Psifr

Release v0.5.2

Neal Morton

Feb 22, 2022

CONTENTS

1 Installation **3**

2 User guide **5**
2.1 Importing data 5
2.2 Scoring data 6
2.3 Recall performance 8
2.4 Recall order 9
2.5 Comparing conditions 13

3 Tutorials **17**

4 API reference **19**
4.1 Free recall analysis 19
4.2 Measures 26
4.3 Transitions 28
4.4 Outputs 32

5 Development **35**
5.1 Transitions 35

Index **37**

In free recall, participants study a list of items and then name all of the items they can remember in any order they choose. Many sophisticated analyses have been developed to analyze data from free recall experiments, but these analyses are often complicated and difficult to implement.

Psifr leverages the Pandas data analysis package to make precise and flexible analysis of free recall data faster and easier.

See the code repository for version release notes.

CHAPTER

ONE

INSTALLATION

You can install the latest stable version of Psifr using pip:

```
pip install psifr
```

You can also install the development version directly from the code repository on GitHub:

```
pip install git+git://github.com/mortonne/psifr
```

CHAPTER

TWO

USER GUIDE

2.1 Importing data

In Psifr, free recall data are imported in the form of a “long” format table. Each row corresponds to one *study* or *recall* event. Study events include any time an item was presented to the participant. Recall events correspond to any recall attempt; this includes *repeats* of items there were already recalled and *intrusions* of items that were not present in the study list.

This type of information is well represented in a CSV spreadsheet, though any file format supported by pandas may be used for input. To import from a CSV, use pandas. For example:

```
import pandas as pd
data = pd.read_csv("my_data.csv")
```

2.1.1 Trial information

The basic information that must be included for each event is the following:

subject Some code (numeric or string) indicating individual participants. Must be unique for a given experiment. For example, `sub-101`.

list Numeric code indicating individual lists. Must be unique within subject.

trial_type String indicating whether each event is a `study` event or a `recall` event.

position Integer indicating position within a given phase of the list. For `study` events, this corresponds to *input position* (also referred to as *serial position*). For `recall` events, this corresponds to *output position*.

item Individual thing being recalled, such as a word. May be specified with text (e.g., `pumpkin`, `Jack Nicholson`) or a numeric code (`682`, `121`). Either way, the text or number must be unique to that item. Text is easier to read and does not require any additional information for interpretation and is therefore preferred if available.

2.1.2 Example

Table 1: Sample data

subject	list	trial_type	position	item
1	1	study	1	absence
1	1	study	2	hollow
1	1	study	3	pupil
1	1	recall	1	pupil
1	1	recall	2	absence

2.1.3 Additional information

Additional fields may be included in the data to indicate other aspects of the experiment, such as presentation time, stimulus category, experimental session, distraction length, etc. All of these fields can then be used for analysis in Psifr.

2.2 Scoring data

After *importing free recall data*, we have a DataFrame with a row for each study event and a row for each recall event. Next, we need to score the data by matching study events with recall events.

2.2.1 Scoring list recall

First, let's create a simple sample dataset with two lists:

```
In [1]: import pandas as pd

In [2]: data = pd.DataFrame({
   ...:     'subject': [
   ...:         1, 1, 1, 1, 1, 1,
   ...:         1, 1, 1, 1, 1, 1,
   ...:     ],
   ...:     'list': [
   ...:         1, 1, 1, 1, 1, 1,
   ...:         2, 2, 2, 2, 2, 2,
   ...:     ],
   ...:     'trial_type': [
   ...:         'study', 'study', 'study', 'recall', 'recall', 'recall',
   ...:         'study', 'study', 'study', 'recall', 'recall', 'recall',
   ...:     ],
   ...:     'position': [
   ...:         1, 2, 3, 1, 2, 3,
   ...:         1, 2, 3, 1, 2, 3,
   ...:     ],
   ...:     'item': [
   ...:         'absence', 'hollow', 'pupil', 'pupil', 'absence', 'empty',
   ...:         'fountain', 'piano', 'pillow', 'pillow', 'fountain', 'pillow',
   ...:     ],
   ...: })
   ...:

In [3]: data
Out[3]:
    subject  list trial_type  position      item
0         1     1      study         1   absence
1         1     1      study         2    hollow
2         1     1      study         3     pupil
3         1     1     recall         1     pupil
4         1     1     recall         2   absence
5         1     1     recall         3     empty
6         1     2      study         1  fountain
7         1     2      study         2     piano
```

(continues on next page)

(continued from previous page)

```
8          1     2      study         3    pillow
9          1     2     recall         1    pillow
10         1     2     recall         2  fountain
11         1     2     recall         3    pillow
```

Next, we'll merge together the study and recall events by matching up corresponding events:

```
In [4]: from psifr import fr

In [5]: merged = fr.merge_free_recall(data)

In [6]: merged
Out[6]: 
   subject  list      item  input  output  study  recall  repeat  intrusion
0        1     1   absence    1.0     2.0   True    True       0      False
1        1     1    hollow    2.0     NaN   True   False       0      False
2        1     1     pupil    3.0     1.0   True    True       0      False
3        1     1     empty    NaN     3.0  False    True       0       True
4        1     2  fountain    1.0     2.0   True    True       0      False
5        1     2     piano    2.0     NaN   True   False       0      False
6        1     2    pillow    3.0     1.0   True    True       0      False
7        1     2    pillow    3.0     3.0  False    True       1      False
```

For each item, there is one row for each unique combination of input and output position. For example, if an item is presented once in the list, but is recalled multiple times, there is one row for each of the recall attempts. Repeated recalls are indicated by the *repeat* column, which is greater than zero for recalls of an item after the first. Unique study events are indicated by the *study* column; this excludes intrusions and repeated recalls.

Items that were not recalled have the *recall* column set to *False*. Because they were not recalled, they have no defined output position, so *output* is set to *NaN*. Finally, intrusions have an output position but no input position because they did not appear in the list. There is an *intrusion* field for convenience to label these recall attempts.

`merge_free_recall()` can also handle additional attributes beyond the standard ones, such as codes indicating stimulus category or list condition. See *Working with custom columns* for details.

