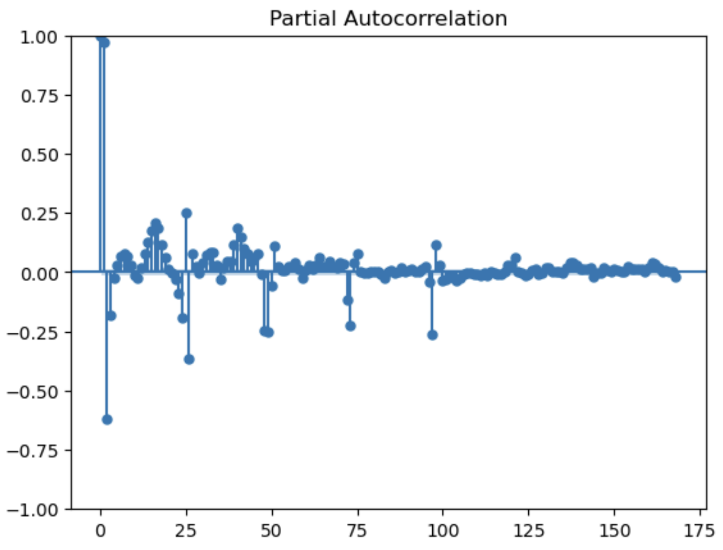
Dual MSc and MSc in CS

Midterm 2: Energy Forecasting



Subject: ECE 5260

Professor: Anna SCAGLIONE **as337**

Author(s): Naman MAKKAR **nbm49** Jan CARBONELL **jc2897**

Due date: April 26th

CORNELL TECH

- Reasoning and work must be shown to gain partial/full credit
- Please include the cover-page on your homework PDF with your name and student ID. Failure of doing so is considered bad citizenship.

In this homework, you are going to learn about various forecasting algorithms and their application to energy systems. Forecasting in energy systems is ubiquitous. Independent System Operators (ISOs) need weather forecasts, load forecast, renewable energy forecasts in day-to-day operations. Generators use fuel price forecast as well as electricity price forecasts to offer electricity in various markets. Every year, multi-million dollar decisions are made based on these forecasts. In this homework, we will forecast the load of an ISO using ARIMA models without exogenous variables and a forecasting method of your choice using exogenous variables. For this exercise, you may find the package statsmodels[1] useful.

1. (4 points) **Fundamentals of forecasting. SARIMA models. Lecture 17, Slide 37 onwards**: In this problem we will use an Seasonal ARIMA model to forecast a univariate time series without exogenous variables. An ARIMA model combines an AR(p), I(d) and MA(q) processes with parameters p,d and q, respectively. A seasonal ARIMA includes additional seasonal terms.[2]

   (a) **(0.25 points)** Using the data from the file *data.csv*, plot the load of the PJM Independent System Operator during the first week. Do you observe any seasonality? Comment your results.

   First we do all the necessary imports

In [1]:
```python
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
from scipy.fftpack import fft, ifft, rfft
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error,
    mean_absolute_percentage_error, mean_absolute_error
import lightgbm as lgb
from statsmodels.tsa.stattools import pacf
from statsmodels.tsa.stattools import acf
from sklearn.preprocessing import StandardScaler
import seaborn
import tensorflow as tf
```

   Then, we import the 3 datasets and proceed with a basic data cleaning

[1]https://www.statsmodels.org/stable/index.html
[2]https://otexts.com/fpp2/seasonal-arima.html

In [2]:
```
data = pd.read_csv('/kaggle/input/midterm-2-dataset/data.csv',
    index_col=0, parse_dates=True).sort_index()
next_day_forecast = pd.read_csv('/kaggle/input/midterm-2-
    dataset/load_forecast_next_day.csv', index_col=0,
    parse_dates=True).sort_index()
weather = pd.read_csv('/kaggle/input/midterm-2-dataset/weather.
    csv', index_col=0, parse_dates=True).sort_index()

data.head(24*7).plot()
```

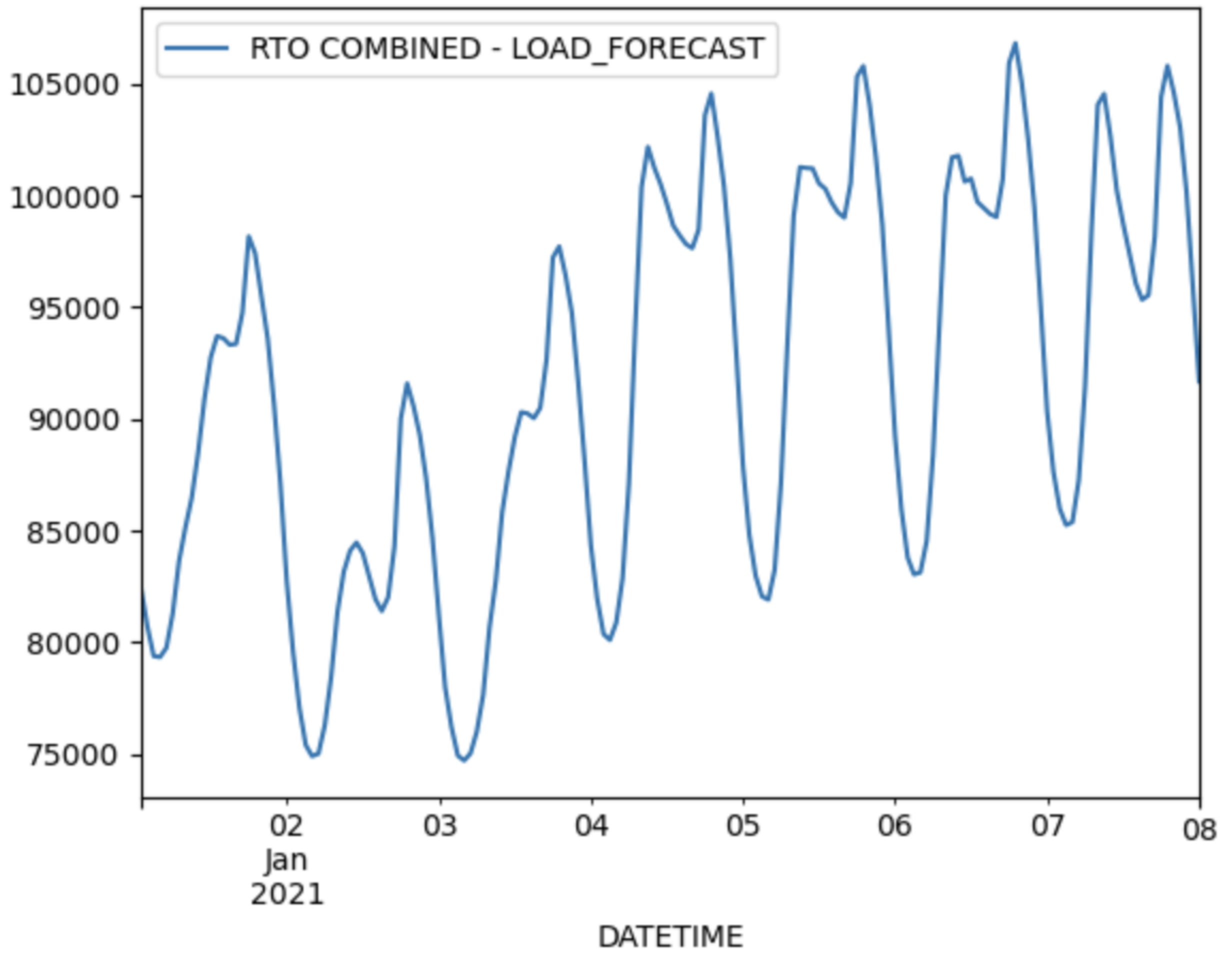


Figure 1: Load Forecast During the First Week

It is hard to tell for sure without more testing in the entire dataset, but there does seems to be a weekly repeating pattern in terms of the peak and a similar consumption period over the weekend. All while having a high inter-day variability. And with a slight increase in consumption over time.

(b) **(0.5 points)** Now, plot the Fast Fourier Transform of the signal to determine if there are any seasonalities. What do you observe? Is the seasonality daily, weekly, monthly or yearly? (**Observation**: You may find "weaker" forms of seasonality but we are concerned about the main seasonality that can be observed from the data, which should be obvious to the naked eye.)

