STUDENT SUMMER INTERNSHIP TECHNICAL REPORT

pyLEnM: A Python Package for Long-Term Soil and Groundwater Monitoring

DOE-FIU SCIENCE & TECHNOLOGY
WORKFORCE DEVELOPMENT PROGRAM

Date submitted:

December 4, 2020

Principal Investigators:

Aurelien Meray (DOE Fellow Student)
Florida International University

Haruko Wainwright (Mentor)
Lawrence Berkeley National Laboratory

Ravi Gudavalli, Ph.D. (Program Manager)
Florida International University

Leonel Lagos, Ph.D., PMP® (Program Director)
Florida International University

Submitted to:

U.S. Department of Energy
Office of Environmental Management
Under Cooperative Agreement # DE-EM0000598

FIU | Applied Research Center
FLORIDA INTERNATIONAL UNIVERSITY

ABSTRACT

Recent technological advances - in situ groundwater sensors, geophysics, drone/satellite-based remote sensing, reactive transport modeling, and AI - have a great potential to establish the new paradigm of long-term monitoring with improved effectiveness and reliability at contaminated groundwater sites. In situ sensors prove to be a powerful alternative to traditional groundwater sampling and laboratory analysis; particularly for monitoring master variables that are often leading indicators of changes prior to plume movement. With these advancements, there are still challenges and problems to solve such as where to place new sensors, which in situ variables contribute the most information, and how to effectively predict plume movement through contaminant concentration estimations. The research described herein involves the development of a suite of machine learning algorithms in the form of a python package to analyze monitoring datasets effectively. Particular focus was on extracting critical information from a historical dataset, by analyzing multiple time series of groundwater contamination data and groundwater quality parameters such as pH, water table, and specific conductance. The algorithms developed analyze and visualize the correlations between different analytes and help identify key parameters that control contaminant concentrations and plume mobilities. In parallel, regression models were developed to predict when the contaminant concentrations are expected to reach below the regulatory standard. In addition, principal component analysis and clustering analysis were used to group existing wells that have similar groundwater dynamics to more effectively select among existing wells for new sensor installations.

TABLE OF CONTENTS

LIST OF FIGURES

1. EXECUTIVE SUMMARY

This research work has been supported by the DOE-FIU Science & Technology Workforce Development Initiative, an innovative program developed by the US Department of Energy's Environmental Management (DOE-EM) Office and Florida International University's Applied Research Center (FIU ARC). During the summer of 2020, a DOE Fellow intern, Aurelien Meray, spent 10 weeks doing a remote summer internship at Lawrence Berkeley National Laboratory under the supervision and guidance of Research Scientist, Haruko Wainwright. The intern's project was initiated on June 8, 2020 and continued through August 14, 2020, with the objective of creating a python package to perform soil and groundwater data analysis and visualization.

2. RESEARCH DESCRIPTION

2.1. Introduction

The ultimate goal of the research was to propel the AI effort for the Advanced Long-Term Monitoring Systems (ALTEMIS) project by developing a python package to perform data analysis and provide visualization tools for soil and groundwater datasets. Before the package could be generalized and used across multiple sites, a dataset from the Savanah River Site (SRS) F-Area was utilized to validate the python package. For over 30 years, nuclear facilities discharged low-level radioactive wastewater at the site, resulting in soil and groundwater contamination with various radioactive and non-radioactive contaminants. Some of these contaminants include uranium, tritium, strontium-90, and iodine-129. Since the 1990s, the Department of Energy (DOE) has installed wells mounted with sensors across the site measuring a variety of parameters such as pH, water table, and specific conductance. For the ALTEMIS project, creating the package is one of many milestones for establishing a long-term monitoring solution for DOE legacy sites.

2.2. Python Package

The long-term objective of creating the pyLEnM package is to help environmental scientists analyze their unique datasets at different DOE sites. A python package is a library of functions that is generalized enough to be used across multiple domains. Creating a package brings many advantages to users; modularity is one of them. Modularity allows the user to focus primarily on the problem at hand rather than on the implementation of how to solve the task. Maintaining this separation between the function logic and usability simplifies and can accelerate the user experience. Another benefit of using a package in developing code is the reusability aspect. The same functionality can be used anywhere in the code without explicitly duplicating code blocks.

Python packages are easily accessible to anyone thanks to the Python Package Index (PyPi). PyPi is the central repository of software for the Python programming language. As can be seen in Figure 1 below, one accomplishment of this internship was publishing of the pyLEnM package on PyPi. Having this repository makes the maintenance of the package’s code very manageable.