2.2.2 Filtering and sorting

Now that we have a merged *DataFrame*, we can use *pandas* methods to quickly get different views of the data. For some analyses, we may want to organize in terms of the study list by removing repeats and intrusions. Because our data are in a *DataFrame*, we can use the *DataFrame.query* method:

```
In [7]: merged.query('study')
Out[7]: 
   subject  list      item  input  output  study  recall  repeat  intrusion
0        1     1   absence    1.0     2.0   True    True       0      False
1        1     1    hollow    2.0     NaN   True   False       0      False
2        1     1     pupil    3.0     1.0   True    True       0      False
4        1     2  fountain    1.0     2.0   True    True       0      False
5        1     2     piano    2.0     NaN   True   False       0      False
6        1     2    pillow    3.0     1.0   True    True       0      False
```

Alternatively, we may also want to get just the recall events, sorted by output position instead of input position:

```
In [8]: merged.query('recall').sort_values(['list', 'output'])
Out[8]:
   subject  list      item  input  output  study  recall  repeat  intrusion
2        1     1     pupil    3.0     1.0   True    True       0      False
0        1     1   absence    1.0     2.0   True    True       0      False
3        1     1     empty    NaN     3.0  False    True       0       True
6        1     2    pillow    3.0     1.0   True    True       0      False
4        1     2  fountain    1.0     2.0   True    True       0      False
7        1     2    pillow    3.0     3.0  False    True       1      False
```

Note that we first sort by list, then output position, to keep the lists together.

2.3 Recall performance

First, load some sample data and create a merged DataFrame:

```
In [1]: from psifr import fr

In [2]: df = fr.sample_data('Morton2013')

In [3]: data = fr.merge_free_recall(df)
```

2.3.1 Raster plot

Raster plots can give you a quick overview of a whole dataset. We'll look at all of the first subject's recalls. This will plot every individual recall, colored by the serial position of the recalled item in the list. Items near the end of the list are shown in yellow, and items near the beginning of the list are shown in purple. Intrusions of items not on the list are shown in red.

```
In [4]: subj = fr.filter_data(data, 1)

In [5]: g = fr.plot_raster(subj).add_legend()
```

2.3.2 Serial position curve

We can calculate average recall for each serial position using *spc()* and plot using *plot_spc()*.

```
In [6]: recall = fr.spc(data)

In [7]: g = fr.plot_spc(recall)
```

Using the same plotting function, we can plot the curve for each individual subject:

```
In [8]: g = fr.plot_spc(recall, col='subject', col_wrap=5)
```

2.3.3 Probability of Nth recall

We can also split up recalls, to test for example how likely participants were to initiate recall with the last item on the list.

```
In [9]: prob = fr.pnr(data)

In [10]: prob
Out[10]: 
                           prob  actual  possible
subject output input                             
1       1      1       0.000000       0        48
               2       0.020833       1        48
               3       0.000000       0        48
               4       0.000000       0        48
               5       0.000000       0        48
...                         ...     ...       ...
47      24     20           NaN       0         0
               21           NaN       0         0
               22           NaN       0         0
               23           NaN       0         0
               24           NaN       0         0

[23040 rows x 3 columns]
```

This gives us the probability of recall by output position (`'output'`) and serial or input position (`'input'`). This is a lot to look at all at once, so it may be useful to plot just the first three output positions. We can plot the curves using *`plot_spc()`*, which takes an optional `hue` input to specify a variable to use to split the data into curves of different colors.

```
In [11]: pfr = prob.query('output <= 3')

In [12]: g = fr.plot_spc(pfr, hue='output').add_legend()
```

This plot shows what items tend to be recalled early in the recall sequence.

2.4 Recall order

A key advantage of free recall is that it provides information not only about what items are recalled, but also the order in which they are recalled. A number of analyses have been developed to charactize different influences on recall order, such as the temporal order in which the items were presented at study, the category of the items themselves, or the semantic similarity between pairs of items.

Each conditional response probability (CRP) analysis involves calculating the probability of some type of transition event. For the lag-CRP analysis, transition events of interest are the different lags between serial positions of items recalled adjacent to one another. Similar analyses focus not on the serial position in which items are presented, but the properties of the items themselves. A semantic-CRP analysis calculates the probability of transitions between items in different semantic relatedness bins. A special case of this analysis is when item pairs are placed into one of two bins, depending on whether they are in the same stimulus category or not. In Psifr, this is referred to as a category-CRP analysis.

2.4.1 Lag-CRP

In all CRP analyses, transition probabilities are calculated conditional on a given transition being available. For example, in a six-item list, if the items 6, 1, and 4 have been recalled, then possible items that could have been recalled next are 2, 3, or 5; therefore, possible lags at that point in the recall sequence are -2, -1, or +1. The number of actual transitions observed for each lag is divided by the number of times that lag was possible, to obtain the CRP for each lag.

First, load some sample data and create a merged DataFrame:

```
In [1]: from psifr import fr

In [2]: df = fr.sample_data('Morton2013')

In [3]: data = fr.merge_free_recall(df, study_keys=['category'])
```

Next, call *lag_crp()* to calculate conditional response probability as a function of lag.

```
In [4]: crp = fr.lag_crp(data)

In [5]: crp
Out[5]:
                   prob  actual  possible
subject lag
1       -23.0  0.020833       1        48
        -22.0  0.035714       3        84
        -21.0  0.026316       3       114
        -20.0  0.024000       3       125
        -19.0  0.014388       2       139
...                 ...     ...       ...
47       19.0  0.061224       3        49
         20.0  0.055556       2        36
         21.0  0.045455       1        22
         22.0  0.071429       1        14
         23.0  0.000000       0         6

[1880 rows x 3 columns]
```

The results show the count of times a given transition actually happened in the observed recall sequences (`actual`) and the number of times a transition could have occurred (`possible`). Finally, the `prob` column gives the estimated probability of a given transition occurring, calculated by dividing the actual count by the possible count.

Use *plot_lag_crp()* to display the results:

```
In [6]: g = fr.plot_lag_crp(crp)
```

The peaks at small lags (e.g., +1 and -1) indicate that the recall sequences show evidence of a temporal contiguity effect; that is, items presented near to one another in the list are more likely to be recalled successively than items that are distant from one another in the list.

2.4.2 Lag rank

We can summarize the tendency to group together nearby items using a lag rank analysis. For each recall, this determines the absolute lag of all remaining items available for recall and then calculates their percentile rank. Then the rank of the actual transition made is taken, scaled to vary between 0 (furthest item chosen) and 1 (nearest item chosen). Chance clustering will be 0.5; clustering above that value is evidence of a temporal contiguity effect.

```
In [7]: ranks = fr.lag_rank(data)

In [8]: ranks
Out[8]:
             rank
subject
1        0.610953
2        0.635676
3        0.612607
4        0.667090
5        0.643923
...           ...
43       0.554024
44       0.561005
45       0.598151
46       0.652748
47       0.621245

[40 rows x 1 columns]

In [9]: ranks.agg(['mean', 'sem'])
Out[9]:
          rank
mean  0.624699
sem   0.006732
```

2.4.3 Category CRP

If there are multiple categories or conditions of trials in a list, we can test whether participants tend to successively recall items from the same category. The category-CRP estimates the probability of successively recalling two items from the same category.

```
In [10]: cat_crp = fr.category_crp(data, category_key='category')

In [11]: cat_crp
Out[11]:
             prob  actual  possible
subject
1        0.801147     419       523
2        0.733456     399       544
3        0.763158     377       494
4        0.814882     449       551
5        0.877273     579       660
...           ...     ...       ...
43       0.809187     458       566
```

(continues on next page)

(continued from previous page)

```
44        0.744376     364        489
45        0.763780     388        508
46        0.763573     436        571
47        0.806907     514        637

[40 rows x 3 columns]

In [12]: cat_crp[['prob']].agg(['mean', 'sem'])
Out[12]:
          prob
mean  0.782693
sem   0.006262
```

The expected probability due to chance depends on the number of categories in the list. In this case, there are three categories, so a category CRP of 0.33 would be predicted if recalls were sampled randomly from the list.

