Version 0.1.4

IGEM User Guider

Integrative Genome-Exposome Method

2023

Table of Contents

[Introduction 3](#_Toc139042855)

[Install 5](#_Toc139042856)

[Users and Roles 9](#_Toc139042857)

[Server Application 18](#_Toc139042858)

[Master Data 18](#_Toc139042859)

[Data Source 19](#_Toc139042860)

[Connector 23](#_Toc139042861)

[Group 28](#_Toc139042862)

[Category 29](#_Toc139042863)

[Term 29](#_Toc139042864)

[Prefix 31](#_Toc139042865)

[Word-Term 31](#_Toc139042866)

[Database Management 32](#_Toc139042867)

[ETL 37](#_Toc139042868)

[Collect 38](#_Toc139042869)

[Prepare 38](#_Toc139042870)

[MAP 39](#_Toc139042871)

[Reduce 39](#_Toc139042872)

[Workflow 39](#_Toc139042873)

[GE Application 42](#_Toc139042874)

[GE.db 42](#_Toc139042875)

[Direct Access to Tables 42](#_Toc139042876)

[Sync GE.db 45](#_Toc139042877)

[GE.filter 46](#_Toc139042878)

[Term Map Report 47](#_Toc139042879)

[Gene Exposome Report 47](#_Toc139042880)

[SNP Exposome Report 48](#_Toc139042881)

[Other Functions and Report 49](#_Toc139042882)

[EPC Application 52](#_Toc139042883)

[Data Load 52](#_Toc139042884)

[Data Description 52](#_Toc139042885)

[Data Modification 53](#_Toc139042886)

[Data Analysis 56](#_Toc139042887)

[Data Visualization 57](#_Toc139042888)

[Anexo A : Functions Map 60](#_Toc139042889)

[Anexo B: Database Schema 61](#_Toc139042890)

[Figures 62](#_Toc139042891)

# Introduction

The IGEM (Integrative Genome-Exposome Method) system is a powerful platform designed to host various applications (APPs) that share common resources and interact with each other through a single database. It provides a flexible and dynamic framework for genomic and exposomic research, enabling the integration of diverse data sources and facilitating complex analyses.

The primary goal of the IGEM system is to enable researchers to study exposure-exposure (ExE) and gene-environment (GxE) interactions in high-dimensional big data sets. By integrating an automated knowledge-based approach, IGEM offers a user-friendly, open-source, and open-access software solution that supports high-throughput quality control (QC), knowledge-driven ExE and GxE filtering, machine learning (ML), regression-based interaction analysis, and big data visualization.

One of the key applications within the IGEM system is the GxE (Gene x Exposome) application. The GxE application focuses on collecting external data sets, identifying key genetic and exposomic information, and building a comprehensive knowledge base. This knowledge base serves as a valuable resource for researchers, providing them with the ability to perform dynamic and exploratory queries, uncover insights, and generate novel hypotheses.

The IGEM system follows a modular architecture (figure 1) and currently includes multiple applications such as GE, OMICS, and EPC. These applications interact with each other, allowing users to query GxE relations, integrate external data, perform regressions, and analyze data seamlessly. The system also provides components that handle database interface operations, file system management, and other essential tasks to support the functionality of the applications.

Access to the IGEM system is available through a web query interface, scripts, or the command line, providing users with flexibility in choosing their preferred mode of interaction. This accessibility, combined with the consolidated view of the IGEM components, enhances the usability and efficiency of the system.

This User Guide aims to provide comprehensive documentation for utilizing the IGEM platform and its various applications. It covers installation instructions, detailed usage examples, and explanations of key functionalities. Whether you are a researcher, data scientist, or domain expert, this guide will assist you in effectively leveraging the IGEM system to drive your genomic and exposomic analyses.

A screen shot of a computer

Description automatically generated with medium confidence

Figure 1: IGEM Architeture

By harnessing the capabilities of the IGEM system, researchers have the ability to extract and integrate external data from diverse sources, perform complex analyses, and gain valuable insights. Despite the ongoing potential for improvement and integration of additional data, the IGEM system represents a significant milestone in the field of genomics and exposomics, offering a unified knowledge base and a path towards implementing IGEM as a web-based solution.

In summary, the IGEM system provides researchers with a powerful platform for integrative genomic and exposomic research. With its user-friendly interface, advanced analysis capabilities, and modular architecture, IGEM empowers users to explore complex interactions, analyze large datasets, and uncover valuable insights. This user guide will serve as your comprehensive resource to navigate and utilize the IGEM platform effectively, driving impactful genomic and exposomic analyses.

# Install

IGEM has two versions, IGEM and IGEM-Lite. Assess your needs and select the version that aligns with your research goals and preferences. Both versions offer valuable tools and access to the IGEM platform, allowing you to leverage genomic and exposure data for meaningful insights and discoveries.

A picture containing screenshot, diagram, text, design

Description automatically generated

Figure 2: IGEM and IGEM Lite components

**IGEM Version**

The IGEM version provides a comprehensive set of tools for filtering, analyzing, and visualizing data. It encompasses all core components of the IGEM architecture, including GE-db, GE-filter, ETL process, and integration with external software packages. We recommend the IGEM Server version for users who need a centralized and powerful system to handle large-scale omics data and create the own knowledge database.

IGEM is available on PyPl or through GitHub. It can be installed in a virtual environment with Python >= 3.9. Run via the command line:

$ pip install igem

**IGEM Lite Version**

The IGEM version provides a comprehensive set of tools for filtering, analyzing, and visualizing data. It encompasses all core components of the IGEM architecture, including GE-db, GE-filter, ETL process, and integration with external software packages. We recommend the IGEM Server version for users who need a centralized and powerful system to handle large-scale omics data and create their own knowledge database.

IGEM Lite is available on PyPl or through GitHub. It can be installed in a virtual environment with Python >= 3.9. Run via the command line:

$ pip install igem-lite

**Database Customization**

IGEM accepts several types of software to manage the database, including, SQLite, MySQL, Postgres, and others. By default, the system is already configured with SQLite.

To change the database manager, open the {package\_path}/igem/src/settings.py file and change the DATABASES parameters. The example below demonstrates a configuration using a Postgres database:

DATABASES = {"default": {

"ENGINE":django.db.backends.postgresql\_psycopg2",

"NAME": "IGEM",

"USER": "postgres",

"PASSWORD": "your\_password",

"HOST": "127.0.0.1",

"PORT": "5432",

}}

IMPORTANT: Changing the database is optional as the system is configured by default to create a local SQLite database.

If you want to use a database created on another computer/server, edit the base path, for example, an SQLLITE base:

DATABASES = {

"default": {

"ENGINE": "django.db.backends.sqlite3",

"NAME": {path} / "db.sqlite3",

}}

Unzip and run the deploy\_db.py script in the environment with IGEM.

The other way to create the database, access the IGEM folder and run the following:

$ python manage.py makemigrations

A screenshot of a computer

Description automatically generated with medium confidence

Figure 3: DB Makemigrations

The next command to create the database with all the IGEM metadata:

$ python manage.py migrate

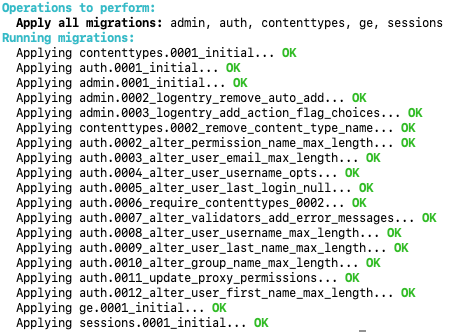


Figure 4: DB Migrate

At this point, we already have IGEM installed, and the database created with the IGEM structure. To check if the system is working correctly, type:

$ python manage.py check

The IGEM system has a layer of security per user and functions. To create the first user, run:

$ python manage.py createsuperuser

Enter your username, email, and security password.

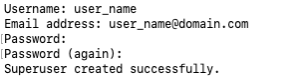


Figure 5: Create user

The system will be ready to parameterize the master data, perform external data load and generate reports.

**Web Interface**

The IGEM system has a web interface for performing activities such as master data registration and simple queries in the database.

To start the WEB service, type:

$ python manage.py runserver

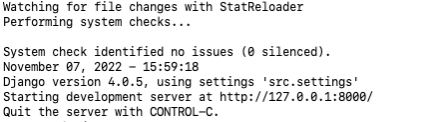


Figure 6: Runserver

In a browser, go to http://127.0.0.1:8000/admin.

The IGEM system will show the authentication page.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 7: Web Login

Enter the username and password created in the previous steps. The administration page will be loaded after authentication.

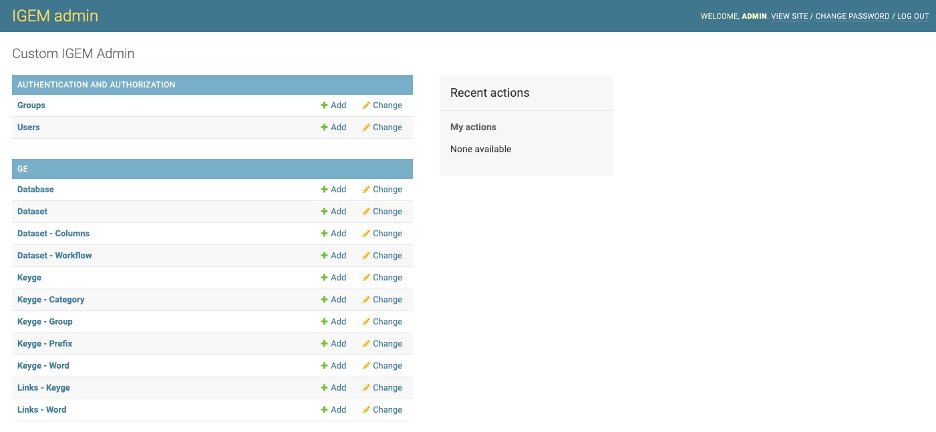


Figure 8: Web IGEM Admin

The features of this interface will be explored in detail in Master Data and Access and Permissions.

