Statistics Formula Compendium

Rune Skov Christensen

Sebastian Hvid Olavsgaard

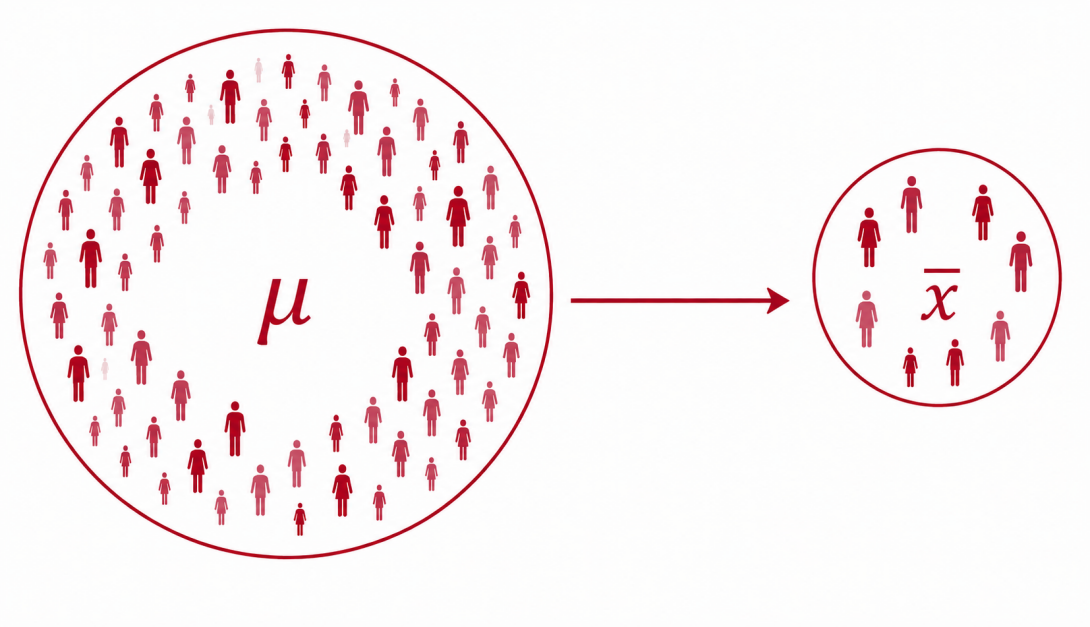


Table of Contents

1 Descriptive statistics

Where n denotes the number of observations in the data sample, and x_i denotes the ith observation.

Collect data in `Python`

```
data = np.array([...])
```

	Description	Formula	Python
	Sample average:	$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$	`np.mean(x)`
	Sample variance:	$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$	`np.var(x, ddof=1)`
	Sample standard deviation:	$s = \sqrt{s^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$	`np.std(x, ddof=1)`
	Coefficient of variation:	$CV = \frac{s}{\bar{x}}$	`np.std(x, ddof=1) / np.mean(x)`
	Median:	$Q_2 = \begin{cases} x_{(\frac{n+1}{2})}, & \text{if } n \text{ is odd} \\ \frac{x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}}{2}, & \text{if } n \text{ is even} \end{cases}$	`np.median(x)`
	Quantile/percentile:	$q_p = \begin{cases} \frac{x_{(np)} + x_{(np+1)}}{2}, & \text{if } np \text{ is integer} \\ x_{(\lceil np \rceil)}, & \text{if } np \text{ is not integer} \end{cases}$	`np.quantile(x, q)` where `q` $\in [0, 1]$ Alternatively: `np.percentile(x, q)` where `q` $\in [0, 100]$

	Description	Formula	Python

	Quartiles:	$Q_0 = q_{0.00} =$ minimum
		$Q_1 = q_{0.25} =$ lower quartile
		$Q_2 = q_{0.50} =$ median
		$Q_3 = q_{0.75} =$ upper quartile
		$Q_4 = q_{1.00} =$ maximum
			`np.quantile(x, 0.00)`
			`np.quantile(x, 0.25)`
			`np.quantile(x, 0.50)`
			`np.quantile(x, 0.75)`
			`np.quantile(x, 1.00)`

	Interquartile range:	$IQR = Q_3 - Q_1$	`stats.iqr(x)`
	Range:	$\text{Range} = \max - \min = Q_4 - Q_0$	`np.ptp(x)`
	Sample covariance: Also denoted: $Cov(x, y)$	$s_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$	`np.cov(x,y,ddof=1)[0,1]`
	Sample correlation coefficient (r): Note: $r = \hat{\rho}(x, y) = \hat{\rho}_{xy}$ $r \in [-1, 1]$	$r = \frac{1}{n-1}\sum_{i=1}^{n}\frac{(x_i - \bar{x})}{s_x}\frac{(y_i - \bar{y})}{s_y} = \frac{s_{xy}}{s_x s_y}$	`np.corrcoef(x, y)[0,1]`

2 Probability distributions

Discrete distributions

The random variable takes values in a finite or countably infinite set of possible outcomes.

General formulas for discrete random variables:

	Description	Formula	Python

	Probability density function (pdf):
	
	Also known as the *probability mass function* (pmf).
		$f(x) = P(X = x)$
		
		Note: $f(x) \geq 0$ and $\sum_{\text{all x}} f(x) = 1$
			`stats.binom.pmf()`
			`stats.hypergeom.pmf()`
			`stats.poisson.pmf()`

	Cumulative distribution function (cdf):	$F(x) = P(X \leq x) = \sum_{t \leq x} P(X = t)$	`stats.binom.cdf()`
			`stats.hypergeom.cdf()`
			`stats.poisson.cdf()`

	Mean of a discrete random variable	$\mu = E[X] = \sum_{\text{all x}} x f(x)$	`stats.binom.mean()`
			`stats.hypergeom.mean()`
			`stats.poisson.mean()`

	Variance of a discrete random variable X:	$\sigma^2 = V[X] = E[(X - \mu)^2] = \sum_{\text{all x}} (x - \mu)^2 f(x)$	`stats.binom.var()`
			`stats.hypergeom.var()`
			`stats.poisson.var()`

Combinatorics

	Description	Formula	Python
	Binomial coefficient: Number of ways to choose b items from a total set of a items without regard to order.	$\binom{a}{b} = \frac{a!}{b!(a-b)!}$	

Binomial distribution $X \sim B(n, p)$

Describes the number of successes X in n independent trials, where each trial has two possible outcomes: success with probability p and failure with probability $1 - p$. The probability p is the same for every trial.

$f(x)$	$F(x)$	$E[X]$	$V[X]$
$\binom{n}{x} p^x (1-p)^{n-x}$	$\sum_{k=0}^{x} \binom{n}{k} p^k (1-p)^{n-k}$	np	$np(1-p)$

Python functions

```
stats.binom
.pmf(k, n, p)
.cdf(k, n, p)
.mean(n, p)
.var(n, p)
.std(n, p)
.ppf(q, n, p)
.rvs(n, p, size)
```

Notation

Book	Python	Meaning
x	`k`	number of successes
n	`n`	number of trials
p	`p`	probability of success

Hypergeometric distribution $X \sim H(n, a, N)$

Describes the probability of obtaining a certain number of successes X when sampling without replacement from a finite population N that contains a fixed number of successes a and failures $N - a$.

$f(x)$	$F(x)$	$E[X]$	$V[X]$
$\frac{\binom{a}{x}\binom{N-a}{n-x}}{\binom{N}{n}}$	$\sum_{k=0}^{\lfloor x \rfloor} \frac{\binom{a}{k}\binom{N-a}{n-k}}{\binom{N}{n}}$	$n\frac{a}{N}$	$n\frac{a}{N}\frac{N-a}{N}\frac{N-n}{N-1}$

Python functions

```
stats.hypergeom
.pmf(k, M, n, N)
.cdf(k, M, n, N)
.mean(M, n, N)
.var(M, n, N)
.std(M, n, N)
.ppf(q, M, n, N)
.rvs(M, n, N, size)
```

Notation

Book	Python	Meaning
x	`k`	number of successes in sample
N	`M`	population size
a	`n`	number of successes in pop.
n	`N`	number of draws

Poisson distribution $X \sim Po(\lambda)$

Describes the number of events X that occur in a given time period or region where the events occur independently and with a constant average rate λ. It can be interpreted as events per unit of time or as a count per interval.

$f(x)$	$F(x)$	$E[X]$	$V[X]$
$\frac{\lambda^x}{x!}e^{-\lambda}$	$\sum_{k=0}^{\lfloor x \rfloor} \frac{\lambda^k}{k!}e^{-\lambda}$	λ	λ

Python functions

```
stats.poisson
.pmf(k, mu)
.cdf(k, mu)
.mean(mu)
.var(mu)
.std(mu)
.ppf(q, mu)
.rvs(mu, size)
```

Notation

Book	Python	Meaning
x	`k`	number of occurrences
λ	`mu`	average rate

Continuous distributions

The random variable can take values over a continuum of possible outcomes.