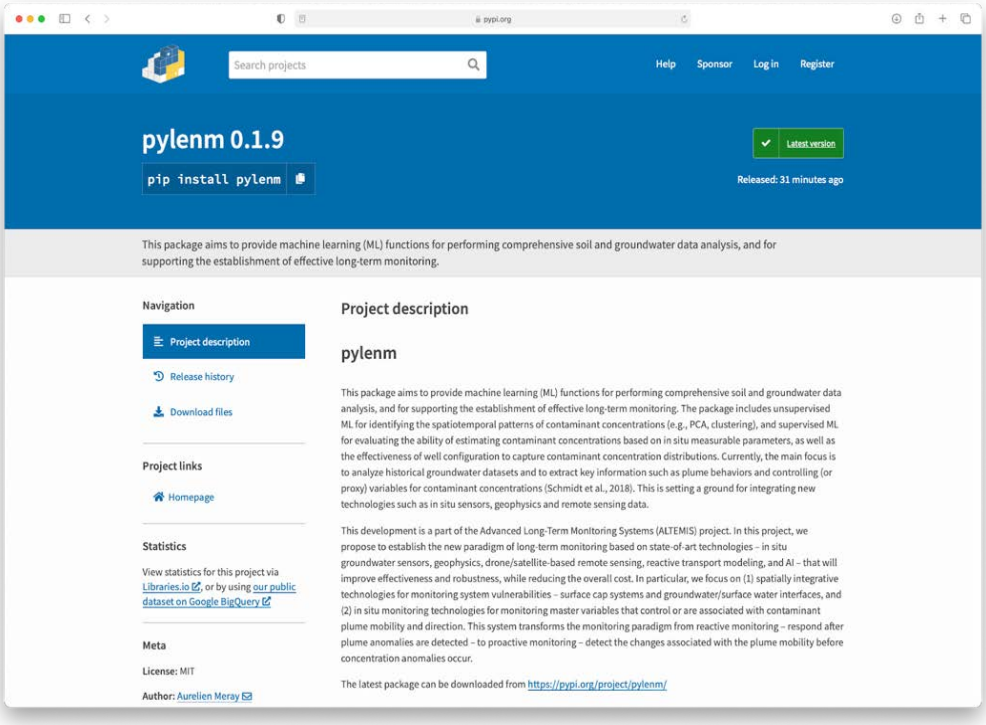

Figure 1. pyLEnM published on the Python Package Index. (https://pypi.org/project/pylenm/)

2.3. F-Area Dataset

As previously mentioned, the data used for this project was the Savanah River Site's F-Area well dataset. The original comma-separated values (csv) file contains over 30 columns, but for analysis, only five (5) useful columns were preserved. These include the analyte collection date, well name (station_id), analyte name, concentration reading (result), and the concentration unit (result_unit). Figure 2 shows a sample of the dataset containing the columns deemed most useful.

	COLLECTION_DATE	STATION_ID	ANALYTE_NAME	RESULT	RESULT_UNITS
0	1990-01-01	FSB 93C	SPECIFIC CONDUCTANCE	345.00	uS/cm
1	1990-01-01	FSB111D	BENZENE	1.00	ug/L
2	1990-01-01	FSB 93C	1,2-DICHLOROPROPANE	1.00	ug/L
3	1990-01-01	FSB105C	FLUORIDE	2.50	mg/L
4	1990-01-01	FSB105C	BARIUM	559.00	ug/L
...	...	...	...	...	...
629770	2015-09-24	FOB 13D	DEPTH_TO_WATER	21.50	ft
629771	2015-09-24	FBI 14D	DEPTH_TO_WATER	17.40	ft
629772	2015-09-24	FBI 17D	DEPTH_TO_WATER	25.16	ft
629773	2015-09-24	FSB117D	DEPTH_TO_WATER	23.46	ft
629774	2015-09-24	FSB 79	DEPTH_TO_WATER	19.76	ft

629775 rows × 5 columns

Figure 2. F-Area dataset sample.

The dataset contains 422 different analytes with the most important ones being tritium, uranium-238, iodine-129, specific conductance, pH, and water table (depth_to_water). In addition, there are over 80 wells collecting and measuring analyte data at multiple depths denoted by the postfix letter of the well name (A being the lowest and D being the highest depth). Lastly, the earliest collection date is in 1990 and the last recorded entry is in 2015.

2.4. Package Functions

Many different functions were created as part of the pyLEnM package to perform various tasks. These functions can be broadly categorized into two groups: 1) the data exploration and transformation functions and 2) the visualization functions. This section will describe the most important of these functions. A list of all the functions can be seen in Figure 3 and the package documentation can be found in Appendix A.

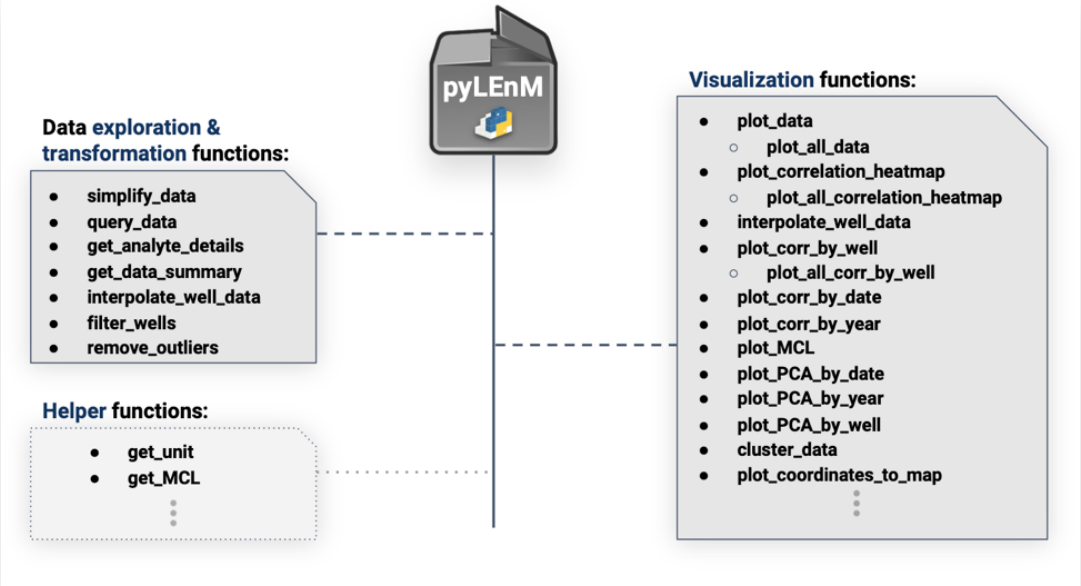

Figure 3. Overview of pyLEnM functions.