**File Structure**

Inside the src directory, we will have:

* */ge/*: all source codes and interfaces for the functioning of APP GE.
* */loader/*: all input files for loading master data and output directory of the FILTER process.
* */psa/*: Persist Store Area to store the database files downloaded and processed by the ETL process. Each DATASET will have its subfolder within the PSA.
* */src/*: hosts the source code of IGEM components, configurations and parameterizations.
* */templates/*: hosts the standard web interfaces in IGEM.

PSA - Store Area Persists

The PSA is a folder that stores the Dataset files loaded in their original format and transformed during the ETL process. Each Database will be a subfolder, and each Dataset a subfolder concerning the Database.

These structures will be created automatically, and if deleted, they will be created again on the following workflow run for the corresponding dataset. Each external Dataset source will be a design solution for the original uploaded file. If you want to keep this file for queries and analyses, configure it in the Dataset register to keep the original file. Important that this file will be kept unzipped. To reduce the amount of system space, it is not recommended to keep these files. The subfolder will also have a transformed version normalized by the rules of the applied dataset.

## Users and Roles

The IGEM system was developed to be flexible, and it is necessary to evaluate the best configuration for the scenario and objectives of each installation. A suggestion would be a model of three functions, being:

* *Administrator*: responsible for installing and updating the environment to receive the IGEM, Customizing the IGEM, configuring the database and monitoring the performance and creating and maintaining users.
* *Super User*: responsible for registering master data such as Database, Dataset, Keyge, among others. He will also be responsible for creating the ETL JOBs and monitoring them via workflow. For this group, we will have access to the WEB interface for parameterization of the registration, the necessary tables, access to processes such as Collect, Prepare, and DB
* *Users*: they will be the clients of the system, performing queries and analysis of the IGEM data. For this group, we will have processes such as GE.filter

**Users**

New users can be created via command line:

$ python manage.py createsuperuser

Through IGEM’s friendly web interface, it will be possible to carry out Users management activities.

Activate the IGEM web service if you have not already done so. Go to the /src/ folder and type the command line:

$ python manage.py runserver

>>> Watching for file changes with StatReloader

Performing system checks...

System check identified no issues (0 silenced)

March 24, 2023 - 12:56:26

Django version 4.1.5, using settings 'src.settings'

Starting development server at <http://127.0.0.1:8000/>

Quit the server with CONTROL-C.

If it returns a port error, you can specify a different port:

$ python manage.py runserver 8080

Access the address in the link provided in Starting development server. Significantly, this address may vary depending on the initial settings performed during installation.

After user authentication and on the initial administration screen, select an option Users.

On the User screen, we will have options to consult, modify, add and eliminate Users.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 9: Web User Admin

On the first screen, we have a view of all available Users. To consult, click a desired User.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 10: Web user interface

On the next screen, we have all the Users fields open for modifications. To modify, change the desired information and select one of the three button options:

* Save and add another: Will save the changes and open a blank User screen to add a new User record.
* Save and Continue editing: Will save the changes and continue on the User screen.
* Save: Will save the changes and return to the screen with the list of User.

In the History button, we can consult all the modifications carried out in the User, this function will be important to track modifications and audit the process.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 11: Web change history admin

The DELETE button will permanently delete the User record.

Caution: when deleting a User, the system will also delete all records dependent on that User.

Deletion can also be performed en bloc. On the Users List screen, select all the User you want to delete, choose the Delete Selected User action and click on the GO button.

Be careful, this elimination operation will be definitive for the User and for all other records dependent on it, as already explained.

For the User, we will have two filter locations:

* First located at the top of the User List screen where we can search broadly.
* Second on the right sidebar, being able to select by status and actives.

To add new User, we will have three different ways:

* by the + Add button on the left sidebar.
* Through the ADD USER + button in the right field of the Users list.
* Via the Save and add another button located within a User record.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 12: Web new user

After entering the username, and password and saving, the system will be directed to the user details page. Inform the personal data of the first name, last name, and email address.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 13: Web save new user

Under permissions, check:

* Active Box to allow user activities.
* Staff to allow the user to access the administration page
* Superuser to give access to all data and system registration. If this option is not checked, it will be necessary to manually add which records and functions the user will have access to or add a group so that the user can access the system.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 14: Web add permisions to user

In groups, inform which groups the user will inherit the accesses to. For Super User, it will not be necessary to advertise any groups as they are given full access.

If you want to customize the user or add more system functionality and access options, access the user’s permissions type.

In user date, we will have how much was the last access and the date when the user was created.

A screenshot of a computer

Description automatically generated with low confidence

Figure 15: Web user log

After performing the new parameterizations, save the new user

**Group**

The groups help maintain access; we can create groups for different functions and assign them to the users who perform them, thus avoiding giving users undue access.

On the Groups screen, we will have options to consult, modify, add and eliminate Groups.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 16: Web group

On the first screen, we have a view of all available Groups. To consult, click a desired Group.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 17: Web setup user group

On the next screen, we have all the Group fields open for modifications. To modify, change the desired information and select one of the three button options:

* Save and add another: Will save the changes and open a blank User screen to add a new Group record.
* Save and Continue editing: Will save the changes and continue on the Group screen.
* Save: Will save the changes and return to the screen with the list of Groups.

In the History button, we can consult all the modifications carried out in the Group, this function will be important to track modifications and audit the process.

A picture containing text, receipt, screenshot, font

Description automatically generated

Figure 18: Web user group history

The DELETE button will permanently delete the Group record.

Caution: when deleting a Group, the system will also delete all records dependent on that Group.

Deletion can also be performed en bloc. On the Users Group screen, select all the Group you want to delete, choose the Delete Selected Group action and click on the GO button.

Be careful, this elimination operation will be definitive for the Group and for all other records dependent on it, as already explained.

For the Group, we will have on filter locations:

* Located at the top of the Group List screen where we can search broadly.

To add new Group, we will have three different ways:

* by the + Add button on the left sidebar.
* Through the ADD GROUP + button in the right field of the Group list.
* Via the Save and add another button located within a Group record.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 19: Web user group permissions

After entering the username, and password and saving, the system will be directed to the user details page.

A group combines different table accesses and access types. select the tables, type by the functional relationship on the left, and click the arrow to take to the box on the right. All combinations added in the correct box will be assigned to users who inherit this access group.

After performing the new parameterizations, save the new Group.

# Server Application

The Server module serves as the backbone of the IGEM project, facilitating the seamless integration of external data into the comprehensive Exposomes and Genomics knowledge base. With its powerful ETL capabilities and efficient database management, the Server module ensures the harmonious incorporation of diverse datasets into the project ecosystem.

By leveraging advanced ETL techniques, the Server module extracts valuable information from external sources, transforms it into a standardized format, and loads it into the central knowledge base. This allows researchers and users to access and analyze a wealth of integrated data related to Exposomes and Genomics effortlessly.

Furthermore, the Server module assumes the critical responsibility of maintaining and managing the IGEM database, ensuring its accuracy, reliability, and accessibility. It oversees the seamless incorporation of new data, performs updates, and organizes the information in a coherent and user-friendly manner.

Through the collaborative efforts of the Server module, IGEM continues to expand its reach and depth of knowledge, empowering researchers and stakeholders to explore the intricate connections between Exposomes and Genomics. By bridging the gap between external data sources and the project’s standardized knowledge base, the Server module serves as a catalyst for groundbreaking discoveries and valuable insights.

This user guide will provide you with comprehensive instructions on utilizing the features and functionalities offered by the Server module, enabling you to harness the full potential of the integrated Exposomes and Genomics database. Let’s embark on this transformative journey together and unlock the power of data-driven exploration in the realm of IGEM.

## Master Data

The master data module plays a vital role in the functioning of the system by directing data flow, filtering information, and establishing connections with the knowledge base. It enables efficient data collection and integration processes while facilitating effective filtering and linking of terms.

Before initiating the data collection, it is necessary to parameterize the master data module, which can be done either in batch processing or through an intuitive web interface. This allows users to configure the module according to their specific requirements, providing flexibility and ease of use. The configuration of master data involves the following components, each building upon the previous one:

* Datasource: Define the source of the data, specifying its origin or location from which it will be collected.
* Connector: Establish a connection between the system and the designated data source, enabling seamless data retrieval and integration.
* Terms: Define individual terms or keywords relevant to the system’s knowledge base. Terms act as key identifiers for data retrieval and linkage. Each term can be associated with attributes such as Group and Category, which help users filter and organize terms based on specific criteria or themes.
* Prefix: Specify prefixes or identifiers to be appended to certain terms, enhancing their contextual meaning and facilitating accurate data interpretation.

By configuring the master data module in this manner, users gain the ability to intelligently handle data origin, perform precise filtering based on Groups and Categories, and establish seamless connections to the knowledge base. This ensures the accuracy, relevance, and integrity of the data, empowering comprehensive analysis and meaningful insights.

The user-friendly approach to managing master data within the Server module enables users to leverage the full potential of the IGEM system. It provides efficient data integration, discovery, and exploration, while facilitating flexible filtering and organization of terms based on their associated attributes. This empowers users to efficiently navigate and extract relevant information from the knowledge base, facilitating their research and analysis activities.

### Data Source

Datasource master data refers to an external data source and groups of Connectors over the same domain. The connection between IGEM and external data sources will be established by Connectors.

