Python Documentation

version

October 20, 2020

Contents

Welcome to moseq2-viz's documentation!

moseq2_viz package

CLI Module

moseq2-viz

```
moseq2-viz [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit. [default: False]

add-group

Change group name in index file given a key-value pair

```
moseq2-viz add-group [OPTIONS] INDEX_FILE
```

Options

-k, --key <key>
Key to search for value [default: SubjectName]

-v, --value <value>
Value to search for [default: Mouse]

-g, --group <group>
Group name to map to [default: Group1]

-e, --exact
Exact match only [default: False]

--lowercase
Lowercase text filter [default: False]

-n, --negative
Negative match (everything that does not match is included) [default: False]

Arguments

INDEX_FILE
Required argument

copy-h5-metadata-to-yaml

Copies metadata within an h5 file to a yaml file.

```
moseq2-viz copy-h5-metadata-to-yaml [OPTIONS]
```

Options

-i, --input-dir <input_dir>
Directory to find h5 files [default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs]

--h5-metadata-path <h5_metadata_path>
Path to acquisition metadata in h5 files [default: /metadata/acquisition]

get-best-model

Returns the model with the closest median duration to the PC Changepoints, given a directory containing multiple models

```
moseq2-viz get-best-model [OPTIONS] MODEL_DIR CP_PATH OUTPUT_FILE
```

Options

--plot-all
Plot all included model results [default: False]

--ext <ext>
Model extensions found in input directory [default: .p]

--fps <fps>
Frames per second [default: 30]

Arguments

MODEL_DIR
Required argument

CP_PATH
Required argument

OUTPUT_FILE
Required argument

make-crowd-movies

Writes movies of overlaid examples of the rodent perform a given syllable

```
moseq2-viz make-crowd-movies [OPTIONS] INDEX_FILE MODEL_PATH
```

Options

--max-syllable <max_syllable>
Index of max syllable to render [default: 40]

-m, --max-examples <max_examples>
Number of examples to show [default: 40]

-t, --threads <threads>
Number of threads to use for rendering crowd movies [default: -1]

--separate-by <separate_by>
Generate crowd movies from individual group sources. [default: default]

Options: default|groups|sessions

--specific-syllable <specific_syllable>
Index of max syllable to render

-s, --session-names <session_names>
SessionNames to create crowd movies from [default:]

--sort <sort>
Sort syllables by usage [default: True]

--count <count>
How to quantify syllable usage [default: usage]

Options: usage|frames

-o, --output-dir <output_dir>
Path to store files [default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs/crowd_movies]

--gaussfilter-space <gaussfilter_space>
Spatial filter for data (Gaussian) [default: 0, 0]

--medfilter-space <medfilter_space>
Median spatial filter [default: 0]

--min-height <min_height>
Minimum height for scaling videos [default: 5]

--max-height <max_height>
Minimum height for scaling videos [default: 80]

--raw-size <raw_size>
Size of original videos [default: 512, 424]

--scale <scale>
Scaling from pixel units to mm [default: 1]

--cmap <cmap>
Name of valid Matplotlib colormap for false-coloring images [default: jet]

--dur-clip <dur_clip>
Exclude syllables more than this number of frames (None for no limit) [default: 300]

--legacy-jitter-fix <legacy_jitter_fix>
Set to true if you notice jitter in your crowd movies [default: False]

--frame-path <frame_path>
Path to depth frames in h5 file [default: frames]

-p, --progress-bar
Show verbose progress bars. [default: False]

Arguments

INDEX_FILE
Required argument

MODEL_PATH
Required argument

plot-group-position-heatmaps

Plots position heatmaps for each group in the index file

```
moseq2-viz plot-group-position-heatmaps [OPTIONS] INDEX_FILE
```

Options

--output-file <output_file>
[default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs/scalars]

Arguments

INDEX_FILE
Required argument

plot-scalar-summary

Plots a scalar summary of the index file data.

```
moseq2-viz plot-scalar-summary [OPTIONS] INDEX_FILE
```

Options

--output-file <output_file>
[default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs/scalars]

-c, --colors <colors>
Colors to plot groups with.

Arguments

INDEX_FILE
Required argument

plot-stats

Plots syllable usages with different sorting,coloring and grouping capabilities

```
moseq2-viz plot-stats [OPTIONS] INDEX_FILE MODEL_FIT
```

Options

--stat <stat>
Statistic to plot. [default: usage]

Options: usage|speed|duration

--output-file <output_file>
Filename to store plot [default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs/syll_stat]

--sort <sort>
Sort syllables by usage [default: True]

--figsize <figsize>
Size dimensions of the plotted figure. [default: 10, 5]

--count <count>
How to quantify syllable usage [default: usage]

Options: usage|frames

--max-syllable <max_syllable>
Index of max syllable to render [default: 40]

-g, --group <group>
Name of group(s) to show

-o, --ordering <ordering>
How to order the groups [default: usage]

Options: usage|speed|duration|mutation

--ctrl-group <ctrl_group>
Name of control group. Only if ordering = 'm'

--exp-group <exp_group>
Name of experimental group. Only if ordering = 'm'

-c, --colors <colors>
Colors to plot groups with.

-f, --fmt <fmt>
Format the scatter plot data. [default: o-]

Arguments

INDEX_FILE
Required argument

MODEL_FIT
Required argument

plot-transition-graph

Plots the transition graph depicting the transition probabilities between syllables.

```
moseq2-viz plot-transition-graph [OPTIONS] INDEX_FILE MODEL_FIT
```

Options

--max-syllable <max_syllable>
Index of max syllable to render [default: 40]

-g, --group <group>
Name of group(s) to show

--output-file <output_file>
Filename to store plot [default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs/transitions]

--normalize <normalize>
How to normalize transition probabilities [default: bigram]

Options: bigram|rows|columns

--edge-threshold <edge_threshold>
Threshold for edges to show [default: 0.001]

--usage-threshold <usage_threshold>
Threshold for nodes to show [default: 0]

--layout <layout>
Default networkx layout algorithm [default: spring]

-k, --keep-orphans
Show orphaned nodes [default: False]

--orphan-weight <orphan_weight>
Weight for non-existent connections [default: 0]

--arrows
Show arrows [default: False]

--sort <sort>
Sort syllables by usage [default: True]

--count <count>
How to quantify syllable usage [default: usage]

Options: usage|frames

--edge-scaling <edge_scaling>
Scale factor from transition probabilities to edge width [default: 250]

--node-scaling <node_scaling>
Scale factor for nodes by usage [default: 10000.0]

--scale-node-by-usage <scale_node_by_usage>
Scale node sizes by usages probabilities [default: True]

--width-per-group <width_per_group>
Width (in inches) for figure canvas per group [default: 8]

Arguments

INDEX_FILE
Required argument

MODEL_FIT
Required argument

plot-verbose-position-heatmaps

Plots a position heatmap for each session in the index file.

