# Executive Summary

# List of Abbreviations

# Introduction

An environmental study requires data. The data depends solely on the typoe of study that is been carried out and the resolution of the data neededfor the research. Many times, a sresearcher has problems just in acquiring data; the amount of sources that can be found online can become overwhelming. Furthermore, once the data has been obtained, it requires preparation that has takes time and efforts from the use rpersepctive. Hydrologic libraries have been used in the scientific communities to simplify the workload that a researcher most due to obtain basic and reliable results. A complete hydrological analysis is done considering

# Statement of Problem

# Purpose of the Study

* Common web browsers have used Javascript since 1990 as the first programming language that was generalized for common usage.
* Similar ideas have been developed through research institutions or open source communities.

# Objectives

Hydrolang uses client-side technology to perform different routines which aim towards the retrieval of data and perform basic analyzes from a data and environmental perspective. This has been done by creating a series of components within the framework working as a library in which the user selects a particular function for data manipulation. The main structure of the Hydrolang is described in more details on the methodology section of this thesis.

The components defined and created within the library follow the next structure:

* Data
* Analyze
* Maps
* Visualization

The data module has been written so that it can retrieve environmental data from open sources such as governmental agencies, free API’s, etc. which solely depend on the availability of information and the final aim of the user. The data is obtained, manipulated and output through functionalities within the data module.

The analyze module contains two different main functions: stats and hydro. Stats submodule contains different tools that process data and performs statistical analyzes which provide a cleanup version of the data as required from the user. The hydro submodule contains subroutines that build up together a complete hydrological analysis. This includes features that work within the framework established by renown authors and community accepted practices.

The maps module consist of mapping tools that allow the user to include layers of data obtained from previous modules or owned by the user (provided they are on the same format) so that it can be included within a map. The layers can be flooding extents, markers or layer of analyzed data. Finally, the visualization module provides the user with charts, reports and other visual tools that aid the user to better analyze the obtained data from previous modules or previously owned.

The modules can be accessed using the chaining properties of high-level modules in JavaScript, and after the functions have been used, they can be saved on array-like objects.

# Definition of terms

* Standardize the use of client, user, developer.
* API: application program interface.
* External library (JavaScript): a library of functions that have been created by third parties which aim to provide with standardized and optimized functionalities for specific problems.
* Lazy-loading: asynchronous loading of interfaces such as images, websites, external libraries, etc.
* EcmaScript: commonly used international agreements for client-side code development of the World Wide Web.
* CDN: content distribution network, used for calling external libraries dynamically.
* REST: acronym for Representational State Transfer; an architectural style for distribution of hypermedia systems introduced by (Fielding, University of California,).
* AEMET: Spanish Meteorological Agency (Agencia Estatal de Meteorologia).
* CORS: acronym for cross-origin requests in browser technology.
* EAUK: Environmental Agency of the United Kingdom.
* FEMA: United States Federal Emergency Management Agency.
* METEOIT: Italian meteorological network.
* Object (JavaScript Environment): available variables, functions, definitions on the language. It is the main idea for the language’s development (object-based paradigm).
* NOAA: National Oceanic and Atmospheric Administration of the United States.
* USGS: United States Geological Service.
* DWD: Deutscher Wetterdiesnt (German Weather).

# Literature review

The library was developed fully using JavaScript. Since its development in the 90’s, the implementation and standardization of the language on the most commonly used web browsers and the different amounts of libraries and frameworks that deal with more responsive and adaptive web technologies create a suitable environment for the scope of the project. Specifically, dealing with a prototypical document object model which define first classes as functions as well (Punchatz, 2017).

The research question that was used to find similar approaches to what Hydrolang was to become was the following:

*“Which applications are available on the literature and on the web with respects to open source, web based environmental, hydrologic and hydraulic analyzes?”*

To answer the latter previous to the development of the project, the following resources were used:

* GitHub
* Science Direct
* Google Scholar
* Research Gate
* Google

From the results, NUMBER were papers and projects that used JavaScript; NUMBER used Java; NUMBER used Python and finally NUMBER used PHP or any other language. Overall, the papers can be categorized on the type of results that are obtained which are based mainly on environmental data, data retrieval from environmental agencies, visualizations applications that used Google Maps or other web services for deployment and statistical sorting and manipulation of data.

# Background

The library has been developed purely on JavaScript. The language has been used for web technologies since the 90’s and, because of this reason, became the first standardized programming language for common web development.

Object oriented architecture has been defined as:

“*the methodology for creating software components in the form of reusable libraries exclusively restrained to a specific domain ontology*”

Different approaches have been taken to create library based analytical tools for research and education purposes.

API’s and data

The basics on data retrieval

Downloading using API interface requests.

Statistics and cleaning

Hydrology

The watyer scycle and the physical processes involved vary greatly depending

Unit hydrograph from gauged and ungauged basins. Area of basin.

Rainfall

Neural networks have been used in hydrology for solving nonlinear problems such as rainfall runoff, groundwater applications and rainfall distributions. Regarding the latter, numerous papers have shown the advantages that using NN models have on prediction of patters and distribution. In comparison to the stochastic distributions.

Neural Networks

The main constraint that the implementation of neural network has is the amount of data that is required for training a model. This issue had been a setback in the early years of artificial intelligence. The reliability of the data that is to be used for both feeder and result has a direct effect over the trained model (Vemuri, 1993). The number of training sets for both input and output of models will showcase the percentage of performance that the trained model will have over new datasets. This comes with a toll, of course on the performance of the environment in which the model has been developed. Nowadays, with the advancements in web technologies, browser applications that are being actively maintained, the usage of a native environment is becoming less necessary. The discussion done by (Ma, et al., 2019) showcases the capabilities that running deep learning on the browser

# Design Methodology

## Requirements

The idea for creating Hydrolang comes from the advantages of running code directly on the client side and explore the extent of what can be done. Specifically, considering that:

* It allows for a quicker and immediate interaction between the user and the library’s functionalities.
* A faster execution time because there are no delays from contacting external servers (except for data retrieval).
* The current usability of web browsers improves using client-side approaches and are up to date with the common programming standards..
* The code runs using the technology that the user has within his/her environment, mainly using the CPU and if required, a more dedicated core.

The efficiency of the code depends solely on the programming approach that has been taken for a given task. The better the algorithm that has been written, the less dependencies and connections between each component and a better performance.

Components

The modules created on Hydrolang are aiming to provide the following:

* Be able to understand, retrieve and transform data in several formats from external sources sources including public and private data.
* Data comprehension using relevant statistical metrics to assess the reliability of the data and conclude if any further actions are required to achieve the latter.
* Hydrological analyzes that vary depending on what the user requires from the library. This includes precipitation, discharge and hydrodynamic routing.
* Visualizing and mapping of data so that the user can have a clearer image of what has been achieved through the library or from external files and what else might be needed for further improvement.

These components have been created accordingly through the applicable literature.

## Architecture

Using a library-oriented architecture, Hydrolang has been developed to explore the functionalities that the dynamic coding style of Javascript has, considering the principles and methodologies of the ontology domain of reusable software libraries. By establishing the latter, functions are accessed using inheritance from classes and export functions as established on ECMAScript 6, in which the syntactic sugar has been reduced and chainage is achieved through the usage of basic dot notation (Engelschall, 2017). Instantiation was not considered because by using classes, a cleaner and responsive code was obtained. Thus, a decision was made to work within the library creating code wrapping of objects to access the functionalities on lower levels. A modular approach was then established which allows for the protection of the execution context of the given modules but enables the usage of lower level functions. In general, the following is the modular tree which the library follows:

* Object mutability. “Objects inherit from objects. What could be more object oriented than that?”
* Inheritance from classes has been used using the ECMA6 version, instead of creating prototype which would require for doing instantiation for each function that is required to be used.
* Most of the functionalities within each library can be fed using by the creation of Javascript objects or arrays. This eases the communication for each module and the client.
* Inheritance in JavaScript is covered by delegation automatism bound the prototype property of constructor functions.
* Important to mention within the review for JavaScript development: event loop and how the stack works for working with different functions within a class.

The code was developed in Javascript, without the usage of any framework. The access of the been created using

Prototypical objects were not considering when developing the library. Instead, object oriented implementation was done

All of the modules are build depending on the level of changing in which they are located.

## External libraries

To achieve a cleaner, concise and version compliant code, different libraries have been used within the framework to create new modules or components. The libraries that have been used (but not limited to):

* JQuery.js
* Google charts API
* Google maps API
* Leaflet.js
* D3.js
* Tensorflow.js
* Documentation.js

JQuery is a JavaScript external library that creates a more compact version for requests and object manipulation in either synchronous or asynchronous environments for HTML and JavaScript (JQuery, 2020). It is mostly used for creating HTML documents and webpages enabling traverse manipulation, event handling and implementation of features for requests using Ajax. The library is very handy regarding data retrieval because of its in-built functionalities and ease of use of calls and requests. It can handle any type of available requests for different types of API’s and databases and allows for cross referencing manipulation.

Google charts is a powerful tool developed by google that enables software developers to use different charting modalities on webpages. The library contains a variety of options that are customizable as required from the developer. The API can be used free of charge and enables the user for a dynamic view of the data due to the protocols and tools implemented within (Google, 2020).

D3 is library that is mainly focused on manipulation of documents that are driven by data. The library allows the user for data binding using a Document Object Module and later apply data driven transformation to the document. The library solves data manipulation by applying the most relevant and applicable algorithms that are dependent on the type of data that is to be handled. Just as with all the other libraries, it can be accessed using an API that is called whenever the functions are required (Bostock, 2020).

Google maps is a service by google which provides information about geographical regions and sites around the world. Its API has become the most popular library for map generation. It has always been an open source library but has changed since 2014 (CHECK), then changed the application to bill the application depending on the amount of calls it on a monthly basis. Nevertheless, because of its ease of implementation, use and flexibility for developers, it is widely used for web page creators, content creators, etc. The API has been written completely in JavaScript and it can be paired with any type of database to load data (Google Maps Platform, 2020).

Leaflet is an open source interactive JavaScript library for map generation and rendering. It has been developed by Vladimir Agafonking and out to the public since 2010 (Agafonking, 2020). Among its functionalities , it provides with maps tailored to options that can be using the library are tile layers, markers, vector layers, image overlays, geoJSON files, interaction features, map control, etc. It can run on most web browsers for either desktop or mobile applications. Because it is lightweight, can be used by CDN calls and no need for external dependencies, plugin creation mainly lead by the community and, the library has been a choice over other map visualizers. Overall, it is a very well documented API and can be coupled with other libraries for creating beautified rendering styles.

Tensorflow is a platform dedicated to machine learning applications. It has been used for creating neural networks models for different programming languages and different approaches. The platform was created by google and it has a simple and flexible architecture that allows to create, train and compile neural network models easily through its high-level APIs. Different options have been implemented by the platform to include different options for models such as recursive algorithms, feed forward networks, convolutional cores, etc. It can also run on both the browser and if needed/required, a GPU (Tensorflow, 2020).

To generate the documentation from code, all of the features in framework have used the script tags comments compliant with documentation.js. It is a framework developed based upon Doc.js which reads through the code for comments that have /\*\*\*/ format and generates documentation from that is later published for usage. It is an open source library that has been used since 2015 and is continuously upgraded to adapt to new programming styles (Dcoumentation.js, 2020)

## Data sources

The data that is being retrieved using Hydrolang comes from APIs developed by governmental or open source institutions. Most of the data has been filtered and cleaned up before it is hosted within the databases, but this limitation is strongly linked to the type of data and the location from which the data is to be retrieved.

The data sources that have been included within the library have been selected because of the varied information that can be retrieved from them, the locations for which they can be obtained and the cross referencing (validation) of the data. Overall, 9 sources (PLEASE EXPAND THIS) have been included within the library and each of the source is explained below.

