

**NURO Analysis Code Manual**

Table of Contents

**Requirements1**

**File Setup1**

**Code Execution1**

Locating the Code2

Running Local App2

Deploying To Server2

**Updating Tests1**

Editing Code Manually2

Using modifyTests.py2

Changing JSON Files2

**Adding Data Columns1**

**Potential Future Improvements4**

Using Weather Data5

Integrating With NURO Connect5

Efficiency5

Adding Modify Menu to modifyTests.py5

**Commands and Terminology4**

**Resources1**

**Requirements**

**Languages**: Python (version 3.9.6), JavaScript, CSS, HTML

**Virtual Environment/Server**: download all libraries in virtual environment or on server using pip install <library name>; see resources on more information on creating a virtual environment

**Libraries**: Flask, Werkzeug, Requests, Pandas, meteostat\*\*

**Minimum Usage**: 30 users at one time; average size of GET request is 330,000 bytes

**Version 1.0:**

* Status: deployed; tested and debugged
* Latest Update: added Help Page
* Contents: new UI, no weather data, hard-coded tests
* Link: <https://github.com/elizabethpursell/Boiler-Web-App-Deploy>

**Version 2.0:**

* Status: not deployed; current version; working consistently
* Latest Update: added weather data and Test 14; used modifyTests.py to dynamically update tests
* Contents: weather data using meteostat library + mapsquest API, Test 14, daily weather chart; modifyTests.py; test execution using JSON files
* Link: <https://github.com/elizabethpursell/Boiler-Web-App-Current>

**API Credentials\*\***: store in config.py for security

* MapQuest: gets coordinates for weather data; large usage limit
  + Request URL (coordsAPI): “<https://www.mapquestapi.com/geocoding/v1/batch>”
* Visual Crossing: gets weather data; unused; replaced by meteostat because usage limits
  + Request URL: “[https://weather.visualcrossing.com/VisualCrossingWebServices/ rest/services/timeline/](https://weather.visualcrossing.com/VisualCrossingWebServices/%20rest/services/timeline/)”
* Tomorrow.io: gets weather data; unused; replaced by meteostat because usage limits
  + Request URL: “<https://api.tomorrow.io/v4/historical>”

**\*\* Only available for Version 2.0**

**File Setup**

**NuroConnect**: functions to get data from NURO Connect API

* **\_\_init\_\_.py**: creates NuroConnect object to get and store data from API

**Static**:

* **CSS**: each page is styled by its own css file; mainly used for positioning, coloring, sizing
* **IMG**: all images are stored here, specifically for the home and help pages
* **JS**: each page is run by its own JavaScript file; used to display data to webpage, generate graphs, execute button functions
  + **analyzeData.js**: creates HTML for each test
  + **generateGraph.js**: generates test graph when its button is pressed
  + **graphFunctions.js**: functions to create info popups
  + **graphOptions.js**: functions for all graph features
  + **availableBoilers.js**: creates HTML for each boiler
* **PDF**: holds NURO Analysis Guide
* **PY**: all Python files that are used for running tests are stored here
  + **analyzeRecentData.py**: runs each test
  + **createTest.py\*\***: gets error/warning sets and other test setup options
  + **executeTest.py\*\***: gets graph data points and styles for each test
* **TXT**: files for boiler models, states, and statuses

**Templates**: each page is setup by its own HTML file; base.html is the parent for all pages

**App.py**: sets up all flask app routes to render HTML for each page and get data from API

**Requirements.txt**: holds all libraries that need to be downloaded to run app; created using pip freeze > requirements.txt

**Config.py\*\***: stores API keys and request URLs

**ModifyTests.py\*\***: generates JSON files for sets and tests based on user input; eliminates hardcoding of tests and sets

**setOptions.json\*\***: file generated by modifyTests.py to hold available sets for analysis; structure explained in setOptionsStructure.txt

**testOptions.json\*\***: file generated by modifyTests.py to hold all tests options; app reads test data from this file to execute correct tests; structure explained in testOptionsStructure.txt

