NumPy Tutorial:

Indexing on “ndarrays”:

# ‘ndarrays’ can be indexed using the standard Python x[obj] syntax, where x is the array and obj the selection.

# Note that in Python, x[(exp1, exp2, ..., expN)] is equivalent to x[exp1, exp2, ..., expN]; the latter is just syntactic sugar for the former.

# Note that if one indexes a multidimensional array with fewer indices than dimensions, one gets a subdimensional array.

# So note that x[0, 2] == x[0][2] though the second case is more inefficient as a new temporary array is created after the first index that is subsequently indexed by 2.

# NumPy uses C-order indexing. That means that the last index usually represents the most rapidly changing memory location, unlike Fortran or IDL, where the first index represents the most rapidly changing location in memory. This difference represents a great potential for confusion.

# Ellipsis ( . . . ) expands to the number of : objects needed for the selection tuple to index all dimensions.

# Each ‘newaxis’ (None) object in the selection tuple serves to expand the dimensions of the resulting selection by one unit-length dimension.

# Advanced indexing always returns a copy of the data (contrast with basic slicing that returns a view).

# Advanced indexing is triggered when the selection object, obj, is a non-tuple sequence object, an ndarray (of data type integer or bool), or a tuple with at least one sequence object or ndarray (of data type integer or bool). There are two types of advanced indexing: integer and Boolean.

## Array creation routines:

## From shape or value

|  |  |
| --- | --- |
| [**empty**](https://numpy.org/doc/stable/reference/generated/numpy.empty.html#numpy.empty)(shape[, dtype, order, like]) | Return a new array of given shape and type, without initializing entries. |
| [**empty\_like**](https://numpy.org/doc/stable/reference/generated/numpy.empty_like.html#numpy.empty_like)(prototype[, dtype, order, subok, ...]) | Return a new array with the same shape and type as a given array. |
| [**eye**](https://numpy.org/doc/stable/reference/generated/numpy.eye.html#numpy.eye)(N[, M, k, dtype, order, like]) | Return a 2-D array with ones on the diagonal and zeros elsewhere. |
| [**identity**](https://numpy.org/doc/stable/reference/generated/numpy.identity.html#numpy.identity)(n[, dtype, like]) | Return the identity array. |
| [**ones**](https://numpy.org/doc/stable/reference/generated/numpy.ones.html#numpy.ones)(shape[, dtype, order, like]) | Return a new array of given shape and type, filled with ones. |
| [**ones\_like**](https://numpy.org/doc/stable/reference/generated/numpy.ones_like.html#numpy.ones_like)(a[, dtype, order, subok, shape]) | Return an array of ones with the same shape and type as a given array. |
| [**zeros**](https://numpy.org/doc/stable/reference/generated/numpy.zeros.html#numpy.zeros)(shape[, dtype, order, like]) | Return a new array of given shape and type, filled with zeros. |
| [**zeros\_like**](https://numpy.org/doc/stable/reference/generated/numpy.zeros_like.html#numpy.zeros_like)(a[, dtype, order, subok, shape]) | Return an array of zeros with the same shape and type as a given array. |
| [**full**](https://numpy.org/doc/stable/reference/generated/numpy.full.html#numpy.full)(shape, fill\_value[, dtype, order, like]) | Return a new array of given shape and type, filled with fill\_value. |
| [**full\_like**](https://numpy.org/doc/stable/reference/generated/numpy.full_like.html#numpy.full_like)(a, fill\_value[, dtype, order, ...]) | Return a full array with the same shape and type as a given array. |

## From existing data

|  |  |
| --- | --- |
| [**array**](https://numpy.org/doc/stable/reference/generated/numpy.array.html#numpy.array)(object[, dtype, copy, order, subok, ...]) | Create an array. |
| [**asarray**](https://numpy.org/doc/stable/reference/generated/numpy.asarray.html#numpy.asarray)(a[, dtype, order, like]) | Convert the input to an array. |
| [**asanyarray**](https://numpy.org/doc/stable/reference/generated/numpy.asanyarray.html#numpy.asanyarray)(a[, dtype, order, like]) | Convert the input to an ndarray, but pass ndarray subclasses through. |
| [**ascontiguousarray**](https://numpy.org/doc/stable/reference/generated/numpy.ascontiguousarray.html#numpy.ascontiguousarray)(a[, dtype, like]) | Return a contiguous array (ndim >= 1) in memory (C order). |
| [**asmatrix**](https://numpy.org/doc/stable/reference/generated/numpy.asmatrix.html#numpy.asmatrix)(data[, dtype]) | Interpret the input as a matrix. |
| [**copy**](https://numpy.org/doc/stable/reference/generated/numpy.copy.html#numpy.copy)(a[, order, subok]) | Return an array copy of the given object. |
| [**frombuffer**](https://numpy.org/doc/stable/reference/generated/numpy.frombuffer.html#numpy.frombuffer)(buffer[, dtype, count, offset, like]) | Interpret a buffer as a 1-dimensional array. |
| [**from\_dlpack**](https://numpy.org/doc/stable/reference/generated/numpy.from_dlpack.html#numpy.from_dlpack)(x, /) | Create a NumPy array from an object implementing the \_\_dlpack\_\_ protocol. |
| [**fromfile**](https://numpy.org/doc/stable/reference/generated/numpy.fromfile.html#numpy.fromfile)(file[, dtype, count, sep, offset, like]) | Construct an array from data in a text or binary file. |
| [**fromfunction**](https://numpy.org/doc/stable/reference/generated/numpy.fromfunction.html#numpy.fromfunction)(function, shape, \*[, dtype, like]) | Construct an array by executing a function over each coordinate. |
| [**fromiter**](https://numpy.org/doc/stable/reference/generated/numpy.fromiter.html#numpy.fromiter)(iter, dtype[, count, like]) | Create a new 1-dimensional array from an iterable object. |
| [**fromstring**](https://numpy.org/doc/stable/reference/generated/numpy.fromstring.html#numpy.fromstring)(string[, dtype, count, like]) | A new 1-D array initialized from text data in a string. |
| [**loadtxt**](https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html#numpy.loadtxt)(fname[, dtype, comments, delimiter, ...]) | Load data from a text file. |

## Creating record arrays (numpy.rec)

**Note: numpy.rec** is the preferred alias for **numpy.core.records**.

|  |  |
| --- | --- |
| [**core.records.array**](https://numpy.org/doc/stable/reference/generated/numpy.core.records.array.html#numpy.core.records.array)(obj[, dtype, shape, ...]) | Construct a record array from a wide-variety of objects. |
| [**core.records.fromarrays**](https://numpy.org/doc/stable/reference/generated/numpy.core.records.fromarrays.html#numpy.core.records.fromarrays)(arrayList[, dtype, ...]) | Create a record array from a (flat) list of arrays |
| [**core.records.fromrecords**](https://numpy.org/doc/stable/reference/generated/numpy.core.records.fromrecords.html#numpy.core.records.fromrecords)(recList[, dtype, ...]) | Create a recarray from a list of records in text form. |
| [**core.records.fromstring**](https://numpy.org/doc/stable/reference/generated/numpy.core.records.fromstring.html#numpy.core.records.fromstring)(datastring[, dtype, ...]) | Create a record array from binary data |
| [**core.records.fromfile**](https://numpy.org/doc/stable/reference/generated/numpy.core.records.fromfile.html#numpy.core.records.fromfile)(fd[, dtype, shape, ...]) | Create an array from binary file data |

## Creating character arrays ([numpy.char](https://numpy.org/doc/stable/reference/routines.char.html" \l "module-numpy.char" \o "numpy.char))

**Note:** [**numpy.char**](https://numpy.org/doc/stable/reference/routines.char.html#module-numpy.char) is the preferred alias for **numpy.core.defchararray**.

|  |  |
| --- | --- |
| [**core.defchararray.array**](https://numpy.org/doc/stable/reference/generated/numpy.core.defchararray.array.html#numpy.core.defchararray.array)(obj[, itemsize, ...]) | Create a **[chararray](https://numpy.org/doc/stable/reference/generated/numpy.char.chararray.html" \l "numpy.char.chararray" \o "numpy.chararray)**. |
| [**core.defchararray.asarray**](https://numpy.org/doc/stable/reference/generated/numpy.core.defchararray.asarray.html#numpy.core.defchararray.asarray)(obj[, itemsize, ...]) | Convert the input to a **[chararray](https://numpy.org/doc/stable/reference/generated/numpy.char.chararray.html" \l "numpy.char.chararray" \o "numpy.chararray)**, copying the data only if necessary. |

## Numerical ranges

|  |  |
| --- | --- |
| [**arange**](https://numpy.org/doc/stable/reference/generated/numpy.arange.html#numpy.arange)([start,] stop[, step,][, dtype, like]) | Return evenly spaced values within a given interval. |
| [**linspace**](https://numpy.org/doc/stable/reference/generated/numpy.linspace.html#numpy.linspace)(start, stop[, num, endpoint, ...]) | Return evenly spaced numbers over a specified interval. |
| [**logspace**](https://numpy.org/doc/stable/reference/generated/numpy.logspace.html#numpy.logspace)(start, stop[, num, endpoint, base, ...]) | Return numbers spaced evenly on a log scale. |
| [**geomspace**](https://numpy.org/doc/stable/reference/generated/numpy.geomspace.html#numpy.geomspace)(start, stop[, num, endpoint, ...]) | Return numbers spaced evenly on a log scale (a geometric progression). |
| [**meshgrid**](https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html#numpy.meshgrid)(\*xi[, copy, sparse, indexing]) | Return a list of coordinate matrices from coordinate vectors. |
| [**mgrid**](https://numpy.org/doc/stable/reference/generated/numpy.mgrid.html#numpy.mgrid) | An instance which returns a dense multi-dimensional "meshgrid". |
| [**ogrid**](https://numpy.org/doc/stable/reference/generated/numpy.ogrid.html#numpy.ogrid) | An instance which returns an open multi-dimensional "meshgrid". |

## Building matrices

|  |  |
| --- | --- |
| [**diag**](https://numpy.org/doc/stable/reference/generated/numpy.diag.html#numpy.diag)(v[, k]) | Extract a diagonal or construct a diagonal array. |
| [**diagflat**](https://numpy.org/doc/stable/reference/generated/numpy.diagflat.html#numpy.diagflat)(v[, k]) | Create a two-dimensional array with the flattened input as a diagonal. |
| [**tri**](https://numpy.org/doc/stable/reference/generated/numpy.tri.html#numpy.tri)(N[, M, k, dtype, like]) | An array with ones at and below the given diagonal and zeros elsewhere. |
| [**tril**](https://numpy.org/doc/stable/reference/generated/numpy.tril.html#numpy.tril)(m[, k]) | Lower triangle of an array. |
| [**triu**](https://numpy.org/doc/stable/reference/generated/numpy.triu.html#numpy.triu)(m[, k]) | Upper triangle of an array. |
| [**vander**](https://numpy.org/doc/stable/reference/generated/numpy.vander.html#numpy.vander)(x[, N, increasing]) | Generate a Vandermonde matrix. |

## The Matrix class

|  |  |
| --- | --- |
| [**mat**](https://numpy.org/doc/stable/reference/generated/numpy.mat.html#numpy.mat)(data[, dtype]) | Interpret the input as a matrix. |
| [**bmat**](https://numpy.org/doc/stable/reference/generated/numpy.bmat.html#numpy.bmat)(obj[, ldict, gdict]) | Build a matrix object from a string, nested sequence, or array. |

