The_Challenge_Soil_organic_matter_content_(SOM)_prediction

March 10, 2022

1 The Challenge: Soil organic matter content (SOM) prediction (Spectroscopic Based Method)

1.1 Objectives:

1.1.1 Familiarize with the dataset (inspect and clean data).

1.1.2 Compare the estimates SOM with the corresponding data

from POLARIS soil database (Identify, Download, and ExtractPOLARIS data). ### Build two models and Evaluate their performance (using the sensor readings and lab values) to predict **SOM** of closeby fields.

1.2 Data:

1.2.1 STENON challege data

1.2.2 The POLARIS soil database

2 Setup program's requirements

2.1 Install the requirement packages

[ ]:
```
# Install the Python package to view the map on google maps
!pip install geemap --quiet

# # Install the Python package to plot charts
# !pip install matplotlib==3.1.1 --quiet

# Install the Python package to download data from website
!pip install wget --quiet

# Install the Python package to handle geo-vector and raster data
!pip install rasterstats --quiet

# # Install the Python package to created to manipulate data geo-vector data
# !pip install geopandas --quiet  # Might introduce conflict

# Important library for many geopython libraries
!apt install gdal-bin python-gdal python3-gdal
```

```
# Install rtree - Geopandas requirment
!apt install python3-rtree
# Install Geopandas
!pip install git+git://github.com/geopandas/geopandas.git --quiet
# Install descartes - Geopandas requirment
!pip install descartes --quiet
# Install Folium for Geographic data visualization
!pip install folium --quiet
# Install plotlyExpress
!pip install plotly_express --quiet
```

2.2 Import Standard libraries

[7]:
```
import os
import sys
import math
import logging
import argparse

from urllib.parse import urlparse

logger = logging.getLogger(__name__)
# import multiprocessing
```

2.3 Import Third party libraries

[3]:
```
import ee

import math
import wget
import json
import csv
import folium
import geojson
import numpy as np
import pandas as pd
from numpy import arange

import scipy as sp
from scipy.stats import pearsonr
from scipy.interpolate import interpn
from scipy.spatial.distance import pdist, squareform

import geemap
import geopandas as gpd
from rasterstats import point_query
```

```
from shapely.geometry import shape, GeometryCollection
from geojson import Feature, FeatureCollection, Point

# Third party imports for regression Model training and validation
import pickle
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

from sklearn import linear_model  # the Linear regression model
from sklearn.ensemble import RandomForestRegressor  # the Random forest␣
↪regression model
import xgboost as xgb  # the XGBoost regression model
from sklearn.linear_model import Ridge  # the Ridge regression model
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import RepeatedKFold
from sklearn.cross_decomposition import PLSRegression # the PLSRegression␣
↪regression model

# Third party imports to plot pretty figures
%matplotlib inline
import matplotlib as mpl
from matplotlib import cm
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from matplotlib.colors import Normalize

mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

font      = {'family':'arial', 'style':'normal', 'size':18}
axis_font = {'family':'arial', 'style':'normal', 'size':18}
plt.rc('font', **font)

import seaborn as sns
sns.set(style = 'whitegrid')

# Colab includes an extension that renders pandas dataframes into interactive␣
↪displays
# that can be filtered, sorted, and explored dynamically.
from google.colab import files
from google.colab import data_table
data_table.disable_dataframe_formatter()
data_table.DataTable.max_columns = 30
```

```
# Enable vertical scrolling in code block/output block
from IPython.display import display, HTML
js = "<script>$('.output_scroll').removeClass('output_scroll')</script>"
display(HTML(js))
```

```
<IPython.core.display.HTML object>
```

2.4 Earth Engine authentication

```
[ ]: # Trigger the authentication flow.
ee.Authenticate()

# Initialize the library.
ee.Initialize()
```

2.5 Google Driver authentication (optional)

```
[ ]: # from google.colab import drive
# drive.mount('/content/drive')
```

3 Challenge actions

3.1 Familiarize with the dataset

```
[5]: uploaded = files.upload()
csv_fn = next(iter(uploaded))
```

```
<IPython.core.display.HTML object>

Saving data.csv to data.csv
```

3.1.1 Read csv file input

```
[8]: def csv_valid(infile, arbitrary_number=2048):
  #arbitrary_number is an entirely arbitrary number. It just needs to be big␣
↪enough to read in at least two or three CSV rows.
  with open(infile, 'rb') as csvfile:
    try:
      sniffer = csv.Sniffer()
      dialect = sniffer.sniff(csvfile.read(arbitrary_number))
      # Perform various checks on the dialect (e.g., lineseparator, delimiter)␣
↪to make sure it's sane
      delimiter = dialect.delimiter
      lineseparator = dialect.lineterminator
      # Reset the read position back to the start of the file before reading␣
↪any entries.
      csvfile.seek(0)
```

```
    # Perform header check to make sure it's sane
    has_header = sniffer.has_header(csvfile.read(arbitrary_number))
    logger.info("CSV file is OK!")

    return delimiter, lineseparator, has_header
  except csv.Error:
    # File appears not to be in CSV format; move along
    logger.error ("Invalid CSV file")
    return None


st_path = "./{}".format(csv_fn)
# delimiter, _, _ = csv_valid(st_path)

if os.path.exists(st_path):
  df_data = pd.read_csv(st_path, sep=';')
  print(df_data.info)
else:
  print ("{} does not exists. The following code blocks won't be execused.␣
↪Please check!".format(st_path))
```

```
<bound method DataFrame.info of       measurement_ID                  lat_lng
location        som  nir_1370  \
0                  0  (41.2799111,-92.0301512)  field_A   2.038524  0.268311
1                  1  (41.2798524,-92.0294544)  field_A  17.443334  0.411358
2                  2  (41.2798545,-92.0294499)  field_A   2.227260  0.402013
3                  3   (41.2796759,-92.027938)  field_A   2.369437  0.506582
4                  4   (41.2796802,-92.027932)  field_A   2.345968  0.475707
…              …                       …       …         …        …
1068            1068  (41.2718012,-92.0390603)  field_B   1.919485  0.421228
1069            1069  (41.2717968,-92.0390726)  field_B   1.702884  0.451167
1070            1070  (41.2717812,-92.0374845)  field_B   1.555839  0.489369
1071            1071  (41.2717374,-92.0375016)  field_B   1.644490  0.507497
1072            1072  (41.2717458,-92.0374918)  field_B   1.591437  0.447848

      nir_1374  nir_1378  nir_1382  nir_1386  nir_1390  …  nir_2486  \
0     0.266145  0.262247  0.256983  0.250741  0.243620  …  0.122705
1     0.407535  0.401783  0.393936  0.384204  0.372926  …  0.165669
2     0.398604  0.393279  0.385631  0.375653  0.363615  …  0.140496
3     0.503565  0.498138  0.489784  0.478757  0.465691  …  0.197408
4     0.472190  0.466343  0.457769  0.446611  0.433262  …  0.173269
…        …        …        …        …        …   …        …
1068  0.417337  0.411337  0.402050  0.389677  0.374866  …  0.140396
1069  0.447152  0.440777  0.430529  0.416616  0.399818  …  0.136339
1070  0.485688  0.480174  0.471506  0.459823  0.445747  …  0.182598
1071  0.503908  0.497956  0.488233  0.475167  0.459762  …  0.172855
```

```
1072  0.444600  0.439651  0.431790  0.421130  0.408224  …  0.165309

      nir_2490  nir_2494  nir_2498  nir_2502  nir_2506  nir_2510  nir_2514  \
0     0.121610  0.120648  0.119822  0.119134  0.118585  0.118178  0.117913
1     0.164012  0.162577  0.161365  0.160374  0.159602  0.159048  0.158710
2     0.138907  0.137509  0.136309  0.135311  0.134522  0.133946  0.133589
3     0.195094  0.193042  0.191262  0.189763  0.188553  0.187640  0.187033
4     0.171223  0.169431  0.167901  0.166637  0.165644  0.164924  0.164483
…          …         …         …         …         …         …         …
1068  0.138827  0.137418  0.136181  0.135125  0.134262  0.133602  0.133156
1069  0.134797  0.133415  0.132209  0.131192  0.130378  0.129781  0.129415
1070  0.180544  0.178695  0.177061  0.175656  0.174493  0.173585  0.172944
1071  0.170890  0.169130  0.167590  0.166287  0.165238  0.164459  0.163967
1072  0.163435  0.161741  0.160241  0.158948  0.157876  0.157039  0.156451

      nir_2518  nir_2522
0     0.117794  0.117820
1     0.158586  0.158674
2     0.133455  0.133550
3     0.186739  0.186768
4     0.164326  0.164455
…          …         …
1068  0.132933  0.132944
1069  0.129294  0.129430
1070  0.172582  0.172511
1071  0.163779  0.163907
1072  0.156125  0.156074

[1073 rows x 293 columns]>
```

3.1.2 Inspect the input data and do primary data cleanning

Data inspection (optional)

[ ]:
```
# print(df_data.head(2))
```

[ ]:
```
# print(df_data.tail(2))
```

[ ]:
```
# df_data.describe()
```

[ ]:
```
# data_table.DataTable(df_data, include_index=False, num_rows_per_page=10)
```

[ ]:
```
# data_table.DataTable(df_data.filter(regex='nir_').describe(),
↪include_index=False, num_rows_per_page=10)
```

Primary data cleaning

```
[9]: # Count number of Rows having NaN or Empty element(s)
nrows_having_NaN = df_data.shape[0] - df_data.dropna().shape[0]
print ("Total number of Rows contain NaN or Empty element(s): {}".
 ↪format(nrows_having_NaN))
```

```
Total number of Rows contain NaN or Empty element(s): 21
```

```
[10]: nrows_index_having_NaN = df_data.index[df_data.isnull().any(axis=1)]
print ("Following Rows: {} having NaN or Empty element(s)".
 ↪format(nrows_index_having_NaN.tolist()))
df_data.iloc[nrows_index_having_NaN.tolist()]
```

```
Following Rows: [6, 7, 12, 148, 380, 410, 459, 548, 549, 605, 817, 827, 942,
943, 956, 1039, 1042, 1056, 1059, 1063, 1068] having NaN or Empty element(s)
```

```
[10]:       measurement_ID                  lat_lng location       som  nir_1370  \
6                    6  (41.2796451,-92.0272683)  field_A  2.404789  0.458376
7                    7  (41.2785352,-92.0258246)  field_A  2.274771  0.405309
12                  12  (41.2789834,-92.0271847)  field_A  2.368392  0.429245
148                148  (41.2749801,-92.0302816)  field_A       NaN  0.548440
380                380  (41.2744286,-92.0309851)  field_A  2.243380  0.262801
410                410  (41.2755399,-92.0302769)  field_A  2.087054  0.391038
459                459  (41.2767103,-92.0294227)  field_A  1.672700  0.366341
548                548   (41.276149,-92.0317264)  field_A  1.859473  0.474475
549                549  (41.2761434,-92.0309735)  field_A  1.928195  0.474540
605                605  (41.2749934,-92.0271311)  field_A  2.118079  0.465279
817                817   (41.2718129,-92.040581)  field_B  1.742657  0.408274
827                827  (41.2717725,-92.0391614)  field_B       NaN  0.483472
942                942   (41.271788,-92.0383153)  field_B  1.825330  0.498465
943                943  (41.2718039,-92.0383111)  field_B  1.743291  0.482633
956                956  (41.2732904,-92.0347316)  field_B  3.360884  0.438682
1039              1039  (41.2700941,-92.0374383)  field_B  2.081310  0.536902
1042              1042  (41.2702266,-92.0390213)  field_B  1.694808  0.466805
1056              1056   (41.271216,-92.0405878)  field_B  1.675076  0.449637
1059              1059  (41.2717577,-92.0405707)  field_B  1.531175  0.495917
1063              1063  (41.2723853,-92.0405988)  field_B  1.629065  0.463084
1068              1068  (41.2718012,-92.0390603)  field_B  1.919485  0.421228

      nir_1374  nir_1378  nir_1382  nir_1386  nir_1390  …  nir_2486  \
6     0.455395  0.450467  0.443263  0.433917  0.422832  …  0.203755
7     0.402589  0.397344  0.389162  0.378638  0.366896  …  0.161977
12    0.426400  0.421677  0.414607  0.405291  0.394182  …  0.180110
148   0.544507  0.539130  0.531397  0.520869  0.507597  …  0.233458
380   0.259948  0.256718  0.252900  0.248281  0.242464  …  0.149032
410   0.388326  0.384833  0.379096  0.370173  0.359023  …  0.159687
459   0.364446  0.360978       NaN  0.350085  0.343373  …  0.177333
548   0.470377  0.464071  0.455043  0.443431  0.429572  …  0.182306
```

549	0.471533	0.466922	0.459742	0.450156	0.438694	…	0.199251
605	0.461459	0.455300	0.446164	0.434476	NaN	…	0.162110
817	0.405335	0.401973	0.396414	0.386440	0.373136	…	0.150218
827	0.480591	0.475051	0.465203	0.450663	0.433068	…	0.144127
942	0.495466	0.491580	0.484143	0.470803	0.453667	…	0.172637
943	0.480355	0.476989	0.470280	0.458405	0.442820	…	0.172898
956	0.437289	0.433614	0.426335	0.415141	0.401353	…	0.163369
1039	0.531439	0.523232	0.510788	0.494440	0.475116	…	0.169982
1042	0.463413	0.457836	0.449059	0.437321	0.423249	…	0.157725
1056	0.445931	0.440272	0.431129	0.418662	0.403592	…	0.142147
1059	0.492432	0.486819	0.477747	0.465435	0.450604	…	0.180223
1063	0.459362	0.453421	0.443682	0.430359	0.414262	…	0.146415
1068	0.417337	0.411337	0.402050	0.389677	0.374866	…	0.140396

	nir_2490	nir_2494	nir_2498	nir_2502	nir_2506	nir_2510	nir_2514 \
6	0.201477	0.199492	0.197807	0.196425	0.195351	0.194587	0.194137
7	0.160195	0.158629	0.157287	0.156176	0.155300	0.154666	0.154279
12	0.178271	0.176629	0.175192	0.173965	0.172953	0.172160	0.171592
148	0.230943	0.228500	0.226186	0.224060	0.222180	0.220608	0.219403
380	0.147511	NaN	0.145008	0.144056	0.143315	0.142793	0.142499
410	0.157733	0.156086	0.154769	0.153770	0.153076	0.152673	0.152548
459	0.175782	0.174269	0.172827	0.171495	0.170306	0.169299	0.168508
548	0.180258	0.178296	0.176461	0.174797	0.173346	0.172151	0.171255
549	0.197084	0.194992	0.193020	0.191218	0.189635	0.188319	0.187320
605	0.160218	0.158387	0.156663	0.155089	0.153713	0.152578	0.151732
817	0.148557	0.147147	0.145996	0.145093	0.144432	0.144004	0.143799
827	0.142438	0.141026	0.139892	0.139024	0.138409	0.138035	0.137888
942	0.170746	0.169156	0.167871	0.166877	0.166160	0.165705	0.165499
943	0.170898	0.169219	0.167863	0.166814	0.166057	0.165576	0.165355
956	0.161599	0.160163	0.159065	0.158281	0.157787	0.157561	0.157578
1039	0.167988	0.166232	0.164726	0.163478	0.162498	0.161795	0.161379
1042	0.155831	0.154173	0.152763	0.151613	0.150734	0.150140	0.149841
1056	0.140462	0.138977	0.137704	0.136652	0.135831	0.135251	0.134923
1059	0.178234	0.176452	0.174895	0.173579	0.172521	0.171737	0.171245
1063	0.144746	0.143276	0.142019	0.140988	0.140197	0.139661	0.139392
1068	0.138827	0.137418	0.136181	0.135125	0.134262	0.133602	0.133156

	nir_2518	nir_2522
6	0.194004	0.194193
7	0.154144	0.154269
12	0.171254	0.171149
148	0.218624	0.218331
380	0.142443	0.142631
410	0.152688	0.153080
459	0.167970	0.167716
548	0.170701	0.170528
549	0.186686	0.186459

```
605   0.151219  0.151080
817   0.143810  0.144028
827   0.137955  0.138225
942   0.165528  0.165778
943   0.165380  0.165633
956   0.157815  0.158248
1039  0.161258  0.161439
1042  0.149850  0.150175
1056  0.134856  0.135059
1059  0.171060  0.171197
1063  0.139404  0.139710
1068  0.132933  0.132944

[21 rows x 293 columns]
```

