GEOparse Documentation

Release 0.1.10

Rafal Gumienny

Jul 21, 2017

Contents

Contents:

CHAPTER 1

GEOparse

Python library to access Gene Expression Omnibus Database (GEO).

GEOparse is python package that can be used to query and retrieve data from Gene Expression Omnibus database (GEO). The inspiration and the base for it is great R library GEOquery.

- Free software: BSD license
- Documentation: https://GEOparse.readthedocs.org.

Features

- Download GEO series, datasets etc. as SOFT files
- Download supplementary files for GEO series to use them locally
- Load GEO SOFT as easy to use and manipulate objects
- Prepare your data for GEO upload

Installation

At the command line:

```
$ pip install GEOparse
```

TODO

There is still work to do so any contribution is welcome. Any bug/error that you report will improve the library.

The main issues are:

- add checking for compatibility with SOFT files
- expand GEOTypes objects with useful functions for differential expression analysis
- share your idea
- add more tests - that’s always good idea :)

CHAPTER 2

Installation

At the command line:

```
$ pip install GEOparse
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv GEOparse
$ pip install GEOparse
```

Note: Not all dependencies will be installed automatically. The Biopython is only optional and it is used to interact with SRA entries. Please, install it if you are going to download SRA files.

CHAPTER 3

Usage

To simplest example of usage:

```
import GEOparse

gse = GEOparse.get_GEO(geo="GSE1563", destdir="./")

print()
print("GSM example:")
for gsm_name, gsm in gse.gsms.items():
    print("Name: ", gsm_name)
    print("Metadata:",)
    for key, value in gsm.metadata.items():
        print(" - %s : %s" % (key, ", ".join(value)))
    print ("Table data:",)
    print (gsm.table.head())
    break

print()
print("GPL example:")
for gpl_name, gpl in gse.gpls.items():
    print("Name: ", gpl_name)
    print("Metadata:",)
    for key, value in gpl.metadata.items():
        print(" - %s : %s" % (key, ", ".join(value)))
    print("Table data:",)
    print(gpl.table.head())
    break
```

Working with GEO accession

In order to download and load to the memory the experiment with specific GEO accession use `GEOparse.get_GEO()`.

Eg. if you would like to download the *GSE1563* the code:

```
import GEOparse
gse = GEOparse.get_GEO(geo="GSE1563", destdir="./")
```

will check the GEO database for this accession ID and it will download it to specified directory (by default this is CWD). The result will be loaded into `GEOparse.GSE`. For more information on how to work with GEOparse objects see working-with-objects.

Working with SOFT files

Reading SOFT file

If you already downloaded some GDS, GSE, GSM or GPL you can read it into object by explicitly specifying path to the file:

```
import GEOparse
gse = GEOparse.get_GEO(filepath="./GSE1563.soft.gz")
```

This will read the file into GSE object.

Crating and saving SOFT file

When doing own experiments one might want to submit the data to the GEO. In this case, one shuld first create all necessary elements of given GEO object, create this object and use `to_soft()` method to save it as SOFT files.

Warning: As for now there is no implementation of any check procedure if the tables and metadata contain all necessary information required by GEO. This could be introduced in the future, however, now it is entirely up to the user to follow the rules of GEO.

Here is the example of a faked GSE that will be saved as SOFT. There is three main components of GSE object: a dictionary of GSM objects, a dictionary of GPL objects and its own metadata:

```
# imports
import GEOparse
import pandas as pd

# prepare data for columns and table
columns_header = ['a', 'b', 'c']
columns_description = ['first column', 'second column', 'third column']
table_data = [[1, 2, 3],
              [4, 5, 6]]

# create table and columns
table = pd.DataFrame(table_data, columns=columns_header)
columns = pd.DataFrame(columns_description, columns_header)
columns.columns = ['description'] # columns header should contain description

# prepare metadata for objects. Each value of the dictionary should be a list
gpl_metadata = {'name': ['FooGPL']}
gsm_metadata = {'name': ['FooGSM']}
metadata = {'name': ['FooGSE']}
```

```
# initialize GPL and GSM object(s)
gpl = GEOparse.GPL(name='FooGPL', table=table, metadata=gpl_metadata, columns=columns)
gsm = GEOparse.GSM(name='FooGSM', table=table, metadata=gsm_metadata, columns=columns)

# prepare attributes for GSE
gsms = {'FooGSM': gsm}
gpls = {'FooGPL': gpl}

# initialize GSE
gse = GEOparse.GSE(name='FooGSE', metadata=metadata, gpls=gpls, gsms=gsms)

# save gse as SOFT file
gse.to_soft("./GSEFoo.soft")
```

This creates file with following content:

```
^SERIES = FooGSE
!Series_name = FooGSE
^SAMPLE = FooGSM
!Sample_name = FooGSM
#a = first column
#b = second column
#c = third column
!sample_table_begin
a    b    c
1    2    3
4    5    6
!sample_table_end
^PLATFORM = FooGPL
!Platform_name = FooGPL
#a = first column
#b = second column
#c = third column
!platform_table_begin
a    b    c
1    2    3
4    5    6
!platform_table_end
```

Of course in this case the file is simpler than the code that generates it but in normal situation this is reversed. For more information on what GEO objects are available and what parameters one need to create them see the working-with-objects section.

Working with GEO objects

BaseGEO

All GEO objects inherit from abstract base class `GEOparse.BaseGEO`. Two main attributes of that class are the *name* and *metadata*.

metadata is a dictionary of useful information about samples which occurs in the SOFT file with bang (!) in the beginning. Each value of this dictionary id a list (even with one element).

GSM (Sample)

A GSM (or a Sample) contains information the conditions and preparation of a Sample. In the GEO database sample is assigned to unique and stable GEO accession number that is composed of 'GSM' followed by numbers eg. GSM906.

In GEOparse Sample is represented by GEOparse.GSM object that contains tree main attributes:

- inherited from BaseGEO `metadata`
- `table` – `pandas.DataFrame` with the data table from SOFT file
- `columns` – `pandas.DataFrame` that contains *description* column with the information about columns in the `table`

See API for more information.

GPL (Platform)

A GPL (or a Platform) contains a tab-delimited table containing the array definition eg. mappings from probe IDs to RefSeq IDs. Similarly to GSM, it is assigned to unique and stable GEO accession number that is composed of 'GPL' followed by numbers eg. GPL2020.