General formulas for continuous random variables

	Description	Formula	Python

	Probability density function (pdf):	$\int_a^b f(x)\,dx = P(a < X \leq b)$
		
		Note: $f(x) \geq 0$ and $\int_{-\infty}^{\infty} f(x)\,dx = 1$
			`stats.norm.pdf()`
			`stats.uniform.pdf()`
			`stats.expon.pdf()`
			`stats.lognorm.pdf()`
			`stats.chi2.pdf()`
			`stats.t.pdf()`
			`stats.f.pdf()`

	Cumulative distribution function (cdf):	$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(u)\,du$	`stats.norm.cdf()`
			`stats.uniform.cdf()`
			`stats.expon.cdf()`
			`stats.lognorm.cdf()`
			`stats.chi2.cdf()`
			`stats.t.cdf()`
			`stats.f.cdf()`

	Mean of a continuous random variable X:	$\mu = E[X] = \int_{-\infty}^{\infty} x f(x)\,dx$	`stats.norm.mean()`
			`stats.uniform.mean()`
			`stats.expon.mean()`
			`stats.lognorm.mean()`
			`stats.chi2.mean()`
			`stats.t.mean()`
			`stats.f.mean()`

	Variance of a continuous random variable X:	$\sigma^2 = E[(X-\mu)^2] = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)\,dx$	`stats.norm.var()`
			`stats.uniform.var()`
			`stats.expon.var()`
			`stats.lognorm.var()`
			`stats.chi2.var()`
			`stats.t.var()`
			`stats.f.var()`

Normal distribution $X \sim \mathcal{N}(\mu, \sigma^2)$

Describes a random variable X that follows a characteristic bell-shaped distribution. The distribution is symmetric about the mean and has spread σ.

$f(x)$	$F(x)$	$E[X]$	$V[X]$
$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$	$\int_{-\infty}^{x} f(u)\,du$	μ	σ^2

Python functions

```
stats.norm
.pdf(x, loc, scale)
.cdf(x, loc, scale)
.mean(loc, scale)
.var(loc, scale)
.std(loc, scale)
.ppf(q, loc, scale)
.rvs(loc, scale, size)
```

Notation

Book	Python	Meaning
x	`x`	observed value
μ	`loc`	mean
σ	`scale`	standard deviation

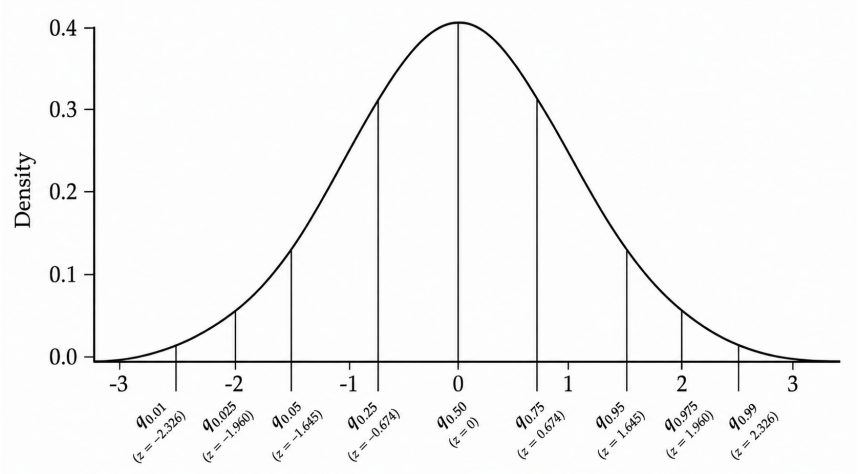
Standard normal distribution

A stochastic variable X from a given distribution can be transformed (standardized) into a standard normal distribution variable via:

$$Z = \frac{X - \mu}{\sigma} \sim N(0, 1^2)$$

Important percentiles and quantiles of the standard normal distribution

Percentile	1%	2.275%	2.5%	5%	15.866%	25%	50%	75%	84.134%	95%	97.5%	97.725%	99%
Quantile	0.01	0.0228	0.025	0.05	0.1587	0.25	0.50	0.75	0.8413	0.95	0.975	0.9772	0.99
Z-score	-2.326	-2.000	-1.960	-1.645	-1.000	-0.674	0.000	0.674	1.000	1.645	1.960	2.000	2.326



Notable quantiles of the standard normal distribution

Lognormal distribution $X \sim \text{LN}(\alpha, \beta^2)$

Describes a positive random variable X whose natural logarithm follows a normal distribution. The distribution is asymmetric and right-skewed.

$f(x)$	$F(x)$	$E[X]$	$V[X]$
$\frac{1}{x\beta\sqrt{2\pi}} \exp\left(-\frac{(\ln(x)-\alpha)^2}{2\beta^2}\right), x > 0$	$\int_0^x f(u)\,du, x > 0$	$e^{\alpha+\frac{\beta^2}{2}}$	$e^{2\alpha+\beta^2}\left(e^{\beta^2}-1\right)$

Python functions

```
stats.lognorm
.pdf(x, s, loc, scale)
.cdf(x, s, loc, scale)
.mean(s, loc, scale)
.var(s, loc, scale)
.std(s, loc, scale)
.ppf(q, s, loc, scale)
.rvs(s, loc, scale, size)
```

Notation

Book	Python	Meaning
x	`x`	observed value
e^{α}	`scale`	scale parameter
β	`s`	std of $\ln(X)$
α		mean of $\ln(X)$

Transformation from Lognormal to Normal

A lognormal distributed variable $X \sim LN(\alpha, \beta^2)$ can be transformed to a normal distributed random variable via: $Y = ln(X) \sim N(\alpha, \beta^2)$

Exponential distribution $X \sim \text{Exp}(\lambda)$

Describes a positive random variable X representing waiting times or lifetimes. It is commonly used to model the time between events in a Poisson process.

$f(x)$	$F(x)$	$E[X]$	$V[X]$
$\lambda e^{-\lambda x}, \quad x \geq 0$	$1 - e^{-\lambda x}, \quad x \geq 0$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$

Python functions

```
stats.expon
.pdf(x, loc, scale)
.cdf(x, loc, scale)
.mean(loc, scale)
.var(loc, scale)
.std(loc, scale)
.ppf(q, loc, scale)
.rvs(loc, scale, size)
```

Notation

Book	Python	Meaning
x	`x`	waiting time
$1/\lambda$	`scale`	mean waiting time
λ		rate parameter

Uniform distribution $X \sim U(\alpha, \beta)$

Describes a random variable X whose values are equally likely within the interval $[\alpha, \beta]$.

$f(x)$	$F(x)$	$E[X]$	$V[X]$
$\begin{cases} 0, & x < \alpha \\ \dfrac{1}{\beta - \alpha}, & \alpha \leq x \leq \beta \\ 0, & x > \beta \end{cases}$	$\begin{cases} 0, & x < \alpha \\ \dfrac{x - \alpha}{\beta - \alpha}, & \alpha \leq x \leq \beta \\ 1, & x > \beta \end{cases}$	$\dfrac{\alpha + \beta}{2}$	$\dfrac{(\beta - \alpha)^2}{12}$

Python functions

```
stats.uniform
.rvs(loc, scale, size)
.pdf(x, loc, scale)
.cdf(x, loc, scale)
.mean(loc, scale)
.var(loc, scale)
.std(loc, scale)
.ppf(q, loc, scale)
```

Notation

Book	Python	Meaning
x	`x`	observed value
α	`loc`	lower limit
β	`loc + scale`	upper limit

Chi-squared distribution $X \sim \chi^2(\nu)$

Describes a random variable that can be expressed as $X = \sum_{i=1}^{\nu} Z_i^2$, the sum of ν squared independent standard normally distributed variables. In the context of a normally distributed sample, it also arises as $X = \dfrac{(n-1)S^2}{\sigma^2} \sim \chi^2(n-1)$, where S^2 is the sample variance and σ^2 is the population variance. It is only defined for non-negative values and is *right-skewed*, with its shape depending on the degrees of freedom ν.

$f(x)$	$F(x)$	$E[X]$	$V[X]$
$\dfrac{1}{2^{\nu/2}\Gamma(\nu/2)} x^{\frac{\nu}{2}-1} e^{-x/2}, x \geq 0$	$\displaystyle\int_0^x f(u)\,du, \quad x \geq 0$	ν	2ν

Python functions

```
stats.chi2
.pdf(x, df, loc, scale)
.cdf(x, df, loc, scale)
.mean(df, loc, scale)
.var(df, loc, scale)
.std(df, loc, scale)
.ppf(q, df, loc, scale)
.rvs(df, loc, scale, size)
```

Notation

Book	Python	Meaning
x	`x`	observed value
ν	`df`	degrees of freedom
$\Gamma(I)$		gamma function

Student's t-distribution $X \sim t(\nu)$

Describes a random variable $X = \frac{Z}{\sqrt{Y/\nu}}$, where $Z \sim \mathcal{N}(0,1)$ and $Y \sim \chi^2(\nu)$ are independent. In statistical inference, it arises as $X = \frac{\bar{X}-\mu}{S/\sqrt{n}} \sim t(n-1)$ for an i.i.d. normal sample. It is symmetric about 0 with heavier tails than the standard normal, with shape depending on ν.

$f(x)$	$F(x)$	$E[X]$	$V[X]$
$\frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\,\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$	$\int_{-\infty}^{x} f(u)\,du$	$0, \quad \nu > 1$	$\frac{\nu}{\nu-2}, \quad \nu > 2$

Python functions

```
stats.t
.pdf(x, df, loc, scale)
.cdf(x, df, loc, scale)
.mean(df, loc, scale)
.var(df, loc, scale)
.std(df, loc, scale)
.ppf(q, df, loc, scale)
.rvs(df, loc, scale, size)
```

Notation

Book	Python	Meaning
x	`x`	observed value
ν	`df`	degrees of freedom
$\Gamma(I)$		gamma function

F-distribution $X \sim F(\nu_1, \nu_2)$

Describes a random variable $X = \dfrac{S_1^2/\sigma_1^2}{S_2^2/\sigma_2^2} \sim F(n_1-1, n_2-1)$ for two independent normal samples. More generally, $X = \dfrac{U/\nu_1}{V/\nu_2}$, where $U \sim \chi^2(\nu_1)$ and $V \sim \chi^2(\nu_2)$ are independent. It is non-negative and right-skewed, with shape depending on ν_1, ν_2.

