Environment Set-Up

Load relevant Python Packages

In [1]:
```
reset -fs
```

In [2]:
```
# Importing the most important modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import pickle
import time
from tqdm.notebook import tqdm

# Import plotly modules to view time series in a more interactive way
import plotly.graph_objects as go
import plotly.offline as pyo
from matplotlib.pyplot import cm
from IPython.display import Image

# Importing time series split for cross validation of time series models
from sklearn.model_selection import TimeSeriesSplit

# For Data Mining
import os, glob
from pandas import read_csv

# For Data Cleaning
from datetime import datetime
import missingno as msno

from matplotlib import pyplot
import matplotlib.dates as mdates

# Importing metrics to evaluate the implemented models
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Importing fbprophet for Prophet Model
from fbprophet import Prophet
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-2-c24ed60a7eb2> in <module>
     33 
     34 # Importing fbprophet for Prophet Model
---> 35 from fbprophet import Prophet

ModuleNotFoundError: No module named 'fbprophet'
```

Global Variables and Settings

In [3]:
```
# Setting the random seed for reproducability and several plotting style parameters
%matplotlib inline
plt.style.use('seaborn')
pyo.init_notebook_mode()
sns.set(rc={'figure.figsize':(14,8)})
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
RSEED = 42
```

Load Data

In [4]:
```
#data has been saved using a .pkl file.
path = './data/df_small.pkl'
df = pd.read_pickle(path)
df.head(2)
```

Out[4]:

	power_available_mw_obsnorm	target_losses_norm	lagged_NetConsumption_MW	lagged_energyprice_euro_MWh	dswrf_sfc_wm2
2018-01-01 06:00:00	0.911849	0.425598	3142.133333	-71.616667	0.0
2018-01-01 06:10:00	0.932739	0.404513	3144.800000	-72.540000	0.0

Global Variable (Starting points of days to test models on)

In [5]:
```
test_timestamps = []
for i in range (10):
    test_timestamps.append(pd.to_datetime(df.index[-1]) - (i+1)*pd.Timedelta(hours=24))
test_timestamps.sort()

val_timestamps = [pd.to_datetime("2019-03-17 06:00:00")]
for i in range (9):
    val_timestamps.append(pd.to_datetime(val_timestamps[0]) + (i+1)*pd.Timedelta(hours=24))
val_timestamps.sort()
```

General Functions

Error Metrics Function (RMSE, R2, MAE, MAPE)

In [6]:
```
def error_metrics(y_pred, y_truth, model_name = "default"):
    """
    Calculate error metrics for a single comparison between predicted and observed values
    """
    # calculating error metrics
    RMSE_return = np.sqrt(mean_squared_error(y_truth, y_pred))
    R2_return = r2_score(y_truth, y_pred)
    MAE_return = mean_absolute_error(y_truth, y_pred)
    MAPE_return = (np.mean(np.abs((y_truth - y_pred) / y_truth)) * 100)

    # saving error metrics in a dataframe and returning it
    name_error = ['RMSE', 'R2', 'MAE', 'MAPE']
    value_error = [RMSE_return, R2_return, MAE_return, MAPE_return/100]
    dict_error = dict()
    for i in range(len(name_error)):
        dict_error[name_error[i]] = [value_error[i]]
    errors = pd.DataFrame(dict_error).T
    errors.rename(columns={0 : model_name}, inplace = True)

    #path = './data/error_metrics_{}.pkl'.format(model_name)
    #errors.to_pickle(path)

    return(errors)
```

FB Prophet Multistep Prediction

Defining Functions for multi-step forecast with Prophet Model and a rolling training window

In [7]:
```
def stan_init(m):
    """Retrieve parameters from a trained model.

    Retrieve parameters from a trained model in the format
    used to initialize a new Stan model.

    Parameters
    ----------
    m: A trained model of the Prophet class.

    Returns
    -------
    A Dictionary containing retrieved parameters of m.

    """
    res = {}
    for pname in ['k', 'm', 'sigma_obs']:
        res[pname] = m.params[pname][0][0]
    for pname in ['delta', 'beta']:
        res[pname] = m.params[pname][0]
    return res
```