2.4.1 Data exploration and transformation functions

The function *get_analyte_details*, seen in Figure 4, provides well information for a given analyte such as the collection date range and the number of samples collected by the well. This function is useful for quickly determining the wells with the highest amounts of data. Another data summarization function is *get_data_summary*, which looks at certain analytes in its entirety. It provides information such as the number of wells the analyte is collected at, with statistical information on the measurements such as the average, quartiles, and minimum and maximum values. A sample result from this function can be seen in Figure 5.

```
pylenm.get_analyte_details('IODINE-129')
```

Well Name	Start Date	End Date	Date Range (days)	Unique samples	Cumulative samples
FSB 87B	1990-01-06	2002-02-14	4422	15	15
FSB 87A	1990-01-06	2006-01-23	5861	19	34
FSB106C	1990-01-06	2006-01-26	5864	19	53
FSB 78A	1990-01-06	2015-01-20	9145	27	80
FSB 87C	1990-01-06	2015-01-20	9145	27	107
...	...	...	...	...	...
FSB146D	2015-04-29	2015-09-09	133	9	4689
FSB145D	2015-04-30	2015-09-09	132	9	4698
FSB143D	2015-05-04	2015-09-09	128	9	4707
FSB144D	2015-05-04	2015-09-09	128	9	4716
FSB142D	2015-05-05	2015-09-09	127	9	4725

Figure 4. ***get_analyte_details*** **function for iodine-129.**

```
pylenm.get_data_summary(analytes=['TRITIUM','IODINE-129','SPECIFIC CONDUCTANCE',
                                  'PH','URANIUM-238', 'DEPTH_TO_WATER'],
                        sort_by='wells')
```

Analyte Name	Start Date	End Date	Date Range (days)	Unique wells	Samples	Result unit	Result mean	Result std	Result min	Result 25%	Result 50%	Result 75%	Result max
TRITIUM	1990-01-01	2015-09-23	9396 days	160	13476	pCi/mL	1248.316406	3261.565925	-0.628	25.300	193.00	934.00	44800.0
PH	1990-01-01	2015-09-23	9396 days	158	18517	pH	5.354886	1.529332	0.000	4.100	5.20	6.40	12.7
SPECIFIC CONDUCTANCE	1990-01-01	2015-09-23	9396 days	156	18276	uS/cm	425.754861	555.068915	0.000	87.000	213.00	544.00	10700.0
DEPTH_TO_WATER	1990-01-01	2015-09-24	9397 days	155	20091	ft	46.814224	38.137888	0.500	16.450	32.17	72.90	177.2
IODINE-129	1990-01-06	2015-09-23	9391 days	150	5405	pCi/L	41.624339	92.511351	-356.900	0.996	9.37	45.10	1620.0
URANIUM-238	1990-01-06	2015-09-23	9391 days	144	5319	pCi/L	66.398114	169.240747	-2.650	0.062	0.64	43.75	1810.0

Figure 5. ***get_data_summary*** **function for 6 analytes.**

One of the main challenges when working with time series data is the inconsistency at which samples are recorded. In this case, some wells were installed in the early 90s while others were only installed a decade later. In addition, some wells have sensors that only record a certain number of analytes and not others. To complicate things further, not every sensor records at the same time interval; some are taken on a biweekly basis while others are recorded as little as once every six months. As one can see, these varying factors can make analysis very difficult, as algorithms require consistency. To combat these issues, the data must be transformed to create equally spaced intervals, a process known as resampling. We resampled each variable at the same interval so that they align on the same dates. As one can imagine, this process creates empty slots of data at certain

dates, so interpolation is necessary. Interpolation is estimating the values between two known points. In this project, we decided to interpolate linearly, but this parameter can easily be changed. This idea of data transformation by resampling and interpolating is shown in Figure 6 below. The function *interpolate_wells_by_analyte* in pyLEnM takes advantage of this concept to generate consistent data.

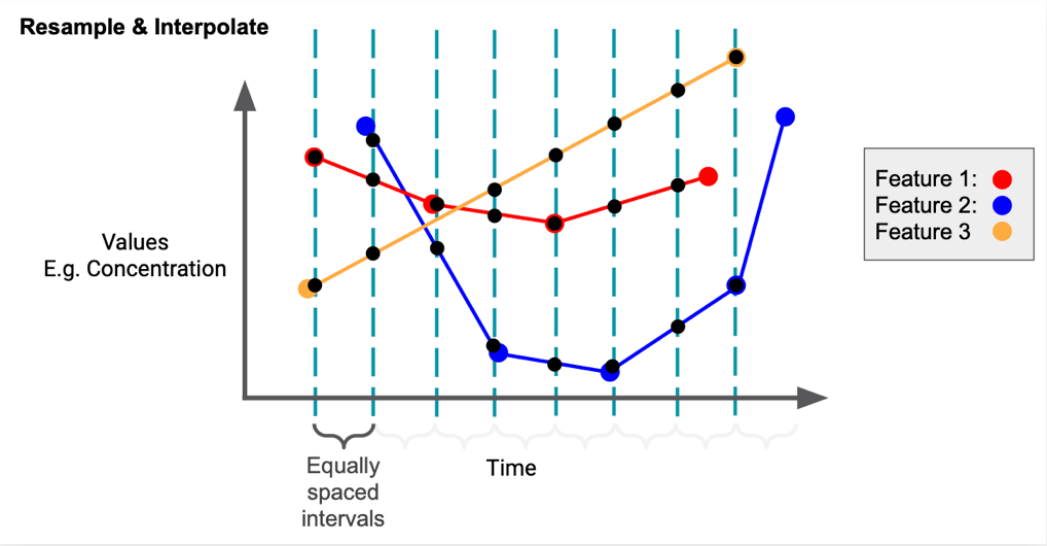

Figure 6. Concept of resampling and interpolating.

2.4.2 Data visualization functions

The pyLEnM package contains a variety of visualization functions. The *plot_data* function allows the user to view the log-concentration values for an analyte at a well with the outliers highlighted in red. As can be seen in Figure 7, tritium levels seem to be diminishing at the FSB 95DR well.