```
moseq2-viz plot-verbose-position-heatmaps [OPTIONS] INDEX_FILE
```

Options

--output-file <output_file>
[default: /Users/aymanzeine/Desktop/moseq/moseq2-viz/docs/scalars]

Arguments

INDEX_FILE
Required argument

GUI Module

GUI front-end operations. This module contains all the functionality and configurable parameters users can alter to most accurately process their data.

Note: These functions perform jupyter notebook specific pre-processing, loads in corresponding parameters from the CLI functions, then call the corresponding wrapper function with the given input parameters.

`moseq2_viz.gui.`**`add_group`** (`index_file`, `by='SessionName'`, `value='default'`, `group='default'`, `exact=False`, `lowercase=False`, `negative=False`)

Updates index file SubjectName group names with user defined group names.

Parameters:
- **index_file (str)** (*path to index file*)
- **value (str)** (*SessionName value to search for*)
- **group (str)** (*group name to allocate.*)
- **exact (bool)** (*indicate whether to search for exact match.*)
- **lowercase (bool)** (*indicate whether to convert all searched for names to lowercase.*)
- **negative (bool)** (*whether to update the inverse of the found selection.*)

Returns:

Return type: None

`moseq2_viz.gui.`**`copy_h5_metadata_to_yaml_command`** (`input_dir`, `h5_metadata_path`)

Reads h5 metadata from a specified metadata h5 path.

Parameters:
- **input_dir (str)** (*path to directory containing h5 file*)
- **h5_metadata_path (str)** (*path to metadata within h5 file*)

Returns:

Return type: None

`moseq2_viz.gui.`**`get_best_fit_model`** (`progress_paths`, `output_file=None`, `plot_all=False`, `fps=30`)

Given a directory containing multiple models, and the path to the pca scores they were trained on,

this function returns the path to the model that has the closest median syllable duration to that of the PC Scores.

Parameters:
- **progress_paths (dict)** (*Dict containing paths the to model directory and pca scores file*)
- **output_file (str)** (*Optional path to save the comparison plot*)
- **plot_all (bool)** (*Indicates whether to plot all the models' changepoint distributions with the PCs, highlighting*) – the best model curve.
- **fps (int)** (*Frames per second.*)

Returns: **best_fit_model (str)**

Return type: Path tp best fit model

`moseq2_viz.gui.`**`get_groups_command`** (`index_file`)

Jupyter Notebook to print index file current metadata groupings.

Parameters: **index_file (str)** (*path to index file*)

Returns: **(int)**

Return type: number of unique groups

`moseq2_viz.gui.`**`make_crowd_movies_command`** (`index_file`, `model_path`, `output_dir`, `max_syllable`, `max_examples`)

Runs CLI function to write crowd movies, due to multiprocessing compatibilty issues with Jupyter notebook's scheduler.

Parameters:
- **index_file (str)** (*path to index file.*)
- **model_path (str)** (*path to fit model.*)
- **output_dir (str)** (*path to directory to save crowd movies in.*)
- **max_syllable (int)** (*number of syllables to make crowd movies for.*)
- **max_examples (int)** (*max number of mice to include in a crowd movie.*)

Returns: **(str)**

Return type: Success string.

`moseq2_viz.gui.`**`plot_mean_group_position_heatmaps_command`** (`index_file`, `output_file`)

Plots the average mouse position in a PDF-derived heatmap for each group found in the inputted index file. :Parameters: * **index_file (str)** (*path to index file.*)

- **output_file (str)** (*filename for syllable duration graph.*)

Returns: **fig (pyplot figure)**

Return type: figure to graph in Jupyter Notebook.

`moseq2_viz.gui.`**`plot_scalar_summary_command`** (`index_file`, `output_file`, `colors=None`, `groupby='group'`)

Creates a scalar summary graph and a position summary graph.

Parameters:
- **index_file (str)** (*path to index file*)
- **output_file (str)** (*prefix name of scalar summary images*)
- **colors (list)** (*list of colors to serve as the sns palette in the scalar summary*)
- **groupby (str)** (*scalar_df column to group sessions by when graphing scalar and position summaries*)

Returns: **scalar_df (pandas DataFrame)**

Return type: DataFrame containing all of scalar values for debugging.

`moseq2_viz.gui.`**`plot_stats_command`** (`model_fit`, `index_file`, `output_file`, `stat='usage'`, `max_syllable=40`, `count='usage'`, `group=None`, `sort=True`, `ordering=None`, `ctrl_group=None`, `exp_group=None`, `colors=None`, `fmt='o-'`, `figsize=(10, 5)`)

Graph given syllable statistic from fit model data. :Parameters: * **model_fit (str)** (*path to fit model.*)

- **index_file (str)** (*path to index file*)
- **output_file (str)** (*name of saved usages graph.*)
- **stat (str)** (*syllable statistic to plot: ['usage', 'speed', 'duration']*)
- **max_syllable (int)** (*max number of syllables to plot.*)
- **count (str)** (*method to calculate syllable usages, either by 'frames' or 'usage'*)
- **group (tuple)** (*groups to include in usage plot. If empty, plots default average of all groups.*)
- **sort (bool)** (*sort by usages.*)
- **ordering (list, range, str, None)** (*order to list syllables. Default is None to graph syllables [0-max_syllable).*) – Setting ordering to "m" will graph mutated syllable usage difference between ctrl_group and exp_group. None to graph default [0,max_syllable] in order. "usage" to plot descending order of usage values.
- **ctrl_group (str)** (*Control group to graph when plotting mutation differences via setting ordering to 'm'.*)
- **exp_group (str)** (*Experimental group to directly compare with control group.*)
- **colors (list)** (*list of colors to serve as the sns palette in the scalar summary. If None, default colors are used.*)
- **fmt (str)** (*scatter plot format. "o-" for line plot with vertices at corresponding usages. "o" for just points.*)
- **figsize (tuple)** (*tuple value of length = 2, representing (columns x rows) of the plotted figure dimensions*)

Returns: **fig (pyplot figure)**
Return type: figure to graph in Jupyter Notebook.

`moseq2_viz.gui.`**`plot_transition_graph_command`** (`index_file`, `model_fit`, `config_file`, `max_syllable`, `group`, `output_file`)
Creates transition graphs given groups to process.

Parameters:
- **index_file (str)** (*path to index file*)
- **model_fit (str)** (*path to fit model*)
- **config_file (str)** (*path to config file*)
- **max_syllable (int)** (*maximum number of syllables to include in graph*)
- **group (tuple)** (*tuple of names of groups to graph transition graphs for*)
- **output_file (str)** (*name of the transition graph saved image*)

Returns: **fig (pyplot figure)**
Return type: figure to graph in Jupyter Notebook.

`moseq2_viz.gui.`**`plot_verbose_position_heatmaps`** (`index_file`, `output_file`)
Plots a PDF-derived heatmap of each session found in the index file titled with the session name and group.

Parameters:
- **index_file (str)** (*path to index file.*)
- **output_file (str)** (*filename for syllable duration graph.*)

Returns: **fig (pyplot figure)**
Return type: figure to graph in Jupyter Notebook.

Utilities Module

General utility functions to facilitate loading and organizing data.

`moseq2_viz.util.`**`camel_to_snake`** (`s`)
Converts CamelCase to snake_case

Parameters: **s (str)** (*string to convert to snake case*)
Returns: **(str)**
Return type: snake_case string

`moseq2_viz.util.`**`clean_dict`** (`dct`)
Casts dict values to numpy arrays

Parameters: **dct (dict)** (*dictionary with values to clean.*)
Returns: **(dict)**
Return type: dictionary with standardized value type:list

`moseq2_viz.util.`**`get_index_hits`** (`config_data`, `metadata`, `key`, `v`)
Searches for matching keys in given index file metadata dict. Returns list of booleans indicating that a session was found.

Parameters:
- **config_data (dict)** (*dictionary containing boolean search filters [lowercase, negative]*)
- **metadata (list)** (*list of session metadata dict objects*)
- **key (str)** (*metadata key being searched for*)
- **v (str)** (*value of the corresponding key to be found*)

Returns: **hits (list)**
Return type: list of booleans indicating the found sessions to be updated in add_group_wrapper()

`moseq2_viz.util.`**`get_sorted_index`** (`index_file: str`) → dict
Just return the sorted index from an index_file path.

Parameters: **index_file (str)** (*path to index file.*)
Returns: **sorted_ind (dict)**
Return type: dictionary of loaded sorted index file contents

`moseq2_viz.util.`**`get_timestamps_from_h5`** (`h5file: str`)
Returns dict of timestamps from h5file.

Parameters: **h5file (str)** (*path to h5 file.*)
Returns: **(dict)**
Return type: dictionary containing timestamp data.

`moseq2_viz.util.`**`h5_filepath_from_sorted`** (`sorted_index_entry: dict`) → str
Gets the h5 extraction file path from a sorted index entry

Parameters: **sorted_index_entry (dict)** (*get filepath from sorted index.*)
Returns: **(str)**
Return type: a str containing the extraction filepath

`moseq2_viz.util.`**`h5_to_dict`** (`h5file`, `path: str = '/'`) → dict
Load h5 dict contents to a dict variable.

Parameters:
- **h5file (str or h5py.File)** (*file path to the given h5 file or the h5 file handle*)
- **path (str)** (*path to the base dataset within the h5 file. Default: /*)

Returns: **out (dict)**
Return type: dictionary of all h5 contents

`moseq2_viz.util.`**`load_changepoints`** (`cpfile`)
Loads PC changepoints array from given changepoints.h5 file.

Parameters: **cpfile (str)** (*Path to changepoints h5 file.*)
Returns: **(1d numpy array)**
Return type: Array of pre-computed principal components changepoints.

`moseq2_viz.util.`**`load_timestamps`** (`timestamp_file`, `col=0`)
Read timestamps from space delimited text file.

Parameters:
- **timestamp_file (str)** (*path to timestamp file*)
- **col (int)** (*column to load.*)

Returns: **ts (numpy array)**
Return type: loaded array of timestamps

`moseq2_viz.util.`**`make_separate_crowd_movies`** (`config_data`, `sorted_index`, `group_keys`, `labels`, `label_uuids`, `output_dir`, `ordering`, `sessions=False`)
