researchpy Documentation

Release 0.1.9

Corey Bryant

Jul 10, 2020

CONTENTS:

researchpy produces Pandas DataFrames that contain relevant statistical testing information that is commonly required for academic research. The information is returned as Pandas DataFrames to make for quick and easy exporting of results to any format/method that works with the traditional Pandas DataFrame.

researchpy is essentially a wrapper that combines various established packages such as pandas, scipy.stats, and statsmodels to get all the standard required information in one method. If analyses were not available in these packages, code was developed to fill the gap.

Formula's are provided with citations if the code originated from researchpy. All output has been tested and verified by comparing to established software packages such as Stata, SAS, SPSS, and/or R.

Note: researchpy is only compatible with Python 3.x. Download using either:

- **pip install researchpy**
  - For standard install
- **conda install -c researchpy researchpy**
  - For installation through conda

CHAPTER

ONE

CODEBOOK()

Prints out descriptive information for a Pandas Series or DataFrame object.

1.1 Arguments

codebook(data)

- **data** must either be a Pandas Series or DataFrame

returns

- Information from a print()

1.2 Examples

```
import researchpy as rp
import pandas as pd
import patsy

pdf = pd.DataFrame(patsy.demo_data("y", "x", "age", "disease",
                                   nlevels=3,
                                   min_rows= 20))
rp.codebook(pdf)
```

CHAPTER

TWO

SUMMARY_CONT()

Returns a nice data table as a Pandas DataFrame that includes the variable name, total number of non-missing observations, standard deviation, standard error, and the 95% confidence interval. This is compatible with Pandas Series, DataFrame, and GroupBy objects.

2.1 Arguments

summary_cont(group1, conf = 0.95, decimals = 4)

- **group1, must either be a Pandas Series or DataFrame with multiple** columns stated
- **conf**, must be entered in decimal format. The default confidence interval being calculated is at 95%
- **decimals**, rounds the output table to the specified decimal.

returns

- Pandas DataFrame

2.2 Examples

```
import numpy, pandas, researchpy

numpy.random.seed(12345678)

df = pandas.DataFrame(numpy.random.randint(10, size= (100, 2)),
                      columns= ['healthy', 'non-healthy'])
df['tx'] = ""
df['tx'].iloc[0:50] = "Placebo"
df['tx'].iloc[50:101] = "Experimental"

df['dose'] = ""
df['dose'].iloc[0:26] = "10 mg"
df['dose'].iloc[26:51] = "25 mg"
df['dose'].iloc[51:76] = "10 mg"
df['dose'].iloc[76:101] = "25 mg"
```

```
# Summary statistics for a Series (single variable)
researchpy.summary_cont(df['healthy'])
```

```
# Summary statistics for multiple Series
researchpy.summary_cont(df[['healthy', 'non-healthy']])
```

```
# Easy to export results, assign to Python object which will have
# the Pandas DataFrame class
results = researchpy.summary_cont(df[['healthy', 'non-healthy']])

results.to_csv("results.csv", index= False)
```

```
# This works with GroupBy objects as well
researchpy.summary_cont(df['healthy'].groupby(df['tx']))
```

```
# Even with a GroupBy object with a hierarchical index
researchpy.summary_cont(df.groupby(['tx', 'dose'])['healthy', 'non-healthy'])
```

```
# Above is the default output, but if the results want to be compared
# above/below each other use .apply()

df.groupby(['tx', 'dose'])['healthy', 'non-healthy'].apply(researchpy.summary_cont)
```

CHAPTER

THREE

SUMMARY_CAT()

Returns a data table as a Pandas DataFrame that includes the counts and percentages of each category. If there are missing data present (numpy.nan), they will be excluded from the counts. However, if the missing data is coded as a string, it will be included as it's own category.