[ ]:
```
# # Select duplicate rows except first occurrence based on all columns
# duplicateRowsDF = df_data[df_data.duplicated()]
# print("Duplicate Rows except first occurrence based on all columns are :")
# print(duplicateRowsDF)

# for col in ['measurement_ID', 'lat_lng']:
#   # Select all duplicate rows based on one column
#   duplicateRowsDF = df_data[df_data.duplicated([col])]
#   print("Duplicate Rows based on a single column are:", duplicateRowsDF,␣
↪sep='\n')
```

[ ]:
```
# # Count number of Rows having  all elements are NaN or Empty
# nrows_all_NaN = df_data.shape[0] - df_data.isnull().all(axis=1).shape[0]
# print ("Total number of Rows having  all elements are NaN or Empty: {}".
↪format(nrows_all_NaN))

# if nrows_all_NaN> 0:
#     print (df_data.loc[nrows_all_NaN])
#     df_data = df_data.loc[df_data.index.drop(nrows_all_NaN)]
```

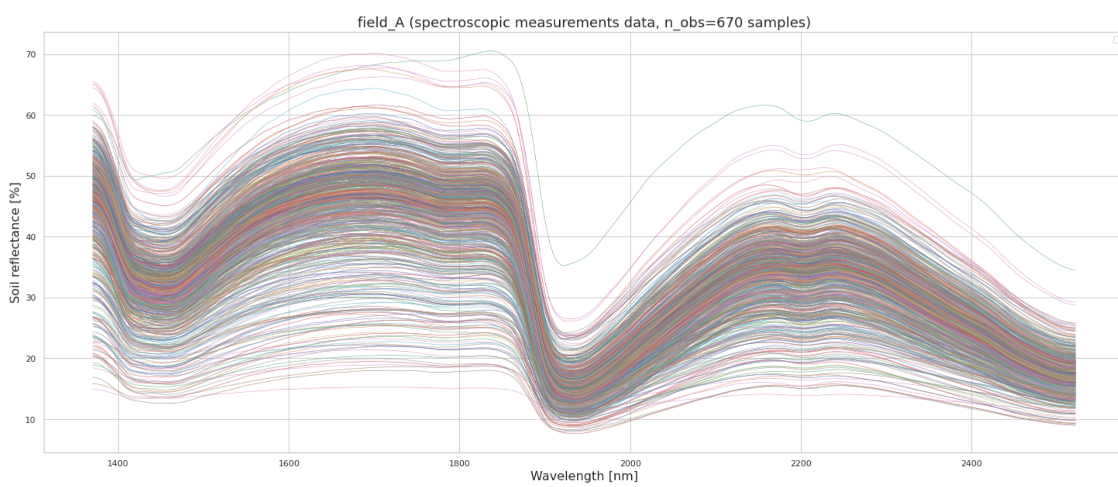
[11]:
```
# Drop Rows with missing values or NaN
df_data = df_data.dropna()

# Extract wave length
wave_length = [int(item.split('_')[1]) for item in df_data.columns.tolist() if␣
↪'nir_' in item]
# print (wave_length)

# Columns of Interest
ioc_name = [col for col in df_data.columns if (("nir_" in col) and␣
↪(df_data[col].dtype == "float64"))]
# print (ioc_name)
```

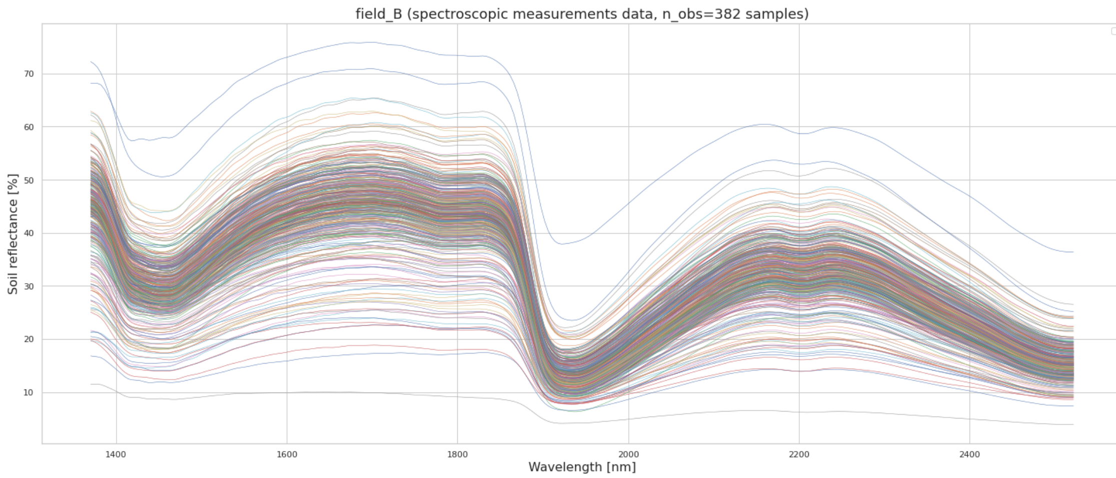
Further data inspection via chartplot(s)

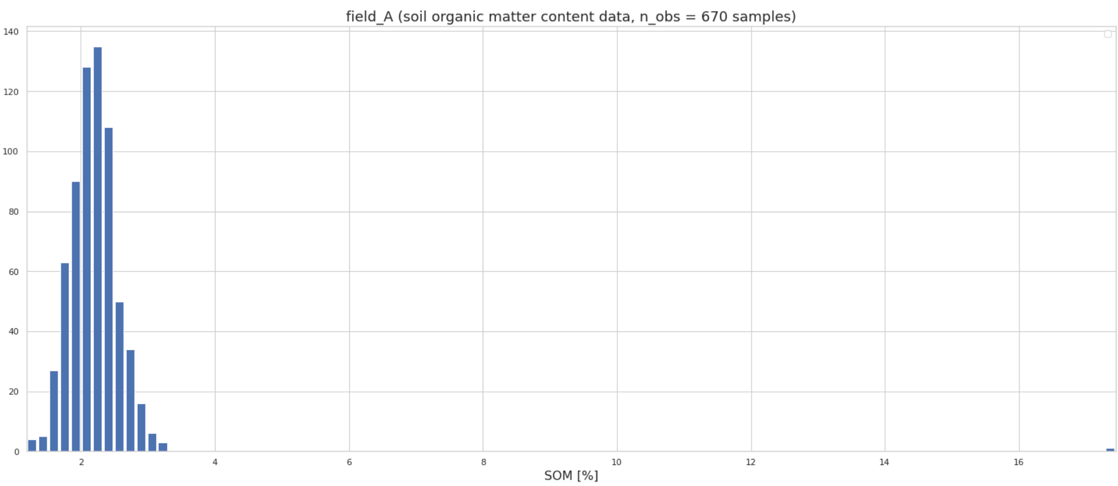
```
[12]: def density_scatter(df, field_name, xcol='nir_2522', ycol='som', ax = None,␣
     ↪sort=True, bins=20, **kwargs )   :
         """
         Scatter plot colored by 2d histogram
         """

         x = df_field[xcol].to_numpy()
         y = df_field[ycol].to_numpy()

         if ax is None :
             fig , ax = plt.subplots(figsize=(10, 8))
         data , x_e, y_e = np.histogram2d( x, y, bins = bins, density = True )
         z = interpn( ( 0.5*(x_e[1:] + x_e[:-1]) , 0.5*(y_e[1:]+y_e[:-1]) ) , data ,␣
     ↪np.vstack([x,y]).T , method = "splinef2d", bounds_error = False)

         #To be sure to plot all data
         z[np.where(np.isnan(z))] = 0.0

         # Sort the points by density, so that the densest points are plotted last
         if sort :
             idx = z.argsort()
             x, y, z = x[idx], y[idx], z[idx]

         ax.scatter( x, y, c=z, **kwargs )
         plt.xlabel(xcol, fontsize=16)
         plt.ylabel(ycol, fontsize=16)

         norm = Normalize(vmin = np.min(z), vmax = np.max(z))
         cbar = fig.colorbar(cm.ScalarMappable(norm = norm), ax=ax)
         cbar.ax.set_ylabel('Density', fontsize=18)
         ax.set_title ("{} (n_obs = {})".format(field_name, df_field[xcol].
     ↪shape[0]), fontsize=18)
         plt.show()

         return ax


     for location_name in set(df_data['location'].tolist()):
       df_field = df_data[df_data['location'] == location_name]
       df_field_nir = df_field.filter(regex='nir_')
       plt.figure(figsize=(25, 10))
       plt.clf

       for _, row in df_field_nir.iterrows():
           plt.plot(wave_length, 100*row, linewidth=0.5)
```

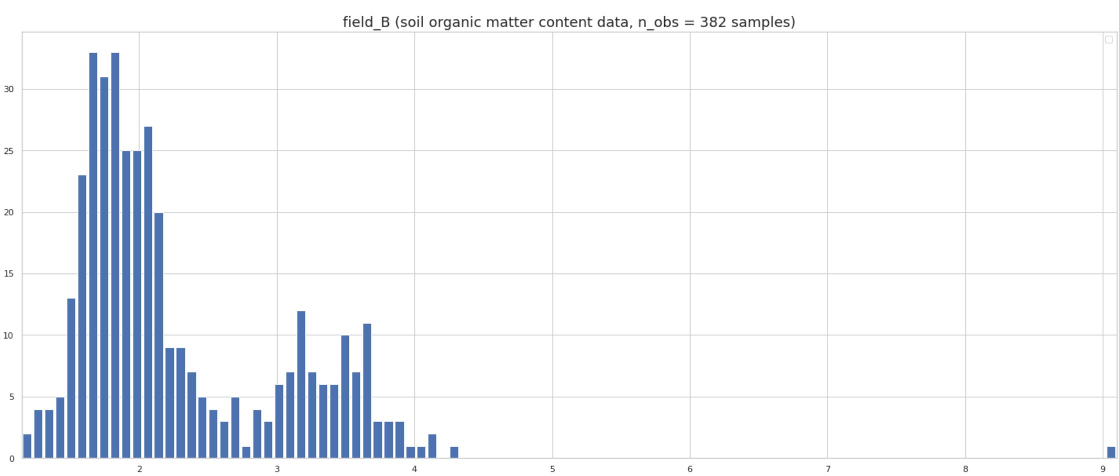
```
  plt.xlabel("Wavelength [nm]", fontsize=16)
  plt.ylabel("Soil reflectance [%]", fontsize=16)
  plt.title ("{} (spectroscopic measurements data, n_obs={} samples)".
 ↪format(location_name, df_field.shape[0]), fontsize=18)
  plt.legend()

for location_name in set(df_data['location'].tolist()):
  df_field = df_data[df_data['location'] == location_name]
  plt.figure(figsize=(25, 10))
  plt.clf

  plt.hist(df_field['som'], bins=100, histtype='bar', rwidth=0.8)
  plt.xlim(df_field['som'].min(), df_field['som'].max())
  plt.xlabel("SOM [%]", fontsize=16)
  plt.title ("{} (soil organic matter content data, n_obs = {} samples)".
 ↪format(location_name, df_field['som'].shape[0]), fontsize=18)
  plt.legend()

for location_name in set(df_data['location'].tolist()):
  df_field = df_data[df_data['location'] == location_name]
  density_scatter(df_field, location_name, xcol='nir_1902', ycol='som',␣
 ↪bins=[100,100])
```

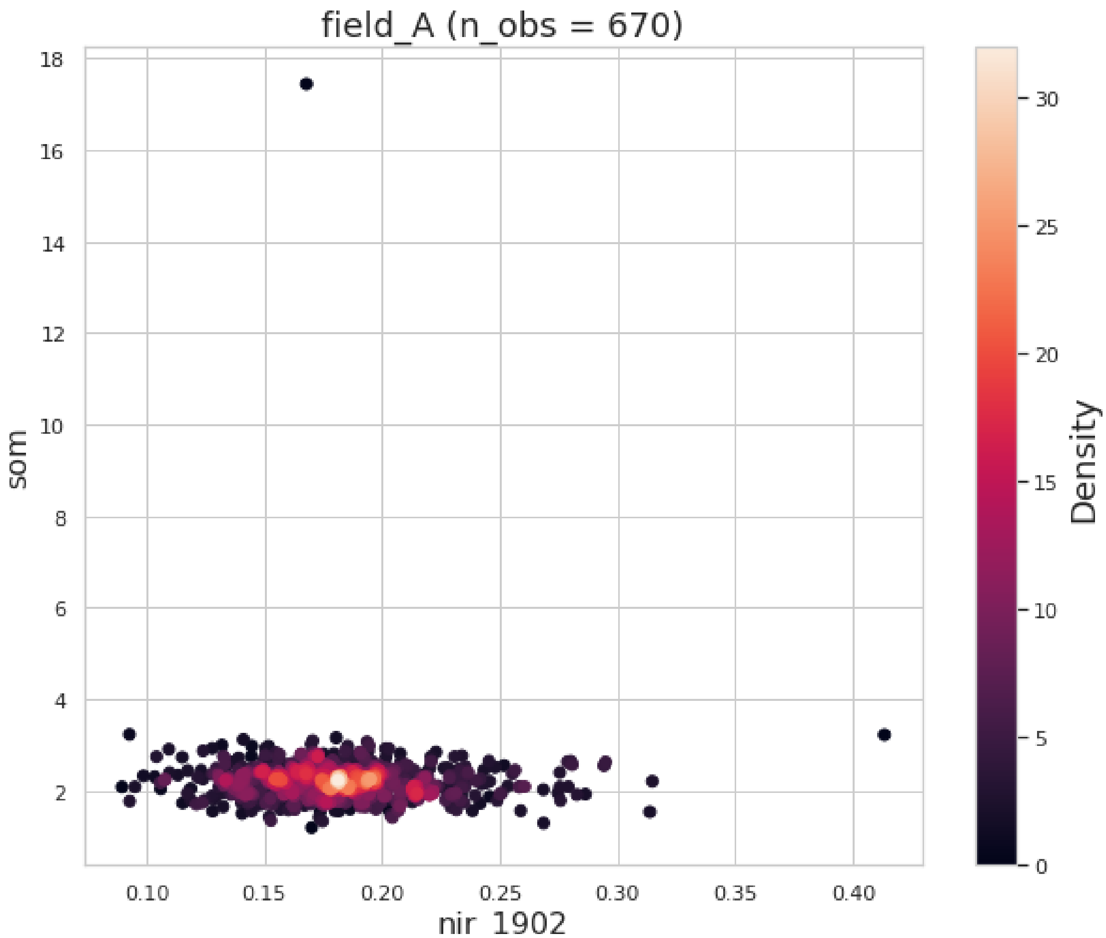
```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
```