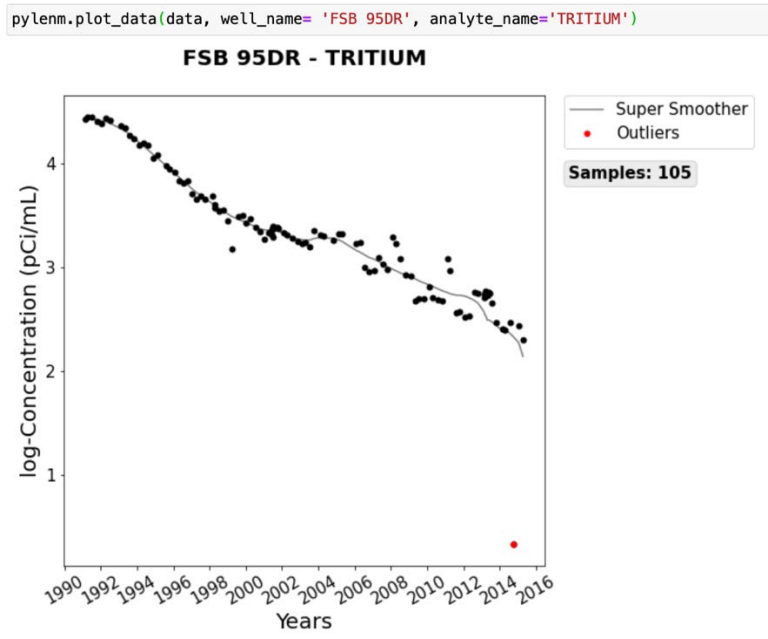

Figure 7. *plot_data* function for tritium at well FSB 95DR.

Having a quick way to predict when log-concentration values will reach the analyte's regulatory limit, or maximum concentration limit (MCL) was a crucial aspect of project. The *plot_MCL* function extends the *plot_data* function by projecting the regression's line of best fit into the future. Each analyte has its own MCL value that is manually specified by the user, and the function finds the intersection point between the constant MCL and the line of best fit. The 95% confidence interval is also calculated which returns a date range rather than just one fixed date. Figure 8 below shows a good example of this function.

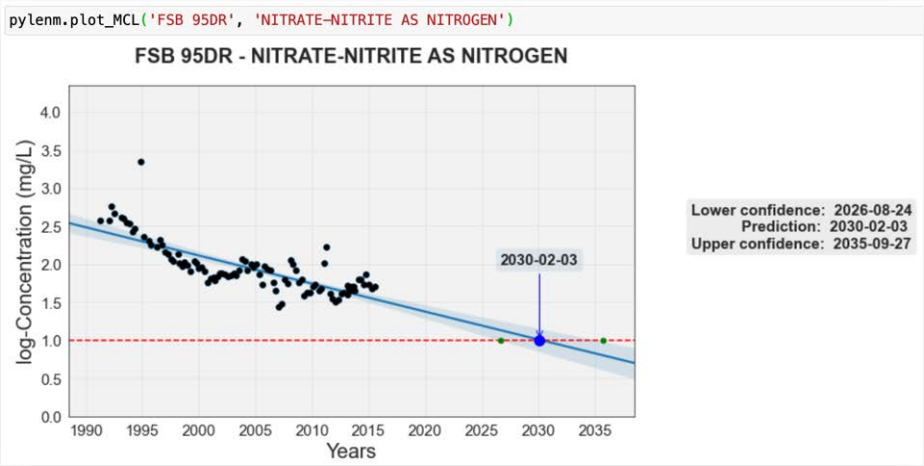

Figure 8. *plot_MCL* function for nitrate at well FSB 95DR.

Another useful tool that was created for the package is a set of functions for correlation analysis. For all of the correlation functions, *plot_corr_by_well*, *plot_corr_by_date*, and *plot_coor_by_year*, the same appearance is shared; it combines a traditional pairplot with a correlation matrix. A pairplot shows the pairwise relationship between two variables and a correlation matrix shows the correlation coefficients, values between 0 and 1, for these two variables. These two plots individually have duplicate information along the diagonal, so combining the two plots into one made sense. The correlation matrix and pairplot are located above and below the diagonal (top left to bottom right) respectively. Figure 9 shows the *plot_coor_by_year* for 2012.

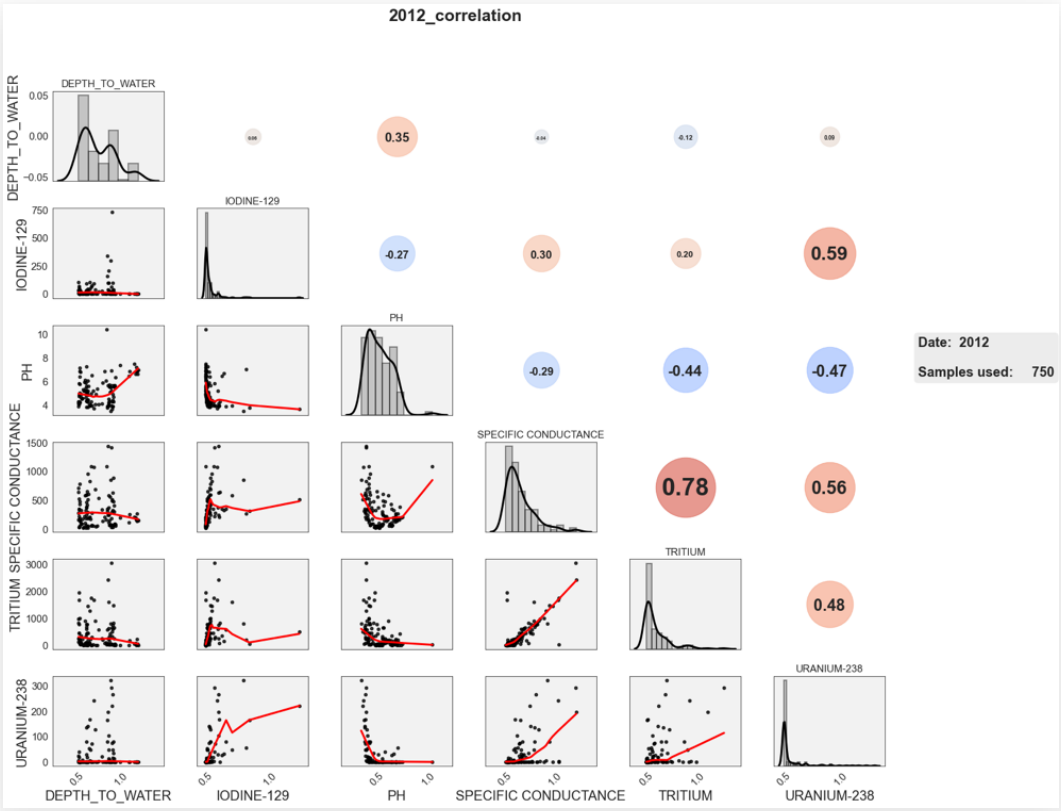

Figure 9. *plot_corr_by_year* function for the year 2012 looking at the top 6 analytes.