3.1 Arguments

summary_cat(group1, ascending= False)

- **group1**, can be a Pandas Series or DataFrame with multiple columns stated
- **ascending, determines the output ascending order or not. Default is** descending.

returns

- Pandas DataFrame

3.2 Examples

```
import numpy, pandas, researchpy

numpy.random.seed(123)

df = pandas.DataFrame(numpy.random.randint(2, size= (101, 2)),
                      columns= ['disease', 'treatment'])
```

```
# Handles a single Pandas Series
researchpy.summary_cat(df['disease'])
```

```
# Can handle multiple Series, although the output is not pretty
researchpy.summary_cat(df[['disease', 'treatment']])
```

```
# If missing is a string, it will show up as it's own category
df['disease'][0] = ""

researchpy.summary_cat(df['disease'])
```

```
# However, is missing is a numpy.nan, it will be excluded from the counts
df['disease'][0] = numpy.nan

researchpy.summary_cat(df['disease'])
```

```
# Results can easily be exported using many methods including the default
# Pandas exporting methods
results = researchpy.summary_cat(df['disease'])

results.to_csv("summary_cats.csv", index= False)
```

```
# This is the default, showing for comparison of immediately below
researchpy.summary_cat(df['disease'], ascending= False)
```

```
researchpy.summary_cat(df['disease'], ascending= True)
```

CHAPTER

FOUR

TTEST()

Returns data tables as Pandas DataFrames with relevant information pertaining to the statistical test conducted. Returns 2 DataFrames so all information can easily be exported, except for Wilcoxon ranked-sign test- only 1 DataFrame is returned.

DataFrame 1 (all except Wilcoxon ranked-sign test) has summary statistic information including variable name, total number of non-missing observations, standard deviation, standard error, and the 95% confidence interval. This is the same information returned from the *summary_cont()* method.

DataFrame 2 (all except Wilcoxon ranked-sign test) has the test results for the statistical tests. Included in this is an effect size measures of r, Cohen's d, Hedge's g, and Glass's Δ for the independent sample t-test, paired sample t-test, and Welch's t-test.

For the Wilcoxon ranked-sign test, the returned DataFrame contains the mean for both comparison points, the T-value, the Z-value, the two-sided p-value, and effect size measure r.

This method can perform the following tests:

- Independent sample t-test [1],
- Paired sample t-test [2],
- Welch's t-test [1], and
- Wilcoxon ranked-sign test [3]

4.1 Arguments

ttest(group1, group2, group1_name= None, group2_name= None, equal_variances= True, paired= False, correction= None)

- **group1** and **group2**, requires the data to be a Pandas Series
- **group1_name** and **group2_name**, will override the series name
- **equal_variances, tells whether equal variances is assumed or not.** If not, Welch's t-test is used if data is unpaired, or Wilcoxon rank-signed test is used if data is paired. The default is True.
- **paired, tells whether the data is paired. If data is paired and equal** variance is assumed, a paired sample t-test is conducted, otherwise a Wilcoxon ranked-sign test is conducted. The default is False.

returns

- **2 Pandas DataFrames as a tuple;**
  - First returned DataFrame is the summary statistics
  - Second returned DataFrame is the test results.

- Except for Wilcoxon ranked-sign test, only 1 DataFrame is returned

Note: Wilcoxon ranked-sign test: a 0 difference between the 2 groups is discarded from the calculation. This is the 'wilcox' method apart of scipy.stats.wilcoxon

4.1.1 Effect size measures formulas

Cohen's d_s (between subjects design)

Cohen's d_s [4] for a between groups design is calculated with the following equation:

$$d_s = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{(n_1-1)SD_1^2+(n_2-1)SD_2^2}{n_1+n_2-2}}}$$

Rosenthal [8] provided the following formula to calculate Cohen's d_s using the t-value and the number of participants from each group. This returns an identical Cohen's d_s value as the original formula.

$$d_s = t\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

Computationally speaking, the formula provided by Rosenthal is faster, therefore it is used to calculate Cohen's d_s.