**\*\* Only available for Version 2.0**

**Code Execution**

**Locating the Code**: follow this tutorial before running locally or deploying to server

**\*\* Requirements**: Git or GitHub

1. Open the Command Prompt/Terminal using the search bar on your computer
2. Navigate to where you want to copy the code to or where the code is on your computer using the command cd foldername. (ex. cd Desktop would go to the Desktop folder)
3. If needed, copy over the code to your current folder
   1. Using Git:
      1. Open the link to the GitHub Repository that has the desired code
      2. Press the green <>Code dropdown
      3. Under the Local tab, select the HTTPS tab and copy the URL
      4. Back in the terminal, run the command git clone GitHubURL, where GitHubURL is the url that you copied from GitHub
   2. Using GitHub Zip File:
      1. Open File Explorer and navigate to the folder where you want the code
      2. Open the link to the GitHub Repository that has the desired code
      3. Press the green <>Code dropdown
      4. Under the Local tab, press the Download ZIP button
      5. Drag the Zip file to the folder where you want the code
      6. Extract the Zip file

**Running Local App**: see resources for more information if needed

**\*\* Requirements**: Python

1. Locate the code on your computer
2. Create a virtual environment. Follow tutorial in resources if needed.
   1. Install virtual environment using pip install –user virtualenv
   2. Create a virtual environment using python –m virtualenv venv
   3. Navigate to the venv/Scripts folder using cd venv and cd Scripts
   4. Activate the virtual environment using activate.bat
3. Download necessary libraries from requirements.txt using pip install –r requirements.txt
4. Run the app using python app.py
5. Open the local link that is given after executing the command

**Deploying to Server**: see resources for more information

**\*\* Requirements**: Python, Git, Azure access

1. Locate the code on your computer and navigate to the folder in the Terminal/Command Prompt using cd foldername
2. Login to the Azure Portal here: <https://portal.azure.com/>
3. On the home page, under Resources, select the NURO Analysis app
4. Press the Deployment Center button on the sidebar
5. Copy the Git Clone Uri link
6. Back in the Terminal, if you are outside the code folder, use cd foldername to open it. You are now in the root directory/folder.
7. Use the command git remote add azure GitCloneUri, where GitCloneUri is the link copied from the Azure Portal, to connect the code to Azure
8. Use the command git push azure main:master to send the code to Azure
   1. If it asks for credentials, in the Azure Deployment Center, under the Local Git/FTPS Credentials tab, find the heading Application Scope
   2. Use the Local Git Username and Password as the credentials to finalize the code update

**Updating Tests**

**Editing Code Manually\***

1. Open analyzeRecentData.py, which is located in the py folder inside the static folder, in an IDE or with Notepad
2. Scroll to the bottom of the file and find the last test function
3. Copy that function and paste it directly underneath it
4. Rename the function to be relevant for the new test
5. To create the errorSet, which identifies error points, use the same format as the copied errorSet. Change the set names, which are surrounded by quotes, as needed. Use & (and) or | (or) to create a compound condition to combine multiple conditions.
   1. Ex) errorSet = data[(data[“setName1”] > 30) & (data[“setName2”] < 20)]
      1. Error points when setName1 > 30 and setName2 < 20
   2. If a duration check needed, use the checkDuration function and pass the errorSet, data, duration in minutes, and overLimit/underLimit based on if you want to trigger an error point if over or under the given duration.
      1. Ex) errorSet = checkDuration(errorSet, data, 30, overLimit)
         1. Error points when original errorSet conditions are met over 30 minutes at a time
6. To create the warningSet, which identifies warning points, use the same procedure as creating an errorSet
7. Assign title to be a description of the error conditions
8. Use the sets list to hold the set names for the sets to be graphed for the test
9. To create the testOptions dictionary, use the same structure as the previous test. Each set in the sets list is a key in the dictionary. Assign a title, axisID, thresholdLabel, thresholdType, thresholdFill, warnVal, thresholdData as needed.
   1. title: legend label for the dataset (str)
   2. axisID: uses left or right axis (str)
   3. thresholdLabel: describes the threshold condition (str)
   4. thresholdType: custom
   5. thresholdFill: where the threshold should be filled (str)
      1. if thresholdType default or custom, can be above, below, or none
      2. if thresholdType exists, can be all
      3. if thresholdType abs, can be between or split
         1. abs between is used when absolute value < warnVal
         2. abs split is used when absolute value > warnVal
   6. warnVal: value that triggers a warning point; used with all thresholdTypes, except custom(float)
   7. thresholdData: list of data points to create threshold line; see test1; only used with thresholdType custom (list)
10. To assign titles to the axes, use the axisTitles dictionary. Set the left and right keys to their corresponding titles.
11. For errorMsg, use the causes for the test failure. This will display on the Error Summary popup
12. Save the changes to analyzeRecentData.py.
13. Open app.py, which is located app’s route directory, in an IDE or with Notepad
14. Scroll down to the function named analyzeDataPost that has all the test function calls
15. Using the same format as the rest of the tests, add the function call to the new test.
16. Save the changes and test the program locally

