pynssp Documentation

Release 0.1.0

Gbedegnon Roseric Azondekon

May 18, 2023

Overview:

Chapter 1

About pynssp

1.1 Overview

pynssp is a Python package for the National Syndromic Surveillance Program (NSSP) and its Community of Practice. A collection of classes and methods to advance the practice of Syndromic Surveillance. It serves as a Python alternative to the Rnssp R package.

1.2 Installation

You can install the development version of pynssp from Github:

```
$ pip install git+https://github.com/CDCgov/pynssp.git
```

1.3 Usage

```
>>> import pandas as pd
>>> from pynssp.utils import *
>>> from datetime import date, timedelta


## Creating a user profile (token)
>>> myProfile = create_token_profile()
```

(continues on next page)

(continued from previous page)

```
## Creating a user profile (username and password)
>>> myProfile = create_profile()

## JSON URL from NSSP-ESSENCE API
>>> url = "https://essence.syndromicsurveillance.org/nssp_essence/api/
 ↪alerts/regionSyndromeAlerts?end_date=31Jan2021&start_date=29Jan2021"

## Update Start and End dates in NSSP-ESSENCE API URL
>>> startDate = date.today() - timedelta(days=30)
>>> endDate = date.today()

>>> url = change_dates(url, start_date = startDate, end_date = endDate)

## Pull Time Series Data from NSSP-ESSENCE
>>> api_data = get_api_data(url, profile=myProfile)

## Inspect data object structure
>>> api_data.columns

## Extract table of interest
>>> api_data = pd.json_normalize(api_data["regionSyndromeAlerts"][0])

## Get a glimpse of the pulled dataset
>>> api_data.head()
```

1.4 Contributing to this project

Should you want to contribute to this project, submit a push request to this Github repository and consider submitting a request to be added as a developer to gazondekon@cdc.gov.

1.5 Getting Help

If you encounter a clear bug, please consider emailing the author at gazondekon@cdc.gov and/or file an issue with a minimal reproducible example.

1.6 Public Domain Standard Notice

This repository constitutes a work of the United States Government and is not subject to domestic copyright protection under 17 USC ? 105. This repository is in the public domain within the United States, and copyright and related rights in the work worldwide are waived through the CC0 1.0 Universal public domain dedication. All contributions to this repository will be released under the CC0 dedication. By submitting a pull request you are agreeing to comply with this waiver of copyright interest.

1.7 License Standard Notice

The repository utilizes code licensed under the terms of the Apache Software License and therefore is licensed under ASL v2 or later.

This source code in this repository is free: you can redistribute it and/or modify it under the terms of the Apache Software License version 2, or (at your option) any later version.

This source code in this repository is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the Apache Software License for more details.

You should have received a copy of the Apache Software License along with this program. If not, see http://www.apache.org/licenses/LICENSE-2.0.html

The source code forked from other open source projects will inherit its license.

1.8 Privacy Standard Notice

This repository contains only non-sensitive, publicly available data and information. All material and community participation is covered by the Disclaimer and Code of Conduct. For more information about CDC's privacy policy, please visit http://www.cdc.gov/other/privacy.html.

1.9 Contributing Standard Notice

Anyone is encouraged to contribute to the repository by forking and submitting a pull request. (If you are new to GitHub, you might start with a basic tutorial.) By contributing to this project, you grant a world-wide, royalty-free, perpetual, irrevocable, non-exclusive, transferable license to all users under the terms of the Apache Software License v2 or later.

All comments, messages, pull requests, and other submissions received through CDC including this GitHub page may be subject to applicable federal law, including but not limited to the Federal Records Act, and may be archived. Learn more at http://www.cdc.gov/other/privacy.html.

1.10 Records Management Standard Notice

This repository is not a source of government records, but is a copy to increase collaboration and collaborative potential. All government records will be published through the CDC web site.

Chapter 2

Installing pynssp

2.1 Development version

You may install the development version with pip by running this command in your terminal.

```
$ pip install git+https://github.com/cdcgov/pynssp.git
```

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

Alternatively, the sources for pynssp can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/cdcgov/pynssp
```

Or download the tarball:

```
$ curl -OJL https://github.com/cdcgov/pynssp/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

Chapter 3

Getting started

3.1 Introduction

The goal of the `pynssp` package is to facilitate the access to the Electronic Surveillance System for the Early Notification of Community-based Epidemics (ESSENCE) via a secure and simplified interface. In addition, `pynssp` provides methods and functions that streamline the data pull by abstracting the complexity of the R codes from the users.

In this vignette, we explained how to create an NSSP user profile, and provide various examples to how to use it to pull data from ESSENCE using the following ESSENCE APIs:

- Time series data table
- Time series png image
- Table builder results
- Data details (line level)
- Summary stats
- Alert list detection table
- Time series data table with stratified, historical alerts (from ESSENCE2)

3.2 Creating an NSSP user profile

We start by loading the `pandas` and the `pynssp` packages.

```
>>> import pandas as pd
>>> from pynssp.utils import *
```

The next step is to create an NSSP user profile by creating an object of the class `Credentials`. Here, we use the `create_profile()` function to create a user profile

```
>>> myProfile = create_profile()

# Save profile object to file for future use
>>> myProfile.pickle()
```

The above code needs to be executed only once. Upon execution, it prompts the user to provide his username and password.

The created myProfile object comes with the `.get_api_response()`, `.get_api_data()`, `.get_api_graph()` and `.get_api_tsgraph()` methods with various parameters to pull ESSENCE data. Alternatively, the `get_api_response()`, `get_api_data()`, `get_api_graph()` functions serve as wrappers to their respective methods. The `get_essence_data()` function may be used for NSSP-ESSENCE API URLs.

In the following sections, we show how to pull data from ESSENCE using the seven APIs listed above.

3.3 Time Series Data Table

```
>>> url = "https://essence.syndromicsurveillance.org/nssp_essence/api/
↪timeSeries?endDate=9Feb2021&medicalGrouping=injury&
↪percentParam=noPercent&geographySystem=hospitaldhhsregion&datasource=va_
↪hospdreg&detector=probrepswitch&startDate=11Nov2020&
↪timeResolution=daily&medicalGroupingSystem=essencesyndromes&userId=455&
↪aqtTarget=TimeSeries"

# Pull time series data
>>> api_data_ts = get_api_data(url, profile=myProfile) # or api_data_ts =␣
↪myProfile.get_api_data(url)

>>> api_data_ts.columns

# Extracting embedded dataframe
```

(continues on next page)

(continued from previous page)

```
>>> api_data_ts = pd.json_normalize(api_data_ts["timeSeriesData"][0])

# Preview data
>>> api_data_ts.head()
```

Alternatively, the example below with the `get_essence_data()` function achieves the same outcome directly extracting the embedded dataframe when needed

```
>>> api_data_ts = get_essence_data(url, profile=myProfile)

# Preview data
>>> api_data_ts.head()
```

3.4 Time Series Graph from ESSENCE

The example below shows how to retrieve the Time Series Graph from ESSENCE and insert it in a Jupyter notebook.

```
>>> url = "https://essence.syndromicsurveillance.org/nssp_essence/api/
 ↪timeSeries/graph?endDate=9Feb2021&medicalGrouping=injury&
 ↪percentParam=noPercent&geographySystem=hospitaldhhsregion&datasource=va_
 ↪hospdreg&detector=probrepswitch&startDate=11Nov2020&
 ↪timeResolution=daily&medicalGroupingSystem=essencesyndromes&userId=455&
 ↪aqtTarget=TimeSeries&graphTitle=National%20-%20Injury%20Syndrome%20Daily
 ↪%20Counts&xAxisLabel=Date&yAxisLabel=Count"

# Data pull from ESSENCE
>>> api_data_graph = get_api_graph(url, profile=myProfile)

# Check the type of api_data_graph
>>> type(api_data_graph)

# Print image file location
>>> print(api_data_graph)

# Insert it into a Jupyter notebook
>>> api_data_graph.plot()
```

From the example above, the variable `api_data_graph` is an `APIGraph` object. In an interactive mode from the Python console, the `.show()` method can be called on the `APIGraph` object to preview the image it contains.

```
>>> api_data_graph.show()
```

For an exaustive list of the methods that may be called on an `APIGraph` object, please check the `pynssp` documentation.

3.5 Table Builder Results

The CSV option of the Table Builder Results API pulls in data in the tabular format seen on ESSENCE. The JSON option on the other hand, pulls in the data in a long, pivoted format. In the following subsections, we demonstrate how to pull the Table Builder results data with both options.

3.5.1 CSV option

```
>>> url = "https://essence.syndromicsurveillance.org/nssp_essence/api/
 ↪tableBuilder/csv?endDate=31Dec2020&ccddCategory=cdc%20opioid%20overdose
 ↪%20v3&percentParam=noPercent&geographySystem=hospitaldhhsregion&
 ↪datasource=va_hospdreg&detector=nodetectordetector&startDate=1Oct2020&
 ↪ageNCHS=11-14&ageNCHS=15-24&ageNCHS=25-34&ageNCHS=35-44&ageNCHS=45-54&
 ↪ageNCHS=55-64&ageNCHS=65-74&ageNCHS=75-84&ageNCHS=85-1000&
 ↪ageNCHS=unknown&timeResolution=monthly&hasBeenE=1&
 ↪medicalGroupingSystem=essencesyndromes&userId=455&
 ↪aqtTarget=TableBuilder&rowFields=timeResolution&
 ↪rowFields=geographyhospitaldhhsregion&columnField=ageNCHS"

# Data Pull from ESSENCE
>>> api_data_tb_csv = get_api_data(url, fromCSV=True, profile=myProfile)

# Preview data
>>> api_data_tb_csv.head()
```

3.5.2 JSON option

While the `get_api_data()` function can equally be used to pull from ESSENCE API URL retruning JSON objects, the `get_essence_data()` function has the advantage of not requiring extra JSON parsing as described in the second section of this tutorial.

```
>>> url = "https://essence2.syndromicsurveillance.org/nssp_essence/api/
 ↪tableBuilder?endDate=31Dec2020&ccddCategory=cdc%20opioid%20overdose
```

(continues on next page)

(continued from previous page)

```
↪%20v3&percentParam=noPercent&geographySystem=hospitaldhhsregion&
↪datasource=va_hospdreg&detector=nodetectordetector&startDate=1Oct2020&
↪ageNCHS=11-14&ageNCHS=15-24&ageNCHS=25-34&ageNCHS=35-44&ageNCHS=45-54&
↪ageNCHS=55-64&ageNCHS=65-74&ageNCHS=75-84&ageNCHS=85-1000&
↪ageNCHS=unknown&timeResolution=monthly&hasBeenE=1&
↪medicalGroupingSystem=essencesyndromes&userId=2362&
↪aqtTarget=TableBuilder&rowFields=timeResolution&
↪rowFields=geographyhospitaldhhsregion&columnField=ageNCHS"

# Data Pull from ESSENCE
>>> api_data_tb_json = get_essence_data(url, profile=myProfile)

# Preview data
>>> api_data_tb_json.head()
```

3.6 Data Details (line level)

Similarly to the Table builder Results API, the Data Details (line level) provides CSV and JSON data outputs.