Datasource is used for selecting and grouping queries and future security and authentication features.

The Datasource data will be stored in the ge\_datasource table of the IGEM DB defined in the initial parameters. The available fields are:

* *ID*: GE.db internal key
* *Datasource*: Abbreviated name of the Datasource
* *Description*: Description for identifying and consulting the Datasource
* *Category*: Category to help identify and group the Datasource
* *Website*: Electronic address of the maintainer of the available data

The inclusion of new data can be performed via the process sql On the command line:

$ python manage.py sql --load\_data "table='datasource, path='{your\_path}/datasource.csv'"

Other commands and functions for manipulating master data can be found in the database management tab.

CAUTION: As GE.db is a correlational base with key integrity, all records linked to the deleted data will also be deleted, which includes Connector and TermMap information

Web Interface

Through IGEM’s friendly web interface, it will be possible to carry out Datasource management activities. Activate the IGEM web service if you have not already done so. Go to the igem folder and type the command line:

$ python manage.py runserver

>>> Watching for file changes with StatReloader

Performing system checks...

System check identified no issues (0 silenced)

March 24, 2023 - 12:56:26

Django version 4.1.5, using settings 'src.settings'

Starting development server at <http://127.0.0.1:8000/>

Quit the server with CONTROL-C.

If it returns a port error, you can specify a different port:

$ python manage.py runserver 8080

Access the address in the link provided in Starting development server. Significantly, this address may vary depending on the initial settings performed during installation.

After user authentication and on the initial administration screen, select an option Datasource. On the Datasource screen, we will have options to consult, modify, add and eliminate Datasource.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 20: Web datasouce

On the first screen, we have a view of all available Datasource. To consult, click a desired Datasource.

A picture containing text, screenshot, receipt

Description automatically generated

Figure 21: Web datasouce change

On the next screen, we have all the Datasource fields open for modifications. To modify, change the desired information and select one of the three button options:

* Save and add another: Will save the changes and open a blank Datasource screen to add a new Datasource record.
* Save and Continue editing: Will save the changes and continue on the Datasource screen.
* Save: Will save the changes and return to the screen with the list of Datasource.

In the History button, we can consult all the modifications carried out in the Datasource, this function will be important to track modifications and audit the process.

A picture containing text, screenshot, receipt, font

Description automatically generated

Figure 22: Web datasource change history

The DELETE button will permanently delete the Datasource record.

Caution: when deleting a Datasource, the system will also delete all records dependent on that Datasource, which include Connector, Parameterizations of transformations and TermMaps

Deletion can also be performed en bloc. On the Datasource List screen, select all the Datasource you want to delete, choose the Delete Selected Datasource action and click on the GO button.

Be careful, this elimination operation will be definitive for the Datasource and for all other records dependent on it, as already explained.

A screenshot of a web page

Description automatically generated with low confidence

Figure 23: Web datasouce delete

To add new Datasource, we will have three different ways:

* by the + Add button on the left sidebar.
* Through the ADD DATASOURCE + button in the right field of the Datasource list.
* Via the Save and add another button located within a Datasource record.

For the Datasource, we will have two filter locations:

* First located at the top of the Datasource List screen where we can search broadly.
* Second on the right sidebar, being able to select by category of Datasource.

### Connector

A Connector record stores all information needed for external extraction and controls the links of related terms by the mapping process. The control of external data extraction occurs by Connector. Each Connector will consist of the fields:

* *Datasource*: Grouping of Connector, controlling access authentication.
* *Connector*: Abbreviation for Connector identification
* *Description*: Brief description of the purpose of Connector
* *Enabled*: Flag informing if the Connector is active and will be considered in the ETL process

Group Attributes: Fields to control extraction path and file type

* *Source path from Internet*: Flag to route the extraction path. If enabled, it will be via HTTP and disabled. It will be considered as a local file path.
* *Source path*: Path where the Connector file is hosted
* *Source file name*: name of the file with the original data
* *Source file format*: file format with the original data. This information will be imported for conversion treatment to the data ingestion format in the ETL process. If compressed, inform only the compression format, type ZIP, GZ.
* *Source file sep*: inform the type of file separator if any. For tabular division, use /n
* *Source file skip row*: Inform the number of lines eliminated in the ETL process. Many files have structural information in their first lines that are not needed in the ETL process.
* *Source compact*: Flag to control if the file is compressed. If not marked, it is considered an uncompressed file.
* *Target file name*: Name of the file after unzipping
* *Target file format*: File format after unzipping, this field will be the actual file format, type CSV, TXT
* *Keep file*: Flag, if selected, will keep the file after data processing for future reference. It is essential to analyze the storage space consumption, as keeping the files may consume unnecessary space. New updates will overwrite existing files.

Group Columns: Controls rules for handling the extracted data in a format compatible with the GE.db system. Consider only columns with standard MEsH NIH codes.

* *Column Sequence*: Number of the column that receives the rule
* *Column Name*: Column name to guide and help identify the applied rule
* *Active*: a flag that informs if the practice is active
* *PREFIX*: inform the prefix of the word that will be considered and added to the column information.

The inclusion of new data can be performed via the process db . On the command line:

$ python manage.py db --load\_data "table='connector,

path='{your\_path}/connector.csv'"

Other commands and functions for manipulating master data can be found in the database management tab.

CAUTION: As GE.db is a correlational base with key integrity, all records linked to the deleted data will also be deleted, which includes Rules and TermMap information

Web Interface

On the Connector screen, we will have options to consult, modify, add and eliminate Connector.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 24: Web connector

For the Datasource, we will have two filter locations:

* First located at the top of the Datasource List screen where we can search broadly.
* Second on the right sidebar, being able to select by category of Datasources.

To consult, click a desired Connector.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 25: Web Connector list

To add new Datasource, we will have three different ways:

* by the + Add button on the left sidebar.
* Through the ADD Datasource + button in the right field of the Datasources list.
* Via the Save and add another button located within a Datasource record.

The Connector interface is divided into three parts with each one performing a different activity in the ETL (Extraction, Transformation and Load) process of external data to IGEM.

PART 1

It has fields for identifying the Connector, among them we will have which Datasource the Connector belongs, an abbreviation that will identify the Connector in processes and queries, and a description and a flag for activating or not the Connector.

If the Activate FLAG is not selected, the Connector will not perform new extraction of external data, but all data already loaded will continue to be available for queries

A picture containing text, screenshot

Description automatically generated

Figure 26: Web change connector – part 1

PART 2

The second part will be composed of the attributes to make the connection with external sources and format of the extracted data.

Each field has already been detailed at the beginning of this session.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 27: Web change connector - part 2

PART 3

The third part stores the transformation rules that the IGEM system needs to interpret the input data and how to handle each column.

If the input files are not in tabular format, as in the case of files with XML extension, the COLLECT process will transform into tabular and keep a new file in the PSA folder as detailed in the [Collect](#_bookmark57)

The sequence of columns does not need to be in order; however it will be necessary to identify the first column with the number 0 and so on.

If a column is not informed, the system will understand it as a basic rule and process the column as detailed in the process [Prepare](#_bookmark59)

To delete a column rule, check the DELETE? On the desired line and save the Connector. To add a new column rule, select the + Add another DST column button

The column name will be a header identifier used in the file generated during the ETL process. This file can be kept for future queries and Data Lake projects for example, as it has already passed through a normalization layer.

The Active field tells the ETL process to read and handle the source column. If the line is not marked, it will be discarded during the PREPARE process and not going through MAPREDUCE.

Important to balance workload, storage space, and other factors before keeping a column active. Many columns are unnecessary information for the purpose of term mapping and would not make sense to keep them active.

The prefix field adds an identification to fields composed only of codes: For example, we will have a column with a code of a Numeric Gene only, for the IGEM to identify that this sequence of numbers is a Gene, we add a prefix that differentiates it from other types of also numerical information.

It is important to maintain a synchronism between the prefixes and the registration of Keyge and Words so that the MAPREDUCE process can identify and ingest the terms found into the GE.db base.

We can perform two maintenance operations on the prefixes using the change buttons and quickly add a new prefix on the Connector screen without having to leave the register. Prefixes are also treated as master data within IGEM.

The SINGLE WORD option field has the function of informing the ETL process if this column is composed of a single word and does not need to go through the process of breaking up and identifying terms. This makes the process faster and reduces memory consumption during the Connector ETL. When we have selected a prefix for the column, the SINGLE WORD will have no function, because with the prefix it is assumed that it is a KEYGE without the replacement processing by the KEYWORD.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 28: Web change connector - part 3

Save Connector

On the next screen, we have all the Connector fields open for modifications. To modify, change the desired information and select one of the three button options:

* Save and add another: Will save the changes and open a blank Connector screen to add a new Connector record.
* Save and Continue editing: Will save the changes and continue on the Connector screen.
* Save: Will save the changes and return to the screen with the list of Connector.

Delete Connector

The DELETE button will permanently delete the Datasource record.

Caution: when deleting a Datasource, the system will also delete all records dependent on that Datasource, which include Connectors, Parameterizations of transformations and KEYLINKS

Deletion can also be performed en bloc. On the Datasource List screen, select all the Datasource you want to delete, choose the Delete Selected Database action and click on the GO button.

Be careful, this elimination operation will be definitive for the Datasources and for all other records dependent on it, as already explained.

### Group

The group master data acts as a qualitative characteristic for the [Term](#_bookmark49) being the highest level of the hierarchical structure, followed by the [Category](#_bookmark47) and then the [Term](#_bookmark49).

The system uses the Group information as a filter in queries and other interfaces. An example of the use of Group will be the [Gene Exposome Report](#_bookmark26), in which the system will use the Group to select which Term will be considered as Exposome.

