EVCI siting tool

MP Ensystems Advisory Pvt Ltd

Aug 2022

Table of contents

# 1. EVCI Siting Tool

A python based mathematical framework to assess utilization capability of EVCI sites

## 1.1 Install

pip install evci\_tool

## 1.2 How to use

The model inputs are provided in the form of excel files (xlsx). The analyze\_sites() is the entry level function and completes the analysis for specified corridor

from evci\_tool.config import \*  
from evci\_tool.model import \*  
from evci\_tool.analysis import \*  
  
analyze\_sites ('chandigarh\_leh')

Initial Analysis  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Number of sites: 48/48

100%|███████████████████████████████████████████████| 48/48 [00:04<00:00, 10.37it/s]

Total capex charges = INR Cr 8.99  
Total opex charges = INR Cr 42.25  
Total Margin = INR Cr nan

:::

# 2. EVCI config

**API**: The inputs are specified through 4 excel sheets for frieght corridor analysis. The python code reads these inputs parameters from the following excel sheets for analysis.

1. model.xlsx - This contains all the global parameters that are model specific. They remain valid for every corridor that needs analysis.
2. sites.xlsx - This contains a list of sites, with their latitude and longitude for analysis. This is an initial list of sites. This file needs to be created for each freight corridor
3. traffic.xlsx - This contains a typical traffic profile around the sites for each freight corridor.
4. grid.xlsx - This contains information about neighboring distribution transformers from where the chargers deployed at each site will draw power from.

from nbdev.showdoc import \*

# 3. Read input data

import os  
import numpy as np  
import pandas as pd  
  
  
def setup\_and\_read\_data(corridor:str, input\_path='input/', output\_path='output/'):  
 "This function sets up paths and reads input excel files for a specified corridor"  
  
 INPUT\_PATH = input\_path + corridor + '/'  
 OUTPUT\_PATH = output\_path + corridor + '/'  
  
 if not os.path.exists(OUTPUT\_PATH):  
 os.mkdir (output\_path + corridor)  
   
 model = pd.read\_excel(input\_path + "model.xlsx",sheet\_name=['charger\_specific','battery\_specific','others'])  
 sites = pd.read\_excel(INPUT\_PATH + "sites.xlsx",sheet\_name=['sites'])  
 traffic = pd.read\_excel(INPUT\_PATH + "traffic.xlsx",sheet\_name=['profile'])  
 grid = pd.read\_excel(INPUT\_PATH + "grid.xlsx",sheet\_name=['grid'])  
   
 return model, sites, traffic, grid, INPUT\_PATH, OUTPUT\_PATH

Arguments:

1. corridor: a string that identifies the corridor being analyzed (e.g. chandigarh\_leh)
2. input\_path: a string denoting base directory under which input files (xlsx and shape files) are available for analysis. Default is 'input/'
3. output\_path: a string denoting base directory under which output files will be stored. Default is 'output/'

Returns:

1. model: dataframe of model parameters (from model.xlsx)
2. sites: dataframe of sites (from sites.xlsx)
3. traffic: dataframe of traffic profile (from traffic.xlsx)
4. grid: dataframe of grid parameters (from grid.xlsx)
5. INPUT\_PATH: a string indicating the input\_path (e.g. input/chandigarh\_leh/)
6. OUTPUT\_PATH: a string indicating the output path (e.g. output/chandigarh\_leh/)

#example usage  
m,s,t,g,INPUT\_PATH,OUTPUT\_PATH = setup\_and\_read\_data('chandigarh\_leh')

# 4. Data availability check

Let’s check if the four excel sheets provided have the correctly named worksheets within them.

model\_sheets = ['charger\_specific', 'battery\_specific', 'others']  
sites\_sheets = ['sites']  
traffic\_sheets = ['profile']  
grid\_sheets = ['grid']  
  
def data\_availability\_check(m,s,t,g):   
 "This function checks if the excel files contain the mandatory worksheets."  
   
 retval = []  
   
 if list(m.keys()) != model\_sheets: retval.append('model')  
 if list(s.keys()) != sites\_sheets: retval.append('sites')  
 if list(t.keys()) != traffic\_sheets: retval.append('traffic')  
 if list(g.keys()) != grid\_sheets: retval.append('grid')  
   
 #assert data\_availability\_check(list(m.keys()),model\_sheets) == True, \  
 # f"model.xlsx must contain the sheets: {model\_sheets}"   
 #assert data\_availability\_check(list(s.keys()),['sites']) == True, \  
 # f"sites.xlsx must contain the sheet {sites\_sheets}"   
 #assert data\_availability\_check(list(t.keys()),['profile']) == True,   
 # f"traffic.xlsx must contain the sheet {traffic\_sheets}"   
 #assert data\_availability\_check(list(g.keys()),['grid']) == True, \  
 # f"grid.xlsx must contain the sheet {grid\_sheets}"  
   
 return retval

Arguments:

1. m: dataframe of model parameters (from model.xlsx)
2. s: dataframe of sites (from sites.xlsx)
3. t: dataframe of traffic profile (from traffic.xlsx)
4. g: dataframe of grid parameters (from grid.xlsx)

Returns:

A list of xlsx file names wiht missing mandatory sheets

data\_availability\_check(m,s,t,g)

[]

# 5. Data integrity check

Let’s now check if any of the mandatory columns in each of the worksheets are all empty!

