PaDuA Documentation

Release 0.1.2

Martin Fitzpatrick

Apr 15, 2017

Contents

PaDuA

Proteomic Data Analysis

PaDuA is a Python package to simplify the processing and analysis of quantified proteomics data. Currently it supports processing and analysis of MaxQuant outputs, providing many of the features available in the GUI analysis tool Perseus. By scripting these processing and analysis steps you can get to your results more quickle and reproducibly.

CHAPTER 1

Getting Started

Installation

The following instructions should allow you to get PaDuA up and running on your Python installation.

Windows

Install Python 2.7 or 3.4 Windows installer from the Python download site.

You can get Windows binaries for most required Python libraries from the Pythonlibs library. At a minimum you will need to install NumPy, SciPy, Scikit-Learn and Matplotlib. Make sure that the installed binaries match the architecture (32bit/64bit) and the installed Python version.

With those installed you should be able to install the latest release of PaDuA with:

```
pip install padua
```

Windows Using Anaconda

Install Anaconda for Windows. Link to the website is http://continuum.io/downloads. Make the decision at this point whether to use 64bit or 32bit versions and stick to it.

Anaconda will install many useful packages for you by default. Open the Anaconda command prompt and ensure they are setup with:

```
conda install numpy scipy scikit-learn matplotlib
```

With those installed you should be able to install the latest release of PaDuA with:

```
pip install padua
```

MacOS X

The simplest approach to setting up a development environment is through the MacOS X package manager Homebrew. It should be feasible to build all these tools from source, but I'd strongly suggest you save yourself the bother.

Install Homebrew as follows:

```
ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)"
```

Ensure Python 2.7 or 3.4 is installed:

```
brew install python
```

Or:

```
brew install python3
```

Next use pip to install all required Python packages. This can be done in a one liner with pip:

```
pip install numpy scipy pandas matplotlib scikit-learn
```

With those installed you should be able to install the latest release of PaDuA with:

```
pip install padua
```

MacOS X Using Anaconda

Install Anaconda for MacOS X. Link to the website is http://continuum.io/downloads.

Anaconda will install many useful packages for you by default. Open the Anaconda command prompt and ensure they are setup with:

```
conda install numpy scipy scikit-learn matplotlib
```

With those installed you should be able to install the latest release of PaDuA with:

```
pip install padua
```

Linux

For Python3 install the following packages:

```
sudo apt-get install g++ python3 python3-dev python3-pip git gfortran libzmq-dev
sudo apt-get install python3-matplotlib python3-requests python3-numpy python3-scipy
```

You can also install the other packages using pip3 (the names on PyPi are the same as for the packages minus the python3- prefix). Once installation of the above has completed you're ready to go.

With those installed you should be able to install the latest release of PaDuA with:

```
pip3 install padua
```

API Reference

The API reference lists all modules and funtions of the PaDuA package.

Analysis

`padua.analysis.`**`anova_1way`**(*df*, **args*, *fdr=0.05*)

Perform Analysis of Variation (ANOVA) on provided dataframe and for specified groups. Groups for analysis can be specified as individual arguments, e.g.

anova(df, “Group A”, “Group B”) anova(df, (“Group A”, 5), (“Group B”, 5))

At least 2 groups must be provided.

Returns Dataframe containing selected groups and P/T/sig value for the comparisons.

`padua.analysis.`**`correlation`**(*df*, *rowvar=False*)

Calculate column-wise Pearson correlations using `numpy.ma.corrcoef`

Input data is masked to ignore NaNs when calculating correlations. Data is returned as a Pandas `DataFrame` of column_n x column_n dimensions, with column index copied to both axes.

Parameters **`df`** – Pandas DataFrame

Returns Pandas DataFrame (n_columns x n_columns) of column-wise correlations

`padua.analysis.`**`enrichment_from_evidence`**(*dfe*, *modification='Phospho (STY)'*)

Calculate relative enrichment of peptide modifications from evidence.txt.

Taking a modifiedsitepeptides `DataFrame` returns the relative enrichment of the specified modification in the table.

The returned data columns are generated from the input data columns.

Parameters **`df`** – Pandas `DataFrame` of evidence

Returns Pandas `DataFrame` of percentage modifications in the supplied data.

`padua.analysis.`**`enrichment_from_msp`**(*dfmsp*, *modification='Phospho (STY)'*)

Calculate relative enrichment of peptide modifications from modificationSpecificPeptides.txt.

Taking a modifiedsitepeptides `DataFrame` returns the relative enrichment of the specified modification in the table.

The returned data columns are generated from the input data columns.

Parameters **`df`** – Pandas `DataFrame` of modificationSpecificPeptides

Returns Pandas `DataFrame` of percentage modifications in the supplied data.

`padua.analysis.`**`go_enrichment`**(*df*, *enrichment='function'*, *organism='Homo sapiens'*, *summary=True*, *fdr=0.05*, *ids_from=['Proteins', 'Protein IDs']*)

Calculate gene ontology (GO) enrichment for a specified set of indices, using the PantherDB GO enrichment service.

Provided with a processed data `DataFrame` will calculate the GO ontology enrichment specified by *enrichment*, for the specified *organism*. The IDs to use for genes are taken from the field *ids_from*, which by default is compatible with both proteinGroups and modified peptide tables. Setting the *fdr* parameter (default=0.05) sets the cut-off to use for filtering the results. If *summary* is `True` (default) the returned `DataFrame` contains just the ontology summary and FDR.