2.4.4 Restricting analysis to specific items

Sometimes you may want to focus an analysis on a subset of recalls. For example, in order to exclude the period of high clustering commonly observed at the start of recall, lag-CRP analyses are sometimes restricted to transitions after the first three output positions.

You can restrict the recalls included in a transition analysis using the optional `item_query` argument. This is built on the Pandas query/eval system, which makes it possible to select rows of a `DataFrame` using a query string. This string can refer to any column in the data. Any items for which the expression evaluates to `True` will be included in the analysis.

For example, we can use the `item_query` argument to exclude any items recalled in the first three output positions from analysis. Note that, because non-recalled items have no output position, we need to include them explicitly using `output > 3 or not recall`.

```
In [13]: crp_op3 = fr.lag_crp(data, item_query='output > 3 or not recall')

In [14]: g = fr.plot_lag_crp(crp_op3)
```

2.4.5 Restricting analysis to specific transitions

In other cases, you may want to focus an analysis on a subset of transitions based on some criteria. For example, if a list contains items from different categories, it is a good idea to take this into account when measuring temporal clustering using a lag-CRP analysis. One approach is to separately analyze within- and across-category transitions.

Transitions can be selected for inclusion using the optional `test_key` and `test` inputs. The `test_key` indicates a column of the data to use for testing transitions; for example, here we will use the `category` column. The `test` input should be a function that takes in the test value of the previous recall and the current recall and returns True or False to indicate whether that transition should be included. Here, we will use a lambda (anonymous) function to define the test.

```
In [15]: crp_within = fr.lag_crp(data, test_key='category', test=lambda x, y: x == y)

In [16]: crp_across = fr.lag_crp(data, test_key='category', test=lambda x, y: x != y)
```

(continues on next page)

(continued from previous page)

```
In [17]: crp_combined = pd.concat([crp_within, crp_across], keys=['within', 'across'],␣
↪axis=0)

In [18]: crp_combined.index.set_names('transition', level=0, inplace=True)

In [19]: g = fr.plot_lag_crp(crp_combined, hue='transition').add_legend()
```

The `within` curve shows the lag-CRP for transitions between items of the same category, while the `across` curve shows transitions between items of different categories.

2.5 Comparing conditions

When analyzing a dataset, it's often important to compare different experimental conditions. Psifr is built on the Pandas DataFrame, which has powerful ways of splitting data and applying operations to it. This makes it possible to analyze and plot different conditions using very little code.

2.5.1 Working with custom columns

First, load some sample data and create a merged DataFrame:

```
In [1]: from psifr import fr

In [2]: df = fr.sample_data('Morton2013')

In [3]: data = fr.merge_free_recall(
   ...:     df, study_keys=['category'], list_keys=['list_type']
   ...: )
   ...:

In [4]: data.head()
Out[4]:
   subject  list      item  input  ...  repeat  intrusion  list_type  category
0        1     1     TOWEL    1.0  ...       0      False       pure       obj
1        1     1     LADLE    2.0  ...       0      False       pure       obj
2        1     1   THERMOS    3.0  ...       0      False       pure       obj
3        1     1      LEGO    4.0  ...       0      False       pure       obj
4        1     1  BACKPACK    5.0  ...       0      False       pure       obj

[5 rows x 11 columns]
```

The *merge_free_recall()* function only includes columns from the raw data if they are one of the standard columns or if they've explictly been included using `study_keys`, `recall_keys`, or `list_keys`. `list_keys` apply to all events in a list, while `study_keys` and `recall_keys` are relevant only for study and recall events, respectively.

We've included a list key here, to indicate that the `list_type` field should be included for all study and recall events in each list, even intrusions. The `category` field will be included for all study events and all valid recalls. Intrusions will have an undefined category.

2.5.2 Analysis by condition

Now we can run any analysis separately for the different conditions. We'll use the serial position curve analysis as an example.

```
In [5]: spc = data.groupby('list_type').apply(fr.spc)

In [6]: spc.head()
Out[6]:
                            recall
list_type subject input
mixed     1       1.0    0.500000
                  2.0    0.466667
                  3.0    0.600000
                  4.0    0.300000
                  5.0    0.333333
```

The spc DataFrame has separate groups with the results for each `list_type`.

Warning: When using `groupby` with order-based analyses like *`lag_crp()`*, make sure all recalls in all recall sequences for a given list have the same label. Otherwise, you will be breaking up recall sequences, which could result in an invalid analysis.

2.5.3 Plotting by condition

We can then plot a separate curve for each condition. All plotting functions take optional `hue`, `col`, `col_wrap`, and `row` inputs that can be used to divide up data when plotting. See the Seaborn documentation for details. Most inputs to `seaborn.relplot()` are supported.

For example, we can plot two curves for the different list types:

```
In [7]: g = fr.plot_spc(spc, hue='list_type').add_legend()
```

We can also plot the curves in different axes using the `col` option:

```
In [8]: g = fr.plot_spc(spc, col='list_type')
```

We can also plot all combinations of two conditions:

```
In [9]: spc_split = data.groupby(['list_type', 'category']).apply(fr.spc)

In [10]: g = fr.plot_spc(spc_split, col='list_type', row='category')
```

2.5.4 Plotting by subject

All analyses can be plotted separately by subject. A nice way to do this is using the `col` and `col_wrap` optional inputs, to make a grid of plots with 6 columns per row:

```
In [11]: g = fr.plot_spc(
   ....:     spc, hue='list_type', col='subject', col_wrap=6, height=2
   ....: ).add_legend()
   ....:
```

CHAPTER

THREE

TUTORIALS

See the psifr-notebooks project for a set of Jupyter notebooks with sample code. These examples go more in depth into the options available for each analysis and how they can be used for advanced analyses such as conditionalizing CRP analysis on specific transitions.

