DVH Analytics

Release 0.9.7

Dan Cutright

May 21, 2021

CONTENTS

The documentation here is largely geared towards advanced uses of DVHA involving custom scripts with python. Basic use of the graphical user interface is documented in the user manual PDFs included with each release on GitHub.

CHAPTER

ONE

DVH ANALYTICS

DVH Analytics (DVHA) is a software application for building a local database of radiation oncology treatment planning data. It imports data from DICOM-RT files (*i.e.*, plan, dose, and structure), creates a SQL database, provides customizable plots, and provides tools for generating linear, multi-variable, and machine learning regressions.

1.1 Documentation

Be sure to check out the latest release for the user manual PDF, which is geared towards the user interface. For power-users, dvha.readthedocs.io contains detailed documentation for backend tools (*e.g.*, if you want to perform queries with python commands).

1.2 Executables

Executable versions of DVHA can be found here. Please keep in mind this software is still in beta. If you have issues, compiling from source may be more informative.

1.3 About

In addition to viewing DVH data, this software provides methods to:

- download queried data
- create time-series plots of various planning and dosimetric variables
- calculate correlations
- generate multi-variable linear and machine learning regressions
- share regression models with other DVHA users
- additional screenshots available here

The code is built with these core libraries:

- wxPython Phoenix - Build a native GUI on Windows, Mac, or Unix systems
- Pydicom - Read, modify and write DICOM files with python code

- dicompyler-core - A library of core radiation therapy modules for DICOM RT
- Bokeh - Interactive Web Plotting for Python
- scikit-learn - Machine Learning in Python

1.4 Installation

To install via pip:

```
$ pip install dvha
```

If you've installed via pip or setup.py, launch from your terminal with:

```
$ dvha
```

If you've cloned the project, but did not run the setup.py installer, launch DVHA with:

```
$ python dvha_app.py
```

See our installation notes for potential Shapely install issues on MS Windows and help setting up a PostgreSQL database if it is preferred over SQLite3.

1.5 Dependencies

- Python >3.5
- wxPython Phoenix >= 4.0.4, < 4.1.0
- Pydicom >=1.4.0
- dicompyler-core >= 0.5.4
- Bokeh >= 1.2.0, < 2.0.0
- PostgreSQL (optional) and psycopg2
- SQLite3
- SciPy
- NumPy
- Shapely < 1.7.0
- Statsmodels >=0.8.0
- Scikit-image
- Scikit-learn >= 0.21.0
- regressors
- RapidFuzz
- selenium
- PhantomJS
- DVHA MLC Analyzer

1.6 Support

If you like DVHA and would like to support our mission, all we ask is that you cite us if we helped your publication, or help the DVHA community by submitting bugs, issues, feature requests, or solutions on the issues page.

1.7 Cite

DOI: https://doi.org/10.1002/acm2.12401 Cutright D, Gopalakrishnan M, Roy A, Panchal A, and Mittal BB. "DVH Analytics: A DVH database for clinicians and researchers." Journal of Applied Clinical Medical Physics 19.5 (2018): 413-427.

The previous web-based version described in the above publication can be found here but is no longer being developed.

1.8 Related Publications

DOI: http://doi.org/10.1002/mp.14795 Roy A, Widjaja R, Wang M, Cutright D, Gopalakrishnan M, Mittal BB. "Treatment plan quality control using multivariate control charts." Medical Physics. (2021).

DOI: https://doi.org/10.1016/j.adro.2019.11.006 Roy A, Cutright D, Gopalakrishnan M, Yeh AB, and Mittal BB. "A Risk-Adjusted Control Chart to Evaluate IMRT Plan Quality." Advances in Radiation Oncology (2019).

1.9 Selected Studies Using DVHA

5,000 Patients National Cancer Institute (5R01CA219013-03): Active 8/1/17 → 7/31/22 Retrospective NCI Phantom-Monte Carlo Dosimetry for Late Effects in Wilms Tumor Brannigan R (Co-Investigator), Kalapurakal J (PD/PI), Kazer R (Co-Investigator)

265 Patients DOI: https://doi.org/10.1016/j.ijrobp.2019.06.2509 Gross J, et al. "Determining the organ at risk for lymphedema after regional nodal irradiation in breast cancer." International Journal of Radiation Oncology* Biology* Physics 105.3 (2019): 649-658.

CHAPTER

TWO

USAGE

2.1 Database Connection

Assuming you've setup a successful connection and imported data through the GUI, you can connect to the SQL database with the DVH_SQL class object. This level of interaction with DVHA requires basic knowledge of SQL. Refer to *Data Dictionary* for table and column names.

Below is an example SQL statement requesting a table with column headers of mrn, roi_name, and dvh_string such that the physician_roi is 'brainstem'.

```
SELECT mrn, roi_name, dvh_string FROM DVHs WHERE physician_roi = 'brainstem';
```

The equivalent code in python:

```
from dvha.db.sql_connector import DVH_SQL

table = 'DVHs'
columns = 'mrn, roi_name, dvh_string'
condition = "physician_roi = 'brainstem'"

with DVH_SQL() as cnx:
    mandible = cnx.query(table, columns, condition)
```

The `with` block is equivalent to:

```
cnx = DVH_SQL()
mandible = cnx.query(table, columns, condition)
cnx.close()
```

If no parameters are provided to DVH_SQL, it will automatically pick up your Group 1 connection settings last used in the GUI. See the *dvha.db.sql_connector.DVH_SQL()* documentation for custom connection settings.

2.2 QuerySQL

Use *dvha.db.sql_to_python.QuerySQL()* if you'd like to query a table and automatically convert the results into a python object more convenient than a list of lists (as in DVH_SQL.query). The equivalent of the previous example, using QuerySQL is below:

```
from dvha.db.sql_to_python import QuerySQL
table = 'DVHs'
columns = ['mrn', 'roi_name', 'dvh_string']
condition = "physician_roi = 'brainstem'"
data = QuerySQL(table, condition, columns=columns)
```

Or if you'd like to just pick up all columns:

```
data = QuerySQL(table, condition)
```

QuerySQL automatically adds properties based on the column name. So the *mrns* are accessible with *data.mrns*, *roi_name* with *data.roi_name*, etc. This works with DVHs, Plans, Rxs, and Beams tables.

2.3 DVH

Some data you may want with each of your DVHs is spread across multiple tables. The *dvha.models.dvh.DVH()* object uses *dvha.db.sql_to_python.QuerySQL()* to query the database, adds some pertinent data from other tables, and provides many commonly used functions dealing with DVHs.