The Group data will be stored in the ge\_group table of the IGEM DB defined in the initial parameters. The available fields are:

* ID: GE.db internal key
* group: Abbreviated name of the Group
* Description: Description for identifying and consulting the Group

The inclusion of new data can be performed via the process sql On the command line:

$ python manage.py db --load\_data "table='term\_group,

path='{your\_path}/term\_group.csv'"

Other commands and functions for manipulating master data can be found in the database management tab.

CAUTION: As GE.db is a correlational base with key integrity, all records linked to the deleted data will also be deleted, which includes Term and TermMap information

As with Connector and Data Source, Group master data can also be managed via the web interface.

### Category

Category master data acts as a grouping of [Term](#_bookmark49) at a lower level than the [Group](#_bookmark6).

The system uses the Category information as a filter in queries and other interfaces. An example of the use of Category will be the [Gene Exposome Report](#_bookmark26), in which the system will use the Category to select all Gene **KEYGE**.

The Category data will be stored in the ge\_category table of the IGEM DB defined in the initial parameters. The available fields are:

* *ID*: GE.db internal key
* *Category*: Abbreviated name of the Category
* *Description*: Description for identifying and consulting the Category

The inclusion of new data can be performed via the process db . On the command line:

$ python manage.py db --load\_data "table='term\_category,

path='{your\_path}/term\_category.csv’”

Other commands and functions for manipulating master data can be found in the database management tab.

CAUTION: As GE.db is a correlational base with key integrity, all records linked to the deleted data will also be deleted, which includes Term and TermMap information

As with Connector and Data Source, Group master data can also be managed via the web interface.

### Term

Term is the main component in GE.db and GE.filter and was created to specify a search term in external data sources. A Term can be assigned to a gene, a chromosome, an SNP, a disease, a chemical, an environmental factor, or any other term necessary to keep in the GE.db knowledge base.

A Term will have as attributes the Group and Category records to qualify and group, helping during searches, queries, and analysis of the GE.db knowledge base.

A Term inside GE.db can be kept as a code number, a prefix + code, or even a word, depending exclusively on the initial planning adopted. Thus allowing high flexibility in the use of the IGEM system.

The system has an interface for mapping external words to a Term, with this link having several external combinations for a single Term. The system does not allow mapping the same external word to more than one Term, a process necessary to guarantee the integrity of the knowledge base.

As described in the introduction, the purpose of GE.db will be to search an external record for all Terms found, correlate these Terms and maintain a frequency and origin, allowing, like GE.filter, to perform searches for combinations between Term in different external data sources quickly and easily.

The Term data will be stored in the ge\_keyge table of the IGEM DB defined in the initial parameters. The available fields are:

* *ID*: GE.db internal key
* *Term*: Abbreviated name of the Term
* *Description*: Description for identifying and consulting the Term
* *Category\_id*: foreign\_key from ge\_category
* *Group\_id*: foreign\_key from ge\_group

The inclusion of new data can be performed via the process db . On the command line:

$ python manage.py db --load\_data "table='term,

path='{your\_path}/term.csv'"

Other commands and functions for manipulating master data can be found in the database management tab.

CAUTION: As GE.db is a correlational base with key integrity, all records linked to the deleted data will also be deleted, which includes TermMap amd WordMap information

As with Connector and Data Source, Group master data can also be managed via the web interface.

### Prefix

Prefixes play a fundamental role in the logic of the IGEM system for the correct identification of ref:*Term*.

Prefixes are assigned as input structure columns by [Connector](#_bookmark44). Necessary due to identification only by the code of terms mapped to [Term](#_bookmark49) and that however conflict with other categories of [Term](#_bookmark49).

For the correct identification of the [Term](#_bookmark49), the system will add a prefix to the code located in the source Connector before MAPREDUCE processing.

Keeping without a record in none, it is important that the record is used during the IGEM ETL where additional cases from prefix to source code do not occur.

The Prefix data will be stored in the ge\_prefixopc table of the IGEM DB defined in the initial parameters. The available fields are:

* *pre\_value*: prefix name

The inclusion of new data can be performed via the process db . On the command line:

$ python manage.py db --load\_data "table='term,

path='{your\_path}/term.csv'"

Other commands and functions for manipulating master data can be found in the database management tab.

CAUTION: As GE.db is a correlational base with key integrity, all records linked to the deleted data will also be deleted, which includes Connector columns rules.

As with Connector and Data Source, Group master data can also be managed via the web interface.

### Word-Term

The WordTerm data will be stored in the ge\_WordTerm table of the IGEM DB defined in the initial parameters. The available fields are:

*word*: The word or set of words that convert to Term (unique)

*term\_id*: foreign\_key to ge\_Term that link word with one Term

*commute*: Flag used to convert. If it is the same criterion between Term and WORD, disable this flag to reduce memory consumption during the ETL process.

*status*: Flag to activate the relationship

The inclusion of new data can be performed via the process db . On the command line:

$ python manage.py db --load\_data "table='term',

path='{your\_path}/term.csv'"

Other commands and functions for manipulating master data can be found in the database management tab.

As with Connector and Data Source, Group master data can also be managed via the web interface.

## Database Management

With the sql process, it will be possible to carry out data extraction operations, data loading, deletion, and cleaning of IGEM tables. The available tables are:

* datasource
* connector
* term\_group
* term\_category
* term
* ds\_column
* prefix
* wordterm
* termmap
* wordmap

Python function

get\_data

The get\_data() function allows extracting data from the GE database and loading this data into a Pandas DataFrame structure or CSV File.

It has an intelligent filter mechanism that allow you to perform data selections simply through a conversion layer of function arguments and SQL syntax. This allows the same input arguments regardless of implemented database management system.

Parameters:

Only the table parameter will be mandatory, the others being optional, and will model the data output. In the case of only informing the table, the function will return a DataFrame with all the columns and values of the table.

table: str datasource, connector, ds\_column, term\_group, term\_category, term, prefix, wordterm, termmap, wordmap

path: str With this parameter, the function will save the selected data in a file in the directory informed as the parameter argument. In this scenario, data will not be returned in the form of a Dataframe; only a Boolean value will be returned, informing whether the file was generated or not

columns: list[“str”] Columns that will be selected for output. They must be informed with the same name as the database. It is possible to load other data from other tables as long as it correlate. For example, suppose the table only has the term field and not the category field. In that case, you can inform as an argument: “term\_id term\_category\_id category”, the system selected the ID of the term, consulted the ID of the category in the Term table, and went to the Category table to choose the category

columns\_out: list[“str”] If you want to rename the header of the output fields to more familiar names, you can use this parameter, passing the desired names in the same sequential sequence in the parameter columns

datasource: Dict{“str”:list[”str”]} Filter argument. It is used to filter datasource, with the dictionary key being the selection argument and the dictionary value being the datasources selected as the filter. Without this parameter, the function will return all datasources

connector: Dict{“str”:list[”str”]} Filter argument. It uses the same logic as the datasource, but applied to the connector field

word: Dict{“str”:list[”str”]} Filter argument. It uses the same logic as the datasource, but applied to the word field

term: Dict{“str”:list[”str”]} Filter argument. It uses the same logic as the datasource, but applied to the term field

term\_category: Dict{“str”:list[”str”]} Filter argument. It uses the same logic as the datasource, but applied to the term\_categorty field

term\_group: Dict{“str”:list[”str”]} Filter argument. It uses the same logic as the datasource, but applied to the term\_group field

Return:

Pandas Dataframe or Boolean (If the parameter path is informed, the function will generate the file; if successful, it will return the TRUE. Otherwise, it will return FALSE)

Examples:

>>> from igem.server import sql

>>> sql.get\_data(

table=”datasource”,datasource={“datasource in”: [“ds\_01”,”ds\_02”]},

columns=[“id”,”datasource”], columns\_out=[“Datasource ID”,

“Datasource Name”], path=”{your\_path}/datasource.csv”

)

>>> df = sql.get\_data( table="connector",

connector={"connector start": ["conn\_ds"]},

datasource={"datasource\_id datasource in": ["ds\_01"]},

columns=["connector", "status"]

)

>>> x = sql.get\_data( table="termmap",

term={"term\_id term": "chem:c112297"}, path="{your\_path},

)

load\_data

Loads data from a CSV file into the IGEM database. This process does not update existing data, it only inserts new records.

Parameters:

table: str datasource, connector, ds\_column, term\_group, term\_category, term, prefix, wordterm, termmap, wordmap

path: str full path and file name to load

Layout of data file:

Datasource: (datasource, description, category, website)

Connector: (connector, datasource, description, update\_ds, source\_path, source\_web, source\_compact, source\_file\_name, source\_file\_format, source\_file\_sep, source\_file\_skiprow, target\_file\_name, target\_file\_format)

Ds\_column: (connector, status, column\_number, column\_name, pre\_value, single\_word)

Term\_group: (term\_group, description)

Term\_category: (term\_category, description)

Term: (term, category, group, description)

Prefix: (pre\_value)

Wordterm: (term, word, status, commute)

Termmap: (ckey, connector, term\_1, term\_2, qtd\_links)

Wordmap: (cword, datasource, connector, term\_1, term\_2, word\_1, word\_2, qtd\_links)

We can generate an example file with the get\_data() function and manipulate and load it with the new data.

Return:

Boolean: (TRUE if the process occurred without errors and FALSE if had some errors). Examples:

>>> from igem.server import sql

>>> sql.load\_data(

table="datasource” path=”{your\_path}/datasource.csv”)

delete\_data

Allows deleting a record from the given table. The deletion will be carried out in all records related to the informed parameter. For example, if we delete a datasource, the connectors, ds\_columns, and termmap associated with the datasource will be deleted.