## Mathematical Functions:

## Trigonometric functions

|  |  |
| --- | --- |
| [**sin**](https://numpy.org/doc/stable/reference/generated/numpy.sin.html#numpy.sin)(x, /[, out, where, casting, order, ...]) | Trigonometric sine, element-wise. |
| [**cos**](https://numpy.org/doc/stable/reference/generated/numpy.cos.html#numpy.cos)(x, /[, out, where, casting, order, ...]) | Cosine element-wise. |
| [**tan**](https://numpy.org/doc/stable/reference/generated/numpy.tan.html#numpy.tan)(x, /[, out, where, casting, order, ...]) | Compute tangent element-wise. |
| [**arcsin**](https://numpy.org/doc/stable/reference/generated/numpy.arcsin.html#numpy.arcsin)(x, /[, out, where, casting, order, ...]) | Inverse sine, element-wise. |
| [**arccos**](https://numpy.org/doc/stable/reference/generated/numpy.arccos.html#numpy.arccos)(x, /[, out, where, casting, order, ...]) | Trigonometric inverse cosine, element-wise. |
| [**arctan**](https://numpy.org/doc/stable/reference/generated/numpy.arctan.html#numpy.arctan)(x, /[, out, where, casting, order, ...]) | Trigonometric inverse tangent, element-wise. |
| [**hypot**](https://numpy.org/doc/stable/reference/generated/numpy.hypot.html#numpy.hypot)(x1, x2, /[, out, where, casting, ...]) | Given the "legs" of a right triangle, return its hypotenuse. |
| [**arctan2**](https://numpy.org/doc/stable/reference/generated/numpy.arctan2.html#numpy.arctan2)(x1, x2, /[, out, where, casting, ...]) | Element-wise arc tangent of x1/x2 choosing the quadrant correctly. |
| [**degrees**](https://numpy.org/doc/stable/reference/generated/numpy.degrees.html#numpy.degrees)(x, /[, out, where, casting, order, ...]) | Convert angles from radians to degrees. |
| [**radians**](https://numpy.org/doc/stable/reference/generated/numpy.radians.html#numpy.radians)(x, /[, out, where, casting, order, ...]) | Convert angles from degrees to radians. |
| [**unwrap**](https://numpy.org/doc/stable/reference/generated/numpy.unwrap.html#numpy.unwrap)(p[, discont, axis, period]) | Unwrap by taking the complement of large deltas with respect to the period. |
| [**deg2rad**](https://numpy.org/doc/stable/reference/generated/numpy.deg2rad.html#numpy.deg2rad)(x, /[, out, where, casting, order, ...]) | Convert angles from degrees to radians. |
| [**rad2deg**](https://numpy.org/doc/stable/reference/generated/numpy.rad2deg.html#numpy.rad2deg)(x, /[, out, where, casting, order, ...]) | Convert angles from radians to degrees. |

## Hyperbolic functions

|  |  |
| --- | --- |
| [**sinh**](https://numpy.org/doc/stable/reference/generated/numpy.sinh.html#numpy.sinh)(x, /[, out, where, casting, order, ...]) | Hyperbolic sine, element-wise. |
| [**cosh**](https://numpy.org/doc/stable/reference/generated/numpy.cosh.html#numpy.cosh)(x, /[, out, where, casting, order, ...]) | Hyperbolic cosine, element-wise. |
| [**tanh**](https://numpy.org/doc/stable/reference/generated/numpy.tanh.html#numpy.tanh)(x, /[, out, where, casting, order, ...]) | Compute hyperbolic tangent element-wise. |
| [**arcsinh**](https://numpy.org/doc/stable/reference/generated/numpy.arcsinh.html#numpy.arcsinh)(x, /[, out, where, casting, order, ...]) | Inverse hyperbolic sine element-wise. |
| [**arccosh**](https://numpy.org/doc/stable/reference/generated/numpy.arccosh.html#numpy.arccosh)(x, /[, out, where, casting, order, ...]) | Inverse hyperbolic cosine, element-wise. |
| [**arctanh**](https://numpy.org/doc/stable/reference/generated/numpy.arctanh.html#numpy.arctanh)(x, /[, out, where, casting, order, ...]) | Inverse hyperbolic tangent element-wise. |

## Rounding

|  |  |
| --- | --- |
| [**round**](https://numpy.org/doc/stable/reference/generated/numpy.round.html#numpy.round)(a[, decimals, out]) | Evenly round to the given number of decimals. |
| [**around**](https://numpy.org/doc/stable/reference/generated/numpy.around.html#numpy.around)(a[, decimals, out]) | Round an array to the given number of decimals. |
| [**rint**](https://numpy.org/doc/stable/reference/generated/numpy.rint.html#numpy.rint)(x, /[, out, where, casting, order, ...]) | Round elements of the array to the nearest integer. |
| [**fix**](https://numpy.org/doc/stable/reference/generated/numpy.fix.html#numpy.fix)(x[, out]) | Round to nearest integer towards zero. |
| [**floor**](https://numpy.org/doc/stable/reference/generated/numpy.floor.html#numpy.floor)(x, /[, out, where, casting, order, ...]) | Return the floor of the input, element-wise. |
| [**ceil**](https://numpy.org/doc/stable/reference/generated/numpy.ceil.html#numpy.ceil)(x, /[, out, where, casting, order, ...]) | Return the ceiling of the input, element-wise. |
| [**trunc**](https://numpy.org/doc/stable/reference/generated/numpy.trunc.html#numpy.trunc)(x, /[, out, where, casting, order, ...]) | Return the truncated value of the input, element-wise. |

## Sums, products, differences

|  |  |
| --- | --- |
| [**prod**](https://numpy.org/doc/stable/reference/generated/numpy.prod.html#numpy.prod)(a[, axis, dtype, out, keepdims, ...]) | Return the product of array elements over a given axis. |
| [**sum**](https://numpy.org/doc/stable/reference/generated/numpy.sum.html#numpy.sum)(a[, axis, dtype, out, keepdims, ...]) | Sum of array elements over a given axis. |
| [**nanprod**](https://numpy.org/doc/stable/reference/generated/numpy.nanprod.html#numpy.nanprod)(a[, axis, dtype, out, keepdims, ...]) | Return the product of array elements over a given axis treating Not a Numbers (NaNs) as ones. |
| [**nansum**](https://numpy.org/doc/stable/reference/generated/numpy.nansum.html#numpy.nansum)(a[, axis, dtype, out, keepdims, ...]) | Return the sum of array elements over a given axis treating Not a Numbers (NaNs) as zero. |
| [**cumprod**](https://numpy.org/doc/stable/reference/generated/numpy.cumprod.html#numpy.cumprod)(a[, axis, dtype, out]) | Return the cumulative product of elements along a given axis. |
| [**cumsum**](https://numpy.org/doc/stable/reference/generated/numpy.cumsum.html#numpy.cumsum)(a[, axis, dtype, out]) | Return the cumulative sum of the elements along a given axis. |
| [**nancumprod**](https://numpy.org/doc/stable/reference/generated/numpy.nancumprod.html#numpy.nancumprod)(a[, axis, dtype, out]) | Return the cumulative product of array elements over a given axis treating Not a Numbers (NaNs) as one. |
| [**nancumsum**](https://numpy.org/doc/stable/reference/generated/numpy.nancumsum.html#numpy.nancumsum)(a[, axis, dtype, out]) | Return the cumulative sum of array elements over a given axis treating Not a Numbers (NaNs) as zero. |
| [**diff**](https://numpy.org/doc/stable/reference/generated/numpy.diff.html#numpy.diff)(a[, n, axis, prepend, append]) | Calculate the n-th discrete difference along the given axis. |
| [**ediff1d**](https://numpy.org/doc/stable/reference/generated/numpy.ediff1d.html#numpy.ediff1d)(ary[, to\_end, to\_begin]) | The differences between consecutive elements of an array. |
| [**gradient**](https://numpy.org/doc/stable/reference/generated/numpy.gradient.html#numpy.gradient)(f, \*varargs[, axis, edge\_order]) | Return the gradient of an N-dimensional array. |
| [**cross**](https://numpy.org/doc/stable/reference/generated/numpy.cross.html#numpy.cross)(a, b[, axisa, axisb, axisc, axis]) | Return the cross product of two (arrays of) vectors. |
| [**trapz**](https://numpy.org/doc/stable/reference/generated/numpy.trapz.html#numpy.trapz)(y[, x, dx, axis]) | Integrate along the given axis using the composite trapezoidal rule. |

## Exponents and logarithms

|  |  |
| --- | --- |
| [**exp**](https://numpy.org/doc/stable/reference/generated/numpy.exp.html#numpy.exp)(x, /[, out, where, casting, order, ...]) | Calculate the exponential of all elements in the input array. |
| [**expm1**](https://numpy.org/doc/stable/reference/generated/numpy.expm1.html#numpy.expm1)(x, /[, out, where, casting, order, ...]) | Calculate exp(x) - 1 for all elements in the array. |
| [**exp2**](https://numpy.org/doc/stable/reference/generated/numpy.exp2.html#numpy.exp2)(x, /[, out, where, casting, order, ...]) | Calculate 2\*\*p for all p in the input array. |
| [**log**](https://numpy.org/doc/stable/reference/generated/numpy.log.html#numpy.log)(x, /[, out, where, casting, order, ...]) | Natural logarithm, element-wise. |
| [**log10**](https://numpy.org/doc/stable/reference/generated/numpy.log10.html#numpy.log10)(x, /[, out, where, casting, order, ...]) | Return the base 10 logarithm of the input array, element-wise. |
| [**log2**](https://numpy.org/doc/stable/reference/generated/numpy.log2.html#numpy.log2)(x, /[, out, where, casting, order, ...]) | Base-2 logarithm of x. |
| [**log1p**](https://numpy.org/doc/stable/reference/generated/numpy.log1p.html#numpy.log1p)(x, /[, out, where, casting, order, ...]) | Return the natural logarithm of one plus the input array, element-wise. |
| [**logaddexp**](https://numpy.org/doc/stable/reference/generated/numpy.logaddexp.html#numpy.logaddexp)(x1, x2, /[, out, where, casting, ...]) | Logarithm of the sum of exponentiations of the inputs. |
| [**logaddexp2**](https://numpy.org/doc/stable/reference/generated/numpy.logaddexp2.html#numpy.logaddexp2)(x1, x2, /[, out, where, casting, ...]) | Logarithm of the sum of exponentiations of the inputs in base-2. |

## Other special functions

|  |  |
| --- | --- |
| [**i0**](https://numpy.org/doc/stable/reference/generated/numpy.i0.html#numpy.i0)(x) | Modified Bessel function of the first kind, order 0. |
| [**sinc**](https://numpy.org/doc/stable/reference/generated/numpy.sinc.html#numpy.sinc)(x) | Return the normalized sinc function. |

## Floating point routines

|  |  |
| --- | --- |
| [**signbit**](https://numpy.org/doc/stable/reference/generated/numpy.signbit.html#numpy.signbit)(x, /[, out, where, casting, order, ...]) | Returns element-wise True where signbit is set (less than zero). |
| [**copysign**](https://numpy.org/doc/stable/reference/generated/numpy.copysign.html#numpy.copysign)(x1, x2, /[, out, where, casting, ...]) | Change the sign of x1 to that of x2, element-wise. |
| [**frexp**](https://numpy.org/doc/stable/reference/generated/numpy.frexp.html#numpy.frexp)(x[, out1, out2], / [[, out, where, ...]) | Decompose the elements of x into mantissa and twos exponent. |
| [**ldexp**](https://numpy.org/doc/stable/reference/generated/numpy.ldexp.html#numpy.ldexp)(x1, x2, /[, out, where, casting, ...]) | Returns x1 \* 2\*\*x2, element-wise. |
| [**nextafter**](https://numpy.org/doc/stable/reference/generated/numpy.nextafter.html#numpy.nextafter)(x1, x2, /[, out, where, casting, ...]) | Return the next floating-point value after x1 towards x2, element-wise. |
| [**spacing**](https://numpy.org/doc/stable/reference/generated/numpy.spacing.html#numpy.spacing)(x, /[, out, where, casting, order, ...]) | Return the distance between x and the nearest adjacent number. |