def data\_integrity\_check(m,s,t,g, verbose=False):  
 "This function checks for integrity of excel data by checking missing values."  
 missing = []  
   
 for x in [m,s,t,g]:  
 tmpx = {}  
 for k in x.keys():  
 total = x[k].shape[0]  
 tmpx[k] = []  
 for c in x[k].columns:  
 if sum(pd.isna(x[k][c])) > 0:  
 if verbose: print(f"Column '{c}' of '{k}' has {sum(pd.isna(x[k][c]))}/{total} missing values")  
 tmpx[k].append(c)  
 missing.append(tmpx)  
   
 return missing

Arguments:

1. m: dataframe of model parameters (from model.xlsx)
2. s: dataframe of sites (from sites.xlsx)
3. t: dataframe of traffic profile (from traffic.xlsx)
4. g: dataframe of grid parameters (from grid.xlsx)

Returns:

A dictionary of missing columns with their corresponding xlsx filename and worksheet name.

data\_integrity\_check(m,s,t,g)

[{'charger\_specific': ['Range',  
 'User Input?',  
 'UoM',  
 '2W',  
 '3WS',  
 '4WS',  
 '4WF',  
 'Unnamed: 9',  
 '2W.1',  
 '3WS.1',  
 '4WS.1',  
 '4WF.1'],  
 'battery\_specific': ['Range', 'User Input?'],  
 'others': ['Range']},  
 {'sites': ['Address',  
 'Traffic congestion',  
 'Year for Site recommendation',  
 'Hoarding/Kiosk (1 is yes & 0 is no)',  
 'Hoarding margin',  
 'Kiosk margin',  
 'Available area',  
 'Upfront cost per sqm (land)',  
 'Yearly cost per sqm (land)',  
 'Upfront cost per sqm (kiosk)',  
 'Yearly cost per sqm (kiosk)',  
 'Upfront cost per sqm (hoarding)',  
 'Yearly cost per sqm (hoarding)',  
 'Battery swap available']},  
 {'profile': []},  
 {'grid': ['Longitude',  
 'Latitude',  
 'Tariff',  
 'Power Outage',  
 'Available load']}]

# verbose output  
data\_integrity\_check(m,s,t,g,verbose=True)

Column 'Range' of 'charger\_specific' has 6/17 missing values  
Column 'User Input?' of 'charger\_specific' has 7/17 missing values  
Column 'UoM' of 'charger\_specific' has 9/17 missing values  
Column '2W' of 'charger\_specific' has 9/17 missing values  
Column '3WS' of 'charger\_specific' has 9/17 missing values  
Column '4WS' of 'charger\_specific' has 9/17 missing values  
Column '4WF' of 'charger\_specific' has 9/17 missing values  
Column 'Unnamed: 9' of 'charger\_specific' has 17/17 missing values  
Column '2W.1' of 'charger\_specific' has 11/17 missing values  
Column '3WS.1' of 'charger\_specific' has 11/17 missing values  
Column '4WS.1' of 'charger\_specific' has 11/17 missing values  
Column '4WF.1' of 'charger\_specific' has 11/17 missing values  
Column 'Range' of 'battery\_specific' has 9/9 missing values  
Column 'User Input?' of 'battery\_specific' has 7/9 missing values  
Column 'Range' of 'others' has 8/8 missing values  
Column 'Address' of 'sites' has 48/48 missing values  
Column 'Traffic congestion' of 'sites' has 48/48 missing values  
Column 'Year for Site recommendation' of 'sites' has 48/48 missing values  
Column 'Hoarding/Kiosk (1 is yes & 0 is no)' of 'sites' has 48/48 missing values  
Column 'Hoarding margin' of 'sites' has 48/48 missing values  
Column 'Kiosk margin' of 'sites' has 48/48 missing values  
Column 'Available area' of 'sites' has 48/48 missing values  
Column 'Upfront cost per sqm (land)' of 'sites' has 48/48 missing values  
Column 'Yearly cost per sqm (land)' of 'sites' has 48/48 missing values  
Column 'Upfront cost per sqm (kiosk)' of 'sites' has 48/48 missing values  
Column 'Yearly cost per sqm (kiosk)' of 'sites' has 48/48 missing values  
Column 'Upfront cost per sqm (hoarding)' of 'sites' has 48/48 missing values  
Column 'Yearly cost per sqm (hoarding)' of 'sites' has 48/48 missing values  
Column 'Battery swap available' of 'sites' has 48/48 missing values  
Column 'Longitude' of 'grid' has 1/4 missing values  
Column 'Latitude' of 'grid' has 1/4 missing values  
Column 'Tariff' of 'grid' has 4/4 missing values  
Column 'Power Outage' of 'grid' has 4/4 missing values  
Column 'Available load' of 'grid' has 4/4 missing values

[{'charger\_specific': ['Range',  
 'User Input?',  
 'UoM',  
 '2W',  
 '3WS',  
 '4WS',  
 '4WF',  
 'Unnamed: 9',  
 '2W.1',  
 '3WS.1',  
 '4WS.1',  
 '4WF.1'],  
 'battery\_specific': ['Range', 'User Input?'],  
 'others': ['Range']},  
 {'sites': ['Address',  
 'Traffic congestion',  
 'Year for Site recommendation',  
 'Hoarding/Kiosk (1 is yes & 0 is no)',  
 'Hoarding margin',  
 'Kiosk margin',  
 'Available area',  
 'Upfront cost per sqm (land)',  
 'Yearly cost per sqm (land)',  
 'Upfront cost per sqm (kiosk)',  
 'Yearly cost per sqm (kiosk)',  
 'Upfront cost per sqm (hoarding)',  
 'Yearly cost per sqm (hoarding)',  
 'Battery swap available']},  
 {'profile': []},  
 {'grid': ['Longitude',  
 'Latitude',  
 'Tariff',  
 'Power Outage',  
 'Available load']}]