Parameters

- **df** – Pandas `DataFrame` to
- **enrichment** – `str` GO enrichment method to use (one of 'function', 'process', 'cellular_location', 'protein_class', 'pathway')
- **organism** – `str` organism name (e.g. "Homo sapiens")
- **summary** – `bool` return full, or summarised dataset
- **fdr** – `float` FDR cut-off to use for returned GO enrichments
- **ids_from** – `list` of `str` containing the index levels to select IDs from (genes, protein IDs, etc.) default=['Proteins','Protein IDs']

Returns Pandas `DataFrame` containing enrichments, sorted by P value.

`padua.analysis.`**`modifiedaminoacids`**(*df*)

Calculate the number of modified amino acids in supplied `DataFrame`.

Returns the total of all modifications and the total for each amino acid individually, as an `int` and a `dict` of `int`, keyed by amino acid, respectively.

Parameters **df** – Pandas `DataFrame` containing processed data.

Returns total_aas `int` the total number of all modified amino acids quants `dict` of `int` keyed by amino acid, giving individual counts for each aa.

`padua.analysis.`**`pca`**(*df*, *n_components=2*, *mean_center=False*, ***kwargs*)

Principal Component Analysis, based on *sklearn.decomposition.PCA*

Performs a principal component analysis (PCA) on the supplied dataframe, selecting the first `n_components` components in the resulting model. The model scores and weights are returned.

For more information on PCA and the algorithm used, see the scikit-learn documentation.

Parameters

- **df** – Pandas `DataFrame` to perform the analysis on
- **n_components** – `int` number of components to select
- **mean_center** – `bool` mean center the data before performing PCA
- **kwargs** – additional keyword arguments to *sklearn.decomposition.PCA*

Returns scores `DataFrame` of PCA scores n_components x n_samples weights `DataFrame` of PCA weights n_variables x n_components

`padua.analysis.`**`plsda`**(*df*, *a*, *b*, *n_components=2*, *mean_center=False*, *scale=True*, ***kwargs*)

Partial Least Squares Discriminant Analysis, based on *sklearn.cross_decomposition.PLSRegression*

Performs a binary group partial least squares discriminant analysis (PLS-DA) on the supplied dataframe, selecting the first `n_components`.

Sample groups are defined by the selectors `a` and `b` which are used to select columns from the supplied dataframe. The result model is applied to the entire dataset, projecting non-selected samples into the same space.

For more information on PLS regression and the algorithm used, see the scikit-learn documentation.

Parameters

- **df** – Pandas `DataFrame` to perform the analysis on
- **a** – Column selector for group a
- **b** – Column selector for group b

- **`n_components`** – `int` number of components to select
- **`mean_center`** – `bool` mean center the data before performing PLS regression
- **`kwargs`** – additional keyword arguments to *sklearn.cross_decomposition.PLSRegression*

Returns scores `DataFrame` of PLSDA scores n_components x n_samples weights `DataFrame` of PLSDA weights n_variables x n_components

`padua.analysis.`**`plsr`**(*df*, *v*, *n_components=2*, *mean_center=False*, *scale=True*, ***kwargs*)

Partial Least Squares Regression Analysis, based on *sklearn.cross_decomposition.PLSRegression*

Performs a partial least squares regression (PLS-R) on the supplied dataframe `df` against the provided continuous variable `v`, selecting the first `n_components`.

For more information on PLS regression and the algorithm used, see the scikit-learn documentation.

Parameters

- **`df`** – Pandas `DataFrame` to perform the analysis on
- **`v`** – Continuous variable to perform regression against
- **`n_components`** – `int` number of components to select
- **`mean_center`** – `bool` mean center the data before performing PLS regression
- **`kwargs`** – additional keyword arguments to *sklearn.cross_decomposition.PLSRegression*

Returns scores `DataFrame` of PLS-R scores n_components x n_samples weights `DataFrame` of PLS-R weights n_variables x n_components

`padua.analysis.`**`sitespeptidesproteins`**(*df*, *site_localization_probability=0.75*)

Generate summary count of modified sites, peptides and proteins in a processed dataset `DataFrame`.

Returns the number of sites, peptides and proteins as calculated as follows:

- *sites* (>0.75; or specified site localization probability) count of all sites > threshold
- *peptides* the set of *Sequence windows* in the dataset (unique peptides)
- *proteins* the set of unique leading proteins in the dataset

Parameters

- **`df`** – Pandas `DataFrame` of processed data
- **`site_localization_probability`** – `float` site localization probability threshold (for sites calculation)

Returns `tuple` of `int`, containing sites, peptides, proteins

Annotations

Filters

`padua.filters.`**`filter_exclude`**(*df*, *s*)

Filter dataframe to exclude matching columns, based on search for "s"

Parameters **`s`** – string to search for, exclude matching columns

`padua.filters.`**`filter_intensity`**(*df*, *label=''*)

Filter to include only the Intensity values with optional specified label, excluding other Intensity measurements, but retaining all other columns.

`padua.filters.`**`filter_intensity_lfq`**(*df*, *label=''*)

Filter to include only the Intensity values with optional specified label, excluding other Intensity measurements, but retaining all other columns.

`padua.filters.`**`filter_localization_probability`**(*df*, *threshold=0.75*)

Remove rows with a localization probability below 0.75

Return a `DataFrame` where the rows with a value < *threshold* (default 0.75) in column 'Localization prob' are removed. Filters data to remove poorly localized peptides (non Class-I by default).

Parameters

- **`df`** – Pandas `DataFrame`
- **`threshold`** – Cut-off below which rows are discarded (default 0.75)

Returns Pandas `DataFrame`

`padua.filters.`**`filter_select_columns`**(*df*, *columns*)

Filter dataframe to include specified columns, retaining any Intensity columns.

