openva_pipeline Documentation

Release 0.9.0

Jason Thomas, Samuel J. Clark, and Martin Bratschi

Jul 02, 2021

Contents:

The OpenVA Pipeline automates the processing of verbal autopsy (VA) data from an ODK Aggregate server, through the openVA library of VA algorithms from the R statistical software, to a DHIS2 server (with the VA DHIS2 Program).

CHAPTER 1

Software Requirements

The following software is required by the openVA pipeline (note: installation instructions are found on a different page)

- **Python 3.6, 3.7, 3.8, or 3.9**
  - PIP (tool for installing Python packages)
- OpenJDK or Java JDK 11,
- R (OpenVA Pipeline was tested on Version 4.1)
- SQLite3
- SQLCipher
- ODK Briefcase (OpenVA Pipeline was tested on Version v1.18.0)

It is also useful to install DB Browser for SQLite. This optional tool is useful for configuring the Pipeline (i.e., configuration tables in the SQLite Database).

CHAPTER 2

Installation Guide

The following instructions guide the installation of the openVA Pipeline on Ubuntu 20.04 operating system.

> **Note:** To make the installation process easier, all of the required software can be installed by downloading and running the bash script install_software_ubuntu_20_04.sh located in the main folder of the openVA_Pipeline repository.

1. Add the apt key (signed by Michael Rutter) to authenticate R package from CRAN and add the CRAN repository to get the latest version of R (for more details see CRAN page: Ubuntu packages for R).

```
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys␣
↪E298A3A825C0D65DFD57CBB651716619E084DAB9
$ sudo bash -c "echo 'deb https://cloud.r-project.org/bin/linux/
↪ubuntu focal-cran40/' >> /etc/apt/sources.list"
```

2. Install Python, OpenJDK, **R**, SQLite3, SQLCipher, and Git by typing the following commands at a terminal prompt (indicated by $)

```
$ sudo apt update
$ sudo apt install python3-pip openjdk-11-jdk r-base sqlite3␣
↪libsqlite3-dev sqlcipher libsqlcipher-dev curl libcurl4-openssl-
↪dev -y
```

3. (*optional*) Download the ODK-Briefcase-v1.18.0.jar file to a folder that will serve as the

openVA_Pipeline working directory.

4. Configure OpenJDK with R by running the following command

```
$ sudo R CMD javareconf
```

5. In a terminal, start **R** by simply typing `R` at the prompt, or use `sudo R` for system-wide installation of the packages. The necessary packages can be installed (with an internet connection) using the following command:

```
> install.packages("openVA")
```

Note that > in the previous command indicates the **R** prompt (not part of the actual command). This command will prompt the user to select a CRAN mirror (choose a mirror close to your geographic location), as well as a location for the library where the R packages will be installed (this does not happen if R is run as sudo). After the installation of the packages has been completed, you can exit **R** with the following command:

```
> q('no')
```

6. Python version 3.8 is included in the Ubuntu 20.04, but we still need to update the package installer, setup a virtual environment with Pipenv, and install the openva-pipeline package. These tasks can be accomplished at the terminal as follows:

```
$ pip3 install --upgrade pip --user
$ hash -d pip3
$ pip3 install --upgrade setuptools --user
$ pip3 install pipenv --user
$ pipenv install openva-pipeline
```

We can enter the virtual environment and import the openva-pipeline package with the following terminal commands:

```
$ pipenv shell
$ python


>>> import openva_pipeline as ovaPL   # import package
>>> help(ovaPL)                       # access primary help file␣
↪(hit q to exit)
>>> help(ovaPL.runPipeline)           # access help file for a␣
↪particular function
>>> quit()                            # return to virtualenv␣
↪terminal shell

$ exit
```

7. Install DB Browser for SQLite with the commands

```
$ sudo apt install sqlitebrowser -y
```

2.1 Alternative Installation Options

2.1.1 Using Java JDK (instead of OpenJDK)

Instructions for installing JDK 11 on Linux can be found here. After installing JDK 11, run the following command at the terminal to properly configure **R**

```
$ sudo R CMD javareconf
```

and then install the **R** packages (as described above).

2.1.2 Installing the Pipeline package with using a virtual environment (and pipenv)

Simply use `pip3` to install the openva-pipeline package as follows

```
$ pip3 install openva-pipeline --user
```

CHAPTER 3

Pipeline Configuration

1. **Create the SQLite database**: The openVA Pipeline uses an SQLite database to store and access configuration settings for ODK Aggregate, openVA in R, and DHIS2. Error and log messages are also stored to this database, along with the VA records downloaded from ODK Aggregate and the assigned COD.

    - While it is possible to create the Transfer Database manually (see the next bullet point) the openva-pipeline package has a built-in function for creating the database with the default settings. Open a terminal shell, change to the Pipeline's working directory, and start a Python session in the virtual environment with the following commands:

        ```
        $ pipenv shell
        $ python
        ```

        Within the Python interpreter load the openVA Pipeline package and call the `createTransferDB()`:

        ```
        >>> import openva_pipline as ovaPL
        >>> ovaPL.createTransferDB('Pipeline.db', '.', 'enilepiP
        ↪')
        >>> quit()
        ```

        This will create an encrypted SQLite database, called *Pipeline.db*, in the working directory with encryption key *enilepiP*.

    - **Manual Installation**

        1. The necessary tables and schema are created in the SQL file pipelineDB.sql, which can be downloaded from the OpenVA_Pipeline

GitHub webpage. Create the SQLite database in the folder that will serve as the Pipeline's working directory.

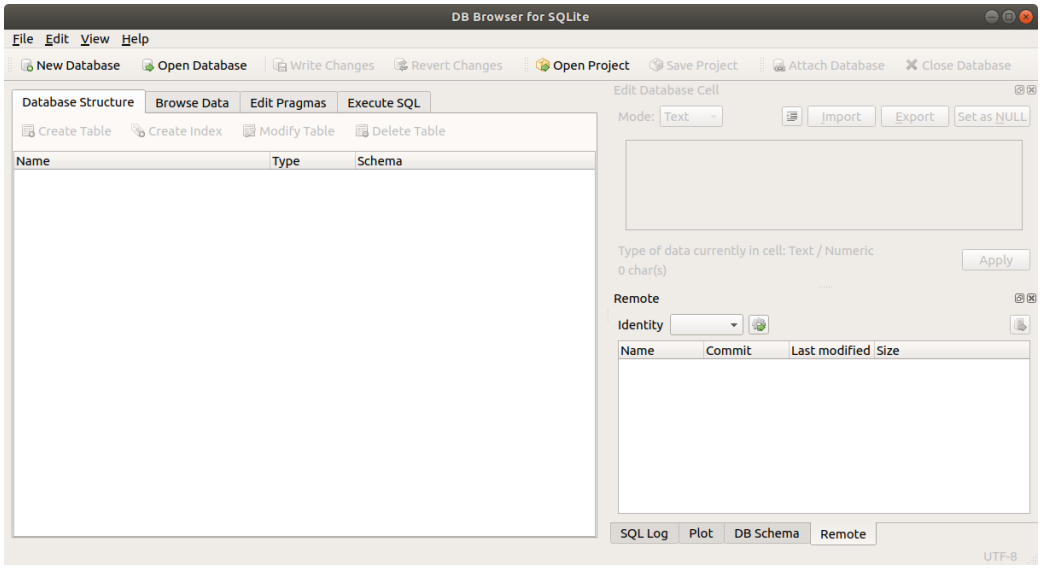
2. Use SQLCipher to create the pipeline database, assign an encryption key, and populate the database using the following commands (note that the `$` is the terminal prompt and `sqlite>` is the SQLite prompt, i.e., not part of the commands).

```
$ sqlcipher
sqlite> .open Pipeline.db
sqlite> PRAGMA key=encryption_key;
sqlite> .read "pipelineDB.sql"
sqlite> .tables
sqlite> -- take a look --
sqlite> .schema ODK_Conf
sqlite> SELECT odkURL from ODK_Conf;
sqlite> .quit
```