## Rational routines

|  |  |
| --- | --- |
| [**lcm**](https://numpy.org/doc/stable/reference/generated/numpy.lcm.html#numpy.lcm)(x1, x2, /[, out, where, casting, order, ...]) | Returns the lowest common multiple of |x1| and |x2| |
| [**gcd**](https://numpy.org/doc/stable/reference/generated/numpy.gcd.html#numpy.gcd)(x1, x2, /[, out, where, casting, order, ...]) | Returns the greatest common divisor of |x1| and |x2| |

## Arithmetic operations

|  |  |
| --- | --- |
| [**add**](https://numpy.org/doc/stable/reference/generated/numpy.add.html#numpy.add)(x1, x2, /[, out, where, casting, order, ...]) | Add arguments element-wise. |
| [**reciprocal**](https://numpy.org/doc/stable/reference/generated/numpy.reciprocal.html#numpy.reciprocal)(x, /[, out, where, casting, ...]) | Return the reciprocal of the argument, element-wise. |
| [**positive**](https://numpy.org/doc/stable/reference/generated/numpy.positive.html#numpy.positive)(x, /[, out, where, casting, order, ...]) | Numerical positive, element-wise. |
| [**negative**](https://numpy.org/doc/stable/reference/generated/numpy.negative.html#numpy.negative)(x, /[, out, where, casting, order, ...]) | Numerical negative, element-wise. |
| [**multiply**](https://numpy.org/doc/stable/reference/generated/numpy.multiply.html#numpy.multiply)(x1, x2, /[, out, where, casting, ...]) | Multiply arguments element-wise. |
| [**divide**](https://numpy.org/doc/stable/reference/generated/numpy.divide.html#numpy.divide)(x1, x2, /[, out, where, casting, ...]) | Divide arguments element-wise. |
| [**power**](https://numpy.org/doc/stable/reference/generated/numpy.power.html#numpy.power)(x1, x2, /[, out, where, casting, ...]) | First array elements raised to powers from second array, element-wise. |
| [**subtract**](https://numpy.org/doc/stable/reference/generated/numpy.subtract.html#numpy.subtract)(x1, x2, /[, out, where, casting, ...]) | Subtract arguments, element-wise. |
| [**true\_divide**](https://numpy.org/doc/stable/reference/generated/numpy.true_divide.html#numpy.true_divide)(x1, x2, /[, out, where, ...]) | Divide arguments element-wise. |
| [**floor\_divide**](https://numpy.org/doc/stable/reference/generated/numpy.floor_divide.html#numpy.floor_divide)(x1, x2, /[, out, where, ...]) | Return the largest integer smaller or equal to the division of the inputs. |
| [**float\_power**](https://numpy.org/doc/stable/reference/generated/numpy.float_power.html#numpy.float_power)(x1, x2, /[, out, where, ...]) | First array elements raised to powers from second array, element-wise. |
| [**fmod**](https://numpy.org/doc/stable/reference/generated/numpy.fmod.html#numpy.fmod)(x1, x2, /[, out, where, casting, ...]) | Returns the element-wise remainder of division. |
| [**mod**](https://numpy.org/doc/stable/reference/generated/numpy.mod.html#numpy.mod)(x1, x2, /[, out, where, casting, order, ...]) | Returns the element-wise remainder of division. |
| [**modf**](https://numpy.org/doc/stable/reference/generated/numpy.modf.html#numpy.modf)(x[, out1, out2], / [[, out, where, ...]) | Return the fractional and integral parts of an array, element-wise. |
| [**remainder**](https://numpy.org/doc/stable/reference/generated/numpy.remainder.html#numpy.remainder)(x1, x2, /[, out, where, casting, ...]) | Returns the element-wise remainder of division. |
| [**divmod**](https://numpy.org/doc/stable/reference/generated/numpy.divmod.html#numpy.divmod)(x1, x2[, out1, out2], / [[, out, ...]) | Return element-wise quotient and remainder simultaneously. |

## Handling complex numbers

|  |  |
| --- | --- |
| [**angle**](https://numpy.org/doc/stable/reference/generated/numpy.angle.html#numpy.angle)(z[, deg]) | Return the angle of the complex argument. |
| [**real**](https://numpy.org/doc/stable/reference/generated/numpy.real.html#numpy.real)(val) | Return the real part of the complex argument. |
| [**imag**](https://numpy.org/doc/stable/reference/generated/numpy.imag.html#numpy.imag)(val) | Return the imaginary part of the complex argument. |
| [**conj**](https://numpy.org/doc/stable/reference/generated/numpy.conj.html#numpy.conj)(x, /[, out, where, casting, order, ...]) | Return the complex conjugate, element-wise. |
| [**conjugate**](https://numpy.org/doc/stable/reference/generated/numpy.conjugate.html#numpy.conjugate)(x, /[, out, where, casting, ...]) | Return the complex conjugate, element-wise. |

## Extrema Finding

|  |  |
| --- | --- |
| [**maximum**](https://numpy.org/doc/stable/reference/generated/numpy.maximum.html#numpy.maximum)(x1, x2, /[, out, where, casting, ...]) | Element-wise maximum of array elements. |
| [**max**](https://numpy.org/doc/stable/reference/generated/numpy.max.html#numpy.max)(a[, axis, out, keepdims, initial, where]) | Return the maximum of an array or maximum along an axis. |
| [**amax**](https://numpy.org/doc/stable/reference/generated/numpy.amax.html#numpy.amax)(a[, axis, out, keepdims, initial, where]) | Return the maximum of an array or maximum along an axis. |
| [**fmax**](https://numpy.org/doc/stable/reference/generated/numpy.fmax.html#numpy.fmax)(x1, x2, /[, out, where, casting, ...]) | Element-wise maximum of array elements. |
| [**nanmax**](https://numpy.org/doc/stable/reference/generated/numpy.nanmax.html#numpy.nanmax)(a[, axis, out, keepdims, initial, where]) | Return the maximum of an array or maximum along an axis, ignoring any NaNs. |
| [**minimum**](https://numpy.org/doc/stable/reference/generated/numpy.minimum.html#numpy.minimum)(x1, x2, /[, out, where, casting, ...]) | Element-wise minimum of array elements. |
| [**min**](https://numpy.org/doc/stable/reference/generated/numpy.min.html#numpy.min)(a[, axis, out, keepdims, initial, where]) | Return the minimum of an array or minimum along an axis. |
| [**amin**](https://numpy.org/doc/stable/reference/generated/numpy.amin.html#numpy.amin)(a[, axis, out, keepdims, initial, where]) | Return the minimum of an array or minimum along an axis. |
| [**fmin**](https://numpy.org/doc/stable/reference/generated/numpy.fmin.html#numpy.fmin)(x1, x2, /[, out, where, casting, ...]) | Element-wise minimum of array elements. |
| [**nanmin**](https://numpy.org/doc/stable/reference/generated/numpy.nanmin.html#numpy.nanmin)(a[, axis, out, keepdims, initial, where]) | Return minimum of an array or minimum along an axis, ignoring any NaNs. |

## Miscellaneous

|  |  |
| --- | --- |
| [**convolve**](https://numpy.org/doc/stable/reference/generated/numpy.convolve.html#numpy.convolve)(a, v[, mode]) | Returns the discrete, linear convolution of two one-dimensional sequences. |
| [**clip**](https://numpy.org/doc/stable/reference/generated/numpy.clip.html#numpy.clip)(a, a\_min, a\_max[, out]) | Clip (limit) the values in an array. |
| [**sqrt**](https://numpy.org/doc/stable/reference/generated/numpy.sqrt.html#numpy.sqrt)(x, /[, out, where, casting, order, ...]) | Return the non-negative square-root of an array, element-wise. |
| [**cbrt**](https://numpy.org/doc/stable/reference/generated/numpy.cbrt.html#numpy.cbrt)(x, /[, out, where, casting, order, ...]) | Return the cube-root of an array, element-wise. |
| [**square**](https://numpy.org/doc/stable/reference/generated/numpy.square.html#numpy.square)(x, /[, out, where, casting, order, ...]) | Return the element-wise square of the input. |
| [**absolute**](https://numpy.org/doc/stable/reference/generated/numpy.absolute.html#numpy.absolute)(x, /[, out, where, casting, order, ...]) | Calculate the absolute value element-wise. |
| [**fabs**](https://numpy.org/doc/stable/reference/generated/numpy.fabs.html#numpy.fabs)(x, /[, out, where, casting, order, ...]) | Compute the absolute values element-wise. |
| [**sign**](https://numpy.org/doc/stable/reference/generated/numpy.sign.html#numpy.sign)(x, /[, out, where, casting, order, ...]) | Returns an element-wise indication of the sign of a number. |
| [**heaviside**](https://numpy.org/doc/stable/reference/generated/numpy.heaviside.html#numpy.heaviside)(x1, x2, /[, out, where, casting, ...]) | Compute the Heaviside step function. |
| [**nan\_to\_num**](https://numpy.org/doc/stable/reference/generated/numpy.nan_to_num.html#numpy.nan_to_num)(x[, copy, nan, posinf, neginf]) | Replace NaN with zero and infinity with large finite numbers (default behaviour) or with the numbers defined by the user using the [**nan**](https://numpy.org/doc/stable/reference/constants.html#numpy.nan), posinf and/or neginf keywords. |
| [**real\_if\_close**](https://numpy.org/doc/stable/reference/generated/numpy.real_if_close.html#numpy.real_if_close)(a[, tol]) | If input is complex with all imaginary parts close to zero, return real parts. |
| [**interp**](https://numpy.org/doc/stable/reference/generated/numpy.interp.html#numpy.interp)(x, xp, fp[, left, right, period]) | One-dimensional linear interpolation for monotonically increasing sample points. |

Linear Algebra (numpy.linalg):

The @ operator

Introduced in NumPy 1.10.0, the @ operator is preferable to other methods when computing the matrix product between 2d arrays. The **[numpy.matmul](https://numpy.org/doc/stable/reference/generated/numpy.matmul.html" \l "numpy.matmul" \o "numpy.matmul)** function implements the @ operator.

Matrix and vector products

|  |  |
| --- | --- |
| [**dot**](https://numpy.org/doc/stable/reference/generated/numpy.dot.html#numpy.dot)(a, b[, out]) | Dot product of two arrays. |
| [**linalg.multi\_dot**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.multi_dot.html#numpy.linalg.multi_dot)(arrays, \*[, out]) | Compute the dot product of two or more arrays in a single function call, while automatically selecting the fastest evaluation order. |
| [**vdot**](https://numpy.org/doc/stable/reference/generated/numpy.vdot.html#numpy.vdot)(a, b, /) | Return the dot product of two vectors. |
| [**inner**](https://numpy.org/doc/stable/reference/generated/numpy.inner.html#numpy.inner)(a, b, /) | Inner product of two arrays. |
| [**outer**](https://numpy.org/doc/stable/reference/generated/numpy.outer.html#numpy.outer)(a, b[, out]) | Compute the outer product of two vectors. |
| [**matmul**](https://numpy.org/doc/stable/reference/generated/numpy.matmul.html#numpy.matmul)(x1, x2, /[, out, casting, order, ...]) | Matrix product of two arrays. |
| [**tensordot**](https://numpy.org/doc/stable/reference/generated/numpy.tensordot.html#numpy.tensordot)(a, b[, axes]) | Compute tensor dot product along specified axes. |
| [**einsum**](https://numpy.org/doc/stable/reference/generated/numpy.einsum.html#numpy.einsum)(subscripts, \*operands[, out, dtype, ...]) | Evaluates the Einstein summation convention on the operands. |
| [**einsum\_path**](https://numpy.org/doc/stable/reference/generated/numpy.einsum_path.html#numpy.einsum_path)(subscripts, \*operands[, optimize]) | Evaluates the lowest cost contraction order for an einsum expression by considering the creation of intermediate arrays. |
| [**linalg.matrix\_power**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.matrix_power.html#numpy.linalg.matrix_power)(a, n) | Raise a square matrix to the (integer) power *n*. |
| [**kron**](https://numpy.org/doc/stable/reference/generated/numpy.kron.html#numpy.kron)(a, b) | Kronecker product of two arrays. |