In GEOparse Platform is represented by GEOparse.GSM object that contains tree main attributes:

- inherited from BaseGEO `metadata`
- `table` – `pandas.DataFrame` with the data table from SOFT file
- `columns` – `pandas.DataFrame` that contains *description* column with the information about columns in the `table`

See API for more information.

GSE (Series)

A GSE (or a Series) is an original submitter-supplied record that summarizes whole study including samples and platforms. GSE is assigned to unique and stable GEO accession number that starts at GSE followed by numbers eg. GSE1563.

In GEOparse Series is represented by GEOparse.GSE object that contains tree main attributes:

- inherited from BaseGEO `metadata`
- `gsms` – `dict` with all samples in this GSE as GSM objects
- `gpls` – `dict` with all platforms in this GSE as GSM objects

See API for more information.

GDS (Dataset)

A GDS (or a Dataset) is a curated file that hold a summarised combination of a Series file and its samples. GDS is assigned to unique and stable GEO accession number that starts at GDS followed by numbers eg. GDS1563.

In GEOparse Dataset is represented by GEOparse.GDS object that contains tree main attributes:

- inherited from BaseGEO `metadata`
- `table` – `pandas.DataFrame` with the data table from SOFT file

- `columns` – `pandas.DataFrame` that contains *description* column with the information about columns in the `table` and additional information according to GDS file

See API for more information.

Examples

Analyse hsa-miR-124a-3p transfection time-course

Note: In order to do this analysis you have to be in the tests directory of GEOparse.

In the paper *Systematic identification of microRNA functions by combining target prediction and expression profiling* Wang and Wang provided a series of microarrays from 7 time-points after miR-124a transfection. The series can be found in GEO under the GSE6207 accession. We use this series to demonstrate general principles of GEOparse.

Warning: Mind that this tutorial is not abut how to properly calculate log fold changes - the approach undertaken here is simplistic.

We start with the imports:

```
import GEOparse
import pandas as pd
import pylab as pl
import seaborn as sns
pl.rcParams['figure.figsize'] = (14, 10)
pl.rcParams['ytick.labelsize'] = 12
pl.rcParams['xtick.labelsize'] = 11
pl.rcParams['axes.labelsize'] = 23
pl.rcParams['legend.fontsize'] = 20
sns.set_style('ticks')
c1, c2, c3, c4 = sns.color_palette("Set1", 4)
```

Now we select the GSMs that are controls. See below on how to generate names of control samples directly from the phenotype data of GSE.

```
controls = ['GSM143386',
            'GSM143388',
            'GSM143390',
            'GSM143392',
            'GSM143394',
            'GSM143396',
            'GSM143398']
```

Using GEOparse we can download experiments and look into the data:

```
gse = GEOparse.get_GEO("GSE6207")
```

```
File already exist: using local version.
Parsing ./GSE6207.soft.gz:
 - DATABASE : GeoMiame
 - SERIES : GSE6207
 - PLATFORM : GPL570
 - SAMPLE : GSM143385
 - SAMPLE : GSM143386
 - SAMPLE : GSM143387
 - SAMPLE : GSM143388
 - SAMPLE : GSM143389
 - SAMPLE : GSM143390
 - SAMPLE : GSM143391
 - SAMPLE : GSM143392
 - SAMPLE : GSM143393
 - SAMPLE : GSM143394
 - SAMPLE : GSM143395
 - SAMPLE : GSM143396
 - SAMPLE : GSM143397
 - SAMPLE : GSM143398
```

The GPL we are interested:

```
gse.gpls['GPL570'].columns
```

And the columns that are available for exemplary GSM:

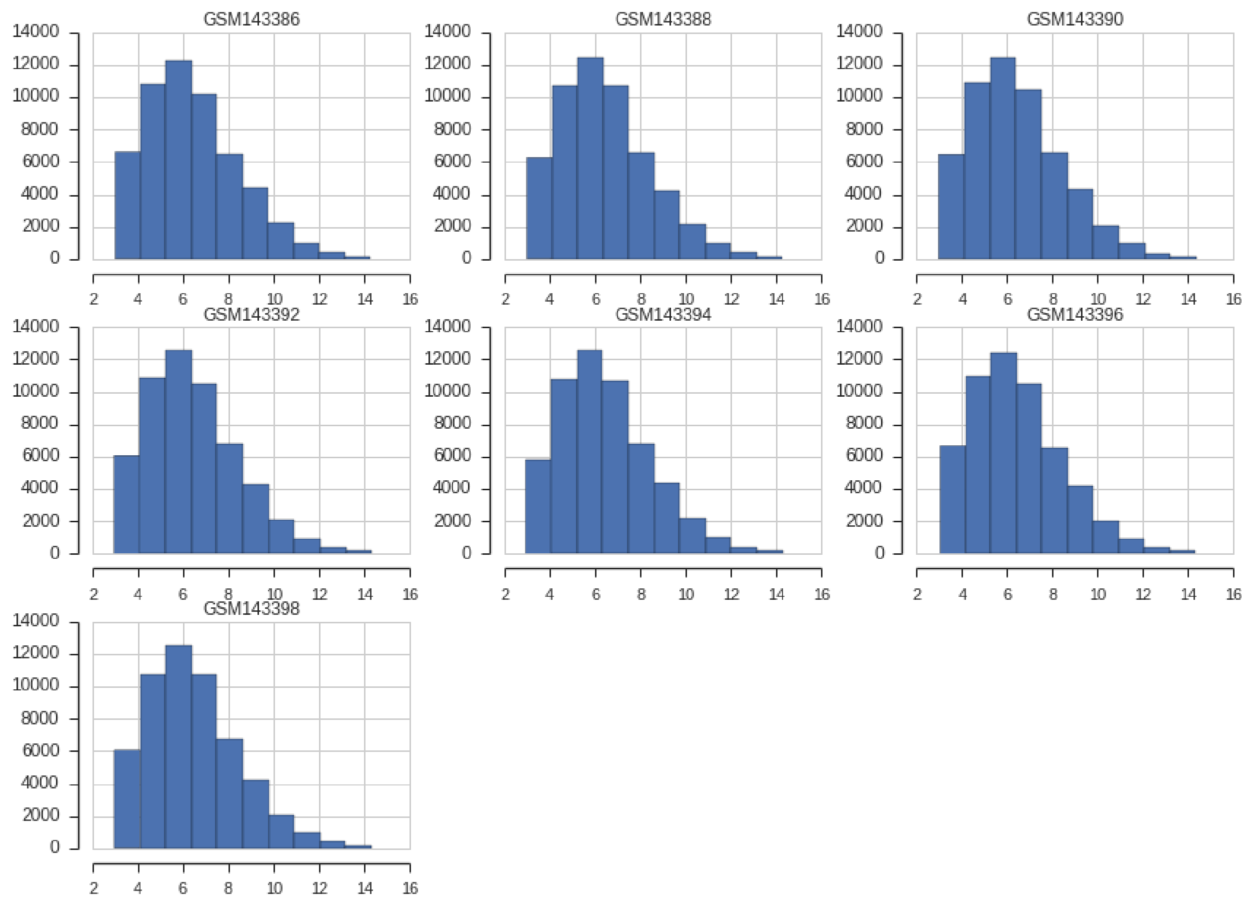
```
gse.gsms["GSM143385"].columns
```

We take the opportunity and check if everything is OK with the control samples. For this we just use simple histogram. To obtain table with each GSM as column, ID_REF as index and VALUE in each cell we use pivot_samples method from GSE object (we restrict the columns to the controls):

```
pivoted_control_samples = gse.pivot_samples('VALUE')[controls]
pivoted_control_samples.head()
```

And we plot:

```
pivoted_control_samples.hist()
sns.despine(offset=10, trim=True)
```



Next we would like to filter out probes that are not expressed. The gene is expressed (in definition here) when its average log2 intensity in control samples is above 0.25 quantile. I.e. we filter out worst 25% genes.

```
pivoted_control_samples_average = pivoted_control_samples.median(axis=1)
print("Number of probes before filtering: ", len(pivoted_control_samples_average))
```

```
Number of probes before filtering:  54675
```

```
expression_threshold = pivoted_control_samples_average.quantile(0.25)
```

```
expressed_probes = pivoted_control_samples_average[pivoted_control_samples_average >=␣
↪expression_threshold].index.tolist()
print("Number of probes above threshold: ", len(expressed_probes))
```

```
Number of probes above threshold:  41006
```

We can see that the filtering succeeded. Now we can pivot all the samples and filter out probes that are not expressed:

```
samples = gse.pivot_samples("VALUE").ix[expressed_probes]
```

The most important thing is to calculate log fold changes. What we have to do is for each time-point identify control and transfected sample and subtract the VALUES (they are provided as log2 transformed already, we subtract transfection from the control).

In order to identify control and transfection samples we will take a look into phenotype data and based on it we decide how to split samples:

```
print gse.phenotype_data[["title", "source_name_ch1"]]
```

```
                                                  title   source_name_ch1
GSM143385              miR-124 transfection for 4 hours  HepG2 cell line
GSM143386     negative control transfection for 4 hours  HepG2 cell line
GSM143387              miR-124 transfection for 8 hours  HepG2 cell line
GSM143388     negative control transfection for 8 hours  HepG2 cell line
GSM143389             miR-124 transfection for 16 hours  HepG2 cell line
GSM143390    negative control transfection for 16 hours  HepG2 cell line
GSM143391             miR-124 transfection for 24 hours  HepG2 cell line
GSM143392    negative control transfection for 24 hours  HepG2 cell line
GSM143393             miR-124 transfection for 32 hours  HepG2 cell line
GSM143394    negative control transfection for 32 hours  HepG2 cell line
GSM143395             miR-124 transfection for 72 hours  HepG2 cell line
GSM143396    negative control transfection for 72 hours  HepG2 cell line
GSM143397            miR-124 transfection for 120 hours  HepG2 cell line
GSM143398   negative control transfection for 120 hours  HepG2 cell line
```

We can see that based on the title of the experiment we can get all the information that we need:

```
experiments = {}
for i, (idx, row) in enumerate(gse.phenotype_data.iterrows()):
    tmp = {}
    tmp["Experiment"] = idx
    tmp["Type"] = "control" if "control" in row["title"] else "transfection"
    tmp["Time"] = re.search(r"for (\d+ hours)", row["title"]).group(1)
    experiments[i] = tmp
experiments = pd.DataFrame(experiments).T
print experiments
```

```
   Experiment       Time          Type
0   GSM143385    4 hours  transfection
1   GSM143386    4 hours       control
2   GSM143387    8 hours  transfection
3   GSM143388    8 hours       control
4   GSM143389   16 hours  transfection
5   GSM143390   16 hours       control
6   GSM143391   24 hours  transfection
7   GSM143392   24 hours       control
8   GSM143393   32 hours  transfection
9   GSM143394   32 hours       control
10  GSM143395   72 hours  transfection
11  GSM143396   72 hours       control
12  GSM143397  120 hours  transfection
13  GSM143398  120 hours       control
```

In the end we create new DataFrame with LFCs:

```
lfc_results = {}
sequence = ['4 hours',
             '8 hours',
             '16 hours',
             '24 hours',
             '32 hours',
             '72 hours',
             '120 hours']
for time, group in experiments.groupby("Time"):
    print(time)
    control_name = group[group.Type == "control"].Experiment.iloc[0]
    transfection_name = group[group.Type == "transfection"].Experiment.iloc[0]
    lfc_results[time] = (samples[transfection_name] - samples[control_name]).to_dict()
lfc_results = pd.DataFrame(lfc_results)[sequence]
```

```
120 hours
16 hours
24 hours
32 hours
4 hours
72 hours
8 hours
```

Let's look at the data sorted by 24-hours time-point:

```
lfc_results.sort("24 hours").head()
```

We are interested in the gene expression changes upon transfection. Thus, we have to annotate each probe with ENTREZ gene ID, remove probes without ENTREZ or with multiple assignments. Although this strategy might not be optimal, after this we average the LFC for each gene over probes.

```
# annotate with GPL
lfc_result_annotated = lfc_results.reset_index().merge(gse.gpls['GPL570'].table[["ID",
↪ "ENTREZ_GENE_ID"]],
                                left_on='index', right_on="ID").set_index('index')
del lfc_result_annotated["ID"]
# remove probes without ENTREZ
lfc_result_annotated = lfc_result_annotated.dropna(subset=["ENTREZ_GENE_ID"])
# remove probes with more than one gene assigned
lfc_result_annotated = lfc_result_annotated[~lfc_result_annotated.ENTREZ_GENE_ID.str.
↪contains("///")]
# for each gene average LFC over probes
lfc_result_annotated = lfc_result_annotated.groupby("ENTREZ_GENE_ID").median()
```

