SOFTWARE USER MANUAL

Version 1.0

Table of Contents

This document describes the User Guide for the OpenBats simulation tool. The tool in its current stage of development can optimize a building load to reduce the cost to the consumer by orchestrating use of solar and storage to offset high price segments of a given day. In addition to this optimization, the tool can also add additional optimization methods easily. The optimization procedure needs to be added as a function in the optimization code and the user interface needs to be updated with that option in its list, and then updated to make an appropriate call to that function when the user selects it in the user interface. All the code for the OpenBats tool, including the optimization and user interface application are all built using python.

Architecture

The architecture of the OpenBats simulation tool is primarily to be able to run various optimization algorithms and then once the simulation of the systems has been executed, a user could apply those methods to an actual building. The different steps of the algorithm are:

A. Collect required data for control and optimization from a building for period of interest (summer, autumn, winter, spring and iterating these over weekday and weekend days)
B. Clean and format the data for AI data modeling
C. Model the load parameters as a function of weather, system parameters and building constraints.
D. Optimization
    a. Select the days to apply the optimization (currently fixed in the optimization source code)
    b. Select the original data file where no optimization has been applied for the selected days
    c. Select the input and output variables
    d. Select the model built earlier based on the data collected
    e. Select the optimization methodology to apply
    f. Run the optimization
    g. Plot data to understand the optimization results

Using the OpenBats simulation tool

To launch the tool, one needs to have python 3.7.12 installed. To start the tool, ensure that relevant python version is installed in the computer (tested on a window 10 machine). Then open a command window (cmd.exe), navigate to the folder that contains the source code for the OpenBats Simulation tool (openbats_v10.py) and execute the script as shown below.

```
C:\Users\user1\openbats_tool> python openbats_v10.py
```

where the location of the OpenBats tool code is present in the directory "C:\Users\user1\openbats_tool" directory.

When the code executes, the following user interface is launched.

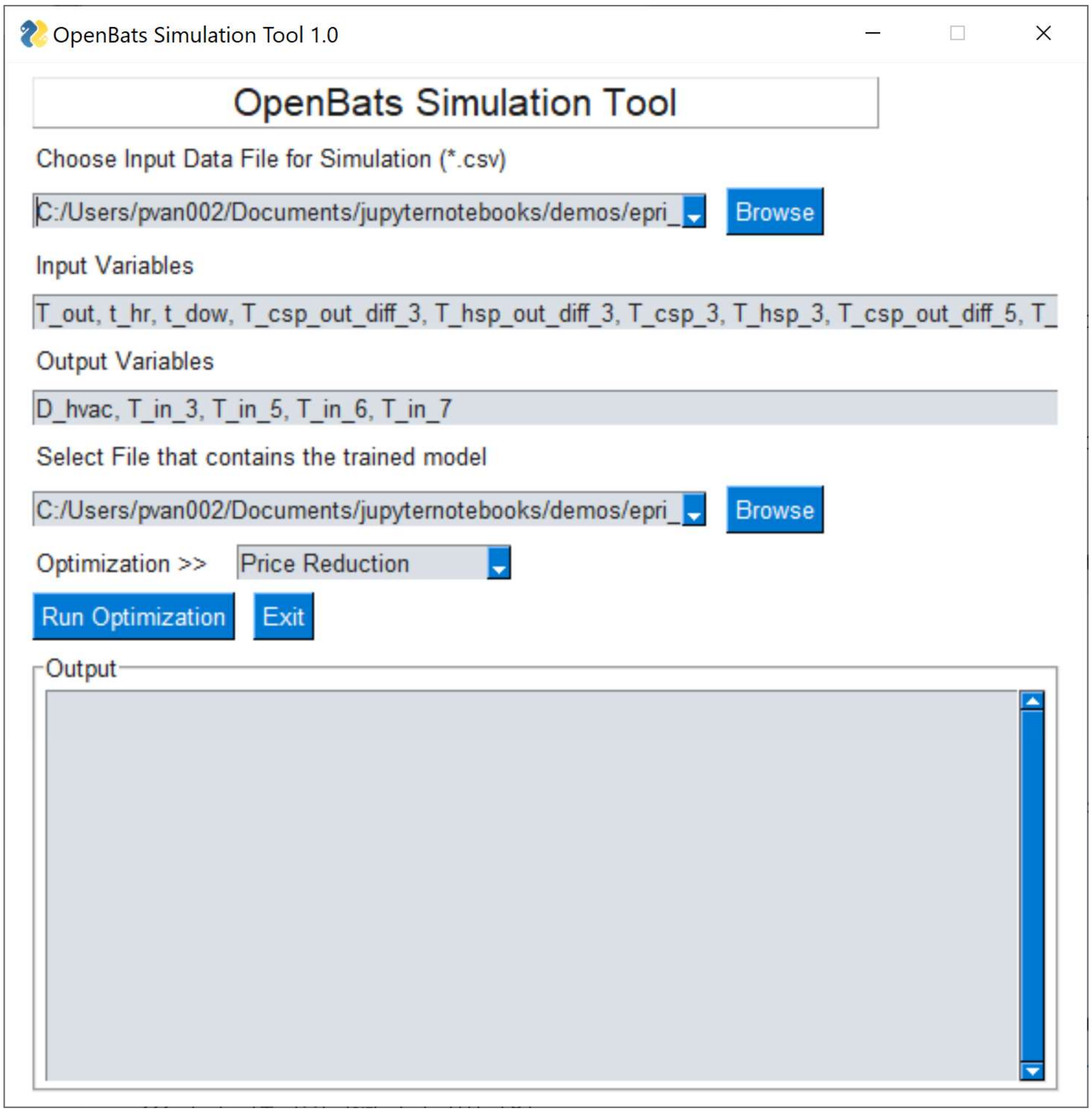

Figure 1: OpenBats Simulation Tool User Interface

The methodology of running the optimization simulation using OpenBats 1.0 will be described next.

As a first step, click on the browse button left of indicator (1) in the **Figure 2** below to select the data file days the optimization simulation is being run.

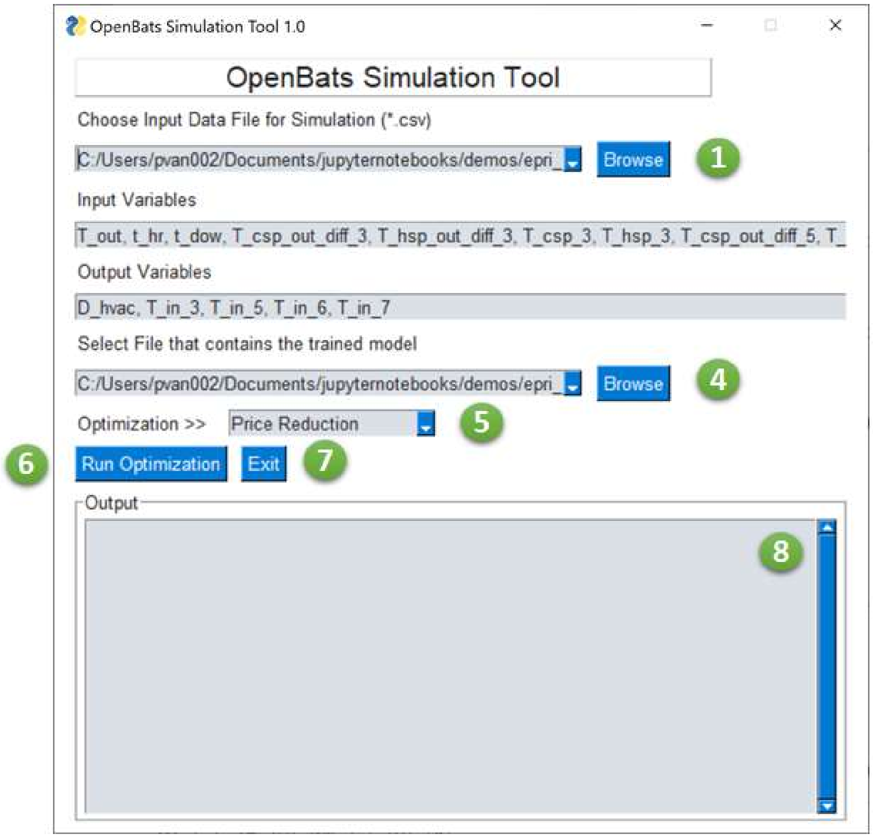

Figure 2: Screenshot of the OpenBats tool with user interface element indicators

On clicking the "Browse" button, the file select file dialog pops up as shown in Figure 3 below.

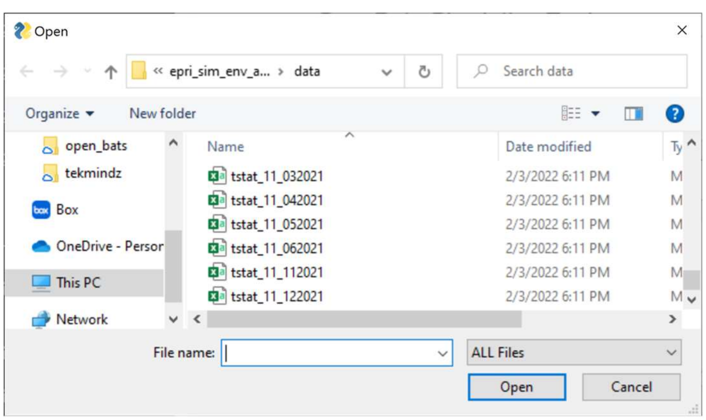

Figure 3: Screenshot of the Select File Dialog

The data file is expected to be a “CSV” file (that is format is a comma separated values file with first line containing the signal names. After selecting the data file, the next step is enter the input variables in indicator (2) of Figure 2, and output variables for the model outputs in indicator (3) of Figure 2. The user needs to ensure that the names of the variables match the values in the data “CSV” file included in the first step.

In the next step click the “Browse” button left of the indicator (4) in Figure 2 to select the model file that built using the original data collected and trained using an AI model.

In the next step, select the Optimization algorithm from the drop down list of the combo box to the left of the indicator (5) in Figure 2. Finally click on the button “Run Optimization” to the right of indicator (6) in Figure 2.

The output data emitted by the optimization will appear on the “Output” pane shown by the indicator (8) in Figure 2.

Any data plots that the optimization generates will be open in a browser after the code execution is complete.

Sample output for the price reduction optimization for the BSA building is shown below.

```
Run Optimization
Running Optimization
 xvars = ['T_out', 't_hr', 't_dow', 'T_csp_out_diff_3', 'T_hsp_out_diff_3', 'T_csp_3', 'T_hsp_3',
'T_csp_out_diff_5', 'T_hsp_out_diff_5', 'T_csp_5', 'T_hsp_5', 'T_csp_out_diff_7', 'T_hsp_out_diff_7',
'T_csp_7', 'T_hsp_7', 'T_csp_out_diff_6', 'T_hsp_out_diff_6', 'T_csp_6', 'T_hsp_6']
 yvars = ['D_hvac', 'T_in_3', 'T_in_5', 'T_in_6', 'T_in_7']
c:\Users\pvan002\Anaconda3\envs\epri_sim\lib\site-packages\pandas\core\generic.py:4150:
PerformanceWarning:

dropping on a non-lexsorted multi-index without a level parameter may impact performance.
```

```
completed sim for the  2021-05-11
completed sim for the  2021-05-12
completed sim for the  2021-05-13
completed sim for the  2021-05-14
completed sim for the  2021-05-15
completed sim for the  2021-05-16
completed sim for the  2021-05-17
completed sim for the  2021-05-18
completed sim for the  2021-05-19
completed sim for the  2021-05-20
completed sim for the  2021-05-21
completed sim for the  2021-05-22
completed sim for the  2021-05-23
completed sim for the  2021-05-24
completed sim for the  2021-05-25
completed sim for the  2021-05-26
completed sim for the  2021-05-27
completed sim for the  2021-05-28
completed sim for the  2021-05-29



Baseline (based on computed HVAC for simulated range)
          Date    Weekday  import_cost  export_cost
0   05/11/2021    Tuesday    16.236628   -25.026564
1   05/12/2021  Wednesday    15.458936   -27.145193
2   05/13/2021   Thursday    15.112718   -27.356455
3   05/14/2021     Friday    15.792430   -16.264383
4   05/15/2021   Saturday    13.342127   -19.254181
5   05/16/2021     Sunday    10.963797   -29.344381
6   05/17/2021     Monday    11.298342   -17.949223
7   05/18/2021    Tuesday    13.100645   -27.759899
8   05/19/2021  Wednesday    11.701935   -35.970223
9   05/20/2021   Thursday    12.074597   -33.895783
10  05/21/2021     Friday    11.813847   -31.985740
11  05/22/2021   Saturday     9.996693   -41.731843
12  05/23/2021     Sunday    10.408633   -30.970331
13  05/24/2021     Monday    13.679155   -29.309328
14  05/25/2021    Tuesday    14.061656   -30.027576
15  05/26/2021  Wednesday    12.038411   -29.587899
16  05/27/2021   Thursday    13.193805   -31.924031
17  05/28/2021     Friday    12.164756   -29.902709
```

```
18  05/29/2021   Saturday    13.200907    -8.388656
peak kw =  11.685336198709786
import -> $ 245.6400197022997   export -> $ -523.7943987581718
total bill -> $ -46.200455511482915




Simulation: (based on computed HVAC for simulated range)
         Date    Weekday  import_cost  export_cost
0   05/11/2021    Tuesday    16.172574   -25.705572
1   05/12/2021  Wednesday    14.201711   -26.613153
2   05/13/2021   Thursday    13.304146   -25.970389
3   05/14/2021     Friday    15.488851   -16.943255
4   05/15/2021   Saturday    11.164372   -17.257745
5   05/16/2021     Sunday     9.669647   -26.310466
6   05/17/2021     Monday    14.998102   -21.861017
7   05/18/2021    Tuesday    14.871845   -31.082817
8   05/19/2021  Wednesday    12.470899   -38.469210
9   05/20/2021   Thursday    12.873106   -35.946567
10  05/21/2021     Friday    12.368339   -33.924502
11  05/22/2021   Saturday     9.324630   -42.039537
12  05/23/2021     Sunday     9.106281   -30.859928
13  05/24/2021     Monday    13.613184   -29.976644
14  05/25/2021    Tuesday    14.637375   -31.487786
15  05/26/2021  Wednesday    11.667069   -30.202120
16  05/27/2021   Thursday    12.519144   -31.986427
17  05/28/2021     Friday    12.040373   -30.966868
18  05/29/2021   Saturday    11.878603    -7.707987
peak kw =  10.873529249878771
import -> $ 242.3702506170079   export -> $ -535.3119915138292
total bill -> $ -77.1021852867276
Optimization done... Plotting data. Data will open up in a browser
Optimization done
```

Sample output generated for the price reduction optimization for the BSA building is shown in Figure 4 below.