Decompositions

|  |  |
| --- | --- |
| [**linalg.cholesky**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.cholesky.html#numpy.linalg.cholesky)(a) | Cholesky decomposition. |
| [**linalg.qr**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.qr.html#numpy.linalg.qr)(a[, mode]) | Compute the qr factorization of a matrix. |
| [**linalg.svd**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html#numpy.linalg.svd)(a[, full\_matrices, compute\_uv, ...]) | Singular Value Decomposition. |

Matrix eigenvalues

|  |  |
| --- | --- |
| [**linalg.eig**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.eig.html#numpy.linalg.eig)(a) | Compute the eigenvalues and right eigenvectors of a square array. |
| [**linalg.eigh**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.eigh.html#numpy.linalg.eigh)(a[, UPLO]) | Return the eigenvalues and eigenvectors of a complex Hermitian (conjugate symmetric) or a real symmetric matrix. |
| [**linalg.eigvals**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.eigvals.html#numpy.linalg.eigvals)(a) | Compute the eigenvalues of a general matrix. |
| [**linalg.eigvalsh**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.eigvalsh.html#numpy.linalg.eigvalsh)(a[, UPLO]) | Compute the eigenvalues of a complex Hermitian or real symmetric matrix. |

Norms and other numbers

|  |  |
| --- | --- |
| [**linalg.norm**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html#numpy.linalg.norm)(x[, ord, axis, keepdims]) | Matrix or vector norm. |
| [**linalg.cond**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.cond.html#numpy.linalg.cond)(x[, p]) | Compute the condition number of a matrix. |
| [**linalg.det**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.det.html#numpy.linalg.det)(a) | Compute the determinant of an array. |
| [**linalg.matrix\_rank**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.matrix_rank.html#numpy.linalg.matrix_rank)(A[, tol, hermitian]) | Return matrix rank of array using SVD method |
| [**linalg.slogdet**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.slogdet.html#numpy.linalg.slogdet)(a) | Compute the sign and (natural) logarithm of the determinant of an array. |
| [**trace**](https://numpy.org/doc/stable/reference/generated/numpy.trace.html#numpy.trace)(a[, offset, axis1, axis2, dtype, out]) | Return the sum along diagonals of the array. |

Solving equations and inverting matrices

|  |  |
| --- | --- |
| [**linalg.solve**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html#numpy.linalg.solve)(a, b) | Solve a linear matrix equation, or system of linear scalar equations. |
| [**linalg.tensorsolve**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.tensorsolve.html#numpy.linalg.tensorsolve)(a, b[, axes]) | Solve the tensor equation a x = b for x. |
| [**linalg.lstsq**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html#numpy.linalg.lstsq)(a, b[, rcond]) | Return the least-squares solution to a linear matrix equation. |
| [**linalg.inv**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.inv.html#numpy.linalg.inv)(a) | Compute the (multiplicative) inverse of a matrix. |
| [**linalg.pinv**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.pinv.html#numpy.linalg.pinv)(a[, rcond, hermitian]) | Compute the (Moore-Penrose) pseudo-inverse of a matrix. |
| [**linalg.tensorinv**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.tensorinv.html#numpy.linalg.tensorinv)(a[, ind]) | Compute the 'inverse' of an N-dimensional array. |

Exceptions

|  |  |
| --- | --- |
| [**linalg.LinAlgError**](https://numpy.org/doc/stable/reference/generated/numpy.linalg.LinAlgError.html#numpy.linalg.LinAlgError) | Generic Python-exception-derived object raised by linalg functions. |

Pandas Tutorial:

# How do I read tabular data file into pandas?

**pandas.read\_table**

**pandas.read\_table(*filepath\_or\_buffer*,***\****, *sep****=\_NoDefault.no\_default***, *delimiter****=None***, *header****='infer'***, *names****=\_NoDefault.no\_default***, *index\_col****=None***, *usecols****=None***, *dtype****=None***, *engine****=None***, *converters****=None***, *true\_values****=None***, *false\_values****=None***, *skipinitialspace****=False***, *skiprows****=None***, *skipfooter****=0***, *nrows****=None***, *na\_values****=None***, *keep\_default\_na****=True***, *na\_filter****=True***, *verbose****=\_NoDefault.no\_default***, *skip\_blank\_lines****=True***, *parse\_dates****=False***, *infer\_datetime\_format****=\_NoDefault.no\_default***, *keep\_date\_col****=\_NoDefault.no\_default***, *date\_parser****=\_NoDefault.no\_default***, *date\_format****=None***, *dayfirst****=False***, *cache\_dates****=True***, *iterator****=False***, *chunksize****=None***, *compression****='infer'***, *thousands****=None***, *decimal****='.'***, *lineterminator****=None***, *quotechar****='"'***, *quoting****=0***, *doublequote****=True***, *escapechar****=None***, *comment****=None***, *encoding****=None***, *encoding\_errors****='strict'***, *dialect****=None***, *on\_bad\_lines****='error'***, *delim\_whitespace****=\_NoDefault.no\_default***, *low\_memory****=True***, *memory\_map****=False***, *float\_precision****=None***, *storage\_options****=None***, *dtype\_backend****=\_NoDefault.no\_default***)**[**[source]**](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/io/parsers/readers.py#L1257-L1405)

Read general delimited file into DataFrame.

Also supports optionally iterating or breaking of the file into chunks.

**Parameters:**

**filepath\_or\_buffer*str, path object or file-like object***

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, gs, and file. For file URLs, a host is expected. A local file could be: [file://localhost/path/to/table.csv](file:///\\localhost\path\to\table.csv).

If you want to pass in a path object, pandas accepts any os.PathLike.

By file-like object, we refer to objects with a read() method, such as a file handle (e.g. via builtin open function) or StringIO.

**sep*str, default ‘\t’ (tab-stop)***

Character or regex pattern to treat as the delimiter. If sep=None, the C engine cannot automatically detect the separator, but the Python parsing engine can, meaning the latter will be used and automatically detect the separator from only the first valid row of the file by Python’s builtin sniffer tool, csv.Sniffer. In addition, separators longer than 1 character and different from '\s+' will be interpreted as regular expressions and will also force the use of the Python parsing engine. Note that regex delimiters are prone to ignoring quoted data. Regex example: '\r\t'.

**delimiter*str, optional***

Alias for sep.

**header*int, Sequence of int, ‘infer’ or None, default ‘infer’***

Row number(s) containing column labels and marking the start of the data (zero-indexed). Default behavior is to infer the column names: if no names are passed the behavior is identical to header=0 and column names are inferred from the first line of the file, if column names are passed explicitly to names then the behavior is identical to header=None. Explicitly pass header=0 to be able to replace existing names. The header can be a list of integers that specify row locations for a **[MultiIndex](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.MultiIndex.html" \l "pandas.MultiIndex" \o "pandas.MultiIndex)** on the columns e.g. [0, 1, 3]. Intervening rows that are not specified will be skipped (e.g. 2 in this example is skipped). Note that this parameter ignores commented lines and empty lines if skip\_blank\_lines=True, so header=0 denotes the first line of data rather than the first line of the file.

**names*Sequence of Hashable, optional***

Sequence of column labels to apply. If the file contains a header row, then you should explicitly pass header=0 to override the column names. Duplicates in this list are not allowed.

**index\_col*Hashable, Sequence of Hashable or False, optional***

Column(s) to use as row label(s), denoted either by column labels or column indices. If a sequence of labels or indices is given, **[MultiIndex](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.MultiIndex.html" \l "pandas.MultiIndex" \o "pandas.MultiIndex)** will be formed for the row labels.

Note: index\_col=False can be used to force pandas to *not* use the first column as the index, e.g., when you have a malformed file with delimiters at the end of each line.

**usecols*Sequence of Hashable or Callable, optional***

Subset of columns to select, denoted either by column labels or column indices. If list-like, all elements must either be positional (i.e. integer indices into the document columns) or strings that correspond to column names provided either by the user in names or inferred from the document header row(s). If names are given, the document header row(s) are not taken into account. For example, a valid list-like usecols parameter would be [0, 1, 2] or ['foo', 'bar', 'baz']. Element order is ignored, so usecols=[0, 1] is the same as [1, 0]. To instantiate a **[DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \o "pandas.DataFrame)** from data with element order preserved use pd.read\_csv(data, usecols=['foo', 'bar'])[['foo', 'bar']] for columns in ['foo', 'bar'] order or pd.read\_csv(data, usecols=['foo', 'bar'])[['bar', 'foo']] for ['bar', 'foo'] order.

If callable, the callable function will be evaluated against the column names, returning names where the callable function evaluates to True. An example of a valid callable argument would be lambda x: x.upper() in ['AAA', 'BBB', 'DDD']. Using this parameter results in much faster parsing time and lower memory usage.

**dtype*dtype or dict of {Hashabledtype}, optional***

Data type(s) to apply to either the whole dataset or individual columns. E.g., {'a': np.float64, 'b': np.int32, 'c': 'Int64'} Use str or object together with suitable na\_values settings to preserve and not interpret dtype. If converters are specified, they will be applied INSTEAD of dtype conversion.

***New in version 1.5.0:***Support for defaultdict was added. Specify a defaultdict as input where the default determines the dtype of the columns which are not explicitly listed.

**engine*{‘c’, ‘python’, ‘pyarrow’}, optional***

Parser engine to use. The C and pyarrow engines are faster, while the python engine is currently more feature-complete. Multithreading is currently only supported by the pyarrow engine.

***New in version 1.4.0:***The ‘pyarrow’ engine was added as an *experimental* engine, and some features are unsupported, or may not work correctly, with this engine.

**converters*dict of {HashableCallable}, optional***

Functions for converting values in specified columns. Keys can either be column labels or column indices.

**true\_values*list, optional***

Values to consider as True in addition to case-insensitive variants of ‘True’.

**false\_values*list, optional***

Values to consider as False in addition to case-insensitive variants of ‘False’.

**skipinitialspace*bool, default False***

Skip spaces after delimiter.

**skiprows*int, list of int or Callable, optional***

Line numbers to skip (0-indexed) or number of lines to skip (int) at the start of the file.

If callable, the callable function will be evaluated against the row indices, returning True if the row should be skipped and False otherwise. An example of a valid callable argument would be lambda x: x in [0, 2].

**skipfooter*int, default 0***

Number of lines at bottom of file to skip (Unsupported with engine='c').

**nrows*int, optional***

Number of rows of file to read. Useful for reading pieces of large files.

**na\_values*Hashable, Iterable of Hashable or dict of {HashableIterable}, optional***

Additional strings to recognize as NA/NaN. If dict passed, specific per-column NA values. By default the following values are interpreted as NaN: “ “, “#N/A”, “#N/A N/A”, “#NA”, “-1.#IND”, “-1.#QNAN”, “-NaN”, “-nan”, “1.#IND”, “1.#QNAN”, “<NA>”, “N/A”, “NA”, “NULL”, “NaN”, “None”, “n/a”, “nan”, “null “.

**keep\_default\_na*bool, default True***

Whether or not to include the default NaN values when parsing the data. Depending on whether na\_values is passed in, the behavior is as follows:

* If keep\_default\_na is True, and na\_values are specified, na\_values is appended to the default NaN values used for parsing.
* If keep\_default\_na is True, and na\_values are not specified, only the default NaN values are used for parsing.
* If keep\_default\_na is False, and na\_values are specified, only the NaN values specified na\_values are used for parsing.
* If keep\_default\_na is False, and na\_values are not specified, no strings will be parsed as NaN.

Note that if na\_filter is passed in as False, the keep\_default\_na and na\_values parameters will be ignored.

**na\_filter*bool, default True***

Detect missing value markers (empty strings and the value of na\_values). In data without any NA values, passing na\_filter=False can improve the performance of reading a large file.

**verbose*bool, default False***

Indicate number of NA values placed in non-numeric columns.

***Deprecated since version 2.2.0.***

**skip\_blank\_lines*bool, default True***

If True, skip over blank lines rather than interpreting as NaN values.