Helper function that writes syllable crowd movies for each given grouping found in group_keys, and returns
a dictionary with session/group name keys paired with paths to their respective generated crowd movies.

Parameters:
- **config_data (dict)** (*Loaded crowd movie writing configuration parameters.*)
- **sorted_index (dict)** (*Loaded index file and sorted files in list.*)
- **group_keys (dict)** (*Dict of group/session name keys paired with UUIDS to match with labels.*)
- **labels (2d list)** (*list of syllable label lists for all sessions.*)
- **label_uuids (list)** (*list of corresponding session UUIDs for all sessions included in labels.*)
- **output_dir (str)** (*Path to output directory to save crowd movies in.*)
- **ordering (list)** (*ordering for the new mapping of the relabeled syllable usages.*)

Returns: **cm_paths (dict)**

Return type: group/session name keys paired with paths to their respectively generated syllable crowd movies.

`moseq2_viz.util.`**`np_cache`** (`function`)

`moseq2_viz.util.`**`parse_index`** (`index_file: str`) → tuple

Load an index file, and use extraction UUIDs as entries in a sorted index.

Parameters: **index_file**

Returns: **index (dict)** (*loaded index file contents in a dictionary*) **uuid_sorted (dict)** (*dictionary of a list of files and pca_score path.*)

`moseq2_viz.util.`**`read_yaml`** (`yaml_path: str`)

`moseq2_viz.util.`**`recursive_find_h5s`** (`root_dir='/Users/aymanzeine/Desktop/moseq/moseq2-viz/docs'`, `ext='.h5'`, `yaml_string='{}.yaml'`)

Recursively find h5 files, along with yaml files with the same basename.

Parameters:

- **root_dir (str)** (*path to directory containing h5*)
- **ext (str)** (*extension to search for.*)
- **yaml_string (str)** (*yaml file format name.*)

Returns: **h5s (list)** (*list of paths to h5 files*) **dicts (list)** (*list of paths to metadata files*) **yamls (list)** (*list of paths to yaml files*)

`moseq2_viz.util.`**`star`**

Apply a function to a tuple of args, by expanding the tuple into each of the function's parameters. It is curried, which allows one to specify one argument at a time.

Parameters:

- **f (function)** (*a function that takes multiple arguments*)
- **args (tuple)** (: a tuple to expand into `f`)

Returns:

Return type: the output of `f`

`moseq2_viz.util.`**`strided_app`** (`a`, `L`, `S`)

Taking subarrays from numpy array given stride

Parameters:

- **a (np.array)** (*array to get subarrays from.*)
- **L (int)** (*window length.*)
- **S (int)** (*stride size.*)

Returns: **(np.ndarray)**

Return type: sliced subarrays

Visualization Module

Visualization model containing all plotting functions and some dependent data pre-processing helper functions.

`moseq2_viz.viz.`**`check_types`** (`function`)

Decorator function to validate user input parameters for plotting syllable statistics, facilitated using functools wraps

Parameters: **function** (*plot_syll_stats_with_sem - the function to check parameters from.*)

Returns:

Return type: wrapped (function) returns the function to run

`moseq2_viz.viz.`**`clean_frames`** (`frames`, `medfilter_space=None`, `gaussfilter_space=None`, `tail_filter=None`, `tail_threshold=5`)

Filters frames using spatial filters such as Median or Gaussian filters.

Parameters:
- **frames (3D numpy array)** (*frames to filter.*)
- **medfilter_space (list)** (*list of len()==1, must be odd. Median space filter kernel size.*)
- **gaussfilter_space (list)** (*list of len()==2. Gaussian space filter kernel size.*)
- **tail_filter (cv2.getStructuringElement)** (*structuringElement to filter out mouse tails.*)
- **tail_threshold (int)** (*filtering threshold value*)

Returns: **out (3D numpy array)**

Return type: filtered numpy array.

`moseq2_viz.viz.`**`make_crowd_matrix`**`(slices, nexamples=50, pad=30, raw_size=(512, 424), frame_path='frames', crop_size=(80, 80), dur_clip=1000, offset=(50, 50), scale=1, center=False, rotate=False, min_height=10, legacy_jitter_fix=False, **kwargs)`

Creates crowd movie video numpy array.

Parameters:
- **slices (numpy array)** (*video slices of specific syllable label*)
- **nexamples (int)** (*maximum number of mice to include in crowd_matrix video*)
- **pad (int)** (*number of frame padding in video*)
- **raw_size (tuple)** (*video dimensions.*)
- **frame_path (str)** (*path to in-h5 frames variable*)
- **crop_size (tuple)** (*mouse crop size*)
- **dur_clip (int)** (*maximum clip duration.*)
- **offset (tuple)** (*centroid offsets from cropped videos*)
- **scale (int)** (*mouse size scaling factor.*)
- **center (bool)** (*indicate whether mice are centered.*)
- **rotate (bool)** (*rotate mice to orient them.*)
- **min_height (int)** (*minimum max height from floor to use.*)
- **legacy_jitter_fix (bool)** (*whether to apply jitter fix for K1 camera.*)
- **kwargs (dict)** (*extra keyword arguments*)

Returns: **crowd_matrix (3D numpy array)**

Return type: crowd movie for a specific syllable.

`moseq2_viz.viz.`**`plot_cp_comparison`**`(model_results, pc_cps, plot_all=False, best_model=None)`

Plot the changepoint-duration distributions of a given 1D arrays of model

and principal component changepoints.

Parameters:
- **model_cps (dict)** (*Multiple parsed model results aggregated into a single dict.*)
- **pc_cps (1D np.array)** (*Computed PC changepoints*)
- **plot_all (bool)** (*Plot all model changepoints for all keys included in model_cps dict.*)
- **best_model (str)** (*key name to the model with the closest median syllable duration*)

Returns: **fig (pyplot figure)** (*syllable usage ordered by frequency, 90% usage marked*) **ax (pyplot axis)** (*plotted scalar axis*)

`moseq2_viz.viz.`**`plot_mean_group_heatmap`**`(pdfs, groups)`

Computes the overall group mean of the computed PDFs and plots them.

Parameters:
- **pdfs (list)** (*list of 2d probability density functions (heatmaps) describing mouse position.*)
- **groups (list)** (*list of groups to compute means and plot*)

Returns: **fig (pyplot figure)**

Return type: plotted scalar scatter plot

`moseq2_viz.viz.`**`plot_syll_stats_with_sem`**(`complete_df`, `stat='usage'`, `ordering=None`, `max_sylls=None`, `groups=None`, `ctrl_group=None`, `exp_group=None`, `colors=None`, `fmt='o-'`, `figsize=(10, 5)`)

Plots a line and/or point-plot of a given pre-computed syllable statistic (usage, duration, or speed), with a SEM error bar with respect to the group. This function is decorated with the check types function that will ensure that the inputted data configurations are safe to plot in matplotlib.

Parameters:

- **complete_df (pd.DataFrame)** (*dataframe containing the statistical information about syllable data [usages, durs, etc.]*)
- **stat (str)** (*choice of statistic to plot: either usage, duration, or speed*)
- **ordering (str, list, None)** (*"m" for mutated, f"{stat}" for descending ordering with respect to original usage ordering.*)
- **max_sylls (int)** (*maximum number of syllable to include in plot*)
- **groups (list)** (*list of groups to include in plot. If groups=None, all groups will be plotted.*)
- **ctrl_group (str)** (*name of control group to base mutation sorting on.*)
- **exp_group (str)** (*name of experimental group to base mutation sorting on.*)
- **colors (list)** (*list of user-selected colors to represent the data*)
- **fmt (str)** (*str to indicate the kind of plot to make. "o-", "o", "–', etc.*)
- **figsize (tuple)** (*tuple value of length = 2, representing (columns x rows) of the plotted figure dimensions*)

Returns: **fig (pyplot figure)** (*plotted scalar scatter plot*) **ax (pyplot axis)** (*plotted scalar axis*)

`moseq2_viz.viz.`**`plot_verbose_heatmap`**(`pdfs`, `sessions`, `groups`, `subjectNames`)

Plots the PDF position heatmap for each session, titled with the group and subjectName.

Parameters:

- **pdfs (list)** (*list of 2d probability density functions (heatmaps) describing mouse position.*)
- **groups (list)** (*list of sessions corresponding to the pdfs indices*)
- **groups (list)** (*list of groups corresponding to the pdfs indices*)
- **subjectNames (list)** (*list of subjectNames corresponding to the pdfs indices*)

Returns: **fig (pyplot figure)**

Return type: plotted scalar scatter plot

`moseq2_viz.viz.`**`position_plot`**(`scalar_df, centroid_vars=['centroid_x_mm', 'centroid_y_mm'], sort_vars=['SubjectName', 'uuid'], group_var='group', sz=50, **kwargs`)

Creates a position summary graph that shows all the mice's centroid path throughout the respective sessions.

Parameters:

- **scalar_df (pandas DataFrame)** (*dataframe containing all scalar data*)
- **centroid_vars (list)** (*list of scalar variables to track mouse position*)
- **sort_vars (list)** (*list of variables to sort the dataframe by.*)
- **group_var (str)** (*groups df column to graph position plots for.*)
- **sz (int)** (*plot size.*)
- **kwargs (dict)** (*extra keyword arguments*)

Returns: **fig (pyplot figure)** (*pyplot figure object*) **ax (pyplot axis)** (*pyplot axis object*)

`moseq2_viz.viz.`**`save_fig`**(`fig`, `output_file`, `name='{}'`, `**kwargs`)

Convenience function for saving created/open matplotlib figures to PNG and PDF formats.

Parameters:
- **fig (pyplot.Figure)** (*open figure to save*)
- **output_file (str)** (*path to save figure to*)
- **name (str)** (*dynamic figure name; allows for overriding name with specific value/prefix*)
- **kwargs (dict)** (*dictionary containing additional figure saving parameters. (check plot-stats in wrappers.py)*)

Returns:

Return type: None

`moseq2_viz.viz.`**`scalar_plot`**`(scalar_df, sort_vars=['group', 'uuid'], group_var='group', show_scalars=['velocity_2d_mm', 'velocity_3d_mm', 'height_ave_mm', 'width_mm', 'length_mm'], headless=False, colors=None, **kwargs)`

Creates scatter plot of given scalar variables representing extraction results.

Parameters:
- **scalar_df (pandas DataFrame)**
- **sort_vars (list)** (*list of variables to sort the dataframe by.*)
- **group_var (str)** (*groups df column to graph position plots for.*)
- **show_scalars (list)** (*list of scalar variables to plot.*)