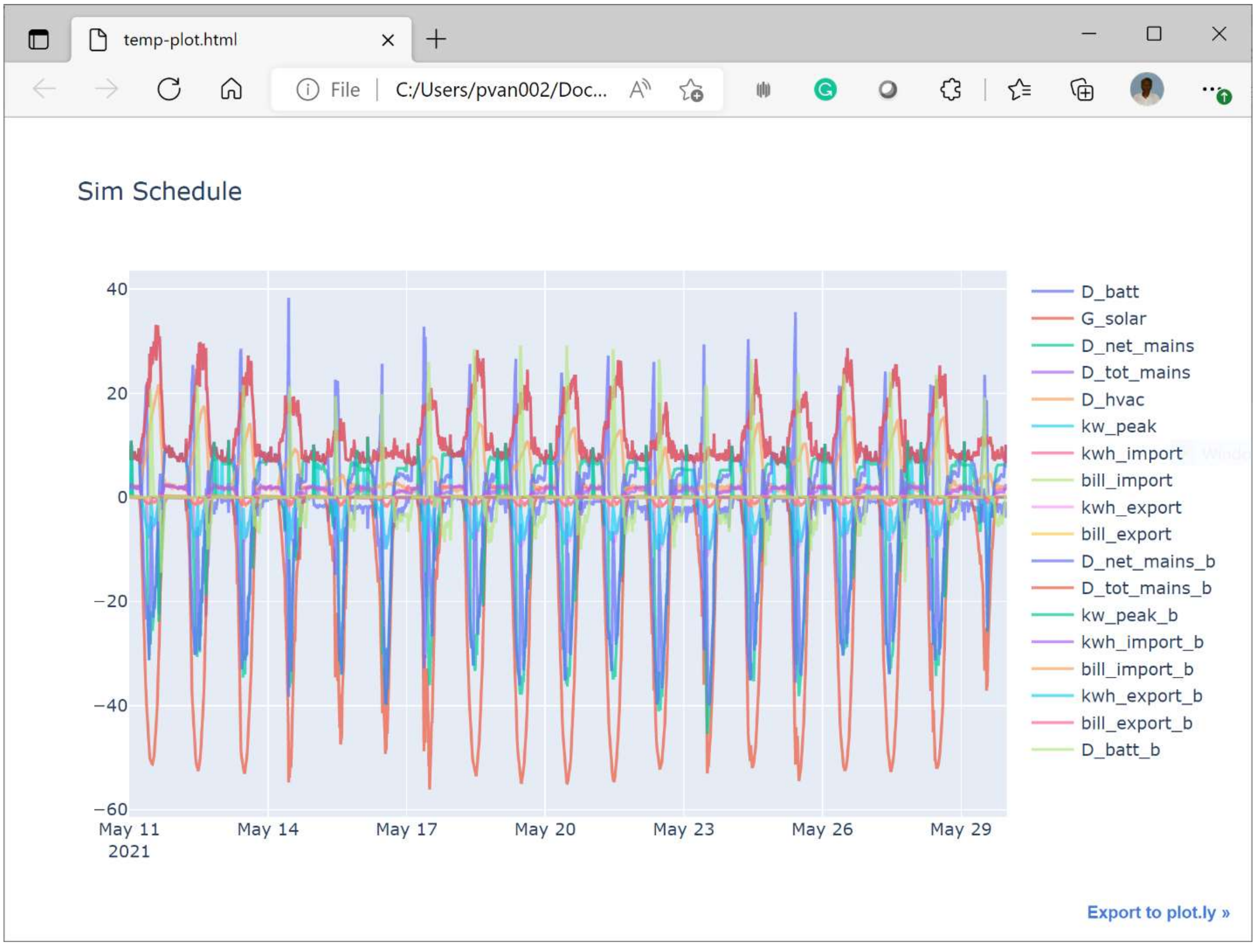

Figure 4: Data plot output as a result of the optimization execution in the local browser window

Source Code for the User interface

The source code for the OpenBats interface is coded in the file openbats_v10.py ("10" indicates a versioning system to keep track of changes).

The python code for the user interface is listed below.

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 24 17:19:08 2022

@author: Viswanath
"""
```

```
# openbats GUI to run the simulation in the background

import threading
import time
import os.path
import PySimpleGUI as sg

import simulate_v3gui as sim_ob


def simulation_thread(window, model_file = None, data_file = None):
    if model_file is None or data_file is None:
        print(f'model file => {model_file}')
        print(f'data file => {data_file}')
        print("both data file and model file are required for optimizaton")
        window.write_event_value('-SIMULATION FAILED-', "")
    else:
        sim_ob.simulate_from_gui(file_model = model_file, data_in_file = data_file)
        window.write_event_value('-SIMULATION DONE-', '')

def do_simulation(model_file = None, data_file = None):
    threading.Thread(target=simulation_thread, args=(window, model_file, data_file),
daemon=True).start()


def make_window(theme):
    sg.theme(theme)

    layout_sim = [
        [sg.Text("OpenBats Simulation Tool",size=(38, 1), justification='center',
font=("Helvetica", 16), relief=sg.RELIEF_RIDGE, k='-TEXT HEADING-',
enable_events=True)],
        [sg.Text("Choose Input Data File for Simulation (*.csv)")],
        [sg.Combo(sg.user_settings_get_entry('-filenames-data-', []),
default_value=sg.user_settings_get_entry('-last filename-data-', ''), size=(50, 1),
key='-FILENAME-DATA-'),
         sg.FileBrowse()],
        [sg.Text("Input Variables")],
        [sg.Input("T_out, t_hr, t_dow, T_csp_out_diff_3, T_hsp_out_diff_3, T_csp_3,
T_hsp_3, T_csp_out_diff_5, T_hsp_out_diff_5, T_csp_5, T_hsp_5, T_csp_out_diff_7,
T_hsp_out_diff_7, T_csp_7, T_hsp_7, T_csp_out_diff_6, T_hsp_out_diff_6, T_csp_6,
T_hsp_6",
                  key='-input-vars-', expand_x = True, expand_y = True)],
        [sg.Text('Output Variables')],
        [sg.Input("D_hvac, T_in_3, T_in_5, T_in_6, T_in_7", key='-output-vars-',
expand_x = True, expand_y = True)],
        [sg.Text("Select File that contains the trained model")],
        [sg.Combo(sg.user_settings_get_entry('-filenames-model-', []),
default_value=sg.user_settings_get_entry('-last filename-model-', ''), size=(50, 1),
key='-FILENAME-MODEL-'),
```

```
         sg.FileBrowse()],
        [sg.Text("Optimization >> "),
             sg.Combo(values=('Price Reduction', 'Reduce Peak Energy'),
default_value='Price Reduction',
             readonly=True, k='-OPT-COMBO-')],
        [sg.Button("Run Optimization"), sg.Button('Exit')],
        [sg.Frame("Output", [[sg.Multiline(size=(75,15), font='Courier 8',
expand_x=True, expand_y=True, write_only=True,
                                    reroute_stdout=True, reroute_stderr=True,
echo_stdout_stderr=True, autoscroll=True, auto_refresh=True)]])]
        ]

    window = sg.Window('OpenBats Simulation Tool 1.0', layout_sim, finalize=True,
keep_on_top=True)
    return window

sg.theme('LightGrey1')
window = make_window(sg.theme())

while True:
    event, values = window.read()
    print(event)
    print(values)
    print(type(event))
    if event == sg.WIN_CLOSED or event == 'Exit':
        break
    elif event is None:
        break
    elif event == 'Run Optimization':
        print ("Running Optimization")
        dat_file = values['-FILENAME-DATA-']
        mod_file = values['-FILENAME-MODEL-']
        tx_vars = values['-input-vars-']
        x_vars= tx_vars.split(',') if tx_vars else []
        x_vars = [s.strip() for s in x_vars]
        ty_vars = values['-output-vars-']
        y_vars = ty_vars.split(',') if ty_vars else []
        y_vars = [s.strip() for s in y_vars]
        print (f' xvars = {x_vars}')
        print (f' yvars = {y_vars}')
        if dat_file is None or mod_file is None or len(dat_file) == 0 or
len(mod_file) == 0:
            print("Cannot do optimizaton, both data and model file required")
            print(f'model file => {mod_file}')
            print(f'data file => {dat_file}')
        else:
            sg.user_settings_set_entry('-filenames-data-',
list(set(sg.user_settings_get_entry('-filenames-data-', []) + [values['-FILENAME-
DATA-'], ])))
            sg.user_settings_set_entry('-last filename-data-', values['-FILENAME-
DATA-'])
```

```
            sg.user_settings_set_entry('-filenames-model-', 
list(set(sg.user_settings_get_entry('-filenames-model-', []) + [values['-FILENAME-
MODEL-'], ])))
            sg.user_settings_set_entry('-last filename-model-', values['-FILENAME-
MODEL-'])
            do_simulation(mod_file, dat_file)
    elif event == '-SIMULATION DONE-':
        print ("Optimization done")


window.close()
```

Source Code for the Optimization and Data Modeling

The source code for Optimization and data modeling is split into 3 files:

1. **darts_model.py** : This file creates the AI based model based on the data, and it uses Darts package to identify and generate the AI model
2. **simulate.py**: This file contains the code that runs the optimization function, and all the necessary related functions are also present in the same file
3. **run_simulate.py**: This is a wrapper file in python which allow the user interface to execute the optimization functions defined in "simulate.py".

The source code for "darts_model.py" is listed below:

```
from datetime import timedelta
from lzma import CHECK_UNKNOWN
#from sqlite3 import Timestamp
import pandas as pd
import numpy as np
import os
import calendar

import plotly.graph_objs as go
import plotly.offline as pyo
import plotly.express as px
from plotly.subplots import make_subplots
import matplotlib.pyplot as plt

from darts import TimeSeries
from darts.models import NBEATSModel
from darts.models import RegressionModel
from darts.models import RNNModel
from darts.models import BlockRNNModel
from darts.dataprocessing.transformers import Scaler
```

```
import torch

class GLOBS():
    filepath_in = "./data/df_historical_20210101-20210601.csv"
    filepath_tstat_alt = "./data/test_data/darts_counterfactual_tstat.csv"
    # TODO: Add multiple sample series inputs to model fitting (non-contiguous time
series)
    start_date = "2021-04-20" # TODO: Add Winter Months to training data (or warmer
summer...?) (DONE)
    end_date = "2021-05-30"
    train_val_split_date = "2021-05-20"
    #n_val_samples = 480 # Deprecated
    train_frac = 0.8
    n_epochs = 50
    forecast_horizon = 24*4
    pred_dates = [
        "2021-05-21", "2021-05-22", "2021-05-23", "2021-05-24",
        "2021-05-25", "2021-05-26", "2021-05-27"
    ]
    #pred_dates = [
    #    "2021-05-01", "2021-05-02", "2021-05-03", "2021-05-04",
    #    "2021-05-05", "2021-05-06", "2021-05-07", "2021-05-08"
    #]
    training_pred_dates = [
        "2021-05-14", "2021-05-15", "2021-05-16", "2021-05-17",
        "2021-05-18", "2021-05-19", "2021-05-20"
    ]
    n_rnn_layers = 2

    #y_var = "D_hvac" # need to include the correlated targets as well (DONE)
    y_var = ["D_hvac", "T_in_3", "T_in_5", "T_in_6", "T_in_7"] # TODO: Try adding
zone 7 and D_hvac to the model targets. (DONE)
    # TODO: Try a RandomForestRegressor (or other future variates model), maybe will
overfit less than RNN
    X_vars = [
        "T_out", "t_hr", "t_dow", #"t_isweekday", #"G_solar", #"T_hd65", "T_cd65",
        "T_csp_out_diff_3", "T_hsp_out_diff_3", "T_csp_3", "T_hsp_3",
        "T_csp_out_diff_5", "T_hsp_out_diff_5", "T_csp_5", "T_hsp_5", #TODO: Figure
out how to account for the non-SP conforming behavior (DONE)
        "T_csp_out_diff_7", "T_hsp_out_diff_7", "T_csp_7", "T_hsp_7",
        "T_csp_out_diff_6", "T_hsp_out_diff_6", "T_csp_6", "T_hsp_6" # no setpoint
changes occur in 2021
    ]
    # TODO: Write code for counterfactual thermostat schedules that imports a csv
table
    # that maps ZONE, HOUR, DAYOFWEEK, to a setpoint that will override the
historical
    # default value

class GLOBS_V2():
    filepath_in = "./data/df_historical_20210101-20210601.csv"
```

```
    filepath_tstat_alt = "./data/test_data/darts_counterfactual_tstat.csv"
    # TODO: Add multiple sample series inputs to model fitting (non-contiguous time
series)
    start_date = "2021-04-20" # TODO: Add Winter Months to training data (or warmer
summer...?) (DONE)
    end_date = "2021-05-30"
    train_val_split_date = "2021-05-20"
    #n_val_samples = 480 # Deprecated
    train_frac = 0.8
    n_epochs = 50
    forecast_horizon = 24*4
    pred_dates = [
        "2021-05-21", "2021-05-22", "2021-05-23", "2021-05-24",
        "2021-05-25", "2021-05-26", "2021-05-27"
    ]
    #pred_dates = [
    #    "2021-05-01", "2021-05-02", "2021-05-03", "2021-05-04",
    #    "2021-05-05", "2021-05-06", "2021-05-07", "2021-05-08"
    #]
    training_pred_dates = [
        "2021-05-14", "2021-05-15", "2021-05-16", "2021-05-17",
        "2021-05-18", "2021-05-19", "2021-05-20"
    ]
    n_rnn_layers = 2

    #y_var = "D_hvac" # need to include the correlated targets as well (DONE)
    y_var = ["D_hvac", "T_in_3", "T_in_5", "T_in_6", "T_in_7"] # TODO: Try adding
zone 7 and D_hvac to the model targets. (DONE)
    # TODO: Try a RandomForestRegressor (or other future variates model), maybe will
overfit less than RNN
    X_vars = [
        "T_out", "t_hr", "t_dow", #"t_isweekday", #"G_solar", #"T_hd65", "T_cd65",
        "T_csp_out_diff_3", "T_hsp_out_diff_3", "T_csp_3", "T_hsp_3",
        "T_csp_out_diff_5", "T_hsp_out_diff_5", "T_csp_5", "T_hsp_5", #TODO: Figure
out how to account for the non-SP conforming behavior (DONE)
        "T_csp_out_diff_7", "T_hsp_out_diff_7", "T_csp_7", "T_hsp_7",
        "T_csp_out_diff_6", "T_hsp_out_diff_6", "T_csp_6", "T_hsp_6" # no setpoint
changes occur in 2021
    ]
    # TODO: Write code for counterfactual thermostat schedules that imports a csv
table
    # that maps ZONE, HOUR, DAYOFWEEK, to a setpoint that will override the
historical
    # default value

    price_offpeak = 0.14181 # $/kWh
    price_onpeak = 0.22501 # $/kWh
    price_partpeak = 0.16988 #$kwh
    price_export = 0.18 # $/kwh
    price_peak = 19.85 # $/kW*month (demand charge)
    customer_charge = 4.59959 # charge / day according to bill
```