In [8]:
```
def rolling_prophet_model(data, tfstart, prediction_window_size_hrs = 24, train_window_size_days = 90,
                          timesteps = 18, lags = 1, logtransformation = True, target_name = "target_los
ses_norm"):
    """
        Predict values with a Prophet Model for a chosen timespan with a rolling-forward training windo
w of a
        chosen size (differenced time series will be predicted).
        - data: input dataframe
        - tfstart: start timestamp of the timespan to predict for
        - prediction_window_size_hrs: size of the prediction window in hours
        - train_window_size_days: size of the training window in days
        - timesteps: number of timesteps that will be predicted ahead on each step
        - lags: number of lags of the target_variable that should be included in the dataframe
        - logtransformation: should the target variable be transformed with the log-function for the pr
ediction
        - target_name: column name of the target variable

    """
    #creating a working data frame to not change the actual input dataframe
    workframe = data.copy(deep = True)

    #if logtransformation is wanted
    if logtransformation == True:
        workframe[target_name] = np.log(workframe[target_name])

    #if lags should be included, they will be generated
    for i in range(lags):
        workframe[f"lag{i+1}"] = workframe[target_name].shift(i+1)

    #nan values after creation of lags will be dropped
    workframe.dropna(inplace = True)

    #creating another copy to keep the undifferenced values for backtransformation
    workframe_real = workframe.copy(deep = True)

    #calculating the differenced values for the target column
    workframe[target_name] = workframe[target_name].diff(1)

    #calculating the differenced values for the included lags
    if lags >= 1:
        for i in range(lags):
            workframe[f"lag{i+1}"] = workframe[f"lag{i+1}"].diff(1)

    #nan values after creation of lags will be dropped
    workframe.dropna(inplace = True)

    #setting start point of initial training window dependent on training window size
    train_start = pd.to_datetime(tfstart) - pd.Timedelta(days = train_window_size_days)

    #setting end point of test set dependent on chosen prediction window size
    tfend = pd.to_datetime(tfstart) + pd.Timedelta(hours = prediction_window_size_hrs)

    #making working dataframe compatible with fbprophet
    workframe.rename(columns={target_name: "y"}, inplace = True)

    #splitting data in train and test
    df_test = workframe[(workframe.index >= tfstart) & (workframe.index <= tfend)]
    df_train = workframe[(workframe.index >= train_start) & (workframe.index < tfstart)]

    #making the copy with the undifferenced target values compatible with prophet
    workframe_real.rename(columns={target_name: "y"}, inplace = True)

    #creating copy of the undifferenced test data for later evalutation against predictions
    y_test = list()
    for i in range(timesteps):
        y_test.append(workframe_real[(workframe_real.index >= tfstart) & (workframe_real.index <= tfend
)]["y"].shift(-i).iloc[:-timesteps])

    #saving all the additional regressors (not the target) in list
    regressors = list(df_train.columns)
    regressors.remove("y")

    #adding datestamps to dataframes for compatibility with fbprophet
    df_test["ds"] = df_test.index
    df_train["ds"] = df_train.index

    # setting up a list to store the prediction results in
    predictions = list()

    #iterating over the test set
    for t in tqdm(range(len(df_test)-timesteps)):

            #initializing new Prophet model
            model = Prophet(yearly_seasonality = False)

            #adding all the regressors with the same hyperparameters
