scikit-posthocs Documentation

Release 0.7.0

Maksim Terpilowski

May 26, 2026

DOCUMENTATION

scikit-posthocs is a Python package which provides post hoc tests for pairwise multiple comparisons that are usually performed in statistical data analysis to assess the differences between group levels if a statistically significant result of ANOVA test has been obtained.

scikit-posthocs is tightly integrated with Pandas DataFrames and NumPy arrays to ensure fast computations and convenient data import and storage.

This package will be useful for statisticians, data analysts, and researchers who use Python in their work.

CHAPTER

ONE

INTRODUCTION

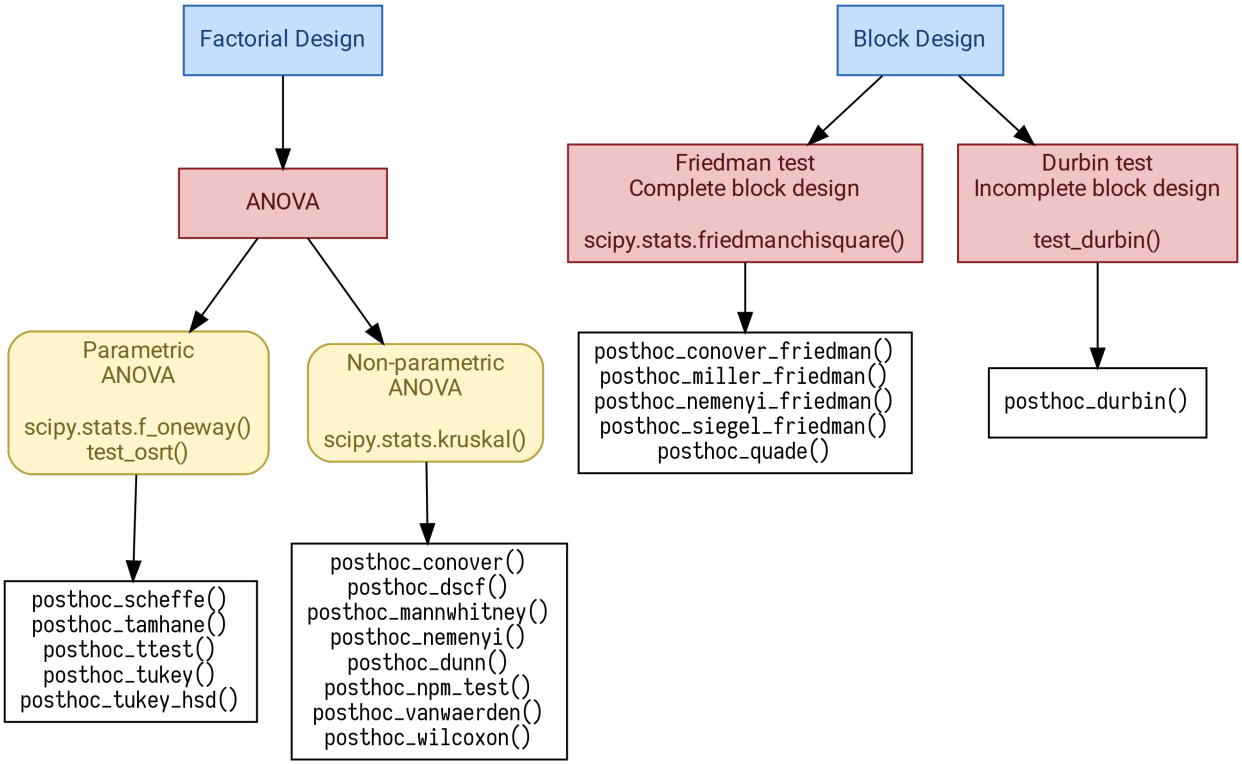
1.1 Background

Python statistical ecosystem is comprised of multiple packages. However, it still has numerous gaps and is surpassed by R packages and capabilities.

SciPy (version 1.2.0) offers *Student*, *Wilcoxon*, and *Mann-Whitney* tests which are not adapted to multiple pairwise comparisons. Statsmodels (version 0.9.0) features *TukeyHSD* test which needs some extra actions to be fluently integrated into a data analysis pipeline. *Statsmodels* also has good helper methods: `allpairtest` (adapts an external function such as `scipy.stats.ttest_ind` to multiple pairwise comparisons) and `multipletests` (adjusts *p* values to minimize type I and II errors). PMCMRplus is a very good R package which has no rivals in Python as it offers more than 40 various tests (including post hoc tests) for factorial and block design data. PMCMRplus was an inspiration and a reference for *scikit-posthocs*.

scikit-posthocs attempts to improve Python statistical capabilities by offering a lot of parametric and nonparametric post hoc tests along with outliers detection and basic plotting methods.

1.2 Features



- *Omnibox* tests:
  - Durbin test (for balanced incomplete block design).
- *Parametric* pairwise multiple comparisons tests:
  - Scheffe test.
  - Student T test.
  - Tamhane T2 test.
  - TukeyHSD test.
- *Non-parametric* tests for factorial design:
  - Conover test.
  - Dunn test.
  - Dwass, Steel, Critchlow, and Fligner test.
  - Mann-Whitney test.
  - Nashimoto and Wright (NPM) test.
  - Nemenyi test.
  - van Waerden test.
  - Wilcoxon test.
- *Non-parametric* tests for block design:
  - Conover test.

  - Durbin and Conover test.
  - Miller test.
  - Nemenyi test.
  - Quade test.
  - Siegel test.
- Other tests:
  - Anderson-Darling test.
  - Mack-Wolfe test.
  - Hayter (OSRT) test.
- Outliers detection tests:
  - Simple test based on interquartile range (IQR).
  - Grubbs test.
  - Tietjen-Moore test.
  - Generalized Extreme Studentized Deviate test (ESD test).
- Plotting functionality:
  - Significance plots.
  - Critical difference diagrams.

All post hoc tests are capable of p value adjustments for multiple pairwise comparisons.

CHAPTER

TWO

INSTALLATION

The latest version can be installed from PyPi using `pip`:

```
pip install scikit-posthocs
```

Or from conda-forge repository using `conda`:

```
conda install -c conda-forge scikit-posthocs
```

You can also use `pip` to install the development version from GitHub:

```
pip install git+https://github.com/maximtrp/scikit-posthocs.git
```

2.1 Dependencies

Package is compatible with both major versions of Python and has the following dependencies:

- NumPy
- SciPy
- Statsmodels
- Pandas
- Seaborn
- Matplotlib

2.2 Bugs

Please report any bugs using issues tracker on GitHub.

CHAPTER

THREE

TUTORIAL

3.1 Parametric ANOVA with post hoc tests

Here is a simple example of the one-way analysis of variance (ANOVA) with post hoc tests used to compare *sepal width* means of three groups (three iris species) in *iris* dataset.

To begin, we will import the dataset using statsmodels `get_rdataset()` method.

```
>>> import statsmodels.api as sa
>>> import statsmodels.formula.api as sfa
>>> import scikit_posthocs as sp
>>> df = sa.datasets.get_rdataset('iris').data
>>> df.head()
   Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

Now, we will build a model and run ANOVA using statsmodels `ols()` and `anova_lm()` methods. Columns `Species` and `Sepal.Width` contain independent (predictor) and dependent (response) variable values, correspondingly.

```
>>> lm = sfa.ols('Sepal.Width ~ C(Species)', data=df).fit()
>>> anova = sa.stats.anova_lm(lm)
>>> print(anova)
                df     sum_sq   mean_sq         F        PR(>F)
C(Species)     2.0  11.344933  5.672467  49.16004  4.492017e-17
Residual     147.0  16.962000  0.115388       NaN           NaN
```

The results tell us that there is a significant difference between groups means (p = 4.49e-17), but does not tell us the exact group pairs which are different in means. To obtain pairwise group differences, we will carry out a posteriori (post hoc) analysis using `scikits-posthocs` package. Student T test applied pairwisely gives us the following p values:

```
>>> sp.posthoc_ttest(df, val_col='Sepal.Width', group_col='Species', p_adjust='holm')
                setosa    versicolor     virginica
setosa     -1.000000e+00  5.535780e-15  8.492711e-09
versicolor  5.535780e-15 -1.000000e+00  1.819100e-03
virginica   8.492711e-09  1.819100e-03 -1.000000e+00
```

Remember to use a FWER controlling procedure, such as Holm procedure, when making multiple comparisons. As seen from this table, significant differences in group means are obtained for all group pairs.

3.2 Non-parametric ANOVA with post hoc tests

If normality and other assumptions are violated, one can use a non-parametric Kruskal-Wallis H test (one-way non-parametric ANOVA) to test if samples came from the same distribution.

Let's use the same dataset just to demonstrate the procedure. Kruskal-Wallis test is implemented in SciPy package. `scipy.stats.kruskal` method accepts array-like structures, but not DataFrames.

```
>>> import scipy.stats as ss
>>> import statsmodels.api as sa
>>> import scikit_posthocs as sp
>>> df = sa.datasets.get_rdataset('iris').data
>>> data = [df.loc[ids, 'Sepal.Width'].values for ids in df.groupby('Species').groups.
↪values()]
```

`data` is a list of 1D arrays containing *sepal width* values, one array per each species. Now we can run Kruskal-Wallis analysis of variance.

```
>>> H, p = ss.kruskal(*data)
>>> p
1.5692820940316782e-14
```

P value tells us we may reject the null hypothesis that the population medians of all of the groups are equal. To learn what groups (species) differ in their medians we need to run post hoc tests. `scikit-posthocs` provides a lot of non-parametric tests mentioned above. Let's choose Conover's test.

```
>>> sp.posthoc_conover(df, val_col='Sepal.Width', group_col='Species', p_adjust = 'holm')
                setosa    versicolor     virginica
setosa     -1.000000e+00  2.278515e-18  1.293888e-10
versicolor  2.278515e-18 -1.000000e+00  1.881294e-03
virginica   1.293888e-10  1.881294e-03 -1.000000e+00
```

Pairwise comparisons show that we may reject the null hypothesis ($p < 0.01$) for each pair of species and conclude that all groups (species) differ in their sepal widths.

3.3 Compact Letter Display

Post hoc test results can be condensed into a compact letter display (CLD), where groups sharing at least one letter are not significantly different from each other. This is useful for annotating plots or summarizing results in tables.

```
>>> x = [[1, 2, 1, 3, 1, 4], [12, 3, 11, 9, 3, 8, 1],
...      [10, 22, 12, 9, 8, 3], [14, 12, 16, 17, 5, 9]]
>>> pc = sp.posthoc_dunn(x, p_adjust='holm')
>>> sp.compact_letter_display(pc, alpha=0.05)
1    a
2    ab
3     b
4     b
Name: letters, dtype: object
```

Groups 1 and 2 share letter `a` (not significantly different), while groups 2, 3 and 4 share letter `b` (not significantly different). Group 1 is significantly different from groups 3 and 4.

3.4 Block design

In block design case, we have a primary factor (e.g. treatment) and a blocking factor (e.g. age or gender). A blocking factor is also called a *nuisance* factor, and it is usually a source of variability that needs to be accounted for.

An example scenario is testing the effect of four fertilizers on crop yield in four cornfields. We can represent the results with a matrix in which rows correspond to the blocking factor (field) and columns correspond to the primary factor (yield).

The following dataset is artificial and created just for demonstration of the procedure:

```
>>> data = np.array([[ 8.82, 11.8 , 10.37, 12.08],
                     [ 8.92,  9.58, 10.59, 11.89],
                     [ 8.27, 11.46, 10.24, 11.6 ],
                     [ 8.83, 13.25,  8.33, 11.51]])
```

First, we need to perform an omnibus test — Friedman rank sum test. It is implemented in `scipy.stats` subpackage:

```
>>> import scipy.stats as ss
>>> ss.friedmanchisquare(*data.T)
FriedmanchisquareResult(statistic=8.700000000000003, pvalue=0.03355726870553798)
```

We can reject the null hypothesis that our treatments have the same distribution, because p value is less than 0.05. A number of post hoc tests are available in `scikit-posthocs` package for unreplicated block design data. In the following example, Nemenyi's test is used:

```
>>> import scikit_posthocs as sp
>>> sp.posthoc_nemenyi_friedman(data)
          0         1         2         3
0 -1.000000  0.220908  0.823993  0.031375
1  0.220908 -1.000000  0.670273  0.823993
2  0.823993  0.670273 -1.000000  0.220908
3  0.031375  0.823993  0.220908 -1.000000
```

This function returns a DataFrame with p values obtained in pairwise comparisons between all treatments. One can also pass a DataFrame and specify the names of columns containing dependent variable values, blocking and primary factor values. The following code creates a DataFrame with the same data:

```
>>> data = pd.DataFrame.from_dict({'blocks': {0: 0, 1: 1, 2: 2, 3: 3, 4: 0, 5: 1, 6:
2, 7: 3, 8: 0, 9: 1, 10: 2, 11: 3, 12: 0, 13: 1, 14: 2, 15: 3}, 'groups': {0:
0, 1: 0, 2: 0, 3: 0, 4: 1, 5: 1, 6: 1, 7: 1, 8: 2, 9: 2, 10: 2, 11: 2, 12: 3,
13: 3, 14: 3, 15: 3}, 'y': {0: 8.82, 1: 8.92, 2: 8.27, 3: 8.83, 4: 11.8, 5:
9.58, 6: 11.46, 7: 13.25, 8: 10.37, 9: 10.59, 10: 10.24, 11: 8.33, 12: 12.08,
13: 11.89, 14: 11.6, 15: 11.51}})
>>> data
    blocks  groups      y
0        0       0   8.82
1        1       0   8.92
2        2       0   8.27
3        3       0   8.83
4        0       1  11.80
5        1       1   9.58
6        2       1  11.46
7        3       1  13.25
8        0       2  10.37
```

(continues on next page)

(continued from previous page)

```
9        1       2  10.59
10       2       2  10.24
11       3       2   8.33
12       0       3  12.08
13       1       3  11.89
14       2       3  11.60
15       3       3  11.51
```

This is a *melted* and ready-to-use DataFrame. Do not forget to pass `melted` argument:

```
>>> sp.posthoc_nemenyi_friedman(data, y_col='y', block_col='blocks', group_col='groups',␣
↪melted=True)
          0         1         2         3
0 -1.000000  0.220908  0.823993  0.031375
1  0.220908 -1.000000  0.670273  0.823993
2  0.823993  0.670273 -1.000000  0.220908
3  0.031375  0.823993  0.220908 -1.000000
```

3.5 Data types

Internally, `scikit-posthocs` uses NumPy ndarrays and pandas DataFrames to store and process data. Python lists, NumPy ndarrays, and pandas DataFrames are supported as *input* data types. Below are usage examples of various input data structures.

3.5.1 Lists and arrays

```
>>> x = [[1,2,1,3,1,4], [12,3,11,9,3,8,1], [10,22,12,9,8,3]]
>>> # or
>>> x = np.array([[1,2,1,3,1,4], [12,3,11,9,3,8,1], [10,22,12,9,8,3]])
>>> sp.posthoc_conover(x, p_adjust='holm')
          1         2         3
1 -1.000000  0.057606  0.007888
2  0.057606 -1.000000  0.215761
3  0.007888  0.215761 -1.000000
```

You can check how it is processed with a hidden function `__convert_to_df()`:

```
>>> sp.__convert_to_df(x)
(    vals  groups
 0      1       1
 1      2       1
 2      1       1
 3      3       1
 4      1       1
 5      4       1
 6     12       2
 7      3       2
 8     11       2
 9      9       2
 10     3       2
 11     8       2
```

(continues on next page)

(continued from previous page)

```
 12     1       2
 13    10       3
 14    22       3
 15    12       3
 16     9       3
 17     8       3
 18     3       3, 'vals', 'groups')
```

It returns a tuple of a DataFrame representation and names of the columns containing dependent (`vals`) and independent (`groups`) variable values.

Block design matrix passed as a NumPy ndarray is processed with a hidden `__convert_to_block_df()` function:

```
>>> data = np.array([[ 8.82, 11.8 , 10.37, 12.08],
                     [ 8.92,  9.58, 10.59, 11.89],
                     [ 8.27, 11.46, 10.24, 11.6 ],
                     [ 8.83, 13.25,  8.33, 11.51]])
>>> sp.__convert_to_block_df(data)
(    blocks groups      y
 0        0      0   8.82
 1        1      0   8.92
 2        2      0   8.27
 3        3      0   8.83
 4        0      1  11.80
 5        1      1   9.58
 6        2      1  11.46
 7        3      1  13.25
 8        0      2  10.37
 9        1      2  10.59
 10       2      2  10.24
 11       3      2   8.33
 12       0      3  12.08
 13       1      3  11.89
 14       2      3  11.60
 15       3      3  11.51, 'y', 'groups', 'blocks')
```

3.5.2 DataFrames

If you are using DataFrames, you need to pass column names containing variable values to a post hoc function:

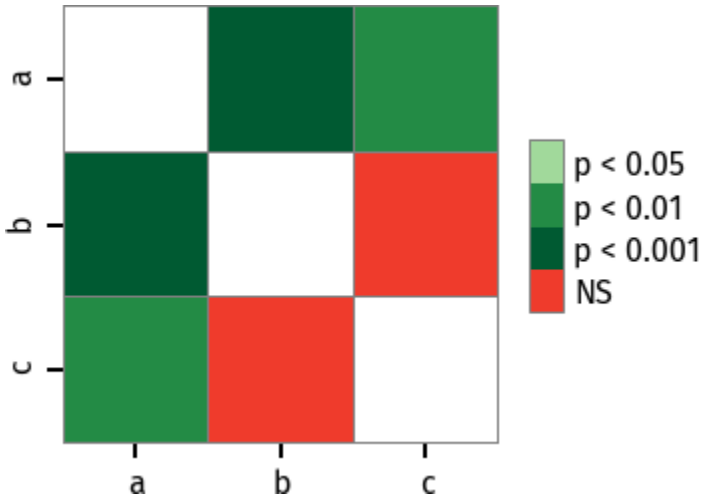
```
>>> import statsmodels.api as sa
>>> import scikit_posthocs as sp
>>> df = sa.datasets.get_rdataset('iris').data
>>> sp.posthoc_conover(df, val_col='Sepal.Width', group_col='Species', p_adjust = 'holm')
```

`val_col` and `group_col` arguments specify the names of the columns containing dependent (response) and independent (grouping) variable values.

3.6 Significance plots

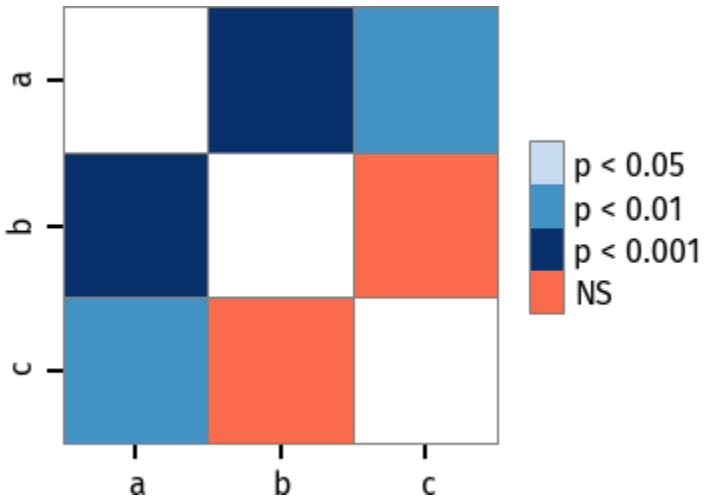
P values can be plotted using a heatmap:

```
pc = sp.posthoc_conover(x, val_col='values', group_col='groups')
heatmap_args = {'linewidths': 0.25, 'linecolor': '0.5', 'clip_on': False, 'square': True,
↪ 'cbar_ax_bbox': [0.80, 0.35, 0.04, 0.3]}
sp.sign_plot(pc, **heatmap_args)
```



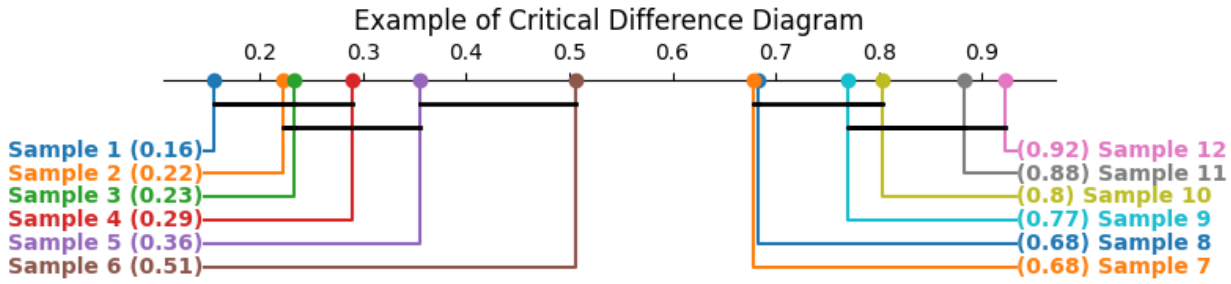
Custom colormap applied to a plot:

```
pc = sp.posthoc_conover(x, val_col='values', group_col='groups')
# Format: diagonal, non-significant, p<0.001, p<0.01, p<0.05
cmap = ['1', '#fb6a4a',  '#08306b',  '#4292c6', '#c6dbef']
heatmap_args = {'cmap': cmap, 'linewidths': 0.25, 'linecolor': '0.5', 'clip_on': False,
↪'square': True, 'cbar_ax_bbox': [0.80, 0.35, 0.04, 0.3]}
sp.sign_plot(pc, **heatmap_args)
```



3.7 Critical difference diagrams

Critical difference diagrams are another interesting way of visualizing post hoc test statistics. Firstly, in a block design scenario, the values within each block are ranked, and the average rank across all blocks for each treatment is plotted along the x axis. A crossbar is then drawn over each group of treatments that do not show a statistically significant difference among themselves.



As an example, suppose we have a set of 8 treatments with 30 measurements (blocks) each, as simulated below. It could, for instance, represent scores for eight machine learning models in a 30-fold cross-validation setting.

```
>>> rng = np.random.default_rng(1)
>>> dict_data = {
    'model1': rng.normal(loc=0.2, scale=0.1, size=30),
    'model2': rng.normal(loc=0.2, scale=0.1, size=30),
    'model3': rng.normal(loc=0.4, scale=0.1, size=30),
    'model4': rng.normal(loc=0.5, scale=0.1, size=30),
    'model5': rng.normal(loc=0.7, scale=0.1, size=30),
    'model6': rng.normal(loc=0.7, scale=0.1, size=30),
    'model7': rng.normal(loc=0.8, scale=0.1, size=30),
    'model8': rng.normal(loc=0.9, scale=0.1, size=30),
}
>>> data = (
  pd.DataFrame(dict_data)
  .rename_axis('cv_fold')
  .melt(
      var_name='estimator',
      value_name='score',
      ignore_index=False,
  )
  .reset_index()
)
>>> data
      cv_fold  estimator     score
0           0     model1  0.234558
1           1     model1  0.282162
2           2     model1  0.233044
3           3     model1  0.069684
4           4     model1  0.290536
..        ...        ...       ...
235        25     model8  0.925956
236        26     model8  0.758762
237        27     model8  0.977032
238        28     model8  0.829890
239        29     model8  0.787381

[240 rows x 3 columns]
```

The average (percentile) ranks could be calculated as follows:

```
>>> avg_rank = data.groupby('cv_fold').score.rank(pct=True).groupby(data.estimator).
```

(continues on next page)

(continued from previous page)

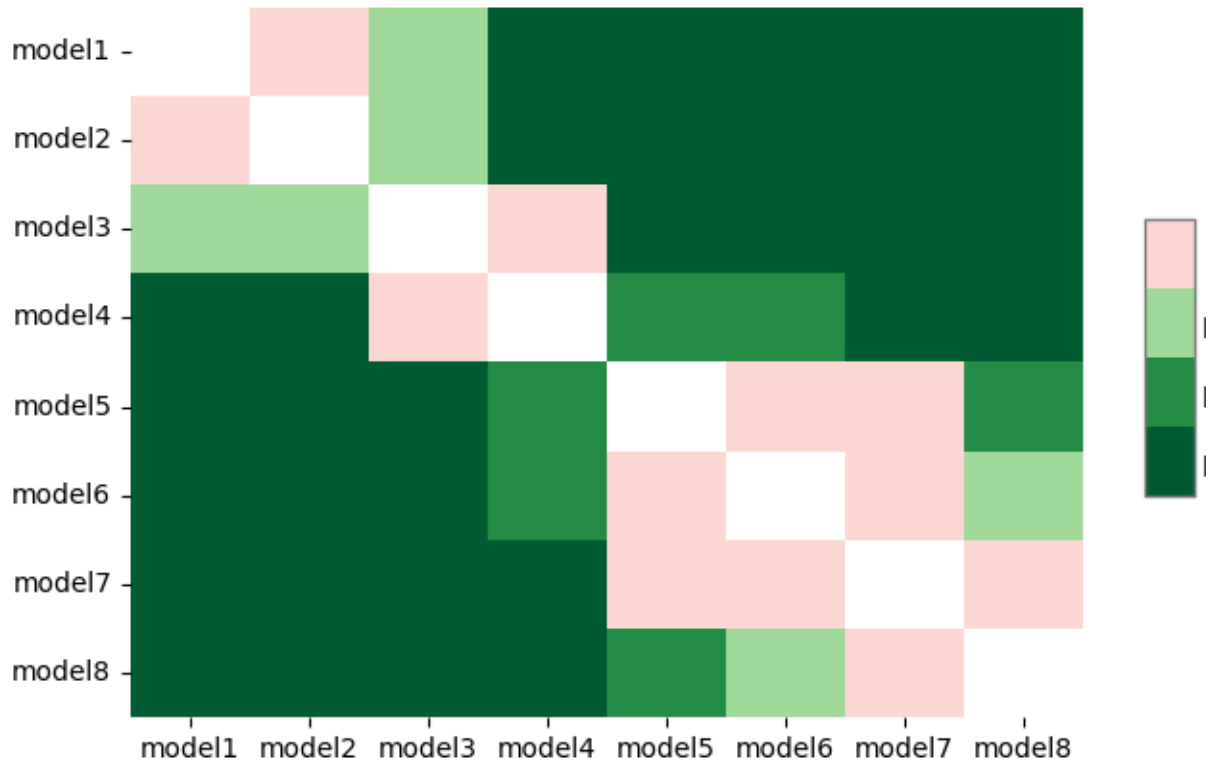
```
↪mean()
>>> avg_rank

estimator
model1    0.208333
model2    0.191667
model3    0.366667
model4    0.495833
model5    0.708333
model6    0.737500
model7    0.850000
model8    0.941667
Name: score, dtype: float64
```

Again, the omnibus test result shows we can confidently reject the null hypothesis that all models come from the same distribution and proceed to the post hoc analysis.

