Statistics

```
import numpy as np
import math
import scipy.stats as stats
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.stats.power as smp
import statsmodels.formula.api as smf
import statsmodels.stats.proportion as smprop
import pandas as pd
x = np.array([168, 161, 167, 179, 184, 166, 198, 187, 191, 179])
```

Sample

There is an underlying population from which a representative sample with
n observations has been drawn.

The sample is usually represented by a vector

$$x = (x_1, x_2, ..., x_n)$$

The sorted sample is then

$$x = (x_{(1)}, x_{(2)}, ..., x_{(n)})$$

```
x.sort()
```

where $x_{(1)}$ denotes the smallest observation and $x_{(n)}$ denotes the largest observation.

▼ Sample mean (average):

We say that $\bar{x}$ is an estimate of the population mean.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

```
# calculate mean of x (average height of students)
np.mean(x)
```

▼ Standard error

$$s.e._x = \frac{\sigma}{\sqrt{n}} = \sqrt{\frac{\sigma^2}{n}}$$

▼ Sample median:

$$\begin{gathered}\text{if n is odd} \\ Q_2 = x_{(\frac{n}{2})} \\ \text{if n is even} \\ Q_2 = \frac{x_{\frac{n+1}{2}} + x_{\frac{n-1}{2}}}{2}\end{gathered}$$

```
np.median(x)
```

▼ Sample Variance and Standard deviation:

The variance and the standard deviation indicate the dispersion ("spread") of the data:

- Variance:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

```
# OBS: need to remember ddof = 1 if you want to calculate the "sample variance".
# We use (n-1) for sample variance because it gives us an unbiased estimator of the population variance. This is known as Bessel's correction.
x.var(ddof=1)
```

- Standard deviation:

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

```
# standard deviation (also remember ddof=1 for "sample standard deviation")
x.std(ddof=1)
```

▼ Coefficient of variation

$$V = \frac{s}{\bar{x}}$$

▼ Percentiles and quantiles

- The median is the 50% percentile.
- The 25, 50, 75 % percentiles are often referred to as the first, second and third quartiles, and denoted Q_1, Q_2, and Q_3, respectively.
- Inter Quartile Range (IQR): $Q_3 - Q_1$

The p'th quantile, also named the 100p'th percentile, can be defined by the following procedure:

1. Compute the sorted sample
2. Compute pn
3. If pn is an integer: Average the pn'th and $(pn+1)$'th ordered observations:

$$\text{The } p\text{'th quantile } = (x_{(np)} + x_{(np+1)})/2$$

4. If pn is a non-integer, take the next ordered observation:

$$\text{The } p\text{'th quantile } = x_{(\lceil np \rceil)}$$

Where $\lceil np \rceil$ is the ceiling of np, that is, the smallest integer larger up

```
x.min()
x.max()
# we can also get other percentiles (50th percentile is the same as the median)
np.percentile(x, [10,20,50,80,90], method='averaged_inverted_cdf')
```

▼ Sample Covariance and correlation

```
# Data from the table
heights = np.array([168, 161, 167, 179, 184, 166, 198, 187, 191, 179])
weights = np.array([65.5, 58.3, 68.1, 85.7, 80.5, 63.4, 102.6, 91.4, 86.7, 78.9])
cov_matrix = np.cov(heights, weights)
```

- The sample covariance is given by:

$$s_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

```
# Covariance calculation
covariance = cov_matrix[0, 1]
```

- The sample correlation coefficient is given by:

$$r = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s_x}\right)\left(\frac{y_i - \bar{y}}{s_y}\right) = \frac{s_{xy}}{s_x \times s_y}$$

```
# Correlation coefficient calculation
correlation = np.corrcoef(heights, weights)[0, 1]
```

where s_x and s_y are the sample standard deviations for x and respectively.

- r is always between −1 and 1: $-1 \leq r \leq 1$.
- r measures the degree of linear relation between x and y.
- $r = \pm 1$ if and only if all points in the scatterplot are exactly on a line.
- $r > 0$ if and only if the general trend in the scatterplot is positive.
- $r < 0$ if and only if the general trend in the scatterplot is negative.

Plots

```
# import the matplotlib.pyplot package
import matplotlib.pyplot as plt
```

▼ Histogram

A histogram is a graphical representation of the distribution of numerical data, where the data is grouped into continuous, non-overlapping intervals called bins, and the height of each bar reflects the frequency (or density) of values falling within each bin.

```
# Create a histogram. This is the simplest way of creating a histogram in matplotlib. It uses the default number of bins and default style settings.
plt.hist(x)
plt.show()
```

Now, we'll customize the histogram by specifying the number of bins (bins=8), adding an edge color and fill color, adjusting the transparency (alpha), and setting density=True to display a probability density rather than a simple count.

```
# Customize your histogram
plt.hist(x, bins=8, edgecolor='black', color='red', alpha=0.7, density=True)
plt.xlabel('x')
plt.ylabel('Density')
plt.title('Histogram Example')
plt.show()
```

Now, we'll manually define the exact bin edges (bins=[160,165,170,175,180,185,190,195,200]) to have full control over the bin intervals, while still applying the same stylistic customizations as in the second example.

```
# specifying bin-edges:
plt.hist(x, bins=[160,165,170,175,180,185,190,195,200], edgecolor='black', color='red', alpha=0.7, density=True)
plt.xlabel('x')
plt.ylabel('Density')
plt.title('Histogram Example')
plt.show()
```

▼ Empirical Cumulative Distribution Function

A **cumulative distribution plot (CDF plot)** shows the proportion (or percentage) of data that falls at or below a given value. Some key, simple uses include:

- **Identifying thresholds and percentiles** (e.g., medians, quartiles).
- **Comparing different distributions** easily on the same figure.
- **Visually estimating probabilities** of data being less than or greater than a specific value.

```
# plot the "Empirical Cumulative Distribution Function"
plt.ecdf(x)
plt.show()
```

```
# lets increase the y-range slightly:
plt.ecdf(x)
plt.ylim(-0.1,1.1)
plt.xlabel('x')
plt.ylabel('ecdf(x)')
plt.title('Empirical Cumulative Distribution Function')
plt.show()
```

▼ Boxplot

A boxplot provides a quick visual summary of the distribution of a dataset, highlighting:

- Median and Quartiles – Displays the central tendency and spread via the median, the first quartile (Q1), and the third quartile (Q3).
- Range of Data – Whiskers typically show the data range or a specific cutoff for outliers.
- Outlier Detection – Points lying beyond the whiskers indicate potential outliers
- Comparison Across Groups – Multiple boxplots can be placed side by side to compare distributions.

```
# make a boxplot
plt.boxplot(x)
plt.show()
```

```
# Adding some explanation:
plt.boxplot(x)
plt.text(1.1, np.percentile(x,  [0]), 'Minimum', color='blue')
plt.text(1.1, np.percentile(x, [25]), 'Q1', color='blue')
plt.text(1.1, np.percentile(x, [50]), 'Median', color='blue')
plt.text(1.1, np.percentile(x, [75]), 'Q3', color='blue')
plt.text(1.1, np.percentile(x,[100]), 'Maximun', color='blue')
plt.title("Basic box plot")
plt.show()
x.sort()
```

▼ Manually Calculate the values required for drawing a boxplot

```
import numpy as np

# Data
x = [168, 161, 167, 179, 184, 166, 198, 187, 191, 179]

# Calculate quartiles
Q1 = np.percentile(x, 25)
Q3 = np.percentile(x, 75)

# Interquartile Range
IQR = Q3 - Q1

# Calculate fences
lower_fence = Q1 - 1.5 * IQR
upper_fence = Q3 + 1.5 * IQR

print("Q1 (25th percentile):", Q1)
print("Q3 (75th percentile):", Q3)
print("IQR:", IQR)
print("Lower fence:", lower_fence)
print("Upper fence:", upper_fence)
```

▼ Adding an outlier to the data

```
# Adding an outlier to the data:
plt.boxplot(np.append(x, [235]))
plt.show()
```

▼ Different types of whiskers

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(8, 4))

# Boxplot with one extreme value appended
ax1.boxplot(np.append(x, [235]))
ax1.set_title('Default Whiskers')

# Boxplot with whiskers at min and max of the data (0th and 100th percentiles)
ax2.boxplot(np.append(x, [235]), whis=(0, 100))
ax2.set_title('Whiskers at (0, 100)')

# Boxplot with whiskers set to 1.5 * IQR
ax3.boxplot(np.append(x, [235]), whis=1.5)
ax3.set_title('Whiskers at 1.5 * IQR')

plt.tight_layout()
plt.show()
```

Working with data - dataframes

```
# import the Pandas library
import pandas as pd
```

▼ About the code:

- Use `'pd.DataFrame()'` from **pandas** to create a table-like data structure.
- The dictionary keys (`'males'` and `'females'`) become the DataFrame columns, and the lists are the row entries.
- Missing values (`np.nan`) allow pandas to handle incomplete data gracefully in further analysis.

▼ Example of a DataFrame

```
# Make a DataFrame:
data = pd.DataFrame({
    'males':  [152, 171, 173, 173, 178, 179, 180, 180, 182, 182, 182, 185,
                   185 ,185, 185, 185 ,186 ,187 ,190 ,190, 192, 192, 197],
    'females':[159, 166, 168 ,168 ,171 ,171 ,172, 172, 173, 174 ,175 ,175,
                   175, 175, 175, 177, 178, np.nan,np.nan,np.nan,np.nan,np.nan,np.nan]
```

```
# The DataFrame has a direct method for making a boxplot:
data.boxplot()
```

Random Variables

```
# import scipy.stats for theoretical distributions (and much more)
import scipy.stats as stats
```

▼ Discrete and continuous random variables

We distinguish between discrete and continuous random variables.

- Discrete:

Number of people in this room who wear glasses.
Number of planes departing from CPH within the next hour.

- Continuous:

Wind speed measurement.
Transport time to DTU.

- Today: Discrete. Next week: Continuous.

Before the experiment is carried out, we have a random variable indicated with capital letters.

$$X \ (or \ X_1, ..., X_n)$$

After the experiment is carried out, we have a realization or observation indicated with lowercase letters.

$$x \ (or \ x_1, ..., x_n)$$

Random variables are used to describe an experiment before it is carried out. We can do this without knowing the outcome using different density functions.

▼ Classification of distributions

A distribution can be classified in several ways. In this course, we focus on:

- The probability density function (pdf, alternatively for discrete variables: probability mass function, pmf)

  It can be defined:

$$f(x) = P(X = x)$$

It describes the probability that X takes the value x when the experiment is carried out.

This function satisfies two properties:

$$f(x) \geq 0 \text{ for all } x \text{ and } \sum_{\text{all } x} f(x) = 1$$

- The cumulative distribution function (cdf)

  It can be defined:

$$F(x) = P(X \leq x) = \sum_{j \text{ where } x_j \leq x} f(x_j)$$

Here's an example of the difference of pdf and cdf density functions

▼ Specific distributions

A number of different statistical distributions exist, which may be used to describe and analyse different types of problems.

▼ The binomial distribution (discrete)

- An experiment with two outcomes, "success" or "failure", is repeated (independent repetitions).
- X is the number of successes after n repetitions
- Then X follows a binomial distribution:

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

- n represents the number of repetitions
- p represents the probability of success in each repetition

▼ Computing (pdf)

Probability density function (pdf):

$$f(x; n, p) = P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$$

$$\text{where } \binom{n}{x} = \frac{n!}{x!(n-x)!}$$

```
# we can calculate the probability using the pdf ("pmf") of the binomial:
stats.binom.pmf(k=x, n=n, p=p)
```

▼ Computing (cdf)

Cumulative density function (pdf):

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)\, dt.$$

As a consequence:

$$f(x) = F'(x)$$

It's particularly useful to note that:

$$P(a < X \leq b) = \int_a^b f(x)\, dx = F(b) - F(a).$$

```
# we can calculate the probability using the cdf of the binomial:
# instead of being P(X=k) its now P(X<=k)
# if we want P(X<k) we choose x-1
stats.binom.cdf(k=x, n=n, p=p)
```

▼ Computing (sf)

Survival density function (sf):

$$g(x) = P(X > x) = 1 - P(X \leq x)$$

```
# we can calculate the probability using the cdf of the binomial:
# instead of being P(x=k) its now P(x>k)
stats.binom.sf(k=x, n=n, p=p)
```

▼ Mean

$$\mu = np,$$

▼ Variance

$$\sigma^2 = np(1-p)$$

▼ The hypergeometric distribution

- X is the number of successes, but now without replacement when repeating.
- X follows the hypergeometric distribution

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

- n is the number draws (repetitions)
- a is the number of successes in the population
- N is the number of elements in the (entire) population

▼ Computing (pdf)

Probability density function (pdf)

$$f(x; n, a, N) = P(X = x) = \frac{\binom{a}{x}\binom{N-a}{n-x}}{\binom{N}{n}}$$

$$\text{where } \binom{n}{x} = \frac{n!}{x!(n-x)!}$$

```
stats.hypergeom.pmf(k=x, M=N, n=a, N=n)
```

▼ Computing (cdf)

Cumulative density function (pdf):

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)\, dt.$$

As a consequence:

$$f(x) = F'(x)$$

It's particularly useful to note that:

$$P(a < X \leq b) = \int_{a}^{b} f(x)\, dx = F(b) - F(a).$$

```
# we can calculate the probability using the cdf of the hypergeometric:
# instead of being P(X=k) its now P(X<=k)
# if we want P(X<k) we choose x-1
stats.hypergeom.cdf(k=x, M=N, n=a, N=n)
```

▼ Computing (sf)

Survival density function (sf):

$$g(x) = P(X > x) = 1 - P(X \leq x)$$