**parse\_dates*bool, list of Hashable, list of lists or dict of {Hashablelist}, default False***

The behavior is as follows:

* bool. If True -> try parsing the index. Note: Automatically set to True if date\_format or date\_parser arguments have been passed.
* list of int or names. e.g. If [1, 2, 3] -> try parsing columns 1, 2, 3 each as a separate date column.
* list of list. e.g. If [[1, 3]] -> combine columns 1 and 3 and parse as a single date column. Values are joined with a space before parsing.
* dict, e.g. {'foo' : [1, 3]} -> parse columns 1, 3 as date and call result ‘foo’. Values are joined with a space before parsing.

If a column or index cannot be represented as an array of datetime, say because of an unparsable value or a mixture of timezones, the column or index will be returned unaltered as an object data type. For non-standard datetime parsing, use **[to\_datetime()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html" \l "pandas.to_datetime" \o "pandas.to_datetime)** after **[read\_csv()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html" \l "pandas.read_csv" \o "pandas.read_csv)**.

Note: A fast-path exists for iso8601-formatted dates.

**infer\_datetime\_format*bool, default False***

If True and parse\_dates is enabled, pandas will attempt to infer the format of the datetime strings in the columns, and if it can be inferred, switch to a faster method of parsing them. In some cases this can increase the parsing speed by 5-10x.

***Deprecated since version 2.0.0:***A strict version of this argument is now the default, passing it has no effect.

**keep\_date\_col*bool, default False***

If True and parse\_dates specifies combining multiple columns then keep the original columns.

**date\_parser*Callable, optional***

Function to use for converting a sequence of string columns to an array of datetime instances. The default uses dateutil.parser.parser to do the conversion. pandas will try to call date\_parser in three different ways, advancing to the next if an exception occurs: 1) Pass one or more arrays (as defined by parse\_dates) as arguments; 2) concatenate (row-wise) the string values from the columns defined by parse\_dates into a single array and pass that; and 3) call date\_parser once for each row using one or more strings (corresponding to the columns defined by parse\_dates) as arguments.

***Deprecated since version 2.0.0:***Use date\_format instead, or read in as object and then apply **[to\_datetime()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html" \l "pandas.to_datetime" \o "pandas.to_datetime)** as-needed.

**date\_format*str or dict of column -> format, optional***

Format to use for parsing dates when used in conjunction with parse\_dates. The strftime to parse time, e.g. **"%d/%m/%Y"**. See [strftime documentation](https://docs.python.org/3/library/datetime.html" \l "strftime-and-strptime-behavior) for more information on choices, though note that **"%f"** will parse all the way up to nanoseconds. You can also pass:

* **“ISO8601”, to parse any**[**ISO8601**](https://en.wikipedia.org/wiki/ISO_8601)

time string (not necessarily in exactly the same format);

* **“mixed”, to infer the format for each element individually. This is risky,**

and you should probably use it along with *dayfirst*.

***New in version 2.0.0.***

**dayfirst*bool, default False***

DD/MM format dates, international and European format.

**cache\_dates*bool, default True***

If True, use a cache of unique, converted dates to apply the datetime conversion. May produce significant speed-up when parsing duplicate date strings, especially ones with timezone offsets.

**iterator*bool, default False***

Return TextFileReader object for iteration or getting chunks with get\_chunk().

**chunksize*int, optional***

Number of lines to read from the file per chunk. Passing a value will cause the function to return a TextFileReader object for iteration. See the [IO Tools docs](https://pandas.pydata.org/pandas-docs/stable/io.html#io-chunking) for more information on iterator and chunksize.

**compression*str or dict, default ‘infer’***

For on-the-fly decompression of on-disk data. If ‘infer’ and ‘filepath\_or\_buffer’ is path-like, then detect compression from the following extensions: ‘.gz’, ‘.bz2’, ‘.zip’, ‘.xz’, ‘.zst’, ‘.tar’, ‘.tar.gz’, ‘.tar.xz’ or ‘.tar.bz2’ (otherwise no compression). If using ‘zip’ or ‘tar’, the ZIP file must contain only one data file to be read in. Set to None for no decompression. Can also be a dict with key 'method' set to one of {'zip', 'gzip', 'bz2', 'zstd', 'xz', 'tar'} and other key-value pairs are forwarded to zipfile.ZipFile, gzip.GzipFile, bz2.BZ2File, zstandard.ZstdDecompressor, lzma.LZMAFile or tarfile.TarFile, respectively. As an example, the following could be passed for Zstandard decompression using a custom compression dictionary: compression={'method': 'zstd', 'dict\_data': my\_compression\_dict}.

***New in version 1.5.0:***Added support for *.tar* files.

***Changed in version 1.4.0:***Zstandard support.

**thousands*str (length 1), optional***

Character acting as the thousands separator in numerical values.

**decimal*str (length 1), default ‘.’***

Character to recognize as decimal point (e.g., use ‘,’ for European data).

**lineterminator*str (length 1), optional***

Character used to denote a line break. Only valid with C parser.

**quotechar*str (length 1), optional***

Character used to denote the start and end of a quoted item. Quoted items can include the delimiter and it will be ignored.

**quoting*{0 or csv.QUOTE\_MINIMAL, 1 or csv.QUOTE\_ALL, 2 or csv.QUOTE\_NONNUMERIC, 3 or csv.QUOTE\_NONE}, default csv.QUOTE\_MINIMAL***

Control field quoting behavior per csv.QUOTE\_\* constants. Default is csv.QUOTE\_MINIMAL (i.e., 0) which implies that only fields containing special characters are quoted (e.g., characters defined in quotechar, delimiter, or lineterminator.

**doublequote*bool, default True***

When quotechar is specified and quoting is not QUOTE\_NONE, indicate whether or not to interpret two consecutive quotechar elements INSIDE a field as a single quotechar element.

**escapechar*str (length 1), optional***

Character used to escape other characters.

**comment*str (length 1), optional***

Character indicating that the remainder of line should not be parsed. If found at the beginning of a line, the line will be ignored altogether. This parameter must be a single character. Like empty lines (as long as skip\_blank\_lines=True), fully commented lines are ignored by the parameter header but not by skiprows. For example, if comment='#', parsing #empty\na,b,c\n1,2,3 with header=0 will result in 'a,b,c' being treated as the header.

**encoding*str, optional, default ‘utf-8’***

Encoding to use for UTF when reading/writing (ex. 'utf-8'). [List of Python standard encodings](https://docs.python.org/3/library/codecs.html#standard-encodings) .

**encoding\_errors*str, optional, default ‘strict’***

How encoding errors are treated. [List of possible values](https://docs.python.org/3/library/codecs.html#error-handlers) .

***New in version 1.3.0.***

**dialect*str or csv.Dialect, optional***

If provided, this parameter will override values (default or not) for the following parameters: delimiter, doublequote, escapechar, skipinitialspace, quotechar, and quoting. If it is necessary to override values, a ParserWarning will be issued. See csv.Dialect documentation for more details.

**on\_bad\_lines*{‘error’, ‘warn’, ‘skip’} or Callable, default ‘error’***

Specifies what to do upon encountering a bad line (a line with too many fields). Allowed values are :

* 'error', raise an Exception when a bad line is encountered.
* 'warn', raise a warning when a bad line is encountered and skip that line.
* 'skip', skip bad lines without raising or warning when they are encountered.

***New in version 1.3.0.***

***New in version 1.4.0:***

* Callable, function with signature (bad\_line: list[str]) -> list[str] | None that will process a single bad line. bad\_line is a list of strings split by the sep. If the function returns None, the bad line will be ignored. If the function returns a new list of strings with more elements than expected, a ParserWarning will be emitted while dropping extra elements. Only supported when engine='python'

***Changed in version 2.2.0:***

* Callable, function with signature as described in [pyarrow documentation](https://arrow.apache.org/docs/python/generated/pyarrow.csv.ParseOptions.html" \l "pyarrow.csv.ParseOptions.invalid_row_handler) when engine='pyarrow'

**delim\_whitespace*bool, default False***

Specifies whether or not whitespace (e.g. ' ' or '\t') will be used as the sep delimiter. Equivalent to setting sep='\s+'. If this option is set to True, nothing should be passed in for the delimiter parameter.

***Deprecated since version 2.2.0:***Use sep="\s+" instead.

**low\_memory*bool, default True***

Internally process the file in chunks, resulting in lower memory use while parsing, but possibly mixed type inference. To ensure no mixed types either set False, or specify the type with the dtype parameter. Note that the entire file is read into a single **[DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \o "pandas.DataFrame)** regardless, use the chunksize or iterator parameter to return the data in chunks. (Only valid with C parser).

**memory\_map*bool, default False***

If a filepath is provided for filepath\_or\_buffer, map the file object directly onto memory and access the data directly from there. Using this option can improve performance because there is no longer any I/O overhead.

**float\_precision*{‘high’, ‘legacy’, ‘round\_trip’}, optional***

Specifies which converter the C engine should use for floating-point values. The options are None or 'high' for the ordinary converter, 'legacy' for the original lower precision pandas converter, and 'round\_trip' for the round-trip converter.

**storage\_options*dict, optional***

Extra options that make sense for a particular storage connection, e.g. host, port, username, password, etc. For HTTP(S) URLs the key-value pairs are forwarded to urllib.request.Request as header options. For other URLs (e.g. starting with “s3://”, and “gcs://”) the key-value pairs are forwarded to fsspec.open. Please see fsspec and urllib for more details, and for more examples on storage options refer [here](https://pandas.pydata.org/docs/user_guide/io.html?highlight=storage_options#reading-writing-remote-files).

**dtype\_backend*{‘numpy\_nullable’, ‘pyarrow’}, default ‘numpy\_nullable’***

Back-end data type applied to the resultant **[DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \o "pandas.DataFrame)** (still experimental). Behaviour is as follows:

* "numpy\_nullable": returns nullable-dtype-backed **[DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \o "pandas.DataFrame)** (default).
* "pyarrow": returns pyarrow-backed nullable **[ArrowDtype](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.ArrowDtype.html" \l "pandas.ArrowDtype" \o "pandas.ArrowDtype)** DataFrame.

***New in version 2.0.***

**Returns:**

**DataFrame or TextFileReader**

A comma-separated values (csv) file is returned as two-dimensional data structure with labeled axes.

[**DataFrame.to\_csv**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_csv.html#pandas.DataFrame.to_csv)**:** Write DataFrame to a comma-separated values (csv) file.

[**read\_csv**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html#pandas.read_csv)**:** Read a comma-separated values (csv) file into DataFrame.

[**read\_fwf**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_fwf.html#pandas.read_fwf)**:** Read a table of fixed-width formatted lines into DataFrame.

# How do I select a Series from a pandas DataFrame:

DataFrames and Series are the two main object types in pandas for data storage: a DataFrame is like a table, and each column of the table is called a Series.

You will often select a Series in order to analyze or manipulate it.

In this video, I'll show you how to select a Series using "bracket notation" and "dot notation", and will discuss the limitations of dot notation. I'll also demonstrate how to create a new Series in a DataFrame.

# Why do some pandas commands have parantheses, while others don’t?

The DataFrame attributes (for example, df.shape, df.dtypes) do not use parantheses

The DataFrame methods (for example, df.head(), df.describe()) do use parantheses. They can also have some mandatory and/or optional arguments.

What are methods? - In Python, a **method** is a function that is available for a given object **because of the object's type**.

For example, if you create my\_list = [1, 2, 3], the append method can be applied to my\_list because it's a Python list: my\_list.append(4). All lists have an append method simply because they are lists.

Pandas.DataFrame.describe( )

**DataFrame.describe(*percentiles****=None***, *include****=None***, *exclude****=None***)**[**[source]**](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/core/generic.py#L11734-L11981)

Generate descriptive statistics.

Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset’s distribution, excluding NaN values.

Analyzes both numeric and object series, as well as DataFrame column sets of mixed data types. The output will vary depending on what is provided. Refer to the notes below for more detail.