`padua.filters.`**`minimum_valid_values_in_any_group`**(*df*, *levels=None*, *n=1*, *invalid=<Mock id='140225958643864'>*)

Filter `DataFrame` by at least n valid values in at least one group.

Taking a Pandas `DataFrame` with a `MultiIndex` column index, filters rows to remove rows where there are less than *n* valid values per group. Groups are defined by the *levels* parameter indexing into the column index. For example, a `MultiIndex` with top and second level Group (A,B,C) and Replicate (1,2,3) using `levels=[0,1]` would filter on *n* valid values per replicate. Alternatively, `levels=[0]` would filter on *n*

valid values at the Group level only, e.g. A, B or C.

By default valid values are determined by *np.nan*. However, alternatives can be supplied via *invalid*.

Parameters

- **`df`** – Pandas `DataFrame`
- **`levels`** – `list` of `int` specifying levels of column `MultiIndex` to group by
- **`n`** – `int` minimum number of valid values threshold
- **`invalid`** – matching invalid value

Returns filtered Pandas `DataFrame`

`padua.filters.`**`remove_columns_containing`**(*df*, *column*, *match*)

Return a `DataFrame` with rows where *column* values containing *match* are removed.

The selected *column* series of values from the supplied Pandas `DataFrame` is compared to *match*, and those rows that contain it are removed from the DataFrame.

Parameters

- **`df`** – Pandas `DataFrame`
- **`column`** – Column indexer
- **`match`** – `str` match target

Returns Pandas `DataFrame` filtered

`padua.filters.`**`remove_columns_matching`**(*df*, *column*, *match*)

Return a `DataFrame` with rows where *column* values match *match* are removed.

The selected *column* series of values from the supplied Pandas `DataFrame` is compared to *match*, and those rows that match are removed from the DataFrame.

Parameters

- **`df`** – Pandas `DataFrame`
- **`column`** – Column indexer
- **`match`** – `str` match target

Returns Pandas `DataFrame` filtered

`padua.filters.`**`remove_contaminants`**(*df*)

Remove rows with a + in the 'Contaminants' column

Return a `DataFrame` where rows where there is a "+" in the column 'Contaminants' are removed. Filters data to remove peptides matched as reverse.

Parameters **`df`** – Pandas `DataFrame`

Returns filtered Pandas `DataFrame`

`padua.filters.`**`remove_only_identified_by_site`**(*df*)

Remove rows with a + in the 'Only identified by site' column

Return a `DataFrame` where rows where there is a "+" in the column 'Only identified by site' are removed. Filters data to remove peptides matched as reverse.

Parameters **`df`** – Pandas `DataFrame`

Returns filtered Pandas `DataFrame`

`padua.filters.`**`remove_potential_contaminants`**(*df*)

Remove rows with a + in the 'Potential contaminant' column

Return a `DataFrame` where rows where there is a "+" in the column 'Contaminants' are removed. Filters data to remove peptides matched as reverse.

Parameters **`df`** – Pandas `DataFrame`

Returns filtered Pandas `DataFrame`

`padua.filters.`**`remove_reverse`**(*df*)

Remove rows with a + in the 'Reverse' column.

Return a `DataFrame` where rows where there is a "+" in the column 'Reverse' are removed. Filters data to remove peptides matched as reverse.

Parameters **`df`** – Pandas `DataFrame`

Returns filtered Pandas `DataFrame`

`padua.filters.`**`search`**(*df, match, columns=['Proteins', 'Protein names', 'Gene names']*)

Search for a given string in a set of columns in a processed `DataFrame`.

Returns a filtered `DataFrame` where *match* is contained in one of the *columns*.

Parameters

- **`df`** – Pandas `DataFrame`
- **`match`** – `str` to search for in columns
- **`columns`** – `list` of `str` to search for match

Returns filtered Pandas `DataFrame`

Imputation

Algorithms for imputing missing values in data

`padua.imputation.`**`gaussian`**(*df*, *width=0.3*, *downshift=-1.8*, *prefix=None*)
Impute missing values by drawing from a normal distribution

Parameters

- **`df`** –
- **`width`** – Scale factor for the imputed distribution relative to the standard deviation of measured values. Can be a single number or list of one per column.
- **`downshift`** – Shift the imputed values down, in units of std. dev. Can be a single number or list of one per column
- **`prefix`** – The column prefix for imputed columns

Returns

`padua.imputation.`**`pls`**(*df*)
A simple implementation of a least-squares approach to imputation using partial least squares regression (PLS).

Parameters **`df`** –

Returns

Input/Output (IO)

`padua.io.`**`read_maxquant`**(*f*, *header=0*, *index_col='id'*, ***kwargs*)
Load the quantified table output from MaxQuant run, e.g.

•Proteingroups.txt

•Phospho (STY)Sites.txt

Parameters **`f`** – Source file

Returns Pandas dataframe of imported data

`padua.io.`**`read_perseus`**(*f*)
Load a Perseus processed data table

Parameters **`f`** – Source file

Returns Pandas dataframe of imported data

`padua.io.`**`write_perseus`**(*f*, *df*)
Export a dataframe to Perseus; recreating the format

Parameters

- **`f`** –
- **`df`** –

Returns

`padua.io.`**`write_phosphopath`**(*df*, *f*, *extra_columns=None*)
Write out the data frame of phosphosites in the following format:

```
protein, protein-Rsite, Rsite, multiplicity
Q13619  Q13619-S10      S10     1
Q9H3Z4  Q9H3Z4-S10      S10     1
Q6GQQ9  Q6GQQ9-S100     S100    1
Q86YP4  Q86YP4-S100     S100    1
Q9H307  Q9H307-S100     S100    1
Q8NEY1  Q8NEY1-S1000    S1000   1
```

The file is written as a comma-separated (CSV) file to file `f`.