**\* Only available for Version 1.0**

**Using modifyTests.py\*\***

\*\* At all times, press CTRL + Fn + Pause Break/PgUp or CTRL + C to terminate the program

1. Using the Command Prompt/Terminal and the cd folderName command, navigate to the app’s code folder.
2. Once in the root directory of the app, use the command python modifyTests.py to run the program. This opens the main menu to update test information.
3. Option 1: Add a Test
   1. To select add a test, enter 1 or add into the command line
   2. The program automatically assigns a test number based on the existing tests
   3. The first task is to assign a title to the test. Enter a description of the error conditions as the title
   4. After the title is entered, a list of all the available sets to choose from is displayed. Enter the keywords of the sets that you want to include in the test. Select the set that you want to use on the left axis first for the correct auto generation of the axes. The keywords are found in the parentheses next to the set titles. Enter the sets one at a time, and enter done once all sets have been added. Verify the set list after submission.
   5. Then, choose whether or not you would like to check the duration of the error/warning conditions.
   6. After that, begin constructing the error conditions that will be used to identify the error points of the test. Enter the set that you want to compare. Decide if you want to compare the selected set to a value, another set, or a custom set.
      1. For value comparisons, first decide if you want to use the absolute value of the selected set in the condition. Then, enter the value that you want to compare the selected set to. Lastly, select the comparison operator that you want for the condition. Verify that the condition is correct.
      2. For set comparisons, first select the set that you want to compare the initial set to. Then, enter the comparison operator that you want for the condition. Verify that the condition is correct.
      3. For custom comparisons, first enter the set that you want to compare the initial set to. Then, input a value to adjust the set by. Next, select the operation that you want to use with the adjustment value (+ - \* /). Lastly, enter the comparison operator that you want for the condition. Verify that the condition is correct.
         1. Ex) To make the condition outlet > hx + 5 , select outlet as the initial set, hx as the comparison set, 5 as the adjustment, + as the adjustment operation, and > as the comparison operator
   7. Verify that the error condition is correct.
      1. If you chose to check a duration, enter the amount of minutes that the error condition needs to occur for to create an error point. Then, select if you want to create an error point if the duration is over or under the duration. Verify the duration condition.
   8. Choose if you want to add another error condition to the test.
      1. If you chose to add another condition, select if you want to AND or OR the two condition together. AND creates an error point only if both conditions are true. OR creates an error point if at least one of the conditions are true. Repeat step f for the new condition.
      2. If not, verify that the overall error condition is correct
   9. Next, follow the same procedure starting from step f but for the warning conditions.
   10. Determine if the auto generated dataset labels are correct. If they aren’t, enter the correct label.
   11. Determine if the auto generated axes are correct. If they aren’t, for each set, select either the left or right axis, and, for each axis, enter the correct axis title
   12. Lastly, enter the information for the test explanation. Input the causes for the test failure. This will appear on the error summary popup and the help page test explanation table. Enter the importance of addressing the test failures. This will appear on the help page test explanation table.
   13. Data is saved to testOptions.json
4. Option 2: Remove a Test
   1. To select remove a test, enter 2 or remove into the command line
   2. Enter the number of the test that you want to delete.
   3. Verify the correct removal of the test.
   4. Data is saved to testOptions.json
5. Option 3: Update Sets
   1. See the Adding Data Columns section
6. Test the program locally