CHAPTER FOUR

API REFERENCE

4.1 Free recall analysis

4.1.1 Managing data

merge_free_recall(data, **kwargs)	Merge standard free recall events.
merge_lists(study, recall[, merge_keys, ...])	Merge study and recall events together for each list.
filter_data(data[, subjects, lists, ...])	Filter data to get a subset of trials.
reset_list(df)	Reset list index in a DataFrame.
split_lists(frame, phase, keys[, names, ...])	Convert free recall data from one phase to split format.

psifr.fr.merge_free_recall

`psifr.fr.`**`merge_free_recall`**(*data*, ***kwargs*)

Merge standard free recall events.

Split study and recall events and then merge them. See *merge_lists* for details.

psifr.fr.merge_lists

`psifr.fr.`**`merge_lists`**(*study*, *recall*, *merge_keys=None*, *list_keys=None*, *study_keys=None*, *recall_keys=None*, *position_key='position'*)

Merge study and recall events together for each list.

Parameters

- **study** (*pandas.DataFrame*) – Information about all study events. Should have one row for each study event.
- **recall** (*pandas.DataFrame*) – Information about all recall events. Should have one row for each recall attempt.
- **merge_keys** (*list, optional*) – Columns to use to designate events to merge. Default is [‘subject’, ‘list’, ‘item’], which will merge events related to the same item, but only within list.
- **list_keys** (*list, optional*) – Columns that apply to both study and recall events.
- **study_keys** (*list, optional*) – Columns that only apply to study events.
- **recall_keys** (*list, optional*) – Columns that only apply to recall events.

- **position_key** (*str, optional*) – Column indicating the position of each item in either the study list or the recall sequence.

Returns

merged – Merged information about study and recall events. Each row corresponds to one unique input/output pair.

The following columns will be added:

input [int] Position of each item in the input list (i.e., serial position).

output [int] Position of each item in the recall sequence.

study [bool] True for rows corresponding to a unique study event.

recall [bool] True for rows corresponding to a unique recall event.

repeat [int] Number of times this recall event has been repeated (0 for the first recall of an item).

intrusion [bool] True for recalls that do not correspond to any study event.

Return type pandas.DataFrame

psifr.fr.filter_data

psifr.fr.**filter_data**(*data*, *subjects=None*, *lists=None*, *trial_type=None*, *positions=None*, *inputs=None*, *outputs=None*)

Filter data to get a subset of trials.

psifr.fr.reset_list

psifr.fr.**reset_list**(*df*)

Reset list index in a DataFrame.

psifr.fr.split_lists

psifr.fr.**split_lists**(*frame*, *phase*, *keys*, *names=None*, *item_query=None*, *as_list=False*)

Convert free recall data from one phase to split format.

Parameters

- **frame** (*pandas.DataFrame*) – Free recall data with separate study and recall events.
- **phase** (*{'study', 'recall', 'raw'}*) – Phase of recall to split. If 'raw', all trials will be included.
- **keys** (*list of str*) – Data columns to include in the split data.
- **names** (*list of str, optional*) – Name for each column in the returned split data. Default is to use the same names as the input columns.
- **item_query** (*str, optional*) – Query string to select study trials to include. See *pandas.DataFrame.query* for allowed format.
- **as_list** (*bool, optional*) – If true, each column will be output as a list; otherwise, outputs will be numpy.ndarray.

Returns **split** – Data in split format. Each included column will be a key in the dictionary, with a list of either numpy.ndarray (default) or lists, containing the values for that column.

Return type dict of str: list

4.1.2 Recall probability

spc(df)	Serial position curve.
pnr(df[, item_query, test_key, test])	Probability of recall by serial position and output position.

psifr.fr.spc

`psifr.fr.spc`(*df*)

Serial position curve.

Parameters `df` (*pandas.DataFrame*) – Merged study and recall data. See merge_lists.

Returns

recall – Index includes:

subject [hashable] Subject identifier.

input [int] Serial position in the list.

Values are:

recall [float] Recall probability for each serial position.

Return type pandas.Series

psifr.fr.pnr

`psifr.fr.pnr`(*df*, *item_query=None*, *test_key=None*, *test=None*)

Probability of recall by serial position and output position.

Calculate probability of Nth recall, where N is each output position. Invalid recalls (repeats and intrusions) are ignored and not counted toward output position.

Parameters

- `df` (*pandas.DataFrame*) – Merged study and recall data. See merge_lists. List length is assumed to be the same for all lists within each subject. Must have fields: subject, list, input, output, study, recall. Input position must be defined such that the first serial position is 1, not 0.
- `item_query` (*str, optional*) – Query string to select items to include in the pool of possible recalls to be examined. See *pandas.DataFrame.query* for allowed format.
- `test_key` (*str, optional*) – Name of column with labels to use when testing transitions for inclusion.
- `test` (*callable, optional*) – Callable that takes in previous and current item values and returns True for transitions that should be included.

Returns **prob** – Analysis results. Has fields: subject, output, input, prob, actual, possible. The prob column for output x and input y indicates the probability of recalling input position y at output position x. The actual and possible columns give the raw tallies for how many times an event actually occurred and how many times it was possible given the recall sequence.

Return type pandas.DataFrame

4.1.3 Transition probability

lag_crp(df[, lag_key, count_unique, ...])	Lag-CRP for multiple subjects.
category_crp(df, category_key[, item_query, ...])	Conditional response probability of within-category transitions.
distance_crp(df, index_key, distances, edges)	Conditional response probability by distance bin.

psifr.fr.lag_crp

`psifr.fr.`**`lag_crp`**(*df*, *lag_key='input'*, *count_unique=False*, *item_query=None*, *test_key=None*, *test=None*)

Lag-CRP for multiple subjects.

Parameters

- **`df`** (*`pandas.DataFrame`*) – Merged study and recall data. See merge_lists. List length is assumed to be the same for all lists. Must have fields: subject, list, input, output, recalled. Input position must be defined such that the first serial position is 1, not 0.
- **`lag_key`** (*`str, optional`*) – Name of column to use when calculating lag between recalled items. Default is to calculate lag based on input position.
- **`count_unique`** (*`bool, optional`*) – If true, possible transitions of the same lag will only be incremented once per transition.
- **`item_query`** (*`str, optional`*) – Query string to select items to include in the pool of possible recalls to be examined. See *pandas.DataFrame.query* for allowed format.
- **`test_key`** (*`str, optional`*) – Name of column with labels to use when testing transitions for inclusion.
- **`test`** (*`callable, optional`*) – Callable that takes in previous and current item values and returns True for transitions that should be included.