3.6.1 CSV option

```
>>> url = "https://essence.syndromicsurveillance.org/nssp_essence/api/
↪dataDetails/csv?medicalGrouping=injury&geography=region%20i&
↪percentParam=noPercent&geographySystem=hospitaldhhsregion&datasource=va_
↪hospdreg&detector=probrepswitch&timeResolution=daily&
↪medicalGroupingSystem=essencesyndromes&userId=455&aqtTarget=TimeSeries&
↪startDate=31Jan2021&endDate=31Jan2021"

# Data Pull from ESSENCE
>>> api_data_dd_csv = get_api_data(url, fromCSV=True, profile=myProfile)

# Preview data
>>> api_data_dd_csv.head()
```

3.6.2 JSON option

```
>>> url = "https://essence.syndromicsurveillance.org/nssp_essence/api/
↪dataDetails?endDate=31Jan2021&medicalGrouping=injury&
↪percentParam=noPercent&geographySystem=hospitaldhhsregion&datasource=va_
↪hospdreg&detector=probrepswitch&startDate=31Jan2021&
↪timeResolution=daily&medicalGroupingSystem=essencesyndromes&userId=455&
↪aqtTarget=DataDetails"

# Data Pull from ESSENCE
>>> api_data_dd_json = get_essence_data(url, profile=myProfile)

# Preview data
>>> api_data_dd_json.head()
```

3.7 Summary Stats

```
>>> url = "https://essence.syndromicsurveillance.org/nssp_essence/api/
↪summaryData?endDate=31Jan2021&medicalGrouping=injury&geography=region
↪%20i&percentParam=noPercent&geographySystem=hospitaldhhsregion&
↪datasource=va_hosp&detector=probrepswitch&startDate=29Jan2021&
↪timeResolution=daily&medicalGroupingSystem=essencesyndromes&userId=455&
↪aqtTarget=TimeSeries"

# Data Pull from ESSENCE
>>> api_data_ss = get_essence_data(url, profile=myProfile)

# Preview data
>>> api_data_ss.head()
```

3.8 Alert List Detection Table

Since the Alert List API provides programmatic access to the Alert List table on the ESSENCE user interface by patient region or by hospital regions, we provide two use cases of data pull in the following subsections:

3.8.1 Alert List Detection Table by Patient Region

```
>>> url = "https://essence.syndromicsurveillance.org/nssp_essence/api/
↪alerts/regionSyndromeAlerts?end_date=31Jan2021&start_date=29Jan2021"

# Data Pull from ESSENCE
>>> api_data_alr = get_essence_data(url, profile=myProfile)

# Preview data
>>> api_data_alr.head()
```

3.8.2 Alert List Detection Table by Hospital Region

```
>>> url = "https://essence.syndromicsurveillance.org/nssp_essence/api/
↪alerts/hospitalSyndromeAlerts?end_date=31Jan2021&start_date=29Jan2021"

# Data Pull from ESSENCE
>>> api_data_alh = get_api_data(url, profile=myProfile)

# Preview data
>>> api_data_alh.head()
```

3.9 Time series data table with stratified, historical alerts (from ESSENCE2)

This functionality as of February 10, 2023 is available from ESSENCE2. Therefore, if your ESSENCE 2 credentials are different from the one you define for ESSENCE above, you will have to recreate another profile object for ESSENCE 2 and use it to run the code below. In this example, it is assumed that the same user profile has been used for both ESSENCE and ESSENCE 2.

```
>>> url = "https://essence2.syndromicsurveillance.org/nssp_essence/api/
↪timeSeries?endDate=9Feb2021&ccddCategory=cdc%20pneumonia%20ccdd%20v1&
↪ccddCategory=cdc%20coronavirus-dd%20v1&ccddCategory=cli%20cc%20with
↪%20cli%20dd%20and%20coronavirus%20dd%20v2&percentParam=ccddCategory&
↪geographySystem=hospitaldhhsregion&datasource=va_hospdreg&
↪detector=probrepswitch&startDate=11Nov2020&timeResolution=daily&
↪hasBeenE=1&medicalGroupingSystem=essencesyndromes&userId=2362&
↪aqtTarget=TimeSeries&stratVal=ccddCategory&multiStratVal=geography&
↪graphOnly=true&numSeries=3&graphOptions=multipleSmall&
```

(continues on next page)

(continued from previous page)

```
↪seriesPerYear=false&nonZeroComposite=false&removeZeroSeries=true&
↪startMonth=January&stratVal=ccddCategory&multiStratVal=geography&
↪graphOnly=true&numSeries=3&graphOptions=multipleSmall&
↪seriesPerYear=false&startMonth=January&nonZeroComposite=false"

# Data Pull from ESSENCE
>>> api_data_tssh = get_essence_data(url, profile=myProfile)

# Preview data
>>> api_data_tssh.head()
```

Chapter 4

Anomaly Detection

4.1 Introduction

In this tutorial, we describe how to perform anomaly detection and trend classification analysis using time series data from NSSP-ESSENCE. This tutorial uses time series data from NSSP-ESSENCE data source for the CLI CC with CLI DD and Coronavirus DD v2 definition, limiting to ED visits (Has been Emergency = "Yes").

We start this tutorial by loading the `pynssp` package.

```
>>> from pynssp import *
```

Then, we load additional packages for visualization purposes:

```
>>> import seaborn as sns
>>> import matplotlib.pyplot as plt
>>> import matplotlib.dates as mdates
```

Next, we create an NSSP user profile by creating an object of the class Credentials.

```
>>> myProfile = create_profile() # Creating an ESSENCE user profile

# save profile object to file for future use
>>> myProfile.pickle()
```

4.2 Data Pull from NSSP-ESSENCE

With the NSSP `myProfile` object, we authenticate to NSSP-ESSENCE and pull in the data using the Time series data table API.

```
>>> url = "https://essence2.syndromicsurveillance.org/nssp_essence/api/
 ↪timeSeries?endDate=20Nov20&ccddCategory=cli%20cc%20with%20cli%20dd%20and
 ↪%20coronavirus%20dd%20v2&percentParam=ccddCategory&
 ↪geographySystem=hospitaldhhsregion&datasource=va_hospdreg&
 ↪detector=probrepswitch&startDate=22Aug20&timeResolution=daily&
 ↪hasBeenE=1&medicalGroupingSystem=essencesyndromes&userId=2362&
 ↪aqtTarget=TimeSeries&stratVal=&multiStratVal=geography&graphOnly=true&
 ↪numSeries=0&graphOptions=multipleSmall&seriesPerYear=false&
 ↪nonZeroComposite=false&removeZeroSeries=true&startMonth=January&
 ↪stratVal=&multiStratVal=geography&graphOnly=true&numSeries=0&
 ↪graphOptions=multipleSmall&seriesPerYear=false&startMonth=January&
 ↪nonZeroComposite=false"

# Data Pull from NSSP-ESSENCE
>>> api_data = get_essence_data(url, profile = myProfile)

# Inspect pulled data frame
>>> api_data.info()
```

Before applying an anomaly detection function, let's first group the data by HHS regions:

```
>>> df_hhs = api_data.groupby("hospitaldhhsregion_display")
```

4.3 Exponentially Weighted Moving Average (EWMA)

The Exponentially Weighted Moving Average (EWMA) compares a weighted average of the most recent visit counts to a baseline expectation. For the weighted average to be tested, an exponential weighting gives the most influence to the most recent observations. This algorithm is appropriate for daily counts that do not have the characteristic features modeled in the regression algorithm. It is more applicable for Emergency Department data from certain hospital groups and for time series with small counts (daily average below 10) because of the limited case definition or chosen geographic region. The EWMA detection algorithm can be performed with `alert_ewma()` function (run `help(alert_ewma)` in your Python console or `alert_ewma?` in Jupyter Notebook or JupyterLab for more).

```
>>> df_ewma = alert_ewma(df_hhs, t = "date", y = "dataCount")
```

Let's subset the dataframe to visualize the time series with the anomalies for Region 4:

```
>>> df_ewma_region = df_ewma[df_ewma["hospitaldhhsregion_display"] ==
↪"Region 4"]
```