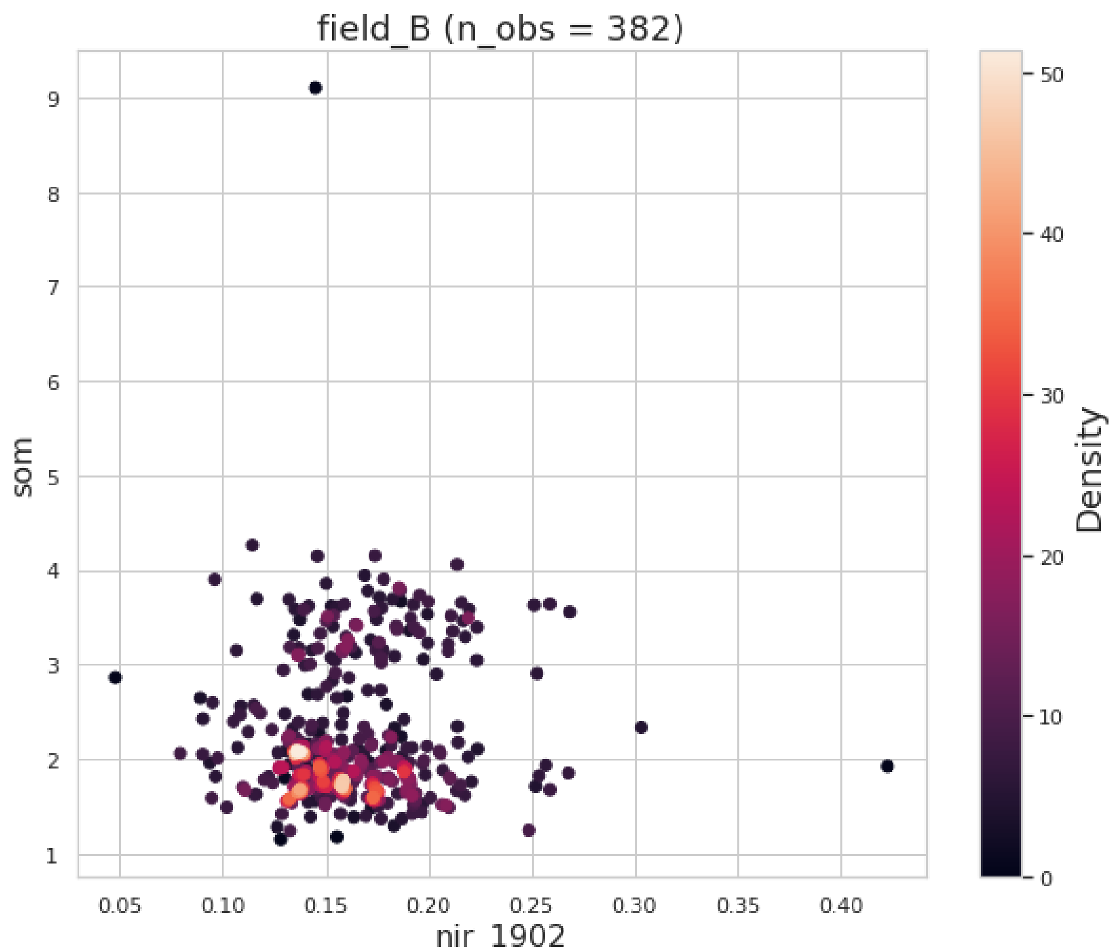



field_B (spectroscopic measurements data, n_obs=382 samples)
Soil reflectance [%]
Wavelength [nm]
10
20
30
40
50
60
70
1400
1600
1800
2000
2200
2400


field_A (soil organic matter content data, n_obs = 670 samples)
SOM [%]
0
20
40
60
80
100
120
140
2
4
6
8
10
12
14
16


field_B (soil organic matter content data, n_obs = 382 samples)
SOM [%]
0
5
10
15
20
25
30
2
3
4
5
6
7
8
9


field_A (n_obs = 670)
som
nir_1902
Density
18
16
14
12
10
8
6
4
2
0.10
0.15
0.20
0.25
0.30
0.35
0.40
30
25
20
15
10
5
0



[ ]:
```
# g = sns.FacetGrid(df_data, col="location", margin_titles=True, height=4)
# g.map(plt.scatter, "nir_2522", "som", color="#338844", edgecolor="white",␣
↪s=50, lw=1)
```

[ ]:
```
# sns.lmplot('nir_1370', 'nir_2522', data=df_data, fit_reg=True)
```

Geo-location data inspection

[13]:
```
# Define a method for displaying Earth Engine image tiles to folium map.
def add_ee_layer(self, ee_image_object, vis_params, name):
  map_id_dict = ee.Image(ee_image_object).getMapId(vis_params)
  folium.raster_layers.TileLayer(
    tiles = map_id_dict['tile_fetcher'].url_format,
    attr = 'Map Data &copy; <a href="https://earthengine.google.com/">Google␣
↪Earth Engine</a>',
    name = name,
    overlay = True,
```

```python
    control = True
  ).add_to(self)


# Add EE drawing method to folium.
folium.Map.add_ee_layer = add_ee_layer

# Conver pandas dataframe into geojson format
def pandas_to_geojson(df, out_geojson=None, latitude_longitude="lat_lng",␣
 ↪encoding="utf-8",):
  """Creates points for a Pandas DataFrame and exports data as a GeoJSON.

  Args:
      df (pandas.DataFrame): The input Pandas DataFrame.
      out_geojson (str): The file path to the exported GeoJSON. Default to None.
      latitude (str, optional): The name of the column containing latitude␣
 ↪coordinates. Defaults to "latitude".
      longitude (str, optional): The name of the column containing longitude␣
 ↪coordinates. Defaults to "longitude".
      encoding (str, optional): The encoding of characters. Defaults to "utf-8".

  """

  if out_geojson is not None:
    out_dir = os.path.dirname(os.path.abspath(out_geojson))
    if not os.path.exists(out_dir):
      os.makedirs(out_dir)

  features = df.apply(
    lambda row: Feature(
        geometry=Point((float(row['lat_lng'][1:-1].split(',')[1]),␣
 ↪float(row['lat_lng'][1:-1].split(',')[0]))),
        properties=dict(row),
    ),
    axis=1,
  ).tolist()

  geojson_obj = FeatureCollection(features=features)
  coords = np.array(list(geojson.utils.coords(geojson_obj)))

  #delete [0, 0] from coords
  # coords = np.delete(coords, [0, 0], axis=0)
  area_extent = coords[:,0].min(), coords[:,0].max(), coords[:,1].min(),␣
 ↪coords[:,1].max()

  if out_geojson is None:
    return [False, geojson_obj, area_extent]
```

```
    else:
      with open(out_geojson, "w", encoding=encoding) as f:
        f.write(json.dumps(geojson_obj))
      return [True, geojson_obj, area_extent]
```

[150]:
```
# df_data = df_data[df_data['lat_lng'] != '(0,0)']
_, geojson_obj, area_extent = pandas_to_geojson(df_data.iloc[:, :4])
fc = geemap.geojson_to_ee(geojson_obj)

Map = geemap.Map(center=(0, 0), zoom=3, lite_mode=True)
Map.addLayer(fc, {}, 'csv to ee 2')
Map
```

```
Map(center=[0, 0], controls=(ZoomControl(options=['position', 'zoom_in_text',␣
↪'zoom_in_title', 'zoom_out_text'…
```

[15]:
```
df_data = df_data[df_data['lat_lng'] != '(0,0)']
_, geojson_obj, area_extent = pandas_to_geojson(df_data.iloc[:, :4])
fc = geemap.geojson_to_ee(geojson_obj)

Map = geemap.Map(center=(0, 0), zoom=3, lite_mode=True)
Map.addLayer(fc, {}, 'csv to ee 2')
Map
```

```
Map(center=[0, 0], controls=(ZoomControl(options=['position', 'zoom_in_text',␣
↪'zoom_in_title', 'zoom_out_text'…
```

Handle wrong geo-location data

[16]:
```
_temp_geo = df_data["lat_lng"].str.slice(start=1,stop=-1).str.split(",", n = 1,␣
↪expand = True)
df_data["lat"] = _temp_geo[0]
df_data['lng'] = _temp_geo[1]
df_data[["lat", "lng"]] = df_data[["lat", "lng"]].apply(pd.to_numeric)
lat_median = df_data["lat"].median()
lon_median = df_data["lng"].median()
```

[17]:
```
_, geojson_obj, area_extent = pandas_to_geojson(df_data.iloc[:, :4])
fc = geemap.geojson_to_ee(geojson_obj)

Map = geemap.Map(center=(lat_median, lon_median), zoom=15, lite_mode=True)
# https://viewer.nationalmap.gov/services/
url = 'https://mt1.google.com/vt/lyrs=y&x={x}&y={y}&z={z}'
Map.add_tile_layer(url, name='Google Satellite', attribution='Google')
Map.addLayer(fc, {}, 'csv to ee 2')
Map
```

```
Map(center=[41.2738562, -92.0308867], controls=(ZoomControl(options=['position',␣
↪'zoom_in_text', 'zoom_in_titl…
```

[18]:
```python
_, geojson_obj, area_extent = pandas_to_geojson(df_data.iloc[:, :
↪4][df_data['location'] == 'field_A'])
fc = geemap.geojson_to_ee(geojson_obj)

Map = geemap.Map(center=(lat_median, lon_median), zoom=15, lite_mode=True)
# https://viewer.nationalmap.gov/services/
url = 'https://mt1.google.com/vt/lyrs=y&x={x}&y={y}&z={z}'
Map.add_tile_layer(url, name='Google Satellite', attribution='Google')
Map.addLayer(fc, {}, 'csv to ee 2')
Map
```

```
Map(center=[41.2738562, -92.0308867], controls=(ZoomControl(options=['position',␣
↪'zoom_in_text', 'zoom_in_titl…
```

[19]:
```python
_, geojson_obj, area_extent = pandas_to_geojson(df_data.iloc[:, :
↪4][df_data['location']== 'field_B'])
fc = geemap.geojson_to_ee(geojson_obj)

Map = geemap.Map(center=(lat_median, lon_median), zoom=15, lite_mode=True)
# https://viewer.nationalmap.gov/services/
url = 'https://mt1.google.com/vt/lyrs=y&x={x}&y={y}&z={z}'
Map.add_tile_layer(url, name='Google Satellite', attribution='Google')
Map.addLayer(fc, {}, 'csv to ee 2')
Map
```

```
Map(center=[41.2738562, -92.0308867], controls=(ZoomControl(options=['position',␣
↪'zoom_in_text', 'zoom_in_titl…
```

[ ]:
```python
# # states = ee.FeatureCollection("TIGER/2018/States")
# # style = {'color': 'black', 'fillColor': "00000000"}
# # Map.addLayer(states.style(**style), {}, "US States")

# Map.remove_labels()

# Map = geemap.Map(center=(lat_median, lon_median), zoom=16, lite_mode=True,␣
↪add_google_map=True)
# # https://viewer.nationalmap.gov/services/
# url = 'https://mt1.google.com/vt/lyrs=y&x={x}&y={y}&z={z}'
# Map.add_tile_layer(url, name='Google Satellite', attribution='Google')
# # Labeling a Pandas DataFrame
# Map.add_labels(
#     df_data[['location', 'lat', 'lng', 'som']],
#     "location",
#     x='lng',
```

```
#     y='lat',
#     font_size="16pt",
#     font_color="red",
#     font_family="arial",
#     # font_weight="bold",
# )
# Map
```

[20]:
```
_,geojson_obj, extent = pandas_to_geojson(df_data, './data.geojson')
if geojson_obj is True:
  print ('You can start download geojson file [](./data.geojson] for further␣
↪local data inspection and correction')
else:
  print ('You would like to go further with data inspection in Colab!')
```

```
You would like to go further with data inspection in Colab!
```

[21]:
```
uploaded = files.upload()
fields_boundary_fn = next(iter(uploaded))
```

```
<IPython.core.display.HTML object>

Saving fields_boundary.geojson to fields_boundary.geojson
```

[22]:
```
poly_gdf = gpd.read_file('./{}'.format(fields_boundary_fn))
point_gdf = gpd.GeoDataFrame.from_features(geojson_obj['features'], crs=4326)
# print(point_gdf)

# Make sure they're using the same projection reference before perform geometry␣
↪Joins
if point_gdf.crs == poly_gdf.crs:
  join_inner_gdf = point_gdf.sjoin(poly_gdf, how="inner")
else:
  print ('The projection reference not the same!')

join_inner_gdf.rename({'location_right': 'location', 'location_left':␣
↪'location_orig'}, axis=1, inplace=True)
df_data_correct_geo = pd.DataFrame(join_inner_gdf.drop(columns='geometry'))
```

[23]:
```
_, geojson_obj, area_extent = pandas_to_geojson(df_data_correct_geo.iloc[:, :
↪4][df_data_correct_geo['location']== 'field_B'])
fc = geemap.geojson_to_ee(geojson_obj)

Map = geemap.Map(center=(lat_median, lon_median), zoom=15, lite_mode=True)
# https://viewer.nationalmap.gov/services/
url = 'https://mt1.google.com/vt/lyrs=y&x={x}&y={y}&z={z}'
Map.add_tile_layer(url, name='Google Satellite', attribution='Google')
```

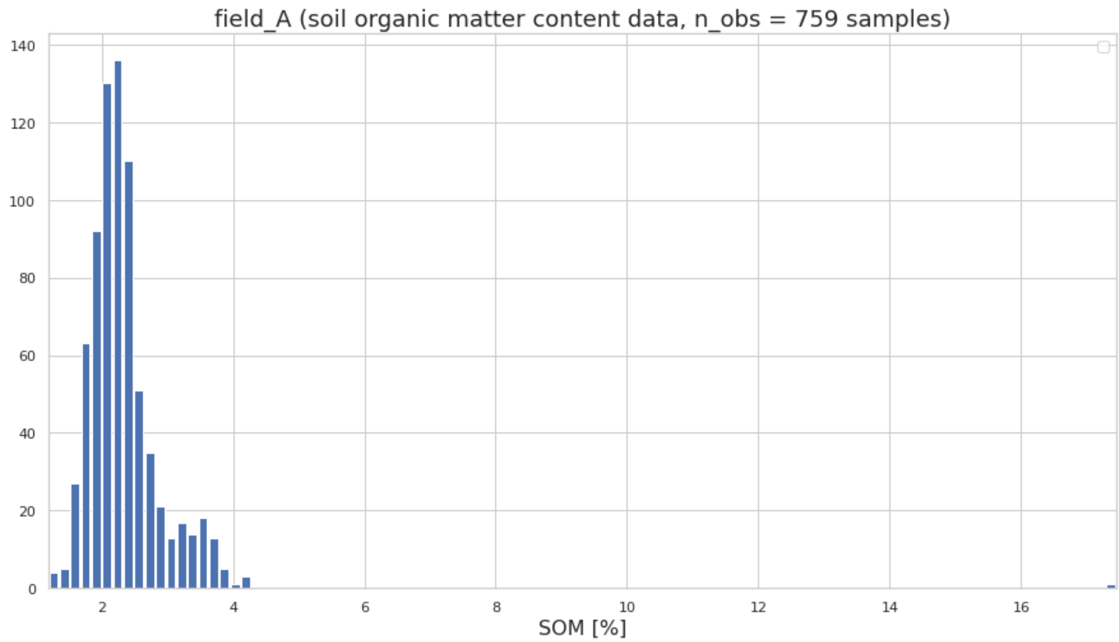
```
Map.addLayer(fc, {}, 'csv to ee 2')
Map
```

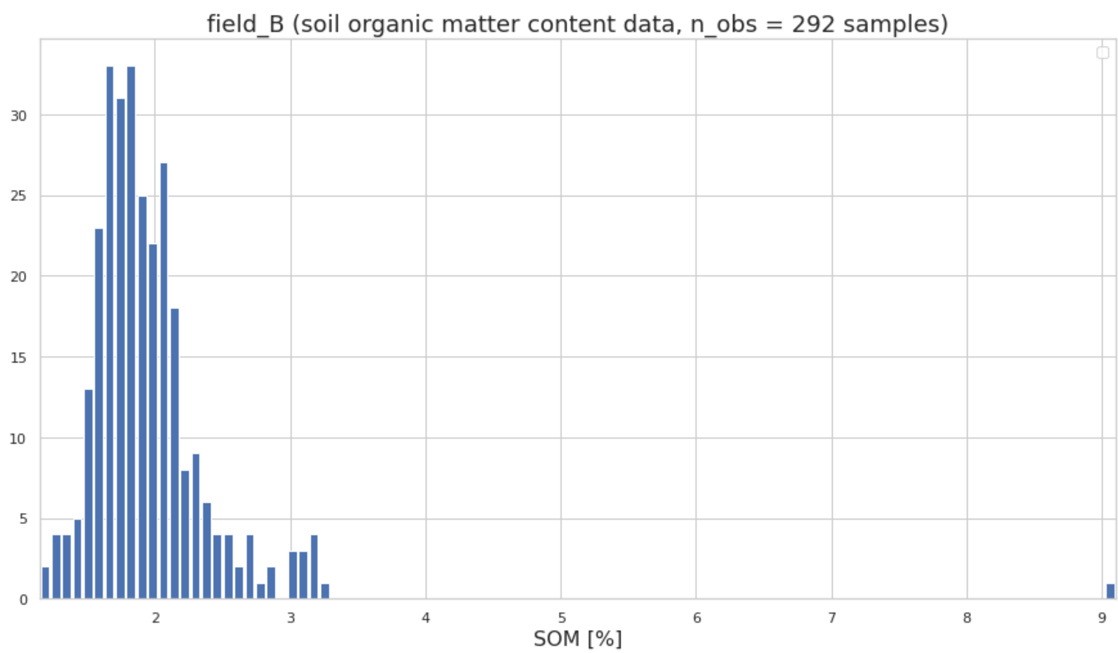
```
Map(center=[41.2738562, -92.0308867], controls=(ZoomControl(options=['position',␣
↪'zoom_in_text', 'zoom_in_titl…
```