Parameters

- **`df`** –
- **`f`** –

Returns

`padua.io.`**`write_phosphopath_ratio`**(*df*, *f*, *v*, *a=None*, *b=None*)

Write out the data frame ratio between two groups protein-Rsite-multiplicity-timepoint ID Ratio Q13619-S10-1-1 0.5 Q9H3Z4-S10-1-1 0.502 Q6GQQ9-S100-1-1 0.504 Q86YP4-S100-1-1 0.506 Q9H307-S100-1-1 0.508 Q8NEY1-S1000-1-1 0.51 Q13541-S101-1-1 0.512 O95785-S1012-2-1 0.514 O95785-S1017-2-1 0.516 Q9Y4G8-S1022-1-1 0.518 P35658-S1023-1-1 0.52

Parameters

- **`df`** –
- **`f`** –
- **`v`** – Value ratio
- **`t`** – Timepoint
- **`a`** –
- **`b`** –

Returns

`padua.io.`**`write_r`**(*df*, *f*, *sep=', '*, *index_join='@'*, *columns_join='.'*)

Export dataframe in a format easily importable to R

Index fields are joined with "@" and column fields by "." by default. :param df: :param f: :param index_join: :param columns_join: :return:

Normalization

`padua.normalization.`**`subtract_column_median`**(*df*, *prefix='Intensity '*)

Apply column-wise normalisation to expression columns.

Default is median transform to expression columns beginning with Intensity

Parameters

- **`df`** –
- **`prefix`** – The column prefix for expression columns

Returns

Process

`padua.process.`**`apply_experimental_design`**(*df*, *f*, *prefix='Intensity '*)

Load the experimental design template from MaxQuant and use it to apply the label names to the data columns.

Parameters

- **`df`** –
- **`f`** – File path for the experimental design template
- **`prefix`** –

Returns dt

`padua.process.`**`build_index_from_design`**(*df*, *design*, *remove=None*, *types=None*, *axis=1*, *auto_convert_numeric=True*, *unmatched_columns='index'*)

Build a MultiIndex from a design table.

Supply with a table with column headings for the new multiindex and a index containing the labels to search for in the data.

Parameters

- **`df`** –
- **`design`** –
- **`remove`** –
- **`types`** –
- **`axis`** –
- **`auto_convert_numeric`** –

Returns

`padua.process.`**`build_index_from_labels`**(*df*, *indices*, *remove=None*, *types=None*, *axis=1*)

Build a MultiIndex from a list of labels and matching regex

Supply with a dictionary of Hierarchy levels and matching regex to extract this level from the sample label

Parameters

- **`df`** –
- **`indices`** – Tuples of indices ('label','regex') matches
- **`strip`** – Strip these strings from labels before matching (e.g. headers)
- **`axis=1`** – Axis (1 = columns, 0 = rows)

Returns

`padua.process.`**`combine_expression_columns`**(*df*, *columns_to_combine*, *remove_combined=True*)

Combine expression columns, calculating the mean for 2 columns

Parameters

- **`df`** – Pandas dataframe
- **`columns_to_combine`** – A list of tuples containing the column names to combine

Returns

`padua.process.`**`expand_side_table`**(*df*)

Perform equivalent of 'expand side table' in Perseus by folding Multiplicity columns down onto duplicate rows

The id is remapped to UID___Multiplicity, which is different to Perseus behaviour, but prevents accidental of non-matching rows from occurring later in analysis.

Parameters **`df`** –

Returns

`padua.process.`**`fold_columns_to_rows`**(*df*, *levels_from=2*)

Take a levels from the columns and fold down into the row index. This destroys the existing index; existing rows will appear as columns under the new column index

Parameters

- **`df`** –
- **`levels_from`** – The level (inclusive) from which column index will be folded

Returns

`padua.process.`**`get_unique_indices`**(*df*, *axis=1*)

Parameters

- **`df`** –
- **`axis`** –

Returns

`padua.process.`**`numeric`**(*s*)

Parameters **`s`** –

Returns

`padua.process.`**`strip_index_labels`**(*df*, *strip*, *axis=1*)

Parameters

- **`df`** –
- **`strip`** –
- **`axis`** –

Returns

`padua.process.`**`transform_expression_columns`**(*df*, *fn=<Mock id='140225958496800'>*, *prefix='Intensity '*)

Apply transformation to expression columns.

Default is log2 transform to expression columns beginning with Intensity

Parameters

- **`df`** –
- **`prefix`** – The column prefix for expression columns

Returns

Utils

`padua.utils.`**`build_combined_label`**(*sl*, *idxs*, *sep=' '*, *label_format=None*)

Generate a combined label from a list of indexes into sl, by joining them with *sep* (str).

Parameters

- **`sl`** (*`dict of str`*) – Strings to combine
- **`idxs`** (*`list of sl keys`*) – Indexes into sl
- **`sep`** –

Returns *str* of combined label

`padua.utils.`**`calculate_s0_curve`**(*s0*, *minpval*, *maxpval*, *minratio*, *maxratio*, *curve_interval=0.1*)

Calculate s0 curve for volcano plot.

Taking an min and max p value, and a min and max ratio, calculate an smooth curve starting from parameter *s0* in each direction.

The *curve_interval* parameter defines the smoothness of the resulting curve.

Parameters

- **`s0`** – *float* offset of curve from interset
- **`minpval`** – *float* minimum p value
- **`maxpval`** – *float* maximum p value
- **`minratio`** – *float* minimum ratio
- **`maxratio`** – *float* maximum ratio
- **`curve_interval`** – *float* stepsize (smoothness) of curve generator

Returns x, y, fn x,y points of curve, and fn generator

`padua.utils.`**`chunks`**(*seq*, *num*)

Separate *seq* (*np.array*) into *num* series of as-near-as possible equal length values.