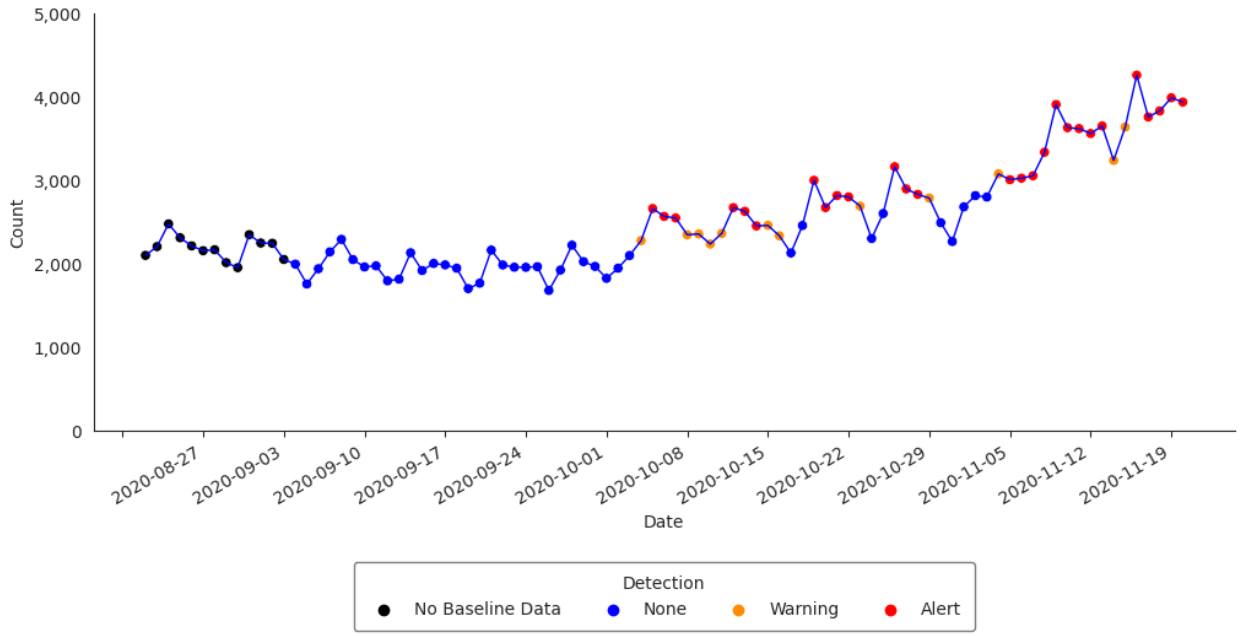
Now, let's visualize the time series with the anomalies

```
>>> sns.set_style('whitegrid') # Set plot grid
>>> sns.set_style("white", {"axes.grid": False})

# Set plot size and y-axis limit
>>> plt.figure(figsize=(12, 6))
>>> plt.gca().set_ylim([0, 5000])

# Plot data
>>> sns.lineplot(data=df_ewma_region, x='date', y='dataCount', color='blue
↪', linewidth=1)

>>> sns.scatterplot(
...     data=df_ewma_region, x='date', y='dataCount', hue='alert',
...     palette=['black', 'blue', 'darkorange', 'red'], alpha=1
... )

# Format dates
>>> plt.gca().yaxis.set_major_formatter(
...     plt.FuncFormatter(lambda x, loc: "{:,}".format(int(x)))
... )
>>> plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator())
>>> plt.gcf().autofmt_xdate()

# Set x and y axes labels
>>> plt.gca().set_xlabel('Date')
>>> plt.gca().set_ylabel('Count')

# Customize legend
>>> legend = plt.legend(
...     title='Detection', loc='lower center', bbox_to_anchor=(0.5, -0.5),
...     ncol=5, frameon=True, framealpha=0.5, edgecolor='black',␣
↪borderpad=0.75
... )

>>> legend.get_texts()[0].set_text('No Baseline Data')
>>> legend.get_texts()[1].set_text('None')
>>> legend.get_texts()[2].set_text('Warning')
>>> legend.get_texts()[3].set_text('Alert')
```

(continues on next page)

(continued from previous page)

```
# Remove the top and right spines from plot
>>> sns.despine()
```



4.4 Adaptive Multiple Regression

The Adaptive Multiple Regression algorithm fits a linear model to a baseline of counts or percentages, and forecasts a predicted value for test dates following a pre-defined buffer period following the baseline. This model includes terms to account for linear trends and day-of-week effects. This implementation does NOT include holiday terms as in the Regression 1.4 algorithm in ESSENCE. The EWMA detection algorithm can be performed with the `alert_regression()` function (run `help(alert_regression)` in your Python console or `alert_regression?` in Jupyter Notebook or JupyterLab for more).

```
>>> df_regression = alert_regression(df_hhs, t = "date", y = "dataCount")
```

Let's filter `df_regression` and visualize the time series with the anomalies for Region 4:

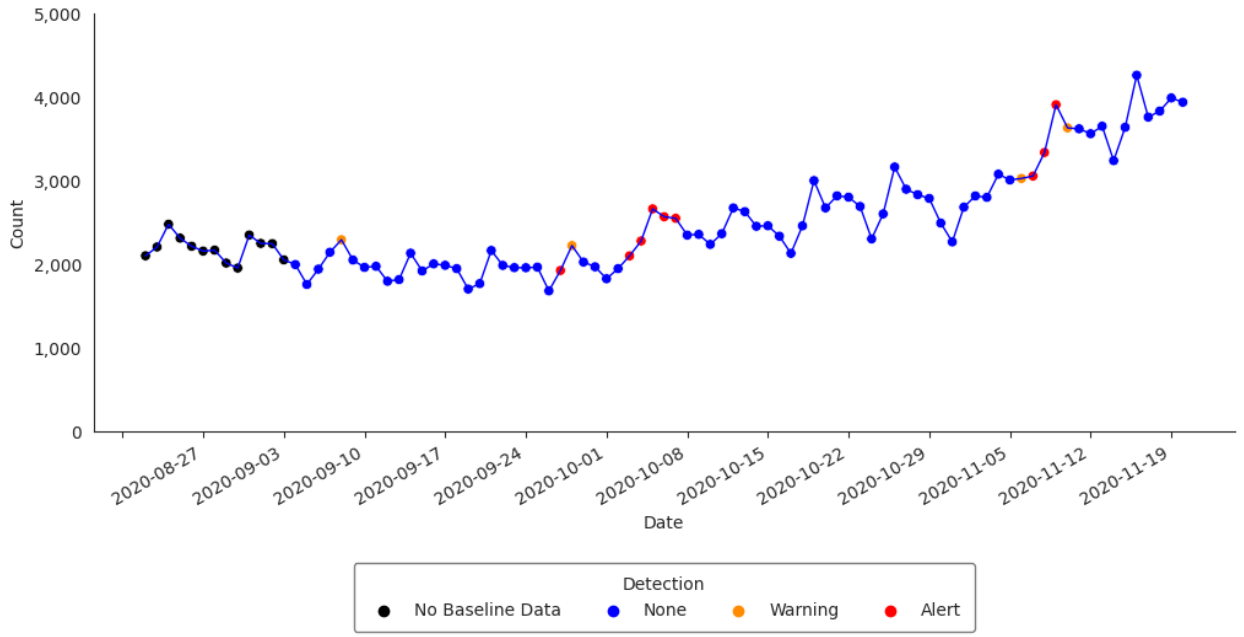
```
>>> df_regression_region = df_regression[df_regression[
↪"hospitaldhhsregion_display"] == "Region 4"]
```

```
>>> sns.set_style('whitegrid') # Set plot grid
>>> sns.set_style("white", {"axes.grid": False})
```

(continues on next page)

(continued from previous page)

```
# Set plot size and y-axis limit
>>> plt.figure(figsize=(12, 6))
>>> plt.gca().set_ylim([0, 5000])

# Plot data
>>> sns.lineplot(data=df_regression_region, x='date', y='dataCount',␣
↪color='blue', linewidth=1)

>>> sns.scatterplot(
...     data=df_regression_region, x='date', y='dataCount', hue='alert',
...     palette=['black', 'blue', 'darkorange', 'red'], alpha=1
... )

# Format dates
>>> plt.gca().yaxis.set_major_formatter(
...     plt.FuncFormatter(lambda x, loc: "{:,}".format(int(x)))
... )
>>> plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator())
>>> plt.gcf().autofmt_xdate()

# Set x and y axes labels
>>> plt.gca().set_xlabel('Date')
>>> plt.gca().set_ylabel('Count')

# Customize legend
>>> legend = plt.legend(
...     title='Detection', loc='lower center', bbox_to_anchor=(0.5, -0.5),
...     ncol=5, frameon=True, framealpha=0.5, edgecolor='black',␣
↪borderpad=0.75
... )

>>> legend.get_texts()[0].set_text('No Baseline Data')
>>> legend.get_texts()[1].set_text('None')
>>> legend.get_texts()[2].set_text('Warning')
>>> legend.get_texts()[3].set_text('Alert')

# Remove the top and right spines from plot
>>> sns.despine()
```



4.5 Regression/EWMA Switch

The NSSP-ESSENCE Regression/EWMA Switch algorithm generalized the Regression and EWMA algorithms by applying the most appropriate algorithm for the data in the baseline. First, adaptive multiple regression is applied where the adjusted R-squared value of the model is examined to see if it meets a threshold of >=0.60 . If this threshold is not met, then the model is considered to not explain the data well. In this case, the algorithm switches to the EWMA algorithm, which is more appropriate for sparser time series that are common with granular geographic levels.

The Regression/EWMA algorithm can be performed with the `alert_switch()` function (run `help(alert_switch)` in your Python console or `alert_switch?` in Jupyter Notebook or JupyterLab for more).

```
>>> df_switch = alert_switch(df_hhs, t = "date", y = "dataCount")
```

Let's visualize the time series with the anomalies for Region 4:

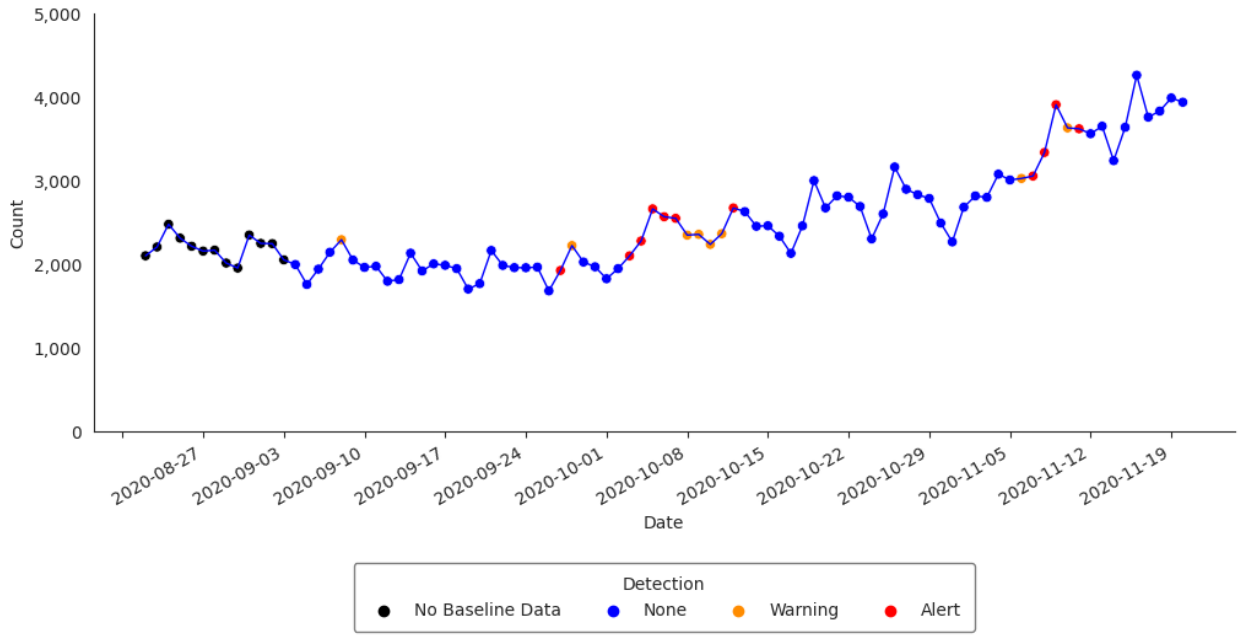
```
>>> df_switch_region = df_switch[df_switch["hospitaldhhsregion_display"]␣
↪== "Region 4"]
```

```
>>> sns.set_style('whitegrid') # Set plot grid
>>> sns.set_style("white", {"axes.grid": False})

# Set plot size and y-axis limit
>>> plt.figure(figsize=(12, 6))
```

(continues on next page)

(continued from previous page)

```
>>> plt.gca().set_ylim([0, 5000])

# Plot data
>>> sns.lineplot(data=df_switch_region, x='date', y='dataCount', color=
↪'blue', linewidth=1)

>>> sns.scatterplot(
...     data=df_switch_region, x='date', y='dataCount', hue='alert',
...     palette=['black', 'blue', 'darkorange', 'red'], alpha=1
... )

# Format dates
>>> plt.gca().yaxis.set_major_formatter(
...     plt.FuncFormatter(lambda x, loc: "{:,}".format(int(x)))
... )
>>> plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator())
>>> plt.gcf().autofmt_xdate()

# Set x and y axes labels
>>> plt.gca().set_xlabel('Date')
>>> plt.gca().set_ylabel('Count')

# Customize legend
>>> legend = plt.legend(
...     title='Detection', loc='lower center', bbox_to_anchor=(0.5, -0.5),
...     ncol=5, frameon=True, framealpha=0.5, edgecolor='black',␣
↪borderpad=0.75
... )

>>> legend.get_texts()[0].set_text('No Baseline Data')
>>> legend.get_texts()[1].set_text('None')
>>> legend.get_texts()[2].set_text('Warning')
>>> legend.get_texts()[3].set_text('Alert')

# Remove the top and right spines from plot
>>> sns.despine()
```



4.6 Negative Binomial Regression

The Negative Binomial Regression algorithm is intended for weekly time series spanning multiple years and fits a negative binomial regression model with a time term and cyclic sine and cosine terms to a baseline period that spans 2 or more years. Inclusion of cyclic terms in the model is intended to account for seasonality common in multi-year weekly time series of counts for syndromes and diseases such as influenza, RSV, and norovirus. Each baseline model is used to make weekly forecasts for all weeks following the baseline period. One-sided upper 95% prediction interval bounds are computed for each week in the prediction period. Alarms are signaled for any week during which the observed weekly count exceeds the upper bound of the prediction interval. The Negative Binomial Regression detector can be applied with the `alert_nbinom()` function (run `help(alert_nbinom)` in your Python console or `alert_nbinom?` in Jupyter Notebook or JupyterLab for more). The example below applies the Negative Binomial Regression detector to our synthetic time series for Scenario #1.

```
>>> synth_ts1 = get_scenario1() # Load synthesized time series data for␣
↪scenario1
>>> synth_ts1.head()
```

Now, applying Negative Binomial detector…

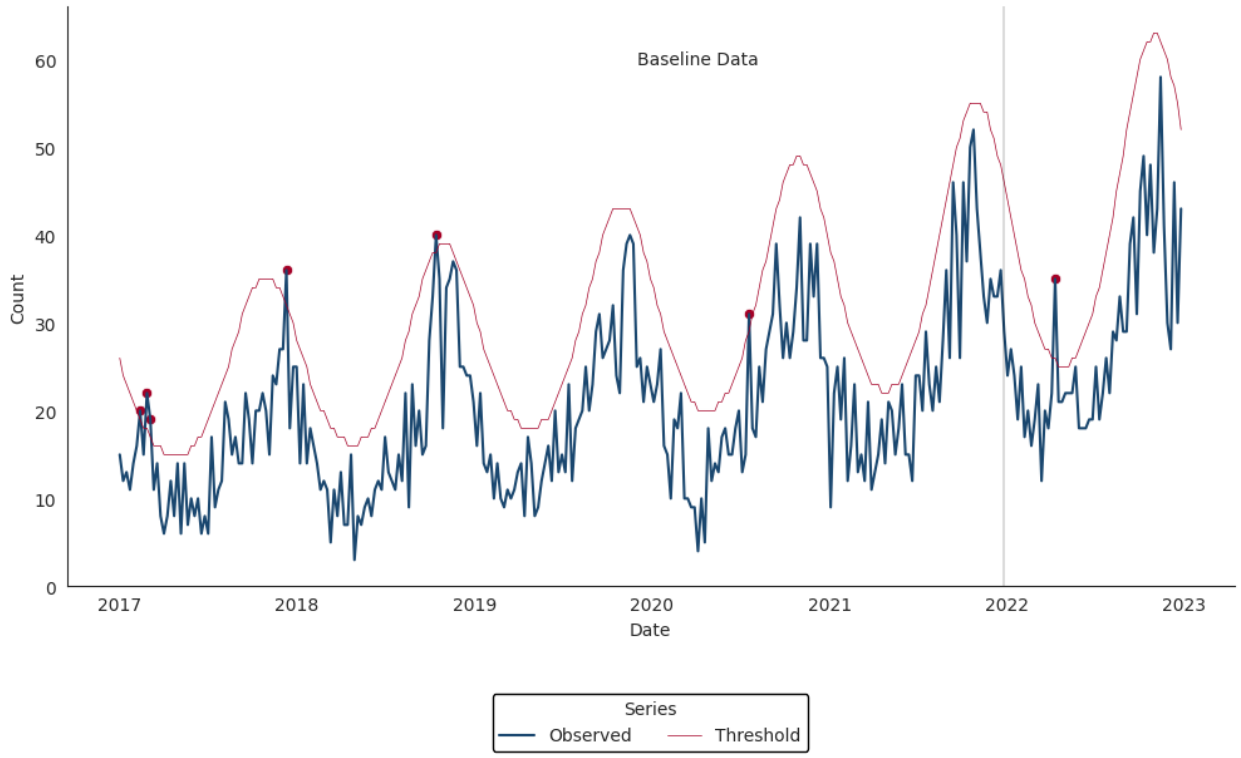
```
>>> df_nbinom = alert_nbinom(synth_ts1, t='date', y='cases', baseline_end=
↪'2021-12-26')
```

```
>>> fig, ax = plt.subplots(figsize=(12, 6))

>>> sns.lineplot(
...     data=df_nbinom, x='date', y='cases',
...     color='#1A476F', label='Observed', ax=ax
... )

>>> sns.lineplot(
...     data=df_nbinom, x='date', y='threshold',
...     color='#A50026', linewidth=0.5, label='Threshold', ax=ax
... )

>>> sns.scatterplot(
...     data=df_nbinom.loc[df_nbinom['alarm']],
...     x='date', y='cases', color='#A50026', ax=ax
... )

# Add vertical line
>>> ax.axvline(x=pd.to_datetime('2021-12-26'), color='black', linewidth=0.
↪2)

# Add text annotation
>>> ax.text(pd.to_datetime('2019-12-01'), 60, 'Baseline Data', ha='left',␣
↪va='center')

# Set axis labels and limits
>>> ax.set_xlabel('Date')
>>> ax.set_ylabel('Count')
>>> ax.set_ylim(bottom=0)
>>> ax.yaxis.set_major_formatter(
...     plt.FuncFormatter(lambda x, loc: "{:,}".format(int(x)))
... )

# Set legend
>>> ax.legend(
...     title="Series", loc='lower center', bbox_to_anchor=(0.5, -0.3),
...     ncol=2, framealpha=1, edgecolor='black'
... )

# Set date axis
>>> ax.xaxis.set_major_locator(mdates.YearLocator())
>>> ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
```

(continues on next page)

(continued from previous page)

```
# Set tick parameters
>>> ax.tick_params(axis='both', length=2.5, width=0.5)

# Set theme
>>> sns.set_style('white')
>>> sns.despine()
```



4.7 Original Serfling Detector

The Original Serfling detector is intended for weekly time series spanning multiple years. It fits a linear regression model with a time term and sine and cosine terms to a baseline period that ideally spans 5 or more years. Inclusion of Fourier terms in the model is intended to account for seasonality common in multi-year weekly time series. This implementation follows the approach of the original Serfling method in which weeks between October of the starting year of a season and May of the ending year of a season are considered to be in the epidemic period. Weeks in the epidemic period are removed from the baseline prior to fitting the regression model. Each baseline model is used to make weekly forecasts for all weeks following the baseline period. One-sided upper 95% prediction interval bounds are computed for each week in the prediction period. Alarms are signaled for any week during which the observed weekly count exceeds the upper bound of the prediction interval. The Original Serfling detector can be applied with the `alert_serfling()` function (run `help(alert_serfling)` in your Python console or `alert_serfling?` in Jupyter Notebook or

JupyterLab for more). Using the same simulated time series from the previous examples, the Negative Binomial Regression detector can be applied as below:

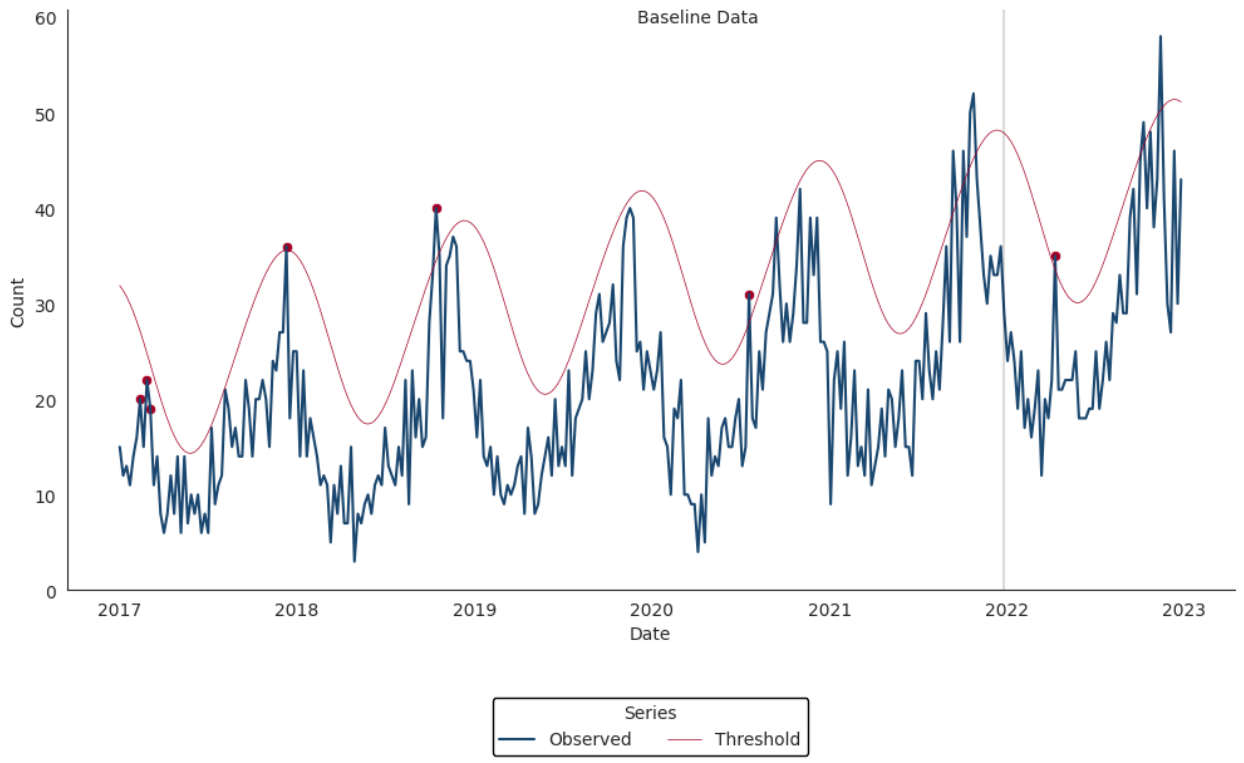
```
>>> df_serfling  = alert_serfling(synth_ts1, t='date', y='cases',␣
↪baseline_end='2021-12-26')
```

```
>>> fig, ax = plt.subplots(figsize=(12, 6))

>>> sns.lineplot(
...     data=df_serfling, x='date', y='cases',
...     color='#1A476F', label='Observed', ax=ax
... )

>>> sns.lineplot(
...     data=df_serfling, x='date', y='threshold',
...     color='#A50026', linewidth=0.5, label='Threshold', ax=ax
... )

>>> sns.scatterplot(
...     data=df_serfling.loc[df_serfling['alarm']],
...     x='date', y='cases', color='#A50026', ax=ax
... )

# Add vertical line
>>> ax.axvline(x=pd.to_datetime('2021-12-26'), color='black', linewidth=0.
↪2)

# Add text annotation
>>> ax.text(pd.to_datetime('2019-12-01'), 60, 'Baseline Data', ha='left',␣
↪va='center')

# Set axis labels and limits
>>> ax.set_xlabel('Date')
>>> ax.set_ylabel('Count')
>>> ax.set_ylim(bottom=0)
>>> ax.yaxis.set_major_formatter(
...     plt.FuncFormatter(lambda x, loc: "{:,}".format(int(x)))
... )

# Set legend
>>> ax.legend(
...     title="Series", loc='lower center', bbox_to_anchor=(0.5, -0.3),
...     ncol=2, framealpha=1, edgecolor='black'
... )
```

(continues on next page)

(continued from previous page)

```
# Set date axis
>>> ax.xaxis.set_major_locator(mdates.YearLocator())
>>> ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

# Set tick parameters
>>> ax.tick_params(axis='both', length=2.5, width=0.5)

# Set theme
>>> sns.set_style('white')
>>> sns.despine()
```



Chapter 5

pynssp

5.1 pynssp package

5.1.1 Subpackages

pynssp.core package

Submodules

pynssp.core.constants module

pynssp.core.container module

class pynssp.core.container.**APIGraph**(*path*, *response*)

Bases: object

A class to store an API graph

plot()

A method to plot an APIGraph object

show()

A method to display an APIGraph object

class pynssp.core.container.**NSSPContainer**(*value*)

Bases: object

A NSSPContainer Class to store a value or an object

An object of class NSSPContainer stores a value or an object The NSSPContainer class encapsulates a value or an object

pynssp.core.credentials module

class pynssp.core.credentials.**Credentials**(*username=None*, *password=None*)

Bases: object

A Token Class Representing a Credentials object

A Credentials object has a username, a password and a key. A Credentials object gets API data via an API URL.

get_api_data(*url*, *fromCSV=False*, *encoding='utf-8'*, ***kwargs*)

Get API data

Parameters

- **url** – a string of API URL
- **fromCSV** – a logical, defines whether data are received in .csv format or .json format (Default value = False)
- **encoding** – an encoding standard (Default value = "utf-8")
- ****kwargs** – Additional keyword arguments to pass to *pandas.read_csv()* if *fromCSV* is True.

Returns
A pandas dataframe

get_api_graph(*url*, *file_ext='.png'*)

Get API graph

Parameters

- **url** – a string of API URL
- **file_ext** – a non-empty character vector giving the file extension. (Default value = ".png")

Returns
an object of type APIGraph

get_api_response(*url*)

Get API response

Parameters
url – a string of API URL

Returns
an object of class response

pickle(*file=None*, *file_ext='.pkl'*)

Save an object of class Credentials to file

Parameters

- **file_ext** – a non-empty character vector giving the file extension. (Default value = ".pkl")
- **file** – (Default value = None)

pynssp.core.token module

class pynssp.core.token.**Token**(*token*, *access_token='Bearer'*)

Bases: object

A Token Class Representing a Token object

A Token object has a token string and a key. A Token object can get API data via an API URL.

get_api_data(*url*, *fromCSV=False*, *encoding='utf-8'*, ***kwargs*)

Get API data

Parameters

- **url** – a string of API URL
- **fromCSV** – a logical, defines whether data are received in .csv format or .json format (Default value = False)
- **encoding** – an encoding standard (Default value = "utf-8")
- ****kwargs** – Additional keyword arguments to pass to *pandas.read_csv()* if *fromCSV* is True.

Returns

A pandas dataframe

get_api_graph(*url*, *file_ext='.png'*)

Get API graph

Parameters

- **url** – a string of API URL
- **file_ext** – a non-empty character vector giving the file extension. (Default value = ".png")

Returns

an object of type APIGraph

get_api_response(*url*)

Get API response

Parameters
url – a string of API URL

Returns
an object of class response

pickle(*file=None*, *file_ext='.pkl'*)

Save an object of class Credentials to file

Parameters

- **file_ext** – a non-empty character vector giving the file extension. (Default value = ".pkl")
- **file** – (Default value = None)

Module contents

pynssp.detectors package

Submodules

pynssp.detectors.ewma module

pynssp.detectors.ewma.**alert_ewma**(*df*, *t='date'*, *y='count'*, *B=28*, *g=2*, *w1=0.4*, *w2=0.9*)

Exponentially Weighted Moving Average (EWMA)

The EWMA compares a weighted average of the most recent visit counts to a baseline expectation. For the weighted average to be tested, an exponential weighting gives the most influence to the most recent observations. This algorithm is appropriate for daily counts that do not have the characteristic features modeled in the regression algorithm. It is more applicable for Emergency Department data from certain hospital groups and for time series with small counts (daily average below 10) because of the limited case definition or chosen geographic region. An alert (red value) is signaled if the statistical test (student"s t-test) applied to the test statistic yields a p-value less than 0.01. If the p-value is greater than or equal to 0.01 and strictly less than 0.05, a warning (yellow value) is signaled. Blue values are returned if an alert or warning does not occur. Grey values represent instances where anomaly detection did not apply (i.e., observations for which baseline data were unavailable).

Parameters

- **df** – A pandas data frame containing time series data
- **t** – Name of the column of type Date containing the dates (Default value = "date")
- **y** – Name of the column of type Numeric containing counts or percentages (Default value = "count")

- **B** – Baseline parameter. The baseline length is the number of days used to calculate rolling averages, standard deviations, and exponentially weighted moving averages. Defaults to 28 days to match ESSENCE implementation.
- **g** – Guardband parameter. The guardband length is the number of days separating the baseline from the current test date. Defaults to 2 days to match ESSENCE implementation.
- **w1** – Smoothing coefficient for sensitivity to gradual events. Must be between 0 and 1 and is recommended to be between 0.3 and 0.5 to account for gradual effects. Defaults to 0.4 to match ESSENCE implementation.
- **w2** – Smoothed coefficient for sensitivity to sudden events. Must be between 0 and 1 and is recommended to be above 0.7 to account for sudden events. Defaults to 0.9 to match ESSENCE implementation and approximate the C2 algorithm.

Returns

Original pandas data frame with detection results.