Solution: The plot for the Fast Fourier Transform can be observed in **Figure 2**

In [3]:
```python
data = pd.read_csv('/kaggle/input/midterm-2-dataset/data.csv', index_col=0, parse_dates=True).sort_index()
next_day_forecast = pd.read_csv('/kaggle/input/midterm-2-dataset/load_forecast_next_day.csv', index_col=0, parse_dates=True).sort_index()
weather = pd.read_csv('/kaggle/input/midterm-2-dataset/weather.csv', index_col=0, parse_dates=True).sort_index()
fft = tf.signal.rfft(data['RTO COMBINED - LOAD_FORECAST'])
f_per_dataset = np.arange(0, len(fft))
n_samples_h = len(data)
hours_per_year = 24 * 365.2524
years_per_dataset = n_samples_h / (hours_per_year)

f_per_year = f_per_dataset / years_per_dataset
plt.step(f_per_year, np.abs(fft))
plt.xscale("log")
plt.ylim(0, 1.2e8)
plt.xlim([0.1, max(plt.xlim())])
plt.xticks([1, 2, 52, 365.2524], labels=["1/Y", "1/6M", "1/W", "1/D"])
_ = plt.xlabel("Frequency (log scale)")
```

In here we can see the effect of the hourly and daily which has a significant impact, well as the season between 1 year and 6 months. There is a minor impact of weekly and monthly but not as significant. This gives us an indication of possible features we can split our datetime in.

(c) **(0.5 points)** Test if the time series is stationary. For this, you may use the Augmented Dickey-Fuller test[3]. Does the test say that your data is stationary? Is it possible that your data is stationary after finding some seasonality in the data? Comment your results.

In [4]:
```python
def adf_test(time_series):
    print('Results of the Dickey Fuller Test for testing stationarity')
    dftest = adfuller(time_series, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value', '#Lags Used', 'Number of Observations Used'])
    for key, value in dftest[4].items():
        dfoutput['Critical Value (%s)' % key] = value
    print(dfoutput)
adf_test(data['RTO COMBINED - LOAD_FORECAST'])
```

[3]https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.adfuller.html

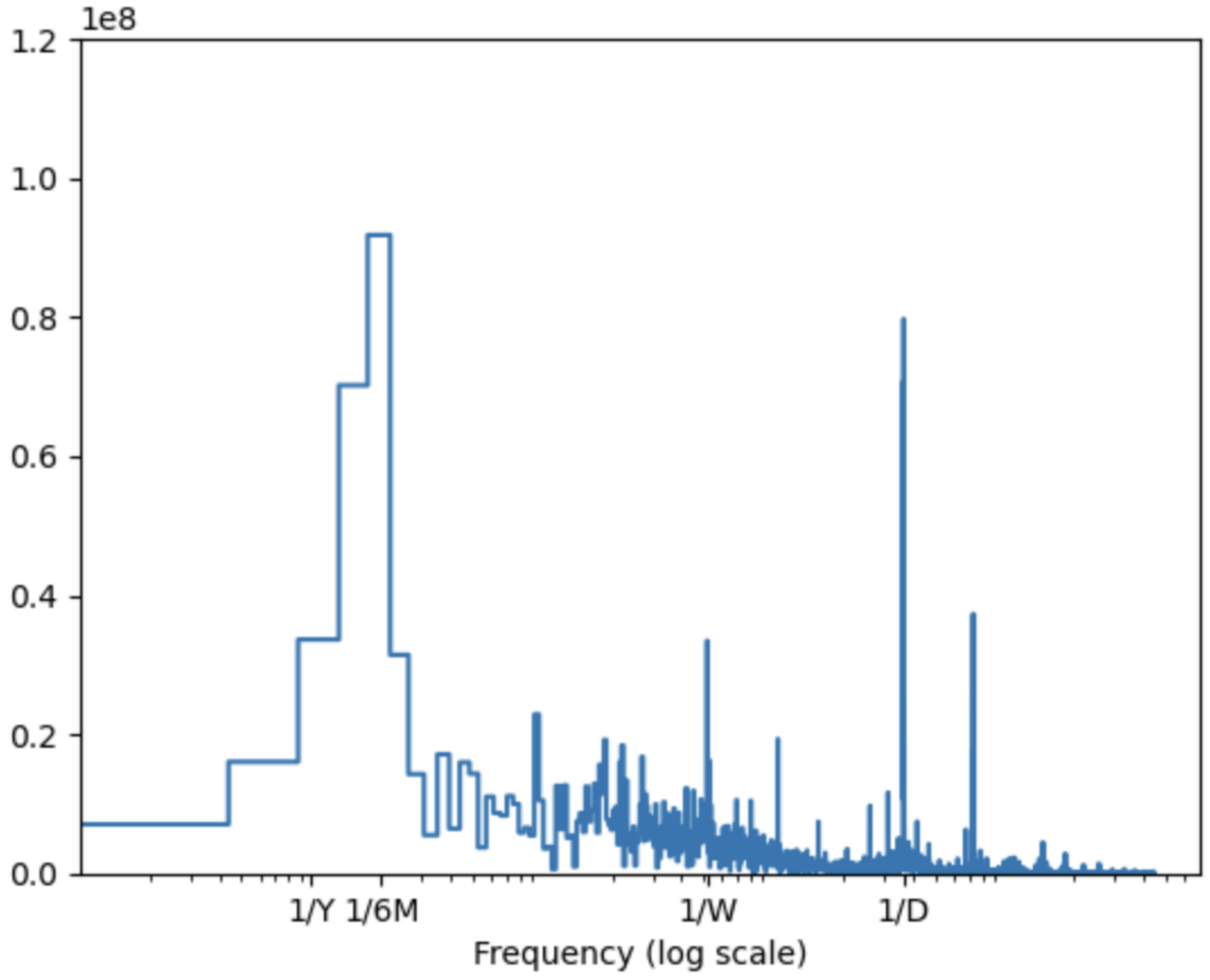


Figure 2: Plot of the Fast Fourier Transform

Out[4]:
```
Results of the Dickey Fuller Test for testing stationarity
Test Statistic                  -7.974911e+00
p-value                          2.718679e-12
#Lags Used                       4.600000e+01
Number of Observations Used      1.990000e+04
Critical Value (1%)             -3.430679e+00
Critical Value (5%)             -2.861685e+00
Critical Value (10%)            -2.566847e+00
dtype: float64
```

In this case, it means that the time series has a high likelihood of being seasonal on an daily and hourly basis and there's a minor periodicity on a daily and weekly basis and the other factors such as season (yearly).

Based on the Augmented Dickey-Fuller test results, the p-value is less than the chosen significance level of 0.05. Therefore, we can reject the null hypothesis and conclude that the time series is stationary.

(d) **(0.5 points)** Plot the partial autocorrelation and autocorrelation functions. Based on these plots, what can you observe? Provide a guess of your SARIMA parameters, (p,d,q) x (P,D,Q,S). You can use this guide[4] for your guess.

In [5]:
```python
plot_pacf(data['RTO COMBINED - LOAD_FORECAST'], method='ywm',
    lags=24*7)
```

In [6]:
```python
plot_acf(data['RTO COMBINED - LOAD_FORECAST'], lags=24*7)
```

As for the partial and total autocorrelation, we can see that the next 4 hours have a high significance (either positively or negatively correlated). Then there is a low effect and the cycle repeats again at the 24h cycle, indicating that the previous hours and days may have significant effect.

(e) **(1 points)** Now, use auto arima [5] with all available data, to find the optimal SARIMA parameters

Solution:

In [7]:
```python
!pip install pmdarima
import pmdarima as pm
model = pm.auto_arima(data['RTO COMBINED - LOAD_FORECAST'], D
    =1, start_P=1, max_P=1, start_Q=0, max_Q=0, seasonal=True, m
    =24, stationary=True, d=1, start_p=1, max_p=3, start_q=1,
    max_q=3)
model.summary()
```

(f) **(1 points)** Using the optimal parameters, implement your model in statsmodels[6]. Fit the

[4]https://arauto.readthedocs.io/en/latest/how_to_choose_terms.html
[5]https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html
[6]https://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html

