Bentley University MA214 in Python

Content extracted from the How to Data Website

This PDF generated on 05 November 2022

Contents

MA214 is an undergraduate statistics course at Bentley University that builds on the basic managerial statistics course taken by all students. The description from the course catalog can be found here.

It covers hypothesis tests, analysis of variance, multiple regression, and contingency tables.

Review of statistical inference

- How to compute a confidence interval for a population mean
- How to compute a confidence interval for a population mean using z-scores
- How to compute a confidence interval for the population proportion
- How to do a hypothesis test for a mean difference (matched pairs)
- How to do a hypothesis test for a population proportion
- How to do a hypothesis test for population variance
- How to do a hypothesis test for the mean with known standard deviation

Two populations

- How to compute a confidence interval for a mean difference (matched pairs)
- How to choose the sample size in a study with two population means
- How to compute a confidence interval for the difference between two means when both population variances are known
- How to compute a confidence interval for the difference between two means when population variances are unknown
- How to compute a confidence interval for the difference between two proportions
- How to compute a confidence interval for the ratio of two population variances
- How to do a hypothesis test for the difference between means when both population variances are known
- How to do a hypothesis test for the difference between two proportions
- How to do a hypothesis test for the ratio of two population variances
- How to do a Kruskal-Wallis test
- How to do a one-sided hypothesis test for two sample means
- How to do a Wilcoxon rank-sum test
- How to do a Wilcoxon signed-rank test
- How to do a Wilcoxon signed-rank test for matched pairs

Variance inference

- How to compute a confidence interval for a single population variance

Chi-squares tests

- How to perform a chi-squared test on a contingency table
- How to do a goodness of fit test for a multinomial experiment

ANOVA

- How to do a one-way analysis of variance (ANOVA)
- How to do a two-way ANOVA test with interaction
- How to do a two-way ANOVA test without interaction
- How to compute Fisher's confidence intervals
- How to perform an analysis of covariance (ANCOVA)
- How to perform post-hoc analysis with Tukey's HSD test
- How to use Bonferroni's Correction method

Regression

- How to fit a linear model to two columns of data
- How to compute a confidence interval for the expected value of a response variable
- How to compute R-squared for a simple linear model
- How to predict the response variable in a linear model

Nonparametric tests

- How to create a QQ-plot
- How to test data for normality with Pearson's chi-squared test
- How to test data for normality with the D'Agostino-Pearson test
- How to test data for normality with the Jarque-Bera test

Content last modified on 07 December 2021.

How to compute a confidence interval for a population mean

Description

If we have a set of data that seems normally distributed, how can we compute a confidence interval for the mean? Assume we have some confidence level already chosen, such as $\alpha = 0.05$.

We will use the t-distribution because we have not assumed that we know the population standard deviation, and we have not assumed anything about our sample size. If you know the population standard deviation or have a large sample size (typically at least 30), then you can use z-scores instead; see how to compute a confidence interval for a population mean using z-scores.

Related tasks:

- How to compute a confidence interval for a population mean using z-scores
- How to do a two-sided hypothesis test for a sample mean (on website)
- How to do a two-sided hypothesis test for two sample means (on website)
- How to compute a confidence interval for a mean difference (matched pairs)
- How to compute a confidence interval for a regression coefficient (on website)
- How to compute a confidence interval for a single population variance
- How to compute a confidence interval for the difference between two means when both population variances are known
- How to compute a confidence interval for the difference between two means when population variances are unknown
- How to compute a confidence interval for the difference between two proportions
- How to compute a confidence interval for the expected value of a response variable
- How to compute a confidence interval for the population proportion
- How to compute a confidence interval for the ratio of two population variances

Solution in Python

This solution uses a 95% confidence level, but you can change that in the first line of code, by specifing a different `alpha`.

When applying this technique, you would have a series of data values for which you needed to compute a confidence interval for the mean. But in order to provide code that runs independently, we create some fake data below. When using this code, replace our fake data with your real data.

```
alpha = 0.05       # replace with your chosen alpha (here, a 95% confidence level)
data = [ 435,542,435,4,54,43,5,43,543,5,432,43,36,7,876,65,5 ] # fake

# We will use NumPy and SciPy to compute some of the statistics below.
import numpy as np
import scipy.stats as stats

# Compute the sample mean, as an estimate for the population mean.
sample_mean = np.mean( data )

# Compute the Standard Error for the sample Mean (SEM).
sem = stats.sem( data )

# The margin of error then has the following formula.
moe = sem * stats.t.ppf( 1 - alpha / 2, len( data ) - 1 )

# The confidence interval is centered on the mean with moe as its radius:
( sample_mean - moe, sample_mean + moe )
```

```
(70.29847811072423, 350.0544630657464)
```

Note: The solution above assumes that the population is normally distributed, which is a common assumption in introductory statistics courses, but we have not verified that assumption here.

Content last modified on 30 November 2021.

See a problem? Tell us or edit the source.

How to compute a confidence interval for a population mean using z-scores

Description

If we have a set of data that seems normally distributed, how can we compute a confidence interval for the mean? Assume we have some confidence level already chosen, such as $\alpha = 0.05$.

We will use the normal distribution, which assumes either that we know the population standard deviation, or we have a large enough sample size (typically at least 30). If neither of these is true in your case, then you can use t-scores instead; see how to compute a confidence interval for a population mean.

Related tasks:

- How to compute a confidence interval for a population mean
- How to do a two-sided hypothesis test for a population mean
- How to do a two-sided hypothesis test for two sample means (on website)
- How to compute a confidence interval for a mean difference (matched pairs)
- How to compute a confidence interval for a regression coefficient (on website)
- How to compute a confidence interval for a single population variance
- How to compute a confidence interval for the difference between two means when both population variances are known
- How to compute a confidence interval for the difference between two means when population variances are unknown
- How to compute a confidence interval for the difference between two proportions
- How to compute a confidence interval for the expected value of a response variable
- How to compute a confidence interval for the population proportion
- How to compute a confidence interval for the ratio of two population variances

Solution in Python

This solution uses a 95% confidence level, but you can change that in the first line of code, by specifing a different `alpha`.

When applying this technique, you would have a series of data values for which you needed to compute a confidence interval for the mean. But in order to provide code that runs independently, we create some fake data below. When using this code, replace our fake data with your real data.

We include the population standard deviation below, assuming it is known. See the notes at the end for what to do if you do not know the population standard deviation in your situation.

```
alpha = 0.05       # replace with your chosen alpha (here, a 95% confidence level)
pop_std = 250      # replace with your population standard devation, if known
data = [ 435,542,435,4,54,43,5,43,543,5,432,43,36,7,876,65,5 ] # fake

# We will use NumPy and SciPy to compute some of the statistics below.
import numpy as np
import scipy.stats as stats

# Compute the sample mean, as an estimate for the population mean.
sample_mean = np.mean( data )

# The margin of error then has the following formula.
z_score = stats.norm.ppf( 1 - alpha / 2 )
moe = pop_std * z_score / np.sqrt( len( data ) )

# The confidence interval is centered on the mean with moe as its radius:
( sample_mean - moe, sample_mean + moe )
```

```
(91.33619807845439, 329.0167430980162)
```

Notes:

1. If you do not have the population standard deviation, but your sample is large enough (typically at least 30), you can approximate the population standard deviation with the sample standard deviation, using the code `std = stats.tstd( data )`. If your sample is not that large, then consider using a different technique instead; see how to compute a confidence interval for a population mean.
2. The solution above assumes that the population is normally distributed, which is a common assumption in introductory statistics courses, but we have not verified that assumption here.

Content last modified on 30 November 2021.

See a problem? Tell us or edit the source.

How to compute a confidence interval for the population proportion

Description

If we have a sample of qualitative data from a normally distributed population, then how do we compute a confidence interval for a population proportion?

Related tasks:

- How to compute a confidence interval for a mean difference (matched pairs)
- How to compute a confidence interval for a regression coefficient (on website)
- How to compute a confidence interval for a population mean
- How to compute a confidence interval for a single population variance
- How to compute a confidence interval for the difference between two means when both population variances are known
- How to compute a confidence interval for the difference between two means when population variances are unknown
- How to compute a confidence interval for the difference between two proportions
- How to compute a confidence interval for the expected value of a response variable
- How to compute a confidence interval for the ratio of two population variances

Solution in Python

We're going to use some fake data here for illustrative purposes, but you can replace our fake data with your real data in the code below.

```
# Replace the next two lines of code with your real data
sample_size = 30
sample_proportion = .39

# Find the margin of error
from scipy import stats
import numpy as np
alpha = 0.05       # replace with your chosen alpha (here, a 95% confidence level)
moe = stats.norm.ppf(1-alpha/2) * np.sqrt(sample_proportion*(1-sample_proportion)/sample_size)

# Find the confidence interval
lower_bound = sample_proportion - moe
upper_bound = sample_proportion + moe
lower_bound, upper_bound
```

```
(0.21546413420702207, 0.564535865792978)
```

Our 95% confidence interval is $[0.2155, 0.5645]$.

Content last modified on 09 September 2021.

See a problem? Tell us or edit the source.

How to do a hypothesis test for a mean difference (matched pairs)

Description

Say we have two sets of data that are not independent of each other and come from a matched-pairs experiment, $(x_1, x_1'), (x_2, x_2'), \ldots, (x_n, x_n')$. We want to perform inference on the mean of the differences between these two samples, that is, the mean of $x_1 - x_1', x_2 - x_2', \ldots, x_n - x_n'$, called μ_D. We want to determine if it is significantly different from, greater than, or less than zero (or any other hypothesized value). We can do so with a two-tailed, right-tailed, or left-tailed hypothesis test for matched pairs.

Related tasks:

- How to compute a confidence interval for a mean difference (matched pairs)
- How to do a hypothesis test for a mean difference (matched pairs)
- How to do a hypothesis test for a population proportion
- How to do a hypothesis test for population variance
- How to do a hypothesis test for the difference between means when both population variances are known
- How to do a hypothesis test for the difference between two proportions
- How to do a hypothesis test for the mean with known standard deviation
- How to do a hypothesis test for the ratio of two population variances
- How to do a hypothesis test of a coefficient's significance (on website)
- How to do a one-sided hypothesis test for two sample means
- How to do a two-sided hypothesis test for a sample mean (on website)
- How to do a two-sided hypothesis test for two sample means (on website)

Solution in Python

We choose a value, $0 \leq \alpha \leq 1$, as the Type I Error rate, and in this case we will set it to be 0.05.

We're going to use fake fata here, but you can replace our fake data with your real data below. Because the data are matched pairs, the samples must be the same size.

```
# Replace the following example data with your real data
sample1 = [15, 10,  7, 22, 17, 14]
sample2 = [ 9,  1, 11, 13,  3,  6]
```

Two-tailed test

In a two-sided hypothesis test, the null hypothesis states that the mean difference is equal to 0 (or some other hypothesized value), $H_0 : \mu_D = 0$.

```
from scipy import stats
stats.ttest_rel(sample1, sample2, alternative = "two-sided")
```

```
Ttest_relResult(statistic=2.8577380332470415, pvalue=0.03550038112896236)
```

Our p-value, 0.0355, is smaller than α, so we have sufficient evidence to reject the null hypothesis and conclude that the mean difference between the two samples is significantly different from zero.

Note that the function above specifically tests whether the mean of $x_i - x_i'$ is zero. If we want instead to test whether it is some other value $d \neq 0$, then that's equivalent to testing whether the mean of $(x_i - d) - x_i'$ is zero. We could do so with the code below, which uses an example value of d. The null hypothesis is now $H_0 : \mu_D = d$.

```
d = 6  # as an example
stats.ttest_rel([ x - d for x in sample1 ], sample2, alternative = "two-sided")
```

```
Ttest_relResult(statistic=0.4082482904638631, pvalue=0.6999865427788738)
```

The above p-value is greater than $\alpha = 0.05$, so we could not conclude that the mean difference is significantly different from our chosen $d = 6$.