Hedges's g_s (between subjects design)

Cohen's d_s gives a biased estimate of the effect size for a population and Hedges and Olkin [5] provides an unbiased estimation. The differences between Hedges's g and Cohen's d is negligible when sample sizes are above 20, but it is still preferable to report Hedges's g [6]. Hedge's g_s is calculated using the following formula:

$$\text{Hedges's } g_s = \text{Cohen's } d_s \times (1 - \frac{3}{4(n_1 + n_2 - 9)})$$

Glass's Δ (between or within subjects design)

Glass's Δ is the mean differences between the two groups divided by the standard deviation of the control group. When used in a within subjects design, it is recommended to use the pre- standard deviation in the denominator [7]; the following formula is used to calculate Glass's Δ:

$$\Delta = \frac{(\bar{x}_1 - \bar{x}_2)}{SD_1}$$

Cohen's d_z (within subject design)

Another version of Cohen's d is used in within subject designs. This is noted by the subscript "z". The formula for Cohen's d_z [4] is as follows:

$$d_z = \frac{M_{diff}}{\sqrt{\frac{\sum(X_{diff}-M_{diff})^2}{N-1}}}$$

Cohen's d_z can also be calculated with the following formula using the t-value and number of participants provided by Rosenthal [8]. This formula is used to calculate Cohen's d_z since it is computationally quicker.

$$d_z = \frac{t}{\sqrt{n}}$$

Pearson correlation coefficient r (between or within subjects design)

Rosenthal [8] provided the following formula to calculate the Pearson correlation coefficient r using the t-value and degrees of freedom:

$$r = \sqrt{\frac{t^2}{t^2 + df}}$$

Rosenthal [8] provided the following formula to calculate the Pearson correlation coefficient r using the z-value and N. This formula is used to calculate the r coefficient for the Wilcoxon ranked-sign test.

$$r = \sqrt{\frac{Z}{\sqrt{N}}}$$

4.2 Examples

```
import numpy, pandas, researchpy

numpy.random.seed(12345678)

df = pandas.DataFrame(numpy.random.randint(10, size= (100, 2)),
                      columns= ['healthy', 'non-healthy'])
```

```
# Independent t-test

# If you don't store the 2 returned DataFrames, it outputs as a tuple and
# is displayed
researchpy.ttest(df['healthy'], df['non-healthy'])
```

```
(      Variable      N   Mean        SD        SE  95% Conf.  Interval
 0      healthy  100.0  4.590  2.749086  0.274909   4.044522  5.135478
 1  non-healthy  100.0  4.160  3.132495  0.313250   3.538445  4.781555
 2     combined  200.0  4.375  2.947510  0.208420   3.964004  4.785996,
                                 Independent t-test    results
 0             Difference (healthy - non-healthy) =     0.4300
 1                          Degrees of freedom =   198.0000
 2                                           t =     1.0317
 3                       Two side test p value =     0.3035
 4                      Difference < 0 p value =     0.8483
 5                      Difference > 0 p value =     0.1517
 6                                  Cohen's d =     0.1459
 7                                  Hedge's g =     0.1454
 8                              Glass's delta =     0.1564
 9                                           r =     0.0731)
```

```
# Otherwise you can store them as objects
des, res = researchpy.ttest(df['healthy'], df['non-healthy'])

des
```

```
res
```

```
# Paired samples t-test
des, res = researchpy.ttest(df['healthy'], df['non-healthy'],
                            paired= True)

des
```

```
res
```

```
# Welch's t-test
des, res = researchpy.ttest(df['healthy'], df['non-healthy'],
                            equal_variances= False)

des
```

```
res
```

```
# Wilcoxon signed-rank test
researchpy.ttest(df['healthy'], df['non-healthy'],
                 equal_variances= False, paired= True)
```

```
# Exporting descriptive table (des) and result table (res) to same
# csv file
des, res = researchpy.ttest(df['healthy'], df['non-healthy'])

des.to_csv("C:\\Users\\...\\test.csv", index= False)
res.to_csv("C:\\Users\\...\\test.csv", index= False, mode= 'a')
```

4.3 References

CHAPTER

FIVE

CROSSTAB()

Returns up to 3 DataFrames depending on what desired. Can calculate row, column, or cell percentages if requested. Otherwise, counts are returned as the default.

DataFrame 1 is always the crosstabulation results, the other 2 DataFrames returned depends on the options selected which is determined by the arguments *test* and *expected_freqs*. If all 3 options are returned, then the order of the returned DataFrames is as follows: crosstabulation results, χ^2 test results with effect size of Cramer's Phi or V depending on size of table, and the expected frequency table.

