IGEM Client

Version 0.1.2

June 24, 2023

Contents

Integrative Genome-Exposome Method

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. The primary goal of IGEM is to provide a flexible and dynamic framework for genomic and exposomic research, enabling the integration of diverse data sources and facilitating complex analyses.

The GE (Gene x Exposome) application developed within the IGEM system is focuses on collecting external data sets, identifying key genetic and exposomic information, and building a comprehensive knowledge base. This knowledge base is readily available for dynamic and exploratory queries, empowering researchers to uncover valuable insights and generate novel hypotheses.

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 help you leverage the IGEM system effectively to drive your genomic and exposomic analyses.

Introduction

The Integrative Genome-Exposome Method (IGEM) is a novel software to study exposure-exposure (ExE) and gene-environment (GxE) interactions in high-dimensional big data sets by integrating an automated knowledge-based user-friendly, open-source, and open-access software.

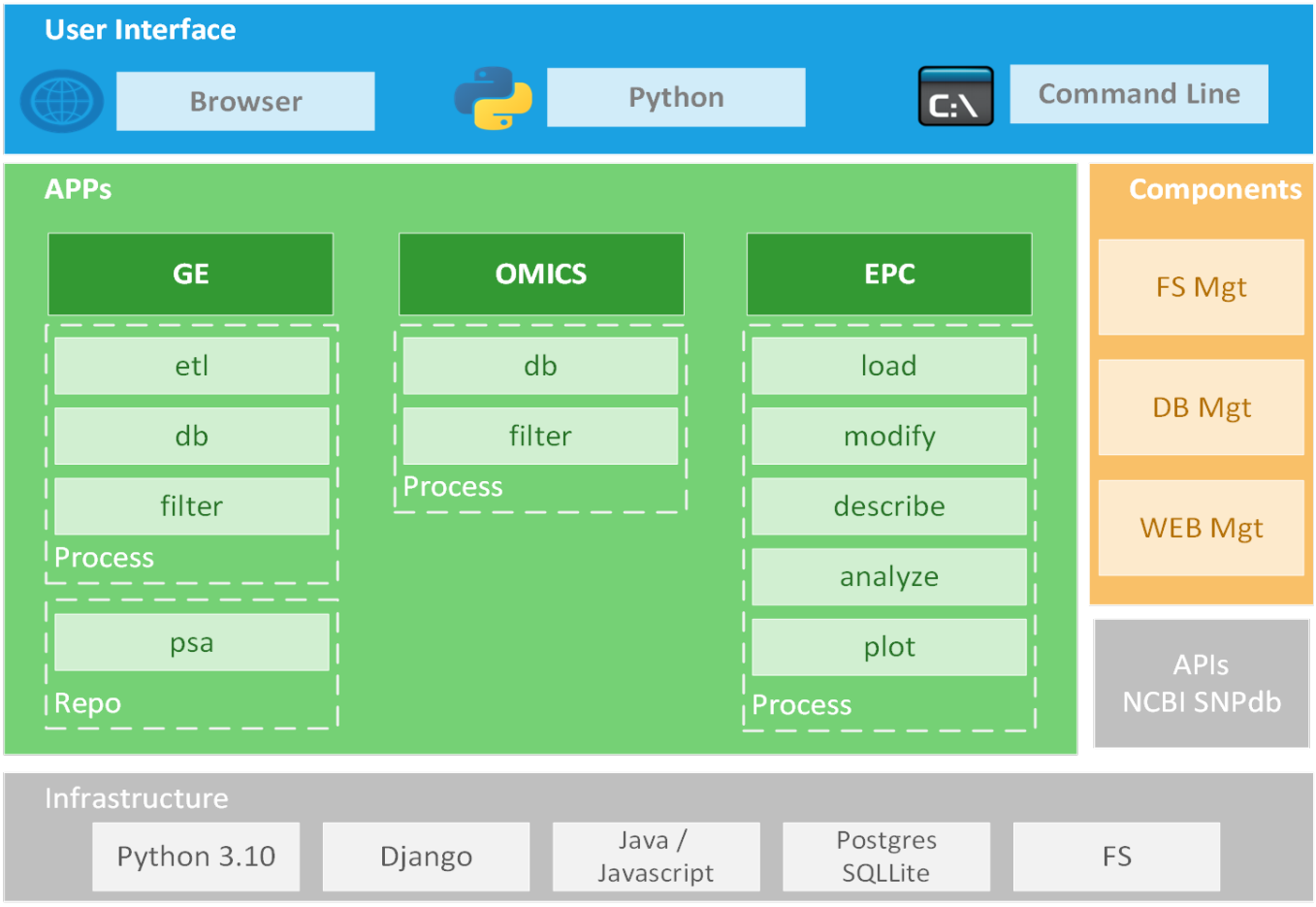
IGEM is a software to perform high-throughput quality control (QC), knowledge-driven ExE and GxE filtering, machine learning (ML) and regression-based interaction analysis, and big data visualization.