            for name in list(regressors):
                model.add_regressor(name, prior_scale = 1, standardize = True, mode='multiplicative')


            #training the model on the current training dataframe with the saved initial parameters fro
m the last model, if there was one
            try:
                model.fit(df_train, init = parameters);
            except NameError:
                model.fit(df_train);

            #saving the parameters of the fitted model for warm-start training of the next model
            parameters = stan_init(model)

            #the timestamp before the current prediciton timestep is calculated
            index_before = df_test.index[0] - pd.Timedelta(minutes = 10)

            #setting up future dataframe (two steps ahead) with all regressors filled in assumption of
 perfect forecast for regressors
            future = df_test.drop(columns = ["y"]).iloc[0:timesteps] #.to_frame().T

            #predicting next timestep
            forecast = model.predict(future)

            predictions_inner_list = list()

            # setting the physically possible boundaries of the predictions (must be between 0 and 1 af
ter backtransformation) depending on the chosen transformations
            for i in range(timesteps):
                if (logtransformation == True):
                    if forecast["yhat"].iloc[0:i].sum() + workframe_real.loc[index_before]["y"] < -30:
                         predictions_inner_list.append(-30)
                    elif forecast["yhat"].iloc[0:i].sum() + workframe_real.loc[index_before]["y"] >= 0:
                         predictions_inner_list.append(0)
                    else:
                         predictions_inner_list.append(forecast["yhat"].iloc[0:i].sum() + workframe_rea
l.loc[index_before]["y"])

                else:
                    if forecast["yhat"].iloc[0:i].sum() + workframe_real.loc[index_before]["y"] < 0:
                        predictions_inner_list.append(0)
                    elif forecast["yhat"].iloc[0:i].sum() + workframe_real.loc[index_before]["y"] >= 1:
                        predictions_inner_list.append(1)
                    else:
                        predictions_inner_list.append(forecast["yhat"].iloc[0:i].sum() + workframe_real
.loc[index_before]["y"])

            predictions.append(predictions_inner_list)

            #dropping the left end point of the training dataframe
            df_train.drop(df_train.index[0], inplace = True)

            #appending the left end point of the test dataframe to the training dataframe
            df_train = df_train.append(df_test.iloc[0].to_frame().T)

            #dropping the left end point of the test dataframe
            df_test.drop(df_test.index[0], inplace = True)

    columnnames = list()
    testcolumnnames = list()

    for i in range(timesteps):
        columnnames.append(f"y_pred{i+1}")
        testcolumnnames.append(f"y_test{i+1}")

    results = pd.DataFrame(columns = columnnames)
    original = pd.DataFrame(columns = testcolumnnames)

    for i in range(timesteps):
        results[f"y_pred{i+1}"] = pd.Series(v for v in [el[i] for el in predictions])
        original[f"y_test{i+1}"] = y_test[i]


    #setting the indices as they were
    results.index = y_test[0].index

    #backtransformation to real values if logtransformation was used
    if logtransformation == True:
        results = np.exp(results)
        original = np.exp(original)

    ##creating the dataframe that will be saved as a file
    #results.to_csv(f".data/{filename}_predictions.csv")
    #original.to_csv(f".data/{filename}_test.csv")
    #print(f"Predictions and test values saved.")

    #returning the dataframes with the results
    return results, original
```