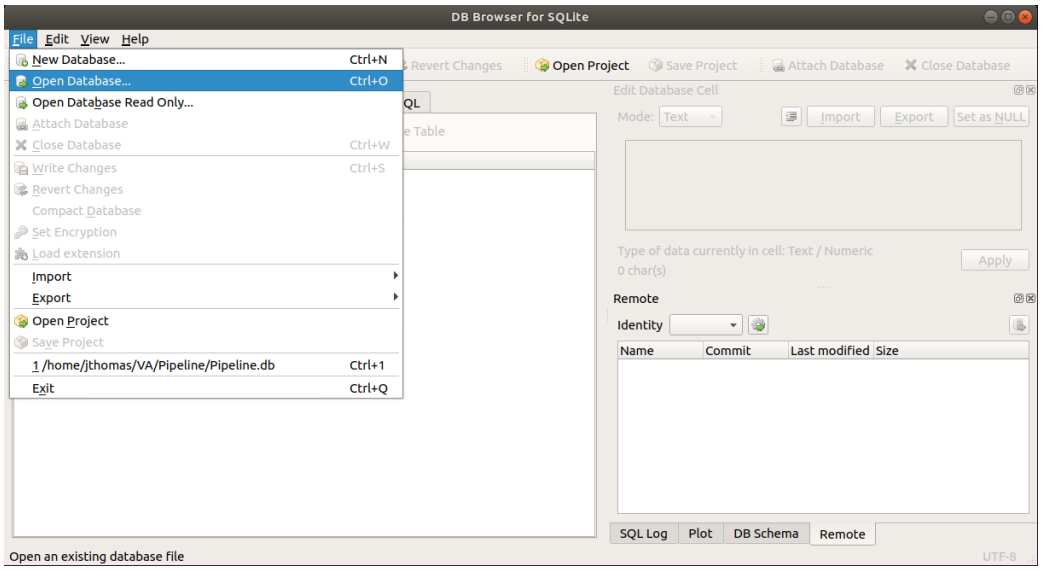
Note how the pipeline database is encrypted, and can be accessed via with SQLite command: `PRAGMA key = "encryption_key;"`

```
$ sqlcipher
sqlite> .open Pipeline.db
sqlite> .tables

Error: file is encrypted or is not a database

sqlite> PRAGMA key = "encryption_key";
sqlite> .tables
sqlite> .quit
```

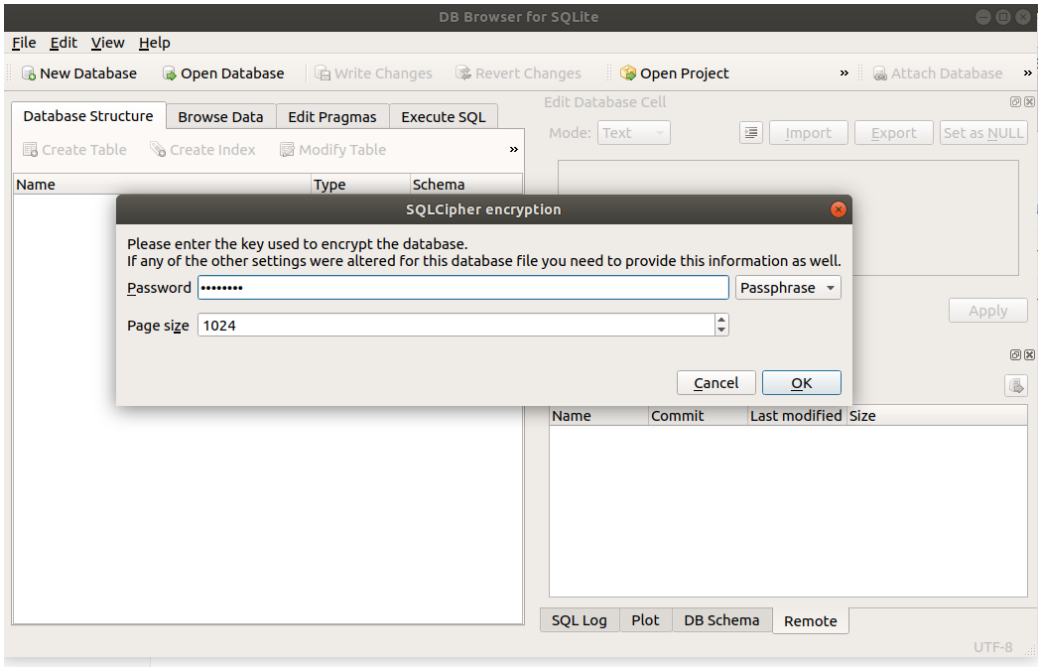
2. **Configure Pipeline**: The pipeline connects to ODK Aggregate and DHIS2 servers and thus requires usernames, passwords, and URLs. Arguments for openVA should also be supplied. We will use DB Browser for SQLite to configure these settings. Start by launching DB Browser from the terminal, which should open the window below `$ sqlitebrowser`



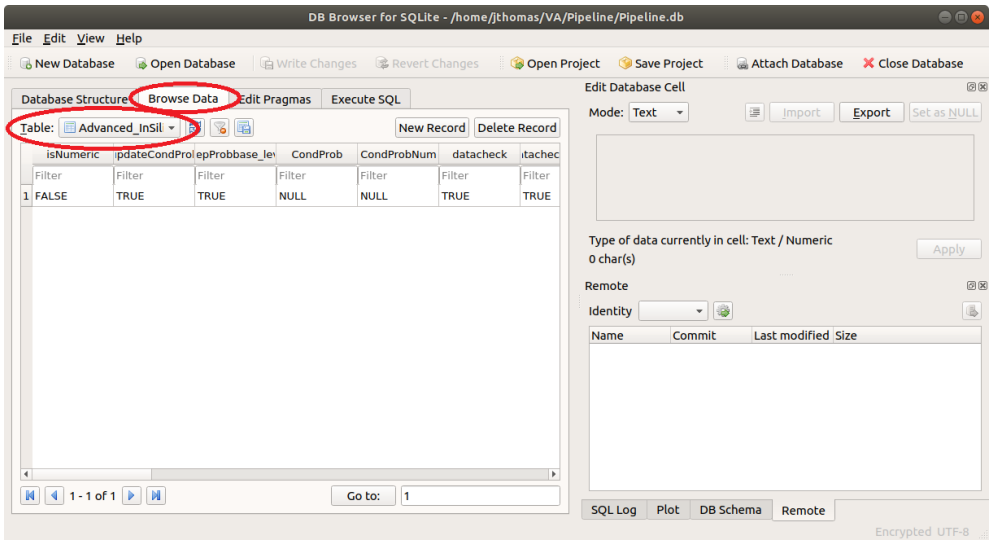
Next, open the database by selecting the menu options: *File -> Open Database...*

