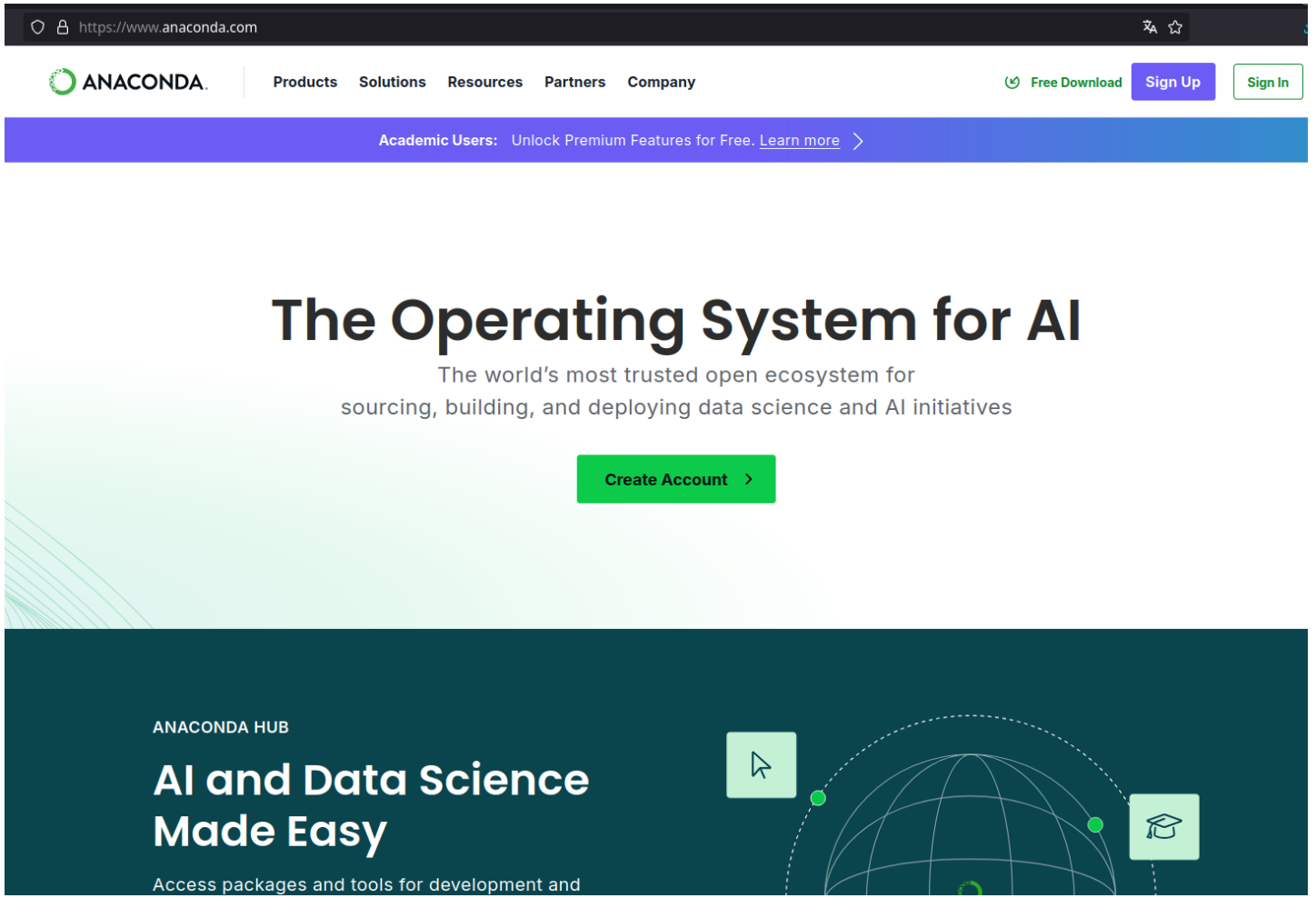
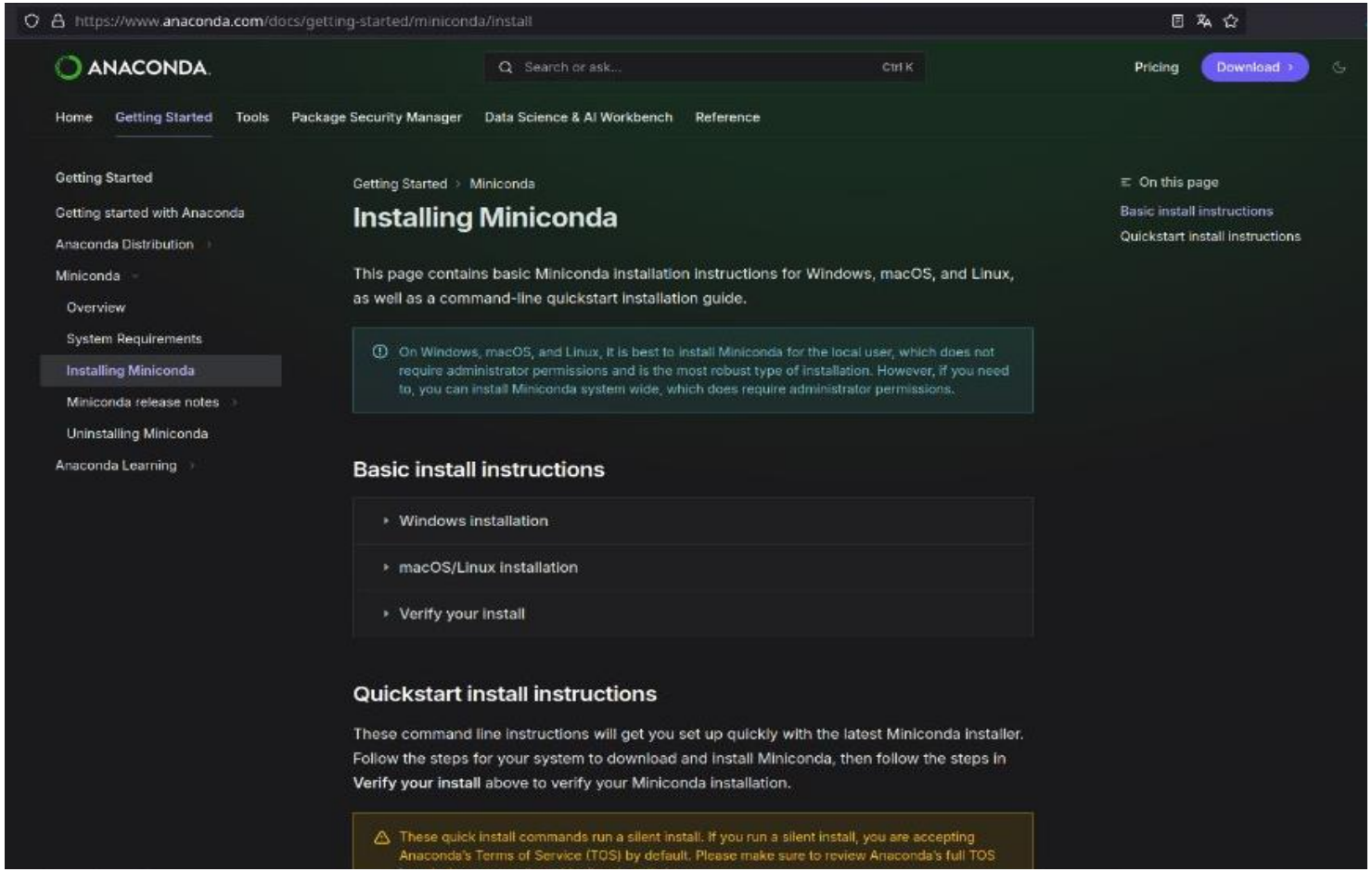
Essential tools for weather and climate data analysis