Right-tailed test

If instead we want to test whether the mean difference is less than or equal to zero, $H_0 : \mu_D \leq 0$, we can use a right-tailed test, as follows.

```
stats.ttest_rel(sample1, sample2, alternative = "greater")
```

```
Ttest_relResult(statistic=2.8577380332470415, pvalue=0.01775019056448118)
```

Our p-value, 0.01775, is smaller than α, so we have sufficient evidence to reject the null hypothesis and conclude that the mean difference between the two samples is significantly greater than zero.

A similar change could be made to the code above to test $H_0 : \mu_D \leq d$, as in the example code further above that uses $d = 6$.

Left-tailed test

If instead we want to test whether the mean difference is greater than or equal to zero, $H_0 : \mu_D \geq 0$, we can use a right-tailed test, as follows.

```
stats.ttest_rel(sample1, sample2, alternative = "less")
```

```
Ttest_relResult(statistic=2.8577380332470415, pvalue=0.9822498094355188)
```

Our p-value, 0.98225, is larger than α, so we do not have sufficient evidence to reject the null hypothesis; we must continue to assume that the mean difference between the two samples is greater than or equal to zero.

A similar change could be made to the code above to test $H_0 : \mu_D \geq d$, as in the example code further above that uses $d = 6$.

Content last modified on 07 December 2021.

See a problem? Tell us or edit the source.

How to do a hypothesis test for a population proportion

Description

When we have qualitative data, we're often interested in performing inference on population proportions. That is, the proportion (between 0.0 and 1.0) of the population that is in a certain category with respect to the qualitative variables. Given a sample proportion, $\bar{p}$, how can we test whether the population proportion is equal to, greater than, or less than some hypothesized value?

Related tasks:

- How to compute a confidence interval for the population proportion
- How to do a hypothesis test for a mean difference (matched pairs)
- How to do a hypothesis test for the difference between means when both population variances are known
- How to do a hypothesis test for the difference between two proportions
- How to do a hypothesis test for the mean with known standard deviation
- How to do a hypothesis test for the ratio of two population variances
- How to do a hypothesis test of a coefficient's significance (on website)
- How to do a one-sided hypothesis test for two sample means
- How to do a two-sided hypothesis test for a sample mean (on website)
- How to do a two-sided hypothesis test for two sample means (on website)

Solution in Python

We're going to use fake data here for illustrative purposes, but you can replace our fake data with your real data in the code below.

Let's say that we've hypothesized that about one-third of Bostonians are unhappy with the Red Sox' performance. To test this hypothesis, we surveyed 460 Bostonians and found that 76 of them were unhappy with the Red Sox' performance.

We summarize this situation with the following variables. We will do a test with a Type I error rate of $\alpha = 0.05$.

```
n = 460               # Number of respondents in sample
x = 76                # Number of respondents in chosen subset
sample_prop = x/n     # Proportion of sample in chosen subset
population_prop = 1/3 # Hypothesized population proportion
```

Two-tailed test

A two-tailed test is for the null hypothesis $H_0 : p = \frac{1}{3}$. It can be done by directly computing the test statistic and p-value using tools from SciPy's `stats` module.

```
import numpy as np
from scipy import stats
test_stat = ( (sample_prop - population_prop) /
    np.sqrt(population_prop*(1 - population_prop)/n) )
stats.norm.sf(abs(test_stat))*2  # p-value
```

```
2.0284218907806657e-14
```

The p-value is less than α, so we can reject the null hypothesis. The proportion of Bostonians unhappy with Red Sox performance is different from $\frac{1}{3}$.

Right-tailed test

A right-tailed test is for the null hypothesis $H_0 : p \leq \frac{1}{3}$. Most of the code is the same as above, but the p-value is computed differently for a one-sided test. We repeat the re-used code to make it easy to copy and paste.

```
import numpy as np
from scipy import stats
test_stat = ( (sample_prop - population_prop) /
    np.sqrt(population_prop*(1 - population_prop)/n) )
stats.norm.sf(test_stat)
```

```
0.9999999999999899
```

The p-value is greater than α, so we cannot reject the null hypothesis. We continue to assume that the proportion of Bostonians unhappy with Red Sox performance is less than or equal to $\frac{1}{3}$.

Left-tailed test

A left-tailed test is for the null hypothesis $H_0 : p \geq \frac{1}{3}$. Most of the code is the same as above, but the p-value is computed differently yet again. We repeat the re-used code to make it easy to copy and paste.

```
import numpy as np
from scipy import stats
test_stat = ( (sample_prop - population_prop) /
    np.sqrt(population_prop*(1 - population_prop)/n) )
stats.norm.sf(abs(test_stat))
```

```
1.0142109453903328e-14
```

The p-value is less than α, so we can reject the null hypothesis. The proportion of Bostonians unhappy with Red Sox performance is less than $\frac{1}{3}$.

Content last modified on 24 October 2021.

See a problem? Tell us or edit the source.

How to do a hypothesis test for population variance

Description

Assume we want to estimate the variability of a quantity across a population, starting from a sample of data, $x_1, x_2, x_3, ... x_k$. How might we test whether the population variance is equal to, greater than, or less than a hypothesized value?

Related tasks:

- How to compute a confidence interval for the population proportion
- How to do a hypothesis test for a mean difference (matched pairs)
- How to do a hypothesis test for a population proportion
- How to do a hypothesis test for the difference between means when both population variances are known
- How to do a hypothesis test for the difference between two proportions
- How to do a hypothesis test for the mean with known standard deviation
- How to do a hypothesis test for the ratio of two population variances
- How to do a hypothesis test of a coefficient's significance (on website)
- How to do a one-sided hypothesis test for two sample means
- How to do a two-sided hypothesis test for a sample mean (on website)
- How to do a two-sided hypothesis test for two sample means (on website)

Solution in Python

How to Data does not yet contain a solution for this task in Python.

How to do a hypothesis test for the mean with known standard deviation

Description

Let's say we are measuring a variable over a population, and we know its standard deviation σ is known, and assume that the variable is normally distributed. We take a sample, $x_1, x_2, x_3, \ldots, x_k$, and compute its mean $\bar{x}$. We want to determine if the sample mean is significantly different from, greater than, or less than some hypothesized value, such as a hypothesized population mean. How do we formulate an appropriate hypothesis test?

Related tasks:

- How to compute a confidence interval for a population mean
- How to do a hypothesis test for a mean difference (matched pairs)
- How to do a hypothesis test for a population proportion
- How to do a hypothesis test for population variance
- How to do a hypothesis test for the difference between means when both population variances are known
- How to do a hypothesis test for the difference between two proportions
- How to do a hypothesis test for the ratio of two population variances
- How to do a hypothesis test of a coefficient's significance (on website)
- How to do a one-sided hypothesis test for two sample means
- How to do a two-sided hypothesis test for a sample mean (on website)
- How to do a two-sided hypothesis test for two sample means (on website)

Solution in Python

We will use the following fake data, but you can insert your actual data in its place. We have a sample of just 5 values and an assumed population standard deviation of 3.

```
sample = [31, 44, 28, 25, 40]  # sample data
pop_std = 3                    # population standard deviation
```

We also choose a value $0 \leq \alpha \leq 1$ as our Type I error rate. We'll let α be 0.05 here, but you can change that in the code below.

Two-tailed test

In a two-tailed test, we compare the unknown population mean to a hypothesized value m using the null hypothesis $H_0 : \mu = m$. Here we'll use $m = 30$, but you can replace that value in the code below with the hypothesis relevant for your situation.

```
from scipy import stats
import numpy as np
m = 30                                          # hypothesized mean
n = len(sample)                                 # number of observations
xbar = np.mean(sample)                          # sample mean
test_stat = (xbar - m) / (pop_std/np.sqrt(n))  # test statistic
2*stats.norm.sf(abs(test_stat))                 # two-tailed p-value
```

```
0.007290358091535614
```

The p-value, 0.00729, is less than α, so we have evidence to reject the null hypothesis and conclude that the actual population mean μ is significantly different from the hypothesized value of $m = 30$.

Right-tailed test

In a right-tailed hypothesis test, the null hypothesis is that the population mean is greater than or equal to a chosen value, $H_0 : \mu \geq m$.

Most of the code below is the same as above, but we repeat it to make it easy to copy and paste. Only the computation of the p-value changes.

```
from scipy import stats
import numpy as np
m = 30                                          # hypothesized mean
n = len(sample)                                 # number of observations
xbar = np.mean(sample)                          # sample mean
test_stat = (xbar - m) / (pop_std/np.sqrt(n))  # test statistic
stats.norm.sf(abs(test_stat))                   # right-tailed p-value
```

```
0.003645179045767807
```

The p-value, 0.003645, is less than α, so we have evidence to reject the null hypothesis and conclude that the actual population mean μ is significantly less than the hypothesized value of $m = 30$.

Left-tailed test

In a left-tailed hypothesis test, the null hypothesis is that the population mean is less than or equal to a chosen value, $H_0 : \mu \leq m$.

Most of the code below is the same as above, but we repeat it to make it easy to copy and paste. Only the computation of the p-value changes.

```
from scipy import stats
import numpy as np
m = 30                                          # hypothesized mean
n = len(sample)                                 # number of observations
xbar = np.mean(sample)                          # sample mean
test_stat = (xbar - m) / (pop_std/np.sqrt(n))  # test statistic
stats.norm.sf(-abs(test_stat))                  # left-tailed p-value
```

```
0.9963548209542322
```

The p-value, 0.99635, is greater than α, so wwe do not have sufficient evidence to conclude that $\mu > m$ and should continue to accept the null hypothesis, that $\mu \leq m$.

Content last modified on 24 October 2021.

See a problem? Tell us or edit the source.

How to compute a confidence interval for a mean difference (matched pairs)

Description

Say we have two sets of data that are not independent of each other and come from a matched-pairs experiment, and we want to construct a confidence interval for the mean difference between these two samples. How do we make this confidence interval? Let's assume we've chosen a confidence level of $\alpha = 0.05$.

Related tasks:

- How to do a hypothesis test for a mean difference (matched pairs)
- How to compute a confidence interval for a regression coefficient (on website)
- How to compute a confidence interval for a population mean
- How to compute a confidence interval for a single population variance
- How to compute a confidence interval for the difference between two means when both population variances are known
- How to compute a confidence interval for the difference between two means when population variances are unknown
- How to compute a confidence interval for the difference between two proportions
- How to compute a confidence interval for the expected value of a response variable
- How to compute a confidence interval for the population proportion
- How to compute a confidence interval for the ratio of two population variances

Solution in Python

We'll use Numpy and SciPy to do some statistics later.

```
import numpy as np
from scipy import stats
```

This example computes a 95% confidence interval, but you can choose a different level by choosing a different value for α.

```
alpha = 0.05
```

We have two samples of data, $x_1, x_2, x_3, \ldots, x_k$ and $x'_1, x'_2, x'_3, \ldots, x'_k$. We're going to use some fake data below just as an example; replace it with your real data.

```
sample1 = np.array([15, 10,  7, 22, 17, 14])
sample2 = np.array([ 9,  1, 11, 13,  3,  6])
```

And now the computations:

```
diff_samples = sample1 - sample2                          # differences between the samples
n = len(sample1)                                          # number of observations per sample
diff_mean = np.mean(diff_samples)                         # mean of the differences
diff_variance = np.var( diff_samples, ddof=1 )            # variance of the differences
critical_val = stats.t.ppf(q = 1-alpha/2, df = n - 1)     # critical value
radius = critical_val*np.sqrt(diff_variance)/np.sqrt(n) # radius of confidence interval
( diff_mean - radius, diff_mean + radius )                # confidence interval
```

```
(0.7033861582274517, 13.296613841772547)
```

Our 95% confidence interval for the mean difference is $[0.70338, 13.2966]$.

Content last modified on 07 December 2021.

See a problem? Tell us or edit the source.

How to choose the sample size in a study with two population means

Description

When designing a study, it is important to choose a sample size that is large enough to perform a useful test but that is also economically feasible. How we choose the sample size depends on what test we plan to run on the data from our study. Here, let's say our data will be used to compare two population means. If we are planning such a study, how do we determine how large it should be in order for the test that compares the population means to have a certain power?