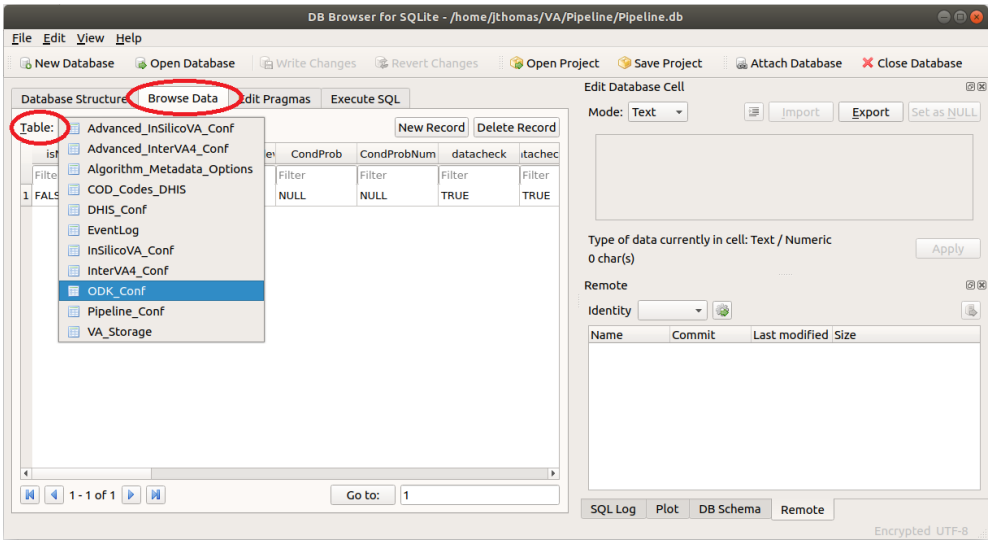


and navigate to the *Pipeline.db* SQLite database and click the *Open* button. This will prompt you to enter in encryption password.

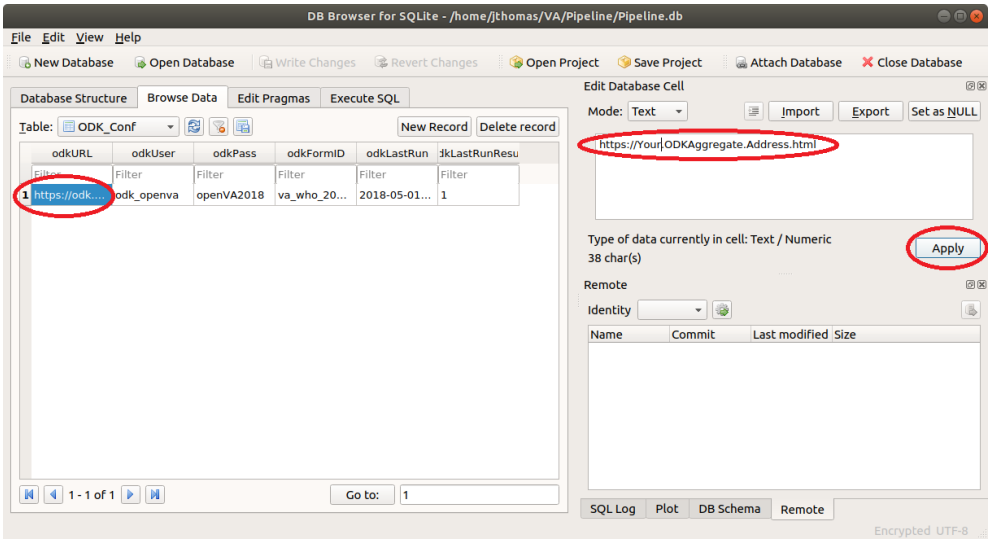


1. **ODK Configuration**: To configure the pipeline connection to ODK Aggregate, click on the *Browse Data* tab and select the ODK_Conf table as shown below.





Now, click on the *odkURL* column, enter the URL for your ODK Aggregate server, and click *Apply*.



Similarly, edit the *odkUser*, *odkPass*, and *odkFormID* columns so they contain a valid user name, password, and Form ID (see Form Management on ODK Aggregate server) of the VA questionnaire of your ODK Aggregate server.

2. **openVA Configuration**: The pipeline configuration for openVA is stored in the *Pipeline_Conf* table. Follow the steps described above (in the ODK Aggregate Configuration section) and edit the following columns:

    - *workingDirectory* – the directory where the pipeline files (i.e., *pipeline.py*, *Pipeline.db* and the ODK Briefcase application, *ODK-Briefcase-v1.10.1.jar*) are stored. Note that the pipeline will create new folders and files in this working directory, and must be run by a user with privileges for writing files to this location.
    - *algorithm* – currently, there are only three acceptable values for the alogrithm:

`Insilico`, `InterVA` or `SmartVA`

- *algorithmMetadataCode* – this column captures the necessary inputs for producing a COD, namely the VA questionnaire, the algorithm, and the symptom-cause information (SCI) (for more details, see the section: *Symptom-Cause Information*). Note that there are also different versions (e.g., InterVA 4.01 and InterVA 4.02, or WHO 2012 questionnare and the WHO 2016 instrument/questionnaire). It is important to keep track of these inputs in order to make the COD determination reproducible and to fully understand the assignment of the COD. A list of all algorith metadata codes is provided in the *dhisCode* column in the *Algorithm_Metadata_Options* table. The logic for each code is

  algorith|algorithm version|SCI|SCI version|instrument|instrument version

- *codSource* – both the InterVA and InSilicoVA algorithms return CODs from a list produced by the WHO, and thus this column should be left at the default value of `WHO`.

3. **DHIS2 Configuration**: The pipeline configuration for DHIS2 is located in the *DHIS_Conf* table, and the following columns should be edited with appropriate values for your DHIS2 server.

   - *dhisURL* – the URL for your DHIS2 server
   - *dhisUser* – the username for the DHIS2 account
   - *dhisPass* – the password for the DHIS2 account
   - *dhisOrgUnit* – the Organization Unit (e.g., districts) UID to which the verbal autopsies are associated. The organisation unit must be linked to the Verbal Autopsy program. For more details, see the DHIS2 Verbal Autopsy program installation guide

3. **SmartVA Configuration**: The pipeline can also be configured to run SmartVA using the command line interface (CLI) available from the ihmeuw/SmartVA-Analyze repository.

   1. Download the smartva CLI from the following repository: https://github.com/ihmeuw/SmartVA-Analyze/releases and save it in the pipeline's working directory (see below).
   2. Update the *Pipeline_Conf* table in the SQLite database with the following values:

      - *workingDirectory* – the directory where the pipeline files are stored – **THIS IS WHERE THE smartva CLI file should be downloaded**.
      - *openVA_Algorithm* – set this field to `SmartVA`
      - *algorithmMetadataCode* – set this field to the appropriate SCI, e.g.

        SmartVA|2.0.0_a8|PHMRCShort|1|PHMRCShort|1

      - *codSource* – set this field to''Tariff''.

