DVHA-Stats

Release 0.2.4

Mar 04, 2021

Contents

CHAPTER 1

dvhastats

A library of prediction and statistical process control tools. Although based on work in DVH Analytics, all tools in this library are generic and not specific to radiation oncology. See our documentation for advanced uses.

1.1 What does it do?

- Read data from CSV, supply as numpy array or dict
- **Basic plotting**
  - Simple one-variable plots from data
  - Control Charts (Univariate, Multivariate, & Risk-Adjusted)
  - Heat Maps (correlations, PCA, etc.)
- Perform Box-Cox transformations
- Calculate Correlation matrices
- Perform Multi-Variable Linear Regressions
- Perform Principal Component Analysis (PCA)

1.2 Other information

- Free software: MIT license
- Documentation: Read the docs
- Tested on Python 3.6, 3.7, 3.8

1.3 Dependencies

- scipy
- numpy
- scikit-learn
- regressors
- matplotlib
- PTable

1.4 Basic Usage

```
>>> from dvhastats.ui import DVHAStats
>>> s = DVHAStats("your_data.csv")  # use s = DVHAStats() for test data

>>> s.var_names
['V1', 'V2', 'V3', 'V4', 'V5', 'V6']

>>> s.show('V1')  # or s.show(0), can provide index or var_name
```

1.4.1 Multivariate Control Chart (w/ non-normal data)

```
>>> ht2_bc = s.hotelling_t2(box_cox=True)
>>> ht2_bc.show()
```

1.4.2 Principal Component Analysis (PCA)

```
>>> pca = s.pca()
>>> pca.show()
```

CHAPTER 2

Installation

At the command line:

```
$ pip install dvha-stats
```

CHAPTER 3

Usage

To use dvha-stats in a project:

Statistical data can be easily accessed with *dvhastats.ui.DVHAStats* class.

3.1 Getting Started

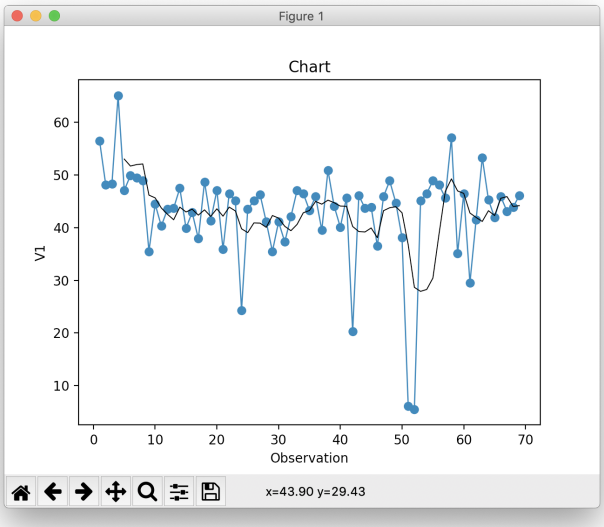
Before attempting the examples below, run these lines first:

```
>>> from dvhastats.ui import DVHAStats
>>> s = DVHAStats("your_data.csv")  # use s = DVHAStats() for test data
```

This assumes that your csv is formatted such that it contains one row per observation (*i.e.*, wide format). If your csv contains multivariate data with one row per dependent value (*i.e.*, narrow format), you can use *dvhastats.utilities.widen_data()*. See Reformatting CSV for an example.

3.2 Basic Plotting

```
>>> s.var_names
['V1', 'V2', 'V3', 'V4', 'V5', 'V6']

>>> s.get_data_by_var_name('V1')
array([56.5, 48.1, 48.3, 65.1, 47.1, 49.9, 49.5, 48.9, 35.5, 44.5, 40.3,
       43.5, 43.7, 47.5, 39.9, 42.9, 37.9, 48.7, 41.3, 47.1, 35.9, 46.5,
       45.1, 24.3, 43.5, 45.1, 46.3, 41.1, 35.5, 41.1, 37.3, 42.1, 47.1,
       46.5, 43.3, 45.9, 39.5, 50.9, 44.1, 40.1, 45.7, 20.3, 46.1, 43.7,
       43.9, 36.5, 45.9, 48.9, 44.7, 38.1,  6.1,  5.5, 45.1, 46.5, 48.9,
       48.1, 45.7, 57.1, 35.1, 46.5, 29.5, 41.5, 53.3, 45.3, 41.9, 45.9,
       43.1, 43.9, 46.1])

>>> s.show('V1')  # or s.show(0), can provide index or var_name
```



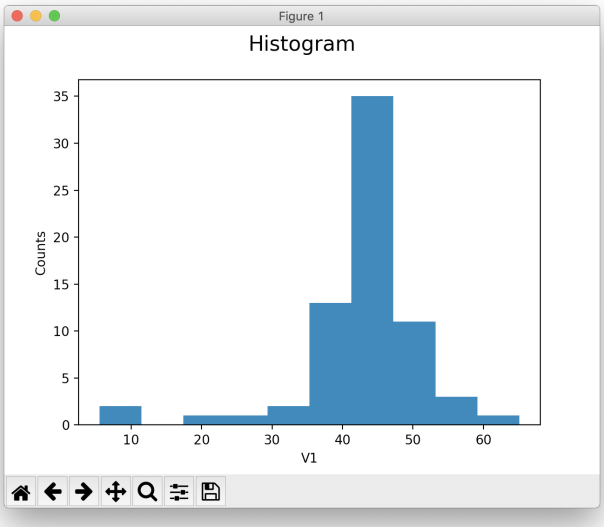
3.3 Histogram

Calculation with numpy.

```
>>> h = s.histogram('V1')
>>> hist, center = h.hist_data
>>> hist
array([ 2,  0,  0,  0,  0,  1,  1,  0,  1,  0,  5,  4,  9, 15, 17, 10,  1,
    1,  1,  0,  1]
>>> center
array([ 6.91904762,  9.75714286, 12.5952381 , 15.43333333, 18.27142857,
       21.10952381, 23.94761905, 26.78571429, 29.62380952, 32.46190476,
       35.3       , 38.13809524, 40.97619048, 43.81428571, 46.65238095,
       49.49047619, 52.32857143, 55.16666667, 58.0047619 , 60.84285714,
       63.68095238])
```

Calculation with matplotlib.

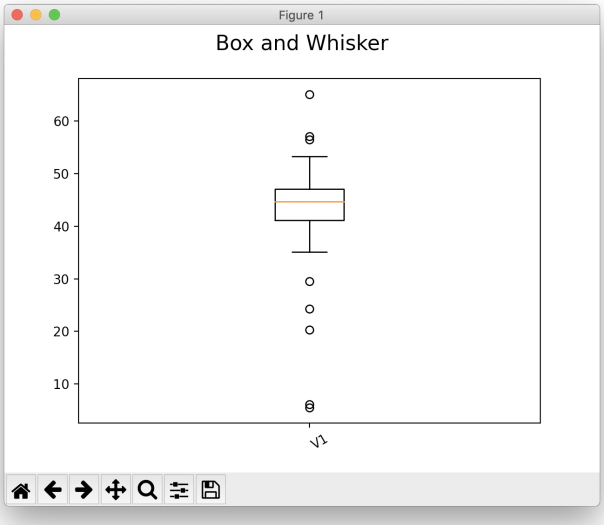
```
>>> s.show(0, plot_type="hist")  # histogram recalculated using matplotlib
```



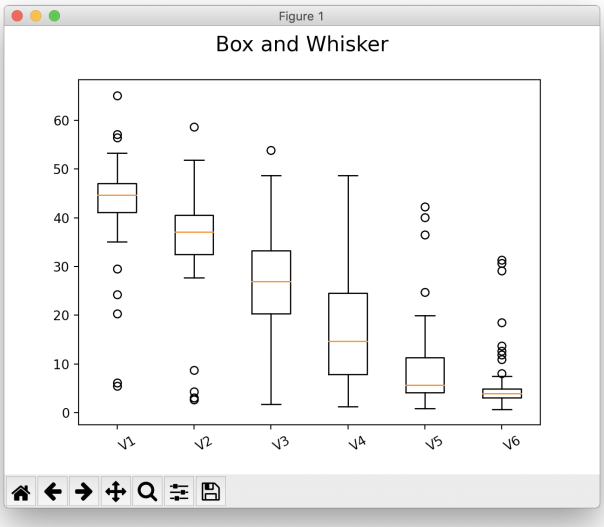
3.4 Box & Whisker Plot

Calculation with matplotlib

```
>>> s.show(0, plot_type="box")
```



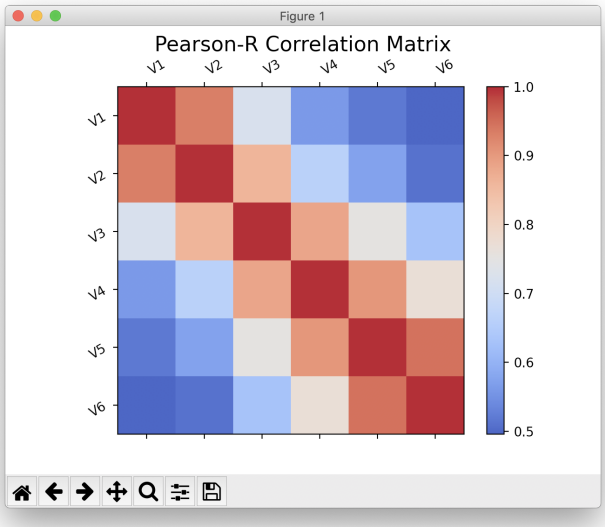

```
>>> s.show(plot_type="box")
```



3.5 Pearson-R Correlation Matrix

Calculation with scipy.