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> from pynssp.detectors.ewma import *
>>> df = pd.DataFrame({
...     "date": pd.date_range("2020-01-01", "2020-12-31"),
...     "count": np.random.randint(0, 101, size=366)
... })
>>> df_ewma = alert_ewma(df)
>>> df_ewma.head()
```

pynssp.detectors.ewma.**ewma_loop**(*df*, *t*, *y*, *B*, *g*, *w1*, *w2*)

Loop for EWMA

Loop for EWMA and adjustment of outlying smoothed values

Parameters

- **df** – A pandas data frame
- **t** – Name of the column of type Date containing the dates
- **y** – Name of the column containing the response variable data
- **mu** – Numeric vector of baseline averages
- **B** – Baseline parameter. The baseline length is the number of days used to calculate rolling averages, standard deviations, and exponen-

tially weighted moving averages. Defaults to 28 days to match NSSP-ESSENCE implementation.

- **g** – Guardband parameter. The guardband length is the number of days separating the baseline from the current test date. Defaults to 2 days to match NSSP-ESSENCE implementation.
- **w1** – Smoothing coefficient for sensitivity to gradual events. Must be between 0 and 1 and is recommended to be between 0.3 and 0.5 to account for gradual effects. Defaults to 0.4 to match NSSP-ESSENCE implementation.
- **w2** – Smoothed coefficient for sensitivity to sudden events. Must be between 0 and 1 and is recommended to be above 0.7 to account for sudden events. Defaults to 0.9 to match NSSP-ESSENCE implementation and approximate the C2 algorithm.

Returns

A pandas data frame with p-values and test statistics

pynssp.detectors.farrington module

pynssp.detectors.farrington.**alert_farrington**(*df*, *t='date'*, *y='count'*, *B=4*, *g=27*, *w=3*, *p=10*, *method='original'*)

Farrington Temporal Detector

The Farrington algorithm is intended for weekly time series of counts spanning multiple years. Original Farrington Algorithm: Quasi-Poisson generalized linear regression models are fit to baseline counts associated with reference dates in the B previous years, including w weeks before and after each reference date. The algorithm checks for convergence with a time term and refits a model with only an intercept term in the scenario the model does not converge. The inclusion of high baseline counts associated with past outbreaks or public health events is known to result in alerting thresholds that are too high and a reduction in sensitivity. An empirically derived weighting function is used to calculate weights from Anscombe residuals that assign low weight to baseline observations with large residuals. A 2/3rds transformation is applied to account for skewness common to time series with lower counts, after which expected value and variance estimates are used to derive upper and lower bounds for the prediction interval. The alert score is defined as the current observation minus the forecast value divided by the upper prediction interval bound minus the forecast value. If this score exceeds 1, an alert (red value) is raised given that the number of counts in the last 4 days is above 5. This algorithm requires that the number of years included in the baseline is 3 or higher. Blue values are returned if an alert does not occur. Grey values represent instances where anomaly detection did not apply (i.e., observations for which baseline data were unavailable).

Modified Farrington Algorithm: In 2012, Angela Noufaily developed a modified implemen-

tation of the original Farrington algorithm that improved performance by including more historical data in the baseline. The modified algorithm includes all weeks from the beginning of the first reference window to the last week proceeding a 27-week guardband period used to separate the test week from the baseline. A 10-level factor is used to account for seasonality throughout the baseline. Additionally, the modified algorithm assumes a negative binomial distribution on the weekly time series counts, where thresholds are computed as quantiles of the negative binomial distribution with plug-in estimates for mu and phi.

Parameters

- **df** – A pandas dataframe
- **t** – A column containing date values (Default value = "date")
- **y** – A column containing time series counts (Default value = "count")
- **B** – Number of years to include in baseline (default is 4)
- **g** – Number of guardband weeks to separate the test date from the baseline (default is 27)
- **w** – Half the number of weeks included in reference window, before and after each reference date (default is 3)
- **p** – Number of seasonal periods for each year in baseline
- **method** – A string of either "original" (default) or "modified" to specify the version of the Farrington algorithm (original vs modified).

Returns

A dataframe

Examples

```
>>> # Example 1
>>> from pynssp import *
>>> df = get_scenario1()
>>> df_farrington_original  = alert_farrington(df, y=
↪'cases')
>>> df_farrington_original.tail()
```

```
>>> # Example 2
>>> from pynssp import *
>>> df = get_scenario1()
>>> df_farrington_modified  = alert_farrington(df, y=
↪'cases', method="modified")
>>> df_farrington_modified.tail()
```

```
>>> # Example 3 (grouped data)
>>> from pynssp import *
>>> df = load_simulated_ts().groupby("id")
>>> df_farrington_group  = alert_farrington(df, y='cases')
>>> df_farrington_group.tail()
```

pynssp.detectors.farrington.**farrington_modified**(*df*, *t='date'*, *y='count'*, *B=4*, *g=27*, *w=3*, *p=10*)

Modified Farrington Algorithm

Parameters

- **df** – A pandas dataframe
- **t** – A column containing date values (Default value = "date")
- **y** – A column containing time series counts (Default value = "count")
- **B** – Number of years to include in baseline (default is 4)
- **g** – Number of guardband weeks to separate the test date from the baseline (default is 27)
- **w** – Half the number of weeks included in reference window, before and after each reference date (default is 3)
- **p** – Number of seasonal periods for each year in baseline

Returns

A pandas dataframe

Examples

```
>>> from pynssp import *
>>> df = get_scenario1()
>>> df_farrington_modified  = farrington_modified(df, y=
↪'cases', method="modified")
>>> df_farrington_modified.tail()
```

pynssp.detectors.farrington.**farrington_original**(*df*, *t='date'*, *y='count'*, *B=4*, *w=3*)

Original Farrington Algorithm

Parameters

- **df** – A pandas dataframe
- **t** – A column containing date values (Default value = "date")
- **y** – A column containing time series counts (Default value = "count")
- **B** – Number of years to include in baseline (default is 4)

- **w** – Half the number of weeks included in reference window, before and after each reference date (default is 3)

Returns

A pandas dataframe

Examples

```
>>> from pynssp import *
>>> df = get_scenario1()
>>> df_farrington_original  = farrington_original(df, y=
↪'cases')
>>> farrington_original.tail()
```

pynssp.detectors.farrington.**seasonal_groups**(*B=4*, *g=27*, *w=3*, *p=10*, *base_length=None*, *base_weeks=None*)

Return 10-level seasonal factor series

Parameters

- **B** – Number of years to include in baseline (default is 4)
- **g** – Number of guardband weeks to separate the test week from the baseline (default is 27)
- **w** – Half the number of weeks included in reference window, before and after each reference date (default is 3)
- **p** – Number of seasonal periods for each year in baseline (default is 10)
- **base_length** – Total number of weeks included in baseline
- **base_weeks** – Indices of baseline weeks

Returns

A pandas Series

pynssp.detectors.nbinom module

pynssp.detectors.nbinom.**alert_nbinom**(*df*, *baseline_end*, *t='date'*, *y='count'*, *include_time=True*)

Negative binomial detection algorithm for weekly counts

The negative binomial regression algorithm fits a negative binomial regression model with a time term and order 1 Fourier terms to a baseline period that spans 2 or more years. Inclusion of Fourier terms in the model is intended to account for seasonality common in multi-year weekly time series of counts. Order 1 sine and cosine terms are included to account for annual seasonality that is common to syndromes and diseases such as influenza, RSV, and norovirus. Each baseline model is used to make weekly forecasts for all weeks following the baseline

period. One-sided upper 95% prediction interval bounds are computed for each week in the prediction period. Alarms are signaled for any week during for which weekly counts fall above the upper bound of the prediction interval.

Parameters

- **df** – A pandas data frame containing time series data
- **t** – Name of the column of type Date containing the dates (Default value = "date")
- **y** – Name of the column of type Numeric containing counts or percentages (Default value = "count")
- **baseline_end** – date of the end of the baseline/training period (in date or string class)
- **include_time** – Indicate whether or not to include time term in regression model (default is True)

Returns

Original pandas dataframe with model estimates, upper prediction interval bounds, a binary alarm indicator field, and a binary indicator field of whether or not a time term was included.

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> df = pd.DataFrame({
...     "date": pd.date_range(start="2014-01-05", end=
↪"2022-02-05", freq="W"),
...     "count": np.random.poisson(lam=25, size=(len(pd.
↪date_range(start="2014-01-05", end="2022-02-05", freq="W
↪")),))
... })
>>> df_nbinom = alert_nbinom(df, baseline_end = "2020-03-
↪01")
>>> df_nbinom.head()
```

pynssp.detectors.nbinom.**nb_model**(*df*, *t*, *y*, *baseline_end*, *include_time*)

Negative binomial regression model for weekly counts

Negative binomial model helper function for monitoring weekly count time series with seasonality

Parameters

- **df** – A pandas data frame
- **t** – Name of the column of type Date containing the dates

- **y** – Name of the column of type Numeric containing counts
- **baseline_end** – Object of type Date defining the end of the baseline/training period
- **include_time** – Logical indicating whether or not to include time term in regression model

Returns

A pandas data frame.

pynssp.detectors.regression module

pynssp.detectors.regression.**adaptive_regression**(*df*, *t*, *y*, *B*, *g*)

Adaptive Regression

Adaptive Regression helper function for Adaptive Multiple Regression

Parameters

- **df** – A pandas data frame containing time series data
- **t** – Name of the column of type Date containing the dates
- **y** – Name of the column containing the response variable data
- **B** – Baseline parameter. The baseline length is the number of days to which each liner model is fit.
- **g** – Guardband parameter. The guardband length is the number of days separating the baseline from the current date in consideration for alerting.

Returns

A pandas data frame with p-values and test statistics

pynssp.detectors.regression.**alert_regression**(*df*, *t='date'*, *y='count'*, *B=28*, *g=2*)

Adaptive Multiple Regression

The adaptive multiple regression algorithm fits a linear model to a baseline of counts or percentages of length B, and forecasts a predicted value g + 1 days later (guard-band). This value is compared to the current observed value and divided by the standard error of prediction in the test-statistic. The model includes terms to account for linear trends and day-of-week effects. Note that this implementation does NOT account for federal holidays as in the Regression 1.2 algorithm in ESSENCE. An alert (red value) is signaled if the statistical test (student"s t-test) applied to the test statistic yields a p-value less than 0.01. If the p-value is greater than or equal to 0.01 and strictly less than 0.05, a warning (yellow value) is signaled. Blue values are returned if an alert or warning does not occur. Grey values represent instances where anomaly detection did not apply (i.e., observations for which baseline data were unavailable).