CHAPTER 4

Miscellaneous Notes

4.1 Symptom-Cause Information

A key component of automated cause assignment methods for VA is the symptom-cause information (SCI) that describes how VA symptoms are related to each cause. It is likely that the relationships of VA symptoms to causes vary in important ways across space and between administrative jurisdictions, and they are likely to change through time as new diseases and conditions emerge and as treatments become available. Consequently, automated cause assignment algorithms used for mortality surveillance should optimally rely on representative SCI that is locally and continuously updated. Furthermore, it is vital to track the SCI used for COD assignment to enable reproducibility and to fully understand the assignment of the COD.

CHAPTER 5

Setting Up and Running the

CHAPTER 6

openVA Pipeline

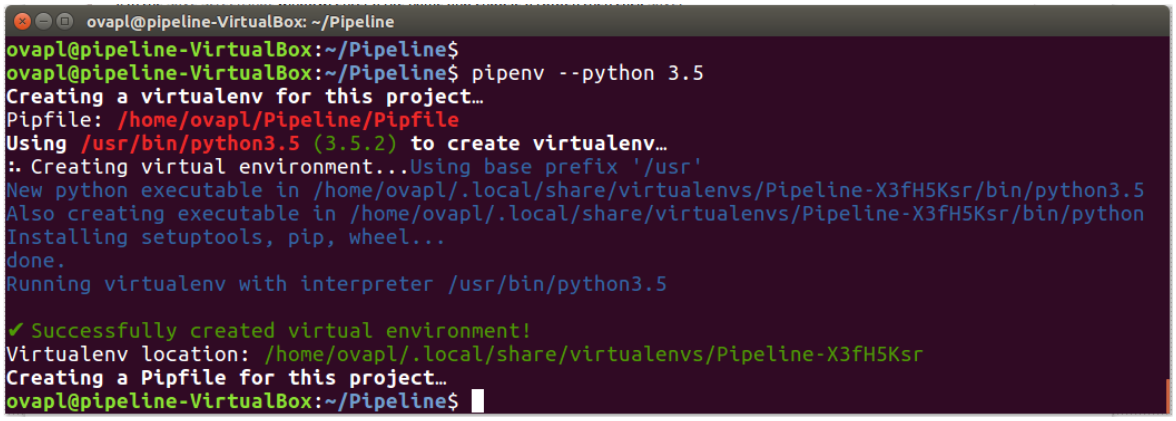
6.1 Quick Demonstration

Once Python3, Java, R, and openVA have been installed on your computer, there are only a few steps needed to install and demonstrate the openVA Pipeline.

1. Start by opening a terminal and changing to the directory that will serve as the working directory for the openVA Pipline. The working directory for this example will be a folder called Pipeline located in the user's home directory

```
$ mkdir -p $HOME/Pipeline
$ cd $HOME/Pipeline
$ pipenv --python 3.8
```

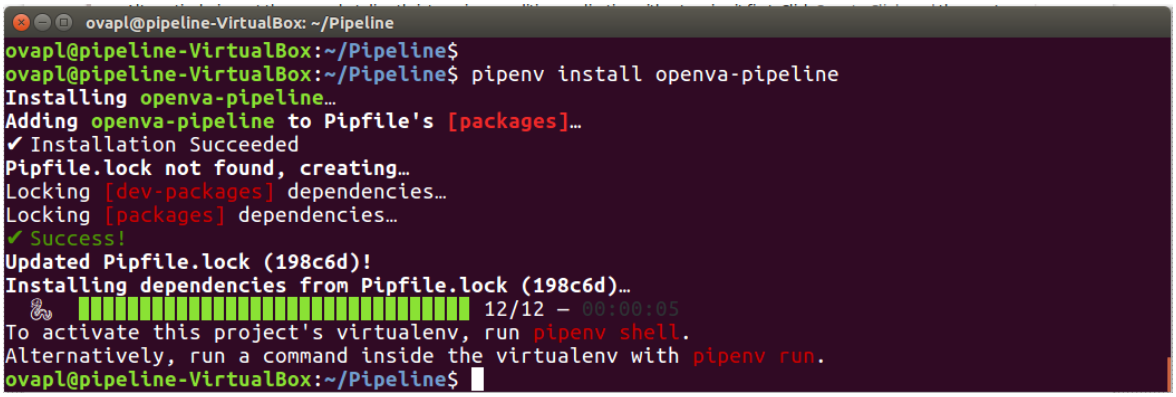
   The last (recommended) step uses Pipenv to create a virtual environment for the pipeline, which should look like the following screenshot



2. Use Pipenv to install the openVA Pipeline package from PyPI

```
$ pipenv install openva-pipeline
```

After this command, the output produced by Pipenv is shown in the following screenshot



3. Open a Pipenv shell load the openVA Pipeline package, download the ODK Briefcase, install the Transfer database, and run the Pipeline. The final command exits out of Python.

```
$ pipenv shell
```

(*Python commands. . .*)

```
>>> import openva_pipeline as ovaPL
>>> ovaPL.downloadBriefcase()
>>> ovaPL.createTransferDB("Pipeline.db", ".", "enilepiP")
>>> ovaPL.runPipeline("Pipeline.db", ".", "enilepiP")
>>> quit()
```

Here are a few more details about each Python command. . .

1. `import openva_pipeline as ovaPL` – load the openVA Pipeline package and use the shortcut name ovaPL. To access the tools inside the openVA Pipeline package, use the nickname, followed by a `.`, and then the name of the function or class.

2. `ovaPL.downloadBriefcase()` – call the function that downloads the ODK Briefcase app (version 1.18.0) from the ODK GitHub page. The app will be downloaded to the current working directory.

3. `ovaPL.createTransferDB("Pipeline.db", ".", "enilepiP")` – create the SQLite database that contains the configuration settings for the Pipeline, stores VA data and results, and includes an event/error log. The arguments are: (1) the name of the Transfer DB file ("Pipeline.db"); (2) the path to the directory where the DB file will be created; and (3) the key for encrypting the DB. The default settings point the Pipeline to an ODK Aggregate and DHIS2 server hosted by the SwissTPH.

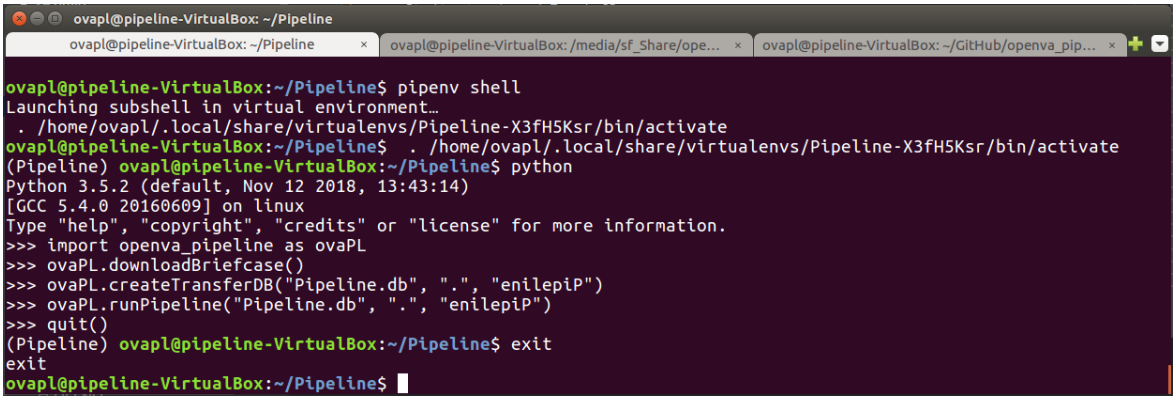
4. `ovaPL.runPipeline("Pipeline.db", ".", "enilepiP")` – run each step of the Pipeline: download records from ODK Aggregate; run openVA to assign causes of death; store the results in the Transfer DB; and post the VA data and assigned causes of death to a DHIS2 server (with the VA Program installed). The arguments are (again): (1) the name of the Transfer DB file ("Pipeline.db"); (2) the path to the working directory for the Pipeline (the ODK Briefcase file and the Transfer DB need to be located in this directory); and (3) the key for encrypting the DB.