We can now look at the data:

```
lfc_result_annotated.sort("24 hours").head()
```

At that point our job is basicaly done. However, we might want to check if the experiments worked out at all. To do this we will use hsa-miR-124a-3p targets predicted by MIRZA-G algorithm. The targets should be downregulated. First we read MIRZA-G results:

```
header = ["GeneID", "miRNA", "Total score without conservation", "Total score with␣
↪conservation"]
miR124_targets = pd.read_table("seed-mirza-g_all_mirnas_per_gene_scores_miR_124a.tab",
↪ names=header)
```

```
miR124_targets.head()
```

We shall extract targets as a simple list of strings:

```
miR124_targets_list = map(str, miR124_targets.GeneID.tolist())
print("Number of targets:", len(miR124_targets_list))
```

```
Number of targets: 2311
```

As can be seen there is a lot of targets (genes that posses a seed match in their 3'UTRs). We will use all of them. As first stem we will annotate genes if they are targets or not and add this information as a column to DataFrame:

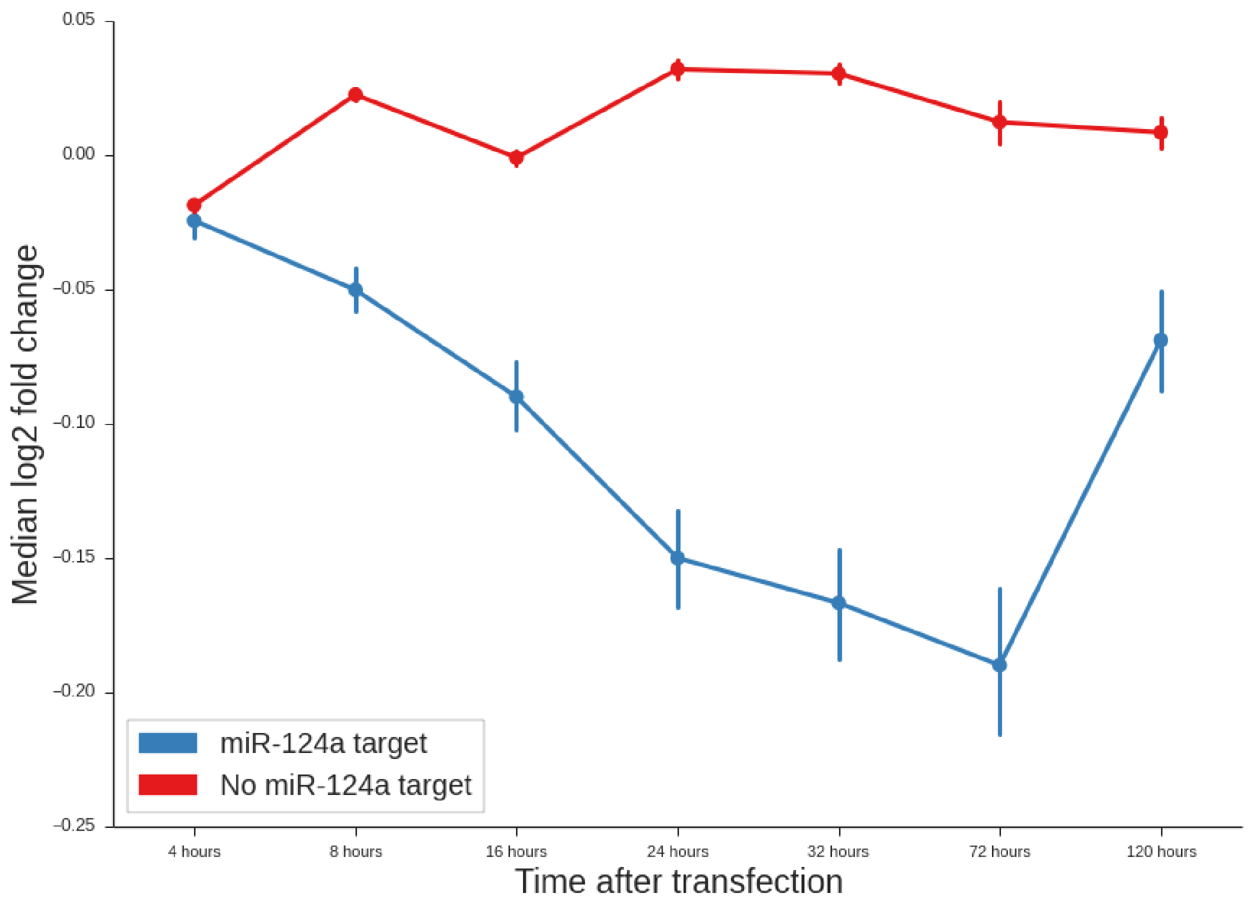
```
lfc_result_annotated["Is miR-124a target"] = [i in miR124_targets_list for i in lfc_
↪result_annotated.index]
```

```
cols_to_plot = [i for i in lfc_result_annotated.columns if "hour" in i]
```

In the end we can plot the results:

```
a = sns.pointplot(data=lfc_result_annotated[lfc_result_annotated["Is miR-124a target
↪"]][cols_to_plot],
                   color=c2,
                   label="miR-124a target")
b = sns.pointplot(data=lfc_result_annotated[~lfc_result_annotated["Is miR-124a target
↪"]][cols_to_plot],
             color=c1,
             label="No miR-124a target")
sns.despine()
pl.legend([pl.mpl.patches.Patch(color=c2), pl.mpl.patches.Patch(color=c1)],
          ["miR-124a target", "No miR-124a target"], frameon=True, loc='lower left')
pl.xlabel("Time after transfection")
pl.ylabel("Median log2 fold change")
```

```
<matplotlib.text.Text at 0x7fe66c094410>
```



As can be seen the targets of hsa-miR-124a-3p behaves in the expected way. With each time-point their downregulation is stronger up the 72 hours. After 120 hours the transfection is probably lost. This means that the experiments worked out.

CHAPTER 4

GEOparse package

Submodules

GEOparse.GEOTypes module

Classes that represent different GEO entities

class `GEOparse.GEOTypes.`**`BaseGEO`**`(`*name*, *metadata*`)`

Bases: `object`

Initialize base GEO object.

Parameters

- **`name`** (`str`) – Name of the object.
- **`metadata`** (`dict`) – Metadata information.