Parameters

- **df** – A pandas data frame containing time series data
- **t** – The name of the column in df that contains the dates or times of observations. Defaults to "date".
- **y** – The name of the column in df that contains the values of the time series. Defaults to "count".
- **B** – The length of the baseline period (in days). Must be a multiple of 7 and at least 7. Defaults to 28.
- **g** – The length of the guard band (in days). Must be non-negative. Defaults to 2.

Returns

Original pandas data frame with detection results.

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> from pynssp.detectors.regression import *
>>> df = pd.DataFrame({
...     "date": pd.date_range("2020-01-01", "2020-12-31"),
...     "count": np.random.randint(0, 101, size=366)
... })
>>> df_regression = alert_regression(df)
>>> df_regression.head()
```

pynssp.detectors.serfling module

pynssp.detectors.serfling.**alert_serfling**(*df*, *baseline_end*, *t='date'*, *y='count'*)

Original Serfling method for weekly time series

The original Serfling algorithm fits a linear regression model with a time term and order 1 Fourier terms to a baseline period that ideally spans 5 or more years. Inclusion of Fourier terms in the model is intended to account for seasonality common in multi-year weekly time series. Order 1 sine and cosine terms are included to account for annual seasonality that is common to syndromes and diseases such as influenza, RSV, and norovirus. Each baseline model is used to make weekly forecasts for all weeks following the baseline period. One-sided upper 95% prediction interval bounds are computed for each week in the prediction period. Alarms are signaled for any week during for which weekly observations fall above the upper bound of the prediction interval. This implementation follows the approach of the original Serfling method in which weeks between October of the starting year of a season

and May of the ending year of a season are considered to be in the epidemic period. Weeks in the epidemic period are removed from the baseline prior to fitting the regression model.

Parameters

- **df** – A pandas data frame containing time series data
- **t** – Name of the column of type Date containing the dates (Default value = "date")
- **y** – Name of the column of type Numeric containing counts or percentages (Default value = "count")
- **baseline_end** – date of the end of the baseline/training period (in date or string class)

Returns

Original pandas dataframe with model estimates, upper prediction interval bounds, a binary alarm indicator field, and a binary indicator

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> df = pd.DataFrame({
...     "date": pd.date_range(start="2014-01-05", end=
↪"2022-02-05", freq="W"),
...     "count": np.random.poisson(lam=25, size=(len(pd.
↪date_range(start="2014-01-05", end="2022-02-05", freq="W
↪")),))
... })
>>> df_serfling  = alert_serfling(df, baseline_end =
↪"2020-03-01")
>>> df_serfling.head()
```

pynssp.detectors.serfling.**serfling_model**(*df*, *t*, *y*, *baseline_end*)

Original Serfling method for weekly time series

Serfling model helper function for monitoring weekly time series with seasonality

Parameters

- **df** – A pandas data frame
- **t** – Name of the column of type Date containing the dates
- **y** – Name of the column of type Numeric containing counts
- **baseline_end** – date of the end of the baseline/training period (date or string class)

Returns
A pandas data frame.

pynssp.detectors.switch module

pynssp.detectors.switch.**alert_switch**(*df*, *t='date'*, *y='count'*, *B=28*, *g=2*, *w1=0.4*, *w2=0.9*)

Regression/EWMA Switch

The NSSP-ESSENCE Regression/EWMA Switch algorithm generalized the Regression and EWMA algorithms by applying the most appropriate algorithm for the data in the baseline. First, multiple adaptive regression is applied where the adjusted R squared value of the model is examined to see if it meets a threshold of 0.60. If this threshold is not met, then the model is considered to not explain the data well. In this case, the algorithm switches to the EWMA algorithm, which is more appropriate for sparser time series that are common with county level trends. The smoothing coefficient for the EWMA algorithm is fixed to 0.4.

Parameters

- **df** – A dataframe containing the time series data.
- **t** – The name of the column in *df* containing the time information. Defaults to "date".
- **y** – The name of the column in *df* containing the values to be analyzed. Defaults to "count".
- **B** – The length of the baseline period in days, must be a multiple of 7 and greater than or equal to 7. Defaults to 28.
- **g** – The length of the guardband period in days. Must be non-negative. Defaults to 2.
- **w1** – Smoothing coefficient for sensitivity to gradual events. Must be between 0 and 1 and is recommended to be between 0.3 and 0.5 to account for gradual effects. Defaults to 0.4 to match NSSP-ESSENCE implementation.
- **w2** – Smoothed coefficient for sensitivity to sudden events. Must be between 0 and 1 and is recommended to be above 0.7 to account for sudden events. Defaults to 0.9 to match NSSP-ESSENCE implementation and approximate the C2 algorithm.

Returns
A dataframe containing the results of the analysis.

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> from pynssp.detectors.switch import *
>>> df = pd.DataFrame({
...     "date": pd.date_range("2020-01-01", "2020-12-31"),
...     "count": np.random.randint(0, 101, size=366)
... })
>>> df_switch = alert_switch(df)
>>> df_switch.head()
```

pynssp.detectors.trend module

Module contents

5.1.2 Submodules

5.1.3 pynssp.data module

pynssp.data.**get_scenario1**()

Return a subset of the simulated time series data ('scenario #1'). :examples:

```
>>> from pynssp.data import *
>>> scenario1_ts = get_scenario1()
>>> scenario1_ts.info()
```

pynssp.data.**get_scenario2**()

Return a subset of the simulated time series data ('scenario #2'). :examples:

```
>>> from pynssp.data import *
>>> scenario2_ts = get_scenario2()
>>> scenario2_ts.info()
```

pynssp.data.**load_nssp_stopwords**()

Return a dataframe of NSSP-curated stopwords.

Examples

```
>>> from pynssp.data import *
>>> stopwords = load_nssp_stopwords()
>>> stopwords.info()
... ## #   Column      Non-Null Count  Dtype
```

(continues on next page)

(continued from previous page)

```
... ## ---  ------      --------------  -----
... ## 1   word        835 non-null    object
... ## 2   type        835 non-null    object
... ## dtypes: int64(1), object(2)
... ## memory usage: 13.2+ KB
```

pynssp.data.**load_simulated_ts**()

Return a dataframe of simulated time series.

Examples

```
>>> from pynssp.data import *
>>> simulated_ts = load_simulated_ts()
>>> simulated_ts.info()
... ## #   Column  Non-Null Count  Dtype
... ## ---  ------  --------------  -----
... ## 0   date    626 non-null    object
... ## 1   week    626 non-null    int64
... ## 2   year    626 non-null    int64
... ## 3   cases   626 non-null    int64
... ## 4   id      626 non-null    object
... ## dtypes: int64(3), object(2)
... ## memory usage: 24.6+ KB
```

5.1.4 pynssp.pynssp module

Main module.

5.1.5 pynssp.utils module

pynssp.utils.**change_dates**(*url*, *start_date=None*, *end_date=None*)

Changes the start and end dates in a given URL to new dates, if provided.

Parameters

- **url** – str): The URL containing the start and end dates to be changed.
- **start_date** – str): A new start date to replace the existing start date in the URL. (Default value = None)
- **end_date** – str): A new end date to replace the existing end date in the URL. (Default value = None)

Returns

The modified URL with the new start and end dates.

Examples

```
>>> from pynssp.utils import change_dates
>>> url = "https://example.com/data?startDate=01Jan2022&
↪endDate=31Dec2022"
>>> change_dates(url, start_date="01Jan2021", end_date=
↪"31Dec2021")
```

pynssp.utils.**create_profile**(*username=None*, *password=None*)

Create a new user profile with the given username and password.

Parameters

- **username** – A string representing the username. If not provided, the user will be prompted to enter it.
- **password** – A string representing the user's password. If not provided, the user will be prompted to enter it securely.

Returns

A new Credentials object with the given username and password.

Examples

```
>>> from pynssp.utils import create_profile
>>> myProfile = create_profile()
```

pynssp.utils.**create_token_profile**(*token=None*, *access_token='Bearer'*)

Create a new token profile with the given token and authentication type.

Parameters

- **token** – A string representing the token. If not provided, the user will be prompted to enter it securely.
- **auth_type** – A string representing the authentication type. Defaults to "Bearer".

Returns

A new Token object with the given token and authentication type.

Examples

```
>>> from pynssp.utils import create_token_profile
>>> myTokenProfile = create_token_profile()
```

pynssp.utils.**get_api_data**(*url*, *fromCSV=False*, *profile=None*, *encoding='utf-8'*, ***kwargs*)

Retrieve data from an API using the provided profile.

Parameters

- **url** – A string representing the URL of the API endpoint.
- **fromCSV** – A boolean indicating whether the data should be retrieved from a CSV file. Defaults to False.
- **profile** – An profile object of class *pynssp.core.credentials.Credentials* or *pynssp.core.token.Token*.
- **kwargs** – Additional keyword arguments to be passed to the profile's get_api_data method.

Returns

The data retrieved from the API.

Examples

```
>>> from pynssp.utils import *
>>> myProfile = create_profile()
>>> url = "http://httpbin.org/json"
>>> api_data = get_api_data(url, profile=myProfile)
```

pynssp.utils.**get_api_graph**(*url*, *file_ext='.png'*, *profile=None*)

Retrieve a graph from an API using the provided profile.

Parameters

- **url** – A string representing the URL of the API endpoint.
- **file_ext** – A string representing the file extension of the graph. Defaults to ".png".
- **profile** – An profile object of class *pynssp.core.credentials.Credentials* or *pynssp.core.token.Token*.

Returns

The graph retrieved from the API.

Examples

```
>>> from pynssp.utils import *
>>> myProfile = create_profile()
>>> url = "http://httpbin.org/image/png"
>>> api_graph = get_api_response(url, profile=myProfile)
```

pynssp.utils.**get_api_response**(*url*, *profile=None*)

Retrieve a response from an API using the provided profile.

Parameters

- **url** – A string representing the URL of the API endpoint.
- **profile** – An profile object of class *pynssp.core.credentials.Credentials* or *pynssp.core.token.Token*.

Returns

The response object returned by the API.

Examples