```
>>> import scipy.stats as ss
>>> ss.friedmanchisquare(*dict_data.values())
FriedmanchisquareResult(statistic=186.90000000000001, pvalue=6.787361102785178e-37)
```

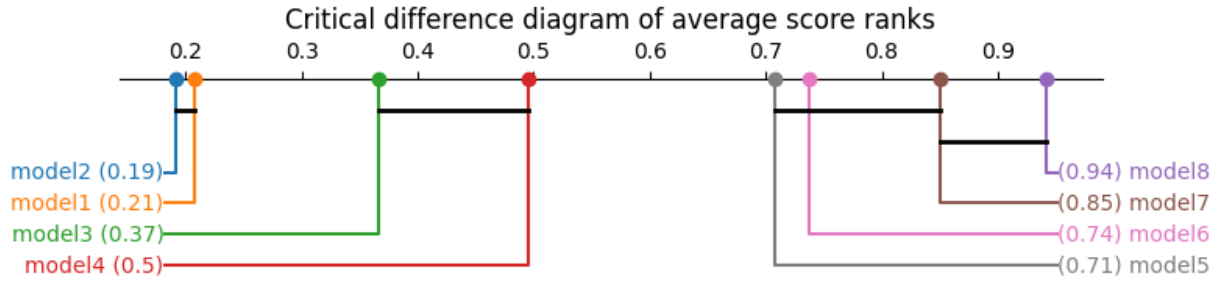
The results of a post hoc Conover test are collected:

```
>>> test_results = sp.posthoc_conover_friedman(
>>>     data,
>>>     melted=True,
>>>     block_col='cv_fold',
>>>     group_col='estimator',
>>>     y_col='score',
>>> )
>>> sp.sign_plot(test_results)
```



Finally, the average ranks and post hoc significance results can be passed to the `critical_difference_diagram()` function to plot the diagram:

```
>>> plt.figure(figsize=(10, 2), dpi=100)
>>> plt.title('Critical difference diagram of average score ranks')
>>> sp.critical_difference_diagram(avg_rank, test_results)
```



The diagram shows that model 8 is significantly better ranked than all models but model 7, that models 1 and 2 are worse than the others, and that 3 and 4 are also worse ranked than models 5, 6 and 7. Other comparisons, however, do not have sufficient statistical evidence to support them.

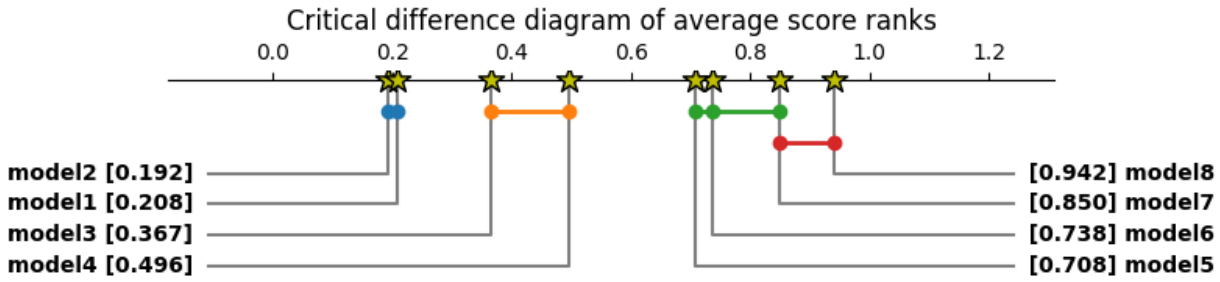
Several style customization options are available:

```
>>> plt.figure(figsize=(10, 2), dpi=100)
```

(continues on next page)

(continued from previous page)

```
>>> plt.title('Critical difference diagram of average score ranks')
>>> sp.critical_difference_diagram(
>>>     ranks=avg_rank,
>>>     sig_matrix=test_results,
>>>     label_fmt_left='{label} [{rank:.3f}]  ',
>>>     label_fmt_right='  [{rank:.3f}] {label}',
>>>     text_h_margin=0.3,
>>>     label_props={'color': 'black', 'fontweight': 'bold'},
>>>     crossbar_props={'color': None, 'marker': 'o'},
>>>     marker_props={'marker': '*', 's': 150, 'color': 'y', 'edgecolor': 'k'},
>>>     elbow_props={'color': 'gray'},
>>> )
```



CHAPTER

FOUR

GLOBAL TESTS API REFERENCE

global_f_test(p_vals[, stat])	Fisher's combination test for global null hypothesis.
global_simes_test(p_vals)	Global Simes test of the intersection null hypothesis.

4.1 scikit_posthocs.global_f_test

scikit_posthocs.**global_f_test**(*p_vals: List | ndarray*, *stat: bool = False*) → float | Tuple[float, float]

Fisher's combination test for global null hypothesis.

Computes the combined p value using chi-squared distribution and T statistic: -2 * sum(log(x))[1].

Parameters

- **p_vals** (*Union[List, ndarray]*) – An array or a list of p values.
- **stat** (*bool*) – Defines if statistic should be returned.

Returns

- **p_value** (*float*) – Global p value.
- **t_stat** (*float*) – Statistic.

References

Examples

```
>>> x = [0.04, 0.03, 0.98, 0.01, 0.43, 0.99, 1.0, 0.002]
>>> sp.global_f_test(x)
```

4.2 scikit_posthocs.global_simes_test

scikit_posthocs.**global_simes_test**(*p_vals: List | ndarray*) → float

Global Simes test of the intersection null hypothesis.

Computes the combined p value as min(np(i)/i), where p(1), ..., p(n) are the ordered p values[1].

Parameters

p_vals (*Union[List, ndarray]*) – An array of p values.

[1] Fisher RA. Statistical methods for research workers, London: Oliver and Boyd, 1932.

[1] Simes, R. J. (1986). An improved Bonferroni procedure for multiple tests of significance. Biometrika, 73(3):751-754.

Returns
p_value – Global p value.

Return type
float

References

Examples

```
>>> arr = [0.04, 0.03, 0.98, 0.01, 0.43, 0.99, 1.0, 0.002]
>>> sp.global_simes_test(arr)
```

CHAPTER

FIVE

OMNIBUS API REFERENCE

test_mackwolfe(data[, val_col, group_col, ...])	Mack-Wolfe Test for Umbrella Alternatives.
test_osrt(data[, val_col, group_col, sort])	Hayter's one-sided studentised range test (OSRT)
test_durbin(data[, y_col, group_col, ...])	Durbin's test whether k groups (or treatments) in a two-way balanced incomplete block design (BIBD) have identical effects.

5.1 scikit_posthocs.test_mackwolfe

scikit_posthocs.**test_mackwolfe**(*data: _Buffer | _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | complex | bytes | str | _NestedSequence[complex | bytes | str] | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *p: int | None = None*, *n_perm: int = 100*, *sort: bool = False*) → tuple[float, float]

Mack-Wolfe Test for Umbrella Alternatives.

In dose-finding studies one may assume an increasing treatment effect with increasing dose level. However, the test subject may actually succumb to toxic effects at high doses, which leads to decresing treatment effects[1],[2].

The scope of the Mack-Wolfe Test is to test for umbrella alternatives for either a known or unknown point P (i.e. dose-level), where the peak (umbrella point) is present.

Parameters

- **data** (*Union[List, numpy.ndarray, DataFrame]*) – An array, any object exposing the array interface or a pandas DataFrame with data values.
- **val_col** (*str = None*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if `a` is a pandas DataFrame object.
- **group_col** (*str = None*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if `a` is a pandas DataFrame object.
- **p** (*int = None*) – The a priori known peak as an ordinal number of the treatment group including the zero dose level, i.e. p = {0, …, k-1}. Defaults to None.
- **n_perm** (*int = 100*) – Permutations number.
- **sort** (*bool = False*) – If `True`, sort data by block and group columns.

[1] Chen, I.Y. (1991) Notes on the Mack-Wolfe and Chen-Wolfe Tests for Umbrella Alternatives. Biom. J., 33, 281-290.
[2] Mack, G.A., Wolfe, D. A. (1981) K-sample rank tests for umbrella alternatives. J. Amer. Statist. Assoc., 76, 175-181.

Returns
P value and statistic.

Return type
tuple[float, float]

References

Examples

```
>>> x = [[22, 23, 35], [60, 59, 54], [98, 78, 50], [60, 82, 59], [22, 44, 33], [23,␣
↪21, 25]]
>>> sp.posthoc_mackwolfe(x)
```

5.2 scikit_posthocs.test_osrt

scikit_posthocs.**test_osrt**(*data: List | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *sort: bool = False*) → tuple[float, float, int]

Hayter's one-sided studentised range test (OSRT)

Tests a hypothesis against an ordered alternative for normal data with equal variances[1].

Parameters

- **data** (*Union[List, numpy.ndarray, DataFrame]*) – An array, any object exposing the array interface or a pandas DataFrame with data values.
- **val_col** (*str = None*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if `a` is a pandas DataFrame object.
- **group_col** (*str = None*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **sort** (*bool = False*) – If True, sort data by block and group columns.

Returns
P value, statistic, and number of degrees of freedom.

Return type
tuple[float, float, int]

Notes

P values are computed from the Tukey distribution.

References

Examples

```
>>> import scikit_posthocs as sp
>>> import pandas as pd
>>> x = pd.DataFrame({"a": [1,2,3,5,1], "b": [12,31,54,62,12], "c": [10,12,6,74,11]}
↪)
```

(continues on next page)

[1] Hayter, A.J.(1990) A One-Sided Studentised Range Test for Testing Against a Simple Ordered Alternative, Journal of the American Statistical Association, 85, 778-785.

(continued from previous page)

```
>>> x = x.melt(var_name='groups', value_name='values')
>>> sp.test_osrt(x, val_col='values', group_col='groups')
```

5.3 scikit_posthocs.test_durbin

scikit_posthocs.**test_durbin**(*data: List | ndarray | DataFrame*, *y_col: str | int | None = None*, *group_col: str | int | None = None*, *block_col: str | int | None = None*, *block_id_col: str | int | None = None*, *melted: bool = False*, *sort: bool = True*) → tuple[float, float, int]

Durbin's test whether k groups (or treatments) in a two-way balanced incomplete block design (BIBD) have identical effects. See references for additional information[1],[2].

Parameters

- **data** (*Union[List, np.ndarray, DataFrame]*) – An array, any object exposing the array interface or a pandas DataFrame with data values.

  If `melted` argument is set to False (default), `a` is a typical matrix of block design, i.e. rows are blocks, and columns are groups. In this case, you do not need to specify col arguments.

  If `a` is an array and `melted` is set to True, y_col, block_col and group_col must specify the indices of columns containing elements of correspondary type.

  If `a` is a Pandas DataFrame and `melted` is set to True, y_col, block_col and group_col must specify columns names (string).

- **y_col** (*Union[str, int] = None*) – Must be specified if `a` is a melted pandas DataFrame object. Name of the column that contains y data.

- **group_col** (*Union[str, int] = None*) – Must be specified if `a` is a melted pandas DataFrame object. Name of the column that contains group names.

- **block_col** (*Union[str, int] = None*) – Must be specified if `a` is a melted pandas DataFrame object. Name of the column that contains block names.

- **block_id_col** (*Union[str, int] = None*) – Must be specified if `a` is a melted pandas DataFrame object. Name of the column that contains identifiers of block names. In most cases, this is the same as *block_col* except for those cases when you have multiple instances of the same blocks.

- **melted** (*bool = False*) – Specifies if data are given as melted columns "y", "blocks", and "groups".

- **sort** (*bool = False*) – If True, sort data by block and group columns.

Returns

P value, statistic, and number of degrees of freedom.

Return type

tuple[float, float, int]

[1] N. A. Heckert, J. J. Filliben. (2003) NIST Handbook 148: Dataplot Reference Manual, Volume 2: Let Subcommands and Library Functions. National Institute of Standards and Technology Handbook Series, June 2003.

[2] W. J. Conover (1999), Practical nonparametric Statistics, 3rd. edition, Wiley.

References

Examples

