CLARITE Documentation

Release 0.10.0

Contributors

May 28, 2020

Contents

Version 0.10.0

CLeaning to Analysis: Reproducibility-based Interface for Traits and Exposures

CHAPTER 1

Motivation

CLARITE was created to provide an easy-to-use tool for analysis of traits and exposures.

It exists as both a python package (for integration into scripts and/or other packages) and as a command line program.

CHAPTER 2

Installation

2.1 Basic Install

At the command line:

```
$ pip install clarite
```

2.2 Running R code from CLARITE

In order to use the *ewas_r* function, it is recommended to install CLARITE using Conda:

1. Create and activate a conda environment with python 3.6 or 3.7:

   ```
   $ conda create -n clarite python=3.7
   $ conda activate clarite
   ```

2. Install rpy2 (optional). CLARITE has a version of the EWAS function that calls R code using the *survey* library:

   ```
   $ conda install -c conda-forge rpy2
   ```

3. Install CLARITE:

   ```
   $ pip install clarite
   ```

4. Install required R packages (such as *survey*) (optional):

   ```
   $ clarite-cli utils install-r-packages
   ```

CHAPTER 3

Citing CLARITE

If you use CLARITE in a scientific publication, please consider citing:

1. Lucas AM, et al (2019) CLARITE facilitates the quality control and analysis process for EWAS of metabolic-related traits. *Frontiers in Genetics*: 10, 1240

BibTeX entry:

```
@article{lucas2019clarite,
  title={CLARITE facilitates the quality control and analysis process for EWAS of␣
↪metabolic-related traits},
  author={Lucas, Anastasia M. and Palmiero, Nicole E. and McGuigan, John and Passero,␣
↪Kristin and Zhou, Jiayan and Orie, Deven and Ritchie, Marylyn D. and Hall, Molly A.}
↪,
  journal={Frontiers in Genetics},
  volume={10},
  pages={1240},
  year={2019},
  publisher={Frontiers},
  url={https://www.frontiersin.org/article/10.3389/fgene.2019.01240},
  doi={10.3389/fgene.2019.01240}
}
```

2. Passero K, et al (2020) Phenome-wide association studies on cardiovascular health and fatty acids considering phenotype quality control practices for epidemiological data. *Pacific Symposium on Biocomputing*: 25, 659

BibTeX entry:

```
@inproceedings{passero2020phenome,
  title={Phenome-wide association studies on cardiovascular health and fatty acids␣
↪considering phenotype quality control practices for epidemiological data.},
  author={Passero, Kristin and He, Xi and Zhou, Jiayan and Mueller-Myhsok, Bertram␣
↪and Kleber, Marcus E and Maerz, Winfried and Hall, Molly A},
  booktitle={Pacific Symposium on Biocomputing},
  volume={25},
  pages={659},
```

(continues on next page)

(continued from previous page)

```
  year={2020},
  organization={World Scientific},
  URL={https://www.worldscientific.com/doi/abs/10.1142/9789811215636_0058},
  DOI={10.1142/9789811215636_0058}
}
```

CHAPTER 4

Usage

4.1 Organization of Functions

CLARITE has many functions organized into several different modules:

Analyze Functions related to calculating EWAS results

Describe Functions used to gather information about data

Load Functions used to load data from different formats or sources

Modify Functions used to filter and/or modify data

Plot Functions that generate plots

Survey Functions and classes related to handling data with a complex survey design

4.2 Coding Style

There are three primary ways of using CLARITE'.

1. Using the CLARITE package as part of a python script or Jupyter notebook

This can be done using the function directly:

```
import clarite
df = clarite.load.from_tsv('data.txt')
df_filtered = clarite.modify.colfilter_min_n(df, n=250)
df_filtered_complete = clarite.modify.rowfilter_incomplete_obs(df_filtered)
clarite.plot.distributions(df_filtered_complete, filename='plots.pdf')
```

Or it can be done using Pandas *pipe*

```
clarite.plot.distributions(df.pipe(clarite.modify.colfilter_min_n, n=250)\
                             .pipe(clarite.modify.rowfilter_incomplete_obs),
                           filename='plots.pdf')
```

2. Using the command line tool

```
clarite-cli load from_tsv data/nhanes.txt results/data.txt --index SEQN
cd results
clarite-cli modify colfilter-min-n data data_filtered -n 250
clarite-cli modify rowfilter-incomplete-obs data_filtered data_filtered_complete
clarite-cli plot distributions data_filtered_complete plots.pdf
```

3. Using the GUI (coming soon)

CHAPTER 5

Example Analysis

CLARITE facilitates the quality control and analysis process for EWAS of metabolic-related traits

[Paper in review]

Data from NHANES was used in an EWAS analysis including utilizing the provided survey weight information. The first two cycles of NHANES (1999-2000 and 2001-2002) are assigned to a 'discovery' dataset and the next two cycles (2003-2004 and 2005-2006) are assigned to a 'replication' datset.

```
import pandas as pd
import numpy as np
from scipy import stats
import clarite
```

```
pd.options.display.max_rows = 10
pd.options.display.max_columns = 6
```

5.1 Load Data

```
data_folder = "../../../../data/NHANES_99-06/"
data_main_table_over18 = data_folder + "MainTable_keepvar_over18.tsv"
data_main_table = data_folder + "MainTable.csv"
data_var_description = data_folder + "VarDescription.csv"
data_var_categories = data_folder + "VarCat_nopf.txt"
output = "."
```

5.1.1 Data of all samples with age >= 18

```
# Data
nhanes = clarite.load.from_tsv(data_main_table_over18, index_col="ID")
nhanes.head()
```

```
Loaded 22,624 observations of 970 variables
```

5.1.2 Variable Descriptions

```
var_descriptions = pd.read_csv(data_var_description)[["tab_desc","module","var","var_
↪desc"]]\
                     .drop_duplicates()\
                     .set_index("var")
var_descriptions.head()
```

```
# Convert variable descriptions to a dictionary for convenience
var_descr_dict = var_descriptions["var_desc"].to_dict()
```

5.1.3 Survey Weights, as provided by NHANES

Survey weight information is used so that the results apply to the US civillian non-institutionalized population.

This includes:

- SDMVPSU (Cluster ID)
- SDMVSTRA (Nested Strata ID)
- 2-year weights
- 4-year weights

Different variables require different weights, as many of them were measured on a subset of the full dataset. For example:

- *WTINT* is the survey weight for interview variables.
- *WTMEC* is the survey weight for variables measured in the Mobile Exam Centers (a subset of interviewed samples)

2-year and 4-year weights are provided. It is important to adjust the weights when combining multiple cycles, by computing the weighted average. In this case 4-year weights (covering the first 2 cycles) are provided by NHANES and the replication weights (the 3rd and 4th cycles) were computed from the 2-year weights prior to loading them here.

```
survey_design_discovery = pd.read_csv(data_folder + "weights/weights_discovery.txt",␣
↪sep="\t")\
                            .rename(columns={'SEQN':'ID'})\
                            .set_index("ID")\
                            .drop(columns="SDDSRVYR")
survey_design_discovery.head()
```

```
survey_design_replication = pd.read_csv(data_folder + "weights/weights_replication_
↪4yr.txt", sep="\t")\
                            .rename(columns={'SEQN':'ID'})\
                            .set_index("ID")\
                            .drop(columns="SDDSRVYR")
survey_design_replication.head()
```

```
# These files map variables to their correct weights, and were compiled by reading␣
↪throught the NHANES codebook
var_weights = pd.read_csv(data_folder + "weights/VarWeights.csv")
var_weights.head()
```

```
# Convert the data to two dictionaries for convenience
weights_discovery = var_weights.set_index('variable_name')['discovery'].to_dict()
weights_replication = var_weights.set_index('variable_name')['replication'].to_dict()
```

5.1.4 Survey Year data

Survey year is found in a separate file and can be matched using the *SEQN* ID value.

```
survey_year = pd.read_csv(data_main_table)[["SEQN", "SDDSRVYR"]].rename(columns={'SEQN
↪':'ID'}).set_index("ID")
nhanes = clarite.modify.merge_variables(nhanes, survey_year, how="left")
```

```
================================================================================
Running merge_variables
--------------------------------------------------------------------------------
left Merge:
    left = 22,624 observations of 970 variables
    right = 41,474 observations of 1 variables
Kept 22,624 observations of 971 variables.
================================================================================
```

5.2 Define the phenotype and covariates

```
phenotype = "BMXBMI"
print(f"{phenotype} = {var_descriptions.loc[phenotype, 'var_desc']}")
covariates = ["female", "black", "mexican", "other_hispanic", "other_eth", "SES_LEVEL
↪", "RIDAGEYR", "SDDSRVYR"]
```

```
BMXBMI = Body Mass Index (kg/m**2)
```

5.3 Initial cleanup / variable selection

5.3.1 Remove any samples missing the phenotype or one of the covariates

```
nhanes = clarite.modify.rowfilter_incomplete_obs(nhanes, only=[phenotype] +␣
↪covariates)
```

```
================================================================================
Running rowfilter_incomplete_obs
--------------------------------------------------------------------------------
Removed 3,687 of 22,624 observations (16.30%) due to NA values in any of 9 variables
================================================================================
```

5.3.2 Remove variables that aren't appropriate for the analysis

Physical fitness measures

These are measurements rather than proxies for environmental exposures

```
phys_fitness_vars = ["CVDVOMAX","CVDESVO2","CVDS1HR","CVDS1SY","CVDS1DI","CVDS2HR",
↪"CVDS2SY","CVDS2DI","CVDR1HR","CVDR1SY","CVDR1DI","CVDR2HR","CVDR2SY","CVDR2DI",
↪"physical_activity"]
for v in phys_fitness_vars:
    print(f"\t{v} = {var_descr_dict[v]}")
nhanes = nhanes.drop(columns=phys_fitness_vars)
```

```
CVDVOMAX = Predicted VO2max (ml/kg/min)
CVDESVO2 = Estimated VO2max (ml/kg/min)
CVDS1HR = Stage 1 heart rate (per min)
CVDS1SY = Stage 1 systolic BP (mm Hg)
CVDS1DI = Stage 1 diastolic BP (mm Hg)
CVDS2HR = Stage 2 heart rate (per min)
CVDS2SY = Stage 2 systolic BP (mm Hg)
CVDS2DI = Stage 2 diastolic BP (mm Hg)
CVDR1HR = Recovery 1 heart rate (per min)
CVDR1SY = Recovery 1 systolic BP (mm Hg)
CVDR1DI = Recovery 1 diastolic BP (mm Hg)
CVDR2HR = Recovery 2 heart rate (per min)
CVDR2SY = Recovery 2 systolic BP (mm Hg)
CVDR2DI = Recovery 2 diastolic BP (mm Hg)
physical_activity = Physical Activity (MET-based rank)
```

Lipid variables

These are likely correlated with BMI in some way

```
lipid_vars = ["LBDHDD", "LBDHDL", "LBDLDL", "LBXSTR", "LBXTC", "LBXTR"]
print("Removing lipid measurement variables:")
for v in lipid_vars:
    print(f"\t{v} = {var_descr_dict[v]}")
nhanes = nhanes.drop(columns=lipid_vars)
```

```
Removing lipid measurement variables:
    LBDHDD = Direct HDL-Cholesterol (mg/dL)
    LBDHDL = Direct HDL-Cholesterol (mg/dL)
    LBDLDL = LDL-cholesterol (mg/dL)
    LBXSTR = Triglycerides (mg/dL)
    LBXTC = Total cholesterol (mg/dL)
    LBXTR = Triglyceride (mg/dL)
```

Indeterminate variables

These variables don't have clear meanings

```
indeterminent_vars = ["house_type","hepa","hepb", "house_age", "current_past_smoking"]
print("Removing variables with indeterminate meanings:")
```

(continues on next page)

(continued from previous page)

```
for v in indeterminent_vars:
    print(f"\t{v} = {var_descr_dict[v]}")
nhanes = nhanes.drop(columns=indeterminent_vars)
```

```
Removing variables with indeterminate meanings:
    house_type = house type
    hepa = hepatitis a
    hepb = hepatitis b
    house_age = house age
    current_past_smoking = Current or Past Cigarette Smoker?
```

5.3.3 Recode “missing” values

```
# SMQ077 and DDB100 have Refused/Don't Know for "7" and "9"
nhanes = clarite.modify.recode_values(nhanes, {7: np.nan, 9: np.nan}, only=['SMQ077',
↪'DBD100'])
```

```
================================================================================
Running recode_values
--------------------------------------------------------------------------------
Replaced 11 values from 18,937 observations in 2 variables
================================================================================
```

5.3.4 Split the data into *discovery* and *replication*

```
discovery = (nhanes['SDDSRVYR']==1) | (nhanes['SDDSRVYR']==2)
replication = (nhanes['SDDSRVYR']==3) | (nhanes['SDDSRVYR']==4)

nhanes_discovery = nhanes.loc[discovery]
nhanes_replication = nhanes.loc[replication]
```

```
nhanes_discovery.head()
```

```
nhanes_replication.head()
```

5.4 QC

5.4.1 Minimum of 200 non-NA values in each variable

Drop variables that have too small of a sample size

```
nhanes_discovery = clarite.modify.colfilter_min_n(nhanes_discovery, skip=[phenotype]␣
↪+ covariates)
nhanes_replication = clarite.modify.colfilter_min_n(nhanes_replication,␣
↪skip=[phenotype] + covariates)
```

```
================================================================================
Running colfilter_min_n
--------------------------------------------------------------------------------
Testing 0 of 0 binary variables
Testing 0 of 0 categorical variables
Testing 936 of 945 continuous variables
    Removed 302 (32.26%) tested continuous variables which had less than 200 non-null␣
↪values.
================================================================================
================================================================================
Running colfilter_min_n
--------------------------------------------------------------------------------
Testing 0 of 0 binary variables
Testing 0 of 0 categorical variables
Testing 936 of 945 continuous variables
    Removed 225 (24.04%) tested continuous variables which had less than 200 non-null␣
↪values.
================================================================================
```

5.4.2 Categorize Variables

This is important, as different variable types must be processed in different ways. The number of unique values for each variable is a good heuristic for determining this. The default settings were used here, but different cutoffs can be specified. CLARITE reports the results in neatly formatted text:

```
nhanes_discovery = clarite.modify.categorize(nhanes_discovery)
nhanes_replication = clarite.modify.categorize(nhanes_replication)
```

```
================================================================================
Running categorize
--------------------------------------------------------------------------------
229 of 643 variables (35.61%) are classified as binary (2 unique values).
19 of 643 variables (2.95%) are classified as categorical (3 to 6 unique values).
336 of 643 variables (52.26%) are classified as continuous (>= 15 unique values).
37 of 643 variables (5.75%) were dropped.
    0 variables had zero unique values (all NA).
    37 variables had one unique value.
22 of 643 variables (3.42%) were not categorized and need to be set manually.
    22 variables had between 6 and 15 unique values
    0 variables had >= 15 values but couldn't be converted to continuous (numeric)␣
↪values
================================================================================
================================================================================
Running categorize
--------------------------------------------------------------------------------
236 of 720 variables (32.78%) are classified as binary (2 unique values).
32 of 720 variables (4.44%) are classified as categorical (3 to 6 unique values).
400 of 720 variables (55.56%) are classified as continuous (>= 15 unique values).
13 of 720 variables (1.81%) were dropped.
    0 variables had zero unique values (all NA).
    13 variables had one unique value.
39 of 720 variables (5.42%) were not categorized and need to be set manually.
    39 variables had between 6 and 15 unique values
    0 variables had >= 15 values but couldn't be converted to continuous (numeric)␣
↪values
```

(continues on next page)

(continued from previous page)

```
================================================================================
```

5.4.3 Checking categorization

Distributions of variables may be plotted using CLARITE:

```
clarite.plot.distributions(nhanes_discovery,
                           filename="discovery_distributions.pdf",
                           continuous_kind='count',
                           nrows=4,
                           ncols=3,
                           quality='medium')
```

One variable needed correcting where the heuristic was not correct

```
v = "L_GLUTAMINE_gm"
print(f"\t{v} = {var_descr_dict[v]}\n")
nhanes_discovery = clarite.modify.make_continuous(nhanes_discovery, only=[v])
nhanes_replication = clarite.modify.make_continuous(nhanes_replication, only=[v])
```

```
    L_GLUTAMINE_gm = L_GLUTAMINE_gm

================================================================================
Running make_continuous
--------------------------------------------------------------------------------
Set 1 of 606 variable(s) as continuous, each with 9,063 observations
================================================================================
================================================================================
Running make_continuous
--------------------------------------------------------------------------------
Set 1 of 707 variable(s) as continuous, each with 9,874 observations
================================================================================
```