- **headless (bool)** (*exclude head of dataframe from plot.*)
- **colors (list)** (*list of color strings to indicate groups*)
- **kwargs (dict)** (*extra keyword variables*)

Returns: **fig (pyplot figure)** (*plotted scalar scatter plot*) **ax (pyplot axis)** (*plotted scalar axis*)

Subpackages

moseq2_viz.helpers package

Helpers - Wrappers Module

Wrapper functions for all functionality included in MoSeq2-Viz that is accessible via CLI or GUI. Each wrapper function executes the functionality from end-to-end given it's dependency parameters are inputted. (See CLI Click parameters)

`moseq2_viz.helpers.wrappers.`**`add_group_wrapper`**`(index_file, config_data)`

Given a pre-specified key and value, the index file will be updated with the respective found keys and values.

Parameters:
- **index_file (str)** (*path to index file*)
- **config_data (dict)** (*dictionary containing the user specified keys and values*)

Returns:

Return type: None

`moseq2_viz.helpers.wrappers.`**`copy_h5_metadata_to_yaml_wrapper`**`(input_dir, h5_metadata_path)`

Copy h5 metadata dictionary contents into the respective file's yaml file.

Parameters:
- **input_dir (str)** (*path to directory that contains h5 files.*)
- **h5_metadata_path (str)** (*path to data within h5 file to update yaml with.*)

Returns:

Return type: None

`moseq2_viz.helpers.wrappers.`**`get_best_fit_model_wrapper`**`(model_dir, cp_file, output_file, plot_all=False, ext='.p', fps=30)`

Given a directory containing multiple models trained on different kappa values, finds the model with the closest median syllable duration to the PC changepoints.

Function also graphs the distributions of the best fit model and PC changepoints

Parameters:
- **model_dir (str)** (*Path to directory containing multiple models.*)
- **cp_file (str)** (*Path to PCA changepoints*)
- **output_file (str)** (*Path to file to save figure to.*)
- **plot_all (bool)** (*Plot all model changepoint distributions.*)
- **ext (str)** (*Model extension to search for*)
- **fps (int)** (*Frames per second*)

Returns: **fig (pyplot figure)** (*syllable usage ordered by frequency, 90% usage marked*) **ax (pyplot axis)** (*plotted scalar axis*)

`moseq2_viz.helpers.wrappers.`**`init_wrapper_function`**(`index_file=None`, `model_fit=None`, `output_dir=None`, `output_file=None`)

Helper function that will optionally load the index file and a trained model given their respective paths. The function will also create any output directories given path to the output file or directory.

Parameters:
- **index_file (str)** (*path to index file to load.*)
- **model_fit (str)** (*path to model to use.*)
- **output_dir (str)** (*path to directory to save plots in.*)
- **output_file (str)** (*path to saved figures.*)

Returns: **index (dict)** (*loaded index file dictionary*) **sorted_index (dict)** (*OrderedDict object representing a sorted version of index*) **model_data (dict)** (*loaded model dictionary containing modeling results*)

`moseq2_viz.helpers.wrappers.`**`make_crowd_movies_wrapper`**(`index_file`, `model_path`, `config_data`, `output_dir`)

Wrapper function to create crowd movie videos and write them to individual files depicting respective syllable labels. Note: function is decorated with function performing initialization operations and saving the results in the kwargs variable. Decorator will retrieve the sorted_index dict and parse the model results into a single dict.

Parameters:
- **index_file (str)** (*path to index file*)
- **model_path (str)** (*path to trained model.*)
- **config_data (dict)** (*dictionary containing the user specified keys and values*)
- **output_dir (str)** (*directory to store crowd movies in.*)

Returns: **cm_paths (dict)**

Return type: Dictionary of syllables and their generated crowd movie paths

`moseq2_viz.helpers.wrappers.`**`plot_mean_group_position_pdf_wrapper`**(`index_file`, `output_file`)

Wrapper function that computes the PDF of the rodent's position throughout the respective sessions, and averages these values with respect to their groups to graph a mean position heatmap for each group.
Note: function is decorated with function performing initialization operations and saving the results in the kwargs variable.
Decorator will retrieve the sorted_index dict.

Parameters:
- **index_file (str)** (*path to index file.*)
- **output_file (str)** (*filename for the group heatmap graph.*)

Returns: **fig (pyplot figure)**

Return type: figure to graph in Jupyter Notebook.

`moseq2_viz.helpers.wrappers.`**`plot_scalar_summary_wrapper`**(`index_file`, `output_file`, `groupby='group'`, `colors=None`)

Wrapper function that plots scalar summary graphs.

Note: function is decorated with function performing initialization operations and saving the results in the kwargs variable.
Decorator will retrieve the sorted_index dict.

Parameters:
- **index_file (str)** (*path to index file.*)
- **output_file (str)** (*path to save graphs.*)
- **groupby (str)** (*scalar_df column to group sessions by when graphing scalar and position summaries*)
- **colors (list)** (*list of colors to serve as the sns palette in the scalar summary*)
- **kwargs (dict)** (*dict containing index dicts from given index file path.*)

Returns: **scalar_df (pandas DataFrame)** (*df containing scalar data per session uuid.*) *(Only accessible through GUI API)*

`moseq2_viz.helpers.wrappers.`**`plot_syllable_stat_wrapper`**`(model_fit, index_file, output_file, stat='usage', sort=True, count='usage', group=None, max_syllable=40, fmt='o-', ordering=None, ctrl_group=None, exp_group=None, colors=None, figsize=(10, 5))`

Wrapper function to plot specified syllable statistic.
Note: function is decorated with function performing initialization operations and saving the results in the kwargs variable. Decorator will retrieve the sorted_index dict and parse the model results into a single dict.

Parameters:
- **model_fit (str)** (*path to trained model file.*)
- **index_file (str)** (*path to index file.*)
- **output_file (str)** (*filename for syllable usage graph.*)
- **stat (str)** (*syllable statistic to plot: ['usage', 'speed', 'duration']*)
- **sort (bool)** (*sort syllables by usage.*)
- **count (str)** (*method to compute usages 'usage' or 'frames'.*)
- **group (tuple, list, None)** (*tuple or list of groups to separately model usages. (None to graph all groups)*)
- **max_syllable (int)** (*maximum number of syllables to plot.*)
- **fmt (str)** (*scatter plot format. "o-" for line plot with vertices at corresponding usages. "o" for just points.*)
- **ordering (list, range, str, None)** (*order to list syllables. Default is None to graph syllables [0-max_syllable).*) – Setting ordering to "m" will graph mutated syllable usage difference between ctrl_group and exp_group. None to graph default [0,max_syllable] in order. "usage" to plot descending order of usage values.
- **ctrl_group (str)** (*Control group to graph when plotting mutation differences via setting ordering to 'm'.*)
- **exp_group (str)** (*Experimental group to directly compare with control group.*)
- **colors (list)** (*list of colors to serve as the sns palette in the scalar summary. If None, default colors are used.*)
- **figsize (tuple)** (*tuple value of length = 2, representing (columns x rows) of the plotted figure dimensions*)

Returns: **plt (pyplot figure)**

Return type: graph to show in Jupyter Notebook.

`moseq2_viz.helpers.wrappers.`**`plot_transition_graph_wrapper`**`(index_file, model_fit, config_data, output_file)`

Wrapper function to plot transition graphs.
Note: function is decorated with function performing initialization operations and saving the results in the kwargs variable.
Decorator will retrieve the sorted_index dict and parse the model results into a single dict.

Parameters:
- **index_file (str)** (*path to index file*)
- **model_fit (str)** (*path to trained model.*)
- **config_data (dict)** (*dictionary containing the user specified keys and values*)
- **output_file (str)** (*filename for syllable usage graph.*)
- **kwargs (dict)** (*dict containing loaded model data and index dicts*)

Returns: **plt (pyplot figure)**

Return type: graph to show in Jupyter Notebook.

`moseq2_viz.helpers.wrappers.`**`plot_verbose_pdfs_wrapper`**`(index_file, output_file)`

Wrapper function that computes the PDF for the mouse position for each session in the index file. Will plot each session's heatmap with a "SessionName: Group"-like title.
Note: function is decorated with function performing initialization operations and saving the results in the kwargs variable.
Decorator will retrieve the sorted_index dict.

Parameters:
- **index_file (str)** (*path to index file.*)
- **output_file (str)** (*filename for the verbose heatmap graph.*)

Returns: **fig (pyplot figure)**

Return type: figure to graph in Jupyter Notebook.

moseq2_viz.info package

Info - Utilities Module

Utility functions for computing syllable usage entropy, and syllable transition entropy rate. These can be used for measuring modeling model performance and group separability.

`moseq2_viz.info.util.`**`entropy`**`(labels, truncate_syllable=40, smoothing=1.0, relabel_by='usage', get_session_sum=True)`

Computes syllable usage entropy, base 2.

Parameters:
- **labels (np.ndarray)** (*array of predicted syllable labels*)
- **truncate_syllable (int)** (*truncate list of relabeled syllables*)
- **smoothing (float)** (*a constant added to label usages before normalization*)
- **relabel_by (str)** (*mode to relabel predicted labels.*)
- **get_session_sum (bool)** (*Compute the sum of syllable usage entropies for each session.*)

Returns: **ent (list)**

Return type: list of entropy values for each syllable label.

`moseq2_viz.info.util.`**`entropy_rate`**`(labels, truncate_syllable=40, normalize='bigram', smoothing=1.0, tm_smoothing=1.0, relabel_by='usage', get_session_sum=True)`

Computes entropy rate, base 2 using provided syllable labels. If syllable labels have not been re-labeled by usage, this function will do so.