   Note: runPipeline() has fourth parameter *export_to_DHIS* with a default argument of True. If you do not want to post VA events to DHIS2, then pass a value of False, i.e. export_to_DHIS = False.

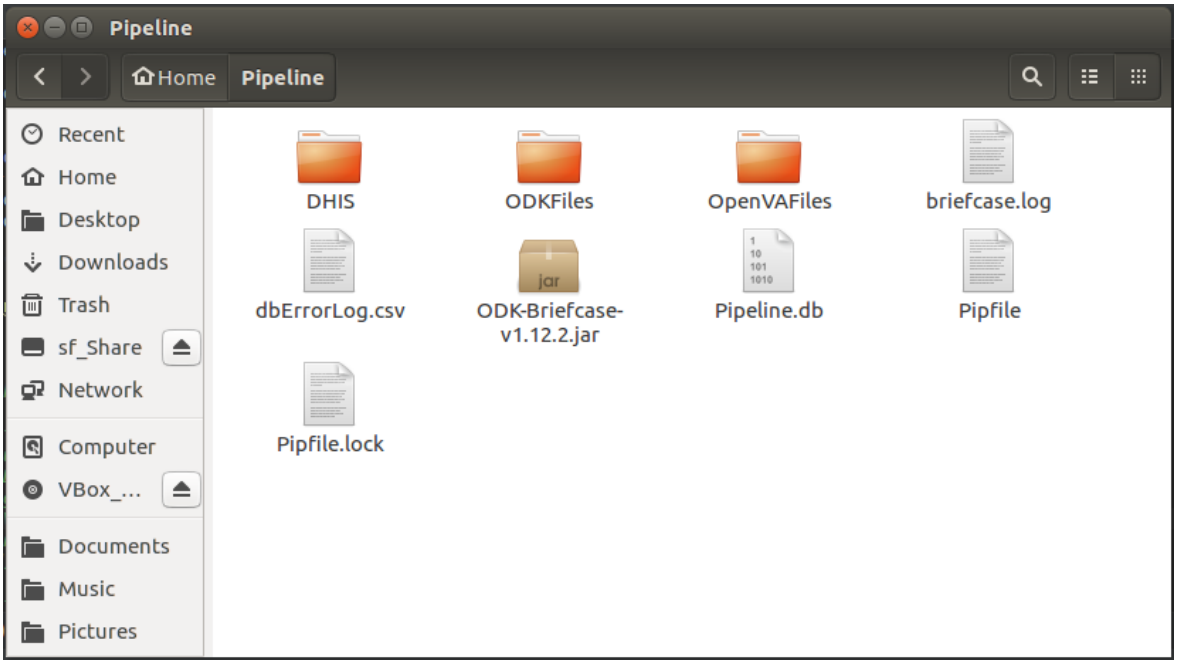
   ---

5. `quit()` – exit out of Python.

A demonstration of these commands is shown in the following screenshot.

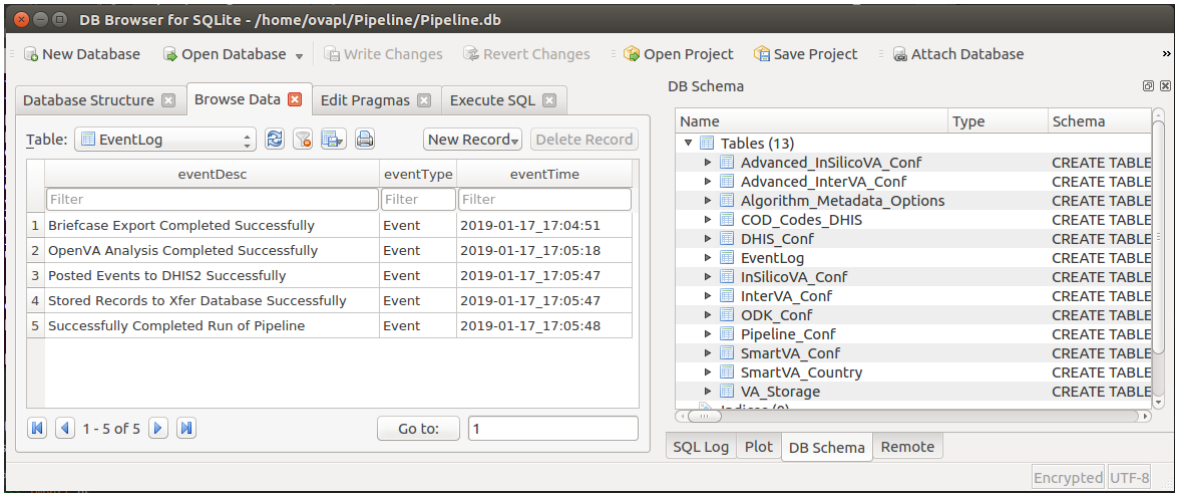


Once the `runPipeline` function has completed, we can check the results in several locations. First, let us take a look in the Pipeline's working directory.



We can see folders where the files associated with the ODK Briefcase export and downloaded records (*ODKFiles*); the R script and log files from openVA (*OpenVAFiles*); and the files containg the VA events posted to the DHIS2 VA program (*DHIS*). The working directory also contains the ODK Briefcase jar file (and log file), the files generated by Pipenv (that keep track of the packages and dependences for our virtual environment); and another file *dbErrorLog.csv* which contains error messages for the Pipeline when it is unable to connect to the Transfer Database. If we take a look at *dbErrorLog.csv*, we see it is blank and thus the Pipeline was able to connect to *Pipeline.db*.

Finally, if we look at the EventLog table in the Transfer database, we see that each step of the Pipeline was successfully completed (along with the date and time when it finished).


DB Browser for SQLite - /home/ovapl/Pipeline/Pipeline.db
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Table: EventLog
New Record
Delete Record
eventDesc
eventType
eventTime
Filter
1 Briefcase Export Completed Successfully Event 2019-01-17_17:04:51
2 OpenVA Analysis Completed Successfully Event 2019-01-17_17:05:18
3 Posted Events to DHIS2 Successfully Event 2019-01-17_17:05:47
4 Stored Records to Xfer Database Successfully Event 2019-01-17_17:05:47
5 Successfully Completed Run of Pipeline Event 2019-01-17_17:05:48
1 - 5 of 5
Go to: 1
DB Schema
Name
Type
Schema
Tables (13)
Advanced_InSilicoVA_Conf CREATE TABLE
Advanced_InterVA_Conf CREATE TABLE
Algorithm_Metadata_Options CREATE TABLE
COD_Codes_DHIS CREATE TABLE
DHIS_Conf CREATE TABLE
EventLog CREATE TABLE
InSilicoVA_Conf CREATE TABLE
InterVA_Conf CREATE TABLE
ODK_Conf CREATE TABLE
Pipeline_Conf CREATE TABLE
SmartVA_Conf CREATE TABLE
SmartVA_Country CREATE TABLE
VA_Storage CREATE TABLE
SQL Log
Plot
DB Schema
Remote
Encrypted
UTF-8

CHAPTER 7

Documentation for classes, functions, and methods

7.1 Running the OpenVA Pipeline

The openva_pipeline package includes two convenience functions for

1. creating the Transfer Database – a database that holds configuration settings, VA data and results, and a table for logging events and errors; and
2. running through all of the steps in the openVA Pipeline

openva_pipeline.runPipeline.**createTransferDB**(*database_file_name*, *database_directory*, *database_key*)

Create the (SQLite encrypted) Transfer Database.