**\*\* Only available for Version 2.0**

**Changing JSON Files\*\***

1. From the root directory, open testOptions.json. It may also be helpful to open testOptionsStructure.txt for further explanation of the JSON structure.
2. To add a new test, copy an existing test. Start from the test number to the end of field called “importance”, including the one closing curly brace and the comma.
3. Add a comma after the last test and paste the copied test.
4. Change the test number to the one more than the last test.
5. Replace the old title with a description of the error conditions of the new test
6. Set the type as default, duration, daily weather, hourly weather, or all weather, depending on the testing that you are doing. Duration is for testing the duration of an error condition. Daily weather is for using daily weather data. Hourly weather is for using hourly weather data. All weather is for using all weather data. Default is for using the default data.
7. For sets, create a list of all the set names that you will graph for the test.
8. For error, create a list of all the error conditions for error points.
   1. Each condition is assigned a type (set, value, custom, abs) based on what values are compared. Set is for comparing two sets. Value is for comparing a set to a value. Custom is for comparing a set to another set that is adjusted by a constant. Abs is for comparing the absolute value of a set to a value.
   2. Each condition also is assigned a set1, which is the initial set to compare, a set2 object, which is the set that set1 is compared to, and a compare, which is how the two sets are compared (< ,<=, >, >=, ==, !=).
   3. The structure of the set2 object is dependent on the type of condition. For set types, name is the name of the set that set1 is compared to, and the other fields are left blank or “”. For value and abs types, value is the number that set1 is compared to, and the other fields are left blank or “”. For custom types, name is the name of the set that set1 is compared to, value is the number that the set is adjusted by, and operation is the operation that is used to adjust the set by the value.
   4. If multiple error conditions are needed, separate the condition objects by “and” or “or” to dictate if an error point occurs only if both are true or if at least one is true.
9. For errorDuration, if the test type is duration, assign value to be the duration to check for and assign compare as > or < to check if the duration is over or under value.
10. Warning and warningDuration have the same structure as error and errorDuration.
11. For options, each set in the sets list gets its own object. They get assigned a title, which is shown on the legend, an axisID to set the dataset to either the left or the right axis, units for labeling the values, a thresholdLabel to describe the threshold condition that will appear on the dataset information popup, a thresholdType (value, none, set, custom, exists, or abs) to identify the comparison type, and a thresholdFill to choose which part gets filled (above, below, none, all, split, between).
12. For axisTitles, assign a title to the left and right axes
13. For causes, describe the causes for a test failure. This will appear on the error summary and help page
14. For importance, describe the importance for addressing the test failure. This will appear on the help page.
15. Save the changes and test the program locally.

**\*\* Only available for Version 2.0**

**Adding Data Columns**

**Updating analyzeRecentData.py**

1. Open analyzeRecentData.py, which is located in the py folder inside the static folder, in an IDE or with Notepad
2. Go to the first function named cleanRawData.
3. Here is where you can add new data columns. All of the relevant data columns are added already, but you can add additional difference columns using the same structure.
4. You can print the names of the columns using print(data.columns, flush=True) and run the program locally. The data frame columns will be printed to your terminal. You can print the whole data frame using print(data, flush=True).