Parameters:

Only the table parameter will always be requested, the others will depend on the selected table, functioning as a record that will be eliminated.

table: str (datasource, connector, ds\_column, term\_group, term\_category, term, prefix, wordterm, termmap, wordmap, workflow)

(Filter argument. It is used to filter the field, with the dictionary key being the selection argument and the dictionary value being the field selected as the filter. Without this parameter, the function will return all values of the field.)

Return:

Boolean: (TRUE if the process occurred without errors and FALSE if had some errors). Examples:

>>> from igem.server import sql

>>> sql.delete\_data( table='datasource',

datasource={'datasource in': [ds\_01]}

)

truncate\_table

Will delete all records from a table, never use this function, with excess if the need is to restart a new instance of the database, free up log table space or in test environments.

Parameters:

table: str (datasource, connector, dst, term\_group, term\_category, term, prefix, wordterm, termmap, wordmap, workflow, logs)

If inform table=”all”, the function will truncate all table on GE database. The other tables of the IGEM system will be maintained.

Return:

Boolean: (TRUE if the process occurred without errors and FALSE if had some errors). Examples:

>>> from igem.server import sql

>>> sql.truncate\_table(

table='datasource'

)

backup

Backup the database with the internal keys. It can be performed at once for all GE.sql tables

Parameters:

table: str (datasource, connector, dst, term\_group, term\_category, term, prefix, wordterm, termmap, wordmap, workflow, logs)

path\_out: str Folder path to store the generated backup files

If inform table=”all”, the function will backup all table on GE database.

Return:

Boolean: (TRUE if the process occurred without errors and FALSE if had some errors).

Examples:

>>> import igem

>>> igem.server.sql.backup( table="", path\_out="/root/back")

restore

Restore the database with the internal keys. It can be performed at once for all GE.sql tables

Parameters:

table: str (datasource, connector, dst, term\_group, term\_category, term, prefix, wordterm, termmap, wordmap, workflow, logs)

path\_out: str Folder path to store the generated backup files

If inform table=”all”, the function will restore all table on GE database.

Return:

Boolean: (TRUE if the process occurred without errors and FALSE if had some errors).

Examples:

>>> import igem

>>> gem.server.sql.restore( table="", path\_out="/root/back")

Command Line

Within the parameters, inform the same ones used for the functions, as well as the arguments, example:

$ python manage.py sql --get\_data 'table="datasource", datasource={“datasource in”: [“ds\_

Get data:

$ python manage.py sql --get\_data {parameters}

Load data:

$ python manage.py sql --load\_data {parameters}

Delete data:

$ python manage.py sql --delete\_data {parameters}

Delete all table:

$ python manage.py sql --truncate\_table {parameters}

Backup (get data with internal ID):

$ python manage.py sql --backup {parameters}

Restore (load data with internal ID):

$ python manage.py sql --restore {parameters}

## ETL

The ETL process is responsible for fetching data from external sources, transforming it into a compatible standard, searching for term relationships, and writing the data to GE.db. It consists of five distinct phases to efficiently manage resources and ensure successful execution:

* Collect: This phase involves gathering data from external sources.
* Prepare: In this phase, the collected data is processed and prepared for further transformation and loading.
* Map: The data is mapped to relevant terms and categories, establishing relationships between them.
* Reduce: Unnecessary or redundant data is filtered out, ensuring that only relevant information is retained.
* Workflow: This phase coordinates the entire ETL process, orchestrating the execution of the preceding phases. Each phase is explained in detail in the respective files:

### Collect

The “Collect” process is responsible for selecting active connectors and checking if new versions of data are available. It performs the following tasks:

Connector Selection: The process selects active connectors to fetch data from various sources. Data Extraction: If a new version of the data is available, the process extracts the latest data.

File Handling: If necessary, the extracted file is uncompressed and stored in the Persists Storage Area (PSA).

Logs and Version Controls: The process updates logs and version controls to track the execution and status of each connector.

Currently, the execution version of the steps in the web interface is still under development.

The process is executed through the command line using the following script:

$ python manage.py etl –collect {all or connector}

If the “all” option is used, the process collects data for all active connectors in the master data table. If a specific connector is provided, only that connector’s data will be collected.

### Prepare

This second phase of the process aims to transform the original data, thus reducing the need for computational resources in the subsequent steps. Based on the briefly configured connector parameters, in this phase, we will have:

* Deleting header lines
* Deleting unnecessary columns
* Transforming ID Columns with Suffix Identifiers
* Replacement the terms
* Deletion of the original file

The output will be a new temporary file for consumption in the next phase:

# python manage.py etl –prepare {all or connector}

It will start the data preparation phase for all connectors or just one specified. Essential to have the file in PSA. Otherwise, the system will display a warning:

The reset option will reset the control for all or a specific connector in the preparation phase and the two later ones.

### MAP

The map will process each line of the file and combine all found words. The result will be recorded in the WordMap table:

# python manage.py etl –map {all or connector}

It will start the data term switching phases for all connectors or just one specified. This is why it is essential to have the file in PSA. Otherwise, the system will display a warning:

The reset option will reset the control for all or a specific connector to the switching phase and the next phase

### Reduce

Reduce is the last step of the process. It has a mechanism to find Term (terms) per line called Mapper and then activates the Reducer subprocess that will count the number of links found in the connector. After all processing, the result will be recorded in the Keylinks table. It is important to note that the new data will fully replace the previous data in the processed connector.

$ python manage.py etl –reduce {all or connector}

The file must be in PSA, otherwise, the system will display a warning. It will start the MapReduce phase of data terms for all Connector or just a specific one. In this phase, there is a large consumption of memory and processing, so it will be essential to allocate resources compatible with the size of the processed data.

Reset option will Press the control to all or a specific connector in the current phase.

### Workflow

In addition to the reset commands shown above to control the process workflow, the system also has a web interface where the user can consult and manage the flow and status of the phases.

Browse http://127.0.0.1:8000/admin/ge/wfcontrol/ or select the Dataset – Workflow option in the GE application in the first one, Connectors that have already been started will be started, with the following references:

* DS STATUS: informs whether the Connector is active or not for processing the 4 phases
* Connector: Abbreviation for Connector
* Last Update Dataset: Date of the previous data update.
* Source file version: Version of the final processed file.

The following four columns display the statuses still of the processes by phase. The green symbol indicates the status-completed successfully and the group not processed.

Data were only available in the GE.be database after all phases had been successfully executed.

A screenshot of a computer

Description automatically generated

Figure 29: Web workflow control

When selecting a Connector, it will be a details screen allowing the specific opening by field.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 30: Web workflow control change

Important: it will not be necessary to include new Connectors in the Workflow monitor. The system will automatically create a new control after the first load of data from the Connector. If one of the workflow records is deleted, it will also be completed after the next data load.

To ensure a successful ETL process, proper parameterization and accurate master data entries are crucial. These files will guide you through each phase, helping you understand and execute the ETL process effectively

# GE Application

The GE module is a powerful component of the system that encompasses various functionalities related to data processing and analysis in the context of genomics and exposomes. It consists of two important components: GE.db and GE.Filter.

1. The GE.db provides direct access to the underlying database tables, allowing users to retrieve information directly from the IGEM Client DB. It offers the capability to query and analyze data stored in the database tables, empowering users to efficiently extract specific information for their research purposes. Additionally, the GE.db facilitates synchronization between the IGEM Client DB and the Hall Lab DB Server, ensuring the availability of up-to-date data. Users can choose between offline and online synchronization options based on their requirements.
2. The GE.Filter offers a range of functions to filter and retrieve information from the IGEM Client DB, specifically focusing on the relationships and reports related to genomics (G), exposomes (E), and their interactions (GxE and ExE).

By leveraging the functionalities of the GE.db and GE.Filter, researchers can efficiently access and analyze data, extract relevant information, and explore the relationships between various elements in the genomics and exposome domains.

Note: The GE module is part of a larger system, and additional submodules and functionalities may exist to further enhance the research and analysis capabilities in genomics and exposomes.

## GE.db

### Direct Access to Tables

Enables direct access to the database tables, allowing users to retrieve information directly from the IGEM Client DB.

This functionality provides a convenient way to query and analyze the data stored in the database tables.

By leveraging this function, users can efficiently retrieve specific information from the IGEM Client DB and utilize it for their research and analysis purposes.

The available tables are:

* datasource
* connector
* term\_group
* term\_category
* term
* prefix
* wordterm
* termmap
* wordmap

get\_data

The get\_data() function allows extracting data from the GE database and loading this data into a Pandas DataFrame structure or CSV File.

It has an intelligent filter mechanism that allow you to perform data selections simply through a conversion layer of function arguments and SQL syntax. This allows the same input arguments regardless of implemented database management system.