Parameters

- **database_file_name** – File name for the Transfer Database.
- **database_directory** – Path of the Transfer Database.
- **datatbase_key** – Encryption key for the Transfer Database
- **export_to_DHIS** – Indicator for posting VA records to a DHIS2 server.

openva_pipeline.runPipeline.**runPipeline**(*database_file_name*, *database_directory*, *database_key*, *export_to_DHIS=True*)

Runs through all steps of the OpenVA Pipeline

This function is a wrapper for the Pipeline class, which runs through all steps of the OpenVA Pipeline – (1) connect to Transfer Database (to retrieve configuration settings); (2) connect to ODK Aggregate to download a CSV file with VA records; (3) run openVA (or SmartVA) to assign cause of death; and (4) store CoD results and VA data in the Transfer Database as well as a DHIS2 VA Program (if requested).

Parameters

- **database_file_name** – File name for the Transfer Database.
- **database_directory** – Path of the Transfer Database.
- **datatbase_key** – Encryption key for the Transfer Database
- **export_to_DHIS** (*(Boolean)*) – Indicator for posting VA records to a DHIS2 server.

7.2 Main Interface

The OpenVA Pipeline is run using the following function

class openva_pipeline.pipeline.**Pipeline**(*dbFileName*, *dbDirectory*, *dbKey*, *useDHIS=True*)

Primary API for the openVA pipeline.

This class calls three others to move verbal autopsy data from an ODK Aggregate server (using the ODK class), through the openVA R package to assign cause of death (using the OpenVA class), and deposits the VA records with assigned causes to either/both a DHIS server (using the DHIS class) or the Transfer database – a local database which also contains configuration settings for the pipeline. The TransferDB class performs the final step of storing the results locally as well as accessing the configuration settings.

Parameters

- **dbFileName** (*string*) – File name of the Tranfser database.
- **dbDirectory** (*string*) – str Path of folder containing the Transfer database.
- **dbKey** (*string*) – Encryption key for the Transfer database.

closePipeline()

Update ODK_Conf ODKLastRun in Transfer DB and clean up files.

This method calls methods in the *TransferDB* class to remove the data files created at each step of the pipeline. More specifically, it runs *TransferDB.cleanODK()* to remove the ODK Briefcase export files ("ODKFiles/odkBCExportNew.csv" and "ODKFiles/odkBCExportPrev.csv") if they exist; *TransferDB.cleanOpenVA()*

to remove the input data file ("OpenVAFiles/openVA_input.csv") and the output files ("OpenVAFiles/recordStorage.csv", "OpenVAFiles/newStorage.csv", and "OpenVAFiles/entityAttributeValue.csv") – note that all of these results are stored in either/both of the Transfer DB and the DHIS2 server's VA program; and, third, the method *TransferDB.cleanDHIS()* is called to remove the blobs posted to the DHIS2 server and stored in the folder "DHIS/blobs". Finally, this method updates the Transfer DB's value in the ODK_Conf table's variable odkLastRun so the next ODK Export file does not include VA records already processed through the pipeline.

config()

Fetch configuration settings from Transfer DB.

This method queries the Transfer database (DB) and returns objects that can be used as the arguments for other methods in this class, i.e., *Pipeline.runODK()*, *Pipeline.runOpenVA()*, and *Pipeline.runDHIS()*.

Parameters

- **dbFileName** (*str*) – File name of the Transfer DB. (e.g., Pipeline.db)
- **dbDirectory** (*str*) – Path to the location of the Transfer DB.
- **dbKey** (*str*) – Encryption key for the Transfer DB
- **plRunDate** (*date*) – Date when pipeline started latest run (YYYY-MM-DD_hh:mm:ss).

Returns Configuration settings for pipeline steps (e.g. connecting to ODK Aggregate, running openVA, or posting records to DHIS)

Return type dictionary

logEvent(*eventDesc*, *eventType*)

Commit event or error message into EventLog table of transfer database.

Parameters

- **eventDesc** (*string*) – Description of the event.
- **eventType** – Type of event (error or information)

runDHIS(*argsDHIS*, *argsPipeline*)

Connect to API and post events.

This method first calls the method *DHIS.connect()* to establish a connection with a DHIS2 server and, second calls the method *DHIS.postVA()* to post VA data, the assigned causes of death, and associated metadata (concerning cause assignment).

Parameters

- **argsDHIS** – Configuration settings for connecting to DHIS2 server.

- **argsPipeline** (*named tuple*) – Configuration settings for OpenVA Pipeline

Returns VA Program ID from the DHIS2 server, the log from the DHIS2 connection, and the number of records posted to DHIS2

Return type dictionary

runODK(*argsODK*, *argsPipeline*)

Run check duplicates, copy file, and briefcase.

This method downloads data from either (1) an ODK Central server, using *ODK.central()*, or (2) an ODK Aggregate server using the Java application ODK Briefcase, by calling the method *ODK.briefcase()*. The configuration settings are taken from the argument argsODK (see *Pipeline.config()*) , and downloads verbal autopsy (VA) records as a (csv) export from an ODK Central/Aggregate server. If there is a previous ODK export file, this method merges the files by keeping only the unique VA records.

Parameters

- **argsODK** (*named tuple*) – Arguments passed to connect and download records from the ODK Central/Aggregate server.
- **argsPipeline** (*named tuple*) – Arguments for configuration the openva pipeline.

Returns Return value from method subprocess.run()

Return type subprocess.CompletedProcess

runOpenVA(*argsOpenVA*, *argsPipeline*, *odkID*, *runDate*)

Create & run script or run smartva.

This method runs the through the suite of methods in the *OpenVA*. class. The list of tasks performed (in order) are: (1) call the method *OpenVA.copyVA()* to copy over CSV files with VA data (retrieved from ODK Aggregate); (2) use the method *OpenVA.rScript()* to create an R script; and (3) call the method *OpenVA.getCOD()* to run the R script that estimates the causes of death and stores the results in "OpenVAFiles/recordStorage.csv" and "OpenVAFiles/entityAttributeValue.csv" (the former serving as the blob posted to DHIS2).

Parameters

- **argsOpenVA** (*named tuple*) – Configuration settings for openVA.
- **argsPipeline** (*named tuple*) – Configuration settings for OpenVA Pipeline
- **odkID** (*string*) – column/variable name of VA record ID in ODK export

- **runDate** (*nowDate.strftime("%Y-%m-%d_%H:%M:%S")*) – date and time when OpenVA Pipeline ran

Returns an indicator of zero VA records in the ODK export

Return type dictionary

storeResultsDB()
Store VA results in Transfer database.