**Updating Weather Data\*\***

1. Open \_\_init\_\_.py, which is located in the NuroConnect folder, in an IDE or with Notepad
2. Go to the functions named getDailyWeather and getHourlyWeather. Either one can be updated. You can add columns by doing calculation, like with Heating Degree Days, or you can add columns to find the change in a column.
3. Save the changes and see the section Applying Changes Using modifyTests.py

**Applying Changes Using modifyTests.py\*\***

\*\* At all times, press CTRL + Fn + Pause Break/PgUp or CTRL + C to terminate the program

1. Using the Command Prompt/Terminal and the cd folderName command, navigate to the app’s code folder.
2. Once in the root directory of the app, use the command python modifyTests.py to run the program. This opens the main menu to update test information.
3. Select Update Sets by entering 3 or update into the command line
4. Before using this menu, follow the procedure under the section Updating analyzeRecentData.py or Updating Weather Data
5. Enter the set keyword, which is the column name that you added, to add it to the modifyTests.py menu
6. Input the legend label for the set
7. Select the dataset that you added the column to
8. Choose the measurement type for the column. Default uses imperial units. Setpoint uses imperial units and is used for any column that stores a boiler setpoint. Difference uses imperial units and is used for any column that finds the difference between two columns or the change of one column. Metric uses metric units, typically used with weather data. Metric differences uses metric units, typically used with weather data.
9. Enter the type of units that should be used. Temperature, current, frequency, pressure, height, degrees, percent, speed, or other for custom unit types
10. If other is chosen, input the custom unit for the column
11. Verify that the units are correct.
12. Data is saved to setOptions.json

**\*\* Only available for Version 2.0**

**Potential Future Improvements**

**Using Weather Data**

* Dynamically apply boiler settings based on weather (ie. If temp = 90 °F, then boiler off)
* Add more tests using weather data

**Integrating With NURO Connect**

* Add button in NURO Connect sidebar to open NURO Analysis

**Creating Mobile App**

* All sizing and functions work with mobile; make mobile app version available

**Efficiency**

* To speed up Available Boilers page load, create dropdown to select site to reduce amount of API

**Adding Modify Menu to modifyTests.py**

* A modify menu would eliminate the need to manually edit the JSON file
* Function Ideas: change title, change threshold label, change warning value, change error value, change dataset label, change axis label, change duration, swap left and right axes, hide dataset threshold, renumber tests

**Test Error Checking**

* Currently, the error checking for individual test execution failures does not fully remove the graph. It is replaced by the next test’s graph. This is not a problem currently because it only occurs if the weather data cannot be retrieved, and that graph is last in the list.

**Commands and Terminology**

**Command Line**:

* Change Directory/Folder: cd folderName
* Execute Python Code: python filename
* List Files/Folders in Current Directory: dir (Windows), ls (Linux)
* Install Library: pip install libraryName

**Git Commands**

* Cloning GitHub Repository: git clone GitHubURL
* Add File/Folder To Repository: git add fileName
* Save Code Changes To Repository: git push

**Terminology**:

* Directory: folder
* Current Working Directory: the folder you are currently in
* Root Directory: folder where all the app’s code is stored

**Resources**

**Virtual Environment**: <https://stackoverflow.com/questions/48911582/virtualenv-to-path-on-windows-10>

**Running Flask Apps Locally**: <https://stackoverflow.com/questions/17309889/how-to-debug-a-flask-app>

**Windows Command Line**: <https://serverspace.us/support/help/windows-cmd-commands-cheat-sheet/>

**Deploying to Azure**: <https://learn.microsoft.com/en-us/azure/app-service/quickstart-python?tabs=flask%2Cwindows%2Cazure-portal%2Clocal-git-deploy%2Cdeploy-instructions-azportal%2Cterminal-powershell%2Cdeploy-instructions-zip-azcli>

**Bootstrap 5 Documentation**: <https://getbootstrap.com/docs/5.0/getting-started/introduction/>

**Meteostat Python Library**: <https://dev.meteostat.net/python/>

**MapQuest Geocoding API**: [https://developer.mapquest.com/documentation/ geocoding-api](https://developer.mapquest.com/documentation/%20geocoding-api)

**Visual Crossing Weather API**: <https://www.visualcrossing.com/resources/documentation/> weather-api/timeline-weather-api/

**Tomorrow.io Weather API**: <https://docs.tomorrow.io/reference/historical-overview/>

**Degree Days Information**: <https://www.degreedays.net/calculation>