After examining all of the uncategorized variables, they are all continuous

```
discovery_types = clarite.describe.get_types(nhanes_discovery)
discovery_unknown = discovery_types[discovery_types == 'unknown'].index
for v in list(discovery_unknown):
    print(f"\t{v} = {var_descr_dict[v]}")
nhanes_discovery = clarite.modify.make_continuous(nhanes_discovery, only=discovery_
↪unknown)
```

```
WARNING: 22 variables need to be categorized into a type manually
    URXUBE = Beryllium, urine (ug/L)
    URXUPT = Platinum, urine (ug/L)
    DRD350BQ = # of times crabs eaten in past 30 days
    DRD350FQ = # of times oysters eaten in past 30 days
    DRD350IQ = # of times other shellfish eaten
    DRD370AQ = # of times breaded fish products eaten
    DRD370DQ = # of times catfish eaten in past 30 days
```

(continues on next page)

(continued from previous page)

```
    DRD370EQ = # of times cod eaten in past 30 days
    DRD370FQ = # of times flatfish eaten past 30 days
    DRD370UQ = # of times other unknown fish eaten
    OMEGA_3_FATTY_ACIDS_mg = OMEGA_3_FATTY_ACIDS_mg
    ALANINE_mg = ALANINE_mg
    ARGININE_mg = ARGININE_mg
    BETA_CAROTENE_mg = BETA_CAROTENE_mg
    CAFFEINE_mg = CAFFEINE_mg
    CYSTINE_mg = CYSTINE_mg
    LYSINE_mg = LYSINE_mg
    PROLINE_mg = PROLINE_mg
    SERINE_mg = SERINE_mg
    TRYPTOPHAN_mg = TRYPTOPHAN_mg
    TYROSINE_mg = TYROSINE_mg
    OTHER_FATTY_ACIDS_mg = OTHER_FATTY_ACIDS_mg
================================================================================
Running make_continuous
--------------------------------------------------------------------------------
Set 22 of 606 variable(s) as continuous, each with 9,063 observations
================================================================================
```

```
replication_types = clarite.describe.get_types(nhanes_replication)
replication_unknown = replication_types[replication_types == 'unknown'].index
for v in list(replication_unknown):
    print(f"\t{v} = {var_descr_dict[v]}")
nhanes_replication = clarite.modify.make_continuous(nhanes_replication,␣
↪only=replication_unknown)
```

```
WARNING: 39 variables need to be categorized into a type manually
    LBXVCT = Blood Carbon Tetrachloride (ng/ml)
    LBXV3A = Blood 1,1,1-Trichloroethene (ng/ml)
    URXUBE = Beryllium, urine (ug/L)
    LBXTO2 = Toxoplasma (IgM)
    LBXPFDO = Perfluorododecanoic acid
    DRD350AQ = # of times clams eaten in past 30 days
    DRD350BQ = # of times crabs eaten in past 30 days
    DRD350DQ = # of times lobsters eaten past 30 days
    DRD350FQ = # of times oysters eaten in past 30 days
    DRD350GQ = # of times scallops eaten past 30 days
    DRD370AQ = # of times breaded fish products eaten
    DRD370DQ = # of times catfish eaten in past 30 days
    DRD370EQ = # of times cod eaten in past 30 days
    DRD370FQ = # of times flatfish eaten past 30 days
    DRD370GQ = # of times haddock eaten in past 30 days
    DRD370NQ = # of times sardines eaten past 30 days
    DRD370RQ = # of times trout eaten in past 30 days
    DRD370UQ = # of times other unknown fish eaten
    ALANINE_mg = ALANINE_mg
    ARGININE_mg = ARGININE_mg
    BETA_CAROTENE_mg = BETA_CAROTENE_mg
    CAFFEINE_mg = CAFFEINE_mg
    CYSTINE_mg = CYSTINE_mg
    HISTIDINE_mg = HISTIDINE_mg
    ISOLEUCINE_mg = ISOLEUCINE_mg
    LEUCINE_mg = LEUCINE_mg
    LYSINE_mg = LYSINE_mg
```

(continues on next page)

(continued from previous page)

```
    PHENYLALANINE_mg = PHENYLALANINE_mg
    PROLINE_mg = PROLINE_mg
    SERINE_mg = SERINE_mg
    THREONINE_mg = THREONINE_mg
    TRYPTOPHAN_mg = TRYPTOPHAN_mg
    TYROSINE_mg = TYROSINE_mg
    VALINE_mg = VALINE_mg
    LBXV2T = Blood trans-1,2-Dichloroethene (ng/mL)
    LBXV4T = Blood 1,1,2,2-Tetrachloroethane (ng/mL)
    LBXVDM = Blood Dibromomethane (ng/mL)
    URXUTM = Urinary Trimethylarsine Oxide (ug/L)
    LBXPFBS = Perfluorobutane sulfonic acid
================================================================================
Running make_continuous
--------------------------------------------------------------------------------
Set 39 of 707 variable(s) as continuous, each with 9,874 observations
================================================================================
```

Types should match across discovery/replication

```
# Take note of which variables were differently typed in each dataset
print("Correcting differences in variable types between discovery and replication")
# Merge current type series
dtypes = pd.DataFrame({'discovery':clarite.describe.get_types(nhanes_discovery),
                       'replication':clarite.describe.get_types(nhanes_replication)
                       })
diff_dtypes = dtypes.loc[(dtypes['discovery'] != dtypes['replication']) &
                         (~dtypes['discovery'].isna()) &
                         (~dtypes['replication'].isna())]

# Discovery

# Binary -> Categorical
compare_bin_cat = list(diff_dtypes.loc[(diff_dtypes['discovery']=='binary') &
                                       (diff_dtypes['replication']=='categorical'),].
↪index)
if len(compare_bin_cat) > 0:
    print(f"Bin vs Cat: {', '.join(compare_bin_cat)}")
    nhanes_discovery = clarite.modify.make_categorical(nhanes_discovery, only=compare_
↪bin_cat)
    print()
# Binary -> Continuous
compare_bin_cont = list(diff_dtypes.loc[(diff_dtypes['discovery']=='binary') &
                                        (diff_dtypes['replication']=='continuous'),].
↪index)
if len(compare_bin_cont) > 0:
    print(f"Bin vs Cont: {', '.join(compare_bin_cont)}")
    nhanes_discovery = clarite.modify.make_continuous(nhanes_discovery, only=compare_
↪bin_cont)
    print()
# Categorical -> Continuous
compare_cat_cont = list(diff_dtypes.loc[(diff_dtypes['discovery']=='categorical') &
                                        (diff_dtypes['replication']=='continuous'),].
↪index)
if len(compare_cat_cont) > 0:
```

(continues on next page)

(continued from previous page)

```
    print(f"Cat vs Cont: {', '.join(compare_cat_cont)}")
    nhanes_discovery = clarite.modify.make_continuous(nhanes_discovery, only=compare_
↪cat_cont)
    print()

# Replication

# Binary -> Categorical
compare_cat_bin = list(diff_dtypes.loc[(diff_dtypes['discovery']=='categorical') &
                                        (diff_dtypes['replication']=='binary'),].index)
if len(compare_cat_bin) > 0:
    print(f"Cat vs Bin: {', '.join(compare_cat_bin)}")
    nhanes_replication = clarite.modify.make_categorical(nhanes_replication,␣
↪only=compare_cat_bin)
    print()
# Binary -> Continuous
compare_cont_bin = list(diff_dtypes.loc[(diff_dtypes['discovery']=='continuous') &
                                         (diff_dtypes['replication']=='binary'),].
↪index)
if len(compare_cont_bin) > 0:
    print(f"Cont vs Bin: {', '.join(compare_cont_bin)}")
    nhanes_replication = clarite.modify.make_continuous(nhanes_replication,␣
↪only=compare_cont_bin)
    print()
# Categorical -> Continuous
compare_cont_cat = list(diff_dtypes.loc[(diff_dtypes['discovery']=='continuous') &
                                         (diff_dtypes['replication']=='categorical'),].
↪index)
if len(compare_cont_cat) > 0:
    print(f"Cont vs Cat: {', '.join(compare_cont_cat)}")
    nhanes_replication = clarite.modify.make_continuous(nhanes_replication,␣
↪only=compare_cont_cat)
    print()
```

```
Correcting differences in variable types between discovery and replication
Bin vs Cat: BETA_CAROTENE_mcg, CALCIUM_Unknown, MAGNESIUM_Unknown
================================================================================
Running make_categorical
--------------------------------------------------------------------------------
Set 3 of 606 variable(s) as categorical, each with 9,063 observations
================================================================================

Bin vs Cont: LBXPFDO
================================================================================
Running make_continuous
--------------------------------------------------------------------------------
Set 1 of 606 variable(s) as continuous, each with 9,063 observations
================================================================================

Cat vs Cont: DRD350AQ, DRD350DQ, DRD350GQ
================================================================================
Running make_continuous
--------------------------------------------------------------------------------
Set 3 of 606 variable(s) as continuous, each with 9,063 observations
================================================================================

```

(continues on next page)

(continued from previous page)

```
Cat vs Bin: VITAMIN_B_12_Unknown
================================================================================
Running make_categorical
--------------------------------------------------------------------------------
Set 1 of 707 variable(s) as categorical, each with 9,874 observations
================================================================================
```

5.4.4 Filtering

These are a standard set of filters with default settings

```
# 200 non-na samples
discovery_1_min_n = clarite.modify.colfilter_min_n(nhanes_discovery)
replication_1_min_n = clarite.modify.colfilter_min_n(nhanes_replication)
```

```
================================================================================
Running colfilter_min_n
--------------------------------------------------------------------------------
Testing 228 of 228 binary variables
    Removed 0 (0.00%) tested binary variables which had less than 200 non-null values.
Testing 15 of 15 categorical variables
    Removed 0 (0.00%) tested categorical variables which had less than 200 non-null␣
↪values.
Testing 363 of 363 continuous variables
    Removed 0 (0.00%) tested continuous variables which had less than 200 non-null␣
↪values.
================================================================================
================================================================================
Running colfilter_min_n
--------------------------------------------------------------------------------
Testing 236 of 236 binary variables
    Removed 0 (0.00%) tested binary variables which had less than 200 non-null values.
Testing 31 of 31 categorical variables
    Removed 0 (0.00%) tested categorical variables which had less than 200 non-null␣
↪values.
Testing 440 of 440 continuous variables
    Removed 0 (0.00%) tested continuous variables which had less than 200 non-null␣
↪values.
================================================================================
```

```
# 200 samples per category
discovery_2_min_cat_n = clarite.modify.colfilter_min_cat_n(discovery_1_min_n, skip=[c␣
↪for c in covariates + [phenotype] if c in discovery_1_min_n.columns] )
replication_2_min_cat_n = clarite.modify.colfilter_min_cat_n(replication_1_min_n,
↪skip=[c for c in covariates + [phenotype] if c in replication_1_min_n.columns])
```

```
================================================================================
Running colfilter_min_cat_n
--------------------------------------------------------------------------------
Testing 222 of 228 binary variables
    Removed 162 (72.97%) tested binary variables which had a category with less than␣
↪200 values.
Testing 14 of 15 categorical variables
    Removed 10 (71.43%) tested categorical variables which had a category with less␣
↪than 200 values.
```

(continues on next page)

(continued from previous page)

```
================================================================================
================================================================================
Running colfilter_min_cat_n
--------------------------------------------------------------------------------
Testing 230 of 236 binary variables
    Removed 154 (66.96%) tested binary variables which had a category with less than␣
↪200 values.
Testing 30 of 31 categorical variables
    Removed 25 (83.33%) tested categorical variables which had a category with less␣
↪than 200 values.
================================================================================
```

```
# 90percent zero filter
discovery_3_pzero = clarite.modify.colfilter_percent_zero(discovery_2_min_cat_n)
replication_3_pzero = clarite.modify.colfilter_percent_zero(replication_2_min_cat_n)
```

```
================================================================================
Running colfilter_percent_zero
--------------------------------------------------------------------------------
Testing 363 of 363 continuous variables
    Removed 28 (7.71%) tested continuous variables which were equal to zero in at␣
↪least 90.00% of non-NA observations.
================================================================================
================================================================================
Running colfilter_percent_zero
--------------------------------------------------------------------------------
Testing 440 of 440 continuous variables
    Removed 30 (6.82%) tested continuous variables which were equal to zero in at␣
↪least 90.00% of non-NA observations.
================================================================================
```

```
# Those without weights
keep = set(weights_discovery.keys()) | set([phenotype] + covariates)
discovery_4_weights = discovery_3_pzero[[c for c in list(discovery_3_pzero) if c in␣
↪keep]]

keep = set(weights_replication.keys()) | set([phenotype] + covariates)
replication_4_weights = replication_3_pzero[[c for c in list(replication_3_pzero) if␣
↪c in keep]]
```

5.4.5 Summarize

```
# Summarize Results
print("\nDiscovery:")
clarite.describe.summarize(discovery_4_weights)
print('-'*50)
print("Replication:")
clarite.describe.summarize(replication_4_weights)
```

```
Discovery:
9,063 observations of 385 variables
    66 Binary Variables
    5 Categorical Variables
```

(continues on next page)

(continued from previous page)

```
    314 Continuous Variables
    0 Unknown-Type Variables

--------------------------------------------------
Replication:
9,874 observations of 428 variables
    77 Binary Variables
    6 Categorical Variables
    345 Continuous Variables
    0 Unknown-Type Variables
```

5.4.6 Keep only variables that passed QC in both datasets

```
both = set(list(discovery_4_weights)) & set(list(replication_4_weights))
discovery_final = discovery_4_weights[both]
replication_final = replication_4_weights[both]
print(f"{len(both)} variables in common")
```

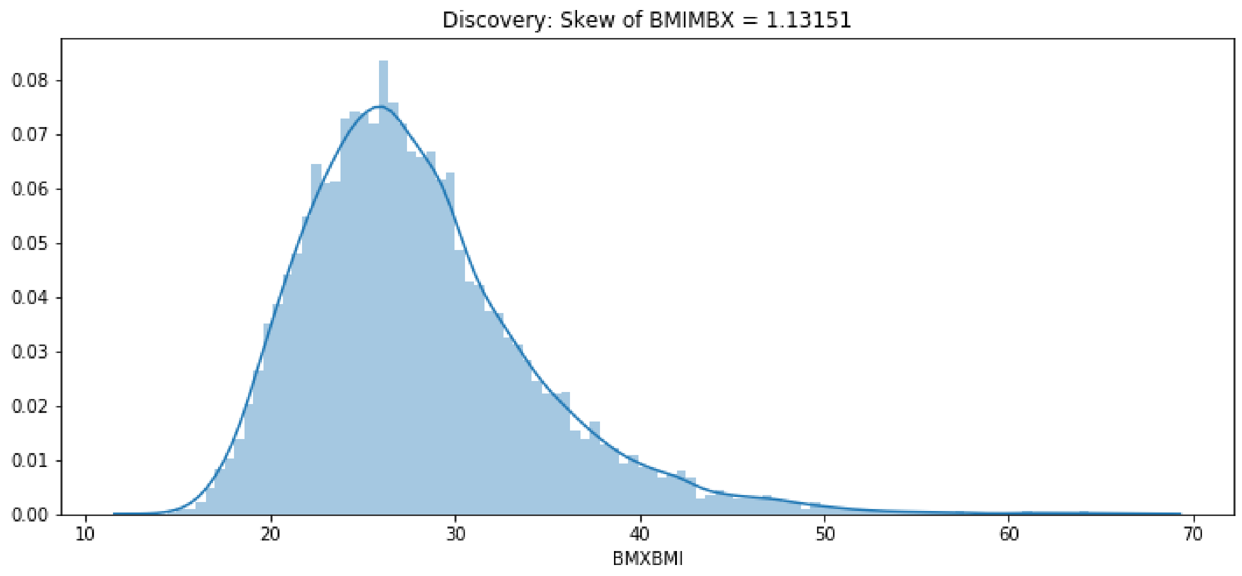
```
341 variables in common
```

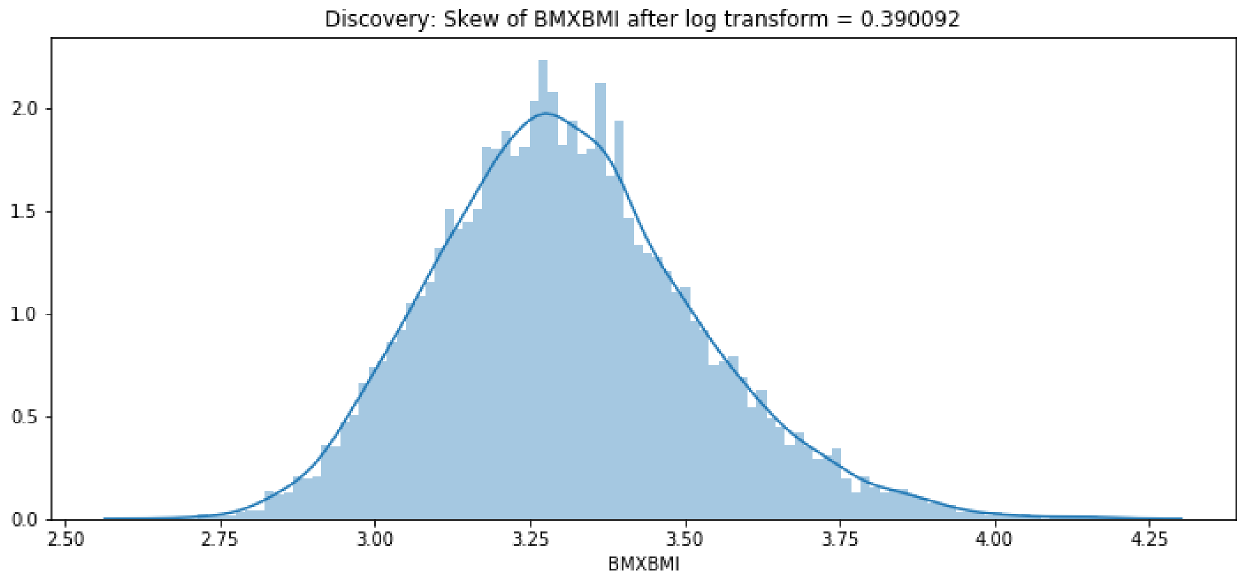
5.5 Checking the phenotype distribution

The phenotype appears to be skewed, so it will need to be corrected. CLARITE makes it easy to plot distributions and to transform variables.

```
title = f"Discovery: Skew of BMIMBX = {stats.skew(discovery_final['BMXBMI']):.6}"
clarite.plot.histogram(discovery_final, column="BMXBMI", title=title, bins=100)
# Log-transform
discovery_final = clarite.modify.transform(discovery_final, transform_method='log',␣
↪only='BMXBMI')
#Plot
title = f"Discovery: Skew of BMXBMI after log transform = {stats.skew(discovery_final[
↪'BMXBMI']):.6}"
clarite.plot.histogram(discovery_final, column="BMXBMI", title=title, bins=100)
```