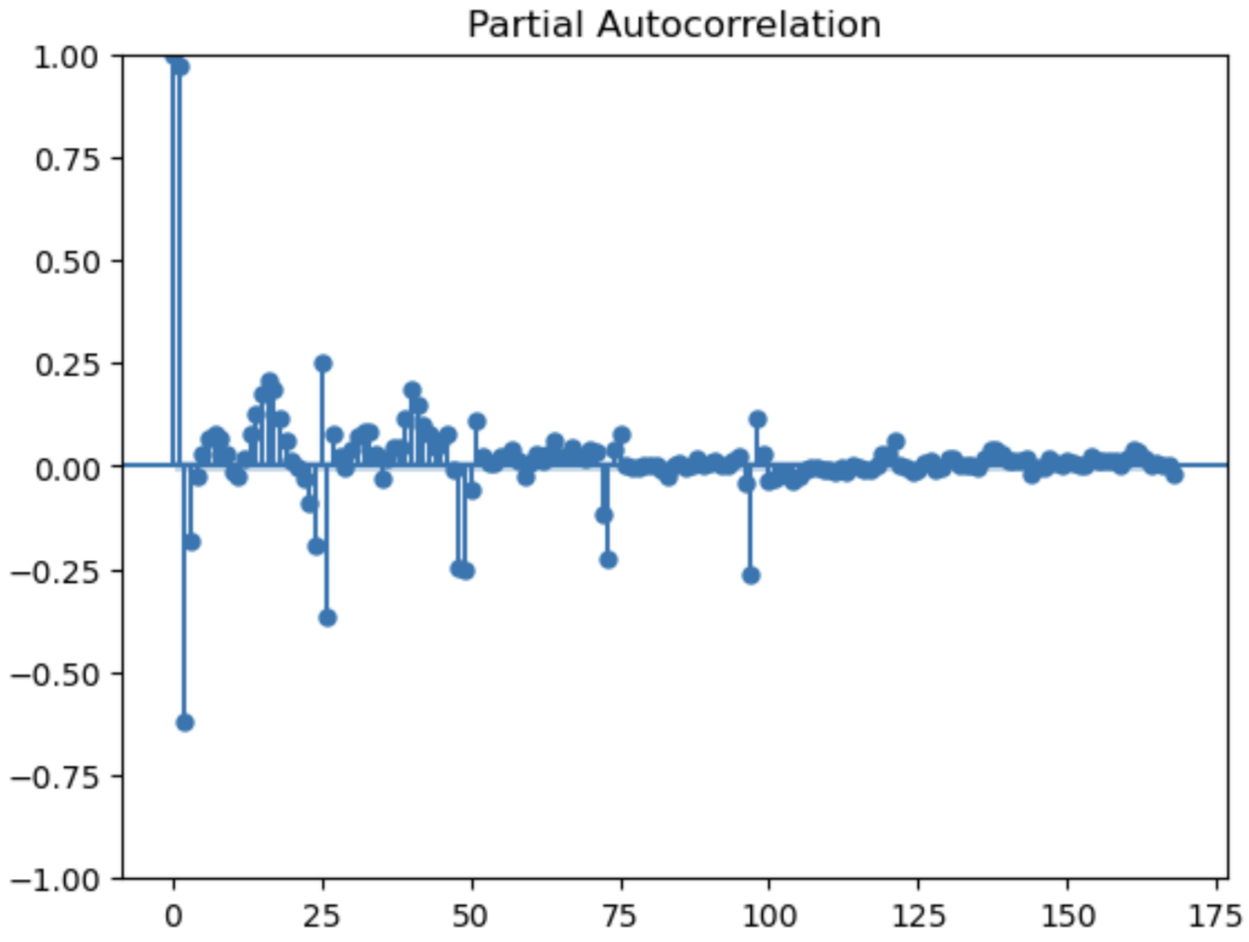


Figure 3: Plot of the Partial Autocorrelation Function

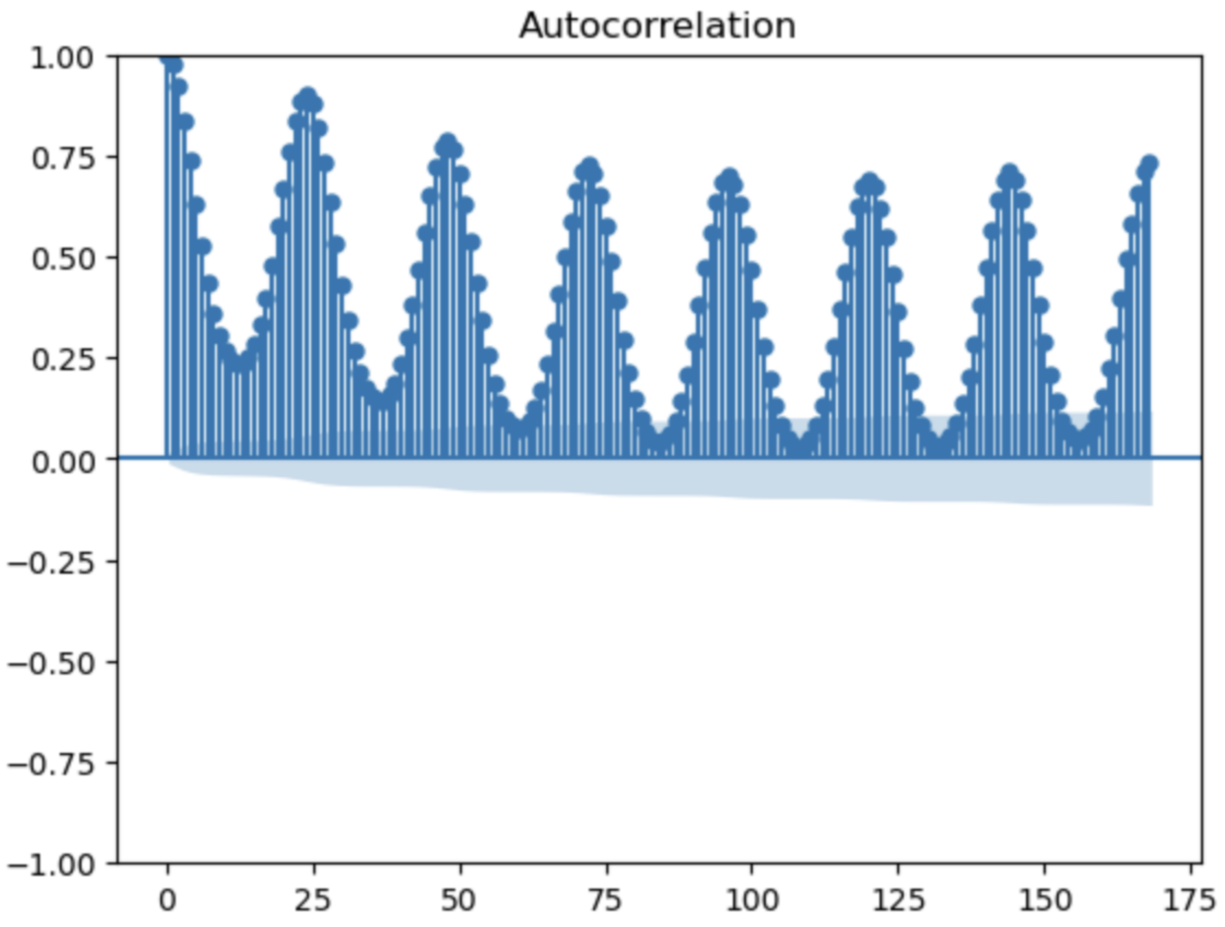
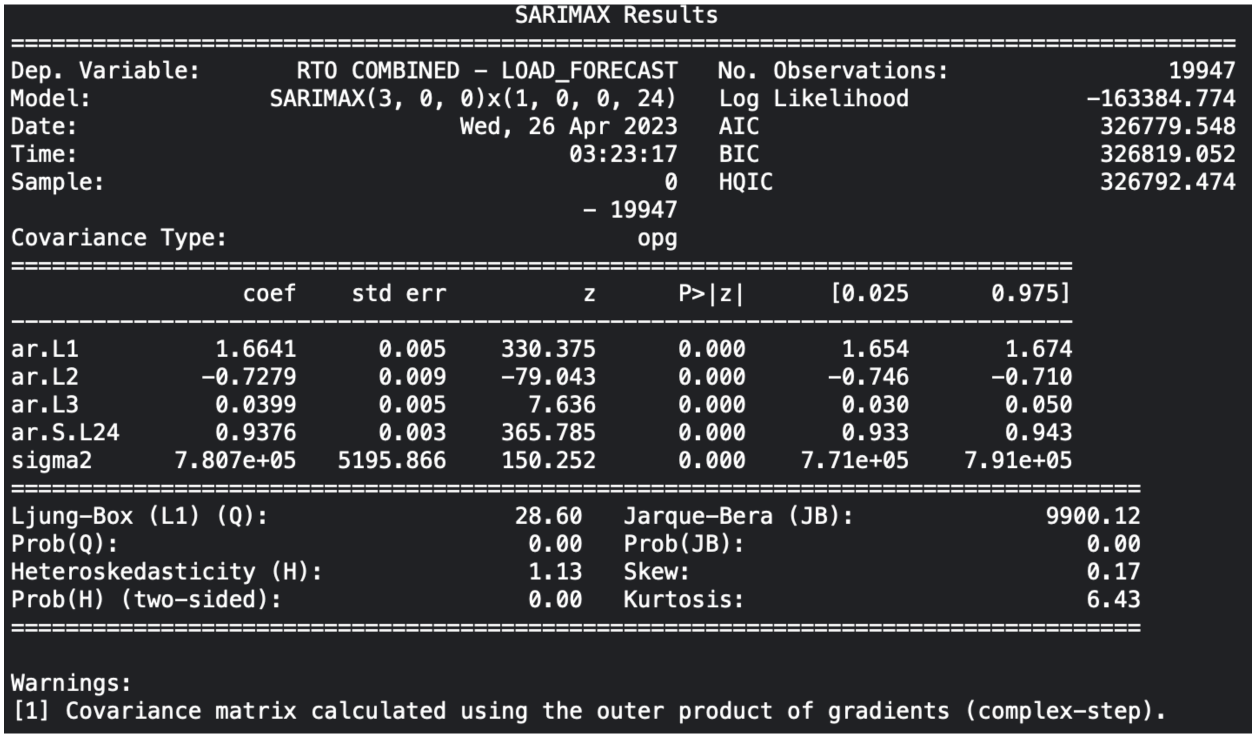