# 6. Read global variables from xlsx

def read\_globals(m,s,t,g):  
 "This function returns all global parameters read from the xlsx."  
   
 r = {}  
 df\_c = m['charger\_specific']  
 df\_b = m['battery\_specific']  
 df\_o = m['others']  
  
 r['M'] = df\_c[df\_c['Parameter']=='vehicle\_types']['Value'].iloc[0].split(',')  
 r['charger\_types'] = r['M'] # to be set from the UI  
 r['years\_of\_analysis'] = max(list(map(int,df\_o[df\_o['Parameter']=='Analysis duration']['Value'].iloc[0].split(','))))  
 r['C'] = df\_c[df\_c['Parameter']=='charger\_types']['Value'].iloc[0].split(',')  
 r['capex\_2W'] = int(df\_c[df\_c['Parameter']=='capex\_2W']['Value'].iloc[0])  
 r['capex\_3WS'] = int(df\_c[df\_c['Parameter']=='capex\_3WS']['Value'].iloc[0])  
 r['capex\_4WS'] = int(df\_c[df\_c['Parameter']=='capex\_4WS']['Value'].iloc[0])  
 r['capex\_4WF'] = int(df\_c[df\_c['Parameter']=='capex\_4WF']['Value'].iloc[0])  
 r['Kj'] = eval(df\_c[df\_c['Parameter']=='Kj']['Value'].iloc[0])  
 r['Dj'] = eval(df\_c[df\_c['Parameter']=='Dj']['Value'].iloc[0])  
 r['Hj'] = eval(df\_c[df\_c['Parameter']=='Hj']['Value'].iloc[0])  
 r['Qj'] = eval(df\_c[df\_c['Parameter']=='Qj']['Value'].iloc[0])  
 r['tj'] = eval(df\_c[df\_c['Parameter']=='tj']['Value'].iloc[0])  
 r['Mj'] = eval(df\_c[df\_c['Parameter']=='Mj']['Value'].iloc[0])  
 r['Gk'] = eval(df\_c[df\_c['Parameter']=='Gk']['Value'].iloc[0])  
 r['K'] = r['years\_of\_analysis']  
  
 r['N'] = 500  
 r['Ng'] = 0  
  
 r['timeslots'] = {k: 24/v for k, v in r['tj'].items()}  
 timeslots = r['timeslots']  
   
 r['Nc'] = s['sites'].shape[0]  
 Nc = r['Nc']  
  
 r['Gi'] = [0]\*Nc  
 r['di'] = [0]\*Nc  
 r['Wi'] = [0]\*Nc  
 r['Ri'] = [0]\*Nc  
 r['Ai'] = [50]\*Nc  
 r['Li'] = [1500]\*Nc  
 r['Bi'] = [0.25 \* 3.5 \* 24 \* 365]\*Nc # 25% of Rs 3.5/KWh per year  
  
 r['Eg'] = {k: [5.5] \* int(v) for k, v in timeslots.items()}  
 r['Er'] = {k: [0] \* int(v) for k, v in timeslots.items()}  
 r['Mg'] = {k: [5.5 \* 0.15] \* int(v) for k, v in timeslots.items()}  
 r['Mr'] = {k: [0] \* int(v) for k, v in timeslots.items()}  
 r['l'] = {k: [1] \* int(v) for k, v in timeslots.items()}  
  
 r['hoarding\_cost'] = 900000 #@param {type:"slider", min:500000, max:1000000, step:50000}  
 r['kiosk\_cost'] = 180000 #@param {type:"slider", min:100000, max:200000, step:20000}  
  
 r['CH'] = [r['hoarding\_cost']]\*Nc  
 r['CK'] = [r['kiosk\_cost']]\*Nc  
  
 r['MH'] = [s['sites'].loc[i]['Hoarding margin'] for i in range(Nc)]  
 r['MK'] = [0.15]\*Nc  
  
 #Traffic Model  
  
 r['year1\_conversion'] = float(df\_o[df\_o['Parameter']=='year1\_conversion']['Value'].iloc[0])  
 r['year2\_conversion'] = float(df\_o[df\_o['Parameter']=='year2\_conversion']['Value'].iloc[0])  
 r['year3\_conversion'] = float(df\_o[df\_o['Parameter']=='year3\_conversion']['Value'].iloc[0])  
  
 r['pj'] = {1: r['year1\_conversion'],   
 2: r['year2\_conversion'],   
 3: r['year3\_conversion']}  
  
 r['Pj'] = max(r['pj'].values())   
  
 # peak vehicles through crowded junctions in a day ~ 1.5L  
  
 peak\_traffic = [  
 4826, 4826, 5228, 5228, 5228, 5630, 6434, 6836, 6836,   
 6434, 6032, 6032, 6032, 6032, 6434, 6836, 7239, 8043,   
 8043, 8043, 6836, 6032, 5630, 5228   
 ]  
  