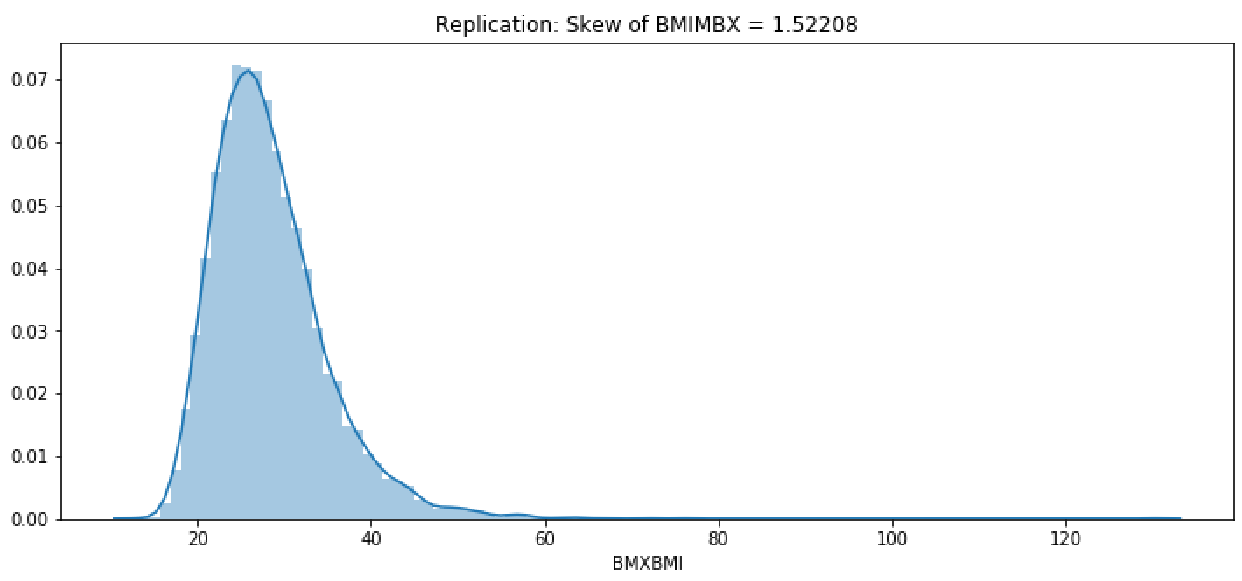
```
================================================================================
Running transform
--------------------------------------------------------------------------------
Transformed 'BMXBMI' using 'log'
================================================================================
```

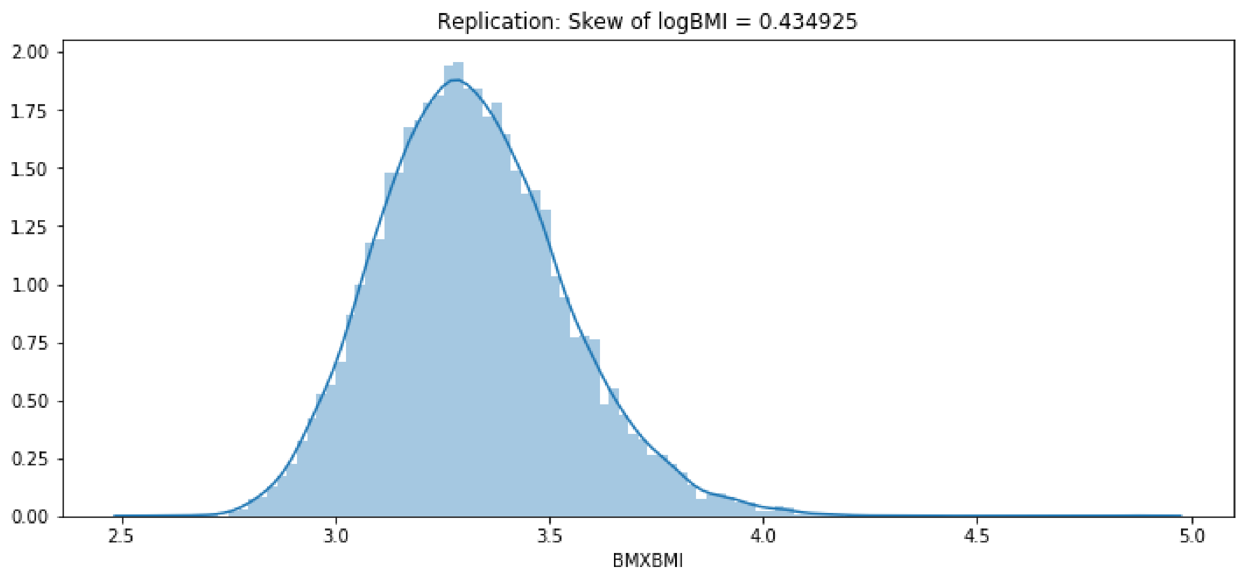




```
title = f"Replication: Skew of BMIMBX = {stats.skew(replication_final['BMXBMI']):.6}"
clarite.plot.histogram(replication_final, column="BMXBMI", title=title, bins=100)
# Log-transform
replication_final = clarite.modify.transform(replication_final, transform_method='log
↪', only='BMXBMI')
#Plot
title = f"Replication: Skew of logBMI = {stats.skew(replication_final['BMXBMI']):.6}"
clarite.plot.histogram(replication_final, column="BMXBMI", title=title, bins=100)
```

```
================================================================================
Running transform
--------------------------------------------------------------------------------
Transformed 'BMXBMI' using 'log'
================================================================================
```





5.6 EWAS

5.6.1 Survey Design Spec

When utilizing survey data, a survey design spec object must be created.

```
sd_discovery = clarite.survey.SurveyDesignSpec(survey_df=survey_design_discovery,
                                                strata="SDMVSTRA",
                                                cluster="SDMVPSU",
                                                nest=True,
                                                weights=weights_discovery,
                                                single_cluster='centered')
```

5.6.2 EWAS

This can then be passed into the EWAS function

```
ewas_discovery = clarite.analyze.ewas(phenotype, covariates, discovery_final, sd_
↪discovery)
```

```
Running EWAS on a continuous variable

####### Regressing 280 Continuous Variables #######

WARNING: DRD370UQ - 3 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBXVID has non-varying covariates(s): SDDSRVYR
WARNING: URXP24 has non-varying covariates(s): SDDSRVYR
WARNING: age_stopped_birth_control has non-varying covariates(s): female
WARNING: DR1TCHOL - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBX206 has non-varying covariates(s): SDDSRVYR
WARNING: DR1TVB1 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBXDIE has non-varying covariates(s): SDDSRVYR
WARNING: DRD350BQ - 2 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBXLYC has non-varying covariates(s): SDDSRVYR
WARNING: LBXF09 has non-varying covariates(s): SDDSRVYR
WARNING: DR1TS160 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TVK has non-varying covariates(s): SDDSRVYR
WARNING: DRD350FQ - 1 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD370TQ - 1 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD370EQ - 1 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TS100 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBXALD has non-varying covariates(s): SDDSRVYR
WARNING: DR1TCOPP - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: URXP20 has non-varying covariates(s): SDDSRVYR
WARNING: DR1TSELE - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBX151 has non-varying covariates(s): SDDSRVYR
WARNING: LBXLUZ has non-varying covariates(s): SDDSRVYR
WARNING: DR1TLZ has non-varying covariates(s): SDDSRVYR
WARNING: DR1TPHOS - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TP204 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBXCBC has non-varying covariates(s): SDDSRVYR
WARNING: DR1TPOTA - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TVB6 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TVB12 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
```

(continues on next page)

(continued from previous page)

```
WARNING: DR1TP184 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TP182 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TMFAT - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: RHQ556 has non-varying covariates(s): female
WARNING: LBXBEC has non-varying covariates(s): SDDSRVYR
WARNING: DR1TSUGR has non-varying covariates(s): SDDSRVYR
WARNING: URXP02 has non-varying covariates(s): SDDSRVYR
WARNING: DRD370AQ - 2 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBXEND has non-varying covariates(s): SDDSRVYR
WARNING: DR1TCRYP has non-varying covariates(s): SDDSRVYR
WARNING: DR1TKCAL - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TFIBE - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TTFAT - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TZINC - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBX110 has non-varying covariates(s): SDDSRVYR
WARNING: how_long_estrogen has non-varying covariates(s): female
WARNING: LBD199 has non-varying covariates(s): SDDSRVYR
WARNING: URXMHH has non-varying covariates(s): SDDSRVYR
WARNING: DR1TTHEO - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TFDFE has non-varying covariates(s): SDDSRVYR
WARNING: URXOP4 - 403 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD350DQ - 1 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TALCO - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: URXUHG has non-varying covariates(s): female
WARNING: URXP22 has non-varying covariates(s): SDDSRVYR
WARNING: URXP21 has non-varying covariates(s): SDDSRVYR
WARNING: DR1TSFAT - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD350HQ - 6 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: URXOP1 - 404 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD370BQ - 5 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: URXOP2 - 404 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TM201 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TFF has non-varying covariates(s): SDDSRVYR
WARNING: URXMOH has non-varying covariates(s): SDDSRVYR
WARNING: DR1TFA has non-varying covariates(s): SDDSRVYR
WARNING: DR1TS120 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: URXMNM has non-varying covariates(s): SDDSRVYR
WARNING: LBX195 has non-varying covariates(s): SDDSRVYR
```

(continues on next page)

(continued from previous page)

```
WARNING: DR1TACAR has non-varying covariates(s): SDDSRVYR
WARNING: DRD370FQ - 1 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TATOC has non-varying covariates(s): SDDSRVYR
WARNING: URXOP3 - 404 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBX189 has non-varying covariates(s): SDDSRVYR
WARNING: DR1TP225 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TP226 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TP183 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBXTHG has non-varying covariates(s): female
WARNING: DR1TBCAR has non-varying covariates(s): SDDSRVYR
WARNING: DRD370MQ - 1 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TPFAT - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TS060 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TM161 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBXCRY has non-varying covariates(s): SDDSRVYR
WARNING: DR1TCALC - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBXIHG has non-varying covariates(s): female
WARNING: DR1TM221 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TIRON - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD370DQ - 1 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: URXOP5 - 403 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TPROT - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TVARA has non-varying covariates(s): SDDSRVYR
WARNING: DR1TCARB - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TMAGN - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TM181 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TS140 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TVC - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBX196 has non-varying covariates(s): SDDSRVYR
WARNING: age_started_birth_control has non-varying covariates(s): female
WARNING: URXP01 has non-varying covariates(s): SDDSRVYR
WARNING: LBXD02 has non-varying covariates(s): SDDSRVYR
WARNING: URXMIB has non-varying covariates(s): SDDSRVYR
WARNING: LBX149 has non-varying covariates(s): SDDSRVYR
WARNING: LBXALC has non-varying covariates(s): SDDSRVYR
WARNING: DR1TS180 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
```

(continues on next page)

(continued from previous page)

```
WARNING: DR1TVB2 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TCAFF - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TLYCO has non-varying covariates(s): SDDSRVYR
WARNING: LBX087 has non-varying covariates(s): SDDSRVYR
WARNING: LBXV3A has non-varying covariates(s): SDDSRVYR
WARNING: DR1TP205 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBX194 has non-varying covariates(s): SDDSRVYR
WARNING: DR1TNIAC - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: URXUUR has non-varying covariates(s): SDDSRVYR
WARNING: DRD350AQ - 1 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: URXMC1 has non-varying covariates(s): SDDSRVYR
WARNING: DR1TS040 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: URXOP6 - 403 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TS080 - 14 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DR1TRET has non-varying covariates(s): SDDSRVYR
WARNING: LBX028 has non-varying covariates(s): SDDSRVYR

####### Regressing 48 Binary Variables #######

WARNING: DRD350A - 6 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD350B - 6 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: current_loud_noise - 925 observation(s) with missing, negative, or zero␣
↪weights were removed
WARNING: LBXBV has non-varying covariates(s): female, SDDSRVYR
WARNING: ordinary_salt - 19 observation(s) with missing, negative, or zero weights␣
↪were removed
WARNING: ordinary_salt has non-varying covariates(s): SDDSRVYR
WARNING: taking_birth_control has non-varying covariates(s): female
WARNING: LBXMS1 has non-varying covariates(s): SDDSRVYR
WARNING: DRD370A - 10 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD370F - 10 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: SXQ280 has non-varying covariates(s): female
WARNING: DRD350F - 6 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD350G - 6 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD370B - 10 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD370U - 10 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD370D - 10 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: LBXHBC - 5808 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD370T - 10 observation(s) with missing, negative, or zero weights were␣
↪removed
```

(continues on next page)

(continued from previous page)

```
WARNING: DRD340 - 22 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD350H - 6 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: RHQ540 has non-varying covariates(s): female
WARNING: DRD350D - 6 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD370M - 10 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD360 - 21 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: no_salt - 19 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: no_salt has non-varying covariates(s): SDDSRVYR
WARNING: DRD370E - 10 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: RHQ510 has non-varying covariates(s): female

####### Regressing 4 Categorical Variables #######

WARNING: DBD100 - 9 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DBD100 has non-varying covariates(s): SDDSRVYR
Completed EWAS
```

There is a separate function for adding pvalues with multiple-test-correction applied.

```
clarite.analyze.add_corrected_pvalues(ewas_discovery)
```

Saving results is straightforward

```
ewas_discovery.to_csv(output + "/BMI_Discovery_Results.txt", sep="\t")
```

5.6.3 Selecting top results

Variables with an FDR less than 0.1 were selected (using standard functionality from the Pandas library, since the ewas results are simply a Pandas DataFrame).

```
significant_discovery_variables = ewas_discovery[ewas_discovery['pvalue_fdr']<0.1].
↪index.get_level_values('Variable')
print(f"Using {len(significant_discovery_variables)} variables based on FDR-corrected␣
↪pvalues from the discovery dataset")
```

```
Using 100 variables based on FDR-corrected pvalues from the discovery dataset
```

5.7 Replication

The variables with low FDR in the discovery dataset were analyzed in the replication dataset

5.7.1 Filter out variables

```
keep_cols = list(significant_discovery_variables) + covariates + [phenotype]
replication_final_sig = clarite.modify.colfilter(replication_final, only=keep_cols)
clarite.describe.summarize(replication_final_sig)
```

```
================================================================================
Running colfilter
--------------------------------------------------------------------------------
Keeping 109 of 341 variables:
    19 of 54 binary variables
    3 of 5 categorical variables
    87 of 282 continuous variables
    0 of 0 unknown variables
================================================================================
9,874 observations of 109 variables
    19 Binary Variables
    3 Categorical Variables
    87 Continuous Variables
    0 Unknown-Type Variables
```