7.3 API for Transfer Database

class openva_pipeline.transferDB.**TransferDB**(*dbFileName*, *dbDirectory*, *dbKey*, *plRunDate*)
This class handles interactions with the Transfer database.

The Pipeline accesses configuration information from the Transfer database, and also stores log messages and verbal autopsy records in the DB. The Transfer database is encrypted using sqlcipher3 (and the pysqlcipher3 module is imported to establish DB connection).

Parameters

Parameters

- **dbFileName** (*str*) – File name of the Tranfser database.
- **dbDirectory** (*str*) – Path of folder containing the Transfer database.
- **dbKey** (*str*) – Encryption key for the Transfer database.
- **plRunDate** (*date*) – Date when pipeline started latest run (YYYY-MM-DD_hh:mm:ss).

checkDuplicates(*conn*)
Search for duplicate VA records.

This method searches for duplicate VA records in ODK Briefcase export file and the Tranfser DB. If duplicates are found, a warning message is logged to the EventLog table in the Transfer database and the duplicate records are removed from the ODK Briefcase export file.

Parameters **conn** (*sqlite3 Connection object*) – A connection to the Transfer Database (e.g. the object returned from *TransferDB.connectDB()*.)

Raises DatabaseConnectionError, PipelineError

cleanDHIS()
Remove DHIS2 blob files.

cleanODK()
Remove ODK Briefcase Export files.

cleanOpenVA()
Remove openVA files with COD results.

configDHIS(*conn*, *algorithm*)
Query DHIS configuration settings from database.

This method is intended to be used in conjunction with (1) *TransferDB.connectDB()*, which establishes a connection to a database with the Pipeline configuration settings; and (2) *DHIS.connect()*, which establishes a connection to a DHIS server. Thus, TransferDB.configDHIS() gets its input from *TransferDB.connectDB()* and the output from TransferDB.config() is a valid argument for *DHIS.connect()*

Parameters

- **conn** (*sqlite3 Connection object*) – A connection to the Transfer Database (e.g. the object returned from *TransferDB.connectDB()*.)
- **algorithm** (*str*) – VA algorithm used by R package openVA

Returns Contains all parameters for *DHIS.connect()*.

Return type tuple

Raises DHISConfigurationError

configODK(*conn*)
Query ODK configuration settings from database.

This method is intended to be used in conjunction with (1) *TransferDB.connectDB()*, which establishes a connection to a database with the Pipeline configuration settings; and (2) *ODK.briefcase()*, which establishes a connection to an ODK Aggregate server. Thus, TransferDB.configODK() gets its input from *TransferDB.connectDB()* and the output from TransferDB.configODK() is a valid argument for *ODK.briefcase()*.

Parameters **conn** (*sqlite3 Connection object*) – A connection to the Transfer Database (e.g. the object returned from *TransferDB.connectDB()*.)

Returns Contains all parameters for *ODK.briefcase()*.

Return type tuple

Raises ODKConfigurationError

configOpenVA(*conn*, *algorithm*, *pipelineDir*)
Query OpenVA configuration settings from database.

This method is intended to receive its input (a Connection object) from *TransferDB.connectDB()*, which establishes a connection to a database with the Pipeline configuration settings. It sets up the configuration for all of the VA algorithms included in the R package openVA. The output from *configOpenVA()* serves as an input to the method `OpenVA.setAlgorithmParameters()`. This is a wrapper function that calls `configInterVA()`, `configInSilicoVA()`, and `configSmartVA()` to actually pull configuration settings from the database.

Parameters

- **conn** (*sqlite3 Connection object*) – A connection to the Transfer Database (e.g. the object returned from *TransferDB.connectDB()*.)
- **algorithm** (*str*) – VA algorithm used by R package openVA
- **pipelineDir** (*str*) – Working directory for the Pipeline

Returns Contains all parameters needed for OpenVA.setAlgorithmParameters().

Rtypes tuple

Raises OpenVAConfigurationError

configPipeline(*conn*)

Grabs Pipline configuration settings.

This method queries the Pipeline_Conf table in Transfer database and returns a tuple with attributes (1) algorithmMetadataCode; (2) codSource; (3) algorithm; and (4) workingDirectory.

Returns Arguments needed to configure the OpenVA Pipeline alogrithmMetadataCode - attribute describing VA data codSource - attribute detailing the source of the Cause of Death list algorithm - attribute indicating which VA algorithm to use workingDirectory - attribute indicating the working directory

Return type tuple

Raises PipelineConfigurationError

connectDB()

Connect to Transfer database.

Uses parameters supplied to the parent class, TransferDB, to connect to the (encrypted) Transfer database.

Returns Used to query (encrypted) SQLite database.

Return type SQLite database connection object

Raises DatabaseConnectionError

makePipelineDirs()
Create directories for storing files (if they don't exist).

The method creates the following folders in the working directory (as set in the Transfer database table Pipeline_Conf): (1) ODKFiles for files containing verbal autopsy records from the ODK Aggregate server; (2) OpenVAFiles containing R scripts and results from the cause assignment algorithms; and (3) DHIS for holding blobs that will be stored in a data repository (DHIS2 server and/or the local Transfer database).

Raises PipelineError

storeVA(*conn*)
Store VA records in Transfer database.

This method is intended to be used in conjunction with the *DHIS* class, which prepares the records into the proper format for storage in the Transfer database.

Parameters **conn** (*sqlite3 Connection object*) – A connection to the Transfer Database (e.g. the object returned from *TransferDB.connectDB()*.)

Raises PipelineError, DatabaseConnectionError

updateODKLastRun(*conn*, *plRunDate*)
Update Transfer Database table ODK_Conf.odkLastRun

Parameters

- **conn** (*sqlite3 Connection object*) – A connection to the Transfer Database (e.g. the object returned from *TransferDB.connectDB()*.)
- **plRunDate** (*date (YYYY-MM-DD_hh:mm:ss)*) – Date when pipeline started latest run

7.4 API for ODK Briefcase

class openva_pipeline.odk.**ODK**(*odkSettings*, *workingDirectory*)
Manages Pipeline's interaction with ODK Aggregate.

This class handles the segment of the pipeline related to ODK. The ODK.connect() method calls ODK Briefcase to connect with an ODK Aggregate server and export VA records. It also checks for previously exported files and updates them as needed. Finally, it logs messages and errors to the pipeline database.

Parameters

- **odkSettings** (*named tuple*) – A named tuple with all of configuration settings as attributes.

- **workingDirectory** (*string*) – Directory where openVA Pipeline should create files.

briefcase()
Calls ODK Briefcase.

This method spawns a new process that runs the ODK Briefcase Java application (via a command-line interface) to download a CSV file with verbal autopsy records from an ODK Aggregate server.

Returns Return value from method subprocess.run()

Return type subprocess.CompletedProcess

Raises ODKError

central()
Connects to ODK Central through api.

This method calls requests.get to download a CSV file with verbal autopsy records from an ODK Collect server.

Returns Returns a string indicating the number of downloaded records.

Return type string

Raises ODKError

mergeToPrevExport()
Merge previous ODK Briefcase export files.

7.5 API for OpenVA

class openva_pipeline.openVA.**OpenVA**(*vaArgs*, *pipelineArgs*, *odkID*, *runDate*)
Assign cause of death (COD) to verbal autopsies (VA) R package openVA.

