Bentley University GR521 in Python

Content extracted from the How to Data Website

This PDF generated on 30 November 2021

Contents

GR521 is a graduate Managerial Statistics course at Bentley University. The description from the course catalog can be found here.

Mathematical topics include random variables, discrete and continuous probability distributions, confidence intervals, hypothesis testing, single-variable linear models, and optionally advanced topics such as data mining, time permitting.

Basics

Random variables and probability distributions

Confidence intervals and hypothesis testing

Linear modeling

Other end-of-semester topics, time permitting

Content last modified on 30 November 2021.

How to do basic mathematical computations

Description

How do we write the most common mathematical operations in a given piece of software? For example, how do we write multiplication, or exponentiation, or logarithms, in Python vs. R vs. Excel, and so on?

Solution in Python

This answer assumes you have imported NumPy as follows.

```
import numpy as np
```

Mathematical notation	Python code	Requires NumPy?
$x+y$	`x+y`	no
$x-y$	`x-y`	no
xy	`x*y`	no
$\frac{x}{y}$	`x/y`	no
$\left\lfloor\frac{x}{y}\right\rfloor$	`x//y`	no
$\left\lfloor\frac{x}{y}\right\rfloor$	`np.floor_divide(x,y)`	yes
remainder of $x \div y$	`x%y`	no
remainder of $x \div y$	`np.remainder(x,y)`	yes
x^y	`x**y`	no
$\|x\|$	`abs(x)`	no
$\|x\|$	`np.abs(x)`	yes
$\ln x$	`np.log(x)`	yes
$\log_a b$	`np.log(b)/np.log(a)`	yes
e^x	`np.exp(x)`	yes
π	`np.pi`	yes
$\sin x$	`np.sin(x)`	yes
$\sin^{-1} x$	`np.asin(x)`	yes
$\sqrt{x}$	`x**0.5`	no
$\sqrt{x}$	`np.sqrt(x)`	yes

Other trigonometric functions are also available besides just `np.sin`, including `np.cos`, `np.tan`, etc.

NumPy automatically applies any of these functions to all entries of a NumPy array or pandas Series, but the built-in Python functions do not have this feature. For example, to square all numbers in an array, see below.

```
import numpy as np
example_array = np.array( [ -3, 2, 0.5, -1, 10, 9.2, -3.3 ] )
example_array ** 2
```

```
array([  9.  ,   4.  ,   0.25,   1.  , 100.  ,  84.64,  10.89])
```

Content last modified on 14 July 2021.

See a problem? Tell us or edit the source.

How to compute summary statistics

Description

The phrase "summary statistics" usually refers to a common set of simple computations that can be done about any dataset, including mean, median, variance, and some of the others shown below.

Related tasks:

- How to summarize a column (on website)
- How to summarize and compare data by groups (on website)

Solution in Python

We first load a famous dataset, Fisher's irises, just to have some example data to use in the code that follows. (See how to quickly load some sample data (on website).)

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

How big is the dataset? The output shows number of rows then number of columns.

```
df.shape
```

```
(150, 5)
```

What are the columns and their data types? Are any values missing?

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Sepal.Length  150 non-null    float64
 1   Sepal.Width   150 non-null    float64
 2   Petal.Length  150 non-null    float64
 3   Petal.Width   150 non-null    float64
 4   Species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

What do the first few rows look like?

```
df.head()  # Default is 5, but you can do df.head(20) or any number.
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

The easiest way to get summary statistics for a pandas DataFrame is with the `describe` function.

```
df.describe()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

The individual statistics are the row headings, and the numeric columns from the original dataset are listed across the top.

We can also compute these statistics (and others) one at a time for any given set of data points. Here, we let `xs` be one column from the above DataFrame, but you could use any NumPy array or pandas DataFrame instead.

```
xs = df['Sepal.Length']

import numpy as np

np.mean( xs )           # mean, or average, or center of mass
np.median( xs )         # 50th percentile
np.percentile( xs, 25 ) # compute any percentile, such as the 25th
np.var( xs )            # variance
np.std( xs )            # standard deviation, the square root of the variance
np.sort( xs )           # data in increasing order
np.sum( xs )            # sum, or total
```

Content last modified on 26 July 2021.

See a problem? Tell us or edit the source.