```
    price_arr = []
    price_arr.append(np.ones(34) * price_offpeak )  # 12:00 Am to 8:30 Am
    price_arr.append(np.ones(14) * price_partpeak ) # 8:30 AM to 12:00 PM
    price_arr.append(np.ones(24) * price_onpeak )   # 12:00 PM to 6:00 PM
    price_arr.append(np.ones(14) * price_partpeak ) # 6:00 PM to 9:30 PM
    price_arr.append(np.ones(10) * price_offpeak )  # 9:30 PM to 12:00 AM
    price_tou = dict(enumerate(price_arr))

    price_arr = []
    price_arr.extend( (np.ones(96) * price_offpeak).tolist() )  # 12:00 Am to 12:00
Am next day
    price_tou_hday = dict(enumerate(price_arr))



class Forecaster():
    def __init__(self, y_var, X_vars):
        self.y_var = y_var
        self.X_vars = X_vars
        self.n_thermal_zones = 11

    def import_data(self, filepath):
        self.df_raw = pd.read_csv(filepath)
        self.df_raw["Timestamp"] = pd.to_datetime(self.df_raw["Timestamp"])
        #self.df_raw["T_in_lag1_diff_3"] = s

    def import_trained_model(self, filepath):
        """
        Import trained model from prior torch_model_run *.pth.tar
        """
        self.model = torch.load(filepath)

    def prep_features(self, df):
        #self.df_raw["T_diff_3"] = self.df_raw["T_out"] - self.df_raw["T_csp_3"]
        #df = self.df_raw.copy()
                # update_indoor_temp features:
        df["t_isweekday"] = df["Timestamp"].dt.dayofweek < 5
        df["t_isweekday"] = df["t_isweekday"].map({True: 1, False: 0})

        # Add a non-flexible load feature for simulation
        df["D_nonflex"] = df["D_mains"] - df["G_solar"] - df["D_hvac"] -
df["D_battery_eg"]

        for zone in range(1,self.n_thermal_zones):
            # Outdoor-SP temp diff (this is not as good as indoor-sp diff, but this
feature is not cooupled to the target or the decision variable)
            df["T_csp_out_diff_{}".format(zone)] = (df["T_out"] -
df["T_csp_{}".format(zone)]).clip(lower=0)
            df["T_hsp_out_diff_{}".format(zone)] = (df["T_hsp_{}".format(zone)] -
df["T_out"]).clip(lower=0)
```

```
            # Indoor-SP temp diff
            df["T_csp_in_diff_{}".format(zone)] = (df["T_in_{}".format(zone)] -
df["T_csp_{}".format(zone)]).clip(lower=0)
            df["T_hsp_in_diff_{}".format(zone)] = (df["T_hsp_{}".format(zone)] -
df["T_in_{}".format(zone)]).clip(lower=0)

            # CD65 & HD65
            df["T_cd65".format(zone)] = (df["T_out"] - 65).clip(lower=0)
            df["T_hd65".format(zone)] = (65 - df["T_out"]).clip(lower=0)

            # Targets:
            df["T_in_lag1_diff_{}".format(zone)] = df["T_in_{}".format(zone)].diff()

            # One more feature...
            df["T_in_lag1_diff_lag1_{}".format(zone)] =
df["T_in_lag1_diff_{}".format(zone)].shift()
        #self.df_raw = df.copy()
        return df

    def prep_data(self, idx_col="Timestamp", start_date="2000-01-01", end_date="2050-
01-01", t_res="15min"):
        """
        Remove non-model variable columns
        """
        # Calculate the "fudged" setpoints to make training data more robust
        df = self.df_raw.copy()
        # If T_csp_* - T_in > 4 --> T_csp_* += uniform(-2,2)
        # If T_in - T_hsp_* > 4 --> T_hsp_* += uniform(-2,2)
        for z in range(1,self.n_thermal_zones):
            df["T_csp_{}".format(z)] += (df["T_csp_{}".format(z)]-
df["T_in_{}".format(z)] >= 1)*np.random.uniform(-0.5, 0.5, df.shape[0])
            df["T_hsp_{}".format(z)] += (df["T_in_{}".format(z)]-
df["T_hsp_{}".format(z)] >= 1)*np.random.uniform(-0.5, 0.5, df.shape[0])
            df["T_csp_{}".format(z)] += (df["T_csp_{}".format(z)]-
df["T_in_{}".format(z)] >= 3)*np.random.uniform(-2, 1, df.shape[0])
            df["T_hsp_{}".format(z)] += (df["T_in_{}".format(z)]-
df["T_hsp_{}".format(z)] >= 3)*np.random.uniform(-1, 2, df.shape[0])
            df["T_csp_{}".format(z)] += (df["T_csp_{}".format(z)]-
df["T_in_{}".format(z)] >= 7)*np.random.uniform(-5, 4, df.shape[0])
            df["T_hsp_{}".format(z)] += (df["T_in_{}".format(z)]-
df["T_hsp_{}".format(z)] >= 7)*np.random.uniform(-4, 5, df.shape[0])
            df["T_csp_{}".format(z)] += (df["T_csp_{}".format(z)]-
df["T_in_{}".format(z)] >= 10)*np.random.uniform(-7, 6, df.shape[0])


        if isinstance(self.y_var, list):
            cols = [idx_col] + self.y_var + self.X_vars
        else:
            cols = [idx_col, self.y_var] + self.X_vars
        #self.df_in = self.df_raw[["Timestamp", "D_mains", "T_out"]]
        self.df_in = df[cols].copy()
```

```
        self.df_in = self.df_in[self.df_in.Timestamp >= start_date]
        self.df_in = self.df_in[self.df_in.Timestamp < end_date]
        self.df_in = self.df_in.set_index("Timestamp").resample(t_res).mean()
        #self.df_in = self.df_in.interpolate(method="spline", order=3)
        self.df_in = self.df_in.interpolate(method="linear").reset_index()
        #self.df_in = self.df_in[self.df_in.Timestamp.dt.dayofweek < 5] # Forecast
does not work

    def prep_altered_data(self, file_tstat_alt=None):
        df = self.df_raw.copy()

        # Import Altered Setpoints and Overwrite from File
        if file_tstat_alt is not None:
            setpoints_alt = pd.read_csv(file_tstat_alt)\
                .drop_duplicates(subset=["zone", "t_hr", "t_dow"])\
                .pivot(index=["t_hr", "t_dow"], columns="zone", values=["T_hsp",
"T_csp"])\
                .T.reset_index()
            setpoints_alt["col"] = setpoints_alt["level_0"].astype(str) + "_" +
setpoints_alt["zone"].astype(str)
            setpoints_alt = setpoints_alt\
                .set_index("col")\
                .drop(["level_0", "zone"], axis=1).T\
                .reset_index()\
                .set_index(["t_hr", "t_dow"])

            df = df.set_index(["t_hr", "t_dow"])
            df.update(setpoints_alt)
            df = df.reset_index()
        # More hack-y way of doing it...should deprecate this after testing
import+overwrite method
        else:
            setpoints_alt_diff = {
                "T_csp_1": 0,
                "T_csp_2": 0,
                "T_csp_3": +2,
                "T_csp_4": 0,
                "T_csp_5": 0,
                "T_csp_6": 0,
                "T_csp_7": -2,
                "T_csp_8": 0,
                "T_csp_9": 0,
                "T_csp_10": 0,
                "T_hsp_1": 0,
                "T_hsp_2": 0,
                "T_hsp_3": +2,
                "T_hsp_4": 0,
                "T_hsp_5": 0,
                "T_hsp_6": 0,
                "T_hsp_7": -3,
                "T_hsp_8": 0,
```

```
            "T_hsp_9": 0,
            "T_hsp_10": 0,
        }
        altered_hours = [9,10,11,12,13,14]
        df["mask"] = df["t_hr"].isin(altered_hours).map({True:1, False:0})
        for col in setpoints_alt_diff.keys():
            df[col] = df[col] + setpoints_alt_diff[col]*df["mask"]

    self.df_altered = self.prep_features(df)

    self.X_altered = TimeSeries.from_dataframe(self.df_altered.reset_index(),
"Timestamp", self.X_vars)
    self.X_altered_scaled = self.scaler_X.transform(self.X_altered)

def init_model(self):
    '''
    # for 15 min resolution
    self.model = NBEATSModel(
        input_chunk_length = 48*4,
        output_chunk_length = 24*4,
        n_epochs = 50,
        random_state = 0
    )
    '''
    # 1-hr model
    self.model = NBEATSModel(
        input_chunk_length = 48,
        output_chunk_length = 24,
        n_epochs = 5,
        random_state = 0
    )

def init_rnn_model(self, n_rnn_layers=2):
    self.model = RNNModel(
        model="RNN",
        input_chunk_length = 24*1,
        training_length = 24*2,
        save_checkpoints = True,
        n_rnn_layers = n_rnn_layers
    )

def init_blockrnn_model(self, n_epochs=50):
    self.model = BlockRNNModel(
        model = "LSTM",
        input_chunk_length = 24*4,
        output_chunk_length = 24*4,
        n_epochs = n_epochs,
        random_state = 0
    )

def fit_rnn_model(self, epochs):
```

```
        self.model.fit(
            self.train_y,
            future_covariates = self.train_X, # Set as bast or future
            epochs = epochs,
            verbose = True
        )

    def fit_blockrnn_model(self):
        self.model.fit(
            self.train_y,
            past_covariates=self.train_X,
            verbose = True
        )

    def init_regr_model(self):
        self.model = RegressionModel(
            lags=None,
            lags_past_covariates=[-24,-12,-2,-1],
            lags_future_covariates=[-24,-12,-2,-1,0],
        )

    def fit_regr_model(self):
        self.model.fit(
            self.train_y,
            past_covariates = self.train_X["T_out"],
            future_covariates = self.train_X["T_csp_3"]
        )

    def split_scale_data(self, split_date):
        self.y = TimeSeries.from_dataframe(self.df_in, "Timestamp", self.y_var)
        self.X = TimeSeries.from_dataframe(self.df_in, "Timestamp", self.X_vars)

        # Add noise
        x_shape = self.X.pd_dataframe().shape
        y_shape = self.y.pd_dataframe().shape
        x_noise = np.random.normal(0, self.X.pd_dataframe().std()*0.09, (x_shape[0],
x_shape[1]))
        y_noise = np.random.normal(0, self.y.pd_dataframe().std()*0.09, (y_shape[0],
y_shape[1]))
        x_noise = TimeSeries.from_dataframe(pd.DataFrame(x_noise,
index=self.X.pd_dataframe().index, columns=self.X.pd_dataframe().columns))
        y_noise = TimeSeries.from_dataframe(pd.DataFrame(y_noise,
index=self.y.pd_dataframe().index, columns=self.y.pd_dataframe().columns))
        self.X_wNoise = self.X + x_noise
        self.y_wNoise = self.y + y_noise

        self.scaler_y = Scaler().fit(self.y_wNoise)
        self.scaler_X = Scaler().fit(self.X_wNoise)
        self.y_scaled = self.scaler_y.transform(self.y_wNoise)
        self.X_scaled = self.scaler_X.transform(self.X_wNoise)
```

```
        self.train_y = self.y_scaled.drop_after(pd.to_datetime(split_date))#[:-
n_val_samples]
        self.val_y = self.y_scaled.drop_before(pd.to_datetime(split_date))#[-
n_val_samples:]
        self.train_X = self.X_scaled.drop_after(pd.to_datetime(split_date))#[:-
n_val_samples]
        self.val_X = self.X_scaled.drop_before(pd.to_datetime(split_date))#[-
n_val_samples:]

    def eval_model(self, past_covariates=None, future_covariates=None,
forecast_horizon=10, start=GLOBS.train_frac, test_name=None):
        if test_name is None:
            test_name= "backtest (n={})".format(forecast_horizon)
        else:
            test_name = test_name
        # Backtest the model on the last 20% of the self.train_y series, with a
horizon of 24 steps
        self.backtest = self.model.historical_forecasts(
            series = self.y_scaled,
            past_covariates=past_covariates,
            future_covariates=future_covariates,
            start=start,
            retrain=False,
            verbose=True,
            forecast_horizon=forecast_horizon
        )
        #self.y[-len(self.backtest)-150:].plot()
        self.y.plot(alpha=0.8)
        b_inv = self.scaler_y.inverse_transform(self.backtest)
        b_inv.plot(
            label = test_name, #label='backtest (n={})'.format(forecast_horizon),
            alpha = 0.8
        )
        self.df_raw.set_index("Timestamp")[["T_out", "T_in_3", "T_csp_3",
"T_hsp_3"]].plot()



        plt.legend()
        plt.show()

    def eval_model_altered(self, past_covariates=None, past_covariates_altered=None,
future_covariates=None, future_covariates_altered=None, forecast_horizon=10,
start=GLOBS.train_frac):
        # Backtest the model on the last 20% of the self.train_y series, with a
horizon of 24 steps
        self.backtest = self.model.historical_forecasts(
            series = self.y_scaled,
            past_covariates=past_covariates,
            future_covariates=future_covariates,
            start=start,
```

```
            retrain=False,
            verbose=True,
            forecast_horizon=forecast_horizon
        )
        #self.y[-len(self.backtest)-150:].plot()
        #self.y.plot(alpha=0.8)
        b_inv = self.scaler_y.inverse_transform(self.backtest)
        #b_inv.plot(
        #    label='backtest (n={})'.format(forecast_horizon),
        #    alpha = 0.8
        #)

        # Run backtest on altered data
        if past_covariates_altered is not None:
            self.backtest_altered = self.model.historical_forecasts(
                series = self.y_scaled,
                past_covariates = past_covariates_altered,
                future_covariates = future_covariates_altered,
                start = start,
                retrain = False,
                verbose = True,
                forecast_horizon = forecast_horizon
            )
        elif future_covariates_altered is not None:
            self.backtest_altered = self.model.historical_forecasts(
                series = self.y_scaled,
                past_covariates = past_covariates_altered,
                future_covariates = future_covariates_altered,
                start = start,
                retrain = False,
                verbose = True,
                forecast_horizon = forecast_horizon
            )
            b_inv_altered = self.scaler_y.inverse_transform(self.backtest_altered)
            #b_inv_altered.plot(
            #    label = "altered X",
            #    alpha = 0.8
            #)

        # Plot temperature data
        y = self.y.pd_dataframe()

        y_pred = b_inv.pd_dataframe() # .rename(columns={0:"b_inv"})
        y_pred_altered = b_inv_altered.pd_dataframe() #
.rename(columns={0:"counterfactual preds"})

        y.columns = self.y_var
        y_pred.columns = self.y_var
        y_pred_altered.columns = self.y_var
```