Raises `TypeError` – Metadata should be a dict.

`geotype` **= None**

`get_accession``()`

Return accession ID of the sample.

Returns GEO accession ID

Return type `str`

`get_metadata_attribute``(`*metaname*`)`

Get the metadata attribute by the name.

Parameters **`metaname`** (`str`) – Name of the attribute

Returns

Value(s) of the requested metadata attribute

Return type `list` or `str`

Raises

- `NoMetadataException` – Attribute error
- `TypeError` – Metadata should be a list

`get_type``()`
Get the type of the GEO.

Returns Type attribute of the GEO

Return type `str`

`show_metadata``()`
Print metadata in SOFT format.

`to_soft``(`*path_or_handle*, *as_gzip=False*`)`
Save the object in a SOFT format.

Parameters

- **`path_or_handle`** (`str` or `file`) – Path or handle to output file
- **`as_gzip`** (`bool`) – Save as gzip

exception `GEOparse.GEOTypes.`**`DataIncompatibilityException`**
Bases: `exceptions.Exception`

class `GEOparse.GEOTypes.`**`GDS`**`(`*name*, *metadata*, *table*, *columns*, *subsets*, *database=None*`)`
Bases: `GEOparse.GEOTypes.SimpleGEO`

Class that represents a dataset from GEO database

Initialize GDS

Parameters

- **`name`** (`str`) – Name of the object.
- **`metadata`** (`dict`) – Metadata information.
- **`table`** (`pandas.DataFrame`) – Table with the data from SOFT file.
- **`columns`** (`pandas.DataFrame`) – description of the columns, number of columns, order, and names represented as index in this DataFrame has to be the same as table.columns.
- **`subsets`** (`dict` of `GEOparse.GDSSubset`) – GDSSubset from GDS soft file.
- **`database`** (`GEOparse.Database`, optional) – Database from SOFT file. Defaults to None.

`geotype` **= 'DATASET'**

class `GEOparse.GEOTypes.`**`GDSSubset`**`(`*name*, *metadata*`)`
Bases: `GEOparse.GEOTypes.BaseGEO`

Class that represents a subset from GEO GDS object.

Initialize base GEO object.

Parameters

- **`name`** (`str`) – Name of the object.
- **`metadata`** (`dict`) – Metadata information.

Raises `TypeError` – Metadata should be a dict.

`geotype` **= 'SUBSET'**

class `GEOparse.GEOTypes.`**`GEODatabase`**(*name*, *metadata*)

Bases: *`GEOparse.GEOTypes.BaseGEO`*

Class that represents a subset from GEO GDS object.

Initialize base GEO object.

Parameters

- **`name`** (`str`) – Name of the object.
- **`metadata`** (`dict`) – Metadata information.

Raises `TypeError` – Metadata should be a dict.

`geotype` = **'DATABASE'**

class `GEOparse.GEOTypes.`**`GPL`**(*name*, *metadata*, *table=None*, *columns=None*, *gses=None*, *gsms=None*, *database=None*)

Bases: *`GEOparse.GEOTypes.SimpleGEO`*

Class that represents platform from GEO database

Initialize GPL.

Parameters

- **`name`** (`str`) – Name of the object
- **`metadata`** (`dict`) – Metadata information
- **`table`** (`pandas.DataFrame`, optional) – Table with actual GPL data
- **`columns`** (`pandas.DataFrame`, optional) – Table with description of the columns. Defaults to None.
- **`gses`** (`dict` of `GEOparse.GSE`, optional) – A dictionary of GSE objects. Defaults to None.
- **`gsms`** (`dict` of `GEOparse.GSM`, optional) – A dictionary of GSM objects. Defaults to None.
- **`database`** (`GEOparse.GEODatabase`, optional) – A database object from SOFT file associated with GPL. Defaults to None.

`geotype` = **'PLATFORM'**

class `GEOparse.GEOTypes.`**`GSE`**(*name*, *metadata*, *gpls=None*, *gsms=None*, *database=None*)

Bases: *`GEOparse.GEOTypes.BaseGEO`*

Class representing GEO series

Initialize GSE.

Parameters

- **`name`** (`str`) – Name of the object.
- **`metadata`** (`dict`) – Metadata information.
- **`gpls`** (`dict` of `GEOparse.GPL`, optional) – A dictionary of GSE objects. Defaults to None.
- **`gsms`** (`dict` of `GEOparse.GSM`, optional) – A dictionary of GSM objects. Defaults to None.
- **`database`** (`GEOparse.Database`, optional) – Database from SOFT file. Defaults to None.

download_SRA(*email*, *directory='series'*, *filterby=None*, ***kwargs*)
Download SRA files for each GSM in series.

Parameters

- **email** (str) – E-mail that will be provided to the Entrez.
- **directory** (str, optional) – Directory to save the data (defaults to the 'series' which saves the data to the directory with the name of the series + '_SRA' ending). Defaults to "series".
- **filterby** (str, optional) – Filter GSM objects, argument is a function that operates on GSM object and return bool eg. lambda x: "brain" not in x.name. Defaults to None.
- ****kwargs** – Any arbitrary argument passed to GSM.download_SRA method. See the documentation for more details.

download_supplementary_files(*directory='series'*, *download_sra=True*, *sra_filetype='fasta'*, *email=None*, *sra_kwargs=None*)
Download supplementary data.

Parameters

- **directory** (str, optional) – Directory to download the data (in this directory function will create new directory with the files), by default this will be named with the series name + _Supp.
- **download_sra** (bool, optional) – Indicates whether to download SRA raw data too. Defaults to True.
- **sra_filetype** (str, optional) – Indicates what file type to download if we specified SRA, can be sra, fasta or fastq. Defaults to "fasta".
- **email** (str, optional) – E-mail that will be provided to the Entrez. Defaults to None.
- **sra_kwargs** (dict, optional) – Kwargs passed to the GSM.download_SRA method. Defaults to None.

Returns Downloaded paths for each of the GSM

Return type dict

geotype = 'SERIES'

merge_and_average(*platform*, *expression_column*, *group_by_column*, *force=False*, *merge_on_column=None*, *gsm_on=None*, *gpl_on=None*)
Merge and average GSE samples.

For given platform prepare the DataFrame with all the samples present in the GSE annotated with given column from platform and averaged over the column.