Proyectos en ingeniería de datos e inteligencia artificial

Anaconda / Miniconda

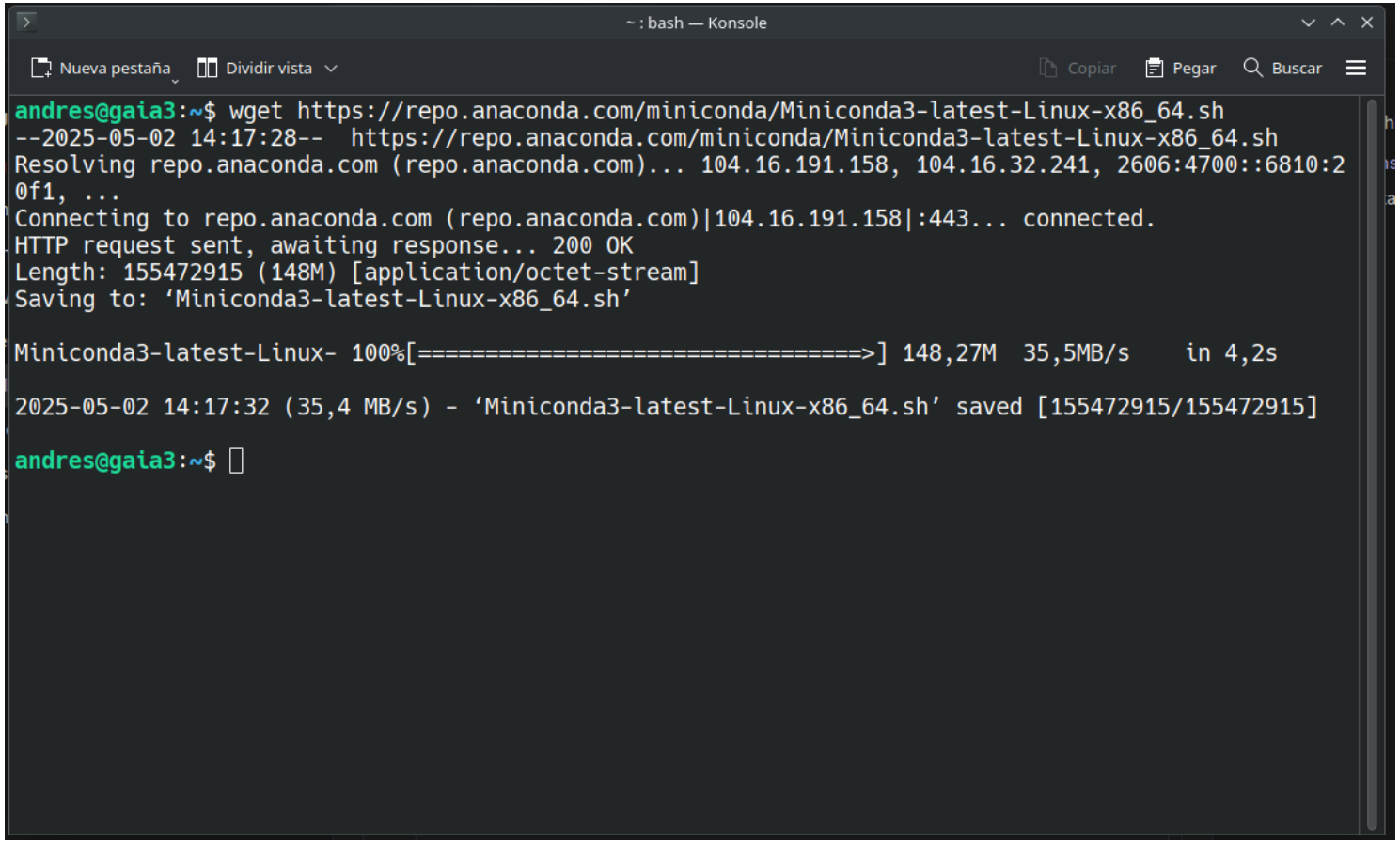


Anaconda / Miniconda



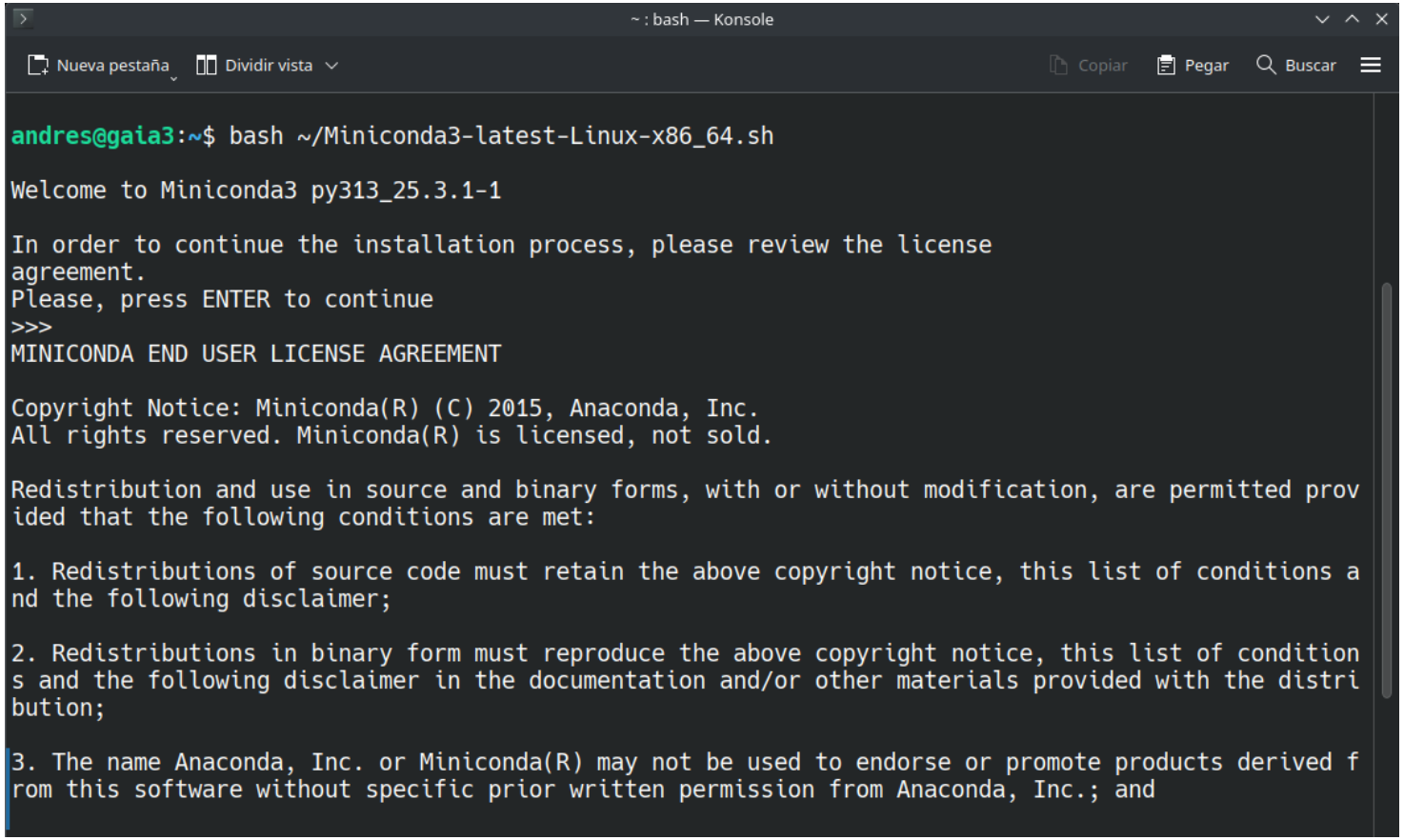
https://www.anaconda.com/docs/getting-started/miniconda/install#macos-linux-installation

Download & Install



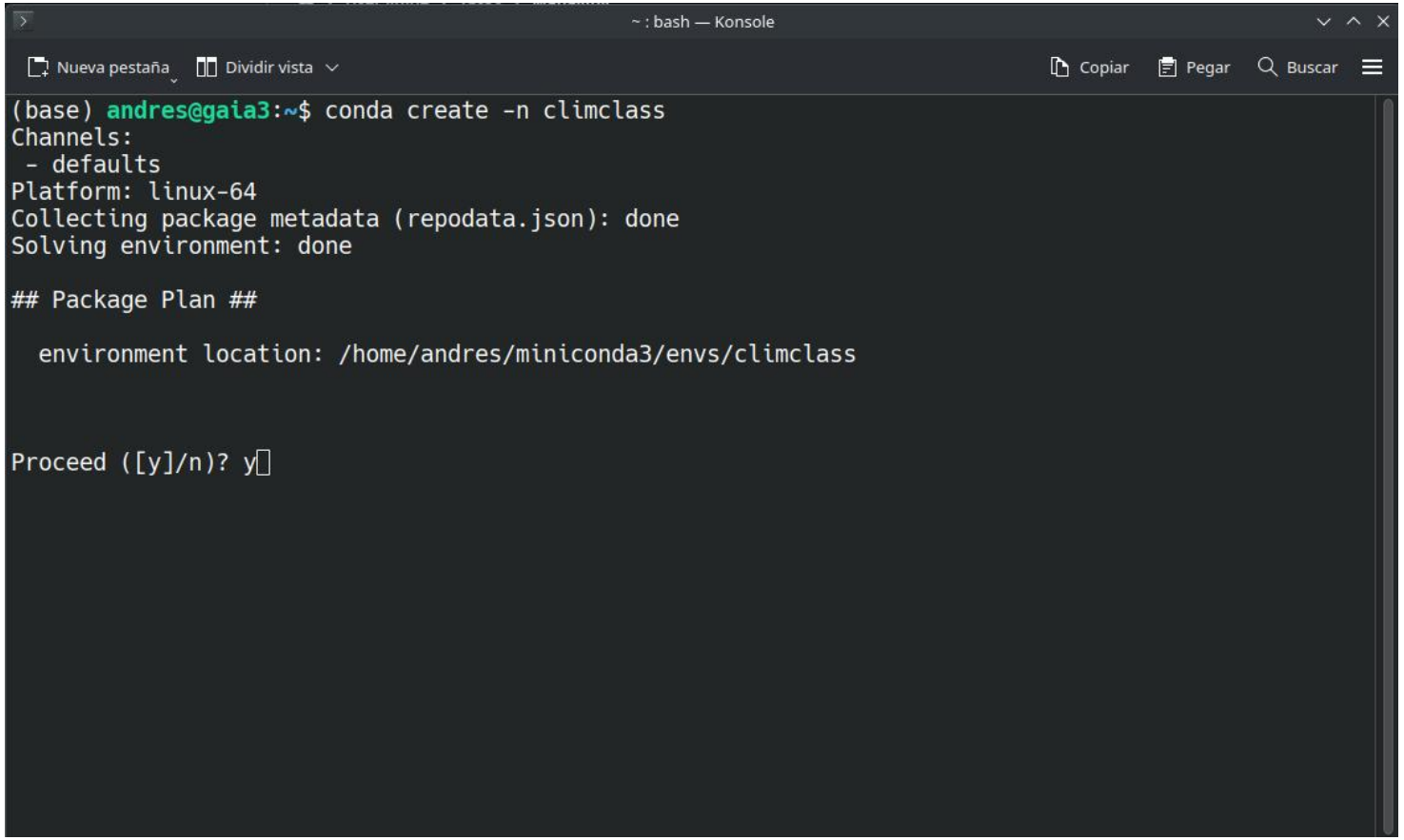

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

Download & Install



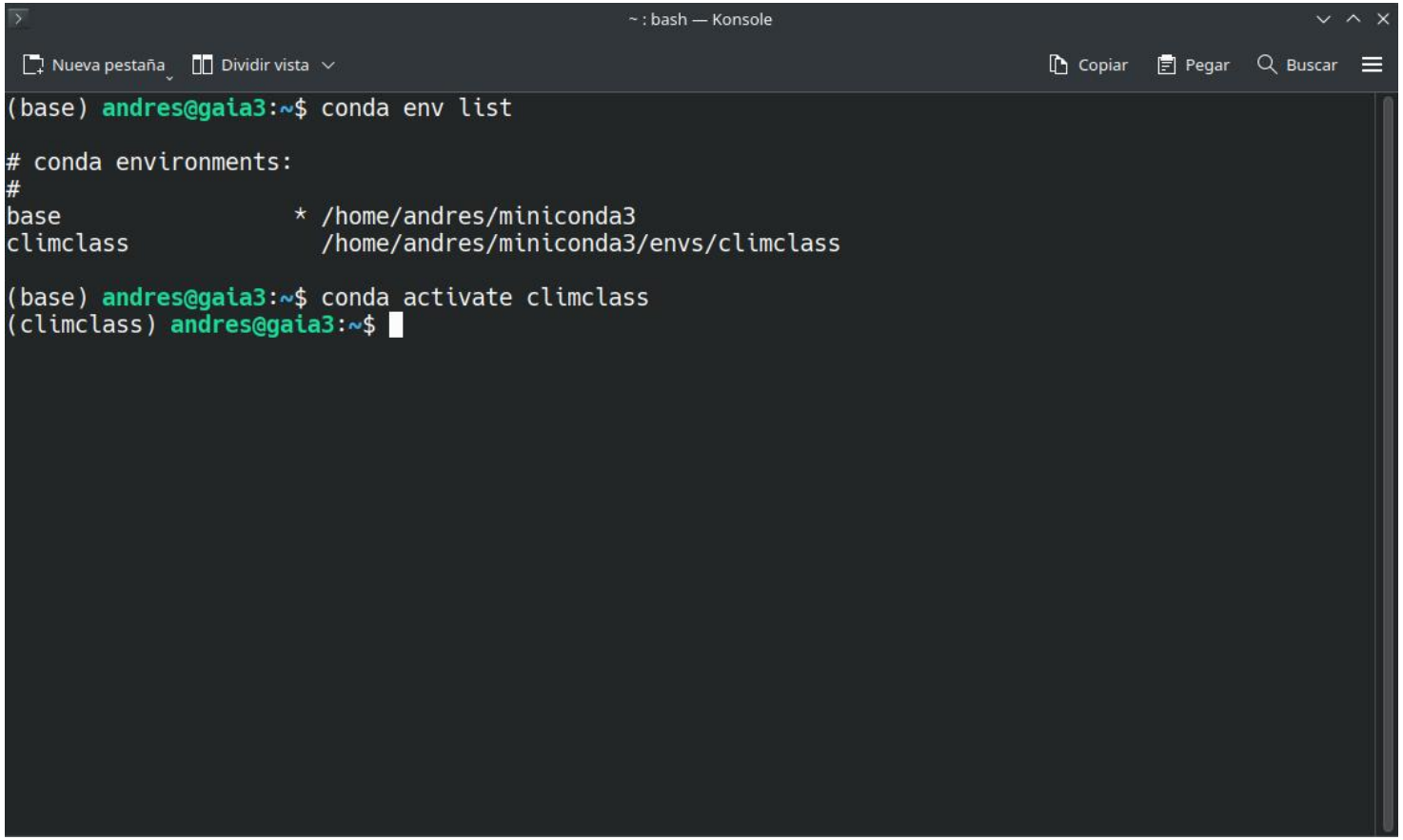

```
bash ~/Miniconda3-latest-Linux-x86_64.sh
```