Related tasks:

- How to compute the power of a test comparing two population means (on website)

Solution in Python

Example: Let's say we're designing a study to assess the effectiveness of a new four-week exercise program for weight loss. Assume that weight loss in four-week exercise programs is normally distributed with a standard deviation of around 5 pounds. The goal is that the new exercise program will have a 4-pound higher weight loss than the average program. (Notice that we will be comparing the means of two populations, the weight loss in each of two programs.)

We choose a value $0 \leq \alpha \leq 1$ as the probability of a Type I error in our test that compares the two means. (Recall, Type I error is for a false positive, finding we should reject H_0 when it's actually true). Let's set α to be 0.05 here.

We choose a value $0 \leq \beta \leq 1$ as the probability of a Type II error (false negative, failing to reject H_0 when it's actually false). Let's set β to be 0.2 here. The test's power is $1 - \beta$, or in this case, 0.8.

What should the sample size be for each group?

```
from statsmodels.stats.power import TTestIndPower

standard_deviation = 5
desired_increase = 4
alpha = 0.05
beta = 0.2

analysis = TTestIndPower()
analysis.solve_power( effect_size=desired_increase / standard_deviation,
                      power=1 - beta, alpha=alpha)
```

```
25.52457250047935
```

Our sample size needs to be 26 participants in order for the power of the study to be 80% with our specified parameters.

Content last modified on 21 June 2022.

See a problem? Tell us or edit the source.

How to compute a confidence interval for the difference between two means when both population variances are known

Description

If we have samples from two independent populations, and both of the population variances are known, how do we construct a confidence interval for the difference between the population means?

Related tasks:

- How to compute a confidence interval for a mean difference (matched pairs)
- How to compute a confidence interval for a regression coefficient (on website)
- How to compute a confidence interval for a population mean
- How to compute a confidence interval for a single population variance
- How to compute a confidence interval for the difference between two means when population variances are unknown
- How to compute a confidence interval for the difference between two proportions
- How to compute a confidence interval for the expected value of a response variable
- How to compute a confidence interval for the population proportion
- How to compute a confidence interval for the ratio of two population variances

Solution in Python

We're going to use some fake data here to illustrate how to make the confidence interval. Replace our fake data and population variances with your actual data and population variances if you use this code.

```
sample1 = [15, 10, 7, 22, 17, 14]
sample2 = [9, 1, 11, 13, 3, 6]
pop1_variance = 2.3
pop2_variance = 3
```

We will need the size and mean of each sample.

```
import numpy as np
n_sample1 = len(sample1)
n_sample2 = len(sample2)
xbar1 = np.mean(sample1)
xbar2 = np.mean(sample2)
```

We can then use that data to create the confidence interval.

```
# Find the critical value from the normal distribution
from scipy import stats
alpha = 0.05       # replace with your chosen alpha (here, a 95% confidence level)
critical_val = stats.norm.ppf(1-alpha/2)

# Find the lower and upper bounds of the confidence interval
upper_bound = (xbar1 - xbar2) + \
    critical_val*np.sqrt((pop1_variance/n_sample1) + (pop2_variance/n_sample2))
lower_bound = (xbar1 - xbar2) - \
    critical_val*np.sqrt((pop1_variance/n_sample1) + (pop2_variance/n_sample2))
lower_bound, upper_bound
```

```
(5.15791188458682, 8.842088115413178)
```

Our 95% confidence interval for the true difference between the population means is $[5.1579, 8.842]$.

Content last modified on 09 September 2021.

See a problem? Tell us or edit the source.

How to compute a confidence interval for the difference between two means when population variances are unknown

Description

If we have samples from two independent populations and both of the population variances are unknown, how do we compute a confidence interval for the difference between the population means?

Related tasks:

- How to compute a confidence interval for a mean difference (matched pairs)
- How to compute a confidence interval for a regression coefficient (on website)
- How to compute a confidence interval for a population mean
- How to compute a confidence interval for a single population variance
- How to compute a confidence interval for the difference between two means when both population variances are known
- How to compute a confidence interval for the difference between two proportions
- How to compute a confidence interval for the expected value of a response variable
- How to compute a confidence interval for the population proportion
- How to compute a confidence interval for the ratio of two population variances

Solution in Python

We're going to use some fake data here to illustrate how to make the confidence interval. Replace our fake data with your actual data if you use this code.

```
sample1 = [15, 10, 7, 22, 17, 14]
sample2 = [9, 1, 11, 13, 3, 6]
```

We will need the sizes, means, and variances of each sample.

```
import numpy as np
n_sample1 = len(sample1)
n_sample2 = len(sample2)
xbar1 = np.mean(sample1)
xbar2 = np.mean(sample2)
var_sample1 = np.var(sample1, ddof = 1)
var_sample2 = np.var(sample2, ddof = 1)
```

Before we can compute the confidence interval, we must ask, can we assume that the two population variances are equal?

IF YES: We compute the degrees of freedom and the radius of the confidence interval as follows.

```
df = n_sample1 + n_sample2 - 2
pooled_var = ((n_sample1-1)*var_sample1 + (n_sample2-1)*var_sample2) / df
radius = pooled_var*(1/n_sample1 + 1/n_sample2)
```

IF NO: We *replace* the above code with the following code instead, which does not make the assumption that the population variances are equal.

```
# ratio1 = var_sample1/n_sample1
# ratio2 = var_sample2/n_sample2
# df = (ratio1 + ratio2)**2 / (ratio1**2/(n_sample1-1) + ratio2**2/(n_sample2-1))
# radius = ratio1 + ratio2
```

Then, whichever of the two methods above was used, we compute the confidence interval as follows.

```
from scipy import stats

# Find the critical value from the normal distribution
alpha = 0.05
critical_val = stats.t.ppf(q = 1-alpha/2, df = df)

# Find the lower and upper bound of the confidence interval
upper_bound = (xbar1 - xbar2) + critical_val*np.sqrt(radius)
lower_bound = (xbar1 - xbar2) - critical_val*np.sqrt(radius)
lower_bound, upper_bound
```

```
(0.5980039236697818, 13.401996076330217)
```

The 95% confidence interval for the true difference between these population means is $[0.598, 13.402]$. That was computed under the assumption that the variances were equal. See the alternative code above for if the variances were not assumed to be equal; in that case, we would get the slightly different result of $[0.5852, 13.4147]$ instead.

Content last modified on 07 September 2021.

See a problem? Tell us or edit the source.

How to compute a confidence interval for the difference between two proportions

Description

When dealing with qualitative data, we often want to construct a confidence interval for the difference between two population proportions. For example, if we are trying a drug on experimental and control groups of patients, we probably want to compare the proportion of patients who got well in one group versus the other.

How do we make such a comparison using a confidence interval?

Related tasks:

- How to compute a confidence interval for a mean difference (matched pairs)
- How to compute a confidence interval for a regression coefficient (on website)
- How to compute a confidence interval for a population mean
- How to compute a confidence interval for a single population variance
- How to compute a confidence interval for the difference between two means when both population variances are known
- How to compute a confidence interval for the difference between two means when population variances are unknown
- How to compute a confidence interval for the expected value of a response variable
- How to compute a confidence interval for the population proportion
- How to compute a confidence interval for the ratio of two population variances

Solution in Python

Here is some fake data for the purposes of this illustration. Let's say we conduct a survey of people in Boston and of people in Nashville and ask them if they prefer chocolate or vanilla ice cream. We want to compare the proportions of people from the two cities who like vanilla.

- Out of 150 people in Boston surveyed, 90 prefer vanilla.
- Out of 135 people in Nashville surveyed, 50 prefer vanilla.

We'll let $\bar{p_1}$ represent the proportion of people from Boston who like vanilla and $\bar{p_2}$ represent the proportion of people from Nashville who like vanilla. You can replace the code for this fake data below with your real data.

```
# number of observations in the samples
n1 = 150
n2 = 135
# proportions in the two samples
p_bar1 = 90/150
p_bar2 = 50/135
```

We now compute the confidence interval using tools from SciPy and NumPy.

```
# Find the critical value to compute the confidence interval
from scipy import stats
alpha = 0.05       # replace with your chosen alpha (here, a 95% confidence level)
critical_value = stats.norm.ppf(1-alpha/2)

# Compute the standard error of the proportions
import numpy as np
std_error = np.sqrt( p_bar1*(1-p_bar1)/n1 + p_bar2*(1-p_bar2)/n2 )

# Compute the upper and lower bounds of the confidence interval
upper_bound = (p_bar1 - p_bar2) + critical_value*std_error
lower_bound = (p_bar1 - p_bar2) - critical_value*std_error
lower_bound, upper_bound
```

```
(0.11657216971616415, 0.3426870895430951)
```

The confidence interval for the difference between these two proportions is $[0.11657, 0.34269]$.

Content last modified on 09 September 2021.

See a problem? Tell us or edit the source.

How to compute a confidence interval for the ratio of two population variances

Description

Let's say we want to compute a confidence interval for two population variances. We take two samples of data, $x_1, x_2, x_3, \ldots, x_k$ and $x'_1, x'_2, x'_3, \ldots, x'_k$, and compute their variances, σ_1^2 and σ_2^2. How do we compute a confidence interval for $\frac{\sigma_1^2}{\sigma_2^2}$?

Related tasks:

- How to compute a confidence interval for a mean difference (matched pairs)
- How to compute a confidence interval for a regression coefficient (on website)
- How to compute a confidence interval for a population mean
- How to compute a confidence interval for a single population variance
- How to compute a confidence interval for the difference between two means when both population variances are known
- How to compute a confidence interval for the difference between two means when population variances are unknown
- How to compute a confidence interval for the difference between two proportions
- How to compute a confidence interval for the expected value of a response variable
- How to compute a confidence interval for the population proportion

Solution in Python

We'll use R's dataset EuStockMarkets as an example; of course you should replace this example data with your actual data when using this code. This dataset has information on the daily closing prices of 4 European stock indices. We're going to compare the variability of Germany's DAX and France's CAC closing prices here. Let's load in the dataset using the process explained in how to quickly load some sample data (on website).

```
from rdatasets import data
import pandas as pd

# Load in the EuStockMarkets data and convert to a DataFrame
EuStockMarkets = data('EuStockMarkets')
df = pd.DataFrame(EuStockMarkets[['DAX', 'CAC']])

# Our two samples are its DAX and CAC columns
sample1 = df['DAX'].tolist()
sample2 = df['CAC'].tolist()
```

Now that we have our data loaded we can compute the confidence interval. You can change the confidence level by changing the value of α below.

```
# The degrees of freedom in each sample is its length minus 1
sample1_df = len(sample1) - 1
sample2_df = len(sample2) - 1

# Compute the ratio of the variances
import statistics
ratio = statistics.variance(sample1) / statistics.variance(sample2)

# Find the critical values from the F-distribution
from scipy import stats
alpha = 0.05       # replace with your chosen alpha (here, a 95% confidence level)
lower_critical_value = 1 / stats.f.ppf(q = 1 - alpha/2, dfn = sample1_df, dfd = sample2_df)
upper_critical_value = stats.f.ppf(q = 1 - alpha/2, dfn = sample2_df, dfd = sample1_df)

# Compute the confidence interval
lower_bound = ratio * lower_critical_value
upper_bound = ratio * upper_critical_value
lower_bound, upper_bound
```

```
(3.190589226470889, 3.827043522824141)
```

The 95% confidence interval for the ratio of the variances for Germany's DAX and France's CAC is $[3.191, 3.827]$.

Content last modified on 09 September 2021.

See a problem? Tell us or edit the source.

How to do a hypothesis test for the difference between means when both population variances are known

Description

Assume we have two samples, $x_1, x_2, \ldots, x_n$ and $x'_1, x'_2, \ldots, x'_n$, that come from normally distributed populations with known variances, and the two sample means are $\bar{x}$ and $\bar{x}'$, respectively. We might want to ask whether the difference $\bar{x} - \bar{x}'$ is significantly different from, greater than, or less than zero.