[ ]:
```
# # states = ee.FeatureCollection("TIGER/2018/States")
# # style = {'color': 'black', 'fillColor': "00000000"}
# # Map.addLayer(states.style(**style), {}, "US States")

# Map.remove_labels()

# Map = geemap.Map(center=(lat_median, lon_median), zoom=16, lite_mode=True,␣
↪add_google_map=True)
# url = 'https://mt1.google.com/vt/lyrs=y&x={x}&y={y}&z={z}'
# Map.add_tile_layer(url, name='Google Satellite', attribution='Google')
# # Labeling a Pandas DataFrame
# Map.add_labels(
#     df_data_correct_geo[['location', 'lat', 'lng', 'som']],
#     "location",
#     x='lng',
#     y='lat',
#     font_size="16pt",
#     font_color="red",
#     font_family="arial",
#     # font_weight="bold",
# )
# Map
```

[24]:
```
for location_name in set(df_data_correct_geo['location'].tolist()):
  df_field = df_data_correct_geo[df_data_correct_geo['location'] ==␣
↪location_name]
  plt.figure(figsize=(15, 8))
  plt.clf

  plt.hist(df_field['som'], bins=100, histtype='bar', rwidth=0.8)
  plt.xlim(df_field['som'].min(), df_field['som'].max())
  plt.xlabel("SOM [%]", fontsize=16)
  plt.title ("{} (soil organic matter content data, n_obs = {} samples)".
↪format(location_name, df_field['som'].shape[0]), fontsize=18)
  plt.legend()
```

```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
```


field_A (soil organic matter content data, n_obs = 759 samples)
0
20
40
60
80
100
120
140
2
4
6
8
10
12
14
16
SOM [%]


field_B (soil organic matter content data, n_obs = 292 samples)
0
5
10
15
20
25
30
2
3
4
5
6
7
8
9
SOM [%]

4 Compare with POLARIS soil database

4.1 Identity POLARIS tile(s) data

[25]:
```
# Default POLARIS setting values
DEFAULT_POLARIS_URL = 'http://hydrology.cee.duke.edu/POLARIS/PROPERTIES/v1.0/
↪{0}/{1}/{2}/lat{3}{4}_lon{5}{6}.tif'

DEFAULT_POLARIS_LAYERS = ['0_5', '5_15', '15_30', '30_60', '60_100', '100_200']
DEFAULT_POLARIS_STATISTICS = ['mean', 'mode', 'p5', 'p50', 'p95']
DEFAULT_POLARIS_PARAMETERS = ['alpha', 'bd', 'clay', 'hb', 'ksat', 'lambda',␣
↪'n', 'om', 'ph', 'sand', 'silt', 'theta_r', 'theta_s']

DEFAULT_POLARIS_WORKING_DIRECTORY = './polaris'

layers = DEFAULT_POLARIS_LAYERS
statistics = DEFAULT_POLARIS_STATISTICS
parameters = DEFAULT_POLARIS_PARAMETERS
template_url = DEFAULT_POLARIS_URL

output_base_path = DEFAULT_POLARIS_WORKING_DIRECTORY
os.makedirs(output_base_path, exist_ok=True)

def download_polaris_data(area_extent, output_base_path=output_base_path,␣
↪layers=layers, statistics=statistics, parameters=parameters):

    # import pdb;pdb.set_trace()
    (minLon, maxLon, minLat, maxLat) = area_extent
    domain_extent = {'lon': [math.floor(minLon), int(math.ceil(maxLon))], 'lat':
↪[math.floor(minLat), int(math.ceil(maxLat))]}

    def generate_url_path(output_base_path, domain_extent):
        ''' TODO': Write discription'''

        url_path_lst = []
        lat_range = range(domain_extent['lat'][0],domain_extent['lat'][1])
        lon_range = range(domain_extent['lon'][0],domain_extent['lon'][1])
        for layer in layers:
            for stat in statistics:
                for param in parameters:
                    for lat in lat_range:
                        for lon in lon_range:
                            url = template_url.
↪format(param,stat,layer,str(lat),str(lat+1),str(lon),str(lon+1))
                            temp_path = os.path.join(output_base_path, '{}/{}/
↪{}/'.format(param,stat,layer))
                            if not os.path.exists(temp_path):
```

```
                        os.makedirs(temp_path)
                    url_path_lst += [[url, temp_path]]

        return url_path_lst


    # def run_process(url, output_path):
    #     wget.download(url, out=output_path)


    # Generate URL path for the data that we intersted in
    url_path_lst = generate_url_path(output_base_path, domain_extent)

    # cpus = multiprocessing.cpu_count()
    # max_pool_size = 6
    # pool = multiprocessing.Pool(cpus if cpus < max_pool_size else␣
↪max_pool_size)

    for url, path in url_path_lst:
        print('Beginning file download with wget module {n}'.format(n=url))
        wget.download(url, out=path)
        # pool.apply_async(run_process, args=(url, path, ))

    # pool.close()
    # pool.join()

    for url, path in url_path_lst:
      if os.path.exists(path):
        print("File(s) in {} downloaded successfully from {}".format(path, url))
      else:
        print("Failed when download file {} from {}".format(path, url))

    return url_path_lst
```

4.2 Download selected POLARIS tile(s) data

[26]:
```
# Default CHALLENGE setting values
DEFAULT_CHALLENGE_LAYERS = ['0_5', '5_15', '15_30']
DEFAULT_CHALLENGE_STATISTICS = ['mean', 'mode', 'p5', 'p50', 'p95']
DEFAULT_CHALLENGE_PARAMETERS = ['om']

_, geojson_obj, area_extent = pandas_to_geojson(df_data_correct_geo, './data.
↪geojson')
url_path_lst = download_polaris_data(area_extent,␣
↪layers=DEFAULT_CHALLENGE_LAYERS, statistics=DEFAULT_CHALLENGE_STATISTICS,␣
↪parameters=DEFAULT_CHALLENGE_PARAMETERS)
```

```
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/mean/0_5/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/mode/0_5/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/p5/0_5/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/p50/0_5/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/p95/0_5/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/mean/5_15/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/mode/5_15/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/p5/5_15/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/p50/5_15/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/p95/5_15/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/mean/15_30/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/mode/15_30/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/p5/15_30/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/p50/15_30/lat4142_lon-93-92.tif
Beginning file download with wget module http://hydrology.cee.duke.edu/POLARIS/P
ROPERTIES/v1.0/om/p95/15_30/lat4142_lon-93-92.tif
File(s) in ./polaris/om/mean/0_5/ downloaded successfully from http://hydrology.
cee.duke.edu/POLARIS/PROPERTIES/v1.0/om/mean/0_5/lat4142_lon-93-92.tif
File(s) in ./polaris/om/mode/0_5/ downloaded successfully from http://hydrology.
cee.duke.edu/POLARIS/PROPERTIES/v1.0/om/mode/0_5/lat4142_lon-93-92.tif
File(s) in ./polaris/om/p5/0_5/ downloaded successfully from http://hydrology.ce
e.duke.edu/POLARIS/PROPERTIES/v1.0/om/p5/0_5/lat4142_lon-93-92.tif
File(s) in ./polaris/om/p50/0_5/ downloaded successfully from http://hydrology.c
ee.duke.edu/POLARIS/PROPERTIES/v1.0/om/p50/0_5/lat4142_lon-93-92.tif
File(s) in ./polaris/om/p95/0_5/ downloaded successfully from http://hydrology.c
ee.duke.edu/POLARIS/PROPERTIES/v1.0/om/p95/0_5/lat4142_lon-93-92.tif
File(s) in ./polaris/om/mean/5_15/ downloaded successfully from http://hydrology
.cee.duke.edu/POLARIS/PROPERTIES/v1.0/om/mean/5_15/lat4142_lon-93-92.tif
File(s) in ./polaris/om/mode/5_15/ downloaded successfully from http://hydrology
.cee.duke.edu/POLARIS/PROPERTIES/v1.0/om/mode/5_15/lat4142_lon-93-92.tif
File(s) in ./polaris/om/p5/5_15/ downloaded successfully from http://hydrology.c
ee.duke.edu/POLARIS/PROPERTIES/v1.0/om/p5/5_15/lat4142_lon-93-92.tif
File(s) in ./polaris/om/p50/5_15/ downloaded successfully from http://hydrology.
cee.duke.edu/POLARIS/PROPERTIES/v1.0/om/p50/5_15/lat4142_lon-93-92.tif
```

```
File(s) in ./polaris/om/p95/5_15/ downloaded successfully from http://hydrology.
cee.duke.edu/POLARIS/PROPERTIES/v1.0/om/p95/5_15/lat4142_lon-93-92.tif
File(s) in ./polaris/om/mean/15_30/ downloaded successfully from http://hydrolog
y.cee.duke.edu/POLARIS/PROPERTIES/v1.0/om/mean/15_30/lat4142_lon-93-92.tif
File(s) in ./polaris/om/mode/15_30/ downloaded successfully from http://hydrolog
y.cee.duke.edu/POLARIS/PROPERTIES/v1.0/om/mode/15_30/lat4142_lon-93-92.tif
File(s) in ./polaris/om/p5/15_30/ downloaded successfully from http://hydrology.
cee.duke.edu/POLARIS/PROPERTIES/v1.0/om/p5/15_30/lat4142_lon-93-92.tif
File(s) in ./polaris/om/p50/15_30/ downloaded successfully from http://hydrology
.cee.duke.edu/POLARIS/PROPERTIES/v1.0/om/p50/15_30/lat4142_lon-93-92.tif
File(s) in ./polaris/om/p95/15_30/ downloaded successfully from http://hydrology
.cee.duke.edu/POLARIS/PROPERTIES/v1.0/om/p95/15_30/lat4142_lon-93-92.tif
```

4.3 Extract POLARIS data

[83]:
```
def ZonalStats(in_vec, in_rst_list, out_vec=None, interest_cols=[],␣
↪attribute_name='som_polaris'):
    """TODO"""
    # in_vec - shapefile path
    # in_rst_list - raster path
    # the result is df as DataFrame

    shape_gdf = gpd.read_file(in_vec)
    for in_rst in in_rst_list:
        file_pattern = '_'.join(in_rst.split('/')[-3:-1])
        column_name = "{}_{}".format(attribute_name, file_pattern)
        zonalSt = point_query(in_vec, in_rst, band=1, nodata=-9999,␣
↪interpolate='nearest')
        df = pd.DataFrame (zonalSt, index=shape_gdf.index, columns =␣
↪[column_name])
        shape_gdf = pd.concat([shape_gdf, df], axis=1)
        interest_cols.append(column_name)

    interest_cols.append('geometry')
    # re-order the columns
    gdf = gpd.GeoDataFrame(shape_gdf, geometry=shape_gdf.geometry)
    gdf = gdf[interest_cols]

    if out_vec is not None:
        # Alternatively, you can write GeoJSON to file:
        gdf.to_file(out_vec, driver="GeoJSON")

    df = gdf.drop(['geometry'], axis=1, errors='ignore')

    return df
```

```
# Extract data from POLARIS soil database")
geojson_fn = './data.geojson'
in_rst_list =['{}{}'.format(item[1], os.path.basename(item[0])) for item in␣
↪url_path_lst]
selected_cols=['measurement_ID', 'location', 'som']
out_geojson_with_polaris = './data_with_SOM_from_polaris.geojson'

df_data_with_polaris_data = ZonalStats(geojson_fn, in_rst_list,␣
↪out_vec=out_geojson_with_polaris, interest_cols=selected_cols)
# df_data_with_polaris_data
```

4.4 Rescale Stenon SOM data

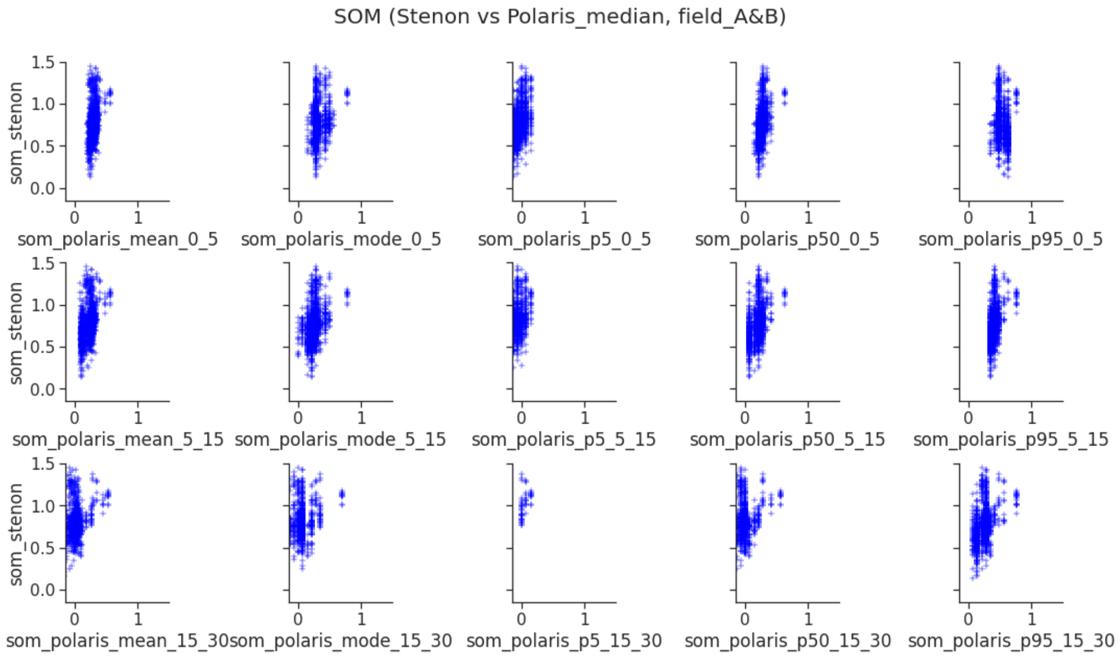
Table 1. Soil Properties That Are Mapped Over the Contiguous United States at a 30-m Spatial Resolution

[84]:
```
# Let's define a UDF(User defined function).
def compute_logData(x):
   return np.log(x)

df_data_with_polaris_data['som_stenon'] = df_data_with_polaris_data[['som']].
↪apply(compute_logData, axis=1)
```