Create new environment *climclass*



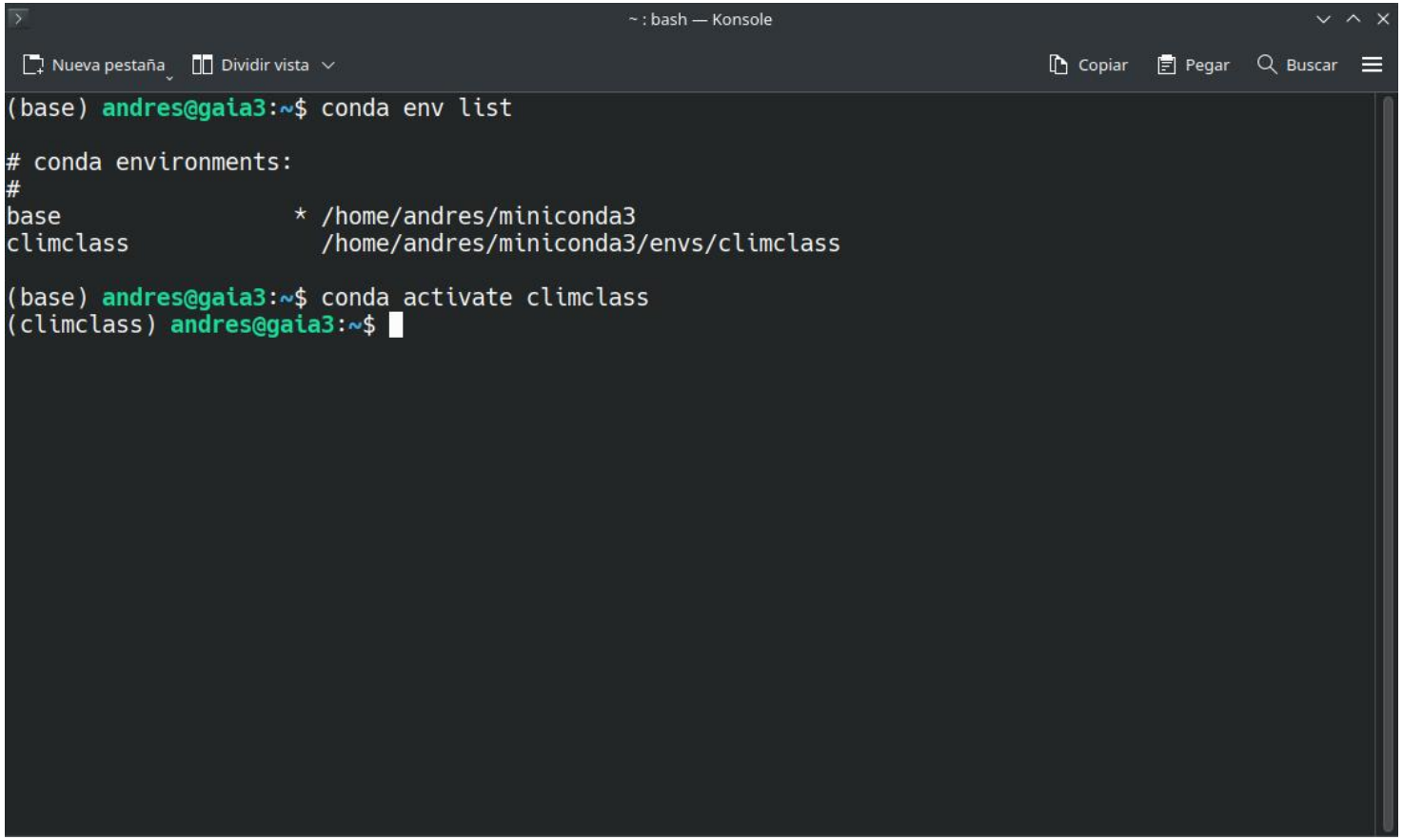
conda create -n climclass

Activate environment *climclass*



conda activate climclass / conda deactivate

Installing required packages



conda activate climclass / conda deactivate

Install Documentation Learn Community About Us News Contribute English

NumPy

The fundamental package for scientific computing with Python

LATEST RELEASE: NUMPY 2.3. VIEW ALL RELEASES

NumPy 2.3.0 released! 2025-06-07

Powerful N-dimensional arrays

Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.

Numerical computing tools

NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

Open source

Distributed under a liberal BSD license, NumPy is developed and maintained publicly on GitHub by a vibrant, responsive, and diverse community.

Interoperable

NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries.

Performant

The core of NumPy is well-optimized C code. Enjoy the flexibility of Python with the speed of compiled code.

Easy to use

NumPy's high level syntax makes it accessible and productive for programmers from any background or experience level.

https://numpy.org/

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

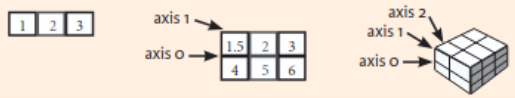
NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

`>>> np.zeros((3,4))`	Create an array of zeros
`>>> np.ones((2,3,4),dtype=np.int16)`	Create an array of ones
`>>> d = np.arange(10,25,5)`	Create an array of evenly spaced values (step value)
`>>> np.linspace(0,2,9)`	Create an array of evenly spaced values (number of samples)
`>>> e = np.full((2,2),7)`	Create a constant array
`>>> f = np.eye(2)`	Create a 2X2 identity matrix
`>>> np.random.random((2,2))`	Create an array with random values
`>>> np.empty((3,2))`	Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

`>>> np.int64`	Signed 64-bit integer types
`>>> np.float32`	Standard double-precision floating point
`>>> np.complex`	Complex numbers represented by 128 floats
`>>> np.bool`	Boolean type storing `TRUE` and `FALSE` values
`>>> np.object`	Python object type
`>>> np.string_`	Fixed-length string type
`>>> np.unicode_`	Fixed-length unicode type

Inspecting Your Array

`>>> a.shape`	Array dimensions
`>>> len(a)`	Length of array
`>>> b.ndim`	Number of array dimensions
`>>> e.size`	Number of array elements
`>>> b.dtype`	Data type of array elements
`>>> b.dtype.name`	Name of data type
`>>> b.astype(int)`	Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b
 array([[-0.5,  0. ,  0. ],
        [-3. , -3. , -3. ]])
>>> np.subtract(a,b)
>>> b + a
 array([[ 2.5,  4. ,  6. ],
        [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)
>>> a / b
 array([[ 0.66666667,  1.        ,  1.        ],
        [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)
>>> a * b
 array([[  1.5,   4. ,   9. ],
        [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
 array([[ 7.,  7.],
        [ 7.,  7.]])
```

Subtraction

Subtraction
Addition

Addition
Division

Division
Multiplication

Multiplication
Exponentiation
Square root
Print sines of an array
Element-wise cosine
Element-wise natural logarithm
Dot product

Comparison

```
>>> a == b
 array([[False,  True,  True],
        [False, False, False]], dtype=bool)
>>> a < 2
 array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)
```

Element-wise comparison

Element-wise comparison

Array-wise comparison

Aggregate Functions

`>>> a.sum()`	Array-wise sum
`>>> a.min()`	Array-wise minimum value
`>>> b.max(axis=0)`	Maximum value of an array row
`>>> b.cumsum(axis=1)`	Cumulative sum of the elements
`>>> a.mean()`	Mean
`>>> b.median()`	Median
`>>> a.corrcoef()`	Correlation coefficient
`>>> np.std(b)`	Standard deviation

Copying Arrays

`>>> h = a.view()`	Create a view of the array with the same data
`>>> np.copy(a)`	Create a copy of the array
`>>> h = a.copy()`	Create a deep copy of the array

Sorting Arrays

`>>> a.sort()`	Sort an array
`>>> c.sort(axis=0)`	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]
3
```
Select the element at the 2nd index

```
>>> b[1,2]
6.0
```
Select the element at row 1 column 2 (equivalent to `b[1][2]`)

Slicing

```
>>> a[0:2]
 array([1, 2])
```
Select items at index 0 and 1

```
>>> b[0:2,1]
 array([ 2.,  5.])
```
Select items at rows 0 and 1 in column 1

```
>>> b[:1]
 array([[1.5, 2., 3.]])
```
Select all items at row 0 (equivalent to `b[0:1, :]`)

```
>>> c[1,...]
 array([[[ 3.,  2.,  1.],
         [ 4.,  5.,  6.]]])
```
Same as `[1,:,:]`

```
>>> a[ : :-1]
 array([3, 2, 1])
```
Reversed array `a`

Boolean Indexing

```
>>> a[a<2]
 array([1])
```
Select elements from `a` less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]
 array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]
 array([[ 4. , 5. , 6. , 4. ],
        [ 1.5, 2. , 3. , 1.5],
        [ 4. , 5. , 6. , 4. ],
        [ 1.5, 2. , 3. , 1.5]])
```
Select elements `(1,0)`, `(0,1)`, `(1,2)` and `(0,0)`

Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)
>>> i.T
```
Permute array dimensions
Permute array dimensions

Changing Array Shape

```
>>> b.ravel()
```
Flatten the array

```
>>> g.reshape(3,-2)
```
Reshape, but don't change data

Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a,[1])
```
Return a new array with shape (2,6)
Append items to an array
Insert items in an array
Delete items from an array

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)
 array([ 1,  2,  3, 10, 15, 20])
```
Concatenate arrays

```
>>> np.vstack((a,b))
 array([[ 1. ,  2. ,  3. ],
        [ 1.5,  2. ,  3. ],
        [ 4. ,  5. ,  6. ]])
```
Stack arrays vertically (row-wise)

```
>>> np.r_[e,f]
```
Stack arrays vertically (row-wise)

```
>>> np.hstack((e,f))
 array([[ 7.,  7.,  1.,  0.],
        [ 7.,  7.,  0.,  1.]])
```
Stack arrays horizontally (column-wise)

```
>>> np.column_stack((a,d))
 array([[ 1, 10],
        [ 2, 15],
        [ 3, 20]])
```
Create stacked column-wise arrays

```
>>> np.c_[a,d]
```
Create stacked column-wise arrays

Splitting Arrays

```
>>> np.hsplit(a,3)
 [array([1]),array([2]),array([3])]
>>> np.vsplit(c,2)
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3. ,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]]])]
```
Split the array horizontally at the 3rd index
Split the array vertically at the 2nd index

https://assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

https://scipy.org/

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

Index Tricks

```
>>> np.mgrid[0:5,0:5]          Create a dense meshgrid
>>> np.ogrid[0:2,0:2]          Create an open meshgrid
>>> np.r_[3,[0]*5,-1:1:10j]    Stack arrays vertically (row-wise)
>>> np.c_[b,c]                 Create stacked column-wise arrays
```

Shape Manipulation

```
>>> np.transpose(b)            Permute array dimensions
>>> b.flatten()                Flatten the array
>>> np.hstack((b,c))           Stack arrays horizontally (column-wise)
>>> np.vstack((a,b))           Stack arrays vertically (row-wise)
>>> np.hsplit(c,2)             Split the array horizontally at the 2nd index
>>> np.vpsplit(d,2)            Split the array vertically at the 2nd index
```

Polynomials

```
>>> from numpy import polyld
>>> p = polyld([3,4,5])        Create a polynomial object
```

Vectorizing Functions

```
>>> def myfunc(a):
        if a < 0:
          return a*2
        else:
          return a/2
>>> np.vectorize(myfunc)       Vectorize functions
```

Type Handling

```
>>> np.real(b)                 Return the real part of the array elements
>>> np.imag(b)                 Return the imaginary part of the array elements
>>> np.real_if_close(c,tol=1000)  Return a real array if complex parts close to 0
>>> np.cast['f'](np.pi)        Cast object to a data type
```

Other Useful Functions

```
>>> np.angle(b,deg=True)       Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5)  Create an array of evenly spaced values
                                    (number of samples)
>>> g [3:] += np.pi
>>> np.unwrap(g)               Unwrap
>>> np.logspace(0,10,3)        Create an array of evenly spaced values (log scale)
>>> np.select([c<4],[c*2])     Return values from a list of arrays depending on
                               conditions
>>> misc.factorial(a)          Factorial
>>> misc.comb(10,3,exact=True)  Combine N things taken at k time
>>> misc.central_diff_weights(3)  Weights for Np-point central derivative
>>> misc.derivative(myfunc,1.0)   Find the n-th derivative of a function at a point
```

Linear Algebra

Also see NumPy

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse

```
>>> A.I                        Inverse
>>> linalg.inv(A)              Inverse
```

Transposition

```
>>> A.T                        Tranpose matrix
>>> A.H                        Conjugate transposition
```

Trace

```
>>> np.trace(A)                Trace
```

Norm

```
>>> linalg.norm(A)             Frobenius norm
>>> linalg.norm(A,1)           L1 norm (max column sum)
>>> linalg.norm(A,np.inf)      L inf norm (max row sum)
```

Rank

```
>>> np.linalg.matrix_rank(C)   Matrix rank
```

Determinant

```
>>> linalg.det(A)              Determinant
```

Solving linear problems

```
>>> linalg.solve(A,b)          Solver for dense matrices
>>> E = np.mat(a).T            Solver for dense matrices
>>> linalg.lstsq(F,E)          Least-squares solution to linear matrix
                               equation
```

Generalized inverse

```
>>> linalg.pinv(C)             Compute the pseudo-inverse of a matrix
                               (least-squares solver)
>>> linalg.pinv2(C)            Compute the pseudo-inverse of a matrix
                               (SVD)
```

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)         Create a 2X2 identity matrix
>>> G = np.mat(np.identity(2)) Create a 2x2 identity matrix
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C)   Compressed Sparse Row matrix
>>> I = sparse.csc_matrix(D)   Compressed Sparse Column matrix
>>> J = sparse.dok_matrix(A)   Dictionary Of Keys matrix
>>> E.todense()                Sparse matrix to full matrix
>>> sparse.isspmatrix_csc(A)   Identify sparse matrix
```

Sparse Matrix Routines

Inverse

```
>>> sparse.linalg.inv(I)       Inverse
```

Norm

```
>>> sparse.linalg.norm(I)      Norm
```

Solving linear problems

```
>>> sparse.linalg.spsolve(H,I) Solver for sparse matrices
```

Sparse Matrix Functions