Parameters

- **platform** (str or GEOparse.GPL) – GPL platform to use.
- **expression_column** (str) – Column name in which "expressions" are represented
- **group_by_column** (str) – The data will be grouped and averaged over this column and only this column will be kept
- **force** (bool) – If the name of the GPL does not match the platform name in GSM proceed anyway
- **merge_on_column** (str) – Column to merge the data on - should be present in both GSM and GPL

- **`gsm_on`** (`str`) – In the case columns to merge are different in GSM and GPL use this column in GSM
- **`gpl_on`** (`str`) – In the case columns to merge are different in GSM and GPL use this column in GPL

Returns Merged and averaged table of results.

Return type `pandas.DataFrame`

`phenotype_data`
Get the phenotype data for each of the sample.

`pivot_and_annotate`(*values*, *gpl*, *annotation_column*, *gpl_on='ID'*, *gsm_on='ID_REF'*)
Annotate GSM with provided GPL.

Parameters

- **`values`** (`str`) – Column to use as values eg. "VALUES"
- **`gpl`** (`pandas.DataFrame` or `GEOparse.GPL`) – A Platform or DataFrame to annotate with.
- **`annotation_column`** (`str`) – Column in table for annotation.
- **`gpl_on`** (`str`, optional) – Use this column in GPL to merge. Defaults to "ID".
- **`gsm_on`** (`str`, optional) – Use this column in GSM to merge. Defaults to "ID_REF".

Returns Pivoted and annotated table of results

Return type pandas.DataFrame

`pivot_samples`(*values*, *index='ID_REF'*)
Pivot samples by specified column.

Construct a table in which columns (names) are the samples, index is a specified column eg. ID_REF and values in the columns are of one specified type.

Parameters

- **`values`** (`str`) – Column name present in all GSMs.
- **`index`** (`str`, optional) – Column name that will become an index in pivoted table. Defaults to "ID_REF".

Returns Pivoted data

Return type `pandas.DataFrame`

class `GEOparse.GEOTypes.`**`GSM`**(*name*, *metadata*, *table*, *columns*)
Bases: *`GEOparse.GEOTypes.SimpleGEO`*

Class that represents sample from GEO database.

Initialize simple GEO object.

Parameters

- **`name`** (`str`) – Name of the object
- **`metadata`** (`dict`) – Metadata information
- **`table`** (`pandas.DataFrame`) – Table with the data from SOFT file
- **`columns`** (`pandas.DataFrame`) – Description of the columns, number of columns, order and names represented as index in this DataFrame has to be the same as table.columns.

Raises

- `ValueError` – Table should be a DataFrame
- `ValueError` – Columns' description should be a DataFrame
- *`DataIncompatibilityException`* – Columns are wrong
- `ValueError` – Description has to be present in columns

`annotate`(*gpl*, *annotation_column*, *gpl_on='ID'*, *gsm_on='ID_REF'*, *in_place=False*)
Annotate GSM with provided GPL

Parameters

- **`gpl`** (`pandas.DataFrame`) – A Platform or DataFrame to annotate with
- **`annotation_column`** (*`str``*) – Column in a table for annotation
- **`gpl_on`** (`str`) – Use this column in GSM to merge. Defaults to "ID".
- **`gsm_on`** (`str`) – Use this column in GPL to merge. Defaults to "ID_REF".
- **`in_place`** (`bool`) – Substitute table in GSM by new annotated table. Defaults to False.

Returns Annotated table or None

Return type `pandas.DataFrame` or `None`

Raises `TypeError` – GPL should be GPL or pandas.DataFrame

`annotate_and_average`(*gpl*, *expression_column*, *group_by_column*, *rename=True*, *force=False*, *merge_on_column=None*, *gsm_on=None*, *gpl_on=None*)
Annotate GSM table with provided GPL.

Parameters

- **`gpl`** (`GEOTypes.GPL`) – Platform for annotations
- **`expression_column`** (`str`) – Column name which "expressions" are represented
- **`group_by_column`** (`str`) – The data will be grouped and averaged over this column and only this column will be kept
- **`rename`** (`bool`) – Rename output column to the self.name. Defaults to True.
- **`force`** (`bool`) – If the name of the GPL does not match the platform name in GSM proceed anyway. Defaults to False.
- **`merge_on_column`** (`str`) – Column to merge the data on. Defaults to None.
- **`gsm_on`** (`str`) – In the case columns to merge are different in GSM and GPL use this column in GSM. Defaults to None.
- **`gpl_on`** (`str`) – In the case columns to merge are different in GSM and GPL use this column in GPL. Defaults to None.

Returns Annotated data

Return type `pandas.DataFrame`

`download_SRA`(*email*, *directory='./'*, ***kwargs*)
Download RAW data as SRA file.

The files will be downloaded to the sample directory created ad hoc or the directory specified by the parameter. The sample has to come from sequencing eg. mRNA-seq, CLIP etc.

An important parameter is a download_type. By default an SRA is accessed by FTP and such file is downloaded. This does not require additional libraries. However in order to produce FASTA of FASTQ files one would need to use SRA-Toolkit. Thus, it is assumed that this library is already installed or it will be installed in the near future. One can immediately specify the download type to fasta or fastq.

Following `**kwargs` can be passed:

- **filetype - str** can be sra, fasta, or fastq - for fasta or fastq SRA-Toolkit need to be installed
- **aspera - bool** use Aspera to download samples, defaults to False
- **keep_sra - bool** keep SRA files after download. Removes SRA file only if the selected file type is different than "sra", defaults to False
- **fastq_dump_options - dict** pass options to fastq-dump (if used, the options has to be in long form eg. –split-files), defaults to:

```
{
    'split-files': None,
    'readids': None,
    'read-filter': 'pass',
    'dumpbase': None,
    'gzip': None
}
```

Parameters

- **`email`** (`str`) – an email (any) - Required by NCBI for access
- **`directory`** (`str`, optional) – The directory to which download the data. Defaults to "./".
- **`**kwargs`** – Arbitrary keyword arguments, see description

Returns List of downloaded files.

Return type `list` of `str`

Raises

- `TypeError` – Type to download unknown
- *`NoSRARelationException`* – No SRAToolkit
- `Exception` – Wrong e-mail
- `HTTPError` – Cannot access or connect to DB

`download_supplementary_files`(*directory='./'*, *download_sra=True*, *email=None*, *sra_kwargs=None*)

Download all supplementary data available for the sample.