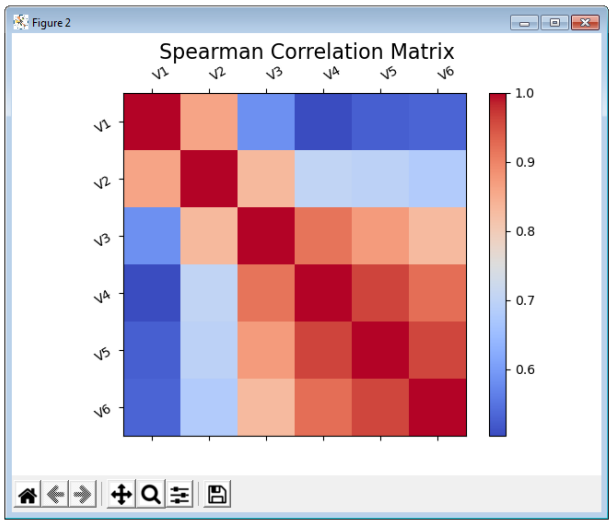
```
>>> pearson_mat = s.correlation_matrix()
>>> pearson_mat.corr  # correlation array
array([[1.        , 0.93160407, 0.72199862, 0.56239953, 0.51856243, 0.49619153],
       [0.93160407, 1.        , 0.86121347, 0.66329274, 0.5737434 , 0.51111648],
       [0.72199862, 0.86121347, 1.        , 0.88436716, 0.7521324 ,  0.63030588],
       [0.56239953, 0.66329274, 0.88436716, 1.        , 0.90411476, 0.76986654],
       [0.51856243, 0.5737434 , 0.7521324 , 0.90411476, 1.        , 0.9464186 ],
       [0.49619153, 0.51111648, 0.63030588, 0.76986654, 0.9464186 , 1.        ]])
>>> pearson_mat.p  # p-values
array([[0.00000000e+00, 3.70567507e-31, 2.54573222e-12, 4.92807604e-07, 5.01004755e-
↪06, 1.45230750e-05],
       [3.70567507e-31, 0.00000000e+00, 2.27411745e-21, 5.28815300e-10, 2.55750429e-
↪07, 7.19979746e-06],
       [2.54573222e-12, 2.27411745e-21, 0.00000000e+00, 7.41613930e-24, 9.37849945e-
↪14, 6.49207976e-09],
       [4.92807604e-07, 5.28815300e-10, 7.41613930e-24, 0.00000000e+00, 1.94118606e-
↪26, 1.06898267e-14],
       [5.01004755e-06, 2.55750429e-07, 9.37849945e-14, 1.94118606e-26, 0.
↪00000000e+00, 1.32389842e-34],
       [1.45230750e-05, 7.19979746e-06, 6.49207976e-09, 1.06898267e-14, 1.32389842e-
↪34, 0.00000000e+00]])
>>> pearson_mat.show()
```



3.6 Spearman Correlation Matrix

Calculation with scipy.

```
>>> spearman_mat = s.correlation_matrix("Spearman")
>>> spearman_mat.show()
```



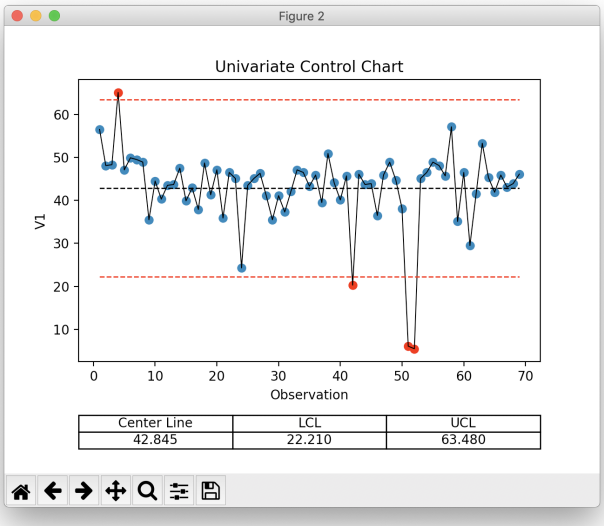
3.7 Univariate Control Chart

```
>>> ucc = s.univariate_control_charts()
>>> ucc['V1']
center_line: 42.845
```

(continues on next page)

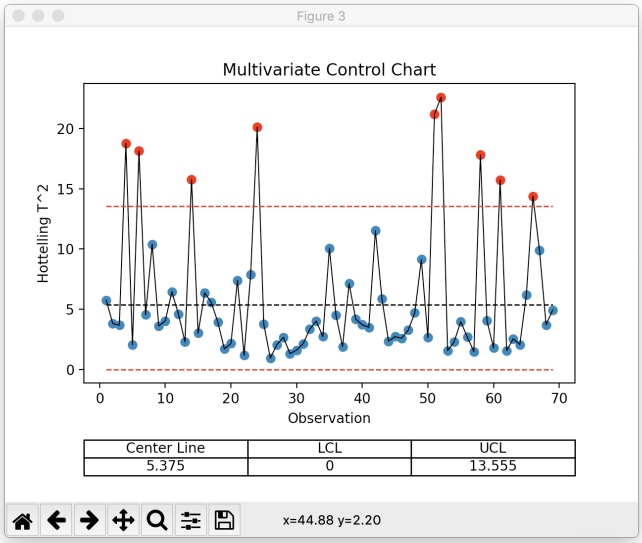
(continued from previous page)

```
control_limits: 22.210, 63.480
out_of_control: [ 3 41 50 51]

>>> ucc['V1'].show()  # or ucc[0].show(), can provide index or var_name
```



3.8 Multivariate Control Chart

```
>>> ht2 = s.hotelling_t2()
>>> ht2
Q: [ 5.75062092  3.80141786  3.67243782 18.80124504  2.03849294 18.15447155
     4.54475048 10.40783971  3.60614333  4.03138994  6.45171623  4.60475303
     2.29185301 15.7891342   3.0102578   6.36058098  5.56477106  3.92950273
     1.70534379  2.14021007  7.3839626   1.16554558  7.89636669 20.13613585
     3.76034723  0.93179106  2.05542886  2.65257506  1.31049764  1.59880892
     2.13839258  3.33331329  4.01060102  2.71837612 10.0744586   4.50776545
     1.87955428  7.13423455  4.1773818   3.70446025  3.49570988 11.52822658
     5.874624    2.34515306  2.71884639  2.58457841  3.2591779   4.69554484
     9.1358149   2.64106059 21.21960037 22.6229493   1.55545875  2.29606726
     3.96926714  2.69041382  1.47639788 17.83532339  4.03627833  1.78953536
    15.7485067   1.56110637  2.53753085  2.04243193  6.20630748 14.39527077
     9.88243129  3.70056854  4.92888799]
center_line: 5.375
control_limits: 0, 13.555
out_of_control: [ 3  5 13 23 50 51 57 60 65]

>>> ht2.show()ht
```



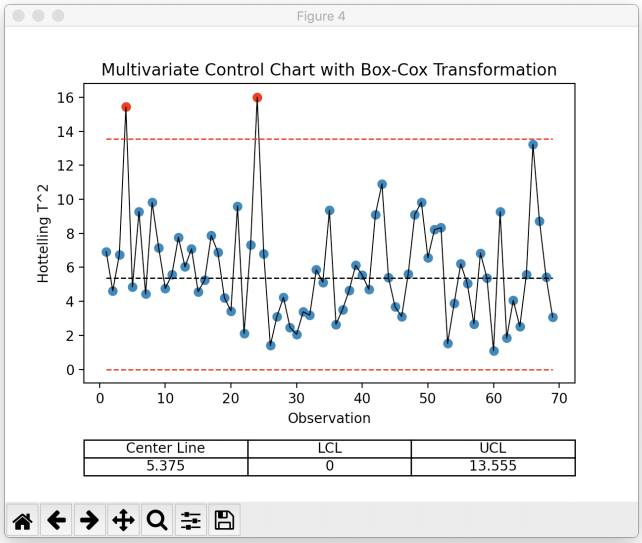
3.9 Box-Cox Transformation (for non-normal data)

Calculation with scipy.

```
>>> bc = s.box_cox_by_index(0)
>>> bc
array([3185.2502073 , 2237.32503551, 2257.79294148, 4346.90639712,
       2136.50469314, 2425.19594298, 2382.73410297, 2319.80580872,
       1148.63472597, 1886.15962058, 1517.3226398 , 1794.37742725,
       1812.53465647, 2176.52932216, 1484.4619302 , 1740.50195077,
       1326.0093692 , 2299.03324672, 1601.1904051 , 2136.50469314,
       1177.23656545, 2077.22485894, 1942.42664844,  499.72380601,
       1794.37742725, 1942.42664844, 2057.66647538, 1584.22036354,
       1148.63472597, 1584.22036354, 1280.36568471, 1670.05579771,
       2136.50469314, 2077.22485894, 1776.31962594, 2018.85154453,
       1451.99231252, 2533.13894266, 1849.14775291, 1500.84335095,
       1999.59482773,  336.62160027, 2038.20873211, 1812.53465647,
       1830.79140224, 1220.85798302, 2018.85154453, 2319.80580872,
       1904.81531264, 1341.41740006,   23.64034973,   18.74313335,
       1942.42664844, 2077.22485894, 2319.80580872, 2237.32503551,
       1999.59482773, 3259.95515527, 1120.41519999, 2077.22485894,
        764.99904232, 1618.25887705, 2802.6765172 , 1961.38246534,
       1652.69148146, 2018.85154453, 1758.36116355, 1830.79140224,
       2038.20873211])
```