```
# we can calculate the probability using the cdf of the binomial:
# instead of being P(x=k) its now P(x>k)
# if we want P(X>=k) we choose x-1
stats.hypergeom.sf(k=x, M=N, n=a, N=n)
```

▼ Mean

$$\mu = n\frac{a}{N},$$

▼ Variance

$$\sigma^2 = n\frac{a(N-a)}{N^2}\frac{N-n}{N-1}$$

▼ The Poisson distribution

- X is the number of successes
- The Poisson distribution is often used as a distribution (model) for counts, which do not have a natural upper bound.
- The Poisson distribution is often characterized by its intensity, which is on the form "number/unit", and often denoted λ.
- X follows the poisson distribution

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

▼ Computing (pdf)

Probability density function (pdf)

$$f(x) = P(X = x) = \frac{\lambda^x}{x!}e^{-\lambda}$$

$$\lambda^{10 \text{ minutes}} = \frac{\lambda^{90 \text{ minutes}}}{9}$$

```
#x is the number of successes in the time period
stats.poisson.pmf(x, λ)
```

▼ Computing (cdf)

Cumulative density function (pdf):

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)\, dt.$$

As a consequence:

$$f(x) = F'(x)$$

It's particularly useful to note that:

$$P(a < X \leq b) = \int_a^b f(x)\,dx = F(b) - F(a).$$

```
# we can calculate the probability using the cdf of the hypergeometric:
# instead of being P(X=k) its now P(X<=k)
# if we want P(X<k) we choose x-1
stats.poisson.cdf(x, λ)
```

▼ Computing (sf)

Survival density function (sf):

$$g(x) = P(X > x) = 1 - P(X \leq x)$$

```
# we can calculate the probability using the cdf of the binomial:
# instead of being P(x=k) its now P(x>k)
# if we want P(X>=k) we choose x-1
stats.poisson.sf(x, λ)
```

▼ Mean

$$\mu = \lambda,$$

▼ Variance

$$\sigma^2 = \lambda$$

▼ Continuous Distributions

- Mean, continuous random variable, definition

  The mean/expected value of a continuous random variable:

  $$\mu = E[X] = \int_{-\infty}^{\infty} x f(x)\,dx$$

  Compare with the mean of a discrete random variable:

  $$\mu = E[X] = \sum_{\text{all } x} x f(x)$$

- Variance, continuous random variable, definition

  The variance of a continuous random variable:

  $$\sigma^2 = E[(X-\mu)^2] = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)\,dx$$

  Compare with the variance of a discrete random variable:

  $$\sigma^2 = E[(X-\mu)^2] = \sum_{\text{every type of } x} (x_i-\mu)^2 f(x_i)$$

- Covariance, continuous random variable, definition

  The covariance between two random variables:

  Let $X$and Y be two random variables. Then, the covariance between X and Y is

  $$\operatorname{Cov}(X,Y) = E[(X - E[X])(Y - E[Y])]$$

  Relationship between covariance and independence:
  If two random variables are independent, their covariance is 0 (The reverse is not necessarily true)

▼ Specific continuous distributions

▼ The Uniform distribution

Let X be a uniform distribuited random variable:

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

where α and β defines the range of possible outcomes

It has the following density function:

$$f(x) = \frac{1}{\beta - \alpha} \text{ for } \alpha \leq x \leq \beta$$

▼ Computing (pdf)

$$f(x) = \begin{cases} \frac{1}{\beta-\alpha} & \text{for } x \in [\alpha, \beta] \\ 0 & \text{otherwise} \end{cases}$$

```
#x is the number you wanna find the probability of
#a is alpha (the beginning of the interval)
#b is beta (the end of the interval)
stats.uniform.pdf(x, loc=a, scale=b)
```

▼ Example of density function for uniform distribuition

▼ Computing (cdf)

$$F(x) = \begin{cases} 0 & \text{for } x < \alpha \\ \frac{1}{\beta - \alpha} & \text{for } x \in [\alpha, \beta) \\ 1 & \text{for } x \geq \beta \end{cases}$$

```
#x is the number in x>= you will recieve probability of
#a is alpha (the beginning of the interval)
#b is beta (the end of the interval)
stats.uniform.cdf(x, loc=a, scale=b)
```

▼ Mean

$$\mu = \frac{\alpha + \beta}{2}$$

▼ Variance

$$\sigma^2 = \frac{1}{12}(\beta - \alpha)^2$$

▼ The Normal distribution

Let X be a normally distributed random variable:

$$X \sim N(\mu, \sigma^2)$$

where μ represents the mean, and σ^2 the variance

It has the following density function:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \text{ for } -\infty < x < \infty$$

▼ Computing (pdf)

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \text{ for } -\infty < x < \infty$$

```
#x is the number you want the probabilty
#m is the mean
#s is the standard deviation
stats.norm.pdf(x, loc=m, scale=s)
```

▼ Computing (cdf)

```
#y is the number you want for the probability of x>=y
#m is the mean
#s is the standard deviation
stats.norm.cdf(y, loc=m, scale=s)
```

▼ Computing (ppf)

```
#m is the mean
#s is the standard deviation
#y is the percentile
#this function will return the cdf until y percent
stats.norm.ppf(y, loc=m scale=s)
```

▼ Mean

$$\mu = \mu$$

▼ Variance

$$\sigma^2 = \sigma^2$$

▼ The Standard Normal distribution

Let Z be a standard normally distributed random variable:

$$Z \sim N(0, 1)$$

The Standard Normal distribution is just a Normal distribution where the mean is 0 and the standard variation is 1.

▼ Standardization

An arbitrary normal distributed variable $X \sim N(\mu, \sigma^2)$ can be standardized by

$$Z = \frac{X - \mu}{\sigma}$$

▼ Computing (pdf)

```
#x is the number you want the probabilty of
stats.norm.pdf(x, loc=0, scale=1)
```

▼ Computing (cdf)

```
#y is the number you want for the probability of x>=y
stats.norm.cdf(y, loc=0, scale=1)
```

▼ Computing (ppf)

```
#y is the percentile
#this function will return the cdf until y percent
stats.norm.ppf(y, loc=0, scale=1)
```

▼ Mean

$$\mu = 0$$

▼ Variance

$$\sigma^2 = 1^2$$

▼ The log-normal distribution

Let X be a log-normally distributed variable:

$$X \sim LN(\alpha, \beta^2)$$

where α represents the mean, and β^2 the variance of the normal distribution obtained when taking the natural logarithm to X

It has the following density function:

$$f(x) = \begin{cases} \frac{1}{\beta\sqrt{2\pi}} x^{-1} e^{-(\ln(x)-\alpha)^2/2\beta^2} & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

▼ Transformation into normally distributed variable

For:

$$Y \sim LN(\alpha, \beta^2)$$

X is the transformed normally distributed function:

$$X = \ln Y$$

Where X is normally distributed with mean α and variance β^2, i.e. $X \sim N(\alpha, \beta^2)$.

By standardizing X through:

$$Z = \frac{X - E[X]}{\sqrt{V[X]}} = \frac{\ln Y - \alpha}{\beta}$$

We get a standard normal distributed variable $Z \sim N(0, 1)$

▼ Computing (pdf)

```
#x is the number you want the probabilty of
#m is the e to the power of the mean
#s is the standard deviation
stats.lognorm.pdf(x, s, loc=0, scale=m)
```

▼ Computing (cdf)

```
#y is the number you want for the probability of x>=y
#m is the e to the power of the mean
#s is the standard deviation
stats.lognorm.cdf(y, s, loc=0, scale=m)
```

▼ Computing (ppf)

```
#m is the e to the power of the mean
#s is the standard deviation
#q is the percentile
#this function will return the cdf until y percent
stats.lognorm.ppf(q,s, loc=0 scale=m)
```

▼ Mean

$$\mu = e^{\alpha+\beta^2/2}$$

▼ Variance

$$\sigma^2 = e^{2\alpha+\beta^2}(e^{\beta^2} - 1)$$

▼ The exponential distribution

Let X be a exponentially distributed variable:

$$X \sim \text{Exp}(\lambda), \text{ where } \lambda > 0$$

Where λ is the average rate of events.

It has the following density function:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

▼ Transformation into normally distributed variable

For:

$$Y \sim LN(\alpha, \beta^2)$$

X is the transformed normally distributed function:

$$X = \ln Y$$

Where X is normally distributed with mean α and variance β^2, i.e. $X \sim N(\alpha, \beta^2)$.

By standardizing X through:

$$Z = \frac{X - E[X]}{\sqrt{V[X]}} = \frac{\ln Y - \alpha}{\beta}$$

We get a standard normal distributed variable $Z \sim N(0, 1)$

▼ Computing (pdf)

```
#x is the number you want the probability of
#m is the inverse of the lambda
stats.expon.pdf(x, loc=0, scale=m)
```

▼ Computing (cdf)

```
#y is the number you want for the probability of x>=y
#m is the inverse of the lambda
stats.expon.cdf(y, loc=0, scale=m)
```

▼ Computing (ppf)

```
#m is the inverse of the lambda
#q is the percentile
#this function will return the cdf until y percent
stats.expon.ppf(q, loc=0 scale=m)
```

▼ Mean

$$\mu = \frac{1}{\lambda}$$

▼ Variance

$$\sigma^2 = \frac{1}{\lambda^2}$$

▼ The Chi-squared distribution

Let $Z_1, \cdots, Z_k$ be an independent standard normal distributed random variable:

$$Z_i \sim N(0, 1^2),\ i \in \{1, \cdots, k\}$$

Then let X be the sum of Z_i^2 like,

$$X = \sum_{i=1}^{k} Z_i^2$$

X is then distributed according to the chi-squared distribution with k degrees of freedom, usually denoted as,

$$X \sim \chi^2(k) \text{ or } X \sim \chi_k^2$$

The Chi-squared distribution is always bigger than 0

▼ Rules for Stochastic variables

These rules apply to both continuous and discrete random variables

Let X be a random variable, while a and b are constant.

- Mean rule

$$E[aX + b] = aE[X] + b$$

- Variance rule

$$V[aX + b] = a^2V[X]$$

▼ Rules for Random variables

Let $X_1, ..., X_n$ be independent random variables

- Mean rule

$$E[a_1X_1 + a_2X_2 + ... + a_nX_n] = \\ = a_1E[X_1] + a_2E[X_2] + ... + a_nE[X_n]$$

- Variance rule

$$V[a_1X_1 + a_2X_2 + ... + a_nX_n] = \\ = a_1^2V[X_1] + ... + a_n^2V[X_n]$$

Even when you are subtracting two random variables you add the variance.

$$V[X + \cdots + X] = nV[X]$$

Confidence intervals

The purpose of an interval estimate is to provide information about how close the point estimate, provided by the sample, is to the value of the population parameter.

The general form of an interval estimate of population mean is:

$$\bar{x} \pm \text{Margin of error}$$

▼ Distribution of the sample mean of i.i.d. normal random variables

Now the sample mean, $\bar{X} = \frac{1}{n}(X_i + \ldots + X_n)$, itself is a random variable (with its own distribution).

Assume that $X_i + \ldots + X_n$ are independent and identically distributed (i.i.d) normal random variables, $X_i \sim N(\mu, \sigma^2), i = 1, ..., n$, then:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} \text{ and } N\left(\mu, \frac{\sigma^2}{n}\right)$$

The mean and variance follow from the rules of calculation:

▼ The mean of $\bar{X}$:

$$E[\bar{X}] = E[\frac{1}{n}X_i + ... + \frac{1}{n}X_n] = \frac{1}{n}\sum_{i=1}^{n} E[X_i] =$$
$$= \frac{1}{n}\sum_{i=1}^{n} \mu = \frac{1}{n} n\mu = \mu$$

▼ The Variance of $\bar{X}$:

$$V[\bar{X}] = V[\frac{1}{n}X_i + ... + \frac{1}{n}X_n] = \frac{1}{n^2}\sum_{i=1}^{n} V[X_i] =$$
$$= \frac{1}{n^2}\sum_{i=1}^{n} \sigma^2 = \frac{1}{n^2} n\sigma^2 = \frac{\sigma^2}{n}$$

▼ Normality of $\bar{X}$:

A linear combination of normally distributed random variables will also be normally distributed (with mean and variance as given above).

▼ Error of $(\bar{X} - \mu)$:

- Expected average error $E[\bar{X} - \mu]$:

$$E[\bar{X} - \mu] = \mu - \mu = 0$$

- Variance of the error $V[\bar{X} - \mu]$:

$$EV[\bar{X} - \mu] = V[\bar{X}] = \frac{\sigma^2}{n}$$

▼ Distribution of standardized sample mean (or standardized error):

Assume that $X_i + \ldots + X_n$ are i.i.d normal random variables, $X_i \sim N(\mu, \sigma^2), i = 1, ..., n$, then:

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

That is, the standardized sample mean Z follows normal distribution.

▼ Confidence interval (CI) for μ: σ assumed known

Use the correct z-distribution to construct the confidence interval:

For a sample $x_1, \ldots, x_n$ the $100(1-\alpha)\%$ confidence interval is given by:

$$\bar{x} \pm z_{1-\alpha/2}\frac{\sigma}{\sqrt{n}}$$

Where $z_{1-\alpha/2}$ is the $100(1-\alpha/2)\%$ quantile from the z distribution or standard normal distribution.

Most commonly using $\alpha = 0.05$:

The most commonly used is the 95% confidence interval:

$$\bar{x} \pm z_{0.975}\frac{\sigma}{\sqrt{n}}$$

▼ Table of most commonly used confidence levels

Confidence level	α	$\alpha/2$
90%	0.1	0.05
95%	0.05	0.025
99%	0.01	0.005

▼ Interval estimate of population mean: σ unknown

We've estimated $\bar{x}$ and s^2 from our sample data. We do not know the "true" μ and σ^2. So we apply the t-distribution ($n-1$ degrees of freedom), the theory deals with uncertainty linked to s.

Instead we consider the random variable: $T = \frac{\bar{X}-\mu}{S/\sqrt{n}}$

The t-distribution takes the uncertainty of s into account:

Assume that $X_i + \ldots + X_n$ are independent and identically distributed (i.i.d) normal random variables, $X_i \sim N(\mu, \sigma^2), i = 1, ..., n$, then:

$$T = \frac{\bar{X} - \mu}{S/\sqrt{n}} \sim t(n-1)$$

Where $t(n-1)$ is the t-distribution with $n-1$ degrees of freedom

▼ T-intervals

We need to choose a "significance level" (α), typically, 0.1 (90% confidence level), 0.05 (95% confidence level), or 0.01 (confidence level).

After that, based on interval boundaries of Z, you can calculate the interval boundaries for $\bar{X} - \mu$