```
        if isinstance(self.y_var, list):
            row_idx = 1
            traces = []
            trace_rows = []
            trace_cols = []
            df_raw_15min =
self.df_raw.set_index("Timestamp").resample("15min").mean()
            for col in self.y_var:

                trace_y = go.Scatter(x=pd.to_datetime(y.index), y=y[col], name=col,
opacity=0.8)
                trace_y_pred = go.Scatter(x=pd.to_datetime(y_pred.index),
y=y_pred[col], name=col+"_pred", opacity=0.8)
                trace_y_altered = go.Scatter(x=pd.to_datetime(y_pred_altered.index),
y=y_pred_altered[col], name=col+"_counterfactual", opacity=0.8)

                traces.append(trace_y)
                traces.append(trace_y_pred)
                traces.append(trace_y_altered)
                trace_rows += [row_idx, row_idx, row_idx]
                trace_cols += [1,1,1]

                if "T_in_" in col:
                    # TODO: Need to only include the CSP/HSP traces if the target
variable is an indoor temperature
                    zone_idx = col.split("_")[-1]
                    hsp_col = "T_hsp_{}".format(zone_idx)
                    csp_col = "T_csp_{}".format(zone_idx)
                    trace_csp = go.Scatter(x=pd.to_datetime(df_raw_15min.index),
y=df_raw_15min[csp_col], name=csp_col, opacity=0.8)
                    trace_hsp = go.Scatter(x=pd.to_datetime(df_raw_15min.index),
y=df_raw_15min[hsp_col], name=hsp_col, opacity=0.8)
                    trace_csp_alt =
go.Scatter(x=pd.to_datetime(self.df_altered["Timestamp"]),
y=self.df_altered[csp_col], name=csp_col+"_counterfactual", opacity=0.6)
                    trace_hsp_alt =
go.Scatter(x=pd.to_datetime(self.df_altered["Timestamp"]),
y=self.df_altered[hsp_col], name=hsp_col+"_counterfactual", opacity=0.6)
                    traces.append(trace_csp)
                    traces.append(trace_hsp)
                    traces.append(trace_csp_alt)
                    traces.append(trace_hsp_alt)
                    trace_rows += [row_idx, row_idx, row_idx, row_idx]
                    trace_cols += [1,1,1,1]

                row_idx += 1

        else:
            row_idx = 1
            trace_y = go.Scatter(x=pd.to_datetime(y.index), y=y, name=self.y_var,
opacity=0.8)
```

```
            trace_y_pred = go.Scatter(x = pd.to_datetime(y_pred.index), y=y_pred,
name=self.y_var+"_pred", opacity=0.8)
            trace_y_pred_altered = go.Scatter(x=pd.to_datetime(y_pred_altered.index),
y=y_pred_altered, name=self.y_var+"_counterfactual", opacity=0.8)


            trace_rows = [row_idx, row_idx, row_idx]
            trace_cols = [1,1,1]
            traces = [trace_y, trace_y_pred, trace_y_pred_altered]


        #for col in ["T_out", "T_csp_3", "T_hsp_3", "T_in_3"]:
        for col in ["T_out"]:
            trace = go.Scatter(
                x = pd.to_datetime(df_raw_15min.index),
                y = df_raw_15min[col],
                name = col,
                opacity = 0.8
            )
            traces.append(trace)
            trace_rows.append(row_idx)
            trace_cols.append(1)

        fig = make_subplots(rows=row_idx, shared_xaxes=True)
        fig.add_traces(traces, rows=trace_rows, cols=trace_cols)
        pyo.plot(fig)


        #self.df_raw.set_index("Timestamp")[["T_out", "T_csp_3", "T_hsp_3", "T_out"]]
        #plt.legend()
        #plt.show()

    def get_episodic_preds(self, pred_dates, past_covariates=None,
past_covariates_altered=None, future_covariates=None, future_covariates_altered=None,
forecast_horizon=10, start=GLOBS.train_frac):
        """
        Run multiple 24-hr predictions
        Args:
            pred_dates: list of dates to run a 24-hr forecast on
            forecast_horizon = 24*4 (15-min intervals)
        """
        self.y_ep_pred = pd.DataFrame()
        self.y_ep_pred_alt = pd.DataFrame()
        for pred_date in pred_dates:
            y_pred = self.model.predict(
                n = 24*4, # Predict a full 24 hrs
                series = self.y_scaled.drop_after(pd.to_datetime(pred_date)),
                future_covariates=future_covariates,
                past_covariates=past_covariates
            )
            y_pred_alt = self.model.predict(
```

```
                n = 24*4,
                series = self.y_scaled.drop_after(pd.to_datetime(pred_date)),
                future_covariates=future_covariates_altered,
                past_covariates = past_covariates_altered
            )

            y_pred = self.scaler_y.inverse_transform(y_pred)
            y_pred_alt = self.scaler_y.inverse_transform(y_pred_alt)

            self.y_ep_pred = self.y_ep_pred.append(y_pred.pd_dataframe())
            self.y_ep_pred_alt = self.y_ep_pred_alt.append(y_pred_alt.pd_dataframe())

    def prep_sim_df(self, df_hist, df_hvac, t_res):
        """
        Merges the historical and simulated HVAC operation into a single dataframe
        :param df_hist: dataframe of historical data needed for the simulation
        :param df_hvac: dataframe of simulated hvac operation
        :param t_res: time resolution of the simulation.
        :return df_sim: the merged dataframe, with only the inner join of simulated
timestamps
        """
        df_hist = df_hist.resample(t_res).mean().interpolate(method="linear")
        df_hvac = df_hvac.resample(t_res).mean().interpolate(method="linear")
        df_sim = pd.concat([df_hist, df_hvac], axis=1, join="inner")
        return df_sim

    def simulate_batt(self, df_sim_in, x_d_max, x_soc_min, batt_cap_kwh,
battery_soc_max=100, t_res_minutes=15, chg_kw_max=29, dchg_kw_max=-22):
        """
        Simulates battery operation
        :param df_sim: merged dataframe with only the inner join of simulated
timestamp
        :param x_d_max: net load demand limite that battery discharge profile
maintains with available capacity
        :param x_soc_min: minimum SOC of battery reserved for backup and other
services
        :param batt_cap_kwh: total kwh of capacity

        dchg_kw_max = 7 # kW
        chg_kw_max = 7 # kW
        # Assumes SOC starts between 0-100%
        available_dchg_kwh = SOC * batt_cap_kwh / 100    # kWh available for
discharging (til 0%)
        available_chg_kwh = batt_cap_kwh - SOC_kwh       # kWh available for charging
(til 100%)

        # Convert to kW
        available_dchg_kw = available_dchg_kwh * 60 / t_res_minutes
        available_chg_kw = available_chg_kwh * 60 / t_res_minutes

        # Constrained by max chg/dchg rates of battery
```

```
        available_dchg_kw = - min(available_dchg_kw, dchg_kw_max) # Adds negative
sign for chg/dchg polarity
        available_chg_kw = min(available_chg_kw, chg_kw_max)

        # Battery kW needed
        chg_needed_kw = -G_solar - D_nonflex - D_hvac - x_d_max

        # Contrained battery kW dispatched
        chg_dispatched_kw = max(min(chg_needed_kw, available_chg_kw),
available_dchg_kw)
        D_batt = chg_dispatched_kw

        # Update SOC with dispatched battery chg/dchg rate (kW)
        SOC += (D_batt * t_res_minutes/60) / battery_capacity * 100
        """
        #t_res_minutes = 15 # TODO: moving to func args...time resolution of
simulation in minutes
        #chg_kw_max = 7 # TODO: moving to func args...max charge (kW)
        #dchg_kw_max = -22 # TODO: moving to func args...max discharge (kW)
        # cap_kwh_max = 20 # TODO: moving to func args...Max batter kWh, appears not
to be used
        #df_sim["X_soc_min"] = x_soc_min
        df_sim = df_sim_in.copy()
        df_sim["soc"] =np.nan
        #df_sim.iloc[0,-1] = 5
        df_sim["D_batt"] = np.nan
        #df_sim.iloc[0,-1] = 0


        # Intial conditions for SOC and battery kW
        soc_next = x_soc_min
        #d_batt_next = 0
        # Step through storage operation
        for i in df_sim.index:
            df_sim.loc[i,"soc"] = soc_next
            #df_sim.loc[i,"D_batt"] = d_batt_next
            row = df_sim.loc[i,:].copy()
            soc_next, d_batt = update_battery(
                row["soc"], row["G_solar"], row["D_nonflex"], row["D_hvac"],
                x_d_max, x_soc_min, battery_soc_max,
                batt_cap_kwh, t_res_minutes,
                chg_kw_max, dchg_kw_max
            )
            df_sim.loc[i,"D_batt"] = d_batt
        df_sim["D_net_mains"] = df_sim[["G_solar", "D_hvac", "D_nonflex",
"D_batt"]].sum(axis=1)
        df_sim["D_tot_mains"] = df_sim[["D_hvac", "D_nonflex"]].sum(axis=1)
        df_sim["soc_kwh"] = df_sim["soc"] * batt_cap_kwh / 100
        return df_sim
```

```
    def simulate_compute_batt_soc(self, current_soc, kw_15mins):
        # kw_15mins could be a float or an array of flaots
        #based on batt data for month of May 2021, linear fit using excel
        # gives the equation for delta_soc = 1.98 * energy_kw + 1.12
        tmp_kw_arr = []
        updated_soc = current_soc
        if type(kw_15mins) == float:
            tmp_kw_arr.append(kw_15mins)
        elif type(kw_15mins) == list:
            tmp_kw_arr.append(kw_15mins)
        else:
            print ("expecting float or list of floats, got")
            print (type(kw_15mins))

        if len(tmp_kw_arr) > 0:
            for x in tmp_kw_arr:
                delta_soc = 1.98 * x + 1.12
                updated_soc = updated_soc + delta_soc

        return updated_soc


    def process_cost(self, df_sim_in, import_key = "bill_import", export_key =
"bill_export"):
        df_sim = df_sim_in.copy()

        # df_sim_cost = pd.DataFrame()
        #split the data rows into rows for each day

        next_day_index = 0
        day_import = []
        day_export = []
        date_data = []
        date_str = []
        weekday_name = []
        while next_day_index < df_sim.shape[0]:
            current_sim_day = df_sim.iloc[next_day_index]["Timestamp"].date()
            next_sim_day = current_sim_day + timedelta(days = 1)
            # extracting all rows for the current_sim_day
            rows_for_day = df_sim[ ( df_sim["Timestamp"] >= str(current_sim_day) ) &
(df_sim["Timestamp"] < str(next_sim_day) )]
            next_day_index = next_day_index + rows_for_day.index[-1] + 1
            if rows_for_day.size > 0:
                #date_data.append(rows_for_day.iloc[0]['Timestamp'].to_pydatetime())
                date_data.append(current_sim_day)
                date_str.append(current_sim_day.strftime("%m/%d/%Y"))
                weekday_name.append(calendar.day_name[current_sim_day.weekday()])
                day_import.append( rows_for_day[import_key].sum() )
                day_export.append( rows_for_day[export_key].sum() )
        df_cost = pd.DataFrame({
                        "Date": date_str,
```

```
                                "Weekday" : weekday_name,
                                "import_cost": day_import,
                                "export_cost": day_export
                             })
        return df_cost



    def prepare_baseline_qhourly(self, df_sim_in, cols = ["G_solar", "D_hvac",
"D_battery_eg", "D_nonflex"], t_res_minutes = 15,
                                price_tou_dict=GLOBS_V2.price_tou, price_export =
0.18, price_peak = 19.85,
                                price_hday = GLOBS_V2.price_tou_hday):

        df_sim = df_sim_in.copy()

        df_sim_comp = pd.DataFrame()
        #split the data rows into rows for each day
        next_day_index = 0
        while next_day_index < df_sim.shape[0]:
            current_sim_day = df_sim.iloc[next_day_index]["Timestamp"].date()
            next_sim_day = current_sim_day + timedelta(days = 1)
            # extracting all rows for the current_sim_day
            rows_for_day = df_sim[ ( df_sim["Timestamp"] >= str(current_sim_day) ) &
(df_sim["Timestamp"] < str(next_sim_day) )]
            next_day_index = rows_for_day.index[-1] + 1
            if rows_for_day.size > 0:
                tmp_arr = ['Timestamp'] + cols
                tmp_rows_day = rows_for_day[tmp_arr].copy()
                tmp_rows_day['consumption'] = tmp_rows_day[['D_hvac',
'D_nonflex']].sum(axis=1)
                tmp_rows_day['consumption_kwh'] =
tmp_rows_day['consumption']*(t_res_minutes/60)
                tmp_rows_day['Hour'] = tmp_rows_day['Timestamp'].dt.hour
                tmp_rows_day['Minute'] = tmp_rows_day['Timestamp'].dt.minute
                tmp_rows_day = tmp_rows_day.reset_index()

                # tmp_dict = {}
                # for x in cols:
                #     tmp_dict[x] = 'sum'
                # tmp_dict['consumption'] = 'sum'
                # tmp_dict['consumption_kwh'] = 'sum'
                # hourly_consumption = tmp_rows_day.resample('H',
on='Timestamp').agg(tmp_dict)
                # hourly_consumption['price'] = price_tou_dict.values()
                # hourly_consumption = hourly_consumption.reset_index()
                wday_int = tmp_rows_day.loc[0, 'Timestamp'].to_pydatetime().weekday()
                if wday_int >= 0 and wday_int < 5:
                    tmp_rows_day['price'] = price_tou_dict.values()
                else:
                    tmp_rows_day['price'] = price_hday.values()
```

```
                tmp_rows_day['D_net_mains_b'] = tmp_rows_day[cols].sum(axis=1)
                tmp_rows_day['D_tot_mains_b'] = tmp_rows_day[["D_hvac", "D_nonflex"
]].sum(axis=1)
                print("max = ", tmp_rows_day.D_net_mains_b.max(), "min = ",
tmp_rows_day.D_net_mains_b.min())

                #temporarily define parameeters
                net_kw_col = "D_net_mains_b"
                #price_export = 0.18 # $/kwh
                #price_peak = 19.5 # $/kW*month
                tmp_rows_day.loc[tmp_rows_day[net_kw_col] >= 0, "kw_import"] =
tmp_rows_day.loc[tmp_rows_day[net_kw_col] >= 0, net_kw_col]
                tmp_rows_day.loc[tmp_rows_day[net_kw_col] <= 0, "kw_export"] =
tmp_rows_day.loc[tmp_rows_day[net_kw_col] <= 0, net_kw_col]
                tmp_rows_day["kwh_import"] = tmp_rows_day["kw_import"].fillna(0) *
t_res_minutes/60
                tmp_rows_day["kwh_export"] = tmp_rows_day["kw_export"].fillna(0) *
t_res_minutes/60
                tmp_rows_day["bill_import"] = tmp_rows_day["kwh_import"] *
tmp_rows_day["price"]
                tmp_rows_day["bill_export"] = tmp_rows_day["kwh_export"] *
price_export
                tmp_peak_idx = tmp_rows_day[["kw_import"]].idxmax()
                tmp_rows_day['kw_peak'] = 0
                tmp_rows_day.at[tmp_peak_idx, 'kw_peak'] =
tmp_rows_day.iloc[tmp_peak_idx]["kw_import"]

                df_sim_comp = pd.concat([df_sim_comp, tmp_rows_day])
                print('adjusted hourly for ',  current_sim_day)

        return df_sim_comp



    def simulate_batt_sch_with_cost(self, df_sim_in, x_d_max, x_soc_min,
batt_cap_kwh, battery_soc_max=100,
                                    t_res_minutes=15, chg_kw_max=29, dchg_kw_max=-22,
                                    price_tou_dict=GLOBS_V2.price_tou, price_export =
0.18,
                                    price_peak = 19.85,
                                    solar_threshold_kw_batt_charge = 2.25,
                                    price_hday = GLOBS_V2.price_tou_hday):
        """
        charge: when power available > GLOBS_V2.start_charge_limit till it reaches
max_soc
        discharge:
            identify max cost for the day = consumption * TOU cost for 1 hour
intervals
            discharge battery to reduce the cost to the next tier, repeat till
battery soc reaches min
```