4.5 Data comparison

[85]:
```
def plot_stenon_polaris(df,
                        field_name,
                        y_name = 'som_stenon',
                        features_names = ['som_polaris_mean_0_5',␣
↪'som_polaris_mode_0_5', 'som_polaris_p5_0_5', 'som_polaris_p50_0_5',␣
↪'som_polaris_p95_0_5',
                                          'som_polaris_mean_5_15',␣
↪'som_polaris_mode_5_15', 'som_polaris_p5_5_15', 'som_polaris_p50_5_15',␣
↪'som_polaris_p95_5_15',
                                          'som_polaris_mean_15_30',␣
↪'som_polaris_mode_15_30', 'som_polaris_p5_15_30', 'som_polaris_p50_15_30',␣
↪'som_polaris_p95_15_30']
,
                        x_lim_max=None,
                        y_lim_max=None
                        ):

  g = sns.FacetGrid(pd.DataFrame(features_names), col=0, col_wrap=5,␣
↪sharex=False)
```

```
  for ax, x_var in zip(g.axes, features_names):
      sns.scatterplot(data=df, x=x_var, y=y_name, ax=ax, marker="+",␣
↪color='blue')
  g.tight_layout()

  g.tight_layout()
  if x_lim_max is not None:
    g.set(xlim=(-0.15, x_lim_max))
  if y_lim_max is not None:
    g.set(ylim=(-0.15, y_lim_max))

  #move overall title up
  g.fig.subplots_adjust(top=0.9)
  g.fig.suptitle('SOM (Stenon vs Polaris_median, {})'.format(field_name),␣
↪fontsize=20)

  plt.savefig('Compare_with_POLARIS_SOM_{}.png'.format(field_name))
  plt.show()


features_names = df_data_with_polaris_data.columns[3:-1].tolist()

# # Plotting: all layers, 0_5, 5_15, 15_30
# df_field_A =␣
↪df_data_with_polaris_data[df_data_with_polaris_data['location']== 'field_A']
# df_field_B =␣
↪df_data_with_polaris_data[df_data_with_polaris_data['location']== 'field_B']
# plot_stenon_polaris(df_field_A, features_names=features_names, field_name =␣
↪'field_A', x_lim_max=1.5, y_lim_max=1.5)
# plot_stenon_polaris(df_field_B, features_names=features_names, field_name =␣
↪'field_B', x_lim_max=1.5, y_lim_max=1.5)

plot_stenon_polaris(df_data_with_polaris_data, features_names=features_names,␣
↪field_name = 'field_A&B', x_lim_max=1.5, y_lim_max=1.5)
```

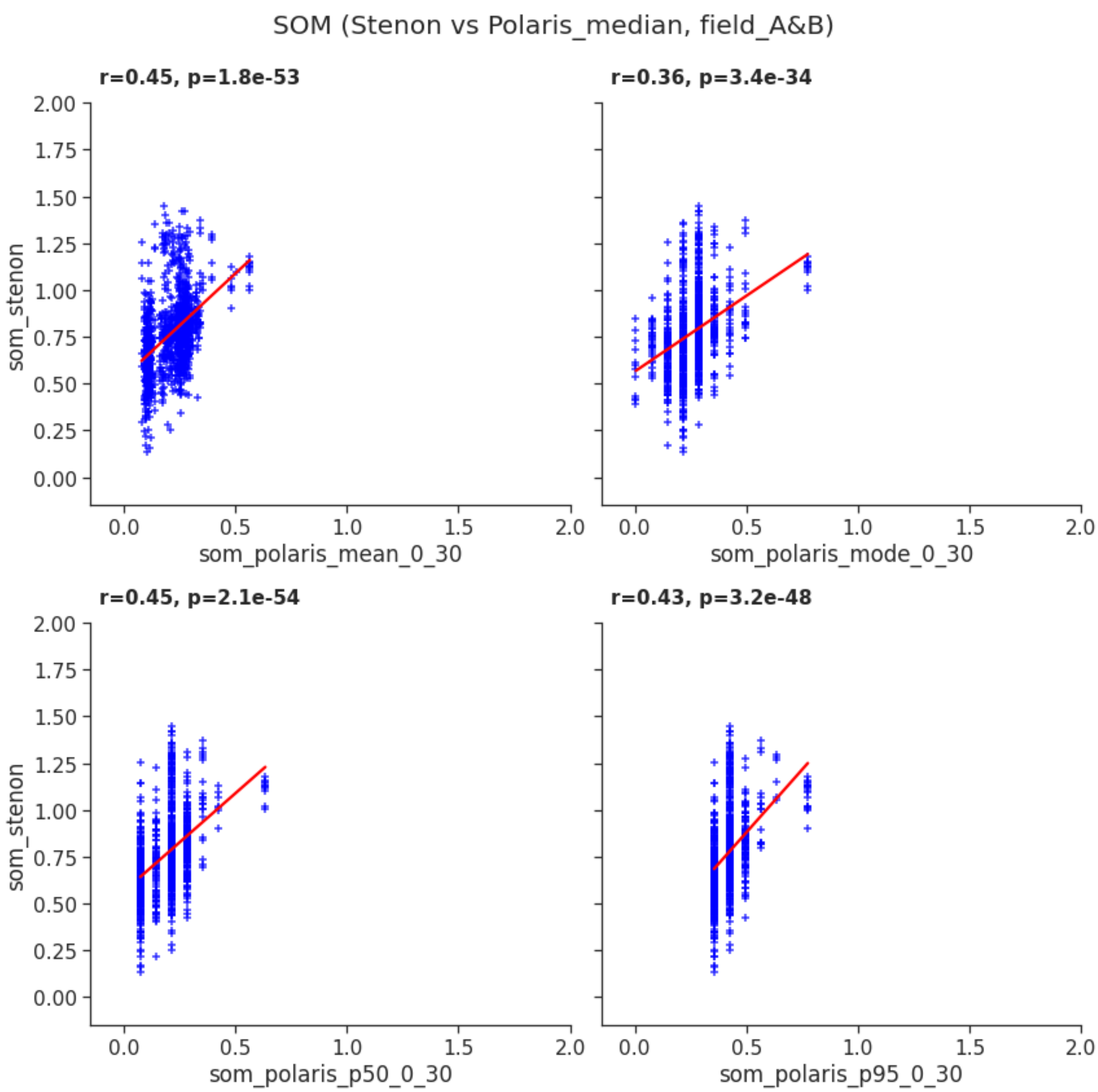


[86]:
```
def plot_stenon_polaris_0_30(df,
                        field_name,
                        y_name = 'som_stenon',
                        features_names = ['som_polaris_mean_0_30',␣
↪'som_polaris_mode_0_30', 'som_polaris_p50_0_30', 'som_polaris_p95_0_30'],
                        t_oulier=None,
                        x_lim_max=None,
                        y_lim_max=None
                        ):

  g = sns.FacetGrid(pd.DataFrame(features_names), col=0, col_wrap=2,␣
↪sharex=False, size = 6)
  for ax, x_var in zip(g.axes, features_names):
    if t_oulier is not None:
      df = df[df[y_name] < t_oulier]

    sns.regplot(data=df, x=x_var, y=y_name, ax=ax,
                scatter_kws={"color": "blue"}, line_kws={"color": "red"},
                scatter = True, ci = 0, fit_reg = True, marker="+")

    r, p = sp.stats.pearsonr(df[x_var], df[y_name])
    ax.text(.02, 1.05, 'r={:.2f}, p={:.2g}'.format(r, p), weight='bold',␣
↪fontsize=15, transform=ax.transAxes)

  g.tight_layout()
```

```
  if x_lim_max is not None:
    g.set(xlim=(-0.15, x_lim_max))
  if y_lim_max is not None:
    g.set(ylim=(-0.15, y_lim_max))

  #move overall title up
  g.fig.subplots_adjust(top=0.9)
  g.fig.suptitle('SOM (Stenon vs Polaris_median, {})'.format(field_name),␣
 ↪fontsize=20)

  plt.savefig('Compare_with_POLARIS_SOM_{}_0_30_median_t_oulier_{}.png'.
 ↪format(field_name, t_oulier))
  plt.show()


# Combine data from multiple-layers to generate 0_30 layer data
new_features_names = []
ts_hilo_corr_attributes = df_data_with_polaris_data.columns[3:-1].tolist()
for i, name in enumerate(DEFAULT_CHALLENGE_STATISTICS):
  df_data_with_polaris_data["som_polaris_%s_0_30"%name] = [np.median(row) for␣
 ↪row in df_data_with_polaris_data[features_names[i::5]].
 ↪itertuples(index=False)]
  new_features_names.append("som_polaris_%s_0_30"%name)

df_field_A_comb =␣
 ↪df_data_with_polaris_data[df_data_with_polaris_data['location']== 'field_A']
df_field_B_comb =␣
 ↪df_data_with_polaris_data[df_data_with_polaris_data['location']== 'field_B']


select_features_names = [item for item in new_features_names if item !=␣
 ↪'som_polaris_p5_0_30']

# Plotting: a combinned layer: 0_30
# # Field A
# plot_stenon_polaris_0_30(df_field_A_comb,␣
 ↪features_names=select_features_names, field_name='field_A', x_lim_max=3,␣
 ↪y_lim_max=3)
# plot_stenon_polaris_0_30(df_field_B_comb,␣
 ↪features_names=select_features_names, field_name='field_B', x_lim_max=3,␣
 ↪y_lim_max=3)
# # Field B
# plot_stenon_polaris_0_30(df_field_A_comb,␣
 ↪features_names=select_features_names, field_name='field_A', t_oulier=2,␣
 ↪x_lim_max=2, y_lim_max=2)
```

```
# plot_stenon_polaris_0_30(df_field_B_comb,␣
↪features_names=select_features_names, field_name='field_B', t_oulier=2,␣
↪x_lim_max=2, y_lim_max=2)

plot_stenon_polaris_0_30(df_data_with_polaris_data,␣
↪features_names=select_features_names, field_name='field_A&B', t_oulier=2,␣
↪x_lim_max=2, y_lim_max=2)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:337: UserWarning: The
`size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



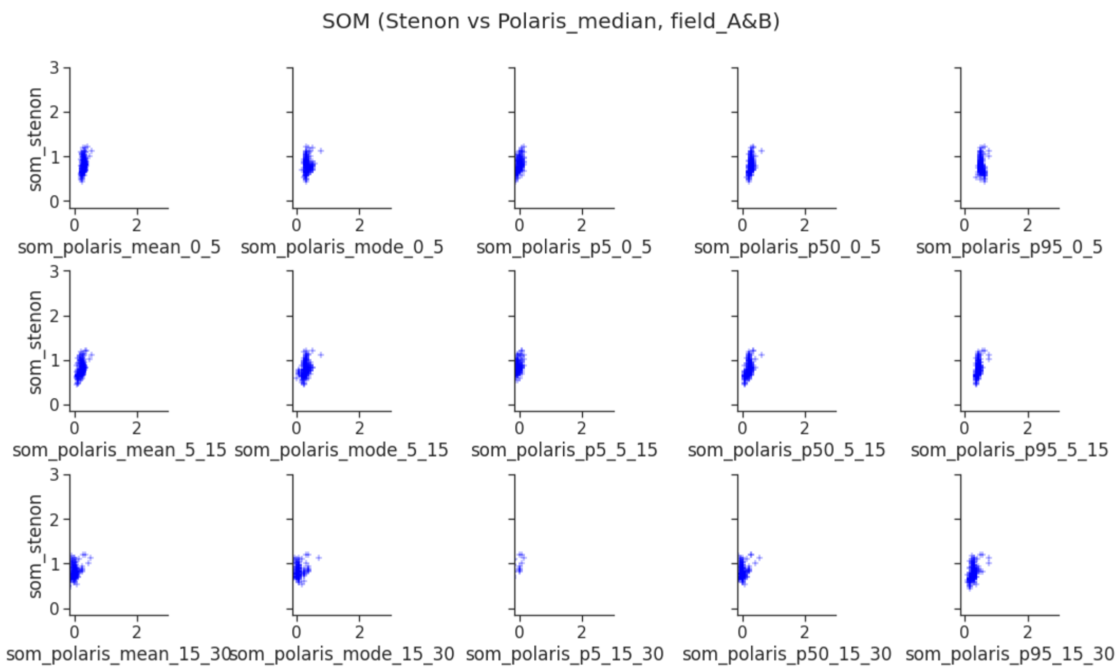
4.6 Group Stenon data by location (POLARIS pixel)

[87]:
```
def group_points_sample(in_vec, out_vec, interest_cols=['lat', 'lng']):
    from sklearn.cluster import DBSCAN

    # https://medium.com/@agarwalvibhor84/
↪lets-cluster-data-points-using-dbscan-278c5459bee5
    df = gpd.read_file(in_vec)

    X = df[interest_cols].to_numpy()
    dbscan = DBSCAN(eps = 0.00007, min_samples = 2)
    model = dbscan.fit(X)

    labels = model.labels_
    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

    # Number of clusters in labels, ignoring noise if present.
    n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
    n_noise_ = list(labels).count(-1)

    print("Estimated number of clusters: %d" % n_clusters_)
    print("Estimated number of noise points: %d" % n_noise_)

    df_label = pd.DataFrame(model.labels_, index=df.index, columns = ["group"])
    shape_gdf = pd.concat([df, df_label], axis=1)
    gdf = gpd.GeoDataFrame(shape_gdf, geometry=shape_gdf.geometry)

    if out_vec is not None:
        # Alternatively, you can write GeoJSON to file:
        gdf.to_file(out_vec, driver="GeoJSON")

    df = gdf.drop(['geometry'], axis=1, errors='ignore')

    return df


# Extract data from POLARIS soil database")
geojson_fn = './data.geojson'
in_rst_list =['{}{}'.format(item[1], os.path.basename(item[0])) for item in␣
↪url_path_lst]
selected_cols=['measurement_ID', 'lat', 'lng', 'location', 'som']
out_geojson_with_polaris = './data_latlng_with_SOM_from_polaris.geojson'

df_latlng_with_polaris_data = ZonalStats(geojson_fn, in_rst_list,␣
↪out_vec=out_geojson_with_polaris, interest_cols=selected_cols)
out_geojson_with_group_by_location = './data_latlng_with_SOM_from_polaris_group.
↪geojson'
```

```
selected_cols = ['lat', 'lng']
df_with_group_by_location = group_points_sample(out_geojson_with_polaris,␣
↪out_vec=out_geojson_with_group_by_location, interest_cols=selected_cols)
df_with_group_by_location.to_csv('./data_clean_with_group_by_location.csv')
```

```
Estimated number of clusters: 156
Estimated number of noise points: 1
```

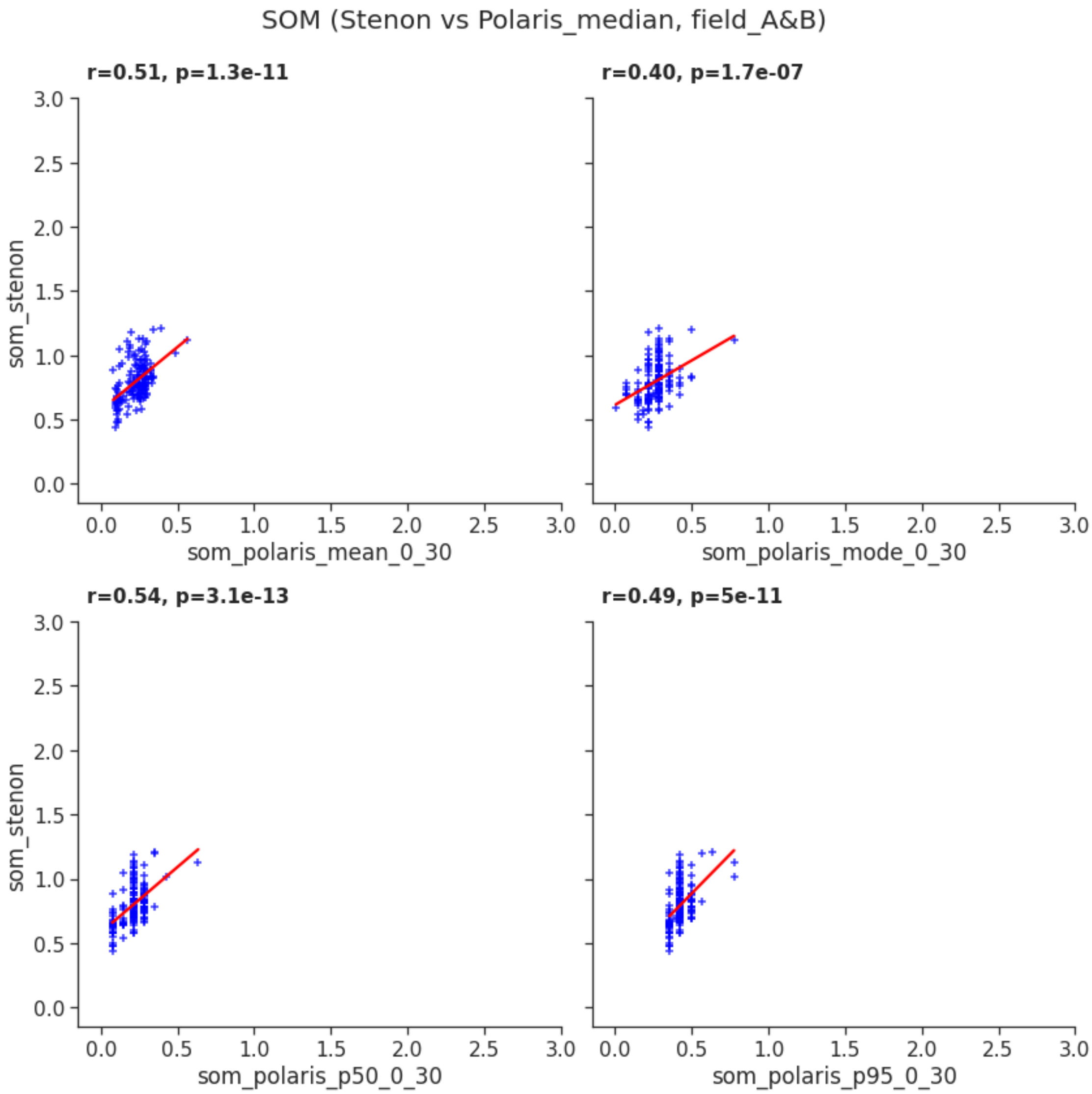
[88]:
```
# # Field A
# df_field_A = df_with_group_by_location[df_with_group_by_location['location']␣
↪== 'field_A'].groupby(['group']).median()
# df_field_A['som_stenon'] = df_field_A[['som']].apply(compute_logData, axis=1)
# features_names = [col for col in df_field_A.columns.tolist() if␣
↪'som_polaris_' in col]