This class creates and executes an R script that copies (and merges) ODK Briefcase exports, runs openVA to assign CODs, and creates outputs for depositing in the Transfers DB and to a DHIS server.

Parameters **algorithm** (*str*) – Which VA algorithm should be used to assign COD.

Raises OpenVAError

copyVA()

Create data file for openVA by merging ODK export files & converting with pycrossva.

Returns Indicator of an empty (i.e. no records) ODK export file

Return type logical

getCOD()
Create and execute R script to assign a COD with openVA; or call the SmartVA CLI to assign COD.

rScript()
Create an R script for running openVA and assigning CODs.

smartVA_to_csv()

Write two CSV files: (1) Entity Value Attribute blob pushed to

DHIS2 (entityAttributeValue.csv)

(2) table for transfer database (recordStorage.csv)

Both CSV files are stored in the OpenVA folder.

7.6 API for DHIS2

class openva_pipeline.dhis.**DHIS**(*dhisArgs*, *workingDirectory*)
Class for transfering VA records (with assigned CODs) to the DHIS2 server.

This class includes methods for importing VA results (i.e. assigned causes of death from openVA or SmartVA) as CSV files, connecting to a DHIS2 server with the Verbal Autopsy Program, and posting the results to the DHIS2 server and/or the local Transfer database.

Parameters

- **dhisArgs** (*(named) tuple*) – Contains parameter values for connected to DHIS2, as returned by transferDB.configDHIS().
- **workingDirectory** (*string*) – Workind direcotry for the openVA Pipeline

Raises DHISError

connect()
Setup connection to DHIS2 server.

This creates a connection to DHIS2's VA Program ID by creating an instance of *API*. This method also checks that the VA Program and the organization unit can both be found on the DHIS2 server. The configuration settings for connecting to the DHIS2 (e.g., URL, username, password, etc.) are taken from the arguments passed to this method's class *DHIS* (these settings can be created using the method *Pipeline.config*).

Returns A class instance for interacting with the DHIS2 API.

Return type Instance of the *API* class

postVA(*apiDHIS*)
Post VA records to DHIS.

This method reads in a CSV file ("entityAttribuesValue.csv") with cause of death results (from openVA) then formats events and posts them to a VA Program (installed on DHIS2 server).

Parameters **apiDHIS** (Instance of the *API* class) – A class instance for interacting with the DHIS2 API created by the method *DHIS.connect*

Returns Log information receieved after posting events to the VA Program on a DHIS2 server (see *API.post*).

Return type dict

Raises DHISError

verifyPost(*postLog*, *apiDHIS*)
Verify that VA records were posted to DHIS2 server.

Parameters

- **postLog** (*dictionary*) – Log information retrieved after posting events to a VA Program on a DHIS2 server; this is the return object from *DHIS.postVA*.
- **apiDHIS** (Instance of the *API* class) – A class instance for interacting with the DHIS2 API created by the method *DHIS.connect*

Raises DHISError

class openva_pipeline.dhis.**API**(*dhisURL*, *dhisUser*, *dhisPass*)
This class provides methods for interacting with the DHIS2 API.

This class is called by an instance of the *DHIS* to retrieve information from and post verbal autopsy records (and results) to a DHIS2 server that has the Verbal Autopsy program installed.

Parameters

- **dhisURL** (*string*) – Web address for DHIS2 server (e.g., "play.dhis2.org/demo").
- **dhisUser** (*string*) – Username for DHIS2 account.
- **dhisPassword** (*string*) – Password for DHIS2 account.

Raises DHISError

get(*endpoint*, *params=None*)
GET method for DHIS2 API.

Return type dict

post(*endpoint*, *data*)
POST method for DHIS2 API.

Return type dict

post_blob(*f*)
Post file to DHIS2 and return created UID for that file

Return type str

class openva_pipeline.dhis.**VerbalAutopsyEvent**(*va_id*, *program*, *dhis_orgunit*, *event_date*, *sex*, *dob*, *age*, *cod_code*, *algorithm_metadata*, *odk_id*, *file_id*)
Create DHIS2 event + a BLOB file resource

Parameters

- **va_id** (*string*) – UID for verbal autopsy record (used as a DHIS2 data element)
- **program** (*string*) – UID of the DHIS2's Verbal Autopsy program
- **dhis_orgunit** (*string*) – UID for the DHIS2 Organization Unit where the event (death) should be registered.
- **event_date** (*datetime.date*) – Date of death with YYYY-MM-DD format
- **sex** (*string or integer*) – Sex of the deceased (used as a DHIS2 data element). Possible values must fit to an option in the VA Program's "Sex" optionSet: female, male, missing, unknown). If SmartVA is used to assign cause of death, then sex is an integer with 1 = male and 2 = female).
- **dob** (*datetime.date*) – Date of birth of the deceased with YYYY-MM-DD format (used as a DHIS2 data element)
- **age** (*integer*) – Age (in years) at time of death
- **cod_code** (*string*) – Coded cause of death (must fit to an option in the VA Program's "CoD codes" optionSet.
- **algorithm_metadata** (*string*) – Code for how the CoD was obtained (must fit in VA Program's "Algorithm Metadata" optionSet.
- **odk_id** (*string*) – UID for the VA record assigned by the ODK Aggregate server

- **file_id** (*string*) – UID for the blob file (containing the VA data and results) posted to (and assigned by) DHIS2 server.

format_to_dhis2(*dhisUser*)
Format object to DHIS2 compatible event for DHIS2 API

Parameters **dhisUser** – DHIS2 username for account posting the event

Returns DHIS2 event

Return type dict

openva_pipeline.dhis.**create_db**(*fName*, *evaList*)
Create a SQLite database with VA data + COD

Parameters **evaList** (*list*) – Event-Value-Attribute data structure with verbal autopsy data, cause of death result, and VA metadata.

Return type None

openva_pipeline.dhis.**getCODCode**(*myDict*, *searchFor*)
Return COD label expected by (DHIS2) VA Program.

Parameters **searchFor** (*string*) – Cause of Death label returned by openVA.

Return type str

openva_pipeline.dhis.**findKeyValue**(*key*, *d*)
Return a key's value in a nested dictionary.

7.7 Exceptions

exception openva_pipeline.exceptions.**PipelineError**
Base class for exceptions in the openva_pipeline module.

exception openva_pipeline.exceptions.**DatabaseConnectionError**
An error occurred connecting to the Transfer database.

exception openva_pipeline.exceptions.**PipelineConfigurationError**
An error occurred accessing the Pipeline_Conf table in the DB.

exception openva_pipeline.exceptions.**ODKConfigurationError**
An error occurred accessing the ODK_Conf table in the DB.

exception openva_pipeline.exceptions.**OpenVAConfigurationError**
An error occurred accessing the OpenVA_Conf table in the DB.

exception openva_pipeline.exceptions.**DHISConfigurationError**
An error occurred accessing the DHIS_Conf table in the DB.

exception openva_pipeline.exceptions.**ODKError**
An error occurred with the odk module.

exception openva_pipeline.exceptions.**OpenVAError**
An error occurred with the openVA module.

exception openva_pipeline.exceptions.**SmartVAError**
An error occurred with the openVA module.

exception openva_pipeline.exceptions.**DHISError**
An error occurred with the dhis module.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index