All of the data sources provide endpoints which retrieve different information depending on what is being required by the user. The data can be extensive and overwhelming, which becomes unhelpful if not filtered properly. Considering that the library deals mainly with environmental information, specifically water-related data, the final endpoints selected for usage considers the latter including endpoints that obtain water-related data. Nonetheless, it is important to mention that the endpoints within the library are just an example usage for hydrological sciences and the idea can be extended to other scientific fields. Each endpoint described below have the following format:

export default {

stations: {

endpoint:

"https://some.url.com/endpointname",

params: {

name1: param1,

name2: param2

},

}

requirements: {

needProxy: true,

requireskey: true,

keyname: "keyname",

method: "GET",

},

}

Snippet . Example code for source usage.

AEMET

Aemet developed an open REST API that allows for the distribution and reutilization of meteorological and climatologic information from Spain and surrounding stations. The API provides data about observation data, forecasts, climatologic values, satellite information, maps and graphs, maritime predictions and observations, ultraviolet readings, radars and others. The API requires for the generation of a key to be passed into the header of the request, along with a “GET” method. The API also requires a proxy server to avoid CORS violation. NEED TO ADD SOURCES.

The endpoints selected to be used within the library are the following:

* Station data (“stations”): conventional data gathering.
* Daily station data (“daily-stations”): retrieves precipitation data for a given station by passing in initial and final observation dates.

EAUK

The EAUK created a hydrological REST API which gives access to historical water information from the UK and surrounding areas. The API is on its alpha state and as of August 2020 it provides information regarding flood monitoring, hydrological data, environmental metrics, etc (Environmental Agency UK, 2020). The API is available freely for the public without the need of a key or any other parameter. The measurements that are provided by the source depend solely on the amount of available measurement stations that have are being considered. Thus, for some locations, the data is unavailable until a certain date and further manipulation of the data might be considered for gap filling or outlier removal. The retrieved data can be in JSON, CSV and XML format and uses a “GET” method for request and it does not require a proxy server. The library also provides geoJSON files.

For each request, added metadata is appended to the first lines of the retrieved object. Due to the available data within the application and the ease of use of the source, the following endpoints were considered for the library:

* Flood warnings (“flood-warnings”): considers instantaneous and historical flood warnings within a certain county or coordinates within the UK. It retrieves the data depending on the type of severity of the warning.
* Flooding prone areas (“flood-areas”): instantaneous and historical areas prone to flooding depending on the coordinates and search strings.
* Flooding prone stations (“flood-stations”): retrieves stations that have been historically cataloged for flooding areas. The endpoint have as parameter the town location, catchment name, etc.
* List stations (“list-stations”): retrieves the stations that are available within a certain location by searching for towns, counties or coordinates.
* Data stations (“data-stations”): finds historic rainfall information for a given rainfall station depending on the location, start and end date, station reference.

FEMA

In accordance to open data initiatives, FEMA developed a read only RESTful API that uses query string parameters to manage query selection. It includes datasets for major disaster declarations across the USA territories sorted depending on city, county, state or nationwide (FEMA, 2020). The API has different operations that can be built depending on what the request for the end user is in the forms of logical operators (i.e. equal, not equal, less than, logical negation, etc.). The formats that the API rpvoide are json or bson and allows for retrieving data through callback functions without the need of a key. The API accepts the “GET” method for retrieval and CORS has been implemented through the API, meaning that it does not require a proxy server.

The requests are appended with the information that has been requested by the user along with metadata. For the library, the following endpoints have been considered:

* Disaster declarations (“disaster-declarations”): it accepts as data fields number of disaster, the county or state; the type of incident and declaration title among others.

METEOIT

METEOIT API contains meteorological data by giving access through different endpoints from the Italian territories. It is a RESTful API which has been developed for obtaining information mainly completely bound to the meteorological stations that are found within the Italian land. The service requires for the creation of an account and the generation of a unique token that must be included within the header of the request. Because of the lack of implementation of CORS on the API, it requires a proxy server to be present with the request, accepting only “POST” methods for retrieval.

The API has been developed since (PLEASE SEARCH FOR DATE) and ever since has deployed 3 different versions. Because of this, a new key is needed for accessing the last version, while the other 2 version have been deprecated but can still be accessed.

For the usage of the library, the following endpoints have been included:

* Daily station data (“station-daily”): retrieves meteorological information for a specific station during the last day previous to the request. It accepts as parameter the station code, the observation data and data quality. This last one is a parameter that can be added to every endpoint to enable the API to fill data in case there is missing.
* Last dates data in station (“stations-lastdays”): it retrieves data for specific data for a single station. This endpoint is on the 2nd version of the API. It requires the station code, the data type for the request and the validity of the station.
* Data per single station per date (“station-singledate”): it retrieves data for multiple stations on a single date. It requires as parameters date and station validity.
* Data per multiple stations per single date (“stations-singledate”): it retrieves the data from multiple stations for a single date on nearby countries. It requires as parameters the date, country code and station validity.
* Nearby stations (“nearstations”): it retrieves data from nearby stations to a query from longitude and latitude. It requires as parameters the coordinates, range of search and station validity.

METEOSTAT

METEOSTAT is a historical weather API done by private developers that use its own climate model to create projections and statistics of single weather stations on geographical points that are used as searching queries. This model allows the API to obtain data from several places throughout the world and guarantees is always up-to-date (METEOSTAT, 2020). The databases from the API has weather stations that regularly report observations and statistics; it also retrieves historical data from stations that are being provided by governmental organizations like NOAA, Deutscher Wetterdienst and Environment Canada. The information that is downloaded through the API contains all of the variables that a certain weather station measure.

A very relevant feature of the source is that it is completely free and allows for the connection to historical data. The developers have also included a bulk data endpoint in case larges chunks of information are required by the user. The API requires a proxy server and a key to obtain the data and uses the “GET” method for retrieving information.

As part of the framework, the following endpoints from METEOSTAT were included:

* Find stations (“find-stations”): through query and limit as parameters, it obtains the stations that are within a country or basin.
* Nearby station(“nearby-stations”): it retrieves the information about stations that are within certain coordinates. The parameters for the endpoint at latitude, longitude, limit and radius.
* Hourly data per station (“hourlydata-stations”): obtains the hourly data from a specific station. It requires as parameter the station ID, the start and end for the query and the model for retrieval.
* Daily data per station (“dailydata-station”): it retrieves daily data from a single weather station given as parameters the station ID, and the date range.
* Hourly data point (“hourlydata-point”): an experimental version inside the API, it allows the user to retrieve data from any given station in hourly intervals. It requires as parameters the coordinates of a certain location, the start and end dates. It allows for 10 days per request.
* Daily data point (“”dailydata-point”): another experimental endpoint, it allows h user to obtain daily information about a specific point in the world by passing by as parameters the coordinates and start and end dates. It allows only 370 days per request.
* Bulk data (“bulkdata”): it allows the user to retrieve information as bulk in gz format. The parameters vary depending on what is required by the end user.

NOAA

NOAA’s API provides through its Climate Data Online (CDO) server access to current data for users. The REST API provides different types of datasets, data categories, types, locations, stations and data depending on what is required by the end user. The coverage of data will depend on the minimum date available of the data as well as the overall technology used for data retrieval (for instance, radar data).

The API does not require a proxy server for access, but it does require an access key that must be included in the header of the request; it uses the “GET” method for retrieval. For the library, the following endpoints were considered:

* Datasets (“datasets”): retrieves the datasets that are available on the NOAA database system. It does not have any parameters.
* Available stations (“availablestations”): retrieves the available stations within a certain location, extent or coordinate within the US territory.
* Precipitation every 15 min (“prec-15min”): retrieves precipitation of a certain location every 15 min for a given period of time. The parameters it accepts are location id, station id, start and end date, units, limit, include metadata.
* Hourly precipitation (“prec-hourly”): retrieves the information from a certain location or station. It requires as parameters location id, station id, start and end date, units, limits, offset.

USGS

USGS REST API services provides several web services that are separated depending on the type of information that is required. These services are separated from instantaneous values, site services, daily values, water quality services, groundwater levels and statistics. Each of the services is access through its own endpoint which contains as parameters data types accepted. All of the sites allow of water data retrieval from thousands of sites that are managed and/or monitored by the USGS throughout the US.

The API requires a proxy server to be accessed but it does not require a key and it uses the “GET” method for retrieving the data.

For the purpose of the framework, the following endpoints were included:

* Instant values (“instant-values”): retrieves instantaneous information from a specific location. The parameters accepted are format, site, stat ID, county ID, start and end date.
* Daily value (“daily-values”): retrieves daily values of a certain location. The parameters accepted are format, site, stat ID, start and end date.
* NEED TO WRITE MORE.

World Bank

The world bank has a REST API interface available for retrieving historical and forecasting data from every country in the world, with the option of going deeper into basin and sub basin areas upon request. The API is based on the derivation and extrapolation of 15 global circulation models, which are used by the Intergovernmental Panel on Climate Change reports. The data within the API are modeled estimates of both temperature and precipitation along with back casting of the data. It is important to notice that the information that the API delivers are based on the more on the models and not as much as in instrumental observed data, this because the difficulty of obtaining data specially during the early 1900s.

The data options within the API are averages for months, years and monthly anomalies and can deliver both precipitation and temperature. The dates that are available have been separated as past and future, from 1920-1999 and 2020-2099 respectively. The requests also can be sent by identifying basin id’s, which is a number varying from 1 to 468 representing regional river basins throughout the world. The responses comes in forms of xml, csv or json and are limited by the anomaly data types, the scenarios for future time periods and the type of ensemble within the request. The API requires a proxy server to be accessed but it does not require a key and it uses the “GET” method for retrieval.

For the purpose of the framework, the following endpoints were included:

* Monthly averages per country (“monavgs-country”): retrieves the monthly averages of precipitation per country.
* Annual averages per country (“annualavgs-country”): retrieves annual averages of precipitation per country.
* Monthly averages per basin (“monavgs-basin”): retrieves monthly averages per basin.
* Annual averages per basin (“annualavgs-basin”): retrieves annual averages per basin.
* Daily precipitation per country (“dailyprec-country”): retrieves daily precipitation per country.
* Daily precipitation per basin (“dailyprec-basin”): retrieves daily precipitation within a basin.

# User interface

Complementing with the current technologies, Hydrolang has been designed using the most current version of JavaScript coupled with HTML and CSS to create a basic and robust user interface.

# Implementation

## Modules

The library was developed mainly on the interoperability of the modules which enables the user to use all the functionalities from the components independently, removing possible dependency and inheritance issues. This means that the user is able to use data retrieved using the functionalities on the library or use external files to perform statistical, hydrological or visual analyzes on data. Each component has dependencies that differ on the functions accessed within as well as the usage of different frameworks.

The functions inside each component have been defined as main and helper functions. The access of each function will depend on the level of chaining in which it is located. For modules with components, the function is located on the fourth level and are called on as static methods inside a class. Whilst functions located on third level are called as exports functions.

hydro[n].module.component.function(arg)

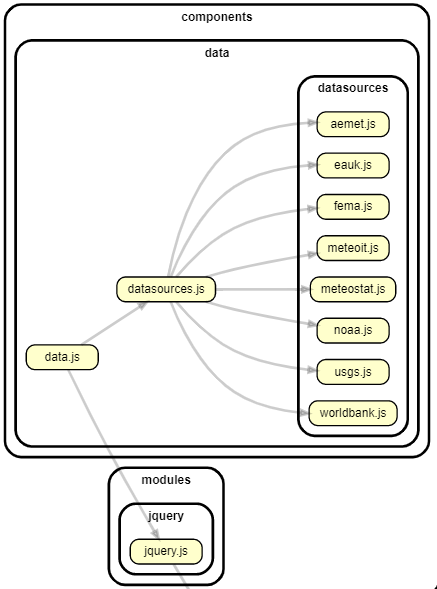
hydro[n].module.function(arg)

## Data

### Functionalities

The data component is subdivided in the data module which handles the data queries coming as inputs from the user; datasources component which is used to export the source parameters from the data components and the data sources providers which contains JSON object examples that contain the necessary query information for the creation of a query.

