leiden Documentation

Release 1.0.3

Andrew Hill

May 15, 2014
These are the documentation pages for Andrew Hill’s project to clean up and validate variants from Leiden Open Variation Database (LOVD) installations - a set of databases housing voluntarily curated mutations associated with specific disease areas as muscular dystrophy.

For more information, see Overview.

**Warning:** The contents of LOVD databases are the intellectual property of the respective curator(s). Any unauthorised use, copying, storage or distribution of this material without written permission from the curator(s) will lead to copyright infringement with possible ensuing litigation. Copyright © 2013-2014. All Rights Reserved. For further details, refer to Directive 96/9/EC of the European Parliament and the Council of March 11 (1996) on the legal protection of databases.

We have used all reasonable efforts to ensure that the information displayed on these pages and contained in the databases is of high quality. We make no warranty, express or implied, as to its accuracy or that the information is fit for a particular purpose, and will not be held responsible for any consequences arising out of any inaccuracies or omissions. Individuals, organisations and companies which use this database do so on the understanding that no liability whatsoever either direct or indirect shall rest upon the curator(s) or any of their employees or agents for the effects of any product, process or method that may be produced or adopted by any part, notwithstanding that the formulation of such product, process or method may be based upon information here provided.

Documentation Contents:
1.1 About

The Leiden Open Variation Database platform is a popular genetic variant database platform whose installations are home to many voluntarily curated mutations implicated in a variety of disease areas.

- A list of all current installations: http://www.lovd.nl/2.0/index_list.php
- LOVD platform: http://www.lovd.nl/3.0/home

Unfortunately these variants are in HGVS format (popular in clinical settings) and in coordinates relative to specific cDNA transcripts, which makes further analysis difficult informatically. Perhaps more concerning is that the standard for submission of disease causing mutations has become much stricter in the time since LOVDs inception. This implies that there are many false positives within this data set. Curation of these databases is completely voluntary, making many variants completely unreliable or unusable. Despite these challenges, there are likely many true positives amongst the noise, many of which may not be in other variant databases. Locating and reporting these true positives is an important goal for the research community.

While LOVD is public access and has provided reST APIs for querying for variants at specific genomic positions or retrieving some information about LOVD variants in specific genes, none of the available services allow the actual variant descriptions (or other submitted information) to be downloaded. This package fills that gap and also facilitates some degree of data validation.

The goals of this project are to provide tools for:

- Extracting variants from these databases
- Remapping these variants to VCF format
- Cross-checking of information about these variants to infer concordance of submissions

Important: Actually implicating variants as being pathogenic requires thorough manual curation by examining the full set of information (including, but not limited to publication references) for validated variants. “Validation” as described here simply implies correctness and consistency of submitted variants, it does not prove true positive implication in any disease.
1.2 General Workflow

In general, the recommended workflow facilitated by the scripts in this package is:

1. Extract raw variants from LOVD, saving one tab-delimited file per gene.
2. Annotate variants with VEP (must have VEP on path) and combine with original data in a single VCF file per gene.
3. Validate annotated variants by cross-checking submitted data with annotation and output a single VCF for all variants.

The scripts I have included (see Driver Scripts and Other Scripts) make it easy to carry out this workflow. There is no need to adhere to this set of scripts if it does not suit your needs. Custom scripts can be developed using the underlying python packages (see Modules for more info)

1.3 Project Structure

- /bin/ - contains all scripts (see Driver Scripts and Other Scripts for more info)
- /leiden/ - python packages for project (see Modules for more info)
- /docs/ - project documentation written in the sphinx framework
- /data/ - sample data formats (see Data section for more info)

Other folders are for build and distribution purposes.
CHAPTER 2

Installation and Dependencies

2.1 Installation for General Use (No Development)

While this project ideally could be installed using a default installation, some dependencies do not always install correctly.

In the next sections I will step through the installation process for the two finicky dependencies.