Figure 4: Plot of the Autocorrelation Function

model to all your data and test it out-of-sample for the next 24 hours. Plot the results obtained and compare them with the output from
load_forecast_next_day.csv, and calculate some statistics, e.g. RMSE, MAE, MAPE. Are your results accurate? Comment your work

Solution:

In [8]:
```
sarima_model = SARIMAX(endog=data['RTO COMBINED - LOAD_FORECAST
    '], order=(3,0,0), seasonal_order=(1,0,0,24),
    simple_differencing=True)
res = sarima_model.fit(disp=False)
print(res.summary())
```



```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:       RTO COMBINED - LOAD_FORECAST   No. Observations:                19947
Model:             SARIMAX(3, 0, 0)x(1, 0, 0, 24)   Log Likelihood              -163384.774
Date:                            Wed, 26 Apr 2023   AIC                          326779.548
Time:                                    03:23:17   BIC                          326819.052
Sample:                                         0   HQIC                         326792.474
                                          - 19947
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.6641      0.005    330.375      0.000       1.654       1.674
ar.L2         -0.7279      0.009    -79.043      0.000      -0.746      -0.710
ar.L3          0.0399      0.005      7.636      0.000       0.030       0.050
ar.S.L24       0.9376      0.003    365.785      0.000       0.933       0.943
sigma2      7.807e+05   5195.866    150.252      0.000    7.71e+05    7.91e+05
===================================================================================
Ljung-Box (L1) (Q):                  28.60   Jarque-Bera (JB):               9900.12
Prob(Q):                              0.00   Prob(JB):                          0.00
Heteroskedasticity (H):               1.13   Skew:                              0.17
Prob(H) (two-sided):                  0.00   Kurtosis:                          6.43
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Figure 5: Fitting a SARIMAX model to the data

To comment on the differences between auto-correlation and sarima:
- **Ljung-Box (Q)**: The p-value (Prob(Q)) is 28.60suggesting that there is some auto-correlation left in the residuals. - **Jarque-Bera (JB)**: The p-value (Prob(JB)) is 9990, which indicates that the residuals are normally distributed. - **Heteroskedasticity (H)**: The p-value (Prob(H)) is 0, which suggests that the residuals have non-constant variance. This is not ideal and might indicate the presence of some unexplained patterns in the data.

In [9]:
```
res.plot_diagnostics(figsize=(16, 8))
plt.show()
```

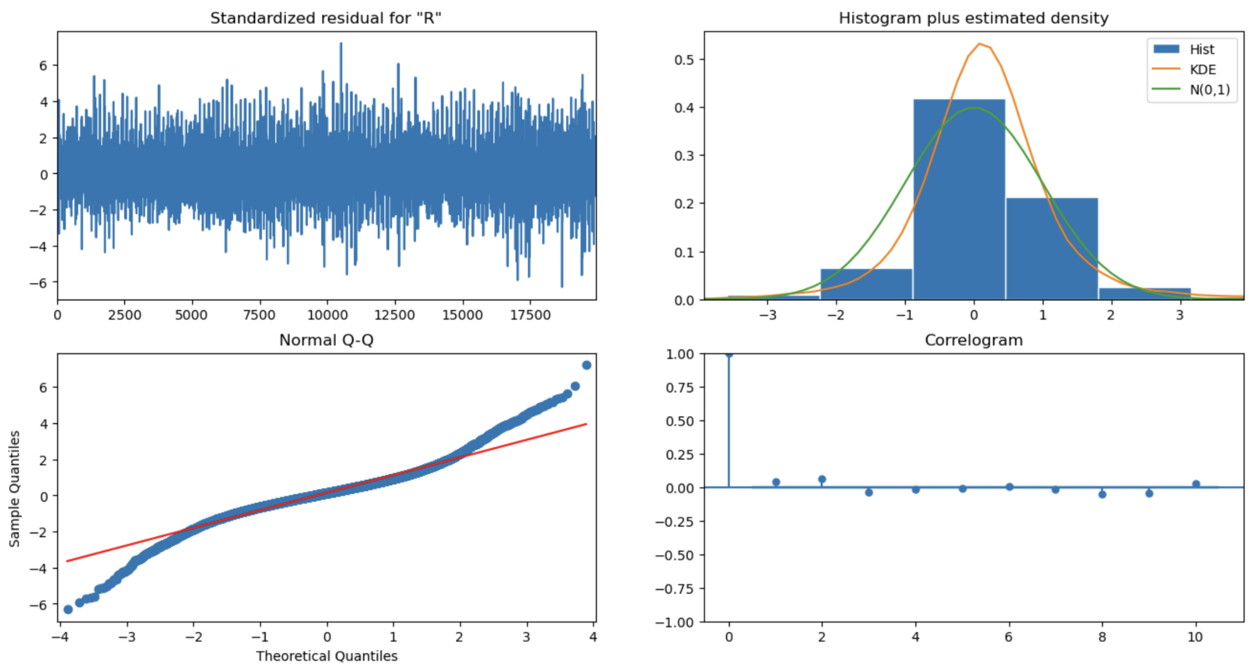


Figure 6: Plots for the SARIMAX model

As we can see from the plots in Figure 6, the standardized residual is not skewed towards any particular direction and the histogram fits more or less a normal probability.

In [10]:
```python
mape_error_sarima = np.mean(np.abs((next_day_forecast_true -
    next_day_forecast_pred)/next_day_forecast_true)*100)
rmse_sarima = np.sqrt(np.mean((next_day_forecast_true -
    next_day_forecast_pred)**2))
mae_sarima = mean_absolute_error(next_day_forecast_true,
    next_day_forecast_pred)
print(f'MAPE: {mape_error_sarima}%')
print(f'RMSE: {rmse_sarima}')
print(f'MAE: {mae_sarima}')
```

Out[10]:
```
MAPE: 3.5392940170951337%
RMSE: 3436.098679120796
MAE: 2763.4971295273263
```

(g) **(0.25 points)** How far out are you forecasting? In other words, what is the forecast lead time in hours and what is the length of your forecast

Based on this model, we are forecasting the next hour ahead effectively, as the comparison between each of the hours of our prediction and the next day's predicted gives us a MAPE of 3.59 percent.

2. (4 points) **Forecasting next-day electricity load**: In this problem, you are given an additional file *weather.csv* with weather data from PJM. The file contains hourly weather data for some of the main cities served by PJM. The columns of the file denote the location followed by the weather variable name.

   - TEMP - Temperature at 2m above ground
   - RH - Relative humidity
   - CLDC - Cloud cover
   - WIND - Wind speed at 10m above ground
   - PRCP - Precipitation
   - DEWP - Dew Point

   The goal is to forecast the electricity load. You may rely on any forecasting method you want. Also, feel free to use the forecasting jupyter notebook from the course.