Returns

results – Has fields:

subject [hashable] Results are separated by each subject.

lag [int] Lag of input position between two adjacent recalls.

prob [float] Probability of each lag transition.

actual [int] Total of actual made transitions at each lag.

possible [int] Total of times each lag was possible, given the prior input position and the remaining items to be recalled.

Return type pandas.DataFrame

psifr.fr.category_crp

`psifr.fr.category_crp`(*df*, *category_key*, *item_query=None*, *test_key=None*, *test=None*)

Conditional response probability of within-category transitions.

Parameters

- **df** (*pandas.DataFrame*) – Merged study and recall data. See merge_lists. List length is assumed to be the same for all lists within each subject. Must have fields: subject, list, input, output, recalled.
- **category_key** (*str*) – Name of column with category labels.
- **item_query** (*str, optional*) – Query string to select items to include in the pool of possible recalls to be examined. See *pandas.DataFrame.query* for allowed format.
- **test_key** (*str, optional*) – Name of column with labels to use when testing transitions for inclusion.
- **test** (*callable, optional*) – Callable that takes in previous and current item values and returns True for transitions that should be included.

Returns

results – Has fields:

subject [hashable] Results are separated by each subject.

prob [float] Probability of each lag transition.

actual [int] Total of actual made transitions at each lag.

possible [int] Total of times each lag was possible, given the prior input position and the remaining items to be recalled.

Return type pandas.DataFrame

psifr.fr.distance_crp

`psifr.fr.distance_crp`(*df*, *index_key*, *distances*, *edges*, *centers=None*, *count_unique=False*, *item_query=None*, *test_key=None*, *test=None*)

Conditional response probability by distance bin.

Parameters

- **df** (*pandas.DataFrame*) – Merged free recall data.
- **index_key** (*str*) – Name of column containing the index of each item in the *distances* matrix.
- **distances** (*numpy.array*) – Items x items matrix of pairwise distances or similarities.
- **edges** (*array-like*) – Edges of bins to apply to the distances.
- **centers** (*array-like, optional*) – Centers to label each bin with. If not specified, the center point between edges will be used.
- **count_unique** (*bool, optional*) – If true, possible transitions to a given distance bin will only count once for a given transition.
- **item_query** (*str, optional*) – Query string to select items to include in the pool of possible recalls to be examined. See *pandas.DataFrame.query* for allowed format.

- **test_key** (*str, optional*) – Name of column with labels to use when testing transitions for inclusion.
- **test** (*callable, optional*) – Callable that takes in previous and current item values and returns True for transitions that should be included.

Returns

crp – Has fields:

subject [hashable] Results are separated by each subject.

bin [int] Distance bin.

prob [float] Probability of each distance bin.

actual [int] Total of actual transitions for each distance bin.

possible [int] Total of times each distance bin was possible, given the prior input position and the remaining items to be recalled.

Return type pandas.DataFrame

4.1.4 Transition rank

lag_rank(df[, item_query, test_key, test])	Calculate rank of the absolute lags in free recall lists.
distance_rank(df, index_key, distances[, ...])	Calculate rank of transition distances in free recall lists.

psifr.fr.lag_rank

`psifr.fr.lag_rank`(*df*, *item_query=None*, *test_key=None*, *test=None*)

Calculate rank of the absolute lags in free recall lists.

Parameters

- **df** (*pandas.DataFrame*) – Merged study and recall data. See merge_lists. List length is assumed to be the same for all lists within each subject. Must have fields: subject, list, input, output, recalled. Input position must be defined such that the first serial position is 1, not 0.
- **item_query** (*str, optional*) – Query string to select items to include in the pool of possible recalls to be examined. See *pandas.DataFrame.query* for allowed format.
- **test_key** (*str, optional*) – Name of column with labels to use when testing transitions for inclusion.
- **test** (*callable, optional*) – Callable that takes in previous and current item values and returns True for transitions that should be included.

Returns **stat** – Has fields ‘subject’ and ‘rank’.

Return type pandas.DataFrame

psifr.fr.distance_rank

`psifr.fr.`**`distance_rank`**(*df*, *index_key*, *distances*, *item_query=None*, *test_key=None*, *test=None*)

Calculate rank of transition distances in free recall lists.

Parameters

- **`df`** (*`pandas.DataFrame`*) – Merged study and recall data. See merge_lists. List length is assumed to be the same for all lists within each subject. Must have fields: subject, list, input, output, recalled. Input position must be defined such that the first serial position is 1, not 0.
- **`index_key`** (*`str`*) – Name of column containing the index of each item in the *distances* matrix.
- **`distances`** (*`numpy.array`*) – Items x items matrix of pairwise distances or similarities.
- **`item_query`** (*`str, optional`*) – Query string to select items to include in the pool of possible recalls to be examined. See *pandas.DataFrame.query* for allowed format.
- **`test_key`** (*`str, optional`*) – Name of column with labels to use when testing transitions for inclusion.
- **`test`** (*`callable, optional`*) – Callable that takes in previous and current item values and returns True for transitions that should be included.

Returns **stat** – Has fields 'subject' and 'rank'.

Return type pandas.DataFrame

4.1.5 Plotting

`plot_raster`(df[, hue, palette, marker, ...])	Plot recalls in a raster plot.
`plot_spc`(recall, **facet_kws)	Plot a serial position curve.
`plot_lag_crp`(recall[, max_lag])	Plot conditional response probability by lag.
`plot_distance_crp`(crp[, min_samples])	Plot response probability by distance bin.
`plot_swarm_error`(data[, x, y, swarm_color, ...])	Plot points as a swarm plus mean with error bars.

psifr.fr.plot_raster

`psifr.fr.`**`plot_raster`**(*df*, *hue='input'*, *palette=None*, *marker='s'*, *intrusion_color=None*, *orientation='horizontal'*, *length=6*, *aspect=None*, *legend='auto'*, ***facet_kws*)

Plot recalls in a raster plot.

psifr.fr.plot_spc

`psifr.fr.`**`plot_spc`**(*recall*, ***facet_kws*)

Plot a serial position curve.

Additional arguments are passed to seaborn.relplot.

Parameters **`recall`** (*`pandas.DataFrame`*) – Results from calling *spc*.

psifr.fr.plot_lag_crp

`psifr.fr.`**`plot_lag_crp`**`(`*recall*, *max_lag=5*, ***facet_kws*`)`

Plot conditional response probability by lag.

Additional arguments are passed to seaborn.FacetGrid.

Parameters

- **`recall`** (*`pandas.DataFrame`*) – Results from calling *lag_crp*.
- **`max_lag`** (*`int`*) – Maximum absolute lag to plot.

psifr.fr.plot_distance_crp

`psifr.fr.`**`plot_distance_crp`**`(`*crp*, *min_samples=None*, ***facet_kws*`)`

Plot response probability by distance bin.