The dependencies of the module described in the following diagram.



The data module contains composed of a retrieve function using as parameters a callback function and a configuration object which contains the source of the data, the type of data that is required, any arguments that the source needs and the type of data (usually JSON format). After the function has initialized, it checks if the source that has been passed in the configuration is in the data sources database of the library. If the source does not exists, it returns : “No data has been found for the given specifications.” If the source exists, the function checks if the data requires a proxy server or not. This is read from within the data sources specifications (for more information, please see the data sources AAAA). If the source requires a proxy, then the user most has have it specified within the configuration object; if that is not the case the function returns: “info: please verify if the resource needs a proxy server.”

Afterwards, the retrieve function verifies the type of headers that each API requires, just as explained on data source. It also checks if the data source requires a key. This information should also be included within the configuration object passed by the user. If the object fails to provide this information, returns: ”info: please verify the keyname of the source.”

If all of the above has been fulfilled, then the callback function uses JQuery Ajax function to retrieve the data given the url (including proxy and endpoint from the data sources), the data required, the type of data, method of retrieval (some sources require either “GET” or “POST” requests) and the header applicable to the source. After the data is retrieved, it is logged into the console of the browser for it to be saved as a temporal variable by the user and postprocessed with the following functions.

The transform function is also included inside the data.js file, which uses objects retrieved from the previous function to transform it according to what the user requires. After the data is retrieved, it is saved as a temporal object which is later passed to a configuration object which is passed into the function. In order to guarantee that the user obtains only the most relevant information, the configuration object must state which are the columns that are to be saved from the retrieved object. An example of function usage is given in the following snippet.

PUT SNIPPET OF CODE THAT SHOWS HOW TO CALL THE TRANSFORM FUNCTION

After the data cleanup, the object is firstly transformed into a JSON file to be later changed into either CSV, XML, JSON or array object; the last one can be used to continue with the analysis of the data using the tools within the library. The function can also be used data that the user has in JSON format to the formats previously mentioned.

To download the data that has been retrieved and cleaned up, a data download function has been created which extends the transformation function. It creates a temporal blob of transformed data appended to the html file which runs the library. To initialize the function, a configuration object must be created with similar specifications to the transformation function: the data that is to be saved and the format of the data.

For simplicity on the algorithms used within the library, data has been handled independently as arrays for each format inside an object or array. This means, in an array containing dates and numbers, the format of the object would be a 2D array: the first array containing dates and the second array containing numbers.

Most of the sources provide the data retrieved in format of a javascript array of JSON objects. For this, a transform function has been created to cleanup the data depending on what the user wants to keep. After the user passes

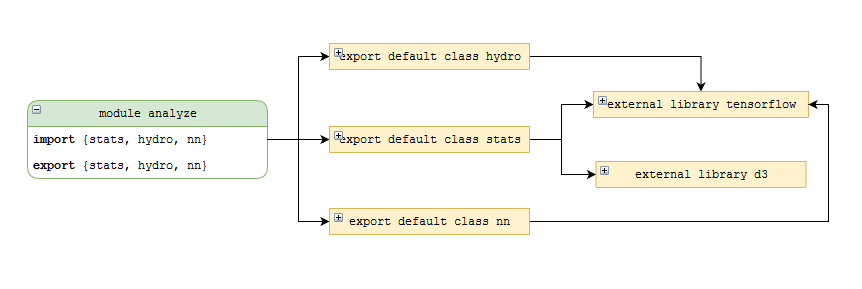
Use example

The data module is located on the 3rd level of the chaining, right after the declaration of a new Hydrolang object from the functions in the second level. the following is an example of the implementation of the data module. Notice that the user must create an object that is adaptable to the data retrieval sources that Hydrolang supports.

The object created for retrieving data will yield a raw object file that will be on the available format that the source can provide. Simultaneously, the user can use the data transformation function to change the data into the formats that the library supports (CSV, JSON, XML, ASCII).

## Analyze

The module was developed with three components that are separated by the final usage required: basic hydrological calculations, statistical analyzes or neural network model development. The module feeds from each of the components and are exported into the main hydro instance to be called when required using chainage.



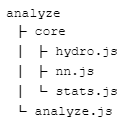


Figure . Dependency and tree diagrams for analyze module.

### Stats

Stats is a component part of the analyze module which has tools to perform statistics sorting and cleaning of data.

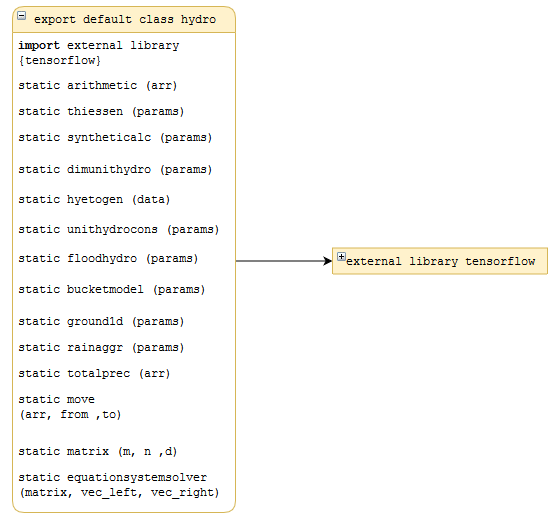
Table . Functions within stats component.

|  |  |  |  |
| --- | --- | --- | --- |
| Module | Component | Function | Summary |
| Analyze | stats | copydata | Makes a deep copy of a dataset. |
|  |  | onearray | Returns the data of a time series 2D array. |
|  |  | datagaps | Identifies gaps in data. Can be fed with time series 2D array or just the data of a time series. |
|  |  | gapremoval | Removes gaps from data. |
|  |  | sum | Sums all data in an array. |
|  |  | mean | Calculates mean of an array. |
|  |  | stddev | Calculates standard deviation of an array. |
|  |  | sumsqrd | Calculates sum of squares of an array. |
|  |  | min | Calculates minimum value of an array. |
|  |  | max | Calculates maximum value of a dataset. |
|  |  | unique | Determine unique values within a dataset. |
|  |  | standardize | Standardize dataset given mean and standard deviation. |
|  |  | quantile | Calculates a given quantile of a dataset. |
|  |  | cleaner | Filters out items in an array that are undefined, NaN, null. |
|  |  | outremove | Removes outliers on a dataset. Requires the identification type (either quantile or normalized). If p1 or p2 are not given, then the default values fo either method will be selected. |
|  |  | itemfilter | Filters out items in an array based on another array. |
|  |  | correlation | Calculates the Pearson coefficient of bivariate data. |
|  |  | variance | Calculates the variance of a dataset. |
|  |  | frequency | Determines the frequency of a dataset. |
|  |  | interoutliers | Calculates the outliers of a dataset using interquartile method. By default, 25 and 75 quantiles are used. |
|  |  | normoutliers | Calculates the outliers of a dataset using the normalized data method. By default, -0.5 and 0.5 are used. |
|  |  | fastfourier | Calculates fast fourier analysis on a dataset to study seasonality effects. |
|  |  | basicstats | Generates an object with basic statistics for a dataset to be ploted using google charts. |
|  |  | joinarray | Joins arrays of data. |
|  |  | numerise | Transforms arrays of data from strings to number. |
|  |  | flatenise | Flatens a dataset |
|  |  | dateparser | Parse data to seconds since 1970 for data manipulation. |
|  |  | arrchange | Interchanges a m x n matrix to n x m. |
|  |  | push | Pushes one array into another array in the end. |
|  |  | copydata | Makes a deep copy of a dataset. |
|  |  | onearray | Returns the data of a time series 2D array. |
|  |  | datagaps | Identifies gaps in data. Can be fed with time series 2D array or just the data of a time series. |

The module also contains a regression analyzer that foresees the correlation that exists between different types of datasets, if needed. For instance, the correlation that exists between air temperature vs evaporation. A regression analyzer has also been included to develop relations between seasonal patterns. In order to aid with the development of the module, the library D3 was used alongside with jStats. Imported as the standalone version, they were used for cross referencing of the results and verification. Overall, some issues were found when using D3 for some of the datatypes that the function can handle, which were overridden by the usage of jStat.

### Hydro

The hydro component contains several hydrological components that aim to retrieve data for the user depending on the level required. This means that the user is able to create datasets within the functionalities without any data; is able to use the data retrieval tools and use that data within the module or is able to import data from a another directory.



|  |  |  |  |
| --- | --- | --- | --- |
| Module | Component | Function | Summary |
| Analyze | hydro | totalprec | Calculates summation of total precipitation of a given event. Rainfall must be evenly distributed and fed as a 1D array. |
|  |  | arithmetic | Calculates arithmetic average for different rain gauges for the same event. Rainfall must be evenly distributed and fed as multiple 1D arrays inside an object with same rainfall period. |
|  |  | thiessen | Calculates pondered average for different rain gauges for a same event given subbasin areas. Rainfall must be evenly distributed through the area. The passed object should contain 1D array with areas and an array of arrays with precipitation with same rainfall period. |
|  |  | bucketmodel | Calculates rainfall-runoff analyses of an event given parameters as evaporation, baseflow, land uses. The rainfall event should be a time series in which the time variable should be either a valid time string type or number. |
|  |  | unithydrocons | Generates a unit hydrograph from both a synthetic rainfall, rainfall event and/or discharge readings. Object should include either of them and state which type of constructor should be called. |
|  |  | floodhydro | Calculates the flooding hydrograph given a U.H. and the physical characteristics of a basin. Object passed to the function should include rainfall event (time in either string or number formats), unit hydrograph (time in either string or unit format), CN number, storm duraation and required timestep. |
|  |  | dimunithydro | Generates a dimensionless unit hydrograph from the physical characterstics of a basin.Object should include timestep, number of hours of event and peak rate flow. |
|  |  | ground1d | Calculates 1D groundwater flow/head steady transport using gaussian elimination. |
|  |  | rainaggr | Calculates both rainfall aggregation and disaggregation. Object should include the aggregation type (agg or disagg), the rainfall event (time as string or object). |
|  |  | move | Moves part of an array into another. Arguments should include the array and locations. |
|  |  | matrix | Creates an m x n matrix filled with 0s or any other number. |
|  |  | equationsystemsolver | Solves linear system in the form Ax = b. Requires a matrix, left vector and right vectors as arguments. |

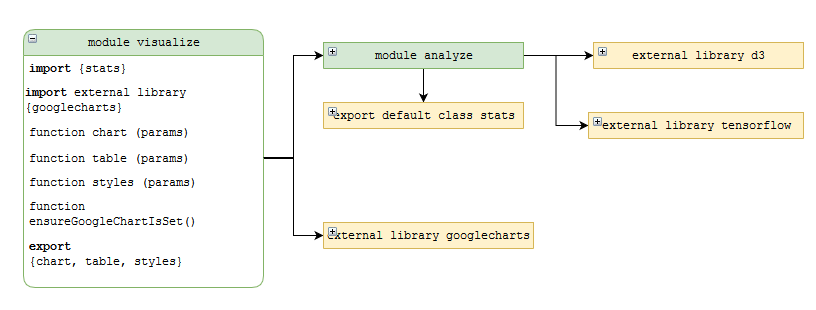
NN (Neural Networks)

The NN component uses the well-known library for artificial intelligence TensorFlow and allows for the creation of different types of neural network models that can be trained, saved and reused. The module included within Hydrolang creates sequential algorithm models, in which the inputs are fed directly to the neurons, with preestablished activation functions. The number of inputs, neurons and outputs are fed to the model by the user depending on the final outcome needed. It creates a 1 layer of neurons, with the number of neurons to be chosen by the user as well as the number of outputs. Consistency of the module needs to be considered so that there are no crashes when creating and training.

|  |  |  |  |
| --- | --- | --- | --- |
| Module | Component | Function | Summary |
| Analyze | NN | createModel | Creates a new sequential neural network model |
|  |  | convertToTensor | Grabs array type data and converts them to tensorflow tensors. |
|  |  | trainModel | Trains an exisiting model with already created tensors. |
|  |  | prediction | Based on a trained model, predict data. |

## Visualize

The visualize module was created as an aid for data visualization using common library such as google charts.