```
>>> sparse.linalg.expm(I)      Sparse matrix exponential
```

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

Matrix Functions

Addition

```
>>> np.add(A,D)                Addition
```

Subtraction

```
>>> np.subtract(A,D)           Subtraction
```

Division

```
>>> np.divide(A,D)             Division
```

Multiplication

```
>>> A @ D                      Multiplication operator (Python 3)
>>> np.multiply(D,A)           Multiplication
>>> np.dot(A,D)                Dot product
>>> np.vdot(A,D)               Vector dot product
>>> np.inner(A,D)              Inner product
>>> np.outer(A,D)              Outer product
>>> np.tensordot(A,D)          Tensor dot product
>>> np.kron(A,D)               Kronecker product
```

Exponential Functions

```
>>> linalg.expm(A)             Matrix exponential
>>> linalg.expm2(A)            Matrix exponential (Taylor Series)
>>> linalg.expm3(D)            Matrix exponential (eigenvalue
                               decomposition)
```

Logarithm Function

```
>>> linalg.logm(A)             Matrix logarithm
```

Trigonometric Functions

```
>>> linalg.sinm(D)             Matrix sine
>>> linalg.cosm(D)             Matrix cosine
>>> linalg.tanm(A)             Matrix tangent
```

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)            Hyperbolic matrix sine
>>> linalg.coshm(D)            Hyperbolic matrix cosine
>>> linalg.tanhm(A)            Hyperbolic matrix tangent
```

Matrix Sign Function

```
>>> np.signm(A)                Matrix sign function
```

Matrix Square Root

```
>>> linalg.sqrtm(A)            Matrix square root
```

Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x)  Evaluate matrix function
```

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)      Solve ordinary or generalized
                               eigenvalue problem for square matrix
>>> l1, l2 = la                Unpack eigenvalues
>>> v[:,0]                     First eigenvector
>>> v[:,1]                     Second eigenvector
>>> linalg.eigvals(A)          Unpack eigenvalues
```

Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)     Singular Value Decomposition (SVD)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N)  Construct sigma matrix in SVD
```

LU Decomposition

```
>>> P,L,U = linalg.lu(C)       LU Decomposition
```

Sparse Matrix Decompositions

```
>>> la, v = sparse.linalg.eigs(F,1)  Eigenvalues and eigenvectors
>>> sparse.linalg.svds(H, 2)         SVD
```

https://www.datacamp.com

pandas

About us | Getting started | Documentation | Community | Contribute

pandas

pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

Install pandas now!

Getting started

- Install pandas
- Getting started
- Try pandas online

Documentation

- User guide
- API reference
- Contributing to pandas
- Release notes

Community

- About pandas
- Ask a question
- Ecosystem

With the support of:

NVIDIA

bodo.ai

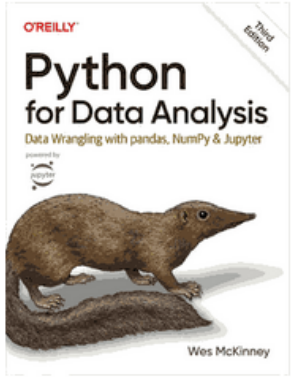
The full list of companies supporting *pandas* is available in the sponsors page.

Latest version: 2.3.1

- What's new in 2.3.1
- Release date: Jul 07, 2025
- Documentation (web)
- Download source code

Follow us

Recommended books



https://pandas.pydata.org/

Data Wrangling

with pandas Cheat Sheet
http://pandas.pydata.org

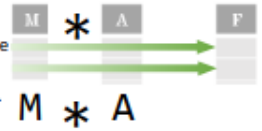
Pandas API Reference Pandas User Guide

Tidy Data – A foundation for wrangling in pandas

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4, 5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```
Specify values for each row.

		a	b	c
N	v			
D	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d', 1), ('d', 2),
         ('e', 2)], names=['n', 'v']))
```
Create DataFrame with a MultiIndex

Method Chaining

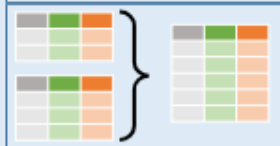
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable':'var',
                'value':'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change layout, sorting, reindexing, renaming

pd.melt(df)
Gather columns into rows.

df.pivot(columns='var', values='val')
Spread rows into columns.



pd.concat([df1,df2])
Append rows of DataFrames

pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg', ascending=False)
Order rows by values of a column (high to low).

df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length', 'Height'])
Drop columns from DataFrame

Subset Observations - rows

```
df[df.Length > 7]
```
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10) Randomly select n rows.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

Subset Variables - columns

```
df[['width', 'length', 'species']]
```
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='regex')
Select columns whose name matches regular expression *regex*.

Using query

query() allows Boolean expressions for filtering rows.

```
df.query('Length > 7')
df.query('Length > 7 and Width < 8')
df.query('Name.str.startswith("abc")',
         engine="python")
```

Subsets - rows and columns

Use **df.loc[]** and **df.iloc[]** to select only rows, only columns or both.
Use **df.at[]** and **df.iat[]** to access a single value by row and column.
First index selects rows, second index columns.

df.iloc[10:20]
Select rows 10-20.

df.iloc[:, [1, 2, 5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[:, 'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.loc[df['a'] > 10, ['a', 'c']]
Select rows meeting logical condition, and only the specific columns .

df.iat[1, 2] Access single value by index

df.at[4, 'A'] Access single value by label

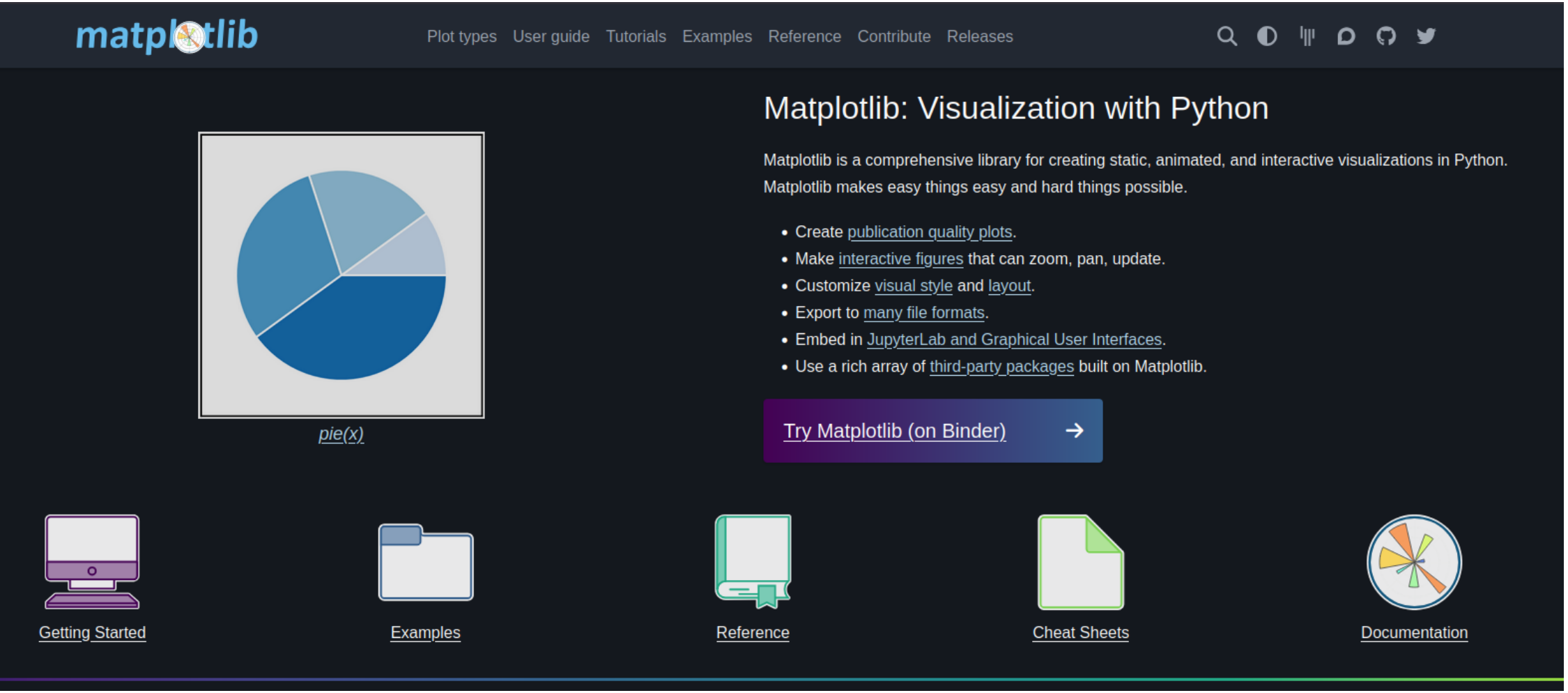
Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	df.column.isin(values)	Group membership
==	Equals	pd.isnull(obj)	Is NaN
<=	Less than or equals	pd.notnull(obj)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

regex (Regular Expressions) Examples

'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf



https://matplotlib.org/

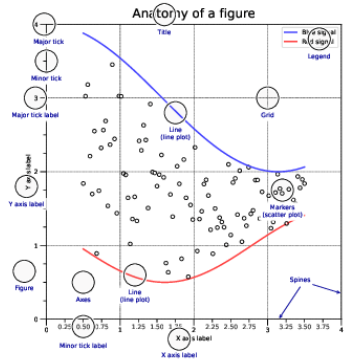
matplotlib Cheat sheet

Version 3.7.4

Quick start

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
plt.show()
```

Anatomy of a figure



Subplots layout

```
subplot[s](rows, cols, …)
fig, axs = plt.subplots(3, 3)
```

```
G = gridspec(rows,cols, …)
ax = G[0, :]
```

```
ax.inset_axes(extent)
```

```
d=make_axes_locatable(ax)
ax = d.new_horizontal('10%')
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- https://gitter.im/matplotlib/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

```
plot([X], Y, [fmt], …)
```
X, Y, fmt, color, marker, linestyle

```
scatter(X, Y, …)
```
X, Y, [s]izes, [c]olors, marker, cmap

```
bar[h](x, height, …)
```
x, height, width, bottom, align, color

```
imshow(Z, …)
```
Z, cmap, interpolation, extent, origin

```
contour[f]([X], [Y], Z, …)
```
X, Y, Z, levels, colors, extent, origin

```
pcolormesh([X], [Y], Z, …)
```
X, Y, Z, vmin, vmax, cmap

```
quiver([X], [Y], U, V, …)
```
X, Y, U, V, C, units, angles

```
pie(X, …)
```
Z, explode, labels, colors, radius


```
text(x, y, text, …)
```
x, y, text, va, ha, size, weight, transform

```
fill[_between][x](…)
```
X, Y1, Y2, color, where

Advanced plots

```
step(X, Y, [fmt], …)
```
X, Y, fmt, color, marker, where

```
boxplot(X, …)
```
X, notch, sym, bootstrap, widths

```
errorbar(X,Y,xerr,yerr, …)
```
X, Y, xerr, yerr, fmt

```
hist(X, bins, …)
```
X, bins, range, density, weights

```
violinplot(D, …)
```
D, positions, widths, vert

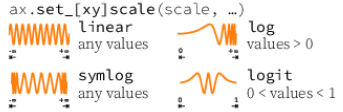
```
barbs([X], [Y], U, V, …)
```
X, Y, U, V, C, length, pivot, sizes

```
eventplot(positions, …)
```
positions, orientation, lineoffsets

```
hexbin(X, Y, C, …)
```
X, Y, C, gridsize, bins

Scales

```
ax.set_[xy]scale(scale, …)
```



Projections

```
subplot(…, projection=p)
```

p='polar'

p='3d'

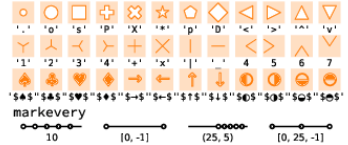
```
p=ccrs.Orthographic()
import cartopy.crs as ccrs
```