How to generate random values from a distribution

Description

There are many famous continuous probability distributions, such as the normal and exponential distributions. How can we get access to them in software, to generate random values from a chosen distribution?

Related tasks:

- How to compute probabilities from a distribution
- How to plot continuous probability distributions
- How to plot discrete probability distributions

Solution in Python

You can import many different random variables from SciPy's `stats` module. The full list of them is online here.

Regardless of whether the distribution is discrete or continuous, the appropriate function to call is `rvs`, which stands for "random values." Here are two examples.

Using a **normal distribution:**

```
from scipy import stats
X = stats.norm( 10, 5 )  # normal random variable with μ=10 and σ=5
X.rvs( 20 )              # 20 random values from X
```

```
array([13.1064664 , 10.91812199, 12.38577231,  7.56388524, 12.76801726,
       15.78585088,  9.91561424, 13.5402871 , 11.99767642,  7.34415886,
       11.74238554, 16.44509778,  9.19625154,  4.53029516, 14.62388518,
        9.32034502, 13.212503  , 10.84916965,  8.68643229, 12.06198166])
```

Using a **uniform distribution:**

(Note that in SciPy, the uniform distribution needs a "location," which is where the sample space begins—in this case 50—and a "scale," which is the width of the sample space—in this case 10.)

```
from scipy import stats
X = stats.uniform( 50, 10 )  # uniform random variable on the interval [50,60]
X.rvs( 20 )                  # 20 random values from X
```

```
array([59.25472277, 52.6586181 , 50.90122306, 56.23730799, 58.86849267,
       52.83377599, 55.42103998, 52.6689307 , 56.5773815 , 50.87022971,
       58.01072153, 55.56971665, 58.54310671, 52.54336368, 57.76300137,
       55.58967496, 50.20936319, 53.47493391, 52.96509439, 57.47859003])
```

Content last modified on 27 May 2021.

See a problem? Tell us or edit the source.

How to compute probabilities from a distribution

Description

There are many famous continuous probability distributions, such as the normal and exponential distributions. How can we get access to them in software, to compute the probability of a value/values occurring?

Related tasks:

- How to generate random values from a distribution
- How to plot continuous probability distributions
- How to plot discrete probability distributions

Solution in Python

You can import many different random variables from SciPy's `stats` module. The full list of them is online here.

To compute a probability from a **discrete** distribution, create a random variable, then use its Probability Mass Function, `pmf`.

```
from scipy import stats

# Create a binomial random variable with 10 trials
# and probability 0.5 of success on each trial
X = stats.binom( 10, 0.5 )

# What is the probability of exactly 3 successes?
X.pmf( 3 )
```

```
0.1171875
```

To compute a probability from a **continuous** distribution, create a random variable, then use its Cumulative Density Function, `cdf`. You can only compute the probability that a random value will fall in an interval $[a, b]$, not the probability that it will equal a specific value.

```
from scipy import stats

# Create a normal random variable with mean μ=10 and standard deviation σ=5
X = stats.norm( 10, 5 )

# What is the probability of the value lying in the interval [12,13]?
X.cdf( 13 ) - X.cdf( 12 )
```

```
0.07032514063960227
```

Content last modified on 27 May 2021.

See a problem? Tell us or edit the source.

How to plot continuous probability distributions

Description

There are many famous continuous probability distributions, such as the normal and exponential distributions. How can we get access to them in software, to plot the distribution as a curve?

Related tasks:

- How to generate random values from a distribution
- How to compute probabilities from a distribution
- How to plot discrete probability distributions

Solution in Python

You can import many different random variables from SciPy's `stats` module. The full list of them is online here.

The challenge with plotting a random variable is knowing the appropriate sample space, because some random variables have sample spaces of infinite width, which cannot be plotted.

But we can just ask SciPy to show us the central 99.98% of a continuous distribution, which is almost always indistinguishable to the human eye from the entire distribution.