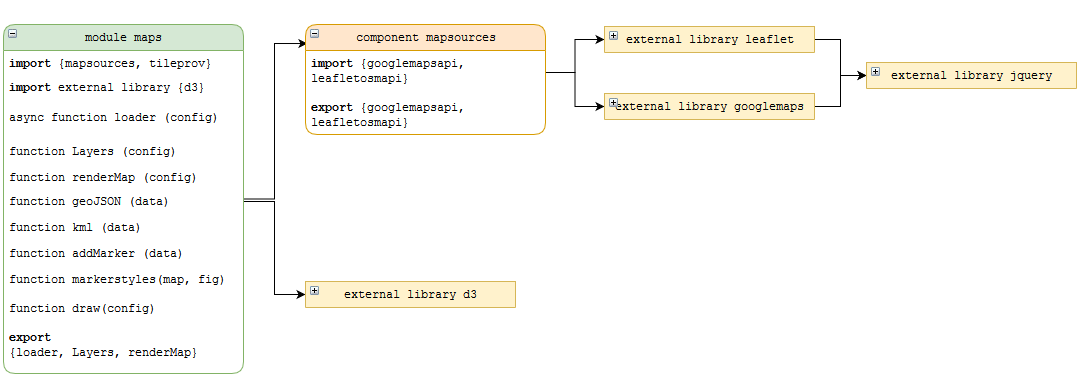


The visualize module calls in the API developed by google for the generation of charts and tables. It is used as a side component for the visualization of data in the analyze module. The functionalities have been implemented using an asynchronous library for google charts that has been included as a component within the framework.

Table . Functions in the visualize module.

|  |  |  |  |
| --- | --- | --- | --- |
| Module | Component | Function | Summary |
| Visualize | visualize | chart | Draws data as a chart depending on the options passed by the user. The data must be in array format. Options available are scatter, column, line, timeline and histograms. |
|  |  | table | Draws a table from data provided by the user. The data must include a header and passed as an array. |
|  |  | styles | Contains different styles available to use by feeding fewer arguments than by directly using the chart and table functions. The drawing styles are meant just as options in case the user does not want to create a style object. |
|  |  | ensureGoogleChartIsSet | Promise for calling the google charts component. Calls in all the charts and tables that are rendered in the library. |

## Maps



The map module was created with the idea that the user will be able to use a map visualization for selection of stations, render geodata files and select from screen locations to generate new queries for data retrieval.

The module uses two different map providers: leaflet and google maps. As explained before, both functionalities provide a map visualization that is added to an HTML page as a divisor—or div—appended to the header or body of the page. The different between each of the map providers is the usage. Because of this, during the development of the module more emphasis was done to add free source functionalities.

The map module contains 7 functions that are explained on the following table. The main functions used for calling in the

Table . Functions of map module.

|  |  |  |  |
| --- | --- | --- | --- |
| Module | Component | Function | Summary |
| Map | maps | loader | Loads maps from leaflet or google. |
|  |  | Layers | Add layers of data for both google or leaflet or both geodata, kml, drawing tools, markers or remove layers. Main function used to draw new layers into an existing map. |
|  |  | renderMap | Appends a map loaded into a div located on the header of an html page. The loader function must be defined first before to load the necessary libraries. |
|  |  | geoJSON | geoJSON function for rendering georeferenced layers into screen. Used through Layers function. |
|  |  | kml | Creates a layer of kml data passed through an object to an exisiting map. Used through Layers function. |
|  |  | Marker | Creates a new maker into an existing map given the coordinates. Used through Layers function. |
|  |  | markerstyles | Changes the style of a marker depending on the marker shape itself. Used through Marker function. |
|  |  | draw | Appends a drawing tool on a map. Can create similar shapes as in markers and layers can be saved. Used through Layers function. |

Case study: rainfall disaggregation using neural networks

Description

As part of a feature within the library, a neural network model has been developed for analyzing rainfall disaggregation patterns from 1 hour to 15 min data. The data requirements are both 15 min and 1 hour intervals with identified 2 hours and 4 hours storm events. The following is the development of the latter by utilizing the features within Hydrolang for data retrieval, cleaning and sorting.

The specifications of the environment used for the development of the case study are the following:

* Model: ASUS GL702VSK.
* Processor: Inter Core i7-7700 HQ CPU @ 2.80 GHz.
* RAM: 12 GB.
* GPU: NVIDIA GeForce GTX 1070 GDDR5 @8 GB.
* Hard drive: 1049GB.
* Code editor: Visual Studio Code.
* Web browser: Google Chrome version 84.0.4147.105.

Library usage

A new HTML file was created containing a script which onloads the Hydrolang library. Inside the body of the file, the function for initializing hydrolang is included to create a new instance of the framework appended to a variable called hydro1. After the variable has been declared, all the functionalities inside the library can be used.

<!DOCTYPE html>

<html>

<head>

<link

rel="stylesheet"

href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"

/>

<title>Hydrolang Use Case</title>

<script

type="module"

onload="initHydrolang()"

src="./hydrolang/hydro.js"

></script>

</head>

<body>

<script>

async function initHydrolang() {

Snippet . Example of Hydrolang Onloader.

Load map for visualization

In order to understand location aspects of the study positions, a new map is loaded into the screen of the HTML webpage to visualize th thypesad amount of locations that a source can provide. To achieve this, the map module from hydrolang is used by calling the functions using the newly created instance of the framework.

Important to mention that in orde to onload the library used for the mapping option, the library used (wether leaflet orgoogle maps) is called asyncronousl. Thus, the onloading of the hydrolang framework should;d be done asynchronously as well. This changes the function caller to add the tag “async”.

Data retrieval

NOAA has been selected as data source since finer resolution precipitation of up to 15 minutes from several stations across the US can be obtained. It is important to mention that the data source requires passing a token to the header of the request, which can be obtained by requesting one at the API web page. The request also should also include a limit for the amount of items that it should contain; if it is not given then the default number is 25. The quality of the data relies on the total coverage within a specific station; the frequency of the readings and the amount of gaps. Three different endpoints were used to select a station that fulfills the data quality requirements (NOAA, 2020). For rainfall information, two different values can be obtained from a query:

* QPCP: amount of precipitation recorded at a station in a given interval in hundreds of inches of tenths of millimeters, depending on the units selected by the user.
* QGAG: volume of precipitation that is calculated by weight on a station in a given interval in hundreds of inches or tenths of millimeters, depending on the units selected by the user.

For purpose of the case study, only the QPCP amounts are being considered in metric system.

The query selects the stations within US territory that have precipitation data of 15 min. Data coverage—between 0 and 1, 1 being the best—is a variable included in the query helps sorting the stations best coverage. For the data analysis module, a function for handling retrieved data is also passed to the body of the HTML, which is used for showing the retrieved data to console. From here, the instance of the hydro is called to use the data library and retrieval function, passing as arguments the created object and the handling data function.

dataRetrievalParams1= {

  source: "noaa",

  dataType: "availablestations",

  type: "json",

  arguments: {

    datasetid: "PRECIP\_15",

locationid: "PRECIP\_15",

    sortfield: "datacoverage",

    sortorder: "desc",

    limit: 1000,

  },

  token: tokennoa,

};

function handleWaterData(data) {

  console.log(data);}

hydro1.data.retrieve(dataRetrievalParams1, handleWaterData);

Snippet . Example commands for downloading data.

By saving the file and opening it in the browser, the downloaded data was prompted to console a soon as it is retrieved. There are more than 3000 stations that provide 15 minute data within the US and most of them have a coverage of around 25%.

As part of the quality, full years of data that have been recorded by a specific station are being considered. The stations that have the best coverages have been queried yearly from the minimum available date until the maximum date (most stations have data until 2014). For example, if a station has data from 1974 through 2014, then the query starts from 01/1974 through 12/1974. The number of values from each query was quantified and classified based on the expected number of features (4-hourly data, 24 hours, 365 days = 35,040). The years with highest percentages of data are then used for processing.

For the case study, the station of Altavista, Virginia was selected because it can provide the most amount of data in a yearly basis. The data was retrieved for years 1984 to 1987. The selection was based on the amount of data that was retrieved on those years. The amount of data that is available per time request is given on the metadata object on screen; the results of the request were saved in a temporary object for further manipulation. Onwards, all the manipulations of the files or functions that are called from the library have been done in the console of the browser; the objects created in the HTML file are available on console.

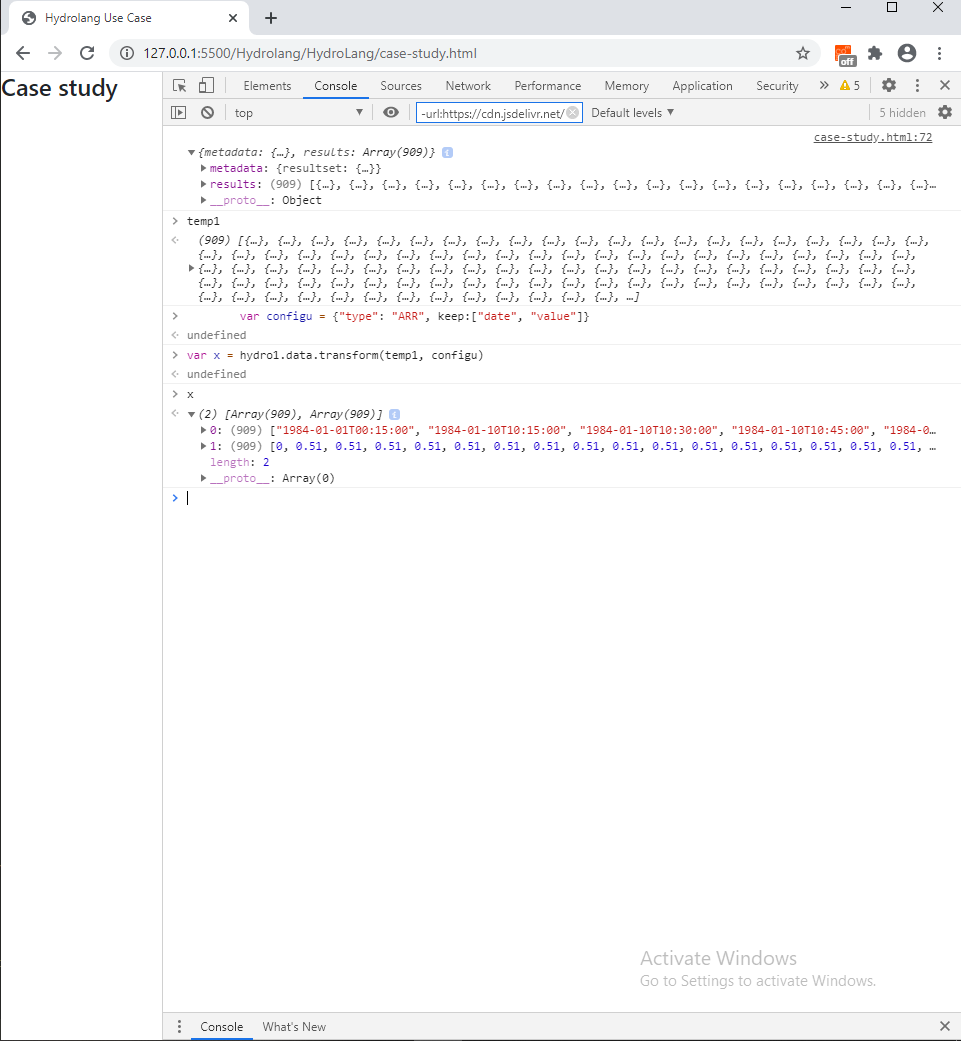


Figure . Response from console after first request.