```
        # Update SOC with dispatched battery chg/dchg rate (kW)
        SOC += (D_batt * t_res_minutes/60) / battery_capacity * 100
        """
        df_sim = df_sim_in.copy()
        df_sim["D_batt"] = np.nan
        df_sim['batt_action'] = 'none'

        df_sch = pd.DataFrame()

        # Intial conditions for SOC and battery kW
        batt_soc = x_soc_min
        #d_batt_next = 0

        #split the data rows into rows for each day
        next_day_index = 0
        while next_day_index < df_sim.shape[0]:
            current_sim_day = df_sim.iloc[next_day_index]["Timestamp"].date()
            next_sim_day = current_sim_day + timedelta(days = 1)
            # extracting all rows for the current_sim_day
            rows_for_day = df_sim[ ( df_sim["Timestamp"] >= str(current_sim_day) ) &
(df_sim["Timestamp"] < str(next_sim_day) )]
            next_day_index = rows_for_day.index[-1] + 1
            if rows_for_day.size > 0:
                tmp_rows_day = rows_for_day[['Timestamp', 'D_hvac', 'D_nonflex',
"G_solar", 'batt_action', 'D_batt']].copy()
                tmp_rows_day['consumption'] = tmp_rows_day[['D_hvac',
'D_nonflex']].sum(axis=1)
                tmp_rows_day['consumption_kwh'] =
tmp_rows_day['consumption']*(t_res_minutes/60)
                tmp_rows_day['Hour'] = tmp_rows_day['Timestamp'].dt.hour
                tmp_rows_day['Minute'] = tmp_rows_day['Timestamp'].dt.minute
                tmp_rows_day['soc'] = batt_soc
                tmp_rows_day.loc[ (-tmp_rows_day.G_solar - tmp_rows_day.consumption)
> solar_threshold_kw_batt_charge, 'batt_action'] = 'charge'
                tmp_rows_day = tmp_rows_day.reset_index()

                for ii in tmp_rows_day.loc[tmp_rows_day['batt_action'] ==
'charge'].index:
                    tmp_rate = -tmp_rows_day.iloc[ii]['G_solar'] -
tmp_rows_day.iloc[ii]['consumption']
                    tmp_soc, tmp_d_batt = update_battery_v2( batt_soc,
                        tmp_rate, x_soc_min, battery_soc_max, batt_cap_kwh,
t_res_minutes,
                        chg_kw_max, dchg_kw_max, 1
                    )
                    tmp_rows_day.at[ii, 'soc'] = tmp_soc
                    batt_soc = tmp_soc
                    tmp_rows_day.at[ii, 'D_batt'] = tmp_d_batt
                    if batt_soc >= battery_soc_max:
                        break
```

```
                wday_int = tmp_rows_day.loc[0, 'Timestamp'].to_pydatetime().weekday()
                if wday_int >= 0 and wday_int < 5:
                    tmp_rows_day['price'] = price_tou_dict.values()
                else:
                    tmp_rows_day['price'] = price_hday.values()
                tmp_rows_day = tmp_rows_day.reset_index()

                tmp_rows_day['computed_cost'] = 0.0
                tmp_rows_day['discharge_kwh'] = 0.0
                tmp_batt_soc = batt_soc
                batt_kwh_available = ((tmp_batt_soc-x_soc_min)/100) * batt_cap_kwh
                while batt_kwh_available > 0:
                    cond_mask = (tmp_rows_day['batt_action'] != 'charge')
                    h_c_selec = tmp_rows_day[cond_mask]
                    tmp_rows_day.loc[cond_mask, 'computed_cost'] = 
(h_c_selec['consumption_kwh'] - h_c_selec['discharge_kwh']  ) * h_c_selec['price']
                    tmp_max_cost = (tmp_rows_day.sort_values('computed_cost', 
ascending=False))['computed_cost'].copy()
                    if max(tmp_max_cost) < 0.01:
                        break
                    arr_max_cost = []
                    arr_max_cost.append(tmp_max_cost.index[0])
                    jj = 1
                    while (jj < tmp_max_cost.shape[0] ) and 
(abs(tmp_max_cost.iloc[jj] -  tmp_max_cost.iloc[0] )  < 0.15) :
                        arr_max_cost.append(tmp_max_cost.index[jj])
                        jj = jj + 1
                    idx_2max_cost = tmp_max_cost.index[jj]
                    batt_kwh_req = 0
                    batt_kwh_req_arr = []
                    for kk in arr_max_cost:
                        tmp_dchg_kwh = (tmp_rows_day.iloc[kk]['computed_cost'] - 
tmp_rows_day.iloc[idx_2max_cost]['computed_cost']) / tmp_rows_day.iloc[kk]['price']
                        tmp_val =  tmp_rows_day.loc[kk]['discharge_kwh']
                        tmp_rows_day.at[kk,'discharge_kwh'] = tmp_val + tmp_dchg_kwh
                        batt_kwh_req = batt_kwh_req + tmp_dchg_kwh
                        batt_kwh_req_arr.append(tmp_dchg_kwh)
                        tmp_rows_day.at[kk,'batt_action'] = 'discharge'
                        tmp_rows_day.at[kk,'D_batt'] = -
tmp_rows_day.iloc[kk]['discharge_kwh'] * 60 / t_res_minutes
                        #print(kk, batt_kwh_req, 
hourly_consumption.iloc[kk]['computed_cost'] - 
hourly_consumption.iloc[idx_2max_cost]['computed_cost'])
                    tmp_val = ( (dchg_kw_max * t_res_minutes)/60)
                    tmp_rate = min(batt_kwh_available, tmp_val)
                    if batt_kwh_req > tmp_rate:
                        tmp_sum = sum(batt_kwh_req_arr)
                        for tt in range(len(arr_max_cost)):
                            kk = arr_max_cost[tt]
                            tmp_val =  tmp_rows_day.loc[kk]['discharge_kwh'] - 
batt_kwh_req_arr[tt]
```

```
                            tmp_rows_day.at[kk,'discharge_kwh'] = tmp_val +
(tmp_rate*batt_kwh_req_arr[tt] )/ tmp_sum
                            tmp_rows_day.at[kk,'D_batt'] = -
tmp_rows_day.iloc[kk]['discharge_kwh'] * 60 / t_res_minutes
                        batt_kwh_req = tmp_rate

                    batt_kwh_available = batt_kwh_available - batt_kwh_req

                # # compute soc for discharge sets
                # for ii in
hourly_consumption.loc[hourly_consumption['discharge_kwh'] > 0 ].index:
                #     tmp_rate = hourly_consumption.loc[ii]['discharge_kwh'] *
t_res_minutes / 60 # hourly obtained from combining four 1`5 mins intervvals
                #     tmp_soc, tmp_d_batt = update_battery_v2( batt_soc,
                #         tmp_rate, x_soc_min, battery_soc_max, batt_cap_kwh, 60,
                #         chg_kw_max, dchg_kw_max, -1
                #     )
                #     hourly_consumption.at[ii, 'soc'] = tmp_soc
                #     batt_soc = tmp_soc
                #     hourly_consumption.at[ii, 'D_batt'] = tmp_d_batt
                #     if batt_soc <= x_soc_min:
                #         break
                #if batt_kwh_available > 0, then batt_soc would be > x_soc_min
                batt_soc = x_soc_min # used up all kwh in battery to support high
cost regions

                tmp_rows_day['D_batt'] = tmp_rows_day['D_batt'].fillna(0)
                tmp_rows_day['D_net_mains'] = tmp_rows_day[["G_solar", "D_hvac",
"D_nonflex", "D_batt"]].sum(axis=1)
                tmp_rows_day['D_tot_mains'] = tmp_rows_day[["D_hvac", "D_nonflex"
]].sum(axis=1)
                print("max = ", tmp_rows_day.D_net_mains.max(), "min = ",
tmp_rows_day.D_net_mains.min())

                #temporarily define parameeters
                net_kw_col = "D_net_mains"
                #price_export = 0.18 # $/kwh
                #price_peak = 19.5 # $/kW*month
                tmp_rows_day.loc[tmp_rows_day[net_kw_col] >= 0, "kw_import"] =
tmp_rows_day.loc[tmp_rows_day[net_kw_col] >= 0, net_kw_col]
                tmp_rows_day.loc[tmp_rows_day[net_kw_col] < 0, "kw_export"] =
tmp_rows_day.loc[tmp_rows_day[net_kw_col] <= 0, net_kw_col]
                tmp_rows_day["kwh_import"] = tmp_rows_day["kw_import"].fillna(0) *
t_res_minutes/60
                tmp_rows_day["kwh_export"] = tmp_rows_day["kw_export"].fillna(0) *
t_res_minutes/60
                tmp_rows_day["bill_import"] = tmp_rows_day["kwh_import"] *
tmp_rows_day["price"]
                tmp_rows_day["bill_export"] = tmp_rows_day["kwh_export"] *
price_export
                tmp_peak_idx = tmp_rows_day[["kw_import"]].idxmax()
```

```
                tmp_rows_day['kw_peak'] = 0
                tmp_rows_day.at[tmp_peak_idx, 'kw_peak'] =
tmp_rows_day.iloc[tmp_peak_idx]["kw_import"]

                df_sch = pd.concat([df_sch, tmp_rows_day])
                print('completed sim for the ',  current_sim_day)

        return df_sch




    def plot_battery_sim(self, df_sim, savepath=None, fig_title="", start="2000-01-
01", end="2100-01-01"):
        plot_cols = ["D_tot_mains", "G_solar", "D_net_mains", "D_batt", "soc_kwh"]
        df_plot = df_sim.loc[(df_sim.index>=start) & (df_sim.index <= end),
plot_cols]
        df_plot.plot()
        plt.title(fig_title)
        #plt.show()
        if savepath is not None:
            plt.savefig(savepath, dpi=200)
            plt.close()



    def calc_bill(self, df_sim_in, net_kw_col="D_net_mains", t_res_minutes=15,
peak_startup=24):
        """
        Calculates a TOU plus Demand Charge Electricity Bill given a kW time series
input
        :param ts_kw: pandas Series with datetime index of average kW values
            ts_kw must be a uniform timeseries with atleast 1-hour resolution.
        :param price_tou: dict of $/kwh price of electricity mapped to each hour of
the day
        :param price_peak: $/kw*month price of peak electricity demand that is
applied on a per month basis
        :return bill: electricity bill normalized to $/month
        """
        # TODO: move price_tou into args
        # TODO: move price_peak into args
        # TODO: need to figure out the actual TOU import and price_export $/kWh
values.
        price_offpeak = 0.14 # $/kWh
        price_onpeak = 0.22 # $/kWh
        price_export = 0.18 # $/kwh
        price_peak = 19.5 # $/kW*month
        price_tou = {
            0: price_offpeak, 1: price_offpeak, 2: price_offpeak,
            3: price_offpeak, 4: price_offpeak, 5: price_offpeak,
            6: price_offpeak, 7: price_offpeak, 8: price_offpeak,
```

```
            9: price_offpeak, 10: price_offpeak, 11: price_offpeak,
            12: price_onpeak, 13: price_onpeak, 14: price_onpeak,
            15: price_onpeak,
            16: price_onpeak, 17: price_onpeak, 18: price_onpeak, 19: price_offpeak,
20: price_offpeak,
            21: price_offpeak, 22: price_offpeak, 23: price_offpeak
        }

        df_sim = df_sim_in.copy()
        df_sim.loc[df_sim[net_kw_col] >= 0, "kw_import"] =
df_sim.loc[df_sim[net_kw_col] >= 0, net_kw_col]
        df_sim.loc[df_sim[net_kw_col] <= 0, "kw_export"] =
df_sim.loc[df_sim[net_kw_col] <= 0, net_kw_col]

        d_peak = df_sim.tail(-peak_startup)[net_kw_col].max()
        df_sim.loc[df_sim.tail(-peak_startup)[net_kw_col].idxmax(), "kw_peak"] =
df_sim.loc[df_sim.tail(-peak_startup)[net_kw_col].idxmax(), net_kw_col]

        df_sim["kwh_import"] = df_sim["kw_import"].fillna(0) * t_res_minutes/60
        df_sim["kwh_export"] = df_sim["kw_export"].fillna(0) * t_res_minutes/60

        df_sim["price_tou"] = df_sim.index.hour.map(price_tou)

        df_sim["bill_import"] = df_sim["kwh_import"] * df_sim["price_tou"]
        df_sim["bill_export"] = df_sim["kwh_export"] * price_export
        df_sim["bill_peak"] = df_sim["kw_peak"] * price_peak

        n_sims_permonth = pd.Timedelta(days=30) / (df_sim.index.max()-
df_sim.index.min()) # Number of simulation periods per 30 days
        df_sim["bill_import_permonth"] = df_sim["bill_import"] * n_sims_permonth
        df_sim["bill_export_permonth"] = df_sim["bill_export"] * n_sims_permonth
        df_sim["bill_peak_permonth"] = df_sim["bill_peak"].fillna(0)
        df_sim["bill_tou_permonth"] = df_sim[["bill_import_permonth",
"bill_export_permonth"]].sum(axis=1)
        df_sim["bill_total_permonth"] = df_sim[["bill_import_permonth",
"bill_export_permonth", "bill_peak_permonth"]].sum(axis=1)

        return df_sim


    def plot_episodic_preds(self, fig_height=None):
        # Convert TimeSeries Objects to DataFrames
        y = self.y.pd_dataframe()
        y_pred = self.y_ep_pred.copy() # b_inv.pd_dataframe() #
.rename(columns={0:"b_inv"})
        y_pred_altered = self.y_ep_pred_alt.copy() # b_inv_altered.pd_dataframe() #
.rename(columns={0:"counterfactual preds"})

        # Name DataFrame Columns
        y.columns = self.y_var
        y_pred.columns = self.y_var
```

```
        y_pred_altered.columns = self.y_var


        if isinstance(self.y_var, list):
            row_idx = 1
            traces = []
            trace_rows = []
            trace_cols = []
            df_raw_15min =
self.df_raw.set_index("Timestamp").resample("15min").mean()
            for col in self.y_var:

                trace_y = go.Scatter(
                    x=pd.to_datetime(y.index),
                    y=y[col],
                    name=col, opacity=0.8,
                    marker=dict(color=px.colors.qualitative.G10[0])
                )
                trace_y_pred = go.Scatter(
                    x=pd.to_datetime(y_pred.index),
                    y=y_pred[col],
                    name=col+"_pred", opacity=0.8,
                    marker=dict(color=px.colors.qualitative.G10[1])
                )
                trace_y_altered = go.Scatter(
                    x=pd.to_datetime(y_pred_altered.index),
                    y=y_pred_altered[col],
                    name=col+"_counterfactual", opacity=0.8,
                    marker=dict(color=px.colors.qualitative.G10[2])
                )

                traces.append(trace_y)
                traces.append(trace_y_pred)
                traces.append(trace_y_altered)
                trace_rows += [row_idx, row_idx, row_idx]
                trace_cols += [1,1,1]

                if "T_in_" in col:
                    # TODO: Need to only include the CSP/HSP traces if the target
variable is an indoor temperature
                    zone_idx = col.split("_")[-1]
                    hsp_col = "T_hsp_{}".format(zone_idx)
                    csp_col = "T_csp_{}".format(zone_idx)
                    trace_csp = go.Scatter(x=pd.to_datetime(df_raw_15min.index),
y=df_raw_15min[csp_col], name=csp_col, opacity=0.8, marker=dict(color="black"))
                    trace_hsp = go.Scatter(x=pd.to_datetime(df_raw_15min.index),
y=df_raw_15min[hsp_col], name=hsp_col, opacity=0.8, marker=dict(color="black"))
                    trace_csp_alt =
go.Scatter(x=pd.to_datetime(self.df_altered["Timestamp"]),
y=self.df_altered[csp_col], name=csp_col+"_counterfactual", opacity=0.5,
line=dict(color="black", dash="dot"))
```