3.10 Multivariate Control Chart (w/ non-normal data)

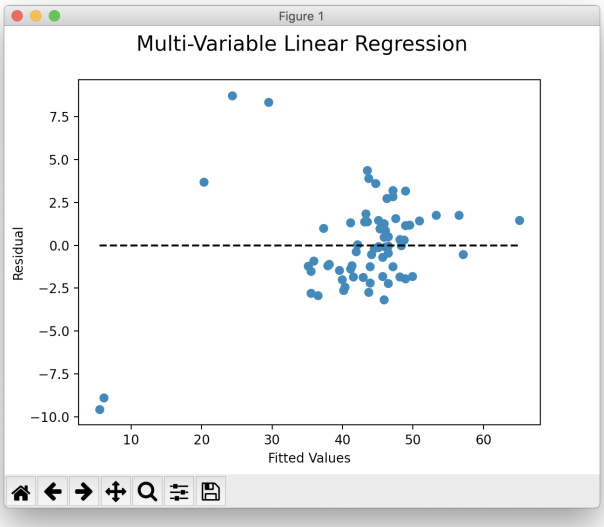
```
>>> ht2_bc = s.hotelling_t2(box_cox=True)
>>> ht2_bc.show()
```



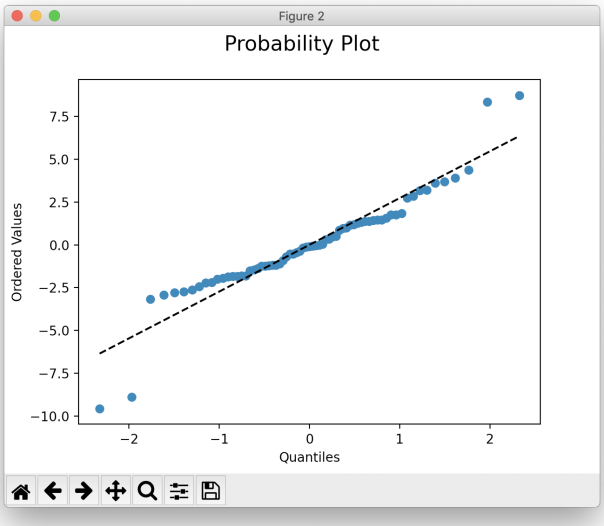
3.11 Multi-Variable Linear Regression

Calculation with sklearn.

```
>>> mvr = s.linear_reg("V1")
>>> mvr

Multi-Variable Regression results/model
R^2: 0.906
MSE: 7.860
f-stat: 121.632
f-stat p-value: 1.000
+-------+------------+-----------+---------+---------+
|       |       Coef | Std. Err. | t-value | p-value |
+-------+------------+-----------+---------+---------+
| y-int |  1.262E+01 | 1.326E+00 |   9.518 |   0.000 |
|   V2  |  1.107E+00 | 7.547E-02 |  14.664 |   0.000 |
|   V3  | -4.442E-01 | 1.135E-01 |  -3.914 |   0.000 |
|   V4  |  1.786E-01 | 1.340E-01 |   1.333 |   0.187 |
|   V5  | -1.789E-01 | 2.538E-01 |  -0.705 |   0.483 |
|   V6  |  2.833E-01 | 2.355E-01 |   1.203 |   0.233 |
+-------+------------+-----------+---------+---------+

>>> mvr.show()
```




```
>>> mvr.show("prob")
```




```
>>> mvr2 = s.linear_reg("V1", back_elim=True)
>>> mvr2

Multi-Variable Regression results/model
R^2: 0.903
MSE: 8.096
f-stat: 202.431
f-stat p-value: 1.000
+-------+-----------+-----------+---------+---------+
|       |      Coef | Std. Err. | t-value | p-value |
+-------+-----------+-----------+---------+---------+
| y-int |  1.276E+01 | 1.321E+00 |   9.656 |   0.000 |
|   V2  |  1.070E+00 | 6.700E-02 |  15.967 |   0.000 |
```

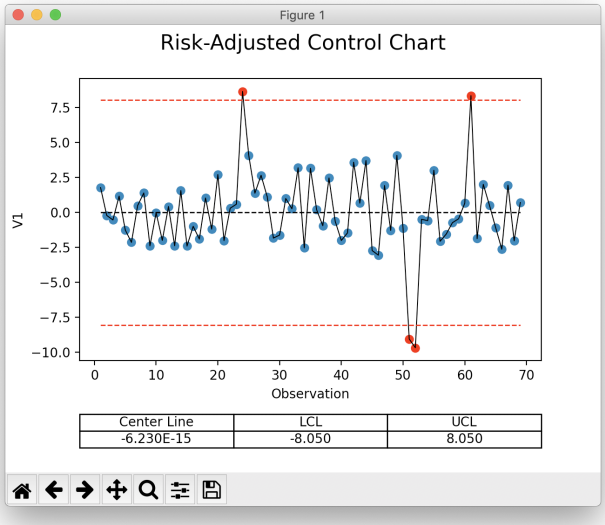
(continues on next page)

(continued from previous page)

```
|    V3   | -3.318E-01 | 6.852E-02 |  -4.843 |    0.000 |
|    V6   |  2.000E-01 | 7.542E-02 |   2.652 |    0.010 |
+---------+------------+-----------+---------+----------+
```

3.12 Risk-Adjusted Control Chart

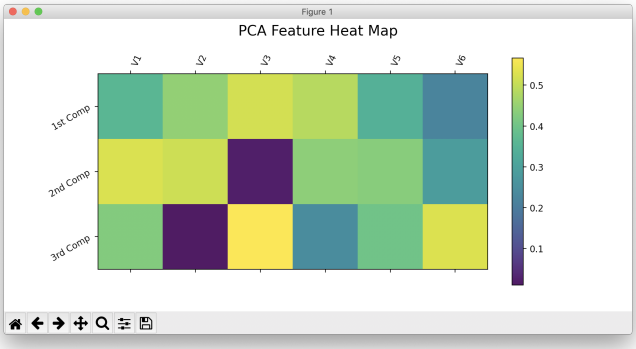
```
>>> ra_cc = s.risk_adjusted_control_chart("V1", back_elim=True)
>>> ra_cc.show()
```



3.13 Principal Component Analysis (PCA)

Calculation with sklearn.

```
>>> pca = s.pca()
>>> pca.feature_map_data
array([[ 0.35795147,  0.44569046,  0.51745294,  0.48745318,  0.34479542, 0.22131141],
       [-0.52601728, -0.51017406, -0.02139406,  0.4386136 ,  0.43258992, 0.28819198],
       [ 0.42660699,  0.01072412, -0.5661977 , -0.24404558,  0.39945093, 0.52743943]])
>>> pca.show()
```



3.14 Reformatting CSV

Below is an example of how to reformat a "narrow" csv (one row per dependent variable value) to a "wide" format (one row per observation). Please see *dvhastats.utilities.widen_data* for additional documentation.

Let's assume the contents of your csv file looks like:

```
patient,plan,field id,image type, date, DD(%), DTA(mm),Threshold(%),Gamma Pass Rate(%)
ANON1234,Plan_name,3,field,6/13/2019 7:27,3,2,10,99.94708217
ANON1234,Plan_name,3,field,6/13/2019 7:27,3,3,5,99.97934552
ANON1234,Plan_name,3,field,6/13/2019 7:27,3,3,10,99.97706894
ANON1234,Plan_name,3,field,6/13/2019 7:27,2,3,5,99.88772435
ANON1234,Plan_name,4,field,6/13/2019 7:27,3,2,10,99.99941874
ANON1234,Plan_name,4,field,6/13/2019 7:27,3,3,5,100
ANON1234,Plan_name,4,field,6/13/2019 7:27,3,3,10,100
ANON1234,Plan_name,4,field,6/13/2019 7:27,2,3,5,99.99533258
```

We can see that all data here is of the same patient, plan, and date. In this example, we want to evaluate the variation of Gamma Pass Rate(%) as a function of DD(%), DTA(mm), and Threshold(%). So, in this context, we really only want two rows of data, one for each field id (*i.e.*, 3 or 4).

```
>>> from dvhastats.utilities import csv_to_dict, widen_data
>>> data_dict = csv_to_dict("path_to_csv_file.csv")
>>> uid_columns = ['patient', 'plan', 'field id']  # only field id really needed in␣
↪this case
>>> x_data_cols = ['DD(%)', 'DTA(mm)', 'Threshold(%)']
>>> y_data_col = 'Gamma Pass Rate(%)'
>>> wide_data = widen_data(data_dict, uid_columns, x_data_cols, y_data_col)
>>> wide_data
    {'uid': ['ANON1234Plan_name3', 'ANON1234Plan_name4'],
     '2/3/5': ['99.88772435', '99.99533258'],
     '3/2/10': ['99.94708217', '99.99941874'],
     '3/3/10': ['99.97706894', '100'],
     '3/3/5': ['99.97934552', '100']}
```

CHAPTER 4

dvha-stats

4.1 ui module

DVHA-Stats classes for user interaction

class dvhastats.ui.**ControlChartUI**(*y*, *std=3*, *ucl_limit=None*, *lcl_limit=None*, *var_name=None*, *x=None*, *plot_title=None*)

Bases: *dvhastats.ui.DVHAStatsBaseClass*, *dvhastats.stats.ControlChart*

Univariate Control Chart

Parameters

- **y** (*list, np.ndarray*) – Input data (1-D)
- **std** (*int, float, optional*) – Number of standard deviations used to calculate if a y-value is out-of-control.
- **ucl_limit** (*float, optional*) – Limit the upper control limit to this value
- **lcl_limit** (*float, optional*) – Limit the lower control limit to this value
- **plot_title** (*str, optional*) – Over-ride the plot title

show()

Display the univariate control chart with matplotlib

Returns The number of the newly created matplotlib figure

Return type int

class dvhastats.ui.**CorrelationMatrixUI**(*X*, *var_names=None*, *corr_type='Pearson'*, *cmap='coolwarm'*)

Bases: *dvhastats.ui.DVHAStatsBaseClass*, *dvhastats.stats.CorrelationMatrix*

Pearson-R correlation matrix UI object

Parameters

- **X** (*np.ndarray*) – Input data (2-D) with N rows of observations and p columns of variables.
- **var_names** (*list, optional*) – Optionally set the variable names with a list of str
- **corr_type** (*str*) – Either "Pearson" or "Spearman"
- **cmap** (*str*) – matplotlib compatible color map

show(*absolute=False*, *corr=True*)
Create a heat map of PCA components

Parameters

- **absolute** (*bool*) – Heat map will display the absolute values in PCA components if True
- **corr** (*bool*) – Plot a p-value matrix if False, correlation matrix if True.