Data analysis

Each of the request was handled separately per year. The number of items that can be included in a request are 1000 and the years for the analysis have more than 1000 items, thus the request were handled separately.

The only fields that are required per each request is the date and the value. For this, a configuration object is created to be passed later to the transformation function inside the data module. The final object, named x, will have all the data from the year.

var configu = {"type": "ARR", keep:["date", "value"]}

var xa = hydro1.data.transform(temp1, configu)

var xo = hydro1.data.transform(temp2, configu)

Snippet . Using the transformation function.

To give an overview of the data, the functionalities of basic statistics, frequency and outliers have been used. To call them, the codes in the following snippet are called into console. The resulting variables can be expanded to show the results of each function. The result of the variable stats contains basic statistics parameters that can be shown in the screen using the table functionality of the visualize module.

var final = hydro1.analyze.stats.push(xa, xo)

var stats = hydro1.analyze.stats.basicstats(final[1])

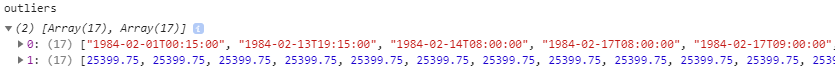
var freq = hydro1.analyze.stats.frequency(final[1])

var outliers = hydro1.analyze.stats.normoutliers(xo)

hydro1.visualize.styles({data: stats, draw: "table", config: {div: "statstable"}})

Snippet . Code for calling statistics for the data.

From the table, it is clear from the standard deviation and variance that there are outliers inside the data. From the results of the variable outliers, the shows that there are 17 outliers that are to be removed in order to further continue. For this, the functionality of outlier remove is used passing in the normalized outliers options. The original data contained 1051 values, from which 17 were outliers. This can also be observed from the freq variable.





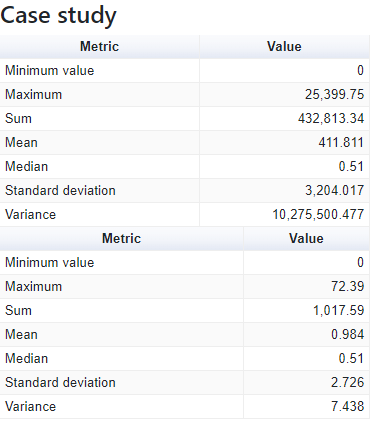


Figure Example of statistics of raw data vs cleaned data.

Once the data has been cleaned from outliers, the data was verified for gaps within the cleaned array in both time or data. The following functions call the gaps for both. The function receives as input the array to be analyzed and the time step in minutes. A timegap is identified when there is a discrepancy between the one date or another regarding the timestep. A data gap is identified when there the a value is not a number.

var timegaps = hydro1.analyze.stats.timegaps(clean, 15)

var datagap = hydro1.analyze.stats.datagaps(clean)

Snippet . Function calls for time and data gaps identifiers.

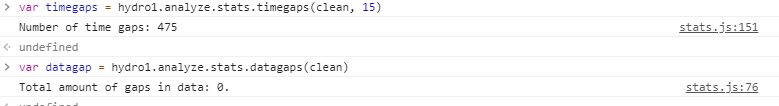


Figure . Identification of time gaps and data gaps.

After cleaning the data and identifying the gaps, the data was downloaded using the download function in the data module.

hydro1.data.download(clean,{"type": "CSV"})

To the search for the next years, the parameter object was changed to retrieve data for required data and the file was saved so that the new results were prompted to screen.

To view further manipulation of the library, please see the attached annex. All the years were retrieved, cleaned and downloaded to be further analyzed using Excel. The four years of data were

Case study:

Overview

So far, the framework provides an easy environment for data obtention and manipulation for the data sources that have been included. The analyzes that have been included just some of the vast majority that can be explored using the same approach. Manipulation of data is an important feature that should be taken close attention to. For instance, the data module has been developed to obtain, transform, download, and upload data; but it still misses more in-depth features. From the type of data that can be downloaded, manipulated, and transformed to the amount of data sources and endpoints that each data source can have.

The library has been developed considering the problems that an environmental data scientist/researcher has when trying to obtain information. So far, the library has become a robust framework that allows for manipulation of data in different formats. The flexibility of using a standardized programming language, open source software and no need for installation provides for scalability and upgrades features and in the hopes that the library can grow to have more functionalities.

More functionalities can be added to the library regarding the hdrological component. Specifically, regarding the amount of hydtrological processe sinvolved in different stahes of an analysis. Anexample of this, is the amount off procesdures that are included in precipitaito nanalysis, where one value for initial abstraction differs with another based solely on the procedure that is taken.

Challenges and limitations

One of the problems found when developing the framework was the need to adapt the functions to what the user can provide. This comes directly from the limitations of data that can be seen in literature and real world cases. For instance, sometimes data that is being retrieved is not compliant with the requirements for temporal resolution and must be manipulated elsewhere using preestablished methods.

The framework has been developed to be downloaded as a complete library to be used on local storage. Nonetheless, most of the functions require internet connection to work due to the different types of external libraries that are used. Although this can be considered a limitation, the framework can be modified in the future for changing its working environment. A migration towards backend development can be done which would give the library still more computational capability for performing special cases.

Time constrain played a key role in the development of the library. Basic functionalities were included to showcase the capabilities that the framework can provide, but a large amount of development is still required to be usable for more complete use cases. Of course, the idea that Hydrolang can become a community based library can help on the expansion of the capabilities of the framework and will showcase that.

Hydrology

Conclusions

The development of Hydrolang has been focused on data. The analytical features within each module have been tailored to adapt to different formats of data that are either retrieved from external sources or uploaded from local storage. Just the data module serves as a strong benchmark in the development of the framework, considering that the capabilities of web browsers nowadays are not limited by the computational power required for running complex algorithms. Like with the usage of external libraries like tensroflow and the capacity of usingGPU instead of CPU for running simulations, the possibilities for the growth of the framework are limitless.

Possible future work

The main idea of the framework is that it can grow as a community based software. This will allow for users from different spaces

Considering that all of the library has been developed in pure javascript with the usage of some external modules, it is possible to move the latter to a more robust client side framework such as react to create more optiojns of interoperability and interactivity.

Features within the data module should be upgraded to handle more types of data. As of now, the retrieval can be done on JSON format, transformed to array and downloaded to CSV after cleansing.

Finally, one last improvement that will be done to the gramework is the delivery method for usage. For instance

References

Ideas to explore

* What is the name of the approach which professor Demir has taking in regard of using the modules of hydrolang with the usage of chainage.
* Which are the type of basic hydrological analyzes that are commonly used by scientists to approach hydrological data.
* Dependencies between components must be declared on top of the files that are used, using ECMAScript 6 import syntax.

Real mean squares error brier score.

Thresholds?

For coding, put the critical parts of the code and if not, implement it inside the appendix.

* Include test cases for each of the modules that are to be retrieved. For instance, for every one of the sources include the parameters that are needed to be included within the JSON file accoridnly to how they are trerived from the source.

Time series considerations

Of course, hydrolang serves as a robust and reliable instance for studying and cleaning data that has been retrieved from external sources and that has not been modified previously. It is important to denote that although the latter is certain, the data must be studied and analyzed by the end user.

Index

Case study

Objects used for data retrieval of the US site

dataRetrievalParams2 = {

  source: "noaa",

  dataType: "datasets",

  type: "json",

  arguments: {

    stationid: "COOP:440166"

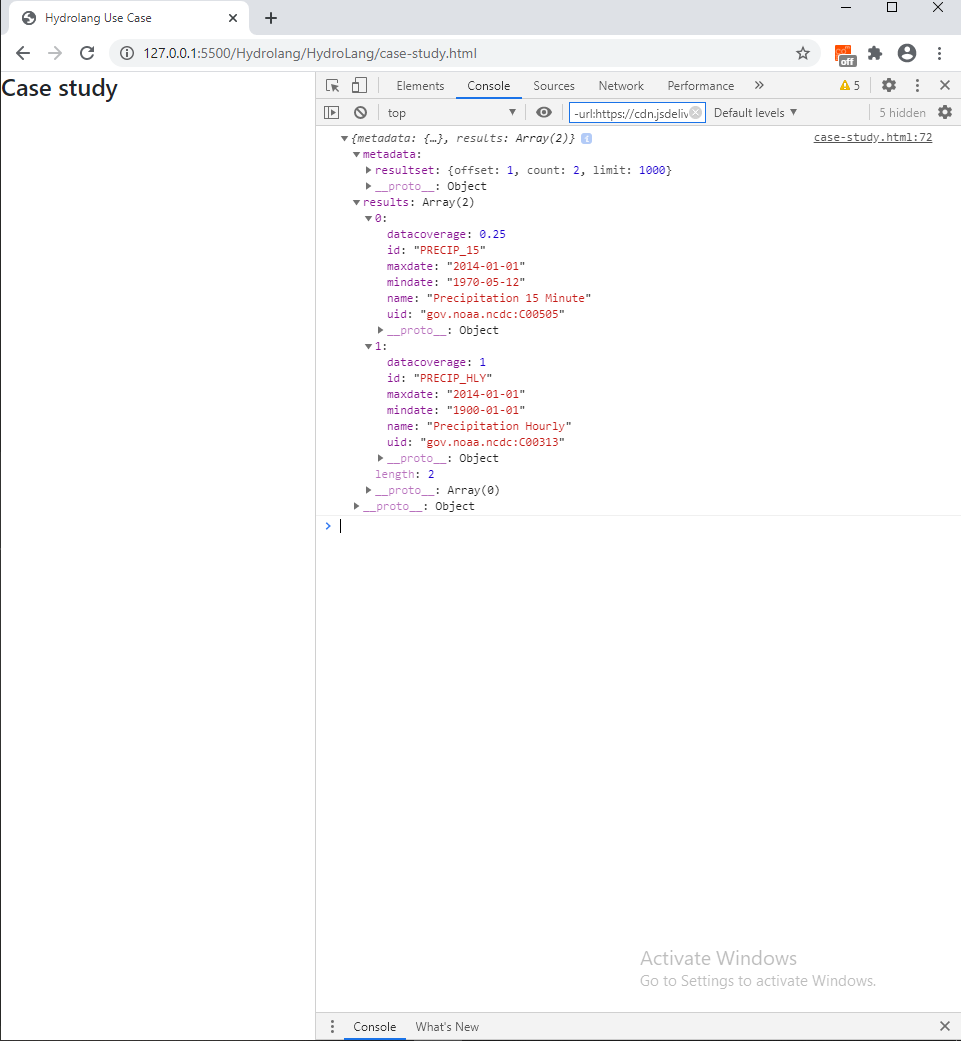
    datasetid: "PRECIP\_15",

    limit: 1000,

  },

  token: tokennoa1,

};