**Important:** Install the dependencies below first and then install leiden.

Once these dependencies are installed, you can do any of the following:

Clone:

```
git clone git@github.com:andrewhill157/leiden.git
```

Clone and Install from Source:

```
git clone git@github.com:andrewhill157/leiden.git
cd leiden
python setup.py install leiden
```

All project dependencies are listed in requirements.txt. If you simply want to install the dependencies for the project without installing the package itself run the command from the package root directory:

```
pip install -r requirements.txt
```

If you want to contribute to the project via Github, please see the *Contributing to Leiden* page.

2.1.1 pygr

pygr requires that you have bsddb3 and Berkeley DB installed. This can be installed using homebrew:

```
brew install berkeley-db --without-java
```

# Note prefix to pip not required on Broad cluster sudo BERKELEYDB_DIR=/usr/local/Cellar/berkeley-db/5.3.15/
pip install bsddb3

Once bbd3 and Berkeley DB are installed, you should be able to install pygr with pip install pygr. If there are still errors, the following may correct the problem.
export CFLAGS=-Qunused-arguments
export CPPFLAGS=-Qunused-arguments
pip install pygr

I have not encountered any additional problems installing pygr.

## 2.1.2 hgvs

HGVS must be installed by cloning the Github repository and installing from source:

```bash
git clone git@github.com:counsyl/hgvs.git
cd hgvs
python setup.py install
cd ..
rm -rf hgvs
```

I have not encountered any additional problems installing hgvs.

**Important:** Unfortunately, this tool depends on a relatively large file that I cannot easily host on Github. This is normally housed in the folder `/leiden/remapping/resources/`. It is a human genome reference sequence (hg19.fa) I have temporarily hosted a copy at at: [http://www.broadinstitute.org/~ahill](http://www.broadinstitute.org/~ahill). This file will need to be decompressed using gunzip and placed in `/leiden/remapping/resources/`. The first time this package is used, two additional files will be generated (takes some time). Subsequent runs will not require this process to be repeated.

## 2.1.3 Other Errors

I have also seen an error stating that `pg_config` executable could not be found. This seems to be an executable included with PostgreSQL, which can be downloaded with homebrew, etc.:

```bash
brew install postgresql
```

## 2.1.4 Variant Effect Predictor

Variant Effect Predictor (VEP) is required to use `generate_annotated_vcf.py` and the `annotate_vcf` module. The downstream script `validate_annotated_vcfs.py` depends on the annotations that VEP provides to perform validation. Since `run_all.py` calls these scripts, it also requires VEP. See [Variant Effect Predictor](http://www.broadinstitute.org/~ahill) for more info.

The `extract_data.py` and the other packages in `leiden` can be used without VEP.

**Important:** VEP must be installed and on your path to generate and validate annotated VCF files.

## 2.2 Development Installation

If you would like to extend or modify the existing code-base or scripts while still having the package installed, you can install in editable or development mode. This differs slightly from the default installation mode.

The easiest way to do this is to install from cloned source.

```bash
git clone git@github.com:andrewhill157/leiden.git
cd leiden
python setup.py develop
```
Either of these methods will make the leiden packages accessible by python, but allow you to edit and call the modified
source without re-installing the package. Note that the dependencies must still be installed according to instructions
in the Installation for General Use section.

If you want to contribute to the project via Github, please see the Contributing to Leiden page.
CHAPTER 3

Variant Effect Predictor

In order to run the annotation and validation software in leiden, you must have Variant Effect Predictor (VEP) installed and available on your PATH.

**Important:** Make sure variant_effect_predictor.pl is on your PATH!

Currently, the parameters for VEP are hard-coded and as follows:

```
--vcf
--cache
--fork 4
--host useastdb.ensembl.org
--format vcf
--force_overwrite
--everything
--compress "gunzip -c"
```

This means that you will also need the human cache and reference sequence available. See VEP Documentation for more info.

Future improvements may seek to allow tweaking of parameters using a config file. If you wish to modify this function to in the meantime, please ensure that --hgvs is used, as this is used for validation purposes in the scripts that are provided with this package.

### 3.1 Optimizing VEP

While not a topic specific to this package, I would like to point out a few adjustments that substantially speed up VEP annotation. These changes can make an enormous different in the runtime.