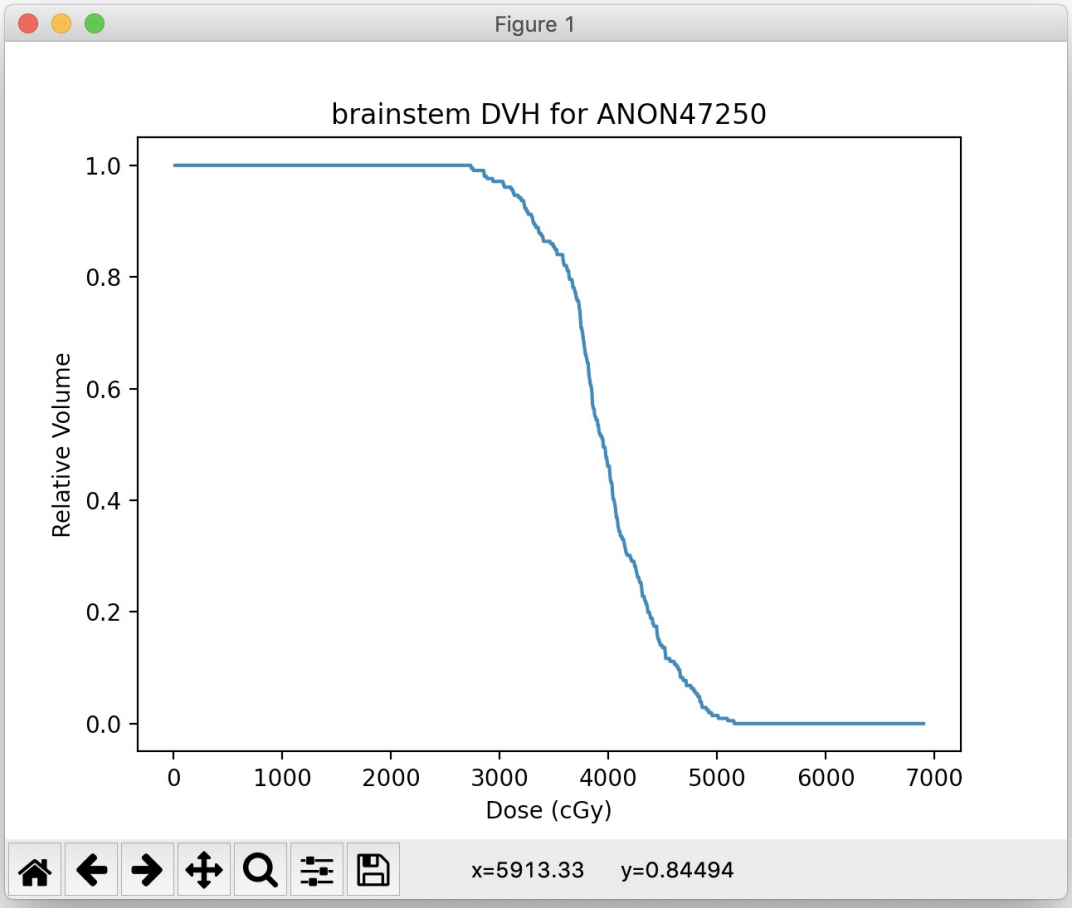
The equivalent of the previous example, using DVH is below:

```
from dvha.models.dvh import DVH
dvh = DVH(dvh_condition="physician_roi = 'brainstem'")
```

2.4 DVH Plotting

```
# Collect the plotting data
i = 0  # change this index to pick a different DVH
x = dvh.x_data[i]
y = dvh.y_data[i]
mrn = dvh.mrn[i]
roi_name = dvh.roi_name[i]
title = '%s DVH for %s' % (roi_name, mrn)

# Create the plot, may need to call plt.show() on some setups
import matplotlib.pyplot as plt
plt.plot(x, y)
plt.title(title)
plt.xlabel('Dose (cGy)')
plt.ylabel('Relative Volume')
```

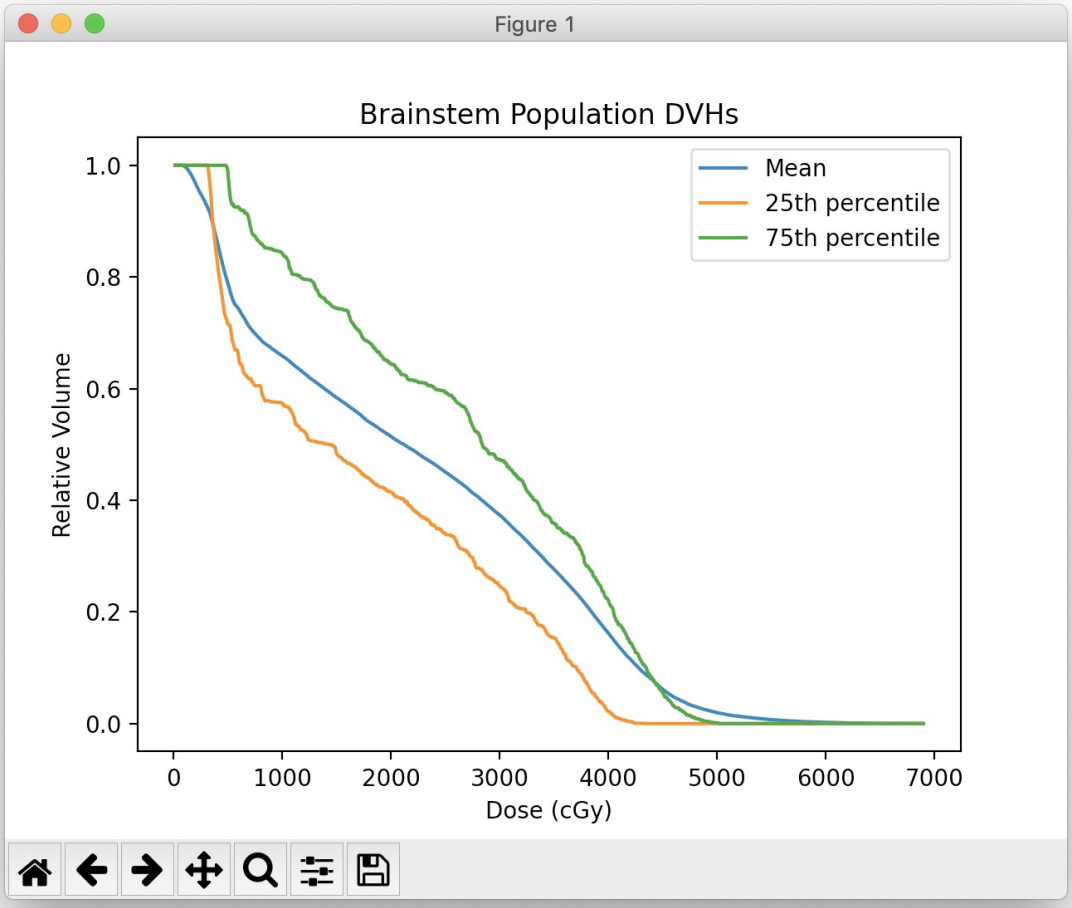


2.5 Population DVH

```
x = dvh.x_data[0]
mean = dvh.get_stat_dvh('mean')
q1 = dvh.get_percentile_dvh(25)
q3 = dvh.get_percentile_dvh(75)

plt.plot(x, mean, label='Mean')
plt.plot(x, q1, label='25th percentile')
plt.plot(x, q3, label='75th percentile')

plt.title('Brainstem Population DVHs')
plt.xlabel('Dose (cGy)')
plt.ylabel('Relative Volume')
plt.legend()
```