**Parameters:**

**percentiles*list-like of numbers, optional***

The percentiles to include in the output. All should fall between 0 and 1. The default is [.25, .5, .75], which returns the 25th, 50th, and 75th percentiles.

**include*‘all’, list-like of dtypes or None (default), optional***

A white list of data types to include in the result. Ignored for Series. Here are the options:

* ‘all’ : All columns of the input will be included in the output.
* A list-like of dtypes : Limits the results to the provided data types. To limit the result to numeric types submit numpy.number. To limit it instead to object columns submit the numpy.object data type. Strings can also be used in the style of select\_dtypes (e.g. df.describe(include=['O'])). To select pandas categorical columns, use 'category'
* None (default) : The result will include all numeric columns.

**exclude*list-like of dtypes or None (default), optional,***

A black list of data types to omit from the result. Ignored for Series. Here are the options:

* A list-like of dtypes : Excludes the provided data types from the result. To exclude numeric types submit numpy.number. To exclude object columns submit the data type numpy.object. Strings can also be used in the style of select\_dtypes (e.g. df.describe(exclude=['O'])). To exclude pandas categorical columns, use 'category'
* None (default) : The result will exclude nothing.

**Returns:**

**Series or DataFrame :** Summary statistics of the Series or Dataframe provided.

# How do I rename columns in a pandas DataFrame?

* 1. **pandas.DataFrame.rename:**

**DataFrame.rename(*mapper****=None***,***\****, *index****=None***, *columns****=None***, *axis****=None***, *copy****=None***, *inplace****=False***, *level****=None***, *errors****='ignore'***)**[**[source]**](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/core/frame.py#L5636-L5776)

Rename columns or index labels.

Function / dict values must be unique (1-to-1). Labels not contained in a dict / Series will be left as-is. Extra labels listed don’t throw an error.

See the [user guide](https://pandas.pydata.org/pandas-docs/stable/user_guide/basics.html#basics-rename) for more.

**Parameters:**

**mapper*dict-like or function***

Dict-like or function transformations to apply to that axis’ values. Use either mapper and axis to specify the axis to target with mapper, or index and columns.

**index*dict-like or function***

Alternative to specifying axis (mapper, axis=0 is equivalent to index=mapper).

**columns*dict-like or function***

Alternative to specifying axis (mapper, axis=1 is equivalent to columns=mapper).

**axis*{0 or ‘index’, 1 or ‘columns’}, default 0***

Axis to target with mapper. Can be either the axis name (‘index’, ‘columns’) or number (0, 1). The default is ‘index’.

**copy*bool, default True***

Also copy underlying data.

Note

The *copy* keyword will change behavior in pandas 3.0. [Copy-on-Write](https://pandas.pydata.org/docs/dev/user_guide/copy_on_write.html) will be enabled by default, which means that all methods with a *copy* keyword will use a lazy copy mechanism to defer the copy and ignore the *copy* keyword. The *copy* keyword will be removed in a future version of pandas.

You can already get the future behavior and improvements through enabling copy on write pd.options.mode.copy\_on\_write = True

**inplace*bool, default False***

Whether to modify the DataFrame rather than creating a new one. If True then value of copy is ignored.

**level*int or level name, default None***

In case of a MultiIndex, only rename labels in the specified level.

**errors*{‘ignore’, ‘raise’}, default ‘ignore’***

If ‘raise’, raise a *KeyError* when a dict-like *mapper*, *index*, or *columns* contains labels that are not present in the Index being transformed. If ‘ignore’, existing keys will be renamed and extra keys will be ignored.

**Returns:**

**DataFrame or None**

DataFrame with the renamed axis labels or None if inplace=True.

**Raises:**

**KeyError**

If any of the labels is not found in the selected axis and “errors=’raise’”.

* 1. ufo.columns = ufo\_cols
  2. ufo = pd.read\_csv('http://bit.ly/uforeports', names=ufo\_cols, header=0)

**pandas.read\_csv:**

**pandas.read\_csv(*filepath\_or\_buffer*,***\****, *sep****=\_NoDefault.no\_default***, *delimiter****=None***, *header****='infer'***, *names****=\_NoDefault.no\_default***, *index\_col****=None***, *usecols****=None***, *dtype****=None***, *engine****=None***, *converters****=None***, *true\_values****=None***, *false\_values****=None***, *skipinitialspace****=False***, *skiprows****=None***, *skipfooter****=0***, *nrows****=None***, *na\_values****=None***, *keep\_default\_na****=True***, *na\_filter****=True***, *verbose****=\_NoDefault.no\_default***, *skip\_blank\_lines****=True***, *parse\_dates****=None***, *infer\_datetime\_format****=\_NoDefault.no\_default***, *keep\_date\_col****=\_NoDefault.no\_default***, *date\_parser****=\_NoDefault.no\_default***, *date\_format****=None***, *dayfirst****=False***, *cache\_dates****=True***, *iterator****=False***, *chunksize****=None***, *compression****='infer'***, *thousands****=None***, *decimal****='.'***, *lineterminator****=None***, *quotechar****='"'***, *quoting****=0***, *doublequote****=True***, *escapechar****=None***, *comment****=None***, *encoding****=None***, *encoding\_errors****='strict'***, *dialect****=None***, *on\_bad\_lines****='error'***, *delim\_whitespace****=\_NoDefault.no\_default***, *low\_memory****=True***, *memory\_map****=False***, *float\_precision****=None***, *storage\_options****=None***, *dtype\_backend****=\_NoDefault.no\_default***)**[**[source]**](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/io/parsers/readers.py#L868-L1026)

Read a comma-separated values (csv) file into DataFrame.

Also supports optionally iterating or breaking of the file into chunks.

Additional help can be found in the online docs for [IO Tools](https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html).

**Parameters:**

**filepath\_or\_buffer*str, path object or file-like object***

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, gs, and file. For file URLs, a host is expected. A local file could be: [file://localhost/path/to/table.csv](file:///\\localhost\path\to\table.csv).

If you want to pass in a path object, pandas accepts any os.PathLike.

By file-like object, we refer to objects with a read() method, such as a file handle (e.g. via builtin open function) or StringIO.

**sep*str, default ‘,’***

Character or regex pattern to treat as the delimiter. If sep=None, the C engine cannot automatically detect the separator, but the Python parsing engine can, meaning the latter will be used and automatically detect the separator from only the first valid row of the file by Python’s builtin sniffer tool, csv.Sniffer. In addition, separators longer than 1 character and different from '\s+' will be interpreted as regular expressions and will also force the use of the Python parsing engine. Note that regex delimiters are prone to ignoring quoted data. Regex example: '\r\t'.

**delimiter*str, optional***

Alias for sep.

**header*int, Sequence of int, ‘infer’ or None, default ‘infer’***

Row number(s) containing column labels and marking the start of the data (zero-indexed). Default behavior is to infer the column names: if no names are passed the behavior is identical to header=0 and column names are inferred from the first line of the file, if column names are passed explicitly to names then the behavior is identical to header=None. Explicitly pass header=0 to be able to replace existing names. The header can be a list of integers that specify row locations for a **[MultiIndex](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.MultiIndex.html" \l "pandas.MultiIndex" \o "pandas.MultiIndex)** on the columns e.g. [0, 1, 3]. Intervening rows that are not specified will be skipped (e.g. 2 in this example is skipped). Note that this parameter ignores commented lines and empty lines if skip\_blank\_lines=True, so header=0 denotes the first line of data rather than the first line of the file.

**names*Sequence of Hashable, optional***

Sequence of column labels to apply. If the file contains a header row, then you should explicitly pass header=0 to override the column names. Duplicates in this list are not allowed.

**index\_col*Hashable, Sequence of Hashable or False, optional***

Column(s) to use as row label(s), denoted either by column labels or column indices. If a sequence of labels or indices is given, **[MultiIndex](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.MultiIndex.html" \l "pandas.MultiIndex" \o "pandas.MultiIndex)** will be formed for the row labels.

Note: index\_col=False can be used to force pandas to *not* use the first column as the index, e.g., when you have a malformed file with delimiters at the end of each line.

**usecols*Sequence of Hashable or Callable, optional***

Subset of columns to select, denoted either by column labels or column indices. If list-like, all elements must either be positional (i.e. integer indices into the document columns) or strings that correspond to column names provided either by the user in names or inferred from the document header row(s). If names are given, the document header row(s) are not taken into account. For example, a valid list-like usecols parameter would be [0, 1, 2] or ['foo', 'bar', 'baz']. Element order is ignored, so usecols=[0, 1] is the same as [1, 0]. To instantiate a **[DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \o "pandas.DataFrame)** from data with element order preserved use pd.read\_csv(data, usecols=['foo', 'bar'])[['foo', 'bar']] for columns in ['foo', 'bar'] order or pd.read\_csv(data, usecols=['foo', 'bar'])[['bar', 'foo']] for ['bar', 'foo'] order.

If callable, the callable function will be evaluated against the column names, returning names where the callable function evaluates to True. An example of a valid callable argument would be lambda x: x.upper() in ['AAA', 'BBB', 'DDD']. Using this parameter results in much faster parsing time and lower memory usage.

**dtype*dtype or dict of {Hashabledtype}, optional***

Data type(s) to apply to either the whole dataset or individual columns. E.g., {'a': np.float64, 'b': np.int32, 'c': 'Int64'} Use str or object together with suitable na\_values settings to preserve and not interpret dtype. If converters are specified, they will be applied INSTEAD of dtype conversion.

***New in version 1.5.0:***Support for defaultdict was added. Specify a defaultdict as input where the default determines the dtype of the columns which are not explicitly listed.

**engine*{‘c’, ‘python’, ‘pyarrow’}, optional***

Parser engine to use. The C and pyarrow engines are faster, while the python engine is currently more feature-complete. Multithreading is currently only supported by the pyarrow engine.

***New in version 1.4.0:***The ‘pyarrow’ engine was added as an *experimental* engine, and some features are unsupported, or may not work correctly, with this engine.

**converters*dict of {HashableCallable}, optional***

Functions for converting values in specified columns. Keys can either be column labels or column indices.

**true\_values*list, optional***

Values to consider as True in addition to case-insensitive variants of ‘True’.

**false\_values*list, optional***

Values to consider as False in addition to case-insensitive variants of ‘False’.

**skipinitialspace*bool, default False***

Skip spaces after delimiter.

**skiprows*int, list of int or Callable, optional***

Line numbers to skip (0-indexed) or number of lines to skip (int) at the start of the file.

If callable, the callable function will be evaluated against the row indices, returning True if the row should be skipped and False otherwise. An example of a valid callable argument would be lambda x: x in [0, 2].

**skipfooter*int, default 0***

Number of lines at bottom of file to skip (Unsupported with engine='c').

**nrows*int, optional***

Number of rows of file to read. Useful for reading pieces of large files.

**na\_values*Hashable, Iterable of Hashable or dict of {HashableIterable}, optional***

Additional strings to recognize as NA/NaN. If dict passed, specific per-column NA values. By default the following values are interpreted as NaN: “ “, “#N/A”, “#N/A N/A”, “#NA”, “-1.#IND”, “-1.#QNAN”, “-NaN”, “-nan”, “1.#IND”, “1.#QNAN”, “<NA>”, “N/A”, “NA”, “NULL”, “NaN”, “None”, “n/a”, “nan”, “null “.

**keep\_default\_na*bool, default True***

Whether or not to include the default NaN values when parsing the data. Depending on whether na\_values is passed in, the behavior is as follows:

* If keep\_default\_na is True, and na\_values are specified, na\_values is appended to the default NaN values used for parsing.
* If keep\_default\_na is True, and na\_values are not specified, only the default NaN values are used for parsing.
* If keep\_default\_na is False, and na\_values are specified, only the NaN values specified na\_values are used for parsing.
* If keep\_default\_na is False, and na\_values are not specified, no strings will be parsed as NaN.

Note that if na\_filter is passed in as False, the keep\_default\_na and na\_values parameters will be ignored.