Tuning Prophet Model on Validation Data

In [9]:
```
y_pred, y_test = rolling_prophet_model(data = df, tfstart = val_timestamps[0], prediction_window_size_h
rs = 240,
                                       train_window_size_days = 60, timesteps = 18, lags = 2, logtransformatio
n = True,
                                       target_name = "target_losses_norm")
```

```
INFO:numexpr.utils:NumExpr defaulting to 4 threads.
```

In [10]:
```
columnnames = list()
for i in range(18):
    columnnames.append(f"FB Prophet Prediction Step {i+1}")

val_errors = pd.DataFrame(columns = columnnames)

for i in range(18):
    val_errors[f"FB Prophet Prediction Step {i+1}"]  = error_metrics(y_pred[f"y_pred{i+1}"], y_test[f"y_
test{i+1}"])["default"]

val_errors = val_errors.T
```

In [11]:
```
val_errors
```

Out[11]:

	RMSE	R2	MAE	MAPE
FB Prophet Prediction Step 1	0.015623	0.990346	0.007050	0.146192
FB Prophet Prediction Step 2	0.022934	0.979131	0.010823	0.212832
FB Prophet Prediction Step 3	0.027418	0.970066	0.013622	0.266374
FB Prophet Prediction Step 4	0.030796	0.962107	0.016033	0.319795
FB Prophet Prediction Step 5	0.033974	0.953746	0.018232	0.369090
FB Prophet Prediction Step 6	0.037599	0.943306	0.020578	0.421615
FB Prophet Prediction Step 7	0.041246	0.931739	0.022789	0.470884
FB Prophet Prediction Step 8	0.044641	0.919920	0.024981	0.515289
FB Prophet Prediction Step 9	0.048089	0.906950	0.027179	0.556268
FB Prophet Prediction Step 10	0.051410	0.893506	0.029248	0.595890
FB Prophet Prediction Step 11	0.054440	0.880519	0.031239	0.629059
FB Prophet Prediction Step 12	0.056910	0.869383	0.032937	0.661596
FB Prophet Prediction Step 13	0.059483	0.857254	0.034798	0.702241
FB Prophet Prediction Step 14	0.062182	0.843938	0.036535	0.736962
FB Prophet Prediction Step 15	0.064746	0.830714	0.038228	0.778356
FB Prophet Prediction Step 16	0.066947	0.818849	0.039628	0.814026
FB Prophet Prediction Step 17	0.068966	0.807527	0.041197	0.851699
FB Prophet Prediction Step 18	0.071453	0.793185	0.042832	0.893329

Tuned Prophet Model on Test Data

In [12]:
```
y_pred, y_test = rolling_prophet_model(data = df, tfstart = test_timestamps[0], prediction_window_size_
hrs = 240,
                                       train_window_size_days = 60, timesteps = 18, lags = 2, logtransformatio
n = True,
                                       target_name = "target_losses_norm")
```

In [13]:
```
columnnames = list()
for i in range(18):
    columnnames.append(f"FB Prophet Prediction Step {i+1}")

test_errors = pd.DataFrame(columns = columnnames)

for i in range(18):
    test_errors[f"FB Prophet Prediction Step {i+1}"]  = error_metrics(y_pred[f"y_pred{i+1}"], y_test[f"y
_test{i+1}"])["default"]

test_errors = test_errors.T
```

In [14]:
```
test_errors
```

Out[14]:

	RMSE	R2	MAE	MAPE
FB Prophet Prediction Step 1	0.012406	0.994166	0.004951	0.111856
FB Prophet Prediction Step 2	0.017815	0.987968	0.007396	0.153848
FB Prophet Prediction Step 3	0.021651	0.982228	0.009409	0.185931
FB Prophet Prediction Step 4	0.025778	0.974805	0.011295	0.222630
FB Prophet Prediction Step 5	0.029558	0.966874	0.013196	0.254354
FB Prophet Prediction Step 6	0.033122	0.958402	0.014911	0.283085
FB Prophet Prediction Step 7	0.036721	0.948873	0.016627	0.313766
FB Prophet Prediction Step 8	0.040103	0.939021	0.018256	0.346486
FB Prophet Prediction Step 9	0.043228	0.929148	0.019644	0.372691
FB Prophet Prediction Step 10	0.046479	0.918086	0.021120	0.397004
FB Prophet Prediction Step 11	0.049676	0.906427	0.022565	0.418086
FB Prophet Prediction Step 12	0.052770	0.894404	0.023978	0.438107
FB Prophet Prediction Step 13	0.055880	0.881588	0.025415	0.460057
FB Prophet Prediction Step 14	0.058870	0.868573	0.026880	0.484461
FB Prophet Prediction Step 15	0.061907	0.854650	0.028253	0.506448
FB Prophet Prediction Step 16	0.064871	0.840392	0.029685	0.526513
FB Prophet Prediction Step 17	0.067645	0.826434	0.031055	0.544764
FB Prophet Prediction Step 18	0.070381	0.812095	0.032314	0.561554

In [ ]:
```
val_errors.to_csv("./Validation Errors/validation_errors_fbprophet.csv", index_label = "Step")
test_errors.to_csv("./Test Errors/test_errors_fbprophet.csv", index_label = "Step")
```

In [3]:
```
print('This cell was last run on: ')
print(datetime.now())
```

```
This cell was last run on: 
2020-11-26 12:55:22.076171
```