```
#a being the significance level
#m being the degree of freedom (n-1)
t_lower = stats.t.ppf(q=a/2,df=m)
t_upper = stats.t.ppf(q=(1-a/2),df=m)
```

▼ Distribution of the sample variance of i.i.d. normal random variables

Assume that $X_i + \ldots + X_n$ are independent and identically distributed (i.i.d) normal random variables, $X_i \sim N(\mu, \sigma^2), i = 1, ..., n$.

The sample variance (variance estimate) follows a χ^2 distribution:

Let

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})$$

It holds that:

$$\chi^2 = \frac{(n-1)S^2}{\sigma^2}$$

follows a χ^2 distribution with $n-1$ degrees of freedom.

χ^2 is always positive and the χ^2 distribution is not symmetric.

We need to choose a "significance level" (α).

Then, we can calculate interval bounds for σ^2 from the interval bounds on χ^2.

▼ χ^2 interval bounds

```
#a being the significance level
#m being the degree of freedom (n-1)
#this will return the interval bounds of chi2 not variance
chi2_lower = stats.chi2.ppf((a/2), df=m)
chi2_upper = stats.chi2.ppf((1-a/2), df=m)
```

▼ σ^2 interval bounds

A $100(1-\alpha)\%$ confidence interval for the variance σ^2 is given by:

$$\left[\frac{(n-1)s^2}{\chi^2_{1-\alpha/2}}; \frac{(n-1)s^2}{\chi^2_{\alpha/2}}\right]$$

```
#a being the significance level
#m being the degree of freedom (n-1)
#this will return the interval bounds of variance
chi2_lower = stats.chi2.ppf((a/2), df=m)
chi2_upper = stats.chi2.ppf((1-a/2), df=m)
#s is the standard deviation
var_hat = s**2
var_lower = m*var_hat/chi2_upper
var_upper = m*var_hat/chi2_lower
```

▼ σ interval bounds

A $100(1-\alpha)\%$ confidence interval for the variance σ^2 is given by:

$$\left[\sqrt{\frac{(n-1)s^2}{\chi^2_{1-\alpha/2}}}; \sqrt{\frac{(n-1)s^2}{\chi^2_{\alpha/2}}}\right]$$

```
#a being the significance level
#m being the degree of freedom (n-1)
#this will return the interval bounds of standard deviation not variance
chi2_lower = stats.chi2.ppf((a/2), df=m)
chi2_upper = stats.chi2.ppf((1-a/2), df=m)
#s is the standard deviation
var_hat = s**2
var_lower = np.sqrt(m*var_hat/chi2_upper)
var_upper = np.sqrt(m*var_hat/chi2_lower)
```

▼ Central Limit Theorem (CLT)

The distribution of the sample mean approaches a normal distribution as the sample size n becomes sufficiently large, regardless of the shape of the population distribution.

Let $\bar{X}$ be the average of a randomly drawn sample of size n taken from a population with mean μ and variance σ^2. Then the distribution of

$$Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$$

Approaches the standard normal distribution, $N(0, 1^2)$, as $n \to \infty$

That is, if n is large enough, we can (approximately) assume:

$$\frac{\bar{X} - \mu}{\sigma/\sqrt{n}} \sim N(0, 1^2)$$

▼ Consequences of central limit theorem:

Confidence intervals for the mean can be calculated based on the t-distribution in almost all situations, as long as n is "large enough."

The assumptions for the t-test are:

1. Observations are independent
2. For small samples, the underlying population from which the sample is drawn should be approximately normally

distributed.

For larger samples, t-test is applicable for CLT.

n is considered "large enough" more or less for $n \geq 30$

▼ Overlapping confidence intervals

When 2 confidence intervals do not overlap the two groups are significantly different.

When 2 confidence intervals overlap no conclusion can be drawn without examining the confidence interval for the difference between the groups.

Model control

To help prove the assumption that some data is normally distributed, we usually look at the histogram of our sample data, however, in some cases, it can be difficult to determine whether the sample is indeed normal, especially if the sample size is small. Additionally, the shape of the histogram depends on the choice of bin size.

Another option would be to plot the empirical distribution function (ECDF).

▼ Q-Q plots

We can also use Q-Q plot (quantile-quantile plot). This plot compares the sorted observations $x_{(1)}, \ldots, x_{(n)}$ against the theoretical quantiles of the normal distribution.

Different definitions of quantiles exist:

- For $n > 10$, the preferred quantiles are: $p_i = \frac{i-0.5}{n}, i = 1, \ldots, n$
- For $n \leq 10$, the preferred quantiles are: $p_i = \frac{i-3/8}{n+1/4}, i = 1, \ldots, n$

▼ Simple observations of Q-Q plot

1. Linearity of points: Check if the points fall approximately along a straight line. This indicates that the data is consistent with the theoretical distribution (e.g., normal).
2. Systematic deviation: Look for any systematic patterns, such as curves or S-shapes, which suggest the presence of skewness in the data.
3. Ends of the plot: Observe if the points at the ends deviate significantly from the line, indicating heavy tails or kurtosis (extreme values more frequent than expected).
4. Outliers: Identify any points that fall far away from the reference line, which could be potential outliers in the dataset.

These basic observations help determine if the data distribution is approximately normal or has deviations that should be considered before performing hypothesis testing.

▼ Step-by-step for creating Q-Q plots

1. Sort the data
2. Calculate the empirical properties
3. Find the theoretical quantiles
4. plot the data quantiles vs. the theoretical quantiles
5. Check the Q-Q line to see if the data is normally distributed

▼ Transformation towards normality

If data is not normally distributed, one option is to transform the data in the hope that the transformed data will better follow a normal distribution.

In such cases, calculations will be performed on the transformed data. It may be necessary to transform the results back to the original scale at the end.

Hypothesis testing

▼ One sample

▼ Null hypothesis

We assume that μ takes a specific value

$$H_0 : \mu = \mu_0$$

Where μ is the true population mean. and we choose a significance level α.

▼ Notation

$$\begin{gathered} \mu_X, \text{ the true mean for } X \\ \sigma_X^2, \text{ the true variance for } X \\ \bar{x}, \text{ sample mean calculated from the actual data} \end{gathered}$$

▼ Stochastic variable

The stochastic variable is

$$\bar{X}$$

The mean of $\bar{X}$

$$\mathbf{E}[\bar{X}] = "\mu_{\bar{X}}" = \mu_X$$

The variance of $\bar{X}$

$$\mathbf{V}[\bar{X}] = "\sigma_{\bar{X}}^2" = \frac{\sigma_X^2}{n_X}$$

The ‘standard error’

$$s.e._{\bar{X}} = \sqrt{\frac{\sigma_X^2}{n_X}}$$
$$s.e._{\bar{x}} = \sqrt{\frac{s_X^2}{n_X}}$$

```
# Compute best estimate for the mean (and the standard error)
mean_hat = x.mean()
se_mean = x.std(ddof=1)/np.sqrt(n)
```

▼ Confidence interval for the mean

```
# confidence interval for the mean: (manually)
mu_lower = mean_hat - stats.t.ppf(0.975, df=9)*se_mean
mu_upper = mean_hat + stats.t.ppf(0.975, df=9)*se_mean
print([mu_lower, mu_upper])
```

▼ Test statistic

To “test” our null hypothesis, we consider the statistic T under the assumption that the null hypothesis is true:

$$T = \frac{\bar{X} - \mu}{S/\sqrt{n}} = \frac{\bar{X} - \mu_0}{S/\sqrt{n}}$$

This statistic follows a t-distribution: $T \sim t(n-1)$

```
stats.t.ppf(0.025, df = n-1)
```

Based on the observed data, we calculate:

$$t_{obs} = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

t_{obs} is our observation of the random variable T (under the assumption that the null hypothesis is true).

```
tobs = (mean_hat - mean_null_hyp) / se_mean
```

▼ p-value

```
# P-value
p_value= 2*stats.t.cdf(-tobs, df=n-1)
```

```
# You can also use the ttest_1samp funtion from scipy.stats:
# popmean=0 is the null hypothesis
print(stats.ttest_1samp(x, popmean=0))
```

▼ Two unpaired samples

▼ Two unpaired samples

▼ Null hypothesis

The test is valid when both samples are large (CLT) or when both samples come from normally distributed populations.

We consider the following null hypothesis about the difference in means between two independent and unpaired samples

$$\delta = \mu_X - \mu_Y$$
$$H_0 : \delta = \delta_0$$

Where δ is the true difference, μ_X and μ_Y are the true means, and we choose a significance level α.

▼ Notation

$$\delta, \text{the true difference}$$
$$d, \text{ the real difference calculated from real data}$$

```
# Difference in means
diff = mean_B - mean_A
```

▼ Stochastic variable

The stochastic variable is

$$D = \bar{X} - \bar{Y}$$

The mean of D

$$\mathbf{E}[D] = \mathbf{E}[\bar{X} - \bar{Y}] = \mathbf{E}[\bar{X}] - \mathbf{E}[\bar{Y}] = \mu_X - \mu_Y$$

The variance of D

$$\mathbf{V}[D] = \mathbf{V}[\bar{X} - \bar{Y}] = 1^2\mathbf{V}[\bar{X}] + (-1)^2\mathbf{V}[\bar{Y}] =$$
$$= \frac{\sigma_X^2}{n_X} + \frac{\sigma_Y^2}{n_Y}$$

The standard deviation of D

$$\sigma_D = \sqrt{\frac{\sigma_X^2}{n_X} + \frac{\sigma_Y^2}{n_Y}} \text{ or } "s.e._D" = \sqrt{(s.e._{\bar{X}})^2 + (s.e._{\bar{Y}})^2}$$

The ‘standard error’

$$s.e._D = \sqrt{\frac{\sigma_X^2}{n_X} + \frac{\sigma_Y^2}{n_Y}}$$
$$s.e._d = \sqrt{\frac{s_X^2}{n_X} + \frac{s_Y^2}{n_Y}}$$

```
s_A = A.std(ddof=1)
s_B = B.std(ddof=1)
n_A = len(A)
n_B = len(B)
# Standard errors
se_A = s_A/np.sqrt(n_A)
se_B = s_B/np.sqrt(n_B)
```

▼ Test statistic in a (Welch) t-test with two unpaired samples

Under the null hypothesis the test statistic is:

$$T = \frac{(\bar{X}_1 - \bar{X}_2) - \delta_0}{\sqrt{S_1^2/n_1 + S_2^2/n_2)}}$$

The test statistic is approximately t-distributed with ν degrees of freedom, where

$$\nu = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{(s_1^2/n_1)^2}{n_1-1} + \frac{(s_2^2/n_2)^2}{n_2-1}}$$

Then the observed test statistic is:

$$t_{obs} = \frac{(\bar{x}_1 - \bar{x}_2) - \delta_0}{\sqrt{s_1^2/n_1 + s_2^2/n_2}} = \frac{d - \delta_0}{s.e._d}$$

```
mean_A = #mean
mean_B = #mean
s_A = #standard deviation
s_B = #standard deviation
n_A = #amount of data
n_B = #amount of data
se_A = s_A/np.sqrt(n_A)
se_B = s_B/np.sqrt(n_B)

# Difference in means
diff = mean_A - mean_B
# Standard error of the difference
se_diff = np.sqrt(se_A**2 + se_B**2)
# Define the null hypothesis
mean_null_hyp = 0 #whatever you want it to be
# Compute the observed test statistic from the data
tobs = (diff - mean_null_hyp) / se_diff #has to be positive
# Compute v (degrees of freedom for difference-test)
v = (se_A**2 + se_B**2)**2 / (se_A**4/(n_A-1) + se_B**4/(n_B-1))
# Compute t_0.975 from t-distribution with df = v
stats.t.ppf(0.975, df=v)
```

▼ p-value

```
# P-value
print(2*stats.t.cdf(-tobs, df=v))
```

```
# You can also use the ttest_ind function from scipy.stats:
# "_ind" is for independent - i.e., two independent samples
test = stats.ttest_ind(B,A,equal_var=False)
```

▼ Two unpaired samples with pooled variance

▼ Null hypothesis

Assuming $\sigma_1^2 = \sigma_2^2$

We consider the following null hypothesis about the difference in means between two independent

$$\delta = \mu_X - \mu_Y$$
$$H_0 : \delta = \delta_0$$

Where δ is the true difference, μ_X and μ_Y are the true means, and we choose a significance level α.

▼ Notation

$$\delta, \text{the true difference}$$
$$d, \text{ the real difference calculated from real data}$$
$$s_p^2, \text{ the pooled variance}$$