An additional way to gain insight on the relationship between different analyte concentrations is through a biplot, which is a combination of a principal component analysis (PCA) plot and a loading plot. For this project, we chose to represent all of the data with just two principal components so that it could easily be viewed on a graph. This plot clusters samples together based on similarity. The loading plot depicted by the red arrows in Figure 10 represents how strongly each characteristic influences a principal component.

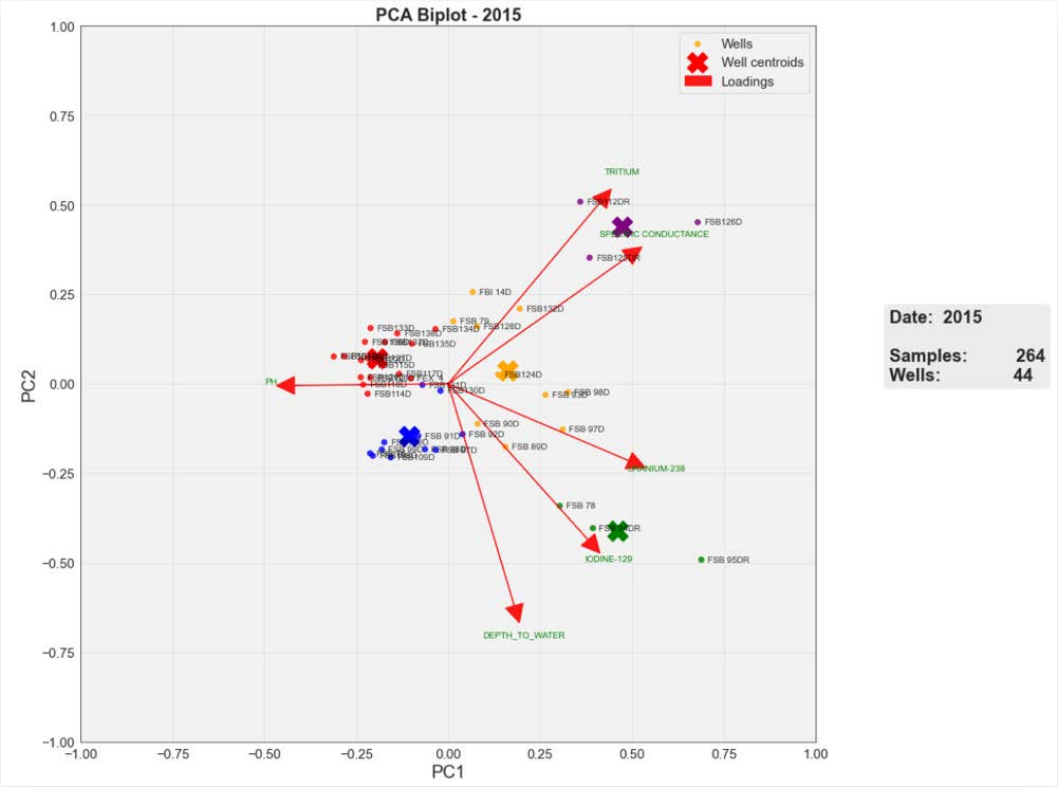

Figure 10. *plot_PCA_by_year* function for the year 2015 looking at the top 6 analytes.

Since there are many wells at the F-Area site, it can be hard to keep track of the concentration values for each individual well. Using a clustering algorithm such as K-means clustering can help group wells based on similar behavior. Once clusters of wells are formed using the *cluster_data* method, it helps to visualize the wells on a map to gain a better understanding of the cluster distributions. This information can be seen in Figure 11 below.

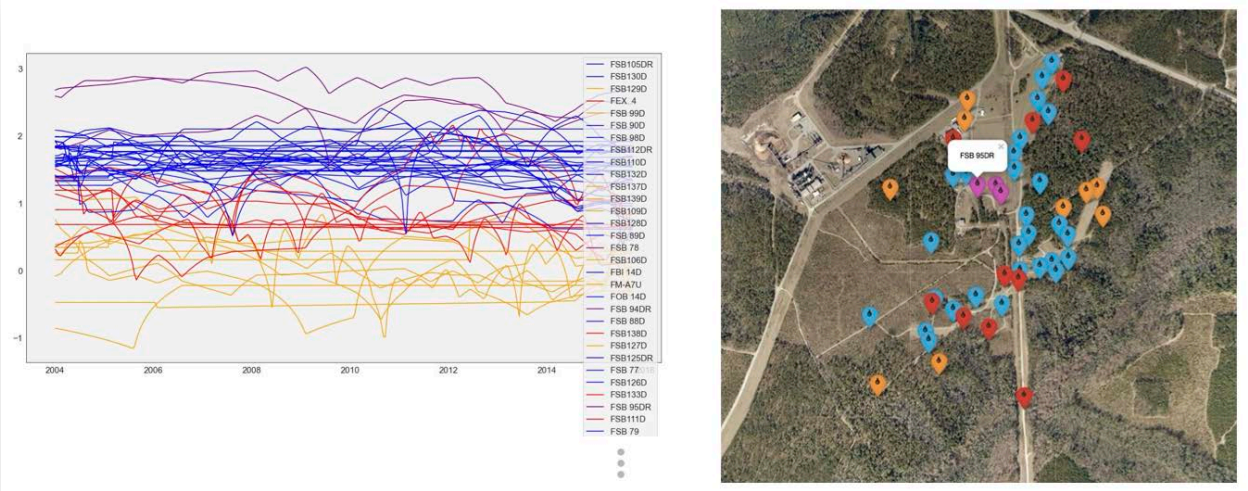

Figure 11. (Left) *cluster_data* function for iodine concentrations filtered at 'D' wells with 4 clusters. (Right) *plot_coordinates_to_map* function with the data from *cluster_data.*