Parameters:

- **labels (list or np.ndarray)** (*a list of label arrays, where each entry in the list*) – is an array of labels for one subject.
- **truncate_syllable (int)** (*the number of labels to keep for this calculation*)
- **normalize (str)** (*the type of transition matrix normalization to perform. Options*) – are: 'bigram', 'rows', or 'columns'.
- **smoothing (float)** (*a constant added to label usages before normalization*)
- **tm_smoothing (float)** (*a constant added to label transtition counts before*) – normalization.
- **relabel_by (str)** (*how to re-order labels. Options are: 'usage' and 'frames'.*)
- **get_session_sum (bool)** (*Compute the sum of syllable usage entropies for each session.*)

Returns: **ent (list)**

Return type: list of entropy rates per syllable label

moseq2_viz.io package

IO - Video Module

Helper functions for handling crowd movie file writing and video metadata maintenance.

`moseq2_viz.io.video.`**`check_video_parameters`**`(index: dict)` → dict

Iterates through each extraction parameter file to verify extraction parameters were the same. If they weren't this function raises a RuntimeError.

Parameters: **index (dict)** (a *sorted_index* dictionary of extraction parameters.)

Returns: **vid_parameters (dict)**

Return type: a dictionary with a subset of the used extraction parameters.

`moseq2_viz.io.video.`**`write_crowd_movie_info_file`**`(model_path, model_fit, index_file, output_dir)`

Creates an info.yaml file in the crowd movie directory that holds model training parameters. This file helps identify the conditions from which the crowd movies were generated.

Parameters:

- **model_path (str)** (*path to model used to generate movies*)
- **model_fit (dict)** (*loaded ARHMM dict*)
- **index_file (str)** (*path to index file used with model*)
- **output_dir (str)** (*path to crowd movies directory to store file in.*)

Returns:

Return type: None

`moseq2_viz.io.video.`**`write_crowd_movies`**`(sorted_index, config_data, ordering, labels, label_uuids, output_dir)`

Creates syllable slices for crowd movies and writes them to files.

Parameters:

- **sorted_index (dict)** (*dictionary of sorted index data.*)
- **config_data (dict)** (*dictionary of visualization parameters.*)
- **filename_format (str)** (*string format that denotes the saved crowd movie file names.*)
- **ordering (list)** (*ordering for the new mapping of the relabeled syllable usages.*)
- **labels (numpy ndarray)** (*list of syllable usages*)
- **label_uuids (list)** (*list of session uuids each series of labels belongs to.*)
- **output_dir (str)** (*path directory where all the movies are written.*)

Returns:

Return type: None

`moseq2_viz.io.video.`**`write_frames_preview`**`(filename, frames=array([], dtype=float64), threads=6, fps=30, pixel_format='rgb24', codec='h264', slices=24, slicecrc=1, frame_size=None, depth_min=0, depth_max=80, get_cmd=False, cmap='jet', text=None, text_scale=1, text_thickness=2, pipe=None, close_pipe=True, progress_bar=True, **kwargs)`

Writes out a false-colored mp4 video. [Duplicate from moseq2-extract]

Parameters:

- **filename (str)**
- **frames (3D numpy array)** (*num_frames * r * c*)
- **threads (int)** (*number of threads to write file*)
- **fps (int)** (*frames per second*)
- **pixel_format (str)** (*ffmpeg image formatting flag.*)
- **codec (str)** (*ffmpeg image encoding flag.*)
- **slices (int)** (*number of slices per thread.*)
- **slicecrc (int)** (*check integrity of slices.*)
- **frame_size (tuple)** (*image dimensions*)
- **depth_min (int)** (*minimum mouse distance from bucket floor*)
- **depth_max (int)** (*maximum mouse distance from bucket floor*)
- **get_cmd (bool)** (*return ffmpeg command instead of executing the command in python.*)
- **cmap (str)** (*color map selection.*)
- **text (range(num_frames)** (*display frame number in output video.*)
- **text_scale (int)** (*text size.*)
- **text_thickness (int)** (*text thickness.*)
- **pipe (subProcess.Pipe object)** (*if not None, indicates that there are more frames to be written.*)
- **close_pipe (bool)** (*indicates whether video is done writing, and to close pipe to file-stream.*)
- **progress_bar (bool)** (*display progress bar.*)
- **kwargs (dict)** (*extra keyword arguments*)

Returns: **(subProcess.Pipe object)**

Return type: if there are more slices/chunks to write to, otherwise None.

moseq2_viz.model package

Model - Dist Module

Utility functions for estimating "behavioral distance" AKA model state similarity analysis.

`moseq2_viz.model.dist.`**`get_behavioral_distance`**`(index, model_file, whiten='all', distances=['ar[init]', 'scalars'], max_syllable=None, resample_idx=-1, dist_options={}, sort_labels_by_usage=True, count='usage')`

Computes the behavioral distance (square) matrices with respect to a predefined set of variables.

Parameters:
- **index (str)** (*Path to index file*)
- **model_file (str)** (*Path to trained model*)
- **whiten (str)** (*Indicates whether to whiten all PCs at once or each one at a time. Options = ['all', 'each']*)
- **distances (list)** (*List of distances to compute.*) – Available options = ['scalars', 'ar[init]', 'ar[dtw]', 'pca[dtw]', 'combined']
- **max_syllable (int)** (*Maximum number of syllables/AR matrices to include in analysis*)
- **resample_idx (int)** (*Indicates the parsing method according to the shape of the labels array.*)
- **dist_options (dict)** (*Dictionary holding each distance operations configurable parameters*)
- **sort_labels_by_usage (bool)** (*Indicates whether to relabel syllables by count ordering*)
- **count (str)** (*Indicates what ordering to relabel syllables by. Options = ['usage', 'frames']*)

Returns: **dist_dict (dict)**

Return type: Dictionary containing all computed behavioral square distance matrices

`moseq2_viz.model.dist.`**`get_behavioral_distance_ar`**(`ar_mat`, `init_point=None`, `sim_points=10`, `max_syllable=40`, `dist='correlation'`, `parallel=False`)

Computes behavioral distance with respect to the model's AutoRegressive matrices. Affords either AR trajectory correlation distance, or computing dynamically time-warped trajectory distances.

Parameters:
- **ar_mat (3D numpy array)** (*Trained model AutoRegressive matrices; shape=(max_syllable, npcs, npcs*nlags+1)*)
- **init_point (list)** (*Initial values as a reference point for distance estimation*)
- **sim_points (int)** (*Number of AR trajectories to simulate*)
- **max_syllable (int)** (*Max number of syllables included in the analysis. Should be equal to ar_mat.shape[0]*)
- **dist (str)** (*Distance operation to compute. Either 'correlation' or 'dtw'.*)
- **parallel (bool)** (*Use multiprocessing to compute dtw distances.*)

Returns: **ar_dist (2D numpy array)** (*Computed AR trajectory distances for each AR matrix/model state.*) *shape=(max_syllable, max_syllable)*

`moseq2_viz.model.dist.`**`get_init_points`**(`pca_scores`, `model_labels`, `max_syllable=40`, `nlags=3`, `npcs=10`)

Compute initial AR trajectories based on a cumulative average of lagged-PC Scores over nlags.

Parameters:
- **pca_scores (2D numpy array)** (*Loaded PCA Scores. Shape=(npcs, nsamples)*)
- **model_labels (2D list)** (*list of 1D numpy arrays of relabeled/sorted syllable labels*)
- **max_syllable (int)** (*Maximum number of syllables to include.*)
- **nlags (int)** (*Number of lagged frames.*)
- **npcs (int)** (*Number of PCs to use in computation.*)

Returns: **syll_average (list)** – PC scores array. Shape = (max_syllables, nlags*2 +1, npcs)

Return type: List containing 2D np arrays of average syllable trajectories over a nlag-strided

`moseq2_viz.model.dist.`**`reformat_dtw_distances`**(`full_mat`, `nsyllables`, `rescale=True`)

Reduce full (max states) dynamically time-warped PC Score distance matrices to only include dimensions for a total of nsyllables. Formatting the 3D matrix (full_mat) to 2D to show the correlation distances from each state pair.

Parameters:
- **full_mat (3D np.ndarray)** (*DTW distance matrices for all model states/syllables.*)
- **nsyllables (int)** (*Number of syllables to include in truncated DTW distance matrix.*)
- **rescale (bool)** (*Rescale truncated dtw-distance matrices to match output distribution.*)

Returns: **rmat (2D np array)**

Return type: Reformatted-Truncated DTW Distance Matrix; shape = (nsyllables, nsyllables)

Model - Label Utilities Module

Syllable label information utility functions. Contains duplicate functions from moseq2-model + additional syllable sorting/relabeling functions.

`moseq2_viz.model.label_util.`**`get_sorted_syllable_stat_ordering`**`(complete_df, stat='usage')`

Computes the sorted ordering of the given DataFrame with respect to the chosen stat.

Parameters:
- **complete_df (pd.DataFrame)** (*DataFrame containing the statistical information about syllable data [usages, durs, etc.]*)
- **stat (str)** (*choice of statistic to order mutations by: {usage, duration, speed}.*)

Returns: **ordering (list)** (*list of newly mapped array (syllable label) indices.*) **relabel_mapping (dict)** (*dict of mappings from old to (descending-order y-label sorting) and ascending-order x range.*)

`moseq2_viz.model.label_util.`**`get_syllable_mutation_ordering`**`(complete_df, ctrl_group, exp_group, max_sylls=None, stat='usage')`

Computes the syllable ordering for the difference of the inputted groups (exp - ctrl). The sorted result will yield an array will indices depicting the largest positive (upregulated) difference between exp and ctrl groups on the left, and vice versa on the right.

Parameters:
- **complete_df (pd.DataFrame)** (*dataframe containing the statistical information about syllable data [usages, durs, etc.]*)
- **ctrl_group (str)** (*Control group.*)
- **exp_group (str)** (*Experimental group.*)
- **max_sylls (int)** (*maximum number of syllables to include in ordering.*)
- **stat (str)** (*choice of statistic to order mutations by: {usage, duration, speed}.*)

Returns: **mutation_ordering (list)**

Return type: list of array indices for the new label mapping.