```
# Difference in means
diff = mean_B - mean_A
```

▼ Stochastic variable

The stochastic variable is

$$D = \bar{X} - \bar{Y}$$

The mean of D

$$\mathbf{E}[D] = \mathbf{E}[\bar{X} - \bar{Y}] = \mathbf{E}[\bar{X}] - \mathbf{E}[\bar{Y}] = \mu_X - \mu_Y$$

The variance of D

$$\mathbf{V}[D] = \mathbf{V}[\bar{X} - \bar{Y}] = 1^2\mathbf{V}[\bar{X}] + (-1)^2\mathbf{V}[\bar{Y}] =$$
$$= \frac{\sigma_X^2}{n_X} + \frac{\sigma_Y^2}{n_Y}$$

The standard deviation of D

$$\sigma_D = \sqrt{\frac{\sigma_X^2}{n_X} + \frac{\sigma_Y^2}{n_Y}} \text{ or } "s.e._D" = \sqrt{(s.e._{\bar{X}})^2 + (s.e._{\bar{Y}})^2}$$

$$s_p^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}$$

The 'standard error'

$$s.e._D = \sqrt{\frac{\sigma_X^2}{n_X} + \frac{\sigma_Y^2}{n_Y}}$$
$$s.e._d = \sqrt{\frac{s_X^2}{n_X} + \frac{s_Y^2}{n_Y}}$$

▼ Test statistic pooled variance for two unpaired samples

Under the null hypothesis the test statistic is:

$$T = \frac{(\bar{X}_1 - \bar{X}_2) - \delta_0}{\sqrt{S_p^2/n_1 + S_p^2/n_2)}}$$

The test statistic is approximately t-distributed with $n_1 + n_2 - 2$ degrees of freedom

Then the observed test statistic is:

$$t_{obs} = \frac{(\bar{x}_1 - \bar{x}_2) - \delta_0}{\sqrt{s_p^2/n_1 + s_p^2/n_2}}$$

```
# Difference in means
diff = mean_B - mean_A
# Standard error of the difference
se_diff = np.sqrt(se_A**2 + se_B**2)
# Define the null hypothesis
mean_null_hyp = 0 #whatever you want it to be
# Compute the observed test statistic from the data
tobs = (diff - mean_null_hyp) / se_diff
# Compute v (degrees of freedom for difference-test)
v = (se_A**2 + se_B**2)**2 / (se_A**4/(n_A-1) + se_B**4/(n_B-1))
# Compute t_0.975 from t-distribution with df = v
stats.t.ppf(0.975, df=v)
```

▼ p-value

```
# P-value
print(2*stats.t.cdf(-tobs, df=v))
```

```
# You can also use the ttest_ind function from scipy.stats:
# "_ind" is for independent - i.e., two independent samples
test = stats.ttest_ind(B,A,equal_var=True)
```

▼ Confidence intervals for the difference in means

For two samples $(x_1, \ldots, x_{n1})$ and $(y_1, \ldots, y_{n2})$, the $(1 - \alpha)$ confidence interval for $\mu_1 - \mu_2$ is given by

$$\bar{x} - \bar{y} \pm t_{1-\alpha/2} \cdot \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

where $t_{1-\alpha/2}$ is the $(1 - \alpha)$-quantile in the t-distribution with ν degrees of freedom

$$d \pm t_{1-\alpha/2} \cdot s.e._d$$

```
# [lower, upper] confidence interval for the difference
diff_lower = diff - stats.t.ppf(0.975, df=v)*se_diff
diff_upper = diff + stats.t.ppf(0.975, df=v)*se_diff
print([diff_lower,diff_upper])
```

▼ Two paired samples

Compares the means of two measurements taken from the same individual, object, or related units. a good example is pre-test and post-test.

```
# equal to a paired t-test for two samples:
test2 = stats.ttest_rel(x1, x2) # "_rel" is for related samples (= paired samples)
print(test2)
print(test2.confidence_interval(0.95))
```

▼ Hypothesis test: t-test

To find out whether the observed values t_{obs} is too "extreme", we test if t_{obs} lies within the interval $\pm\, t_{1-\alpha/2}$ (which depends on our chosen significance level). The test result is either Yes/No, along with the chosen significance level.

If $|t_{obs}| > t_{1-\alpha/2}$, we conclude that $|t_{obs}|$ is too large to believe in the null hypothesis, and we say we reject the null hypothesis

▼ p-value

For a (quantitative) situation with one sample, the p-value is given by:

$$p\text{-value} = 2 \cdot P(T > |t_{obs}|) =$$
$$= P(T < |t_{obs}|) + P(T > |t_{obs}|)$$

Where T follows a t-distribution with $(n-1)$ degrees of freedom.

Lastly, we compare the p-value with the chosen significance level α.

▼ Definition and interpretation of p-value

▼ Statistical significance and terminology

A null hypothesis is rejected if $p\text{-value} < \alpha$.

Otherwise, the null hypothesis cannot be rejected.

Statistical significance: An effect is said to be (statistically) significant if the p-value is less than the significance level α.

This terminology is most meaningful when the null hypothesis is defined as: $H_0 : \mu = \mu_0 = 0$, which translates to the null hypothesis of “no effect”.

It can also be said that the μ is significantly different from μ_0

▼ Type I and Type II errors

There are two types of errors (only one at a time):

▼ Type I:

Type I: Rejecting H_0 when H_0 is true.

The Type I error is called a false positive.

In this terminology, $H_0 =$” negative” (”no effect”)

The risk of the error is referred to as,

$$P(\text{Type I error}) = \alpha$$

▼ Type II:

Type II: Accepting (not rejecting) H_0 when H_1 is true

The Type II error is called a false negative.

In this terminology, $H_1 =$” positive” (there is an ”effect”)

The risk of the error is referred to as,

$$P(\text{Type II error}) = \beta$$

A smaller α = larger β and vice-versa

▼ Normality assumptions

To check whether data comes from underlying normal distributions, we must do qq plots of the data.

For samples with $n \leq 10$ the formula is

$$p_i = \frac{i - 3/8}{n + 1/4}$$

```
sm.qqplot(A,line="q",a=1/2\)
```

For samples with $n > 10$ the formula is

$$p_i = \frac{i - 0.5}{n}$$

```
sm.qqplot(A,line="q",a=3/8\)
```

Power and sample size

▼ Experimental planning with precision requirements

Margin of Error (ME) is defined as,

$$ME = t_{1-\alpha/2}\frac{\sigma}{\sqrt{n}}$$

for sufficiently large samples, we can approximate $t_{1-\alpha/2}$ (from the t-distribution) with $z_{1-\alpha/2}$ (from the normal distribution)

▼ Sample size for confidence interval Based on a sample

If σ is known, or estimated to be a certain value, we can calculate the required sample size to achieve a given margin of error, with probability $1-\alpha$

$$n = \left(\frac{z_{1-\alpha/2} \cdot \sigma}{ME}\right)^2$$

▼ Planning (power and sample size)

Experimental planning and “power” are closely related to the

▼ Type I:

Type I: Rejecting H_0 when H_0 is true.

The Type I error is called a false positive.

In this terminology, $H_0 =$” negative” (”no effect”)

The risk of the error is referred to as,

$$P(\text{Type I error}) = \alpha$$

▼ Type II:

Type II: Accepting (not rejecting) H_0 when H_1 is true

The Type II error is called a false negative.

In this terminology, $H_1 =$” positive” (there is an ”effect”)

The risk of the error is referred to as,

$$P(\text{Type II error}) = \beta$$

A smaller α = larger β and vice-versa

In this graph, the red curve represents the null hypothesis and the green curve the alternative hypothesis. α is represented by the red area, β by the green area, and $(1-\beta)$ is called power.

Power is defined by the probability of correctly rejecting H_0 or the probability of detecting a (proposed) effect)

Typical desired values for desired power are 80%, 90% (something relatively large)

If we know (or assume) four out of five following quantities, we can find the missing one:

- Sample size, n
- Significance level, α, at which we are testing.
- Difference in means (effect size), $\mu_0 - \mu_1$
- Population standard deviation, σ
- Power, $1 - \beta$

▼ Formula for sample size with one sample

for a t-test with one sample, where α, β, and σ are given

$$n = \left(\sigma \frac{z_{1-\beta} + z_{1-\alpha/2}}{\mu_0 - \mu_1}\right)^2$$

Here $\mu_0 - \mu_1$ is the difference in means that we want to measure, while $z_{1-\beta}$ and $z_{1-\alpha/2}$ are quantiles of the standard normal distribution (they must always be positive).

```
delta = #mean difference
sd = #standard deviation
```

```
# we can also use the python function TTestIndPower().solve_power (for one sample power calculations):
print(smp.TTestPower().solve_power(effect_size=delta/sd, alpha=0.05, power=0.90, ratio=k))
```

```
# we can also use the python function TTestIndPower().solve_power (for one sample power calculations):
print(smp.TTestPower().solve_power(effect_size=delta/sd, alpha=0.05, nobs1=10, ratio=k))
```

```
# we can also use the python function TTestIndPower().solve_power (for one sample power calculations):
print(smp.TTestPower().solve_power(alpha=0.05, power=0.90, nobs1=10, ratio=k))
```

▼ Formula for sample size with two samples

We will assume that the variance (σ^2)is equal in the two populations

for a t-test with two samples, where α, β, and σ are given

$$n_1 = (k+1)\left(\sigma \frac{z_{1-\beta} + z_{1-\alpha/2}}{\mu_0 - \mu_1}\right)^2$$

Here $\mu_0 - \mu_1$ is the difference in means that we want to measure, while $z_{1-\beta}$ and $z_{1-\alpha/2}$ are quantiles of the standard normal distribution. Additionally $k = n_1/n_2$ is the ratio of the two sample sizes, so we have $n_2 = n_1/k$

Simulation

You can simulate from (almost) any distribution by inverting the distribution function F and inserting uniformly distributed numbers.

▼ All distributions can be “derived” from the uniform distribution

According to Theorem 2.51

If $U \sim \text{Uniform}(0,1)$ and F is the distribution for a given probability distribution, then $F^{-1}(U)$ will follow the distribution given by F

▼ In Python

Many distributions are available via the `scipy.stats` package, and you can simulate random numbers using the `.rvs()` function (stands for random variates)

```
stats.binom.rvs()
stats.poisson.rvs()
stats.hypergeom.rvs()
stats.norm.rvs()
stats.lognorm.rvs()
stats.expon.rvs()
stats.uniform.rvs()
stats.t.rvs()
stats.chi2.rvs()
stats.f.rvs()
```

```
m = #mean
s = #standard deviation
n = #ammount of data
x = stats.norm.rvs(loc=m, scale=s, size =n)
```

▼ Propagation of error

We wish to find:

$$\sigma^2_{f(X_1,...,X_n)} = \text{Var}(f(X_1,...,X_n))$$

Linear combination of independent variables:

$$\sigma^2_{f(X_1,...,X_n)} = \sum_{i=1}^{n} a_i^2\sigma_i^2 \text{ when}$$
$$f(X_1,...X_n) = \sum_{i=1}^{n} a_i X_i \text{ (with independence)}$$

For non-linear functions of independent variables $X_1, \dots X_n$:

$$\sigma^2_{f(X_1,...,X_n)} \approx \sum_{i=1}^{n} \left(\frac{\partial f}{\partial x_i}\right)^2 \sigma_i^2$$

▼ Error propagation - by simulation

Assuming we have (actual) measurements $x_1, \dots, x_n$ with known/assumed (estimated) variances $\sigma_1^2, \dots, \sigma_n^2$.

- Simulate k outcomes of all n measurements from the assumed distributions, for example $X_i^{(j)} \sim N(x_i\sigma_i^2)$ for $j = 1, \dots, k$, $i = 1, \dots, n$.
- Calculate the standard deviation as the observed standard deviation of the k simulated values of $f(X_1^{(j)}, \dots X_n^{(j)})$:

$$s^{\text{sim}}_{f(X_1,...,X_n)} = \sqrt{\frac{1}{k-1}\sum_{i=1}^{k}(f_j - \bar{f})^2}$$
$$\text{where } f_j = f(X_1^{(j)}, \dots X_n^{(j)})$$

▼ Bootstrapping

▼ Parametric bootstrap

Simulate repeated samples from the assumed (and estimated) distribution.

▼ Confidence interval for any θ by parametric bootstrap

Assume we have actual observations $x_1, .., x_n$, and that they came from some probability distribution f (pdf)

- Simulate $k \times n$ observations from the assumed pdf (with $\mu = \bar{x}$).
- Calculate the estimate $\hat{\theta}$ for each of the k samples, $\hat{\theta}_1^*, \dots, \hat{\theta}_k^*$.
- Find the $\alpha/2$- and the $(1-\alpha/2)$-quantiles in $\hat{\theta}_1^*, \dots, \hat{\theta}_k^*$, so that we get a $(1-\alpha)$-confidence interval:

$$\left[q^*_{\alpha/2},\ q^*_{1-\alpha/2}\right]$$

▼ Python

▼ Mean

```
data =
n = len(data)
k = 100000
sim_data = stats. .rvs(size=(k,n), loc=data.mean(), scale=data.std(ddof=1))
sim_means = sim_data.mean(axis=1)
# From simulated means we can find the 95% CI for the mean:
CI = np.percentile(sim_means, [2.5, 97.5], method="averaged_inverted_cdf")
print(CI)
# visualisation of the CI for the mean:
plt.hist(sim_means, density=True, bins=30)
plt.plot([12.57, 12.57], [0,0.05], '--', color="black")
plt.plot([44.35, 44.35], [0,0.05], '--', color="black")
plt.show()
```

▼ Median