```
>>> x = np.array([[31,27,24],[31,28,31],[45,29,46],[21,18,48],[42,36,46],[32,17,
↪40]])
>>> sp.test_durbin(x)
```

CHAPTER

SIX

OUTLIERS API REFERENCE

outliers_iqr(x[, ret, coef])	Simple detection of potential outliers based on interquartile range (IQR).
outliers_gesd(x[, outliers, hypo, report, alpha])	The generalized (Extreme Studentized Deviate) ESD test is used to detect one or more outliers in a univariate data set that follows an approximately normal distribution [1]_.
outliers_grubbs(x[, hypo, alpha])	Grubbs' Test for Outliers [1]_.
outliers_tietjen(x, k[, hypo, alpha])	Tietjen-Moore test [1]_ to detect multiple outliers in a univariate data set that follows an approximately normal distribution.

6.1 scikit_posthocs.outliers_iqr

scikit_posthocs.**outliers_iqr**(*x: List | ndarray*, *ret: str = 'filtered'*, *coef: float = 1.5*) → ndarray

Simple detection of potential outliers based on interquartile range (IQR). Data that lie within the lower and upper limits are considered non-outliers. The lower limit is the number that lies 1.5 IQRs below (coefficient may be changed with an argument, see Parameters) the first quartile; the upper limit is the number that lies 1.5 IQRs above the third quartile.

Parameters

- **x** (*Union[List, np.ndarray]*) – An array, any object exposing the array interface, containing p values.
- **ret** (*str = 'filtered'*) – Specifies object to be returned. Available options are:
  - `filtered`: return a filtered array (default)
  - `outliers`: return outliers
  - `indices`: return indices of non-outliers
  - `outliers_indices`: return indices of outliers
- **coef** (*float = 1.5*) – Coefficient by which IQR is multiplied.

Returns

One of the following objects:

- Filtered array (default) if `ret` is set to `filtered`.
- Array with indices of elements lying within the specified limits if `ret` is set to `indices`.
- Array with outliers if `ret` is set to `outliers`.

- Array with indices of outlier elements if `ret` is set to `outliers_indices`.

Return type
numpy.ndarray

Examples

```
>>> x = np.array([4, 5, 6, 10, 12, 4, 3, 1, 2, 3, 23, 5, 3])
>>> outliers_iqr(x, ret = 'outliers')
array([12, 23])
```

6.2 scikit_posthocs.outliers_gesd

scikit_posthocs.**outliers_gesd**(*x: _Buffer | _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | complex | bytes | str | _NestedSequence[complex | bytes | str]*, *outliers: int = 5*, *hypo: bool = False*, *report: bool = False*, *alpha: float = 0.05*) → ndarray

The generalized (Extreme Studentized Deviate) ESD test is used to detect one or more outliers in a univariate data set that follows an approximately normal distribution[1].

Parameters

- **x** (*Union[List, np.ndarray]*) – An array, any object exposing the array interface, containing data to test for outliers.
- **outliers** (*int = 5*) – Number of potential outliers to test for. Test is two-tailed, i.e. maximum and minimum values are checked for potential outliers.
- **hypo** (*bool = False*) – Specifies whether to return a bool value of a hypothesis test result. Returns True when we can reject the null hypothesis. Otherwise, False. Available options are: 1) True - return a hypothesis test result. 2) False - return a filtered array without an outlier (default).
- **report** (*bool = False*) – Specifies whether to print a summary table of the test.
- **alpha** (*float = 0.05*) – Significance level for a hypothesis test.

Returns
If hypo is True, returns a boolean array where True indicates an outlier. If hypo is False, returns the filtered array with outliers removed.

Return type
np.ndarray

Notes

Examples

```
>>> data = np.array([-0.25, 0.68, 0.94, 1.15, 1.2, 1.26, 1.26, 1.34,
    1.38, 1.43, 1.49, 1.49, 1.55, 1.56, 1.58, 1.65, 1.69, 1.7, 1.76,
    1.77, 1.81, 1.91, 1.94, 1.96, 1.99, 2.06, 2.09, 2.1, 2.14, 2.15,
    2.23, 2.24, 2.26, 2.35, 2.37, 2.4, 2.47, 2.54, 2.62, 2.64, 2.9,
    2.92, 2.92, 2.93, 3.21, 3.26, 3.3, 3.59, 3.68, 4.3, 4.64, 5.34,
    5.42, 6.01])
>>> outliers_gesd(data, 5)
```

(continues on next page)

[1] Rosner, Bernard (May 1983), Percentage Points for a Generalized ESD Many-Outlier Procedure,Technometrics, 25(2), pp. 165-172.

(continued from previous page)

```
array([-0.25,  0.68,  0.94,  1.15,  1.2 ,  1.26,  1.26,  1.34,  1.38,
        1.43,  1.49,  1.49,  1.55,  1.56,  1.58,  1.65,  1.69,  1.7 ,
        1.76,  1.77,  1.81,  1.91,  1.94,  1.96,  1.99,  2.06,  2.09,
        2.1 ,  2.14,  2.15,  2.23,  2.24,  2.26,  2.35,  2.37,  2.4 ,
        2.47,  2.54,  2.62,  2.64,  2.9 ,  2.92,  2.92,  2.93,  3.21,
        3.26,  3.3 ,  3.59,  3.68,  4.3 ,  4.64])
>>> outliers_gesd(data, outliers = 5, report = True)
H0: no outliers in the data
Ha: up to 5 outliers in the data
Significance level:   = 0.05
Reject H0 if Ri > Critical Value (i)
Summary Table for Two-Tailed Test
---------------------------------------
      Exact           Test     Critical
  Number of      Statistic    Value, i
Outliers, i      Value, Ri          5 %
---------------------------------------
          1          3.119        3.159
          2          2.943        3.151
          3          3.179        3.144 *
          4           2.81        3.136
          5          2.816        3.128
```

6.3 scikit_posthocs.outliers_grubbs

scikit_posthocs.**outliers_grubbs**(*x: List | ndarray*, *hypo: bool = False*, *alpha: float = 0.05*) → ndarray | bool

Grubbs' Test for Outliers[1]. This is the two-sided version of the test. The null hypothesis implies that there are no outliers in the data set.

Parameters

- **x** (*Union[List, np.ndarray]*) – An array, any object exposing the array interface, containing data to test for an outlier in.
- **hypo** (*bool = False*) – Specifies whether to return a bool value of a hypothesis test result. Returns `True` when we can reject the null hypothesis. Otherwise, `False`. Available options are:
  - `True`: return a hypothesis test result
  - `False`: return a filtered array without an outlier (default)
- **alpha** (*float = 0.05*) – Significance level for a hypothesis test.

Returns

Returns a filtered array if alternative hypothesis is true, otherwise an unfiltered array. Returns null hypothesis test result instead of an array if `hypo` argument is set to `True`.

Return type

Union[np.ndarray, bool]

[1] http://www.itl.nist.gov/div898/handbook/eda/section3/eda35h1.htm

Notes

Examples

```
>>> x = np.array([199.31,199.53,200.19,200.82,201.92,201.95,202.18,245.57])
>>> ph.outliers_grubbs(x)
array([ 199.31,  199.53,  200.19,  200.82,  201.92,  201.95,  202.18])
```

6.4 scikit_posthocs.outliers_tietjen

scikit_posthocs.**outliers_tietjen**(*x: List | ndarray*, *k: int*, *hypo: bool = False*, *alpha: float = 0.05*) → ndarray | bool

Tietjen-Moore test[1] to detect multiple outliers in a univariate data set that follows an approximately normal distribution. The Tietjen-Moore test[2] is a generalization of the Grubbs' test to the case of multiple outliers. If testing for a single outlier, the Tietjen-Moore test is equivalent to the Grubbs' test.

The null hypothesis implies that there are no outliers in the data set.

Parameters

- **x** (*Union[List, np.ndarray]*) – An array, any object exposing the array interface, containing data to test for an outlier in.
- **k** (*int*) – Number of potential outliers to test for. Function tests for outliers in both tails.
- **hypo** (*bool = False*) – Specifies whether to return a bool value of a hypothesis test result. Returns `True` when we can reject the null hypothesis. Otherwise, `False`. Available options are:
  - `True`: return a hypothesis test result
  - `False`: return a filtered array without outliers (default).
- **alpha** (*float = 0.05*) – Significance level for a hypothesis test.

Returns

Returns a filtered array if alternative hypothesis is true, otherwise an unfiltered array. Returns null hypothesis test result instead of an array if `hypo` argument is set to True.

Return type

Union[numpy.ndarray, bool]

Notes

Examples

```
>>> x = np.array([-1.40, -0.44, -0.30, -0.24, -0.22, -0.13, -0.05, 0.06,
0.10, 0.18, 0.20, 0.39, 0.48, 0.63, 1.01])
>>> outliers_tietjen(x, 2)
array([-0.44, -0.3 , -0.24, -0.22, -0.13, -0.05,  0.06,  0.1 ,  0.18,
0.2 ,  0.39,  0.48,  0.63])
```

[1] Tietjen and Moore (August 1972), Some Grubbs-Type Statistics for the Detection of Outliers, Technometrics, 14(3), pp. 583-597.
[2] http://www.itl.nist.gov/div898/handbook/eda/section3/eda35h2.htm

CHAPTER

SEVEN

PLOTTING API REFERENCE

sign_array(p_values[, alpha])	Significance array.
sign_table(p_values[, lower, upper])	Significance table.
sign_plot(x[, g, flat, labels, cmap, ...])	Significance plot, a heatmap of p values (based on Seaborn).
critical_difference_diagram(ranks, sig_matrix, *)	Plot a Critical Difference diagram from ranks and post-hoc results.
compact_letter_display(p_values[, alpha, ...])	Compact Letter Display (CLD) for pairwise comparison results.

7.1 scikit_posthocs.sign_array

scikit_posthocs.**sign_array**(*p_values: List | ndarray | DataFrame*, *alpha: float = 0.05*) → ndarray

Significance array.

Converts an array with p values to a significance array where 0 is False (not significant), 1 is True (significant), and -1 is for diagonal elements.

Parameters

- **p_values** (*Union[List, np.ndarray, DataFrame]*) – Any object exposing the array interface and containing p values.
- **alpha** (*float = 0.05*) – Significance level. Default is 0.05.

Returns

result – Array where 0 is False (not significant), 1 is True (significant), and -1 is for diagonal elements.

Return type

numpy.ndarray

Examples

```
>>> p_values = np.array([[ 1.        ,  0.00119517,  0.00278329],
                         [ 0.00119517,  1.        ,  0.18672227],
                         [ 0.00278329,  0.18672227,  1.        ]])
>>> ph.sign_array(p_values)
array([[-1,  1,  1],
       [ 1, -1,  0],
       [ 1,  0, -1]])
```

7.2 scikit_posthocs.sign_table

scikit_posthocs.**sign_table**(*p_values: List | ndarray | DataFrame*, *lower: bool = True*, *upper: bool = True*) → DataFrame | ndarray

Significance table.

Returns table that can be used in a publication. P values are replaced with asterisks: * - p < 0.05, ** - p < 0.01, *** - p < 0.001.

Parameters

- **p_values** (*Union[List, np.ndarray, DataFrame]*) – Any object exposing the array interface and containing p values.
- **lower** (*bool*) – Defines whether to return the lower triangle.
- **upper** (*bool*) – Defines whether to return the upper triangle.

Returns

result – P values masked with asterisks.

Return type

Union[DataFrame, np.ndarray]

Examples