`moseq2_viz.model.label_util.`**`syll_duration`**`(labels: numpy.ndarray)` → numpy.ndarray

Computes the duration of each syllable.

Parameters: **labels (np.ndarray)** (*array of syllable labels for a mouse.*)

Returns: **durations (np.ndarray)**

Return type: array of syllable durations.

`moseq2_viz.model.label_util.`**`syll_id`**`(labels: numpy.ndarray)` → numpy.ndarray

Returns the syllable label at each syllable transition.

Parameters: **labels (np.ndarray)** (*array of syllable labels for a mouse.*)

Returns: **labels[onsets] (np.ndarray)**

Return type: an array of compressed labels.

`moseq2_viz.model.label_util.`**`syll_onset`**`(labels: numpy.ndarray)` → numpy.ndarray

Finds indices of syllable onsets.

Parameters: **labels (np.ndarray)** (*array of syllable labels for a mouse.*)

Returns: **indices (np.ndarray)**

Return type: an array of indices denoting the beginning of each syllables.

`moseq2_viz.model.label_util.`**`to_df`**`(labels, uuid)` → pandas.core.frame.DataFrame
Convert labels numpy.ndarray to pandas.DataFrame

Parameters:
- **labels (np.ndarray)** (*array of syllable labels for a mouse.*)
- **uuid (list)** (*list of session uuids representing each series of labels.*)

Returns: **df (pd.DataFrame)**

Return type: DataFrame of syllables, durations, onsets, and session uuids.

Model - Utilities Module

Utility functions specifically responsible for handling model data during pre and post processing.

`moseq2_viz.model.util.`**`calculate_label_durations`**`(label_arr: Union[dict, numpy.ndarray])` → Union[dict, numpy.ndarray]
Calculates syllable label durations.

Parameters: **label_arr (dict or np.ndarray)** (*list or dict of predicted syllable labels.*)

Returns: **np.diffs(inds) (np.ndarray)**

Return type: list of durations for each syllable in respective label order.

`moseq2_viz.model.util.`**`calculate_syllable_usage`**`(labels: Union[dict, pandas.core.frame.DataFrame])`
Calculates a dictionary of uuid to syllable usage key-values pairs.

Parameters: **label_arr (dict or pd.DataFrame)** (*list or DataFrame of predicted syllable labels.*)

Returns: **(dict)**

Return type: dictionary of syllable usage probabilities.

`moseq2_viz.model.util.`**`compress_label_sequence`**`(label_arr: Union[dict, numpy.ndarray])` → numpy.ndarray
Removes repeating values from a label sequence. It assumes the first label is '-5', which is unused for behavioral analysis, and removes it.

Parameters: **label_arr (dict or np.ndarray)** (*list or dict of predicted syllable labels.*)

Returns: **label_arr[inds] (dict or np.ndarray)**

Return type: the compressed version of the label arrays.

`moseq2_viz.model.util.`**`compute_model_changepoints`**`(model, fps=30.0)`
Computes the given trained model's syllable label changepoints.

Parameters:
- **model (dict)** (*dict of parsed trained model results.*)
- **fps (int)** (*frames per second.*)

Returns: **model_cps (1d np.array)**

Return type: 1-dimensional list of syllable block durations.

`moseq2_viz.model.util.`**`find_label_transitions`**`(label_arr: Union[dict, numpy.ndarray])` → numpy.ndarray
Finds indices where a label transitions into another label. This function is cached to increase performance because it is called frequently.

Parameters: **label_arr (dict or np.ndarray)** (*list or dict of predicted syllable labels.*)

Returns: **inds (np.ndarray)**

Return type: Array of syllable transition indices for each session uuid.

`moseq2_viz.model.util.`**`get_best_fit`**`(cp_path, model_results)`
Returns the model with the closest median syllable duration to the PCA changepoints.

Parameters:
- **cp_path (str)** (*Path to PCA Changepoints h5 file.*)
- **model_results (dict)** (*dict of pairs of model names paired with dict containing their respective changepoints.*)

Returns: **best_model (str)** (*Computed best-fit model key.*) **pca_cps (1D array)** (*1-dimensional list of pc score block durations.*)

`moseq2_viz.model.util.`**`get_frame_label_df`**`(labels, uuids, groups)`

Returns a DataFrame with rows for each session, frame indices as columns, and syllable label values corresponding to these frames+sessions for each frame.

Parameters:
- **labels (2D np.array)** (*list of np arrays containing syllable labels with respect to individually labeled frames.*) – Index by uuids.
- **uuids (list)** (*list of uuid strings corresponding to each "row" in labels.*)
- **groups (list)** (*list of group strings corresponding to each "row" in labels.*)

Returns: **label_df (pd.DataFrame)** (*Dataframe of shape (nsessions x max(len(labels))). At columns exceeding session's labeled frame*) *count, values will be np.NaN. Rows are indexed by Multi-Index([['group', 'uuid'],…)*

`moseq2_viz.model.util.`**`get_mouse_syllable_slices`**`(syllable: int, labels: numpy.ndarray)` → Iterator[slice]

Return a generator containing slices of *syllable* indices for a mouse.

Parameters:
- **syllable (list)** (*list of syllables to get slices from.*)
- **labels (np.ndarrary)** (*list of label predictions for each session.*)

Returns: **slices (list)**

Return type: list of syllable label slices; e.g. [slice(3, 6, None), slice(9, 12, None)]

`moseq2_viz.model.util.`**`get_syllable_slices`**

Get the indices that correspond to a specific syllable for each animal in a modeling run.

Parameters:
- **syllable (int)** (*syllable number to get slices of.*)
- **labels (np.ndarrary)** (*list of label predictions for each session.*)
- **label_uuids (list)** (*list of uuid keys corresponding to each session.*)
- **index (dict)** (*index file contents contained in a dict.*)
- **trim_nans (bool)** (*flag to use the pca scores file for removing time points that contain NaNs.*)
- **Only use if you have not already trimmed NaNs previously (i.e. in `scalars_to_dataframe`).**

Returns: **syllable_slices (list)** (a list of indices for *syllable* in the *labels* array. Each item in the list) *is a tuple of (slice, uuid, h5_file).*

`moseq2_viz.model.util.`**`get_syllable_statistics`**`(data, fill_value=-5, max_syllable=100, count='usage')`

Compute the syllable statistics from a set of model labels

Parameters:
- **data (list of np.array of ints)** (*labels loaded from a model fit.*)
- **fill_value (int)** (*lagged label values in the labels array to remove.*)
- **max_syllable (int)** (*maximum syllable to consider.*)
- **count (str)** (*how to count syllable usage, either by number of emissions (usage), or number of frames (frames).*)

Returns: **usages (OrderedDict)** (*default dictionary of usages*) **durations (OrderedDict)** (*default dictionary of durations*)

moseq2_viz.model.util.**get_syllable_usages**(model_data, max_syllable=100, count='usage')
Computes the overall syllable usages, and returns a 1D array of their corresponding usage values.

Parameters:
- **model_data (dict)** (*dict object of modeling results*)
- **max_syllable (int)** (*number of syllables to compute the mean usage for.*)
- **count (str)** (*option for whether to count syllable usages; by 'frames', or 'usage'.*)

Returns: **syllable_usages (1D np array)**

Return type: array of sorted syllable usages for all syllables in model

moseq2_viz.model.util.**labels_to_changepoints**(labels, fs=30.0)
Compute the transition matrix from a set of model labels.

Parameters:
- **labels (list of np.array of ints)** (*labels loaded from a model fit.*)
- **fs (float)** (*sampling rate of camera.*)

Returns: **cp_dist (list of np.array of floats)**

Return type: list of block durations per element in labels list.

moseq2_viz.model.util.**make_separate_crowd_movies**(config_data, sorted_index, group_keys, labels, label_uuids, output_dir, ordering, sessions=False)

Helper function that writes syllable crowd movies for each given grouping found in group_keys, and returns

a dictionary with session/group name keys paired with paths to their respective generated crowd movies.

Parameters:
- **config_data (dict)** (*Loaded crowd movie writing configuration parameters.*)
- **sorted_index (dict)** (*Loaded index file and sorted files in list.*)
- **group_keys (dict)** (*Dict of group/session name keys paired with UUIDS to match with labels.*)
- **labels (2d list)** (*list of syllable label lists for all sessions.*)
- **label_uuids (list)** (*list of corresponding session UUIDs for all sessions included in labels.*)
- **output_dir (str)** (*Path to output directory to save crowd movies in.*)
- **ordering (list)** (*ordering for the new mapping of the relabeled syllable usages.*)
- **sessions (bool)** (*indicates whether session crowd movies are being generated.*)

Returns: **cm_paths (dict)**

Return type: group/session name keys paired with paths to their respectively generated syllable crowd movies.

moseq2_viz.model.util.**merge_models**(model_dir, ext='p', count='usage')
Merges model states by using the Hungarian Algorithm: a minimum distance state matching algorithm. User inputs a directory containing models to merge, (and the name of the latest-trained model) to match other model states to.

Parameters:
- **model_dir (str)** (*path to directory containing all the models to merge.*)
- **ext (str)** (*model extension to search for.*)
- **count (str)** (*method to compute usages 'usage' or 'frames'.*)

Returns: **model_data (dict)** (*a dictionary containing all the new*) *keys and state-matched labels.*

moseq2_viz.model.util.**normalize_pcs**(pca_scores: dict, method: str = 'z') → dict
Normalize PC scores. Options are: demean, zscore, ind-zscore. demean: subtract the mean from each score.

Parameters:
- **pca_scores (dict)** (*dict of uuid to PC-scores key-value pairs.*)
- **method (str)** (*the type of normalization to perform (demean, zscore, ind-zscore)*)

Returns: **norm_scores (dict)**

Return type: a dictionary of normalized PC scores.

`moseq2_viz.model.util.`**`parse_batch_modeling`**(`filename`)

Reads model parameter scan training results into a single dictionary.

Parameters: **filename (str)** (*path to h5 manifest file containing all the model results.*)