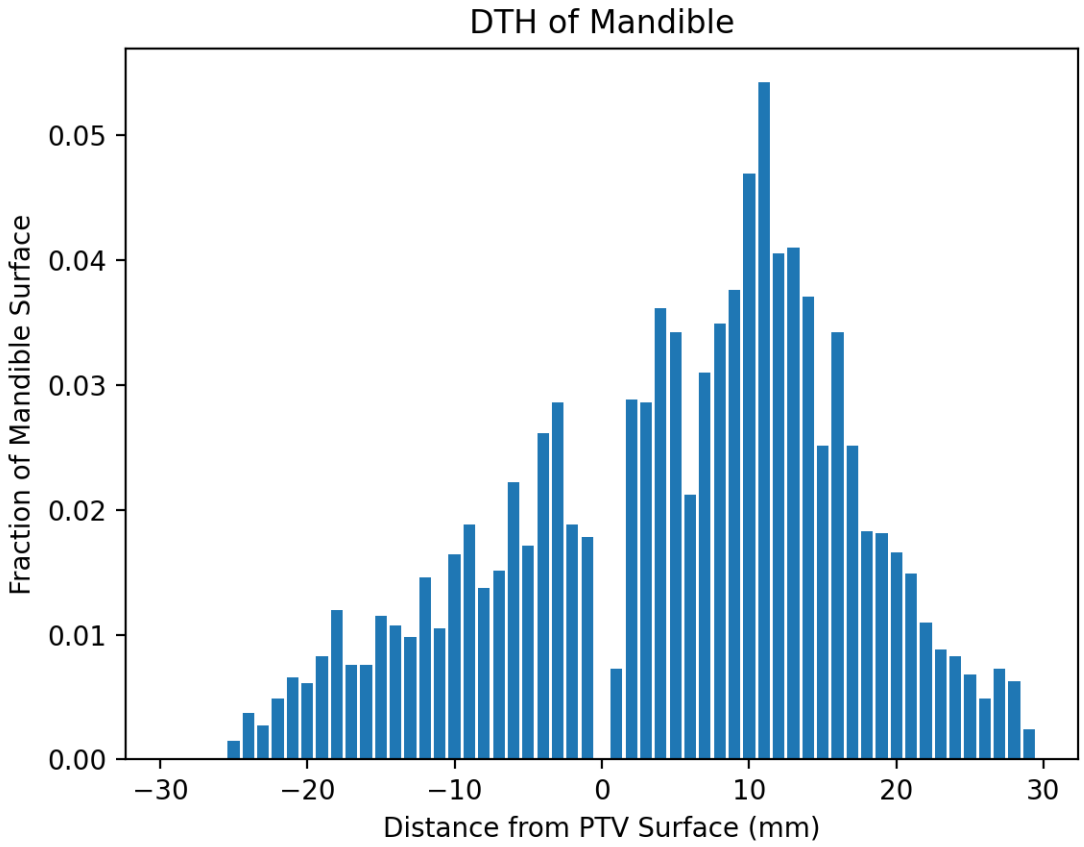


2.6 DTH

Although not accessible in the GUI or DVHA session data, DTHs can be extracted from the database as shown below.

```
from dvha.db.sql_connector import DVH_SQL
import matplotlib.pyplot as plt
from dvha.tools.roi_geometry import process_dth_string
with DVH_SQL() as cnx:
    condition = "mrn = 'ANON11264' and physician_roi = 'mandible'"
    mandible = cnx.query('DVHs', 'dth_string', condition)[0][0]
bins, counts = process_dth_string(mandible)

figure, axes = plt.subplots()
axes.bar(bins, counts)
axes.set_title('DTH of Mandible')
axes.set_xlabel('Distance from PTV Surface (mm)')
axes.set_ylabel('Fraction of Mandible Surface')
```


DTH of Mandible
Fraction of Mandible Surface
0.00
0.01
0.02
0.03
0.04
0.05
−30
−20
−10
0
10
20
30
Distance from PTV Surface (mm)

CHAPTER

THREE

DVH ANALYTICS

3.1 dvha.models.dvh

Class to retrieve DVH data from SQL, calculate parameters dependent on DVHs, and extract plotting data

class dvha.models.dvh.**DVH**(*uid=None*, *dvh_condition=None*, *dvh_bin_width=5*, *group=1*)

Bases: object

This class will retrieve DVHs and other data in the DVH SQL table meeting the given constraints, it will also parse the DVH_string into python lists and retrieve the associated Rx dose

Parameters

- **uid** (*list*) – study_instance_uid's to be included in results
- **dvh_condition** (*str*) – a string in SQL syntax applied to a DVH Table query
- **dvh_bin_width** (*int*) – retrieve every nth value from dvh_string in SQL
- **group** (*int*) – either 1 or 2

dvhs_to_abs_vol(*dvhs*)

Get DVHs in absolute volume

Parameters **dvhs** (*np.ndarray*) – relative DVHs (dvh[bin, roi_index])

Returns absolute DVHs

Return type np.ndarray

get_cds_data(*keys=None*)

Get data from this class in a format compatible with bokeh's ColumnDataSource.data

Parameters **keys** (*list, optional*) – Specify a list of DVH properties to be included

Returns data from this class

Return type dict

get_dose_to_volume(*volume*, *volume_scale='absolute'*, *dose_scale='absolute'*, *compliment=False*)

Get the minimum dose to a specified volume

Parameters

- **volume** (*int, float*) – the specified volume in cm^3
- **volume_scale** (*str, optional*) – either 'relative' or 'absolute'
- **dose_scale** (*str, optional*) – either 'relative' or 'absolute'
- **compliment** (*bool, optional*) – return the max dose - value

Returns the dose in Gy to the specified volume

Return type list

get_percentile_dvh(*percentile*)
Get a population DVH based on percentile

Parameters **percentile** (*float*) – the percentile to calculate for each dose-bin. Passed into np.percentile()

Returns a single DVH such that each bin is the given percentile of each bin over the whole sample

Return type np.ndarray

get_plan_values(*plan_column*)
Get values from the Plans table and store in order matching mrn / study_instance_uid

Parameters **plan_column** (*str*) – name of the SQL column to be returned

Returns values from the Plans table for the DVHs stored in this class

Return type list

get_resampled_x_axis(*resampled_bin_count=5000*)
Get the x_axis of a resampled dvh

Parameters **resampled_bin_count** (*int, optional*) – Specify the number of bins used

Returns the x axis of a resampled dvh

Return type np.ndarray

get_rx_values(*rx_column*)
Get values from the Rxs table and store in order matching mrn / study_instance_uid

Parameters **rx_column** (*str*) – name of the SQL column to be returned

Returns values from the Rxs table for the DVHs stored in this class

Return type list

get_standard_stat_dvh(*dose_scale='absolute'*, *volume_scale='relative'*)
Get a min, mean, median, max, and quartiles DVHs

Parameters

- **dose_scale** (*str, optional*) – either 'absolute' or 'relative'
- **volume_scale** (*str, optional*) – either 'absolute' or 'relative'

Returns a standard set of statistical dvhs ('min', 'q1', 'mean', 'median', 'q1', and 'max')

Return type dict

get_stat_dvh(*stat_type='mean'*, *dose_scale='absolute'*, *volume_scale='relative'*)
Get population DVH by statistical function

Parameters

- **stat_type** (*str*) – either 'min', 'mean', 'median', 'max', or 'std'
- **dose_scale** (*str, optional*) – either 'absolute' or 'relative'
- **volume_scale** (*str, optional*) – either 'absolute' or 'relative'

Returns a single dvh where each bin is the stat_type of each bin for the entire sample

Return type np.ndarray

get_summary()
Get a summary of the data in this class. Used in bottom left of main GUI

Returns Summary data

Return type str

get_volume_of_dose(*dose*, *dose_scale='absolute'*, *volume_scale='absolute'*, *compliment=False*)
Get the volume of an isodose contour defined by dose

Parameters

- **dose** (*int, float*) – input dose use to calculate a volume of dose for entire sample
- **dose_scale** (*str, optional*) – either 'absolute' or 'relative'
- **volume_scale** (*str, optional*) – either 'absolute' or 'relative'
- **compliment** (*bool, optional*) – return the ROI volume - value

Returns a list of V_dose

Return type list

property has_data
Check that this class has queried data

Returns True if there is data returned from SQL

Return type bool

resample_dvh(*resampled_bin_count=5000*)
Resample DVHs with a new bin count

Parameters **resampled_bin_count** (*int*) – Number of bins in reampled DVH

Returns x-axis, y-axis of resampled DVHs

Return type tuple

property x_axis
Get the x-axis for plotting

Returns Dose axis

Return type list

property x_data
Get the x-axes for plotting (one row per DVH)

Returns x data for plotting

Return type list

property y_data
Get y-values of the DVHs

Returns all DVHs in order (i.e., same as mrn, study_instance_uid)

Return type list

dvha.models.dvh.**calc_eud**(*dvh*, *a*, *dvh_bin_width=1*)
EUD = sum[v(i) * D(i)^a] ^ [1/a]

Parameters

- **dvh** (*np.ndarray*) – a single DVH as a list of numpy 1D array with 1cGy bins
- **a** (*float*) – standard a-value for EUD calculations, organ and dose fractionation specific

- **dvh_bin_width** (*int, optional*) – dose bin width of dvh

Returns equivalent uniform dose

Return type float

dvha.models.dvh.**calc_tcp**(*gamma*, *td_tcd*, *eud*)
Tumor Control Probability / Normal Tissue Complication Probability 1.0 / (1.0 + (td_tcd / eud) ^ (4.0 * gamma))

Parameters

- **gamma** (*float*) – Gamma_50
- **td_tcd** (*float*) – Either TD_50 or TCD_50
- **eud** (*float*) – equivalent uniform dose

Returns TCP or NTCP

Return type float

dvha.models.dvh.**dose_to_volume**(*dvh*, *rel_volume*, *dvh_bin_width=1*)
Calculate the minimum dose to a relative volume for one DVH

Parameters

- **dvh** (*np.ndarray*) – a single dvh
- **rel_volume** (*float*) – fractional volume
- **dvh_bin_width** (*int, optional*) – dose bin width of dvh

Returns minimum dose in Gy of specified volume

Return type float

dvha.models.dvh.**volume_of_dose**(*dvh*, *dose*, *dvh_bin_width=1*)
Calculate the relative volume of an isodose line definted by dose for one DVH