Related tasks:

- How to compute a confidence interval for the difference between two means when both population variances are known
- How to do a hypothesis test for a mean difference (matched pairs)
- How to do a hypothesis test for a population proportion
- How to do a hypothesis test for population variance
- How to do a hypothesis test for the difference between two proportions
- How to do a hypothesis test for the mean with known standard deviation
- How to do a hypothesis test for the ratio of two population variances
- How to do a hypothesis test of a coefficient's significance (on website)
- How to do a one-sided hypothesis test for two sample means
- How to do a two-sided hypothesis test for a sample mean (on website)
- How to do a two-sided hypothesis test for two sample means (on website)

Solution in Python

We're going to use fake data here, but you can replace our fake data with your real data below. You will need not only the samples but also the known population standard deviations.

```
sample1 = [ 5,  8, 10,  3,  6,  2]
sample2 = [13, 20, 16, 12, 18, 15]
population1_sd = 2.4
population2_sd = 3
```

We must compute the sizes and means of the two samples.

```
import numpy as np
n1 = len(sample1)
n2 = len(sample2)
sample1_mean = np.mean(sample1)
sample2_mean = np.mean(sample2)
```

We choose a value $0 \leq \alpha \leq 1$ as the probability of a Type I error (a false positive, finding we should reject H_0 when it's actually true). We will use $\alpha = 0.05$ in this example.

Two-tailed test

In a two-tailed test, the null hypothesis is that the difference is zero, $H_0 : \bar{x} - \bar{x}' = 0$. We compute a test statistic and p-value as follows.

```
from scipy import stats
test_statistic = ( (sample1_mean - sample2_mean) /
    np.sqrt(population1_sd**2/n1 + population2_sd**2/n2) )
2*stats.norm.sf(abs(test_statistic))  # two-tailed p-value
```

```
1.8204936819059392e-10
```

Our p-value is less than α, so we have sufficient evidence to reject the null hypothesis. The difference between the means is significantly different from zero.

Right-tailed test

In the right-tailed test, the null hypothesis is $H_0 : \bar{x} - \bar{x}' \leq 0$. That is, we are testing whether the difference is greater than zero.

The code is very similar to the previous, except only in computing the p-value. We repeat the code that's in common, to make it easier to copy and paste the examples.

```
from scipy import stats
test_statistic = ( (sample1_mean - sample2_mean) /
    np.sqrt(population1_sd**2/n1 + population2_sd**2/n2) )
stats.norm.sf(test_statistic)  # right-tailed p-value
```

```
0.9999999999089754
```

Our p-value is greater than α, so we do not have sufficient evidence to reject the null hypothesis. We would continue to assume that the difference in means is less than or equal to zero.

Left-tailed test

In a left-tailed test, the null hypothesis is $H_0 : \bar{x} - \bar{x}' \geq 0$. That is, we are testing whether the difference is less than zero.

The code is very similar to the previous, except only in computing the p-value. We repeat the code that's in common, to make it easier to copy and paste the examples.

```
from scipy import stats
test_statistic = ( (sample1_mean - sample2_mean) /
    np.sqrt(population1_sd**2/n1 + population2_sd**2/n2) )
stats.norm.sf(-test_statistic)  # left-tailed p-value
```

```
9.102468409529696e-11
```

Our p-value is less than α, so we have sufficient evidence to reject the null hypothesis. The difference between the means is significantly less than zero.

Content last modified on 24 October 2021.

How to do a hypothesis test for the difference between two proportions

Description

When dealing with qualitative data, we typically measure what proportion of the population falls into various categories (e.g., which religion a survey respondent adheres to, if any). We might want to compare two proportions by measuring their difference, and asking whether it is equal, greater, or less than zero. How can we perform such a test?

Related tasks:

- How to compute a confidence interval for the difference between two proportions
- How to do a hypothesis test for a mean difference (matched pairs)
- How to do a hypothesis test for a population proportion
- How to do a hypothesis test for population variance
- How to do a hypothesis test for the difference between means when both population variances are known
- How to do a hypothesis test for the mean with known standard deviation
- How to do a hypothesis test for the ratio of two population variances
- How to do a hypothesis test of a coefficient's significance (on website)
- How to do a one-sided hypothesis test for two sample means
- How to do a two-sided hypothesis test for a sample mean (on website)
- How to do a two-sided hypothesis test for two sample means (on website)

Solution in Python

We will use some fake data in this example, but you can replace it with your real data. Imagine we conduct a survey of people in Boston and of people in Nashville and ask them if they prefer chocolate or vanilla ice cream. We get data like the following.

City	Prefer chocolate	Prefer vanilla	Total
Boston	60	90	150
Nashville	85	50	135

We want to compare the proportions of people from the two cities who like vanilla.

Let $\bar{p}_1$ represent the proportion of people from Boston who like vanilla and $\bar{p}_2$ represent the proportion of people from Nashville who like vanilla.

```
n1 = 150         # number of observations in sample 1
n2 = 135         # number of observations in sample 2
p_bar1 = 90/150  # proportion in sample 1
p_bar2 = 50/135  # proportion in sample 2
```

We choose a value $0 \leq \alpha \leq 1$ as our Type 1 error rate. For this example, we will use $\alpha = 0.05$.

Two-tailed test

In a two-tailed test, the null hypothesis states that the difference between the two proportions equals a hypothesized value; let's choose zero, $H_0 : \bar{p}_1 - \bar{p}_2 = 0$. We perform this test by computing a test statistic and p-value as shown below, then comparing the p-value to our chosen α.

```
import numpy as np
p_bar = (90 + 50) / (150 + 135)                    # overall proportion
std_error = np.sqrt(p_bar*(1-p_bar)*(1/n1+1/n2))  # standard error
test_statistic = (p_bar1 - p_bar2)/std_error       # test statistic

from scipy import stats
2*stats.norm.sf(abs(test_statistic))               # two-tailed p-value
```

```
0.00010802693662804402
```

Our p-value, 0.000108, is smaller than α, so we can reject the null hypothesis and conclude that the difference between the two proportions is different from zero.

But we did not need to compare the difference to zero; we could have used any hypothesized difference for comparison. Let's repeat the above test, comparing the difference to 0.15 instead, as an example.

```
import numpy as np
hyp_diff = 0.15                                          # hypothesized difference
std_error = np.sqrt(p_bar1*(1-p_bar1)/n1
                  + p_bar2*(1-p_bar2)/n2)                # standard error
test_statistic = ((p_bar1 - p_bar2) - hyp_diff)/std_error  # test statistic

from scipy import stats
2*stats.norm.sf(abs(test_statistic))                     # two-tailed p-value
```

```
0.16744531573658772
```

Our p-value, 0.1674, is greater than α, so we cannot reject the null hypothesis and cannot conclude that the difference between these two proportions is significantly different from 0.15.

Right-tailed test

In a right-tailed test, the null hypothesis states that the difference between the two proportions is less than or equal to a hypothesized value. Let's begin by using zero as our hypothesized value, $H_0 : \bar{p}_1 - \bar{p}_2 \leq 0$.

We repeat some code below that we've seen above, just to make it easy to copy and paste the example elsewhere.

```
import numpy as np
p_bar = (90 + 50) / (150 + 135)                    # overall proportion
std_error = np.sqrt(p_bar*(1-p_bar)*(1/n1+1/n2))  # standard error
test_statistic = (p_bar1 - p_bar2)/std_error       # test statistic

from scipy import stats
stats.norm.sf(abs(test_statistic))                 # right-tailed p-value
```

```
5.401346831402201e-05
```

Our p-value is smaller than α, so we can reject the null hypothesis and conclude that the difference between the two proportions is significantly greater than zero.

But we did not need to compare the difference to zero; we could have used any hypothesized difference for comparison. Let's repeat the above test, comparing the difference to 0.15 instead, as an example.

```
import numpy as np
hyp_diff = 0.15                                          # hypothesized difference
std_error = np.sqrt(p_bar1*(1-p_bar1)/n1
                  + p_bar2*(1-p_bar2)/n2)                # standard error
test_statistic = ((p_bar1 - p_bar2) - hyp_diff)/std_error  # test statistic

from scipy import stats
stats.norm.sf(abs(test_statistic))                       # right-tailed p-value
```

```
0.08372265786829386
```

Our p-value, 0.0837, is greater than α, so we cannot reject the null hypothesis and cannot conclude that the difference between these two proportions is significantly greater than 0.15.

Left-tailed test

In a left-tailed test, the null hypothesis states that the difference between the two proportions is greater than or equal to a hypothesized value. Let's begin by using zero as our hypothesized value, $H_0 : \bar{p}_1 - \bar{p}_2 \geq 0$.

We repeat some code below that we've seen above, just to make it easy to copy and paste the example elsewhere.

```
import numpy as np
p_bar = (90 + 50) / (150 + 135)                  # overall proportion
std_error = np.sqrt(p_bar*(1-p_bar)*(1/n1+1/n2))  # standard error
test_statistic = (p_bar1 - p_bar2)/std_error     # test statistic

from scipy import stats
stats.norm.sf(-test_statistic)                   # left-tailed p-value
```

```
0.999945986531686
```

Our p-value, 0.9999, is greater than α, so we cannot reject the null hypothesis and cannot conclude that the difference between the two proportions is significantly less than zero.

But we did not need to compare the difference to zero; we could have used any hypothesized difference for comparison. Let's repeat the above test, comparing the difference to 0.15 instead, as an example.

```
import numpy as np
hyp_diff = 0.15                                          # hypothesized difference
std_error = np.sqrt(p_bar1*(1-p_bar1)/n1
                  + p_bar2*(1-p_bar2)/n2)                # standard error
test_statistic = ((p_bar1 - p_bar2) - hyp_diff)/std_error  # test statistic

from scipy import stats
stats.norm.sf(-test_statistic)                           # left-tailed p-value
```

```
0.9162773421317061
```

Our p-value, 0.91627, is greater than α, so we cannot reject the null hypothesis and cannot conclude that the difference between these two proportions is significantly less than 0.15.

Content last modified on 24 October 2021.

See a problem? Tell us or edit the source.

How to do a hypothesis test for the ratio of two population variances

Description

Let's say we want to compare the variability of two populations. We take two samples of data, $x_1, x_2, x_3, \ldots, x_k$ from population 1 and $x'_1, x'_2, x'_3, \ldots, x'_k$ from population 2. What hypothesis tests can help us compare the population variances?

Related tasks:

- How to compute a confidence interval for the difference between two proportions
- How to do a hypothesis test for a mean difference (matched pairs)
- How to do a hypothesis test for a population proportion
- How to do a hypothesis test for population variance
- How to do a hypothesis test for the difference between means when both population variances are known
- How to do a hypothesis test for the difference between two proportions
- How to do a hypothesis test for the mean with known standard deviation
- How to do a hypothesis test of a coefficient's significance (on website)
- How to do a one-sided hypothesis test for two sample means
- How to do a two-sided hypothesis test for a sample mean (on website)
- How to do a two-sided hypothesis test for two sample means (on website)

Solution in Python

We'll use R's dataset `EuStockMarkets` to do an example. This dataset has information on the daily closing prices of 4 European stock indices. We're going to compare the variability of Germany's DAX and France's CAC closing prices.

Let's load the dataset. (See how to quickly load some sample data (on website).) If using your own data, place it into the `sample1` and `sample2` variables instead of using the code below.

```
from rdatasets import data
import pandas as pd

# Load in the EuStockMarkets data and place it in a pandas DataFrame
EuStockMarkets = data('EuStockMarkets')
df = pd.DataFrame(EuStockMarkets[['DAX', 'CAC']])

# Choose the two columns we want to analyze
# (You can replace the two lines below with your actual data.)
sample1 = df['DAX']
sample2 = df['CAC']
```

For all tests below, we will use $\alpha = 0.05$ as our Type I Error Rate, but any value between 0.0 and 1.0 can be used.

Two-tailed test

We can use a two-tailed test to test whether the two population variances are equal. Specifically, the null hypothesis will be:

$$H_0 : \frac{\sigma_1^2}{\sigma_2^2} = 1$$

```
from scipy import stats
sample1_df = len(sample1) - 1                    # degrees of freedom
sample2_df = len(sample2) - 1                    # degrees of freedom
test_statistic = sample1.var() / sample2.var()  # test statistic
stats.f.sf(test_statistic, dfn = sample1_df, dfd = sample2_df)*2  # p-value
```

```
7.729079251495416e-151
```

Our p-value is smaller than our chosen alpha, so we have sufficient evidence to reject the null hypothesis. The ratio of the variance of the closing prices on Germany's DAX and France's CAC is significantly different than 1, so the variances are not equal.