$f(x)$	$F(x)$	$E[X]$	$V[X]$
$\frac{1}{B\left(\frac{\nu_1}{2},\frac{\nu_2}{2}\right)}\left(\frac{\nu_1}{\nu_2}\right)^{\nu_1/2} \times x^{\frac{\nu_1}{2}-1}\left(1+\frac{\nu_1}{\nu_2}x\right)^{-\frac{\nu_1+\nu_2}{2}}, \quad x \geq 0,\ \nu_1,\nu_2 > 0$	$\int_0^x f(u)\,du, x \geq 0$	$\frac{\nu_2}{\nu_2-2}, \nu_2 > 2$	$\frac{2\nu_2^2(\nu_1+\nu_2-2)}{\nu_1(\nu_2-2)^2(\nu_2-4)}, \nu_2 > 4$

Python functions

```
stats.f
.pdf(x, dfn, dfd, loc, scale)
.cdf(x, dfn, dfd, loc, scale)
.mean(dfn, dfd, loc, scale)
.var(dfn, dfd, loc, scale)
.std(dfn, dfd, loc, scale)
.ppf(q, dfn, dfd, loc, scale)
.rvs(dfn, dfd, loc, scale, size)
```

Notation

Book	Python	Meaning
x	`x`	observed value
ν_1	`dfn`	df numerator
ν_2	`dfd`	df denominator
$B(I_1, I_2)$		beta function

3 Linear functions and relationships of random variables

Rules for linear functions and combinations of random variables

The following rules apply to both discrete and continuous random variables. Let X and Y denote random variables.

	Description	Formula	Notes
	Mean of linear functions: $Y = aX + b$.	$E(Y) = E(aX + b) = aE(X) + b$	
	Variance of linear functions: $Y = aX + b$.	$V(Y) = \mathrm{V}(aX + b) = a^2\mathrm{V}(X)$ $V(X) = \frac{V(Y)}{a^2}$	
	Standard deviation of linear functions: $Y = aX + b$.	$\sigma_Y = \sqrt{V(Y)} = \lvert a\rvert\,\sigma_X$ $\sigma_X = \sqrt{V(X)} = \frac{\sigma_Y}{\lvert a\rvert}$	
	Mean of linear combinations: $Y = a_1X_1 + \cdots + a_nX_n$. for independent random variables.	$E(Y) = E(a_1X_1 + \cdots + a_nX_n)$ $= a_1E(X_1) + a_2E(X_2) + \cdots + a_nE(X_n)$	
	Variance of a linear combination: $Y = a_1X_1 + \cdots + a_nX_n$. for independent random variables.	$V(Y) = V(a_1X_1 + a_2X_2 + \cdots + a_nX_n)$ $= a_1^2\mathrm{V}(X_1) + a_2^2\mathrm{V}(X_2) + \cdots + a_n^2\mathrm{V}(\mathrm{X_n})$	

Independence and properties of random variables

	Description	Formula	Notes
	Independence of discrete random variables: Proof of independence between two random variables.	$P(X = x, Y = y) = P(X = x)P(Y = y)$	
	Independence of continuous random variables: Two continuous random variables X and Y are said to be independent if and only if	$f(x, y) = f(x)f(y)$	
	Properties of independent random variables: Requirement: X and Y are independent.	$E(XY) = E(X)E(Y)$ $Cov(X, Y) = 0$	

Measures of linear relationship between two random variables

	Description	Formula	Notes
	Covariance: Also denoted σ_{XY}. The covariance between X and Y.	$Cov(X, Y) = E[(X - E(X))(Y - E(Y))]$	
	Correlation coefficient: Measures the strength of the linear relationship between two stochastic variables X and Y.	$\rho(X, Y) = \frac{Cov(X, Y)}{\sigma_X \sigma_Y} \in [-1, 1]$	

4 Classic sample inference

Given a sample $\{x_1, x_2, \ldots, x_n\}$ from a population with mean μ and variance σ^2, sample statistics can be used to estimate population parameters. In particular, the Central Limit Theorem (CLT) states that, for sufficiently large $n \geq 30$, the sample mean is approximately normally distributed $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \sim \mathcal{N}\left(\mu, \frac{\sigma^2}{n}\right)$. For already normally distributed variables $n \geq 30$ is not required. The CLT therefore provides the theoretical basis for the statistical inference methods introduced in the following section.

Hypothesis tests and confidence intervals

One-sample

Collect data in `Python`

```
data = np.array([...])
```

	Description	Formula	Python
	Variation of the mean ($\sigma^2_{\bar{x}}$):	$\hat{\sigma}^2_{\bar{x}} = \frac{s^2}{n}$	
	Standard error of the mean ($\sigma_{\bar{x}}$):	$\hat{\sigma}_{\bar{x}} = \sqrt{\frac{s^2}{n}} = \frac{s}{\sqrt{n}}$	`SEM = stats.sem(data)`
	Confidence interval of the mean:	$\bar{x} \pm t_{1-\alpha/2,(n-1)} \frac{s}{\sqrt{n}}$	`CI = stats.ttest_1samp(data,popmean=`μ_0`).confidence_interval(1-`α`)`
	Margin of error:	$ME = t_{1-\alpha/2,(n-1)} \frac{s}{\sqrt{n}}$	

	Confidence interval for the variance and standard deviation:
	
	Rule of thumb:
	Only applies to normally distributed data; unlike mean CIs, the CLT does not extend to variance CIs.
		$\sigma^2 : \left[\frac{(n-1)s^2}{\chi^2_{1-\alpha/2,(n-1)}}, \frac{(n-1)s^2}{\chi^2_{\alpha/2,(n-1)}}\right]$
		
		$\sigma : \left[\sqrt{\frac{(n-1)s^2}{\chi^2_{1-\alpha/2,(n-1)}}}, \sqrt{\frac{(n-1)s^2}{\chi^2_{\alpha/2,(n-1)}}}\right]$
			lower quantile:
			`stats.chi2.ppf(1-`$\alpha/2$`, df=n-1)`
			
			upper quantile:
			`stats.chi2.ppf(`$\alpha/2$`, df=n-1)`

	Description	Formula	Python
	t-test statistic (t_{obs}): $H_0 : \mu = \mu_0$ $H_1 : \mu \neq \mu_0$	$t_{obs} = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$	`t_obs, pval = stats.ttest_1samp( data, popmean=`μ_0`)`
	Critical value (t_{crit}):	$t_{crit} = t_{1-\alpha/2,(n-1)}$ if $t_{obs} \notin [-t_{crit}, t_{crit}] \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`crit = stats.t.ppf( 1-`α`/2, df=n-1)`
	P-value:	$p_{value} = 2 \cdot P(T > \lvert t_{obs} \rvert)$ if $p_{value} < \alpha \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`pval = 2 * (1 - stats.t.cdf( abs(t_obs), df=n-1))`

Sample size n for one sample

	Description	Formula	Python

	Sample size n for specified confidence interval:	$n = \left\lceil \left(\frac{z_{1-\alpha/2} \cdot \sigma}{ME} \right)^2 \right\rceil$	`z = stats.norm.ppf(1-`α`/2)`
			
			`n = np.ceil( (z*sigma/ME)**2)`

	Sample size n for given power $(1-\beta)$:	$n = \left\lceil \left(\sigma \frac{z_{1-\beta} + z_{1-\alpha/2}}{\mu_0 - \mu_1} \right)^2 \right\rceil$	`n = smp.TTestPower(). solve_power(effect_ size=(`$\mu_0 - \mu_1$`)/`σ`, alpha=`α`, power=1-`β`)`

Two-samples

Paired

When two "paired" samples are available, they can be transformed into a single sample of measured differences. The data is therefore analyzed using a standard one-sample t-test.

Collect data in `Python`

```
x1 = np.array([...])  # e.g.  patient results before clinical trial
x2 = np.array([...])  #  patient results after clinical trial
diff_data = x1 - x2
```

Hypothesis test

For a paired t-test the proposed hypotheses are usually

$$H_0 : \mu = 0,$$
$$H_1 : \mu \neq 0$$

since we want to test if the mean difference is statistically significant from 0.

```
# Method 1
t_obs, p_val = stats.ttest_1samp(diff_data, popmean=μ0)

# Method 2
t_obs, p_val = stats.ttest_rel(A, B) # note this method only computes for μ0 = 0
```

Confidence interval

For a paired confidence interval, we usually check to see if 0 is included. If 0 is included in the interval, there is not enough evidence to suggest that the two population means are statistically significant from one another.

```
# Method 1
CI = stats.ttest_1samp(diff_data, popmean=μ0).confidence_interval(1-α)

# Method 2
CI = stats.ttest_rel(A, B).confidence_interval(1-α)
```

Welch

The Welch t-test is used when we assume unequal variances in each distribution.