- Download relevant cache (27) and a .fasta file to use --cache
- Use --host useastdb.ensembl.org (USA users)
- use --fork 4
- Convert the cache using tabix (see Databases and Caches for more info)

Unfortunately --hgvs (required for validation) requires internet access, so some database connection must be made. Specifying the US mirror as the host (as specified above) greatly speeds up runtime because of this.

**Warning:** Failing to optimize your VEP installation will result in prohibitively long execution time.
3.2 Remapping HGVS with VEP

Note that it is possible to use HGVS formatted variants as input to VEP. At the time this package was developed, this was incredibly slow. For this reason, I have used the third-party python module (HGVS) to convert variants to VCF format prior to annotation.

This package is very fast and produced very comparable results to those of VEP and other available tools. This tool is also useful for stand-alone HGVS to VCF (or vice versa) conversion. The leiden.remapping module provides functions for remapping.
Contributing to Leiden

The project is hosted on Github and can be cloned with the following command:

```
git clone git@github.com:andrewhill1157/leiden.git
```

**Note:** Even if you do not install the package, you must install in development mode or add the leiden package to your PYTHONPATH to use leiden (see *Installation and Dependencies* page).

If you fork my project on Github, I will gladly review any pull requests. If you want your changes to be incorporated into the main project, this is the best approach. Please include a detailed description with any requests.
This project contains a number of distinct python modules.

Note that there are tests for most modules that are not simple wrappers for other scripts or libraries. These tests are included alongside modules in files named `test_<module_name>.py`. These tests are not 100% comprehensive.

Tests are compatible with the nose unit testing platform. This is an extension of the default unittest platform that makes tests very easy to develop and run. To run all unit tests for this project run the following from the root directory.

```
nosetests
```

Note that the nose python package must be installed to run tests. Tests for specific individual modules can also be run. Please see nose documentation for more information.

**Tip:** The scripts included with this package provide example usage of the functions in these modules.

### 5.1 leiden_database

These classes allow a user to extract tables of data (mutations listed for a specific gene in the database) and other useful information from any Leiden Open Variation Database (LOVD) installation, such as [http://www.dmd.nl/nmdb2/](http://www.dmd.nl/nmdb2/). Unfortunately, it has been necessary to do this by downloading the HTML for relevant pages on the database and parsing out the necessary data, as they do not provide an easy way to access the data otherwise. Note that I have chosen to use beautifulsoup4 internally for HTML parsing.

The usage for these classes is as follows:

```python
leiden_url = 'http://www.dmd.nl/nmdb2/'  # base URL of LOVD installation
gene_id = 'ACTA1'  # External name for Gene

database = make_leiden_database(leiden_url)  # factory method automatically chooses right database version
database.genes()  # get a list of available genes
database.version()  # get LOVD version

# Get data about a gene
acta1 = database.get_gene_data(gene_id)
reference_transcript = acta1.transcript_refseqid()
number_of_variants = acta1.variant_count()
column_labels = acta1.columns()  # get column labels
variants = acta1.variants()  # get table of all variants
```

Note that `make_leiden_database` returns a LeidenDatabase object. There are two subclasses of this type (one for each version of LOVD). This method ensures that the correct subclass is chosen for the provided URL.
Unit tests for this module are currently quite slow because they actually make requests for HTML data. Ideally, this would be replaced with Mock Objects where we have canned HTML responses saved on disk.

### 5.1.1 Member Descriptions

`class leiden.leiden_database.GeneData (leiden_url, gene_id)`
Should not construct directly. Use get_gene_data method of LeidenDatabase objects constructed with make_leiden_database to construct.

Class for interfacing with data provided on genes on LOVD installations.

- `columns()`
  Returns the column labels from the table of variants in the Leiden Database for the set gene.
  **Returns:**
  - list of str: column labels from the table of variants in the Leiden Database variant listing for the object’s gene_id. Returned in left to right order as they appear on the Leiden Database. Empty list returned if no labels are found.