Right-tailed test

In a right-tailed test, the null hypothesis is that the ratio is less than or equal to 1. This is equivalent to asking if $\sigma_1^2 \leq \sigma_2^2$.

$$H_0 : \frac{\sigma_1^2}{\sigma_2^2} \leq 1$$

We repeat below some of the code above to make each example easy to copy and paste.

```
from scipy import stats
sample1_df = len(sample1) - 1                    # degrees of freedom
sample2_df = len(sample2) - 1                    # degrees of freedom
test_statistic = sample1.var() / sample2.var()  # test statistic
stats.f.sf(test_statistic, dfn = sample1_df, dfd = sample2_df)  # p-value
```

```
3.864539625747708e-151
```

Our p-value is smaller than our chosen alpha, so we have sufficient evidence to reject the null hypothesis. The ratio of the variance of the closing prices on Germany's DAX and France's CAC is significantly greater than 1, so the variance of closing prices on Germany's DAX is greater than that of closing prices on France's CAC.

To test whether $\sigma_1^2 \geq \sigma_2^2$, simply swap the roles of the two data columns in the above code.

Content last modified on 24 October 2021.

How to do a Kruskal-Wallis test

Description

If we have samples from several independent populations, we might want to test whether the population medians are equal. We may not be able to assume anything about the populations' variances, nor whether they are normally distributed, but we do assume that the populations have distributions that are approximately the same shape. The Kruskal-Wallis Test will allow us to test the medians for equality. It is similar to a One-Way ANOVA but using medians instead of means. How do we perform a Kruskal-Wallis Test?

Related tasks:

- How to do a one-way analysis of variance (ANOVA)
- How to use Bonferroni's Correction method
- How to do a Wilcoxon rank-sum test

Solution in Python

For the purposes of this example, let's say we have a sample of GPAs from matriculated students at three Ivy League institutions: Harvard, Dartmouth, and Columbia. This is example data, and you can replace it with your actual data when you re-use this code.

SciPy requires our data to be in NumPy arrays, as shown below. Note that pandas Series (e.g., columns in a DataFrame) are also NumPy arrays.

```
import numpy as np
# Replace the fake data below with your real data
harvard   = np.array([3.40, 3.66, 3.90, 3.55, 3.90, 3.58])
dartmouth = np.array([3.90, 3.97, 3.92, 3.83, 4.00, 3.68])
columbia  = np.array([4.00, 3.75, 3.34])
```

The Kruskal-Willis Test uses a null hypothesis that the category medians are equal, $H_0 : m_C = m_H = m_D \leq 0$. We choose α, or the Type I error rate, as 0.05 and run the test as shown below.

```
from scipy import stats
stats.kruskal(harvard, dartmouth, columbia)
```

```
KruskalResult(statistic=3.706006006006005, pvalue=0.15676569090635095)
```

The p-value, 0.1568, is greater than α, so we fail to reject the null hypothesis. We do not have sufficient evidence to conclude that the median GPAs of matriculated students at these three schools are different from each other.

Content last modified on 05 October 2021.

See a problem? Tell us or edit the source.

How to do a one-sided hypothesis test for two sample means

Description

If we have two samples, $x_1, \ldots, x_n$ and $x'_1, \ldots, x'_n$, and we compute the mean of each one, we might want to ask whether one mean is less than the other. Or more precisely, is their difference significantly less than zero?

Related tasks:

- How to compute a confidence interval for a population mean
- How to do a two-sided hypothesis test for a sample mean (on website)
- How to do a two-sided hypothesis test for two sample means (on website)
- How to do a one-way analysis of variance (ANOVA)
- How to do a hypothesis test for a mean difference (matched pairs)
- How to do a hypothesis test for a population proportion

Solution in Python

If we call the mean of the first sample $\bar{x}_1$ and the mean of the second sample $\bar{x}_2$, then this is a two-sided test with the null hypothesis $H_0 : \bar{x}_1 - \bar{x}_2 \geq 0$. We choose a value $0 \leq \alpha \leq 1$ as the probability of a Type I error (false positive, finding we should reject H_0 when it's actually true). Let's use $\alpha = 0.10$ as an example.

```
from scipy import stats

# Replace these first three lines with the values from your situation.
sample1 = [ 6, 9, 7, 10, 10, 9 ]
sample2 = [ 12, 14, 10, 17, 9 ]

# Run a one-sample t-test and print out alpha, the p value,
# and whether the comparison says to reject the null hypothesis.
stats.ttest_ind( sample1, sample2, equal_var=False, alternative="less" )
```

```
Ttest_indResult(statistic=-2.4616581720814326, pvalue=0.025486418709238467)
```

The output says that the p-value is about 0.0255, which is less than $\alpha = 0.10$. Therefore the samples give us enough evidence to reject the null hypothesis at the $\alpha = 0.10$ level. That is, the data suggest that $\bar{x}_1 < \bar{x}_2$.

The `equal_var` parameter tells SciPy *not* to assume that the two samples have equal variances. If in your case they do, you can omit that parameter, and it will revert to its default value of `True`.

Content last modified on 07 December 2021.

See a problem? Tell us or edit the source.

How to do a Wilcoxon rank-sum test

Description

Assume we have two independent samples of data, $x_1, x_2, x_3, \ldots x_n$ and $x'_1, x'_2, x'_3, \ldots x'_m$, each from a different population. Also assume that the sample sizes are small or the populations are not normally distributed, but that the two population distributions are approximately the same shape. How can we test whether there is a significant difference between the two medians (or if one is significantly greater than or less than the other)? One method is the Wilcoxon Rank-Sum Test.

Related tasks:

- How to do a Kruskal-Wallis test
- How to do a Wilcoxon signed-rank test
- How to do a Wilcoxon signed-rank test for matched pairs

Solution in Python

We're going to use fake data for illustrative purposes, but you can replace our fake data with your real data. Say our first sample, $x_1, x_2, x_3, \ldots x_n$, has median m_1, and our second sample, $x'_1, x'_2, x'_3, \ldots x'_m$, has median m_2.

```
import numpy as np
# Replace sample1 and sample2 with your data
sample1 = np.array([56, 31, 190, 176, 119, 15, 140, 152, 167])
sample2 = np.array([45, 36, 78, 54, 12, 25, 39, 48, 52, 70, 85])
```

We choose a value, $0 \leq \alpha \leq 1$, as the Type I Error Rate. We'll let α be 0.05.

Two-tailed test

To test the null hypothesis $H_0 : m_1 - m_2 = 0$, that is, $m_1 = m_2$, we use a two-tailed test:

```
from scipy import stats
from scipy.stats import ranksums
ranksums(sample1, sample2)
```

```
RanksumsResult(statistic=2.0892772350933626, pvalue=0.03668277440246522)
```

Our p-value, 0.03668, is less than $\alpha = 0.05$, so we have sufficient evidence to reject the null hypothesis. The population medians are significantly different from each other.

(The output above is slightly different from the output you would get by running this test in R, because SciPy uses a normal distribution internally, but R uses a Wilcoxon distribution.)

Right-tailed test

To test the null hypothesis $H_0 : m_1 - m_2 \leq 0$, that is, $m_1 \leq m_2$, we use a right-tailed test:

```
ranksums(sample1, sample2, alternative = 'greater')
```

```
RanksumsResult(statistic=2.0892772350933626, pvalue=0.01834138720123261)
```

Our p-value, 0.01834, is less than $\alpha = 0.05$, so we have sufficient evidence to reject the null hypothesis. The first population medians is significantly greater second.

(The output above is slightly different from the output you would get by running this test in R, because SciPy uses a normal distribution internally, but R uses a Wilcoxon distribution.)

Left-tailed test

To test the null hypothesis $H_0 : m_1 - m_2 \geq 0$, that is, $m_1 \geq m_2$, we use a left-tailed test:

```
ranksums(sample1, sample2, alternative = 'less')
```

```
RanksumsResult(statistic=2.0892772350933626, pvalue=0.9816586127987674)
```

Our p-value, 0.98165, is greater than α, so we do not have sufficient evidence to reject the null hypothesis. The first population median is not significantly smaller than the second population median.

(The output above is slightly different from the output you would get by running this test in R, because SciPy uses a normal distribution internally, but R uses a Wilcoxon distribution.)

Content last modified on 30 November 2021.

See a problem? Tell us or edit the source.

How to do a Wilcoxon signed-rank test

Description

Assume we a sample of data, $x_1, x_2, x_3, \ldots x_k$ and either the sample size is small or the population is not normally distributed. But we still want to perform tests that compare the sample median to a hypothesized value (equal, greater, or less). One method is the Wilcoxon Signed-Rank Test.

Related tasks:

- How to do a Kruskal-Wallis test
- How to do a Wilcoxon rank-sum test
- How to do a Wilcoxon signed-rank test for matched pairs

Solution in Python

We're going to use fake data for illustrative purposes, but you can replace our fake data with your real data. Say our sample, $x_1, x_2, x_3, \ldots x_k$, has median m.

```
import numpy as np
# Replace the next line with your data
sample = np.array([19, 4, 23, 16, 1, 8, 30, 25, 13])
```

We choose a value, $0 \leq \alpha \leq 1$, as the Type I Error Rate. We'll let α be 0.05. In the examples below, we will be comparing the median m to a hypothesized value of $a = 10$, but you can use any value for a.

Two-tailed test

To test the null hypothesis $H_0 : m = a$, we use a two-tailed test:

```
from scipy import stats
from scipy.stats import wilcoxon
a = 10  # or your chosen value for comparison
wilcoxon(sample - a)
```

```
WilcoxonResult(statistic=10.0, pvalue=0.1640625)
```

Our p-value, 0.1640625, is greater than $\alpha = 0.05$, so we do not have sufficient evidence to reject the null hypothesis. We may continue to assume the population median is equal to 10.

Right-tailed test

To test the null hypothesis $H_0 : m \geq a$, we use a right-tailed test:

```
wilcoxon(sample - a, alternative = 'less')
```

```
WilcoxonResult(statistic=35.0, pvalue=0.935546875)
```

Our p-value, 0.935546875, is greater than $\alpha = 0.05$, so we do not have sufficient evidence to reject the null hypothesis. We may continue to assume the population median is less than (or equal to) 10.

Left-tailed test

To test the null hypothesis $H_0 : m \leq a$, we use a left-tailed test:

```
wilcoxon(sample - a, alternative = 'greater')
```

```
WilcoxonResult(statistic=35.0, pvalue=0.08203125)
```

Our p-value, 0.08203125, is greater than α, so we do not have sufficient evidence to reject the null hypothesis. We may continue to assume the population median is greater than (or equal to) 10.

Content last modified on 24 October 2021.

See a problem? Tell us or edit the source.

How to do a Wilcoxon signed-rank test for matched pairs

Description

Assume we have two samples of data that come in matched pairs, $x_1, x_2, x_3, \ldots x_k$ and $x'_1, x'_2, x'_3, \ldots x'_k$, which we might pair up as $(x_1, x'_1), (x_2, x'_2), \ldots, (x_k, x'_k)$. The two samples may be from different populations. Also assume that the sample sizes are small or the populations are not normally distributed.

Consider measuring the difference in each pair, $x_1 - x'_1, x_2 - x'_2, \ldots, x_k - x'_k$. We want to perform tests that compare the median of those differences, m_D, to a hypothesized value (equal, greater, or less). One method is the Wilcoxon Signed-Rank Test for Matched Pairs.

Related tasks:

- How to do a Kruskal-Wallis test
- How to do a Wilcoxon rank-sum test
- How to do a Wilcoxon signed-rank test

Solution in Python

The method we will use is equivalent to subtracting the two samples and then performing the signed-rank test. See how to do a Wilcoxon signed-rank test to compare the two methods.

We're going to use fake data for illustrative purposes, but you can replace our fake data with your real data.

```
import numpy as np
# Replace sample1 and sample2 with your data
sample1 = np.array([156, 133, 90, 176, 119, 120, 40, 52, 167, 80])
sample2 = np.array([45, 36, 78, 54, 12, 25, 39, 48, 52, 70])
```

We choose a value, $0 \leq \alpha \leq 1$, as the Type I Error Rate. We'll let α be 0.05.