 # Average traffic approx 80% of peak  
 avg\_traffic = [i\*.8 for i in peak\_traffic]  
 # 2W and 4W assumed to be 60% and 20% respectively  
 avg\_traffic\_2W = [i\*.6 for i in avg\_traffic]  
 avg\_traffic\_3W = [i\*.4 for i in avg\_traffic]  
 avg\_traffic\_4W = [i\*.2 for i in avg\_traffic]  
 djworking\_hourly\_2W = [i/5 for i in avg\_traffic\_2W]  
 djworking\_hourly\_3WS = djworking\_hourly\_2W  
 djworking\_hourly = [i/5 for i in avg\_traffic\_4W]  
 djworking\_half\_hourly = [val for val in djworking\_hourly   
 for \_ in (0, 1)]  
 djworking\_one\_and\_half\_hourly = list(np.mean(np.array(djworking\_half\_hourly).reshape(-1, 3), axis=1))  
 r['djworking\_hourly\_2W'] = djworking\_hourly\_2W  
 r['djworking\_hourly\_3WS'] = djworking\_hourly\_2W  
 r['djworking\_hourly'] = djworking\_hourly  
 r['djworking\_half\_hourly'] = djworking\_half\_hourly  
 r['djworking\_one\_and\_half\_hourly'] = djworking\_one\_and\_half\_hourly  
   
 djworking = {}  
 djworking['2W'] = [np.round(i,0) for i in djworking\_hourly\_2W]  
 djworking['3WS'] = [np.round(i,0) for i in djworking\_hourly\_3WS]  
 djworking['4WF'] = [np.round(i,0) for i in djworking\_half\_hourly]  
 djworking['4WS'] = [np.round(i,0) for i in djworking\_one\_and\_half\_hourly]  
 r['djworking'] = djworking  
   
 holiday\_percentage = .3 #@param {type:"slider", min:0, max:1, step:0.1}  
 r['holiday\_percentage'] = holiday\_percentage  
   
 djholiday = {}  
 djholiday['2W'] = [np.round(i\*holiday\_percentage,0) for i in djworking\_hourly\_2W]  
 djholiday['3WS'] = [np.round(i\*holiday\_percentage,0) for i in djworking\_hourly\_3WS]  
 djholiday['4WF'] = [np.round(i\*holiday\_percentage,0) for i in djworking\_half\_hourly]  
 djholiday['4WS'] = [np.round(i\*holiday\_percentage,0) for i in djworking\_one\_and\_half\_hourly]  
 r['djholiday'] = djholiday  
  
 fast\_charging = float(df\_o[df\_o['Parameter']=='slow charger margin']['Value'].iloc[0])  
 slow\_charging = float(df\_o[df\_o['Parameter']=='fast charger margin']['Value'].iloc[0])  
 r['fast\_charging'] = fast\_charging  
 r['slow\_charging'] = slow\_charging  
  
 r['qjworking'] = {'4WS': [slow\_charging] \* int(timeslots['4WS']),   
 '4WF': [fast\_charging] \* int(timeslots['4WF']),   
 '3WS': [fast\_charging + slow\_charging] \* int(timeslots['3WS']),   
 '2W' : [fast\_charging + slow\_charging] \* int(timeslots['2W']), }  
 r['qjholiday'] = {'4WS': [slow\_charging] \* int(timeslots['4WS']),   
 '4WF': [fast\_charging] \* int(timeslots['4WF']),   
 '3WS': [fast\_charging + slow\_charging] \* int(timeslots['3WS']),   
 '2W' : [fast\_charging + slow\_charging] \* int(timeslots['2W']), }  
  
 r['Cij'] = {'2W': [4]\*Nc, '3WS':[1]\*Nc, '4WS': [1]\*Nc, '4WF':[1]\*Nc}  
   
 return r

r = read\_globals(m,s,t,g)  
r['Kj']

{'2W': 2500, '3WS': 112000, '4WS': 250000, '4WF': 1500000}

# 7. EVCI siting model

**API**: The mathematical formulation is defined here.

from nbdev.showdoc import \*

import numpy as np  
import pandas as pd  
import geopandas as gpd  
  
import shapely  
  
import os  
from tqdm import tqdm  
  
import matplotlib.pyplot as plt  
  
from scipy.cluster.vq import kmeans2, whiten  
from scipy.cluster.hierarchy import dendrogram, linkage  
from scipy.cluster.hierarchy import fcluster  
  
from evci\_tool.config import \*  
  
import warnings  
warnings.filterwarnings("ignore")

def score(r,s\_df\_distances,j,i,hj,k,backoff=True, backoff\_factor=1):  
 "This function computes the utilization score of each site."  
   
 distance\_from\_i = s\_df\_distances[s\_df\_distances > 0][i].sort\_values()/1e3  
 closer\_to\_i = distance\_from\_i[distance\_from\_i <= 5.0]  
 try:  
 congestion = float(s\_df.loc[i]['Traffic congestion'])  
 except:  
 congestion = 1.0  
  
 nw = r['qjworking'][j][hj] \* r['djworking'][j][hj] \* r['pj'][k] \* congestion  
 nh = r['qjholiday'][j][hj] \* r['djholiday'][j][hj] \* r['pj'][k] \* congestion  
  
 if backoff:  
 for el in closer\_to\_i:  
 nw \*= (1 - np.exp(-el\*backoff\_factor))  
 nh \*= (1 - np.exp(-el\*backoff\_factor))  
  
 tw = th = 0  
 if (r['Cij'][j][i] > 0): tw = nw \* (r['tj'][j]/r['Cij'][j][i])  
 if (r['Cij'][j][i] > 0): th = nh \* (r['tj'][j]/r['Cij'][j][i])  
  
 uw = uh = r['tj'][j]  
 if (tw <= r['tj'][j]): uw = tw   
 if (th <= r['tj'][j]): uh = th  
  
 vw = vh = 0  
 if (tw > r['tj'][j]): vw = (tw - r['tj'][j]) \* (r['Cij'][j][i]/r['tj'][j])  
 if (th > r['tj'][j]): vh = (th - r['tj'][j]) \* (r['Cij'][j][i]/r['tj'][j])  
  
 norm\_uw, norm\_uh = uw/r['tj'][j], uh/r['tj'][j]  
 norm\_vw = vw/nw  
 norm\_vh = vh/nh  
  
 return norm\_uw, norm\_uh, norm\_vw, norm\_vh

Arguments:

1. r: a dictionary of global parameters read from the xlsx files.
2. s\_df\_distances: a dataframe of Euclidean distances of each site from all others. (NxN matrix)
3. j: string indicating specific charger type
4. i: integer indicating a specific site
5. hj:
6. k: integer year (1, 2 or 3 of the policy)
7. backoff: a boolean indicating whether backoff should be incorporated
8. backoff\_factor: a float weighting factor for the backoff (mostly empirically selected)

Returns:

1. norm\_uw: float indicating normalized utilization on a typical working day
2. norm\_uh: float indicating normalized utilization on a typical holiday
3. norm\_vw: float indicating number of vehicles not utilizing charging on a working day
4. norm\_vh: float indicating number of vehicles not utilizing charging on a holiday

def capex(r,i):  
 "This function computes the capex requirements of each site"  
 retval = 0  
 for j in r['C']:  
 retval += r['Cij'][j][i]\*r['Kj'][j] + r['Wi'][i] \* r['di'][i] \* r['Cij'][j][i]  
 return retval

Arguments:

1. r: a dictionary of global parameters read from the xlsx files.
2. i: integer indicating a specific site

Returns:

integer. Capex value for a given site

def opex(r,s\_df\_distances,i):  
 "This function computes the opex for each site."  
 op\_e = 0  
 op\_l = 0  
  
 for k in r['Gk']:  
 for j in r['C']:  
 for h in range(int(r['timeslots'][j])):  
 sw, sh, \_, \_ = score (r,s\_df\_distances,j,i,h,k)  
 op\_e += 300 \* r['Cij'][j][i] \* sw \* r['tj'][j] \* r['Dj'][j] \* (r['l'][j][h] \* r['Eg'][j][h] + (1-r['l'][j][h]) \* r['Er'][j][h])  
 op\_e += 65 \* r['Cij'][j][i] \* sh \* r['tj'][j] \* r['Dj'][j] \* (r['l'][j][h] \* r['Eg'][j][h] + (1-r['l'][j][h]) \* r['Er'][j][h])  
 op\_l = r['Li'][i] \* r['Ai'][i] + r['CH'][i] + r['CK'][i]  
 return op\_e + op\_l

Arguments:

1. r: a dictionary of global parameters read from the xlsx files.
2. s\_df\_distances: a dataframe of Euclidean distances of each site from all others. (NxN matrix)
3. i: integer indicating a specific site

Returns:

integer opex value for a given site

def margin(r,s\_df\_distances,i):  
 "This function computes the margins per site."  
 margin\_e = 0  
 margin\_l = 0  
  
 for k in r['Gk']:  
 for j in r['C']:  
 for h in range(int(r['timeslots'][j])):  
 sw, sh, \_, \_ = score (r,s\_df\_distances,j,i,h,k)  
 margin\_e += 300 \* r['Cij'][j][i] \* sw \* r['tj'][j] \* r['Dj'][j] \* (r['l'][j][h] \* r['Mg'][j][h] + (1-r['l'][j][h]) \* r['Mr'][j][h])  
 margin\_e += 65 \* r['Cij'][j][i] \* sh \* r['tj'][j] \* r['Dj'][j] \* (r['l'][j][h] \* r['Mg'][j][h] + (1-r['l'][j][h]) \* r['Mr'][j][h])  
 margin\_l = r['Bi'][i] \* r['Ai'][i] + r['MH'][i] + r['MK'][i]  
 return margin\_e + margin\_l

Arguments:

1. r: a dictionary of global parameters read from the xlsx files.
2. s\_df\_distances: a dataframe of Euclidean distances of each site from all others. (NxN matrix)
3. i: integer indicating a specific site

Returns:

integer margin value of a site

def run\_analysis(m,s,t,g,s\_df,backoff\_factor=1):  
 "This function runs analysis for a given set of sites."  
  
 r = read\_globals(m,s,t,g)  
   
 u\_df = pd.DataFrame(columns=['utilization',   
 'unserviced',   
 'capex',   
 'opex',   
 'margin',   
 'max vehicles',   
 'estimated vehicles'  
 ])   
 s\_df\_crs = gpd.GeoDataFrame(s\_df, crs='EPSG:4326')  
 s\_df\_crs = s\_df\_crs.to\_crs('EPSG:5234')  
 s\_df\_distances = s\_df\_crs.geometry.apply(lambda g: s\_df\_crs.distance(g))   
   