- `transcript_refseqid()`
  Returns the transcript refSeq ID for gene (denoted by NM_<ID> on the gene homepage on the given gene_id). For example, the ACTA1 homepage is [http://www.dmd.nl/nmdb2/home.php](http://www.dmd.nl/nmdb2/home.php) and the RefSeq ID is “NM_001100.3”.
  **Returns:** string: transcript refSeqID for set gene. Returns an empty string if no refSeq ID is found.

- `variant_count()`
  Get the total number of variants for gene. This is the total number of variant entries in table of variants, not the number of unique entries.
  **Returns:** int: Number of variants listed for current gene
  **Raises:** ValueError: if the number of entries could not be found on web page

- `variants()`
  Returns the table of variants for gene.
  **Returns:**
  - list of list of str: table of variants from the gene 1st dimension is rows, 2nd is columns

`class leiden.leiden_database.LeidenDatabase (leiden_url)`
Should not construct directly. Use make_leiden_database factory method to obtain instances of LeidenDatabase objects.

Class providing functions to extract information about a variants listed under a specified gene on a specified lovd Leiden Database installation. For example, [http://www.dmd.nl/nmdb2/](http://www.dmd.nl/nmdb2/), is a particular installation for variants in genes associated with Muscular Dystrophy. A list of all known installations of lovd databases can be found at [http://www.lovd.nl/2.0/index_list.php](http://www.lovd.nl/2.0/index_list.php).

- `genes()`
  Returns a list of gene IDs available on this database.
  **Returns:** list of str: list of gene IDs available on this database.

- `get_gene_data (gene_id)`
  Returns a _GeneData object that provides a interface to data on the specified gene. Available functions allow access to variants, column labels, reference transcript, etc. See _GeneData for documentation.
  **Args:**

---

*leiden Documentation, Release 1.0.3*
gene_id (str): a string with the Gene ID of the gene of interest. For example, ACTA1 is the gene ID for actin, as specified on the Leiden Muscular Dystrophy Pages at http://www.dmd.nl/nmdb2/home.php?

Returns: _GeneData: _GeneData object for specified gene ID.

version_number()

Return version number of lovd in use for lovd.

Returns: float: version number of lovd in use for lovd

leiden.leiden_database.make_leiden_database (leiden_url)

Factory method that returns appropriate LeidenDatabase object for lovd version installed at specified URL. Only LOVD2 and LOVD3 installations are supported.

Args:

leiden_url(str): The base URL of the particular Leiden lovd to be used. For example, the Leiden muscular dystrophy pages LOVD2 homepage is http://www.dmd.nl/nmdb2/. This must be a valid URL to base page of lovd. For LOVD3 installations, such as the Genetic Eye Disorder (GEI) Variation Database, the base url will be similar to http://mseqdr.lumc.edu/GEDI/. Extensions of this URL, such as http://mseqdr.lumc.edu/GEDI/genes or http://mseqdr.lumc.edu/GEDI/variants should not be used.

gene_id (str): a string with the Gene ID of the gene of interest. For example, ACTA1 is the gene ID for actin, as specified on the Leiden Muscular Dystrophy Pages at http://www.dmd.nl/nmdb2/home.php?


Raises: Exception: If the lovd version installed at specified URL is unsupported (anything other than version 2 or 3)

5.2 utilities

These are general utility functions, some of which are used in leiden_database.py as general purpose functions.

5.2.1 Member Descriptions

leiden.utilities.correct_hgvs_parentheses (hgvs_notation)

Normalizes all hgvs notation string to the same use of parentheses. End result is c.<variant_notation> with no use of brackets or parentheses to surround the variant such as c.(), c.[], c. (), etc.

Args: hgvs_notation (str): hgvs notation variant with no transcript (only c. or p. notation)

Returns: str: hgvs notation with no parenthesis or whitespace surrounding the variant description.

leiden.utilities.deep_copy (nested_list)

Makes a deep copy of lists that may or may not contain nested lists. Nested items that are not lists will not be deep copied, they will be shallow copied.

Args: nested_list (list): a list that may or may not contain nested lists. Nested lists may contain additional nestedlists.