Two-tailed test

To test the null hypothesis $H_0 : m_D = 0$, we use a two-tailed test:

```
from scipy import stats
from scipy.stats import wilcoxon
wilcoxon(sample1 - sample2)
```

```
WilcoxonResult(statistic=0.0, pvalue=0.001953125)
```

Our p-value, 0.001953125, is less than $\alpha = 0.05$, so we have sufficient evidence to reject the null hypothesis. The median difference is significantly different from zero.

Right-tailed test

To test the null hypothesis $H_0 : m_D \leq 0$, we use a right-tailed test:

```
wilcoxon(sample1 - sample2, alternative = 'greater')
```

```
WilcoxonResult(statistic=55.0, pvalue=0.0009765625)
```

Our p-value, 0.0009765625, is less than $\alpha = 0.05$, so we have sufficient evidence to reject the null hypothesis. The median difference is significantly greater than zero.

Left-tailed test

To test the null hypothesis $H_0 : m_D \geq 0$, we use a left-tailed test:

```
wilcoxon(sample1 - sample2, alternative = 'less')
```

```
WilcoxonResult(statistic=55.0, pvalue=1.0)
```

Our p-value, 1.0, is greater than α, so we do not have sufficient evidence to reject the null hypothesis. We should continue to assume that the mean difference may be less than (or equal to) zero.

Content last modified on 24 October 2021.

See a problem? Tell us or edit the source.

How to compute a confidence interval for a single population variance

Description

Let's say we want to compute a confidence interval for the variability of a population. We take a sample of data, $x_1, x_2, x_3, \ldots, x_k$ and compute its variance, s^2. How do we construct a confidence interval for the population variance σ^2?

Related tasks:

- How to compute a confidence interval for a mean difference (matched pairs)
- How to compute a confidence interval for a regression coefficient (on website)
- How to compute a confidence interval for a population mean
- How to compute a confidence interval for the difference between two means when both population variances are known
- How to compute a confidence interval for the difference between two means when population variances are unknown
- How to compute a confidence interval for the difference between two proportions
- How to compute a confidence interval for the expected value of a response variable
- How to compute a confidence interval for the population proportion
- How to compute a confidence interval for the ratio of two population variances

Solution in Python

We'll use R's dataset EuStockMarkets here. This dataset has information on the daily closing prices of 4 European stock indices. We're going to look at the variability of Germany's DAX closing prices. For more information on how this is done, see how to quickly load some sample data (on website). You can replace the example data below with your actual data.

```
# Load in EuStockMarkets data
from rdatasets import data
import pandas as pd
df = data('EuStockMarkets')

# Select the column for Germany's DAX closing prices
sample = df['DAX']
```

Now that we have our sample data loaded, let's go ahead and make the confidence interval using SciPy.

```
from scipy import stats

# Find the critical values from the right and left tails of the Chi-square distribution
alpha = 0.05       # replace with your chosen alpha (here, a 95% confidence level)
n = len( sample )
left_critical_val = stats.chi2.ppf(1-alpha/2, df=n-1)
right_critical_val = stats.chi2.ppf(alpha/2, df=n-1)

# Find the upper and lower bounds of the confidence interval and print them out
lower_bound = ((n - 1)*sample.var())/left_critical_val
upper_bound = ((n - 1)*sample.var())/right_critical_val
lower_bound, upper_bound
```

```
(1104642.2801539514, 1256248.1273200295)
```

Our 95% confidence interval for the standard deviation of Germany's DAX closing prices is [1104642, 1256248].

Content last modified on 09 September 2021.

See a problem? Tell us or edit the source.

How to perform a chi-squared test on a contingency table

Description

If we have a contingency table showing the frequencies observed in two categorical variables, how can we run a χ^2 test to see if the two variables are independent?

Solution in Python

Here we will use nested Python lists to store a contingency table of education vs. gender, taken from Penn State University's online stats review website. You should use your own data, and it can be in Python lists or NumPy arrays or a pandas DataFrame.

```
data = [
    # HS  BS  MS  Phd
    [ 60, 54, 46, 41 ], # females
    [ 40, 44, 53, 57 ]  # males
]
```

The χ^2 test's null hypothesis is that the two variables are independent. We choose a value $0 \leq \alpha \leq 1$ as the probability of a Type I error (false positive, finding we should reject H_0 when it's actually true).

SciPy's stats package provides a `chi2_contingency` function that does exactly what we need.

```
alpha = 0.05  # or choose your own alpha here

from scipy import stats
# Run a chi-squared and print out alpha, the p value,
# and whether the comparison says to reject the null hypothesis.
# (The dof and ex variables are values we don't need here.)
chi2_statistic, p_value, dof, ex = stats.chi2_contingency( data )
reject_H0 = p_value < alpha
alpha, p_value, reject_H0
```

```
(0.05, 0.045886500891747214, True)
```

In this case, the samples give us enough evidence to reject the null hypothesis at the $\alpha = 0.05$ level. The data suggest that the two categorical variables are not independent.

Content last modified on 28 May 2021.

See a problem? Tell us or edit the source.

How to do a goodness of fit test for a multinomial experiment

Description

If we have historical values for multiple population proportions, plus more recent samples from those same populations, we may want to compare to see if the proportions appear to have changed. This is called a goodness of fit test for a multinomial experiment. How can we execute it?

Solution in Python

Let's say we have a dataset with the previous population proportions for four categories. (This is contrived data, but the code below can be used on your actual data.)

Category	Frequency	Proportion
A	43	0.25
B	62	0.36
C	52	0.30
D	16	0.09

We have also taken a more recent sample and found the number of observations from it that belong to each category. We want to determine if the proportions coming from the recent sample are equal to the previous proportions.

SciPy expects that we will have two lists, one with the expected number of observations in each group (from the previous, or hypothesized proportions) and the other with the actual number of observations in each group (from the more recent sample). SciPy also expects that the total number of observations in each list is the same. We'll create two lists below with the fake data from above, but you can replace them with your real data

```
# Replace your data in the next two lines
old_observations = [43, 62, 52, 16]
new_observations = [56, 80, 12, 25]
```

We set the null hypothesis to be that the proportions of each category from the recent sample are equal to the previous proportions

$$H_0 : p_A = 0.25 \text{ and } \ p_B = 0.36 \text{ and } \ p_C = 0.30 \text{ and } \ p_D = 0.09.$$

We choose a value $0 \leq \alpha \leq 1$ as our Type 1 error rate. We'll let α be 0.05 here.

```
# Run the Chi-square test, giving the test statistic and p-value
from scipy import stats
stats.chisquare(f_obs=new_observations, f_exp=old_observations)
```

```
Power_divergenceResult(statistic=44.98776977898321, pvalue=9.30824439694332e-10)
```

Our p-value is less than α, so we have sufficient evidence to reject the null hypothesis. It does appear that the proportion of at least one of the four categories is significantly different now from what it was previously.

Content last modified on 23 September 2021.

See a problem? Tell us or edit the source.

How to do a one-way analysis of variance (ANOVA)

Description

If we have multiple independent samples of the same quantity (such as students' SAT scores from several different schools), we may want to test whether the means of each of the samples are the same. Analysis of Variance (ANOVA) can determine whether any two of the sample means differ significantly. How can we do an ANOVA?

Related tasks:

- How to do a two-sided hypothesis test for two sample means (on website) (which is just an ANOVA with only two samples)
- How to do a two-way ANOVA test with interaction
- How to do a two-way ANOVA test without interaction
- How to compare two nested linear models (on website)
- How to conduct a mixed designs ANOVA (on website)
- How to conduct a repeated measures ANOVA (on website)
- How to perform an analysis of covariance (ANCOVA)
- How to do a Kruskal-Wallis test

Solution in Python

Let's assume we have our samples in several different Python lists. (Although anything like a list is also supported, including pandas Series.) Here I'll construct some made-up data about SAT scores at four different schools.

```
school1_SATs = [ 1100, 1250, 1390, 970, 1510 ]
school2_SATs = [ 1010, 1050, 1090, 1110 ]
school3_SATs = [ 900, 1550, 1300, 1270, 1210 ]
school4_SATs = [ 900, 850, 1110, 1070, 910, 920 ]
```

ANOVA tests the null hypothesis that all group means are equal. You choose α, the probability of Type I error (false positive, finding we should reject H_0 when it's actually true). I will use $\alpha = 0.05$ in this example.

```
alpha = 0.05

# Run a one-way ANOVA and print out alpha, the p value,
# and whether the comparison says to reject the null hypothesis.
from scipy import stats
F_statistic, p_value = stats.f_oneway(
    school1_SATs, school2_SATs, school3_SATs, school4_SATs )
reject_H0 = p_value < alpha
alpha, p_value, reject_H0
```

```
(0.05, 0.0342311478489849, True)
```

The result we see above is to reject H_0, and therefore conclude that at least one pair of means is statistically significantly different.

Content last modified on 07 December 2021.

See a problem? Tell us or edit the source.

How to do a two-way ANOVA test with interaction

Description

When we analyze the impact that two factors have on a response variable, we often consider the possible relationship between the two factors. That is, does their interaction term affect the response variable? A two-way ANOVA test with interaction can answer that question.

Related tasks:

- How to do a one-way analysis of variance (ANOVA)
- How to do a two-way ANOVA test without interaction
- How to compare two nested linear models (on website) using ANOVA
- How to conduct a mixed designs ANOVA (on website)
- How to conduct a repeated measures ANOVA (on website)
- How to perform an analysis of covariance (ANCOVA)

Solution in Python

We're going to use R's `esoph` dataset, about esophageal cancer cases. We will focus on the impact of age group (`agegp`) and alcohol consumption (`alcgp`) on the number of cases of the cancer (`ncases`). We ask, does the interaction between these two factors affect the number of cases?

First, we load in the dataset. (See how to quickly load some sample data (on website).)

```
from rdatasets import data
data = data('esoph')
data.head()
```

	agegp	alcgp	tobgp	ncases	ncontrols
0	25-34	0-39g/day	0-9g/day	0	40
1	25-34	0-39g/day	10-19	0	10
2	25-34	0-39g/day	20-29	0	6
3	25-34	0-39g/day	30+	0	5
4	25-34	40-79	0-9g/day	0	27

Next, we create a model that includes the response variable we care about, plus the two categorical variables we will be testing, as well as their interaction term.

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
# C(...) means the variable is categorical, and : means multiplication.
model = ols('ncases ~ C(alcgp) + C(agegp) + C(alcgp):C(agegp)', data = data).fit()
```

A two-way ANOVA with interaction tests the following three null hypotheses.

1. There is no interaction between the two categorical variables. (If we reject this we do not test the other two hypotheses.)
2. The mean response is the same across all groups of the first factor. (In our example, that says the mean `ncases` is the same for all age groups.)
3. The mean response is the same across all groups of the second factor. (In our example, that says the mean `ncases` is the same for all alcohol consumption groups.)

We choose a value, $0 \leq \alpha \leq 1$, as the Type I Error Rate. Let's let $\alpha = 0.05$ here.

```
sm.stats.anova_lm(model, typ=2)
```

	sum_sq	df	F	PR(>F)
C(alcgp)	52.695287	3.0	4.723387	4.862447e-03
C(agegp)	267.026108	5.0	14.361068	2.021935e-09
C(alcgp):C(agegp)	107.557743	15.0	1.928206	3.632710e-02
Residual	238.000000	64.0	NaN	NaN

The p-value for the interaction of age group and alcohol consumption is in the third row, final column, 3.63271×10^{-2}. It is less than α, so we can reject the null hypothesis that age group and alcohol consumption do not interact with regards to the number of esophageal cancer cases. That is, we have reason to believe that their interaction does effect the outcome.

As we stated when we listed the hypotheses to test, since we rejected the first null hypothesis, we will not test the other two. In the case where you failed to reject the first null hypothesis, you could consider each p-value in the first two rows of the above table, one for each of the two factors.

Content last modified on 24 October 2021.

See a problem? Tell us or edit the source.

How to do a two-way ANOVA test without interaction

Description

When we analyze the impact that two factors have on a response variable, we may know in advance that the two factors do not interact. How can we use a two-way ANOVA test to test for an effect from each factor without including an interaction term for the two factors?

Related tasks:

- How to do a one-way analysis of variance (ANOVA)
- How to do a two-way ANOVA test without interaction
- How to compare two nested linear models (on website) using ANOVA
- How to conduct a mixed designs ANOVA (on website)
- How to conduct a repeated measures ANOVA (on website)
- How to perform an analysis of covariance (ANCOVA)

Solution in Python

We're going to use R's `esoph` dataset, about esophageal cancer cases. We will focus on the impact of age group (`agegp`) and alcohol consumption (`alcgp`) on the number of cases of the cancer (`ncases`). We ask, does either of these two factors affect the number of cases?

First, we load in the dataset. (See how to quickly load some sample data (on website).)