# # Field B
# df_field_B = df_with_group_by_location[df_with_group_by_location['location']␣
↪== 'field_B'].groupby(['group']).median()
# df_field_B['som_stenon'] = df_field_B[['som']].apply(compute_logData, axis=1)
# features_names = [col for col in df_field_B.columns.tolist() if␣
↪'som_polaris_' in col]

# Field A&B
df_field_AB = df_with_group_by_location.groupby(['group']).median()
df_field_AB['som_stenon'] = df_field_AB[['som']].apply(compute_logData, axis=1)
features_names = [col for col in df_field_AB.columns.tolist() if 'som_polaris_'␣
↪in col]

# Plotting: all layers, 0_5, 5_15, 15_30
# plot_stenon_polaris(df_field_A, features_names=features_names, field_name =␣
↪'field_A', x_lim_max=3, y_lim_max=3)
# plot_stenon_polaris(df_field_B, features_names=features_names, field_name =␣
↪'field_B', x_lim_max=3, y_lim_max=3)
plot_stenon_polaris(df_field_AB, features_names=features_names, field_name =␣
↪'field_A&B', x_lim_max=3, y_lim_max=3)
```



[89]:
```
# Combine data from multiple-layers to generate 0_30 layer data
new_features_names = []
for i, name in enumerate(DEFAULT_CHALLENGE_STATISTICS):
  df_with_group_by_location["som_polaris_%s_0_30"%name] = [np.median(row) for␣
 ↪row in df_with_group_by_location[features_names[i::5]].
 ↪itertuples(index=False)]
  new_features_names.append("som_polaris_%s_0_30"%name)

# # Field A
# df_with_group_by_location_field_A =␣
 ↪df_with_group_by_location[df_with_group_by_location['location'] ==␣
 ↪'field_A'].groupby(['group']).median()
# df_with_group_by_location_field_A['som_stenon'] =␣
 ↪df_with_group_by_location_field_A[['som']].apply(compute_logData, axis=1)

# # Field B
# df_with_group_by_location_field_B =␣
 ↪df_with_group_by_location[df_with_group_by_location['location'] ==␣
 ↪'field_B'].groupby(['group']).median()
# df_with_group_by_location_field_B['som_stenon'] =␣
 ↪df_with_group_by_location_field_B[['som']].apply(compute_logData, axis=1)

# Field A&B
df_with_group_by_location_field_AB = df_with_group_by_location.
 ↪groupby(['group']).median()
```

```
df_with_group_by_location_field_AB['som_stenon'] =␣
↪df_with_group_by_location_field_AB[['som']].apply(compute_logData, axis=1)

select_features_names = [item for item in new_features_names if item !=␣
↪'som_polaris_p5_0_30']
# Plotting: a combinned layer: 0_30
# plot_stenon_polaris_0_30(df_with_group_by_location_field_A,␣
↪features_names=select_features_names, field_name='field_A', x_lim_max=3,␣
↪y_lim_max=3)
# plot_stenon_polaris_0_30(df_with_group_by_location_field_B,␣
↪features_names=select_features_names, field_name='field_B', x_lim_max=3,␣
↪y_lim_max=3)
plot_stenon_polaris_0_30(df_with_group_by_location_field_AB,␣
↪features_names=select_features_names, field_name='field_A&B', x_lim_max=3,␣
↪y_lim_max=3)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:337: UserWarning: The
`size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



5 Regression Model building

5.1 Outliers removal (optional)

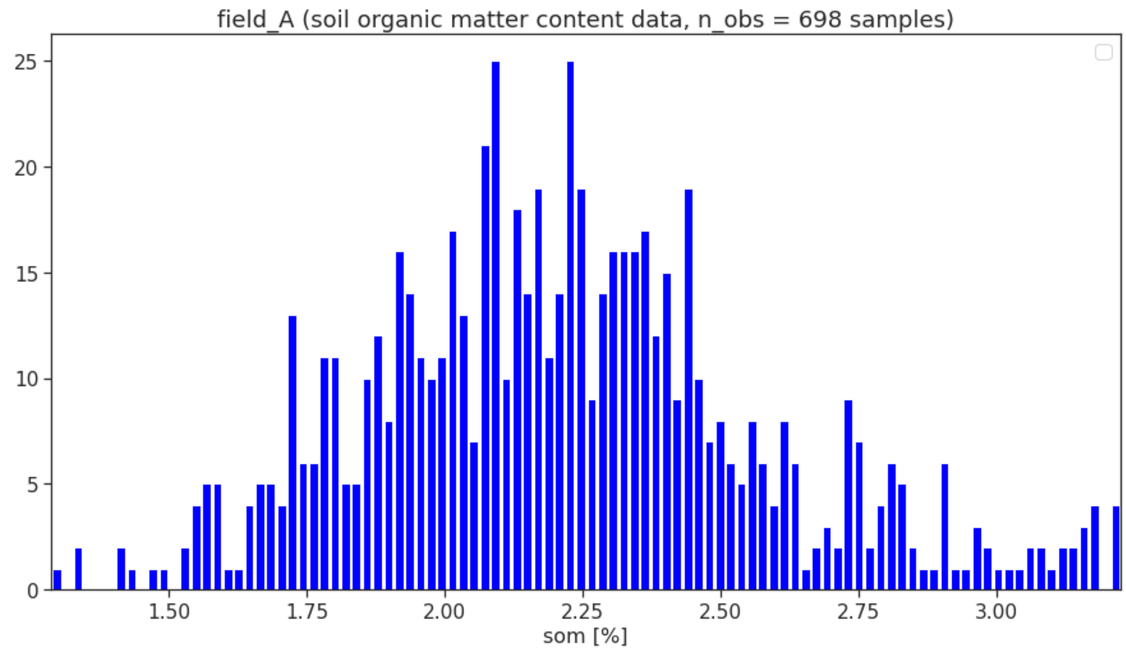
```
[92]: def outlier_removal(df, location_name, col='location', attribute='som'):
        df_field = df[df[col] == location_name]

        Q1 = df_field[attribute].quantile(0.25) # Same as np.percentile but maps␣
      ↪(0,1) and not (0,100)
        Q3 = df_field[attribute].quantile(0.75)
        IQR = Q3 - Q1
```

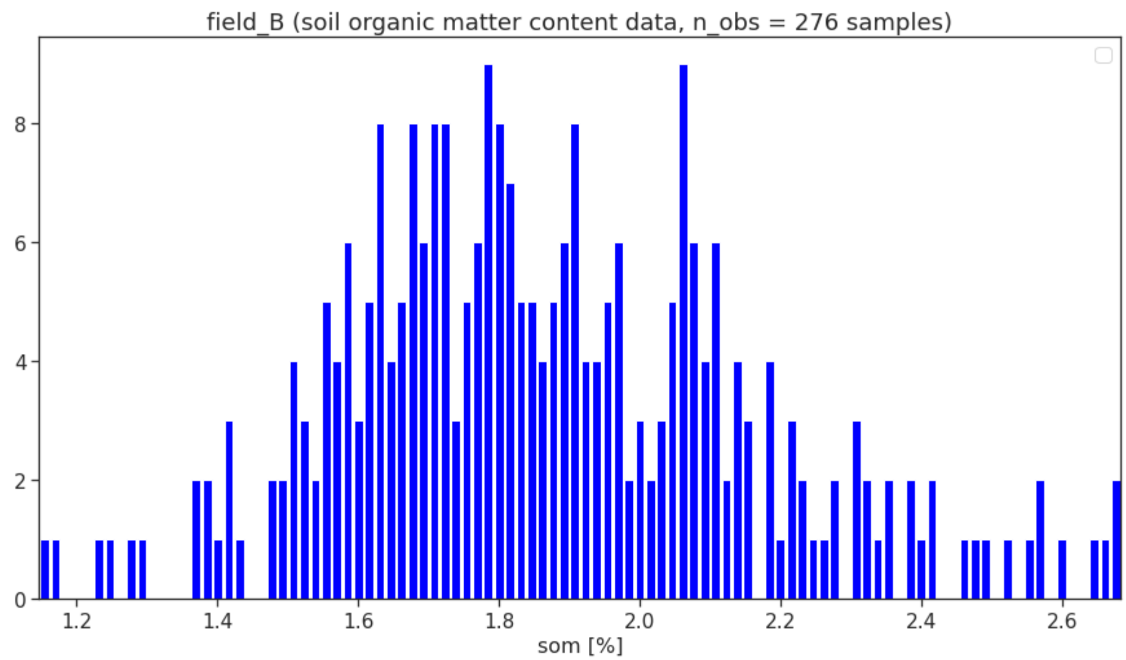
```
  # Return a boolean array of the rows with (any) non-outlier column values
  condition = ~((df_field[attribute] < (Q1 - 1.5 * IQR)) | (df_field[attribute]␣
↪> (Q3 + 1.5 * IQR)))
  filtered_df_field = df_field.loc[condition]

  plt.figure(figsize=(15, 8))
  plt.clf

  plt.hist(filtered_df_field[attribute], bins=100, histtype='bar', rwidth=0.8,␣
↪color='blue')
  plt.xlim(filtered_df_field[attribute].min(), filtered_df_field[attribute].
↪max())
  plt.xlabel("{} [%]".format(attribute), fontsize=16)
  plt.title ("{} (soil organic matter content data, n_obs = {} samples)".
↪format(location_name, filtered_df_field[attribute].shape[0]), fontsize=18)
  plt.legend()

  return filtered_df_field


# for location_name in set(df_data_correct_geo['location'].tolist()):
#   outlier_removal(df_data_correct_geo, location_name)
```

[93]:

```
blunder_removal_action = True
if blunder_removal_action is True:
  df_data_correct_geo_field_A_filtered = outlier_removal(df_data_correct_geo,␣
↪'field_A')
  df_data_correct_geo_field_B_filtered = outlier_removal(df_data_correct_geo,␣
↪'field_B')

  df_data_correct_geo_clean = pd.concat([df_data_correct_geo_field_A_filtered,␣
↪df_data_correct_geo_field_B_filtered])
  df_data_correct_geo_clean.to_csv('./data_clean.csv')
else:
  df_data_correct_geo_clean = df_data_correct_geo
```

```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
```



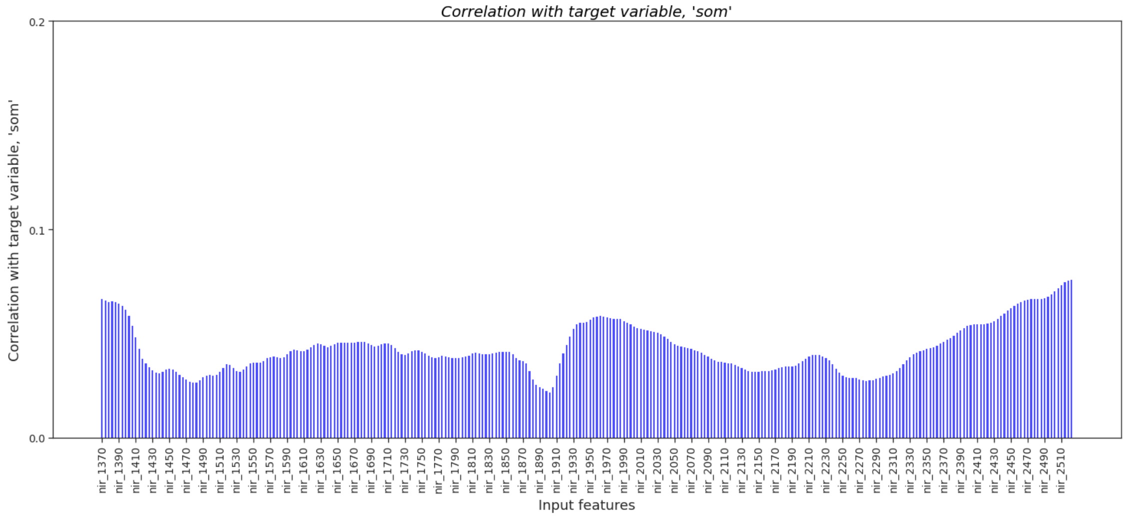


[94]:
```
''' Data preparation for train the model '''
# Full data (Field_A and Field_B)
train_targets_df = df_data_correct_geo_clean['som']
train_features_df = df_data_correct_geo_clean.filter(regex='nir_')
```

```
train_targets_arr = train_targets_df.to_numpy()
train_features_arr = train_features_df.to_numpy()

# Field_A data only
train_field_A_features_df = df_data_correct_geo_clean.
 ↪loc[df_data_correct_geo_clean['location']=='field_A'].filter(regex='nir_')
train_field_A_targets_df = df_data_correct_geo_clean.
 ↪loc[df_data_correct_geo_clean['location']=='field_A']['som']
train_field_A_features_arr = train_field_A_features_df.to_numpy()
train_field_A_targets_arr = train_field_A_targets_df.to_numpy()

# Field_B data only
train_field_B_features_df = df_data_correct_geo_clean.
 ↪loc[df_data_correct_geo_clean['location']=='field_B'].filter(regex='nir_')
train_field_B_targets_df = df_data_correct_geo_clean.
 ↪loc[df_data_correct_geo_clean['location']=='field_B']['som']
train_field_B_features_arr = train_field_B_features_df.to_numpy()
train_field_B_targets_arr = train_field_B_targets_df.to_numpy()
```

5.2 Relationship between input features and target variable

[97]:
```
plot_X_labels = train_features_df.columns.tolist()
plot_correls = np.zeros((train_features_arr.shape[-1]))

for i, val in  enumerate(range(0, train_features_arr.shape[-1])):
  b = np.corrcoef(train_features_arr[:, i], train_targets_arr, rowvar=False)
  plot_correls[i] = b[0,1]

# Figure
fig  = plt.figure(figsize=(25, 10))
ax   = fig.add_subplot(111)
axis_font = {'family':'arial', 'style':'normal', 'size':18}
in_width = 0.5
ar_idxes = np.arange(plot_correls.shape[0])
p1=ax.bar(ar_idxes, plot_correls, in_width, color='blue', edgecolor='none',␣
 ↪alpha=0.7)
plt.xticks(ar_idxes[::5], plot_X_labels[::5], fontsize=14, rotation=90)
plt.yticks(np.arange(0, 0.3, 0.1), fontsize=14)
ax.set_xlabel('Input features', axis_font, labelpad=8)
ax.set_ylabel("Correlation with target variable, 'som'", axis_font, labelpad=8)
plt.title("Correlation with target variable, 'som'", fontsize = 20,␣
 ↪color='Black', fontstyle='italic')
plt.show()
```