5.1 Arguments

crosstab(group1, group2, prop= None, test = False, margins= True, correction = None, cramer_correction = None, exact = False, expected_freqs= False)

- **group1** and **group2**, requires the data to be a Pandas Series
- **prop**, can either be 'row', 'col', or 'cell'. 'row' will calculate the row percentages, 'column' will calculate the column percentages, and 'cell' will calculate the cell percentage based on the entire sample
- **test, can take "chi-square", "g-test", "mcnemar", or "fisher".**
  - If "chi-square", the chi-square (χ^2) test of independence [1] will be calculated and returned in a second DataFrame.
  - If "g-test", will conduct the G-test (likelihood-ratio χ^2) [1] and the results will be returned in a second DataFrame.
  - If "fisher", will conduct Fisher's exact test [2].
  - If "mcnemar", will conduct the McNemar χ^2 [3] test for paired nominal data.
- **margins**, if False will return a crosstabulation table without the total counts for each group. This argument is only supported for counts; the margins will always be returned for the percentages
- **correction**, if True, applies the Yates' correction for continuity. Valid argument for *chi-square"*, *"g-test"*, and *"mcnemar"*.
- **cramer_correction**, if True, applies the bias correction developed by Tschuprow (1925) to Cramer's V.
- **exact**, is only a valid option for when the *"mcnemar"* test is selected. In that case, *exact = True* will then the binomal distribution will be used. If false (default), the χ^2 distribution is used.
- **expected_freqs**, if True, will return a DataFrame that contains the expected counts for each cell. Not a valid argurment for *mcnemar* test.

returns

- **Up to 3 Pandas DataFrames as a tuple;**

- First DataFrame is always the crosstab table with either the counts, cell, row, or column percentages
- Second DataFrame is either the test results or the expected frequencies. If a test is selected and expected frequencies are desired, the second DataFrame will be the test results; otherwise, if just expected frequencies are desired, the second DataFrame will be that and there will not be a third DataFrame returned.
- Third DataFrame is always the expected frequencies

Note: If conducting a McNemar test, make sure the outcomes in both variables are labelled the same.

5.2 Effect size measures formulas

Note: If adjusted χ^2 values are used in the test's calculation, then those adjusted χ^2 values are also used to calculate effect size.

5.2.1 Cramer's Phi (2x2 table)

For analyses were it's a 2x2 table, the following formula is used to calculate Cramer's Phi (ϕ) [4]:

$$\phi = \sqrt{\frac{\chi^2}{N}}$$

Where N = total number of observations in the analysis

5.2.2 Cramer's V (RxC where R or C > 2)

For analyses were it's a table that is larger than a 2x2, the following formula is used to calculate Cramer's V [4]:

$$V = \sqrt{\frac{\chi^2}{(N * (k-1))}}$$

Where K is the number of categories for either R or C (whichever has fewer categories)

$$\tilde{V} = \sqrt{\frac{\tilde{\phi}^2}{\min(\tilde{r}-1, \tilde{c}-1)}}$$

Where r is the number of rows and c is the number of columns, and

$$\tilde{\phi}^2 = \max(0, \frac{\chi^2}{n} - \frac{(c-1)(r-1)}{n-1})$$

$$\tilde{c} = c - \frac{(c-1)^2}{n-1}$$

$$\tilde{r} = r - \frac{(r-1)^2}{n-1}$$

5.3 Examples

```
import researchpy, pandas, numpy

numpy.random.seed(123)

df = pandas.DataFrame(numpy.random.randint(3, size= (101, 3)),
                      columns= ['disease', 'severity', 'alive'])

df.head()
```

```
# If only two Series are passed it will output a crosstabulation with margin totals.
# This is the same as pandas.crosstab(), except for researchpy.crosstab() returns
# a table with hierarchical indexing for better exporting format style.

researchpy.crosstab(df['disease'], df['alive'])
```

```
# Demonstration of calculating cell proportions

crosstab = researchpy.crosstab(df['disease'], df['alive'], prop= "cell")

crosstab
```

```
# Demonstration of calculating row proportions

crosstab = researchpy.crosstab(df['disease'], df['alive'], prop= "row")

crosstab
```

```
# Demonstration of calculating column proportions

crosstab = researchpy.crosstab(df['disease'], df['alive'], prop= "col")

crosstab
```

```
# To conduct a Chi-square test of independence, pass "chi-square" in the "test ="␣
↪argument.
# This will also output an effect size; either Cramer's Phi if it a 2x2 table, or
# Cramer's V is larger than 2x2.

# This will return 2 DataFrames as a tuple, 1 with the crosstabulation and the other␣
↪with the
# test results. It's rather ugly, the recommended way to output is in the next example

researchpy.crosstab(df['disease'], df['alive'], test= "chi-square")
```

```
(        alive
             0   1   2  All
 disease
 0           9  14   7   30
 1           7   9  15   31
 2           7  17  16   40
 All        23  40  38  101,                Chi-square test  results
 0  Pearson Chi-square ( 4.0) =    5.1573
```