Collect data in `Python`

```
A = np.array([...])
B = np.array([...])
```

	Description	Formula	Python
	Difference between two samples (d):	$d = \bar{x}_1 - \bar{x}_2$	`d = np.mean(A) - np.mean(B)`
	Variance of the difference (σ_d^2):	$\hat{\sigma}_d^2 = \frac{s_1^2}{n_1} + \frac{s_2^2}{n_2} = (\hat{\sigma}_{\bar{x}_1})^2 + (\hat{\sigma}_{\bar{x}_2})^2$	
	Standard error of the difference (σ_d):	$\hat{\sigma}_d = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} = \sqrt{(\hat{\sigma}_{\bar{x}_1})^2 + (\hat{\sigma}_{\bar{x}_2})^2}$	`SED = np.sqrt( stats.sem(A)**2 + stats.sem(B)**2)`
	Welch's degrees of freedom (ν):	$\nu = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{\left(s_1^2/n_1\right)^2}{n_1 - 1} + \frac{\left(s_2^2/n_2\right)^2}{n_2 - 1}} = \frac{(\hat{\sigma}_{\bar{x}_1} + \hat{\sigma}_{\bar{x}_2})^4}{\frac{(\hat{\sigma}_{\bar{x}_1})^4}{n_1 - 1} + \frac{(\hat{\sigma}_{\bar{x}_2})^4}{n_2 - 1}}$	
	Confidence interval of difference:	$\bar{x}_1 - \bar{x}_2 \pm t_{1-\alpha/2,\nu}\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$	`CI = stats.ttest_ind(A, B, equal_var=False).conf idence_interval(1-`α`)`
	Margin of error:	$ME = t_{1-\alpha/2,\nu}\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$	

	Description	Formula	Python
	Welch's t-test statistic (t_{obs}): H_0: $\mu_1 - \mu_2 = 0$ H_1: $\mu_1 - \mu_2 \neq 0$	$$t_{obs} = \frac{(\bar{x}_1 - \bar{x}_2) - 0}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} = \frac{d - 0}{\hat{\sigma}_d}$$	`t_obs, pval = stats.ttest_ind(A, B, equal_var=False)`
	Critical value (t_{crit}):	$t_{crit} = t_{1-\alpha/2,\nu}$ if $t_{obs} \notin [-t_{crit}, t_{crit}] \Rightarrow$ reject else $\Rightarrow$ accept H_0	`crit = stats.t.ppf(1-`α`/2, df=`ν`)`
	P-value:	$p_{value} = 2 \cdot P(T > \lvert t_{obs} \rvert)$ if $p_{value} < \alpha \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`pval = 2 * (1 - stats.t.cdf(abs(t_obs), df=`ν`))`

Pooled

The pooled t-test is used when we assume equal variances in each population.

Collect data in `Python`

```
A = np.array([...])
B = np.array([...])
```

	Description	Formula	Python
	Difference between two samples (d):	$d = \bar{x}_1 - \bar{x}_2$	`d = np.mean(A) - np.mean(B)`
	Pooled variance (s_p^2):	$s_p^2 = \frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{n_1+n_2-2}$	
	Pooled standard deviation (s_p):	$s_p = \sqrt{s_p^2}$	
	Variation of the difference (σ_d^2):	$\hat{\sigma}_d^2 = \frac{s_p^2}{n_1} + \frac{s_p^2}{n_2} = s_p^2\left(\frac{1}{n_1} + \frac{1}{n_2}\right)$	
	Standard error of the difference (σ_d):	$\hat{\sigma}_d = \sqrt{\frac{s_p^2}{n_1} + \frac{s_p^2}{n_2}} = s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$	
	Confidence interval of the difference:	$\bar{x}_1 - \bar{x}_2 \pm t_{1-\alpha/2,(n_1+n_2-2)}\sqrt{\frac{\sigma_p^2}{n_1} + \frac{\sigma_p^2}{n_2}}$	`CI = stats.ttest_ind(A, B, equal_var=True).confidence_interval(1-`α`)`
	Margin of error:	$ME = t_{1-\alpha/2,(n_1+n_2-2)}\sqrt{\frac{\sigma_p^2}{n_1} + \frac{\sigma_p^2}{n_2}}$	

	Description	Formula	Python
	Pooled t-test statistic (t_{obs}): H_0: $\mu_1 - \mu_2 = 0$ H_1: $\mu_1 - \mu_2 \neq 0$	$t_{obs} = \frac{(\bar{x}_1 - \bar{x}_2) - 0}{\sqrt{\frac{s_p^2}{n_1} + \frac{s_p^2}{n_2}}} = \frac{d - 0}{\hat{\sigma}_d}$	`t_obs, pval = stats.ttest_ind(A, B, equal_var=True)`

	Critical value (t_{crit}):	$t_{crit} = t_{1-\alpha/2,(n_1+n_2-2)}$
		if $t_{obs} \notin [-t_{crit}, t_{crit}] \Rightarrow$ reject H_0
		else $\Rightarrow$ accept H_0
			`n1, n2 = len(A), len(B)`
			
			`crit = stats.t.ppf(1-`α`/2, df=n1+n2-2)`

	P-value:	$p_{value} = 2 \cdot P(T > \lvert t_{obs} \rvert)$ if $p_{value} < \alpha \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`pval = 2 * (1 - stats.t.cdf(abs(t_obs), df=n1+n2-2))`

Sample size n for pooled sample

	Description	Python	Notes

	Sample size n for given power $(1 - \beta)$:
	
	Where σ is estimated with s_p and where $k = n_1/n_2$, that is the ratio of the two sample sizes. Typically $k = 1$.
		`n = smp.TTestIndPower().`
		`solve_power(effect_size =`$\mu_0 - \mu_1)/\sigma$`,`
		`alpha=`α`, power=1-`β`, ratio=`k`)`	

Statistical errors

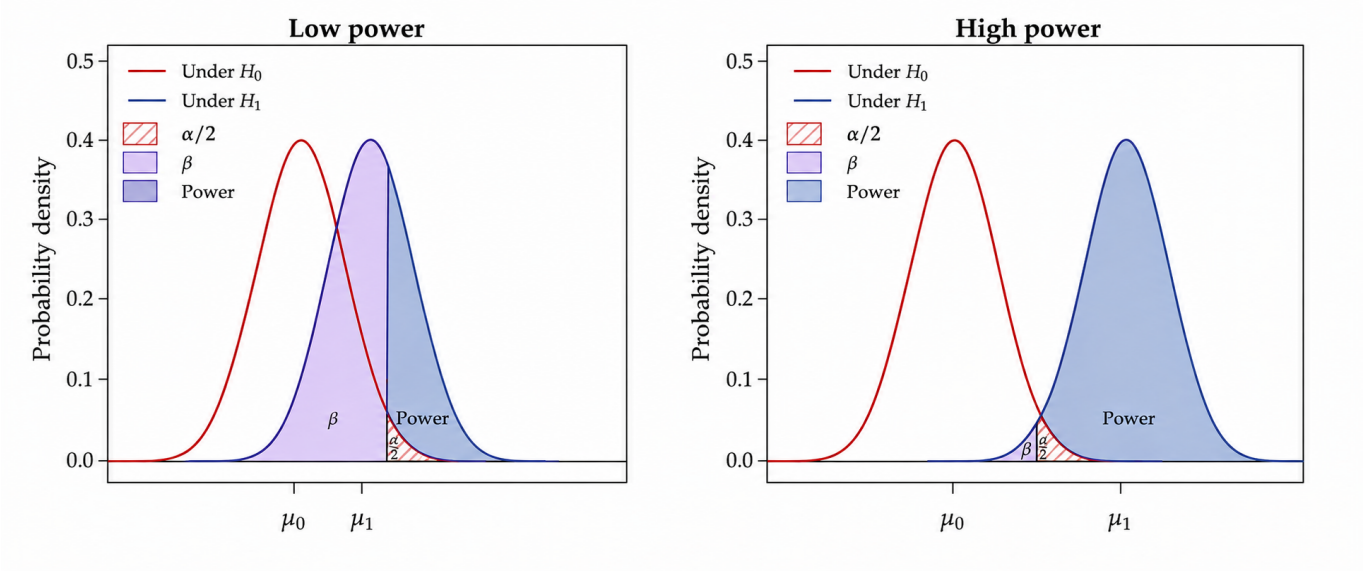
When carrying out hypothesis tests there are two types of errors you can commit:

Type I error

Also known as a false positive. It's the rejection of the null hypothesis, H_0, when H_0 is true.
Probability of a Type I error: $P(\text{Type I error}) = \alpha$.
Confidence is therefore defined: $P(\text{reject } H_1 \,|H_0 \text{ true}) = 1 - \alpha$.

Type II error

Also known as a false negative. It's the acceptance of the null hypothesis, H_0, when H_1 is true.
Probability of a Type II error: $P(\text{Type II error}) = \beta$.
Power is therefore defined: $P(\text{reject } H_0 \,|H_1 \text{ true}) = 1 - \beta$.



Red striped area (α): probability of type I error. Lavender area (β): probability of type II error.

Note: When the sample size and effect size are fixed α and β share an inverse relationship, meaning if you decrease one the other increases. A larger sample size is therefore necessary in order to increase power without decreasing confidence.

5 Simulation-based inference and error propagation

Python commands for simulation

1. **Simulate from a specified discrete sample space**
   To obtain the same simulated results each time the code is run, set a seed:

   `np.random.seed(#)`

   Use `np.random.choice()` when the possible outcomes and their probabilities are specified directly:

   `np.random.choice(a=, size=, replace=, p=)`

   a — List containing the possible outcomes (sample space).
   size — Number (or shape) of observations to simulate.
   replace — Whether sampling is with replacement (`True`) or without replacement (`False`).
   p — List containing the probability of each outcome. If `None`, all outcomes are equally likely.

2. **Simulate from a probability distribution**
   If the random variable follows a known probability distribution, observations can be simulated directly using:

   `stats.<distribution>.rvs(..., size=, random_state=)`

   `size` — specifies the number (or shape) of simulated observations.
   `random_state` — sets the random seed for reproducible simulation results; `None`, no fixed seed is used.