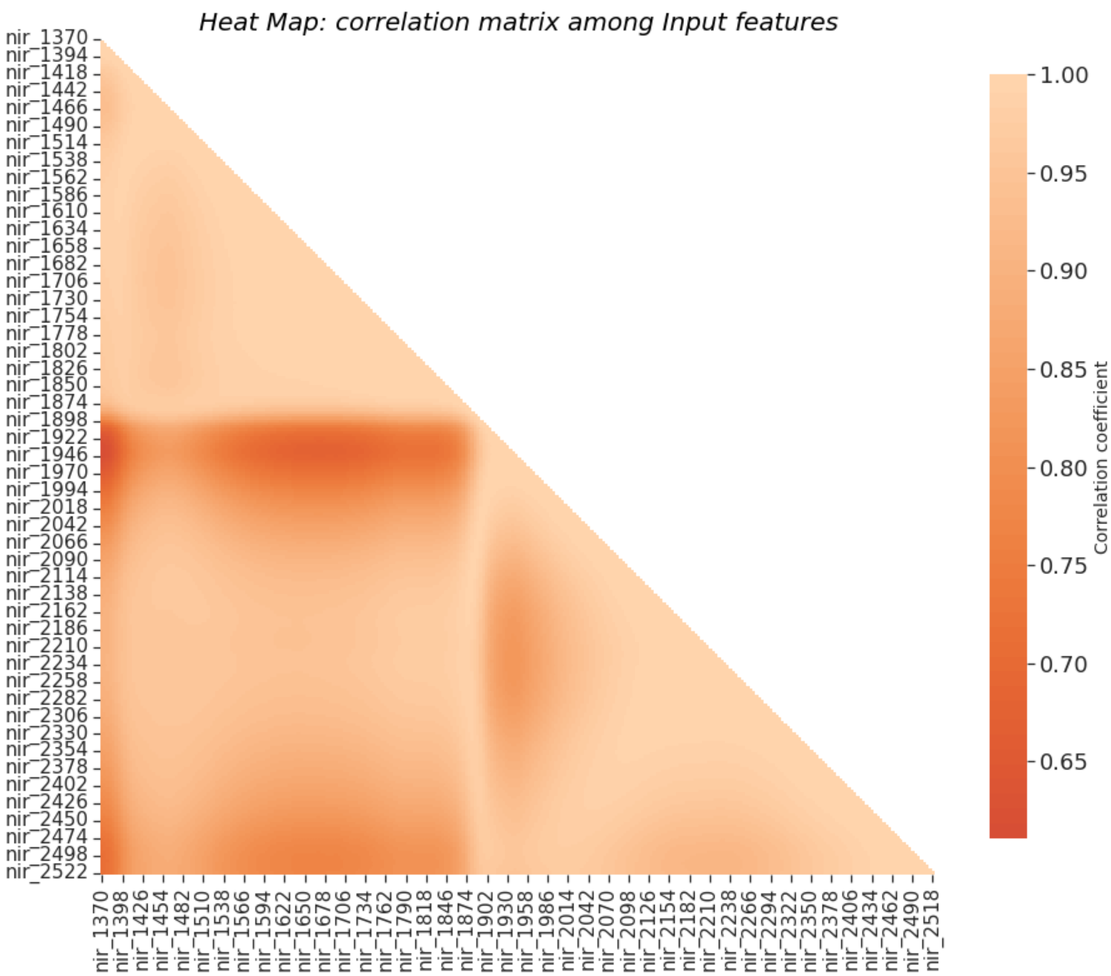

```
[ ]: # for st_label, fo_corr in zip(ve_X_labels, ve_correls):
     #     print(f"{st_label:20} = {fo_corr:5.2f}")
```

```
[100]: # correlation matrix - contains correlation among attributes
       ar_corr_mat = train_features_df.corr().abs()

       # Generate a mask for the upper triangle
       ar_mask = np.zeros_like(ar_corr_mat, dtype=np.bool)
       ar_mask[np.triu_indices_from(ar_mask)] = True

       # Set up the matplotlib figure
       f, ax = plt.subplots(figsize=(15, 15))

       # Generate a custom diverging colormap
       # cmap = sns.diverging_palette(100, 10, as_cmap=True)
       cmap = sns.diverging_palette(220, 20, sep=20, as_cmap=True)

       # Draw the heatmap with the mask and correct aspect ratio
       ax=sns.heatmap(ar_corr_mat, mask=ar_mask, vmax=1.0,
                   center=0, square=True, linewidths=.0, cbar_kws={'label':␣
        ↪'Correlation coefficient', "shrink": .75})

       ax.figure.axes[-1].yaxis.label.set_size(14)
       cbar = ax.collections[0].colorbar
       cbar.ax.tick_params(labelsize=18)

       plt.title("Heat Map: correlation matrix among Input features", fontsize = 20,␣
        ↪color='Black', fontstyle='italic')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5:
```

```
DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To
silence this warning, use `bool` by itself. Doing this will not modify any
behavior and is safe. If you specifically wanted the numpy scalar type, use
`np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  """
```

[100]:
```
Text(0.5, 1.0, 'Heat Map: correlation matrix among Input features')
```



[39]:
```
'''remove highly correlated ones'''
# # Select upper triangle of correlation matrix
# upper = ar_corr_mat.where((np.triu(np.ones(ar_corr_mat.shape), k=1)).
 ↪astype(bool))
# print (upper.shape)

# # Find features with correlation greater than 0.9
```

```
# to_drop = [column for column in upper.columns if any(upper[column] > 0.9)]
# print (to_drop)

# # Drop features
# ar_corr_mat.drop(to_drop, axis=1, inplace=True)
# ar_corr_mat.shape
```

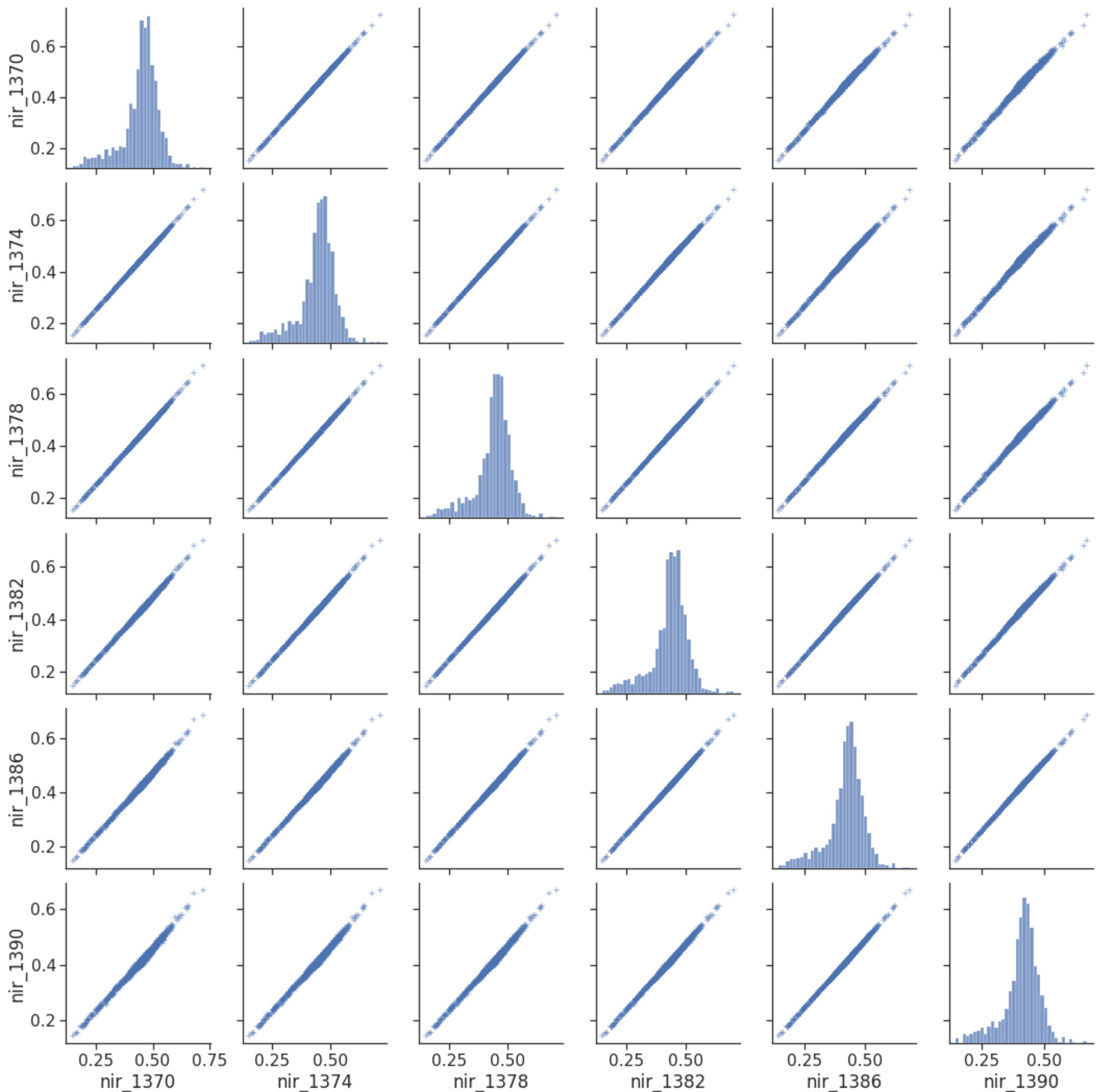
[39]:
```
'remove highly correlated ones'
```

[ ]:
```
# ar_corr_mat['indicator'] = ar_corr_mat[ar_corr_mat.columns.tolist()].ge(0.6).
↪all(axis=1)
# ar_corr_mat[ar_corr_mat['indicator'] == True].shape
```

###Strongly related features (an example)

[106]:
```
# sns.set(style="ticks", color_codes=True, font_scale=1.4)
sns.pairplot(train_features_df.iloc[:, :6], markers='+')
```

[106]:
```
<seaborn.axisgrid.PairGrid at 0x7fab5596a390>
```



5.3 Build Models

[107]:
```
def SOM_models(features, targets, model_selection='Linear', out_model_fn='all',␣
↪n_estimators=100, test_size=0):

  if model_selection in models:
    model_build = False
    # Split the data into training and testing sets
    if test_size>0:
      train_features, test_features, train_target, test_targets =␣
↪train_test_split(features, targets, test_size=test_size)
    else:
      train_features, train_targets = features, targets
```

```
    test_features, test_targets = features, targets

  if model_selection == 'Linear':
    model = linear_model.LinearRegression()
    model.fit(train_features, train_targets)
    model_build = True
  if model_selection == 'Ridge':
    # define cross-validation method to evaluate model
    cv = RepeatedKFold(n_splits=5, n_repeats=3, random_state=1) #
    model = RidgeCV(alphas=arange(0, 1, 0.01), cv=cv,␣
↪scoring='neg_mean_absolute_error')
    model.fit(train_features, train_targets)
    model_build = True
  elif model_selection == 'PLS':
    model = PLSRegression(n_components=150)
    model.fit(train_features, train_targets)
    model_build = True
  elif model_selection == 'RF':
    if n_estimators >0 and isinstance(n_estimators, int):
      # Instantiate model with n_estimators decision trees
      model = RandomForestRegressor(n_estimators=n_estimators)
      # Train the model on training data
      model.fit(train_features, train_targets)
      model_build = True
    else:
      print ('n_estimators must be integer!')
  else:
    # convert the dataset into an optimized data structure called Dmatrix␣
↪that XGBoost supports
    data_dmatrix = xgb.DMatrix(data=train_features,label=train_targets)
    model = xgb.XGBRegressor(objective='reg:squarederror',
                             colsample_bytree=0.1,
                             learning_rate=0.1,
                             max_depth=10,
                             alpha=5,
                             n_estimators=80)

    model.fit(train_features, train_targets)
    model_build = True

  if model_build:
    pickle.dump(model, open('./{}_{}.pkl'.format(model_selection,␣
↪out_model_fn), 'wb'))
    return model
 else:
  print ('Model {} is not implimented yet!'.format(model_selection))
```

5.4 Train Models

[ ]:
```
# List of Models for performance assessment
models = ['Ridge', 'PLS']

out_model_fn = 'all'
for index, model_selection in enumerate(models):
  SOM_models(train_features_arr, train_targets_arr,␣
 ↪model_selection=model_selection, out_model_fn=out_model_fn)

out_model_fn = 'field_A'
for index, model_selection in enumerate(models):
  SOM_models(train_field_A_features_arr, train_field_A_targets_arr,␣
 ↪model_selection=model_selection, out_model_fn=out_model_fn)

out_model_fn = 'field_B'
for index, model_selection in enumerate(models):
  SOM_models(train_field_B_features_arr, train_field_B_targets_arr,␣
 ↪model_selection=model_selection, out_model_fn=out_model_fn)
```

5.5 Evaluate the performance of the Model

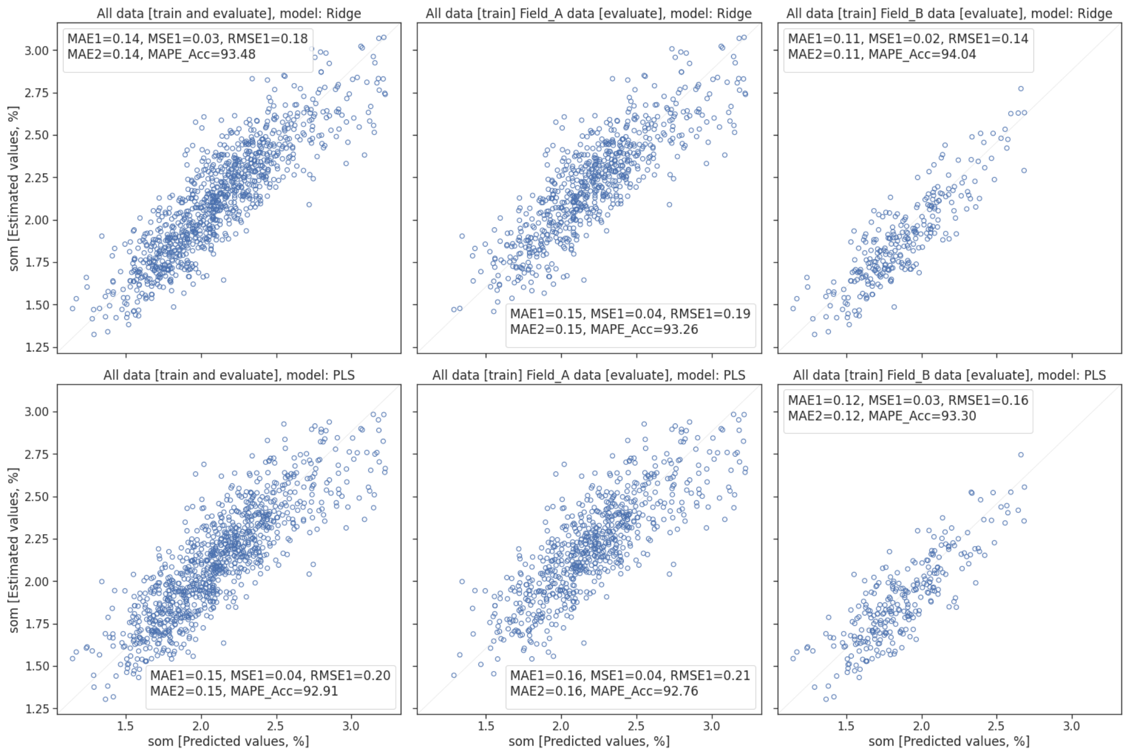
[135]:
```
def model_validation_plot(predictions, estimations, ax, title):

  ### Plot the scatter of the result and R2 calculation:
  # r_squared = r2_score(estimations, predictions)
  # Calculate the absolute errors
  errors = abs(predictions - estimations)
  # Calculate mean absolute percentage error (MAPE)
  mape = 100 * (errors / estimations)
  # Calculate and display accuracy
  accuracy = 100 - np.mean(mape)

  # # Errors test
  MAE1 = metrics.mean_absolute_error(estimations, predictions)
  MSE1 = metrics.mean_squared_error(estimations, predictions)
  RMSE1 = np.sqrt(metrics.mean_squared_error(estimations, predictions))
  MAE2 = round(np.mean(errors), 2)
  Acc = round(accuracy, 2)

  x = np.unique(estimations)
  y = np.poly1d(np.polyfit(estimations, predictions, 1))(np.unique(estimations))

  ax.scatter(predictions, estimations, facecolors='none', edgecolors='b')
  # ax.plot(x, y, color="r", lw=2, label='Fitting line')
  ax.set_xlim(1, 4)
  ax.set_ylim(1, 4)
```

```
    ax.set_title(title)
    # ax.legend(title='MAE1=%.2e, MSE1=%0.2e, RMSE1=%0.2e\nMAE2=%0.2e␣
↪\nMAPE_Acc=%0.2e'%(MAE1, MSE1, RMSE1, MAE2, Acc))
    ax.legend(title='MAE1=%.2f, MSE1=%0.2f, RMSE1=%0.2f\nMAE2=%0.2f, MAPE_Acc=%0.
↪2f'%(MAE1, MSE1, RMSE1, MAE2, Acc))

    import matplotlib.lines as mlines
    line = mlines.Line2D([0, 1], [0, 1], color='k', linestyle='--', linewidth=0.1)
    transform = ax.transAxes
    line.set_transform(transform)
    ax.add_line(line)
```