Lines

```
linestyle or ls
```


```
capstyle or dash_capstyle
```

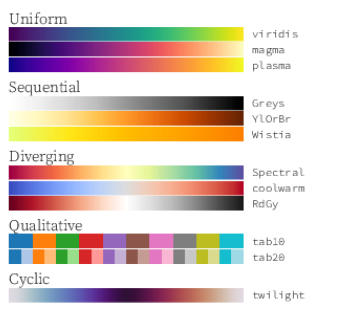
Markers



Colors

Colormaps

```
plt.get_cmap(name)
```



Tick locators

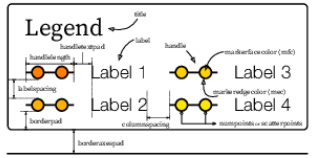
```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
ticker.NullLocator()
ticker.MultipleLocator(0.5)
ticker.FixedLocator([0, 1, 5])
ticker.LinearLocator(numticks=3)
ticker.IndexLocator(base=0.5, offset=0.25)
ticker.AutoLocator()
ticker.MaxNLocator(n=4)
ticker.LogLocator(base=10, numticks=15)
```

Tick formatters

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
ticker.NullFormatter()
ticker.FixedFormatter(['zero', 'one', 'two', …])
ticker.FuncFormatter(lambda x, pos: "[%.2f]" % x)
ticker.FormatStrFormatter('>%d<')
ticker.ScalarFormatter()
ticker.StrMethodFormatter('{x}')
ticker.PercentFormatter(xmax=5)
```

Ornaments

```
ax.legend(…)
```
handles, labels, loc, title, frameon




```
ax.colorbar(…)
```
mappable, ax, cax, orientation

```
ax.annotate(…)
```
text, xy, xytext, xycoords, textcoords, arrowprops

Event handling

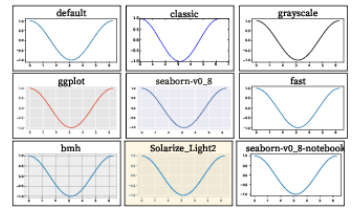
```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

```
plt.style.use(style)
```