   (a) **(1 points)** Propose a method to forecast PJM's next-day electricity load. **You should be able to forecast the hourly load for the next 24 hours at 12AM**, every single day. Do not forecast next-hour load like you did in the previous exercise. Describe your method and the features you need. You may use additional features that are not included in the csv file. Also, you could use dummy variables to encode special calendar dates, e.g. holidays, weekends, etc.

   The following proposal does not necessarily detail what we implemented but the ideal scenario we would have linked to follow if we had more time:

   - We should **clean the data** so that we can decompose the time series effect into individual features (hour, week, month, season, year).
   - **Dropping missing values** (for example, an entire day)
   - Create **lag features** that indicate the difference between the t-1 hour and the current one, week, month, year, etc.
   - Play with **auto-correlation** to see if the residuals that are resulting follow a specific pattern or distribution or are more or less random and create additional features to compensate for that.
   - Then, **integrate weather data**, as it may have a high incidence on AC usage and overall energy consumption and availability of wind and solar.
   - Normalize the data on the train set.
   - Finally, we do regression using the LGBM while finetuning its parameters (leaves, gbdt boosting, 0.2 learning rate, 0.9 feature fraction, 0.8 bagging fraction and 5 bagging frequency.
   - Finally, for the next day load 24 hour batches we take a moving window of 24h in order to predict the next day.

   And the following are theoretical approaches for which we did not have time to incorporate into our model:

   - We could create specific categorical columns (0-1) to factor for holidays or special celebrations such as the Superbowl that could drive consumption.

- Incorporate the model for forecasting the next hour based on the previous one as a feature in the dataset to have more accuracy in the hours that we are predicting further out. We did not do this in our current solution but it would be ideal.

(b) **(0.25 points)** Analyze your data, e.g. look for missing values/outliers. You may need to clean the data and do some pre-processing. A good way of doing this is to use seaborn's pairplot function[7].
Solution:

In [11]:
```python
fig = seaborn.pairplot(weather.iloc[:,0:5])
plt.show()