Parameters

- **`seq`** (*`np.array`*) – Sequence to split
- **`num`** (*`int`*) – Number of parts to split sequence into

Returns np.array of split parts

`padua.utils.`**`find_nearest_idx`**(*array*, *value*)

Parameters

- **`array`** –
- **`value`** –

Returns

`padua.utils.`**`get_index_list`**(*l*, *ms*)

Parameters

- **`l`** –
- **`ms`** –

Returns

padua.utils.**get_protein_id**(*s*)

Return a shortened string, split on spaces, underlines and semicolons.

Extract the first, highest-ranked protein ID from a string containing protein IDs in MaxQuant output format: e.g. P07830;P63267;Q54A44;P63268

Long names (containing species information) are eliminated (split on ' ') and isoforms are removed (split on '_').

Parameters **s** (*str or unicode*) – protein IDs in MaxQuant format

Returns string

padua.utils.**get_protein_id_list**(*df*, *level=0*)

Return a complete list of shortform IDs from a DataFrame

Extract all protein IDs from a dataframe from multiple rows containing protein IDs in MaxQuant output format: e.g. P07830;P63267;Q54A44;P63268

Long names (containing species information) are eliminated (split on ' ') and isoforms are removed (split on '_').

Parameters

- **df** (*pandas.DataFrame*) – DataFrame
- **level** (*int or str*) – Level of DataFrame index to extract IDs from

Returns list of string ids

padua.utils.**get_protein_ids**(*s*)

Return a list of shortform protein IDs.

Extract all protein IDs from a string containing protein IDs in MaxQuant output format: e.g. P07830;P63267;Q54A44;P63268

Long names (containing species information) are eliminated (split on ' ') and isoforms are removed (split on '_').

Parameters **s** (*str or unicode*) – protein IDs in MaxQuant format

Returns list of string ids

padua.utils.**get_shortstr**(*s*)

Return the first part of a string before a semicolon.

Extract the first, highest-ranked protein ID from a string containing protein IDs in MaxQuant output format: e.g. P07830;P63267;Q54A44;P63268

Parameters **s** (*str or unicode*) – protein IDs in MaxQuant format

Returns string

padua.utils.**hierarchical_match**(*d*, *k*, *default=None*)

Match a key against a dict, simplifying element at a time

Parameters

- **df** (*pandas.DataFrame*) – DataFrame
- **level** (*int or str*) – Level of DataFrame index to extract IDs from

Returns hiearchically matched value or default

padua.utils.**qvalues**(*pv*, *m=None*, *verbose=False*, *lowmem=False*, *pi0=None*)

Copyright (c) 2012, Nicolo Fusi, University of Sheffield All rights reserved.

Estimates q-values from p-values

m: number of tests. If not specified m = pv.size verbose: print verbose messages? (default False) lowmem: use memory-efficient in-place algorithm pi0: if None, it's estimated as suggested in Storey and Tibshirani, 2003.

For most GWAS this is not necessary, since pi0 is extremely likely to be 1

Parameters

- **`pv`** –
- **`m`** –
- **`verbose`** –
- **`lowmem`** –
- **`pi0`** –

Returns

Visualize

Visualization tools for proteomic data, using standard Pandas dataframe structures from imported data. These functions make some assumptions about the structure of data, but generally try to accomodate.

Depends on scikit-learn for PCA analysis

`padua.visualize.`**`box`**(*df*, *s=None*, *title_from=None*, *subplots=False*, *figsize=(18, 6)*, *groups=None*, *fcol=None*, *ecol=None*, *hatch=None*, *ylabel=''*, *xlabel=''*)

Generate a box plot from pandas DataFrame with sample grouping.

Plot group mean, median and deviations for specific values (proteins) in the dataset. Plotting is controlled via the *s* param, which is used as a search string along the y-axis. All matching values will be returned and plotted. Multiple search values can be provided as a *list* of *str* and these will be searched as an *and* query.

Box fill and edge colors can be controlled on a full-index basis by passing a *dict* of indexer:color to *fcol* and *ecol* respectively. Box hatching can be controlled by passing a *dict* of indexer:hatch to *hatch*.

Parameters

- **`df`** – Pandas *DataFrame*
- **`s`** – *str* search y-axis for matching values (case-insensitive)
- **`title_from`** – *list* of *str* of index levels to generate title from
- **`subplots`** – *bool* use subplots to separate plot groups
- **`figsize`** – *tuple* of *int* size of resulting figure
- **`groups`** –
- **`fcol`** – *dict* of *str* indexer:color where color is hex value or matplotlib color code
- **`ecol`** – *dict* of *str* indexer:color where color is hex value or matplotlib color code
- **`hatch`** – *dict* of *str* indexer:hatch where hatch is matplotlib hatch descriptor
- **`ylabel`** – *str* ylabel for boxplot
- **`xlabel`** – *str* xlabel for boxplot

Returns *list* of *Figure*

`padua.visualize.`**`column_correlations`**(*df*, *cmap=<Mock id='140225958691560'>*)

Parameters

- **df** –
- **cmap** –

Returns

`padua.visualize.comparedist`(*df1*, *df2*, *bins=50*)

Compare the distributions of two DataFrames giving visualisations of:

- individual and combined distributions
- distribution of non-common values
- distribution of non-common values vs. each side

Plot distribution as area (fill_between) + mean, median vertical bars.