Parameters:

Only the table parameter will be mandatory, the others being optional, and will model the data output. In the case of only informing the table, the function will return a DataFrame with all the columns and values of the table.

table: str datasource, connector, ds\_column, term\_group, term\_category, term, prefix, wordterm, termmap, wordmap

path: str With this parameter, the function will save the selected data in a file in the directory informed as the parameter argument. In this scenario, data will not be returned in the form of a Dataframe; only a Boolean value will be returned, informing whether the file was generated or not

columns: list[“str”] Columns that will be selected for output. They must be informed with the same name as the database. It is possible to load other data from other tables as long as it correlate. For example, suppose the table only has the term field and not the category field. In that case, you can inform as an argument: “term\_id term\_category\_id category”, the system selected the ID of the term, consulted the ID of the category in the Term table, and went to the Category table to choose the category

columns\_out: list[“str”] If you want to rename the header of the output fields to more familiar names, you can use this parameter, passing the desired names in the same sequential sequence in the parameter columns

datasource: Dict{“str”:list[”str”]} Filter argument. It is used to filter datasource, with the dictionary key being the selection argument and the dictionary value being the datasources selected as the filter. Without this parameter, the function will return all datasources

connector: Dict{“str”:list[”str”] Filter argument. It uses the same logic as the datasource, but applied to the connector field

word: Dict{“str”:list[”str”]} Filter argument. It uses the same logic as the datasource, but applied to the word field

term: Dict{“str”:list[”str”]} Filter argument. It uses the same logic as the datasource, but applied to the term field

term\_category: Dict{“str”:list[”str”]} Filter argument. It uses the same logic as the datasource, but applied to the term\_categorty field

term\_group: Dict{“str”:list[”str”]} Filter argument. It uses the same logic as the datasource, but applied to the term\_group field

Return:

Pandas Dataframe or Boolean (If the parameter path is informed, the function will generate the file; if successful, it will return the TRUE. Otherwise, it will return FALSE)

Examples:

>>> from igem.ge import db

>>> db.get\_data(

table=”datasource”,

datasource={“datasource in”: [“ds\_01”,”ds\_02”]}, columns=[“id”,”datasource”],

columns\_out=[“Datasource ID”, “Datasource Name”], path=”{your\_path}/datasource.csv”

)

>>> df = db.get\_data( table="connector",

connector={"connector start": ["conn\_ds"]},

datasource={"datasource\_id

datasource in": ["ds\_01"]}, columns=["connector", "status"]

)

>>> x = db.get\_data(

table="termmap",

term={"term\_id term": "chem:c112297"},

path="{your\_path},

)

Command Line

Within the parameters, inform the same ones used for the functions, as well as the arguments, example:

$ python manage.py db --get\_data 'table="datasource",

datasource={“datasource in”: [“ds\_0”]}’

Get data:

$ python manage.py db --get\_data {parameters}

### Sync GE.db

The second function of the Database Management is to synchronize the IGEM Client DB with the latest data from the Hall Lab DB Server.

This synchronization process ensures that the IGEM Client DB is up to date with the most recent information available. The function offers both offline and online synchronization options.

Offline Sync:

In the offline synchronization mode, users manually acquire the necessary DB files from a designated source. They can obtain the latest versions of the DB files from an authorized repository and update the IGEM Client DB accordingly. This mode is suitable for situations where internet connectivity is limited or when users prefer to have full control over the synchronization process.

Examples:

>>> from igem.ge import db

>>> db.db.sync\_db(table="all", source="{your\_path}")

Online Sync:

The online synchronization mode automates the process of fetching the latest data from the web repository. The submodule accesses the web repository and retrieves the most recent versions of the DB files, ensuring that the IGEM Client DB is synchronized with the Hall Lab DB Server. This mode is ideal for users who prefer a seamless and automated synchronization process, without the need for manual intervention.

Examples:

>>> from igem.ge import db

>>> db.db.sync\_db(table="all")

## GE.filter

The GE.db submodule provides researchers with a comprehensive set of tools to access and synchronize the IGEM Client DB. Whether it’s directly querying database tables or ensuring up-to-date information through synchronization, this submodule facilitates efficient data management and enhances the research capabilities of users.

The GE.filter module serves as a crucial component of the GE (Genomics and Exposomes) system, specifically designed to facilitate the exploration and analysis of the Knowledge Database, referred to as GE.db. This Knowledge Database contains a wealth of information related to genomics, exposomes, and their interconnectedness.

By utilizing the functions provided by GE.filter, users gain the ability to efficiently retrieve and filter data from GE.db, enabling them to uncover valuable insights and relationships.

Whether it’s examining term connections, exploring reports on GxE (Gene-Environment) interactions or ExE (Exposome-Environment) associations, accessing gene-level information in relation to SNPs (Single Nucleotide Polymorphisms), or converting words to IGEM terms, the GE.filter module empowers users to extract pertinent information and generate comprehensive reports.

These functionalities play a crucial role in understanding the complex interplay between genomics and exposomes, supporting various research and analytical endeavors

term\_map: The term\_map function provides the mapping between IGEM terms and their associated metadata. It enables you to explore the attributes and properties of different terms stored in the GE.db, aiding in data exploration and analysis.

word\_to\_term: This function allows you to convert individual words or a list of words into their corresponding IGEM terms. It helps in mapping user-provided words to the relevant terms stored in the GE.db, providing a standardized representation for further processing.

gene\_exposome: The gene\_exposome function retrieves information about the gene-exposome relationship from the GE.db. It helps in understanding the interaction between genes and environmental factors, facilitating studies related to genomics and exposomes.

snp\_exposome: With the snp\_exposome function, you can access reports and information about the impact of single nucleotide polymorphisms (SNPs) on exposomes. It helps in understanding the influence of genetic variations on environmental exposures and their potential effects on health outcomes.

word\_map: In the Word-Map function, all words mapped from an external dataset are stored in a temporary table within GE.db. This feature proves particularly useful for researchers who wish to list the relationships between words on a record-by-record basis, without relying on the IGEM pre-computing mapping process that converts external words to the standardized IGEM Terms. It allows users to perform analysis and retrieve word relationships specific to their research needs. However, it’s important to note that this temporary table should be used judiciously due to its high memory consumption on the database. Users are advised to run the function on a specific dataset, extract the desired relationships for their analysis, and subsequently clean up this information to optimize database performance. By providing a flexible and efficient way to explore word relationships, the Word-Map function empowers researchers in their investigations and enhances their understanding of the data.

### Term Map Report

ge.filter.term\_ma (\*args, \*\*kwargs)

TermMap table query function.

* path\_in: parameter file path with filter information, aggregation, and result file path.
* path\_out: result file path
* term: list of terms to filter passed through the function. If you inform the file with the parameters, the values passed by this parameter will be disregarded.

It may return a boolean value if you have informed an output per file (path\_out) or a DataFrame if you have not informed an output file.

Example

from igem.ge import filter

filter.term\_map(path\_in="../../file.csv", path\_out="../../outcome.csv")

df\_result = filter.term\_map( term=["gene:246126"])

This function queries the TermMap table in GE.db and retrieves relationships between terms. The results can be saved in a specified output file path or returned as a DataFrame.

### Gene Exposome Report

ge.filter.gene\_exposome (\*args, \*\*kwargs)

Queries GE.db and returns links between genes and exposomes based on input parameters or the parameter file.

* path\_in: parameter file path with filter information, aggregation, and result file path.
* path\_out: result file path.
* term: List of terms to filter passed through the function. If you inform the file with the parameters, the values passed by this parameter will be disregarded.

It may return a boolean value if you have informed an output per file (path\_out) or a DataFrame if you have not informed an output file.

Example

from igem.ge import filter

filter.gene\_exposome(path\_in="../file.csv", path\_out="../outcome.csv")

df\_result = filter.gene\_exposome( term=["gene:246126"])

This function queries GE.db and generates results showing links between genes and exposomes based on the provided parameters. The results can be saved in a specified output file path or returned as a DataFrame.

### SNP Exposome Report

ge.filter.snp\_exposome (\*args, \*\*kwargs)

Queries GE.db and returns links between SNPs and exposomes based on input parameters or the parameter file.

* path\_in**:** parameter file path with filter information, aggregation, and result file path.
* path\_out**:** result file path.
* term**:** list of terms to filter passed through the function. If you inform the file with the parameters, the values passed by this parameter will be disregarded.

It may return a boolean value if you have informed an output per file (path\_out) or a DataFrame if you have not informed an output file.

Example

from igem.ge import filter

filter.snp\_exposome(path\_in="../file.csv", path\_out="../outcome.csv")

df\_result = filter.snp\_exposome( term=["gene:246126"])

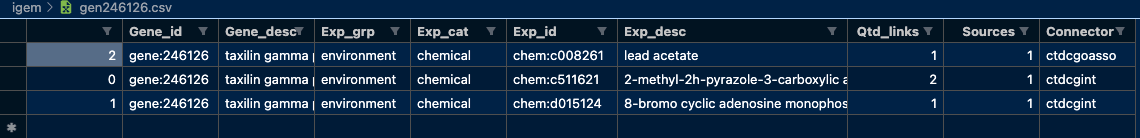


Figure 31: SNP x Exposome Filter

This function queries GE.db and generates results showing links between SNPs and exposomes based on the provided parameters. The results can be saved in a specified output file path or returned as a DataFrame.

### Other Functions and Report

TAG

A TAG in the context of the GE.Filter function is a unique identifier that helps you track and identify the version of the external dataset used in the IGEM query. It serves as a reference to the specific dataset version, allowing you to reproduce the same query in the future with the same dataset version.

When you process (ETL) a specific external dataset, the TAG will indicate the version of the dataset used. For example, if you process one external dataset with version or ETAG 1, the TAG will show that the data comes from this dataset.

In case the IGEM database is updated with a newer version of the external dataset, the TAG will reflect the most recent version. This helps researchers know which external dataset was used in their IGEM query and enables them to control and ensure the consistency of results when they want to replicate the same query in the future.

By referring to the TAG, researchers can track and document the specific dataset version used, providing transparency and facilitating reproducibility in their research.

In summary, the TAG serves as a unique identifier that indicates the version of the external dataset used in the IGEM query, allowing researchers to reproduce the same query with the same dataset version in the future.

Create tag

ge.filter.create\_tag (connectors)

Function to create a TAG with Current connector in TermMap. The IDs generated in the TAG are the WFControl table IDs with Current status.

* connectors: List of connector names.