Data from 1984

dataRetrievalParams3 = {

  source: "noaa",

  dataType: "prec-15min",

  type: "json",

  arguments: {

    stationid: "COOP:440166"

    units: "metric",

    startdate: "1984-01-01",

    enddate: "1984-12-31",

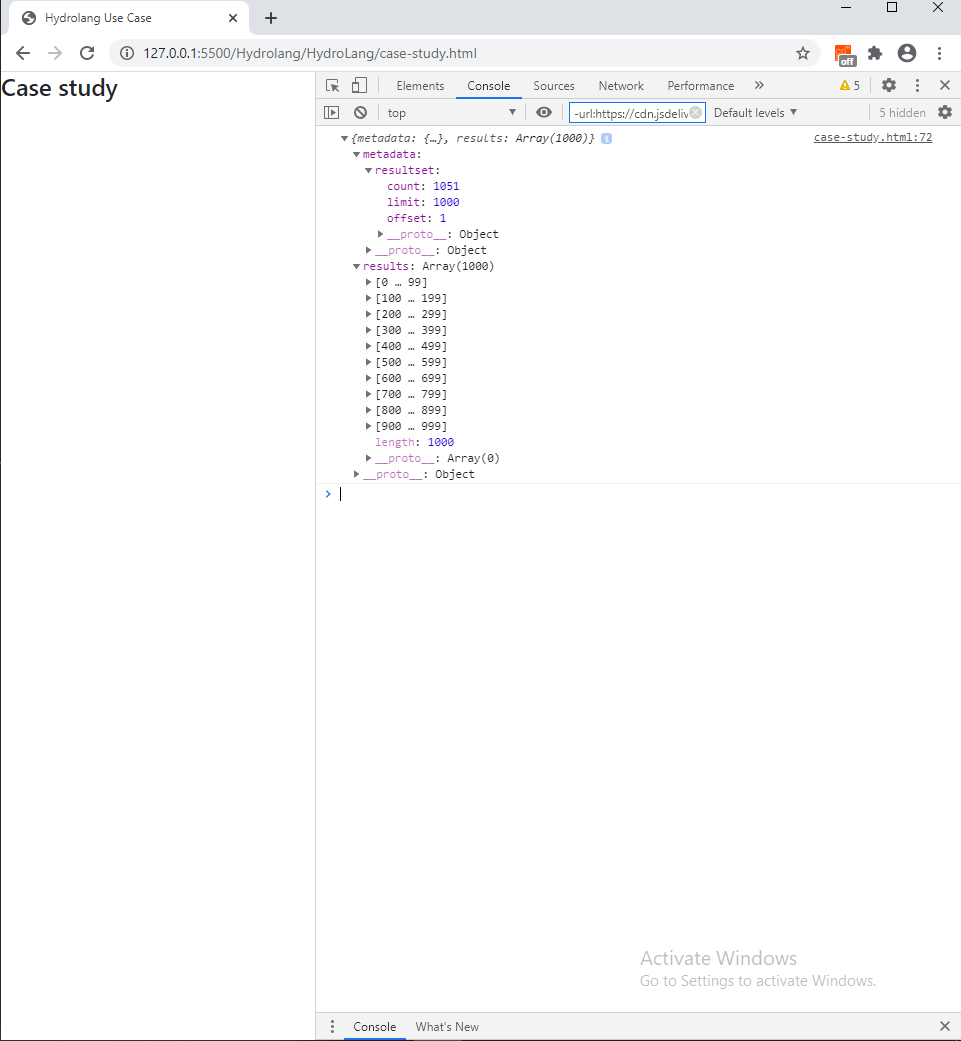
    limit: 1000,

    datatype: "QPCP",

  },

  token: tokennoa1,

};



Data from 1985

Object requests

dataRetrievalParams1 = {

  source: "noaa",

  dataType: "prec-15min",

  type: "json",

  arguments: {

    stationid: "COOP:440166"

    units: "metric",

    startdate: "1985-01-01",

    enddate: "1985-10-31",

    limit: 1000,

    datatype: "QPCP",

  },

  token: tokennoa1,

};

dataRetrievalParams2 = {

  source: "noaa",

  dataType: "prec-15min",

  type: "json",

  arguments: {

    stationid: "COOP:440166"

    units: "metric",

    startdate: "1985-11-01",

    enddate: "1985-12-31",

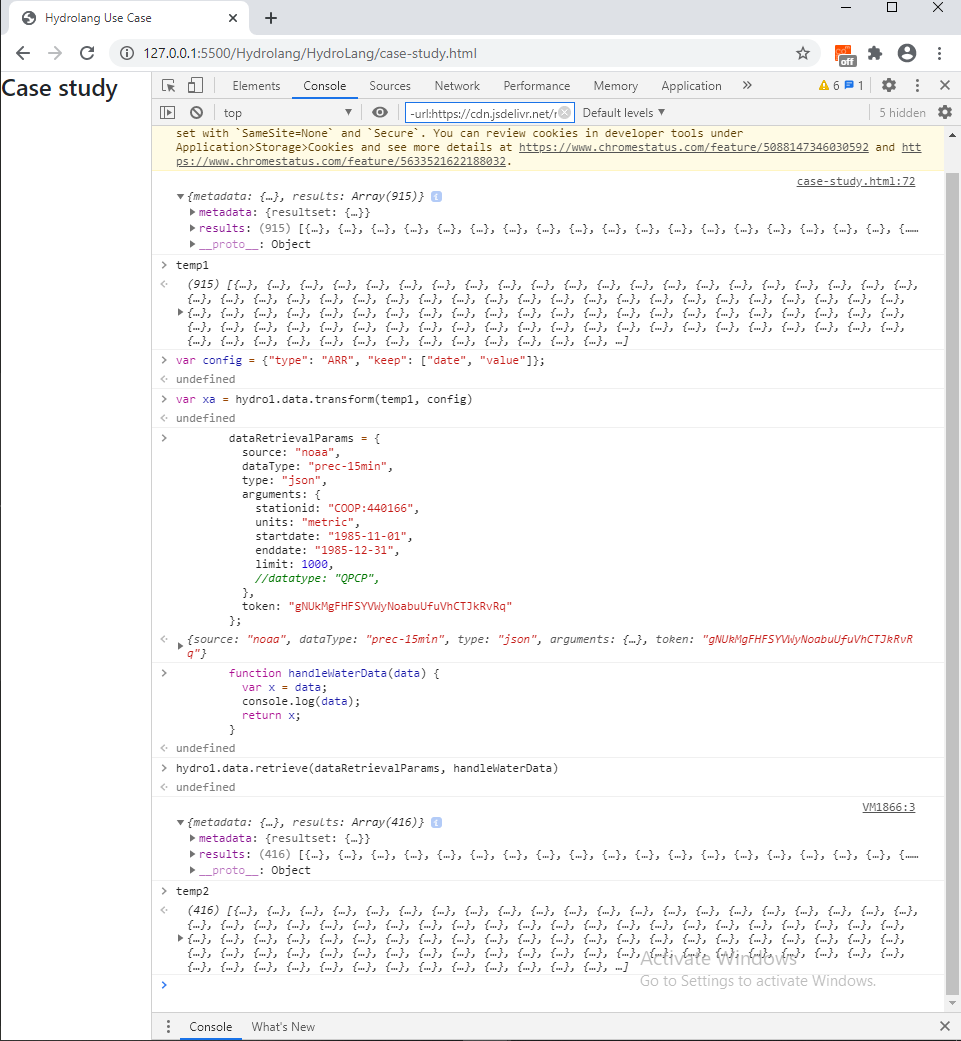
    limit: 1000,

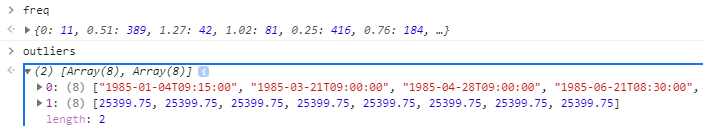
    datatype: "QPCP",

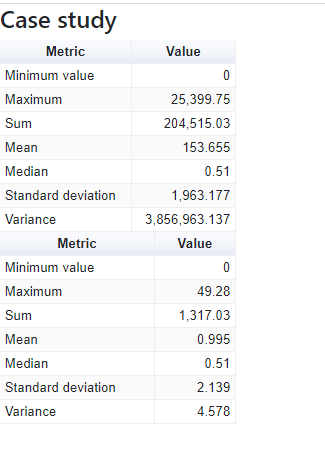
  },

  token: tokennoa1,

};







Raw versus cleaned data.

Data for 1986

dataRetrievalParams1 = {

  source: "noaa",

  dataType: "prec-15min",

  type: "json",

  arguments: {

    stationid: "COOP:440166"

    units: "metric",

    startdate: "1986-01-01",

    enddate: "1986-10-31",

    limit: 1000,

    datatype: "QPCP",

  },

  token: tokennoa1,

};

dataRetrievalParams2 = {

  source: "noaa",

  dataType: "prec-15min",

  type: "json",

  arguments: {

    stationid: "COOP:440166"