Parameters

- **df1** – *pandas.DataFrame*
- **df2** – *pandas.DataFrame*
- **bins** – *int* number of bins for histogram

Returns Figure

`padua.visualize.correlation`(*df*, *cm=<Mock id='140225958014424'>*, *vmin=None*, *vmax=None*, *labels=None*, *show_scatter=False*)

Generate a column-wise correlation plot from the provided data.

The columns of the supplied dataframes will be correlated (using *analysis.correlation*) to generate a Pearson correlation plot heatmap. Scatter plots of correlated samples can also be generated over the redundant half of the plot to give a visual indication of the protein distribution.

Parameters

- **df** – *pandas.DataFrame*
- **cm** – Matplotlib colormap (default cm.PuOr_r)
- **vmin** – Minimum value for colormap normalization
- **vmax** – Maximum value for colormap normalization
- **labels** – Index column to retrieve labels from
- **show_scatter** – Show overlaid scatter plots for each sample in lower-left half. Note that this is slow for large numbers of samples.

Returns *matplotlib.Figure* generated Figure.

`padua.visualize.enrichment`(*dfenr*, *include=None*)

Generates an enrichment pie chart series from a calculate enrichment table :param df: :return:

`padua.visualize.hierarchical`(*df*, *cluster_cols=True*, *cluster_rows=False*, *n_col_clusters=False*, *n_row_clusters=False*, *row_labels=True*, *col_labels=True*, *fcol=None*, *z_score=0*, *method='ward'*, *cmap=<Mock id='140225958198520'>*, *return_clusters=False*, *rdistance_fn=<Mock id='140225958420264'>*, *cdistance_fn=<Mock id='140225958006400'>*)

Hierarchical clustering of samples or proteins

Peform a hiearchical clustering on a pandas DataFrame and display the resulting clustering as a heatmap. The axis of clustering can be controlled with *cluster_cols* and *cluster_rows*. By default clustering is performed along the X-axis, therefore to cluster samples transpose the DataFrame as it is passed, using *df.T*.

Samples are z-scored along the 0-axis (y) by default. To override this use the *z_score* param with the axis to *z_score* or alternatively, *None*, to turn it off.

If a *n_col_clusters* or *n_row_clusters* is specified, this defines the number of clusters to identify and highlight in the resulting heatmap. At *least* this number of clusters will be selected, in some instances there will be more if 2 clusters rank equally at the determined cutoff.

If specified *fcol* will be used to colour the axes for matching samples.

Parameters

- **`df`** – Pandas `DataFrame` to cluster
- **`cluster_cols`** – `bool` if `True` cluster along column axis
- **`cluster_rows`** – `bool` if `True` cluster along row axis
- **`n_col_clusters`** – `int` the ideal number of highlighted clusters in cols
- **`n_row_clusters`** – `int` the ideal number of highlighted clusters in rows
- **`fcol`** – `dict` of label:colors to be applied along the axes
- **`z_score`** – `int` to specify the axis to Z score or *None* to disable
- **`method`** – `str` describing cluster method, default ward
- **`cmap`** – matplotlib colourmap for heatmap
- **`return_clusters`** – `bool` return clusters in addition to axis

Returns matplotlib axis, or axis and cluster data

`padua.visualize.`**`kegg_pathway`**(*df*, *pathway*, *a*, *b=None*, *ids_from='Proteins'*, *cmap=<Mock id='140225958022504'>*, *is_log2=False*, *fillna=None*, *z_score=1*)

Visualize data on a kegg pathway.

Parameters

- **`df`** –
- **`pathway`** –
- **`a`** –
- **`b`** –
- **`ids_from`** –
- **`cmap`** –
- **`is_log2`** –
- **`fillna`** –
- **`z_score`** –

Returns

`padua.visualize.`**`modificationlocalization`**(*df*)

Plot the % of Class I, II and III localised peptides according to standard thresholds.

Generates a pie chart showing the % of peptides that fall within the Class I, II and III classifications based on localisation probability. These definitions are:

```
Class I     0.75 > x
Class II    0.50 > x <= 0.75
Class III   0.25 > x <= 0.50
```

Any peptides with a localisation score of <= 0.25 are excluded.

Parameters **df** –

Returns matplotlib axis

padua.visualize.**modifiedaminoacids**(*df*, *kind='pie'*)

Generate a plot of relative numbers of modified amino acids in source DataFrame.

Plot a pie or bar chart showing the number and percentage of modified amino acids in the supplied data frame. The amino acids displayed will be determined from the supplied data/modification type.

Parameters

- **df** – processed DataFrame
- **kind** – *str* type of plot; either "pie" or "bar"

Returns matplotlib ax

padua.visualize.**pca**(*df*, *n_components=2*, *mean_center=False*, *fcol=None*, *ecol=None*, *marker='o'*, *markersize=40*, *threshold=None*, *label_threshold=None*, *label_weights=None*, *label_scores=None*, *return_df=False*, *show_covariance_ellipse=False*, **args*, ***kwargs*)

Perform Principal Component Analysis (PCA) from input DataFrame and generate scores and weights plots.

Principal Component Analysis is a technique for identifying the largest source of variation in a dataset. This function uses the implementation available in scikit-learn. The PCA is calculated via *analysis.pca* and will therefore give identical results.

Resulting scores and weights plots are generated showing the distribution of samples within the resulting PCA space. Sample color and marker size can be controlled by label, lookup and calculation (lambda) to generate complex plots highlighting sample separation.

For further information see the examples included in the documentation.