3. RESULTS AND ANALYSIS

The functions that were created as a part of the package proved to be very effective. The project's effort thus far is showing promising results.

Using the correlation functions, it quickly became evident that specific conductance and tritium concentrations are closely related as can be seen in Figure 12. This information is extremely beneficial since specific conductance can be measured in-situ, while traditionally tritium concentrations need to be measured from collected water samples from each well. Between collecting the water samples and running the experiments in the laboratory, this process can take days or even weeks to gain tangible results. Identifying correlations between analytes, in this case, tritium and specific conductance, implies that in-situ sensors can predict contaminant concentrations with a high degree of confidence much closer to real-time and at a lower cost.

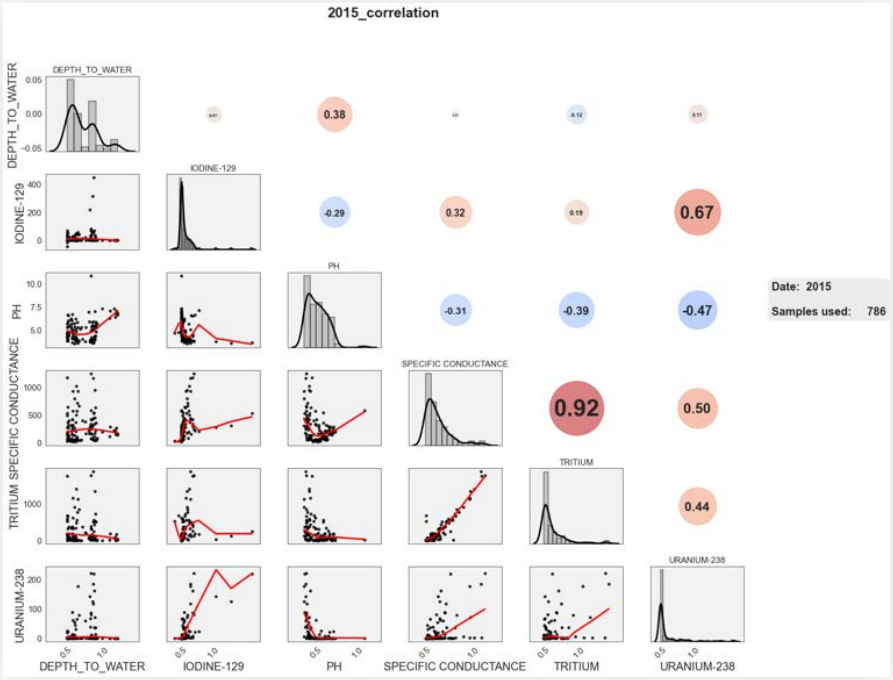

Figure 12. *plot_corr_by_year* function for the year 2015 looking at the top 6 analytes.

4. CONCLUSION

The pyLEnM package is meant to serve as a tool for accelerating and simplifying the analysis of soil and groundwater data. During the internship, we were able to build the foundation for the package by creating useful functions for analysis. The package contains tools to summarize the dataset, remove outliers, interpolate temporally, and many interesting visualization functions. An overview of the functions available in pyLEnM is shown in Figure 3. At the moment pyLEnM works well with the F-Area dataset but the hope is to expand its capability to work on other similar-structured datasets. In the near future, we would like to see the package grow in complexity, such as incorporating more machine learning-based algorithms to further expose meaningful insights in the data.

5. REFERENCES

[1] Denham, Miles E., et al. “A New Paradigm for Long Term Monitoring at the F-Area Seepage Basins, Savannah River Site.” 2019, DOI: 10.2172/1504623.

[2] Schmidt, Franziska, et al. “In Situ Monitoring of Groundwater Contamination Using the Kalman Filter.” *Environmental Science & Technology*, vol. 52, no. 13, 2018, pp. 7418–7425., DOI: 10.1021/acs.est.8b00017.

[3] Wainwright, Haruko, et al. “Objective.” *ALTEMIS*, 19 Aug. 2020, altemis.lbl.gov/about/.

APPENDIX A

Package Documentation

Appendix A serves to describe the functions and parameters in the pyLEnM package.

Color Key:

- Red = required parameters
- Blue = optional parameters (already has defaults)

1. **simplify_data**(data=None, inplace=False, columns=None, save_csv=False, file_name='data_simplified', save_dir='data/')
   - Description:
     - Removes all columns except 'COLLECTION_DATE', 'STATION_ID', 'ANALYTE_NAME', 'RESULT', and 'RESULT_UNITS'.
     - If the user specifies additional columns in addition to the ones listed above, those columns will be kept.
     - The function returns a dataframe and has an optional parameter to be able to save the dataframe to a csv file.
   - Parameters:
     - data (dataframe): data to simplify
     - inplace (bool): save data to current working dataset
     - columns (list of strings): list of any additional columns on top of ['COLLECTION_DATE', 'STATION_ID', 'ANALYTE_NAME', 'RESULT', and 'RESULT_UNITS'] to be kept in the dataframe.
     - save_csv (bool): flag to determine whether or not to save the dataframe to a csv file.
     - file_name (string): name of the csv file you want to save
     - save_dir (string): name of the directory you want to save the csv file to

2. **get_MCL(**analyte_name**)**
   - Description:
     - Returns the Maximum Concentration Limit value for the specified analyte.
     - Example: 'TRITIUM' returns 1.3
   - Parameters:
     - analyte_name (string): name of the analyte to be processed

3. **get_unit(**analyte_name**)**
   - Description:
     - Returns the unit of the analyte you specify.