Parameters

- **`directory`** (`str`) – Directory to download the data (in this directory function will create new directory with the files). Defaults to "./".
- **`download_sra`** (`bool`) – Indicates whether to download SRA raw data too. Defaults to True.
- **`email`** (`str`) – E-mail that will be provided to the Entrez. It is mandatory if download_sra=True. Defaults to None.
- **`sra_kwargs`** (`dict`, optional) – Kwargs passed to the download_SRA method. Defaults to None.

Returns A key-value pair of name and paths downloaded

Return type `dict`

`geotype` = 'SAMPLE'

exception `GEOparse.GEOTypes.`**`NoMetadataException`**
Bases: `exceptions.Exception`

exception `GEOparse.GEOTypes.`**`NoSRARelationException`**
Bases: `exceptions.Exception`

exception `GEOparse.GEOTypes.`**`NoSRAToolkitException`**
Bases: `exceptions.Exception`

class `GEOparse.GEOTypes.`**`SimpleGEO`**(*name*, *metadata*, *table*, *columns*)
Bases: *`GEOparse.GEOTypes.BaseGEO`*

Initialize simple GEO object.

Parameters

- **`name`** (`str`) – Name of the object
- **`metadata`** (`dict`) – Metadata information
- **`table`** (`pandas.DataFrame`) – Table with the data from SOFT file
- **`columns`** (`pandas.DataFrame`) – Description of the columns, number of columns, order and names represented as index in this DataFrame has to be the same as table.columns.

Raises

- `ValueError` – Table should be a DataFrame
- `ValueError` – Columns' description should be a DataFrame
- *`DataIncompatibilityException`* – Columns are wrong
- `ValueError` – Description has to be present in columns

`head`()
Print short description of the object.

`show_columns`()
Print columns in SOFT format.

`show_table`(*number_of_lines=5*)
Show few lines of the table the table as pandas.DataFrame.

Parameters **`number_of_lines`** (`int`) – Number of lines to show. Defaults to 5.

GEOparse.GEOparse module

exception `GEOparse.GEOparse.`**`NoEntriesException`**
Bases: `exceptions.Exception`

Raised when no entries could be found in the SOFT file.

exception `GEOparse.GEOparse.`**`UnknownGEOTypeException`**
Bases: `exceptions.Exception`

Raised when the GEO type that do not correspond to any known.

`GEOparse.GEOparse.`**`get_GEO`**(*geo=None*, *filepath=None*, *destdir='./'*, *how='full'*, *annotate_gpl=False*, *geotype=None*, *include_data=False*, *silent=False*, *aspera=False*)

Get the GEO entry.

The GEO entry is taken directly from the GEO database or read it from SOFT file.

Parameters

- **`geo`** (`str`) – GEO database identifier.
- **`filepath`** (`str`) – Path to local SOFT file. Defaults to None.
- **`destdir`** (`str`, optional) – Directory to download data. Defaults to None.
- **`how`** (`str`, optional) – GSM download mode. Defaults to "full".
- **`annotate_gpl`** (`bool`, optional) – Annotate GPL object. Defaults to False.
- **`geotype`** (`str`, optional) – Type of GEO entry. By default it is inferred from the ID or the file name.
- **`include_data`** (`bool`, optional) – Full download of GPLs including series and samples. Defaults to False.
- **`silent`** (`bool`, optional) – Do not print anything. Defaults to False.
- **`aspera`** (`bool`, optional) – EXPERIMENTAL Download using Aspera Connect. Follow Aspera instructions for further details. Defaults to False.

Returns A GEO object of given type.

Return type `GEOparse.BaseGEO`

`GEOparse.GEOparse.`**`get_GEO_file`**(*geo*, *destdir=None*, *annotate_gpl=False*, *how='full'*, *include_data=False*, *silent=False*, *aspera=False*)

Download corresponding SOFT file given GEO accession.

Parameters

- **`geo`** (`str`) – GEO database identifier.
- **`destdir`** (`str`, optional) – Directory to download data. Defaults to None.
- **`annotate_gpl`** (`bool`, optional) – Annotate GPL object. Defaults to False.
- **`how`** (`str`, optional) – GSM download mode. Defaults to "full".
- **`include_data`** (`bool`, optional) – Full download of GPLs including series and samples. Defaults to False.
- **`silent`** (`bool`, optional) – Do not print anything. Defaults to False.
- **`aspera`** (`bool`, optional) – EXPERIMENTAL Download using Aspera Connect. Follow Aspera instructions for further details. Defaults to False.

Returns Path to downloaded file and and the type of GEO object.

Return type `2-tuple` of `str` and `str`

`GEOparse.GEOparse.`**`parse_GDS`**(*filepath*)

Parse GDS SOFT file.

Parameters **`filepath`** (`str`) – Path to GDS SOFT file.

Returns A GDS object.

Return type `GEOparse.GDS`

`GEOparse.GEOparse.`**`parse_GDS_columns`**`(`*lines*, *subsets*`)`
Parse list of line with columns description from SOFT file of GDS.

Parameters

- **`lines`** (`Iterable`) – Iterator over the lines.
- **`subsets`** (`dict` of `GEOparse.GDSSubset`) – Subsets to use.

Returns Columns description.

Return type `pandas.DataFrame`

`GEOparse.GEOparse.`**`parse_GPL`**`(`*filepath*, *entry_name=None*`)`
Parse GPL entry from SOFT file.

Parameters

- **`filepath`** (`str` or `Iterable`) – Path to file with 1 GPL entry or list of lines representing GPL from GSE file.
- **`entry_name`** (`str`, optional) – Name of the entry. By default it is inferred from the data.

Returns A GPL object.

Return type `GEOparse.GPL`

`GEOparse.GEOparse.`**`parse_GSE`**`(`*filepath*`)`
Parse GSE SOFT file.

Parameters **`filepath`** (`str`) – Path to GSE SOFT file.

Returns A GSE object.

Return type `GEOparse.GSE`

`GEOparse.GEOparse.`**`parse_GSM`**`(`*filepath*, *entry_name=None*`)`
Parse GSM entry from SOFT file.

Parameters

- **`filepath`** (`str` or `Iterable`) – Path to file with 1 GSM entry or list of lines representing GSM from GSE file.
- **`entry_name`** (`str`, optional) – Name of the entry. By default it is inferred from the data.

Returns A GSM object.

Return type `GEOparse.GSM`

`GEOparse.GEOparse.`**`parse_columns`**`(`*lines*`)`
Parse list of lines with columns description from SOFT file.