Returns: **results_dict (dict)** (*dictionary containing each model's training results,*) *concatenated into a single list. Maintaining the original structure as though it was a single model's results.*

`moseq2_viz.model.util.`**`parse_model_results`**(`model_obj`, `restart_idx=0`, `resample_idx=-1`, `map_uuid_to_keys: bool = False`, `sort_labels_by_usage: bool = False`, `count: str = 'usage'`) → dict

Reads model file and returns dictionary containing modeled results and some metadata.

Parameters:

- **model_obj (str or results returned from joblib.load)** (*path to the model fit or a loaded model fit*)
- **restart_idx (int)** (*Select which model restart to load. (Only change for models with multiple restarts used)*)
- **resample_idx (int)** (*Indicates the parsing method according to the shape of the labels array.*)
- **map_uuid_to_keys (bool)** (*for labels, make a dictionary where each key, value pair*)
- **contains the uuid and the labels for that session.**
- **sort_labels_by_usage (bool)** (*sort labels by their usages.*)
- **count (str)** (*how to count syllable usage, either by number of emissions (usage),*)
- **or number of frames (frames).**

Returns: **output_dict (dict)**

Return type: dictionary with labels and model parameters

`moseq2_viz.model.util.`**`relabel_by_usage`**(`labels`, `fill_value=-5`, `count='usage'`)

Resort model labels by their usages.

Parameters:

- **labels (list of np.array of ints)** (*labels loaded from a model fit*)
- **fill_value (int)** (*value prepended to modeling results to account for nlags*)
- **count (str)** (*how to count syllable usage, either by number of emissions (usage), or number of frames (frames)*)

Returns: **labels (list of np.array of ints)** (*labels resorted by usage*) **sorting (list)** (*the new label sorting. The index corresponds to the new label,*) *while the value corresponds to the old label.*

`moseq2_viz.model.util.`**`results_to_dataframe`**(`model_dict, index_dict, sort=False, count='usage', max_syllable=40, include_meta=['SessionName', 'SubjectName', 'StartTime'], compute_labels=False`)

Converts inputted model dictionary to DataFrame with user specified metadata columns. Also generates a DataFrame containing frame-by-frame syllable labels for all sessions.

Parameters:

- **model_dict (dict)** (*loaded model results dictionary.*)
- **index_dict (dict)** (*loaded index file dictionary*)
- **sort (bool)** (*indicate whether to relabel syllables by usage.*)
- **count (str)** (*indicate what to sort the labels by: usage, or frames*)
- **normalize (bool)** (*unused.*)
- **max_syllable (int)** (*maximum number of syllables to include in dataframe.*)
- **include_meta (list)** (*mouse metadata to include in dataframe.*)

Returns: **df (pd.DataFrame)** (*DataFrame containing model results and metadata.*) **label_df (pd.DataFrame)** (*DataFrame containing syllable labels at each frame (nsessions rows x max(nframes) cols)*)

`moseq2_viz.model.util.`**`retrieve_pcs_from_slices`**(`slices`, `pca_scores`, `max_dur=60`, `min_dur=3`, `max_samples=100`, `npcs=10`, `subsampling=None`, `remove_offset=False`, `**kwargs`)

Subsample Principal components from syllable slices

Parameters:
- **slices (np.ndarray)** (*syllable slice or subarray to compute PCs for*)
- **pca_scores (np.ndarray)** (*PC scores for respective session.*)
- **max_dur (int)** (*maximum slice length.*)
- **min_dur (int)** (*minimum slice length.*)
- **max_samples (int)** (*maximum number of samples to slices to retrieve.*)
- **npcs (int)** (*number of pcs to use.*)
- **subsampling (int)** (*number of neighboring PCs to subsample from.*)
- **remove_offset (bool)** (*indicate whether to remove lag values.*)
- **kwargs (dict)** (*unused.*)

Returns: **syllable_matrix (np.ndarray)**

Return type: 3D matrix of subsampled PC projected syllable slices.

`moseq2_viz.model.util.`**`simulate_ar_trajectory`**(`ar_mat`, `init_points=None`, `sim_points=100`)

Simulate auto-regressive trajectory matrices from optionally randomly projected initalized points.

Parameters:
- **ar_mat (3D np.ndarray)** (*numpy array representing the autoregressive matrix of each model state.*)
- **init_points (2D np.ndarray)** (*pre-initialzed array of the same shape as the ar-matrices.*)
- **sim_points (int)** (*number of trajectories to simulate.*)

Returns: **sim_mat[nlags**

Return type:] simulated AR matrices excluding lagged values.

`moseq2_viz.model.util.`**`sort_batch_results`**(`data`, `averaging=True`, `filenames=None`, `**kwargs`)

Sort modeling results from batch/parameter scan.

Parameters:
- **data (np.ndarray)** (*model AR-matrices.*)
- **averaging (bool)** (*return an average of all the model AR-matrices.*)
- **filenames (list)** (*list of paths to fit models.*)
- **kwargs (dict)** (*dict of extra keyword arguments.*)

Returns: **new_matrix (np.ndarray)** (*either average of all AR-matrices, or top sorted matrix*) **param_dict (dict)** (*model parameter dict*) **filename_index (list)** (*list of filenames associated with each model.*)

`moseq2_viz.model.util.`**`syllable_slices_from_dict`**

Reads dictionary of syllable labels, and returning a dict of syllable slices.

Parameters:
- **syllable (list)** (*list of syllables to get slices from.*)
- **labels (np.ndarrary)** (*list of label predictions for each session.*)
- **index (dict)** (*index file contents contained in a dict.*)
- **filter_nans (bool)** (*replace NaN values with 0.*)

Returns: **vals (dict)**

Return type: key-value pairs of syllable slices per session uuid.

`moseq2_viz.model.util.`**`whiten_pcs`**(`pca_scores`, `method='all'`, `center=True`)

Whiten PC scores using Cholesky whitening

Args:

pca_scores (dict): dictionary where values are pca_scores (2d np arrays) method (str): 'all' to whiten using the covariance estimated from all keys, or 'each' to whiten each separately center (bool): whether or not to center the data

Returns:

whitened_scores (dict): dictionary of whitened pc scores

Examples:

Load in pca_scores and whiten

>> from moseq2_viz.util import h5_to_dict >> from moseq2_viz.model.util import whiten_pcs >> pca_scores = h5_to_dict('pca_scores.h5', '/scores') >> whitened_scores = whiten_pcs(pca_scores, method='all')

moseq2_viz.scalars package

Scalars - Utilities Module

Utility functions responsible for handling all scalar data-related operations.

`moseq2_viz.scalars.util.`**`compute_all_pdf_data`**(`scalar_df, normalize=False, centroid_vars=['centroid_x_mm', 'centroid_y_mm'], key='SubjectName'`)

Computes a position PDF for all sessions and returns the pdfs with corresponding lists of

groups, session uuids, and subjectNames.

Parameters:

- **scalar_df (pd.DataFrame)** (*DataFrame containing all scalar data + uuid columns for all stacked sessions*)
- **normalize (bool)** (*Indicates whether normalize the pdfs.*)
- **centroid_vars (list)** (*list of strings for column values to use when computing mouse position.*)
- **key (str)** (*metadata column to return info from.*)

Returns: **pdfs (list)** (*list of 2d np.arrays of PDFs for each session.*) **groups (list)** (*list of strings of groups corresponding to pdfs index.*) **sessions (list)** (*list of strings of session uuids corresponding to pdfs index.*) **subjectNames (list)** (*list of strings of subjectNames corresponding to pdfs index.*)

`moseq2_viz.scalars.util.`**`compute_mean_syll_scalar`**(`complete_df`, `scalar_df`, `label_df`, `scalar='centroid_speed_mm'`, `groups=None`, `max_sylls=40`)

Computes the mean syllable scalar-value based on the time-series scalar dataframe and the selected scalar. Finds the frame indices with corresponding each of the label values (up to max syllables) and looks up the scalar values in the dataframe.

Parameters:

- **complete_df (pd.DataFrame)** (*DataFrame containing syllable statistic results for each uuid.*)
- **scalar_df (pd.DataFrame)** (*DataFrame containing all scalar data + uuid columns for all stacked sessions*)
- **label_df (pd.DataFrame)** (*DataFrame containing syllable labels at each frame (nsessions rows x max(nframes) cols)*)
- **scalar (str)** (*Selected scalar column to compute mean value for syllables*)
- **groups (list)** (*list of strings corresponding to group names to only compute scalars for.*)
- **max_sylls (int)** (*maximum amount of syllables to include in output.*)

Returns: **complete_df (pd.DataFrame)**

Return type: updated input dataframe with a speed value for each syllable merge in as a new column.

`moseq2_viz.scalars.util.`**`compute_mouse_dist_to_center`**(`roi`, `centroid_x_px`, `centroid_y_px`)

Given the session's ROI shape and the frame-by-frame (x,y) pixel centroid location

to compute the mouse's relative distance to the center of the bucket.

Parameters:
- **roi (tuple)** (*Tuple of session's arena dimensions.*)
- **centroid_x_px (1D np.array)** (*x-coordinate of the mouse centroid throughout the recording*)
- **centroid_y_px (1D np.array)** (*y-coordinate of the mouse centroid throughout the recording*)

Returns: **dist_to_center (1D np.array)**

Return type: array of normalized mouse centroid distance to the bucket center.

`moseq2_viz.scalars.util.`**`compute_session_centroid_speeds`**`(scalar_df, grouping_keys=['uuid', 'group'], centroid_keys=['centroid_x_mm', 'centroid_y_mm'])`

Computes the centroid speed float value of the mouse given the Series of mm x and y coordinates

from the scalar_df DataFrame.

Parameters:
- **scalar_df (pd.DataFrame)** (*DataFrame containing all scalar data + uuid columns for all stacked sessions*)
- **grouping_keys (list)** (*list of column names to group the df keys by*)
- **centroid_keys (list)** (*list of column names containing the centroid values.*)

Returns: **sc_speed (pd.DataFrame)** (*single column of a DataFrame containing centroid value to be appended*) *as new column to scalar_df*

`moseq2_viz.scalars.util.`**`compute_syllable_position_heatmaps`**`(complete_df, scalar_df, label_df, centroid_keys=['centroid_x_mm', 'centroid_y_mm'], syllables=range(0, 40))`

Computes position PDFs for the given syllables in each of the sessions included in the results and label dataframes.