Quick reminder

```
ax.grid()
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(ticks, [labels])
ax.set_[xy]ticklabels(labels)
ax.set_title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

fig.suptitle(title)
fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
[fig|ax].patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

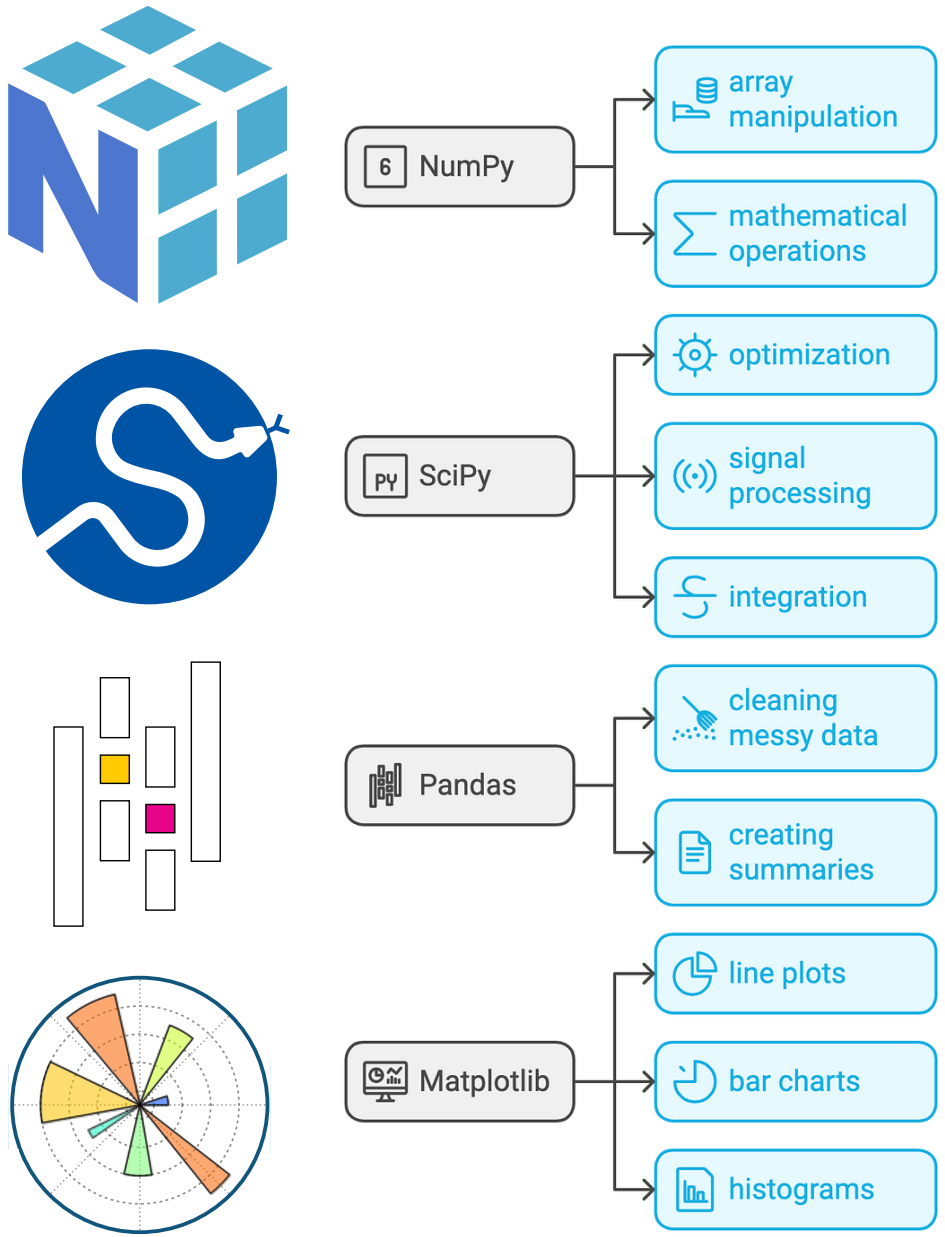
Keyboard shortcuts

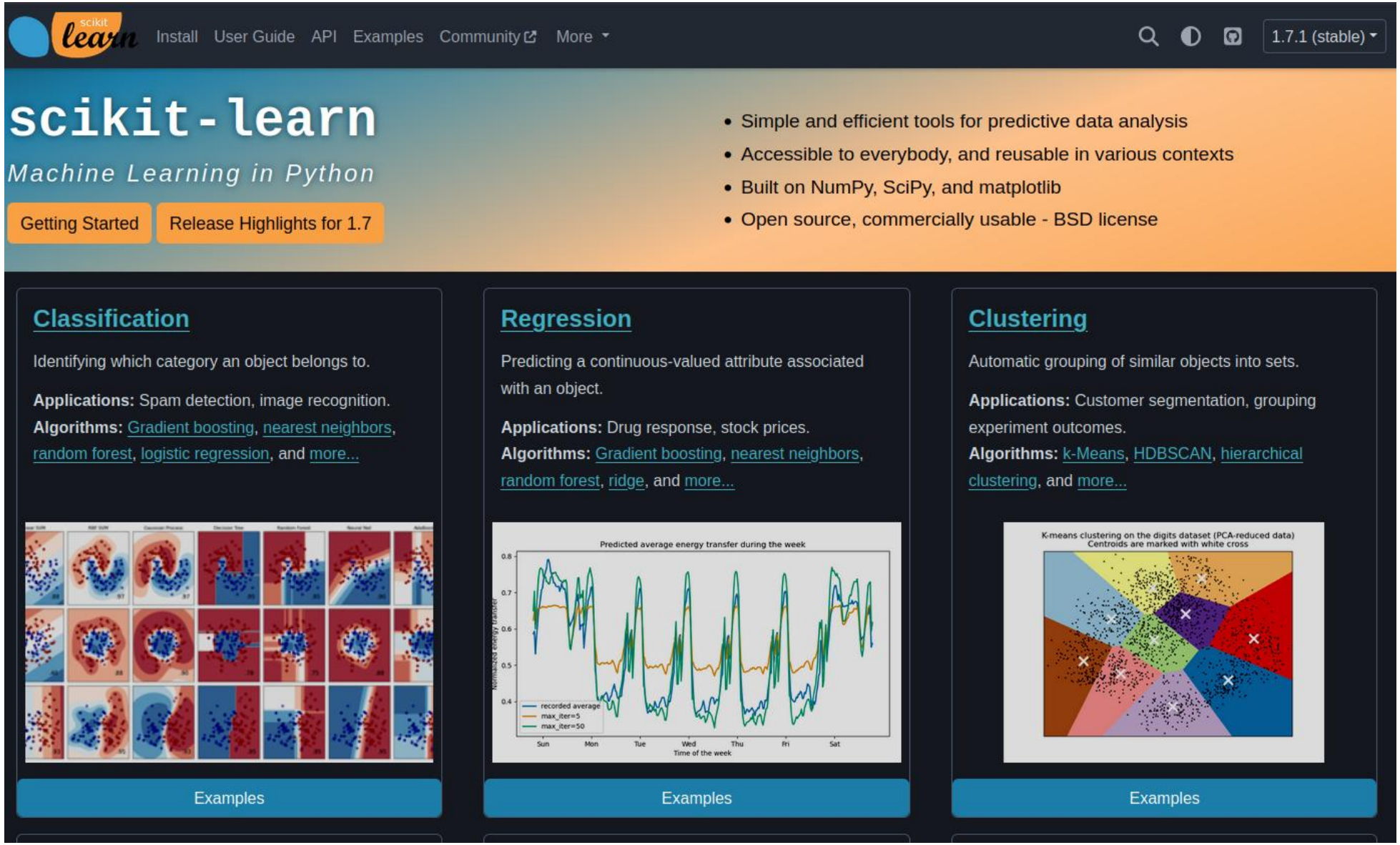
ctrl + s Save	ctrl + w Close plot
r Reset view	f Fullscreen 0/1
f View forward	b View back
p Pan view	o Zoom to rect
x X pan/zoom	y Y pan/zoom
g Minor grid 0/1	G Major grid 0/1
l X axis log/linear	L Y axis log/linear

Ten simple rules

1. Know your audience
2. Identify your message
3. Adapt the figure
4. Captions are not optional
5. Do not trust the defaults
6. Use color effectively
7. Do not mislead the reader
8. Avoid "chartjunk"
9. Message trumps beauty
10. Get the right tool

https://matplotlib.org/cheatsheets/


NumPy
array manipulation
mathematical operations
SciPy
optimization
signal processing
integration
Pandas
cleaning messy data
creating summaries
Matplotlib
line plots
bar charts
histograms



https://scikit-learn.org

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```
Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```
Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

https://www.datacamp.com

etting started Gallery Installation API reference What's new

Search Ctrl + K

Introduction

Cartopy is a Python package designed for geospatial data processing in order to produce maps and other geospatial data analyses.

Cartopy makes use of the powerful PROJ, NumPy and Shapely libraries and includes a programmatic interface built on top of Matplotlib for the creation of publication quality maps.

Key features of cartopy are its object oriented projection definitions, and its ability to transform points, lines, vectors, polygons and images between those projections.

You will find cartopy especially useful for large area / small scale data, where Cartesian assumptions of spherical data traditionally break down. If you've ever experienced a singularity at the pole or a cut-off at the dateline, it is likely you will appreciate cartopy's unique features!

Note

The v0.20 release uses pyproj for transformations, which could be slower in some situations. If you need to increase the speed of plots and don't need to worry about thread safety in your application, you can set the environment variable `PYPROJ_GLOBAL_CONTEXT=ON` to make the projection calculations faster.

On this page

Introduction

Getting involved

Show Source

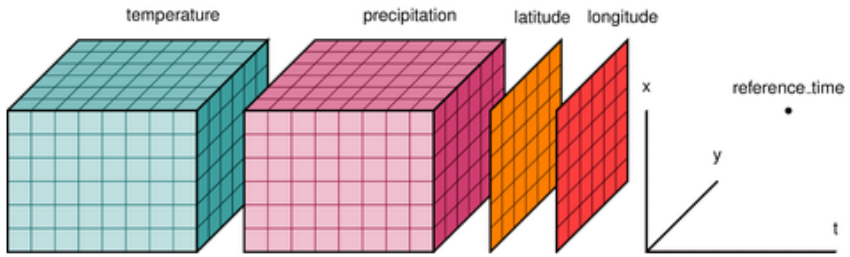
https://scitools.org.uk/cartopy/docs/latest/index.html

xarray

Features Try Ecosystem Sponsors Donate Blog Documentation

Xarray

N-D labeled arrays and datasets in Python



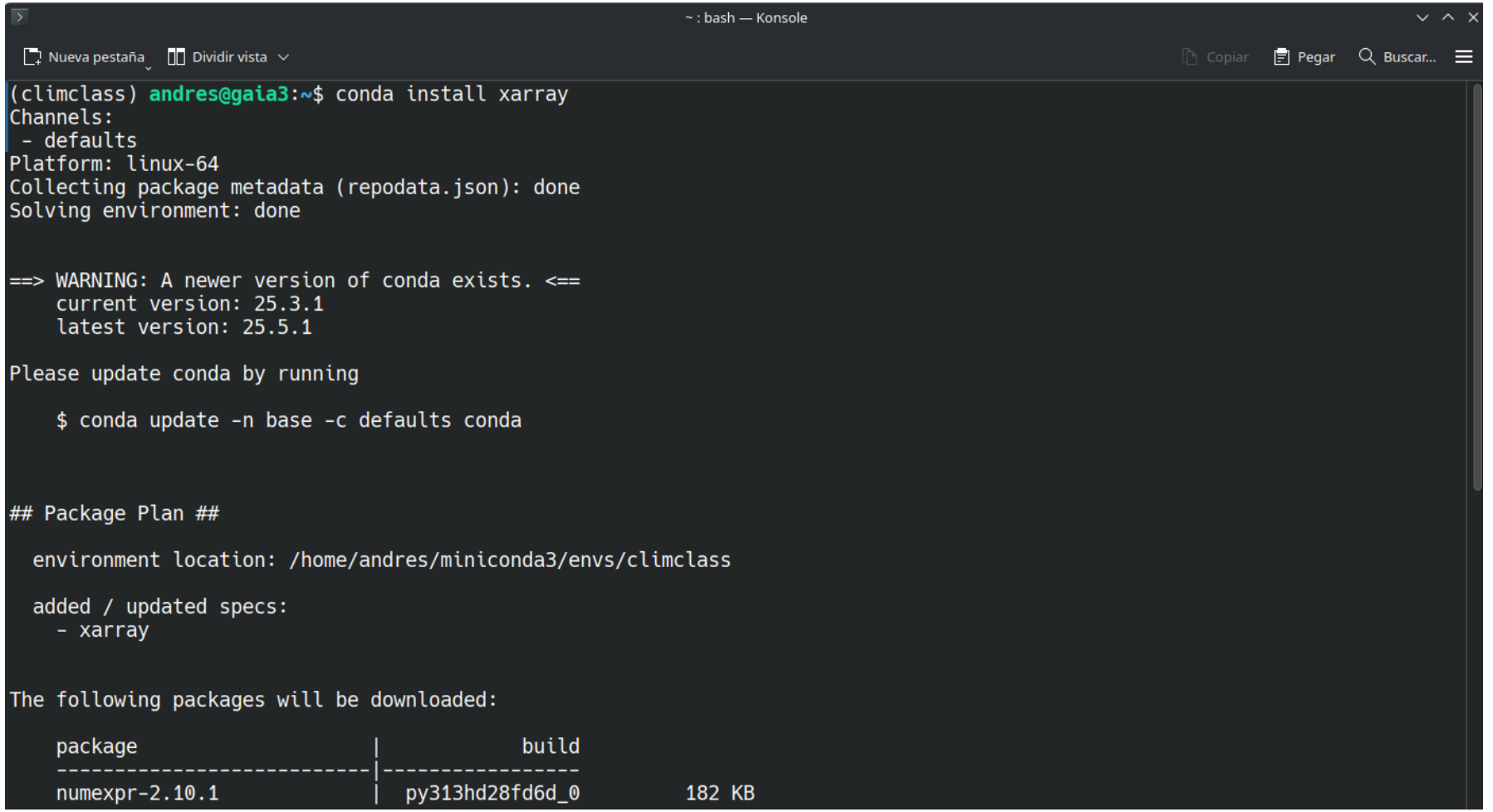