```
>>> p_values = np.array([[-1.        ,  0.00119517,  0.00278329],
                         [ 0.00119517, -1.        ,  0.18672227],
                         [ 0.00278329,  0.18672227, -1.        ]])
>>> ph.sign_table(p_values)
array([['-', '**', '**'],
       ['**', '-', 'NS'],
       ['**', 'NS', '-']], dtype=object)
```

7.3 scikit_posthocs.sign_plot

scikit_posthocs.**sign_plot**(*x: List | ndarray | DataFrame*, *g: List | ndarray | None = None*, *flat: bool = False*, *labels: bool = True*, *cmap: List | None = None*, *cbar_ax_bbox: Tuple[float, float, float, float] | None = None*, *ax: Axes | None = None*, ***kwargs*) → Axes | Tuple[Axes, Colorbar]

Significance plot, a heatmap of p values (based on Seaborn).

Parameters

- **x** (*Union[List, np.ndarray, DataFrame]*) – If *flat* is False (default), *x* must be a square array, any object exposing the array interface, containing p values. If *flat* is True, *x* must be a sign_array (returned by *scikit_posthocs.sign_array()* function).
- **g** (*Union[List, np.ndarray]*) – An array, any object exposing the array interface, containing group names.
- **flat** (*bool*) – If *flat* is True, plots a significance array as a heatmap using seaborn. If *flat* is False (default), plots an array of p values. Non-flat mode is useful if you need to differentiate significance levels visually. It is the preferred mode.
- **labels** (*bool*) – Plot axes labels (default) or not.

- **cmap** (*list*) – 1) If flat is False (default): List consisting of five elements, that will be exported to ListedColormap method of matplotlib. First is for diagonal elements, second is for non-significant elements, third is for p < 0.001, fourth is for p < 0.01, fifth is for p < 0.05.

  2) If flat is True: List consisting of three elements, that will be exported to ListedColormap method of matplotlib. First is for diagonal elements, second is for non-significant elements, third is for significant ones. 3) If not defined, default colormaps will be used.

- **cbar_ax_bbox** (*list*) – Colorbar axes position rect [left, bottom, width, height] where all quantities are in fractions of figure width and height. Refer to *matplotlib.figure.Figure.add_axes* for more information. Default is [0.95, 0.35, 0.04, 0.3].

- **ax** (*SubplotBase*) – Axes in which to draw the plot, otherwise use the currently-active Axes.

- **kwargs** – Keyword arguments to be passed to seaborn heatmap method. These keyword args cannot be used: cbar, vmin, vmax, center.

Returns

- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – Axes object with the heatmap.

- **cbar** (*matplotlib.colorbar.Colorbar*) – ColorBar object if *flat* is set to False.

Examples

```
>>> x = np.array([[ 1, 1, 1],
                  [ 1, 1, 0],
                  [ 1, 0, 1]])
>>> ph.sign_plot(x, flat = True)
```

7.4 scikit_posthocs.critical_difference_diagram

scikit_posthocs.**critical_difference_diagram**(*ranks: dict | Series*, *sig_matrix: DataFrame*, ** (Keyword-only parameters separator (PEP 3102))*, *cd: float | None = None*, *alpha: float = 0.05*, *ax: Axes | None = None*, *label_fmt_left: str = '{label} ({rank:.2g})'*, *label_fmt_right: str = '({rank:.2g}) {label}'*, *label_props: dict | None = None*, *marker_props: dict | None = None*, *elbow_props: dict | None = None*, *crossbar_props: dict | None = None*, *color_palette: Dict[str, str] | List | None = None*, *text_h_margin: float = 0.01*, *left_only: bool = False*, *hue: Dict | Series | None = None*, *hue_order: List | None = None*) → Dict[str, list]

Plot a Critical Difference diagram from ranks and post-hoc results.

The diagram arranges the average ranks of multiple groups on the x axis in order to facilitate performance comparisons between them. The groups that could not be statistically deemed as different are linked by a horizontal crossbar[1],[2].

```
             rank markers
X axis ---------0----0------------------0-0-----------0---------
                |----|                  | |           |
                |    |                  |---crossbar---|
```

(continues on next page)

[1] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. The Journal of Machine learning research, 7, 1-30.
[2] https://mirkobunse.github.io/CriticalDifferenceDiagrams.jl/stable/

(continued from previous page)

```
      clf1 ----|    |                  | |            |---- clf3
      clf2 ---------|                  | |------------------ clf4
                                       |-------------------- clf5
          |____|
      text_h_margin
```

In the drawing above, the two crossbars indicate that clf1 and clf2 cannot be statistically differentiated, the same occurring between clf3, clf4 and clf5. However, clf1 and clf2 are each significantly lower ranked than clf3, clf4 and clf5.

Parameters

- **ranks** (*dict or Series*) – Indicates the rank value for each sample or estimator (as keys or index).
- **sig_matrix** (*DataFrame*) – The corresponding p-value matrix outputted by post-hoc tests, with indices matching the labels in the ranks argument.
- **alpha** (*float, optional = 0.05*) – Significance level. Default is 0.05. Values below this will be considered statistically different.
- **ax** (*matplotlib.SubplotBase, optional*) – The object in which the plot will be built. Gets the current Axes by default (if None is passed).
- **cd** (*float, optional*) – Critical difference value to be plotted as a horizontal line above the axis.
- **label_fmt_left** (*str, optional*) – The format string to apply to the labels on the left side. The keywords label and rank can be used to specify the sample/estimator name and rank value, respectively, by default '{label} ({rank:.2g})'.
- **label_fmt_right** (*str, optional*) – The same, but for the labels on the right side of the plot. By default '({rank:.2g}) {label}'.
- **label_props** (*dict, optional*) – Parameters to be passed to pyplot.text() when creating the labels, by default None.
- **marker_props** (*dict, optional*) – Parameters to be passed to pyplot.scatter() when plotting the rank markers on the axis, by default None.
- **elbow_props** (*dict, optional*) – Parameters to be passed to pyplot.plot() when creating the elbow lines, by default None.
- **crossbar_props** (*dict, optional*) – Parameters to be passed to pyplot.plot() when creating the crossbars that indicate lack of statistically significant difference. By default None.
- **color_palette** (*dict or list, optional*) – Parameters to be passed when you need specific colors for each category
- **text_h_margin** (*float, optional*) – Space between the text labels and the nearest vertical line of an elbow, by default 0.01.
- **left_only** (*boolean, optional*) – Set all labels in a single left-sided block instead of splitting them into two block, one for the left and one for the right.
- **hue** (*dict or Series, optional*) – Maps each estimator name (same keys/index as `ranks`) to a group label. When provided, colors are assigned per group rather than per estimator. By default None.

- **hue_order** (*list, optional*) – Order of the group levels for color assignment. Must contain all unique values present in `hue`. If None and `hue` is provided, the order is determined by the first appearance in `hue`. By default None.

Returns
Lists of Artists created.

Return type
dict[str, list[matplotlib.Artist]]

Examples

See the *Tutorial*.

References

7.5 scikit_posthocs.compact_letter_display

scikit_posthocs.**compact_letter_display**(*p_values: DataFrame | ndarray*, *alpha: float = 0.05*, *names: List | None = None*, *maxiter: int = 100*) → Series

Compact Letter Display (CLD) for pairwise comparison results.

Assigns letters to groups based on pairwise significance. Groups sharing at least one letter are not significantly different from each other. This provides a compact summary of pairwise comparison results that is commonly used in publications (e.g. as annotations on bar plots).

Parameters

- **p_values** (*Union[DataFrame, np.ndarray]*) – Symmetric matrix of p-values from any `posthoc_*` function. Diagonal values are ignored (treated as non-significant).
- **alpha** (*float = 0.05*) – Significance level. Pairs with p-values `>=` alpha are considered not significantly different.
- **names** (*Optional[List] = None*) – Group names used as the index of the returned Series. If `None` and `p_values` is a DataFrame, its index is used; otherwise integer indices `0, 1, ..., k-1` are used.
- **maxiter** (*int = 100*) – Maximum number of iterations for the set-intersection step.

Returns
result – Series mapping each group name to a letter string. Groups sharing a letter are not significantly different. Spaces indicate that a group does not belong to that letter group.

Return type
pandas.Series

Raises
ValueError – If the number of letter groups exceeds the available alphabet length (52: a-z then A-Z).

Warns
RuntimeWarning – If the algorithm does not converge within `maxiter` iterations.

Notes

Uses the set-intersection algorithm described by Perktold (unpublished) which is equivalent to the sweep-and-absorb algorithm of Piepho (2004) for standard use cases. Groups are sorted by their smallest member index to ensure a deterministic letter assignment.

References

Piepho, Hans-Peter (2004). An Algorithm for a Letter-Based Representation of All-Pairwise Comparisons. Journal of Computational and Graphical Statistics, 13(2), 456-466. https://doi.org/10.1198/1061860043515

Examples

```
>>> import scikit_posthocs as sp
>>> x = [[1, 2, 1, 3, 1, 4], [12, 3, 11, 9, 3, 8, 1],
...      [10, 22, 12, 9, 8, 3], [14, 12, 16, 17, 5, 9]]
>>> pc = sp.posthoc_dunn(x, p_adjust='holm')
>>> sp.compact_letter_display(pc, alpha=0.05)
1    a
2    ab
3     b
4     b
Name: letters, dtype: str
```

CHAPTER

EIGHT

POST-HOCS API REFERENCE

posthoc_conover(a[, val_col, group_col, ...])	Post hoc pairwise test for multiple comparisons of mean rank sums (Conover´s test).
posthoc_dunn(a[, val_col, group_col, ...])	Post hoc pairwise test for multiple comparisons of mean rank sums (Dunn's test).
posthoc_nemenyi(a[, val_col, group_col, ...])	Post hoc pairwise test for multiple comparisons of mean rank sums (Nemenyi's test).
posthoc_nemenyi_friedman(a[, y_col, ...])	Calculate pairwise comparisons using Nemenyi post hoc test for unreplicated blocked data.
posthoc_conover_friedman(a[, y_col, ...])	Calculate pairwise comparisons using Conover post hoc test for unreplicated blocked data.
posthoc_siegel_friedman(a[, y_col, ...])	Siegel and Castellan´s All-Pairs Comparisons Test for Unreplicated Blocked Data.
posthoc_miller_friedman(a[, y_col, ...])	Miller´s All-Pairs Comparisons Test for Unreplicated Blocked Data.
posthoc_npm_test(a[, val_col, group_col, ...])	Calculate pairwise comparisons using Nashimoto and Wright´s all-pairs comparison procedure (NPM test) for simply ordered mean ranksums.
posthoc_durbin(a[, y_col, group_col, ...])	Pairwise post hoc test for multiple comparisons of rank sums according to Durbin and Conover for a two-way balanced incomplete block design (BIBD).
posthoc_anderson(a[, val_col, group_col, ...])	Anderson-Darling Pairwise Test for k-samples.
posthoc_quade(a[, y_col, group_col, ...])	Calculate pairwise comparisons using Quade's post hoc test for unreplicated blocked data.
posthoc_vanwaerden(a[, val_col, group_col, ...])	Van der Waerden's test for pairwise multiple comparisons between group levels.
posthoc_tukey_hsd(a[, val_col, group_col, sort])	Pairwise comparisons with TukeyHSD confidence intervals.
posthoc_ttest(a[, val_col, group_col, ...])	Pairwise T test for multiple comparisons of independent groups.
posthoc_mannwhitney(a[, val_col, group_col, ...])	Pairwise comparisons with Mann-Whitney rank test.
posthoc_wilcoxon(a[, val_col, group_col, ...])	Pairwise comparisons with Wilcoxon signed-rank test.
posthoc_scheffe(a[, val_col, group_col, sort])	Scheffe's all-pairs comparisons test for normally distributed data with equal group variances.
posthoc_tamhane(a[, val_col, group_col, ...])	Tamhane's T2 all-pairs comparison test for normally distributed data with unequal variances.
posthoc_tukey(a[, val_col, group_col, sort])	Performs Tukey's all-pairs comparisons test for normally distributed data with equal group variances.

continues on next page

Table 1 – continued from previous page

posthoc_dscf(a[, val_col, group_col, sort])	Dwass, Steel, Critchlow and Fligner all-pairs comparison test for a one-factorial layout with non-normally distributed residuals.
posthoc_dunnett(a[, val_col, group_col, ...])	Dunnett's test [1, 2, 3] for multiple comparisons against a control group, used after parametric ANOVA.

8.1 scikit_posthocs.posthoc_conover

scikit_posthocs.**posthoc_conover**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *p_adjust: str | None = None*, *sort: bool = True*) → DataFrame

Post hoc pairwise test for multiple comparisons of mean rank sums (Conover´s test). May be used after Kruskal-Wallis one-way analysis of variance by ranks to do pairwise comparisons[1].

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame. Array must be two-dimensional. Second dimension may vary, i.e. groups may have different lengths.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.
- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **p_adjust** (*str, optional*) – Method for adjusting p values. See *statsmodels.sandbox.stats.multicomp* for details. Available methods are: 'bonferroni' : one-step correction 'sidak' : one-step correction 'holm-sidak' : step-down method using Sidak adjustments 'holm' : step-down method using Bonferroni adjustments 'simes-hochberg' : step-up method (independent) 'hommel' : closed method based on Simes tests (non-negative) 'fdr_bh' : Benjamini/Hochberg (non-negative) 'fdr_by' : Benjamini/Yekutieli (negative) 'fdr_tsbh' : two stage fdr correction (non-negative) 'fdr_tsbky' : two stage fdr correction (non-negative)
- **sort** (*bool, optional*) – Specifies whether to sort DataFrame by *group_col* or not. Recommended unless you sort your data manually.

Returns

result – P values.

Return type

pandas.DataFrame

Notes

A tie correction are employed according to Conover[1].

[1] W. J. Conover and R. L. Iman (1979), On multiple-comparisons procedures, Tech. Rep. LA-7677-MS, Los Alamos Scientific Laboratory.

References

Examples