Parameters

- **dvh** (*np.ndarray*) – a single dvh
- **dose** (*float*) – dose in cGy
- **dvh_bin_width** (*int, optional*) – dose bin width of dvh

Returns volume in cm^3 of roi receiving at least the specified dose

Return type float

3.2 dvha.db

3.2.1 SQL Connector

Tools used to communicate with the SQL database

class dvha.db.sql_connector.**DVH_SQL**(**config*, *db_type=None*, *group=1*)
Bases: object

This class is used to communicate to the SQL database

Parameters

- **config** (*dict*) – optional SQL login credentials, stored values used if nothing provided. Allowed keys are 'host', 'port', 'dbname', 'user', 'password'
- **db_type** (*str, optional*) – either 'pgsql' or 'sqlite'
- **group** (*int, optional*) – use a group-specific connection, either 1 or 2

change_mrn(*old*, *new*)
Edit all mrns in database

Parameters

- **old** (*str*) – current mrn
- **new** (*str*) – new mrn

change_uid(*old*, *new*)
Edit study instance uids in database

Parameters

- **old** (*str*) – current study instance uid
- **new** (*str*) – new study instance uid

check_table_exists(*table_name*)
Check if a table exists

Parameters **table_name** (*st*) – the SQL table to check

Returns True if table_name exists

Return type bool

close()
Close the SQL DB connection

delete_dvh(*roi_name*, *study_instance_uid*)
Delete a specified DVHs table row

Parameters

- **roi_name** (*str*) – the roi name for the row to be deleted
- **study_instance_uid** (*str*) – the associated study instance uid

delete_rows(*condition_str*, *ignore_tables=None*)
Delete all rows from all tables not in ignore_table for a given condition. Useful when deleting a plan/patient

Parameters

- **condition_str** (*str: str*) – a condition in SQL syntax
- **ignore_tables** (*list, optional*) – tables to be excluded from row deletions

does_db_exist()
Check if database exists

Returns True if the database exists

Return type bool

drop_table(*table*)
Delete a table in the database if it exists

Parameters **table** (*str*) – SQL table

drop_tables()
Delete all tables in the database if they exist

execute_file(*sql_file_name*)
Executes lines within provided text file to SQL

Parameters **sql_file_name** (*str*) – absolute file path of a text file containing SQL commands

execute_str(*command_str*)
Execute and commit a string in proper SQL syntax, can handle multiple lines split by

Parameters **command_str** (*str*) – command or commands to be executed and committed

export_to_sqlite(*file_path*, *callback=None*, *force=False*)
Create a new SQLite database and import this database's data

Parameters

- **file_path** (*str*) – Path where the new SQLite database will be created
- **callback** (*callable, optional*) – optional function to be called on each row insertion. Should accept table (str), current row (int), total_row_count (int) as parameters
- **force** (*bool, optional*) – ignore duplicate StudyInstanceUIDs if False

get_column_names(*table_name*)
Get all of the column names for a specified table

Parameters **table_name** (*str*) – SQL table

Returns All columns names in `table_name`

Return type list

get_dicom_file_paths(*mrn=None*, *uid=None*)
Lookup the dicom file paths of imported data

Parameters

- **mrn** (*str, optional*) – medical record number
- **uid** (*str, optional*) – study instance uid

Returns Query return from DICOM_Files table

Return type list

get_max_value(*table*, *column*, *condition=None*)
Get the maximum value in the database for a given table and column

Parameters

- **table** (*str*) – SQL table
- **column** (*str*) – SQL column
- **condition** (*str, optional*) – Condition in SQL syntax

Returns The maximum value for table.column

Return type any

get_min_value(*table*, *column*, *condition=None*)
Get the minimum value in the database for a given table and column

Parameters

- **table** (*str*) – SQL table

- **column** (*str*) – SQL column
- **condition** (*str, optional*) – Condition in SQL syntax

Returns The minimum value for table.column

Return type any

`get_ptv_counts`()
Get number of PTVs for each study instance uid

Returns PTV counts stored by `study_instance_uid`

Return type dict

`get_roi_count_from_query`(*uid=None*, *dvh_condition=None*)
Counts the DVH rows that match the provided conditions

Parameters

- **uid** (*str, optional*) – study instance uid
- **dvh_condition** (*str, optional*) – condition in SQL syntax for the DVHs table

Returns Number of rows in the DVHs table matching the provided parameters

Return type int

`get_row_count`(*table*, *condition=None*)

Parameters

- **table** (*str*) – SQL table
- **condition** (*str*) – SQL condition

Returns Number of rows in `table` meeting `condition`

Return type int

`get_sql_function_value`(*func*, *table*, *column*, *condition=None*, *first_value_only=True*)
Helper function used by get_min_values and get_max_values

Parameters

- **func** – SQL compatible function
- **table** (*str*) – SQL table
- **column** (*str*) – SQL column
- **condition** (*str, optional*) – Condition in SQL syntax (Default value = None)
- **first_value_only** (*bool, optional*) – if true, only return the first value, otherwise all values returned

Returns Results of `cursor.fetchone()` or `cursor.fetchone()[0]` if `first_value_only` is True

Return type list, any

`get_unique_values`(*table*, *column*, **condition*, ***kwargs*)
Uses SELECT DISTINCT to get distinct values in database

Parameters

- **table** (*str*) – SQL table

- **column** (*str*) – SQL column
- **condition** (*str, optional*) – Condition in SQL syntax
- **kwargs** – option to ignore null values in return

Returns Unique values in table.column

Return type list

ignore_dvh(*variation*, *study_instance_uid*, *unignore=False*)

Change an uncategorized roi name to ignored so that it won't show up in the list of uncategorized rois, so that the user doesn't have to evaluate its need everytime they cleanup the misc rois imported

Parameters

- **variation** (*str*) – roi name
- **study_instance_uid** (*str*) – the associated study instance uid
- **unignore** (*bool, optional*) – if set to True, sets the variation to 'uncategorized'

static import_db(*cnx_src*, *cnx_dst*, *callback=None*, *force=False*)

Parameters

- **cnx_src** (DVH_SQL) – the source DVHA DB connection
- **cnx_dst** (DVH_SQL) – the destination DVHA DB connection
- **callback** (*callable, optional*) – optional function to be called on each row insertion. Should accept table (str), current row (int), total_row_count (int) as parameters
- **force** (*bool, optional*) – ignore duplicate StudyInstanceUIDs if False

initialize_database()

Ensure that all of the latest SQL columns exist in the database

insert_data_set(*data_set*)

Insert an entire data set for a plan

Parameters **data_set** (*dict*) – a dictionary of data with table names for keys, and a list of row data for values

insert_row(*table*, *row*)

Generic function to import data to the database

Parameters

- **table** (*str*) – SQL table name
- **row** (*dict*) – data returned from DICOM_Parser.get_<table>_row()

is_mrn_imported(*mrn*)

Check all tables to see if MRN is used

Parameters **mrn** (*str*) – medical record number

Returns True if mrn exists in any table

Return type bool

is_mrn_in_table(*table_name*, *mrn*)

Parameters

- **table_name** (*str*) – SQL table name
- **mrn** (*str*) – medical record number

Returns True if mrn exists in the mrn column of table_name

Return type bool

is_roi_imported(*roi_name*, *study_instance_uid*)
Check if a study is already using a specified roi name

Parameters

- **roi_name** (*str*) – roi name to check
- **study_instance_uid** (*str*) – restrict search to this study_instance_uid

Returns True if roi_name is used in the DVHs table for the given study_instance_uid

Return type bool

is_sql_table_empty(*table*)
Check if specified SQL table is empty

Parameters **table** (*str*) – SQL table

Returns True if table is empty

Return type bool

is_sqlite_column_datetime(*table_name*, *column*)
Check if a sqlite column is a datetime data type

Parameters

- **table_name** (*str*) – SQL table
- **column** (*str*) – SQL column

Returns True if the table_name.column store datetime data

Return type bool

is_study_instance_uid_in_table(*table_name*, *study_instance_uid*)
Check if a study instance uid exists in the provided table