Xarray is an open source project and Python package that introduces labels in the form of dimensions, coordinates, and attributes on top of raw NumPy-like arrays, which allows for more intuitive, more concise, and less error-prone user experience.

Xarray includes a large and growing library of domain-agnostic functions for advanced analytics and visualization with these data structures.

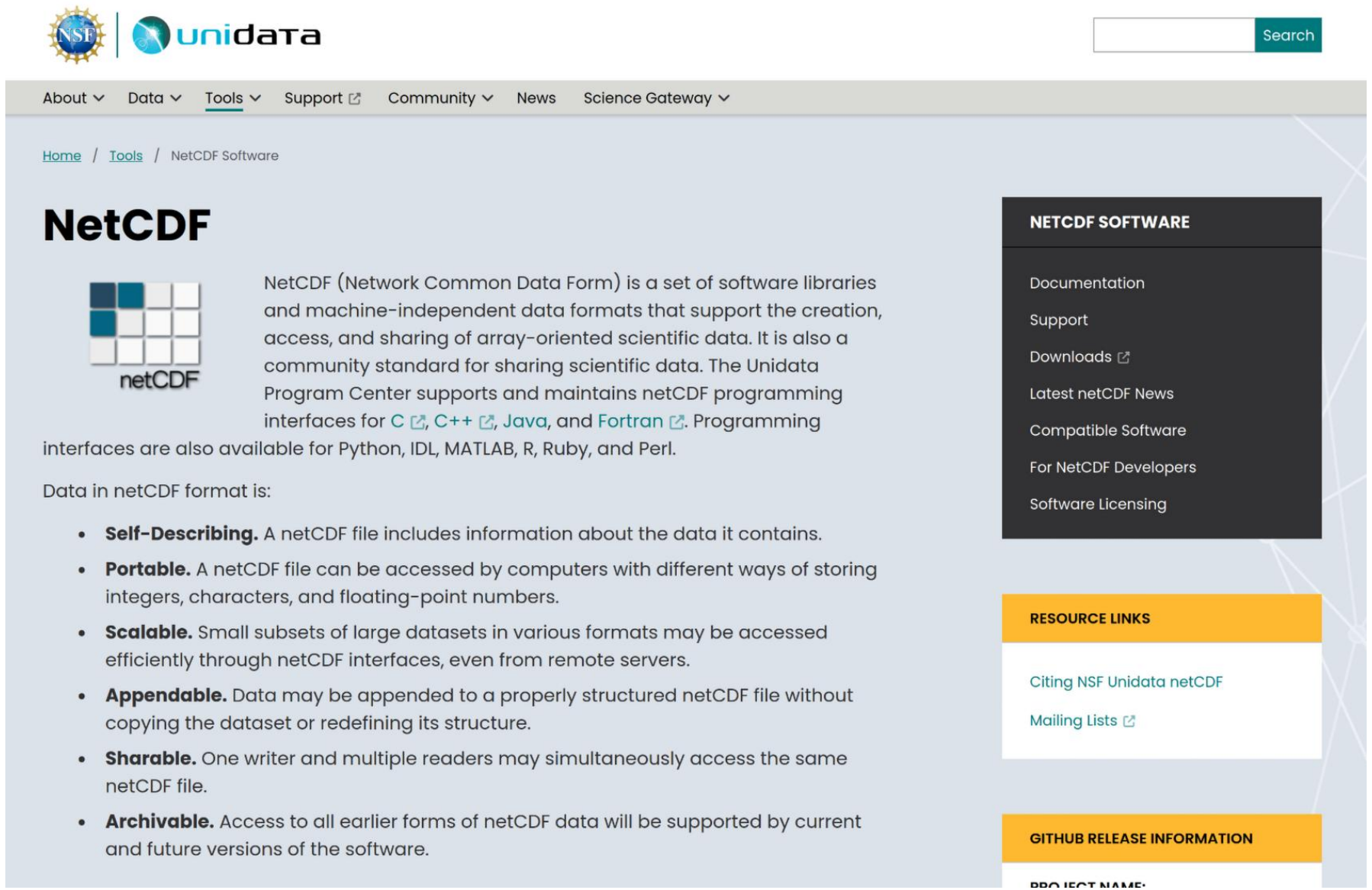
Get Started ↗ Why Xarray? ↗

https://xarray.dev/

Xarray

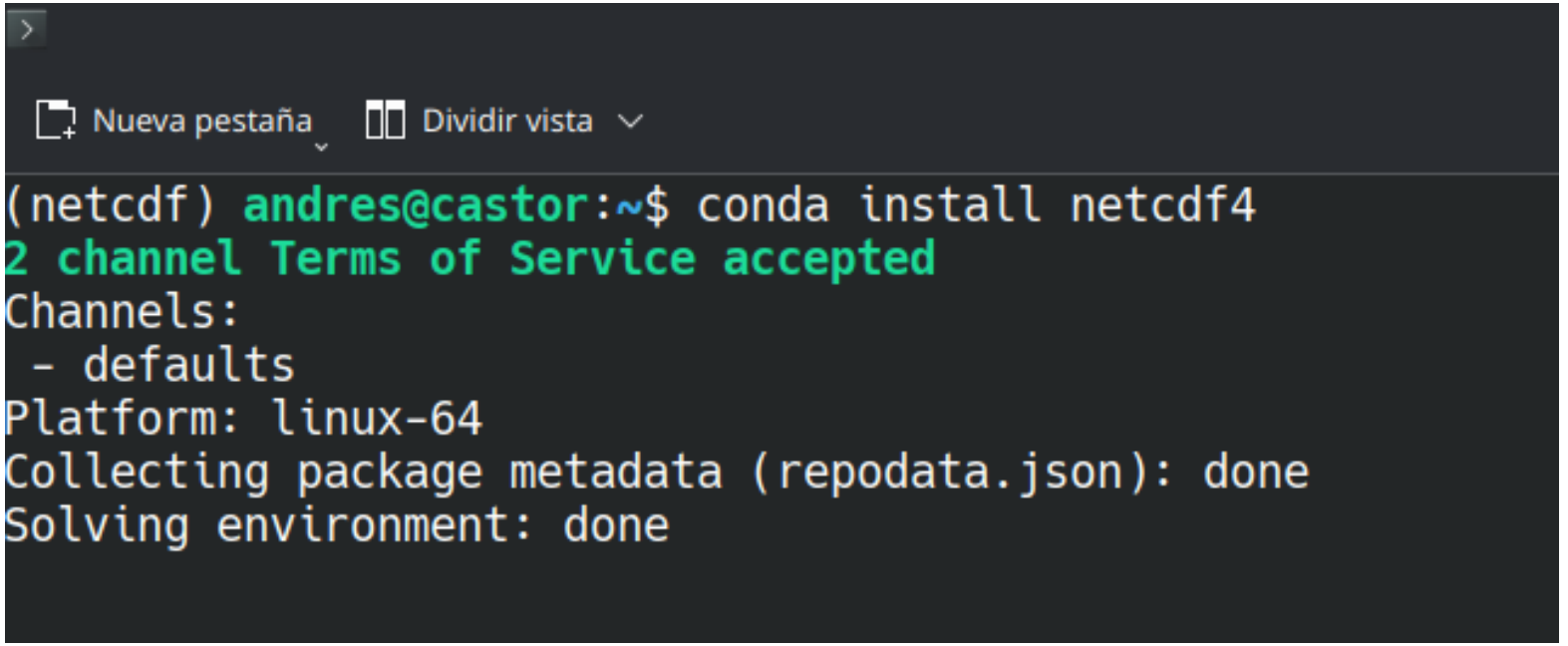

~ : bash — Konsole
Nueva pestaña
Dividir vista
Copiar
Pegar
Buscar...
(climclass) andres@gaia3:~$ conda install xarray
Channels:
 - defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done
==> WARNING: A newer version of conda exists. <==
    current version: 25.3.1
    latest version: 25.5.1
Please update conda by running
    $ conda update -n base -c defaults conda
Package Plan
  environment location: /home/andres/miniconda3/envs/climclass
  added / updated specs:
    - xarray
The following packages will be downloaded:
    package | build
    numexpr-2.10.1 | py313hd28fd6d_0   182 KB

netcdf4



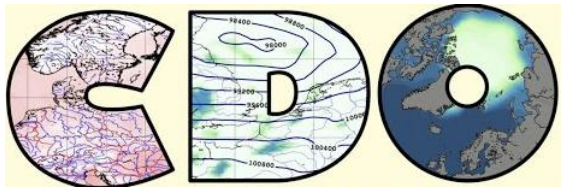
https://www.unidata.ucar.edu/software/netcdf

netcdf4



conda install netcdf4

CDO **C**limate **D**ata **O**perators



https://code.mpimet.mpg.de/projects/cdo

Inicio Proyectos Imprint + Privacy Policy Ayuda — Iniciar sesión Registrar

Max-Planck-Institut für Meteorologie **CDO**

Búsqueda: CDO

Vistazo Actividad Noticias Wiki Foros Archivos Documentation

Vistazo

Climate Data Operators

CDO is a collection of command line Operators to manipulate and analyse Climate and NWP model Data.
Supported data formats are GRIB 1/2, netCDF 3/4, SERVICE, EXTRA and IEG. There are more than 600 operators available.

- Documentation
- FAQ
- Downloads
- Community

Please register yourself for reporting bugs or postings

Últimas noticias

Version 2.5.1 released
Añadido por Uwe Schulzweida hace alrededor de 2 meses

Version 2.5.0 released
Añadido por Uwe Schulzweida hace 4 meses

Version 2.4.4 released
Añadido por Uwe Schulzweida hace 7 meses

Version 2.4.3 released
Añadido por Uwe Schulzweida hace 8 meses

Version 2.4.2 released
Añadido por Uwe Schulzweida hace 10 meses

Ver todas las noticias

Miembros

Manager: Luis Kornblueh, Ralf Mueller, Uwe Schulzweida

Developer: Dian Putrasahan, Fabian Wachsmann, Irina Fast, Joerg Behrens, Karin Meier-Fleischer, Karl-Hermann Wieners, Mathis Rosenhauer, Oliver Heidmann, Ralf Quast, Thomas Jahns

Reporter: Brendan DeTracey, David Gobbett, Didier Monselesan, Frank Kaspar, Jan Sellmann, Jin-Song von Storch, Stefan Fronzek, William Sawyer

Moderator: Estanislao Gavilan, Karin Meier-Fleischer

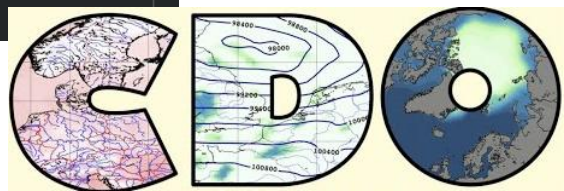
https://code.mpimet.mpg.de/projects/cdo

~ : bash — Konsole

Nueva pestaña | Dividir vista | Copiar | Pegar | Buscar

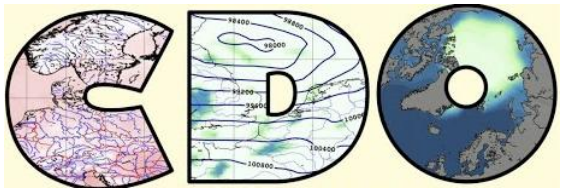
```
(climclass) andres@gaia3:~$ conda install conda-forge::cdo
Channels:
 - defaults
 - conda-forge
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/andres/miniconda3/envs/climclass