    - Example: 'DEPTH_TO_WATER' returns 'ft'
- Parameters:
    - analyte_name (string): name of the analyte to be processed

4. **filter_wells(**units**)**
    - Description:
        - Returns a list of the well names filtered by the unit(s) specified.
    - Parameters:
        - units (list of strings): Letter of the well to be filtered (e.g. [‘A’] or [‘A’, ‘D’])

5. **remove_outliers(**data, z_threshold=4**)**
    - Description:
        - Removes outliers from a dataframe based on the z_scores and returns the new dataframe.
    - Parameters:
        - data (dataframe): data for the outliers to removed from
        - z_threshold (float): z_score threshold to eliminate.

6. **get_analyte_details(**analyte_name, save_dir='analyte_details'**)**
    - Description:
        - Returns a csv file saved to save_dir with details pertaining to the specified analyte.
        - Details include the well names, the date ranges and the number of unique samples.
    - Parameters:
        - analyte_name (string): name of the analyte to be processed
        - save_dir (string): name of the directory you want to save the csv file to

7. **get_data_summary(**analytes=None, sort_by='date', ascending=False**)**
    - Description:
        - Returns a dataframe with a summary of the data for certain analytes.
        - Summary includes the date ranges and the number of unique samples and other statistics for the analyte results.
    - Parameters:
        - analytes (list of strings): list of analyte names to be processed. If left empty, a list of all the analytes in the data will be used.
        - sort_by (string): {‘date’, ‘samples’, ‘wells’} sorts the data by either the dates by entering: ‘date’, the samples by entering: ‘samples’, or by unique well locations by entering ‘wells’.
        - ascending (bool): flag to sort in ascending order.

8. **query_data**(well_name, analyte_name)
    - Description:
        - Filters data by passing the data and specifying the well_name and analyte_name
    - Parameters:
        - well_name (string): name of the well to be processed
        - analyte_name (string): name of the analyte to be processed

9. **plot_data**(well_name, analyte_name, log_transform=True, alpha=0, year_interval=2, plot_inline=True, save_dir='plot_data')
    - Description:
        - Plot concentrations over time of a specified well and analyte with a smoothed curve on interpolated data points.
    - Parameters:
        - well_name (string): name of the well to be processed
        - analyte_name (string): name of the analyte to be processed
        - log_transform (bool): choose whether or not the data should be transformed to log base 10 values
        - alpha (int): value between 0 and 10 for line smoothing
        - year_interval (int): plot by how many years to appear in the axis e.g.(1 = every year, 5 = every 5 years, ...)
        - plot_inline (bool): choose whether or not to show plot inline
        - save_dir (string): name of the directory you want to save the plot to

10. **plot_all_data**(log_transform=True, alpha=0, year_interval=2, plot_inline=True, save_dir='plot_data')
    - Description:
        - Plot concentrations over time for every well and analyte with a smoothed curve on interpolated data points.
    - Parameters:
        - log_transform (bool): choose whether or not the data should be transformed to log base 10 values
        - alpha (int): value between 0 and 10 for line smoothing
        - year_interval (int): plot by how many years to appear in the axis e.g.(1 = every year, 5 = every 5 years, ...)
        - plot_inline (bool): choose whether or not to show plot inline
        - save_dir (string): name of the directory you want to save the plot to

11. **plot_correlation_heatmap**(well_name, show_symmetry=True, color=True, save_dir='plot_correlation_heatmap')

- Description:
  - Plots a heatmap of the correlations of the important analytes over time for a specified well.
- Parameters:
  - well_name (string): name of the well to be processed
  - show_symmetry (bool): choose whether or not the heatmap should show the same information twice over the diagonal
  - color (bool): choose whether or not the plot should be in color or in greyscale
  - save_dir (string): name of the directory you want to save the plot to

12. **plot_all_correlation_heatmap**(show_symmetry=True, color=True, save_dir='plot_correlation_heatmap')
    - Description:
      - Plots a heatmap of the correlations of the important analytes over time for each well in the dataset.
    - Parameters:
      - show_symmetry (bool): choose whether or not the heatmap should show the same information twice over the diagonal
      - color (bool): choose whether or not the plot should be in color or in greyscale
      - save_dir (string): name of the directory you want to save the plot to

13. **interpolate_wells_by_analyte**(analyte, frequency='2W', rm_outliers=True, z_threshold=3)
    - Description:
      - Resamples analyte data based on the frequency specified and interpolates the values in between. NaN values are replaced with the average value per well.
    - Parameters:
      - analyte (string): analyte name for interpolation of all present wells.
      - frequency (string): {'D', 'W', 'M', 'Y'} frequency to interpolate. See https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html for valid frequency inputs. (e.g. 'W' = every week, 'D '= every day, '2W' = every 2 weeks)
      - rm_outliers (bool): flag to remove outliers in the data
      - z_threshold (int): z_score threshold to eliminate outliers

14. **interpolate_well_data**(well_name, analytes, frequency='2W')
    - Description:

    - Resamples the data based on the frequency specified and interpolates the values of the analytes.
- Parameters:
    - well_name (string): name of the well to be processed
    - analytes (list of strings): list of analyte names to use
    - frequency (string): {'D', 'W', 'M', 'Y'} frequency to interpolate. See https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html for valid frequency inputs. (e.g. 'W' = every week, 'D '= every day, '2W' = every 2 weeks)