Parameters

- **`crp`** (*`pandas.DataFrame`*) – Results from *fr.distance_crp*.
- **`min_samples`** (*`int`*) – Minimum number of samples a bin must have per subject to include in the plot.
- **`**facet_kws`** – Additional inputs to pass to *seaborn.relplot*.

psifr.fr.plot_swarm_error

`psifr.fr.`**`plot_swarm_error`**`(`*data*, *x=None*, *y=None*, *swarm_color=None*, *swarm_size=5*, *point_color='k'*, ***facet_kws*`)`

Plot points as a swarm plus mean with error bars.

4.2 Measures

4.2.1 Transition measure base class

`TransitionMeasure`(items_key, label_key[, ...])	Measure of free recall dataset with multiple subjects.
`TransitionMeasure.split_lists`(data, phase[, ...])	Get relevant fields and split by list.
`TransitionMeasure.analyze`(data)	Analyze a free recall dataset with multiple subjects.
`TransitionMeasure.analyze_subject`(subject, ...)	Analyze a single subject.

psifr.measures.TransitionMeasure

`class psifr.measures.`**`TransitionMeasure`**`(`*items_key*, *label_key*, *item_query=None*, *test_key=None*, *test=None*`)`

Measure of free recall dataset with multiple subjects.

Parameters

- **`items_key`** (*`str`*) – Data column with item identifiers.
- **`label_key`** (*`str`*) – Data column with trial labels to use for the measure.

- **item_query** (*str*) – Query string to indicate trials to include in the measure.
- **test_key** (*str*) – Data column with labels to use when testing for trial inclusion.
- **test** (*callable*) – Test of trial inclusion. Takes the previous and current test values and return True if the transition should be included.

keys

List of columns to use for the measure.

Type dict of {str: str}

item_query

Query string to indicate trials to include in the measure.

Type str

test

Test of trial inclusion.

Type callable

__init__(*items_key*, *label_key*, *item_query=None*, *test_key=None*, *test=None*)

Methods

__init__(items_key, label_key[, item_query, ...])	
analyze(data)	Analyze a free recall dataset with multiple subjects.
analyze_subject(subject, pool_lists, ...)	Analyze a single subject.
split_lists(data, phase[, item_query])	Get relevant fields and split by list.

psifr.measures.TransitionMeasure.split_lists

TransitionMeasure.**split_lists**(*data*, *phase*, *item_query=None*)

Get relevant fields and split by list.

Parameters

- **data** (*pandas.DataFrame*) – Raw free recall data.
- **phase** (*str*) – Phase to split ('study' or 'recall').
- **item_query** (*str, optional*) – Query string to determine included trials.

psifr.measures.TransitionMeasure.analyze

TransitionMeasure.**analyze**(*data*)

Analyze a free recall dataset with multiple subjects.

Parameters **data** (*pandas.DataFrame*) – Raw (not merged) free recall data.

Returns **stat** – Statistics calculated for each subject.

Return type pandas.DataFrame

psifr.measures.TransitionMeasure.analyze_subject

`abstract TransitionMeasure.`**`analyze_subject`**(*subject*, *pool_lists*, *recall_lists*)

Analyze a single subject.

Parameters

- **subject** (*int or str*) – Identifier of the subject to analyze.
- **pool_lists** (*dict of lists of numpy.ndarray*) – Information about the item pool for each list, with keys for items, label, and test arrays.
- **recall_lists** (*dict of lists of numpy.ndarray*) – Information about the recall sequence for each list, with keys for items, label, and test arrays.

Returns Results of the analysis for one subject. Should include a 'subject' column in the index.

Return type pandas.DataFrame

4.2.2 Transition measures

`TransitionOutputs`(list_length[, item_query, ...])	Measure recall probability by input and output position.
`TransitionLag`(list_length[, lag_key, ...])	Measure conditional response probability by lag.
`TransitionLagRank`([item_query, test_key, test])	Measure lag rank of transitions.
`TransitionCategory`(category_key[, ...])	Measure conditional response probability by category transition.
`TransitionDistance`(index_key, distances, edges)	Measure conditional response probability by distance.
`TransitionDistanceRank`(index_key, distances)	Measure transition rank by distance.

4.3 Transitions

4.3.1 Counting transitions

`count_lags`(list_length, pool_items, recall_items)	Count actual and possible serial position lags.
`count_category`(pool_items, recall_items, ...)	Count within-category transitions.
`count_distance`(distances, edges, pool_items, ...)	Count transitions within distance bins.

psifr.transitions.count_lags

`psifr.transitions.`**`count_lags`**(*list_length*, *pool_items*, *recall_items*, *pool_label=None*, *recall_label=None*, *pool_test=None*, *recall_test=None*, *test=None*, *count_unique=False*)

Count actual and possible serial position lags.

Parameters

- **list_length** (*int*) – Number of items in each list.
- **pool_items** (*list*) – List of the serial positions available for recall in each list. Must match the serial position codes used in *recall_items*.
- **recall_items** (*list*) – List indicating the serial position of each recall in output order (NaN for intrusions).

- **pool_label** (*list, optional*) – List of the positions to use for calculating lag. Default is to use *pool_items*.
- **recall_label** (*list, optional*) – List of position labels in recall order. Default is to use *recall_items*.
- **pool_test** (*list, optional*) – List of some test value for each item in the pool.
- **recall_test** (*list, optional*) – List of some test value for each recall attempt by output position.
- **test** (*callable*) – Callable that evaluates each transition between items n and n+1. Must take test values for items n and n+1 and return True if a given transition should be included.
- **count_unique** (*bool, optional*) – If true, only unique values will be counted toward the possible transitions. If multiple items are avilable for recall for a given transition and a given bin, that bin will only be incremented once. If false, all possible transitions will add to the count.

psifr.transitions.count_category

`psifr.transitions.`**`count_category`**(*pool_items*, *recall_items*, *pool_category*, *recall_category*, *pool_test=None*, *recall_test=None*, *test=None*)

Count within-category transitions.

psifr.transitions.count_distance

`psifr.transitions.`**`count_distance`**(*distances*, *edges*, *pool_items*, *recall_items*, *pool_index*, *recall_index*, *pool_test=None*, *recall_test=None*, *test=None*, *count_unique=False*)

Count transitions within distance bins.