**na\_filter*bool, default True***

Detect missing value markers (empty strings and the value of na\_values). In data without any NA values, passing na\_filter=False can improve the performance of reading a large file.

**verbose*bool, default False***

Indicate number of NA values placed in non-numeric columns.

***Deprecated since version 2.2.0.***

**skip\_blank\_lines*bool, default True***

If True, skip over blank lines rather than interpreting as NaN values.

**parse\_dates*bool, list of Hashable, list of lists or dict of {Hashablelist}, default False***

The behavior is as follows:

* bool. If True -> try parsing the index. Note: Automatically set to True if date\_format or date\_parser arguments have been passed.
* list of int or names. e.g. If [1, 2, 3] -> try parsing columns 1, 2, 3 each as a separate date column.
* list of list. e.g. If [[1, 3]] -> combine columns 1 and 3 and parse as a single date column. Values are joined with a space before parsing.
* dict, e.g. {'foo' : [1, 3]} -> parse columns 1, 3 as date and call result ‘foo’. Values are joined with a space before parsing.

If a column or index cannot be represented as an array of datetime, say because of an unparsable value or a mixture of timezones, the column or index will be returned unaltered as an object data type. For non-standard datetime parsing, use **[to\_datetime()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html" \l "pandas.to_datetime" \o "pandas.to_datetime)** after **[read\_csv()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html" \l "pandas.read_csv" \o "pandas.read_csv)**.

Note: A fast-path exists for iso8601-formatted dates.

**infer\_datetime\_format*bool, default False***

If True and parse\_dates is enabled, pandas will attempt to infer the format of the datetime strings in the columns, and if it can be inferred, switch to a faster method of parsing them. In some cases this can increase the parsing speed by 5-10x.

***Deprecated since version 2.0.0:***A strict version of this argument is now the default, passing it has no effect.

**keep\_date\_col*bool, default False***

If True and parse\_dates specifies combining multiple columns then keep the original columns.

**date\_parser*Callable, optional***

Function to use for converting a sequence of string columns to an array of datetime instances. The default uses dateutil.parser.parser to do the conversion. pandas will try to call date\_parser in three different ways, advancing to the next if an exception occurs: 1) Pass one or more arrays (as defined by parse\_dates) as arguments; 2) concatenate (row-wise) the string values from the columns defined by parse\_dates into a single array and pass that; and 3) call date\_parser once for each row using one or more strings (corresponding to the columns defined by parse\_dates) as arguments.

***Deprecated since version 2.0.0:***Use date\_format instead, or read in as object and then apply **[to\_datetime()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html" \l "pandas.to_datetime" \o "pandas.to_datetime)** as-needed.

**date\_format*str or dict of column -> format, optional***

Format to use for parsing dates when used in conjunction with parse\_dates. The strftime to parse time, e.g. **"%d/%m/%Y"**. See [strftime documentation](https://docs.python.org/3/library/datetime.html" \l "strftime-and-strptime-behavior) for more information on choices, though note that **"%f"** will parse all the way up to nanoseconds. You can also pass:

* **“ISO8601”, to parse any**[**ISO8601**](https://en.wikipedia.org/wiki/ISO_8601)

time string (not necessarily in exactly the same format);

* **“mixed”, to infer the format for each element individually. This is risky,**

and you should probably use it along with *dayfirst*.

***New in version 2.0.0.***

**dayfirst*bool, default False***

DD/MM format dates, international and European format.

**cache\_dates*bool, default True***

If True, use a cache of unique, converted dates to apply the datetime conversion. May produce significant speed-up when parsing duplicate date strings, especially ones with timezone offsets.

**iterator*bool, default False***

Return TextFileReader object for iteration or getting chunks with get\_chunk().

**chunksize*int, optional***

Number of lines to read from the file per chunk. Passing a value will cause the function to return a TextFileReader object for iteration. See the [IO Tools docs](https://pandas.pydata.org/pandas-docs/stable/io.html#io-chunking) for more information on iterator and chunksize.

**compression*str or dict, default ‘infer’***

For on-the-fly decompression of on-disk data. If ‘infer’ and ‘filepath\_or\_buffer’ is path-like, then detect compression from the following extensions: ‘.gz’, ‘.bz2’, ‘.zip’, ‘.xz’, ‘.zst’, ‘.tar’, ‘.tar.gz’, ‘.tar.xz’ or ‘.tar.bz2’ (otherwise no compression). If using ‘zip’ or ‘tar’, the ZIP file must contain only one data file to be read in. Set to None for no decompression. Can also be a dict with key 'method' set to one of {'zip', 'gzip', 'bz2', 'zstd', 'xz', 'tar'} and other key-value pairs are forwarded to zipfile.ZipFile, gzip.GzipFile, bz2.BZ2File, zstandard.ZstdDecompressor, lzma.LZMAFile or tarfile.TarFile, respectively. As an example, the following could be passed for Zstandard decompression using a custom compression dictionary: compression={'method': 'zstd', 'dict\_data': my\_compression\_dict}.

***New in version 1.5.0:***Added support for *.tar* files.

***Changed in version 1.4.0:***Zstandard support.

**thousands*str (length 1), optional***

Character acting as the thousands separator in numerical values.

**decimal*str (length 1), default ‘.’***

Character to recognize as decimal point (e.g., use ‘,’ for European data).

**lineterminator*str (length 1), optional***

Character used to denote a line break. Only valid with C parser.

**quotechar*str (length 1), optional***

Character used to denote the start and end of a quoted item. Quoted items can include the delimiter and it will be ignored.

**quoting*{0 or csv.QUOTE\_MINIMAL, 1 or csv.QUOTE\_ALL, 2 or csv.QUOTE\_NONNUMERIC, 3 or csv.QUOTE\_NONE}, default csv.QUOTE\_MINIMAL***

Control field quoting behavior per csv.QUOTE\_\* constants. Default is csv.QUOTE\_MINIMAL (i.e., 0) which implies that only fields containing special characters are quoted (e.g., characters defined in quotechar, delimiter, or lineterminator.

**doublequote*bool, default True***

When quotechar is specified and quoting is not QUOTE\_NONE, indicate whether or not to interpret two consecutive quotechar elements INSIDE a field as a single quotechar element.

**escapechar*str (length 1), optional***

Character used to escape other characters.

**comment*str (length 1), optional***

Character indicating that the remainder of line should not be parsed. If found at the beginning of a line, the line will be ignored altogether. This parameter must be a single character. Like empty lines (as long as skip\_blank\_lines=True), fully commented lines are ignored by the parameter header but not by skiprows. For example, if comment='#', parsing #empty\na,b,c\n1,2,3 with header=0 will result in 'a,b,c' being treated as the header.

**encoding*str, optional, default ‘utf-8’***

Encoding to use for UTF when reading/writing (ex. 'utf-8'). [List of Python standard encodings](https://docs.python.org/3/library/codecs.html#standard-encodings) .

**encoding\_errors*str, optional, default ‘strict’***

How encoding errors are treated. [List of possible values](https://docs.python.org/3/library/codecs.html#error-handlers) .

***New in version 1.3.0.***

**dialect*str or csv.Dialect, optional***

If provided, this parameter will override values (default or not) for the following parameters: delimiter, doublequote, escapechar, skipinitialspace, quotechar, and quoting. If it is necessary to override values, a ParserWarning will be issued. See csv.Dialect documentation for more details.

**on\_bad\_lines*{‘error’, ‘warn’, ‘skip’} or Callable, default ‘error’***

Specifies what to do upon encountering a bad line (a line with too many fields). Allowed values are :

* 'error', raise an Exception when a bad line is encountered.
* 'warn', raise a warning when a bad line is encountered and skip that line.
* 'skip', skip bad lines without raising or warning when they are encountered.

***New in version 1.3.0.***

***New in version 1.4.0:***

* Callable, function with signature (bad\_line: list[str]) -> list[str] | None that will process a single bad line. bad\_line is a list of strings split by the sep. If the function returns None, the bad line will be ignored. If the function returns a new list of strings with more elements than expected, a ParserWarning will be emitted while dropping extra elements. Only supported when engine='python'

***Changed in version 2.2.0:***

* Callable, function with signature as described in [pyarrow documentation](https://arrow.apache.org/docs/python/generated/pyarrow.csv.ParseOptions.html" \l "pyarrow.csv.ParseOptions.invalid_row_handler) when engine='pyarrow'

**delim\_whitespace*bool, default False***

Specifies whether or not whitespace (e.g. ' ' or '\t') will be used as the sep delimiter. Equivalent to setting sep='\s+'. If this option is set to True, nothing should be passed in for the delimiter parameter.

***Deprecated since version 2.2.0:***Use sep="\s+" instead.

**low\_memory*bool, default True***

Internally process the file in chunks, resulting in lower memory use while parsing, but possibly mixed type inference. To ensure no mixed types either set False, or specify the type with the dtype parameter. Note that the entire file is read into a single **[DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \o "pandas.DataFrame)** regardless, use the chunksize or iterator parameter to return the data in chunks. (Only valid with C parser).

**memory\_map*bool, default False***

If a filepath is provided for filepath\_or\_buffer, map the file object directly onto memory and access the data directly from there. Using this option can improve performance because there is no longer any I/O overhead.

**float\_precision*{‘high’, ‘legacy’, ‘round\_trip’}, optional***

Specifies which converter the C engine should use for floating-point values. The options are None or 'high' for the ordinary converter, 'legacy' for the original lower precision pandas converter, and 'round\_trip' for the round-trip converter.

**storage\_options*dict, optional***

Extra options that make sense for a particular storage connection, e.g. host, port, username, password, etc. For HTTP(S) URLs the key-value pairs are forwarded to urllib.request.Request as header options. For other URLs (e.g. starting with “s3://”, and “gcs://”) the key-value pairs are forwarded to fsspec.open. Please see fsspec and urllib for more details, and for more examples on storage options refer [here](https://pandas.pydata.org/docs/user_guide/io.html?highlight=storage_options#reading-writing-remote-files).

**dtype\_backend*{‘numpy\_nullable’, ‘pyarrow’}, default ‘numpy\_nullable’***

Back-end data type applied to the resultant **[DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \o "pandas.DataFrame)** (still experimental). Behaviour is as follows:

* "numpy\_nullable": returns nullable-dtype-backed **[DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \o "pandas.DataFrame)** (default).
* "pyarrow": returns pyarrow-backed nullable **[ArrowDtype](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.ArrowDtype.html" \l "pandas.ArrowDtype" \o "pandas.ArrowDtype)** DataFrame.

***New in version 2.0.***

**Returns:**

**DataFrame or TextFileReader**

A comma-separated values (csv) file is returned as two-dimensional data structure with labeled axes.

**4. pandas.Series.str.replace**

**Series.str.replace(*pat*, *repl*, *n****=-1***, *case****=None***, *flags****=0***, *regex****=False***)**[**[source]**](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/core/strings/accessor.py#L1419-L1570)

Replace each occurrence of pattern/regex in the Series/Index.

Equivalent to **[str.replace()](https://docs.python.org/3/library/stdtypes.html" \l "str.replace" \o "(in Python v3.12))** or **[re.sub()](https://docs.python.org/3/library/re.html" \l "re.sub" \o "(in Python v3.12))**, depending on the regex value.

**Parameters:**

**pat*str or compiled regex***

String can be a character sequence or regular expression.

**repl*str or callable***

Replacement string or a callable. The callable is passed the regex match object and must return a replacement string to be used. See **[re.sub()](https://docs.python.org/3/library/re.html" \l "re.sub" \o "(in Python v3.12))**.

**n*int, default -1 (all)***

Number of replacements to make from start.

**case*bool, default None***

Determines if replace is case sensitive:

* If True, case sensitive (the default if *pat* is a string)
* Set to False for case insensitive
* Cannot be set if *pat* is a compiled regex.

**flags*int, default 0 (no flags)***

Regex module flags, e.g. re.IGNORECASE. Cannot be set if *pat* is a compiled regex.