```
data =
n = len(data)
k = 100000
sim_data = stats. .rvs(size=(k,n), loc=data.mean(), scale=data.std(ddof=1))
sim_median = np.median(sim_data, axis=1)
# From simulated medians we can find the 95% CI for the median:
CI = np.percentile(sim_median, [2.5, 97.5], method="averaged_inverted_cdf")
print(CI)
plt.hist(sim_medians, density=True, bins=30)
plt.plot([CI[0], CI[0]], [0,0.05], '--', color="black")
plt.plot([CI[1], CI[1]], [0,0.05], '--', color="black")
plt.show()
```

▼ Confidence interval for any comparison $\theta_1 - \theta_2$ based on two samples by parametric bootstrap

Assume we have actual observations $x_1, .., x_n$, and that they came from probability distributions f_1 and f_2 (The distributions are assumed to be independent)

- Simulate k groups of 2 samples with n_1 and $n_2$observations, respectively, from the assumed distributions with set to $\hat{\mu}_1 = \bar{x}$ and $\hat{\mu}_2 = \bar{y}$.
- Calculate the difference between the sample statistics in each of the k samples, $\hat{\theta}^*_{x1} - \hat{\theta}^*_{y1}, \dots, \hat{\theta}^*_{xk} - \hat{\theta}^*_{yk}$.
- Find the $\alpha/2$- and the $(1-\alpha/2)$-quantiles in these, $q^*_{\alpha/2}$ and $q^*_{1-\alpha/2}$, to obtain a $(1-\alpha)$-confidence interval:

$$\left[q^*_{\alpha/2},\ q^*_{1-\alpha/2}\right]$$

▼ Python

▼ Mean

```
x =
y =
n1 = len(x)
n2 = len(y)
k = 100000
x_sim= stats. .rvs(size=(k,n1), scale=x.mean())
y_sim= stats. .rvs(size=(k,n2), scale=y.mean())

x_means = x_sim.mean(axis=0)
y_means = y_sim.mean(axis=0)
```

```
diffs = x_means - y_means
# find 95% confidence interval for the difference
CI = np.percentile(diffs, [2.5, 97.5], method="averaged_inverted_cdf")
print(CI)

plt.hist(diffs, bins=30)
plt.plot([CI[0], CI[0]], [0,.2], '--', color="black")
plt.plot([CI[1], CI[1]], [0,.2], '--', color="black")
plt.show()
```

▼ Note

Other parameters in the distribution should also match the data as well as possible.

- For the normal distribution, choose μ and σ to match the samples $\bar{x}$ and s.
- Some distributions have more than one parameter
- Generally, one should also use the so-called maximum likelihood approach to match the distribution to the sample data.

▼ Non-parametric bootstrap

Simulate repeated samples directly from the data.

▼ Confidence interval for any θ by non-parametric bootstrap

Assume we have observations $x_1, .., x_n$.

- Simulate $k \times n$ observations by random sampling (with replacement) from the observed data (re-sampling).
- Calculate the estimate $\hat{\theta}$ for each of the k samples, $\hat{\theta}_1^*, \ldots, \hat{\theta}_k^*$.
- Find the $\alpha/2$- and the $(1-\alpha/2)$-quantiles in $\hat{\theta}_1^*, \ldots, \hat{\theta}_k^*$, so that we get a $(1-\alpha)$-confidence interval:

$$\left[q^*_{\alpha/2},\ q^*_{1-\alpha/2}\right]$$

▼ Python

▼ Mean

```
data =
n = len(data)
k = 100000
sim_data = np.random.choice(data,size=(n,k))
sim_means = sim_data.mean(axis=0)
# From simulated means we can find the 95% CI for the mean:
CI = np.percentile(sim_means, [2.5, 97.5], method="averaged_inverted_cdf")
print(CI)
# always visualise :-)
plt.hist(sim_means, density=True,bins=30)
plt.plot([CI[0], CI[0]], [0,.2], '--', color="black")
plt.plot([CI[1], CI[1]], [0,.2], '--', color="black")
plt.show()
```

▼ Median

```
data =
n = len(data)
k = 100000
sim_data = np.random.choice(data,size=(n,k))
sim_median = np.median(sim_data,axis=0)
# From simulated median we can find the 95% CI for the mean:
CI = np.percentile(sim_median, [2.5, 97.5], method="averaged_inverted_cdf")
print(CI)
plt.hist(sim_medians, density=True)
plt.plot([CI[0], CI[0]], [0,.2], '--', color="black")
plt.plot([CI[1], CI[1]], [0,.2], '--', color="black")
plt.show()
```

▼ Confidence interval for any comparison $\theta_1 - \theta_2$ based on two samples by parametric bootstrap

Assume we have actual observations $x_1, .., x_n$, and $y_1, \ldots, y_n$

- Simulate k pairs of bootstrap samples with n_1 and $n_2$observations, respectively, (by random sampling with replacement).
- Calculate the difference between the estimates in each of the k pairs of bootstrap samples, $\hat{\theta}^*_{x1} - \hat{\theta}^*_{y1}, \ldots, \hat{\theta}^*_{xk} - \hat{\theta}^*_{yk}$.
- Find the $\alpha/2$- and the $(1-\alpha/2)$-quantiles in these, $q^*_{\alpha/2}$ and $q^*_{1-\alpha/2}$, to obtain a $(1-\alpha)$-confidence interval:

$$\left[q^*_{\alpha/2},\ q^*_{1-\alpha/2}\right]$$

If the null hypothesis is not in the confidence interval then it is rejected.

▼ Python

▼ Mean

```
x =
y =
n1 = len(x)
n2 = len(y)
k = 100000
x_sim= stats.random.choice(x, size=(n1, k))
y_sim= stats.random.choice(y, size=(n2, k))

sim_mean_dif = x_sim.mean(axis=0) - y_sim.mean(axis=0)

# find 95% confidence interval for the difference
CI = np.percentile(sim_mean_dif, [2.5, 97.5], method="averaged_inverted_cdf")
print(CI)

plt.hist(seam_mean_dif, bins=30)
plt.plot([CI[0], CI[0]], [0,.2], '--', color="black")
plt.plot([CI[1], CI[1]], [0,.2], '--', color="black")
```

```
plt.show()
```

Linear regression

▼ Simple linear regression models

▼ Scatter plot

We have n pairs of data points (x_i, y_i). If the data points lie on a straight line, the relationship between x and y values can be described by the equation:

$$y_i = \beta_0 + \beta_1 x_i$$

To describe the random variation we denote the linear regression model as,

$$\begin{gathered} Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \text{ for } i = 1, ..., n \\ Y_i, \text{ is the dependent variable} \\ x_i, \text{ is the explanatory variable} \\ \varepsilon_i, \text{ is the deviation (residual)} \end{gathered}$$

We assume $\varepsilon_i \sim N(0, \sigma^2)$ (and i.i.d.).

```
x = np.array([])
y = np.array([])
data = pd.DataFrame({'x': x,'y': y})
fitdata=smf.ols(formula = 'y ~ x', data=data).fit()
print(fitdata.summary(slim=True)
```

▼ Least squares method

To estimate the parameters β_0 and β_1. we minimize the sum of the squared residuals (Residual Sum of Squares, RSS):

$$RSS(\beta_0, \beta_2) = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$

That is, we choose $\hat{\beta}_0$ and $\hat{\beta}_1$ so that they minimize RSS.

▼ (Theorem 5.4) 'Least squares' estimator for β_0 and β_1 are given by:

$$\begin{gathered} \hat{\beta}_1 = \frac{\sum_{i=1}^{n}(Y_i - \bar{Y})(x_i - \bar{x})}{S_{xx}}, \\ \hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{x} \\ \text{where } S_{xx} = \sum_{i=1}^{n} (x_i - \bar{x})^2 \end{gathered}$$

▼ (Theorem 5.4) 'Least squares' estimates for β_0 and β_1 are given by:

$$\begin{gathered} \hat{\beta}_1 = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(x_i - \bar{x})}{S_{xx}}, \\ \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \\ \text{where } S_{xx} = \sum_{i=1}^{n} (x_i - \bar{x})^2 \end{gathered}$$

```
### 1 ###  Estimate parameters beta_0_hat and bata_1_hat *manually*
x =
y =
Sxx = np.sum(x - x.mean())**2

beta_1_hat = np.sum((x - x.mean())*(y - y.mean())) / Sxx

beta_0_hat = y.mean() - beta_1_hat*x.mean()

print(beta_0_hat, beta_1_hat)
```

```
data["y_pred"] = beta_0_hat + beta_1_hat*data["x"]
data["residuals"] = data["y"] - data["y_pred"]
```

▼ (Theorem 5.8) Distributions and standard deviations of $\hat{\beta}_0$ and $\hat{\beta}_1$

The estimators $\hat{\beta}_0$ and $\hat{\beta}_1$ are normally distributed with variances:

$$\begin{gathered} V[\hat{\beta}_0] = \frac{\sigma^2}{n} + \frac{\bar{x}^2 \sigma^2}{S_{xx}}, \\ V[\hat{\beta}_1] = \frac{\sigma^2}{S_{xx}}, \\ Cov[\hat{\beta}_0, \hat{\beta}_1] = -\frac{\bar{x}\sigma^2}{S_{xx}} \end{gathered}$$

Since σ^2 is unknown, we use the central estimate for σ^2:

$$\hat{\sigma}^2 = \frac{RSS(\hat{\beta}_0, \hat{\beta}_1)}{n-2} = \frac{\sum_{i=1}^{n} e_i^2}{n-2}$$

Thus, we estimate the variance (standard deviation) for the error and, thereby, also the variances (standard deviations) of the estimators. We denote these $\hat{\sigma}^2_{\hat{\beta}_0}$ and $\hat{\sigma}^2_{\hat{\beta}_1}$.

We obtain the following estimates of the standard deviations for $\hat{\beta}_0$ and $\hat{\beta}_1$:

$$\hat{\sigma}_{\beta_0} = \hat{\sigma}\sqrt{\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}},\ \hat{\sigma}_{\beta_1} = \hat{\sigma}\sqrt{\frac{1}{S_{xx}}}$$

```
n = len(x)
RSS = np.sum(data["residuals"]**2)
sigma_hat = np.sqrt(RSS/(n-2))
se_beta_0_hat = sigma_hat*np.sqrt(1/n + data["x"].mean()**2 / Sxx)
se_beta_1_hat = sigma_hat*np.sqrt(1/Sxx)

print(se_beta_0_hat, se_beta_1_hat)
```

▼ Standard deviation of the linear regression model

```
np.sqrt(fitdata.scale) # or np.sqrt(fitdata.mse_resid)
```

▼ Python cheat code

```
x = np.array([])
y = np.array([])
data = pd.DataFrame({'x': x,'y': y})

fitdata = smf.ols(formula = 'y ~ x', data=data).fit()  # OBS: use the statsmodels.formula.api library (smf)
print(fitdata.summary(slim=True))
```

▼ Hypothesis tests and confidence intervals for β_0 and β_1

We can conduct hypothesis tests for the parameters in a linear regression model:

$$H_{0,i}:\ \beta_i = \beta_{0,i},$$
$$H_{1,i}:\ \beta_i \neq \beta_{0,i}.$$

Under the null hypothesis ($\beta_0 = \beta_{0,0}$ and $\beta_1 = \beta_{0,1}$), the test statistics are,

$$T_{\beta_0} = \frac{\hat{\beta}_0 - \beta_{0,0}}{\hat{\sigma}_{\beta_0}},\ T_{\beta_1} = \frac{\hat{\beta}_1 - \beta_{0,1}}{\hat{\sigma}_{\beta_1}}$$

t-distributed with $n - 2$ degrees of freedom

$(1 - \alpha)$ confidence intervals for β_0 and β_1 are given by:

$$\hat{\beta}_0 \pm t_{1-\alpha/2}\hat{\sigma}_{\beta_0},$$
$$\hat{\beta}_1 \pm t_{1-\alpha/2}\hat{\sigma}_{\beta_1}$$

▼ Python cheat code

```
x = np.array([])
y = np.array([])
data = pd.DataFrame({'x': x,'y': y})

fitdata = smf.ols(formula = 'y ~ x', data=data).fit()  # OBS: use the statsmodels.formula.api library (smf)
print(fitdata.summary(slim=True))
```

```
print(fit.pvalues) # if the p-values are too smallprint
print(fitdata.conf_int(a)) #confidence intervals with different alphas
```

▼ Confidence interval for the regression line

A simple linear regression model can be written as

$$Y \sim N(\beta_0 + \beta_1 x, \sigma^2)$$
$$\text{or } Y \sim N(\mu(x), \sigma^2),\ \text{where } \mu(x) = \beta_0 + \beta_1 x$$

For $x = x_0$, we can find a confidence interval for $\mu(x_0) = \beta_0 + \beta_1 x_0$.

The $(1 - \alpha)$ confidence interval for the regression line at $x = x_0$ (for $\mu(x_0)$) can be found by:

$$(\hat{\beta}_0 + \hat{\beta}_1 x_0) \pm t_{\alpha/2} \cdot \hat{\sigma}\sqrt{\frac{1}{n} + \frac{(x_0 - \bar{x})^2}{S_{xx}}}$$

```
x = np.array([])
y = np.array([])
data = pd.DataFrame({'x': x, 'y': y})
# fit a linear regression model
linfit = smf.ols(formula = 'y ~ x', data=data).fit()
print(linfit.summary(slim=True))
mini = #depending on your data the minimum value it might reach
maxi = #the max
step = #the step
x_pred = pd.DataFrame({'x': np.arange(mini, maxi, step)})
pred = linfit.get_prediction(x_pred).summary_frame(alpha=0.05)
print(pred.head()) #mean_ci_lower and upper
```

```
fitdata.conf_int(alpha=0.005)
```

▼ Prediction interval for a new observation

We want a prediction interval for a new observation Y_0 and $x = x_0$

The $(1-\alpha)$ prediction interval for a new observation Y_0 at $x = x_0$ can be found by:

$$(\hat{\beta}_0 + \hat{\beta}_1 x_0) \pm t_{\alpha/2} \cdot \hat{\sigma}\sqrt{1 + \frac{1}{n} + \frac{(x_0 - \bar{x})^2}{S_{xx}}}$$

The prediction interval will contain the observed y_0 in $100(1-\alpha)\%$ of cases.

For fixed α, the prediction interval is always larger than the confidence interval.