```
                    trace_hsp_alt =
go.Scatter(x=pd.to_datetime(self.df_altered["Timestamp"]),
y=self.df_altered[hsp_col], name=hsp_col+"_counterfactual", opacity=0.5,
line=dict(color="black", dash="dot"))
                    trace_t_out = go.Scatter(x=pd.to_datetime(df_raw_15min.index),
y=df_raw_15min["T_out"], name="T_out", opacity=0.4, marker=dict(color="black"))
                    traces.append(trace_csp)
                    traces.append(trace_hsp)
                    traces.append(trace_csp_alt)
                    traces.append(trace_hsp_alt)
                    traces.append(trace_t_out)
                    trace_rows += [row_idx, row_idx, row_idx, row_idx, row_idx]
                    trace_cols += [1,1,1,1,1]

                row_idx += 1

        else:
            row_idx = 1
            trace_y = go.Scatter(x=pd.to_datetime(y.index), y=y, name=self.y_var,
opacity=0.8, marker=dict(color=px.colors.qualitative.G10[0]))
            trace_y_pred = go.Scatter(x = pd.to_datetime(y_pred.index), y=y_pred,
name=self.y_var+"_pred", opacity=0.8,
marker=dict(color=px.colors.qualitative.G10[1]))
            trace_y_pred_altered = go.Scatter(x=pd.to_datetime(y_pred_altered.index),
y=y_pred_altered, name=self.y_var+"_counterfactual", opacity=0.8,
marker=dict(color=px.colors.qualitative.G10[2]))


            trace_rows = [row_idx, row_idx, row_idx]
            trace_cols = [1,1,1]
            traces = [trace_y, trace_y_pred, trace_y_pred_altered]

        """
        #for col in ["T_out", "T_csp_3", "T_hsp_3", "T_in_3"]:
        for col in ["T_out"]:
            trace = go.Scatter(
                x = pd.to_datetime(df_raw_15min.index),
                y = df_raw_15min[col],
                name = col,
                opacity = 0.8,
                marker = dict(color="black")
            )
            traces.append(trace)
            trace_rows.append(row_idx)
            trace_cols.append(1)
        """


        fig = make_subplots(rows=row_idx, shared_xaxes=True)
        fig.add_traces(traces, rows=trace_rows, cols=trace_cols)
        fig.update_xaxes(showticklabels=True)
```

```
        if fig_height is not None:
            fig.update_layout(height=fig_height)
        pyo.plot(fig)
        self.fig = fig

    def fit_model(self):
        series_list = [self.train_y] + [self.train_X[c] for c in
self.train_X.components]
        self.model.fit(series_list, verbose=True)

    def pred_model(self, n_pred_samples):
        self.pred = self.model.predict(n=n_pred_samples, series=self.train_y)
        self.pred = self.scaler_y.inverse_transform(self.pred)

    def plot_results(self, fig_title="", savepath=None):
        pred = self.pred.pd_dataframe()
        actual = self.y.pd_dataframe()
        pred["Timestamp"] = pd.to_datetime(pred.index)

        pred = pred.reset_index(drop=True)\
            .rename(columns={self.y_var: "forecast"})\
            .set_index("Timestamp")

        actual["Timestamp"] = pd.to_datetime(actual.index)

        actual = actual.reset_index(drop=True)\
            .rename(columns={self.y_var:"actual"})\
            .set_index("Timestamp")

        df = pd.concat([actual, pred], axis=1)

        traces = []
        for col in df:
            trace = go.Scatter(
                x = df.index,
                y = df[col],
                name = col,
                opacity = 0.7
            )
            traces.append(trace)
        fig = go.Figure(data=traces)
        pyo.plot(fig)
        self.fig = fig

def update_battery(soc, G_solar, D_nonflex, D_hvac, x_d_max, x_soc_min, x_soc_max,
batt_cap_kwh, t_res_minutes, chg_kw_max, dchg_kw_max):
    #dchg_kw_max = 20 # kW, TODO: move to fn args
    #chg_kw_max = 7 # kW, TODO: move to fn args
    # Assumes soc starts between 0-100%
    available_dchg_kwh = (soc - x_soc_min)/100 * batt_cap_kwh  # kWh available for
discharging (til x_soc_min%)
```

```
    #available_chg_kwh = batt_cap_kwh - available_dchg_kwh       # kWh available for
charging (til 100%)
    available_chg_kwh = (x_soc_max - soc)/100 * batt_cap_kwh

    # Convert to kW
    available_dchg_kw = available_dchg_kwh * 60 / t_res_minutes
    available_chg_kw = available_chg_kwh * 60 / t_res_minutes

    # Constrained by max chg/dchg rates of battery
    available_dchg_kw = - min(available_dchg_kw, dchg_kw_max) # Adds negative sign
for chg/dchg polarity
    available_chg_kw = min(available_chg_kw, chg_kw_max)

    # Battery kW needed
    chg_needed_kw = -G_solar - D_nonflex - D_hvac + x_d_max

    # Contrained battery kW dispatched
    chg_dispatched_kw = max(min(chg_needed_kw, available_chg_kw), available_dchg_kw)
    D_batt = chg_dispatched_kw

    # Update soc with dispatched battery chg/dchg rate (kW)
    soc += (D_batt * t_res_minutes/60) / batt_cap_kwh * 100
    return soc, D_batt


def update_battery_v2(soc, x_d_max, x_soc_min, x_soc_max, batt_cap_kwh,
t_res_minutes, chg_kw_max, dchg_kw_max, perform_action):
    # av: if perform_action == 1, charge at x_d_max, if available
    #        else if it is -1, discharge at x_d_max if available

    #dchg_kw_max = 20 # kW, TODO: move to fn args
    #chg_kw_max = 7 # kW, TODO: move to fn args
    # Assumes soc starts between 0-100%
    available_dchg_kwh = (soc - x_soc_min)/100 * batt_cap_kwh  # kWh available for
discharging (til x_soc_min%)
    available_chg_kwh = (x_soc_max - soc)/100 * batt_cap_kwh

    # Convert to kW
    available_dchg_kw = available_dchg_kwh * 60 / t_res_minutes
    available_chg_kw = available_chg_kwh * 60 / t_res_minutes

    # Constrained by max chg/dchg rates of battery
    available_dchg_kw = min(available_dchg_kw, dchg_kw_max) # Adds negative sign for
chg/dchg polarity
    available_chg_kw = min(available_chg_kw, chg_kw_max)

    if perform_action == 1:
        D_batt = min(x_d_max, available_chg_kw)
    elif perform_action == -1:
        D_batt = -min(x_d_max, abs(available_dchg_kw))
    else:
```

```
        D_batt = 0

    # Update soc with dispatched battery chg/dchg rate (kW)
    soc = soc + ( (D_batt * t_res_minutes/60) / batt_cap_kwh * 100 )
    return soc, D_batt


if __name__ == "__main__":
    d = Forecaster(
        GLOBS.y_var,
        GLOBS.X_vars
    )
    d.import_data(GLOBS.filepath_in)
    d.df_raw = d.prep_features(d.df_raw)
    d.prep_data(start_date=GLOBS.start_date, end_date=GLOBS.end_date)
    d.split_scale_data(GLOBS.train_val_split_date)
    d.prep_altered_data(GLOBS.filepath_tstat_alt)

   # Fit RNN Model with Future Covariates
    d.init_rnn_model(n_rnn_layers=GLOBS.n_rnn_layers)
    d.fit_rnn_model(GLOBS.n_epochs)
    #d.eval_model_altered(future_covariates=d.X_scaled,
future_covariates_altered=d.X_altered_scaled,
forecast_horizon=GLOBS.forecast_horizon)

    # Run Episodic Predictions
    d.get_episodic_preds(
        future_covariates=d.X_scaled,
        future_covariates_altered=d.X_altered_scaled,
        pred_dates = GLOBS.pred_dates,
        forecast_horizon = GLOBS.forecast_horizon
    )
    d.plot_episodic_preds(fig_height = 1600)

    # earlier predictions during the training timeperiod
    #d.get_episodic_preds(
    #    future_covariates=d.X_scaled,
    #    future_covariates_altered=d.X_altered_scaled,
    #    pred_dates = GLOBS.training_pred_dates,
    #    forecast_horizon = GLOBS.forecast_horizon
    #)
    #d.plot_episodic_preds(fig_height = 1600)

    # Fit Block RNN Model with Past Covariates
    #d.init_blockrnn_model(n_epochs=GLOBS.n_epochs)
    #d.fit_blockrnn_model()
    #d.eval_model_altered(past_covariates=d.X_scaled,
past_covariates_altered=d.X_altered_scaled, forecast_horizon=GLOBS.forecast_horizon)
```

```
    # Run NBEATS moel
    #d.init_model()
    #d.fit_model()
    #d.eval_model(fig_title="NBEATS")

    #d.pred_model(GLOBS.n_val_samples)
    #d.plot_results()

    print("Test DONE!")
```

The source code for the simulate.py is listed below

```
import pandas as pd
import numpy as np
import os

import plotly.graph_objs as go
import plotly.offline as pyo
import plotly.express as px
from plotly.subplots import make_subplots
import matplotlib.pyplot as plt

from darts import TimeSeries
import test_darts_forecast_v3gui as forecaster

import torch

#simulate v3: vis
#   convert data to hourly data for day, and discharge according to higher cost hours
#   keep 15 min interval data, and discharge according to higher cost qurters hours
with hourly-price split into quarter for simulation


#simulate v2: vis
#   convert data to hourly data for day, and discharge according to higher cost hours
#   issue: PG&e peak pricing is 8:30 to 12:30 summer and therefor cannot align this
price in simulation correctly

#batt model linear regression delta_soc = 1.66 * x + 0.5 , where x = kw in 15 min
sampling



class GLOBS():
    # Old simulate.py GLOBS:
```

```
    #filepath_historical = "./data/df_historical.csv"
    #filepath_model = "./.darts/checkpoints/2021-10-
02_21.15.59.427539_torch_model_run_8836/checkpoint_19.pth.tar"

    #filepath_X = "./X_scaled.P"
    #filepath_X_alt = "./X_altered_scaled.P"

    #start_ts = pd.to_datetime("2021-05-02")
    #n_startup = 24*45
    #forecast_horizon = 4

    # New simulate.py GLOBS:
    start_date = "2021-04-20" # TODO: Add Winter Months to training data (or warmer
summer...?) (DONE)
    end_date = "2021-05-30"
    plot_start = "2021-05-15"
    plot_end = "2021-05-19"

    # eval_darts_forecast GLOBS:
    filepath_in = "./data/df_historical_20210101-20210601.csv"
    filepath_tstat_alt = "./data/test_data/darts_counterfactual_tstat.csv"
    # trained model from Corey's machine
    #file_trained_model = ".darts/checkpoints/2021-10-
09_14.01.57.041023_torch_model_run_1228/checkpoint_49.pth.tar"
    # trained model from run on Vis machine
    file_trained_model = ".darts/checkpoints/2022-01-
05_16.35.18.059790_torch_model_run_27084/checkpoint_49.pth.tar"
    #file_trained_model =
"C:/Users/pvan002/Documents/jupyternotebooks/demos/epri_sim_env_ananconda/data/model4
sim.pth.tar"
    pred_dates = [
        "2021-05-11", "2021-05-12", "2021-05-13", "2021-05-14",
        "2021-05-15", "2021-05-16", "2021-05-17", "2021-05-18",
        "2021-05-19", "2021-05-20", "2021-05-21", "2021-05-22",
        "2021-05-23", "2021-05-24", "2021-05-25", "2021-05-26",
        "2021-05-27", "2021-05-28", "2021-05-29"
    ]
    forecast_horizon = 24*4
    t_res_minutes = 15

    # Storage Operational Configuration
    # Uncomment the battery_test_case declaration for the appropriate simulation test
case.
    # Currently implemented for "TUNE HISTORICAL BASELINE" and "X_d_max SENSITIVITY
ANALYSIS"
    #battery_test_case = "TUNE HISTORICAL BASELINE"
    #battery_test_case = "X_d_max SENSITIVITY ANALYSIS"
    battery_test_case = "bill_reduction"

    if battery_test_case == "TUNE HISTORICAL BASELINE":
```

```
        # Battery Params (tuned to historical usage from 2021-05-11 through 2021-05-
14)
        batt_cap_kwh = 52 # kWh
        battery_soc_max = 63 # SOC% maximum setting
        x_soc_min = 30
        X_d_max = [2.7] # kW
        chg_kw_max = 29
        dchg_kw_max = 29
    elif battery_test_case == "X_d_max SENSITIVITY ANALYSIS":
        # Grid Seach optimization
        batt_cap_kwh = 52 # Obtained from
        battery_soc_max = 100
        x_soc_min = 30
        X_d_max = [2,4,5,6,7,8,9,10,11,12] # kW
        chg_kw_max = 29
        dchg_kw_max = 29
    elif battery_test_case == "bill_reduction":
        # new algorithm using batt to reduce high price ranges
        batt_cap_kwh = 52 # Obtained from
        battery_soc_max = 100
        x_soc_min = 30
        X_d_max = 2.7 # kW
        chg_kw_max = 29
        dchg_kw_max = 29

    start_charge_limit = -4 # kW

    price_offpeak = 0.14181 # $/kWh
    price_onpeak = 0.22501 # $/kWh
    price_partpeak = 0.16988 #$kwh
    price_export = 0.18 # $/kwh
    price_peak = 19.85 # $/kW*month (demand charge)
    customer_charge = 4.59959 # charge / day according to bill
    price_arr = []
    price_arr.extend( (np.ones(34) * price_offpeak).tolist() )  # 12:00 Am to 8:30 Am
    price_arr.extend( (np.ones(14) * price_partpeak).tolist() ) # 8:30 AM to 12:00 PM
    price_arr.extend( (np.ones(24) * price_onpeak).tolist() )   # 12:00 PM to 6:00 PM
    price_arr.extend( (np.ones(14) * price_partpeak).tolist() ) # 6:00 PM to 9:30 PM
    price_arr.extend( (np.ones(10) * price_offpeak).tolist() )  # 9:30 PM to 12:00 AM
    price_tou = dict(enumerate(price_arr))

    price_arr = []
    price_arr.extend( (np.ones(96) * price_offpeak).tolist() )  # 12:00 Am to 12:00
Am next day
    price_tou_hday = dict(enumerate(price_arr))



#if __name__ == "__main__":
def simulate_from_gui(x_vars = None, y_var = None,
```