**regex*bool, default False***

Determines if the passed-in pattern is a regular expression:

* If True, assumes the passed-in pattern is a regular expression.
* If False, treats the pattern as a literal string
* Cannot be set to False if *pat* is a compiled regex or *repl* is a callable.

**Returns:**

**Series or Index of object**

A copy of the object with all matching occurrences of *pat* replaced by *repl*.

**Raises:**

**ValueError**

* if *regex* is False and *repl* is a callable or *pat* is a compiled regex
* if *pat* is a compiled regex and *case* or *flags* is set

# How do I remove columns from a pandas DataFrame?

**pandas.DataFrame.drop:**

**DataFrame.drop(*labels****=None***,***\****, *axis****=0***, *index****=None***, *columns****=None***, *level****=None***, *inplace****=False***, *errors****='raise'***)**[**[source]**](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/core/frame.py#L5433-L5589)

Drop specified labels from rows or columns.

Remove rows or columns by specifying label names and corresponding axis, or by directly specifying index or column names. When using a multi-index, labels on different levels can be removed by specifying the level. See the [user guide](https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#advanced-shown-levels) for more information about the now unused levels.

**Parameters:**

**labels*single label or list-like***

Index or column labels to drop. A tuple will be used as a single label and not treated as a list-like.

**axis*{0 or ‘index’, 1 or ‘columns’}, default 0***

Whether to drop labels from the index (0 or ‘index’) or columns (1 or ‘columns’).

**index*single label or list-like***

Alternative to specifying axis (labels, axis=0 is equivalent to index=labels).

**columns*single label or list-like***

Alternative to specifying axis (labels, axis=1 is equivalent to columns=labels).

**level*int or level name, optional***

For MultiIndex, level from which the labels will be removed.

**inplace*bool, default False***

If False, return a copy. Otherwise, do operation in place and return None.

**errors*{‘ignore’, ‘raise’}, default ‘raise’***

If ‘ignore’, suppress error and only existing labels are dropped.

**Returns:**

**DataFrame or None**

Returns DataFrame or None DataFrame with the specified index or column labels removed or None if inplace=True.

**Raises:**

**KeyError**

If any of the labels is not found in the selected axis.

# How do I sort a pandas DataFrame or Series?

1. **pandas.DataFrame.sort\_values:**

**DataFrame.sort\_values(*by*,***\****, *axis****=0***, *ascending****=True***, *inplace****=False***, *kind****='quicksort'***, *na\_position****='last'***, *ignore\_index****=False***, *key****=None***)**[**[source]**](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/core/frame.py#L6997-L7232)

Sort by the values along either axis.

**Parameters:**

**by*str or list of str***

Name or list of names to sort by.

* if *axis* is 0 or *‘index’* then *by* may contain index levels and/or column labels.
* if *axis* is 1 or *‘columns’* then *by* may contain column levels and/or index labels.

**axis*“{0 or ‘index’, 1 or ‘columns’}”, default 0***

Axis to be sorted.

**ascending*bool or list of bool, default True***

Sort ascending vs. descending. Specify list for multiple sort orders. If this is a list of bools, must match the length of the by.

**inplace*bool, default False***

If True, perform operation in-place.

**kind*{‘quicksort’, ‘mergesort’, ‘heapsort’, ‘stable’}, default ‘quicksort’***

Choice of sorting algorithm. See also **[numpy.sort()](https://numpy.org/doc/stable/reference/generated/numpy.sort.html" \l "numpy.sort" \o "(in NumPy v1.26))** for more information. *mergesort* and *stable* are the only stable algorithms. For DataFrames, this option is only applied when sorting on a single column or label.

**na\_position*{‘first’, ‘last’}, default ‘last’***

Puts NaNs at the beginning if *first*; *last* puts NaNs at the end.

**ignore\_index*bool, default False***

If True, the resulting axis will be labeled 0, 1, …, n - 1.

**key*callable, optional***

Apply the key function to the values before sorting. This is similar to the *key* argument in the builtin **sorted()** function, with the notable difference that this *key* function should be *vectorized*. It should expect a Series and return a Series with the same shape as the input. It will be applied to each column in *by* independently.

**Returns:**

**DataFrame or None**

DataFrame with sorted values or None if inplace=True.

2. **pandas.Series.sort\_values:**

**Series.sort\_values(***\****, *axis****=0***, *ascending****=True***, *inplace****=False***, *kind****='quicksort'***, *na\_position****='last'***, *ignore\_index****=False***, *key****=None***)**[**[source]**](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/core/series.py#L3687-L3886)

Sort by the values.

Sort a Series in ascending or descending order by some criterion.

**Parameters:**

**axis*{0 or ‘index’}***

Unused. Parameter needed for compatibility with DataFrame.

**ascending*bool or list of bools, default True***

If True, sort values in ascending order, otherwise descending.

**inplace*bool, default False***

If True, perform operation in-place.

**kind*{‘quicksort’, ‘mergesort’, ‘heapsort’, ‘stable’}, default ‘quicksort’***

Choice of sorting algorithm. See also **[numpy.sort()](https://numpy.org/doc/stable/reference/generated/numpy.sort.html" \l "numpy.sort" \o "(in NumPy v1.26))** for more information. ‘mergesort’ and ‘stable’ are the only stable algorithms.

**na\_position*{‘first’ or ‘last’}, default ‘last’***

Argument ‘first’ puts NaNs at the beginning, ‘last’ puts NaNs at the end.

**ignore\_index*bool, default False***

If True, the resulting axis will be labeled 0, 1, …, n - 1.

**key*callable, optional***

If not None, apply the key function to the series values before sorting. This is similar to the *key* argument in the builtin **sorted()** function, with the notable difference that this *key* function should be *vectorized*. It should expect a Series and return an array-like.

**Returns:**

**Series or None**

Series ordered by values or None if inplace=True.

# How do I filter rows from a pandas DataFrame based on their column values?

1. booleans = [ ]

for length in movies['duration']:

  if length >= 200:

    booleans.append(True)

  else:

    booleans.append(False)

is\_long = pd.Series(booleans)

movies[is\_long]

2. is\_long = movies['duration'] >= 200

movies[is\_long]

3. movies[movies['duration'] >= 200]

**pandas.DataFrame.loc:**

***property*DataFrame.loc[[source]](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/core/indexing.py" \l "L305-L609)**

Access a group of rows and columns by label(s) or a boolean array.

.loc[] is primarily label based, but may also be used with a boolean array.

Allowed inputs are:

* A single label, e.g. 5 or 'a', (note that 5 is interpreted as a *label* of the index, and **never** as an integer position along the index).
* A list or array of labels, e.g. ['a', 'b', 'c'].
* A slice object with labels, e.g. 'a':'f'.

Warning

Note that contrary to usual python slices, **both** the start and the stop are included

* A boolean array of the same length as the axis being sliced, e.g. [True, False, True].
* An alignable boolean Series. The index of the key will be aligned before masking.
* An alignable Index. The Index of the returned selection will be the input.
* A callable function with one argument (the calling Series or DataFrame) and that returns valid output for indexing (one of the above)

See more at [Selection by Label](http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#indexing-label).

**Raises:**

**KeyError**

If any items are not found.

**IndexingError**

If an indexed key is passed and its index is unalignable to the frame index.

# How do I apply multiple filter criteria in a pandas DataFrame?

**pandas.Series.isin:**

**Series.isin(*values*)**[**[source]**](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/core/series.py#L5486-L5562)

Whether elements in Series are contained in *values*.

Return a boolean Series showing whether each element in the Series matches an element in the passed sequence of *values* exactly.

**Parameters:**

**values*set or list-like***

The sequence of values to test. Passing in a single string will raise a TypeError. Instead, turn a single string into a list of one element.

**Returns:**

**Series**

Series of booleans indicating if each element is in values.

**Raises:**

**TypeError**

* If *values* is a string

# How do I use the “axis” parameter in pandas?

**pandas.DataFrame.drop:**

**DataFrame.drop(*labels****=None***,***\****, *axis****=0***, *index****=None***, *columns****=None***, *level****=None***, *inplace****=False***, *errors****='raise'***)**[**[source]**](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/core/frame.py#L5433-L5589)

Drop specified labels from rows or columns.

Remove rows or columns by specifying label names and corresponding axis, or by directly specifying index or column names. When using a multi-index, labels on different levels can be removed by specifying the level. See the [user guide](https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#advanced-shown-levels) for more information about the now unused levels.

**Parameters:**

**labels*single label or list-like***

Index or column labels to drop. A tuple will be used as a single label and not treated as a list-like.

**axis*{0 or ‘index’, 1 or ‘columns’}, default 0***

Whether to drop labels from the index (0 or ‘index’) or columns (1 or ‘columns’).

**index*single label or list-like***

Alternative to specifying axis (labels, axis=0 is equivalent to index=labels).

**columns*single label or list-like***

Alternative to specifying axis (labels, axis=1 is equivalent to columns=labels).

**level*int or level name, optional***

For MultiIndex, level from which the labels will be removed.

**inplace*bool, default False***

If False, return a copy. Otherwise, do operation in place and return None.

**errors*{‘ignore’, ‘raise’}, default ‘raise’***

If ‘ignore’, suppress error and only existing labels are dropped.

**Returns:**

**DataFrame or None**

Returns DataFrame or None DataFrame with the specified index or column labels removed or None if inplace=True.

**Raises:**

**KeyError**

If any of the labels is not found in the selected axis.

**pandas.DataFrame.mean:**

**DataFrame.mean(*axis****=0***, *skipna****=True***, *numeric\_only****=False***,***\*\*****kwargs*)**[**[source]**](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/core/frame.py#L11685-L11696)

Return the mean of the values over the requested axis.

**Parameters:**

**axis*{index (0), columns (1)}***

Axis for the function to be applied on. For *Series* this parameter is unused and defaults to 0.

For DataFrames, specifying axis=None will apply the aggregation across both axes.

***New in version 2.0.0.***

**skipna*bool, default True***

Exclude NA/null values when computing the result.

**numeric\_only*bool, default False***

Include only float, int, boolean columns. Not implemented for Series.

**\*\*kwargs**

Additional keyword arguments to be passed to the function.

**Returns:**

**Series or scalar**

**# How do I use string methods in pandas?**

**Pandas.series.str.method( )**

**# How do I change the data type of a pandas Series?**

**pandas.Series.astype:**

**Series.astype(*dtype*, *copy=None*, *errors='raise'*)**[**[source]**](https://github.com/pandas-dev/pandas/blob/v2.2.2/pandas/core/generic.py#L6463-L6660)

**Cast a pandas object to a specified dtype dtype.**

**Parameters:**

**dtype*str, data type, Series or Mapping of column name -> data type***

**Use a str, numpy.dtype, pandas.ExtensionDtype or Python type to cast entire pandas object to the same type. Alternatively, use a mapping, e.g. {col: dtype, …}, where col is a column label and dtype is a numpy.dtype or Python type to cast one or more of the DataFrame’s columns to column-specific types.**

**copy*bool, default True***

**Return a copy when copy=True (be very careful setting copy=False as changes to values then may propagate to other pandas objects).**

**Note**

**The *copy* keyword will change behavior in pandas 3.0.**[**Copy-on-Write**](https://pandas.pydata.org/docs/dev/user_guide/copy_on_write.html)**will be enabled by default, which means that all methods with a *copy* keyword will use a lazy copy mechanism to defer the copy and ignore the *copy* keyword. The *copy* keyword will be removed in a future version of pandas.**

**You can already get the future behavior and improvements through enabling copy on write pd.options.mode.copy\_on\_write = True**

**errors*{‘raise’, ‘ignore’}, default ‘raise’***

**Control raising of exceptions on invalid data for provided dtype.**

* **raise : allow exceptions to be raised**
* **ignore : suppress exceptions. On error return original object.**

**Returns:**

**same type as caller**

MatPlotLib Tutorial:

Link for notebook:

https://colab.research.google.com/drive/1iWTAoHSBViIl6eo3AMSxtyMUk7cWaSI3?usp=sharing