Parameters

- **table_name** (*str*) – SQL table name
- **study_instance_uid** (*str*) – study instance uid

Returns True if study_instance_uid exists in the study_instance_uid column of table_name

Return type bool

is_uid_imported(*uid*)
Check all tables to see if study instance uid is used

Parameters **uid** (*str*) – study instance uid

Returns True if uid exists in any table

Return type bool

is_value_in_table(*table_name*, *value*, *column*)
Check if a str value exists in a SQL table

Parameters

- **table_name** (*sr*) – SQL table name
- **value** (*str*) – value of interest (str only)
- **column** – SQL table column

Returns True if `value` exists in `table_name.column`

Return type bool

property now
Get a datetime object for now

Returns The current time reported by the database

Return type datetime

process_value(*value*)

query(*table_name*, *return_col_str*, **condition_str*, ***kwargs*)
A generalized query function for DVHA

Parameters

- **table_name** (*str*) – DVHs', 'Plans', 'Rxs', 'Beams', or 'DICOM_Files'
- **return_col_str** (*str: str*) – a csv of SQL columns to be returned
- **condition_str** (*str: str*) – a condition in SQL syntax
- **kwargs** – optional parameters order, order_by, and bokeh_cds

Returns Returns a list of lists by default, or a dict of lists if bokeh_cds in kwargs and is true

Return type list, dict

query_generic(*query_str*)
A generic query function that executes the provided string

Parameters **query_str** (*str*) – SQL command

Returns Results of `cursor.fetchall()`

Return type list

reinitialize_database()
Delete all data and create all tables with latest columns

save_to_json(*file_path*, *callback=None*)
Export SQL database to a JSON file

Parameters

- **file_path** (*str*) – file_path to new JSON file
- **callback** (*callable, optional*) – optional function to be called on each table insertion. Should accept table_name (str), table (int), table_count (int) as parameters

property sql_cmd_now
Get the SQL command for now, based on database type

Returns SQL command for now

Return type str

update(*table_name*, *column*, *value*, *condition_str*)
Change the data in the database

Parameters

- **table_name** (*str*) – DVHs', 'Plans', 'Rxs', 'Beams', or 'DICOM_Files'
- **column** (*str*) – SQL column to be updated
- **value** (*str, float, int*) – value to be set
- **condition_str** (*str*) – a condition in SQL syntax

update_multicolumn(*table_name*, *columns*, *values*, *condition_str*)
Change the data in the database

Parameters

- **table_name** (*str*) – DVHs', 'Plans', 'Rxs', 'Beams', or 'DICOM_Files'
- **columns** (*list*) – list of SQL column to be updated
- **values** (*list*) – list value to be set
- **condition_str** (*str*) – a condition in SQL syntax

vacuum()
Call to reclaim space in the database

dvha.db.sql_connector.**echo_sql_db**(*config=None*, *db_type='pgsql'*, *group=1*)
Echo the database using stored or provided credentials

Parameters

- **config** (*dict, optional*) – database login credentials
- **db_type** (*str, optional*) – either 'pgsql' or 'sqlite'
- **group** (*int, optional*) – either group 1 or 2

Returns True if echo is successful

Return type bool

dvha.db.sql_connector.**initialize_db**()
Initialize the database

dvha.db.sql_connector.**is_file_sqlite_db**(*sqlite_db_file*)
Check if file is a sqlite database

Parameters **sqlite_db_file** (*str*) – path to file to be checked

Returns True if sqlite_db_file is a sqlite database

Return type bool

dvha.db.sql_connector.**truncate_string**(*input_string*, *character_limit*)
Used to truncate a string to ensure it may be imported into database

Parameters

- **input_string** (*str*) – string to be truncated
- **character_limit** (*int*) – the maximum number of allowed characters

Returns truncated string

Return type str

dvha.db.sql_connector.**write_test**(*config=None*, *db_type='pgsql'*, *group=1*, *table=None*, *column=None*, *value=None*)
Write test data to database, verify with a query

Parameters

- **config** (*dict, optional*) – database login credentials
- **db_type** (*str, optional*) – either 'pgsql' or 'sqlite'
- **group** (*int, optional*) – either group 1 or 2
- **table** (*str, optional*) – SQL table
- **column** (*str, optional*) – SQL column
- **value** (*str, optional*) – test value

Returns Write and Read test statuses

Return type dict

3.2.2 SQL to Python

Query a DVHA SQL table and parse the return into a python object

class dvha.db.sql_to_python.**QuerySQL**(*table_name*, *condition_str*, *unique=False*, *columns=None*, *group=1*)

Bases: `object`

Object to generically query a specified table. Each column is stored as a property of the object

For example, if you query 'dvhs' with condition string of "mrn = 'some_mrn'" you can access any column name 'some_column' with QuerySQL.some_column which will return a list of values for 'some_column'. All properties contain lists with the order of their values synced, unless unique=True

Parameters

- **table_name** (*str*) – Beams', 'DVHs', 'Plans', or 'Rxs'
- **condition_str** (*str*) – condition in SQL syntax
- **unique** (*bool, optional*) – If True, only unique values stored
- **columns** (*list, optional*) – A list of SQL column names to be included in the return. If left as `None`, all columns will be returned
- **group** (*int, optional*) – either 1 or 2

cursor_to_list(*force_date=False*)

Convert a cursor return into a list of values

Parameters **force_date** (*bool, optional*) – Apply dateutil.parser to values

Returns queried data

Return type list

dvha.db.sql_to_python.**get_database_tree**()

Query SQL to get all columns of each table

Returns column data sorted by table

Return type dict

dvha.db.sql_to_python.**get_unique_list**(*input_list*)

Remove duplicates in list and retain order

Parameters **input_list** (*list*) – any list of objects

Returns input_list without duplicates

Return type list

3.2.3 DB Updater

3.2.4 DICOM Parser

3.3 dvha.tools

3.3.1 Name Prediction

Implementation of rapidfuzz for ROI name prediction

class dvha.tools.name_prediction.**ROINamePredictor**(*roi_map*, *weight_simple=1.0*, *weight_partial=0.6*, *threshold=0.0*)

Bases: `object`

ROI Name Prediction class object

Parameters

- **roi_map** (*DatabaseROIs*) – ROI map object
- **weight_simple** (*float, optional*) – Scaling factor for fuzz.ratio for combined score
- **weight_partial** (*float, optional*) – Scaling factor for fuzz.partial_ratio for combined score
- **threshold** (*float, optional*) – Set a minimum score for a prediction to be returned

static combine_scores(*score_1*, *score_2*, *mode='average'*)

Get a combined fuzz score

Parameters

- **score_1** (*float*) – A fuzz ratio score
- **score_2** (*float*) – Another fuzz ratio score
- **mode** (*str, optional*) – Method for combining `score_1` and `score_2`. Options are 'geom_mean', 'product', and 'average'

Returns Combined score

Return type float

get_best_roi_match(*roi*, *physician*, *return_score=False*)

Check all ROI variations for best match, return physician ROI

Parameters

- **roi** (*str*) – An ROI name
- **physician** (*str*) – Physician as stored in ROI Map
- **return_score** (*bool, optional*) – If true, return a tuple: prediction, score

Returns The physician ROI associated with the ROI variation that is has the highest combined fuzz score for `roi`

Return type str

get_combined_fuzz_score(*a*, *b*, *mode='geom_mean'*)

Return `combine_scores` for strings `a` and `b`

Parameters

- **a** (*str*) – Any string
- **b** (*str*) – Another string for comparison
- **mode** (*str, optional*) – Method for combining `fuzz.ratio` and `fuzz.partial_ratio`. Options are 'geom_mean', 'product', and 'average'

Returns Results from `combine_scores` for `a` and `b`

Return type float

get_combined_fuzz_scores(*string*, *list_of_strings*)
Compare a string against many

Parameters

- **string** (*str*) – A string to compare against each string in `list_of_strings`
- **list_of_strings** (*list*) – A list of strings for comparison

Returns A list of tuples (score, string) in order of score

Return type list

3.3.2 MLC Analyzer

The code for DVHA's MLC analysis has been exported to a stand-alone library.