fig.savefig('temp_plot_dataset.png')
```

In [12]:
```python
fig = seaborn.pairplot(weather.iloc[:,5:10])
plt.show()
fig.savefig('rh_plot_dataset.png')
```

In [13]:
```python
fig = seaborn.pairplot(weather.iloc[:,15:20])
plt.show()
fig.savefig('wind_plot_dataset.png')
```

In [14]:
```python
fig = seaborn.pairplot(weather.iloc[:,10:15])
plt.show()
fig.savefig('cldc_plot_dataset.png')
```

In [15]:
```python
fig = seaborn.pairplot(weather.iloc[:,20:25])
plt.show()
fig.savefig('PRCP_plot_dataset.png')
```

In [16]:
```python
fig = seaborn.pairplot(weather.iloc[:,25:30])
plt.show()
fig.savefig('DEWP_dataset.png')
```

- As we can see from the figures, the temperature data has a high correlation across the differences in each of the cities and more or less follows a normal or bi-modal distribution.
- As for the humidity data, it can be observed that there is no correlation (all random) and it could be worth dropping from our feature set to avoid introducing random noise to the model.
- For wind data, there is some layer of correlation in some cities, but it more or less follows a random distribution.
- The cloud data follows and absolutely random distribution and should be dropped to increase the effects of our model

[7]https://seaborn.pydata.org/generated/seaborn.pairplot.html

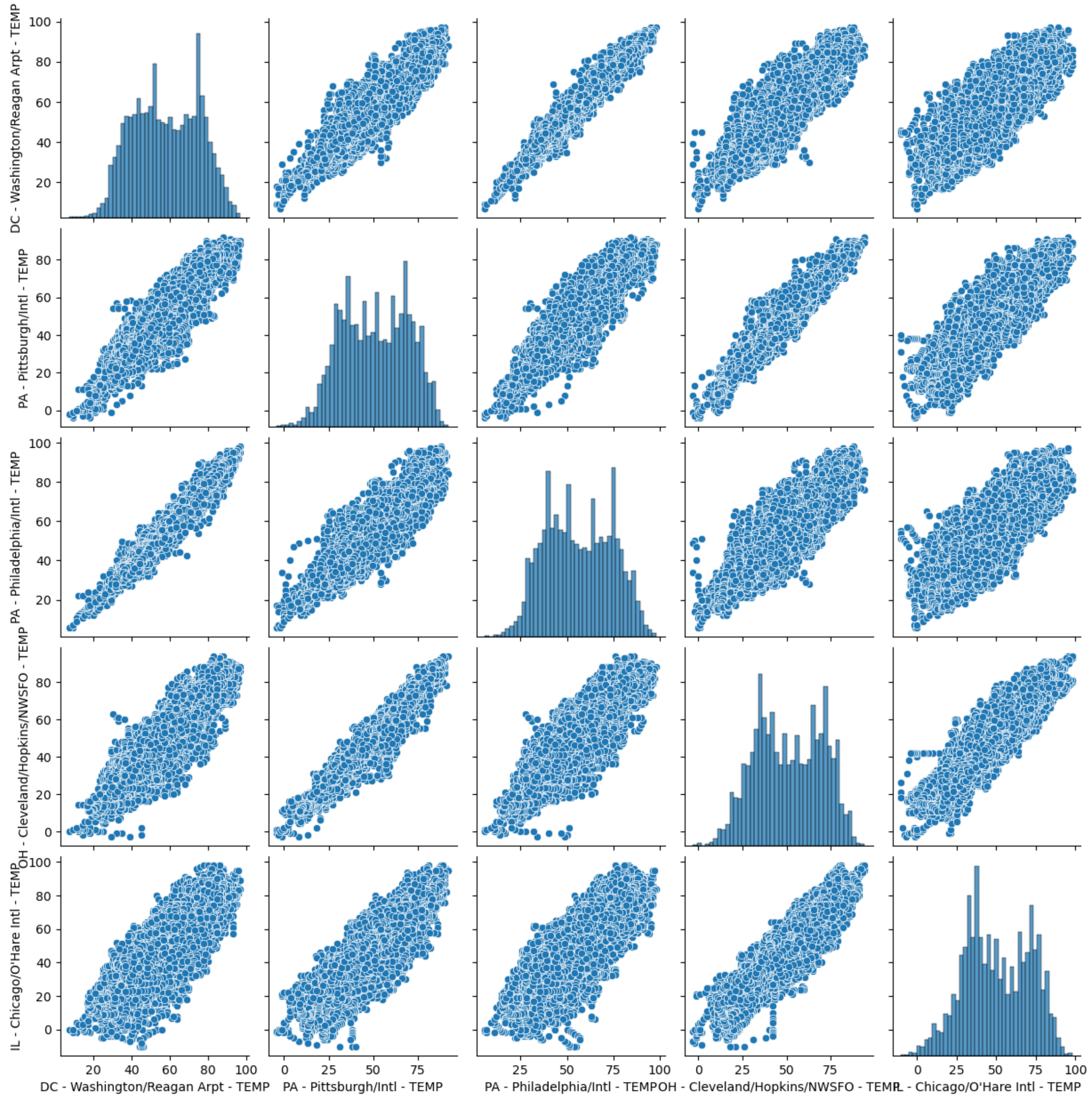


Figure 7: Plot of the Temperature Data

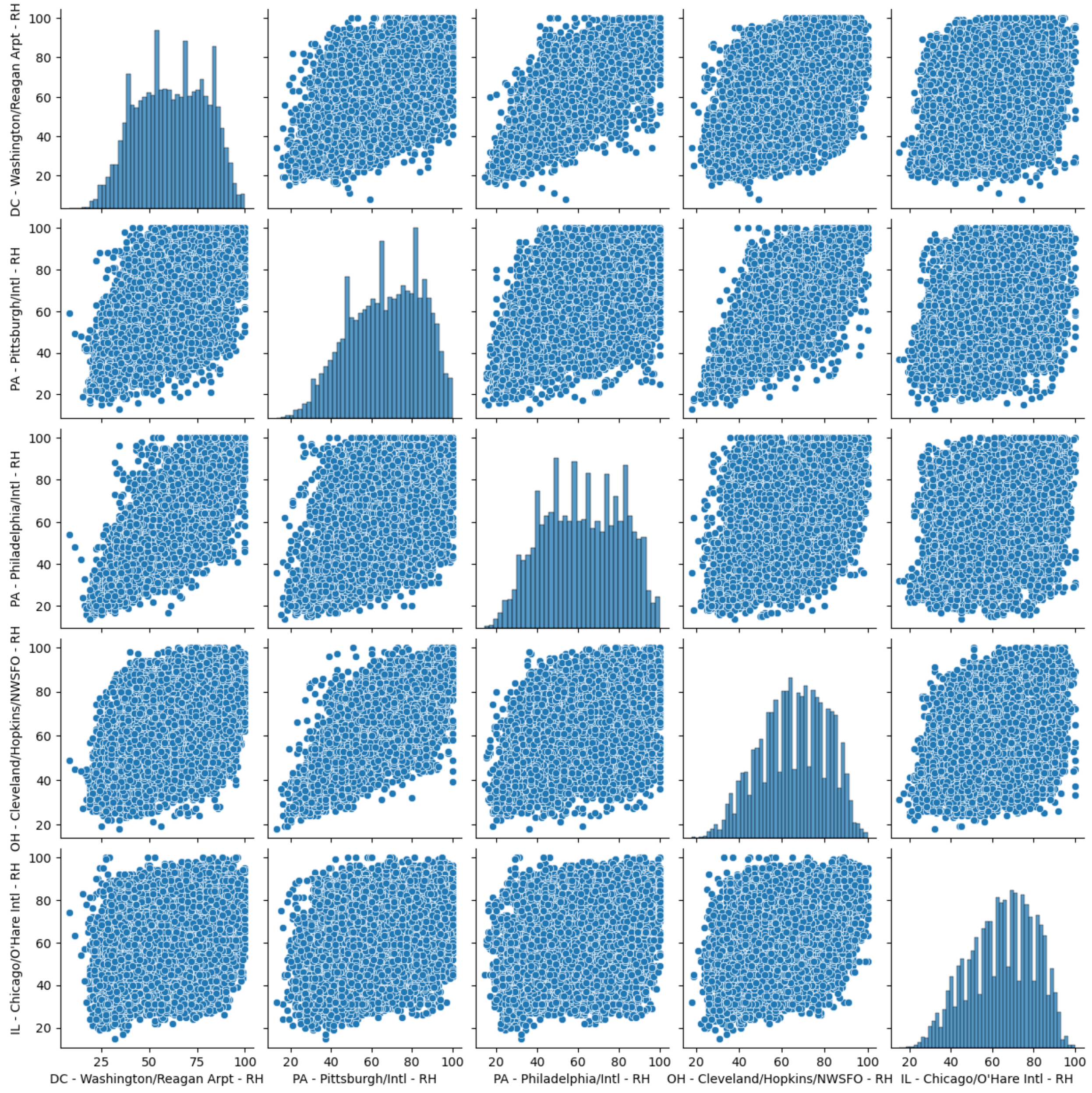


Figure 8: Plot of the Relative Humidity Data

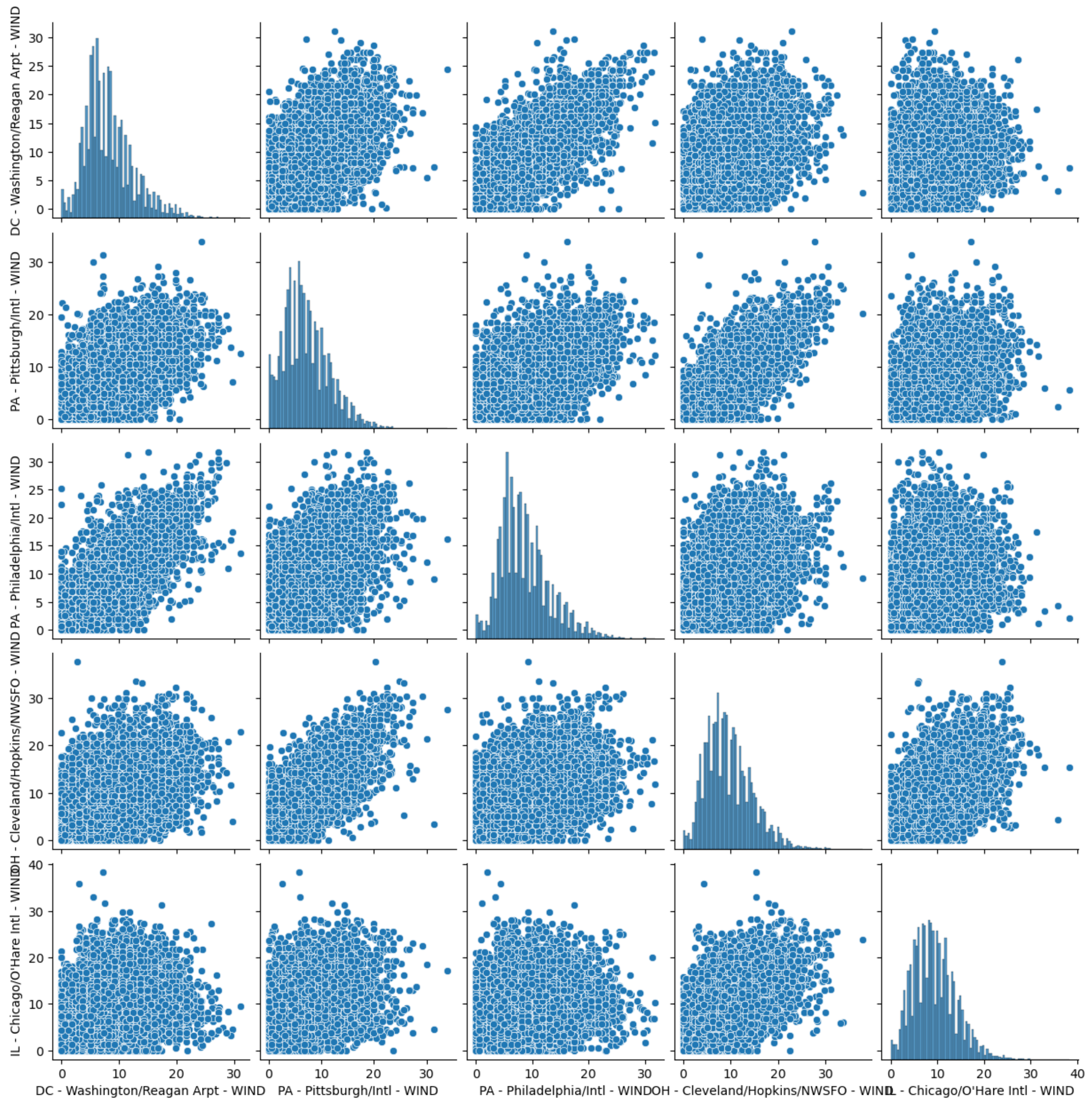


Figure 9: Plot of Wind Data

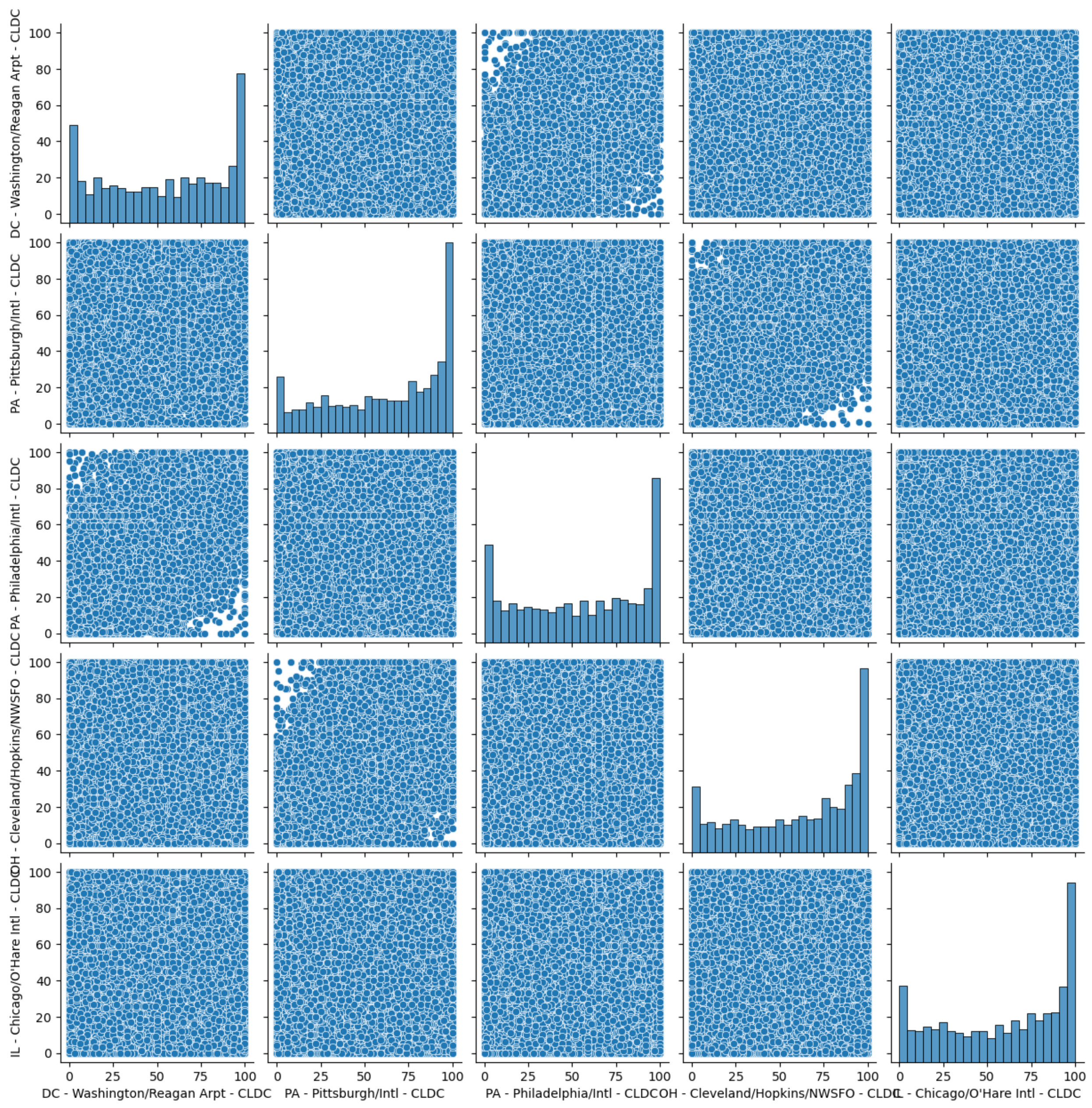


Figure 10: Plot of Cloud Cover Data

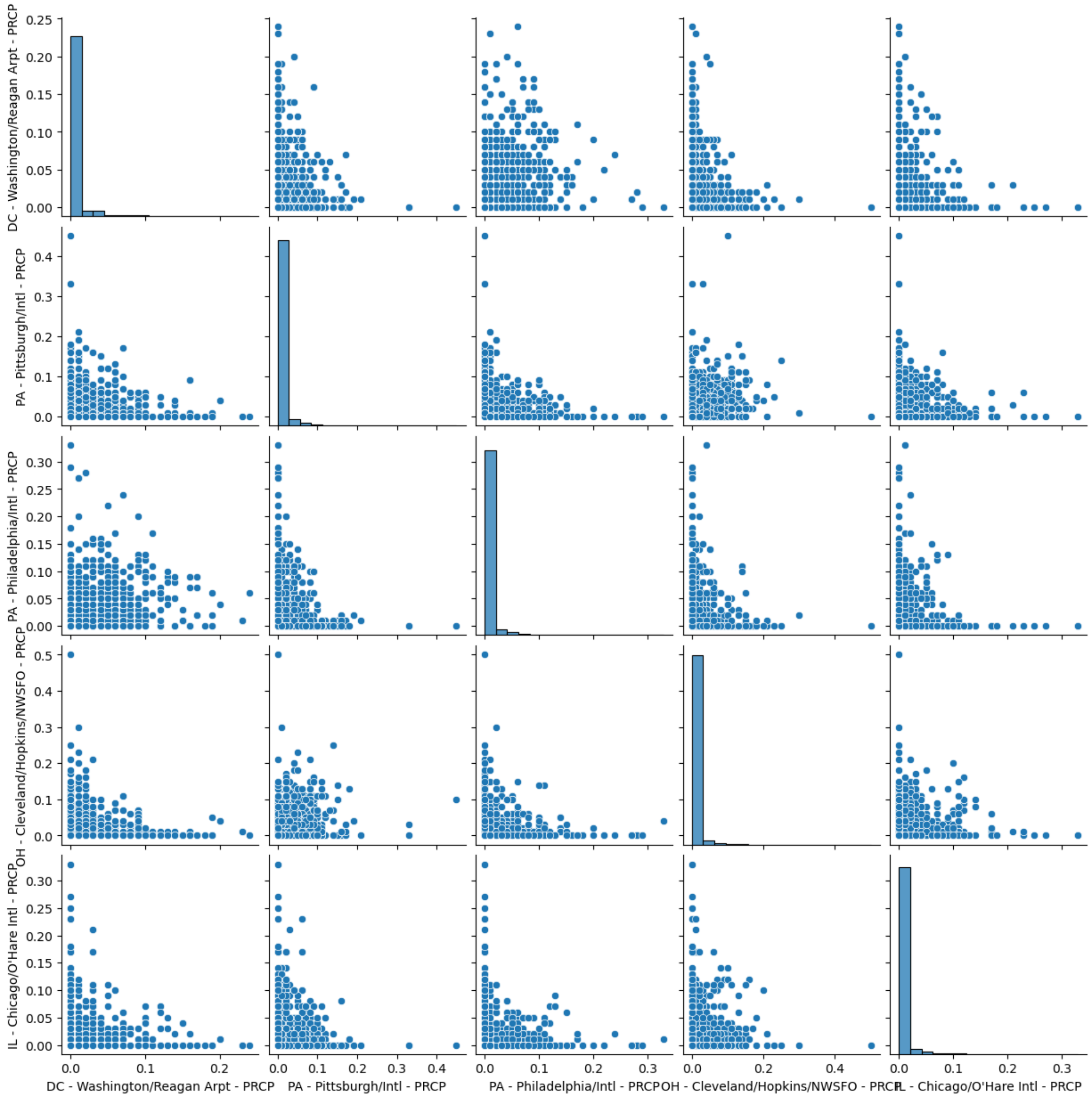


Figure 11: Plot of the Precipitation data

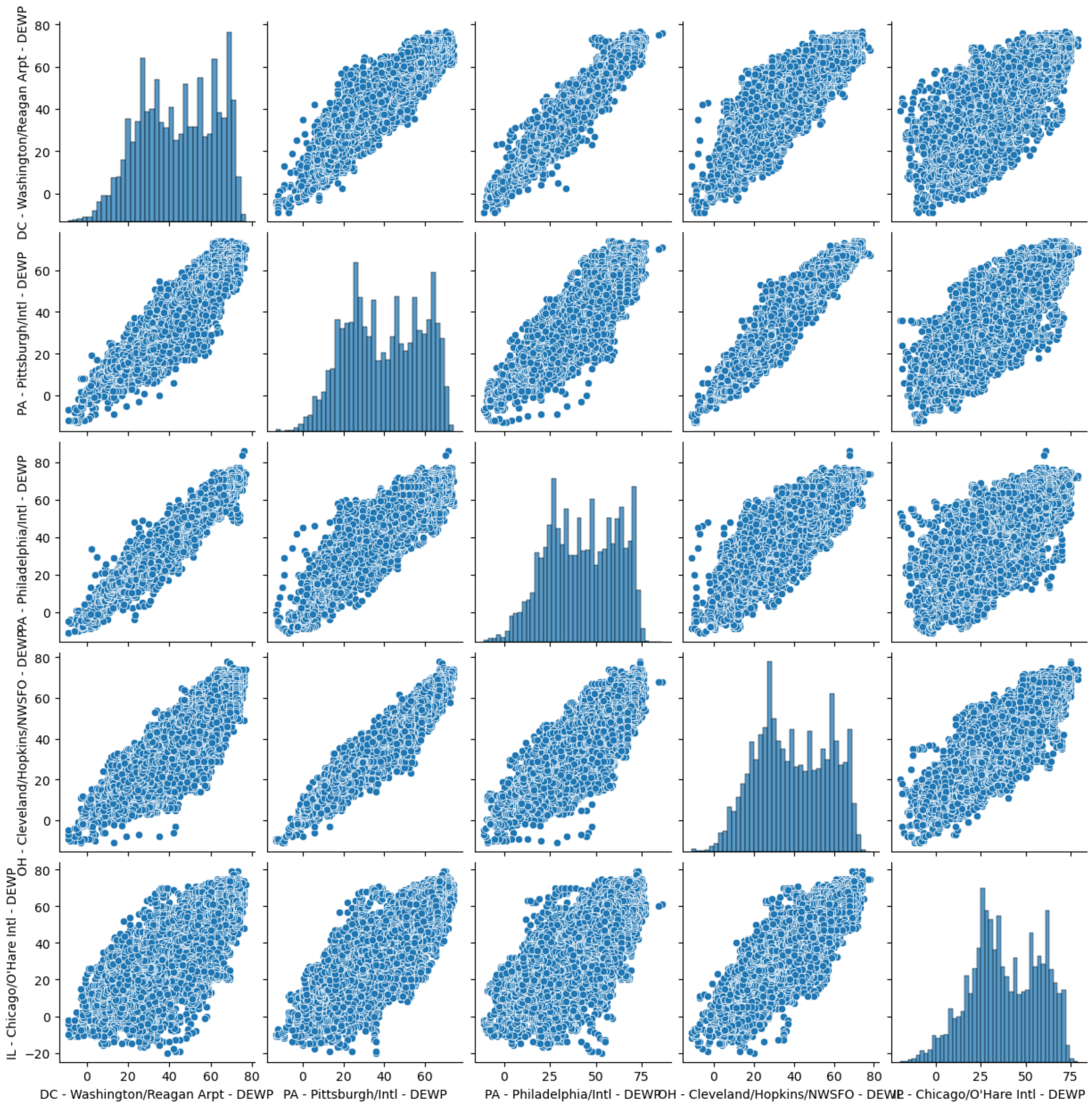


Figure 12: Plot of the Dew Point Data

- The precipitation data is interesting because it informs of a subset of the cities but in a very skewed distribution
- the Dew data is informative and shows signs of correlation inter-cities.

(c) **(0.25 points)** Look at the distribution of electricity load values. Are the values right-skewed or left-skewed? Propose a transformation to correct the skewness of the distribution. See this resource[8]. If you think more transformations are necessary, describe them and add them to the report.

Solution:

```
In [17]: skewness = data.skew()
         skewness = list(skewness)
         skewness
```

```
Out[17]:
         [0.7472950117985941]
```

The positive value tells us that the data distribution is right skewed and we need to make use of lower degree transformations in order to transform the data distribution. An appropriate transformation for this is the log transformation that we implement on the labels.

```
In [18]: data['RTO COMBINED - LOAD_FORECAST'] = np.log(data['RTO
             COMBINED - LOAD_FORECAST'])
```

(d) **(0.25 points)** Separate the dataset into training and testing. You can train on data from Jan 2021 to May 2022, and test on the rest of the dataset.

Solution:

```
In [19]: y_train = data['2021-01-01':'2022-05-31']['RTO COMBINED -
             LOAD_FORECAST']
         y_test = data['2022-06-01':]['RTO COMBINED - LOAD_FORECAST']
         X_train = weather['2021-01-01':'2022-05-31']
         X_test = weather['2022-06-01':]
```

(e) **(0.25 points)** Normalize the data using the mean and standard deviation of the training set.

Solution:

[8] http://seismo.berkeley.edu/~kirchner/eps_120/Toolkits/Toolkit_03.pdf

In [20]:
```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = pd.DataFrame(scaler.transform(X_train),
    columns=X_train.columns, index=X_train.index)
X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns=
    X_test.columns, index=X_test.index)
```

(f) **(1 points)** Fit a model to your data and provide the in-sample hourly RMSE, MAPE and MAE of the fitted values.
Solution:

In [21]:
```
lgb_train = lgb.Dataset(X_train_scaled, y_train)
lgb_eval =  lgb.Dataset(X_test_scaled, y_test, reference=
    lgb_train)
params = {
    'boosting_type':'gbdt',
    'objective':'regression',
    'metric':{'l2','l1'},
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'verbose':0
}
print('Training....')
gbm =  lgb.train(params, lgb_train, num_boost_round=20,
    valid_sets=lgb_eval, callbacks=[lgb.early_stopping(
    stopping_rounds=5)])
print('Saving Model....')
gbm.save_model('model.txt')
print('Running Inference....')
y_pred = gbm.predict(X_test_scaled, num_iteration=gbm.
    best_iteration)
```

Out[21]:
```
Training....
[LightGBM] [Warning] Found whitespace in feature_names, replace
    with underlines
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the
    overhead of testing was 0.003749 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Warning] Found whitespace in feature_names, replace
    with underlines
Training until validation scores don't improve for 5 rounds
Did not meet early stopping. Best iteration is:
[20]  valid_0's l2: 0.0106965 valid_0's l1: 0.0856539
Saving Model....
Running Inference....
```

In [22]:
```
y_test = np.array(y_test)
y_train = np.array(y_train)
y_pred_in_samples = gbm.predict(X_train_scaled, num_iteration=
    gbm.best_iteration)
mape_lgbm_in_sample = mean_absolute_percentage_error(np.exp(
    y_train), np.exp(y_pred_in_samples))
mae_lgbm_in_sample = mean_absolute_error(np.exp(y_train), np.
    exp(y_pred_in_samples))
rmse_train = mean_squared_error(np.exp(y_train), np.exp(
    y_pred_in_samples))**0.5
print(f'RMSE of in samples using LightGBM regressor is {
    rmse_train}')
print(f'In sample MAPE: {mape_lgbm_in_sample*100}%')
print(f'In sample MAE: {mae_lgbm_in_sample}')
```

Out[22]:
```
RMSE of in samples using LightGBM regressor is 9123.060057680126
In sample MAPE: 8.294488632691506%
In sample MAE: 7377.174213264255
```

In [23]:
```
mape_lgbm_out_of_sample = mean_absolute_percentage_error(np.exp
    (y_test), np.exp(y_pred))#np.mean(np.abs((y_test - y_pred)/
    y_test)*100)
mae_lgbm_out_of_sample = mean_absolute_error(np.exp(y_test), np
    .exp(y_pred))
rmse_test = mean_squared_error(np.exp(y_test), np.exp(y_pred))
    **0.5
print(f'RMSE of prediction using LightGBM regressor is: {
    rmse_test}')
print(f'Out of sample MAPE: {mape_lgbm_out_of_sample*100}%')
print(f'Out of sample MAE: {mae_lgbm_out_of_sample}')
```

Out[23]:
```
RMSE of prediction using LightGBM regressor is: 9794.496246930374
Out of sample MAPE: 8.49300157037133%
Out of sample MAE: 7872.5363211894155
```

(g) **(1 points)** Forecast the next-day load in 24 hour batches. You can recycle the model you previously fitted, or you can refit your model every time the model sees new data. Provide the out-of-sample RMSE, MAPE and MAE.

Solution:

In [24]:
```
def lgb_model_train(train_df, train_y, test_df, test_y,
    model_idx):
    lgb_train = lgb.Dataset(train_df, train_y)
    lgb_test = lgb.Dataset(test_df, test_y, reference=lgb_train
    )
    params = {
    'boosting_type':'gbdt',
    'objective':'regression',
    'metric':{'l2','l1'},
    'num_leaves': 31,
    'learning_rate': 0.003,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'verbose':0 }
    gbm_new =  lgb.train(params, lgb_train_big, num_boost_round
    =20, valid_sets=lgb_test_big, callbacks=[lgb.early_stopping(
    stopping_rounds=5)])
    gbm_new.save_model(f'improved_model_{model_idx}.txt')
    pred_y = gbm_new.predict(test_df, num_iteration=gbm.
    best_iteration)
    y_train_pred = gbm_new.predict(train_df, num_iteration=gbm.
    best_iteration)
    return pred_y, y_train_pred

X = df_lags.iloc[:,:-1]
X = (X - X.mean())/X.std()
y = df_lags.iloc[:,-1]
n_train = 12264
n_records = n_train + 7416
y_trains = []
y_train_preds = []
y_tests= []
y_preds = []

for idx in range(n_train, n_records, 24):
    train_df, test_df = X.iloc[:idx], X.iloc[idx:idx+24]
    y_train, y_test = y.iloc[:idx], y.iloc[idx:idx+24]

    y_pred, y_train_pred = lgb_model_train(train_df, y_train,
    test_df, y_test, idx)
    y_trains = np.concatenate((y_trains, y_train))
    y_train_preds = np.concatenate((y_train_preds, y_train_pred
    ))
    y_tests = np.concatenate((y_tests, y_test))
    y_preds = np.concatenate((y_preds, y_pred))
```

In [25]:
```
mape_lgbm_out_of_sample = mean_absolute_percentage_error(np.exp
    (y_tests), np.exp(y_preds))#np.mean(np.abs((y_test - y_pred)
    /y_test)*100)
mae_lgbm_out_of_sample = mean_absolute_error(np.exp(y_tests),
    np.exp(y_preds))
rmse_test = mean_squared_error(np.exp(y_tests), np.exp(y_preds)
    )**0.5
print(f'RMSE of prediction using LightGBM regressor is: {
    rmse_test}')
print(f'Out of sample MAPE: {mape_lgbm_out_of_sample*100}%')
print(f'Out of sample MAE: {mae_lgbm_out_of_sample}')
```

Out[25]:
```
RMSE of prediction using LightGBM regressor is: 15365.964199906239
Out of sample MAPE: 12.115340084172582%
Out of sample MAE: 11519.104187649507
```

(h) (**Extra Credit. 2pts or 20% of the grade**). Propose a method that achieves less than 5% MAPE **out-of-sample**.

Solution: The solution makes use of lagging, and the feature values are lagged for 144 hours. The training and test datasets are normalised and then a lightGBM regression model is trained with a learning rate of 0.2. The model achieves an MAPE of 4.03% on the out of sample test dataset.

In [26]:
```python
combined_weather_data = weather.join(data)
df_lags = combined_weather_data.copy()
for lag in np.arange(24, 144+1, 24):
    df_lag = combined_weather_data.shift(lag, freq='h')
    df_lag.columns = [str(col) + f'_lag_{lag}' for col in
   df_lag.columns]
    df_lags = df_lags.join(df_lag)
df_lags.dropna(inplace=True)
df_lags = df_lags[~df_lags.index.duplicated()]
train_df = df_lags.iloc[:,:-1]['2021-01-01':'2022-06-01']
test_df = df_lags.iloc[:,:-1]['2022-06-01':]
train_y = df_lags.iloc[:,-1]['2021-01-01':'2022-06-01']
test_y = df_lags.iloc[:,-1]['2022-06-01':]
train_mean = train_df.mean()
train_std = train_df.std()
train_df = (train_df - train_mean)/train_std
test_df = (test_df - train_mean)/train_std
lgb_train_big = lgb.Dataset(train_df, train_y)
lgb_test_big = lgb.Dataset(test_df, test_y, reference=
    lgb_train_big)
params = {
    'boosting_type':'gbdt',
    'objective':'regression',
    'metric':{'l2','l1'},
    'num_leaves': 31,
    'learning_rate': 0.2,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'verbose':0
}
print('Training....')
gbm_new =  lgb.train(params, lgb_train_big, num_boost_round=20,
     valid_sets=lgb_test_big, callbacks=[lgb.early_stopping(
    stopping_rounds=5)])
print('Saving Model....')
gbm_new.save_model('improved_model.txt')
print('Running Inference....')
pred_y = gbm_new.predict(test_df, num_iteration=gbm.
    best_iteration)
```

Out[26]:
```
Training....
[LightGBM] [Warning] Found whitespace in feature_names, replace
    with underlines
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the
    overhead of testing was 0.023913 seconds.
You can set ‘force_col_wise=true‘ to remove the overhead.
[LightGBM] [Warning] Found whitespace in feature_names, replace
    with underlines
Training until validation scores don’t improve for 5 rounds
Did not meet early stopping. Best iteration is:
[20]  valid_0’s l2: 0.0028245 valid_0’s l1: 0.0402918
Saving Model....
Running Inference....
```

In [27]:
```python
test_y = np.array(test_y)
pred_y = np.array(pred_y)
mape_lgbm_out_of_sample = mean_absolute_percentage_error(np.exp
    (test_y), np.exp(pred_y))#np.mean(np.abs((y_test - y_pred)/
    y_test)*100)
mae_lgbm_out_of_sample = mean_absolute_error(np.exp(test_y), np
    .exp(pred_y))
rmse_test = mean_squared_error(np.exp(test_y), np.exp(pred_y))
    **0.5
print(f'RMSE of prediction using LightGBM regressor is: {
    rmse_test}')
print(f'Out of sample MAPE: {mape_lgbm_out_of_sample*100}%')
print(f'Out of sample MAE: {mae_lgbm_out_of_sample}')
```

Out[27]:
```
RMSE of prediction using LightGBM regressor is: 5086.112686575689
Out of sample MAPE: 4.034659257763l295%
Out of sample MAE: 3754.7286648229365
```