    units: "metric",

    startdate: "1986-11-01",

    enddate: "1987-12-31",

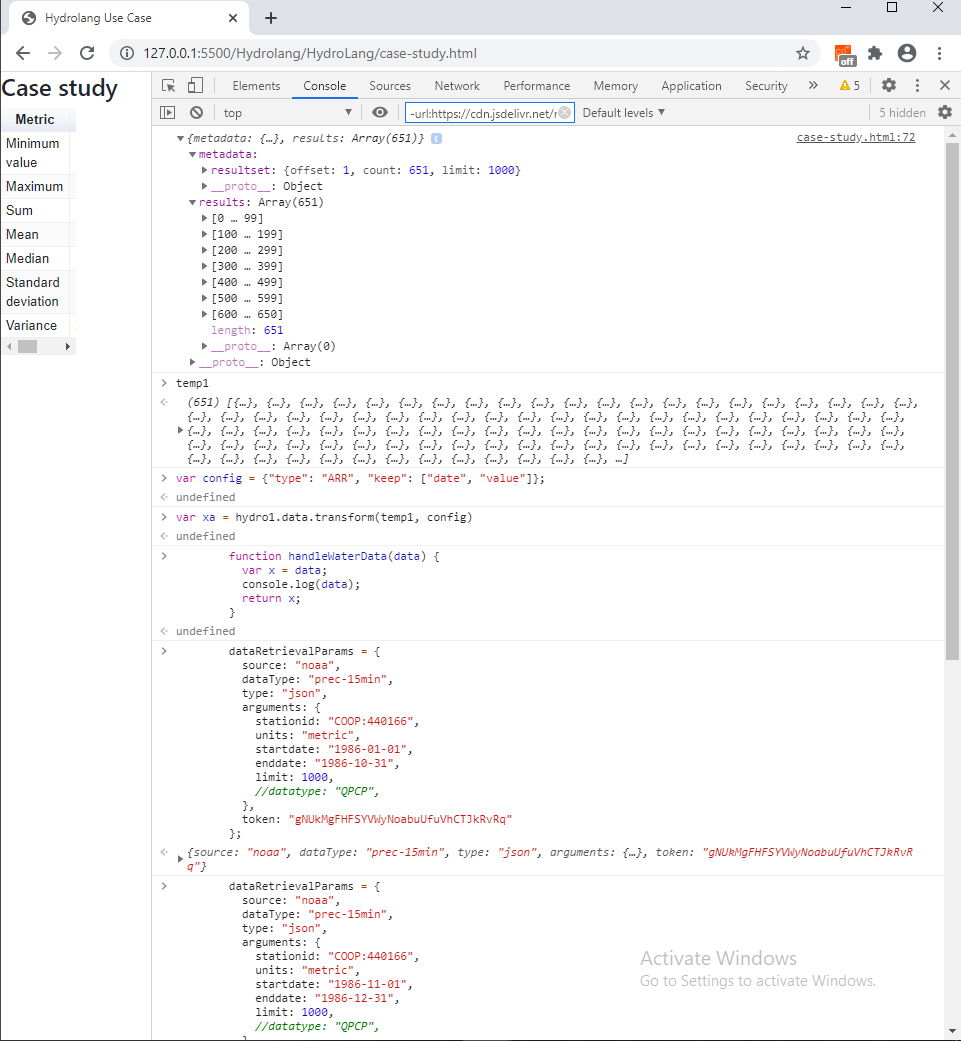
    limit: 1000,

    datatype: "QPCP",

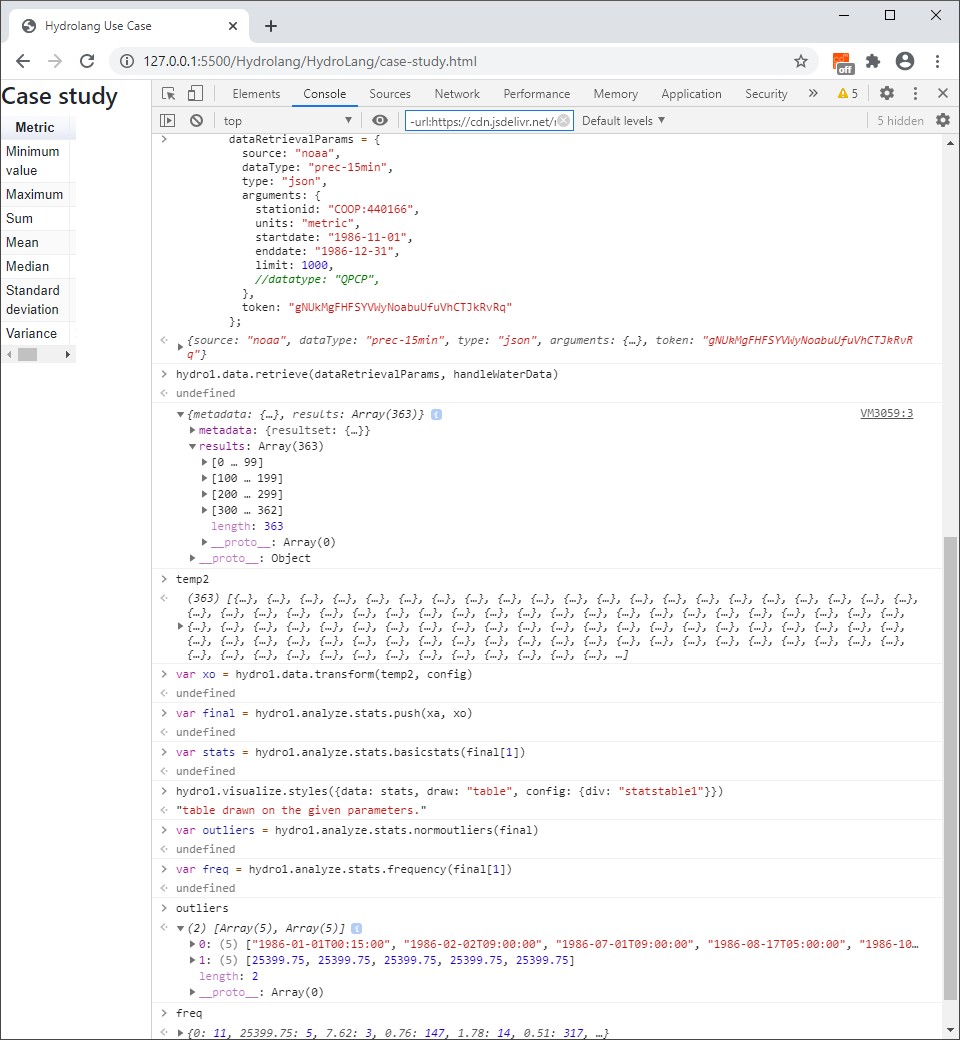
  },

  token: tokennoa1,

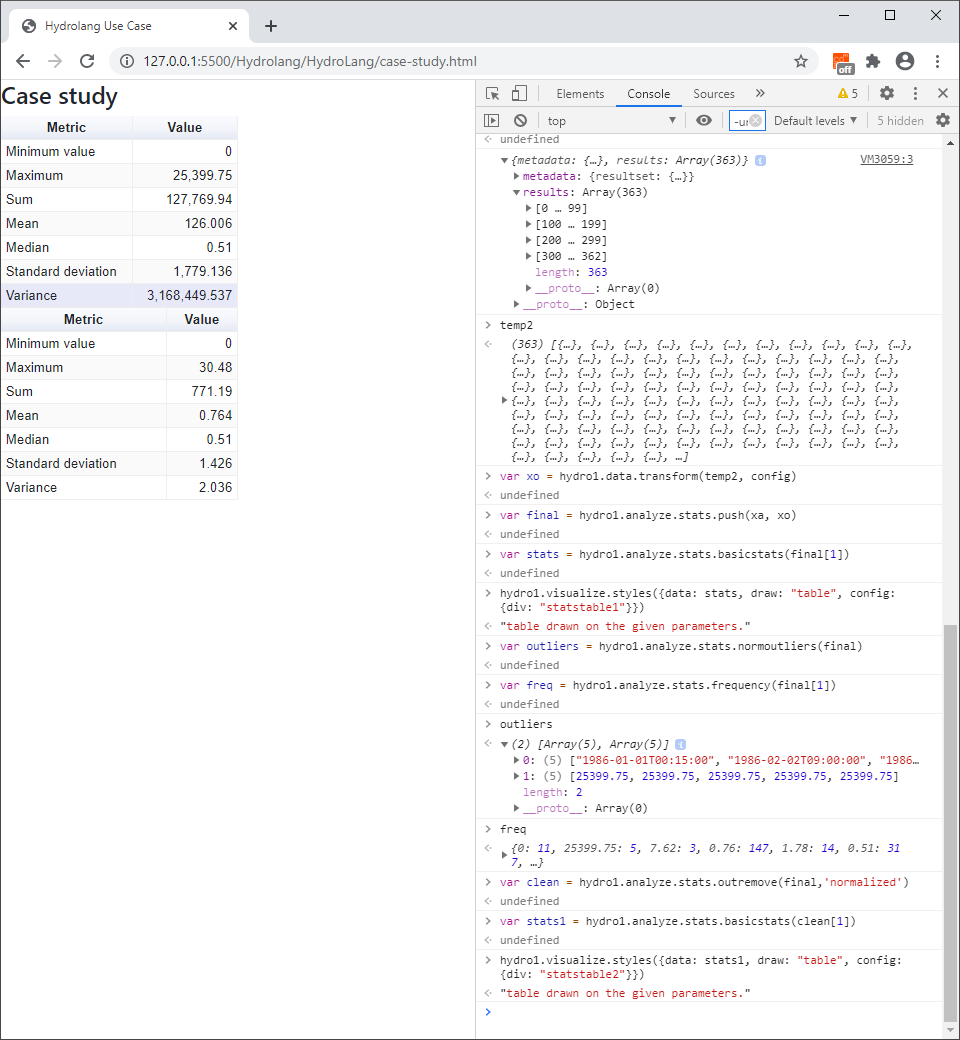
};



Response from the first request.



Second request, join two requests, identify outliers and frequency.



Raw data versus cleaned data.

Data for 1987

dataRetrievalParams1 = {

  source: "noaa",

  dataType: "prec-15min",

  type: "json",

  arguments: {

    stationid: "COOP:440166"

    units: "metric",

    startdate: "1987-01-01",

    enddate: "1987-10-31",

    limit: 1000,

    datatype: "QPCP",

  },

  token: tokennoa1,

};

dataRetrievalParams2 = {

  source: "noaa",

  dataType: "prec-15min",

  type: "json",

  arguments: {

    stationid: "COOP:440166"

    units: "metric",

    startdate: "1987-11-01",

    enddate: "1987-12-31",

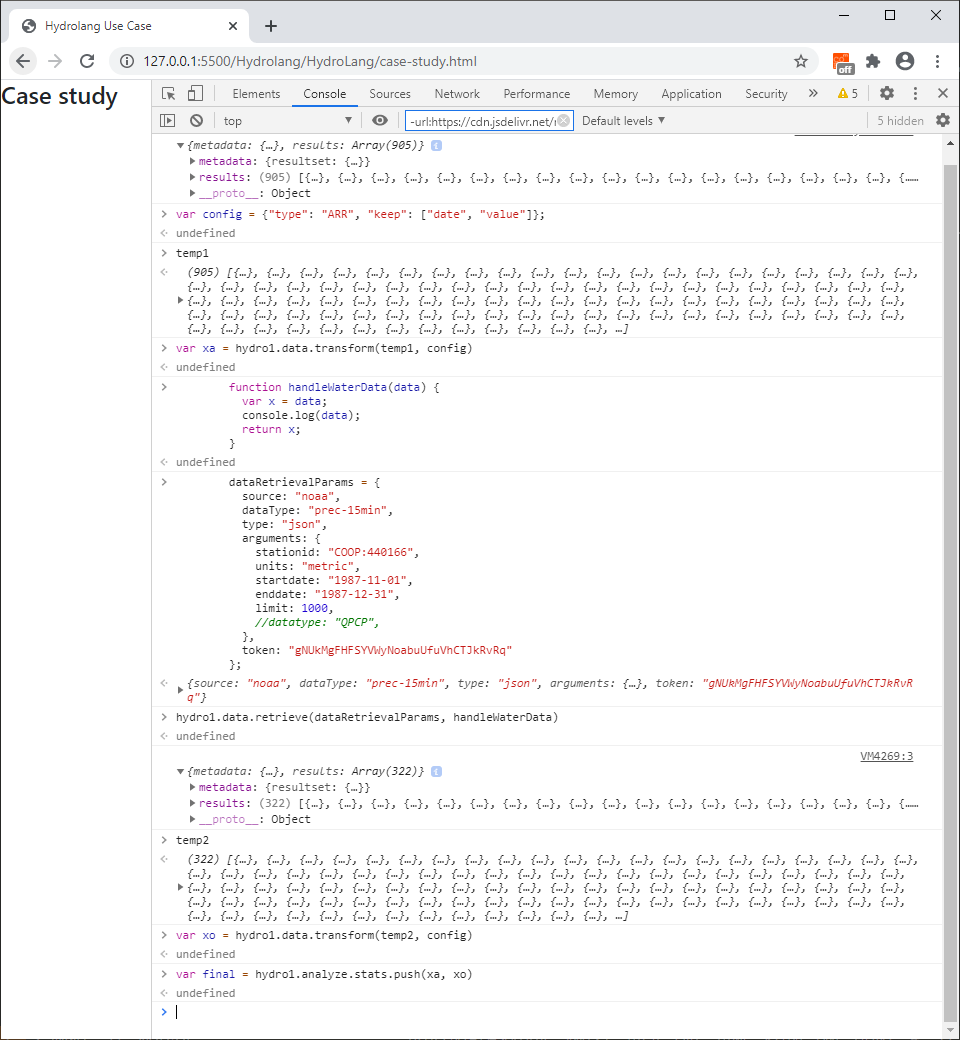
    limit: 1000,

    datatype: "QPCP",

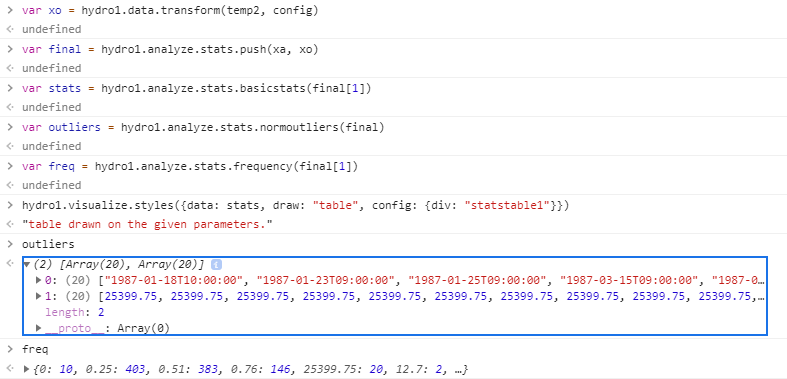
  },

  token: tokennoa1,

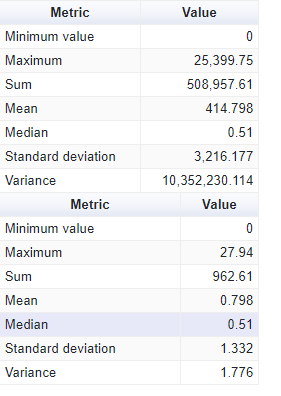
};



Response for both requests and joining the two sets of data.