```
                      start_date = None, end_date = None, train_val_split_date =
None,
                      file_model = None, data_in_file = None
                      ):
    # TODO: Take the eval_darts_forecast main, and expand to incorporate battery
operation response to
    # HVAC alternate operation (counterfactual operation)
    if x_vars is not None:
        t_x_vars = x_vars
    else:
        t_x_vars = forecaster.GLOBS.X_vars
    if y_var is not None:
        t_y_var= y_var
    else:
        t_y_var = forecaster.GLOBS.y_var

    f = forecaster.Forecaster(
        t_y_var,
        t_x_vars
    )

    # TODO: start_date, end_date, train_val_split_date usage

    if data_in_file is None:
        f.import_data(GLOBS.filepath_in)
    else:
        f.import_data(data_in_file)

    f.df_raw = f.prep_features(f.df_raw)
    f.prep_data(start_date=GLOBS.start_date, end_date=GLOBS.end_date)
    f.split_scale_data(forecaster.GLOBS.train_val_split_date)

    f.prep_altered_data(GLOBS.filepath_tstat_alt)
    if file_model is None:
        f.import_trained_model(GLOBS.file_trained_model)
    else:
        f.import_trained_model(file_model)

    # Run Episodic Predictions
    f.get_episodic_preds(
        future_covariates=f.X_scaled,
        future_covariates_altered=f.X_altered_scaled,
        pred_dates = GLOBS.pred_dates,
        forecast_horizon = GLOBS.forecast_horizon
    )
    #f.plot_episodic_preds(fig_height = 1600)


    # Simulate Battery Operation accounting for simulated HVAC
    df_sim_baseline = f.prep_sim_df(
```

```
        df_hist = f.df_raw.set_index("Timestamp")[["G_solar", "D_nonflex",
"D_battery_eg"]],
        df_hvac = f.y_ep_pred,
        t_res = "{}min".format(GLOBS.t_res_minutes),
    )

    if GLOBS.battery_test_case == "X_d_max SENSITIVITY ANALYSIS":

        df_sim_baseline = f.simulate_batt(
            df_sim_baseline,
            x_d_max = 2.7, # kW
            x_soc_min = GLOBS.x_soc_min, # percent,
            batt_cap_kwh = GLOBS.batt_cap_kwh, # kWh, TODO: update with actual values
            battery_soc_max = GLOBS.battery_soc_max,
            t_res_minutes = GLOBS.t_res_minutes, # TODO: moving to func args...time
resolution of simulation in minutes
            chg_kw_max = GLOBS.chg_kw_max, # TODO: moving to func args...max charge
(kW)
            dchg_kw_max = GLOBS.dchg_kw_max # TODO: moving to func args...max
discharge (kW)
            # cap_kwh_max = 20 # TODO: moving to func args...Max batter kWh, appears
not to be used
        )
        f.plot_battery_sim(
            df_sim_baseline,
            fig_title = "Simulated Baseline",
            savepath = "timeseries-plot_sim-baseline.png",
            start = GLOBS.plot_start,
            end = GLOBS.plot_end
        )

        # Alternate battery setting
        df_sim_alt = f.prep_sim_df(
            df_hist = f.df_raw.set_index("Timestamp")[["G_solar", "D_nonflex"]],
            df_hvac = f.y_ep_pred,
            t_res = "{}min".format(GLOBS.t_res_minutes),
        )

        df_sim_alt = f.simulate_batt(
            df_sim_alt,
            x_d_max=7.5,
            x_soc_min = GLOBS.x_soc_min,
            batt_cap_kwh = GLOBS.batt_cap_kwh,
            battery_soc_max = GLOBS.battery_soc_max,
            t_res_minutes=GLOBS.t_res_minutes,
            chg_kw_max=GLOBS.chg_kw_max,
            dchg_kw_max=GLOBS.dchg_kw_max
        )
        f.plot_battery_sim(
            df_sim_alt,
            fig_title = "Simulated Alternate Battery Setting",
```

```
            savepath = "timeseries-plot_sim-altbatt.png",
            start = GLOBS.plot_start,
            end = GLOBS.plot_end
        )

        # Simulate x_d_max sensitivity analysis
        results = {}
        bills = {}
        bills_tou = {}
        peaks = {}
        for x_d_max in GLOBS.X_d_max:
            results[x_d_max] = f.calc_bill(
                f.simulate_batt(
                    df_sim_alt,
                    x_d_max=x_d_max,
                    x_soc_min = GLOBS.x_soc_min,
                    batt_cap_kwh=GLOBS.batt_cap_kwh,
                    battery_soc_max = GLOBS.battery_soc_max,
                    t_res_minutes=GLOBS.t_res_minutes,
                    chg_kw_max=GLOBS.chg_kw_max,
                    dchg_kw_max=GLOBS.dchg_kw_max
                )
            )
            bills[x_d_max] = results[x_d_max]['bill_total_permonth'].sum()
            bills_tou[x_d_max] = results[x_d_max]['bill_tou_permonth'].sum()
            peaks[x_d_max] = results[x_d_max]['D_net_mains'].max()
            print("Dem Limit: {} kW, TOU Bill: {:.2f}, Peak Dem: {:.2f}, Bill:
${:.2f}".format(x_d_max, bills_tou[x_d_max], peaks[x_d_max], bills[x_d_max]))
            # Export Results to CSV
            results[x_d_max].to_csv("timeseries-results_{}kw.csv".format(x_d_max))
        pd.Series(bills).plot()
        plt.xlabel("Net Load Demand Limit (kW)")
        plt.ylabel("Monthly Bill ($/month)")
        plt.savefig('sensitivity-analysis.png', dpi=200)
        plt.close()

        # Investigate Timeseries Plot Outputs

        # 6 kW Limit Plots
        results[6][['bill_import', 'bill_export']].plot(figsize=[12,4])
        plt.title("6 kW Limit")
        plt.savefig('timeseries-bills_6kw.png')
        plt.close()

        results[6][["D_net_mains"]].plot(figsize=[12,4])
        plt.title("6 kW Limit")
        plt.savefig('timeseries-netmains_6kw.png')
        plt.close()

        # 8 kW Limit Plots
```

```
        # Plot Historical Baseline
        df_hist_baseline = f.df_raw.set_index("Timestamp")
        df_hist_baseline =
df_hist_baseline[(df_hist_baseline.index>=df_sim_baseline.index.min()) &
(df_hist_baseline.index <= df_sim_baseline.index.max())]
        df_hist_baseline["D_tot"] = df_hist_baseline["D_mains"] -
df_hist_baseline["G_solar"] - df_hist_baseline["D_battery_eg"]
        df_hist_baseline["SOC_kwh"] = df_hist_baseline["SOC"] * GLOBS.batt_cap_kwh /
100
        #df_hist_baseline[["D_tot", "G_solar", "D_mains", "D_battery_eg",
"SOC_kwh"]].plot()
        #plt.title("Historical Baseline")

        print('Done %s' % GLOBS.battery_test_case)

    elif GLOBS.battery_test_case == "bill_reduction":
        # Timestamp is set as index, later on in the computation, it is converted to
use hourly data, so specifically setting Timestamp
        df_sim_baseline['Timestamp'] = df_sim_baseline.index
        df_sim_baseline = df_sim_baseline.reset_index(drop=True)

        df_sim_hr = f.simulate_batt_sch_with_cost(
            df_sim_baseline,
            x_d_max=7.5,
            x_soc_min = GLOBS.x_soc_min,
            batt_cap_kwh = GLOBS.batt_cap_kwh,
            battery_soc_max = GLOBS.battery_soc_max,
            t_res_minutes= 15,
            chg_kw_max=GLOBS.chg_kw_max*4,
            dchg_kw_max=GLOBS.dchg_kw_max*4,
            price_tou_dict=GLOBS.price_tou,
            price_export = GLOBS.price_export,
            price_peak = GLOBS.price_peak,
            price_hday = GLOBS.price_tou_hday
        )
        df_sim_hr.set_index('Timestamp')

        print('     ')

        df_sim_baseline_hr = f.prepare_baseline_qhourly(df_sim_baseline,
                                                       cols = ["G_solar", "D_hvac",
"D_battery_eg", "D_nonflex"],
                                                       t_res_minutes=15,

price_tou_dict=GLOBS.price_tou,
                                                       price_export =
GLOBS.price_export,
                                    price_peak = GLOBS.price_peak,
                                    price_hday = GLOBS.price_tou_hday)
```

```
        #df_sim_hr[['D_batt', 'G_solar', "D_net_mains", 'D_tot_mains', 'D_hvac',
'soc']].plot()


        df_sim_baseline_hr.rename(
            columns={"G_solar":"G_solar_b", "D_hvac":"D_hvac_b",
                        "D_nonflex":"D_nonflex_b",
                        "price": "price_b",
                        "consumption": "consumption_b",
                        "consumption_kwh": "consumption_kwh_b",
                        "kw_import": "kw_import_b",
                        "kw_export" : "kw_export_b",
                        "kw_peak" : "kw_peak_b",
                        "kwh_import": "kwh_import_b",
                        "kwh_export" : "kwh_export_b",
                        "bill_import" : "bill_import_b",
                        "bill_export" : "bill_export_b"
                        }
                  ,inplace=True)

        df_cost_sim = f.process_cost(df_sim_hr)
        peak_sim = df_sim_hr.kw_peak.max()
        df_cost_baseline = f.process_cost(df_sim_baseline_hr,
import_key='bill_import_b', export_key = "bill_export_b")
        peak_baseline = df_sim_baseline_hr.kw_peak_b.max()

        print("Baseline (based on computed HVAC for simulated rabge")
        print(df_cost_baseline )
        print("peak kw = ", peak_baseline)
        tmp_import_cost = df_cost_baseline.import_cost.sum()
        tmp_export_cost = df_cost_baseline.export_cost.sum()
        total_baseline_cost = GLOBS().price_peak * peak_baseline + tmp_import_cost +
tmp_export_cost
        print("import -> $", tmp_import_cost, "  export -> $", tmp_export_cost)
        print("total bill -> $", total_baseline_cost)

        print("Simulation: (based on computed HVAC for simulated rabge")
        print(df_cost_sim )
        print("peak kw = ", peak_sim)
        tmp_import_cost = df_cost_sim.import_cost.sum()
        tmp_export_cost = df_cost_sim.export_cost.sum()
        total_sim_cost = GLOBS().price_peak * peak_sim + tmp_import_cost +
tmp_export_cost
        print("import -> $", tmp_import_cost, "  export -> $", tmp_export_cost)
        print("total bill -> $", total_sim_cost)


        df_sim_baseline_hr.set_index('Timestamp', inplace=True)
        df_sim_hr.set_index('Timestamp', inplace=True)
        df_sim_hr.drop(['index', 'level_0', 'Hour', 'Minute'], inplace=True, axis=1)
        df_sim_baseline_hr.drop('index', inplace=True, axis=1)
```

```
        tmpdf = df_sim_hr.join(df_sim_baseline_hr)
        tmpdf.to_csv('./temp_sim_comp_hr.csv')


        # df_sim_hr_bill = f.calc_bill(df_sim_hr, net_kw_col='D_net_mains',
t_res_minutes=15)
        # df_sim_baseline_hr_bill = f.calc_bill(df_sim_baseline_hr,
net_kw_col='D_net_mains_b', t_res_minutes=15)

        # bills = df_sim_hr_bill['bill_total_permonth'].sum()
        # bills_tou = df_sim_hr_bill['bill_tou_permonth'].sum()
        # peaks = df_sim_hr_bill['D_net_mains'].max()
        # print('Simulated')
        # print("TOU Bill: {:.2f}, Peak Dem: {:.2f}, Bill: ${:.2f}".format(bills_tou,
peaks, bills))


        # bills = df_sim_baseline_hr_bill['bill_total_permonth'].sum()
        # bills_tou = df_sim_baseline_hr_bill['bill_tou_permonth'].sum()
        # peaks = df_sim_baseline_hr_bill['D_net_mains_b'].max()
        # print('baseline')
        # print("TOU Bill: {:.2f}, Peak Dem: {:.2f}, Bill: ${:.2f}".format(bills_tou,
peaks, bills))

        #df_sim_baseline['D_net_mains'] = df_sim_baseline[["G_solar", "D_hvac",
"D_nonflex"]].sum(axis=1)
        #df_sim_baseline['D_tot_mains'] = df_sim_baseline[["D_hvac", "D_nonflex"
]].sum(axis=1)

        #df_sim_baseline.set_index('Timestamp')
        #df_sim_baseline[['G_solar', "D_net_mains", 'D_tot_mains', 'D_hvac']].plot()

        df_sim_hr.reset_index(inplace=True)
        df_sim_baseline_hr.reset_index(inplace=True)

        print("Optimization done... Plotting data. Data will open up in a browser")

        traces = []
        #for col in ['D_batt', 'G_solar', "D_net_mains", 'D_tot_mains', 'D_hvac',
'soc']:
        for col in ['D_batt', 'G_solar', "D_net_mains", 'D_tot_mains', 'D_hvac',
"kw_peak",
                   "kwh_import", "bill_import", "kwh_export", "bill_export"]:
            trace = go.Scatter(
                x = df_sim_hr.Timestamp,
                y = df_sim_hr[col],
                name = col,
                opacity = 0.7
            )
            traces.append(trace)
```

```
        for col in ['D_net_mains_b', 'D_tot_mains_b', "kw_peak_b",
                    "kwh_import_b", "bill_import_b", "kwh_export_b",
"bill_export_b"]:
            trace = go.Scatter(
                x = df_sim_baseline_hr.Timestamp,
                y = df_sim_baseline_hr[col],
                name = col,
                opacity = 0.7
            )
            traces.append(trace)
        trace = go.Scatter(
            x = df_sim_baseline_hr.Timestamp,
            y = df_sim_baseline_hr['D_battery_eg'],
            name = "D_batt_b",
            opacity = 0.7
        )
        traces.append(trace)

        fig = go.Figure(data=traces, layout={'title': "Sim Schedule"})
        pyo.plot(fig, 'all_data')
```

The source code for the run_simulate.py is listed below.

```
# -*- coding: utf-8 -*-
"""

@author: Viswanath
"""

import subprocess
import io

proc3 = subprocess.Popen('python simulate_v3.py', stdout = subprocess.PIPE,
stderr=subprocess.STDOUT, shell=True)

for line in io.TextIOWrapper(proc3.stdout, encoding='utf-8'):
    print(line)
```

Appendix A : Library dependencies for OpenBats simulation tool.

As with any tool, there is a list of dependencies for executing the various parts of the code, for modeling, for optimization, and for the user interface. A brief list of dependent libraries are presented in this section.

To easily manage packages and their version across teams, anaconda-navigator is suggested. Any other scheme would also work equally well. Sample list of packages exported for this simulation environment is listed below. Using a file containing this information, a new environment with these dependencies could be created with a single click in anaconda-navigator tool.