We style the plot below so that it is clear the sample space is continuous.

```
from scipy import stats
X = stats.norm( 10, 5 )      # use a normal distribution with μ=10 and σ=5

xmin = X.ppf( 0.0001 )       # compute min x as the 0.0001 quantile
xmax = X.ppf( 0.9999 )       # compute max x as the 0.9999 quantile
import numpy as np
xs = np.linspace( xmin, xmax, 100 )  # create 100 x values in that range

import matplotlib.pyplot as plt
plt.plot( xs, X.pdf( xs ) )  # plot the shape of the distribution
plt.show()
```

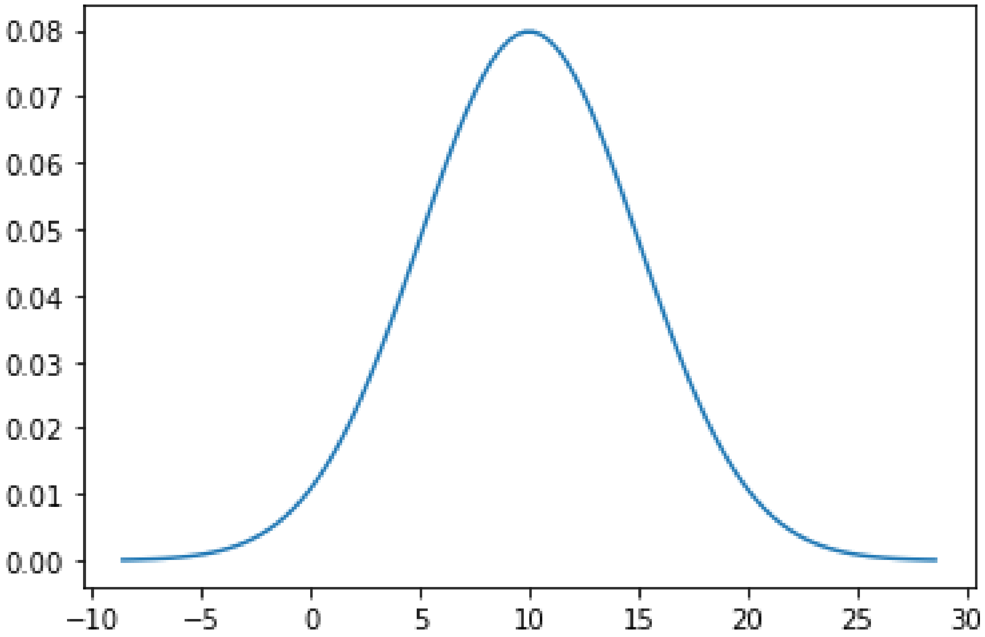

Figure 1: png

Content last modified on 28 May 2021.

See a problem? Tell us or edit the source.

How to plot discrete probability distributions

Description

There are many famous discrete probability distributions, such as the binomial and geometric distributions. How can we get access to them in software, to plot the distribution as a series of points?

Related tasks:

- How to generate random values from a distribution
- How to compute probabilities from a distribution
- How to plot continuous probability distributions

Solution in Python

You can import many different random variables from SciPy's `stats` module. The full list of them is online here.

The challenge with plotting a random variable is knowing the appropriate sample space, because some random variables have sample spaces of infinite width, which cannot be plotted.

The example below uses a geometric distribution, whose sample space is $\{1, 2, 3, \dots\}$. We specify that we just want to use x values in the set $\{1, 2, \dots, 10\}$. (In some software, the geometric distribution's sample space begins at 0, but not in SciPy.)

We style the plot below so that it is clear the sample space is discrete.

```python
from scipy import stats
X = stats.geom( 0.5 )     # use a geometric distribution with p=0.5

import numpy as np
xs = np.arange( 1, 11 )   # specify the range to be 1,2,3,...,10

import matplotlib.pyplot as plt
ys = X.pmf( xs )          # compute the shape of the distribution
plt.plot( xs, ys, 'o' )   # plot circles...
plt.vlines( xs, 0, ys )   # ...and lines
plt.ylim( bottom=0 )      # ensure sensible bottom border
plt.show()
```