Outliers and frequency for raw data.



Function and object examples.

* The objects passed as arguments must contain the name of the key as explained on the table.

Usages

If an object has been established as argument, the names inside the description represent the keys of the object. If it is an array instead, then the type of data will be specified. Time series can be both represented in the following formats:

var timeseries1: [["date1", "date2", "date3"], [1,2,3]]

var timeseries2: [[0.5,1,1.5],[1,2,3]]

If an argument example mentions: “If time series, either in date string or number”, one of the two array types from above are expected. For more detailed explanations or examples on usage, please reference to the repository.

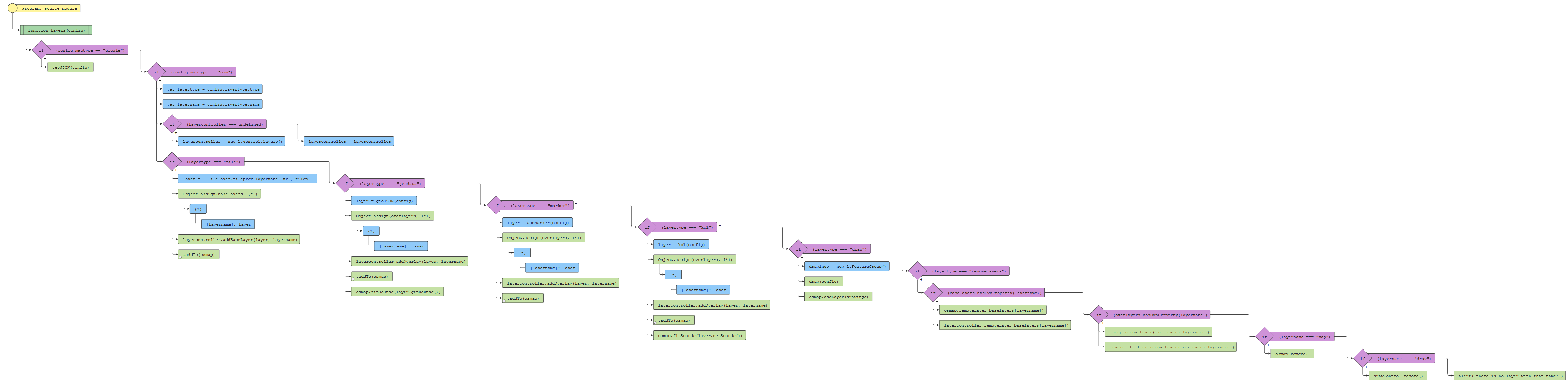
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Function | Arguments | Example Arguments | Example function call |
| Analyze | hydro | Totalprec | arr - array of arrays | var arr = n\*array | hydro[n].analyze.hydro.totalprec(arr) |
|  |  | arithmetic | arr - array of numbers. | var arr = array | hydro[n].analyze.hydro.arithmetic(arr) |
|  |  | thiessen | Object - precipitation and areas. | var obj = {rainfall: 1darray(numbers), areas: 1darray(numbers)} | hydro[n].analyze.hydro.thiessen(obj) |
|  |  | bucketmodel | Object - precipitation, baseflow, evaporation as object, landuse as object, infiltration\* | var obj = {rainfall: 1d or 2darray(time series or just rainfall event. If time series, either in date string or number for date.), baseflow: number, evaporation:{data: 1d or 2darray(time series or just rainfall event. If time series, either in date string or number for date.)} landuse: {agriculture: number, barerock: number, grassland: number, forest: number, moorland: number}, infiltration: number } | hydro[n].analyze.hydro.bucketmodel(obj) |
|  |  | floodhydro | Object - precipitation, unithydro, cn, stormduration, timestep | var obj = {rainfall: 2d array(time series either in date string or number for date), unithydro: 2d array(timeseries in number), cn: number, stormduration: number, timestep: number} | hydro[n].analyze.hydro.floodhydro(obj) |
|  |  | dimunithydro | Object - distribution as object, timestep, numhours | var obj = {distribution: {type: string, PRF: number}, timestep: number, numhours: number} | hydro[n].analyze.hydro.dimunithydro(obj) |
|  |  | ground1d | Object - length, k, nodes, w0, w1, hL, q0, qL | var obj = {length: number, k: number, w0: number, w1: number, hL: number, q0: number, qL: number} | hydro[n].analyze.hydro.ground1d(obj) |
|  |  | move | arr - array of numbers, from - number, to - number | var arr = array of numbers, from - number, to - number | hydro[n].analyze.hydro.move(arr, from, to) |
|  |  | matrix | m - number, n - number, d - number | m - number, n - number, d - number | hydro[n].analyze.hydro.matrix(m, n, d) |
|  |  | equationsystemsolver | matrix - m\*array, vec\_left - 1d array, vec\_right - 1d array | matrix - m\*array, vec\_left - 1d array, vec\_right - 1d array | hydro[n].analyze.hydro.equationsystemsolver(m, n, d) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Function | Arguments | Example Arguments | Example function call |
| Analyze | stats | copydata | data - multidim array or object. | var data = data (timeseries or data) | hydro[n].analyze.stats.copydata(data) |
|  |  | onearray | data - array with data | var data = 2darray (timeseries) | hydro[n].analyze.stats.onearray(data) |
|  |  | datagaps | arr - array with data | var arr = arr (1d or 2darray. Timeseries in date string or number) | hydro[n].analyze.stats.datagaps(arr) |
|  |  | gapremoval | arr - array with data | var arr = arr (1d or 2darray. Timeseries in date string or number for date). | hydro[n].analyze.stats.gapremoval(arr) |
|  |  | sum | arr - array with numbers | var arr = 1darr(values) | hydro[n].analyze.stats.sum(arr) |
|  |  | mean | arr - array with numbers | var arr = 1darr(values) | hydro[n].analyze.stats.mean(arr) |
|  |  | stddev | arr - array with numbers | var arr = 1darr(values) | hydro[n].analyze.stats.stddev(arr) |
|  |  | sumsqrd | arr - array with numbers | var arr = 1darr(values) | hydro[n].analyze.stats.sumsqrd(arr) |
|  |  | min | arr - array with numbers | var arr = 1darr(values) | hydro[n].analyze.stats.min(arr) |
|  |  | max | arr - array with numbers | var arr = 1darr(values) | hydro[n].analyze.stats.max(arr) |
|  |  | unique | arr - array with numbers | var arr = 1darr(values) | hydro[n].analyze.stats.unique(arr) |
|  |  | standardize | arr - array with numbers | var arr = 1darr(values) | hydro[n].analyze.stats.standardize(arr) |
|  |  | quantile | arr - array with numbers, q - number | var arr = 1darr(values), var q = number | hydro[n].analyze.stats.quantile(arr, q) |
|  |  | cleaner | arr - array with data | var arr = 1d array (data) | hydro[n].analyze.stats.cleaner(arr) |
|  |  | outremove | arr - array with data, type - string, p1 - number, p2 - number | var arr = 1d or 2d array (Timeseries or just rainfall event. If time series, then date string or number for date.) var type = string (normalized or interquartile). | hydro[n].analyze.stats.outremove(arr, type,p1, p2) |
|  |  | itemfilter | arr1 - array with data to be cleaned, arr2 - array with data to be removed. | var arr1, arr2 - 1d array of dates (number or string) or values | hydro[n].analyze.stats.itemfilter(arr1, arr2) |
|  |  | correlation | Object - q1, q2 | var obj = {q1: 1darray (data), q2: 1darray (data)} | hydro[n].analyze.stats.correlation(obj) |
|  |  | variance | arr - array with numbers | var arr = 1darr(values) | hydro[n].analyze.stats.variance(arr) |
|  |  | frequency | arr - array with numbers or strings | var arr = 1darray (numbers or strings) | hydro[n].analyze.stats.frequency(arr) |
|  |  | interoutliers | arr - array with data, q1 - number, q2 - number | var arr = 1d or 2d array (Timeseries or just rainfall event. If time series, then date string or number for date.) var q1 = number, q2 = number | hydro[n].analyze.stats.interoutliers(arr, q1, q2) |
|  |  | normoutliers | arr - array with data, low - number, high - number | var arr = 1d or 2d array (Timeseries or just rainfall event. If time series, then date string or number for date), var low= number, var high = number | hydro[n].analyze.stats.normoutliers(arr, low, high) |
|  |  | fastfourier | arr - array with numbers | var arr = 1darray (numbers) | hydro[n].analyze.stats.fastfourier(arr) |
|  |  | basicstats | arr - array with numbers | var arr = 1darray (numbers) | hydro[n].analyze.stats.basicstats(arr) |
|  |  | joinarray | arr - array with numbers or strings | var arr = 1darray (numbers or strings) | hydro[n].analyze.stats.joinarray(arr) |
|  |  | numerise | arr - array with strings | var arr = 1darray (strings) | hydro[n].analyze.stats.numerise(arr) |
|  |  | flatenise | Object - Columns, graphdata | var obj = {Columns: 1darray (strings), graphdata: 1darray(numbers)} | hydro[n].analyze.stats.flatenise(obj) |
|  |  | dateparser | arr - array with strings | var arr = 1darray (strings) | hydro[n].analyze.stats.dateparser(arr) |
|  |  | arrchange | arr - array (numbers or strings) | var arr = n\*m array | hydro[n].analyze.stats.arrchange(arr) |
|  |  | push | arr1 - array which data will be pushed, arr2 - array with data to be pushed | arr1 = 1darray (numbers or strings), arr2 = 1darray (numbers or strings) | hydro[n].analyze.hydro.push(arr1, arr2) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Function | Arguments | Example Arguments | Example function call |
| Data | data | retrieve | Object - source, dataType, type, arguments as object, token (if required), proxy (if required) | var obj = {source: string, dataTyep: string, arguments: {options: strings}, token (if required): string, proxy (if required): string} | hydro[n].data.retrieve(obj, callback) |
|  |  | transform | data - Object with data, Object - configuration options | data = Object (retrieved data or JS arrays), var obj = {type: string, keep : 1d array} | hydro[n].data.transform(data, obj) |
|  |  | upload | type - string | type - string (either CSV or JSON) | hydro[n].data.upload(type) |
|  |  | download | data - Object with data, Object - configuration options | data = Object (retrieved data or JS arrays), var obj = {type: string, keep : 1d array} | hydro[n].data.download(data, obj) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Function | Arguments | Example Arguments | Example function call |
| Visualize | visualize | chart | Object - chartType, data, divID, options | var obj = {chartType: string (column, line, scatter, histogram, timeline), divID: string, data: 1d or 2d array, options: {}} | hydro[n].visualize.chart(obj) |
|  |  | table | Object - chartType, data, divID, options | var obj = {chartType: string (table), divID: string, data: 1d array, dataType: 1d array (same size as data), options: {}} | hydro[n].visualize.table(obj) |
|  |  | styles | Object - chartType, data, divID, options as object | var obj = {chartType: string (charts or table), data: 1d or 2darray depending on chartType, divID: string, options: {title: string}} | hydro[n].visualize.styles(obj) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Function | Arguments | Example Arguments | Example function call |
| Map | maps | loader | Object - map | var obj = {maptype: string} | hydro[n].map.loader(obj) |
|  |  | Layers | Object - maptype, layertype as object | var obj2 = {maptype: string, layertype: {type: string, markertype: string, coord: 1d array, name: string}} var obj3 = {maptype: string, layertype: {type: string}} | hydro[n].map.Layers(obj) |
|  |  | renderMap | Object - maptype, lat, lon, zoom, layertype as object | var obj = {maptype: string, lat: number, lon: number, zoom, number, layertype: {type: string, name: string}} | hydro[n].map.renderMap(obj) |



Flowcharts

Map module