```
x = np.array([])
y = np.array([])
data = pd.DataFrame({'x': x, 'y': y})
# fit a linear regression model
linfit = smf.ols(formula = 'y ~ x', data=data).fit()
print(linfit.summary(slim=True))
mini = #depending on your data the minimum value it might reach
maxi = #the max
step = #the step
x_pred = pd.DataFrame({'x': np.arange(mini, maxi, step)})
pred = linfit.get_prediction(x_pred).summary_frame(alpha=0.05)
print(pred.head()) #obs_ci_lower and upper
```

```
k = #what you want the prediction
x_pred = pd.DataFrame({'x': np.array([k])})
pred = fitdata.get_prediction(x_pred).summary_frame(alpha=0.05)
print(pred.head()) #obs_ci_lower and upper
```

▼ Variance and correlation

The explained variance in a model is R^2 (R-squared) calculated with,

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$
$$\text{where } \hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_1$$

The proportion of the total variance explained by the model.

The correlation ρ is a measure of linear relationship between two stochastic variables.

The estimated (i.e., empirical) correlation satisfies,

$$\hat{\rho} = R = \sqrt{R^2} sign(\hat{\beta}_1)$$
$$\text{where } sign(\hat{\beta}_1) = \begin{cases} -1 & \text{if } \hat{\beta}_1 \leq 0 \\ 1 & \text{if } \hat{\beta}_1 > 0 \end{cases}$$

Thus, positive correlation with positive slope and negative correlation with negative slope.

▼ Python cheat code

```
x = np.array([])
y = np.array([])
data = pd.DataFrame({'x': x,'y': y})

fitdata = smf.ols(formula = 'y ~ x', data=data).fit()  # OBS: use the statsmodels.formula.api library (smf)
print(fitdata.summary(slim=True))
```

▼ Test for significant correlation

Test for significant correlation (linear relationship) between two variables:

$$H_0 : \rho = 0$$
$$H_1 : \rho \neq 0$$

Is equivalent to

$$H_0 : \beta_1 = 0$$
$$H_1 : \beta_1 \neq 0$$

Where β_1 is the slope in the simple linear regression model.

▼ Model control - Analysis of residuals (validity of assumptions)

Check the normality assumption with a QQ-plot.

Check for any systematic deviations by plotting the residuals (e_i) as a function of the fitted values ($\hat{y}_i$)

▼ Multiple linear regression

An extension of the simple linear regression model, where several explanatory/independent variables are included.

In a multiple linear regression with p explanatory variables, the deterministic variables are denoted $x_1, x_2, \ldots x_p$.

We model a linear relationship between Y and $x_1, x_2, \ldots x_p$, using a regression model of the form,

$$Y_i = \beta_0 + \beta_1 x_{1,i} + ... + \beta_p x_{p,i} + \varepsilon_i$$

Where the errors are independent and identically distributed (i.i.d) with $\varepsilon_i \sim N(0, \sigma^2)$.

▼ Multiple linear regression parameters ($\hat{\beta}_i$)

▼ Representation of $\hat{\beta}_i$

The parameter $\hat{\beta}_i$ represents

- The expected change in y when x_i changes by one unit.

- The effect of x_i given the other variables.
- The effect of x_i adjusted for the effects of the other variables.
- The effects of x_i "when the other variables are unchanged

▼ Estimation of $\hat{\beta}_i$ and $\hat{\sigma}_{\beta_i}$

Estimation and prediction are performed as in the simple linear regression model.

The parameter estimates are found by minimizing RSS:

$$(\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_p) = \operatorname{argmin} \sum_{i=1}^{n} RSS(b_0, b_1, ..., b_p)$$
$$\operatorname{argmin} \sum_{i=1}^{n} RSS(b_0, b_1, ..., b_p) = \operatorname{argmin} \sum_{i=1}^{n} e_i^2 =$$
$$= \operatorname{argmin} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

The predicted (fitted) values are found by:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i,1} + ... + \hat{\beta}_p x_{i,p}$$

and the residuals are then:

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

▼ Python cheat code

```
x1 = np.array([])
x2 = np.array([])
y = np.array([])
data = pd.DataFrame({'x1': x1,'x2': x2,'y': y})

fitdata = smf.ols(formula = 'y ~ x1 + x2', data=data).fit()  # OBS: use the statsmodels.formula.api library (smf)
print(fitdata.summary(slim=True))
sigma = np.sqrt(fitdata.mse_resid) # fit.mse_resid or fit.scale
print(sigma)
```

▼ Hypothesis tests and confidence intervals for $\hat{\beta}_i$

We can conduct hypothesis tests for the parameters in a linear regression model:

$$H_{0,i}: \ \beta_i = \beta_{0,i},$$
$$H_{1,i}: \ \beta_i \neq \beta_{0,i}.$$

Under the null hypothesis ($\beta_i = \beta_{0,i}$) the test statistics are,

$$T_{\beta_i} = \frac{\hat{\beta}_i - \beta_{0,i}}{\hat{\sigma}_{\beta_i}}$$

t-distributed with $n - (p + 1)$ degrees of freedom

$(1 - \alpha)$ confidence intervals for β_i is given by:

$$\hat{\beta}_i \pm t_{1-\alpha/2} \hat{\sigma}_{\beta_i}$$

▼ Confidence intervals

```
x1 = np.array([])
x2 = np.array([])
x3 = np.array([])
y = np.array([])
data = pd.DataFrame({'x1': x1,'x2': x2,'x3': x3,'y': y})
fitdata = smf.ols(formula = 'y ~ x1 + x2 + x3', data=data).fit()
#mini is the minimum that value takes, maxi the maximum and n is the number of datapoints
New_data = pd.DataFrame({'x1':   np.linspace(mini1, maxi2, n),
                         'x2':   np.linspace(mini2, maxi2, n),
                         'x3':   data["x3"].mean()})
pred = fitdata.get_prediction(New_data).summary_frame(alpha=0.05)
print(pred.head(10)) #mean_ci_lower and upper
```

▼ Prediction intervals

```
x1 = np.array([])
x2 = np.array([])
x3 = np.array([])
y = np.array([])
data = pd.DataFrame({'x1': x1,'x2': x2,'x3': x3,'y': y})
fitdata = smf.ols(formula = 'y ~ x1 + x2 + x3', data=data).fit()
#mini is the minimum that value takes, maxi the maximum and n is the number of datapoints
New_data = pd.DataFrame({'x1':   np.linspace(mini1, maxi2, n),
                         'x2':   np.linspace(mini2, maxi2, n),
                         'x3':   data["x3"].mean()})
pred = fitdata.get_prediction(New_data).summary_frame(alpha=0.05)
print(pred.head(10)) #obs_ci_lower and upper
```

▼ Model selection

It is not always better to include more variables. There are 2 main ways of selecting,
Backward selection: Remove the least significant variables (with the largest
p-value) one at a time, until all remaining parameters are significant.

Forward selection: Start with the most significant variable and add one at a time until there are no more significant extensions

There is no certain method to find the best model as selecting a model requires subjective decisions.

▼ Model control (validity of assumptions)

Consists on analyzing the residuals to check if the assumptions are met

QQ-plot is used to check the normality of the residuals (we want it to follow a straight line), while the scatter plot is used to look for patterns in the residuals (we want this to look random)

▼ QQ-plot and Residuals vs fitted values in Python

```
x1 = np.array([])
x2 = np.array([])
x3 = np.array([])
y = np.array([])
data = pd.DataFrame({'x1': x1,'x2': x2,'x3': x3,'y': y})
fitdata = smf.ols(formula = 'y ~ x1 + x2 + x3', data=data).fit()
residuals = fitdata.resid
fittedvalues = fitdata.fittedvalues

fig, (ax1, ax2) = plt.subplots(1,2,figsize=(12,4))
# qq-plot of residuals:
sm.qqplot(residuals,ax=ax1, line='q')
ax1.set_title("QQ plot of residuals")

# plot residuals versus fitted values:
ax2.scatter(fittedvalues,residuals)
ax2.set_xlabel("fitted values")
ax2.set_ylabel("residuals")

plt.tight_layout()
plt.show()
```

▼ Residuals vs individual explanatory variables in Python

```
x1 = np.array([])
x2 = np.array([])
x3 = np.array([])
y = np.array([])
data = pd.DataFrame({'x1': x1,'x2': x2,'x3': x3,'y': y})
fitdata = smf.ols(formula = 'y ~ x1 + x2 + x3', data=data).fit()
residuals = fitdata.resid
fittedvalues = fitdata.fittedvalues
data["residuals"] = residuals

# plot residuals versus fitted values:
# first subplot
ax0.scatter(data["x1"],data["residuals"])
ax0.set_xlabel("x1")
ax0.set_ylabel("residuals")
# second subplot
ax1.scatter(data["x2"],data["residuals"])
ax1.set_xlabel("x2")
ax1.set_ylabel("residuals")
# third subplot
ax2.scatter(data["x3"],data["residuals"])
ax2.set_xlabel("x3")
ax2.set_ylabel("residuals")

plt.tight_layout() # this makes sure that the plots do not overlap
plt.show()
```

▼ Curvilinearity

Regression models for non-linear data based on Taylor expansions. If we want to use a model of the type,

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i$$

Then we can use a multiple linear regression model,

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \varepsilon_i$$
$$\text{where } x_{i,1} = x_i,\ x_{i,2} = x_i^2$$

And use the same methods as for multiple linear regression.

▼ Collinearity

If two (or more) explanatory variables have perfect linear relationship, then we cannot determine which is explanatory. It is also an issue if the relationship is close to linear.

With two highly correlated x-variables, together neither may have a “unique” effect, but separately they may have a large effect

Categorical and count data

▼ Different analyses and data types

▼ Means in quantitative data

- Hypothesis test for a single mean based on one sample
- Hypothesis test for two means based on two samples
- Hypothesis test for multiple means based on several samples

▼ Proportions in qualitative data

- Hypothesis test for a single proportion based on one sample
- Hypothesis test for two proportions based on two samples
- Hypothesis test for multiple proportions based on several samples

▼ Estimation of proportions

We define the random variable P as the number of “successes” (X) out of a total (n):

$$P = \frac{X}{n}$$

From sample data with x "successes" (sample size n), we estimate the proportions as:

$$\hat{p} = \frac{x}{n}$$

Note that $P \in [0; 1]$, p is the "true" population probability of a "success"

▼ Binomial distribution and proportions

If we assume the outcome/interest variable y_i to be binary (yes/no, success/failure) than the number of "successes" follows a binomial distribution with the density function:

$$f(x; n, p) = \binom{n}{x} p^x (1-p)^{n-x}$$

Mean and variance in the binomial distribution

$$\mathbf{E}[X] = np$$
$$\mathbf{V}[X] = np(1-p)$$

▼ Mean and variance for proportions

$$\mathbf{E}[P] = \mathbf{E}\left[\frac{X}{n}\right] = \frac{np}{n} = p$$
$$\mathbf{V}[P] = \mathbf{V}\left[\frac{X}{n}\right] = \frac{1}{n^2}\mathbf{V}[X] = \frac{p(1-p)}{n}$$

Thus, we can define,

$$\sigma_P = \sqrt{\frac{p(1-p)}{n}}$$

Note that σ_P is largest when $p = 1/2$

▼ Confidence interval for a single proportion

If the sample is large, then the $(1-\alpha)$-confidence interval for p is given by:

$$\hat{p} \pm z_{1-\alpha/2}\sigma_P$$

In practice, $\hat{p}$ is substituted for p in the formula $\sigma_P = \sqrt{p(1-p)/n}$

```
n = # total number data in the sample
x = # number of data in the sample that is in the category
p_hat = x/n
se_p_hat = np.sqrt(p_hat*(1-p_hat)/n)
z = stats.norm.ppf(0.975, loc=0, scale=1)
print([p_hat - z*se_p_hat, p_hat + z*se_p_hat])
```

Assume $X \sim \text{bin}(n, p)$. The normal distribution is a good approximation for the binomial distribution if np and $n(1-p)$ (expected number of successes and failures) are both at least 15.

If the sample is not large, we use $\tilde{x} = x + 2$ and $\tilde{n} = n + 4$. And the $(1-\alpha)$-confidence interval for p is given by:

$$\tilde{p} \pm z_{1-\alpha/2}\sqrt{\tilde{p}(1-\tilde{p})/\tilde{n}}$$

```
n = # total number data in the sample
x = # number of data in the sample that is in the category
p_tilde = (x+2)/(n+4)
se_p_tilde = np.sqrt(p_tilde*(1-p_tilde)/(n+4))
z = stats.norm.ppf((1-a/2), loc=0, scale=1)
print([p_tilde - z*se_p_tilde, p_tilde + z*se_p_tilde])
```

▼ Margin of error (ME)

Margin of error corresponds to half the width of the $(1-\alpha)$-confidence interval, and describes the expected precision (minimum desired precision) of the estimate $\hat{p}$

At a $(1-\alpha)$-confidence level, the margin of error is:

$$ME = z_{1-\alpha/2}\sqrt{\frac{p(1-p)}{n}}$$

Where we estimate p with $\hat{p} = \frac{x}{n}$.

▼ Python for large samples

```
n = # total number data in the sample
x = # number of data in the sample that is in the category
p_hat = x/n
se_p_hat = np.sqrt(p_hat*(1-p_hat)/n)
z = stats.norm.ppf(0.975, loc=0, scale=1)
print([2*z*se_p_hat])
```

▼ Python for small samples

```
n = # total number data in the sample
x = # number of data in the sample that is in the category
p_tilde = (x+2)/(n+4)
se_p_tilde = np.sqrt(p_tilde*(1-p_tilde)/(n+4))
print([z*se_p_tilde])
```

▼ Precision and sample size

For a given margin of error in a $(1-\alpha)$-confidence interval, the required sample size is:

$$n = p(1-p)\left(\frac{z_{1-\alpha/2}}{ME}\right)^2$$

Where p is a reasonable guess.