- GitHub: mlca.dvhanalytics.com
- Docs: dvha-mlca.readthedocs.io

3.3.3 ROI Formatter

Formatting tools for roi data (dicompyler, Shapely, DVHA)

dvha.tools.roi_formatter.**dicompyler_roi_coord_to_db_string**(*coord*)

Parameters **coord** – dicompyler structure coordinates from GetStructureCoordinates()

Returns roi string representation of an roi as formatted in the SQL database (roi_coord_string)

Return type str

dvha.tools.roi_formatter.**dicompyler_roi_to_sets_of_points**(*coord*)

Parameters **coord** – dicompyler structure coordinates from GetStructureCoordinates()

Returns a "sets of points" formatted dictionary

Return type dict

dvha.tools.roi_formatter.**get_contour_sample**(*polygon*, *dth_res=0.5*) → tuple
Get 3D points uniformly distributed in the perimeter space

Parameters

- **polygon** (*Polygon*) – shapely object
- **dth_res** (*int, float*) – Sampling distance in perimeter space (mm)

Returns

- *np.ndarray* – x coordinates of sampled contour
- *np.ndarray* – y coordinates of sampled contour

dvha.tools.roi_formatter.**get_planes_from_string**(*roi_coord_string*)

Parameters **roi_coord_string** (*string: str*) – roi string representation of an roi as formatted in the SQL database

Returns a "sets of points" formatted dictionary

Return type dict

dvha.tools.roi_formatter.**get_roi_coordinates_from_planes**(*sets_of_points*)

Parameters **sets_of_points** (*dict*) – a "sets of points" formatted dictionary

Returns a list of numpy arrays, each array is the x, y, z coordinates of the given point

Return type list

dvha.tools.roi_formatter.**get_roi_coordinates_from_shapely**(*shapely_dict*, *sample_res=None*)

Parameters

- **shapely_dict** (*dict*) – output from get_shapely_from_sets_of_points
- **sample_res** (*int, float*) – If set to a numeric value, sample each polygon with this resolution (mm)

Returns a list of numpy arrays, each array is the x, y, z coordinates of the given point

Return type list

dvha.tools.roi_formatter.**get_roi_coordinates_from_string**(*roi_coord_string*)

Parameters **roi_coord_string** (*string: str*) – roi string representation of an roi as formatted in the SQL database

Returns a list of numpy arrays, each array is the x, y, z coordinates of the given point

Return type list

dvha.tools.roi_formatter.**get_shapely_from_sets_of_points**(*sets_of_points*, *tolerance=None*, *preserve_topology=True*)

Parameters

- **sets_of_points** (*dict*) – a "sets of points" formatted dictionary
- **tolerance** (*bool, optional*) – If set to a number, will use Shapely's simplify on each contour with the given tolerance
- **preserve_topology** (*bool, optional*) – Passed to Shapely's simplify if `simplify_tolerance` is set

Returns roi_slices which is a dictionary of lists of z, thickness, and a Shapely Polygon class object

Return type dict

dvha.tools.roi_formatter.**points_to_shapely_polygon**(*sets_of_points*)

Parameters **sets_of_points** (*dict*) – a "sets of points" formatted dictionary

Returns a composite polygon as a shapely object (either polygon or multipolygon)

Return type type

3.3.4 ROI Geometry

Tools for geometric calculations

dvha.tools.roi_geometry.**centroid**(*roi*)

Parameters **roi** – a "sets of points" formatted dictionary

Returns centroid or the roi in x, y, z dicom coordinates (mm)

Return type list

dvha.tools.roi_geometry.**cross_section**(*roi*)
Calculate the cross section of a given roi

Parameters **roi** (*dict*) – a "sets of points" formatted dictionary

Returns max and median cross-sectional area of all slices in cm^2

Return type dict

dvha.tools.roi_geometry.**dth**(*min_distances*)

Parameters **min_distances** – the output from min_distances_to_target

Returns histogram of distances in 1mm bin widths

Return type numpy.array

dvha.tools.roi_geometry.**is_point_inside_roi**(*point*, *roi*)
Check if a point is within an ROI

Parameters

- **point** (*list*) – x, y, z
- **roi** (*dict*) – roi: a "sets of points" formatted dictionary

Returns Whether or not the point is within the roi

Return type bool

dvha.tools.roi_geometry.**min_distances_to_target**(*oar_coordinates*, *target_coordinates*, *factors=None*)
Calculate all OAR-point-to-Target-point euclidean distances

Parameters

- **oar_coordinates** (*list*) – numpy arrays of 3D points defining the surface of the OAR
- **target_coordinates** (*list*) – numpy arrays of 3D points defining the surface of the PTV
- **factors** – (Default value = None)

Returns min_distances: all minimum distances (cm) of OAR-point-to-Target-point pairs

Return type list

dvha.tools.roi_geometry.**overlap_volume**(*oar*, *tv*)
Calculate the overlap volume of two rois

Parameters

- **oar** (*dict*) – organ-at-risk as a "sets of points" formatted dictionary
- **tv** (*dict*) – treatment volume as a "sets of points" formatted dictionary

dvha.tools.roi_geometry.**planes_to_voxel_centers**(*planes*, *res=1*, *max_progress=None*)
Convert a sets of points into a 3D voxel centers within ROI

Parameters

- **planes** (*dict*) – a "sets of points" dictionary representing the union of the rois
- **res** (*float*) – resolution factor for voxels
- **max_progress** (*float*) – if not None, set the maximum progress bar value (with update_dvh_progress)

Returns A list of 3D points inside the ROI defined by `planes`

Return type list

dvha.tools.roi_geometry.**process_dth_string**(*dth_string*)
Convert a dth_string from the database into data and bins DVHA stores 1-mm binned surface DTHs with an odd number of bins, middle bin is 0.

Parameters **dth_string** – a value from the dth_string column

Returns counts, bin positions (mm)

Return type type

dvha.tools.roi_geometry.**spread**(*roi*)

Parameters **roi** – a "sets of points" formatted dictionary

Returns x, y, z dimensions of a rectangular prism encompassing roi

Return type list

dvha.tools.roi_geometry.**surface_area**(*coord*, *coord_type='dicompyler'*)
Calculate the surface of a given roi

Parameters

- **coord** – dicompyler structure coordinates from GetStructureCoordinates() or a sets_of_points dictionary
- **coord_type** – either 'dicompyler' or 'sets_of_points' (Default value = 'dicompyler')

Returns surface_area in cm^2

Return type float

dvha.tools.roi_geometry.**union**(*rois*)
Calculate the geometric union of the provided rois

Parameters **rois** (*list*) – rois formatted as "sets of points" dictionaries

Returns a "sets of points" dictionary representing the union of the rois

Return type dict

dvha.tools.roi_geometry.**volume**(*roi*)

Parameters **roi** – a "sets of points" formatted dictionary

Returns volume in cm^3 of roi

Return type float

3.3.5 Stats

The code from DVHA's statistical modules have been exported to a stand-alone library, however, DVHA does not use this internally (yet).

- GitHub: stats.dvhanalytics.com
- Docs: dvha-stats.readthedocs.io

Take numerical data from main app and convert to a format suitable for statistical analysis in Regression and Control Chart tabs

class dvha.tools.stats.**MultiVariableRegression**(*X*, *y*, *saved_reg=None*)

Bases: object

Perform a multi-variable regression using sklearn

Parameters

- **X** (*np.array*) – independent data
- **y** (*list*) – dependent data

class dvha.tools.stats.**StatsData**(*dvhs*, *table_data*, *group=1*)

Bases: object

Class used to to collect data for Regression and Control Chart This process is different than for Time Series since regressions require all variables to be the same length

Parameters

- **dvhs** (DVH) – data from DVH query
- **table_data** (*dict*) – table data other than from DVHs. Has keys of 'Plans', 'Rxs', 'Beams' with values of QuerySQL objects

add_variable(*variable*, *values*, *units=''*)

Add a new variable to StatsData.data, will not over-write

Parameters

- **variable** (*str*) – variable name to be used as a key and plot title
- **values** (*list*) – values to be stored for variable
- **units** (*str, optional*) – Define units for display on plot

del_variable(*variable*)

Delete a variable from StatsData.data

Parameters **variable** (*str*) – variable name

get_X_and_y(*y_variable*, *x_variables*, *include_patient_info=False*)