   Discrete distributions:

   - Binomial: `stats.binom.rvs(n=, p=, loc=0, size=, random_state=)`
   - Hypergeometric: `stats.hypergeom.rvs(M=, n=, N=, loc=0, size=, random_state=)`
   - Poisson: `stats.poisson.rvs(mu=, loc=0, size=, random_state=)`

   Continuous distributions:

   - Normal: `stats.norm.rvs(loc=`μ`, scale=`σ`, size=, random_state=)`
   - Uniform: `stats.uniform.rvs(loc=`α`, scale=`$\beta-\alpha$`, size=, random_state=)`
   - Exponential: `stats.expon.rvs(loc=0, scale=`$1/\lambda$`, size=, random_state=)`
   - Log-normal: `stats.lognorm.rvs(s=`σ`, loc=0, scale=exp(`μ`), size=, random_state=)`
   - Chi-squared: `stats.chi2.rvs(df=, loc=0, scale=1, size=, random_state=)`
   - Student's t: `stats.t.rvs(df=, loc=0, scale=1, size=, random_state=)`
   - F: `stats.f.rvs(dfn=, dfd=, loc=0, scale=1, size=, random_state=)`

	Description	Formula	Notes
	Error propagation: The variance of a function $f(X_1,\ldots,X_n)$ of random variables.	$\sigma^2_{f(X_1,\ldots,X_n)} = \sum_{i=1}^{n}\left(\frac{\partial f}{\partial x_i}\sigma_i\right)^2$	Where it is assumed x_i and x_j are independent for all i's and j's.

6 Linear regression

$$Y_i = \beta_0 + \beta_1 X_{1,i} + \cdots + \beta_p X_{p,i} + \varepsilon_i$$

Description of components:

- Y_i — the response variable (observed outcome).
- $X_{1,i}, X_{2,i}, ..., X_{p,i}$ — the explanatory variables. There are p explanatory variables (features).
- ε_i — the random error and the model assumption is $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$, $\quad \varepsilon_1, \ldots, \varepsilon_n$ i.i.d.

Fitting the model in `Python`

```
data = pd.DataFrame({'y':  [...], 'x1':  [...], ..., 'xp':  [...]})
model = smf.ols(formula='y ~ x1 + x2 + ...  + xp', data=data).fit()
print(model.summary(slim=True, alpha=α))
```

	Description	Formula	Python
	Sample estimated linear regression model:	$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{1,i} + \cdots + \hat{\beta}_p x_{p,i}$	`fittedvalues = model.fittedvalues`
	Residual (e_i):	$e_i = y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{1,i} - \cdots - \hat{\beta}_p x_{p,i}$	`residuals = model.resid`
	Residual sum of squares (RSS):	$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} e_i^2$	`RSS = model.ssr`

	Estimate of variance (σ^2):	$\hat{\sigma}^2 = \frac{RSS}{n-(p+1)}$	`var = model.scale`
			Alternatively:
			`var = model.mse_resid`

	Estimate of standard deviation (σ):	$\hat{\sigma} = \sqrt{\hat{\sigma}^2} = \sqrt{\frac{RSS}{n-(p+1)}}$	`sigma = np.sqrt(model.scale)`

Simple linear regression-specific formulas ($Y_i = \beta_0 + \beta_1 X_{1,i} + \varepsilon_i$)

	Description	Formula	Python
	Spread of x-values:	$S_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2$	`S_xx = sum((x-np.mean(x))**2)`
	Sum of cross-products:	$S_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$	`S_xy = sum((x-np.mean(x)) *(y-np.mean(y)))`
	Slope (β_1) and intercept (β_0):	$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{S_{xx}} = \frac{S_{xy}}{S_{xx}}$ $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$	`print(model.params)`
	Covariance of β_0 and β_1:	$Cov(\hat{\beta}_0, \hat{\beta}_1) = -\frac{\bar{x}\sigma^2}{S_{xx}}$	`print(model.cov_ params().iloc[0,1])`
	Variance of β_0 ($\sigma^2_{\hat{\beta}_0}$): Also denoted $V[\hat{\beta}_0]$	$\hat{\sigma}^2_{\hat{\beta}_0} = \frac{\sigma^2}{n} + \frac{\bar{x}^2\sigma^2}{S_{xx}}$	`print(model.bse**2)`
	Variance of β_1 ($\sigma^2_{\hat{\beta}_1}$): Also denoted $V[\hat{\beta}_1]$	$\hat{\sigma}^2_{\hat{\beta}_1} = \frac{\sigma^2}{S_{xx}}$	`print(model.bse**2)`
	Standard error of β_0 (σ_{β_0}):	$\hat{\sigma}_{\hat{\beta}_0} = \sqrt{\hat{\sigma}^2_{\hat{\beta}_0}} = \sqrt{\frac{\sigma^2}{n} + \frac{\bar{x}^2\sigma^2}{S_{xx}}} = \hat{\sigma}\sqrt{\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}}$	`print(model.bse)`
	Standard error of β_1 (σ_{β_1}):	$\hat{\sigma}_{\hat{\beta}_1} = \sqrt{\hat{\sigma}^2_{\hat{\beta}_1}} = \frac{\sigma}{\sqrt{S_{xx}}} = \hat{\sigma}\sqrt{\frac{1}{S_{xx}}}$	`print(model.bse)`

General linear regression continued

	Description	Formula	Python
	Spread of y-values:	$S_{yy} = \sum_{i=1}^{n} (y_i - \bar{y})^2$	`S_yy = sum((y-np.mean(y))**2)`
	t-test for parameters: $H_0 : \beta_i = \beta_{0,i}$ $H_1 : \beta_i \neq \beta_{0,i}$.	$t_{obs,\beta_i} = \dfrac{\hat{\beta}_i - \beta_{0,i}}{\hat{\sigma}_{\hat{\beta}_i}}$	`print(model.tvalues)`
	Critical value (t_{crit}):	$t_{crit} = t_{1-\alpha/2,\, n-(p+1)}$ if $t_{obs,\beta_i} \notin [-t_{crit}, t_{crit}] \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`crit = stats.t.ppf(1-`α`/2, df=n-(p+1))`

	P-value:	$p_{value} = 2 \cdot P(T > \lvert t_{obs,\beta_i} \rvert)$
		if $p_{value} < \alpha \Rightarrow$ reject H_0
		else $\Rightarrow$ accept H_0
			`pvals = model.pvalues`
			
			Alternatively:
			
			`pval = 2 * (1 - stats.t.cdf(abs(`t_{obs,β_i}`), df=n-(p+1)))`

	Confidence interval for parameters: Note: $\hat{\beta}_i$ and $\hat{\sigma}_{\hat{\beta}_i}$ are computed in Python when $p > 1$.	$\hat{\beta}_i \pm t_{1-\alpha/2,\, n-(p+1)}\, \hat{\sigma}_{\hat{\beta}_i}$	`confint = model.conf_int(alpha=`α`)`

	Confidence interval for the regression function for a given point $\mathbf{x}_0$:
	
	Fitted value:
	$\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_{1,0} + \cdots + \hat{\beta}_p x_{p,0}$
		$\hat{y}_0 \pm t_{1-\alpha/2, n-p-1}\, \sigma_{\hat{y}_0}$
		
		Simple linear regression:
		
		$\sigma_{\hat{y}_0} = \hat{\sigma}\sqrt{\dfrac{1}{n} + \dfrac{(x_0 - \bar{x})^2}{S_{xx}}}$
			`x_0 = pd.DataFrame({"x1": [...],..., "xp": [...]})`
			
			`pred = model.get_prediction(x_0).summary_frame(alpha=`α`)`
			
			`print(pred.head())`

	Description	Formula	Python

	Prediction interval for a given point $\mathbf{x}_0$:
	
	Fitted value:
	$\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_{1,0} + \cdots + \hat{\beta}_p x_{p,0}$
		$\hat{y}_0 \pm t_{1-\alpha/2,n-p-1}\sigma_{\tilde{y}_0}$
		
		Simple linear regression:
		
		$\sigma_{\tilde{y}_0} = \hat{\sigma}\sqrt{1 + \frac{1}{n} + \frac{(x_0 - \bar{x})^2}{S_{xx}}}$
			`pred = model.get_prediction(x_0).summary_frame(alpha=`α`)`
			
			`print(pred.head())`

	Coefficient of determination R^2:	$R^2 = 1 - \frac{RSS}{S_{yy}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$	`print(model.rsquared)`
			
			adjusted R-squared:
			
			`print(model.rsquared_adj)`

	Correlation coefficient r:	Simple linear regression: $r = \hat{\rho} = \text{sign}(\hat{\beta}_1)\sqrt{R^2}$	

Backward selection

Start with the full model and iteratively remove the least significant variable, refitting the model and testing the remaining parameters after each removal. Continue until all remaining explanatory variables are significant.