```
from rdatasets import data
data = data('esoph')
data.head()
```

	agegp	alcgp	tobgp	ncases	ncontrols
0	25-34	0-39g/day	0-9g/day	0	40
1	25-34	0-39g/day	10-19	0	10
2	25-34	0-39g/day	20-29	0	6
3	25-34	0-39g/day	30+	0	5
4	25-34	40-79	0-9g/day	0	27

Next, we create a model that includes the response variable we care about, plus the two categorical variables we will be testing. We simply omit the interaction term. (If you wish to include it, see how to do a two-way ANOVA test with interaction.)

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
# C(...) means the variable is categorical, below
model = ols('ncases ~ C(alcgp) + C(agegp)', data = data).fit()
```

A two-way ANOVA with interaction tests the following two null hypotheses.

1. The mean response is the same across all groups of the first factor. (In our example, that says the mean `ncases` is the same for all age groups.)
2. The mean response is the same across all groups of the second factor. (In our example, that says the mean `ncases` is the same for all alcohol consumption groups.)

We choose a value, $0 \leq \alpha \leq 1$, as the Type I Error Rate. Let's let $\alpha = 0.05$ here.

```
sm.stats.anova_lm(model, typ=2)
```

	sum_sq	df	F	PR(>F)
C(alcgp)	52.695287	3.0	4.015660	1.029452e-02
C(agegp)	267.026108	5.0	12.209284	8.907998e-09
Residual	345.557743	79.0	NaN	NaN

The p-value for the alcohol consumption factor is in the first row, final column, 1.029452×10^{-2}. It is less than α, so we can reject the null hypothesis that alcohol consumption does not affect the number of esophageal cancer cases. That is, we have reason to believe that it does affect the number of cases.

The p-value for the age group factor is in the second row, final column, 8.907998×10^{-9}. It is less than α, so we can reject the null hypothesis that age group does not affect the number of esophageal cancer cases. Again, we have reason to believe that it does affect the number of cases.

Content last modified on 24 October 2021.

See a problem? Tell us or edit the source.

How to compute Fisher's confidence intervals

Description

If we run a one-way ANOVA test and find that there is a significant difference between population means, we might want to know which means are actually different from each other. One way to do so is with Fisher's Least Significant Difference Confidence Intervals, which forms a confidence interval for each pair of samples. How do we go about making these confidence intervals?

Solution in Python

How to Data does not yet contain a solution for this task in Python.

How to perform an analysis of covariance (ANCOVA)

Description

Recall that covariates are variables that may be related to the outcome but are unaffected by treatment assignment. In a randomized experiment with one or more observed covariates, an analysis of covariance (ANCOVA) addresses this question: How would the mean outcome in each treatment group change if all groups were equal with respect to the covariate? The goal is to remove any variability in the outcome associated with the covariate from the unexplained variability used to determine statistical significance.

Related tasks:

- How to do a one-way analysis of variance (ANOVA)
- How to compare two nested linear models (on website)
- How to conduct a mixed designs ANOVA (on website)
- How to conduct a repeated measures ANOVA (on website)

Solution in Python

The solution below uses an example dataset about car design and fuel consumption from a 1974 Motor Trend magazine. (See how to quickly load some sample data (on website).)

```
from rdatasets import data
df = data('mtcars')
```

Let's use ANCOVA to check the effect of the engine type (0 = V-shaped, 1 = straight, in the variable `vs`) on the miles per gallon when considering the weight of the car as a covariate. We will use the `ancova` function from the `pingouin` package to conduct the test.

```
from pingouin import ancova
ancova(data=df, dv='mpg', covar='wt', between='vs')
```

	Source	SS	DF	F	p-unc	np2
0	vs	54.228061	1	7.017656	1.292580e-02	0.194839
1	wt	405.425409	1	52.466123	5.632548e-08	0.644024
2	Residual	224.093877	29	NaN	NaN	NaN

```
/opt/conda/lib/python3.9/site-packages/outdated/utils.py:14: OutdatedPackageWarning: The package pingouin is out
Set the environment variable OUTDATED_IGNORE=1 to disable these warnings.
  return warn(
```

The p-value for each variable is in the `p-unc` column.

The p-value for the `wt` variable tests the null hypothesis, "The quantities `wt` and `mpg` are not related." Since it is below 0.05, we reject the null hypothesis, and conclude that `wt` is significant in predicting `mpg`.

The p-value for the `vs` variable tests the null hypothesis, "The quantities `vs` and `mpg` are not related if we hold `wt` constant." Since it is below 0.05, we reject the null hypothesis, and conclude that `vs` is significant in predicting `mpg` even among cars with equal weight (`wt`).

Note: Unfortunately, a two-factor ANCOVA is not possible in pingouin. However, a model with more than one covariate is possible, as you can provide a list as the `covar` parameter when calling `ancova`.

Content last modified on 07 December 2021.

See a problem? Tell us or edit the source.

How to perform post-hoc analysis with Tukey's HSD test

Description

If we run a one-way ANOVA test and find that there is a significant difference between population means, we might want to know which means are actually different from each other. One way to do so is with Tukey's Honestly Significant Differences (HSD) method. It creates confidence intervals for each pair of samples, while controlling for Type I error rate across all pairs. Thus the resulting intervals are a little wider than those produced using Fisher's LSD method. How do we make these confidence intervals, with an appropriate visualization?

Solution in Python

We load here the same data that appears in the solution for how to perform pairwise comparisons (on website). That solution used ANOVA to determine which pairs of groups have significant differences in their means; follow its link for more details.

```
from rdatasets import data
df = data('InsectSprays')
df
```

	count	spray
0	10	A
1	7	A
2	20	A
3	14	A
4	14	A
...	...	...
67	10	F
68	26	F
69	26	F
70	24	F
71	13	F

72 rows × 2 columns

We now want to perform an unplanned comparison test on the data to determine the magnitudes of the differences between pairs of groups. We do this by applying Tukey's HSD approach to perform pairwise comparisons and generate confidence intervals that maintain a specified experiment-wide error rate. Before that, the `pairwise_tukeyhsd` module needs to be imported from the `statsmodels` package.

```
from statsmodels.stats.multicomp import pairwise_tukeyhsd
tukey = pairwise_tukeyhsd(endog=df['count'], groups=df['spray'], alpha=0.05)
print(tukey)
```

```
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
group1 group2 meandiff p-adj   lower    upper  reject
----------------------------------------------------
     A      B   0.8333    0.9  -3.8659  5.5326  False
     A      C -12.4167  0.001 -17.1159 -7.7174   True
     A      D  -9.5833  0.001 -14.2826 -4.8841   True
     A      E    -11.0  0.001 -15.6992 -6.3008   True
     A      F   2.1667  0.728  -2.5326  6.8659  False
     B      C   -13.25  0.001 -17.9492 -8.5508   True
     B      D -10.4167  0.001 -15.1159 -5.7174   True
     B      E -11.8333  0.001 -16.5326 -7.1341   True
     B      F   1.3333    0.9  -3.3659  6.0326  False
     C      D   2.8333 0.4921  -1.8659  7.5326  False
     C      E   1.4167    0.9  -3.2826  6.1159  False
     C      F  14.5833  0.001   9.8841 19.2826   True
     D      E  -1.4167    0.9  -6.1159  3.2826  False
     D      F    11.75  0.001   7.0508 16.4492   True
     E      F  13.1667  0.001   8.4674 17.8659   True
----------------------------------------------------
```

Because the above table contains a lot of information, it's often helpful to visualize these intervals. Python's `statsmodels` package does not have a built-in way to do so, but we can create our own as follows.

```
import matplotlib.pyplot as plt
rows = tukey.summary().data[1:]
plt.hlines( range(len(rows)), [row[4] for row in rows], [row[5] for row in rows] )
plt.vlines( 0, -1, len( rows )-1, linestyles='dashed' )
plt.gca().set_yticks( range( len( rows ) ) )
plt.gca().set_yticklabels( [ f'{x[0]}-{x[1]}' for x in rows ] )
plt.show()
```

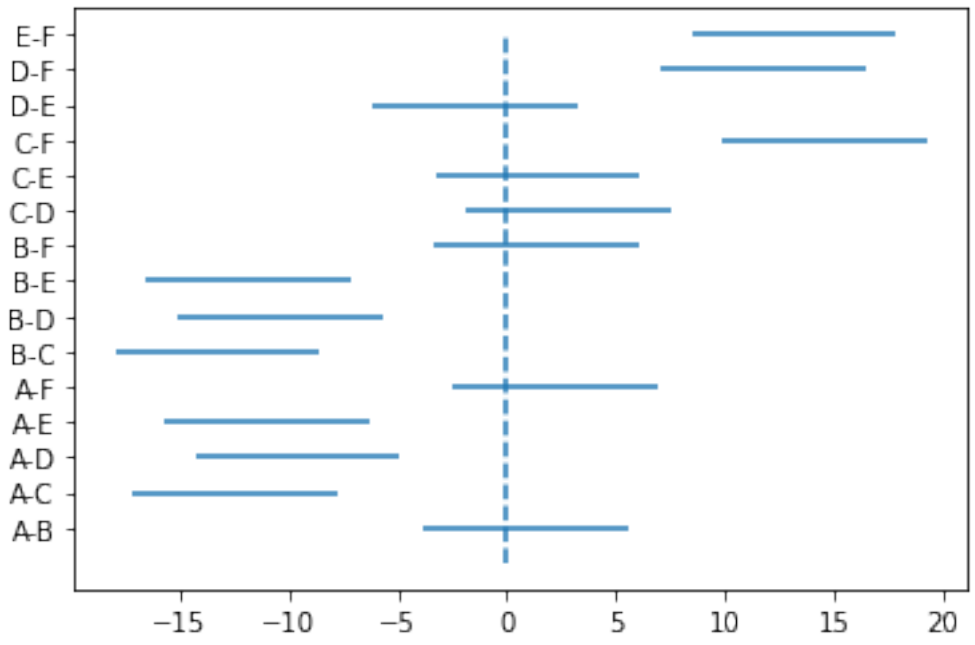

Figure 1: png

Confidence intervals that cross the vertical, dashed line at $x = 0$ are those in which the means across those groups may be equal. Other intervals have mean differences whose 95% confidence intervals do not include zero.

Content last modified on 10 September 2021.

See a problem? Tell us or edit the source.

How to use Bonferroni's Correction method

Description

If we run a one-way ANOVA test and find that there is a significant difference between population means, we might want to know which means are actually different from each other. One way to do so is with the Bonferroni correction. This method runs a t-test for each pair of categories using a conservative confidence level.

Related tasks:

- How to do a one-way analysis of variance (ANOVA)
- How to do a two-sided hypothesis test for two sample means (on website) (which is just an ANOVA with only two samples)
- How to do a Kruskal-Wallis test

Solution in Python

How to Data does not yet contain a solution for this task in Python.

How to fit a linear model to two columns of data

Description

Let's say we have two columns of data, one for a single independent variable x and the other for a single dependent variable y. How can I find the best fit linear model that predicts y based on x?

In other words, what are the model coefficients β_0 and β_1 that give me the best linear model $\hat{y} = \beta_0 + \beta_1 x$ based on my data?

Related tasks:

- How to compute R-squared for a simple linear model
- How to fit a multivariate linear model (on website)
- How to predict the response variable in a linear model

Solution in Python

This solution uses a pandas DataFrame of fake example data. When using this code, replace our fake data with your real data.