Collect data for input into multi-variable regression

Parameters

- **y_variable** (*str*) – dependent variable
- **x_variables** (*list*) – independent variables
- **include_patient_info** (*bool*) – If True, return mrn, uid, dates with X and y

Returns X, y or X, y, mrn, uid, dates

Return type type

`get_axis_title`(*variable*)
Get the plot axis title for `variable`

Parameters **`variable`** (*str*) – A key of `StatsData.data`

Returns `variable` with units if stored

Return type str

`get_beam_indices`(*uid*)
Get the indices of the Beams table with `uid`

Parameters **`uid`** (*str*) – StudyInstanceUID as stored in the SQL database

Returns Beams table indices that match `uid`

Return type list

`get_bokeh_data`(*x*, *y*)
Get data in a format compatible with bokeh's ColumnDataSource.data

Parameters

- **`x`** (*str*) – x-variable name
- **`y`** (*str*) – y-variable name

Returns x and y data

Return type dict

`get_corr_matrix_data`(*options*, *included_vars=None*, *extra_vars=None*)
Get a Pearson-R correlation matrix

Parameters

- **`options`** (*dvha.options.Options*) – DVHA options class object. Used to get colors.
- **`included_vars`** (*list, optional*) – variables to be included in matrix
- **`extra_vars`** (*list, optional*) – variables to be excluded from the matrix

Returns The dictionary has keys of 'source_data', 'x_factors' and 'y_factors'. *source_data* is used for bokeh plotting, the factors are for axis tick labels. The 2nd parameter of the tuple is a list of removed mrns

Return type tuple (dict, list)

`get_plan_index`(*uid*)
Get the index of `uid` from the Plans table

Parameters **`uid`** (*str*) – StudyInstanceUID as stored in the SQL database

Returns Plans table index for `uid`

Return type int

`property mrns`
MRNs from DVH object

Returns `DVH.mrn`

Return type list

set_variable_data(*variable*, *data*, *units=None*)
Replace the data for the given variable in StatsData.data

Parameters

- **variable** (*str*) – variable name
- **data** (*list*) – new data
- **units** (*str, optional*) – Define units for display on plot

set_variable_units(*variable*, *units*)
Set the units for the given variable in StatsData.data

Parameters

- **variable** (*str*) – variable name
- **units** (*str*) – units for display on plot

property sim_study_dates
Simulation dates from Plans table

Returns Simulation dates

Return type list

property uids
StudyInstanceUIDs from DVH object

Returns DVH.study_instance_uid

Return type list

update_endpoints_and_radbio()
Update endpoint and radbio data in self.data. This function is needed since all of these values are calculated after a query and user may change these values.

property variables
Get variable names for plotting

Returns keys of StatsData.data sans 'Simulation Date'

Return type list

property vars_with_nan_values
Find variable names that contain non-numerical values

Returns Variable names that cannot be converted to float

Return type list

dvha.tools.stats.**get_control_limits**(*y*, *std_devs=3*)
Calculate control limits for Control Chart

Parameters

- **y** (*list*) – data
- **std_devs** (*int or float*) – values greater than std_devs away are out-of-control (Default value = 3)

Returns center line, upper control limit, and lower control limit

Return type type

dvha.tools.stats.**get_index_of_nan**(*numpy_array*)
Find indices of np.nan values

Parameters **numpy_array** (*np.ndarray*) – A numpy array

Returns indices of `numpy_array` that are `np.nan`

Return type list

dvha.tools.stats.**get_p_values**(*X*, *y*, *predictions*, *params*)
Get p-values using sklearn based on https://stackoverflow.com/questions/27928275/find-p-value-significance-in-scikit-learn-linearregression

Parameters

- **X** (*np.ndarray*) – independent data
- **y** (*np.ndarray*) – dependent data
- **predictions** – output from linear_model.LinearRegression.predict
- **params** – np.array([y_incercept, slope])

Returns p-values

Return type list

dvha.tools.stats.**str_starts_with_any_in_list**(*string_a*, *string_list*)
Check if string_a starts with any string the provided list of strings

Parameters

- **string_a** (*str*) – Any string
- **string_list** (*list*) – A list of strings

Returns True if any `string_a` starts with any string in `string_list`

Return type bool

dvha.tools.stats.**sync_variables_in_stats_data_objects**(*stats_data_1*, *stats_data_2*)
Ensure both stats_data objects have the same variables

Parameters

- **stats_data_1** (StatsData) – A StatsData object (e.g., Group 1)
- **stats_data_2** (StatsData) – Another StatsData object (e.g., Group 2)

CHAPTER

FOUR

DATA DICTIONARY

Each table below describes a SQL table (of the same name as the section header). This is manually generated, so best to check out `dvha.db.create_tables.sql` and `dvha.db.create_tables_sqlite.sql` for any changes.

4.1 DVHs

Storage of DVHs and other ROI specific data.

Column	Data Type	Units	DICOM Tag	Description
mrn	text	–	(0010, 0020)	Medical Record Number (PatientID)
study_instance_uid	text	–	(0020, 000D)	Unique ID tied to planning image set
centroid	varchar(35)	–	–	DVHA custom function
centroid_dist_to_iso_max	–	–	–	DVHA custom function
centroid_dist_to_iso_min	–	–	–	DVHA custom function
cross_section_max	real	cm^2	–	DVHA custom function with Shapely
cross_section_median	real	cm^2	–	DVHA custom function with Shapely
dist_to_ptv_centroids	real	cm	–	DVHA custom function
dist_to_ptv_max	real	cm	–	Calculated with scipy's cdist function
dist_to_ptv_75	real	cm	–	Calculated with scipy's cdist function
dist_to_ptv_mean	real	cm	–	Calculated with scipy's cdist function
dist_to_ptv_median	real	cm	–	Calculated with scipy's cdist function
dist_to_ptv_25	real	cm	–	Calculated with scipy's cdist function
dist_to_ptv_min	real	cm	–	Calculated with scipy's cdist function
dth_string	text	cm	–	numpy histogram of scipy cdist with PTV
dvh_string	text	cGy	–	CSV of DVH in 1 cGy bins
import_time_stamp	timestamp	–	–	Time per SQL at time of import
institutional_roi	varchar(50)	–	–	Standard ROI name for all physician
max_dose	real	Gy	–	Max ROI dose per dicompyler
mean_dose	real	Gy	–	Mean ROI dose per dicompyler
min_dose	real	Gy	–	Min ROI dose per dicompyler
ovh_max	real	cm	–	Custom DVHA overlap volume histogram calc
ovh_75	real	cm	–	Custom DVHA overlap volume histogram calc
ovh_mean	real	cm	–	Custom DVHA overlap volume histogram calc
ovh_median	real	cm	–	Custom DVHA overlap volume histogram calc
ovh_25	real	cm	–	Custom DVHA overlap volume histogram calc
ovh_min	real	cm	–	Custom DVHA overlap volume histogram calc
physician_roi	varchar(50)	–	–	Standard ROI name for patient's physician
ptv_overlap	real	cm^3	–	DVHA custom function with Shapely

continues on next page

Table 1 – continued from previous page

Column	Data Type	Units	DICOM Tag	Description
roi_coord_string	text	–	(3006, 0050)	Single string containing all ROI points
roi_name	varchar(50)	–	(3006, 0026)	ROI name as in plan
roi_type	varchar(20)	–	(3006, 00A4)	ROI categegoy (e.g., ORGAN, PTV)
spread_x	real	cm	(3006, 0050)	Max distance in x-dim of ROI
spread_y	real	cm	(3006, 0050)	Max distance in y-dim of ROI
spread_z	real	cm	(3006, 0050)	Max distance in z-dim of ROI
surface_area	real	cm^2	–	DVHA custom function, needs validation
toxicity_grade	smallint	–	–	Not yet implemented
volume	real	cm^3	–	ROI volume per dicompyler
integral_dose	real	$Gy*cm^3$	–	ROI mean dose times volume per dicompyler

4.2 Plans

Storage of information applicable across an entire plan / StudyInstanceUID.