Parameters **`lines`** (`Iterable`) – Iterator over the lines.

Returns Columns description.

Return type `pandas.DataFrame`

`GEOparse.GEOparse.`**`parse_entry_name`**`(`*nameline*`)`
Parse line that starts with ^ and assign the name to it.

Parameters **`nameline`** (`str`) – A line to process.

Returns Entry name.

Return type `str`

`GEOparse.GEOparse.`**`parse_metadata`**(*lines*)

Parse list of lines with metadata information from SOFT file.

Parameters **`lines`** (`Iterable`) – Iterator over the lines.

Returns Metadata from SOFT file.

Return type `dict`

`GEOparse.GEOparse.`**`parse_table_data`**(*lines*)

"Parse list of lines from SOFT file into DataFrame.

Parameters **`lines`** (`Iterable`) – Iterator over the lines.

Returns Table data.

Return type `pandas.DataFrame`

GEOparse.logger module

`GEOparse.logger.`**`set_verbosity`**(*level*)

Set the log level.

Parameters **`level`** (`str`) – Level name eg. DEBUG or ERROR

`GEOparse.logger.`**`add_log_file`**(*path*)

Add log file.

Parameters **`path`** (`str`) – Path to the log file.

GEOparse.utils module

`GEOparse.utils.`**`download_aspera`**(*url*, *dest_path*, *user='anonftp'*, *ftp='ftp-trace.ncbi.nlm.nih.gov'*)

Download file with Aspera Connect.

For details see the documentation ov Aspera Connect

Parameters

- **`url`** (`str`) – URL to the file
- **`dest_path`** (`str`) – Destination path.
- **`user`** (`str`, optional) – User. Defaults to anonftp.
- **`ftp`** (`str`, optional) – FTP path. Defaults to "ftp-trace.ncbi.nlm.nih.gov".

`GEOparse.utils.`**`download_from_url`**(*url*, *destination_path*, *force=False*, *aspera=False*, *silent=False*)

Download file from remote server

Parameters

- **`url`** (`str`) – Path to the file on remote server (including file name)
- **`destination_path`** (`str`) – Path to the file on local machine (including file name)
- **`force`** (`bool`) – If file exist force to overwrite it. Defaults to False.
- **`aspera`** (`bool`) – Download with Aspera Connect. Defaults to False.

- **`silent`** (`bool`) – Do not print any message. Defaults to False.

`GEOparse.utils.`**`mkdir_p`**(*path_to_dir*)

Make directory(ies).

This function behaves like mkdir -p.

Parameters **`path_to_dir`** (`str`) – Path to the directory to make.

`GEOparse.utils.`**`smart_open`**(**args*, ***kwds*)

Open file intelligently depending on the source and python version.

Parameters **`filepath`** (`str`) – Path to the file.

Yields Context manager for file handle.

`GEOparse.utils.`**`which`**(*program*)

Check if executable exists.

The code is taken from: https://stackoverflow.com/questions/377017/test-if-executable-exists-in-python :param program: Path to the executable. :type program: `str`

Returns Path to the program or None.

Return type `str` or `None`

Module contents

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/guma44/GEOparse/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

GEOparse could always use more documentation, whether as part of the official GEOparse docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/guma44/GEOparse/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *GEOparse* for local development.

1. Fork the *GEOparse* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/GEOparse.git
   $ git checkout -b develop origin/develop
   ```

   Develop branch should be the start of every feature.
3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv GEOparse
   $ cd GEOparse/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.
5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ python setup.py test
   $ tox
   ```

   To get tox, just pip install them into your virtualenv.
6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

CHAPTER 6

Credits

Development Lead

- Rafal Gumienny <guma44@gmail.com>, <r.gumienny@unibas.ch>

Contributors

- Simon van Heeringen <simon.vanheeringen@gmail.com>
- Tycho Bismeijer <tychobismeijer@gmail.com>
- Haries Ramdhani
- KwatME

CHAPTER 7

History

CHAPTER 8

1.0.0 (2017-07-21)

- Many small bug fixes: * unknown subset types added to columns * silent=True is really silent * correct treatment of duplicated columns * illegal file names and no filtering of user input from GEO to create the file names * platform was not imported but used * fixed issues of python 2 and 3 compatibility
- Logging replaced stdout and stderr + ability to set verbosity and log file
- Return downloaded paths from download functions
- Updated documentation according to Google docstring style guide
- Tests update
- Code refactored to be more PEP-8 friendly

CHAPTER 9

0.1.10 (2017-03-27)

- Important fix for SRA download
- Fix duplicated columns issue
- Python 2 and 3 open compatibility

CHAPTER 10

0.1.9 (2017-03-10)

- Added property phenotype_data to access phenotype data of GSE
- Fixed windows issue with file names
- replaced default download function with wgetter
- Update documentation
- Various bugfixes

CHAPTER 11

0.1.8 (2016-11-02)

Thanks to Tycho Bismeijer:

- Python 3 Compatibility
- Bio.Entrez dependency optional

CHAPTER 12

0.1.7 (2016-05-30)

Thanks to Simon van Heeringen:

- bugfix in datasets with multiple associated relations
- –split-files to fastq-dump to support paired-end experiments by default
- parse a GPL that also contains series and sample information
- gsm2fastq command to make download easier
- initial Aspera download support

CHAPTER 13

0.1.6 (2016-04-12)

- Bugfixes
- SRA function of GSE can now filter GSMs

CHAPTER 14

0.1.5 (2016-02-03)

- Added functions to download supplementary files including raw files from SRA

CHAPTER 15

0.1.4 (2015-09-27)

- Updated documentation including example
- Updated tests: they now cover 80% of library with all important functions
- Added pivot_and_annotate method to GSE object
- Bugfixes

CHAPTER 16

0.1.3 (2015-08-30)

- Updated documentation
- Added pivot_samples to GSE object
- Code of GEOTypes was refactored
- All objects now have to_soft function
- Various bugfixes

CHAPTER 17

0.1.2 (2015-08-23)

- Added GDS support
- Added to_soft methods to GSE, GSM and GPL
- Added DATABASE entry support to GSE and GDS

CHAPTER 18

0.1.1 (2015-08-16)

- Brown-Bag release

CHAPTER 19

0.1.0 (2015-08-16)

- First release on PyPI.

CHAPTER 20

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index