(range(0,15)), header=0)
WtiOil_Price = pd.read_csv('/Users/kenotsu/Dropbox/Master_thesis_datasets/wti-crude-oil-prices-10-year-daily-chart.csv', sep=',', skiprows=list(range(0,15)), header=0)

# Rename the columns
Gas_Price = Gas_Price.rename(columns={'date': 'Date', ' value': 'Gas_Price'})
BrentOil_Price = BrentOil_Price.rename(columns={'date': 'Date', ' value': 'BrentOil_Price'})
WtiOil_Price = WtiOil_Price.rename(columns={'date': 'Date', ' value': 'WtiOil_Price'})

# Merge the datasets and adjust
Commodity_price = pd.merge(Gas_Price, BrentOil_Price, how='left', on='Date')
Commodity_price = pd.merge(Commodity_price, WtiOil_Price, how='left', on='Date')
Commodity_price["Oil_Price"] = round((Commodity_price["BrentOil_Price"] + Commodity_price["WtiOil_Price"]) / 2, 2)
Commodity_price = Commodity_price.drop(["BrentOil_Price", "WtiOil_Price"], axis=1)
Commodity_price["Date"] = pd.to_datetime(Commodity_price["Date"])
Commodity_price = Commodity_price[Commodity_price["Date"] >= "2016-04-01"]
Commodity_price = Commodity_price[Commodity_price["Date"] <= "2020-12-31"].reset_index(drop=True)
Commodity_price
```

Out[75]:

	Date	Gas_Price	Oil_Price
0	2016-04-01	1.88	37.73
1	2016-04-04	1.96	36.70
2	2016-04-05	1.96	36.88
3	2016-04-06	1.91	38.80
4	2016-04-07	1.96	38.34
...	...	...	...
1204	2020-12-23	2.76	49.58
1205	2020-12-24	2.68	49.56
1206	2020-12-28	2.39	49.25
1207	2020-12-29	2.40	49.22
1208	2020-12-30	2.36	49.57

1209 rows × 3 columns

In [76]:

```
# Plot
commodity_plot = Commodity_price.set_index("Date")
commodity_plot.plot()
```

Out[76]:

<matplotlib.axes._subplots.AxesSubplot at 0x1459a6090>

In [77]:

```
# Compared with close price
close = close_price[["Date", "Close"]]
CommodityPrice_withClose = pd.merge(close, Commodity_price, how="left", on="Date")
CommodityPrice_withClose.interpolate(method='linear', inplace=True)
CommodityPrice_withClose = CommodityPrice_withClose.set_index("Date")
CommodityPrice_withClose
```

Out[77]:

	Close	Gas_Price	Oil_Price
Date			
2016-04-01	7.69	1.88	37.73
2016-04-01	7.45	1.88	37.73
2016-04-01	7.21	1.88	37.73
2016-04-01	7.06	1.88	37.73
2016-04-01	7.21	1.88	37.73
...	...	...	...
2020-12-31	70.00	2.36	49.57
2020-12-31	45.48	2.36	49.57
2020-12-31	41.33	2.36	49.57
2020-12-31	36.66	2.36	49.57
2020-12-31	26.46	2.36	49.57

83328 rows × 3 columns

In [78]:

```
# Close price correlation matrix
corrmat = CommodityPrice_withClose.corr()

k = 3 # The number of variables on the heatmap
cols = corrmat.nlargest(k, 'Close')['Close'].index
cm = np.corrcoef(CommodityPrice_withClose[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}, ytic
klabels=cols.values, xticklabels=cols.values)
plt.show()
```

In [79]:

```
# Joint plot
g = sns.jointplot(data=CommodityPrice_withClose, x="Gas_Price", y="Close", height=8)
g.plot_marginals(sns.rugplot, color="r", height=-.15, clip_on=False)
```

Out[79]:

<seaborn.axisgrid.JointGrid at 0x142bb5dd0>

In [80]:

```
# Joint plot
g = sns.jointplot(data=CommodityPrice_withClose, x="Oil_Price", y="Close", height=8)
g.plot_marginals(sns.rugplot, color="r", height=-.15, clip_on=False)
```

Out[80]:

<seaborn.axisgrid.JointGrid at 0x148648650>

Add two features to all_data

Make all_data

Merge all input data

In [81]:

```
# Make a table for QH_number
df_intra_HH = df_intra.reset_index()
HH_table = pd.DataFrame(df_intra_HH["HH"])
HH_table = HH_table.drop_duplicates()
HH_table['Time'] = pd.date_range('2020/01/01', periods=48,  freq='30min').strftime('%H:%M')
HH_table['Time'] = pd.to_datetime(HH_table['Time']).dt.time

HH_table.head()
```

Out[81]:

	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 [82]:

```
all_data = df_intra.copy()

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

# HH_table_forFIT (Will drop after merged)
all_data = pd.merge(all_data, HH_table_forFIT, 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_Fit_forecast
all_data = pd.merge(all_data, df_Fit_forecast, how="left", on=['Date', 'Time_forFIT'])
all_data = all_data.drop("Time_forFIT", axis=1)