Parameters

- **df** – Pandas *DataFrame*
- **n_components** – *int* number of Principal components to return
- **mean_center** – *bool* mean center the data before performing PCA
- **fcol** – *dict* of indexers:colors, where colors are hex colors or matplotlib color names
- **ecol** – *dict* of indexers:colors, where colors are hex colors or matplotlib color names
- **marker** – *str* matplotlib marker name (default "o")
- **markersize** – *int* or *callable* which returns an *int* for a given indexer
- **threshold** – *float* weight threshold for plot (horizontal line)
- **label_threshold** – *float* weight threshold over which to draw labels
- **label_weights** – *list* of *str*
- **label_scores** – *list* of *str*
- **return_df** – *bool* return the resulting scores, weights as pandas DataFrames
- **show_covariance_ellipse** – *bool* show the covariance ellipse around each group

- **`args`** – additional arguments passed to analysis.pca
- **`kwargs`** – additional arguments passed to analysis.pca

Returns

`padua.visualize.`**`plot_cov_ellipse`**(*cov*, *pos*, *nstd=2*, ***kwargs*)

Plots an *nstd* sigma error ellipse based on the specified covariance matrix (*cov*). Additional keyword arguments are passed on to the ellipse patch artist.

cov : The 2x2 covariance matrix to base the ellipse on pos : The location of the center of the ellipse. Expects a 2-element

sequence of [x0, y0].

nstd [The radius of the ellipse in numbers of standard deviations.] Defaults to 2 standard deviations.

Additional keyword arguments are pass on to the ellipse patch.

A matplotlib ellipse artist

`padua.visualize.`**`plot_point_cov`**(*points*, *nstd=2*, ***kwargs*)

Plots an *nstd* sigma ellipse based on the mean and covariance of a point "cloud" (points, an Nx2 array).

points : An Nx2 array of the data points. nstd : The radius of the ellipse in numbers of standard deviations.

Defaults to 2 standard deviations.

Additional keyword arguments are pass on to the ellipse patch.

A matplotlib ellipse artist

`padua.visualize.`**`plsda`**(*df*, *a*, *b*, *n_components=2*, *mean_center=False*, *scale=True*, *fcol=None*, *ecol=None*, *marker='o'*, *markersize=40*, *threshold=None*, *label_threshold=None*, *label_weights=None*, *label_scores=None*, *return_df=False*, *show_covariance_ellipse=False*, **args*, ***kwargs*)

Partial Least Squares Regression Analysis, based on *sklearn.cross_decomposition.PLSRegression*

Performs a partial least squares regression (PLS-R) on the supplied dataframe `df` against the provided continuous variable `v`, selecting the first `n_components`.

For more information on PLS regression and the algorithm used, see the scikit-learn documentation.

Resulting scores and weights plots are generated showing the distribution of samples within the resulting PCA space. Sample color and marker size can be controlled by label, lookup and calculation (lambda) to generate complex plots highlighting sample separation.

For further information see the examples included in the documentation.

Parameters

- **`df`** – Pandas *DataFrame*
- **`a`** – Column selector for group a
- **`b`** – Column selector for group b
- **`n_components`** – *int* number of Principal components to return
- **`mean_center`** – *bool* mean center the data before performing PCA
- **`fcol`** – *dict* of indexers:colors, where colors are hex colors or matplotlib color names

- **ecol** – *dict* of indexers:colors, where colors are hex colors or matplotlib color names
- **marker** – *str* matplotlib marker name (default "o")
- **markersize** – *int* or *callable* which returns an *int* for a given indexer
- **threshold** – *float* weight threshold for plot (horizontal line)
- **label_threshold** – *float* weight threshold over which to draw labels
- **label_weights** – *list* of *str*
- **label_scores** – *list* of *str*
- **return_df** – *bool* return the resulting scores, weights as pandas DataFrames
- **show_covariance_ellipse** – *bool* show the covariance ellipse around each group
- **args** – additional arguments passed to analysis.pca
- **kwargs** – additional arguments passed to analysis.pca

Returns

`padua.visualize.`**`plsr`**(*df*, *v*, *n_components=2*, *mean_center=False*, *scale=True*, *fcol=None*, *ecol=None*, *marker='o'*, *markersize=40*, *threshold=None*, *label_threshold=None*, *label_weights=None*, *label_scores=None*, *return_df=False*, *show_covariance_ellipse=False*, **args*, ***kwargs*)

Partial Least Squares Regression Analysis, based on *sklearn.cross_decomposition.PLSRegression*

Performs a partial least squares regression (PLS-R) on the supplied dataframe `df` against the provided continuous variable `v`, selecting the first `n_components`.

For more information on PLS regression and the algorithm used, see the scikit-learn documentation.

Resulting scores, weights and regression plots are generated showing the distribution of samples within the resulting PCA space. Sample color and marker size can be controlled by label, lookup and calculation (lambda) to generate complex plots highlighting sample separation.

For further information see the examples included in the documentation.

Parameters

- **df** – Pandas *DataFrame*
- **v** – Continuous variable to perform regression against
- **n_components** – *int* number of Principal components to return
- **mean_center** – *bool* mean center the data before performing PCA
- **fcol** – *dict* of indexers:colors, where colors are hex colors or matplotlib color names
- **ecol** – *dict* of indexers:colors, where colors are hex colors or matplotlib color names
- **marker** – *str* matplotlib marker name (default "o")
- **markersize** – *int* or *callable* which returns an *int* for a given indexer
- **threshold** – *float* weight threshold for plot (horizontal line)
- **label_threshold** – *float* weight threshold over which to draw labels
- **label_weights** – *list* of *str*
- **label_scores** – *list* of *str*
- **return_df** – *bool* return the resulting scores, weights as pandas DataFrames
- **show_covariance_ellipse** – *bool* show the covariance ellipse around each group

- **args** – additional arguments passed to analysis.pca
- **kwargs** – additional arguments passed to analysis.pca

Returns

`padua.visualize.`**`quality_control`**(*df*)

`padua.visualize.`**`rankintensity`**(*df*, *colors=None*, *labels_from='Protein names'*, *number_of_annotations=3*, *show_go_enrichment=False*, *go_ids_from=None*, *go_enrichment='function'*, *go_max_labels=8*, *go_fdr=None*, *progress_callback=None*)

Rank intensity plot, showing intensity order vs. raw intensity value S curve.