This function generates a TAG string based on the current connector in TermMap. It takes a list of connector names as input and retrieves the last IDs with a “Current” status from the WFControl table. It then generates a TAG string using the retrieved IDs. The TAG string helps researchers know which external dataset was used in the IGEM query and provides control to use the same version of the dataset in future consultations to IGEM.

The generated TAG string follows the format “GE.db-TAG:<tag\_ids>”, where “<tag\_ids>” is a hyphen-separated string of the WFControl table IDs.

Get tag

ge.filter.get\_tag (tag)

Function to retrieve WFControl data based on a TAG.

* tag: TAG string.

This function retrieves WFControl data based on a TAG string. It takes a TAG string as input and parses the TAG to obtain the corresponding WFControl table IDs. It then retrieves the WFControl data for the specified IDs and returns it as a DataFrame.

Get tag data

ge.filter.get\_tag\_data (tag, path)

Function to save TermMap and WFControl data to CSV files based on a TAG.

* tag: TAG string.
* path: Path to save the files.

This function saves the TermMap and WFControl data associated with a TAG string to CSV files. It takes a TAG string and a path as input. First, it retrieves the corresponding WFControl data using the get\_tag function. Then, it retrieves the TermMap data related to the connector IDs in the WFControl data. Finally, it saves both the TermMap and WFControl data to separate compressed CSV files in the specified path. The function returns True if the files were successfully created and False otherwise.

Parameters File

ge.filter.parameters\_file (path=None)

Generates a model file to be used as a parameter file in query functions

* path**:** path where the file will be generated.

In the file structure, new lines for the index filter can be included with additional values, and each filter line must contain only a single value. The output index and path must be unique, as they will be applied to the entire corresponding field (parameter).

In the example below, let’s select all terms from two data sources from a single group. Also, the Datasource and Connector fields will be aggregated and will not appear on the results:

index,parameter,value

filter,datasource,ds\_01

filter,datasource,ds\_02

filter,connector,

filter,term\_group,Chemical

filter,term\_category,

filter,word,

output,datasource,no

output,connector,no

output,term\_group,

output,term\_category,

output,term,

output,word,no

path,path,/../output\_file.csv

It returns a boolean value if the file was created

Example

from igem.ge import filter filter.parameters\_file(

path="../../folder"

)

This function generates a file template with parameters created in the specified path

# EPC Application

The EPC (Extend Process Call) module in the IGEM software provides a comprehensive set of functionalities that enable users to create an end-to-end pipeline for data analysis. This module offers various tools and functions to load external datasets, perform data description, and modify the data to adapt it to different types of analyses such as EWAS, Association Study, and ExE Pairwise analysis. Here is an overview of the key functionalities offered by the EPC module

## Data Load

Allows users to seamlessly load external datasets into the script. It supports loading data from CSV and TSV files. This functionality enables researchers to integrate their data with the IGEM ecosystem for further analysis.

In the above example, the *from\_tsv* function loads data from a tab-separated file, while the *from\_csv* function loads data from a comma-separated file. Both functions return a DataFrame, where the index column is used for merging. The examples demonstrate how to use these functions to load files and provide information about the number of observations and variables loaded.

Example:

>>> import igem

>>> df = igem.epc.load.from\_tsv('{your\_file.txt}',

index\_col="{your\_index}")

Loaded 22,624 observations of 970 variables

>>> import igem

>>> df = igem.epc.load.from\_csv('{your\_file.csv},

index\_col=”{your\_index}")

Loaded 22,624 observations of 970 variables

## Data Description

Users can obtain a comprehensive description of their datasets. This includes calculating correlations between variables, generating frequency tables for categorical variables, determining data types of variables, calculating the percentage of missing values, computing skewness of variables, and generating summary statistics for variables. These descriptive statistics provide valuable insights into the dataset and help researchers understand its characteristics.

Example:

>>> import igem

>>> correlations = igem.epc.describe.correlations(df, threshold=0.9)

>>> correlations.head()

var1 var2 correlation

0 supplement\_count DSDCOUNT 1.000000

1 DR1TM181 DR1TMFAT 0.997900

2 DR1TP182 DR1TPFAT 0.996172

3 DRD370FQ DRD370UQ 0.987974

4 DR1TS160 DR1TSFAT 0.984733

>>> igem.epc.describe.freq\_table(df).head(n=10)

variable value count

0 SDDSRVYR 2 4872

1 SDDSRVYR 1 4191

2 female 1 4724

3 female 0 4339

4 how\_many\_years\_in\_house 5 2961

5 how\_many\_years\_in\_house 3 1713

6 how\_many\_years\_in\_house 2 1502

7 how\_many\_years\_in\_house 1 1451

8 how\_many\_years\_in\_house 4 1419

9 LBXPFDO <Non-Categorical Values> 1032

>>> igem.epc.describe.get\_types(df).head()

RIDAGEYR continuous

female binary

black binary

mexican binary

other\_hispanic binary

dtype: object

>>> igem.epc.describe.percent\_na(df)

variable percent\_na

0 SDDSRVYR 0.00000

1 female 0.00000

2 LBXHBC 4.99321

3 LBXHBS 4.98730

>>> igem.epc.describe.skewness(df)

Variable type skew zscore pvalue

0 pdias categorical NaN NaN NaN

1 longindex categorical NaN NaN NaN

2 durflow continuous 2.754286 8.183515 2.756827e-16

3 height continuous 0.583514 2.735605 6.226567e-03

4 begflow continuous -0.316648 -1.549449 1.212738e-01

>>> igem.epc.describe.get\_types(df).head()

RIDAGEYR continuous

female binary

black binary

mexican binary

other\_hispanic binary

dtype: object

## Data Modification

Offers a wide range of data modification functions to prepare the dataset for specific analyses. Users can categorize variables based on defined criteria, filter columns based on specific conditions, convert variables to binary or categorical format, merge observations or variables based on specified conditions, move variables within the dataset, record specific values for variables, remove outliers, filter rows with incomplete observations, and perform transformations on variables. These data modification functions enable researchers to tailor the dataset to their analysis requirements.

Example:

>>> import igem

>>> igem.epc.modify.categorize(nhanes)

362 of 970 variables (37.32%) are classified as binary (2 unique values).

47 of 970 variables (4.85%) are classified as categorical (3 to 6 unique).

483 of 970 variables (49.79%) are classified as continuous (>= 15 unique).

42 of 970 variables (4.33%) were dropped.

10 variables had zero unique values (all NA).

32 variables had one unique value.

36 of 970 variables (3.71%) were not categorized and need to be set.

36 variables had between 6 and 15 unique values

0 variables had >= 15 values but couldn't be converted to

continuous (numeric) values

>>> f\_logBMI = igem.epc.modify.colfilter(nhanes, only=['BMXBMI', 'female'])

==============================================================================

Running colfilter

--------------------------------------------------------------------------------

Keeping 2 of 945 variables:

0 of 0 binary variables

0 of 0 categorical variables

2 of 945 continuous variables

0 of 0 unknown variables

================================================================================

>>> nhanes\_filtered = igem.epc.modify.colfilter\_min\_cat\_n(nhanes)

================================================================================

Running colfilter\_min\_cat\_n

--------------------------------------------------------------------------------

WARNING: 36 variables need to be categorized into a type manually

Testing 362 of 362 binary variables

Removed 248 (68.51%) tested binary variables which had a category

with less than 200 values

Testing 47 of 47 categorical variables

Removed 36 (76.60%) tested categorical variables which had a

category with less than 200 values

>>> nhanes\_filtered = igem.epc.modify.colfilter\_min\_n(nhanes)

================================================================================

Running colfilter\_min\_n

--------------------------------------------------------------------------------

WARNING: 36 variables need to be categorized into a type manually

Testing 362 of 362 binary variables

Removed 12 (3.31%) tested binary variables which had less than 200

non-null values

Testing 47 of 47 categorical variables

Removed 8 (17.02%) tested categorical variables which had less

than 200 non-null values

Testing 483 of 483 continuous variables

Removed 8 (1.66%) tested continuous variables which had less than

200 non-null values

>>> nhanes\_filtered = igem.epc.modify.colfilter\_percent\_zero(

nhanes\_filtered)

================================================================================

Running colfilter\_percent\_zero

--------------------------------------------------------------------------------

WARNING: 36 variables need to be categorized into a type manually

Testing 483 of 483 continuous variables

Removed 30 (6.21%) tested continuous variables which were equal to

zero in at least 90.00% of non-NA observations.

>>> nhanes = igem.epc.modify.make\_binary(

nhanes,

only=['female', 'black', 'mexican', 'other\_hispanic'])

================================================================================

Running make\_binary

--------------------------------------------------------------------------------

Set 4 of 970 variable(s) as binary, each with 22,624 observations

>>> df = igem.epc.modify.make\_categorical(df)

================================================================================

Running make\_categorical

--------------------------------------------------------------------------------

Set 12 of 12 variable(s) as categorical, each with 4,321 observations

>>> df = igem.epc.modify.make\_continuous(df)

================================================================================

Running make\_categorical

--------------------------------------------------------------------------------

Set 128 of 128 variable(s) as continuous, each with 4,321 observations

>>> df = igem.epc.modify.merge\_variables(df\_bin, df\_cat, how='outer')

>>> df\_cat, df\_cont = igem.epc.modify.move\_variables(

df\_cat, df\_cont,

only=["DRD350AQ", "DRD350DQ", "DRD350GQ"])

Moved 3 variables.

>>> discovery\_check, discovery\_cont =

igem.epc.modify.move\_variables(

discovery\_check,

discovery\_cont)

Moved 39 variables.