Returns The number of the newly created matplotlib figure

Return type int

class dvhastats.ui.**DVHAStats**(*data=None*, *var_names=None*, *x_axis=None*, *avg_len=5*, *del_const_vars=False*)
Bases: *dvhastats.ui.DVHAStatsBaseClass*

The main UI class object for DVHAStats

Parameters

- **data** (*numpy.array, dict, str, None*) – Input data (2-D) with N rows of observations and p columns of variables. The CSV file must have a header row for column names. Test data is loaded if None
- **var_names** (*list of str, optional*) – If data is a numpy array, optionally provide the column names.
- **x_axis** (*numpy.array, list, optional*) – Specify x_axis for plotting purposes. Default is based on row number in data
- **avg_len** (*int*) – When plotting raw data, a trend line will be plotted using this value as an averaging length. If N < avg_len + 1 will not plot a trend line
- **del_const_vars** (*bool*) – Automatically delete any variables that have constant data. The names of these variables are stored in the excluded_vars attr. Default value is False.

box_cox(*alpha=None*, *lmbda=None*, *const_policy='propagate'*)
Apply box_cox_by_index for all data

box_cox_by_index(*index*, *alpha=None*, *lmbda=None*, *const_policy='propagate'*)

Parameters

- **index** (*int, str*) – The index corresponding to the variable data to have a box-cox transformation applied. If index is a string, it will be assumed to be the var_name
- **lmbda** (*None, scalar, optional*) – If lmbda is not None, do the transformation for that value. If lmbda is None, find the lambda that maximizes the log-likelihood function and return it as the second output argument.
- **alpha** (*None, float, optional*) – If alpha is not None, return the 100 * (1-alpha)% confidence interval for lmbda as the third output argument. Must be between 0.0 and 1.0.

- **const_policy** (*str*) – {'propagate', 'raise', 'omit'} Defines how to handle when data is constant. The following options are available (default is 'propagate'): 'propagate': returns nan 'raise': throws an error 'omit': remove

Returns Results from stats.box_cox

Return type np.ndarray

constant_var_indices
Get a list of all constant variable indices

Returns Indices of variables with no variation

Return type list

constant_vars
Get a list of all constant variables

Returns Names of variables with no variation

Return type list

correlation_matrix(*corr_type='Pearson'*)
Get a Pearson-R or Spearman correlation matrices

Parameters **corr_type** (*str*) – Either "Pearson" or "Spearman"

Returns A CorrelationMatrixUI class object

Return type *CorrelationMatrixUI*

del_const_vars()
Permanently remove variables with no variation

del_var(*var_name*)
Determine if data by var_name is constant

Parameters **var_name** (*int, str*) – The var_name to delete (or index of variable)

get_data_by_var_name(*var_name*)
Get the single variable array based on var_name

Parameters **var_name** (*int, str*) – The name (str) or index (int) of the variable of interest

Returns The column of data for the given var_name

Return type np.ndarray

get_index_by_var_name(*var_name*)
Get the variable index by var_name

Parameters **var_name** (*int, str*) – The name (str) or index (int) of the variable of interest

Returns The column index for the given var_name

Return type int

histogram(*var_name*, *bins='auto'*, *nan_policy='omit'*)
Get a Histogram class object

var_name [str, int] The name (str) or index (int) of teh variable to plot

bins [int, list, str, optional] See https://numpy.org/doc/stable/reference/generated/numpy.histogram.html for details

nan_policy [str] Value must be one of the following: 'propagate', 'raise', 'omit' Defines how to handle when input contains nan. The following options are available (default is 'omit'): 'propagate': returns nan 'raise': throws an error 'omit': performs the calculations ignoring nan values

hotelling_t2(*alpha=0.05*, *box_cox=False*, *box_cox_alpha=None*, *box_cox_lmbda=None*, *const_policy='omit'*)

Calculate control limits for a standard univariate Control Chart

Parameters

- **alpha** (*float*) – Significance level used to determine the upper control limit (ucl)
- **box_cox** (*bool, optional*) – Set to true to perform a Box-Cox transformation on data prior to calculating the control chart statistics
- **box_cox_alpha** (*float, optional*) – If alpha is not None, return the 100 * (1-alpha)% confidence interval for lmbda as the third output argument. Must be between 0.0 and 1.0.
- **box_cox_lmbda** (*float, optional*) – If lmbda is not None, do the transformation for that value. If lmbda is None, find the lambda that maximizes the log-likelihood function and return it as the second output argument.
- **const_policy** (*str*) – {'raise', 'omit'} Defines how to handle when data is constant. The following options are available (default is 'raise'): 'raise': throws an error 'omit': exclude constant variables from calculation

Returns HotellingT2UI class object

Return type *HotellingT2UI*

is_constant(*var_name*)

Determine if data by var_name is constant

Parameters **var_name** (*int, str*) – The var_name to check (or index of variable)

Returns True if all values of var_name are the same (i.e., no variation)

Return type bool

linear_reg(*y*, *y_var_name=None*, *reg_vars=None*, *saved_reg=None*, *back_elim=False*, *back_elim_p=0.05*)

Initialize a MultiVariableRegression class object

Parameters

- **y** (*np.ndarray, list, str, int*) – Dependent data based on DVHAStats.data. If y is str or int, then it is assumed to be the var_name or index of data to be set as the dependent variable
- **y_var_name** (*int, str, optional*) – Optionally provide name of the dependent variable. Automatically set if y is str or int
- **reg_vars** (*list, optional*) – Optionally specify variable names or indices of data to be used in the regression
- **saved_reg** (MultiVariableRegression, *optional*) – If supplied, predicted values (y-hat) will be calculated with DVHAStats.data and the regression from saved_reg. This is useful if testing a regression model on new data.
- **back_elim** (*bool*) – Automatically perform backward elimination if True
- **back_elim_p** (*float*) – p-value threshold for backward elimination

Returns A LinearRegUI class object.

Return type *LinearRegUI*

non_const_data
Return self.data excluding any constant variables

Returns Data with constant variables removed. This does not alter the data property.

Return type np.ndarray

observations
Number of observations in data

Returns Number of rows in data

Return type int

pca(*n_components=0.95*, *transform=True*, ***kwargs*)
Return an sklearn PCA-like object, see PCA object for details

Parameters

- **n_components** (*int, float, None or str*) – Number of components to keep. if n_components is not set all components are kept: n_components == min(n_samples, n_features)

  If n_components == 'mle' and svd_solver == 'full', Minka's MLE is used to guess the dimension. Use of n_components == 'mle' will interpret svd_solver == 'auto' as svd_solver == 'full'.

  If 0 < n_components < 1 and svd_solver == 'full', select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components.

  If svd_solver == 'arpack', the number of components must be strictly less than the minimum of n_features and n_samples.

- **transform** (*bool*) – Fit the model and apply the dimensionality reduction
- **kwargs** (*any*) – Provide any keyword arguments for sklearn.decomposition.PCA: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Returns A principal component analysis object inherited from sklearn.decomposition.PCA

Return type *PCAUI*

risk_adjusted_control_chart(*y*, *std=3*, *ucl_limit=None*, *lcl_limit=None*, *saved_reg=None*, *y_name=None*, *reg_vars=None*, *back_elim=False*, *back_elim_p=0.05*)
Calculate control limits for a Risk-Adjusted Control Chart

Parameters

- **y** (*list, np.ndarray*) – 1-D Input data (dependent data)
- **std** (*int, float, optional*) – Number of standard deviations used to calculate if a y-value is out-of-control.
- **ucl_limit** (*float, optional*) – Limit the upper control limit to this value
- **lcl_limit** (*float, optional*) – Limit the lower control limit to this value
- **saved_reg** (*MultiVariableRegression, optional*) – Optionally provide a previously calculated regression
- **y_name** (*int, str, optional*) – Optionally provide name of the dependent variable. Automatically set if y is str or int

- **reg_vars** (*list, optional*) – Optionally specify variable names or indices of data to be used in the regression
- **saved_reg** – If supplied, predicted values (y-hat) will be calculated with DVHAStats.data and the regression from saved_reg. This is useful if testing a regression model on new data.
- **back_elim** (*bool*) – Automatically perform backward elimination if True
- **back_elim_p** (*float*) – p-value threshold for backward elimination

show(*var_name=None*, *plot_type='trend'*, ***kwargs*)
Display a plot of var_name with matplotlib