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

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

all_data.head()
```

Out[82]:

	Date	HH	Open	High	Low	Close	Average	Volume(MWh/h)	Volume(Tick count)	Time	Sys
0	2016-04-01	1	7.69	7.69	7.69	7.69	7.69	0.7	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 [83]:

```
all_data.shape
```

Out[83]:

```
(84281, 138)
```

In [84]:

```
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
ForecastPV_Hokkaido        953
Forecast_PV_Chubu          953
Forecast_Wind_Chubu        953
ForecastPV_Hokuriku        953
ForecastWind_Kansai        953
Temp_Tokyo               42621
SunLight(Time)_Tokyo     42633
WindSpeed(m/s)_Tokyo     42635
WindDirection_Tokyo      42635
SunLight(MJ/㎡)_Tokyo    42633
Gas_Price                26249
Oil_Price                27545
dtype: int64
```

In [85]:

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

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

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
ForecastPV_Hokkaido      0
Forecast_PV_Chubu        0
Forecast_Wind_Chubu      0
ForecastPV_Hokuriku      0
ForecastWind_Kansai      0
Temp_Tokyo               0
SunLight(Time)_Tokyo     0
WindSpeed(m/s)_Tokyo     0
WindDirection_Tokyo      0
SunLight(MJ/㎡)_Tokyo    0
Gas_Price                0
Oil_Price                0
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 [86]:

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

all_data.shape
```

Out[86]:

```
(83328, 138)
```

- Add date information

In [87]:

```
# 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.timestamp).astype(in
t) / 86400 - 16891)

# Month column
all_data['month'] = pd.to_datetime(all_data["Date"]).dt.month

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

# 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)
```

In [88]:

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

Out[88]:

89

- Remove the features that have high correlation with each other

In [89]:

```
# Close price correlation matrix
corrmat = all_data.corr()

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

In [90]:

```
# from df_HourlyDemand_All  HourlyDemand
all_data = all_data.drop(['HourlyDemand_Tohoku',  'HourlyDemand_Chubu', 'HourlyDemand_Chugok
u',
                          'HourlyDemand_Shikoku', 'HourlyDemand_Kyushu', 'HourlyDemand_Hokuriku'], axis=1
)

# Drop features that have high correlation with each other as much as possible
all_data = all_data.drop([
                                "TotalDemand_TOH", "TotalDemand_CHB", "TotalDemand_CHG", "TotalDe
mand_SHI",
                                "TotalDemand_KYU", "TotalDemand_HKU", "TotalDemand_TKO","TotalDe
mand_KAN","TotalDemand_OKI", "TotalDemand_HOK",
                                "Thermal_CHB", "Thermal_TKO", "Thermal_KAN", "Thermal_TOH", "Therm
al_KYU", "Thermal_HKU", "Thermal_SHI", "Thermal_HOK",
                                "PumpedStorage_TKO", "PumpedStorage_KAN", "PumpedStorage_SHI",
"PumpedStorage_KYU", "PumpedStorage_CHG",
                                "PumpedStorage_HOK", "PumpedStorage_CHB", "Water_KAN", "Water_C
HB", "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_WindCu
rtailment", "Allarea_PV", "Allarea_PVCurtailment", "Biomass_CHG",
                                "Biomass_HKU", "Biomass_SHI", "Biomass_HOK", "Biomass_TKO", "Bioma
ss_KYU", "Biomass_HKU", "Biomass_TOH", "PV_OKI",
                                "Allarea_Nuclear", "Allarea_Wind"
                                ], axis=1)

all_data.shape
```

Out[90]:

(83328, 80)

In [91]:

```
all_data.describe()
```

Out[91]:

	HH	Open	High	Low	Close	Average
count	83328.000000	83328.000000	83328.000000	83328.000000	83328.000000	83328.000000
mean	24.500000	9.939637	13.278383	6.364566	8.635154	8.773715
std	13.853482	19.366047	20.441033	3.144575	4.666505	4.827427
min	1.000000	0.010000	1.000000	0.010000	0.010000	0.010000
25%	12.750000	6.740000	8.900000	4.700000	6.130000	6.530000
50%	24.500000	8.680000	11.040000	5.990000	7.930000	7.950000
75%	36.250000	11.000000	14.370000	7.640000	10.000000	9.810000
max	48.000000	801.000000	801.000000	75.200000	150.000000	194.650000

In [92]:

```
# Close price correlation matrix
corrmat = all_data.corr()

k = 30 # The number of variables on the heatmap
cols = corrmat.nlargest(k, 'Close')['Close'].index
cm = np.corrcoef(all_data[cols].values.T)
sns.set(font_scale=1.00)
f, ax = plt.subplots(figsize=(12,12))
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 8}, yticklabels=cols.values, xticklabels=cols.values)

plt.title('Correlation matrix for all_data')
plt.show()
```

Save the all_data as a csv file

In [93]:

```
all_data.head()
```

Out[93]:

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

In [94]:

```
all_data.to_csv('all_data.csv', index=False)
```