```
# This file may be used to create an environment using:
# $ conda create --name <env> --file <this file>
# platform: win-64
@EXPLICIT
https://conda.anaconda.org/conda-forge/win-64/ca-certificates-2021.10.8-h5b45459_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/intel-openmp-2021.4.0-h57928b3_3556.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/mkl-include-2021.4.0-h0e2418a_729.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/msys2-conda-epoch-20160418-1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pandoc-2.16.2-h8ffe710_0.tar.bz2
https://conda.anaconda.org/pytorch/noarch/pytorch-mutex-1.0-cuda.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/ucrt-10.0.20348.0-h57928b3_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/winpty-0.4.3-4.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-gmp-6.1.0-2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-headers-git-5.0.0.4636.c0ad18a-2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-isl-0.16.1-2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-libiconv-1.14-6.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-libmangle-git-5.0.0.4509.2e5a9a2-2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-libwinpthread-git-5.0.0.4634.697f757-
2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-make-4.1.2351.a80a8b8-2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-windows-default-manifest-6.4-3.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/vs2015_runtime-14.29.30037-h902a5da_5.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-crt-git-5.0.0.4636.2595836-2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-gcc-libs-core-5.3.0-7.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-mpfr-3.1.4-4.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-pkg-config-0.29.1-2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/vc-14.2-hb210afc_5.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/bzip2-1.0.8-h8ffe710_4.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/cudatoolkit-11.1.1-heb2d755_9.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/icu-68.2-h0e60522_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/jbig-2.1-h8d14728_2003.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/jpeg-9d-h8ffe710_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/lerc-3.0-h0e60522_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libbrotlicommon-1.0.9-h8ffe710_6.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libclang-11.1.0-default_h5c34c98_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libdeflate-1.8-h8ffe710_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libsodium-1.0.18-h8d14728_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libspatialindex-1.9.3-h39d44d4_4.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libuv-1.42.0-h8ffe710_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libzlib-1.2.11-h8ffe710_1013.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/lz4-c-1.9.3-h8ffe710_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-gcc-libgfortran-5.3.0-6.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-mpc-1.0.3-3.tar.bz2
```

```
https://conda.anaconda.org/conda-forge/win-64/m2w64-winpthreads-git-5.0.0.4634.697f757-
2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/mysql-connector-c-6.1.11-h001a745_1007.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/openssl-1.1.1l-h8ffe710_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/sqlite-3.37.0-h8ffe710_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/tbb-2021.4.0-h2d74725_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/tk-8.6.11-h8ffe710_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/xz-5.2.5-h62dcd97_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/yaml-0.2.5-he774522_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/krb5-1.19.2-h20d022d_3.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libbrotlidec-1.0.9-h8ffe710_6.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libbrotlienc-1.0.9-h8ffe710_6.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-gcc-libs-5.3.0-7.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/mkl-2021.4.0-h0e2418a_729.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/python-3.7.12-h7840368_100_cpython.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/zeromq-4.3.4-h0e60522_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/zlib-1.2.11-h8ffe710_1013.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/alabaster-0.7.12-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/argh-0.26.2-pyh9f0ad1d_1002.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/async_generator-1.10-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/asynctest-0.13.0-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/atomicwrites-1.4.0-pyh9f0ad1d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/attrs-21.2.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/backcall-0.2.0-pyh9f0ad1d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/backports-1.0-py_2.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/blinker-1.4-py_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/brotli-bin-1.0.9-h8ffe710_6.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/cachetools-4.2.4-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/charset-normalizer-2.0.9-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/cloudpickle-2.0.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/colorama-0.4.4-pyh9f0ad1d_0.tar.bz2
https://repo.anaconda.com/pkgs/main/win-64/console_shortcut-0.1.1-4.conda
https://conda.anaconda.org/conda-forge/noarch/cycler-0.11.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/dataclasses-0.8-pyhc8e2a94_3.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/decorator-5.1.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/defusedxml-0.7.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/diff-match-patch-20200713-pyh9f0ad1d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/entrypoints-0.3-pyhd8ed1ab_1003.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/hdf4-4.2.15-h0e5069d_3.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/hijri-converter-2.2.2-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/idna-3.1-pyhd3deb0d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/imagesize-1.3.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/inflection-0.5.1-pyh9f0ad1d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/ipython_genutils-0.2.0-py_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/korean_lunar_calendar-0.2.1-pyh9f0ad1d_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libblas-3.9.0-12_win64_mkl.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libpng-1.6.37-h1d00b33_2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libprotobuf-3.19.1-h7755175_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libssh2-1.10.0-h680486a_2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libzip-1.8.0-hfed4ece_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-bzip2-1.0.6-6.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-tools-git-5.0.0.4592.90b8472-2.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/mccabe-0.6.1-py_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/mkl-devel-2021.4.0-h57928b3_730.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/munkres-1.1.4-pyh9f0ad1d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/nest-asyncio-1.5.4-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/olefile-0.46-pyh9f0ad1d_1.tar.bz2
```

```
https://conda.anaconda.org/conda-forge/noarch/pandocfilters-1.5.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/parso-0.8.3-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pathspec-0.9.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pickleshare-0.7.5-py_1003.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/platformdirs-2.3.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/poyo-0.5.0-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/prometheus_client-0.12.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/ptyprocess-0.7.0-pyhd3deb0d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pyasn1-0.4.8-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pycodestyle-2.8.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pycparser-2.21-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pyflakes-2.4.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pyjwt-2.3.0-pyhd8ed1ab_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pymeeus-0.5.10-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pyparsing-3.0.6-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/python_abi-3.7-2_cp37m.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pytz-2021.3-pyhd8ed1ab_0.tar.bz2
https://repo.anaconda.com/pkgs/main/noarch/qdarkstyle-3.0.2-pyhd3eb1b0_0.conda
https://conda.anaconda.org/conda-forge/noarch/qtpy-1.11.3-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/rope-0.22.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/send2trash-1.8.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/six-1.16.0-pyh6c4a22f_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/snowballstemmer-2.2.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/sortedcontainers-2.4.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/sphinxcontrib-applehelp-1.0.2-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/sphinxcontrib-devhelp-1.0.2-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/sphinxcontrib-htmlhelp-2.0.0-
pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/sphinxcontrib-jsmath-1.0.1-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/sphinxcontrib-qthelp-1.0.3-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/tenacity-8.0.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/tensorboard-plugin-wit-1.8.0-
pyh44b312d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/testpath-0.5.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/textdistance-4.2.2-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/threadpoolctl-3.0.0-pyh8a188c0_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/toml-0.10.2-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/tomli-1.2.2-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/traitlets-5.1.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/typing_extensions-3.10.0.2-pyha770c72_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/webencodings-0.5.1-py_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/wheel-0.37.0-pyhd8ed1ab_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/whichcraft-0.6.1-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/yapf-0.31.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/zipp-3.6.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/zstd-1.5.0-h6255e5f_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/absl-py-1.0.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/autopep8-1.6.0-pyhd8ed1ab_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/babel-2.9.1-pyh44b312d_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/brotli-1.0.9-h8ffe710_6.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/certifi-2021.10.8-py37h03978a9_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/cffi-1.15.0-py37hd8e9650_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/chardet-4.0.0-py37h03978a9_2.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/convertdate-2.3.2-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/cython-0.29.26-py37hf2a7229_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/debugpy-1.5.1-py37hf2a7229_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/docutils-0.15.2-py37h03978a9_3.tar.bz2
```

```
https://conda.anaconda.org/conda-forge/win-64/ephem-4.1.3-py37hcc03f2d_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/freetype-2.10.4-h546665d_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/frozenlist-1.2.0-py37hcc03f2d_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/future-0.18.2-py37h03978a9_4.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/importlib-metadata-4.10.0-py37h03978a9_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/importlib_resources-5.4.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/intervaltree-3.0.2-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/jedi-0.18.1-py37h03978a9_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/jellyfish-0.8.9-py37hcc03f2d_2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/kiwisolver-1.3.2-py37h8c56517_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/lazy-object-proxy-1.7.1-py37hcc03f2d_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libcblas-3.9.0-12_win64_mkl.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libcurl-7.80.0-h789b8ee_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/liblapack-3.9.0-12_win64_mkl.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libtiff-4.3.0-hd413186_2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-zlib-1.2.8-10.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/markupsafe-2.0.1-py37hcc03f2d_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/matplotlib-inline-0.1.3-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/mistune-0.8.4-py37hcc03f2d_1005.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/multidict-5.2.0-py37hcc03f2d_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/mypy_extensions-0.4.3-py37h03978a9_4.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/packaging-21.3-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pexpect-4.8.0-pyh9f0ad1d_2.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/plotly-5.4.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/psutil-5.8.0-py37hcc03f2d_2.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pyasn1-modules-0.2.7-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pydocstyle-6.1.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pyqt5-sip-4.19.18-py37hf2a7229_8.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pyrsistent-0.18.0-py37hcc03f2d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/python-dateutil-2.8.2-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pyu2f-0.1.5-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pywin32-302-py37hcc03f2d_2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pywin32-ctypes-0.2.0-py37h03978a9_1004.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pywinpty-1.1.6-py37h7f67f24_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pyyaml-6.0-py37hcc03f2d_3.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pyzmq-22.3.0-py37hcce574b_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/qt-5.12.9-h5909a2a_4.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/qtawesome-1.1.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/rsa-4.8-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/rtree-0.9.7-py37h13cc57e_3.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/setuptools-59.7.0-py37h03978a9_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/tensorboard-data-server-0.6.0-
py37h03978a9_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/three-merge-0.1.1-pyh9f0ad1d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/tinycss2-1.1.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/tornado-6.1-py37hcc03f2d_2.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/tqdm-4.62.3-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/typed-ast-1.5.1-py37hcc03f2d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/typing-extensions-3.10.0.2-hd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/ujson-5.0.0-py37hf2a7229_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/unicodedata2-14.0.0-py37hcc03f2d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/werkzeug-2.0.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/win_inet_pton-1.1.0-py37h03978a9_3.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/wrapt-1.13.3-py37hcc03f2d_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/aiosignal-1.2.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/argon2-cffi-21.1.0-py37hcc03f2d_2.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/arrow-1.2.1-pyhd8ed1ab_0.tar.bz2
```

```
https://conda.anaconda.org/conda-forge/win-64/astroid-2.9.0-py37h03978a9_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/async-timeout-4.0.2-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/backports.functools_lru_cache-1.6.4-
pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/binaryornot-0.4.4-py_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/bleach-4.1.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/brotlipy-0.7.0-py37hcc03f2d_1003.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/click-8.0.3-py37h03978a9_1.tar.bz2
https://repo.anaconda.com/pkgs/main/win-64/cryptography-36.0.0-py37h21b164f_0.conda
https://conda.anaconda.org/conda-forge/win-64/curl-7.80.0-h789b8ee_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/flake8-4.0.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/fonttools-4.28.5-py37hcc03f2d_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/grpcio-1.43.0-py37h04d2302_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/hdf5-1.12.1-nompi_h2a0e4a3_103.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/holidays-0.11.3.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/importlib_metadata-4.10.0-hd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/isort-5.10.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/jinja2-3.0.3-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/joblib-1.1.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/jsonschema-4.3.2-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/jupyter_core-4.9.1-py37h03978a9_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/lcms2-2.12-h2a16943_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/liblapacke-3.9.0-12_win64_mkl.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libpython-2.1-py37h03978a9_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/lunarcalendar-0.0.9-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-binutils-2.25.1-5.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/markdown-3.3.6-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/numpy-1.21.4-py37h940b05c_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/openjpeg-2.4.0-hb211442_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pip-21.3.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/protobuf-3.19.1-py37hf2a7229_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pygments-2.10.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pynacl-1.4.0-py37ha54c9ec_3.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pyqt-impl-5.12.3-py37hf2a7229_8.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pysocks-1.7.1-py37h03978a9_4.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/python-lsp-jsonrpc-1.0.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/qstylizer-0.2.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/terminado-0.12.1-py37h03978a9_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/watchdog-2.1.6-py37h03978a9_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/yarl-1.7.2-py37hcc03f2d_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/aiohttp-3.8.1-py37hcc03f2d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/argcomplete-1.12.3-pyhd8ed1ab_2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/bcrypt-3.2.0-py37hcc03f2d_2.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/black-21.12b0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/blas-devel-3.9.0-12_win64_mkl.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/cftime-1.5.1.1-py37hec80d1f_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/jinja2-time-0.2.0-py_2.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/jupyter_client-7.1.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/jupyterlab_pygments-0.1.2-pyh9f0ad1d_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/keyring-23.4.0-py37h03978a9_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/libnetcdf-4.8.1-nompi_h1cc8e9d_101.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-gcc-5.3.0-6.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/nbformat-5.1.3-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/oauthlib-3.1.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pandas-1.3.5-py37h9386db6_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pillow-8.4.0-py37hd7d9ad0_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pluggy-1.0.0-py37h03978a9_2.tar.bz2
```

```
https://conda.anaconda.org/conda-forge/noarch/pylint-2.12.2-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pyopenssl-21.0.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pyqtchart-5.12-py37hf2a7229_8.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pyqtwebengine-5.12.1-py37hf2a7229_8.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/scipy-1.7.3-py37hb6553fb_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/wcwidth-0.2.5-pyh9f0ad1d_2.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/blas-2.112-mkl.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-gcc-ada-5.3.0-6.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-gcc-fortran-5.3.0-6.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-gcc-objc-5.3.0-6.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/matplotlib-base-3.5.1-py37h4a79c79_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/nbclient-0.5.9-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/netcdf4-1.5.8-nompi_py37h8860187_101.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/paramiko-2.8.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/patsy-0.5.2-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/prompt-toolkit-3.0.24-pyha770c72_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pyqt-5.12.3-py37h03978a9_8.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/python-lsp-server-1.3.3-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/scikit-learn-1.0.1-py37hcabfae0_3.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/urllib3-1.26.7-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/xarray-0.20.2-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/arviz-0.11.4-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/filterpy-1.4.5-py_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/ipython-7.30.1-py37h03978a9_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/lightgbm-3.3.1-py37hf2a7229_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-toolchain-5.3.0-7.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/nbconvert-6.3.0-py37h03978a9_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/pyls-spyder-0.4.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/python-lsp-black-1.0.1-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/pytorch/win-64/pytorch-1.10.1-py3.7_cuda11.1_cudnn8_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/requests-2.26.0-pyhd8ed1ab_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/statsmodels-0.13.1-py37hec80d1f_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/cookiecutter-1.6.0-py37_1000.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/google-auth-2.3.3-pyh6c4a22f_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/ipykernel-6.6.0-py37h4038f58_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/m2w64-toolchain_win-64-2.4.0-0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/plotly_express-0.4.1-py_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pmdarima-1.8.4-py37hcc03f2d_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/requests-oauthlib-1.3.0-pyh9f0ad1d_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/u8darts-0.14.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/google-auth-oauthlib-0.4.6-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/notebook-6.4.6-pyha770c72_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/pystan-2.19.1.1-py37h631819c_3.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/qtconsole-5.2.2-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/spyder-kernels-2.2.0-py37h03978a9_0.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/prophet-1.0.1-py37h7813e69_3.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/tensorboard-2.7.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/u8darts-all-0.14.0-pyhd8ed1ab_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/numpydoc-1.1.0-py_1.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/sphinx-4.3.2-pyh6c4a22f_0.tar.bz2
https://conda.anaconda.org/conda-forge/noarch/sphinxcontrib-serializinghtml-1.1.5-
pyhd8ed1ab_1.tar.bz2
https://conda.anaconda.org/conda-forge/win-64/spyder-5.2.1-py37h03978a9_0.tar.bz2
```