Generates a plot showing detected protein intensity plotted against protein intensity rank. A series of colors can be provided to segment the S curve into regions. Gene ontology enrichments (as calculated via *analysis.go_enrichment*) can be overlaid on the output. Note that since the ranking reflects simple abundance there is little meaning to enrichment (FDR will remove most if not all items) and it is best considered an annotation of the 'types' of proteins in that region.

Parameters

- **df** – Pands DataFrame
- **colors** – *list* of colors to segment the plot into
- **labels_from** – Take labels from this column
- **number_of_annotations** – Number of protein annotations at each tip
- **show_go_enrichment** – Overlay plot with GO enrichment terms
- **go_ids_from** – Get IDs for GO enrichment from this column
- **go_enrichment** – Type of GO enrichment to show
- **go_max_labels** – Maximum number of GO enrichment labels per segment
- **go_fdr** – FDR cutoff to apply to the GO enrichment terms

Returns matplotlib Axes

`padua.visualize.`**`sitespeptidesproteins`**(*df*, *labels=None*, *colors=None*, *site_localization_probability=0.75*)

Plot the number of sites, peptides and proteins in the dataset.

Generates a plot with sites, peptides and proteins displayed hierarchically in chevrons. The site count is limited to Class I (<=0.75 site localization probability) by default but may be altered using the *site_localization_probability* parameter.

Labels and alternate colours may be supplied as a 3-entry iterable.

Parameters

- **df** – pandas DataFrame to calculate numbers from
- **labels** – list/tuple of 3 strings containing labels
- **colors** – list/tuple of 3 colours as hex codes or matplotlib color codes
- **site_localization_probability** – the cut-off for site inclusion (default=0.75; Class I)

Returns

`padua.visualize.`**`venn`**`(`*df1*, *df2*, *df3=None*, *labels=None*, *ix1=None*, *ix2=None*, *ix3=None*, *return_intersection=False*, *fcols=None*`)`

Plot a 2 or 3-part venn diagram showing the overlap between 2 or 3 pandas DataFrames.

Provided with two or three Pandas DataFrames, this will return a venn diagram showing the overlap calculated between the DataFrame indexes provided as ix1, ix2, ix3. Labels for each DataFrame can be provided as a list in the same order, while *fcol* can be used to specify the colors of each section.

Parameters

- **`df1`** – Pandas DataFrame
- **`df2`** – Pandas DataFrame
- **`df3`** – Pandas DataFrame (optional)
- **`labels`** – List of labels for the provided dataframes
- **`ix1`** – Index level name of of Dataframe 1 to use for comparison
- **`ix2`** – Index level name of of Dataframe 2 to use for comparison
- **`ix3`** – Index level name of of Dataframe 3 to use for comparison
- **`return_intersection`** – Return the intersection of the supplied indices
- **`fcols`** – List of colors for the provided dataframes

Returns ax, or ax with intersection

`padua.visualize.`**`volcano`**`(`*df*, *a*, *b=None*, *fdr=0.05*, *figsize=(8, 10)*, *show_numbers=True*, *threshold=2*, *minimum_sample_n=0*, *estimate_qvalues=False*, *labels_from=None*, *labels_for=None*, *title=None*, *label_format=None*, *markersize=64*, *s0=1e-05*, *draw_fdr=True*, *is_log2=False*, *fillna=None*, *label_sig_only=True*, *ax=None*, *xlim=None*, *ylim=None*, *fc='grey'*, *fc_sig='blue'*, *fc_sigr='red'*`)`

Volcano plot of two sample groups showing t-test p value vs. log2(fc).

Generates a volcano plot for two sample groups, selected from *df* using *a* and *b* indexers. The mean of each group is calculated along the y-axis (per protein) and used to generate a log2 ratio. If a log2-transformed dataset is supplied set *islog2=True* (a warning will be given when negative values are present).

A two-sample independent t-test is performed between each group. If *minimum_sample_n* is supplied, any values (proteins) without this number of samples will be dropped from the analysis.

Individual data points can be labelled in the resulting plot by passing *labels_from* with a index name, and *labels_for* with a list of matching values for which to plot labels.

Parameters

- **`df`** – Pandas *dataframe*
- **`a`** – *tuple* or *str* indexer for group A
- **`b`** – *tuple* or *str* indexer for group B
- **`fdr`** – *float* false discovery rate cut-off
- **`threshold`** – *float* log2(fc) ratio cut -off
- **`minimum_sample_n`** – *int* minimum sample for t-test
- **`estimate_qvalues`** – *bool* estimate Q values (adjusted P)
- **`labels_from`** – *str* or *int* index level to get labels from
- **`labels_for`** – *list* of *str* matching labels to show

- **`title`** – *str* title for plot
- **`markersize`** – *int* size of markers
- **`s0`** – *float* smoothing factor between fdr/fc cutoff
- **`draw_fdr`** – *bool* draw the fdr/fc curve
- **`is_log2`** – *bool* is the data log2 transformed already?
- **`fillna`** – *float* fill NaN values with value (default: 0)
- **`label_sig_only`** – *bool* only label significant values
- **`ax`** – matplotlib *axis* on which to draw
- **`fc`** – *str* hex or matplotlib color code, default color of points

Returns

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index