```
# Given values
p = # estimated population proportion
z = stats.norm.ppf(0.975,loc=0,scale=1)
ME = # margin of error

# Sample size calculation, when we can estimate
n = p * (1 - p) * (z / ME) ** 2

print(f"Required sample size: {math.ceil(n)}")
```

In the case, we do not have a reasonable guess of p the sample size can be calculated as:

$$n = \frac{1}{4}\left(\frac{z_{1-\alpha/2}}{ME}\right)^2$$

because the “worst case” is $p = \frac{1}{2}$

```
z = 1.96  # z-value for 95% confidence
ME = 0.01  # margin of error

# Sample size calculation, without the best guess or estimate
n = 1/4 * (z / ME) ** 2

print(f"Required sample size: {math.ceil(n)}")
```

▼ Hypothesis test for a single proportion

We consider a null hypothesis for a single proportion p and choose a significance level α:

$$H_0 : p = p_0$$
$$H_1 : p \neq p_0$$

▼ Test statistic

If the sample is large enough $(np_0 > 15 \wedge n(1-p0) > 15)$ we use the test statistic:

$$z_{obs} = \frac{x - np_0}{\sqrt{np_0(1-p_0)}} = \frac{\hat{p} - p_0}{\sqrt{p_0(1-p_0)/n}}$$

Under the null hypothesis, the test statistic approximately follows a standard normal distribution.

▼ p-value and conclusion

Find the p-value (evidence against the null hypothesis):

$$2P(Z > |z_{\text{obs}}|)$$

Testing using critical values:

We reject the null hypothesis if $z_{\text{obs}} < -z_{1-\alpha/2} \vee z_{\text{obs}} > z_{1-\alpha/2}$

```
p_hat =  #estimated proportion
p0 = 0.5 #null hypothesis generally 0.5
n =  #number of samples

z_obs = (p_hat-p0)/(np.sqrt(p0*(1-p0)/n))
p_value = 2*(1-stats.norm.cdf(abs(z_obs)))
critical_value = stats.norm.ppf(0.975)
z_obs, p_value, critical_value
```

▼ Confidence interval for two proportions

Confidence interval for the difference of two proportions

$$(\hat{p}_1 - \hat{p}_2) \pm z_{1-\alpha/2} \cdot \hat{\sigma}_{\hat{p}_1-\hat{p}_2}$$
$$\text{where } \hat{\sigma}_{\hat{p}_1-\hat{p}_2} = \sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}$$

Both $n_i\hat{p}_i \geq 10 \wedge n_i(1-\hat{p}_i) \geq 10$ must be true for $i = 1, 2$

```
q = #(1-a/2) quantile depends on confidence level
p1_hat =
p2_hat =
n1 =
n2 =
se_pdif = np.sqrt(p1*(1-p1)/n1 + p2*(1-p2)/n2)
z = stats.norm.ppf(q, loc=0, scale=1)
pdif = p1_hat - p2_hat
ci_lower = pdif - z*se_pdif
ci_upper = pdif + z*se_pdif
print([ci_lower, ci_upper])
```

▼ Hypothesis test for two proportions

When comparing two proportions (shown here for a two-sided alternative hypothesis):

$$H_0 : p_1 = p_2,$$
$$H_1 : p_1 \neq p_2.$$

Use the test statistic:

$$z_{\text{obs}} = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}(1-\hat{p})(\frac{1}{n_2} + \frac{1}{n_2})}}$$
$$\text{where } \hat{p} = \frac{x_1 + x_2}{n_1 + n_2}$$

```
n1 = #total number of data in 1
n2 = #total number of data in 2
x1 = #number of categorical data in 1
x2 = #number of categorical data in 2
diff = x1/n1 - x2/n2
p_pooled = (x1+x2)/(n1+n2)
z_obs = diff / np.sqrt(p_pooled*(1-p_pooled)*(1/n1 + 1/n2))
print("p_hat or p_pooled:", p_pooled)
print("Test statistic or z_obs:", z_obs)
print("p-value:", 2 * stats.norm.cdf(-z_obs, loc=0, scale=1))
```

```
n1 = #total number of data in 1
n2 = #total number of data in 2
x1 = #number of categorical data in 1
x2 = #number of categorical data in 2
p = #measured differece
p0 = #null hypothesis value

z_obs,p_value = smprop.proportions_ztest(count = [x1, x2], nobs = [n1, n2], value=p, prop_var=p0)
print(z_obs, p_value)
```

▼ Hypothesis test for multiple proportions

When comparing c proportions the null hypothesis is:

$$H_0 : p_1 = p_2 = ... = p_c = p$$

Against the alternative hypothesis that these proportions are not equal (i.e., at least one is different).

Table of observed counts for c samples:

	Sample 1	Sample 2	...	Sample c	Total
Success	x_1	x_2	...	x_c	x
Failure	$n_1 - x_1$	$n_2 - x_2$	...	$n_c - x_c$	$n - x$
Total	n_1	n_2	...	n_c	n

Common (average) estimate:

Under the null hypothesis, the estimate for p is:

$$\hat{p} = \frac{x}{n}$$

If the null hypothesis is true, we expect the jth group to have e_{1j} successes and e_{2j} failures, where:

$$e_{1j} = n_j \cdot \hat{p} = \frac{n_j \cdot x}{n}$$
$$e_{2j} = n_j(1 - \hat{p}) = \frac{n_j \cdot (n - x)}{n}$$

Table with the expected counts in the c samples:

e_{ij}	Sample 1	Sample 2	...	Sample c	Total
Success	e_{11}	e_{12}	...	e_{1c}	x
Failure	e_{21}	e_{22}	...	e_{2c}	$n - x$
Total	n_1	n_2	...	n_c	n

The general formula for calculating expected values in contingency tables is:

$$e_{ij} = \frac{(\text{Row total } i) \cdot (\text{Column total } j)}{\text{total}}$$

The test statistic is:

$$\chi^2_{\text{obs}} = \sum_{i=1}^{2} \sum_{j=1}^{c} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}$$

Where o_{ij} is the observed count in cell (i, j) and e_{ij} is the expected count in cell (i, j).

p-value is found by sampling a χ^2 distribution with $(c - 1)$ degrees of freedom (approximate)

With critical values, if $\chi^2 > \chi^2_{1-\alpha}(c - 1)$, we reject then null hypothesis.

Rule of thumb: All expected values $e_{ij} \geq 5$.

```
# The data in a table:
table_data = np.array([[#row 1
                                    ],[#row 2
                                    ]])

chi2, p_val, dof, (expected) = stats.chi2_contingency(table_data, correction=False)
print(expected) # expected frequencies under the null hypothesis
print("Chi-square test statistic:", chi2)
```

```
print("P-value:", p_val)
print(dof) #degrees of freedom
```

```
#critical value of chi2
#a is (1-confidence level)
#m is the degrees of freedom
stats.chi2.ppf(a, df=m)
```

▼ Analysis of a contingency table

In a contingency table with r rows and c columns, the test statistic is:

$$\chi^2_{\text{obs}} = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(o_{ij}-e_{ij})^2}{e_{ij}}$$

Where o_{ij} is the observed count in cell (i,j) and e_{ij} is the expected count in cell (i,j) (under the null hypothesis).

The contribution to the test statistic from a specific cell (a,b) is calculated as:

$$\frac{(o_{ab}-e_{ab})^2}{e_{ab}}$$

The number of contributions to the test statistic is the number of cells in the contingency table.

The general formula for calculating expected values in contingency tables is:

$$e_{ij} = \frac{(\text{Row total } i)\cdot(\text{Column total } j)}{\text{total}}$$

p-value is found by sampling a χ^2 distribution with $(c-1)$ degrees of freedom (approximate)

With critical values, if $\chi^2 > \chi^2_{1-\alpha}(c-1)(r-1)$, we reject then null hypothesis.

All expected values $e_{ij} \geq 5$.

```
# The data in a table:
table_data = np.array([[#row 1
                                    ],[#row 2
                                    ],[#row 3
                                    ]])

chi2, p_val, dof, expected = stats.chi2_contingency(table_data, correction=False)
print(expected) # expected frequencies under the null hypothesis
print("Chi-square test statistic:", chi2)
print("P-value:", p_val)
print(dof) #degrees of freedom
```

Analysis of Variance - ANOVA

▼ Definition

ANalysis Of VAriance (ANOVA) was introduced by R:A: Fisher about 100 years ago as a systematic way to analyze groups and has since been fundamental to the development of statistics.

▼ One classification criterion (one-way ANOVA)

The One-Way ANOVA helps compare (more than 2) means. The model is written as:

$$\begin{gathered}Y_{ij} = \mu + \alpha_i + \varepsilon_{ij} \\ \mu, \text{ is the overall mean} \\ \alpha_i, \text{ indicates the effect group (treatment) } i \\ Y_{ij}, \text{ is measurement } j \text{ in group } i \ (j \text{ ranges from 1 to } n_i)\end{gathered}$$

Where it is assumed that ε_{ij} are i.i.d. with $\varepsilon_{ij} \sim N(0,\sigma^2)$

To compare means $(\mu+\alpha_i)$ in the model, a hypothesis test is made where,

$$\begin{gathered}H_0 : \alpha_i = 0 \text{ for all } i \\ H_1 : \alpha_i \neq 0 \text{ for at least one } i\end{gathered}$$

▼ One-Way parameter estimates

$$\begin{gathered}Y_{ij} = \mu + \alpha_i + \varepsilon_{ij} \\ \hat{\mu} = \bar{y} \\ \hat{\alpha}_i = \bar{y}_i - \bar{y} \\ \hat{\sigma}^2 = MSE = \frac{SSE}{n-k}\end{gathered}$$

```
# Make pandas dataframe with grouped data:
data = pd.DataFrame({
    'value':  [],
    'group':  []})
# Compute the overall mean and add to dataframe:
data['overall_mean'] = data["value"].mean()

# compute the mean within each group and add to dataframe:
data['group_mean'] = data.groupby("group")['value'].transform('mean')

# compute the "alpha" for each group and add to dataframe:
data["alpha"] = data["group_mean"] - data["overall_mean"]
```

▼ Variance decomposition

With the model,

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

The total variation in data can be decomposed into,

$$SST = SS(Tr) + SSE$$

'One-Way' implies that there is only one factor in the experiment (with $k$levels).
The method is called analysis of variance because testing is done by comparing variances.

Total variation,

$$SST = \sum_{i=i}^{k} \sum_{j=i}^{n_i} (y_{ij} - \bar{y})^2$$

Variation within groups (Residual variation left after the model):

$$SSE = \sum_{i=i}^{k} \sum_{j=i}^{n_i} (y_{ij} - \bar{y}_i)^2$$

Variation between groups (Variation explained by the model):

$$SS(Tr) = \sum_{i=i}^{k} n_i (\bar{y}_i - \bar{y})^2$$

```
n = #total number of observation
k = #number of groups

# calculate the individual contribution to SSE and add to dataframe:
data['sse_contribution'] = (data['value']-data['group_mean'])**2

# calculate SSE and MSE:
SSE = data["sse_contribution"].sum()
MSE = SSE / (n-k)
print([SSE, MSE])
# calculate SST contribution of each datapoint:
data["sst_contribution"] = (data["value"] - data["overall_mean"])**2
SST = data["sst_contribution"].sum()
print([SSE, MSE, SST])
# We know SST = SSTr + SSE
# Calculate SSTr and MSTr:
SSTr = SST - SSE
MSTr = SSTr / (k-1)
print([SSTr, MSTr])
```

▼ ANOVA Table

Source of Variation	Treatment	Residual	Total
Degrees of freedoms	$k-1$	$n-k$	$n-1$
Sums of squares	$SS(Tr)$	SSE	SST
Mean sum of squares	$MS(Tr) = \frac{SS(Tr)}{k-1}$	$MSE = \frac{SSE}{n-k}$	
Test-statistic F	$F_{\text{obs}} = \frac{MS(Tr)}{MSE}$		
p-value	$P(F > F_{\text{obs}})$		

```
# Make the ANOVA table:
fit = smf.ols("value ~ group", data=data).fit()
anova_table = sm.stats.anova_lm(fit)
print(anova_table)
```

▼ Hypothesis testing (F-test)

We have Theorem 8.2 stating that:

$$SST = SS(Tr) + SSE$$

From this, the test statistic can be derived:

$$F = \frac{SS(Tr)/(k-1)}{SSE/(n-k)} = \frac{MS(Tr)}{MSE}$$

$$MS(Tr), \text{ is defined as the variation between groups}$$
$$MSE, \text{ Is defined as the variation within groups}$$
$$k, \text{ is the number of groups}$$
$$n, \text{ is the number of observations}$$

We choose a significance level α and compute the test statistic F.

Lastly, we compare the test statistic with the $(1-\alpha)$ quantile in the F distribution:

$$F \sim F(k-1, n-k)$$

```
# Corn yield data (bushels per acre)
x = #lists
y =
z =
```

```
# Combine data
data = [x, y, z]

# Perform one-way ANOVA
f_statistic, p_value = stats.f_oneway(x,y,z)

print("ANOVA Results:")
print(f"F-statistic: {f_statistic}")
print(f"P-value: {p_value}")
```

▼ Variability and relation with the t-test for two samples

The residual sum of squares, SSE, divided by $n-k$, also called residual mean square, $MSE = SSE/(n-k)$, is the average within-group variability:

$$MSE = \frac{SSE}{n-k} = \frac{(n_1-1)s_1^2 + ... + (n_k-1)s_k^2}{n-k}$$
$$s_i^2 = \frac{1}{n_i - 1}\sum_{i=1}^{n_1}(y_{ij} - \bar{y}_i)$$

ONLY when $k = 2$:

$$MSE = s_p^2 = \frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{n-2}$$
$$F_{\text{obs}} = t_{\text{obs}}^2$$
$$t_{\text{obs}}, \text{ is the pooled } t\text{-test statistic}$$