```
>>> x = [[1,2,3,5,1], [12,31,54, np.nan], [10,12,6,74,11]]
>>> sp.posthoc_conover(x, p_adjust = 'holm')
```

8.2 scikit_posthocs.posthoc_dunn

scikit_posthocs.**posthoc_dunn**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *p_adjust: str | None = None*, *sort: bool = True*) → DataFrame

Post hoc pairwise test for multiple comparisons of mean rank sums (Dunn's test). May be used after Kruskal-Wallis one-way analysis of variance by ranks to do pairwise comparisons[1],[2].

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame. Array must be two-dimensional. Second dimension may vary, i.e. groups may have different lengths.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.
- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **p_adjust** (*str, optional*) – Method for adjusting p values. See *statsmodels.sandbox.stats.multicomp* for details. Available methods are: 'bonferroni' : one-step correction 'sidak' : one-step correction 'holm-sidak' : step-down method using Sidak adjustments 'holm' : step-down method using Bonferroni adjustments 'simes-hochberg' : step-up method (independent) 'hommel' : closed method based on Simes tests (non-negative) 'fdr_bh' : Benjamini/Hochberg (non-negative) 'fdr_by' : Benjamini/Yekutieli (negative) 'fdr_tsbh' : two stage fdr correction (non-negative) 'fdr_tsbky' : two stage fdr correction (non-negative)
- **sort** (*bool, optional*) – Specifies whether to sort DataFrame by group_col or not. Recommended unless you sort your data manually.

Returns

result – P values.

Return type

pandas.DataFrame

Notes

A tie correction will be employed according to Glantz (2012).

[1] O.J. Dunn (1964). Multiple comparisons using rank sums. Technometrics, 6, 241-252.
[2] S.A. Glantz (2012), Primer of Biostatistics. New York: McGraw Hill.

References

Examples

```
>>> x = [[1,2,3,5,1], [12,31,54, np.nan], [10,12,6,74,11]]
>>> sp.posthoc_dunn(x, p_adjust = 'holm')
```

8.3 scikit_posthocs.posthoc_nemenyi

scikit_posthocs.**posthoc_nemenyi**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *dist: str = 'chi'*, *sort: bool = True*) → DataFrame

Post hoc pairwise test for multiple comparisons of mean rank sums (Nemenyi's test). May be used after Kruskal-Wallis one-way analysis of variance by ranks to do pairwise comparisons[1].

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame. Array must be two-dimensional. Second dimension may vary, i.e. groups may have different lengths.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.
- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **dist** (*str, optional*) – Method for determining the p value. The default distribution is "chi" (chi-squared), else "tukey" (studentized range).
- **sort** (*bool, optional*) – Specifies whether to sort DataFrame by group_col or not. Recommended unless you sort your data manually.

Returns
result – P values.

Return type
pandas.DataFrame

Notes

A tie correction will be employed according to Glantz (2012).

References

Examples

```
>>> x = [[1,2,3,5,1], [12,31,54, np.nan], [10,12,6,74,11]]
>>> sp.posthoc_nemenyi(x)
```

[1] Lothar Sachs (1997), Angewandte Statistik. Berlin: Springer. Pages: 395-397, 662-664.

8.4 scikit_posthocs.posthoc_nemenyi_friedman

scikit_posthocs.**posthoc_nemenyi_friedman**(*a: list | ndarray | DataFrame*, *y_col: str | None = None*, *group_col: str | None = None*, *block_col: str | None = None*, *block_id_col: str | None = None*, *melted: bool = False*, *sort: bool = False*) → DataFrame

Calculate pairwise comparisons using Nemenyi post hoc test for unreplicated blocked data. This test is usually conducted post hoc if significant results of the Friedman's test are obtained. The statistics refer to upper quantiles of the studentized range distribution (Tukey)[1],[2],[3].

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame.

  If *melted* is set to False (default), *a* is a typical matrix of block design, i.e. rows are blocks, and columns are groups. In this case you do not need to specify col arguments.

  If *a* is an array and *melted* is set to True, y_col, block_col and group_col must specify the indices of columns containing elements of correspondary type.

  If *a* is a Pandas DataFrame and *melted* is set to True, y_col, block_col and group_col must specify columns names (strings).

- **y_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains y data.
- **block_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains blocking factor values.
- **group_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains treatment (group) factor values.
- **melted** (*bool, optional*) – Specifies if data are given as melted columns "y", "blocks", and "groups".
- **sort** (*bool, optional*) – If True, sort data by block and group columns.

Returns
result – P values.

Return type
pandas.DataFrame

Notes

A one-way ANOVA with repeated measures that is also referred to as ANOVA with unreplicated block design can also be conducted via Friedman's test. The consequent post hoc pairwise multiple comparison test according to Nemenyi is conducted with this function.

This function does not test for ties.

References

[1] J. Demsar (2006), Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research, 7, 1-30.
[2] P. Nemenyi (1963) Distribution-free Multiple Comparisons. Ph.D. thesis, Princeton University.
[3] L. Sachs (1997), Angewandte Statistik. Berlin: Springer. Pages: 668-675.

Examples

```
>>> # Non-melted case, x is a block design matrix, i.e. rows are blocks
>>> # and columns are groups.
>>> x = np.array([[31,27,24],[31,28,31],[45,29,46],[21,18,48],[42,36,46],[32,17,
↪40]])
>>> sp.posthoc_nemenyi_friedman(x)
```

8.5 scikit_posthocs.posthoc_conover_friedman

scikit_posthocs.**posthoc_conover_friedman**(*a: list | ndarray | DataFrame*, *y_col: str | None = None*, *block_col: str | None = None*, *block_id_col: str | None = None*, *group_col: str | None = None*, *melted: bool = False*, *sort: bool = False*, *p_adjust: str | None = None*) → DataFrame

Calculate pairwise comparisons using Conover post hoc test for unreplicated blocked data. This test is usually conducted post hoc after significant results of the Friedman test. The statistics refer to the Student t distribution[1],[2].

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame.

  If *melted* is set to False (default), *a* is a typical matrix of block design, i.e. rows are blocks, and columns are groups. In this case you do not need to specify col arguments.

  If *a* is an array and *melted* is set to True, y_col, block_col and group_col must specify the indices of columns containing elements of correspondary type.

  If *a* is a Pandas DataFrame and *melted* is set to True, y_col, block_col and group_col must specify columns names (strings).

- **y_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains y data.

- **block_col** (*str or int*) – Must be specified if *a* is a melted pandas DataFrame object. Name of the column that contains blocking factor values.

- **block_id_col** (*str or int*) – Must be specified if *a* is a melted pandas DataFrame object. Name of the column that contains identifiers of blocking factor values.

- **group_col** (*str or int*) – Must be specified if *a* is a melted pandas DataFrame object. Name of the column that contains treatment (group) factor values.

- **melted** (*bool, optional*) – Specifies if data are given as melted columns "y", "blocks", and "groups".

- **sort** (*bool, optional*) – If True, sort data by block and group columns.

- **p_adjust** (*str, optional*) – Method for adjusting p values. See statsmodels.sandbox.stats.multicomp for details. Available methods are: 'bonferroni' : one-step correction 'sidak' : one-step correction 'holm-sidak' : step-down method using Sidak adjustments 'holm' : step-down method using Bonferroni adjustments 'simes-hochberg' : step-up method (independent) 'hommel' : closed method based on Simes tests (non-negative) 'fdr_bh' : Benjamini/Hochberg (non-negative) 'fdr_by' : Benjamini/Yekutieli (negative) 'fdr_tsbh' : two stage fdr correction (non-negative) 'fdr_tsbky' : two stage fdr correction (non-negative) 'single-step' : uses Tukey distribution for multiple comparisons

[1] W. J. Conover and R. L. Iman (1979), On multiple-comparisons procedures, Tech. Rep. LA-7677-MS, Los Alamos Scientific Laboratory.
[2] W. J. Conover (1999), Practical nonparametric Statistics, 3rd. Edition, Wiley.

Returns
result – P values.

Return type
pandas.DataFrame

Notes

A one-way ANOVA with repeated measures that is also referred to as ANOVA with unreplicated block design can also be conducted via the friedman.test. The consequent post hoc pairwise multiple comparison test according to Conover is conducted with this function.

If y is a matrix, than the columns refer to the treatment and the rows indicate the block.

References

Examples

```
>>> x = np.array([[31,27,24],[31,28,31],[45,29,46],[21,18,48],[42,36,46],[32,17,
↪40]])
>>> sp.posthoc_conover_friedman(x)
```

8.6 scikit_posthocs.posthoc_siegel_friedman

scikit_posthocs.**posthoc_siegel_friedman**(*a: list | ndarray | DataFrame*, *y_col: str | None = None*, *group_col: str | None = None*, *block_col: str | None = None*, *block_id_col: str | None = None*, *p_adjust: str | None = None*, *melted: bool = False*, *sort: bool = False*) → DataFrame

Siegel and Castellan´s All-Pairs Comparisons Test for Unreplicated Blocked Data. See authors' paper for additional information[1].

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame.

  If *melted* is set to False (default), *a* is a typical matrix of block design, i.e. rows are blocks, and columns are groups. In this case you do not need to specify col arguments.

  If *a* is an array and *melted* is set to True, y_col, block_col and group_col must specify the indices of columns containing elements of correspondary type.

  If *a* is a Pandas DataFrame and *melted* is set to True, y_col, block_col and group_col must specify columns names (strings).

- **y_col** (*str or int*) – Must be specified if *a* is a melted pandas DataFrame object. Name of the column that contains y data.
- **block_col** (*str or int*) – Must be specified if *a* is a melted pandas DataFrame object. Name of the column that contains blocking factor values.
- **group_col** (*str or int*) – Must be specified if *a* is a melted pandas DataFrame object. Name of the column that contains treatment (group) factor values.
- **block_id_col** (*str or int*) – Must be specified if *a* is a melted pandas DataFrame object. Name of the column that contains identifiers of blocking factor values.

[1] S. Siegel, N. J. Castellan Jr. (1988), Nonparametric Statistics for the Behavioral Sciences. 2nd ed. New York: McGraw-Hill.

- **melted** (*bool, optional*) – Specifies if data are given as melted columns "y", "blocks", and "groups".
- **sort** (*bool, optional*) – If True, sort data by block and group columns.
- **p_adjust** (*str, optional*) – Method for adjusting p values. See statsmodels.sandbox.stats.multicomp for details. Available methods are: 'bonferroni' : one-step correction 'sidak' : one-step correction 'holm-sidak' : step-down method using Sidak adjustments 'holm' : step-down method using Bonferroni adjustments 'simes-hochberg' : step-up method (independent) 'hommel' : closed method based on Simes tests (non-negative) 'fdr_bh' : Benjamini/Hochberg (non-negative) 'fdr_by' : Benjamini/Yekutieli (negative) 'fdr_tsbh' : two stage fdr correction (non-negative) 'fdr_tsbky' : two stage fdr correction (non-negative)

Returns
result – P values.

Return type
pandas.DataFrame

Notes

For all-pairs comparisons in a two factorial unreplicated complete block design with non-normally distributed residuals, Siegel and Castellan's test can be performed on Friedman-type ranked data.

References

Examples

```
>>> x = np.array([[31,27,24],[31,28,31],[45,29,46],[21,18,48],[42,36,46],[32,17,
↪40]])
>>> sp.posthoc_siegel_friedman(x)
```

8.7 scikit_posthocs.posthoc_miller_friedman

scikit_posthocs.**posthoc_miller_friedman**(*a: list | ndarray | DataFrame*, *y_col: str | None = None*, *group_col: str | None = None*, *block_col: str | None = None*, *block_id_col: str | None = None*, *melted: bool = False*, *sort: bool = False*) → DataFrame

Miller´s All-Pairs Comparisons Test for Unreplicated Blocked Data. The p-values are computed from the chi-square distribution[1],[2],[3].

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame.

  If *melted* is set to False (default), *a* is a typical matrix of block design, i.e. rows are blocks, and columns are groups. In this case you do not need to specify col arguments.

  If *a* is an array and *melted* is set to True, y_col, block_col and group_col must specify the indices of columns containing elements of correspondary type.

  If *a* is a Pandas DataFrame and *melted* is set to True, y_col, block_col and group_col must specify columns names (strings).

[1] J. Bortz J, G. A. Lienert, K. Boehnke (1990), Verteilungsfreie Methoden in der Biostatistik. Berlin: Springerself.
[2] R. G. Miller Jr. (1996), Simultaneous statistical inference. New York: McGraw-Hill.
[3] E. L. Wike (2006), Data Analysis. A Statistical Primer for Psychology Students. New Brunswick: Aldine Transaction.

- **y_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains y data.
- **group_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains treatment (group) factor values.
- **block_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains blocking factor values.
- **block_id_col** (*str or int*) – Must be specified if *a* is a melted pandas DataFrame object. Name of the column that contains identifiers of blocking factor values.
- **melted** (*bool, optional*) – Specifies if data are given as melted columns "y", "blocks", and "groups".
- **sort** (*bool, optional*) – If True, sort data by block and group columns.

Returns
result – P values.

Return type
pandas.DataFrame

Notes

For all-pairs comparisons in a two factorial unreplicated complete block design with non-normally distributed residuals, Miller's test can be performed on Friedman-type ranked data.

References

Examples