Matrix formulation for linear regression

	Description	Formula	Python
	Linear regression model:	$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \varepsilon_i \sim N(0, \sigma^2)$ $\mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,p} \\ 1 & x_{2,1} & \dots & x_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & \dots & x_{n,p} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$	
	Residual $\boldsymbol{e}$:	$\boldsymbol{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$	`e = np.array([...])`
	Residual sum of squares (RSS):	$RSS = \boldsymbol{e}^T\boldsymbol{e}$	`RSS = e @ e.T`
	Estimated $\boldsymbol{\beta}$:	$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_p \end{bmatrix}$	
	Sample estimated linear regression model: The fitted value $\hat{Y}_i$ for vector $\mathbf{x}_i$	$\mathbf{x}_i = [1, x_{1,i}, \dots, x_{p,i}]$ $\hat{Y}_i = \mathbf{x}_i\hat{\boldsymbol{\beta}}$	

	Description	Formula	Python
	Estimated variance and covariance of $\boldsymbol{\beta}$:	$$\mathbf{V}[\hat{\boldsymbol{\beta}}] = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}$$ $$= \begin{bmatrix} \mathrm{V}[\hat{\beta}_0] & \mathrm{Cov}(\hat{\beta}_0, \hat{\beta}_1) & \cdots & \mathrm{Cov}(\hat{\beta}_0, \hat{\beta}_p) \\ \mathrm{Cov}(\hat{\beta}_1, \hat{\beta}_0) & \mathrm{V}[\hat{\beta}_1] & \cdots & \mathrm{Cov}(\hat{\beta}_1, \hat{\beta}_p) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(\hat{\beta}_p, \hat{\beta}_0) & \mathrm{Cov}(\hat{\beta}_p, \hat{\beta}_1) & \cdots & \mathrm{V}[\hat{\beta}_p] \end{bmatrix}$$	
	Confidence interval for the regression line:	$$\mathbf{x_0} = [\,1,\, x_{1,0}, \ldots, x_{p,0}]$$ $$\begin{aligned} V(\hat{Y}_0) &= V(\mathbf{x_0}\hat{\boldsymbol{\beta}}) \\ &= \mathbf{x_0}\,\mathbf{V}[\hat{\boldsymbol{\beta}}]\,\mathbf{x_0}^T \\ &= \sigma^2\,\mathbf{x_0}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x_0}^T \end{aligned}$$ $$\hat{Y}_0 = \mathbf{x}_0\hat{\boldsymbol{\beta}},$$ $$\hat{Y}_0 \;\pm\; t_{1-\alpha/2,n-p-1}\sqrt{V(\hat{Y}_0)}$$	
	Prediction interval for the regression line:	$$\mathbf{x_0} = [\,1,\, x_{1,0}, \ldots, x_{p,0}]$$ $$\begin{aligned} V(\tilde{Y}_0) &= V(\mathbf{x_0}\hat{\beta} + \varepsilon_0) \\ &= \mathbf{x_0}\,\mathbf{V}[\hat{\boldsymbol{\beta}}]\,\mathbf{x_0}^T + \sigma^2 \\ &= \sigma^2\left(1 + \mathbf{x_0}(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{x_0}^T\right) \end{aligned}$$ $$\hat{Y}_0 = \mathbf{x}_0\hat{\boldsymbol{\beta}},$$ $$\hat{Y}_0 \;\pm\; t_{1-\alpha/2,n-p-1}\sqrt{V(\tilde{Y}_0)}$$	

7 Inference for proportions

Where x denotes the number of successes out of n independent trials with probability of success $\hat{p}$ and probability of failure $1 - \hat{p}$.

One-proportion

Collect data in `Python`

```
data = np.array([0, 0, 1, 0, 1, 1, ...])
x = sum(data)
n = len(data)
```

	Description	Formula	Python
	Sample proportion $\hat{p}$:	$\hat{p} = \frac{x}{n}$	
	Variance of a proportion ($\sigma^2_{\hat{p}}$): Also denoted $V(\hat{p})$	$\hat{\sigma}^2_{\hat{p}} = \frac{\hat{p}(1-\hat{p})}{n}$	
	Standard error of p ($\sigma_{\hat{p}}$):	$\hat{\sigma}_{\hat{p}} = \sqrt{\hat{\sigma}^2_{\hat{p}}} = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$	
	Confidence interval for a proportion: **Rule of thumb**: $np > 15$ and $n(1-p) > 15$	$\hat{p} \pm z_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$	`confint = smprop.proportion_confint(count=x, nobs=n, alpha=`α`, method="normal")`
	Margin of error:	$ME = z_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$	
	Plus 2 method: Used when $np \leq 15$ and $n(1-p) \leq 15$.	$\tilde{p} = \frac{\tilde{x}}{\tilde{n}} = \frac{x+2}{n+4}$ $\tilde{p} \pm z_{1-\alpha/2}\sqrt{\frac{\tilde{p}(1-\tilde{p})}{\tilde{n}}}$	`confint = smprop.proportion_confint(count=x, nobs=n, alpha=`α`, method="agresti_coull")`

	Description	Formula	Python
	z-test statistic (z_{obs}): $H_0 : p = p_0$ $H_1 : p \neq p_0$	$z_{obs} = \frac{x - n \cdot p_0}{\sqrt{n \cdot p_0(1-p_0)}}$ $= \frac{\hat{p} - p_0}{\sqrt{p_0(1-p_0)/n}} \sim \mathcal{N}(0, 1^2)$	`z_obs, pval = smprop.proportions_ztest(count=x, nobs=n, value=`p_0`, prop_var=0.5)`
	Critical value (z_{crit}):	$z_{crit} = z_{1-\alpha/2}$ if $z_{obs} \notin [-z_{crit}, z_{crit}] \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`crit = stats.norm.ppf(1-`α`/2)`
	P-value:	$p_{value} = 2 \cdot P(Z > \lvert z_{obs} \rvert)$ if $p_{value} < \alpha \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`pval = 2 * (1 - stats.norm.cdf( abs(z_obs)))`

Sample size n for one sample

	Description	Formula	Python
	Sample size n for specified confidence interval: For unknown p (worst case), use $p = 0.5$.	$n = \left\lceil p(1-p) \left(\frac{z_{1-\alpha/2}}{ME} \right)^2 \right\rceil$	
	Sample size n for given power $(1-\beta)$:	$n = \left\lceil \frac{[z_{1-\alpha/2}\sqrt{p_0(1-p_0)} + z_{1-\beta}\sqrt{p_1(1-p_1)}]^2}{(p_1 - p_0)^2} \right\rceil$	

Two-proportions

Collect data in Python

```
A = np.array([0, 0, 1, 0, 1, 1, ...])
B = np.array([0, 1, 0, 1, 1, 1, ...])
x1, x2 = sum(A), sum(B)
n1, n2 = len(A), len(B)
```

	Description	Formula	Python
	Difference between two proportions ($\hat{d}$):	$\hat{d} = \hat{p}_1 - \hat{p}_2$	
	Variance of the difference $\sigma^2_{\hat{p}_1-\hat{p}_2}$: Also denoted $V(\hat{p}_1 - \hat{p}_2)$	$\hat{\sigma}^2_{\hat{p}_1-\hat{p}_2} = \hat{\sigma}^2_{\hat{p}_1} + \hat{\sigma}^2_{\hat{p}_2}$	
	Standard error of the difference ($\sigma_{\hat{d}}$): **Rule of thumb:** $n_i p_i \geq 10$ and $n_i(1-p_i) \geq 10$ for $i = 1, 2$.	$\hat{\sigma}_{\hat{d}} = \sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}$ $= \sqrt{(\hat{\sigma}_{\hat{p}_1})^2 + (\hat{\sigma}_{\hat{p}_2})^2}$	
	Confidence interval for the difference: **Rule of thumb:** $n_i p_i \geq 10$ and $n_i(1-p_i) \geq 10$ for $i = 1, 2$.	$(\hat{p}_1 - \hat{p}_2) \pm z_{1-\alpha/2}\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}$	`confint = confint_proportions_2indep(count1=x1, nobs1=n1, count2=x2, nobs2=n2, alpha=`α`, method="normal")`
	Margin of Error:	$ME = z_{1-\alpha/2}\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}$	

	Description	Formula	Python
	z-test statistic (z_{obs}): $H_0 : p_1 - p_2 = 0$ $H_1 : p_1 - p_2 \neq 0$	$$\hat{p} = \frac{x_1 + x_2}{n_1 + n_2}$$ $$z_{obs} = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}(1-\hat{p})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$	`z_obs, pval = proportions_ztest( count=[x1, x2], nobs=[n1, n2])`
	Critical value (z_{crit}):	$z_{crit} = z_{1-\alpha/2}$ if $z_{obs} \notin [-z_{crit}, z_{crit}] \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`crit = stats.norm.ppf(1-`α`/2)`
	P-value:	$p_{value} = 2 \cdot P(Z > \lvert z_{obs} \rvert)$ if $p_{value} < \alpha \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`pval = 2 * (1 - stats.norm.cdf( abs(z_obs)))`

Sample size n for two samples

	Description	Formula	Python
	Sample size n for specified confidence interval: $n_1 = n_2 = n$ If p_1 or p_2 are unknown maximize variance by setting $p_1 = p_2 = 0.5$	$$n = \left\lceil \frac{(z_{1-\alpha/2})^2[p_1(1-p_1) + p_2(1-p_2)]}{ME^2} \right\rceil$$	
	Sample size n for given power $(1-\beta)$: $n = n_1 = n_2$	$\bar{p} = \frac{p_1 + p_2}{2}$ $$n = \left\lceil \frac{\left(z_{1-\alpha/2}\sqrt{2\bar{p}(1-\bar{p})} + z_{1-\beta}\sqrt{p_1(1-p_1) + p_2(1-p_2)}\right)^2}{(p_1 - p_2)^2} \right\rceil$$	

Contingency-tables

An $r \times c$ contingency table summarizes the joint distribution of two categorical variables by recording observed cell counts.