15. **plot_corr_by_well**(well_name, remove_outliers=True, z_threshold=4, interpolate=False, frequency='2W', save_dir='plot_correlation')
- Description:
    - Plots the correlations with the physical plots as well as the correlations of the important analytes over time for a specified well.
- Parameters:
    - well_name (string): name of the well to be processed
    - remove_outliers (bool): choose whether or not to remove the outliers.
    - z_threshold (float): z_score threshold to eliminate outliers
    - interpolate (bool): choose whether or not to interpolate the data
    - frequency (string): {'D', 'W', 'M', 'Y'} frequency to interpolate. See https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html for valid frequency inputs. (e.g. 'W' = every week, 'D '= every day, '2W' = every 2 weeks)
    - save_dir (string): name of the directory you want to save the plot to

16. **plot_all_corr_by_well**(remove_outliers=True, z_threshold=4, interpolate=False, frequency='2W', save_dir='plot_correlation')
- Description:
    - Plots the correlations with the physical plots as well as the important analytes over time for each well in the dataset.
- Parameters:
    - remove_outliers (bool): choose whether or to remove the outliers.
    - z_threshold (float): z_score threshold to eliminate outliers
    - interpolate (bool): choose whether or to interpolate the data
    - frequency (string): {'D', 'W', 'M', 'Y'} frequency to interpolate. See https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html for valid frequency inputs. (e.g. 'W' = every week, 'D '= every day, '2W' = every 2 weeks)
    - save_dir (string): name of the directory you want to save the plot to

17. **plot_corr_by_date**(date, min_samples=48, save_dir='plot_corr_by_date')
    - Description:
        - Plots the correlations with the physical plots as well as the correlations of the important analytes for ALL the wells on a specified date.
    - Parameters:
        - date (string): date to be analyzed
        - min_samples (int): minimum number of samples the result should contain in order to execute.
        - save_dir (string): name of the directory you want to save the plot to

18. **plot_corr_by_year**(year, min_samples=500, save_dir='plot_corr_by_year')
    - Description:
        - Plots the correlations with the physical plots as well as the correlations of the important analytes for ALL the wells in specified year.
    - Parameters:
        - year (int): year to be analyzed
        - min_samples (int): minimum number of samples the result should contain in order to execute.
        - save_dir (string): name of the directory you want to save the plot to
19. **plot_MCL**(well_name, analyte_name, year_interval=5, save_dir='plot_MCL')
    - Description:
        - Plots the linear regression line of data given the analyte_name and well_name. The plot includes the prediction where the line of best fit intersects with the Maximum Concentration Limit (MCL).
    - Parameters:
        - well_name (string): name of the well to be processed
        - analyte_name (string): name of the analyte to be processed
        - year_interval (int): plot by how many years to appear in the axis e.g.(1 = every year, 5 = every 5 years, ...)
        - save_dir (string): name of the directory you want to save the plot to

20. **plot_PCA_by_date**(date, n_clusters=4, min_samples=48,
    filter=False, filter_well_by=['D'], return_clusters=False,
    show_labels=True, save_dir='plot_corr_by_date')
    - Description:
        - Gernates a PCA biplot (PCA score plot + loading plot) of the data given a date in the dataset. Only uses the 6 important analytes. The data is also clustered into n_clusters.
    - Parameters:
        - date (string): date to be analyzed

- n_clusters (int): number of clusters to split the data into.
- filter (bool): Flag to indicate well filtering.
- filter_well_by (list of strings): Letter of the well to be filtered (e.g. [‘A’] or [‘A’, ‘D’])
- return_clusters (bool): Flag to return the cluster data to be used for spatial plotting.
- min_samples (int): minimum number of samples the result should contain in order to execute.
- show_labels (bool): choose whether or not to show the name of the wells.
- save_dir (string): name of the directory you want to save the plot to

21. **plot_PCA_by_year**(year, n_clusters=4, min_samples=48, filter=False, filter_well_by=['D'], return_clusters=False, show_labels=True, save_dir=‘plot_corr_by_year’)
    - Description:
        - Gernates a PCA biplot (PCA score plot + loading plot) of the data given a year in the dataset. Only uses the 6 important analytes. The data is also clustered into n_clusters.
    - Parameters:
        - year (int): date to be analyzed
        - n_clusters (int): number of clusters to split the data into.
        - filter (bool): Flag to indicate well filtering.
        - filter_well_by (list of strings): Letter of the well to be filtered (e.g. [‘A’] or [‘A’, ‘D’])
        - return_clusters (bool): Flag to return the cluster data to be used for spatial plotting.
        - min_samples (int): minimum number of samples the result should contain in order to execute.
        - show_labels (bool): choose whether or not to show the name of the wells.
        - save_dir (string): name of the directory you want to save the plot to

22. **plot_PCA_by_well**(well_name, interpolate=False, frequency='2W', min_samples=48, show_labels=True, save_dir=‘plot_PCA_by_well’)
    - Description:
        - Gernates a PCA biplot (PCA score plot + loading plot) of the data given a well_name in the dataset. Only uses the 6 important analytes.
    - Parameters:
        - well_name (string): name of the well to be processed
        - interpolate (bool): choose whether or to interpolate the data
        - frequency (string): {‘D’, ‘W’, ‘M’, ‘Y’} frequency to interpolate. See https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html

for valid frequency inputs. (e.g. ‘W’ = every week, ‘D ’= every day, ‘2W’ = every 2 weeks)
    - min_samples (int): minimum number of samples the result should contain in order to execute.
    - show_labels (bool): choose whether or not to show the name of the wells.
    - save_dir (string): name of the directory you want to save the plot to

23. **plot_coordinates_to_map**(gps_data, center=[33.271459, -81.675873], zoom=14)
    - Description:
        - Plots the well locations on an interactive map given coordinates.
    - Parameters:
        - gps_data (dataframe): Data frame with the following column names: station_id, latitude, longitude, color. If the color column is not passed, the default color will be blue.
        - center (list with 2 floats): latitude and longitude coordinates to center the map view.
        - zoom (int): value to determine the initial scale of the map