```
>>> from pynssp.utils import *
>>> myProfile = create_profile()
>>> url = "http://httpbin.org/json"
>>> response = get_api_response(url, profile=myProfile)
```

pynssp.utils.**get_essence_data**(*url*, *start_date=None*, *end_date=None*, *profile=None*, ***kwargs*)

Retrieve data from the NSSP-ESSENCE API using the provided profile.

Parameters

- **url** – A string representing the URL of the NSSP-ESSENCE API endpoint.
- **start_date** – A string representing the start date of the data to retrieve.
- **end_date** – A string representing the end date of the data to retrieve.
- **profile** – An profile object of class *pynssp.core.credentials.Credentials* or *pynssp.core.token.Token*.
- **kwargs** – Additional arguments to be passed to the get_api_data function.

Returns

The data retrieved from the NSSP-ESSENCE API.

Examples

```
>>> from pynssp.utils import *
>>> myProfile = create_profile()
>>> url = "https://essence2.syndromicsurveillance.org/
↪nssp_essence/api/timeSeries/graph?endDate=25Jun2022&
↪geography=&percentParam=noPercent&datasource=va_hosp&
↪startDate=25Jun2021&
↪medicalGroupingSystem=essencesyndromes&userId=3751&
```

(continues on next page)

(continued from previous page)

```
↪aqtTarget=TimeSeries&ccddCategory=&
↪geographySystem=hospitalregion&detector=probrepswitch&
↪timeResolution=daily"
>>> api_data = get_essence_data(url, profile=myProfile)
>>> api_data.info()
```

pynssp.utils.**webscrape_icd**(*icd_version='ICD10'*, *year=None*)

ICD Code Web Scraper

Function to web scrape ICD discharge diagnosis code sets from the CDC FTP server (for ICD-10) or CMS website (for ICD-9). If pulling ICD-10 codes, by default the function will search for the most recent year's code set publication by NCHS. Users can specify earlier publication years back to 2019 if needed. The ICD-9 option will only web scrape the most recent, final ICD-9 code set publication (2014) from the CMS website. This function will return an error message if the FTP server or CMS website is unresponsive or if a timeout of 60 seconds is reached. The result is a dataframe with 3 fields: code, description, and set (ICD version concatenated with year). Codes are standardized to upper case with punctuation and extra leading/tailing white space removed to enable successful joining.

Parameters

- **icd_version** – The version of ICD codes to retrieve. Default is 'ICD10'.
- **year** – The year for which to retrieve the ICD codes. If not provided, the current year will be used. (Default value = None)

Returns

A DataFrame containing the ICD codes and descriptions.

Examples

```
>>> from pynssp.utils import *
# Example 1
>>> icd9 = webscrape_icd(icd_version = "ICD9")
>>> icd9.head()
# Example 2
>>> icd10_2021 = webscrape_icd(icd_version="ICD10",␣
↪year=2021)
>>> icd10_2021.info()
# Example 3
>>> icd10_2020 = webscrape_icd(icd_version="ICD10",␣
↪year=2020)
>>> icd10_2020.info()
```

5.1.6 Module contents

Top-level package for pynssp.

Chapter 6

Release Notes

6.1 0.1.0 (2023-??-??)

- First release on PyPI (upcoming - not yet released on PyPI).

Chapter 7

Credits

7.1 Development Lead

- Gbedegnon Roseric Azondekon

7.2 Contributors

None yet. Why not be the first?

Chapter 8

Contributing

8.1 Welcome!

Thank you for contributing to CDC's Open Source projects! If you have any questions or doubts, don't be afraid to send them our way. We appreciate all contributions, and we are looking forward to fostering an open, transparent, and collaborative environment.

Before contributing, we encourage you to also read or LICENSE, README, and code-of-conduct files, also found in this repository. If you have any inquiries or questions not answered by the content of this repository, feel free to contact us.

8.2 Public Domain

This project is in the public domain within the United States, and copyright and related rights in the work worldwide are waived through the CC0 1.0 Universal public domain dedication. All contributions to this project will be released under the CC0 dedication. By submitting a pull request you are agreeing to comply with this waiver of copyright interest.

8.3 Requesting Changes

Our pull request/merging process is designed to give the CDC Surveillance Team and other in our space an opportunity to consider and discuss any suggested changes. This policy affects all CDC spaces, both on-line and off, and all users are expected to abide by it.

8.3.1 Open an issue in the repository

If you don't have specific language to submit but would like to suggest a change or have something addressed, you can open an issue in this repository. Team members will respond to the issue as soon as possible.

8.3.2 Submit a pull request

If you would like to contribute, please submit a pull request. In order for us to merge a pull request, it must:

- Be at least seven days old. Pull requests may be held longer if necessary to give people the opportunity to assess it.
- Receive a +1 from a majority of team members associated with the request. If there is significant dissent between the team, a meeting will be held to discuss a plan of action for the pull request.

Chapter 9

Disclaimer

Use of this service is limited only to **non-sensitive and publicly available data**. Users must not use, share, or store any kind of sensitive data like health status, provision or payment of healthcare, Personally Identifiable Information (PII) and/or Protected Health Information (PHI), etc. under **ANY** circumstance.

Administrators for this service reserve the right to moderate all information used, shared, or stored with this service at any time. Any user that cannot abide by this disclaimer and Code of Conduct may be subject to action, up to and including revoking access to services.

The material embodied in this software is provided to you "as-is" and without warranty of any kind, express, implied or otherwise, including without limitation, any warranty of fitness for a particular purpose. In no event shall the Centers for Disease Control and Prevention (CDC) or the United States (U.S.) government be liable to you or anyone else for any direct, special, incidental, indirect or consequential damages of any kind, or any damages whatsoever, including without limitation, loss of profit, loss of use, savings or revenue, or the claims of third parties, whether or not CDC or the U.S. government has been advised of the possibility of such loss, however caused and on any theory of liability, arising out of or in connection with the possession, use or performance of this software.

Chapter 10

Creating a Culture of Innovation

We aspire to create a culture where people work joyfully, communicate openly about things that matter, and provide great services globally. We would like our team and communities (both government and private sector) to reflect on diversity of all kinds, not just the classes protected in law. Diversity fosters innovation. Diverse teams are creative teams. We need a diversity of perspective to create solutions for the challenges we face.

This is our code of conduct (adapted from 18F's Code of Conduct). We follow all Equal Employment Opportunity laws and we expect everyone we work with to adhere to the GSA Anti-harrasment Policy, even if they do not work for the Centers for Disease Control and Prevention or GSA. We expect every user to follow this code of conduct and the laws and policies mentioned above.

10.1 Be Empowering

Consider what you can do to encourage and support others. Make room for quieter voices to contribute. Offer support and enthusiasm for great ideas. Leverage the low cost of experimentation to support your colleagues' ideas, and take care to acknowledge the original source. Look for ways to contribute and collaborate, even in situations where you normally wouldn't. Share your knowledge and skills. Prioritize access for and input from those who are traditionally excluded from the civic process.

10.2 Rules of Behavior

- I understand that I must complete security awareness and records management training annually in order to comply with the latest security and records management policies.
- I understand that I must also follow the Rules of Behavior for use of HHS Information Resources
- I understand that I must not use, share, or store any kind of sensitive data (health status, provision or payment of healthcare, PII, etc.) under ANY circumstance.
- I will not knowingly conceal, falsify, or remove information.
- I understand that I can only use non-sensitive and/or publicly available data.
- I understand that all passwords I create to set up accounts need to comply with CDC's password policy.
- I understand that the stewards reserves the right to moderate all data at any time.

10.3 Boundaries

Create boundaries to your own behavior and consider how you can create a safe space that helps prevent unacceptable behavior by others. We can't list all instances of unacceptable behavior, but we can provide examples to help guide our community in thinking through how to respond when we experience these types of behavior, whether directed at ourselves or others.

If you are unsure if something is appropriate behavior, it probably is not. Each person we interact with can define where the line is for them. Impact matters more than intent. Ensuring that your behavior does not have a negative impact is your responsibility. Problems usually arise when we assume that our way of thinking or behavior is the norm for everyone.

10.3.1 Here are some examples of unacceptable behavior

- Negative or offensive remarks based on the protected classes as listed in the GSA Anti-harrasment Policy of race, religion, color, sex, national origin, age, disability, genetric information, sexual orientation, gender identity, parental status, maritual status, and political affiliation as well as gender expression, mental illness, socioeconomic status or backgrounds, neuro(a)typicality, physical appearance, body size, or clothing. Consider that calling attention to differences can feel alienating.
- Sustained disruption of meetings, talks, or discussions, including chatrooms.
- Patronizing language or behavior.

- Aggresive behavior, such as unconstructive criticism, providing correction that do not improve the conversation (sometimes referred to as "well actually's"), repeatedly interrupting or talking over someone else, feigning surprise at someone's lack of knowledge or awareness about a topic, or subtle prejudice.
- Referring to people in a way that misidentifies their gender and/or rejects the validity of their gender identity; for instance by using incorrect pronouns or forms of address (misgendering).
- Retaliating against anyone who files a formal complaint that someone has violated these codes or laws.

10.4 Background

CDC Scientific Clearance is the process of obtaining approvals by appropriate CDC officials before a CDC information product is released to the public or CDC's external public health partners. Information products that require formal clearance include print, electronic, or oral materials, that CDC employees author or co-author, whether published by CDC or outside CDC. CDC contractors developing content on behalf of CDC for the public or CDC's external public health partners are also required to put their content through the formal clearance process. The collaborative functions related to the projects include blogs, wikis, forums, bug tracking sites, source control and others deemed as necessary.

For those individuals within the CDC, adherence to the following policies are required:

- CDC "Clearance of Information Products Disseminated Outside CDC for Public Use"
- HHS "Ensuring the Quality of Information Disseminated by HHS agencies"

All collaborative materials will be controlled by the rules contained within this document. This will allow for the real-time collaboration opportunities among CDC employees, CDC contractors and CDC public health partners.

10.5 Credit

This code of conduct was mainly adapted from 18F's Code of Conduct and the CDC's Informatics Innovation Unit R&D Lab's code of conduct.

10.6 Relevant Legal Considerations

- Laws enforced by the Equal Employment Opportunity Commission
- Types of discrimination prohibited by law
- New and proposed regulations

Chapter 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

N

P

S

T

W