Although the solution below uses plain Python lists of data, it also works if the data are stored in NumPy arrays or pandas Series.

```
# Here is the fake data you should replace with your real data.
xs = [ 393, 453, 553, 679, 729, 748, 817 ]
ys = [  24,  25,  27,  36,  55,  68,  84 ]

# We will use SciPy to build the model
import scipy.stats as stats

# If you need the model coefficients stored in variables for later use, do:
model = stats.linregress( xs, ys )
beta0 = model.intercept
beta1 = model.slope

# If you just need to see the coefficients (and some other related data),
# do this alone:
stats.linregress( xs, ys )
```

```
LinregressResult(slope=0.1327195637885226, intercept=-37.32141898334582, rvalue=0.8949574425541466, pvalue=0.006
```

The linear model in this example is approximately $y = 0.133x - 37.32$.

Content last modified on 28 May 2021.

See a problem? Tell us or edit the source.

How to compute a confidence interval for the expected value of a response variable

Description

If we have a simple linear regression model, $y = \beta_0 + \beta_1 x + \epsilon$, where ϵ is some random error, then given any x input, y can be veiwed as a random variable because of ϵ. Let's consider its expected value. How do we construct a confidence interval for that expected value, given a value for the predictor x?

Related tasks:

- How to compute a confidence interval for a mean difference (matched pairs)
- How to compute a confidence interval for a regression coefficient (on website)
- How to compute a confidence interval for a population mean
- How to compute a confidence interval for a single population variance
- How to compute a confidence interval for the difference between two means when both population variances are known
- How to compute a confidence interval for the difference between two means when population variances are unknown
- How to compute a confidence interval for the difference between two proportions
- How to compute a confidence interval for the population proportion
- How to compute a confidence interval for the ratio of two population variances

Solution in Python

How to Data does not yet contain a solution for this task in Python.

How to compute R-squared for a simple linear model

Description

Let's say we have fit a linear model to two columns of data, one for a single independent variable x and the other for a single dependent variable y. How can we compute R^2 for that model, to measure its goodness of fit?

Related tasks:

- How to fit a linear model to two columns of data
- How to compute adjusted R-squared (on website)

Solution in Python

We assume you have already fit a linear model to the data, as in the code below, which is explained fully in a separate task, how to fit a linear model to two columns of data.

```
import scipy.stats as stats
xs = [ 393, 453, 553, 679, 729, 748, 817 ]
ys = [  24,  25,  27,  36,  55,  68,  84 ]
model = stats.linregress( xs, ys )
```

The R value is part of the model object that `stats.linregress` returns.

```
model.rvalue
```

```
0.8949574425541466
```

You can compute R^2 just by squaring it.

```
model.rvalue ** 2
```

```
0.8009488239830586
```

Content last modified on 01 June 2021.

See a problem? Tell us or edit the source.

How to predict the response variable in a linear model

Description

If we have a linear model and a value for each explanatory variable, how do we predict the corresponding value of the response variable?

Related tasks:

- How to fit a linear model to two columns of data
- How to fit a multivariate linear model (on website)

Solution in Python

Let's assume that you've already built a linear model. We do an example below with fake data, but you can use your own actual data. For more information on the following code, see how to fit a multivariate linear model (on website).

```
import pandas as pd
df = pd.DataFrame( {
    'x1' : [ 2,  7,  4,  3, 11, 18,   6, 15,   9,  12],
    'x2' : [ 4,  6, 10,  1, 18, 11,   8, 20,  16,  13],
    'x3' : [11, 16, 20,  6, 14,  8,   5, 23,  13,  10],
    'y'  : [24, 60, 32, 29, 90, 45, 130, 76, 100, 120]
} )

import statsmodels.api as sm
model = sm.OLS( df['y'], sm.add_constant( df[['x1','x2','x3']] ) ).fit()
```

Let's say we want to estimate y given that $x_1 = 5$, $x_2 = 12$, and $x_3 = 50$. We can use the model's `predict()` function as shown below, but we must add an entry for the constant term in the model—we can use any value, but we choose 1.

```
model.predict( [ 1, 5, 12, 50 ] )
```

```
array([-91.71014402])
```

For the given values of the explanatory variables, our predicted response variable is −91.71014402.

Note that if you want to compute the predicted values for all the data on which the model was trained, simply call `model.predict()` with no arguments, and it defaults to using the training data.

```
model.predict()
```

```
array([ 47.5701159 ,  24.35988296,  42.21531274,  47.27613825,
       110.86526185,  70.03097584,  95.12689978,  70.91290879,
       106.52986696,  91.11263692])
```

Content last modified on 07 December 2021.

See a problem? Tell us or edit the source.

How to create a QQ-plot

Description

We often want to know whether a set of data is normally distributed, so that we can deduce what inference tests are appropriate to conduct. If we have a set of data and want to figure out if it comes from a population that follows a normal distribution, one tool that can help is a QQ plot. How do we make and interpret one?

Related tasks:

- How to test data for normality with Pearson's chi-squared test
- How to test data for normality with the D'Agostino-Pearson test
- How to test data for normality with the Jarque-Bera test

Solution in Python

We're going to use some fake data here by generating random numbers, but you can replace our fake data with your real data in the code below.

```
# Replace this with your data, such as a variable or column in a DataFrame
import numpy as np
values = np.random.normal(0, 1, 50)  # 50 random values
```

If the data is normally distributed, then we expect that the QQ plot will show the observed values (blue dots) falling very clsoe to the red line (the quantiles for the normal distribution).

```
from scipy import stats
import matplotlib.pyplot as plt

stats.probplot(values, dist="norm", plot=plt)
plt.show()
```

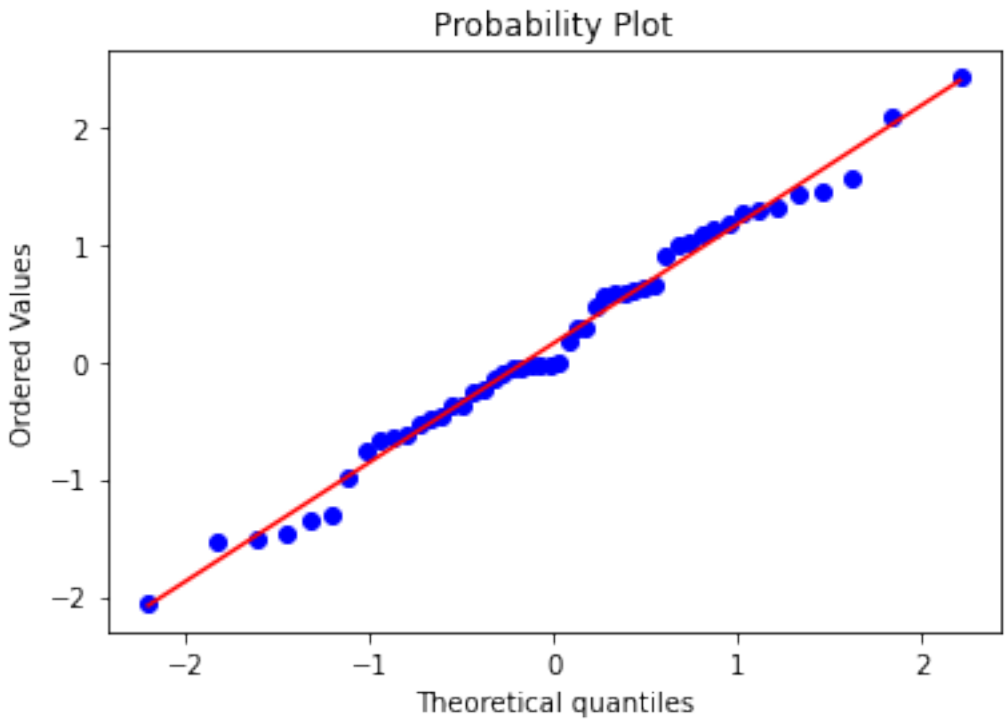

Figure 2: png

Our observed values fall pretty close to the reference line. In this case, we expected that, because we created fake data that was normally distributed. But for real data, it may not stay so close to the red line.

Content last modified on 14 September 2021.

See a problem? Tell us or edit the source.

How to test data for normality with Pearson's chi-squared test

Description

We often want to know whether a set of data is normally distributed, so that we can deduce what inference tests are appropriate to conduct. If we have a set of data and want to figure out if it comes from a population that follows a normal distribution, one tool that can help is Pearson's χ^2 test. How do we perform it?

Related tasks:

- How to create a QQ-plot
- How to test data for normality with the D'Agostino-Pearson test
- How to test data for normality with the Jarque-Bera test

Solution in Python

How to Data does not yet contain a solution for this task in Python.

How to test data for normality with the D'Agostino-Pearson test

Description

We often want to know whether a set of data is normally distributed, so that we can deduce what inference tests are appropriate to conduct. If we have a set of data and want to figure out if it comes from a population that follows a normal distribution, one tool that can help is the D'Agostino-Pearson test (sometimes also called the D'Agostino-Pearson omnibus test, or the D'Agostino-Pearson k^2 test). How do we perform it?

Related tasks:

- How to create a QQ-plot
- How to test data for normality with Pearson's chi-squared test
- How to test data for normality with the Jarque-Bera test

Solution in Python

We're going to use some fake restaurant data, but you can replace our fake data with your real data in the code below. The values in our fake data represent the amount of money that customers spent on a Sunday morning at the restaurant.

```
import numpy as np

# Replace your data here
spending = [34, 12, 19, 56, 54, 34, 45, 37, 13, 22, 65, 19,
            16, 45, 19, 50, 36, 23, 28, 56, 40, 61, 45, 47, 37]

np.mean(spending), np.std(spending, ddof=1)
```

```
(36.52, 15.772127313713899)
```

We will now conduct a test of the following null hypothesis: The data comes from a population that is normally distributed with mean 36.52 and standard deviation 15.77.

We will use a value $\alpha = 0.05$ as our Type I error rate. The `normaltest()` function in SciPy's `stats` package can perform the D'Agostino-Pearson test for normality, which uses the skew and kurtosis of the data.

```
from scipy import stats
stats.normaltest(spending)
```

```
NormaltestResult(statistic=3.0866213696851097, pvalue=0.21367252674488552)
```

The p-value is apprximately 0.21367, which is greater than $\alpha = 0.05$, so we fail to reject our null hypothesis. We would continue to operate under our original assumption that the data come from a normally distributed population.

Content last modified on 30 November 2021.

See a problem? Tell us or edit the source.

How to test data for normality with the Jarque-Bera test

Description

We often want to know whether a set of data is normally distributed, so that we can deduce what inference tests are appropriate to conduct. If we have a set of data and want to figure out if it comes from a population that follows a normal distribution, one tool that can help is the Jarque-Bera test for normality. How do we perform it?

Related tasks:

- How to create a QQ-plot
- How to test data for normality with the D'Agostino-Pearson test
- How to test data for normality with Pearson's chi-squared test

Solution in Python

We're going to use some fake restaurant data, but you can replace our fake data with your real data in the code below. The values in our fake data represent the amount of money that customers spent on a Sunday morning at the restaurant.

```
# Replace your data here
spending = [ 34, 12, 19, 56, 54, 34, 45, 37, 13, 22, 65, 19,
             16, 45, 19, 50, 36, 23, 28, 56, 40, 61, 45, 47, 37 ]
```

If we assume that the skewness coefficient S and the kurtosis coefficient K are both equal to zero, then our null hypothesis is $H_0 : S = K = 0$, or that the sample data comes from a normal distribution. We choose a value $0 \leq \alpha \leq 1$ as our Type 1 error rate. We'll let α be 0.05 here.

We can use the `jarque_bera()` function in SciPy's stats package to run the hypothesis test.

```
from scipy import stats
stats.jarque_bera( spending )
```

```
Jarque_beraResult(statistic=1.3347292970972013, pvalue=0.5130588882194845)
```

Our p-value of about 0.5131 is greater than α, so we fail to reject our null hypothesis. We would continue to operate under our original assumption that the data come from a normally distributed population.

Content last modified on 23 September 2021.

See a problem? Tell us or edit the source.