Returns: list: deep copy of nested_list

leiden.utilities.find_string_index (string_list, search_string)

Given a list of strings and a string to search for, returns the index of the first element in the list that contains the search string. Note that the comparison is not sensitive to case or leading or trailing whitespace characters.

5.2. utilities 15
Args:  
  string_list (list of str): list of strings search_string: a string to search for in elements of string_list

Returns:

  int: index of the first instance of search_string as a substring of element in string_list. Returns -1 if 
  search_string is not found in the string_list.

leiden.utilities.get_omimid(link_url)
Given a URL to an entry on OMIM, return a string containing the OMIM ID for the entry.

Args:

  link_url (str): URL to the entry on OMIM. Assumed to be a valid link to an entry on PUBMED.  
  For example, U(http://www.omim.org/entry/102610#0003) is a valid link to an OMIM entry on the  
  ACTA1 gene. The url must contain the gene ID followed by the entry number in the URL separated  
  by a hash mark (such as, 102610#0003 in the example URL). URL may not contain other instances 
  of this pattern.

Returns:

  str: OMIM entry associated with the URL. This consists of the gene ID (such as 102610 for ACTA1  
  and a specific entry number (0003) separated by a hash mark (102610#0003 in the example above).  
  Raises: Value Error: if link does not contain a valid OMIM ID.

leiden.utilities.get_pmid(link_url)
Given a URL to a publication listed on PUBMED, return a string containing the PUBMED ID of the publication.

Args:

  link_url(str): URL to the publication on PUBMED. Assumed to be a valid link to a publication on PUBMED.  
  For example, U(http://www.ncbi.nlm.nih.gov/pubmed/19562689) is a valid pubmed publication URL. 
  The url must contain the PMID in the URL (19562689 in the example here) and contain no other 4  
  digit or longer numbers.

Returns: str: PUBMED ID associated with link_url (as specified by the N digit ID included in PUBMED 
  URLs).

Raises: ValueError: if there is no 4+ digit number in link_url

leiden.utilities.remove_times_reported(hgvs_notation)
Removes (Reported N times)substring from input if exists. Comparison is not case sensitive.

Args: hgvs_notation (str): typically an entry in the DNA Change column in table_data for a given variant on 
  an lvod installation.

Returns:

  str: hgvs_notation with instances of (Reported N times) removed. Whitespace surrounding this sub-
  string is removed in returned string.

leiden.utilities.swap(list, i, j)
Swaps elements at indices i and j in list. Indices must be within bounds of array. Index swapped with itself 
leaves the list unchanged.

Args: list (list): list of elements i (int): index of element to be swapped with element at index j. Must be within 
  bounds of array. j (int): index of element to be swapped with element at index i. Must be within bounds of 
  array.

Returns: list: list with elements at indices i and j swapped. If i and j are equal, list is unchanged.
5.3 file_io

This module has functions for reading and writing delimited files to and from 2D lists where first dimension is rows and the second is columns.

5.3.1 Member Descriptions

leiden.file_io.format_vcf_text(vcf_format_variants, info_column_entries)

Create formatted VCF file data from remapped variants.

**Args:**
- vcf_format_variants (list of tuples of str): list of tuples that contain chromosome_number, coordinate, ref, alt in that order
- info_column_entries (dict): dictionary where keys are tags to place in info column. The values are tuples that contain data_type, description, and a list of values (one per entry in vcf_format_variant)

**Returns:**
- list of lists: list of lists where each inner list is a row of the VCF file, and each element in the inner list represents the value of the respective column in a given row.

leiden.file_io.read_table_from_file(file_name, column_delimiter='\t')

Returns table of data from tab-delimited file. Alternate delimiter can be specified.

**Args:**
- file_name (str): path to tab-delimited file with column_delimiter (str, optional): column delimiter (tab by default)

**Returns:**
- list of lists: table of data from tab-delimited file 1st dimension is rows, 2nd is columns

leiden.file_io.write_table_to_file(file_name, table, column_delimiter='\t')

Writes table to tab-delimited file. Alternate delimiter can be specified.