Parameters

- **distances** (*numpy.array*) – Items x items matrix of pairwise distances or similarities.
- **edges** (*array-like*) – Edges of bins to apply to distances.
- **pool_items** (*list of list*) – Unique item codes for each item in the pool available for recall.
- **recall_items** (*list of list*) – Unique item codes of recalled items.
- **pool_index** (*list of list*) – Index of each item in the distances matrix.
- **recall_index** (*list of list*) – Index of each recalled item.
- **pool_test** (*list of list, optional*) – Test value for each item in the pool.
- **recall_test** (*list of list, optional*) – Test value for each recalled item.
- **test** (*callable*) – Called as test(prev, curr) or test(prev, poss) to screen actual and possible transitions, respectively.
- **count_unique** (*bool, optional*) – If true, only unique values will be counted toward the possible transitions. If multiple items are avilable for recall for a given transition and a given bin, that bin will only be incremented once. If false, all possible transitions will add to the count.

Returns

- **actual** (*pandas.Series*) – Count of actual transitions made for each bin.

- **possible** (*pandas.Series*) – Count of possible transitions for each bin.

4.3.2 Ranking transitions

percentile_rank(actual, possible)	Get percentile rank of a score compared to possible scores.
rank_lags(pool_items, recall_items[, ...])	Calculate rank of absolute lag for free recall lists.
rank_distance(distances, pool_items, ...[, ...])	Calculate percentile rank of transition distances.

psifr.transitions.percentile_rank

`psifr.transitions.`**`percentile_rank`**(*actual*, *possible*)

Get percentile rank of a score compared to possible scores.

psifr.transitions.rank_lags

`psifr.transitions.`**`rank_lags`**(*pool_items*, *recall_items*, *pool_label=None*, *recall_label=None*, *pool_test=None*, *recall_test=None*, *test=None*)

Calculate rank of absolute lag for free recall lists.

Parameters

- **`pool_items`** (`list`) – List of the serial positions available for recall in each list. Must match the serial position codes used in *recall_items*.
- **`recall_items`** (`list`) – List indicating the serial position of each recall in output order (NaN for intrusions).
- **`pool_label`** (`list, optional`) – List of the positions to use for calculating lag. Default is to use *pool_items*.
- **`recall_label`** (`list, optional`) – List of position labels in recall order. Default is to use *recall_items*.
- **`pool_test`** (`list, optional`) – List of some test value for each item in the pool.
- **`recall_test`** (`list, optional`) – List of some test value for each recall attempt by output position.
- **`test`** (`callable`) – Callable that evaluates each transition between items n and n+1. Must take test values for items n and n+1 and return True if a given transition should be included.

Returns **rank** – Absolute lag percentile rank for each included transition. The rank is 0 if the lag was the most distant of the available transitions, and 1 if the lag was the closest. Ties are assigned to the average percentile rank.

Return type list

psifr.transitions.rank_distance

`psifr.transitions.`**`rank_distance`**(*distances*, *pool_items*, *recall_items*, *pool_index*, *recall_index*, *pool_test=None*, *recall_test=None*, *test=None*)

Calculate percentile rank of transition distances.

Parameters

- **`distances`** (*`numpy.array`*) – Items x items matrix of pairwise distances or similarities.
- **`pool_items`** (*`list of list`*) – Unique item codes for each item in the pool available for recall.
- **`recall_items`** (*`list of list`*) – Unique item codes of recalled items.
- **`pool_index`** (*`list of list`*) – Index of each item in the distances matrix.
- **`recall_index`** (*`list of list`*) – Index of each recalled item.
- **`pool_test`** (*`list of list, optional`*) – Test value for each item in the pool.
- **`recall_test`** (*`list of list, optional`*) – Test value for each recalled item.
- **`test`** (*`callable`*) – Called as test(prev, curr) or test(prev, poss) to screen actual and possible transitions, respectively.

Returns **rank** – Distance percentile rank for each included transition. The rank is 0 if the distance was the largest of the available transitions, and 1 if the distance was the smallest. Ties are assigned to the average percentile rank.

Return type list

4.3.3 Iterating over transitions

`transitions_masker`(pool_items, recall_items, ...)	Iterate over transitions with masking.

psifr.transitions.transitions_masker

`psifr.transitions.`**`transitions_masker`**(*pool_items*, *recall_items*, *pool_output*, *recall_output*, *pool_test=None*, *recall_test=None*, *test=None*)

Iterate over transitions with masking.

Transitions are between a “previous” item and a “current” item. Non-included transitions will be skipped. A transition is yielded only if it matches the following conditions:

(1) Each item involved in the transition is in the pool. Items are removed from the pool after they appear as the previous item.

(2) Optionally, an additional check is run based on test values associated with the items in the transition. For example, this could be used to only include transitions where the category of the previous and current items is the same.

The masker will yield “output” values, which may be distinct from the item identifiers used to determine item repeats.

Parameters

- **`pool_items`** (*`list`*) – Items available for recall. Order does not matter. May contain repeated values. Item identifiers must be unique within pool.

- **`recall_items`** (*`list`*) – Recalled items in output position order.
- **`pool_output`** (*`list`*) – Output values for pool items. Must be the same order as pool.
- **`recall_output`** (*`list`*) – Output values in output position order.
- **`pool_test`** (*`list, optional`*) – Test values for items available for recall. Must be the same order as pool.
- **`recall_test`** (*`list, optional`*) – Test values for items in output position order.
- **`test`** (*`callable, optional`*) – Used to test whether individual transitions should be included, based on test values.

  test(prev, curr) - test for included transition

  test(prev, poss) - test for included possible transition

Yields

- **prev** (*object*) – Output value for the "from" item on this transition.
- **curr** (*object*) – Output value for the "to" item.
- **poss** (*numpy.array*) – Output values for all possible valid "to" items.

4.4 Outputs

4.4.1 Counting recalls by serial position and output position

`count_outputs`(list_length, pool_items, ...)	Count actual and possible recalls for each output position.

psifr.outputs.count_outputs

`psifr.outputs.`**`count_outputs`**(*list_length*, *pool_items*, *recall_items*, *pool_label*, *recall_label*, *pool_test=None*, *recall_test=None*, *test=None*, *count_unique=False*)

Count actual and possible recalls for each output position.

Parameters

- **`list_length`** (*`int`*) – Number of items in each list.
- **`pool_items`** (*`list`*) – List of the serial positions available for recall in each list. Must match the serial position codes used in *recall_items*.
- **`recall_items`** (*`list`*) – List indicating the serial position of each recall in output order (NaN for intrusions).
- **`pool_label`** (*`list, optional`*) – List of the positions to use for calculating lag. Default is to use *pool_items*.
- **`recall_label`** (*`list, optional`*) – List of position labels in recall order. Default is to use *recall_items*.
- **`pool_test`** (*`list, optional`*) – List of some test value for each item in the pool.
- **`recall_test`** (*`list, optional`*) – List of some test value for each recall attempt by output position.