Parameters

- **var_name** (*str, int, None*) – The name (str) or index (int) of the variable to plot. If None and plot_type="boxplot", all variables will be plotted.
- **plot_type** (*str*) – Either "trend", "hist", "box"
- **kwargs** (*any*) – If plot_type is "hist", pass any of the matplotlib hist key word arguments

Returns The number of the newly created matplotlib figure

Return type int

univariate_control_chart(*var_name*, *std=3*, *ucl_limit=None*, *lcl_limit=None*, *box_cox=False*, *box_cox_alpha=None*, *box_cox_lmbda=None*, *const_policy='propagate'*)
Calculate control limits for a standard univariate Control Chart

Parameters

- **var_name** (*str, int*) – The name (str) or index (int) of teh variable to plot
- **std** (*int, float, optional*) – Number of standard deviations used to calculate if a y-value is out-of-control
- **ucl_limit** (*float, optional*) – Limit the upper control limit to this value
- **lcl_limit** (*float, optional*) – Limit the lower control limit to this value
- **box_cox** (*bool, optional*) – Set to true to perform a Box-Cox transformation on data prior to calculating the control chart statistics
- **box_cox_alpha** (*float, optional*) – If alpha is not None, return the 100 * (1-alpha)% confidence interval for lmbda as the third output argument. Must be between 0.0 and 1.0.
- **box_cox_lmbda** (*float, optional*) – If lmbda is not None, do the transformation for that value. If lmbda is None, find the lambda that maximizes the log-likelihood function and return it as the second output argument.
- **const_policy** (*str*) – {'propagate', 'raise', 'omit'} Defines how to handle when data is constant. The following options are available (default is 'propagate'): 'propagate': returns nan 'raise': throws an error 'omit': remove NaN data

Returns stats.ControlChart class object

Return type *stats.ControlChart*

univariate_control_charts(***kwargs*)
Calculate Control charts for all variables

Parameters **kwargs** (*any*) – See univariate_control_chart for keyword parameters

Returns ControlChart class objects stored in a dictionary with var_names and indices as keys (can use var_name or index)

Return type dict

variable_count
Number of variables in data

Returns Number of columns in data

Return type int

class dvhastats.ui.**DVHAStatsBaseClass**
Bases: object

Base Class for DVHAStats objects and child objects

close(*figure_number*)
Close a plot by figure_number

class dvhastats.ui.**HotellingT2UI**(*data*, *alpha=0.05*, *plot_title=None*)
Bases: *dvhastats.ui.DVHAStatsBaseClass*, *dvhastats.stats.HotellingT2*

Hotelling's t-squared statistic for multivariate hypothesis testing

Parameters

- **data** (*np.ndarray*) – A 2-D array of data to perform multivariate analysis. (e.g., DVHAStats.data)
- **alpha** (*float*) – The significance level used to calculate the upper control limit (UCL)
- **plot_title** (*str, optional*) – Over-ride the plot title

show()
Display the multivariate control chart with matplotlib

Returns The number of the newly created matplotlib figure

Return type int

class dvhastats.ui.**LinearRegUI**(*X*, *y*, *saved_reg=None*, *var_names=None*, *y_var_name=None*, *back_elim=False*, *back_elim_p=0.05*)
Bases: *dvhastats.ui.DVHAStatsBaseClass*, *dvhastats.stats.MultiVariableRegression*

A MultiVariableRegression class UI object

Parameters

- **y** (*np.ndarray, list*) – Dependent data based on DVHAStats.data
- **saved_reg** (*MultiVariableRegression, optional*) – If supplied, predicted values (y-hat) will be calculated with DVHAStats.data and the regression from saved_reg. This is useful if testing a regression model on new data.
- **var_names** (*list, optional*) – Optionally provide names of the independent variables
- **y_var_name** (*int, str, optional*) – Optionally provide name of the dependent variable
- **back_elim** (*bool*) – Automatically perform backward elimination if True
- **back_elim_p** (*float*) – p-value threshold for backward elimination

show(*plot_type='residual'*)
Create a Residual or Probability Plot

Parameters **plot_type** (*str*) – Either "residual" or "prob"

Returns The number of the newly created matplotlib figure

Return type int

class dvhastats.ui.**PCAUI**(*X*, *var_names=None*, *n_components=0.95*, *transform=True*, ***kwargs*)
Bases: *dvhastats.ui.DVHAStatsBaseClass*, *dvhastats.stats.PCA*

Hotelling's t-squared statistic for multivariate hypothesis testing

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – Training data, where n_samples is the number of samples and n_features is the number of features.
- **var_names** (*str, optional*) – Names of the independent variables in X
- **n_components** (*int, float, None or str*) – Number of components to keep. if n_components is not set all components are kept: n_components == min(n_samples, n_features) If n_components == 'mle' and svd_solver == 'full', Minka's MLE is used to guess the dimension. Use of n_components == 'mle' will interpret svd_solver == 'auto' as svd_solver == 'full'. If 0 < n_components < 1 and svd_solver == 'full', select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components. If svd_solver == 'arpack', the number of components must be strictly less than the minimum of n_features and n_samples.
- **transform** (*bool*) – Fit the model and apply the dimensionality reduction
- **kwargs** (*any*) – Provide any keyword arguments for sklearn.decomposition.PCA: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

show(*plot_type='feature_map'*, *absolute=True*)
Create a heat map of PCA components

Parameters

- **plot_type** (*str*) – Select a plot type to display. Options include: feature_map.
- **absolute** (*bool*) – Heat map will display the absolute values in PCA components if True

Returns The number of the newly created matplotlib figure

Return type int

class dvhastats.ui.**RiskAdjustedControlChartUI**(*X*, *y*, *std=3*, *ucl_limit=None*, *lcl_limit=None*, *x=None*, *y_name=None*, *var_names=None*, *saved_reg=None*, *plot_title=None*, *back_elim=False*, *back_elim_p=0.05*)
Bases: *dvhastats.ui.DVHAStatsBaseClass*, *dvhastats.stats.RiskAdjustedControlChart*

Risk-Adjusted Control Chart using a Multi-Variable Regression

Parameters

- **X** (*array-like*) – Input array (independent data)
- **y** (*list, np.ndarray*) – 1-D Input data (dependent data)

- **std** (*int, float, optional*) – Number of standard deviations used to calculate if a y-value is out-of-control.
- **ucl_limit** (*float, optional*) – Limit the upper control limit to this value
- **lcl_limit** (*float, optional*) – Limit the lower control limit to this value
- **x** (*list, np.ndarray, optional*) – x-axis values
- **plot_title** (*str, optional*) – Over-ride the plot title
- **saved_reg** (MultiVariableRegression, *optional*) – Optionally provide a previously calculated regression
- **var_names** (*list, optional*) – Optionally provide names of the variables
- **back_elim** (*bool*) – Automatically perform backward elimination if True
- **back_elim_p** (*float*) – p-value threshold for backward elimination

show()
Display the risk-adjusted control chart with matplotlib

Returns The number of the newly created matplotlib figure

Return type int

4.2 stats module

Statistical calculations and class objects

class dvhastats.stats.**ControlChart**(*y*, *std=3*, *ucl_limit=None*, *lcl_limit=None*, *x=None*)
Bases: object

Calculate control limits for a standard univariate Control Chart"

Parameters

- **y** (*list, np.ndarray*) – Input data (1-D)
- **std** (*int, float, optional*) – Number of standard deviations used to calculate if a y-value is out-of-control.
- **ucl_limit** (*float, optional*) – Limit the upper control limit to this value
- **lcl_limit** (*float, optional*) – Limit the lower control limit to this value

avg_moving_range
Avg moving range based on 2 consecutive points

Returns Average moving range. Returns NaN if arr is empty.

Return type np.ndarray, np.nan

center_line
Center line of charting data (i.e., mean value)

Returns Mean value of y with np.mean() or np.nan if y is empty

Return type np.ndarray, np.nan

chart_data
JSON compatible dict for chart generation

Returns Data used for Histogram visuals. Keys include 'x', 'y', 'out_of_control', 'center_line', 'lcl', 'ucl'

Return type dict

control_limits
Calculate the lower and upper control limits

Returns

- **lcl** (*float*) – Lower Control Limit (LCL)
- **ucl** (*float*) – Upper Control Limit (UCL)

out_of_control
Get the indices of out-of-control observations

Returns An array of indices that are not between the lower and upper control limits

Return type np.ndarray

out_of_control_high
Get the indices of observations > ucl

Returns An array of indices that are greater than the upper control limit

Return type np.ndarray

out_of_control_low
Get the indices of observations < lcl

Returns An array of indices that are less than the lower control limit

Return type np.ndarray

sigma
UCL/LCL = center_line +/- sigma * std

Returns sigma or np.nan if arr is empty

Return type np.ndarray, np.nan

class dvhastats.stats.**CorrelationMatrix**(*X*, *corr_type='Pearson'*)
Bases: object

Pearson-R correlation matrix

Parameters

- **X** (*np.ndarray*) – Input data (2-D) with N rows of observations and p columns of variables.
- **corr_type** (*str*) – Either "Pearson" or "Spearman"

chart_data
JSON compatible dict for chart generation

Returns Data used for Histogram visuals. Keys include 'corr', 'p', 'norm', 'norm_p'

Return type dict

normality
The normality and normality p-value of the input array

Returns

- **statistic** (*np.ndarray*) – Normality calculated with scipy.stats.normaltest

- **p-value** (*np.ndarray*) – A 2-sided chi squared probability for the hypothesis test.