5.7.2 Run Replication EWAS

```
survey_design_replication
```

```
sd_replication = clarite.survey.SurveyDesignSpec(survey_df=survey_design_replication,
                                          strata="SDMVSTRA",
                                          cluster="SDMVPSU",
                                          nest=True,
                                          weights=weights_replication,
                                          single_cluster='centered')

ewas_replication = clarite.analyze.ewas(phenotype, covariates, replication_final_sig,␣
↪sd_replication)
clarite.analyze.add_corrected_pvalues(ewas_replication)
ewas_replication.to_csv(output + "/BMI_Replication_Results.txt", sep="\t")
```

```
Running EWAS on a continuous variable

####### Regressing 85 Continuous Variables #######

WARNING: URXP24 has non-varying covariates(s): SDDSRVYR
WARNING: age_stopped_birth_control has non-varying covariates(s): female
WARNING: LBXODT has non-varying covariates(s): SDDSRVYR
WARNING: LBX206 has non-varying covariates(s): SDDSRVYR
WARNING: LBX170 has non-varying covariates(s): SDDSRVYR
WARNING: LBX099 has non-varying covariates(s): SDDSRVYR
WARNING: URXP20 has non-varying covariates(s): SDDSRVYR
WARNING: LBX156 has non-varying covariates(s): SDDSRVYR
WARNING: URXP11 has non-varying covariates(s): SDDSRVYR
WARNING: LBX118 has non-varying covariates(s): SDDSRVYR
WARNING: LBX153 has non-varying covariates(s): SDDSRVYR
WARNING: LBXD05 has non-varying covariates(s): SDDSRVYR
WARNING: LBD199 has non-varying covariates(s): SDDSRVYR
```

(continues on next page)

(continued from previous page)

```
WARNING: LBXHPE has non-varying covariates(s): SDDSRVYR
WARNING: URXOP1 has non-varying covariates(s): SDDSRVYR
WARNING: URXP15 has non-varying covariates(s): SDDSRVYR
WARNING: LBXMIR has non-varying covariates(s): SDDSRVYR
WARNING: URXOP3 has non-varying covariates(s): SDDSRVYR
WARNING: LBXHXC has non-varying covariates(s): SDDSRVYR
WARNING: LBXME has non-varying covariates(s): SDDSRVYR
WARNING: LBX180 has non-varying covariates(s): SDDSRVYR
WARNING: LBX196 has non-varying covariates(s): SDDSRVYR
WARNING: age_started_birth_control has non-varying covariates(s): female
WARNING: LBXF04 has non-varying covariates(s): SDDSRVYR
WARNING: URXP03 has non-varying covariates(s): SDDSRVYR
WARNING: LBXIRN has non-varying covariates(s): female
WARNING: LBX194 has non-varying covariates(s): SDDSRVYR
WARNING: DUQ110 has non-varying covariates(s): SDDSRVYR

####### Regressing 13 Binary Variables #######

WARNING: DUQ100 has non-varying covariates(s): SDDSRVYR
WARNING: LBXHBC - 6318 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: SMQ210 has non-varying covariates(s): SDDSRVYR
WARNING: ever_loud_noise_gt3 has non-varying covariates(s): SDDSRVYR
WARNING: ever_loud_noise_gt3_2 has non-varying covariates(s): SDDSRVYR
WARNING: DRD370M - 19 observation(s) with missing, negative, or zero weights were␣
↪removed
WARNING: DRD370E - 19 observation(s) with missing, negative, or zero weights were␣
↪removed

####### Regressing 2 Categorical Variables #######

Completed EWAS
```

```
## Compare results
```

```
# Combine results
ewas_keep_cols = ['pvalue', 'pvalue_bonferroni', 'pvalue_fdr']
combined = pd.merge(ewas_discovery[['Variable_type'] + ewas_keep_cols],
                    ewas_replication[ewas_keep_cols],
                    left_index=True, right_index=True, suffixes=("_disc", "_repl"))

# FDR < 0.1 in both
fdr_significant = combined.loc[(combined['pvalue_fdr_disc'] <= 0.1) & (combined[
↪'pvalue_fdr_repl'] <= 0.1),]
fdr_significant = fdr_significant.assign(m=fdr_significant[['pvalue_fdr_disc',
↪'pvalue_fdr_repl']].mean(axis=1))\
                                 .sort_values('m').drop('m', axis=1)
fdr_significant.to_csv(output + "/Significant_Results_FDR_0.1.txt", sep="\t")
print(f"{len(fdr_significant)} variables had FDR < 0.1 in both discovery and␣
↪replication")

# Bonferroni < 0.05 in both
bonf_significant05 = combined.loc[(combined['pvalue_bonferroni_disc'] <= 0.05) &␣
↪(combined['pvalue_bonferroni_repl'] <= 0.05),]
bonf_significant05 = bonf_significant05.assign(m=fdr_significant[['pvalue_bonferroni_
↪disc', 'pvalue_bonferroni_repl']].mean(axis=1))\
```

(continues on next page)

(continued from previous page)

```
                                        .sort_values('m').drop('m', axis=1)
bonf_significant05.to_csv(output + "/Significant_Results_Bonferroni_0.05.txt", sep="\t
↪")
print(f"{len(bonf_significant05)} variables had Bonferroni < 0.05 in both discovery␣
↪and replication")

# Bonferroni < 0.01 in both
bonf_significant01 = combined.loc[(combined['pvalue_bonferroni_disc'] <= 0.01) &␣
↪(combined['pvalue_bonferroni_repl'] <= 0.01),]
bonf_significant01 = bonf_significant01.assign(m=fdr_significant[['pvalue_bonferroni_
↪disc', 'pvalue_bonferroni_repl']].mean(axis=1))\
                                        .sort_values('m').drop('m', axis=1)
bonf_significant01.to_csv(output + "/Significant_Results_Bonferroni_0.01.txt", sep="\t
↪")
print(f"{len(bonf_significant01)} variables had Bonferroni < 0.01 in both discovery␣
↪and replication")

bonf_significant01.head()
```

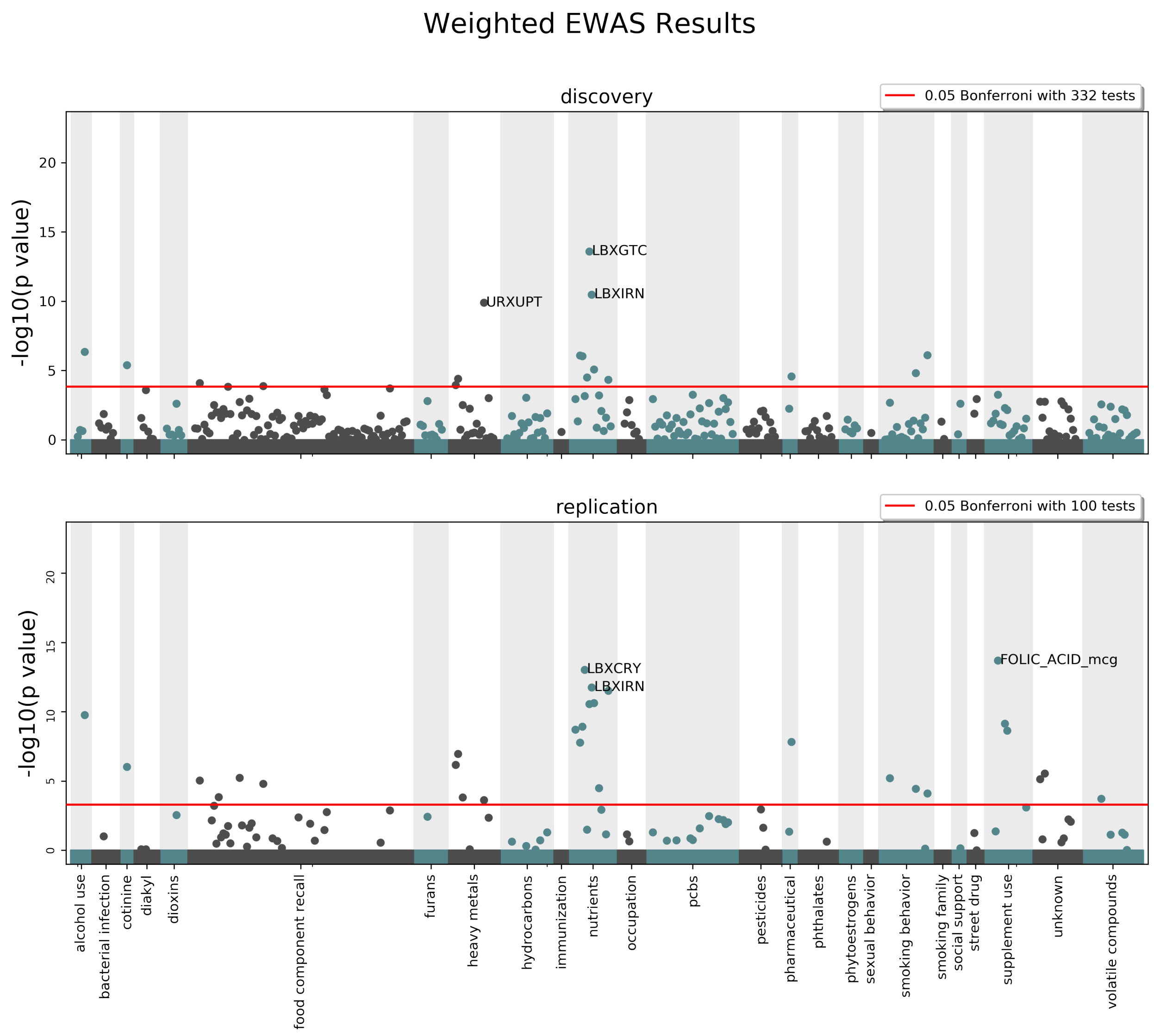
```
63 variables had FDR < 0.1 in both discovery and replication
16 variables had Bonferroni < 0.05 in both discovery and replication
10 variables had Bonferroni < 0.01 in both discovery and replication
```

5.8 Manhattan Plots

CLARITE provides functionality for generating highly customizable Manhattan plots from EWAS results

```
data_categories = pd.read_csv(data_var_categories, sep="\t").set_index('Variable')
data_categories.columns = ['category']
data_categories = data_categories['category'].to_dict()

clarite.plot.manhattan({'discovery': ewas_discovery, 'replication': ewas_replication},
                       categories=data_categories, title="Weighted EWAS Results",␣
↪filename=output + "/ewas_plot.png",
                       figsize=(14, 10))
```


Weighted EWAS Results
discovery
0.05 Bonferroni with 332 tests
-log10(p value)
0
5
10
15
20
LBXGTC
LBXIRN
URXUPT
replication
0.05 Bonferroni with 100 tests
LBXCRY
LBXIRN
FOLIC_ACID_mcg
alcohol use
bacterial infection
cotinine
diakyl
dioxins
food component recall
furans
heavy metals
hydrocarbons
immunization
nutrients
occupation
pcbs
pesticides
pharmaceutical
phthalates
phytoestrogens
sexual behavior
smoking behavior
smoking family
social support
street drug
supplement use
unknown
volatile compounds

CHAPTER 6

Complex Survey Data

CLARITE provides preliminary support for handling complex survey designs, similar to how the r-package *survey* works.

A SurveyDesignSpec can be created, which is used to obtain survey design objects for specific variables:

```
sd_discovery = clarite.survey.SurveyDesignSpec(survey_df=survey_design_discovery,
                                               strata="SDMVSTRA",
                                               cluster="SDMVPSU",
                                               nest=True,
                                               weights=weights_discovery,
                                               single_cluster='scaled')
```

There are a few different options for the 'single_cluster' parameter, which controls how strata with single clusters are handled in the linearized covariance calculation:

- *error* - Throw an error
- *scaled* - Use the average value of other strata
- *centered* - Use the average of all observations
- *certainty* - Single-cluster strata don't contribute to the variance

After a SurveyDesignSpec is created, it can be passed into the ewas function to utilize the survey design parameters:

```
ewas_discovery = clarite.analyze.ewas("logBMI", covariates, nhanes_discovery_bin,␣
↪nhanes_discovery_cat, nhanes_discovery_cont, sd_discovery, cov_method='stata')
```

CHAPTER 7

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

7.1 API Reference

CLARITE functions are organized into several modules:

7.1.1 Analyze

EWAS and associated calculations

`ewas`(phenotype, covariates, data, . . .)	Run an EWAS on a phenotype.
`add_corrected_pvalues`(ewas_result)	Add bonferroni and FDR pvalues to an ewas result and sort by increasing FDR (in-place)

clarite.analyze.ewas

`clarite.analyze.`**`ewas`**(*phenotype: str, covariates: List[str], data: pandas.core.frame.DataFrame, survey_design_spec: Union[clarite.modules.survey.survey_design.SurveyDesignSpec, NoneType] = None, cov_method: Union[str, NoneType] = 'stata', min_n: Union[int, NoneType] = 200*)

Run an EWAS on a phenotype.

Note:

- Binary variables are treated as continuous features, with values of 0 and 1.
- The results of a likelihood ratio test are used for categorical variables, so no Beta values or SE are reported.

- The regression family is automatically selected based on the type of the phenotype. * Continuous phenotypes use gaussian regression * Binary phenotypes use binomial regression (the larger of the two values is counted as "success")
- Categorical variables run with a survey design will not report Diff_AIC

Parameters

phenotype: string The variable to be used as the output of the regressions

covariates: list (strings), The variables to be used as covariates. Any variables in the DataFrames not listed as covariates are regressed.

data: pd.DataFrame The data to be analyzed, including the phenotype, covariates, and any variables to be regressed.

survey_design_spec: SurveyDesignSpec or None A SurveyDesignSpec object is used to create SurveyDesign objects for each regression.

cov_method: str or None Covariance calculation method (if survey_design_spec is passed in). 'stata' or 'jackknife'

min_n: int or None Minimum number of complete-case observations (no NA values for phenotype, covariates, variable, or weight) Defaults to 200

Returns

df: pd.DataFrame EWAS results DataFrame with these columns: ['variable_type', 'N', 'beta', 'SE', 'var_pvalue', 'LRT_pvalue', 'diff_AIC', 'pvalue']

Examples

```
>>> ewas_discovery = clarite.analyze.ewas("logBMI", covariates, nhanes_
↪discovery)
Running EWAS on a continuous variable
```

clarite.analyze.add_corrected_pvalues

`clarite.analyze.add_corrected_pvalues(`*ewas_result*`)`

Add bonferroni and FDR pvalues to an ewas result and sort by increasing FDR (in-place)

Parameters

ewas_result: pd.DataFrame EWAS results DataFrame with these columns: ['Variable_type', 'Converged', 'N', 'Beta', 'SE', 'Variable_pvalue', 'LRT_pvalue', 'Diff_AIC', 'pvalue']

Returns

None

Examples

```
>>> clarite.analyze.add_corrected_pvalues(ewas_discovery)
```

7.1.2 Describe

Functions that are used to gather information about some data

correlations(data, threshold)	Return variables with pearson correlation above the threshold
freq_table(data)	Return the count of each unique value for all binary and categorical variables.
get_types(data)	Return the type of each variable
percent_na(data)	Return the percent of observations that are NA for each variable
skewness(data, dropna)	Return the skewness of each continuous variable
summarize(data)	Print the number of each type of variable and the number of observations

clarite.describe.correlations

clarite.describe.**correlations**(*data: pandas.core.frame.DataFrame*, *threshold: float = 0.75*)

Return variables with pearson correlation above the threshold

Parameters

data: pd.DataFrame The DataFrame to be described

threshold: float, between 0 and 1 Return a dataframe listing pairs of variables whose absolute value of correlation is above this threshold

Returns

result: pd.DataFrame DataFrame listing pairs of correlated variables and their correlation value

Examples

```
>>> import clarite
>>> correlations = clarite.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
```

clarite.describe.freq_table

clarite.describe.**freq_table**(*data: pandas.core.frame.DataFrame*)

Return the count of each unique value for all binary and categorical variables. Other variables will return a single row with a value of '<Non-Categorical Values>' and the number of non-NA values.