**Args:**
- file_name (str): name of output file with extension (can include path) column_delimiter (str): column delimiter (tab by default) table (list of lists): table data to output to file. 1st dimension is rows, 2nd is columns.

5.4 web_io

This module has functions for reading HTML data from URLs. Essentially, this is just wrapping the library used to make HTML requests to make it easier to change if needed.

5.4.1 Member Descriptions

leiden.web_io.get_page_html(page_url)

Returns the html describing the page at the specified URL.

**Args:**
- page_url (str): URL to a specified website

**Returns:**
- str: HTML describing the specified page

**Raises:**
- ValueError: if requested URL not found
- IOError: if page could not be reached
5.5 annotate_vcf

This module provides a function to run Variant Effect Predictor annotation on VCF files.

5.5.1 Member Descriptions

leiden.annotate_vcf.annotate_vep(input_file, output_file)
Annotate VCF file with Variant Effect Predictor.

Args: input_file (str): input VCF file path output_file (str): output VCF file path (VEP annotation added to file).

5.6 validation

This module contains functions that are used to help with validation of annotated variants.

leiden.validation.get_ucsc_location_link(chromosome_number, start_coordinate, end_coordinate)
Returns link to relevant range in the UCSC genome browser. All parameters must be in valid range.

Args: chromosome_number (str): the chromosome number of the range to link to start_coordinate (str): the start coordinate of range to link to end_coordinate (str): the end coordinate of range to link to

Returns: str: URL link to display region in UCSC genome browser

leiden.validation.is_concordant(protein_change_1, protein_change_2)
Compares two protein change values to determine whether or not they are equivalent. Inputs are converted using normalized to a common representation before comparison. Returns False if either argument is an empty string.

Args: protein_change_1 (str): HGVS protein change notation protein_change_2 (str): HGVS protein change notation

Returns:

bool: True if HGVS protein change notations are equivalent; False if either notation is an empty string or if the two notations are not equivalent.

leiden.validation.normalize_protein_notation(protein_change_notation)
Tries to convert protein notations to a uniform format for equality comparison. Converts to lower-case, removes reference protein ID, removes p. notation, and converts all common stop-codon notations to a ‘*’.

Args: protein_change_notation (str): HGVS protein change notation

Returns: str: protein change notation normalized to uniform format.

leiden.validation.remove_p_dot_notation(annotation_text)
Removes the p-dot notation from a description of protein change. Accepted formats are p.change, p.(change, or p.[change], where change is returned and is the description of the protein change.

Args: annotation_text (str): p-dot notation describing the protein change

Returns: str: Annotation_text with the p-dot notation removed

Exception: ValueError: if the p-dot notation is not in one of the expected formats.
5.7 vcf

This module contains functions for parsing VCF files, providing data structures that provide easier access to information in annotated VCF files, and converting pandas dataframes with HGVS notation variants and associated data to VCF format.
In addition to the individual scripts installed with this package, I have also included an example driver script to run the entire pipeline.

**Important:** This script runs annotations serially with Variant Effect Predictor, which can take some time to execute. If you have access to a distributed computing cluster, you may want to develop your own driver script that runs the annotation portion of the pipeline in parallel.

**Tip:** All scripts are implemented using argparse and have built-in help, which accessible via:

```python
python <script_name>.py --help
```

### 6.1 run_all.py

run_all.py runs the full data extraction and validation process, producing a VCF file with only validated variants. Discordant variants are saved a separate VCF. See *Data* for more information.

#### 6.1.1 Example Usage

There are a few use-cases for run_all.py:

1. You are starting completely from scratch (no data has been downloaded from LOVD)

   ```python
   python run_all.py -u http://www.dmd.nl/nmdb2/ -output_directory my_output_directory
   ```

   This will download data from all genes on the specified LOVD URL, saving one .txt file (`<gene_name>.txt`) with raw data as well as one VCF file per gene (`<gene_name>_ANNOTATED.vcf`) with variants in VCF format and annotations (Variant Effect Predictor along with original data from LOVD table).

   Note that files are not saved for genes with no listed variants at the specified URL. Variants that fail to remap to VCF format are saved to `<gene_name>_remapping_errors.log` in their original LOVD table format.