Column	Data Type	Units	DICOM Tag	Description
mrn	text	–	(0010, 0020)	Medical Record Number (PatientID)
study_instance_uid	text	–	(0020, 000D)	Unique ID tied to planning image set
age	smallint	years	–	Patient age on day of simulation
baseline	boolean	–	–	Not yet implemented
birth_date	date	–	(0010, 0030)	–
complexity	real	–	–	Plan complexity score
dose_grid_res	varchar(16)	mm	(0028, 0030)	Resolution of dose grid
–	–	–	(0018, 0050)	–
dose_time_stamp	timestamp	–	(3006, 0012)	Timestamp for dose file
–	–	–	(3006, 0013)	Timestamp for dose file
fxs	int	–	(300a, 0078)	NumberOfFractionsPlanned
heterogeneity_correction	varchar(30)	–	(3004, 0014)	CSV of heterogeneity correction
import_time_stamp	timestamp	–	–	Time per SQL at time of import
patient_orientation	varchar(3)	–	(0018, 5100)	Acronym of patient's sim orientation
patient_sex	char(1)	–	(0010, 0040)	Patient's sex
physician	varchar(50)	–	(0010, 0048)	PhysiciansOfRecord or
–	–	–	(0008, 0090)	ReferringPhysiciansName
plan_time_stamp	timestamp	–	(300A, 0006)	Timestamp for plan
–	–	–	(300A, 0007)	Timestamp for plan
protocol	text	–	–	Not yet implemented
ptv_cross_section_max	real	cm^2	–	Area of largest PTV slice for plan
ptv_cross_section_median	real	cm^2	–	Median slice area of PTV for plan
ptv_max_dose	real	Gy	–	per dicompyler-core
ptv_min_dose	real	Gy	–	per dicompyler-core
ptv_spread_x	real	cm	–	Largest x-dim distance of PTV for plan
ptv_spread_y	real	cm	–	Largest y-dim distance of PTV for plan
ptv_spread_z	real	cm	–	Largest z-dim distance of PTV for plan
ptv_surface_area	real	cm^2	–	Surface area of PTV for plan
ptv_volume	real	cm^3	–	Volume of PTV for plan
rx_dose	real	Gy	(300A, 0026)	TargetPrescriptionDose
sim_study_date	date	–	(0008, 0020)	Date of simulation imaging

continues on next page

Table 2 – continued from previous page

Column	Data Type	Units	DICOM Tag	Description
struct_time_stamp	timestamp	–	(3006, 0008)	Timestamp for structure set
–	–	–	(3006, 0009)	Timestamp for structure set
total_mu	real	–	(300a, 0086)	Total MU to be delivered to the patient
toxicity_grades	text	–	–	Not yet implemented
tps_manufacturer	varchar(50)	–	(0008, 0070)	Manufacturer in RTPlan
tps_software_name	varchar(50)	–	(0008, 1090)	ManufacturerModelName in RTPlan
tps_software_version	varchar(30)	–	(0018, 1020)	CSV of SoftwareVersions in RTPlan
tx_modality	varchar(30)	–	(300A, 00C6)	Based on RadiationType, includes 3D or arc
–	–	–	(300A, 011E)	–
tx_site	varchar(50)	–	(300A, 0002)	RTPlanLabel
tx_time	time	–	(300A, 0286)	For brachy plans

4.3 Rxs

Storage of information for a given prescription.

Column	Data Type	Units	DICOM Tag	Description
mrn	text	–	(0010, 0020)	Medical Record Number (PatientID)
study_instance_uid	text	–	(0020, 000D)	Unique ID tied to planning image set
fx_dose	real	–	–	rx_dose / fxs
fx_grp_count	smallint	–	–	Number of fraction groups in RTPlan
fx_grp_name	varchar(30)	–	(300A, 0071)	Primarily for Pinnacle with special POIs
fx_grp_number	smallint	–	(300A, 0071)	–
fxs	smallint	–	(300A, 0078)	–
import_time_stamp	timestamp	–	–	Time per SQL at time of import
normalization_method	varchar(30)	–	(300A, 0014)	–
normalization_object	varchar(30)	–	–	Intended for special POIs
plan_name	varchar(50)	–	(300A, 0002)	–
rx_dose	real	–	(300A, 0026)	Per dicompyler if not found
rx_percent	real	–	–	Currently only available with special POIs

4.4 Beams

Storage of information per beam.

Column	Data Type	Units	DICOM Tag	Description
mrn	text	–	(0010, 0020)	Medical Record Number (PatientID)
study_instance_uid	text	–	(0020, 000D)	Unique ID tied to planning image set
area_max	real	–	–	–
area_mean	real	–	–	–
area_median	real	–	–	–
area_min	real	–	–	–
beam_dose	real	–	(300A, 008B)	–
beam_dose_pt	varchar(35)	–	(300A, 0082)	–
beam_energy_max	real	–	(300A, 0114)	–

continues on next page

Table 3 – continued from previous page

Column	Data Type	Units	DICOM Tag	Description
beam_energy_min	real	–	(300A, 0114)	–
beam_mu	real	–	(300A, 0086)	–
beam_mu_per_cp	real	–	–	–
beam_mu_per_deg	real	–	–	–
beam_name	varchar(30)	–	(300A, 00C3)	Beam Description or
–	–	–	(300A, 00C2)	Beam Name
beam_number	int	–	(300A, 00C0)	–
beam_type	varchar(30)	–	(300A, 00C4)	–
collimator_end	real	–	(300A, 0120)	–
collimator_max	real	–	(300A, 0120)	–
collimator_min	real	–	(300A, 0120)	–
collimator_range	real	–	(300A, 0120)	–
collimator_rot_dir	varchar(5)	–	(300A, 0121)	–
collimator_start	real	–	(300A, 0120)	–
complexity	real	–	–	–
complexity_max	real	–	–	–
complexity_mean	real	–	–	–
complexity_median	real	–	–	–
complexity_min	real	–	–	–
control_point_count	int	–	–	–
couch_end	real	–	(300A, 0120)	–
couch_max	real	–	(300A, 0120)	–
couch_min	real	–	(300A, 0120)	–
couch_range	real	–	(300A, 0120)	–
couch_rot_dir	varchar(5)	–	(300A, 0123)	–
couch_start	real	–	(300A, 0122)	–
cp_mu_max	real	–	–	–
cp_mu_mean	real	–	–	–
cp_mu_median	real	–	–	–
cp_mu_min	real	–	–	–
fx_count	int	–	–	See Rxs table
fx_grp_beam_count	smallint	–	–	See Rxs table
fx_grp_number	smallint	–	–	See Rxs table
gantry_end	real	–	(300A, 011E)	–
gantry_max	real	–	(300A, 011E)	–
gantry_min	real	–	(300A, 011E)	–
gantry_range	real	–	(300A, 011E)	–
gantry_rot_dir	varchar(5)	–	(300A, 011F)	–
gantry_start	real	–	(300A, 011E)	–
import_time_stamp	timestamp	–	–	Time per SQL at time of import
isocenter	varchar(35)	–	(300A, 012C)	–
perim_max	real	–	–	–
perim_mean	real	–	–	–
perim_median	real	–	–	–
perim_min	real	–	–	–
radiation_type	varchar(30)	–	(300A, 00C6)	–
scan_mode	varchar(30)	–	(300A, 0308)	–
scan_spot_count	real	–	(300A, 0392)	–
ssd	real	–	(300A, 0130)	Average of these values

continues on next page

Table 3 – continued from previous page

Column	Data Type	Units	DICOM Tag	Description
treatment_machine	varchar(30)	–	(300A, 00B2)	–
tx_modality	varchar(30)	–	–	–
x_perim_max	real	–	–	–
x_perim_mean	real	–	–	–
x_perim_median	real	–	–	–
x_perim_min	real	–	–	–
y_perim_max	real	–	–	–
y_perim_mean	real	–	–	–
y_perim_median	real	–	–	–
y_perim_min	real	–	–	–

CHAPTER

FIVE

CREDITS

5.1 Development Lead

- Dan Cutright

5.2 Contributors

- Mahesh Gopalakrishnan
- Arkajyoti Roy
- Aditya Panchal
- Max Grohmann

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

H

I

M

N

O

P

Q

R

S

T

U

V

W

X

Y