```
>>> x = np.array([[31,27,24],[31,28,31],[45,29,46],[21,18,48],[42,36,46],[32,17,
↪40]])
>>> sp.posthoc_miller_friedman(x)
```

8.8 scikit_posthocs.posthoc_npm_test

scikit_posthocs.**posthoc_npm_test**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *alternative: Literal['greater', 'less'] = 'greater'*, *nperm: int = 1000*, *sort: bool = False*) → DataFrame

Calculate pairwise comparisons using Nashimoto and Wright´s all-pairs comparison procedure (NPM test) for simply ordered mean ranksums.

NPM test is basically an extension of Nemenyi´s procedure for testing increasingly ordered alternatives[1].

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.

[1] Nashimoto, K., Wright, F.T., (2005), Multiple comparison procedures for detecting differences in simply ordered means. Comput. Statist. Data Anal. 48, 291–306.

- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **sort** (*bool, optional*) – If True, sort data by block and group columns.
- **alternative** (*str, optional*) – Alternative hypothesis being tested. Can be either "greater" (by default) or "less".
- **nperm** (*int, optional*) – Number of permutations to perform for calculating p values.

Returns
result – P values.

Return type
pandas.DataFrame

Notes

An asymetric permutation test is conducted for calculating p values.

References

Examples

```
>>> x = np.array([[102,109,114,120,124],
                  [110,112,123,130,145],
                  [132,141,156,160,172]])
>>> sp.posthoc_npm_test(x)
```

8.9 scikit_posthocs.posthoc_durbin

scikit_posthocs.**posthoc_durbin**(*a: list | ndarray | DataFrame*, *y_col: str | None = None*, *group_col: str | None = None*, *block_col: str | None = None*, *block_id_col: str | None = None*, *p_adjust: str | None = None*, *melted: bool = False*, *sort: bool = False*) → DataFrame

Pairwise post hoc test for multiple comparisons of rank sums according to Durbin and Conover for a two-way balanced incomplete block design (BIBD). See references for additional information[1],[2].

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame.

  If *melted* is set to False (default), *a* is a typical matrix of block design, i.e. rows are blocks, and columns are groups. In this case you do not need to specify col arguments.

  If *a* is an array and *melted* is set to True, y_col, block_col and group_col must specify the indices of columns containing elements of correspondary type.

  If *a* is a Pandas DataFrame and *melted* is set to True, y_col, block_col and group_col must specify columns names (string).

- **y_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains y data.

[1] W. J. Conover and R. L. Iman (1979), On multiple-comparisons procedures, Tech. Rep. LA-7677-MS, Los Alamos Scientific Laboratory.
[2] W. J. Conover (1999), Practical nonparametric Statistics, 3rd. edition, Wiley.

- **group_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains treatment (group) factor values.
- **block_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains blocking factor values.
- **block_id_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains identifiers of blocking factor values.
- **melted** (*bool, optional*) – Specifies if data are given as melted columns "y", "blocks", and "groups".
- **sort** (*bool, optional*) – If True, sort data by block and group columns.
- **p_adjust** (*str, optional*) – Method for adjusting p values. See statsmodels.sandbox.stats.multicomp for details. Available methods are: 'bonferroni' : one-step correction 'sidak' : one-step correction 'holm-sidak' : step-down method using Sidak adjustments 'holm' : step-down method using Bonferroni adjustments 'simes-hochberg' : step-up method (independent) 'hommel' : closed method based on Simes tests (non-negative) 'fdr_bh' : Benjamini/Hochberg (non-negative) 'fdr_by' : Benjamini/Yekutieli (negative) 'fdr_tsbh' : two stage fdr correction (non-negative) 'fdr_tsbky' : two stage fdr correction (non-negative)

Returns
result – P values.

Return type
pandas.DataFrame

References

Examples

```
>>> x = np.array([[31,27,24],[31,28,31],[45,29,46],[21,18,48],[42,36,46],[32,17,
↪40]])
>>> sp.posthoc_durbin(x)
```

8.10 scikit_posthocs.posthoc_anderson

scikit_posthocs.**posthoc_anderson**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *midrank: bool = True*, *p_adjust: str | None = None*, *sort: bool = False*) → DataFrame

Anderson-Darling Pairwise Test for k-samples. Tests the null hypothesis that k-samples are drawn from the same population without having to specify the distribution function of that population[1].

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.
- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.

[1] F.W. Scholz, M.A. Stephens (1987), K-Sample Anderson-Darling Tests, Journal of the American Statistical Association, Vol. 82, pp. 918-924.

- **midrank** (*bool, optional*) – Type of Anderson-Darling test which is computed. If set to True (default), the midrank test applicable to continuous and discrete populations is performed. If False, the right side empirical distribution is used.
- **sort** (*bool, optional*) – If True, sort data by block and group columns.
- **p_adjust** (*str, optional*) – Method for adjusting p values. See statsmodels.sandbox.stats.multicomp for details. Available methods are: 'bonferroni' : one-step correction 'sidak' : one-step correction 'holm-sidak' : step-down method using Sidak adjustments 'holm' : step-down method using Bonferroni adjustments 'simes-hochberg' : step-up method (independent) 'hommel' : closed method based on Simes tests (non-negative) 'fdr_bh' : Benjamini/Hochberg (non-negative) 'fdr_by' : Benjamini/Yekutieli (negative) 'fdr_tsbh' : two stage fdr correction (non-negative) 'fdr_tsbky' : two stage fdr correction (non-negative)

Returns

result – P values.

Return type

pandas.DataFrame

References

Examples

```
>>> x = np.array([[2.9, 3.0, 2.5, 2.6, 3.2], [3.8, 2.7, 4.0, 2.4], [2.8, 3.4, 3.7,␣
↪2.2, 2.0]])
>>> sp.posthoc_anderson(x)
```

8.11 scikit_posthocs.posthoc_quade

scikit_posthocs.**posthoc_quade**(*a: list | ndarray | DataFrame*, *y_col: str | None = None*, *group_col: str | None = None*, *block_col: str | None = None*, *block_id_col: str | None = None*, *dist: str = 't'*, *p_adjust: str | None = None*, *melted: bool = False*, *sort: bool = False*) → DataFrame

Calculate pairwise comparisons using Quade's post hoc test for unreplicated blocked data. This test is usually conducted if significant results were obtained by the omnibus test[1],[2],[3].

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame.

  If *melted* is set to False (default), *a* is a typical matrix of block design, i.e. rows are blocks, and columns are groups. In this case you do not need to specify col arguments.

  If *a* is an array and *melted* is set to True, y_col, block_col and group_col must specify the indices of columns containing elements of correspondary type.

  If *a* is a Pandas DataFrame and *melted* is set to True, y_col, block_col and group_col must specify columns names (string).

[1] W. J. Conover (1999), Practical nonparametric Statistics, 3rd. Edition, Wiley.

[2] N. A. Heckert and J. J. Filliben (2003). NIST Handbook 148: Dataplot Reference Manual, Volume 2: Let Subcommands and Library Functions. National Institute of Standards and Technology Handbook Series, June 2003.

[3] D. Quade (1979), Using weighted rankings in the analysis of complete blocks with additive block effects. Journal of the American Statistical Association, 74, 680-683.

- **y_col** (*str or int, optional*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains y data.
- **block_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains blocking factor values.
- **group_col** (*str or int*) – Must be specified if *a* is a pandas DataFrame object. Name of the column that contains treatment (group) factor values.
- **dist** (*str, optional*) – Method for determining p values. The default distribution is "t", else "normal".
- **melted** (*bool, optional*) – Specifies if data are given as melted columns "y", "blocks", and "groups".
- **sort** (*bool, optional*) – If True, sort data by block and group columns.
- **p_adjust** (*str, optional*) – Method for adjusting p values. See statsmodels.sandbox.stats.multicomp for details. Available methods are: 'bonferroni' : one-step correction 'sidak' : one-step correction 'holm-sidak' : step-down method using Sidak adjustments 'holm' : step-down method using Bonferroni adjustments 'simes-hochberg' : step-up method (independent) 'hommel' : closed method based on Simes tests (non-negative) 'fdr_bh' : Benjamini/Hochberg (non-negative) 'fdr_by' : Benjamini/Yekutieli (negative) 'fdr_tsbh' : two stage fdr correction (non-negative) 'fdr_tsbky' : two stage fdr correction (non-negative)

Returns
result – P values.

Return type
pandas.DataFrame

References

Examples

```
>>> x = np.array([[31,27,24],[31,28,31],[45,29,46],[21,18,48],[42,36,46],[32,17,
↪40]])
>>> sp.posthoc_quade(x)
```

8.12 scikit_posthocs.posthoc_vanwaerden

scikit_posthocs.**posthoc_vanwaerden**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *sort: bool = False*, *p_adjust: str | None = None*) → DataFrame

Van der Waerden's test for pairwise multiple comparisons between group levels. See references for additional information[1],[2].

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.

[1] W. J. Conover and R. L. Iman (1979), On multiple-comparisons procedures, Tech. Rep. LA-7677-MS, Los Alamos Scientific Laboratory.
[2] B. L. van der Waerden (1952) Order tests for the two-sample problem and their power, Indagationes Mathematicae, 14, 453-458.

- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **sort** (*bool, optional*) – If True, sort data by block and group columns.
- **p_adjust** (*str, optional*) – Method for adjusting p values. See statsmodels.sandbox.stats.multicomp for details. Available methods are: 'bonferroni' : one-step correction 'sidak' : one-step correction 'holm-sidak' : step-down method using Sidak adjustments 'holm' : step-down method using Bonferroni adjustments 'simes-hochberg' : step-up method (independent) 'hommel' : closed method based on Simes tests (non-negative) 'fdr_bh' : Benjamini/Hochberg (non-negative) 'fdr_by' : Benjamini/Yekutieli (negative) 'fdr_tsbh' : two stage fdr correction (non-negative) 'fdr_tsbky' : two stage fdr correction (non-negative)

Returns
result – P values.

Return type
pandas.DataFrame

Notes

For one-factorial designs with samples that do not meet the assumptions for one-way-ANOVA and subsequent post hoc tests, the van der Waerden test using normal scores can be employed. Provided that significant differences were detected by this global test, one may be interested in applying post hoc tests according to van der Waerden for pairwise multiple comparisons of the group levels.

There is no tie correction applied in this function.

References

Examples

```
>>> x = np.array([[10,'a'], [59,'a'], [76,'b'], [10, 'b']])
>>> sp.posthoc_vanwaerden(x, val_col = 0, group_col = 1)
```

8.13 scikit_posthocs.posthoc_tukey_hsd

scikit_posthocs.**posthoc_tukey_hsd**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *sort: bool = True*) → DataFrame

Pairwise comparisons with TukeyHSD confidence intervals. This is a convenience function to make statsmodels *pairwise_tukeyhsd* method more applicable for further use.

Parameters

- **x** (*array_like or pandas Series object, 1d*) – An array, any object exposing the array interface, containing dependent variable values (test or response variable). Values should have a non-nominal scale. NaN values will cause an error (please handle manually).
- **g** (*array_like or pandas Series object, 1d*) – An array, any object exposing the array interface, containing independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical).
- **alpha** (*float, optional*) – Significance level for the test. Default is 0.05.

Returns

result – DataFrame with 0, 1, and -1 values, where 0 is False (not significant), 1 is True (significant), and -1 is for diagonal elements.

Return type

pandas.DataFrame

Examples

```
>>> x = [[1,2,3,4,5], [35,31,75,40,21], [10,6,9,6,1]]
>>> g = [['a'] * 5, ['b'] * 5, ['c'] * 5]
>>> sp.posthoc_tukey_hsd(np.concatenate(x), np.concatenate(g))
```

8.14 scikit_posthocs.posthoc_ttest

scikit_posthocs.**posthoc_ttest**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *pool_sd: bool = False*, *equal_var: bool = True*, *p_adjust: str | None = None*, *sort: bool = False*) → DataFrame

Pairwise T test for multiple comparisons of independent groups. May be used after a parametric ANOVA to do pairwise comparisons.

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame. Array must be two-dimensional.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.
- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **equal_var** (*bool, optional*) – If True (default), perform a standard independent test that assumes equal population variances[1]. If False, perform Welch's t-test, which does not assume equal population variance[2].
- **pool_sd** (*bool, optional*) – Calculate a common SD for all groups and use that for all comparisons (this can be useful if some groups are small). This method does not actually call scipy ttest_ind() function, so extra arguments are ignored. Default is False.
- **p_adjust** (*str, optional*) – Method for adjusting p values. See statsmodels.sandbox.stats.multicomp for details. Available methods are: 'bonferroni' : one-step correction 'sidak' : one-step correction 'holm-sidak' : step-down method using Sidak adjustments 'holm' : step-down method using Bonferroni adjustments 'simes-hochberg' : step-up method (independent) 'hommel' : closed method based on Simes tests (non-negative) 'fdr_bh' : Benjamini/Hochberg (non-negative) 'fdr_by' : Benjamini/Yekutieli (negative) 'fdr_tsbh' : two stage fdr correction (non-negative) 'fdr_tsbky' : two stage fdr correction (non-negative)
- **sort** (*bool, optional*) – Specifies whether to sort DataFrame by group_col or not. Recommended unless you sort your data manually.

[1] http://en.wikipedia.org/wiki/T-test#Independent_two-sample_t-test
[2] http://en.wikipedia.org/wiki/Welch%27s_t_test

Returns
result – P values.

Return type
pandas.DataFrame

References

Examples