Parameters

data: pd.DataFrame The DataFrame to be described

Returns

result: pd.DataFrame DataFrame listing variable, value, and count for each categorical variable

Examples

```
>>> import clarite
>>> clarite.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
```

clarite.describe.get_types

`clarite.describe.get_types`(*data: pandas.core.frame.DataFrame*)

Return the type of each variable

Parameters

data: pd.DataFrame The DataFrame to be described

Returns

result: pd.Series Series listing the CLARITE type for each variable

Examples

```
>>> import clarite
>>> clarite.describe.get_types(df).head()
RIDAGEYR          continuous
female                binary
black                 binary
mexican               binary
other_hispanic        binary
dtype: object
```

clarite.describe.percent_na

`clarite.describe.percent_na`(*data: pandas.core.frame.DataFrame*)

Return the percent of observations that are NA for each variable

Parameters

data: pd.DataFrame The DataFrame to be described

Returns

result: pd.DataFrame DataFrame listing percent NA for each variable

Examples

```
>>> import clarite
>>> clarite.describe.percent_na(df)
   variable  percent_na
0  SDDSRVYR     0.00000
1    female     0.00000
2    LBXHBC     4.99321
3    LBXHBS     4.98730
```

clarite.describe.skewness

`clarite.describe.`**`skewness`**(*data: pandas.core.frame.DataFrame*, *dropna: bool = False*)

Return the skewness of each continuous variable

Parameters

data: pd.DataFrame The DataFrame to be described

dropna: bool If True, drop rows with NA values before calculating skew. Otherwise the NA values propagate.

Returns

result: pd.DataFrame DataFrame listing three values for each continuous variable and NA for others: skew, zscore, and pvalue The test null hypothesis is that the skewness of the samples population is the same as the corresponding

normal distribution. The pvalue is the two-sided pvalue for the hypothesis test

Examples

```
>>> import clarite
>>> clarite.describe.skewness(df)
     Variable      skew    zscore        pvalue
0       pdias       NaN       NaN           NaN
1   longindex       NaN       NaN           NaN
2     durflow  2.754286  8.183515  2.756827e-16
3      height  0.583514  2.735605  6.226567e-03
4     begflow -0.316648 -1.549449  1.212738e-01
```

clarite.describe.summarize

`clarite.describe.`**`summarize`**(*data: pandas.core.frame.DataFrame*)

Print the number of each type of variable and the number of observations

Parameters

data: pd.DataFrame The DataFrame to be described

Returns

result: None

Examples

```
>>> import clarite
>>> clarite.describe.get_types(df).head()
RIDAGEYR          continuous
female                binary
black                 binary
mexican               binary
other_hispanic        binary
dtype: object
```

7.1.3 Load

Load data from different formats or sources

from_tsv(filename, index_col, int, ...)	Load data from a tab-separated file into a DataFrame
from_csv(filename, index_col, int, ...)	Load data from a comma-separated file into a DataFrame

clarite.load.from_tsv

`clarite.load.`**`from_tsv`**(*filename: str, index_col: Union[str, int, NoneType] = 0, **kwargs*)

Load data from a tab-separated file into a DataFrame

Parameters

filename: str or Path File with data to be used in CLARITE

index_col: int or string (default 0) Column to use as the row labels of the DataFrame.

****kwargs:** Other keword arguments to pass to pd.read_csv

Returns

DataFrame The index column will be used when merging

Examples

Load a tab-delimited file with an “ID” column

```
>>> import clarite
>>> df = clarite.import.from_tsv('nhanes.txt', index_col="SEQN")
Loaded 22,624 observations of 970 variables
```

clarite.load.from_csv

`clarite.load.`**`from_csv`**(*filename: str, index_col: Union[str, int, NoneType] = 0, **kwargs*)

Load data from a comma-separated file into a DataFrame

Parameters

filename: str or Path File with data to be used in CLARITE

index_col: int or string (default 0) Column to use as the row labels of the DataFrame.

****kwargs:** Other keword arguments to pass to pd.read_csv

Returns

DataFrame The index column will be used when merging

Examples

Load a tab-delimited file with an "ID" column

```
>>> import clarite
>>> df = clarite.import.from_csv('nhanes.csv', index_col="SEQN")
Loaded 22,624 observations of 970 variables
```

7.1.4 Modify

Functions used to filter and/or change some data, always taking in one set of data and returning one set of data.

categorize(data, cat_min, cat_max, cont_min)	Classify variables into constant, binary, categorical, continuous, and 'unknown'.
colfilter(data, skip, List[str], …)	Remove some variables (skip) or keep only certain variables (only)
colfilter_percent_zero(data, filter_percent, …)	Remove continuous variables which have <proportion> or more values of zero (excluding NA)
colfilter_min_n(data, n, skip, List[str], …)	Remove variables which have less than <n> non-NA values
colfilter_min_cat_n(data, n, skip, …)	Remove binary and categorical variables which have less than <n> occurences of each unique value
make_binary(data, skip, List[str], …)	Set variable types as Binary
make_categorical(data, skip, List[str], …)	Set variable types as Categorical
make_continuous(data, skip, List[str], …)	Set variable types as Numeric
merge_observations(top, bottom)	Merge two datasets, keeping only the columns present in both.
merge_variables(left, …)	Merge a list of dataframes with different variables side-by-side.
move_variables(left, right, …)	Move one or more variables from one DataFrame to another
recode_values(data, replacement_dict, skip, …)	Convert values in a dataframe.
remove_outliers(data, method[, cutoff])	Remove outliers from continuous variables by replacing them with np.nan
rowfilter_incomplete_obs(data, skip, …)	Remove rows containing null values
transform(data, transform_method, skip, …)	Apply a transformation function to a variable

clarite.modify.categorize

`clarite.modify.`**`categorize`**(*data: pandas.core.frame.DataFrame*, *cat_min: int = 3*, *cat_max: int = 6*, *cont_min: int = 15*)

Classify variables into constant, binary, categorical, continuous, and 'unknown'. Drop variables that only have NaN values.

Parameters

data: pd.DataFrame The DataFrame to be processed

cat_min: int, default 3 Minimum number of unique, non-NA values for a categorical variable

cat_max: int, default 6 Maximum number of unique, non-NA values for a categorical variable

cont_min: int, default 15 Minimum number of unique, non-NA values for a continuous variable

Returns

result: pd.DataFrame or None If inplace, returns None. Changes the datatypes on the input DataFrame.

Examples

```
>>> import clarite
>>> clarite.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␣
↪values).
483 of 970 variables (49.79%) are classified as continuous (>= 15 unique␣
↪values).
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␣
↪manually.
        36 variables had between 6 and 15 unique values
        0 variables had >= 15 values but couldn't be converted to␣
↪continuous (numeric) values
```

clarite.modify.colfilter

`clarite.modify.`**`colfilter`**(*data*, *skip: Union[str, List[str], NoneType] = None*, *only: Union[str, List[str], NoneType] = None*)

Remove some variables (skip) or keep only certain variables (only)

Parameters

data: pd.DataFrame The DataFrame to be processed and returned

skip: str, list or None (default is None) List of variables to remove

only: str, list or None (default is None) List of variables to keep

Returns

data: pd.DataFrame The filtered DataFrame

Examples

```
>>> import clarite
>>> female_logBMI = clarite.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
================================================================================
```

clarite.modify.colfilter_percent_zero

clarite.modify.**colfilter_percent_zero**(*data: pandas.core.frame.DataFrame, filter_percent: float = 90.0, skip: Union[str, List[str], NoneType] = None, only: Union[str, List[str], NoneType] = None*)

Remove continuous variables which have <proportion> or more values of zero (excluding NA)

Parameters

data: pd.DataFrame The DataFrame to be processed and returned

filter_percent: float, default 90.0 If the percentage of rows in the data with a value of zero is greater than or equal to this value, the variable is filtered out.

skip: str, list or None (default is None) List of variables that the filter should *not* be applied to

only: str, list or None (default is None) List of variables that the filter should *only* be applied to

Returns

data: pd.DataFrame The filtered DataFrame

Examples

```
>>> import clarite
>>> nhanes_filtered = clarite.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.
```

clarite.modify.colfilter_min_n

`clarite.modify.colfilter_min_n`(*data: pandas.core.frame.DataFrame, n: int = 200, skip: Union[str, List[str], NoneType] = None, only: Union[str, List[str], NoneType] = None*)

Remove variables which have less than <n> non-NA values

Parameters

data: pd.DataFrame The DataFrame to be processed and returned

n: int, default 200 The minimum number of unique values required in order for a variable not to be filtered

skip: str, list or None (default is None) List of variables that the filter should *not* be applied to

only: str, list or None (default is None) List of variables that the filter should *only* be applied to

Returns

data: pd.DataFrame The filtered DataFrame

Examples

```
>>> import clarite
>>> nhanes_filtered = clarite.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
```

clarite.modify.colfilter_min_cat_n

`clarite.modify.colfilter_min_cat_n`(*data, n: int = 200, skip: Union[str, List[str], NoneType] = None, only: Union[str, List[str], NoneType] = None*)

Remove binary and categorical variables which have less than <n> occurences of each unique value

Parameters

data: pd.DataFrame The DataFrame to be processed and returned

n: int, default 200 The minimum number of occurences of each unique value required in order for a variable not to be filtered

skip: str, list or None (default is None) List of variables that the filter should *not* be applied to

only: str, list or None (default is None) List of variables that the filter should *only* be applied to

Returns

data: pd.DataFrame The filtered DataFrame

Examples

```
>>> import clarite
>>> nhanes_filtered = clarite.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
```

clarite.modify.make_binary

`clarite.modify.make_binary`(*data: pandas.core.frame.DataFrame, skip: Union[str, List[str], NoneType] = None, only: Union[str, List[str], NoneType] = None*)

Set variable types as Binary

Checks that each variable has at most 2 values and converts the type to pd.Categorical.

Note: When these variables are used in regression, they are ordered by value. For example, Sex (Male=1, Female=2) will encode "Male" as 0 and "Female" as 1 during the EWAS regression step.

Parameters

data: pd.DataFrame or pd.Series Data to be processed

skip: str, list or None (default is None) List of variables that should *not* be made binary

only: str, list or None (default is None) List of variables that are the *only* ones to be made binary

Returns

data: pd.DataFrame DataFrame with the same data but validated and converted to binary types

Examples

```
>>> import clarite
>>> nhanes = clarite.modify.make_binary(nhanes, only=['female', 'black',
↪'mexican', 'other_hispanic'])
=============================================================================
Running make_binary
```

(continues on next page)

(continued from previous page)

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

clarite.modify.make_categorical

`clarite.modify.make_categorical(data: pandas.core.frame.DataFrame, skip: Union[str, List[str], NoneType] = None, only: Union[str, List[str], NoneType] = None)`

Set variable types as Categorical

Converts the type to pd.Categorical

Parameters

data: pd.DataFrame or pd.Series Data to be processed

skip: str, list or None (default is None) List of variables that should *not* be made categorical

only: str, list or None (default is None) List of variables that are the *only* ones to be made categorical

Returns

data: pd.DataFrame DataFrame with the same data but validated and converted to categorical types

Examples

```
>>> import clarite
>>> df = clarite.modify.make_categorical(df)
================================================================================
Running make_categorical
--------------------------------------------------------------------------
↪-------
Set 12 of 12 variable(s) as categorical, each with 4,321 observations
```

clarite.modify.make_continuous

`clarite.modify.make_continuous(data: pandas.core.frame.DataFrame, skip: Union[str, List[str], NoneType] = None, only: Union[str, List[str], NoneType] = None)`

Set variable types as Numeric

Converts the type to numeric

Parameters

data: pd.DataFrame or pd.Series Data to be processed

skip: str, list or None (default is None) List of variables that should *not* be made continuous

only: str, list or None (default is None) List of variables that are the *only* ones to be made continuous

Returns

data: pd.DataFrame DataFrame with the same data but validated and converted to numeric types

Examples

```
>>> import clarite
>>> df = clarite.modify.make_continuous(df)
==========================================================================================
Running make_categorical
--------------------------------------------------------------------------------
↪-------
Set 128 of 128 variable(s) as continuous, each with 4,321 observations
```

clarite.modify.merge_observations

`clarite.modify.`**`merge_observations`**(*top: pandas.core.frame.DataFrame*, *bottom: pandas.core.frame.DataFrame*)

Merge two datasets, keeping only the columns present in both. Raise an error if a datatype conflict occurs.

Parameters

top: pd.DataFrame "top" DataFrame

bottom: pd.DataFrame "bottom" DataFrame

Returns

result: pd.DataFrame

clarite.modify.merge_variables

`clarite.modify.`**`merge_variables`**(*left: Union[pandas.core.frame.DataFrame, pandas.core.series.Series], right: Union[pandas.core.frame.DataFrame, pandas.core.series.Series], how: str = 'outer'*)

Merge a list of dataframes with different variables side-by-side. Keep all observations ('outer' merge) by default.

Parameters

left: pd.Dataframe or pd.Series "left" DataFrame or Series

right: pd.DataFrame or pd.Series "right" DataFrame or Series which uses the same index

how: merge method, one of {'left', 'right', 'inner', 'outer'} Keep only rows present in the left data, the right data, both datasets, or either dataset.

Examples

```
>>> import clarite
>>> df = clarite.modify.merge_variables(df_bin, df_cat, how='outer')
```

clarite.modify.move_variables

`clarite.modify.`**`move_variables`**(*left: pandas.core.frame.DataFrame, right: Union[pandas.core.frame.DataFrame, pandas.core.series.Series], skip: Union[str, List[str], NoneType] = None, only: Union[str, List[str], NoneType] = None*)

Move one or more variables from one DataFrame to another

Parameters

left: pd.Dataframe DataFrame containing the variable(s) to be moved

right: pd.DataFrame or pd.Series DataFrame or Series (which uses the same index) that the variable(s) will be moved to

skip: str, list or None (default is None) List of variables that will *not* be moved

only: str, list or None (default is None) List of variables that are the *only* ones to be moved

Returns

left: pd.DataFrame The first DataFrame with the variables removed

right: pd.DataFrame The second DataFrame with the variables added

Examples

```
>>> import clarite
>>> df_cat, df_cont = clarity.modify.move_variables(df_cat, df_cont,␣
↪only=["DRD350AQ", "DRD350DQ", "DRD350GQ"])
Moved 3 variables.
>>> discovery_check, discovery_cont = clarite.modify.move_
↪variables(discovery_check, discovery_cont)
Moved 39 variables.
```

clarite.modify.recode_values

`clarite.modify.`**`recode_values`**(*data, replacement_dict, skip: Union[str, List[str], NoneType] = None, only: Union[str, List[str], NoneType] = None*)

Convert values in a dataframe. By default, replacement occurs in all columns but this may be modified with 'skip' or 'only'. Pandas has more powerful 'replace' methods for more complicated scenarios.

Parameters

data: pd.DataFrame The DataFrame to be processed and returned

replacement_dict: dictionary A dictionary mapping the value being replaced to the value being inserted

skip: str, list or None (default is None) List of variables that the replacement should *not* be applied to

only: str, list or None (default is None) List of variables that the replacement should *only* be applied to

Examples

```
>>> import clarite
>>> clarite.modify.recode_values(df, {7: np.nan, 9: np.nan}, only=[
↪'SMQ077', 'DBD100'])
================================================================================
Running recode_values
--------------------------------------------------------------------------
↪-------
Replaced 17 values from 22,624 observations in 2 variables
>>> clarite.modify.recode_values(df, {10: 12}, only=['SMQ077', 'DBD100'])
================================================================================
Running recode_values
--------------------------------------------------------------------------
↪-------
No occurences of replaceable values were found, so nothing was replaced.
```

clarite.modify.remove_outliers

`clarite.modify.`**`remove_outliers`**(*data, method: str = 'gaussian', cutoff=3, skip: Union[str, List[str], NoneType] = None, only: Union[str, List[str], NoneType] = None*)

Remove outliers from continuous variables by replacing them with np.nan

Parameters

data: pd.DataFrame The DataFrame to be processed and returned

method: string, 'gaussian' (default) or 'iqr' Define outliers using a gaussian approach (standard deviations from the mean) or inter-quartile range

cutoff: positive numeric, default of 3 Either the number of standard deviations from the mean (method='gaussian') or the multiple of the IQR (method='iqr') Any values equal to or more extreme will be replaced with np.nan

skip: str, list or None (default is None) List of variables that the replacement should *not* be applied to

only: str, list or None (default is None) List of variables that the replacement should *only* be applied to

Examples

```
>>> import clarite
>>> nhanes_rm_outliers = clarite.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)
```

(continues on next page)

(continued from previous page)

```
>>> nhanes_rm_outliers = clarite.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)
```

clarite.modify.rowfilter_incomplete_obs

`clarite.modify.rowfilter_incomplete_obs`(*data, skip: Union[str, List[str], NoneType] = None, only: Union[str, List[str], NoneType] = None*)

Remove rows containing null values

Parameters

data: pd.DataFrame The DataFrame to be processed and returned

skip: str, list or None (default is None) List of columns that are not checked for null values

only: str, list or None (default is None) List of columns that are the only ones to be checked for null values

Returns

data: pd.DataFrame The filtered DataFrame

Examples

```
>>> import clarite
>>> nhanes_filtered = clarite.modify.rowfilter_incomplete_obs(nhanes,␣
↪only=[phenotype] + covariates)
=================================================================================
Running rowfilter_incomplete_obs
--------------------------------------------------------------------------------
↪-------
Removed 3,687 of 22,624 observations (16.30%) due to NA values in any of␣
↪8 variables
```

clarite.modify.transform

`clarite.modify.transform`(*data: pandas.core.frame.DataFrame, transform_method: str, skip: Union[str, List[str], NoneType] = None, only: Union[str, List[str], NoneType] = None*)

Apply a transformation function to a variable

Parameters

data: pd.DataFrame or pd.Series Data to be processed

transform_method: str Name of the transformation (Python function or NumPy ufunc to apply)

skip: str, list or None (default is None) List of variables that will *not* be transformed

only: str, list or None (default is None) List of variables that are the *only* ones to be transformed

Returns

data: pd.DataFrame DataFrame with variables that have been transformed

Examples

```
>>> import clarite
>>> df = clarite.modify.transform(df, 'log', only=['BMXBMI'])
=================================================================================
Running transform
--------------------------------------------------------------------------
↪-------
Transformed 'BMXBMI' using 'log'.
```

7.1.5 Plot

Functions that generate plots

histogram(data, column, figsize, int] = (12, . . .)	Plot a histogram of the values in the given column.
distributions(data, filename, . . .)	Create a pdf containing histograms for each binary or categorical variable, and one of several types of plots for each continuous variable.
manhattan(dfs, pandas.core.frame.DataFrame], . . .)	Create a Manhattan-like plot for a list of EWAS Results
manhattan_fdr(dfs, . . .)	Create a Manhattan-like plot for a list of EWAS Results using FDR significance
manhattan_bonferroni(dfs, . . .)	Create a Manhattan-like plot for a list of EWAS Results using Bonferroni significance
top_results(ewas_result, pvalue_name, . . .)	Create a dotplot for EWAS Results showing pvalues and beta coefficients

clarite.plot.histogram

clarite.plot.**histogram**(*data*, *column: str*, *figsize: Tuple[int, int] = (12, 5)*, *title: Union[str, NoneType] = None*, *figure: Union[figure, NoneType] = None*, ***kwargs*)

Plot a histogram of the values in the given column. Takes kwargs for seaborn's distplot.