>>> df\_cat, df\_cont = igem.epc.modify.move\_variables(

df\_cat, df\_cont,

only=["DRD350AQ", "DRD350DQ", "DRD350GQ"])

Moved 3 variables.

>>> discovery\_check, discovery\_cont = igem.epc.modify.move\_variables(

discovery\_check,

discovery\_cont)

Moved 39 variables.

>>> nhanes\_rm\_outliers = igem.epc.modify.remove\_outliers(

nhanes,

method='iqr',

cutoff=1.5,

only=['DR1TVB1',

'URXP07',

'SMQ077']

)

================================================================================

Running remove\_outliers

--------------------------------------------------------------------------------

WARNING: 36 variables need to be categorized into a type manually

Removing outliers from 2 continuous variables with values < 1st Quartile -

(1.5 \* IQR) or > 3rd quartile + (1.5 \* IQR)

Removed 0 low and 430 high IQR outliers from URXP07

(outside -153.55 to 341.25)

Removed 0 low and 730 high IQR outliers from DR1TVB1

(outside -0.47 to 3.48)

>>> nhanes\_rm\_outliers = igem.epc.modify.remove\_outliers(

nhanes,

only=['DR1TVB1',

'URXP07'])

================================================================================

Running remove\_outliers

--------------------------------------------------------------------------------

WARNING: 36 variables need to be categorized into a type manually

Removing outliers from 2 continuous variables with values more than 3

standard deviations from the mean

Removed 0 low and 42 high gaussian outliers from URXP07

(outside -1,194.83 to 1,508.13)

Removed 0 low and 301 high gaussian outliers from DR1TVB1

(outside -1.06 to 4.27)

>>> nhanes\_filtered = igem.epc.modify.rowfilter\_incomplete\_obs(

nhanes,

only=[outcome] + covariates

)

================================================================================

Running rowfilter\_incomplete\_obs

--------------------------------------------------------------------------------

Removed 3,687 of 22,624 observations (16.30%) due to NA values in any of 8

>>> df = igem.epc.modify.transform(df, 'log', only=['BMXBMI'])

================================================================================

Running transform

--------------------------------------------------------------------------------

Transformed 'BMXBMI' using 'log'.

>>> df = igem.epc.modify.drop\_extra\_categories(df, only=['SDDSRVYR'])

============================================================================

Running drop\_extra\_categories

----------------------------------------------------------------------------

SDDSRVYR had categories with no occurrences: 3, 4

## Data Analysis

The EPC module includes functionalities specifically designed for conducting:

Environment-Wide Association Studies (EWAS). Researchers can leverage these functions to analyze the association between epigenetic modifications and phenotypic traits. The EPC module provides dedicated tools to perform statistical tests, correct p-values, and generate graphical representations such as Manhattan plots.

Association Study by providing tools to analyze the relationships between variables in the dataset. Users can perform association tests and explore the strength and significance of associations between variables. This functionality is particularly useful for identifying potential relationships and dependencies within the data.

ExE (Exposure by Exposure) Pairwise analysis, allowing researchers to examine the pairwise relationships between exposures. By applying this analysis, users can identify potential interactions or dependencies between different exposures in the dataset.

>>> import igem

>>> results = igem.epc.analyze.association\_study(

outcomes="HI\_CHOL",

covariates=["race", "agecat"],

data=df,

standardize\_data=True,

)

>>> ewas\_discovery = igem.epc.analyze.ewas(

"logBMI", covariates, nhanes\_discovery

)

Running on a continuous variable

>>> igem.epc.analyze.add\_corrected\_pvalues(ewas\_discovery)

>>> igem.epc.analyze.add\_corrected\_pvalues(

interaction\_result,

pvalue='Beta\_pvalue'

)

>>> igem.epc.analyze.add\_corrected\_pvalues(

interaction\_result,

pvalue='LRT\_pvalue',

groupby=["Term1", "Term2"]

)

Users can define survey designs with specific sampling strategies and create survey models for analyzing survey data. These features cater to researchers working with survey datasets and provide specialized tools for accurate analysis.

>>> import igem

>>> igem.epc.analyze.SurveyDesignSpec(

survey\_df=survey\_design\_replication,

strata="SDMVSTRA",

cluster="SDMVPSU",

nest=True,

weights=weights\_replication,

fpc=None,

single\_cluster='fail')

## Data Visualization

The EPC module provides various plot functions to visualize the data and gain deeper insights. These plot functions include:

Distributions: Generate visual representations of variable distributions, such as histograms and kernel density plots. These plots help researchers understand the underlying distribution of variables in the dataset.

Histograms: Create histograms to visualize the distribution of a single variable. This plot provides a visual summary of the frequency distribution of values in the dataset.

Manhattan Plot: Generate a Manhattan plot, commonly used in genetic association studies, to visualize the genomic location of associations. This plot displays the significance of associations along the genome.

Manhattan Plot with Bonferroni Correction: Similar to the Manhattan plot, this function incorporates Bonferroni correction to account for multiple hypothesis testing. It helps identify significant associations while controlling for the family-wise error rate.

Manhattan Plot with False Discovery Rate (FDR): This function applies the False Discovery Rate (FDR) correction to the associations in the Manhattan plot. It allows researchers to control the expected proportion of false discoveries while identifying significant associations.

Top Results Plot: Create a plot displaying the top results of an analysis, such as the most significant associations or the highest-ranked variables. This plot helps researchers focus on the most important findings in the data.

By utilizing the functionalities offered by the EPC module, users can create a streamlined and comprehensive pipeline for data analysis within the IGEM software. This module empowers researchers to load external datasets, describe the data, modify it to suit specific analyses, and perform advanced statistical tests and visualizations.

>>> import igem

>>> igem.epc.plot.distributions(

df[['female', 'occupation', 'LBX074']], filename="test"

)

.. image:: ../\_static/plot/distributions\_count.png

>>> igem.epc.plot.distributions(

df[['female', 'occupation', 'LBX074']],

filename="test",

continuous\_kind='box'

)

.. image:: ../\_static/plot/distributions\_box.png

>>> igem.epc.plot.distributions(

df[['female', 'occupation', 'LBX074']],

filename="test",

continuous\_kind='violin'

)

.. image:: ../\_static/plot/distributions\_violin.png

>>> igem.epc.plot.distributions(

df[['female', 'occupation', 'LBX074']],

filename="test",

continuous\_kind='qq'

)

.. image:: ../\_static/plot/distributions\_qq.png

>>> x = f"Discovery: Skew of BMIMBX = stats.skew(nhanes['BMXBMI']):.6}"

>>> igem.epc.plot.histogram(

nhanes\_discovery\_cont,

column="BMXBMI",

title=x,

bins=100

)

.. image:: ../\_static/plot/histogram.png

>>> igem.epc.plot.manhattan(

{'discovery':disc\_df, 'replication':repl\_df},

categories=data\_categories,

title="EWAS Results"

)

.. image:: ../\_static/plot/manhattan.png

>>> igem.epc.plot.manhattan\_bonferroni(

{'discovery':disc\_df, 'replication':repl\_df},

categories=data\_categories,

title="EWAS Results"

)

.. image:: ../\_static/plot/manhattan\_bonferroni.png

>>> igem.epc.plot.manhattan\_fdr(

{'discovery':disc\_df, 'replication':repl\_df},

categories=data\_categories,

title="EWAS Results"

)

.. image:: ../\_static/plot/manhattan\_fdr.png

>>> igem.epc.plot.top\_results(ewas\_result)

.. image:: ../\_static/plot/top\_results.png

# Anexo A : Functions Map

A screenshot of a computer

Description automatically generated with medium confidence

# Anexo B: Database Schema

A picture containing text, screenshot, parallel, design

Description automatically generated

# Figures

[Figure 1: IGEM Architeture 4](#_Toc139042892)

[Figure 2: IGEM and IGEM Lite differences components 5](#_Toc139042893)

[Figure 3: DB Makemigrations 6](#_Toc139042894)

[Figure 4: DB Migrate 7](#_Toc139042895)

[Figure 5: Create user 7](#_Toc139042896)

[Figure 6: Runserver 8](#_Toc139042897)

[Figure 7: Web Login 8](#_Toc139042898)

[Figure 8: Web IGEM Admin 8](#_Toc139042899)

[Figure 9: Web User Admin 11](#_Toc139042900)

[Figure 10: Web user interface 11](#_Toc139042901)

[Figure 11: Web change history admin 12](#_Toc139042902)

[Figure 12: Web new user 12](#_Toc139042903)

[Figure 13: Web save new user 13](#_Toc139042904)

[Figure 14: Web add permisions to user 14](#_Toc139042905)

[Figure 15: Web user log 15](#_Toc139042906)

[Figure 16: Web group 15](#_Toc139042907)

[Figure 17: Web setup user group 16](#_Toc139042908)

[Figure 18: Web user group history 16](#_Toc139042909)

[Figure 19: Web user group permissions 17](#_Toc139042910)

[Figure 20: Web datasouce 21](#_Toc139042911)

[Figure 21: Web datasouce change 21](#_Toc139042912)

[Figure 22: Web datasource change history 22](#_Toc139042913)

[Figure 23: Web datasouce delete 22](#_Toc139042914)

[Figure 24: Web connector 24](#_Toc139042915)

[Figure 25: Web Connector list 25](#_Toc139042916)

[Figure 26: Web change connector – part 1 25](#_Toc139042917)

[Figure 27: Web change connector - part 2 26](#_Toc139042918)

[Figure 28: Web change connector - part 3 27](#_Toc139042919)

[Figure 29: Web workflow control 40](#_Toc139042920)

[Figure 30: Web workflow control change 41](#_Toc139042921)

[Figure 31: SNP x Exposome Filter 48](#_Toc139042922)