[116]:
```
out_model_fn = 'all'
fig, axs = plt.subplots(2, 3, figsize=(21,14), constrained_layout=True,␣
↪sharex=True, sharey=True)
# fig.tight_layout(pad=3.0)
for index, model_selection in enumerate(['Ridge', 'PLS']):
    pickled_model = pickle.load(open('./{}_{}.pkl'.format(model_selection,␣
↪out_model_fn), 'rb'))

    # All (Field_A and Field_B)
    predicted_targets = pickled_model.predict(train_features_arr).flatten()
    # For Field_A only
    predicted_field_A_targets = pickled_model.predict(train_field_A_features_arr).
↪flatten()
    # For Field_B only
    predicted_field_B_targets = pickled_model.predict(train_field_B_features_arr).
↪flatten()

    model_validation_plot(train_targets_arr, predicted_targets, axs[index, 0],␣
↪'All data [train and evaluate], model: {}'.format(model_selection))
    model_validation_plot(train_field_A_targets_arr, predicted_field_A_targets,␣
↪axs[index, 1], 'All data [train] Field_A data [evaluate], model: {}'.
↪format(model_selection))
    model_validation_plot(train_field_B_targets_arr, predicted_field_B_targets,␣
↪axs[index, 2], 'All data [train] Field_B data [evaluate], model: {}'.
↪format(model_selection))

for ax in axs.flat:
    ax.set(xlabel='som [Predicted values, %]', ylabel='som [Estimated values,␣
↪%]')

# Hide x labels and tick labels for top plots and y ticks for right plots.
for ax in axs.flat:
    ax.label_outer()
```

```
No handles with labels found to put in legend.
```

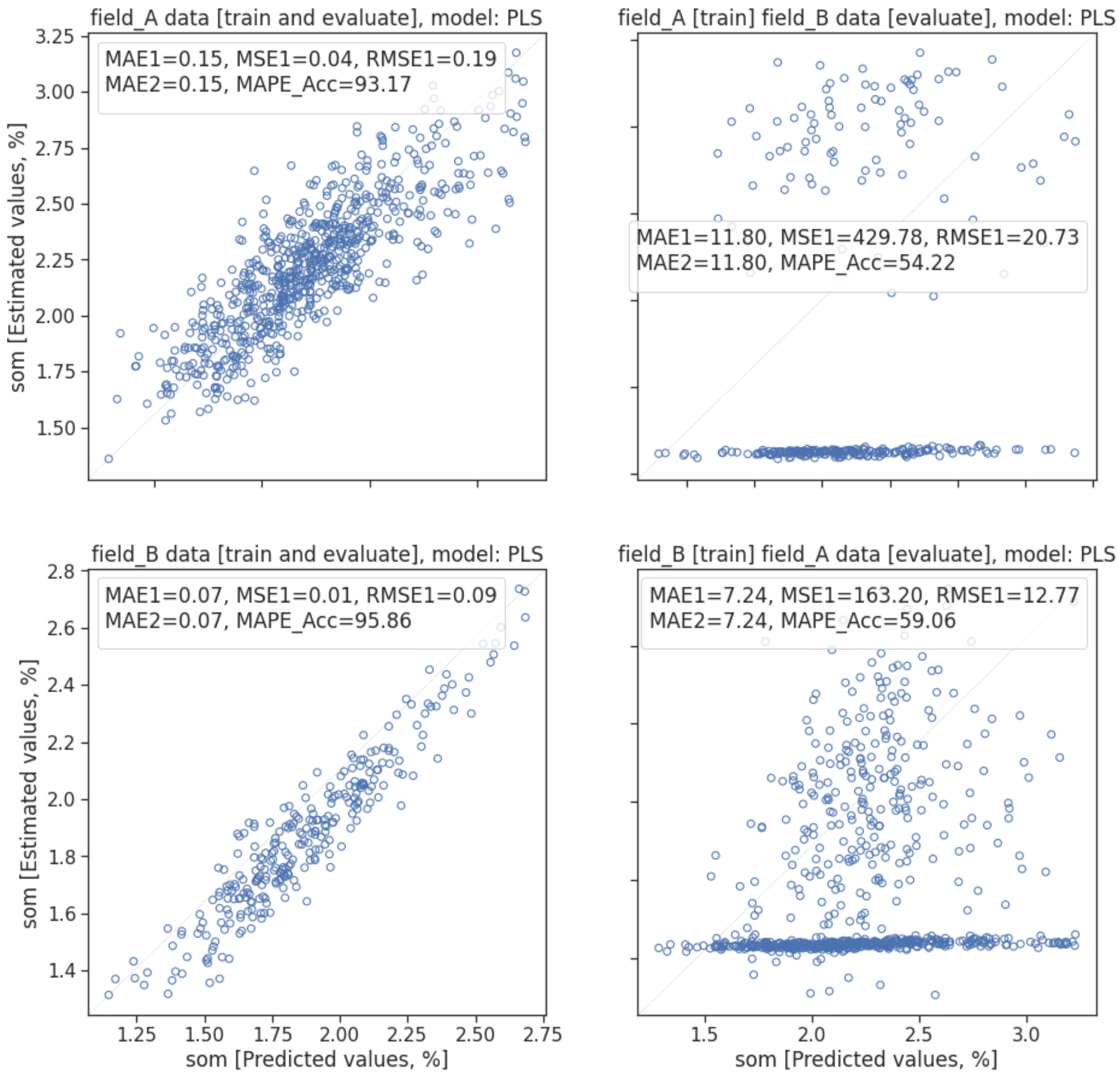
```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
```



[117]:
```
fig, axs = plt.subplots(2, 2, figsize=(15,15))

model_selection='PLS'

# Field_A based
out_model_fn = 'field_A'
pickled_model = pickle.load(open('./{}_{}.pkl'.format(model_selection,␣
↪out_model_fn), 'rb'))
# For Field_A only
predicted_field_A_targets = pickled_model.predict(train_field_A_features_arr).
↪flatten()
# For Field_B only
predicted_field_B_targets = pickled_model.predict(train_field_B_features_arr).
↪flatten()
model_validation_plot(train_field_A_targets_arr, predicted_field_A_targets,␣
↪axs[0, 0], '{} data [train and evaluate], model: {}'.format(out_model_fn,␣
↪model_selection))
```

```
model_validation_plot(train_field_B_targets_arr, predicted_field_B_targets,␣
↪axs[0, 1], '{} [train] field_B data [evaluate], model: {}'.
↪format(out_model_fn, model_selection))

# Field_B based
out_model_fn = 'field_B'
pickled_model = pickle.load(open('./{}_{}.pkl'.format(model_selection,␣
↪out_model_fn), 'rb'))
# For Field_A only
predicted_field_A_targets = pickled_model.predict(train_field_A_features_arr).
↪flatten()
# For Field_B only
predicted_field_B_targets = pickled_model.predict(train_field_B_features_arr).
↪flatten()
model_validation_plot(train_field_B_targets_arr, predicted_field_B_targets,␣
↪axs[1, 0], '{} data [train and evaluate], model: {}'.format(out_model_fn,␣
↪model_selection))
model_validation_plot(train_field_A_targets_arr, predicted_field_A_targets,␣
↪axs[1, 1], '{} [train] field_A data [evaluate], model: {}'.
↪format(out_model_fn, model_selection))

for ax in axs.flat:
    ax.set(xlabel='som [Predicted values, %]', ylabel='som [Estimated values,␣
↪%]')

# Hide x labels and tick labels for top plots and y ticks for right plots.
for ax in axs.flat:
    ax.label_outer()
```

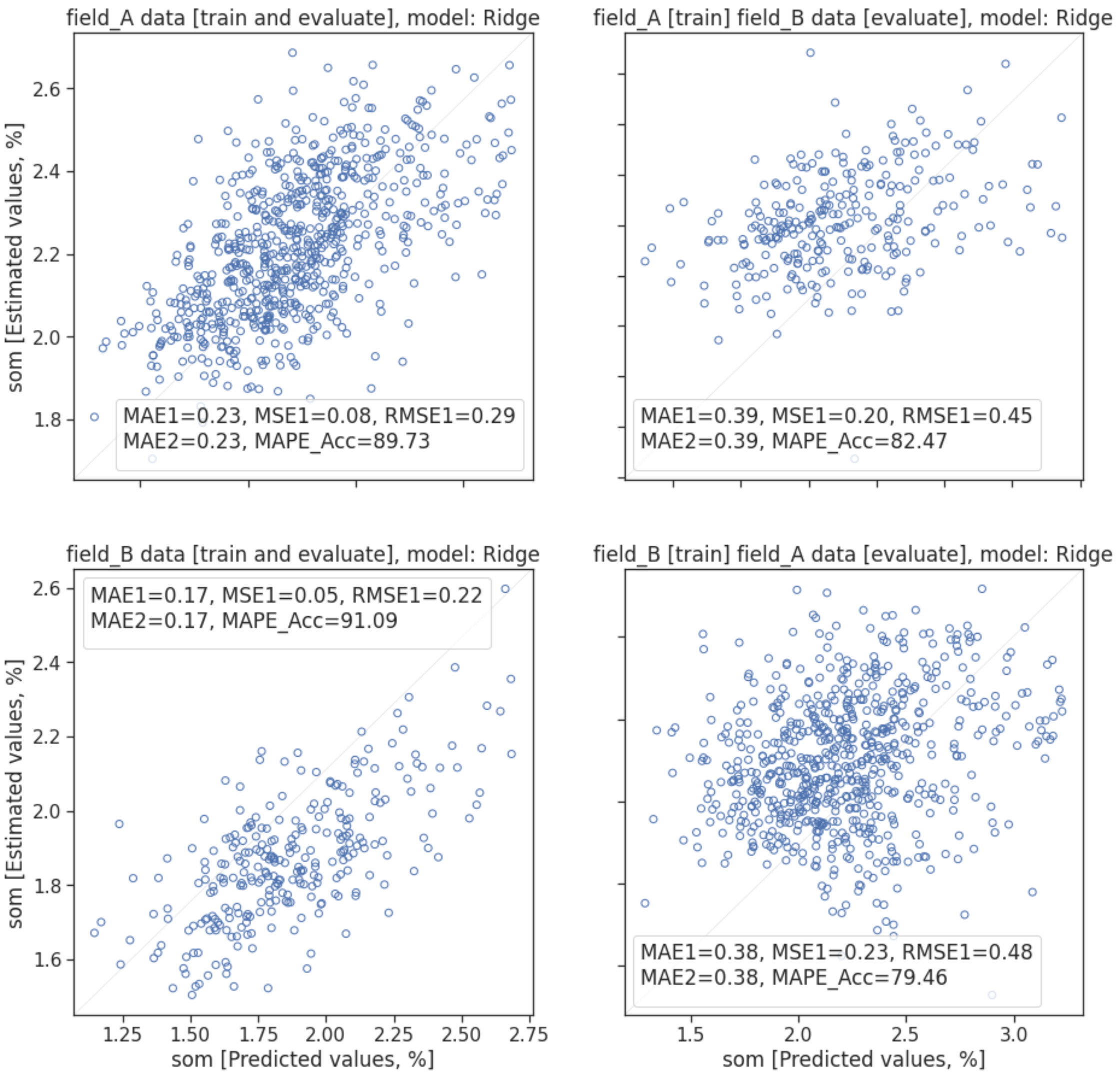
```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
```



[118]:
```
fig, axs = plt.subplots(2, 2, figsize=(15,15))

model_selection='Ridge'

# Field_A based
out_model_fn = 'field_A'
pickled_model = pickle.load(open('./{}_{}.pkl'.format(model_selection,␣
↪out_model_fn), 'rb'))
# For Field_A only
predicted_field_A_targets = pickled_model.predict(train_field_A_features_arr).
↪flatten()
# For Field_B only
predicted_field_B_targets = pickled_model.predict(train_field_B_features_arr).
↪flatten()
```

```
model_validation_plot(train_field_A_targets_arr, predicted_field_A_targets,␣
↪axs[0, 0], '{} data [train and evaluate], model: {}'.format(out_model_fn,␣
↪model_selection))
model_validation_plot(train_field_B_targets_arr, predicted_field_B_targets,␣
↪axs[0, 1], '{} [train] field_B data [evaluate], model: {}'.
↪format(out_model_fn, model_selection))

# Field_B based
out_model_fn = 'field_B'
pickled_model = pickle.load(open('./{}_{}.pkl'.format(model_selection,␣
↪out_model_fn), 'rb'))
# For Field_A only
predicted_field_A_targets = pickled_model.predict(train_field_A_features_arr).
↪flatten()
# For Field_B only
predicted_field_B_targets = pickled_model.predict(train_field_B_features_arr).
↪flatten()
model_validation_plot(train_field_B_targets_arr, predicted_field_B_targets,␣
↪axs[1, 0], '{} data [train and evaluate], model: {}'.format(out_model_fn,␣
↪model_selection))
model_validation_plot(train_field_A_targets_arr, predicted_field_A_targets,␣
↪axs[1, 1], '{} [train] field_A data [evaluate], model: {}'.
↪format(out_model_fn, model_selection))

for ax in axs.flat:
    ax.set(xlabel='som [Predicted values, %]', ylabel='som [Estimated values,␣
↪%]')

# Hide x labels and tick labels for top plots and y ticks for right plots.
for ax in axs.flat:
    ax.label_outer()
```

```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
```


field_A data [train and evaluate], model: Ridge
MAE1=0.23, MSE1=0.08, RMSE1=0.29
MAE2=0.23, MAPE_Acc=89.73
field_A [train] field_B data [evaluate], model: Ridge
MAE1=0.39, MSE1=0.20, RMSE1=0.45
MAE2=0.39, MAPE_Acc=82.47
field_B data [train and evaluate], model: Ridge
MAE1=0.17, MSE1=0.05, RMSE1=0.22
MAE2=0.17, MAPE_Acc=91.09
field_B [train] field_A data [evaluate], model: Ridge
MAE1=0.38, MSE1=0.23, RMSE1=0.48
MAE2=0.38, MAPE_Acc=79.46
som [Estimated values, %]
som [Predicted values, %]
1.8
2.0
2.2
2.4
2.6
1.6
1.25
1.50
1.75
2.00
2.25
2.50
2.75
1.5
2.0
2.5
3.0