(continues on next page)

(continued from previous page)

```
 1                    p-value =     0.2715
 2                  Cramer's V =     0.3196)
```

```
# To clean up the output, assign each DataFrame to an object. This allows
# for a cleaner view and each DataFrame to be exported

crosstab, res = researchpy.crosstab(df['disease'], df['alive'], test= "chi-square")

crosstab
```

```
res
```

```
# To get the expected frequencies, pass "True" in "expected_freqs="

crosstab, res, expected = researchpy.crosstab(df['disease'], df['alive'], test= "chi-
↪square", expected_freqs= True)

expected
```

```
# Can also conduct the G-test (likelihood-ratio chi-square)

crosstab, res = researchpy.crosstab(df['disease'], df['alive'], test= "g-test")

res
```

```
# Can also conduct Fisher's exact test

# Need 2x2 data for Fisher's test.
numpy.random.seed(345)

df = pandas.DataFrame(numpy.random.randint(2, size= (90, 2)),
                      columns= ['tx', 'cured'])

crosstab, res = researchpy.crosstab(df['tx'], df['cured'], test= "fisher")

crosstab
```

```
res
```

```
# Lastly, the McNemar test
# Make sure your outcomes are labelled the same in
# both variables
numpy.random.seed(345)

df = pandas.DataFrame(numpy.random.randint(2, size= (90, 2)),
                      columns= ['time1', 'time2'])

crosstab, res = researchpy.crosstab(df['time1'], df['time2'], test= "mcnemar")

crosstab
```

```
res
```

5.4 References

CHAPTER

SIX

CORR_CASE()

Conducts Pearson (default method), Spearman rank, or Kendall's Tau-b correlation analysis using case wise deletion. Returns the relevant information and results in 3 DataFrames for easy exporting.

DataFrame 1 is the testing information, i.e. the type of correlation analysis conducted and the number of observations used.

DataFrame 2 contains the r value results in a matrix style look.

DataFrame 3 contains the p-values in a matrix style look.

6.1 Arguments

def corr_case(dataframe, method = "pearson")

- **dataframe** can either be a single Pandas Series or multiple Series/an entire DataFrame.
- **method** takes the values of "pearson" [2] (the default if nothing is passed), "spearman" [3], or "kendall" [1].

6.2 Examples

```
import researchpy, numpy, pandas


numpy.random.seed(12345)

df = pandas.DataFrame(numpy.random.randint(10, size= (100, 2)),
                      columns= ['beck', 'srq'])
```

```
# Since it returns 3 DataFrames for easy exporting, if the DataFrames
# aren't assigned to an object the outputting tuple is rather messy
# and ugly

researchpy.correlation.corr_case(df[['beck', 'srq']])
```

```
(  Pearson correlation test using list-wise deletion
 0                     Total observations used = 100,         beck     srq
 beck       1  0.0029
 srq   0.0029       1,         beck     srq
 beck  0.0000  0.9775
 srq   0.9775  0.0000)
```

```
# As noted above, the 3 return DataFrames are information, r values,
# and p-values
# The 3 DataFrame design was decided on so each DataFrame can easily
# be exported using already supported Pandas methods

i, r, p  = researchpy.correlation.corr_case(df[['beck', 'srq']])

i
```

Pearson correlation test using list-wise deletion
Total observations used = 100

```
r
```

	beck	srq
beck	1	0.0029
srq	0.0029	1

```
p
```

	beck	srq
beck	0.0000	0.9775
srq	0.9775	0.0000

6.3 References

CHAPTER

SEVEN

CORR_PAIR()

Conducts Pearson (default method), Spearman rank, or Kendall's Tau-b correlation analysis using pair wise deletion. Returns the relevant information and results in 1 DataFrame for easy exporting.