 Nc = s\_df.shape[0]  
   
 for i in tqdm(range(Nc)):  
 max\_vehicles = 0  
 for j in r['C']:  
 max\_vehicles += r['timeslots'][j]\*r['Cij'][j][i]  
 op\_e = 0  
 op\_l = 0  
 margin\_e = 0  
 margin\_l = 0  
 year\_u\_avg = np.array([])  
 year\_v\_avg = np.array([])  
 for k in r['Gk']:  
 chargertype\_u\_avg = np.array([])  
 chargertype\_v\_avg = np.array([])  
 for j in r['C']:  
 uw\_day\_avg = np.array([])  
 uh\_day\_avg = np.array([])  
 vw\_day\_avg = np.array([])  
 vh\_day\_avg = np.array([])   
 for h in range(int(r['timeslots'][j])):  
 uw, uh, vw, vh = score (r,s\_df\_distances,j,i,h,k,backoff\_factor=backoff\_factor)  
 uw\_day\_avg = np.append(uw\_day\_avg, uw)  
 uh\_day\_avg = np.append(uh\_day\_avg, uh)  
 vw\_day\_avg = np.append(vw\_day\_avg, vw)  
 vh\_day\_avg = np.append(vh\_day\_avg, vh)   
 op\_e += 300 \* r['Cij'][j][i] \* uw \* r['tj'][j] \* r['Dj'][j] \* (r['l'][j][h] \* r['Eg'][j][h] + (1-r['l'][j][h]) \* r['Er'][j][h])  
 op\_e += 65 \* r['Cij'][j][i] \* uh \* r['tj'][j] \* r['Dj'][j] \* (r['l'][j][h] \* r['Eg'][j][h] + (1-r['l'][j][h]) \* r['Er'][j][h])   
 margin\_e += 300 \* r['Cij'][j][i] \* uw \* r['tj'][j] \* r['Dj'][j] \* (r['l'][j][h] \* r['Mg'][j][h] + (1-r['l'][j][h]) \* r['Mr'][j][h])  
 margin\_e += 65 \* r['Cij'][j][i] \* uh \* r['tj'][j] \* r['Dj'][j] \* (r['l'][j][h] \* r['Mg'][j][h] + (1-r['l'][j][h]) \* r['Mr'][j][h])   
 weighted\_u = (300.0\*uw\_day\_avg.mean() + 65.0\*uh\_day\_avg.mean()) / 365.0  
 weighted\_v = (300.0\*vw\_day\_avg.mean() + 65.0\*vh\_day\_avg.mean()) / 365.0  
 chargertype\_u\_avg = np.append(chargertype\_u\_avg, weighted\_u)  
 chargertype\_v\_avg = np.append(chargertype\_v\_avg, weighted\_v)  
 year\_u\_avg = np.append(year\_u\_avg, chargertype\_u\_avg.mean())  
 year\_v\_avg = np.append(year\_v\_avg, chargertype\_v\_avg.mean())  
 op\_l += r['Li'][i] \* r['Ai'][i] + r['CH'][i] + r['CK'][i]  
 margin\_l += r['Bi'][i] \* r['Ai'][i] + r['MH'][i] + r['MK'][i]  
 site\_capex = capex(r,i)  
 estimated\_vehicles = np.round(year\_u\_avg.mean()\*max\_vehicles,0)  
 u\_df.loc[i] = [ year\_u\_avg.mean(),   
 year\_v\_avg.mean(),  
 site\_capex,  
 op\_e + op\_l,  
 margin\_e + margin\_l,  
 max\_vehicles,  
 estimated\_vehicles  
 ]  
 return u\_df

Arguments:

m,s,t,g,s\_df,backoff\_factor=1

1. m: dataframe of model parameters (from model.xlsx)
2. s: dataframe of sites (from sites.xlsx)
3. t: dataframe of traffic profile (from traffic.xlsx)
4. g: dataframe of grid parameters (from grid.xlsx)
5. s\_df: pre-processed geopandas dataframe with each point stored as shapely point object
6. backoff\_factor: a float value of backoff to cater for neighborhood (empirical)

Returns:

A dataframe of utilization values for all sites.

# 8. EVCI maps

**API**: The API includes functions to read shapfiles and render them. The rendered outputs can be saved as images.

from nbdev.showdoc import \*

import geopandas as gpd  
import matplotlib.pyplot as plt  
import numpy as np  
import shapely  
import os  
  
import pandas as pd  
import tqdm  
  
from evci\_tool.config import \*  
  
import warnings  
warnings.filterwarnings("ignore")

def show\_map(corridor:str):  
 "This function reads corresponding shape file and plots the points"  
   
 m,s,t,g,INPUT\_PATH,OUTPUT\_PATH = setup\_and\_read\_data (corridor)  
  
 s\_df = gpd.read\_file(INPUT\_PATH + 'shape\_files/' + corridor + '.shp')  
  
 data = s['sites']  
 data['Latitude'] = pd.to\_numeric(data['Latitude'])  
 data['Longitude'] = pd.to\_numeric(data['Longitude'])  
 data['geometry'] = [shapely.geometry.Point(xy) for xy in zip(data['Longitude'],  
 data['Latitude'])]  
  
 #@title Create grid  
 # total area for the grid  
 xmin, ymin, xmax, ymax= s\_df.total\_bounds  
 print('Bounding box: ', xmin,xmax,ymin,ymax)  
 # how many cells across and down  
 n\_cells=30  
 cell\_size = (xmax-xmin)/n\_cells  
  
 crs = s\_df.crs  
  
 grid\_cells = []  
 for x0 in np.arange(xmin, xmax+cell\_size, cell\_size ):  
 for y0 in np.arange(ymin, ymax+cell\_size, cell\_size):  
 # bounds  
 x1 = x0-cell\_size  
 y1 = y0+cell\_size  
 grid\_cells.append( shapely.geometry.box(x0, y0, x1, y1) )  
 cell = gpd.GeoDataFrame(grid\_cells, columns=['geometry'], crs=crs)  
  
 grid\_s\_df = gpd.overlay(s\_df, cell, how='intersection')  
   
 data\_df = {}  
 data\_df = gpd.GeoDataFrame(data,geometry=data['geometry'])  
  