```
n = #total number of observation
k = #number of groups

# calculate the individual contribution to SSE and add to dataframe:
data['sse_contribution'] = (data['value']-data['group_mean'])**2

# calculate SSE and MSE:
SSE = data["sse_contribution"].sum()
MSE = SSE / (n-k)
print([SSE, MSE])
# calculate SST contribution of each datapoint:
data["sst_contribution"] = (data["value"] - data["overall_mean"])**2
SST = data["sst_contribution"].sum()
print([SSE, MSE, SST])
# We know SST = SSTr + SSE
# Calculate SSTr and MSTr:
SSTr = SST - SSE
MSTr = SSTr / (n-1)
print([SSTr, MSTr])

#Now we can calculate the test-statistic F = MSTr / MSE
Fobs = MSTr / MSE
print(Fobs)

# From Fobs we get a p-value:
pvalue = 1 - stats.f.cdf(Fobs, dfn = k-1, dfd = n-k)
print(pvalue)

# compare with critical value
print(stats.f.ppf(0.95, dfn = k-1, dfd = n-k))
```

▼ Post hoc comparisons

ANOVA tests the overall null hypothesis that all group means are equal $H_0 : \mu_1 = \mu_2 = \ldots = \mu_k$.

If ANOVA shows a significant result (i.e., p-value$< \alpha$), it only indicates that at least one group mean differs, but it doesn't specify which groups are different from each other.

Post-hoc comparisons are performed to pinpoint which specific group means are different.

▼ Post hoc confidence interval

A single planned comparison of the difference between treatment i and j is found by:

$$\bar{y}_i - \bar{y}_j \pm t_{1-\alpha/2}\sqrt{\frac{SSE}{n-k}\left(\frac{1}{n_i} + \frac{1}{n_j}\right)}$$

Where $t_{1-\alpha/2}$ is from the t-distribution with $n-k$ degrees of freedom.

Note the fewer degrees of freedom, since more parameters are estimated in calculating $MSE = SSE/(n-k) = s_p^2$ (the pooled variance estimate).

If all $M = k(k-1)/2$ combinations of pairwise confidence intervals are calculated, use the formula M times, each time with:

$$\alpha_{\text{Bonferroni}} = \frac{\alpha}{M}$$

▼ Post hoc pairwise hypothesis test

For a single planned hypothesis test

$$H_0 : \mu_i = \mu_j, H_1 : \mu_i \neq \mu_j, \; i \neq j$$

A t-test with $n-k$ degrees of freedom can be used with test statistic

$$t_{\text{obs}} = \frac{\bar{y}_i - \bar{y}_j}{\sqrt{\frac{SSE}{n-k}\left(\frac{1}{n_i} + \frac{1}{n_j}\right)}}$$

If all $M = k(k-1)/2$ combinations of pairwise tests are done, then the α level can be adjusted to control the type I error rate using the Bonferroni approach,

$$\alpha_{\text{Bonferroni}} = \frac{\alpha}{M}$$

▼ Model control (cheat code for residuals)

```
data["residual"] = fit.resid

# QQplot:
sm.qqplot(data["residual"], line='q',a=1/2)
plt.tight_layout()
plt.show()
```

▼ Two classification criteria (two-way ANOVA)

The Two-Way ANOVA helps compare (more than 2) means. The model is written as:

$$\begin{gathered} Y_{ij} = \mu + \alpha_i + \beta_j + \varepsilon_{ij} \\ \mu, \text{ is the overall mean} \\ \alpha_i, \text{ indicates the effect of treatment } i \in \{1, \cdots, k\} \\ \beta_j, \text{ indicates the effect of block } j \in \{1, \cdots, l\} \\ \text{There are } k \text{ treatments and } l \text{ blocks} \end{gathered}$$

Where it is assumed that ε_{ij} are i.i.d. with $\varepsilon_{ij} \sim N(0, \sigma^2)$

In this course, we only have one observation in each cell (i.e., with the same α and the same β) when performing two-way ANOVA.

The goal is to compare the treatment effects $(\text{means } \alpha_i)$ in the model, and a hypothesis test is made where,

$$\begin{gathered} H_{0,Tr} : \alpha_i = 0 \text{ for all } i \\ H_{1,Tr} : \alpha_i \neq 0 \text{ for at least one } i \end{gathered}$$

▼ Two-Way ANOVA data

The table of data can be really weird as such an example,

If we are given data like

We can write it in as,

```
data = pd.DataFrame({
    'value':  [100.391,101.506,102.241,103.058,103.766,104.801,100.495,101.455,102.468,103.350,104.230,105.391,100.617,101.623,102.750,103.617,104.411,105.346],
    'treatment': [1, 1, 1, 1, 1, 1,2,2,2,2,2,2,"R","R","R","R","R","R"],
    'block':  ["0","1","2","3","4","5","0","1","2","3","4","5","0","1","2","3","4","5"]})
```

▼ Two-Way parameter estimates

$$\begin{gathered} Y_{ij} = \mu + \alpha_i + \beta_j + \varepsilon_{ij} \\ \hat{\mu} = \frac{1}{k \cdot l} \sum_{i=1}^{k} \sum_{j=1}^{l} y_{ij} = \bar{y} \\ \hat{\alpha}_i = \left(\frac{1}{l} \sum_{j=1}^{l} y_{ij} \right) - \hat{\mu} = \bar{y}_i - \bar{y} \\ \hat{\beta}_j = \left(\frac{1}{k} \sum_{i=1}^{k} y_{ij} \right) - \hat{\mu} = \bar{y}_j - \bar{y} \end{gathered}$$

$\hat{\alpha}_i$ and $\hat{\beta}_j$ describe the estimates of the marginal effects of being in a specific treatment group or block.

```
# Make pandas dataframe with grouped data:
data = pd.DataFrame({
    'value':  [],
    'treatment':  [],
    'block':  []})
# Compute the overall mean and add to dataframe:
data['overall_mean'] = data["value"].mean()

# compute the mean within each treatment-group and add to dataframe:
data['treatment_mean'] = data.groupby("treatment")['value'].transform('mean')

# also add the corresponding alpha for each treatment-group:
data["treatment_alpha"] = data["treatment_mean"] - data["overall_mean"]

# compute the mean within each person (block) and add to dataframe:
data['block_mean'] = data.groupby("block")['value'].transform('mean')

# also add the corresponding beta for each person (block):
data["block_beta"] = data["block_mean"] - data["overall_mean"]
```

▼ Variance decomposition

With the model,

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

The total variation in the data can be decomposed as:

$$SST = SS(Tr) + SS(Bl) + SSE$$

‘Two-way’ refers to the presence of two factors in the experiment.

The method is called analysis of variance because the testing involves comparing variances.

Total sum of squares, total variation (same as for one-way analysis),

$$SST = \sum_{i=1}^{k}\sum_{j=1}^{l}(y_{ij} - \hat{\mu})^2$$

Treatment sum of squares, variation between treatment/groups (variation explained by treatments),

$$SS(Tr) = l \cdot \sum_{i=1}^{k}(\bar{y}_i - \hat{\mu})^2 = l \cdot \sum_{i=1}^{k}\hat{\alpha}_i^2$$

Block sum of squares, variation between blocks/persons (variation explained by blocks),

$$SS(Bl) = k \cdot \sum_{j=1}^{l}(\bar{y}_j - \hat{\mu})^2 = k \cdot \sum_{j=1}^{l}\hat{\beta}_j^2$$

Sum squared errors, variation of residuals (variation not explained by the model),

$$SSE = \sum_{i=1}^{k}\sum_{j=1}^{l}(y_{ij} - \hat{y}_{ij})^2 = \sum_{i=1}^{k}\sum_{j=1}^{l}(y_{ij} - \hat{\alpha}_i - \hat{\beta}_j - \hat{\mu})^2$$

```
data['residuals'] = (data['value']-(data['overall_mean']+data["treatment_alpha"]+data["block_beta"]))**2
data['sse_contribution'] = data['residual']**2

# calculate SSE and MSE:
SSE = data['sse_contribution'].sum()
print(SSE)

# calculate SST contribution of each datapoint:
data["sst_contribution"] = (data["value"] - data["overall_mean"])**2
SST = data["sst_contribution"].sum()
print(SST)
# Calculate SSTr and SSBl:
data['sstr_contribution'] = (data['treatment_mean'] - data['overall_mean'])**2
SSTr = data['sstr_contribution'].sum()

data['ssbl_contribution'] = (data['block_mean'] - data['overall_mean'])**2
SSBl = data['ssbl_contribution'].sum()
print([SSTr, SSBl])

# sanity check:
print(SST)
print(SSTr + SSBl + SSE)
```

▼ ANOVA Table

Source of Variation	Treatment	Block	Residual	Total
Degrees of freedoms	$k-1$	$l-1$	$(k-1)(l-1)$	$n-1$
Sums of squares	$SS(Tr)$	$SS(Bl)$	SSE	SST
Mean sum of squares	$MS(Tr) = \frac{SS(Tr)}{k-1}$	$MS(Bl) = \frac{SS(Bl)}{l-1}$	$MSE = \frac{SSE}{(k-1)(l-1)}$	
Test-statistic F	$F_{\text{Tr}} = \frac{MS(Tr)}{MSE}$	$F_{Bl} = \frac{MS(Bl)}{MSE}$		
p-value	$P(F > F_{\text{Tr}})$	$P(F > F_{Bl})$		

```
# Make the ANOVA table:
fit = smf.ols("value ~ treatment + block", data=data).fit()
anova_table = sm.stats.anova_lm(fit)
print(anova_table)
```

▼ Hypothesis testing (F-test)

▼ Hypothesis about different treatment effects Theorem 8.22

The goal is to compare the treatment effects $(\text{means } \alpha_i)$ in the model,

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

The null hypothesis test of no difference/effect among treatments can be formulated as,

$$H_{0,Tr} : \alpha_i = 0 \text{ for all } i$$
$$H_{1,Tr} : \alpha_i \neq 0 \text{ for at least one } i$$

Under $H_{0,Tr}$, the test statistic,

$$F_{Tr} = \frac{SS(Tr)/(k-1)}{SSE/((k-1)(l-1))}$$

Is F-distributed with $k-1$ and $(k-1)(l-1)$ degrees of freedom.

```
k =
l =
# Compute F-statistic (for treatment groups)
FTr = (SSTr/(k-1)) / (SSE/((k-1)*(l-1)))
print(FTr)

# compute critical value:
stats.f.ppf(0.95, dfn = (k-1), dfd = (k-1)*(l-1) )

# p-value
```

```
p_Tr = 1 - stats.f.cdf(FTr, dfn = (k-1), dfd = (k-1)*(l-1) )
print(p_Tr)
```

▼ Hypothesis about different block effects Theorem 8.22

The goal is to compare the block effects $(\text{means } \beta_j)$ in the model,

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

The null hypothesis test of no difference/effect among blocks can be formulated as,

$$H_{0,Tr} : \beta_j = 0 \text{ for all } j$$
$$H_{1,Tr} : \beta_j \neq 0 \text{ for at least one } j$$

Under $H_{0,Bl}$, the test statistic,

$$F_{Bl} = \frac{SS(Bl)/(l-1)}{SSE/((k-1)(l-1))}$$

Is F-distributed with $l-1$ and $(k-1)(l-1)$ degrees of freedom.

```
k =
l =
# Compute F-statistic (for treatment groups)
FBl = (SSBl/(l-1)) / (SSE/((k-1)*(l-1)))
print(FBl)

# compute critical value:
stats.f.ppf(0.95, dfn = (l-1), dfd = (k-1)*(l-1) )

# p-value
p_Bl = 1 - stats.f.cdf(FBl, dfn = (l-1), dfd = (k-1)*(l-1) )
print(p_Bl)
```

▼ p-value

A high p-value means they have no significant value on the result

▼ Model control (cheat code for residuals)

```
data["residual"] = fit.resid

# QQplot:
sm.qqplot(data["residual"], line='q',a=1/2)
plt.tight_layout()
plt.show()
```

▼ Post hoc Confidence Intervals

Can be done for either treatments or blocks

A single pre-planned comparison of the difference between treatment $a$and b is given by,

$$\bar{y}_a - \bar{y}_b \pm t_{1-\alpha/2}\sqrt{\frac{SSE}{(k-1)(l-1)}\left(\frac{1}{n_a}+\frac{1}{n_b}\right)}$$

$$\bar{y}_a - \bar{y}_b \pm t_{1-\alpha/2}\sqrt{MSE\left(\frac{1}{n_a}+\frac{1}{n_b}\right)}$$

Where $t_{1-\alpha/2}$ is from the t-distribution with $(k-1)(l-1)$ degrees of freedom.

If M combinations of pairwise confidence intervals are calculated, use the formula M times, but each time with $\alpha_{\text{Bonferroni}} = \alpha/M$.

▼ Post hoc pairwise hypothesis test

For a single pre-planned hypothesis test,

$$H_0 : \alpha_a = \alpha_b,\ H_1 : \alpha_a \neq \alpha_b$$

Compute the test statistic as,

$$t_{obs} = \frac{\bar{y}_a - \bar{y}_b}{\sqrt{MSE\left(\frac{1}{n_a}+\frac{1}{n_b}\right)}}$$

And the p-value as,

$$p = 2P(T > |t_{obs}|)$$

Where T follows a t-distribution with $(k-1)(l-1)$ degrees of freedom.

If M combinations of pairwise confidence intervals are calculated, use the adjusted significance level: $\alpha_{\text{Bonferroni}} = \alpha/M$

▼ Missing or additional observations

Our two-way ANOVA is very “neat” in our course since we have exactly one observation per row and column. But in practice:

- Observations are often missing in a group
- There are more than one observation in some cases
- The model can be easily adjusted