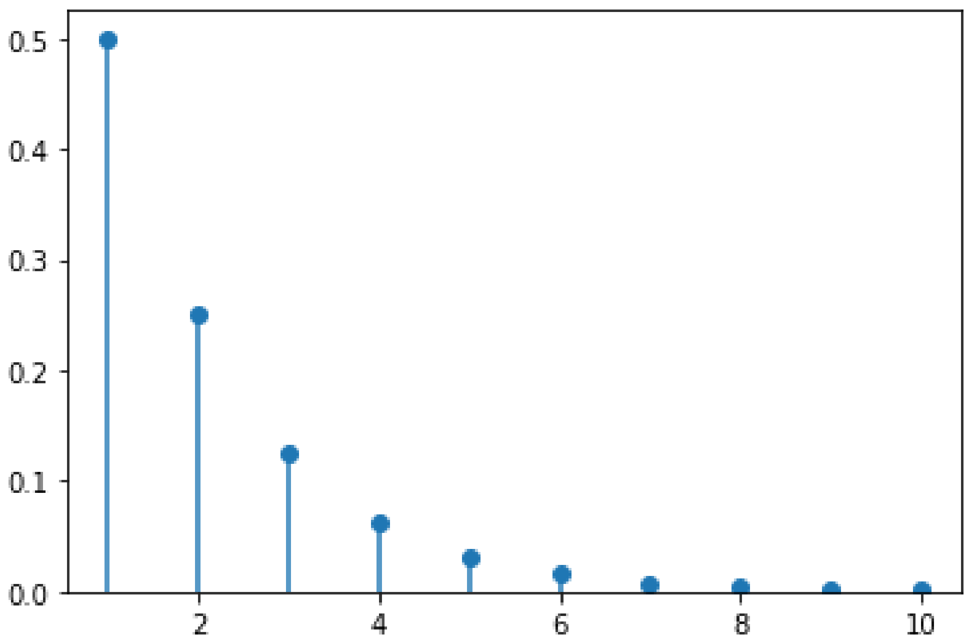

Figure 2: png

Content last modified on 28 May 2021.

See a problem? Tell us or edit the source.

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 (on website).

Related tasks:

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

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 do a two-sided hypothesis test for a sample mean

Description

Say we have a population whose mean μ is known. We take a sample $x_1, \ldots, x_n$ and compute its mean, $\bar{x}$. We then ask whether this sample is significantly different from the population at large, that is, is $\mu = \bar{x}$?

Related tasks:

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

Solution in Python

This is a two-sided test with the null hypothesis $H_0 : \mu = \bar{x}$. We choose a value $0 \le \alpha \le 1$ as the probability of a Type I error (false positive, finding we should reject H_0 when it's actually true).

```
from scipy import stats

# Replace these first three lines with the values from your situation.
alpha = 0.05
pop_mean = 10
sample = [ 9, 12, 14, 8, 13 ]

# Run a one-sample t-test and print out alpha, the p value,
# and whether the comparison says to reject the null hypothesis.
t_statistic, p_value = stats.ttest_1samp( sample, pop_mean )
reject_H0 = p_value < alpha
alpha, p_value, reject_H0
```

```
(0.05, 0.35845634462296455, False)
```

In this case, the sample does not give us enough information to reject the null hypothesis. We would continue to assume that the sample is like the population, $\mu = \bar{x}$.

Content last modified on 05 October 2021.

See a problem? Tell us or edit the source.

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

Description

If we have two samples, $x_1, \ldots, x_n$ and $x'_1, \ldots, x'_m$, and we compute the mean of each one, we might want to ask whether the two means seem approximately equal. Or more precisely, is their difference statistically significant at a given level?

Related tasks:

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

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 $H_0 : \bar{x}_1 = \bar{x}_2$. We choose a value $0 \le \alpha \le 1$ as the probability of a Type I error (false positive, finding we should reject H_0 when it's actually true).

```python
from scipy import stats

# Replace these first three lines with the values from your situation.
alpha = 0.10
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.
t_statistic, p_value = stats.ttest_ind( sample1, sample2, equal_var=False )
reject_H0 = p_value < alpha
alpha, p_value, reject_H0
```

```
(0.1, 0.050972837418476934, True)
```

In this case, the samples give us enough evidence to reject the null hypothesis at the $\alpha = 0.10$ level. The data suggest that $\bar{x}_1 \neq \bar{x}_2$.

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 28 May 2021.

See a problem? Tell us or edit the source.

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 multiple linear regression model (on website)
- How to predict the response variable in a linear model (on website)

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 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 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 (which is just an ANOVA with only two samples)
- How to do a two-way ANOVA test with interaction (on website)
- How to do a two-way ANOVA test without interaction (on website)
- 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) (on website)
- How to do a Kruskal-Wallis test (on website)

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 28 May 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.