 #total area  
 s\_df['geometry'].to\_crs(6933).map(lambda p: p.area/ 1e6).loc[0]  
  
 # plot the grid and the shapefiles  
 base = grid\_s\_df.plot(color='none', edgecolor='grey', alpha=0.4, figsize=(12,8))  
 s\_df.plot(ax=base, color='none', edgecolor='black')  
   
 data\_df.plot(ax=base, markersize=100, legend=True)

Arguments:

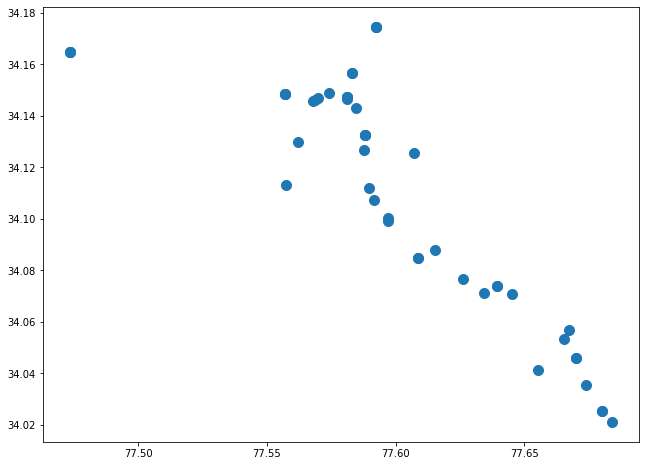
corridor: a string describing the corridor to be analyzed

Returns:

None

show\_map("chandigarh\_leh")

Bounding box: 77.473473 77.6839609 34.0211074 34.174654



# 9. EVCI analysis

**API**: The utilization scores for each site are computed here.

from nbdev.showdoc import \*

## 9.1 Import libraries

import numpy as np  
import pandas as pd  
import geopandas as gpd  
  
import shapely  
  
import os  
from tqdm import tqdm  
  
import matplotlib.pyplot as plt  
  
from scipy.cluster.vq import kmeans2, whiten  
from scipy.cluster.hierarchy import dendrogram, linkage  
from scipy.cluster.hierarchy import fcluster  
  
from evci\_tool.config import \*  
from evci\_tool.model import \*  
  
import warnings  
warnings.filterwarnings("ignore")

def run\_episode(m,s,t,g,s\_df,txt,OUTPUT\_PATH,corridor):  
 "This function runs a full episode of analysis on a set of sites."  
   
 print('\n' + txt.capitalize() + ' Analysis')  
 print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n')  
 total = s\_df.shape[0]  
   
 #s\_df = s\_df[s\_df['year 1'] == 1]  
 #s\_df = s\_df.reset\_index(drop=True)  
   
 Nc = s\_df.shape[0]  
 print(f'Number of sites: {Nc}/{total}')  
  
 #@title Compute scores  
  
 backoff\_factor = 2 #@param {type:"slider", min:1, max:5, step:1}  
  
 u\_df = run\_analysis(m,s,t,g,s\_df,backoff\_factor=backoff\_factor)  
  
 print(f'Total capex charges = INR Cr {sum(u\_df.capex)/1e7:.2f}')  
 print(f'Total opex charges = INR Cr {sum(u\_df.opex)/1e7:.2f}')  
 print(f'Total Margin = INR Cr {sum(u\_df.margin)/1e7:.2f}')   
  
 #@title Prepare data  
 s\_u\_df = s\_df.copy()  
  
 s\_u\_df['utilization'] = u\_df.utilization  
 s\_u\_df['unserviced'] = u\_df.unserviced  
 s\_u\_df['capex'] = u\_df.capex  
 s\_u\_df['opex'] = u\_df.opex  
 s\_u\_df['margin'] = u\_df.margin  
 s\_u\_df['max vehicles'] = u\_df['max vehicles']  
 s\_u\_df['estimated vehicles'] = u\_df['estimated vehicles']  
  
 #@title Save initial analysis to Excel  
 output\_df = s\_u\_df.copy()  
 output\_df.drop('geometry', axis=1, inplace=True)  
   
 # Save output dataframe as both xlsx and json  
 output\_df.to\_excel(OUTPUT\_PATH + '/' + txt + '\_evci\_analysis.xlsx')  
 output\_df.to\_json(OUTPUT\_PATH + '/' + txt + '\_evci\_analysis.json', orient='columns')  
   
 return s\_u\_df

Arguments:

1. m: dataframe of model parameters (from model.xlsx)
2. s: dataframe of sites (from sites.xlsx)
3. t: dataframe of traffic profile (from traffic.xlsx)
4. g: dataframe of grid parameters (from grid.xlsx)
5. s\_df: pre-processed geopandas dataframe with each point stored as shapely point object
6. txt: a string that identifies the episode (e.g. initial, final, with\_cluster etc)
7. OUTPUT\_PATH: the directory path where the generated output files will be stored
8. corridor: a string that identifies the corridor being analyzed (e.g. chandigarh\_leh)

Returns:

A pandas dataframe: s\_u\_df is a dataframe with computed utilization values for each site.

def analyze\_sites(corridor:str, cluster:bool=False, use\_defaults=False):  
 "The function analyzes sites specified as part of a corridor."  
  
 #@title Read data from excel sheets  
 model,site,traffic,grid, INPUT\_PATH, OUTPUT\_PATH = setup\_and\_read\_data(corridor)  
   
 #check if mandatory worksheets in xlsx files are available  
 avail = data\_availability\_check(model,site,traffic,grid)  
 assert len(avail) == 0, f"{avail} sheets missing from the xlsx. Please try again."   
   