class dvhastats.stats.**Histogram**(*y*, *bins*, *nan_policy='omit'*)

Bases: object

Basic histogram plot using matplotlib histogram calculation

Parameters

- **y** (*array-like*) – Input array.
- **bins** (*int, list, str, optional*) – If bins is an int, it defines the number of equal-width bins in the given range (10, by default). If bins is a sequence, it defines a monotonically increasing array of bin edges, including the rightmost edge, allowing for non-uniform bin widths. If bins is a string, it defines the method used to calculate the optimal bin width, as defined by histogram_bin_edges. 'auto' - Maximum of the 'sturges' and 'fd' estimators. Provides good all around performance. 'fd' - (Freedman Diaconis Estimator) Robust (resilient to outliers) estimator that takes into account data variability and data size. 'doane' - An improved version of Sturges' estimator that works better with non-normal datasets. 'scott' - Less robust estimator that that takes into account data variability and data size. 'stone' - Estimator based on leave-one-out cross-validation estimate of the integrated squared error. Can be regarded as a generalization of Scott's rule. 'rice' - Estimator does not take variability into account, only data size. Commonly overestimates number of bins required. 'sturges' - R's default method, only accounts for data size. Only optimal for gaussian data and underestimates number of bins for large non-gaussian datasets. 'sqrt' - Square root (of data size) estimator, used by Excel and other programs for its speed and simplicity.
- **nan_policy** (*str*) – Value must be one of the following: 'propagate', 'raise', 'omit' Defines how to handle when input contains nan. The following options are available (default is 'omit'): 'propagate': returns nan 'raise': throws an error 'omit': performs the calculations ignoring nan values

chart_data

JSON compatible dict for chart generation

Returns Data used for Histogram visuals. Keys include 'x', 'y', 'mean', 'median', 'std', 'normality', 'normality_p'

Return type dict

hist_data

Get the histogram data

Returns

- **hist** (*np.ndarray*) – The values of the histogram
- **center** (*np.ndarray*) – The centers of the bins

mean

The mean value of the input array

Returns Mean value of y with np.mean()

Return type np.ndarray

median

The median value of the input array

Returns Median value of y with np.median()

Return type np.ndarray

normality
The normality and normality p-value of the input array

Returns

- **statistic** (*float*) – Normality calculated with scipy.stats.normaltest
- **p-value** (*float*) – A 2-sided chi squared probability for the hypothesis test.

std
The standard deviation of the input array

Returns Standard deviation of y with np.std()

Return type np.ndarray

class dvhastats.stats.**HotellingT2**(*data*, *alpha=0.05*, *const_policy='raise'*)
Bases: object

Hotelling's t-squared statistic for multivariate hypothesis testing

Parameters

- **data** (*np.ndarray*) – A 2-D array of data to perform multivariate analysis. (e.g., DVHAStats.data)
- **alpha** (*float*) – The significance level used to calculate the upper control limit (UCL)
- **const_policy** (*str*) – {'raise', 'omit'} Defines how to handle when data is constant. The following options are available (default is 'raise'): 'raise': throws an error 'omit': exclude constant variables from calculation

Q
Calculate Hotelling T^2 statistic (Q) from a 2-D numpy array

Returns A numpy array of Hotelling T^2 (1-D of length N)

Return type np.ndarray

center_line
Center line for the control chart

Returns Median value of beta distribution.

Return type float

chart_data
JSON compatible dict for chart generation

Returns Data used for Histogram visuals. Keys include 'x', 'y', 'out_of_control', 'center_line', 'lcl', 'ucl'

Return type dict

control_limits
Lower and Upper control limits

Returns

- **lcl** (*float*) – Lower Control Limit (LCL). This is fixed to 0 for Hotelling T2
- **ucl** (*float*) – Upper Control Limit (UCL)

get_control_limit(*x*)
Calculate a Hotelling T^2 control limit using a beta distribution

Parameters **x** (*float*) – Value where the beta function is evaluated

Returns The control limit for a beta distribution

Return type float

observations
Number of observations in data

Returns Number of rows in data

Return type int

out_of_control
Indices of out-of-control observations

Returns An array of indices that are greater than the upper control limit. (NOTE: Q is never negative)

Return type np.ndarray

ucl
Upper control limit

Returns **ucl** – Upper Control Limit (UCL)

Return type float

variable_count
Number of variables in data

Returns Number of columns in data

Return type int

class dvhastats.stats.**MultiVariableRegression**(*X*, *y*, *saved_reg=None*, *var_names=None*, *y_var_name=None*, *back_elim=False*, *back_elim_p=0.05*)

Bases: object

Multi-variable regression using scikit-learn

Parameters

- **X** (*array-like*) – Independent data
- **y** (*array-like*) – Dependent data
- **saved_reg** (MultiVariableRegression, *optional*) – Optionally provide a previously calculated regression
- **var_names** (*list, optional*) – Optionally provide names of the variables
- **y_var_name** (*int, str, optional*) – Optionally provide name of the dependent variable
- **back_elim** (*bool*) – Automatically perform backward elimination if True
- **back_elim_p** (*float*) – p-value threshold for backward elimination

backward_elimination(*p_value=0.05*)
Remove insignificant variables from regression

p_value [float] Iteratively remove the least significant variable until all variables have p-values less than p_value or only one variable remains.

chart_data
JSON compatible dict for chart generation

Returns Data used for residual visuals. Keys include 'x', 'y', 'pred', 'resid', 'coef', 'r_sq', 'mse', 'std_err', 't_value', 'p_value'

Return type dict

coef
Coefficients for the regression

Returns An array of regression coefficients (i.e., y_intercept, 1st var slope, 2nd var slope, etc.)

Return type np.ndarray

df_error
Error degrees of freedom

Returns Degrees of freedom for the error

Return type int

df_model
Model degrees of freedom

Returns Degrees of freedom for the model

Return type int

f_p_value
p-value of the f-statistic

Returns p-value of the F-statistic of beta coefficients using scipy

Return type float

f_stat
The F-statistic of the regression

Returns F-statistic of beta coefficients using regressors.stats

Return type float

mse
Mean squared error of the linear regression

Returns A non-negative floating point value (the best value is 0.0), or an array of floating point values, one for each individual target.

Return type float, nd.array

prob_plot
Calculate quantiles for a probability plot

Returns Data for generating a probablity plot. Keys include: 'x', 'y', 'y_intercept', 'slope', 'x_trend', 'y_trend'

Return type dict

r_sq
R^2 (coefficient of determination) regression score function.

Returns The R^2 score

Return type float

residuals
Residuals of the prediction and sample data

Returns y - predictions

Return type np.ndarray

slope
The slope of the linear regression

Returns Estimated coefficients for the linear regression problem. If multiple targets are passed during the fit (y 2D), this is a 2D array of shape (n_targets, n_features), while if only one target is passed, this is a 1D array of length n_features.

Return type np.ndarray

y_intercept
The y-intercept of the linear regression

Returns Independent term in the linear model.

Return type float

class dvhastats.stats.**PCA**(*X*, *var_names=None*, *n_components=0.95*, *transform=True*, ***kwargs*)
Bases: sklearn.decomposition._pca.PCA

Principal Component Analysis with sklearn.decomposition.PCA

Parameters

- **X** (*np.ndarray*) – Training data (2-D), where n_samples is the number of samples and n_features is the number of features. shape (n_samples, n_features)
- **var_names** (*list, optional*) – Optionally provide names of the features
- **n_components** (*int, float, None or str*) – Number of components to keep. if n_components is not set all components are kept: n_components == min(n_samples, n_features) If n_components == 'mle' and svd_solver == 'full', Minka's MLE is used to guess the dimension. Use of n_components == 'mle' will interpret svd_solver == 'auto' as svd_solver == 'full'. If 0 < n_components < 1 and svd_solver == 'full', select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components. If svd_solver == 'arpack', the number of components must be strictly less than the minimum of n_features and n_samples.
- **transform** (*bool*) – Fit the model and apply the dimensionality reduction
- **kwargs** (*any*) – Provide any keyword arguments for sklearn.decomposition.PCA: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

component_labels
Get component names

Returns Labels for plotting. (1st Comp, 2nd Comp, 3rd Comp, etc.)

Return type list

feature_map_data
Used for feature analysis heat map

Returns Principal axes in feature space, representing the directions of maximum variance in the data. The components are sorted by explained_variance.