Parameters:
- **complete_df (pd.DataFrame)** (*DataFrame containing syllable statistic results for each uuid.*)
- **scalar_df (pd.DataFrame)** (*DataFrame containing all scalar data + uuid columns for all stacked sessions*)
- **label_df (pd.DataFrame)** (*DataFrame containing syllable labels at each frame (nsessions rows x max(nframes) cols)*)
- **centroid_keys (list)** (*list of column names containing the centroid values used to compute mouse position.*)
- **syllables (list)** (*List of syllables to compute heatmaps for.*)

Returns: **complete_df (pd.DataFrame)** – new PDF column corresponding to each session-syllable pair.

Return type: Inputted model results dataframe with a

`moseq2_viz.scalars.util.`**`convert_legacy_scalars`**`(old_features, force: bool = False, true_depth: float = 673.1)` → dict

Converts scalars in the legacy format to the new format, with explicit units.

Parameters:
- **old_features (str, h5 group, or dictionary of scalars)** (*filename, h5 group,*)
- **or dictionary of scalar values.**
- **force (bool)** (*force the conversion of centroid_[xy]_px into mm.*)
- **true_depth (float)** (*true depth of the floor relative to the camera (673.1 mm by default)*)

Returns: **features (dict)**

Return type: dictionary of scalar values

`moseq2_viz.scalars.util.`**`convert_pxs_to_mm`**`(coords, resolution=(512, 424), field_of_view=(70.6, 60), true_depth=673.1)`

Converts x, y coordinates in pixel space to mm # http://stackoverflow.com/questions/17832238/kinect-intrinsic-parameters-from-field-of-view/18199938#18199938 # http://www.imaginativeuniversal.com/blog/post/2014/03/05/quick-reference-kinect-1-vs-kinect-2.aspx # http://smeenk.com/kinect-field-of-view-comparison/

Parameters:
- **coords (list)** (*list of [x,y] pixel coordinate lists.*)
- **resolution (tuple)** (*video frame size.*)
- **field_of_view (tuple)** (*camera focal lengths.*)
- **true_depth (float)** (*detected distance between depth camera and bucket floor.*)

Returns: **new_coords (list)**

Return type: list of same [x,y] coordinates in millimeters.

`moseq2_viz.scalars.util.`**`find_and_load_feedback`**`(extract_path, input_path)`

`moseq2_viz.scalars.util.`**`generate_empty_feature_dict`**`(nframes)` → dict

Generates a dict of numpy array of zeros of length nframes for each feature parameter.

Parameters: **nframes (int)** (*length of video*)

Returns: **(dict)**

Return type: dictionary feature to numpy 0 arrays of length nframes key-value pairs.

`moseq2_viz.scalars.util.`**`get_scalar_map`**`(index, fill_nans=True, force_conversion=False)`

Returns a dictionary of scalar values loaded from an index dictionary.

Parameters:
- **index (dict)** (*dictionary of index file contents.*)
- **fill_nans (bool)** (*indicate whether to replace NaN values with 0.*)
- **force_conversion (bool)** (*force the conversion of centroid_[xy]_px into mm.*)

Returns: **scalar_map (dict)**

Return type: dictionary of all the scalar values acquired after extraction.

`moseq2_viz.scalars.util.`**`get_scalar_triggered_average`**`(scalar_map, model_labels, max_syllable=40, nlags=20, include_keys=['velocity_2d_mm', 'velocity_3d_mm', 'width_mm', 'length_mm', 'height_ave_mm', 'angle'], zscore=False)`

Get averages of selected scalar keys for each syllable.

Parameters:
- **scalar_map (dict)** (*dictionary of all the scalar values acquired after extraction.*)
- **model_labels (dict)** (*dictionary of uuid to syllable label array pairs.*)
- **max_syllable (int)** (*maximum number of syllables to use.*)
- **nlags (int)** (*number of lags to use when averaging over a series of PCs.*)
- **include_keys (list)** (*list of scalar values to load averages of.*)
- **zscore (bool)** (*indicate whether to z-score loaded values.*)

Returns: **syll_average (dict)**

Return type: dictionary of scalars for each syllable sequence.

`moseq2_viz.scalars.util.`**`get_syllable_pdfs`**`(pdf_df, normalize=True, syllables=range(0, 40), groupby='group')`

Computes the mean syllable position PDF/Heatmap for the given groupings. Either mean of modeling groups: groupby='group', or a verbose list of all the session's syllable PDFs groupby='SessionName'

Parameters:
- **pdf_df (pd.DataFrame)** (*model results dataframe including a position PDF column containing 2D numpy arrays.*)
- **normalize (bool)** (*Indicates whether normalize the pdf scales.*)
- **syllables (list)** (*list of syllables to get a grouping of.*)
- **groupby (str)** (*column name to group the df keys by. (either group, or SessionName)*)

Returns: **group_syll_pdfs (list)** (*2D list of computed pdfs of shape ngroups x nsyllables*) **groups (list)** (*list of corresponding names to each row in the group_syll_pdfs list*)

`moseq2_viz.scalars.util.`**`handle_feedback_data`**(`scalar_dict`, `dct`, `pth`, `input_file`, `nframes`)

Reads recorded neural stimulation timestamps from the given input file or

h5 path. Appends the feedback information to the scalar dict to include in the outputted scalar_df.

Parameters:
- **scalar_dict (dict)** (*Session scalar dictionary to add feedback info to*)
- **dct (dict)** (*Loaded h5 file dict*)
- **pth (str)** (*Path to feedback data in h5 file*)
- **input_file (str)** (*Path to feedback timestamps file*)
- **nframes (int)** (*Number of frames included in current session*)

Returns: **scalar_dict (dict)** (*Inputted scalar dict with appended feedback status info key-pairs*) **skip (bool)** (*Indicator for whether loading feedback data has failed, triggers a "continue" in scalars_to_dataframe()*)

`moseq2_viz.scalars.util.`**`is_legacy`**(`features: dict`)

Checks a dictionary of features to see if they correspond with an older version of moseq.

Parameters: **features**

Returns: **(bool)**

Return type: true if the dict is from an old dataset

`moseq2_viz.scalars.util.`**`make_a_heatmap`**(`position`)

Uses a kernel density function to create a heatmap representing the mouse position throughout a single session.

Parameters: **position (2d numpy array)** (*2d array of mouse centroid coordinates (for a single session),*) – computed from compute_session_centroid_speeds.

Returns: **pdf (2d numpy array)**

Return type: shape (50, 50) representing the PDF for the mouse position over the whole session.

`moseq2_viz.scalars.util.`**`nanzscore`**(`data`)

Z-score numpy array that may contain NaN values.

Parameters: **data (np.ndarray)** (*array of scalar values.*)

Returns: **data (np.ndarray)**

Return type: z-scored data.

`moseq2_viz.scalars.util.`**`process_scalars`**(`scalar_map: dict`, `include_keys: list`, `zscore: bool = False`) → dict

Fill NaNs and possibly zscore scalar values.

Parameters:
- **scalar_map (dict)** (*dictionary of all the scalar values acquired after extraction.*)
- **include_keys (list)** (*scalar keys to process.*)
- **zscore (bool)** (*indicate whether to z-score loaded values.*)

`moseq2_viz.scalars.util.`**`remove_nans_from_labels`**(`idx`, `labels`)

Removes the frames from *labels* where *idx* has NaNs in it.

Parameters:
- **idx (list)** (*indices to remove NaN values at.*)
- **labels (list)** (*label list containing NaN values.*)

Returns: **(list)**

Return type: label list excluding NaN values at given indices

`moseq2_viz.scalars.util.`**`scalars_to_dataframe`**`(index: dict, include_keys: list = ['SessionName', 'SubjectName', 'StartTime'], disable_output=False, include_feedback=None, force_conversion=True)`

Generates a dataframe containing scalar values over the course of a recording session. If a model string is included, then return only animals that were included in the model Called to sort scalar metadata information when graphing in plot-scalar-summary.

Parameters:
- **index (dict)** (a sorted_index generated by *parse_index* or *get_sorted_index*)
- **include_keys (list)** (*a list of other moseq related keys to include in the dataframe*)
- **include_model (str)** (*path to an existing moseq model*)
- **disable_output (bool)** (*indicate whether to show tqdm output.*)
- **include_feedback (bool)** (*indicate whether to include timestamp data*)
- **force_conversion (bool)** (*force the conversion of centroid_[xy]_px into mm.*)

Returns: **scalar_df (pandas DataFrame)**

Return type: DataFrame of loaded scalar values with their selected metadata.

`moseq2_viz.scalars.util.`**`star_valmap`**`(func, d)`

Index

- **genindex**

Index

Symbols

A

C

--version

moseq2-viz-add-group command line option

--exact

--group <group>

--key <key>

--lowercase

--negative

--value <value>

-e

-g

-k

-n

-v

INDEX_FILE

moseq2-viz-copy-h5-metadata-to-yaml command line option

--h5-metadata-path <h5_metadata_path>

--input-dir <input_dir>

-i

moseq2-viz-get-best-model command line option

--ext <ext>

--fps <fps>

--plot-all

CP_PATH

MODEL_DIR

OUTPUT_FILE

moseq2-viz-make-crowd-movies command line option

--cmap <cmap>

--count <count>

--dur-clip <dur_clip>

--frame-path <frame_path>

--gaussfilter-space <gaussfilter_space>

--legacy-jitter-fix <legacy_jitter_fix>

--max-examples <max_examples>

--max-height <max_height>

--max-syllable <max_syllable>

--medfilter-space <medfilter_space>

--min-height <min_height>

--output-dir <output_dir>

--progress-bar

--raw-size <raw_size>

--scale <scale>

--separate-by <separate_by>

--session-names <session_names>

--sort <sort>

--specific-syllable <specific_syllable>

--threads <threads>

-m

-o

-p

-s

-t

INDEX_FILE

MODEL_PATH

moseq2-viz-plot-group-position-heatmaps command line option

--output-file <output_file>

INDEX_FILE

moseq2-viz-plot-scalar-summary command line option

--colors <colors>

--output-file <output_file>

-c

INDEX_FILE

moseq2-viz-plot-stats command line option

--colors <colors>

--count <count>

--ctrl-group <ctrl_group>

--exp-group <exp_group>

--figsize <figsize>

--fmt <fmt>

--group <group>

--max-syllable <max_syllable>

--ordering <ordering>

--output-file <output_file>

--sort <sort>

--stat <stat>

-c

-f

-g

-o

INDEX_FILE

MODEL_FIT

moseq2-viz-plot-transition-graph command line option

--arrows

N

O

P

R

S

T

W

Python Module Index

m