Parameters

data: pd.DataFrame The DataFrame containing data to be plotted

column: string The name of the column that will be plotted

figsize: tuple(int, int), default (12, 5) The figure size of the resulting plot

title: string or None, default None The title used for the plot

figure: matplotlib Figure or None, default None Pass in an existing figure to plot to that instead of creating a new one (ignoring figsize)

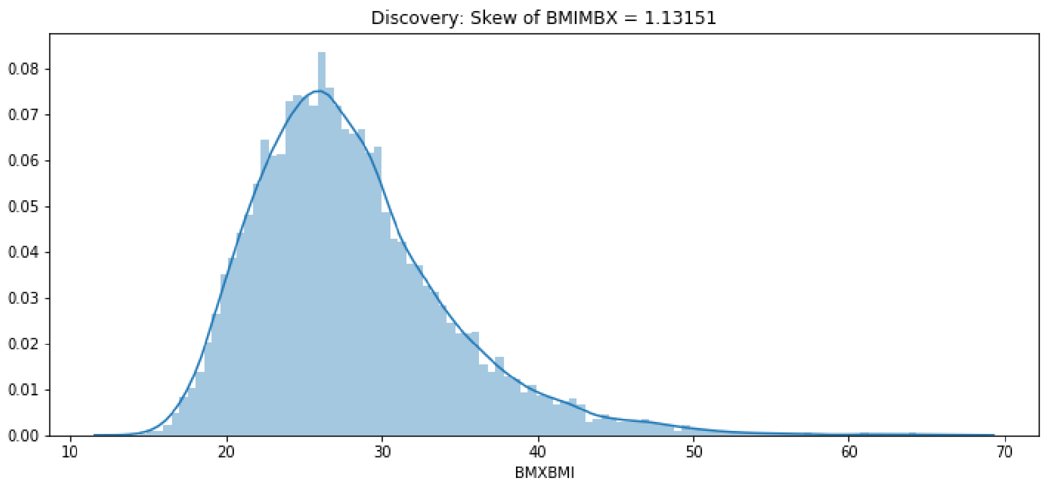
****kwargs:** Other keyword arguments to pass to the distplot function of Seaborn

Returns

None

Examples

```
>>> import clarite
>>> title = f"Discovery: Skew of BMIMBX = {stats.skew(nhanes_discovery_
↪cont['BMXBMI']):.6}"
>>> clarite.plot.histogram(nhanes_discovery_cont, column="BMXBMI",␣
↪title=title, bins=100)
```



clarite.plot.distributions

`clarite.plot.distributions`(*data, filename: str, continuous_kind: str = 'count', nrows: int = 4, ncols: int = 3, quality: str = 'medium', variables: Union[List[str], NoneType] = None, sort: bool = True*)

Create a pdf containing histograms for each binary or categorical variable, and one of several types of plots for each continuous variable.

Parameters

data: pd.DataFrame The DataFrame containing data to be plotted

filename: string Name of the saved pdf file. The extension will be added automatically if it was not included.

continuous_kind: string What kind of plots to use for continuous data. Binary and Categorical variables will always be shown with histograms. One of {'count', 'box', 'violin', 'qq'}

nrows: int (default=4) Number of rows per page

ncols: int (default=3) Number of columns per page

quality: 'low', 'medium', or 'high' Adjusts the DPI of the plots (150, 300, or 1200)

variables: List[str] or None Which variables to plot. If None, all variables are plotted.

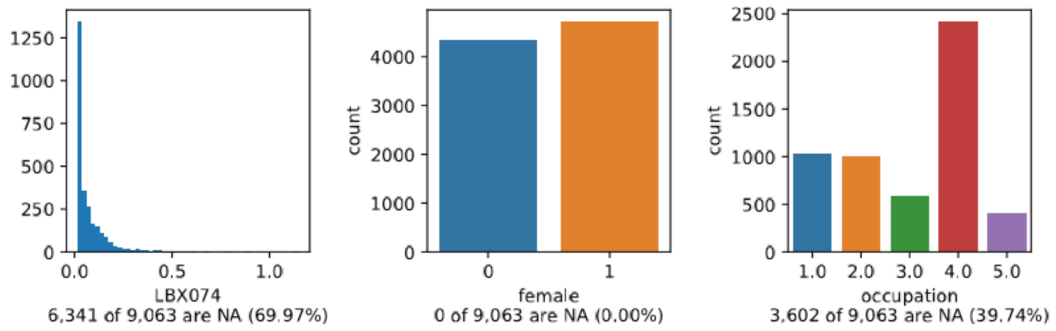
sort: Boolean (default=True) Whether or not to sort variable names

Returns

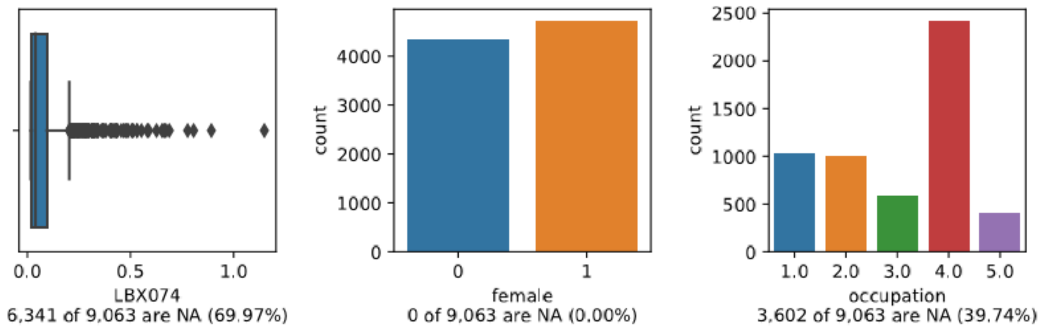
None

Examples

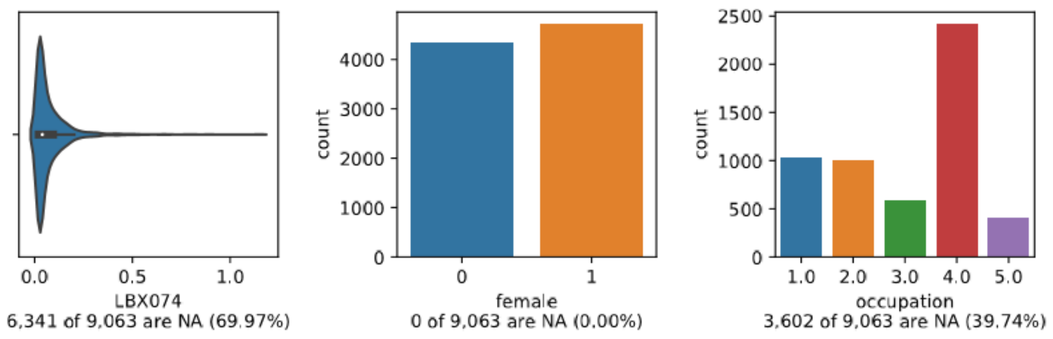
```
>>> import clarite
>>> clarite.plot.distributions(df[['female', 'occupation', 'LBX074']],␣
↪filename="test")
```



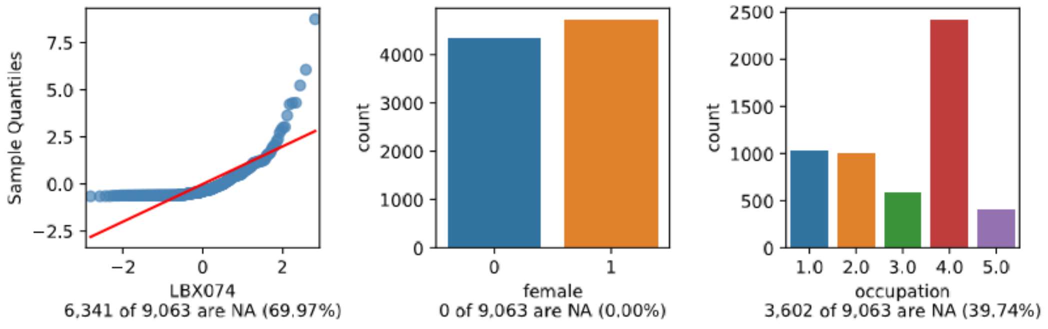
```
>>> clarite.plot.distributions(df[['female', 'occupation', 'LBX074']],␣
↪filename="test", continuous_kind='box')
```



```
>>> clarite.plot.distributions(df[['female', 'occupation', 'LBX074']],␣
↪filename="test", continuous_kind='violin')
```



```
>>> clarite.plot.distributions(df[['female', 'occupation', 'LBX074']],
↪filename="test", continuous_kind='qq')
```



clarite.plot.manhattan

`clarite.plot.manhattan`(*dfs: Dict[str, pandas.core.frame.DataFrame], categories: Dict[str, str] = {}, bonferroni: Union[float, NoneType] = 0.05, fdr: Union[float, NoneType] = None, num_labeled: int = 3, label_vars: List[str] = [], figsize: Tuple[int, int] = (12, 6), dpi: int = 300, title: Union[str, NoneType] = None, figure: Union[figure, NoneType] = None, colors: List[str] = ['#53868B', '#4D4D4D'], background_colors: List[str] = ['#EBEBEB', '#FFFFFF'], filename: Union[str, NoneType] = None*)

Create a Manhattan-like plot for a list of EWAS Results

Parameters

dfs: DataFrame Dictionary of dataset names to pandas dataframes of ewas results (requires certain columns)

categories: dictionary (string: string) A dictionary mapping each variable name to a category name

bonferroni: float or None (default 0.05) Show a cutoff line at the pvalue corresponding to a given bonferroni-corrected pvalue

fdr: float or None (default None) Show a cutoff line at the pvalue corresponding to a given fdr

num_labeled: int, default 3 Label the top <num_labeled> results with the variable name

label_vars: list of strings, default empty list Label the named variables

figsize: tuple(int, int), default (12, 6) The figure size of the resulting plot in inches

dpi: int, default 300 The figure dots-per-inch

title: string or None, default None The title used for the plot

figure: matplotlib Figure or None, default None Pass in an existing figure to plot to that instead of creating a new one (ignoring figsize and dpi)

colors: List(string, string), default ["#53868B", "#4D4D4D"] A list of colors to use for alternating categories (must be same length as 'background_colors')

background_colors: List(string, string), default ["#EBEBEB", "#FFFFFF"] A list of background colors to use for alternating categories (must be same length as 'colors')

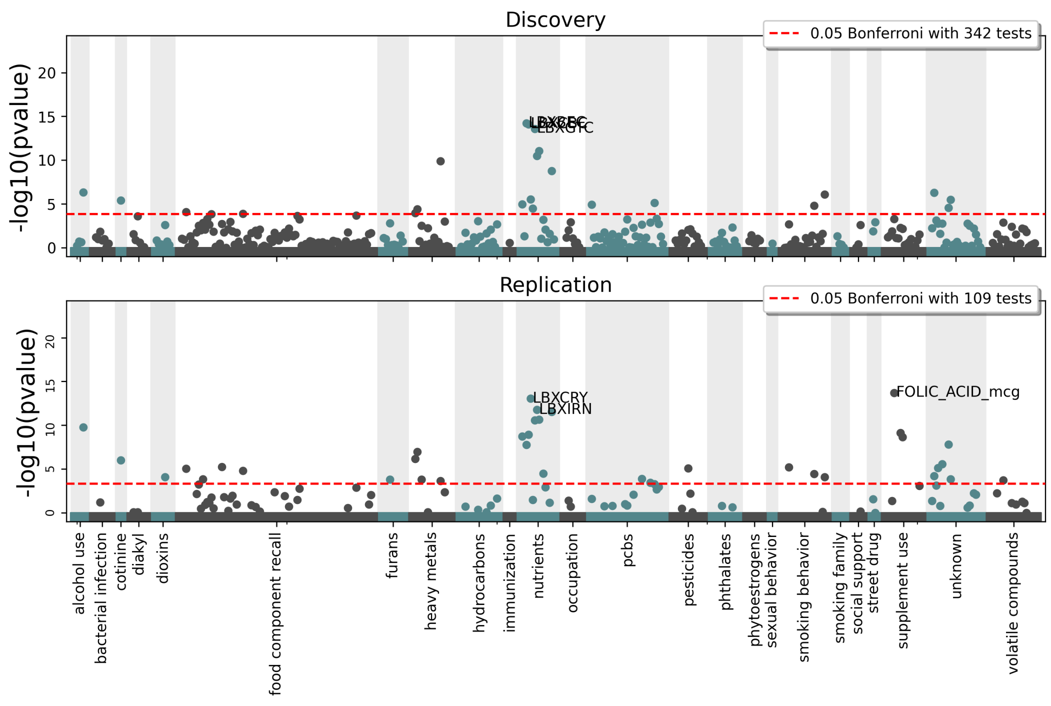
filename: Optional str If provided, a copy of the plot will be saved to the specified file

Returns

None

Examples

```
>>> clarite.plot.manhattan({'discovery':disc_df, 'replication':repl_df},␣
↪categories=data_categories, title="EWAS Results")
```



clarite.plot.manhattan_fdr

`clarite.plot.manhattan_fdr`(*dfs: Dict[str, pandas.core.frame.DataFrame], categories: Dict[str, str] = {}, cutoff: Union[float, NoneType] = 0.05, num_labeled: int = 3, label_vars: List[str] = [], figsize: Tuple[int, int] = (12, 6), dpi: int = 300, title: Union[str, NoneType] = None, figure: Union[figure, NoneType] = None, colors: List[str] = ['#53868B', '#4D4D4D'], background_colors: List[str] = ['#EBEBEB', '#FFFFFF'], filename: Union[str, NoneType] = None*)

Create a Manhattan-like plot for a list of EWAS Results using FDR significance

Parameters

dfs: DataFrame Dictionary of dataset names to pandas dataframes of ewas results (requires certain columns)

categories: dictionary (string: string) A dictionary mapping each variable name to a category name

cutoff: float or None (default 0.05) The pvalue to draw the FDR significance line at (None for no line)

num_labeled: int, default 3 Label the top <num_labeled> results with the variable name

label_vars: list of strings, default empty list Label the named variables

figsize: tuple(int, int), default (12, 6) The figure size of the resulting plot in inches

dpi: int, default 300 The figure dots-per-inch

title: string or None, default None The title used for the plot

figure: matplotlib Figure or None, default None Pass in an existing figure to plot to that instead of creating a new one (ignoring figsize and dpi)

colors: List(string, string), default ["#53868B", "#4D4D4D"] A list of colors to use for alternating categories (must be same length as 'background_colors')

background_colors: List(string, string), default ["#EBEBEB", "#FFFFFF"] A list of background colors to use for alternating categories (must be same length as 'colors')