	Description	Formula	Python
	Expected value (e_{ij}): Also denoted E.	$e_{ij} = \frac{(row\ total\ i)(column\ total\ j)}{grand\ total}$	

	χ^2-test for multi-sample proportions:
	
	$H_0 : p_1 = \cdots = p_c$
	
	Rule of thumb:
	The test is valid if all $e_{ij} \geq 5$.
		$\chi^2_{obs} = \sum_{i=1}^{2}\sum_{j=1}^{c} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}$	`x = [x1,x2,...,xn]`
			`n = [n1,n2,...,n]`
			
			`chi2_obs, pval, (obs, exp) = smprop.proportions_chisquare(x, n)`

	χ^2-test for $r \times c$ frequency table:
	
	Test of the hypothesis $H_0 : p_{i1} = \cdots = p_{ic} = p_i, \quad i = 1, 2, ..., r$
	
	Test for independence between rows and columns.
	
	Rule of thumb:
	The test is valid if all $e_{ij} \geq 5$
		$\chi^2_{obs} = \sum_{i=1}^{r}\sum_{j=1}^{c} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}$	`X = [[`x_{11}`, `x_{12}`,..., `x_{1c}`],`
			`[`x_{21}`, `x_{22}`,..., `x_{2c}`],`
			`⋮`
			`[`x_{r1}`, `x_{r2}`,..., `x_{rc}`]]`
			
			`chi2_obs, pval, dof, exp = stats.chi2_contingency(X, correction=False)`

	Critical value (χ^2_{crit}):	$\chi^2_{crit} = \chi^2_{1-\alpha,(r-1)(c-1)}$ if $\chi^2_{obs} > \chi^2_{crit} \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`crit = stats.chi2.ppf(1-`α`, df=(r-1)*(c-1))`
	P-value:	$p_{value} = P(\chi^2 > \chi^2_{obs})$ if $p_{value} < \alpha \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`pval = 1 - stats.chi2.cdf( chi2_obs, df=(r-1)*(c-1))`

8 One-way ANOVA

$$Y_{ij} = \mu + \alpha_i + \varepsilon_{ij}$$

Description of components:

- Y_{ij} — the response from group i for observation j, where $j \in \{1, ..., n_i\}$
- μ — the grand (overall) mean.
- α_i — the effect of group (treatment) i, $i \in \{1, \ldots, k\}$, where there are k groups each with n_i observations.
- ε_{ij} — the random error and the model assumption is $\varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2)$, ε_{ij} are i.i.d.
- $\bar{y}_i$ — mean in group i.

One-way ANOVA

Source of variation	**Degrees of freedom**	**Sum of squares**	**Mean square**	**Test statistic F**	**p-value**
Treatment	$k-1$	$SS(Tr)$	$MS(Tr) = \frac{SS(Tr)}{k-1}$	$F_{obs} = \frac{MS(Tr)}{MSE}$	$P(F > F_{obs})$
Residual (Error)	$n-k$	SSE	$MSE = \hat{\sigma}^2 = \frac{SSE}{n-k}$		
Total	$n-1$	SST			

Fitting the model in `Python`

```
data = pd.DataFrame('y':  [...], 'group':  ["A", "A",..., "B",...,"K"]
model = smf.ols('y ~ group', data=data).fit()
anova = sm.stats.anova_lm(model)
print(anova)
```

	Description	**Formula**	**Python**
	Grand mean (μ):	$\bar{y} = \hat{\mu} = \frac{1}{n}\sum_{i=1}^{k}\sum_{j=1}^{n_i} y_{ij}$	
	Mean of group i:	$\bar{y}_i = \frac{1}{n_i}\sum_{j=1}^{n_i} y_{ij}$	
	Marginal effect α_i:	$\hat{\alpha}_i = \bar{y}_i - \bar{y}$	

	Description	Formula	Python
	Total sum of squares (SST):	$SST = \sum_{i=1}^{k}\sum_{j=1}^{n_i}(y_{ij} - \bar{y})^2 = (n-1)\sigma_y^2$	`SST = model.centered_tss`
	Treatment sum of squares ($SS(Tr)$):	$SS(Tr) = \sum_{i=1}^{k} n_i(\bar{y}_i - \bar{y})^2 = \sum_{i=1}^{k} n_i\hat{\alpha}_i^2$	`SSTr = model.ess`
	Sum of squared errors (SSE):	$SSE = \sum_{i=1}^{k}\sum_{j=1}^{n_i}(y_{ij} - \bar{y}_i)^2 = \sum_{i=1}^{k}(n_i-1)\sigma_i^2$	`SSE = model.ssr`
	Variance decomposition:	$SST = SS(Tr) + SSE$	
	Mean squared errors (MSE): Also denoted $\hat{\sigma}^2$	$MSE = \frac{SSE}{n-k} = \frac{(n_1-1)s_1^2 + \cdots + (n_k-1)s_k^2}{n-k}$ $s_i^2 = \frac{1}{n_i-1}\sum_{j=1}^{n_i}(y_{ij} - \bar{y}_i)^2$	`MSE = model.mse_resid`
	F-test for differences in group means: $H_0 : \alpha_i = 0$ $H_1 : \alpha_i \neq 0, i = 1, \ldots, k$	$F_{obs} = \frac{SS(Tr)/(k-1)}{SSE/(n-k)} = \frac{MS(Tr)}{MSE}$	`print(anova)`
	Critical value (F_{crit}):	$F_{crit} = F_{1-\alpha,k-1,n-k}$ if $F_{obs} > F_{crit} \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`crit = stats.f.ppf(` `1-`α`, dfn=k-1, dfd=n-k)`

	P-value:	$p_{value} = P(F > F_{obs})$
		if $p_{value} < \alpha \Rightarrow$ reject H_0
		else $\Rightarrow$ accept H_0
			`pval = 1 -`
			`stats.f.cdf(`F_{obs}`,`
			`dfn=k-1, dfd=n-k)`

Post-hoc pairwise tests

When performing multiple hypothesis tests, the probability of a Type I error increases. Therefore, use the Bonferroni-corrected significance level $\alpha_{\text{Bonferroni}} = \alpha/M$, where M is the number of pairwise comparisons. Among k groups the total possible comparisons are found as $M = \frac{k(k-1)}{2}$.
Note: the Bonferroni correction is conservative and may increase the probability of Type II errors.

Important distinction! for post hoc pairwise tests $\bar{y}_j$ is not to be confused with the mean of block j but rather the mean of group j you are comparing against group i with.

	Description	**Formula**	**Python**
	Confidence intervals:	$\bar{y}_i - \bar{y}_j \pm t_{1-\alpha/2,n-k}\sqrt{MSE\left(\frac{1}{n_i}+\frac{1}{n_j}\right)}$	
	Least significant difference (LSD): Formula requirement: $m = n_1 = n_k$	$LSD_\alpha = t_{1-\alpha/2,n-k}\sqrt{2 \cdot MSE/m}$ $\bar{y}_i - \bar{y}_j > LSD_\alpha \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	
	t-test statistic (t_{obs}): $H_0 : \mu_i = \mu_j$ $H_1 : \mu_i \neq \mu_j$	$t_{obs} = \frac{\bar{y}_i - \bar{y}_j}{\sqrt{MSE\left(\frac{1}{n_i}+\frac{1}{n_j}\right)}}$	
	Critical value (t_{crit}):	$t_{crit} = t_{1-\alpha/2,n-k}$ if $t_{obs} \notin [-t_{crit}, t_{crit}] \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`crit = stats.t.ppf( 1-α/2, df=n-k)`
	P-value:	$p_{value} = 2 \cdot P(T > \lvert t_{obs} \rvert)$ if $p_{value} < \alpha \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`pval = 2 * (1 - stats.t.cdf(abs(t_obs), df=n-k))`

9 Two-way ANOVA

$$Y_{ij} = \mu + \alpha_i + \beta_j + \varepsilon_{ij}$$

Description of components:

- Y_{ij} — the response for treatment i, block j.
- μ — the grand (overall) mean.
- α_i — the effect of group (Treatment), $i \in \{1, ..., k\}$, where there are k groups each with n_i observations.
- β_j — the effect of block, $j \in \{1, \ldots, l\}$, where there are j blocks.
- ε_{ij} — the random error and the model assumption is $\varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2)$, ε_{ij} are i.i.d.

Two-way ANOVA

Source of variation	**Degrees of freedom**	**Sum of squares**	**Mean square**	**Test statistic F**	**p-value**
Treatment	$k-1$	$SS(Tr)$	$MS(Tr) = \frac{SS(Tr)}{k-1}$	$F_{Tr} = \frac{MS(Tr)}{MSE}$	$P(F > F_{Tr})$
Block	$l-1$	$SS(Bl)$	$MS(Bl) = \frac{SS(Bl)}{l-1}$	$F_{Bl} = \frac{MS(Bl)}{MSE}$	$P(F > F_{Bl})$
Residual (Error)	$(l-1)(k-1)$	SSE	$MSE = \hat{\sigma}^2 = \frac{SSE}{(l-1)(k-1)}$		
Total	$n-1$	SST			

Fitting the model in `Python`

```
data = pd.DataFrame({'y': [...], 'treatment': ["A",..., "B",...,"K" ], 'block': ["1", ..., "2", ..., "l"]})
model = smf.ols('y ~ treatment + block', data=data).fit()
anova = sm.stats.anova_lm(model)
print(anova)
```

	Description	**Formula**	**Python**
	Grand mean (μ):	$\bar{y} = \hat{\mu} = \frac{1}{k \cdot l} \sum_{i=1}^{k} \sum_{j=1}^{l}$	
	Mean of group i and block j:	$\bar{y}_i = \frac{1}{l} \sum_{j=1}^{l} y_{ij}$ $\bar{y}_j = \frac{1}{k} \sum_{i=1}^{k} y_{ij}$	