```
>>> x = [[1,2,3,5,1], [12,31,54, np.nan], [10,12,6,74,11]]
>>> sp.posthoc_ttest(x, p_adjust = 'holm')
array([[-1.        ,  0.04600899,  0.31269089],
       [ 0.04600899, -1.        ,  0.6327077 ],
       [ 0.31269089,  0.6327077 , -1.        ]])
```

8.15 scikit_posthocs.posthoc_mannwhitney

scikit_posthocs.**posthoc_mannwhitney**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *use_continuity: bool = True*, *alternative: str = 'two-sided'*, *p_adjust: str | None = None*, *sort: bool = True*) → DataFrame

Pairwise comparisons with Mann-Whitney rank test.

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame. Array must be two-dimensional.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.
- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **use_continuity** (*bool, optional*) – Whether a continuity correction (1/2.) should be taken into account. Default is True.
- **alternative** (*['two-sided', 'less', or 'greater'], optional*) – Whether to get the p-value for the one-sided hypothesis ('less' or 'greater') or for the two-sided hypothesis ('two-sided'). Defaults to 'two-sided'.
- **p_adjust** (*str, optional*) – Method for adjusting p values. See statsmodels.sandbox.stats.multicomp for details. Available methods are: 'bonferroni' : one-step correction 'sidak' : one-step correction 'holm-sidak' : step-down method using Sidak adjustments 'holm' : step-down method using Bonferroni adjustments 'simes-hochberg' : step-up method (independent) 'hommel' : closed method based on Simes tests (non-negative) 'fdr_bh' : Benjamini/Hochberg (non-negative) 'fdr_by' : Benjamini/Yekutieli (negative) 'fdr_tsbh' : two stage fdr correction (non-negative) 'fdr_tsbky' : two stage fdr correction (non-negative)
- **sort** (*bool, optional*) – Specifies whether to sort DataFrame by group_col or not. Recommended unless you sort your data manually.

Returns
result – P values.

Return type
pandas.DataFrame

Notes

Refer to *scipy.stats.mannwhitneyu* reference page for further details.

Examples

```
>>> x = [[1,2,3,4,5], [35,31,75,40,21], [10,6,9,6,1]]
>>> sp.posthoc_mannwhitney(x, p_adjust = 'holm')
```

8.16 scikit_posthocs.posthoc_wilcoxon

scikit_posthocs.**posthoc_wilcoxon**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *method: str = 'auto'*, *zero_method: str = 'wilcox'*, *correction: bool = False*, *p_adjust: str | None = None*, *sort: bool = False*) → DataFrame

Pairwise comparisons with Wilcoxon signed-rank test.

It is a non-parametric version of the paired T-test for use with non-parametric ANOVA.

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame. Array must be two-dimensional.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.
- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **method** (*{"auto", "exact", "approx"}, optional*) – Method to calculate the p-value. Default is "auto".
- **zero_method** (*{"pratt", "wilcox", "zsplit"}, optional*) – "pratt": Pratt treatment, includes zero-differences in the ranking process (more conservative) "wilcox": Wilcox treatment, discards all zero-differences "zsplit": Zero rank split, just like Pratt, but spliting the zero rank between positive and negative ones
- **correction** (*bool, optional*) – If True, apply continuity correction by adjusting the Wilcoxon rank statistic by 0.5 towards the mean value when computing the z-statistic. Default is False.
- **p_adjust** (*str, optional*) – Method for adjusting p values. See statsmodels.sandbox.stats.multicomp for details. Available methods are: 'bonferroni' : one-step correction 'sidak' : one-step correction 'holm-sidak' : step-down method using Sidak adjustments 'holm' : step-down method using Bonferroni adjustments 'simes-hochberg' : step-up method (independent) 'hommel' : closed method based on Simes tests (non-negative) 'fdr_bh' : Benjamini/Hochberg (non-negative) 'fdr_by' : Benjamini/Yekutieli (negative) 'fdr_tsbh' : two stage fdr correction (non-negative) 'fdr_tsbky' : two stage fdr correction (non-negative)
- **sort** (*bool, optional*) – Specifies whether to sort DataFrame by group_col and val_col or not. Default is False.

Returns
result – P values.

Return type
pandas.DataFrame

Notes

Refer to *scipy.stats.wilcoxon* reference page for further details[1].

References

Examples

```
>>> x = [[1,2,3,4,5], [35,31,75,40,21], [10,6,9,6,1]]
>>> sp.posthoc_wilcoxon(x)
```

8.17 scikit_posthocs.posthoc_scheffe

scikit_posthocs.**posthoc_scheffe**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *sort: bool = False*) → DataFrame

Scheffe's all-pairs comparisons test for normally distributed data with equal group variances. For all-pairs comparisons in an one-factorial layout with normally distributed residuals and equal variances Scheffe's test can be performed with parametric ANOVA[1],[2],[3].

A total of m = k(k-1)/2 hypotheses can be tested.

Parameters

- **a** (*Union[list, np.ndarray, DataFrame]*) – An array, any object exposing the array interface or a pandas DataFrame.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.
- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **sort** (*bool, optional*) – If True, sort data by block and group columns.

Returns
result – P values.

Return type
pandas.DataFrame

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html
[1] J. Bortz (1993) Statistik für Sozialwissenschaftler. 4. Aufl., Berlin: Springer.
[2] L. Sachs (1997) Angewandte Statistik, New York: Springer.
[3] H. Scheffe (1953) A Method for Judging all Contrasts in the Analysis of Variance. Biometrika 40, 87-110.

Notes

The p values are computed from the F-distribution.

References

Examples

```
>>> import scikit_posthocs as sp
>>> import pandas as pd
>>> x = pd.DataFrame({"a": [1,2,3,5,1], "b": [12,31,54,62,12], "c": [10,12,6,74,11]}
↪)
>>> x = x.melt(var_name='groups', value_name='values')
>>> sp.posthoc_scheffe(x, val_col='values', group_col='groups')
```

8.18 scikit_posthocs.posthoc_tamhane

scikit_posthocs.**posthoc_tamhane**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *welch: bool = True*, *sort: bool = False*) → DataFrame

Tamhane's T2 all-pairs comparison test for normally distributed data with unequal variances. Tamhane's T2 test can be performed for all-pairs comparisons in an one-factorial layout with normally distributed residuals but unequal groups variances. A total of m = k(k-1)/2 hypotheses can be tested. The null hypothesis is tested in the two-tailed test against the alternative hypothesis[1].

Parameters

- **a** (*Union[list, np.ndarray, DataFrame]*) – An array, any object exposing the array interface or a pandas DataFrame.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.
- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **welch** (*bool, optional*) – If True, use Welch's approximate solution for calculating the degree of freedom. T2 test uses the usual df = N - 2 approximation.
- **sort** (*bool, optional*) – If True, sort data by block and group columns.

Returns

result – P values.

Return type

pandas.DataFrame

Notes

The p values are computed from the t-distribution and adjusted according to Dunn-Sidak.

[1] A.C. Tamhane (1979), A Comparison of Procedures for Multiple Comparisons of Means with Unequal Variances. Journal of the American Statistical Association, 74, 471-480.

References

Examples

```
>>> import scikit_posthocs as sp
>>> import pandas as pd
>>> x = pd.DataFrame({"a": [1,2,3,5,1], "b": [12,31,54,62,12], "c": [10,12,6,74,11]}
↪)
>>> x = x.melt(var_name='groups', value_name='values')
>>> sp.posthoc_tamhane(x, val_col='values', group_col='groups')
```

8.19 scikit_posthocs.posthoc_tukey

scikit_posthocs.**posthoc_tukey**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *sort: bool = False*) → DataFrame

Performs Tukey's all-pairs comparisons test for normally distributed data with equal group variances. For all-pairs comparisons in an one-factorial layout with normally distributed residuals and equal variances Tukey's test can be performed. A total of m = k(k-1)/2 hypotheses can be tested. The null hypothesis is tested in the two-tailed test against the alternative hypothesis[1],[2].

Parameters

- **a** (*Union[list, np.ndarray, DataFrame]*) – An array, any object exposing the array interface or a pandas DataFrame.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.
- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **sort** (*bool, optional*) – If True, sort data by block and group columns.

Returns
result – P values.

Return type
pandas.DataFrame

Notes

The p values are computed from the Tukey-distribution.

References

Examples

```
>>> import scikit_posthocs as sp
>>> import pandas as pd
>>> x = pd.DataFrame({"a": [1,2,3,5,1], "b": [12,31,54,62,12], "c": [10,12,6,74,11]}
```

(continues on next page)

[1] L. Sachs (1997) Angewandte Statistik, New York: Springer.

[2] J. Tukey (1949) Comparing Individual Means in the Analysis of Variance, Biometrics 5, 99-114.

(continued from previous page)

```
↪)
>>> x = x.melt(var_name='groups', value_name='values')
>>> sp.posthoc_tukey(x, val_col='values', group_col='groups')
```

8.20 scikit_posthocs.posthoc_dscf

scikit_posthocs.**posthoc_dscf**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *sort: bool = False*) → DataFrame

Dwass, Steel, Critchlow and Fligner all-pairs comparison test for a one-factorial layout with non-normally distributed residuals. As opposed to the all-pairs comparison procedures that depend on Kruskal ranks, the DSCF test is basically an extension of the U-test as re-ranking is conducted for each pairwise test[1],[2],[3].

Parameters

- **a** (*Union[list, np.ndarray, DataFrame]*) – An array, any object exposing the array interface or a pandas DataFrame.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.
- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **sort** (*bool, optional*) – If True, sort data by block and group columns.

Returns

result – P values.

Return type

pandas.DataFrame

Notes

The p values are computed from the Tukey-distribution.

References

Examples

```
>>> import scikit_posthocs as sp
>>> import pandas as pd
>>> x = pd.DataFrame({"a": [1,2,3,5,1], "b": [12,31,54,62,12], "c": [10,12,6,74,11]}
↪)
>>> x = x.melt(var_name='groups', value_name='values')
>>> sp.posthoc_dscf(x, val_col='values', group_col='groups')
```

[1] Douglas, C. E., Fligner, A. M. (1991) On distribution-free multiple comparisons in the one-way analysis of variance, Communications in Statistics - Theory and Methods, 20, 127-139.

[2] Dwass, M. (1960) Some k-sample rank-order tests. In Contributions to Probability and Statistics, Edited by: I. Olkin, Stanford: Stanford University Press.

[3] Steel, R. G. D. (1960) A rank sum test for comparing all pairs of treatments, Technometrics, 2, 197-207.

8.21 scikit_posthocs.posthoc_dunnett

scikit_posthocs.**posthoc_dunnett**(*a: list | ndarray | DataFrame*, *val_col: str | None = None*, *group_col: str | None = None*, *control: str | None = None*, *alternative: Literal['two-sided', 'less', 'greater'] = 'two-sided'*, *sort: bool = False*, *to_matrix: bool = True*) → Series | DataFrame

Dunnett's test [1, 2, 3] for multiple comparisons against a control group, used after parametric ANOVA. The control group is specified by the *control* parameter.

Parameters

- **a** (*array_like or pandas DataFrame object*) – An array, any object exposing the array interface or a pandas DataFrame. Array must be two-dimensional.
- **val_col** (*str, optional*) – Name of a DataFrame column that contains dependent variable values (test or response variable). Values should have a non-nominal scale. Must be specified if *a* is a pandas DataFrame object.
- **group_col** (*str, optional*) – Name of a DataFrame column that contains independent variable values (grouping or predictor variable). Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame object.
- **control** (*str, optional*) – Name of the control group within the *group_col* column. Values should have a nominal scale (categorical). Must be specified if *a* is a pandas DataFrame.
- **alternative** (*['two-sided', 'less', or 'greater'], optional*) – Whether to get the p-value for the one-sided hypothesis ('less' or 'greater') or for the two-sided hypothesis ('two-sided'). Defaults to 'two-sided'.
- **sort** (*bool, optional*) – Specifies whether to sort DataFrame by group_col or not. Recommended unless you sort your data manually.
- **to_matrix** (*bool, optional*) – Specifies whether to return a DataFrame or a Series. If True, a DataFrame is returned with some NaN values since it's not pairwise comparison. Default is True.

Returns

result – P values.

Return type

pandas.Series or pandas.DataFrame

References

INDEX