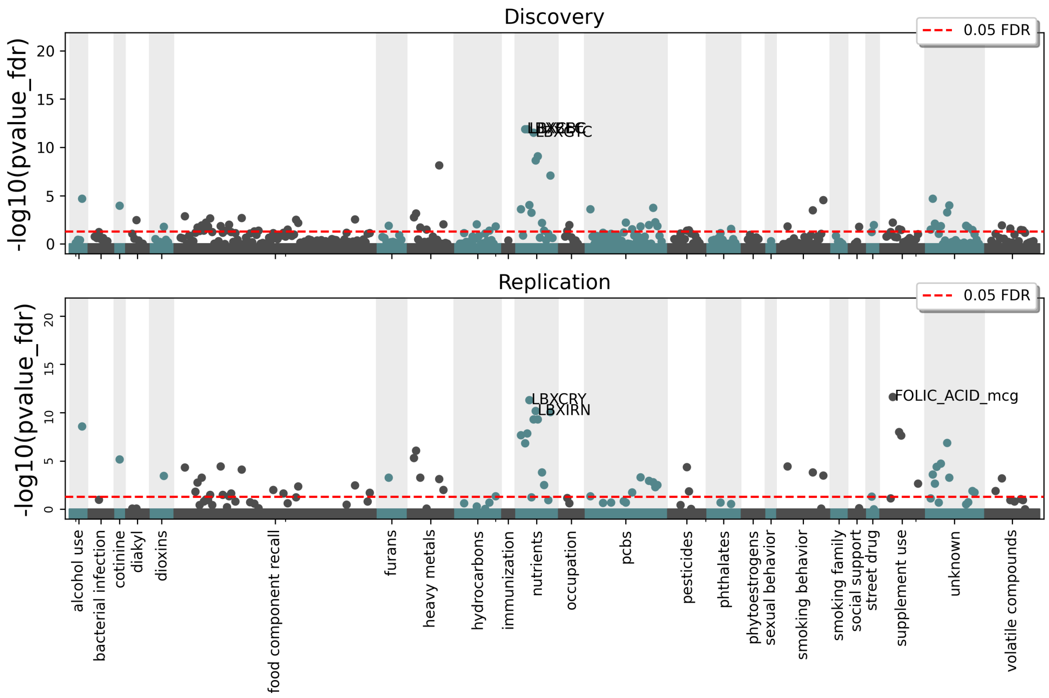
filename: Optional str If provided, a copy of the plot will be saved to the specified file

Returns

None

Examples

```
>>> clarite.plot.manhattan_fdr({'discovery':disc_df, 'replication':repl_
↪df},
 categories=data_categories, title="EWAS Results")
```



clarite.plot.manhattan_bonferroni

`clarite.plot.manhattan_bonferroni`(*dfs: Dict[str, pandas.core.frame.DataFrame], categories: Dict[str, str] = {}, cutoff: Union[float, NoneType] = 0.05, num_labeled: int = 3, label_vars: List[str] = [], figsize: Tuple[int, int] = (12, 6), dpi: int = 300, title: Union[str, NoneType] = None, figure: Union[figure, NoneType] = None, colors: List[str] = ['#53868B', '#4D4D4D'], background_colors: List[str] = ['#EBEBEB', '#FFFFFF'], filename: Union[str, NoneType] = None*)

Create a Manhattan-like plot for a list of EWAS Results using Bonferroni significance

Parameters

dfs: DataFrame Dictionary of dataset names to pandas dataframes of ewas results (requires certain columns)

categories: dictionary (string: string) A dictionary mapping each variable name to a category name

cutoff: float or None (default 0.05) The pvalue to draw the Bonferroni significance line at (None for no line)

num_labeled: int, default 3 Label the top <num_labeled> results with the variable name

label_vars: list of strings, default empty list Label the named variables

figsize: tuple(int, int), default (12, 6) The figure size of the resulting plot in inches

dpi: int, default 300 The figure dots-per-inch

title: string or None, default None The title used for the plot

figure: matplotlib Figure or None, default None Pass in an existing figure to plot to that instead of creating a new one (ignoring figsize and dpi)

colors: List(string, string), default ["#53868B", "#4D4D4D"] A list of colors to use for alternating categories (must be same length as 'background_colors')

background_colors: List(string, string), default ["#EBEBEB", "#FFFFFF"] A list of background colors to use for alternating categories (must be same length as 'colors')

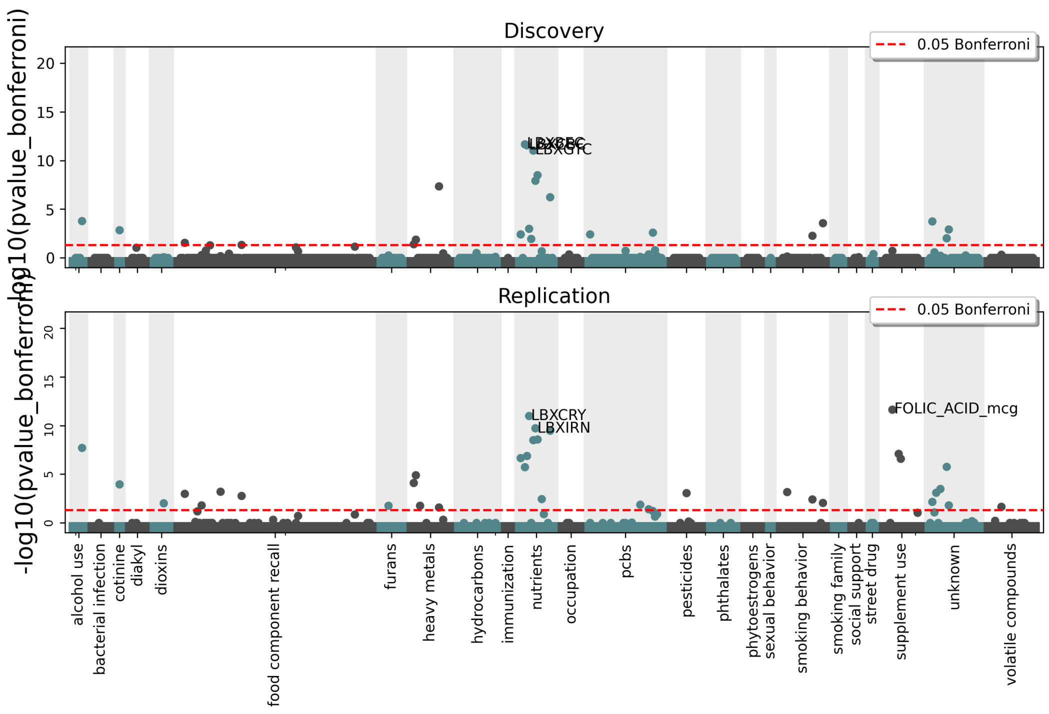
filename: Optional str If provided, a copy of the plot will be saved to the specified file

Returns

None

Examples

```
>>> clarite.plot.manhattan_bonferroni({'discovery':disc_df, 'replication
↪':repl_df},
categories=data_categories, title="EWAS Results")
```



clarite.plot.top_results

`clarite.plot.`**`top_results`**(*ewas_result: pandas.core.frame.DataFrame*, *pvalue_name: str = 'pvalue'*, *cutoff: float = 0.05*, *num_rows: int = 20*, *filename: Union[str, NoneType] = None*)

Create a dotplot for EWAS Results showing pvalues and beta coefficients

Parameters

ewas_result: DataFrame EWAS Result to plot

pvalue_name: str 'pvalue', 'pvalue_fdr', or 'pvalue_bonferroni'

cutoff: float (default 0.05) A vertical line is drawn in the pvalue column to show a significance cutoff

num_rows: int (default 20) How many rows to show in the plot

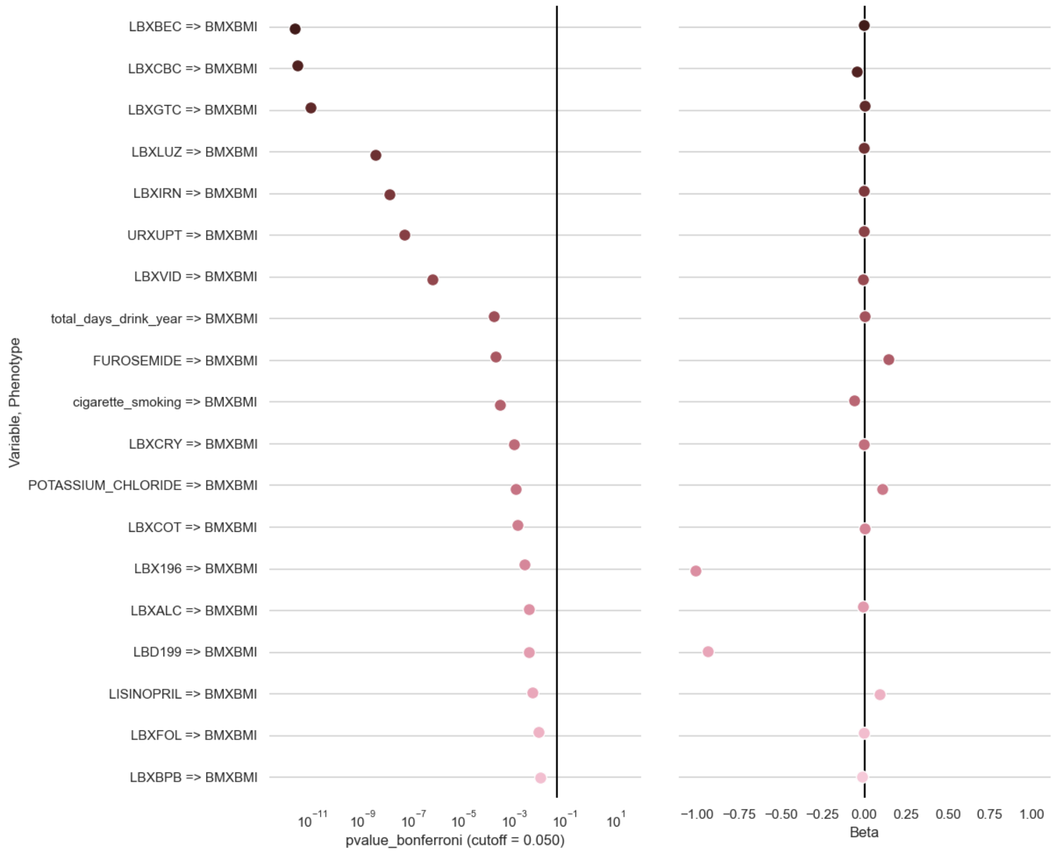
filename: Optional str If provided, a copy of the plot will be saved to the specified file

Returns

None

Examples

```
>>> clarite.plot.top_results(ewas_result)
```



7.1.6 Survey

Complex survey design

SurveyDesignSpec(survey_df, strata, cluster, ...)	Holds parameters for building a statsmodels SurveyDesign object

clarite.survey.SurveyDesignSpec

class clarite.survey.**SurveyDesignSpec**(*survey_df: pandas.core.frame.DataFrame*, *strata: Optional[str] = None*, *cluster: Optional[str] = None*, *nest: bool = False*, *weights: Union[str, Dict[str, str]] = None*, *fpc: Optional[str] = None*, *single_cluster: Optional[str] = 'fail'*, *drop_unweighted: bool = False*)

Holds parameters for building a statsmodels SurveyDesign object

Parameters

survey_df: pd.DataFrame A DataFrame containing Cluster, Strata, and/or weights data

strata: string or None The name of the strata variable in the survey_df

cluster: string or None The name of the cluster variable in the survey_df

nest: bool, default False Whether or not the clusters are nested in the strata (The same cluster IDs are repeated in different strata)

weights: string or dictionary(string:string) The name of the weights variable in the survey_df, or a dictionary mapping variable names to weight names

fpc: string or None The name of the variable in the survey_df that contains the finite population correction information. This reduces variance when a substantial portion of the population is sampled. May be specified as the total population size, or the fraction of the population that was sampled.

single_cluster: str Setting controlling variance calculation in single-cluster ('lonely psu') strata 'fail': default, throw an error 'adjust': use the average of all observations (more conservative) 'average': use the average value of other strata 'certainty': that strata doesn't contribute to the variance (0 variance)

drop_unweighted: bool, default False If True, drop observations that are missing a weight value. This may not be statistically sound. Otherwise the result for variables with missing weights (when the variable is not missing) is NULL.

Examples

```
>>> import clarite
>>> clarite.analyze.SurveyDesignSpec(survey_df=survey_design_replication,
                                     strata="SDMVSTRA",
                                     cluster="SDMVPSU",
                                     nest=True,
                                     weights=weights_replication,
                                     fpc=None,
                                     single_cluster='fail')
```

__init__(*self*, *survey_df: pandas.core.frame.DataFrame*, *strata: Union[str, NoneType] = None*, *cluster: Union[str, NoneType] = None*, *nest: bool = False*, *weights: Union[str, Dict[str, str]] = None*, *fpc: Union[str, NoneType] = None*, *single_cluster: Union[str, NoneType] = 'fail'*, *drop_unweighted: bool = False*)

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(self, survey_df, strata, . . .)	Initialize self.
`get_survey_design`(self, regression_variable, . . .)	Build a survey design based on the regression variable

CHAPTER 8

CLI Reference

Documentation for using the CLI

8.1 CLI Reference

Once CLARITE is installed, the command line interface can be run using the `clarte-cli` command.

The command line interface has command groups that are the same as the modules in the package (except for *survey*).

The `--help` option will show documentation when run with any command or command group:

```
$ clarite-cli --help
Usage: clarite-cli [OPTIONS] COMMAND [ARGS]...

Options:
--help  Show this message and exit.

Commands:
  analyze
  describe
  load
  modify
  plot
```

8.1.1 –skip and –only

Many commands in the CLI have the *skip* and *only* options. These will limit the command to specific variables. If *skip* is specified, all variables except the specified ones will be processed. If *only* is specified, only the specified variables will be processed.

Only one or the other option may be used in a single command. They may be passed in any combination of two ways:

1. As the name of a file containing one variable name per line

2. As the variable name specfied directly in the terminal

For example:

results in:

```
--------------------------------------------------------------------------------------
↪-----------------------------------
--only: 1 variable(s) specified directly
        8 variable(s) loaded from 'covars.txt'
===============================================================================================================
Running rowfilter_incomplete_obs
--------------------------------------------------------------------------------------
↪-----------------------------------
Removed 3,687 of 22,624 observations (16.30%) due to NA values in any of 9 variables
===============================================================================================================
```

8.1.2 Commands

clarite-cli analyze

```
clarite-cli analyze [OPTIONS] COMMAND [ARGS]...
```

add-corrected-pvals

Get FDR-corrected and Bonferroni-corrected pvalues

```
clarite-cli analyze add-corrected-pvals [OPTIONS] EWAS_RESULT OUTPUT
```

Arguments

EWAS_RESULT
Required argument

OUTPUT
Required argument

ewas

Run an EWAS analysis

```
clarite-cli analyze ewas [OPTIONS] PHENOTYPE DATA OUTPUT
```

Options

-c, --covariate `<covariate>`
Covariates

--covariance-calc `<covariance_calc>`
Covariance calculation method

Options stata|jackknife

--min-n <min_n>
Minimum number of complete cases needed to run a regression

--survey-data <survey_data>
Tab-separated data file with survey weights, strata IDs, and/or cluster IDs. Must have an 'ID' column.

--strata <strata>
Name of the strata column in the survey data

--cluster <cluster>
Name of the cluster column in the survey data

--nested, --not-nested
Whether survey data is nested or not

--weights-file <weights_file>
Tab-delimited data file with 'Variable' and 'Weight' columns to match weights from the survey data to specific variables

-w, --weight <weight>
Name of a survey weight column found in the survey data. This option can't be used with –weights-file

--fpc <fpc>
Name of the finite population correction column in the survey data

--single-cluster <single_cluster>
How to handle singular clusters

Options fail|adjust|average|certainty

Arguments

PHENOTYPE
Required argument

DATA
Required argument

OUTPUT
Required argument

ewas-r

Run an EWAS analysis using R

```
clarite-cli analyze ewas-r [OPTIONS] PHENOTYPE DATA OUTPUT
```

Options

-c, --covariate <covariate>
Covariates

--covariance-calc <covariance_calc>
Covariance calculation method

Options stata|jackknife

--min-n <min_n>
Minimum number of complete cases needed to run a regression

--survey-data <survey_data>
Tab-separated data file with survey weights, strata IDs, and/or cluster IDs. Must have an 'ID' column.