Return type np.ndarray Shape (n_components, n_features)

class dvhastats.stats.**RiskAdjustedControlChart**(*X*, *y*, *std=3*, *ucl_limit=None*, *lcl_limit=None*, *x=None*, *saved_reg=None*, *var_names=None*, *back_elim=False*, *back_elim_p=0.05*)
Bases: *dvhastats.stats.ControlChart*

Calculate a risk-adjusted univariate Control Chart (with linear MVR)

Parameters

- **X** (*array-like*) – Independent data
- **y** (*list, np.ndarray*) – Input data (1-D)
- **std** (*int, float, optional*) – Number of standard deviations used to calculate if a y-value is out-of-control.
- **ucl_limit** (*float, optional*) – Limit the upper control limit to this value
- **lcl_limit** (*float, optional*) – Limit the lower control limit to this value
- **saved_reg** (MultiVariableRegression, *optional*) – Optionally provide a previously calculated regression
- **var_names** (*list, optional*) – Optionally provide names of the variables
- **back_elim** (*bool*) – Automatically perform backward elimination if True
- **back_elim_p** (*float*) – p-value threshold for backward elimination

dvhastats.stats.**avg_moving_range**(*arr*, *nan_policy='omit'*)

Calculate the average moving range (over 2-consecutive point1)

Parameters

- **arr** (*array-like (1-D)*) – Input array. Must be positive 1-dimensional.
- **nan_policy** (*str, optional*) – Value must be one of the following: {'propagate', 'raise', 'omit'} Defines how to handle when input contains nan. The following options are available (default is 'omit'): 'propagate': returns nan 'raise': throws an error 'omit': performs the calculations ignoring nan values

Returns Average moving range. Returns NaN if arr is empty

Return type np.ndarray, np.nan

dvhastats.stats.**box_cox**(*arr*, *alpha=None*, *lmbda=None*, *const_policy='propagate'*)

Parameters

- **arr** (*np.ndarray*) – Input array. Must be positive 1-dimensional.
- **lmbda** (*None, scalar, optional*) – If lmbda is not None, do the transformation for that value. If lmbda is None, find the lambda that maximizes the log-likelihood function and return it as the second output argument.
- **alpha** (*None, float, optional*) – If alpha is not None, return the 100 * (1-alpha)% confidence interval for lmbda as the third output argument. Must be between 0.0 and 1.0.
- **const_policy** (*str*) – {'propagate', 'raise', 'omit'} Defines how to handle when data is constant. The following options are available (default is 'propagate'): 'propagate': returns nan 'raise': throws an error

Returns **box_cox** – Box-Cox power transformed array

Return type np.ndarray

dvhastats.stats.**get_lin_reg_p_values**(*X*, *y*, *predictions*, *y_intercept*, *slope*)

Get p-values of a linear regression 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, list*) – Dependent data
- **predictions** (*np.ndarray, list*) – Predictions using the linear regression. (Output from linear_model.LinearRegression.predict)
- **y_intercept** (*float, np.ndarray*) – The y-intercept of the linear regression
- **slope** (*float, np.ndarray*) – The slope of the linear regression

Returns

- **p_value** (*np.ndarray*) – p-value of the linear regression coefficients
- **std_errs** (*np.ndarray*) – standard errors of the linear regression coefficients
- **t_value** (*np.ndarray*) – t-values of the linear regression coefficients

dvhastats.stats.**get_ordinal**(*n*)
Convert number to its ordinal (e.g., 1 to 1st)

Parameters **n** (*int*) – Number to be converted to ordinal

Returns the ordinal of n

Return type str

dvhastats.stats.**is_arr_constant**(*arr*)
Determine if data by var_name is constant

Parameters **arr** (*array-like*) – Input array or object that can be converted to an array

Returns True if all values the same (i.e., no variation)

Return type bool

dvhastats.stats.**is_nan_arr**(*arr*)
Check if array has only NaN elements

Parameters **arr** (*np.ndarray*) – Input array

Returns True if all elements are np.nan

Return type bool

dvhastats.stats.**moving_avg**(*y*, *avg_len*, *x=None*, *weight=None*)
Calculate the moving (rolling) average of a set of data

Parameters

- **y** (*array-like*) – data (1-D) to be averaged
- **avg_len** (*int*) – Data is averaged over this many points (current value and avg_len - 1 prior points)
- **x** (*np.ndarray, list, optional*) – Optionally specify the x-axis values. Otherwise index+1 is used.
- **weight** (*np.ndarray, list, optional*) – A weighted moving average is calculated based on the provided weights. weight must be of same length as y. Weights of one are assumed by default.

Returns

- **x** (*np.ndarray*) – Resulting x-values for the moving average
- **moving_avg** (*np.ndarray*) – moving average values

dvhastats.stats.**pearson_correlation_matrix**(*X*)
Calculate a correlation matrix of Pearson-R values

Parameters **X** (*array-like, shape (n_samples, n_features)*) – Training data, where n_samples is the number of samples and n_features is the number of features.

Returns

- **r** (*np.ndarray*) – Array (2-D) of Pearson-R correlations between the row indexed and column indexed variables
- **p** (*np.ndarray*) – Array (2-D) of p-values associated with r

dvhastats.stats.**process_nan_policy**(*arr*, *nan_policy*)
Calculate the average moving range (over 2-consecutive point1)

Parameters

- **arr** (*array-like (1-D)*) – Input array. Must be positive 1-dimensional.
- **nan_policy** (*str*) – Value must be one of the following: {'propagate', 'raise', 'omit'} Defines how to handle when input contains nan. The following options are available (default is 'omit'): 'propagate': returns nan 'raise': throws an error 'omit': performs the calculations ignoring nan values

Returns Input array evaluated per nan_policy

Return type np.ndarray, np.nan

dvhastats.stats.**remove_const_column**(*arr*)
Remove all columns with zero variance

Parameters **arr** (*np.ndarray*) – Input array (2-D)

Returns Input array with columns of a constant value removed

Return type np.ndarray

dvhastats.stats.**remove_nan**(*arr*)
Remove indices from 1-D array with values of np.nan

Parameters **arr** (*np.ndarray (1-D)*) – Input array. Must be positive 1-dimensional.

Returns arr with NaN values deleted

Return type np.ndarray

dvhastats.stats.**spearman_correlation_matrix**(*X*, *nan_policy='omit'*)
Calculate a Spearman correlation matrix

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – Training data, where n_samples is the number of samples and n_features is the number of features.
- **nan_policy** (*str*) – Value must be one of the following: 'propagate', 'raise', 'omit' Defines how to handle when input contains nan. The following options are available (default is 'omit'): 'propagate': returns nan 'raise': throws an error 'omit': performs the calculations ignoring nan values

Returns

- **correlation** (*float or ndarray (2-D square)*) – Spearman correlation matrix or correlation coefficient (if only 2 variables are given as parameters. Correlation matrix is square with length equal to total number of variables (columns or rows) in a and b combined.

- **p-value** (*float*) – The two-sided p-value for a hypothesis test whose null hypothesis is that two sets of data are uncorrelated, has same dimension as rho.

4.3 plot module

Basic plotting class objects for DVHA-Stats based on matplotlib

class dvhastats.plot.**BoxPlot**(*data*, *title='Box and Whisker'*, *xlabel=''*, *ylabel=''*, *xlabels=None*, ***kwargs*)

Bases: *dvhastats.plot.DistributionChart*

Box and Whisker plotting class object

Parameters

- **data** (*array-like*) – Input array (1-D or 2-D)
- **title** (*str, optional*) – Set the plot title
- **xlabel** (*str, optional*) – Set the x-axis title
- **xlabels** (*array-like, optional*) – Set the xtick labels (e.g., variable names for each box plot)
- **ylabel** (*str, optional*) – Set the y-axis title
- **kwargs** (*any, optional*) – Any keyword argument may be set per matplotlib histogram: https://matplotlib.org/3.3.1/api/_as_gen/matplotlib.pyplot.boxplot.html

class dvhastats.plot.**Chart**(*title=None*, *fig_init=True*)

Bases: object

Base class for charts

Parameters

- **title** (*str, optional*) – Set the title suptitle
- **fig_init** (*bool*) – Automatically call pyplot.figure, store in Chart.figure

activate()

Activate this figure

close()

Close this figure

show()

Display this figure

class dvhastats.plot.**ControlChart**(*y*, *out_of_control*, *center_line*, *lcl=None*, *ucl=None*, *title='Control Chart'*, *xlabel='Observation'*, *ylabel='Charting Variable'*, *line_color='black'*, *line_width=0.75*, *center_line_color='black'*, *center_line_width=1.0*, *center_line_style='–'*, *limit_line_color='red'*, *limit_line_width=1.0*, *limit_line_style='–'*, ***kwargs*)

Bases: *dvhastats.plot.Plot*

ControlChart class object

Parameters

- **y** (*np.ndarray, list*) – Charting data

- **out_of_control** (*np.ndarray, list*) – The indices of y that are out-of-control
- **center_line** (*float, np.ndarray*) – The center line value (e.g., np.mean(y))
- **lcl** (*float, optional*) – The lower control limit (LCL). Line omitted if lcl is None.
- **ucl** (*float, optional*) – The upper control limit (UCL). Line omitted if ucl is None.
- **title** (*str*) – Set the plot title
- **xlabel** (*str*) – Set the x-axis title
- **ylabel** (*str*) – Set the y-axis title
- **line_color** (*str, optional*) – Specify the line color
- **line_width** (*float, int*) – Specify the line width
- **kwargs** (*any*) – Any additional keyword arguments applicable to the Plot class

add_center_line(*color=None*, *line_width=None*, *line_style=None*)
Add the center line to the plot

add_control_limit_line(*limit*, *color=None*, *line_width=None*, *line_style=None*)
Add a control limit line to plot

add_scatter()
Set scatter data, add in- and out-of-control circles

class dvhastats.plot.**DistributionChart**(*data*, *title='Chart'*, *xlabel='Bins'*, *ylabel='Counts'*, ***kwargs*)

Bases: *dvhastats.plot.Chart*

Distribution plotting class object (base for histogram / boxplot

Parameters

- **data** (*array-like*) – Input array (1-D or 2-D)
- **title** (*str*) – Set the plot title
- **xlabel** (*str*) – Set the x-axis title
- **ylabel** (*str*) – Set the y-axis title
- **kwargs** (*any*) – Any keyword argument may be set per matplotlib histogram: https://matplotlib.org/3.3.1/api/_as_gen/matplotlib.pyplot.hist.html

class dvhastats.plot.**HeatMap**(*X*, *xlabels=None*, *ylabels=None*, *title=None*, *cmap='viridis'*, *show=True*)