	Description	Formula	Python
	Marginal effects α_i and β_j:	$\hat{\alpha}_i = \bar{y}_i - \bar{y}$ $\hat{\beta}_j = \bar{y}_j - \bar{y}$	
	Total sum of squares (SST):	$SST = \sum_{i=1}^{k}\sum_{j=1}^{l}(y_{ij} - \bar{y})^2$	
	Treatment sum of squares ($SS(Tr)$):	$SS(Tr) = l\sum_{i=1}^{k}(\bar{y}_i - \bar{y})^2 = l\sum_{i=1}^{k}\hat{\alpha}_i^2$	
	Block sum of squares ($SS(Bl)$):	$SS(Bl) = k\sum_{j=1}^{l}(\bar{y}_j - \bar{y})^2 = k\sum_{j=1}^{l}\hat{\beta}_j^2$	
	Sum of squared errors (SSE):	$SSE = \sum_{i=1}^{k}\sum_{j=1}^{l}\hat{\varepsilon}_{ij}^2$ $= \sum_{i=1}^{k}\sum_{j=1}^{l}(y_{ij} - \hat{y}_{ij})^2$ $= \sum_{i=1}^{k}\sum_{j=1}^{l}(y_{ij} - \bar{y} - \hat{\alpha}_i - \hat{\beta}_j)^2$ $= \sum_{i=1}^{k}\sum_{j=1}^{l}(y_{ij} - \bar{y}_i - \bar{y}_j + \bar{y})^2$	
	Variance decomposition:	$SST = SS(Tr) + SS(Bl) + SSE$	
	Mean squared errors (MSE):	$MSE = \hat{\sigma}^2 = \frac{SSE}{(l-1)(k-1)}$	`MSE = model.mse_resid`

Group hypothesis testing

	Description	Formula	Python
	F-test for group mean differences: $H_{0,Tr} : \alpha_i = 0$ $H_{1,Tr} : \alpha_i \neq 0,$ $i = 1, \ldots, k$	$F_{Tr} = \frac{SS(Tr)/(k-1)}{SSE/((k-1)(l-1))} = \frac{MS(Tr)}{MSE}$	`print(anova)`
	Critical value ($F_{crit,Tr}$):	$F_{crit,Tr} = F_{1-\alpha,k-1,(k-1)(l-1)}$ if $F_{Tr} > F_{crit,Tr} \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`crit = 1 - stats.f.ppf(1-`α`, dfn=k-1, dfd=(k-1)*(l-1))`
	P-value:	$p_{value} = P(F > F_{Tr})$ if $p_{value} < \alpha \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`pval = 1 - stats.f.cdf(`F_{Tr}`, dfn=k-1, dfd=(k-1)*(l-1))`

Block hypothesis testing

	Description	Formula	Python
	F-test for block mean differences: $H_{0,Bl} : \beta_j = 0$ $H_{1,Bl} : \beta_j \neq 0$ $j = 1, \ldots, l$	$F_{Bl} = \frac{SS(Bl)/(l-1)}{SSE/((k-1)(l-1))} = \frac{MS(Bl)}{MSE}$	`print(anova)`
	Critical value ($F_{crit,Bl}$):	$F_{crit,Bl} = F_{1-\alpha,l-1,(k-1)(l-1)}$ if $F_{Bl} > F_{crit,Bl} \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`crit = 1 - stats.f.ppf(1-`α`, dfn=l-1, dfd=(k-1)*(l-1))`
	P-value:	$p_{value} = P(F > F_{Bl})$ if $p_{value} < \alpha \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`pval = 1 - stats.f.cdf(`F_{Bl}`, dfn=l-1, dfd=(k-1)*(l-1))`

Post-hoc pairwise tests

When performing multiple hypothesis tests, the probability of a Type I error increases. Therefore, use the Bonferroni-corrected significance level $\alpha_{\text{Bonferroni}} = \alpha/M$, where M is the number of pairwise comparisons. Among k groups the total possible comparisons are found as $M = \frac{k(k-1)}{2}$.
Note: the Bonferroni correction is conservative and may increase the probability of Type II errors.

Important distinction! for post hoc pairwise tests $\bar{y}_j$ is not to be confused with the mean of block j but rather the mean of group j you are comparing against group i with.

	Description	**Formula**	**Python**
	Confidence intervals:	$\bar{y}_i - \bar{y}_j \pm t_{1-\alpha/2,(k-1)(l-1)} \sqrt{MSE\left(\frac{1}{n_i} + \frac{1}{n_j}\right)}$	
	Least significant difference (LSD): Formula requirement: $m = n_1 = n_k$.	$LSD_\alpha = t_{1-\alpha/2,(k-1)(l-1)} \sqrt{\frac{2 \cdot MSE}{m}}$ $\bar{y}_i - \bar{y}_j > LSD \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	
	t-test statistic (t_{obs}): $H_0 : \mu_i = \mu_j$ $H_1 : \mu_i \neq \mu_j$	$t_{obs} = \frac{\bar{y}_i - \bar{y}_j}{\sqrt{MSE\left(\frac{1}{n_i} + \frac{1}{n_j}\right)}}$	
	Critical value (t_{crit}):	$t_{crit} = t_{1-\alpha/2,(k-1)(l-1)}$ if $t_{obs} \notin [-t_{crit}, t_{crit}] \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`crit = stats.t.ppf(1-α/2, df=(k-1)*(l-1))`
	P-value:	$p_{value} = 2 \cdot P(T > \lvert t_{obs} \rvert)$ if $p_{value} < \alpha \Rightarrow$ reject H_0 else $\Rightarrow$ accept H_0	`pval = 2 * (1 - stats.t.cdf(abs(t_obs), df=(k-1)*(l-1)))`

10 Model diagnostics and residual analysis

To assess whether the model assumptions are reasonable, we examine the estimated residuals from the fitted model. For each observation, the residual is defined as

$$e_i = y_i - \hat{y}_i,$$

where y_i is the observed response and $\hat{y}_i$ is the corresponding fitted value. This general diagnostic procedure applies to linear regression, one-way ANOVA, and two-way ANOVA.

Checking the normality assumption

The normality assumption is assessed using the pooled set of residuals from the fitted model. The following graphical diagnostics can be used:

1. **Q–Q plot of the residuals.**

   If the residuals are approximately normally distributed, the points should lie close to the straight reference line. Small deviations are generally not problematic, whereas systematic curvature, large deviations in the tails, or extreme observations may indicate departures from normality.

2. **Histogram of the residuals.**

   The histogram provides a supplementary visual check. Under approximate normality, the residual distribution should be roughly symmetric and bell-shaped around zero. In general, the Q–Q plot is more informative for assessing normality.

Other residual diagnostics

In addition to normality, the residuals can be examined to assess the other model assumptions:

- **Constant variance:** Plot the residuals against the fitted values. The residuals should have approximately the same vertical spread across the range of fitted values.
- **Independence and model fit:** The residuals should be randomly scattered around zero without any clear systematic pattern.
- **Group comparisons (ANOVA):** Boxplots of the response within each group can be used as an additional visual check for differences in spread and potential outliers.

11 Notation and terminology

Symbol	Definition
Random variables and distributions	
X	Random variable representing an outcome of any distribution.
$\bar{X}$	Random variable representing the sample mean.
S^2	Random variable representing the sample variance.
S	Random variable representing the sample standard deviation.
Z	Random variable representing an outcome from the standard normal distribution.
T	Random variable representing an outcome from the Student's t-distribution.
χ^2	Random variable representing an outcome from the χ^2-distribution.
F	Random variable representing an outcome from the F-distribution.
Population and sample quantities	
N	Population size, the total number of elements in a population.
μ	Population mean (also denoted $E[X]$).
σ^2	Population variance (also denoted $Var(X)$ or $V[X]$).
σ	Population standard deviation.
n	Sample size, the number of observations in a sample.
$\bar{x}$	Sample mean (also denoted $\hat{\mu}$).
x_i	The i-th observation in a sample of size n.
s^2	Sample variance.
s	Sample standard deviation.
Standard errors	
$\hat{\sigma}_{\bar{x}}$	Estimated standard error of the mean.
$\hat{\sigma}_d$	Estimated standard error of the difference between two means (also denoted $\hat{\sigma}_{\bar{x}_1-\bar{x}_2}$).
$\hat{\sigma}_{\hat{p}}$	Estimated standard error of a proportion.
$\hat{\sigma}_{\hat{d}}$	Estimated standard error of the difference between two proportions (also denoted $\hat{\sigma}_{\hat{p}_1-\hat{p}_2}$).
Hypothesis testing	
H_0	Null-hypothesis.
H_1	Alternative hypothesis.
z_{obs}	Observed z-statistic.
t_{obs}	Observed t-statistic.
χ^2_{obs}	Observed χ^2-statistic.
F_{obs}	Observed F-statistic.
z_{crit}	Critical quantile from the standard normal distribution.
t_{crit}	Critical quantile from the Student's t-distribution.
χ^2_{crit}	Critical quantile from the chi-squared distribution.
F_{crit}	Critical quantile from the F-distribution.
p_{value}	Probability of observing a test statistic at least as extreme as the observed one under H_0.

Correlation and regression

s_{xy}	Sample covariance: Measures the direction of the linear relationship between two variables (unit-dependent).
r	Sample correlation coefficient (also denoted $\hat{\rho}$) it's the estimate of the population correlation ρ: Measures the strength and direction of the linear relationship between two variables, in the range $[-1, 1]$.
R^2	Coefficient of determination in linear regression. The proportion of variance in Y explained by X, in the range $[0, 1]$.

12 Python libraries

```
import numpy as np
import pandas as pd
import scipy.stats as stats
import statsmodels.api as sm
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
import statsmodels.stats.power as smp
import statsmodels.stats.proportion as smprop
```