--strata <strata>
Name of the strata column in the survey data

--cluster <cluster>
Name of the cluster column in the survey data

--nested, --not-nested
Whether survey data is nested or not

--weights-file <weights_file>
Tab-delimited data file with 'Variable' and 'Weight' columns to match weights from the survey data to specific variables

-w, --weight <weight>
Name of a survey weight column found in the survey data. This option can't be used with –weights-file

--fpc <fpc>
Name of the finite population correction column in the survey data

--single-cluster <single_cluster>
How to handle singular clusters

Options fail|adjust|average|certainty

Arguments

PHENOTYPE
Required argument

DATA
Required argument

OUTPUT
Required argument

get-significant

filter out non-significant results

```
clarite-cli analyze get-significant [OPTIONS] EWAS_RESULT OUTPUT
```

Options

--fdr, --bonferroni
Use FDR (–fdr) or Bonferroni pvalues (–bonferroni). FDR by default.

-p, --pvalue <pvalue>
Keep results with a pvalue <= this value (0.05 by default)

Arguments

EWAS_RESULT
Required argument

OUTPUT
Required argument

clarite-cli describe

```
clarite-cli describe [OPTIONS] COMMAND [ARGS]...
```

correlations

Report top correlations between variables

```
clarite-cli describe correlations [OPTIONS] DATA OUTPUT
```

Options

-t, --threshold `<threshold>`
Report correlations with R >= this value

Arguments

DATA
Required argument

OUTPUT
Required argument

freq-table

Report the number of occurences of each value for each variable

```
clarite-cli describe freq-table [OPTIONS] DATA OUTPUT
```

Arguments

DATA
Required argument

OUTPUT
Required argument

get-types

Get the type of each variable

```
clarite-cli describe get-types [OPTIONS] DATA OUTPUT
```

Arguments

DATA
Required argument

OUTPUT
Required argument

percent-na

Report the percent of observations that are NA for each variable

```
clarite-cli describe percent-na [OPTIONS] DATA OUTPUT
```

Arguments

DATA
Required argument

OUTPUT
Required argument

skewness

Report and test the skewness for each continuous variable

```
clarite-cli describe skewness [OPTIONS] DATA OUTPUT
```

Options

--dropna, --keepna
Omit NA values before calculating skew

Arguments

DATA
Required argument

OUTPUT
Required argument

clarite-cli load

```
clarite-cli load [OPTIONS] COMMAND [ARGS]...
```

from-csv

Load data from a comma-separated file and save it in the standard format

```
clarite-cli load from-csv [OPTIONS] INPUT OUTPUT
```

Options

-i, --index `<index>`
Name of the column to use as the index. Default is the first column.

-s, --skip `<skip>`
variables to skip. Either individual names, or a file containing one name per line.

-o, --only `<only>`
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

INPUT
Required argument

OUTPUT
Required argument

from-tsv

Load data from a tab-separated file and save it in the standard format

```
clarite-cli load from-tsv [OPTIONS] INPUT OUTPUT
```

Options

-i, --index `<index>`
Name of the column to use as the index. Default is the first column.

-s, --skip `<skip>`
variables to skip. Either individual names, or a file containing one name per line.

-o, --only `<only>`
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

INPUT
Required argument

OUTPUT
Required argument

clarite-cli modify

```
clarite-cli modify [OPTIONS] COMMAND [ARGS]...
```

categorize

Categorize data based on the number of unique values

```
clarite-cli modify categorize [OPTIONS] DATA OUTPUT
```

Options

--cat_min <cat_min>
Minimum number of unique values in a variable to make it a categorical type

--cat_max <cat_max>
Maximum number of unique values in a variable to make it a categorical type

--cont_min <cont_min>
Minimum number of unique values in a variable to make it a continuous type

Arguments

DATA
Required argument

OUTPUT
Required argument

colfilter

Remove some variables from a dataset

```
clarite-cli modify colfilter [OPTIONS] DATA OUTPUT
```

Options

-s, --skip <skip>
variables to skip. Either individual names, or a file containing one name per line.

-o, --only <only>
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

DATA
Required argument

OUTPUT
Required argument

colfilter-min-cat-n

Filter variables based on a minimum number of non-NA observations per category

```
clarite-cli modify colfilter-min-cat-n [OPTIONS] DATA OUTPUT
```

Options

-n <n>
Remove variables with less than this many non-na observations in each category

-s, --skip <skip>
variables to skip. Either individual names, or a file containing one name per line.

-o, --only <only>
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

DATA
Required argument

OUTPUT
Required argument

colfilter-min-n

Filter variables based on a minimum number of non-NA observations

```
clarite-cli modify colfilter-min-n [OPTIONS] DATA OUTPUT
```

Options

-n <n>
Remove variables with less than this many non-na observations

-s, --skip <skip>
variables to skip. Either individual names, or a file containing one name per line.

-o, --only <only>
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

DATA
Required argument

OUTPUT
Required argument

colfilter-percent-zero

Filter variables based on the fraction of observations with a value of zero

```
clarite-cli modify colfilter-percent-zero [OPTIONS] DATA OUTPUT
```

Options

-p, --filter-percent <filter_percent>
Remove variables when the percentage of observations equal to 0 is >= this value (0 to 100)

-s, --skip <skip>
variables to skip. Either individual names, or a file containing one name per line.

-o, --only <only>
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

DATA
Required argument

OUTPUT
Required argument

drop-extra-categories

Remove extra categories from categorical datatypes

```
clarite-cli modify drop-extra-categories [OPTIONS] DATA OUTPUT
```

Options

-s, --skip <skip>
variables to skip. Either individual names, or a file containing one name per line.

-o, --only <only>
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

DATA
Required argument

OUTPUT
Required argument

make-binary

Set the type of variables to 'binary'

```
clarite-cli modify make-binary [OPTIONS] DATA OUTPUT
```

Options

-s, --skip <skip>
variables to skip. Either individual names, or a file containing one name per line.

-o, --only <only>
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

DATA
Required argument

OUTPUT
Required argument

make-categorical

Set the type of variables to 'categorical'

```
clarite-cli modify make-categorical [OPTIONS] DATA OUTPUT
```

Options

-s, --skip <skip>
variables to skip. Either individual names, or a file containing one name per line.

-o, --only <only>
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

DATA
Required argument

OUTPUT
Required argument

make-continuous

Set the type of variables to 'continuous'

```
clarite-cli modify make-continuous [OPTIONS] DATA OUTPUT
```

Options

-s, --skip <skip>
variables to skip. Either individual names, or a file containing one name per line.

-o, --only <only>
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

DATA
Required argument

OUTPUT
Required argument

merge-observations

Merge observations from two different datasets into one

```
clarite-cli modify merge-observations [OPTIONS] TOP BOTTOM OUTPUT
```

Arguments

TOP
Required argument

BOTTOM
Required argument

OUTPUT
Required argument

merge-variables

Merge variables from two different datasets into one

```
clarite-cli modify merge-variables [OPTIONS] LEFT RIGHT OUTPUT
```

Options

-h, --how <how>
Type of Merge

Options left|right|inner|outer

Arguments

LEFT
Required argument

RIGHT
Required argument

OUTPUT
Required argument

move-variables

Move variables from one dataset to another

```
clarite-cli modify move-variables [OPTIONS] LEFT RIGHT
```

Options

--output_left `<output_left>`

--output_right `<output_right>`

-s, --skip `<skip>`
variables to skip. Either individual names, or a file containing one name per line.

-o, --only `<only>`
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

LEFT
Required argument

RIGHT
Required argument

recode-values

Replace values in the data with other values.The value being replaced ('current') and the new value ('replacement') are specified with their type, and only one may be included for each. If it is not specified, the value being replaced or being inserted is None.

```
clarite-cli modify recode-values [OPTIONS] DATA OUTPUT
```

Options

--current-str `<cs>`
Replace occurences of this string value

--current-int `<ci>`
Replace occurences of this integer value

--current-float <cf>
Replace occurences of this float value

--replacement-str <rs>
Insert this string value

--replacement-int <ri>
Insert this integer value

--replacement-float <rf>
Insert this float value

-s, --skip <skip>
variables to skip. Either individual names, or a file containing one name per line.

-o, --only <only>
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

DATA
Required argument

OUTPUT
Required argument

remove-outliers

Replace outlier values with NaN. Outliers are defined using a gaussian or IQR approach.

```
clarite-cli modify remove-outliers [OPTIONS] DATA OUTPUT
```

Options

-m, --method <method>

Options gaussian|iqr

-c, --cutoff <cutoff>

-s, --skip <skip>
variables to skip. Either individual names, or a file containing one name per line.

-o, --only <only>
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

DATA
Required argument

OUTPUT
Required argument

rowfilter

Select some rows from a dataset using a simple comparison, keeping rows where the comparison is True.

```
clarite-cli modify rowfilter [OPTIONS] DATA OUTPUT COLUMN
```

Options

--value-str `<vs>`
Compare values in the column to this string

--value-int `<vi>`
Compare values in the column to this integer

--value-float `<vf>`
Compare values in the column to this floating point number

-c, --comparison `<comparison>`
Keep rows where the value of the column is lt (<), lte (<=), eq (==), gte (>=), or gt (>) the specified value. Eq by default.

Options lt|lte|eq|gte|gt

Arguments

DATA
Required argument

OUTPUT
Required argument

COLUMN
Required argument

rowfilter-incomplete-obs

Filter out observations that are not complete cases (contain no NA values)

```
clarite-cli modify rowfilter-incomplete-obs [OPTIONS] DATA OUTPUT
```

Options

-s, --skip `<skip>`
variables to skip. Either individual names, or a file containing one name per line.

-o, --only `<only>`
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

DATA
Required argument

OUTPUT
Required argument

transform-variable

Apply a function to each value of a variable

```
clarite-cli modify transform-variable [OPTIONS] DATA OUTPUT TRANSFORM_METHOD
```

Options

-s, --skip <skip>
variables to skip. Either individual names, or a file containing one name per line.

-o, --only <only>
variables to process, skipping all others. Either individual names, or a file containing one name per line.

Arguments

DATA
Required argument

OUTPUT
Required argument

TRANSFORM_METHOD
Required argument

clarite-cli plot

```
clarite-cli plot [OPTIONS] COMMAND [ARGS]...
```

distributions

Generate a pdf containing distribution plots for each variable

```
clarite-cli plot distributions [OPTIONS] DATA OUTPUT
```

Options

-k, --kind <kind>
Kind of plot used for continuous data. Non-continuous always shows a count plot.

Options count|box|violin|qq

--nrows <nrows>
Number of rows per page

--ncols <ncols>
Number of columns per page

-q, --quality <quality>
Quality of the generated plots: low (150 dpi), medium (300 dpi), or high (1200 dpi).

Options low|medium|high

--sort, --no-sort
Sort variables alphabetically

Arguments

DATA
Required argument

OUTPUT
Required argument

histogram

Create a histogram plot of a variable

```
clarite-cli plot histogram [OPTIONS] DATA OUTPUT VARIABLE
```

Arguments

DATA
Required argument

OUTPUT
Required argument

VARIABLE
Required argument

manhattan

Generate a manhattan plot of EWAS results

```
clarite-cli plot manhattan [OPTIONS] EWAS_RESULT OUTPUT
```

Options

-c, --categories <categories>
tab-separate file with two columns: ‘Variable’ and ‘category’

--bonferroni <bonferroni>
cutoff value to plot bonferroni-adjusted pvalue line

--fdr <fdr>
cutoff value to plot fdr-adjusted pvalue line

-o, --other <other>
other datasets to include in the plot

--nlabeled <nlabeled>
label top n points

--label <label>
label points by name

Arguments

EWAS_RESULT
Required argument

OUTPUT
Required argument

manhattan-bonferroni

Generate a manhattan plot of EWAS results showing Bonferroni-corrected pvalues

```
clarite-cli plot manhattan-bonferroni [OPTIONS] EWAS_RESULT OUTPUT
```

Options

-c, --categories <categories>
tab-separate file with two columns: 'Variable' and 'category'

--cutoff <cutoff>
cutoff value for plotting the significance line

--fdr <fdr>
cutoff value to plot Bonferroni-adjusted pvalue line

-o, --other <other>
other datasets to include in the plot

--nlabeled <nlabeled>
label top n points

--label <label>
label points by name

Arguments

EWAS_RESULT
Required argument

OUTPUT
Required argument

manhattan-fdr

Generate a manhattan plot of EWAS results showing FDR-corrected pvalues

```
clarite-cli plot manhattan-fdr [OPTIONS] EWAS_RESULT OUTPUT
```

Options

-c, --categories `<categories>`
: tab-separate file with two columns: 'Variable' and 'category'

--cutoff `<cutoff>`
: cutoff value for plotting the significance line

--fdr `<fdr>`
: cutoff value to plot fdr-adjusted pvalue line

-o, --other `<other>`
: other datasets to include in the plot

--nlabeled `<nlabeled>`
: label top n points

--label `<label>`
: label points by name

Arguments

EWAS_RESULT
: Required argument

OUTPUT
: Required argument

CHAPTER 9

Additional Notes

Release History, etc

9.1 Release History

9.1.1 v0.10.0 (2020-05-28)

Enhancements

- Manhattan plot split into three functions (raw, bonferroni, and fdr) and now has a custom threshold parameter
- Use Pandas v1.0+
- Refactored regression objects to simplify internal code and potentially allow for more types of regression in the future
- Added an ewas_r function that seamlessly runs the ewas analysis in R, using the R *survey* library * This is recommended when using weights, as the python version has some inconsistencies in some edge cases
- Added a skewness function
- Added a *top_results* plot
- Add a *drop_unweighted* parameter to the *SurveyDesignSpec* to provide an easy (if potentially incorrect) workaround for observations with missing weights

Fixes

- Provide a warning and a convenience function when categorical types have categories with no occurrences
- Catch errors when categorizing variables with many unique string values
- Corrected some edge-case EWAS results when using weights in the presence of missing values
- Avoid some cryptic errors by ensuring the input to some functions is a DataFrame and not a Series

Tests

Many additional tests were added, especially related to EWAS

9.1.2 v0.9.1 (2019-11-20)

Minor documentation update

9.1.3 v0.9.0 (2019-10-31)

Enhancements

- Add a *figure* parameter to histogram and manhattan plots in order to plot to an existing figure
- *SurveyDesignSpec* can now utilize more parameters, such as *fpc*
- The larger (numeric or alphabetic) binary variable is always treated as the success case for binary phenotypes
- Improved logging during EWAS, including printing the survey design information
- Extensively updated documentation
- CLARITE now has a logo!

Fixes

- Corrected an indexing error that sometimes occurred when removing rows with missing weights
- Improve precision in EWAS results for weighted analyses by using sf instead of 1-cdf
- Change some column names in the EWAS output to be more clear

Tests

An R script and the output of that script is now included. The R output is compared to the python output in the test suite in order to ensure analysis result concordance between R and Python for several analysis scenarios.

9.1.4 v0.8.0 (2019-09-03)

Enhancements

- Allow file input in the command line for skip/only
- Make the manhattan plot function less restrictive of the data passed into it
- Use skip/only in the transform function

Fixes

- Categorization would silently fail if there was only one variable of a given type

9.1.5 v0.7.0 (2019-07-23)

Enhancements

- Improvements to the CLI and printed log messages.
- The functions from the 'Process' module were put into the 'Modify' module.
- Datasets are no longer split apart when categorizing.

9.1.6 v0.6.0 (2019-07-11)

Extensive changes in organization, but limited new functionality (not counting the CLI).

Enhancements

- Reorganize functions - https://github.com/HallLab/clarite-python/pull/13
- Add a CLI - https://github.com/HallLab/clarite-python/pull/11

9.1.7 v0.5.0 (2019-06-28)

Enhancements

- Added a function to recode values - https://github.com/HallLab/clarite-python/issues/4
- Added a function to filter outlier values - https://github.com/HallLab/clarite-python/issues/5
- Added a function to generate manhattan plots for multiple datasets together - https://github.com/HallLab/clarite-python/issues/9

Fixes

- Add some validation of input DataFrames to prevent some errors in calculations

Tests

- Added an initial batch of tests

9.1.8 v0.4.0 (2019-06-18)

Support EWAS with binary outcomes. Additional handling of NA values in covariates and the phenotype. Add a 'min_n' parameter to the ewas function to require a minimum number of observations after removing incomplete cases. Add additional functions including 'plot_distributions', 'merge_variables', 'get_correlations', 'get_freq_table', and 'get_percent_na'

9.1.9 v0.3.0 (2019-05-31)

Add support for complex survey designs

9.1.10 v0.2.1 (2019-05-02)

Added documentation for existing functions

9.1.11 v0.2.0 (2019-04-30)

First functional version. Mutliple methods are available under a 'clarite' Pandas accessor.

9.1.12 v0.1.0 (2019-04-23)

Initial Release

Python Module Index

Index

Symbols