- **`test`** (*`callable`*) – Callable that evaluates each transition between items n and n+1. Must take test values for items n and n+1 and return True if a given transition should be included.
- **`count_unique`** (*`bool`*) – If true, possible recalls with the same label will only be counted once.

4.4.2 Iterating over output positions

`outputs_masker`(pool_items, recall_items, ...)	Iterate over valid outputs.

psifr.outputs.outputs_masker

`psifr.outputs.`**`outputs_masker`**(*pool_items*, *recall_items*, *pool_output*, *recall_output*, *pool_test=None*, *recall_test=None*, *test=None*)

Iterate over valid outputs.

Parameters

- **`pool_items`** (*`list`*) – Items available for recall. Order does not matter. May contain repeated values. Item identifiers must be unique within pool.
- **`recall_items`** (*`list`*) – Recalled items in output position order.
- **`pool_output`** (*`list`*) – Output values for pool items. Must be the same order as pool.
- **`recall_output`** (*`list`*) – Output values in output position order.
- **`pool_test`** (*`list, optional`*) – Test values for items available for recall. Must be the same order as pool.
- **`recall_test`** (*`list, optional`*) – Test values for items in output position order.
- **`test`** (*`callable, optional`*) – Used to test whether output recalls and possible recalls should be included, based on their test values.

Yields

- **curr** (*object*) – Output value for the item at this valid output position.
- **poss** (*numpy.array*) – Output values for all possible items that could be recalled at this output position.
- **output** (*int*) – Current output position.

CHAPTER FIVE

DEVELOPMENT

5.1 Transitions

Psifr has a core set of tools for analyzing transitions in free recall data. These tools focus on measuring what transitions actually occurred, and which transitions were possible given the order in which participants recalled items.

5.1.1 Actual and possible transitions

Calculating a conditional response probability involves two parts: the frequency at which a given event actually occurred in the data and frequency at which a given event could have occurred. The frequency of possible events is calculated conditional on the recalls that have been made leading up to each transition. For example, a transition between item i and item j is not considered "possible" in a CRP analysis if item i was never recalled. The transition is also not considered "possible" if, when item i is recalled, item j has already been recalled previously.

Repeated recall events are typically excluded from the counts of both actual and possible transition events. That is, the transition event frequencies are conditional on the transition not being either to or from a repeated item.

Calculating a CRP measure involves tallying how many transitions of a given type were made during a free recall test. For example, one common measure is the serial position lag between items. For a list of length N, possible lags are in the range $[-N + 1, N - 1]$. Because repeats are excluded, a lag of zero is never possible. The count of actual and possible transitions for each lag is calculated first, and then the CRP for each lag is calculated as the actual count divided by the possible count.

5.1.2 The transitions masker

The *psifr.transitions.transitions_masker()* is a generator that makes it simple to iterate over transitions while "masking" out events such as intrusions of items not on the list and repeats of items that have already been recalled.

On each step of the iterator, the previous, current, and possible items are yielded. The *previous* item is the item being transitioned from. The *current* item is the item being transitioned to. The *possible* items includes an array of all items that were valid to be recalled next, given the recall sequence up to that point (not including the current item).

```
In [1]: from psifr.transitions import transitions_masker

In [2]: pool = [1, 2, 3, 4, 5, 6]

In [3]: recs = [6, 2, 3, 6, 1, 4]

In [4]: masker = transitions_masker(pool_items=pool, recall_items=recs,
```

(continues on next page)

(continued from previous page)

```
   ...:                                 pool_output=pool, recall_output=recs)
   ...:

In [5]: for prev, curr, poss in masker:
   ...:     print(prev, curr, poss)
   ...:
6 2 [1 2 3 4 5]
2 3 [1 3 4 5]
1 4 [4 5]
```

Only valid transitions are yielded, so the code for a specific analysis only needs to calculate the transition measure of interest and count the number of actual and possible transitions in each bin of interest.

Four inputs are required:

pool_items List of identifiers for all items available for recall. Identifiers can be anything that is unique to each item in the list (e.g., serial position, a string representation of the item, an index in the stimulus pool).

recall_items List of identifiers for the sequence of recalls, in order. Valid recalls must match an item in *pool_items*. Other items are considered intrusions.

pool_output Output codes for each item in the pool. This should be whatever you need to calculate your transition measure.

recall_output Output codes for each recall in the sequence of recalls.

By using different values for these four inputs and defining different transition measures, a wide range of analyses can be implemented.

INDEX

Symbols

`__init__()` (*psifr.measures.TransitionMeasure method*), 27

A

`analyze()` (*psifr.measures.TransitionMeasure method*), 27

`analyze_subject()` (*psifr.measures.TransitionMeasure method*), 28

C

`category_crp()` (*in module psifr.fr*), 23

`count_category()` (*in module psifr.transitions*), 29

`count_distance()` (*in module psifr.transitions*), 29

`count_lags()` (*in module psifr.transitions*), 28

`count_outputs()` (*in module psifr.outputs*), 32

D

`distance_crp()` (*in module psifr.fr*), 23

`distance_rank()` (*in module psifr.fr*), 25

F

`filter_data()` (*in module psifr.fr*), 20

I

`item_query` (*psifr.measures.TransitionMeasure attribute*), 27

K

`keys` (*psifr.measures.TransitionMeasure attribute*), 27

L

`lag_crp()` (*in module psifr.fr*), 22

`lag_rank()` (*in module psifr.fr*), 24

M

`merge_free_recall()` (*in module psifr.fr*), 19

`merge_lists()` (*in module psifr.fr*), 19

O

`outputs_masker()` (*in module psifr.outputs*), 33

P

`percentile_rank()` (*in module psifr.transitions*), 30

`plot_distance_crp()` (*in module psifr.fr*), 26

`plot_lag_crp()` (*in module psifr.fr*), 26

`plot_raster()` (*in module psifr.fr*), 25

`plot_spc()` (*in module psifr.fr*), 25

`plot_swarm_error()` (*in module psifr.fr*), 26

`pnr()` (*in module psifr.fr*), 21

R

`rank_distance()` (*in module psifr.transitions*), 31

`rank_lags()` (*in module psifr.transitions*), 30

`reset_list()` (*in module psifr.fr*), 20

S

`spc()` (*in module psifr.fr*), 21

`split_lists()` (*in module psifr.fr*), 20

`split_lists()` (*psifr.measures.TransitionMeasure method*), 27

T

`test` (*psifr.measures.TransitionMeasure attribute*), 27

`TransitionMeasure` (*class in psifr.measures*), 26

`transitions_masker()` (*in module psifr.transitions*), 31