DataFrame 1 contains the variables being compared in the index, followed by the corresponding r value, p-value, and N for the groups being compared.

7.1 Arguments

corr_pair(dataframe, method= "pearson")

- **dataframe** can either be a single Pandas Series or multiple Series/an entire DataFrame.
- **method** takes the values of "pearson" [2] (the default if nothing is passed), "spearman" [3], or "kendall" [1].

7.2 Examples

```
import researchpy, numpy, pandas


numpy.random.seed(12345)

df = pandas.DataFrame(numpy.random.randint(10, size= (100, 4)),
                      columns= ['mental_score', 'physical_score', 'emotional_score',
                                'happiness_index'])
```

```
# Can pass the entire DataFrame or multiple Series

researchpy.correlation.corr_pair(df)
```

```
# Demonstrating how the output looks if there are different Ns for groups
df['happiness_index'][0:30] = numpy.nan

researchpy.correlation.corr_pair(df)
```

7.3 References

CHAPTER

EIGHT

INSTALLING RESEARCHPY

researchpy is available via pip and through conda.

To install using pip use:

- pip install researchpy

To install using conda use:

- conda install -c researchpy researchpy

BIBLIOGRAPHY

[1] *scipy.stats.ttest_ind*. The SciPy community, 2018. Retrieved when last updated on May 5, 2018. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html.

[2] *scipy.stats.ttest_rel*. The SciPy community, 2018. Retrieved when last updated on May 5, 2018. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_rel.html.

[3] *scipy.stats.wilcoxon*. The SciPy community, 2018. Retrieved when last updated on May 5, 2018. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html.

[4] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, second edition, 1988. ISBN 0-8058-0283-5.

[5] Larry Hedges and Ingram Olkin. *Journal of Educational Statistics*, chapter Statistical Methods in Meta-Analysis. Volume 20. Academic Press, Inc., 1985, 10.2307/1164953.

[6] Rex B. Kline. *Beyond significance testing: Reforming data analysis methods in behavioral research*. American Psychological Association, 2004. http://dx.doi.org/10.1037/10693-000.

[7] Daniel Lakens. Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and anovas. *Frontiers in Psychology*, November 2013. doi:10.3389/fpsyg.2013.00863.

[8] Robert Rosenthal. *The hand-book of research synthesis*, chapter Parametric measures of effect size, pages 231–244. New York, NY: Russel Sage Foundation, 1994.

[1] *scipy.stats.chi2_contingency*. The Scipy community, 2016. Retrived when last updated May 12, 2016. URL: http://lagrange.univ-lyon1.fr/docs/scipy/0.17.1/generated/scipy.stats.chi2_contingency.html.

[2] *scipy.stats.fisher_exact*. The SciPy community, 2018. Retrieved when last updated on May 5, 2018. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.fisher_exact.html.

[3] *statsmodels.stats.contingency_tables.mcnemar*. Statsmodels-developers, 2018. URL: https://www.statsmodels.org/dev/generated/statsmodels.stats.contingency_tables.mcnemar.html.

[4] Harald Cramér. *Mathematical methods of statistics (PMS-9)*. Volume 9. Princeton university press, 2016.

[1] *scipy.stats.kendalltau*. 2018. Retrieved when last updated on May 5, 2018. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kendalltau.html.

[2] *scipy.stats.pearsonr*. The SciPy community, 2018. Retrieved when last updated on May 11, 2014. URL: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.pearsonr.html.

[3] *scipy.stats.spearmanr*. The SciPy community, 2018. Retrieved when last updated on May 11, 2014. URL: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.spearmanr.html.

[1] *scipy.stats.kendalltau*. 2018. Retrieved when last updated on May 5, 2018. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kendalltau.html.

[2] *scipy.stats.pearsonr*. The SciPy community, 2018. Retrieved when last updated on May 11, 2014. URL: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.pearsonr.html.

[3] *scipy.stats.spearmanr*. The SciPy community, 2018. Retrieved when last updated on May 11, 2014. URL: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.spearmanr.html.