The IGEM system has a modular architecture, initially designed with three applications: GE, OMICS, and EPC, detailed throughout this document.

All applications interact with each other. For example, we can query a GxE relation from GE.db, integrate with other external data, perform regressions, and analyze without additional software.

To support the applications, components were implemented that work transparently for the user, and we performed database interface operations, with the file system, among others.

The IGEM can be accessed through a WEB query interface, Python strings, or the command line.
Below is a consolidated view of the IGEM components.



Every IGEM application has processes; we can access them through the available functions and their respective arguments.

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 exposomes domains.

These capabilities significantly enhance the research capabilities and contribute to a deeper understanding of complex biological systems.

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.

Database Management

The Database Management within the GE module provides two main functions:

- Direct access to database tables for retrieving information
- Synchronization of the IGEM Client DB with the latest data from the Hall Lab DB Server.

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
- ds_column
- 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},
        )
    If x:
        print("file created")
```

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}
```

Synchronization with the Hall Lab DB Server

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")
```

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.

Reports

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.

Parameters File

ge.filter.**parameters_file** (path=None)

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

- path**: str**

  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,Chemica
l 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 return 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.

Word Map

ge.filter.**word_map** (*args, **kwargs)

Queries GE.db and returns links between words without terms.

- path_in**: str**

  parameter file path with filter information, aggregation, and result file path.

- path_out**: str**

  result file path.

- term**: list[str]**

  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.word_map(
      path_in="../../file.csv",
      path_out="../../outcome.csv"
      )
```

This function queries GE.db and generates results showing links between words without terms. The results can be saved in a specified output file path or returned as a DataFrame.

Term Map

`ge.filter.`**`term_map`**`(*args, **kwargs)`

TermMap table query function.

- `path_in`**: str**

  parameter file path with filter information, aggregation, and result file path.

- `path_out`**: str**

  result file path.

- `term`**: list[str]**

  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.

Words to Terms

`ge.filter.`**`word_to_term`**`(path=None)`

Perform a search for terms from a string base with the same ETL engine.

- `path`**: str**

  File with the strings for conversion into terms. Only the first column of the file will be processed.

A file will be generated with the results in the same folder as the input strings file.

Example

```
from igem.ge import filter
filter.word_to_term(
      path='../../file.csv'
      )
```

This function searches for terms from a string base using the ETL engine. It takes a file path as input, reads the strings from the file, and converts them into terms. The results are saved in a CSV file in the same folder as the input file.

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`**: str**

  parameter file path with filter information, aggregation, and result file path.

- `path_out`**: str**

  result file path.

- `term`**: list[str]**

  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`**: str**

  parameter file path with filter information, aggregation, and result file path.

- `path_out`**: str**

  result file path.

- `term`**: list[str]**

  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"]
      )
```

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.

Tags

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.

`ge.filter.`**`get_tag`**`(tag)`

Function to retrieve WFControl data based on a TAG.

- `tag`**: str**

  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.

`ge.filter.`**`get_tag_data`**`(tag, path)`

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

- `tag`**: str**

  TAG string.

- `path`**: str**

    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.

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('nhanes.txt', index_col="SEQN")
Loaded 22,624 observations of 970 variables

>>> import igem
>>> df = igem.epc.load.from_csv('nhanes.txt', index_col="SEQN")
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"]
                )
```

Survey Design and Modeling

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')
```

Plot Functions

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
```