 #check if any missingness  
 missing = data\_integrity\_check(model,site,traffic,grid)  
 #assert len(missing) > 0, f"{missing} sheets contain missing data."   
  
 #if missing values found, defaults shall be assumed for debug purposes. This is not for production version  
 if use\_defaults and len(missing) > 0:  
 site['sites']['Traffic congestion'] = 1  
 site['sites']['Year for Site recommendation'] = 1  
 site['sites']['Hoarding/Kiosk (1 is yes & 0 is no)'] = 1  
 site['sites']['Hoarding margin'] = 270000   
   
 #@title Read required data sheets only  
 df = gpd.read\_file(INPUT\_PATH + '/shape\_files/' + corridor + '.shp')  
  
 data = site['sites']  
 data['Name'] = data['Name']  
 data['Latitude'] = pd.to\_numeric(data['Latitude'])  
 data['Longitude'] = pd.to\_numeric(data['Longitude'])  
 data['geometry'] = [shapely.geometry.Point(xy) for xy in   
 zip(data['Longitude'], data['Latitude'])]  
  
 data\_df = {}  
  
 data\_df = gpd.GeoDataFrame(data, geometry=data['geometry'])  
  
 s\_df = pd.DataFrame(columns=['Name',  
 'Latitude', 'Longitude',  
 'Traffic congestion',  
 'year 1',  
 'kiosk hoarding',  
 'hoarding margin',  
 'geometry'])  
  
 s\_df = s\_df.reset\_index(drop=True)  
  
 for i in range(data\_df.shape[0]):  
 s\_df.loc[i] = [  
 data\_df.loc[i].Name,   
 data\_df.loc[i].Latitude,   
 data\_df.loc[i].Longitude,   
 data\_df.loc[i]['Traffic congestion'],  
 data\_df.loc[i]['Year for Site recommendation'],  
 data\_df.loc[i]['Hoarding/Kiosk (1 is yes & 0 is no)'],  
 data\_df.loc[i]['Hoarding margin'],  
 data\_df.loc[i].geometry  
 ]   
  
 s\_u\_df = run\_episode(model,site,traffic,grid,s\_df,'initial',OUTPUT\_PATH, corridor)  
  
 #@title Threshold and cluster  
 if cluster:  
 #clustering\_candidates = s\_u\_df[(s\_u\_df.utilization <= 0.2) & (s\_u\_df['year 1'] == 1)]  
 clustering\_candidates = s\_u\_df[s\_u\_df.utilization <= 0.2]  
 print('candidates for clustering: ', clustering\_candidates.shape[0])  
 points = np.array((clustering\_candidates.apply(lambda x: list([x['Latitude'], x['Longitude']]),axis=1)).tolist())  
 Z = linkage (points, method='complete', metric='euclidean');  
 #plt.figure(figsize=(14,8))  
 #dendrogram(Z);  
 max\_d = 0.01  
 clusters = fcluster(Z, t=max\_d, criterion='distance')  
 clustered\_candidates = gpd.GeoDataFrame(clustering\_candidates)  
 #base = grid\_df.plot(color='none', alpha=0.2, edgecolor='black', figsize=(8,8))  
 #clustered\_candidates.plot(ax=base, column=clusters, legend=True)  
  
 #@title Build final list of sites  
 confirmed\_sites = s\_u\_df[s\_u\_df.utilization > 0.2]  
 print('confirmed sites with utilization > 20%: ', confirmed\_sites.shape[0])  
 if cluster:  
 val, ind = np.unique (clusters, return\_index=True)  
 clustered\_sites = clustered\_candidates.reset\_index(drop=True)  
 clustered\_sites = clustered\_sites.iloc[clustered\_sites.index.isin(ind)]  
 final\_list\_of\_sites = pd.concat([confirmed\_sites, clustered\_sites], axis=0)  
 else:  
 final\_list\_of\_sites = confirmed\_sites.copy()  
  
 if cluster:  
 print('final list: ', final\_list\_of\_sites.shape[0])  
 s\_df = final\_list\_of\_sites.copy()  
 s\_df = s\_df.reset\_index(drop=True)  
   
 s\_u\_df = run\_episode(model,site,traffic,grid,s\_df,'clustered',OUTPUT\_PATH, corridor)  
   
 return s\_u\_df

Arguments:

1. corridor: a string that identifies the corridor being analyzed (e.g. chandigarh\_leh)
2. cluster: a boolean flag that indicates whether clustering algorithm should be run. Default is False

Returns:

s\_u\_df: a dataframe containing the list of sites and their utilization numbers.

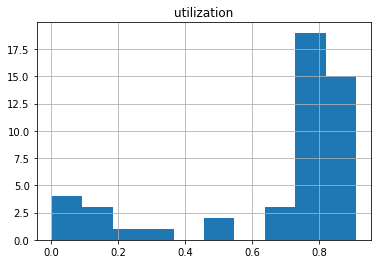
s\_u\_df = analyze\_sites('chandigarh\_leh', use\_defaults=True)

Initial Analysis  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Number of sites: 48/48

100%|███████████████████████████████████████████████| 48/48 [00:04<00:00, 10.32it/s]

Total capex charges = INR Cr 8.99  
Total opex charges = INR Cr 42.25  
Total Margin = INR Cr 13.25  
confirmed sites with utilization > 20%: 41

s\_u\_df.hist(column='utilization');



s\_u\_df.hist(column='unserviced')

array([[<AxesSubplot:title={'center':'unserviced'}>]], dtype=object)