Bases: *dvhastats.plot.Chart*

Create a heat map using matplotlib.pyplot.matshow

Parameters

- **X** (*np.ndarray*) – Input data (2-D) with N rows of observations and p columns of variables.
- **xlabels** (*list, optional*) – Optionally set the variable names with a list of str
- **ylabels** (*list, optional*) – Optionally set the variable names with a list of str
- **title** (*str, optional*) – Set the title suptitle
- **cmap** (*str*) – matplotlib compatible color map
- **show** (*bool*) – Automatically show the figure

class dvhastats.plot.**Histogram**(*data*, *bins=10*, *title='Histogram'*, *xlabel='Bins'*, *ylabel='Counts'*, ***kwargs*)

Bases: *dvhastats.plot.DistributionChart*

Histogram plotting class object

Parameters

- **data** (*array-like*) – Input array (1-D)
- **bins** (*int, sequence, str*) – default: rcParams["hist.bins"] (default: 10) If bins is an integer, it defines the number of equal-width bins in the range. If bins is a sequence, it defines the bin edges, including the left edge of the first bin and the right edge of the last bin; in this case, bins may be unequally spaced. All but the last (righthand-most) bin is half-open. In other words, if bins is: [1, 2, 3, 4] then the first bin is [1, 2) (including 1, but excluding 2) and the second [2, 3). The last bin, however, is [3, 4], which includes 4. If bins is a string, it is one of the binning strategies supported by numpy.histogram_bin_edges: 'auto', 'fd', 'doane', 'scott', 'stone', 'rice', 'sturges', or 'sqrt'.
- **title** (*str*) – Set the plot title
- **xlabel** (*str*) – Set the x-axis title
- **ylabel** (*str*) – Set the y-axis title
- **kwargs** (*any*) – Any keyword argument may be set per matplotlib histogram: https://matplotlib.org/3.3.1/api/_as_gen/matplotlib.pyplot.hist.html

class dvhastats.plot.**PCAFeatureMap**(*X*, *features=None*, *cmap='viridis'*, *show=True*, *title='PCA Feature Heat Map'*)

Bases: *dvhastats.plot.HeatMap*

Specialized Heat Map for PCA feature evaluation

Parameters

- **X** (*np.ndarray*) – Input data (2-D) with N rows of observations and p columns of variables.
- **features** (*list, optional*) – Optionally set the feature names with a list of str
- **title** (*str, optional*) – Set the title suptitle
- **cmap** (*str*) – matplotlib compatible color map
- **show** (*bool*) – Automatically show the figure

get_comp_labels(*n_components*)

Get ylabels for HeatMap

static get_ordinal(*n*)

Convert number to its ordinal (e.g., 1 to 1st)

Parameters **n** (*int*) – Number to be converted to ordinal

Returns the ordinal of n

Return type str

class dvhastats.plot.**Plot**(*y*, *x=None*, *show=True*, *title='Chart'*, *xlabel='Independent Variable'*, *ylabel='Dependent Variable'*, *line=True*, *line_color=None*, *line_width=1.0*, *line_style='-'*, *scatter=True*, *scatter_color=None*)

Bases: *dvhastats.plot.Chart*

Generic plotting class with matplotlib

Parameters

- **y** (*np.ndarray, list*) – The y data to be plotted (1-D only)
- **x** (*np.ndarray, list, optional*) – Optionally specify the x-axis values. Otherwise index+1 is used.
- **show** (*bool*) – Automatically plot the data if True
- **title** (*str*) – Set the plot title
- **xlabel** (*str*) – Set the x-axis title
- **ylabel** (*str*) – Set the y-axis title
- **line** (*bool*) – Plot the data as a line series
- **line_color** (*str, optional*) – Specify the line color
- **line_width** (*float, int*) – Specify the line width
- **line_style** (*str*) – Specify the line style
- **scatter** (*bool*) – Plot the data as a scatter plot (circles)
- **scatter_color** (*str, optional*) – Specify the scatter plot circle color

add_default_line()
Add line data to figure

add_line(*y*, *x=None*, *line_color=None*, *line_width=None*, *line_style=None*)
Add another line with the provided data

Parameters

- **y** (*np.ndarray, list*) – The y data to be plotted (1-D only)
- **x** (*np.ndarray, list, optional*) – Optionally specify the x-axis values. Otherwise index+1 is used.
- **line_color** (*str, optional*) – Specify the line color
- **line_width** (*float, int*) – Specify the line width
- **line_style** (*str*) – Specify the line style

add_scatter()
Add scatter data to figure

dvhastats.plot.**get_new_figure_num**()
Get a number for a new matplotlib figure

Returns Figure number

Return type int

4.4 utilities module

Common functions for the DVHA-Stats.

dvhastats.utilities.**apply_dtype**(*value*, *dtype*)
Convert value with the provided data type

Parameters

- **value** (*any*) – Value to be converted

- **dtype** (*function, None*) – python reserved types, e.g., int, float, str, etc. However, dtype could be any callable that raises a ValueError on failure.

Returns The return of dtype(value) or numpy.nan on ValueError

Return type any

dvhastats.utilities.**csv_to_dict**(*csv_file_path*, *delimiter=','*, *dtype=None*, *header_row=True*)
Read in a csv file, return data as a dictionary

Parameters

- **csv_file_path** (*str*) – File path to the CSV file to be processed.
- **delimiter** (*str*) – Specify the delimiter used in the csv file (default = ',')
- **dtype** (*callable, type, optional*) – Optionally force values to a type (e.g., float, int, str, etc.).
- **header_row** (*bool, optional*) – If True, the first row is interpreted as column keys, otherwise row indices will be used

Returns CSV data as a dict, using the first row values as keys

Return type dict

dvhastats.utilities.**dict_to_array**(*data*, *key_order=None*)
Convert a dict of data to a numpy array

Parameters

- **data** (*dict*) – Dictionary of data to be converted to np.array.
- **key_order** (*None, list of str*) – Optionally the order of columns

Returns A dictionary with keys of 'data' and 'columns', pointing to a numpy array and list of str, respectively

Return type dict

dvhastats.utilities.**get_sorted_indices**(*list_data*)
Get original indices of a list after sorting

Parameters **list_data** (*list*) – Any python sortable list

Returns list_data indices of sorted(list_data)

Return type list

dvhastats.utilities.**import_data**(*data*, *var_names=None*)
Generalized data importer for np.ndarray, dict, and csv file

Parameters

- **data** (*numpy.array, dict, str*) – Input data (2-D) with N rows of observations and p columns of variables. The CSV file must have a header row for column names.
- **var_names** (*list of str, optional*) – If data is a numpy array, optionally provide the column names.

Returns A tuple: data as an array and variable names as a list

Return type np.ndarray, list

dvhastats.utilities.**is_numeric**(*val*)
Check if value is numeric (float or int)

Parameters **val** (*any*) – Any value

Returns Returns true if float(val) doesn't raise a ValueError

Return type bool

dvhastats.utilities.**sort_2d_array**(*arr*, *index*, *mode='col'*)

Sort a 2-D numpy array

Parameters

- **arr** (*np.ndarray*) – Input 2-D array to be sorted
- **index** (*int, list*) – Index of column or row to sort arr. If list, will sort by each index in the order provided.
- **mode** (*str*) – Either 'col' or 'row'

dvhastats.utilities.**str_arr_to_date_arr**(*arr*, *date_parser_kwargs=None*, *force=False*)

Convert an array of datetime strings to a list of datetime objects

Parameters

- **arr** (*array-like*) – Array of datetime strings compatible with dateutil.parser.parse
- **date_parser_kwargs** (*dict, optional*) – Keyword arguments to be passed into dateutil.parser.parse
- **force** (*bool*) – If true, failed parsings will result in original value. If false, dateutil.parser.parse's error will be raised on failures.

Returns list of datetime objects

Return type list

dvhastats.utilities.**widen_data**(*data_dict*, *uid_columns*, *x_data_cols*, *y_data_col*, *date_col=None*, *sort_by_date=True*, *remove_partial_columns=False*, *multi_val_policy='first'*, *dtype=None*, *date_parser_kwargs=None*)

Convert a narrow data dictionary into wide format (i.e., from one row per dependent value to one row per observation)

Parameters

- **data_dict** (*dict*) – Data to be converted. The length of each array must be uniform.
- **uid_columns** (*list*) – Keys of data_dict used to create an observation uid
- **x_data_cols** (*list*) – Keys of columns representing independent data
- **y_data_col** (*int, str*) – Key of data_dict representing dependent data
- **date_col** (*int, str, optional*) – Key of date column
- **sort_by_date** (*bool, optional*) – Sort output by date (date_col required)
- **remove_partial_columns** (*bool, optional*) – If true, any columns that have a blank row will be removed
- **multi_val_policy** (*str*) – Either 'first', 'last', 'min', 'max'. If multiple values are found for a particular combination of x_data_cols, one value will be selected based on this policy.
- **dtype** (*function*) – python reserved types, e.g., int, float, str, etc. However, dtype could be any callable that raises a ValueError on failure.
- **date_parser_kwargs** (*dict, optional*) – Keyword arguments to be passed into dateutil.parser.parse

Returns data_dict reformatted to one row per UID

Return type dict

CHAPTER 5

Credits

5.1 Development Lead

- Dan Cutright

5.2 Contributors

- Arkajyoti Roy
- Aditya Panchal

CHAPTER 6

Change log for IQDM-PDF

6.1 v0.2.5 (TBD)

- Machine Learning module (WIP)

6.2 v0.2.4 (2021.03.04)

- Update for IQDM Analytics compatibility

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

S

U

V

W

Y