2. You already have the txt files containing raw data from LOVD, but want to re-run the rest of the process. Note that this was primarily useful during development, but may still have some utility for others.

   ```python
   python run_all.py --no_download -output_directory my_output_directory
   ```

   **Note:** This assumes that the .txt files containing data extracted from LOVD are located in the specified output directory.
3. By default, run_all will not overwrite any existing annotated VCF files. This can be useful if annotation partially completed and you want to resume, etc. To force an overwrite:

    python run_all.py --force_overwrite -output_directory my_output_directory
    # OR
    python run_all.py --no_download -output_directory my_output_directory
Other Scripts

These scripts are called in sequence by the example driver script - run_all.py. However, they are also callable individually. Note that the python packages included with this project can also be used to write entirely new scripts if needed. Feel free to expand and write your own tools!

Tip: Note all scripts are made with argsparse, so contain built-in help. To access help simply execute: python <script_name>.py --help

Important: The leiden package must be installed or on your PYTHONPATH to run these scripts.

7.1 extract_data.py

extract_data.py allows raw data from the any leiden open variation database installation to be downloaded and saved to text files (one per gene). There are options to allow either data from all genes or a specific list of genes to be downloaded as needed. It also allows users to print a list of all available genes at a given URL, which is useful if you want to check what is available.

Note: Note that both LOVD versions 2 and 3 are supported for this script.

7.1.1 Example Usage

Download data for all genes from a given url to a specified output directory:

```python
python extract_data.py --all --leiden_url http://www.dmd.nl/nmdb2/ --output_directory my_directory
```

Download a list of specified genes from a given url to a specified output directory:

```python
python extract_data.py --leiden_url http://www.dmd.nl/nmdb2/ --output_directory my_directory --gene_list ACTA1 DYSF
```

Print a list of available genes at a specified URL:

```python
python extract_data.py --genes_available --leiden_url http://www.dmd.nl/nmdb2/
```
7.2 generate_annotated_vcf.py

This script utilizes VEP to annotate variants and output a VCF file (<original_file_name>.vcf). The original data from tables of data downloaded from LOVD are also added to the VCF in a format similar to VEP's CSQ tag in VCF annotation. Variants that could not be converted to VCF format are not saved to output file.

7.2.1 Example Usage

Run on list of files contained in a file (improved efficiency over multiple script calls):

```python
python generate_annotated_vcfs.py -f file_names.txt
```

7.3 validate_annotated_vcfs.py

This script takes either a single file or a list of annotated VCF files as input and outputs a single VCF file with all concordant variants from all input files. Variants that are not concordant are saved to a separate VCF.

How is concordance determined?

VEP provides a HGVS protein change prediction, which is compared to the protein change reported in LOVD. If neither LOVD or VEP report a protein change (intronic variants, splice variants, etc.), we instead... TODO

7.3.1 Example Usage

Using a file containing names of all input files:

```python
python extract_data.py --output_file output.vcf --discordant_output_file discordant.vcf -f file_list.txt
```
CHAPTER 8

Data

I have included examples of the major data formats output from the scripts in this package.

8.1 Extracted Data

The naming convention is <gene_name>.txt.
Contains original variant data as found on LOVD. Files are saved per gene by extract_data.py. See /data/ACTA1.txt

8.2 Annotated VCF Files

Naming convention is <gene_name>.vcf.
Contains original data in VCF format with Variant Effect Predictor annotation. Files are saved per input file by generate_annotated_vcf.py. See /data/ACTA1.vcf

8.3 Final VCF Files

This is the file format output by validate_annotated_vcfs.py. They are the VCF files that contains discordant and concordant variants respectively. See /data/lovd_discordant_variants.vcf and /data/lovd_validated_variants.vcf.
Indices and tables

- genindex
- modindex
- search
| leiden.annotate_vcf,?? |
| leiden.file_io,?? |
| leiden.leiden_database,?? |
| leiden.utilities,?? |
| leiden.validation,?? |
| leiden.web_io,?? |