  added / updated specs:
    - conda-forge::cdo


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    blas-1.0                   |        openblas          46 KB
    blosc-1.21.6               |      he440d0b_1          47 KB  conda-forge
    c-ares-1.34.5              |      hb9d3cd8_0         202 KB  conda-forge
    cairo-1.18.4               |      h3394656_0         955 KB  conda-forge
    cdo-2.4.4                  |      h1f03bf2_2        58.1 MB  conda-forge
    eccodes-2.39.0             |      hf413ef6_1         4.2 MB  conda-forge
    expat-2.6.4                |      h6a678d5_0         180 KB
```

conda install conda-forge::cdo



CDO Cheatsheet

https://code.mpimet.mpg.de/projects/cdo/wiki/tutorial



CDO Reference Card

Climate Data Operators
Version 1.0.8
June 2007

Uwe Schulzweida
Max-Planck-Institute for Meteorology

Syntax

cdo [Options] Operators

Options

-a	Convert from a relative to an absolute time axis
-b <*nbits*>	Set the number of bits for the output precision (32/64 for nc, nc2, srv, ext, ieg; 1 - 32 for grb)
-f <*format*>	Output file format (grb, nc, nc2, srv, ext, ieg)
-g <*grid*>	Grid name or file Available grids: t<RES>grid, r<NX>x<NY>
-h	Help information for the operators
-m <*missval*>	Set the default missing value (default: -9e+33)
-R	Convert GRIB data from reduced to regular grid
-r	Convert from an absolute to a relative time axis
-t <*table*>	Set the parameter table name or file Predefined tables: echam4 echam5 mpiom1
-V	Print the version number
-v	Print extra details for some operators

Operators

Information

info	Dataset information listed by code number
infov	Dataset information listed by variable name
map	Dataset information and simple map
Syntax	<*operator*> ifiles
sinfo	Short dataset information listed by code number
sinfov	Short dataset information listed by variable name
Syntax	<*operator*> ifile
diff	Compare two datasets listed by code number
diffv	Compare two datasets listed by variable name
Syntax	<*operator*> ifile1 ifile2
npar	Number of parameters
nlevel	Number of levels
nyear	Number of years
nmon	Number of months
ndate	Number of dates
ntime	Number of time steps
Syntax	<*operator*> ifile
showformat	Show file format
showcode	Show code numbers
showname	Show variable names
showstdname	Show standard names
showlevel	Show levels
showltype	Show GRIB level types
showyear	Show years
showmon	Show months
showdate	Show dates
showtime	Show time steps
Syntax	<*operator*> ifile
pardes	Parameter description
griddes	Grid description
vct	Vertical coordinate table
Syntax	<*operator*> ifile

File operations

copy	Copy datasets
cat	Concatenate datasets
Syntax	<*operator*> ifiles ofile
replace	Replace variables
Syntax	replace ifile1 ifile2 ofile
merge	Merge datasets with different fields
mergetime	Merge datasets sorted by date and time
Syntax	<*operator*> ifiles ofile
splitcode	Split code numbers
splitname	Split variable names
splitlevel	Split levels
splitgrid	Split grids
splitzaxis	Split zaxis
Syntax	<*operator*> ifile oprefix
splithour	Split hours
splitday	Split days
splitmon	Split months
splitseas	Split seasons
splityear	Split years
Syntax	<*operator*> ifile oprefix
splitsel	Split time selection
Syntax	splitsel,*nsets*[,*noffset*[,*nskip*]] ifile oprefix

Selection

selcode	Select variables by code number
delcode	Delete variables by code number
Syntax	<*operator*>,*codes* ifile ofile
selname	Select variables by name
delname	Delete variables by name
Syntax	<*operator*>,*vars* ifile ofile
selstdname	Select variables by standard name
Syntax	selstdname,*stdnames* ifile ofile
sellevel	Select levels
Syntax	sellevel,*levels* ifile ofile
selgrid	Select grids
Syntax	selgrid,*grids* ifile ofile
selgridname	Select grids by name
Syntax	selgridname,*gridnames* ifile ofile
selzaxis	Select zaxes
Syntax	selzaxis,*zaxes* ifile ofile
selzaxisname	Select zaxes by name
Syntax	selzaxisname,*zaxisnames* ifile ofile
selltype	Select GRIB level types
Syntax	selltype,*ltypes* ifile ofile
seltabnum	Select parameter table numbers
Syntax	seltabnum,*tabnums* ifile ofile
seltimestep	Select time steps
Syntax	seltimestep,*timesteps* ifile ofile
seltime	Select times
Syntax	seltime,*times* ifile ofile
selhour	Select hours
Syntax	selhour,*hours* ifile ofile
selday	Select days
Syntax	selday,*days* ifile ofile
selmon	Select months
Syntax	selmon,*months* ifile ofile
selyear	Select years
Syntax	selyear,*years* ifile ofile
selseas	Select seasons
Syntax	selseas,*seasons* ifile ofile
seldate	Select dates
Syntax	seldate,*date1*[,*date2*] ifile ofile
selsmon	Select single month
Syntax	selsmon,*month*[,*nts1*[,*nts2*]] ifile ofile
sellonlatbox	Select a longitude/latitude box
Syntax	sellonlatbox,*lon1,lon2,lat1,lat2* ifile ofile
selindexbox	Select an index box
Syntax	selindexbox,*idx1,idx2,idy1,idy2* ifile ofile

Conditional selection

ifthen	If then
ifnotthen	If not then
Syntax	<*operator*> ifile1 ifile2 ofile
ifthenelse	If then else
Syntax	ifthenelse ifile1 ifile2 ifile3 ofile
ifthenc	If then constant
ifnotthenc	If not then constant
Syntax	<*operator*>,*c* ifile ofile

Comparison

eq	Equal
ne	Not equal
le	Less equal
lt	Less than
ge	Greater equal
gt	Greater than
Syntax	<*operator*> ifile1 ifile2 ofile
eqc	Equal constant
nec	Not equal constant
lec	Less equal constant
ltc	Less then constant
gec	Greater equal constant
gtc	Greater then constant
Syntax	<*operator*>,*c* ifile ofile

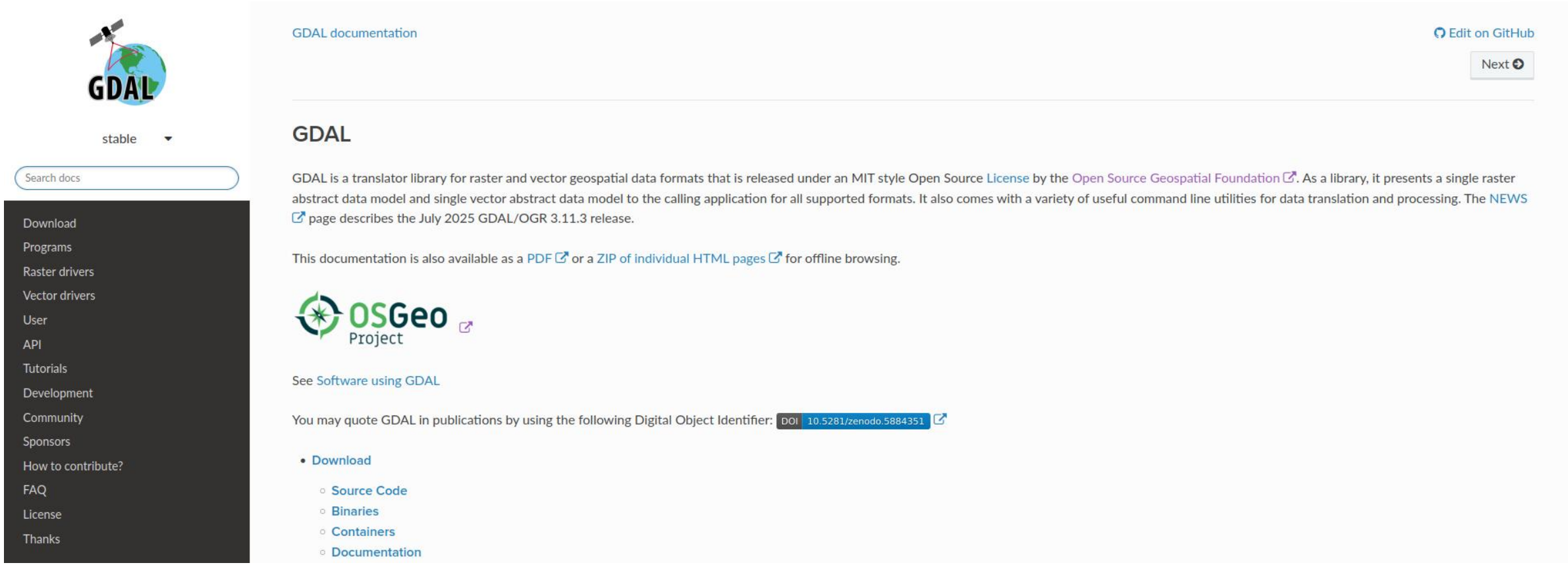
Modification

setpartab	Set parameter table
Syntax	setpartab,*table* ifile ofile
setcode	Set code number
Syntax	setcode,*code* ifile ofile
setname	Set variable name
Syntax	setname,*name* ifile ofile
setlevel	Set level
Syntax	setlevel,*level* ifile ofile
setltype	Set GRIB level type
Syntax	setltype,*ltype* ifile ofile
setdate	Set date
Syntax	setdate,*date* ifile ofile
settime	Set time
Syntax	settime,*time* ifile ofile
setday	Set day
Syntax	setday,*day* ifile ofile
setmon	Set month
Syntax	setmon,*month* ifile ofile
setyear	Set year
Syntax	setyear,*year* ifile ofile
settunits	Set time units
Syntax	settunits,*units* ifile ofile
settaxis	Set time axis
Syntax	settaxis,*date,time*[,*inc*] ifile ofile
setreftime	Set reference time
Syntax	setreftime,*date,time* ifile ofile
setcalendar	Set calendar
Syntax	setcalendar,*calendar* ifile ofile
shifttime	Shift time steps
Syntax	shifttime,*sval* ifile ofile
chcode	Change code number
Syntax	chcode,*oldcode,newcode*[,...] ifile ofile
chname	Change variable name
Syntax	chname,*ovar,nvar*,... ifile ofile
chlevel	Change level
Syntax	chlevel,*oldlev,newlev*,... ifile ofile
chlevelc	Change level of one code
Syntax	chlevelc,*code,oldlev,newlev* ifile ofile
chlevelv	Change level of one variable
Syntax	chlevelv,*var,oldlev,newlev* ifile ofile
setgrid	Set grid
Syntax	setgrid,*grid* ifile ofile
setgridtype	Set grid type
Syntax	setgridtype,*gridtype* ifile ofile
setzaxis	Set zaxis
Syntax	setzaxis,*zaxis* ifile ofile
setgatt	Set global attribute
Syntax	setgatt,*attname,attstring* ifile ofile
setgatts	Set global attributes
Syntax	setgatts,*attfile* ifile ofile
invertlat	Invert latitude
invertlon	Invert longitude
invertlatdes	Invert latitude description
invertlondes	Invert longitude description
invertlatdata	Invert latitude data
invertlondata	Invert longitude data
Syntax	<*operator*> ifile ofile
maskregion	Mask regions
Syntax	maskregion,*regions* ifile ofile
masklonlatbox	Mask a longitude/latitude box
Syntax	masklonlatbox,*lon1,lon2,lat1,lat2* ifile ofile
maskindexbox	Mask an index box
Syntax	maskindexbox,*idx1,idx2,idy1,idy2* ifile ofile
setclonlatbox	Set a longitude/latitude box to constant
Syntax	setclonlatbox,*c,lon1,lon2,lat1,lat2* ifile ofile
setcindexbox	Set an index box to constant
Syntax	setcindexbox,*c,idx1,idx2,idy1,idy2* ifile ofile
enlarge	Enlarge fields
Syntax	enlarge,*grid* ifile ofile
setmissval	Set a new missing value
Syntax	setmissval,*miss* ifile ofile
setctomiss	Set constant to missing value
setmisstoc	Set missing value to constant
Syntax	<*operator*>,*c* ifile ofile
setrtomiss	Set range to missing value
Syntax	setrtomiss,*rmin,rmax* ifile ofile

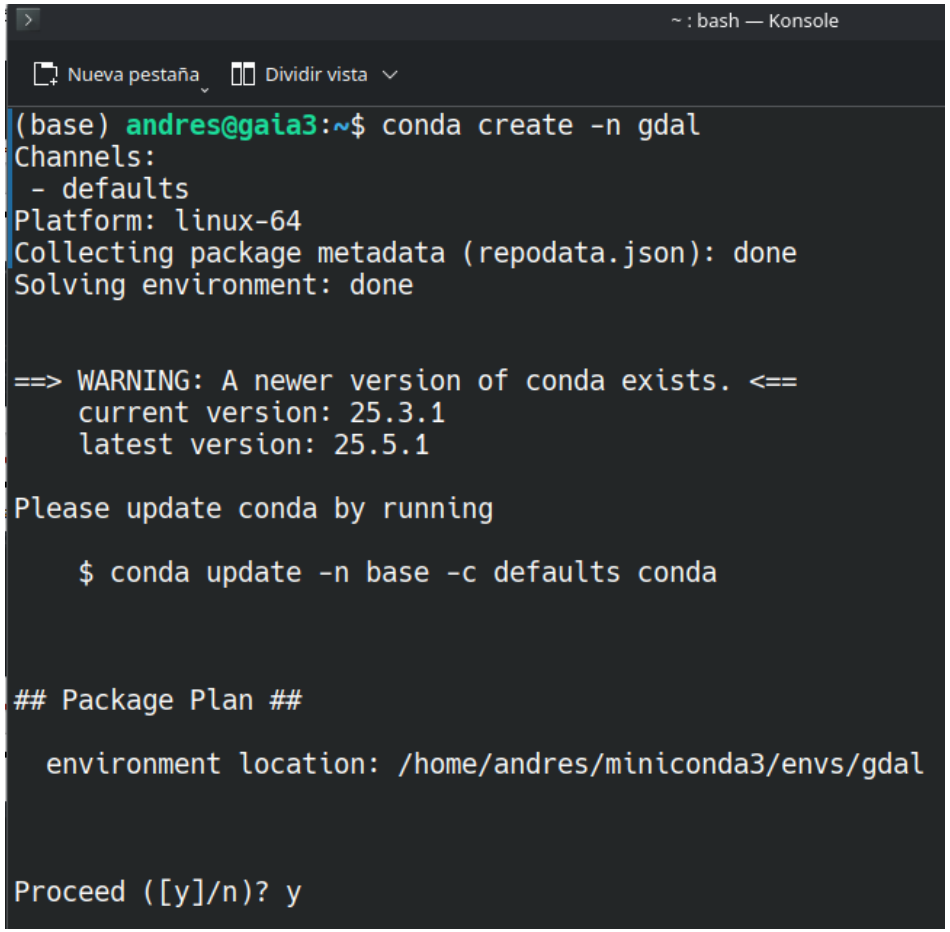
Arithmetic

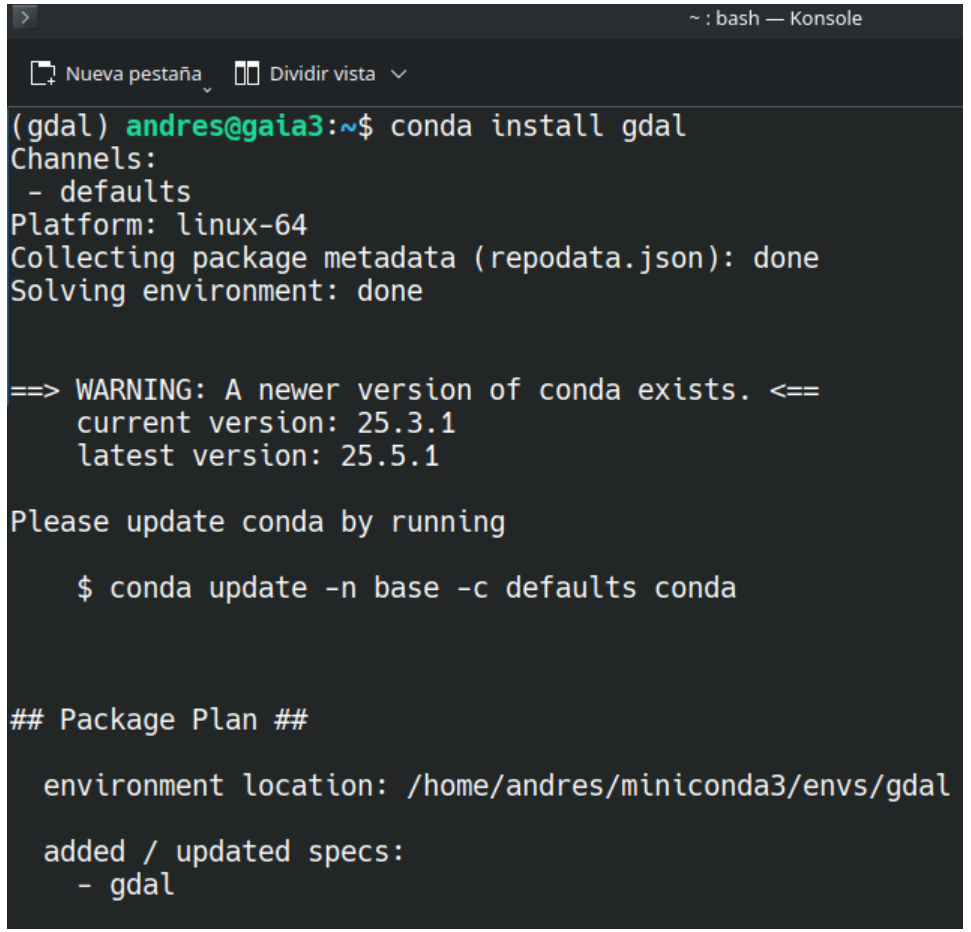
expr	Evaluate expressions
Syntax	expr,*instr* ifile ofile
exprf	Evaluate expressions from script file
Syntax	exprf,*filename* ifile ofile
abs	Absolute value
int	Integer value
nint	Nearest integer value
sqr	Square
sqrt	Square root
exp	Exponential
ln	Natural logarithm
log10	Base 10 logarithm
sin	Sine
cos	Cosine
tan	Tangent
asin	Arc sine
acos	Arc cosine
atan	Arc tangent
Syntax	<*operator*> ifile ofile

Geospatial **D**ata **L**ibrary **A**bstraction



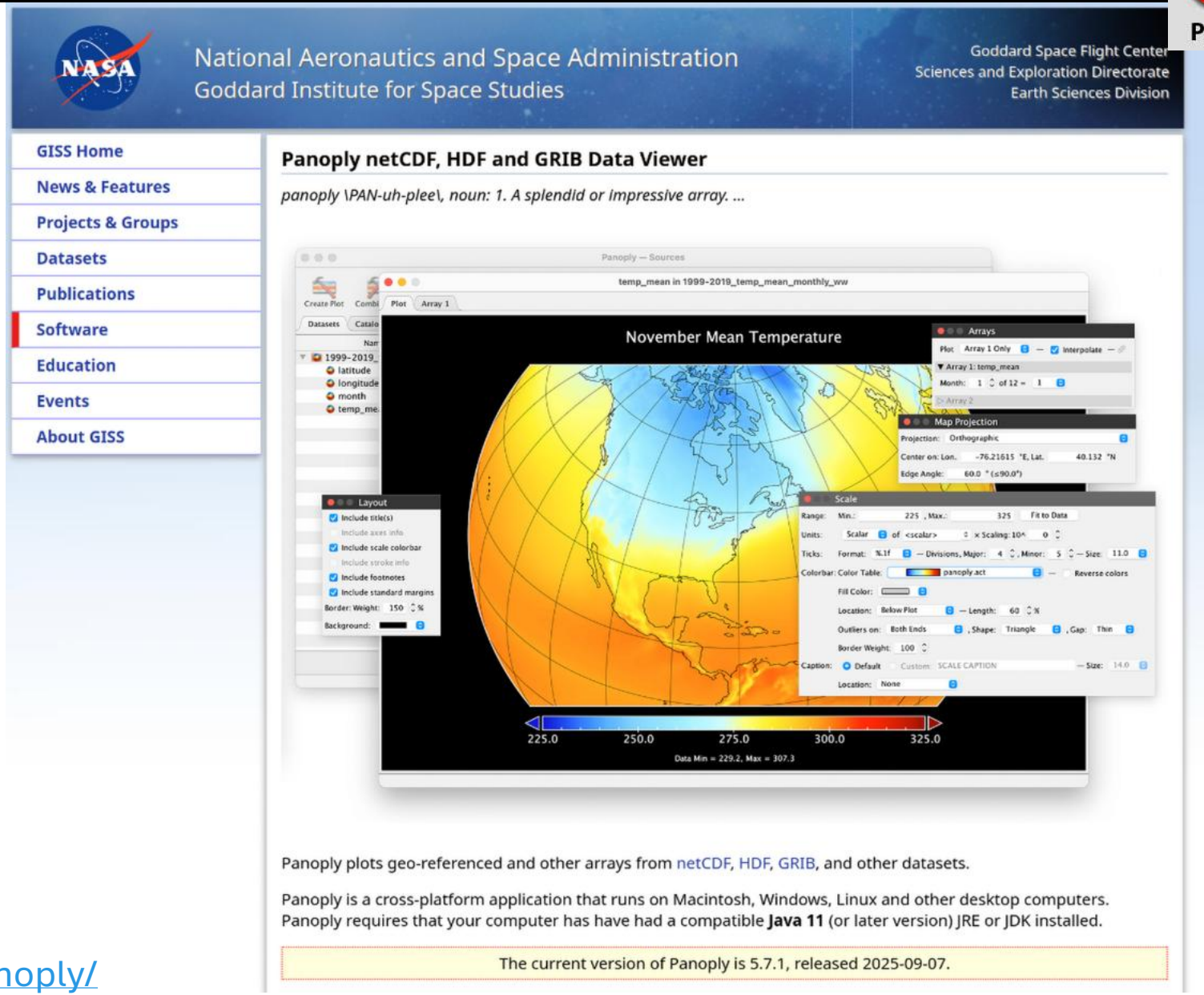
https://gdal.org/en/stable/





conda create –n gdal → conda activate gdal → conda install gdal

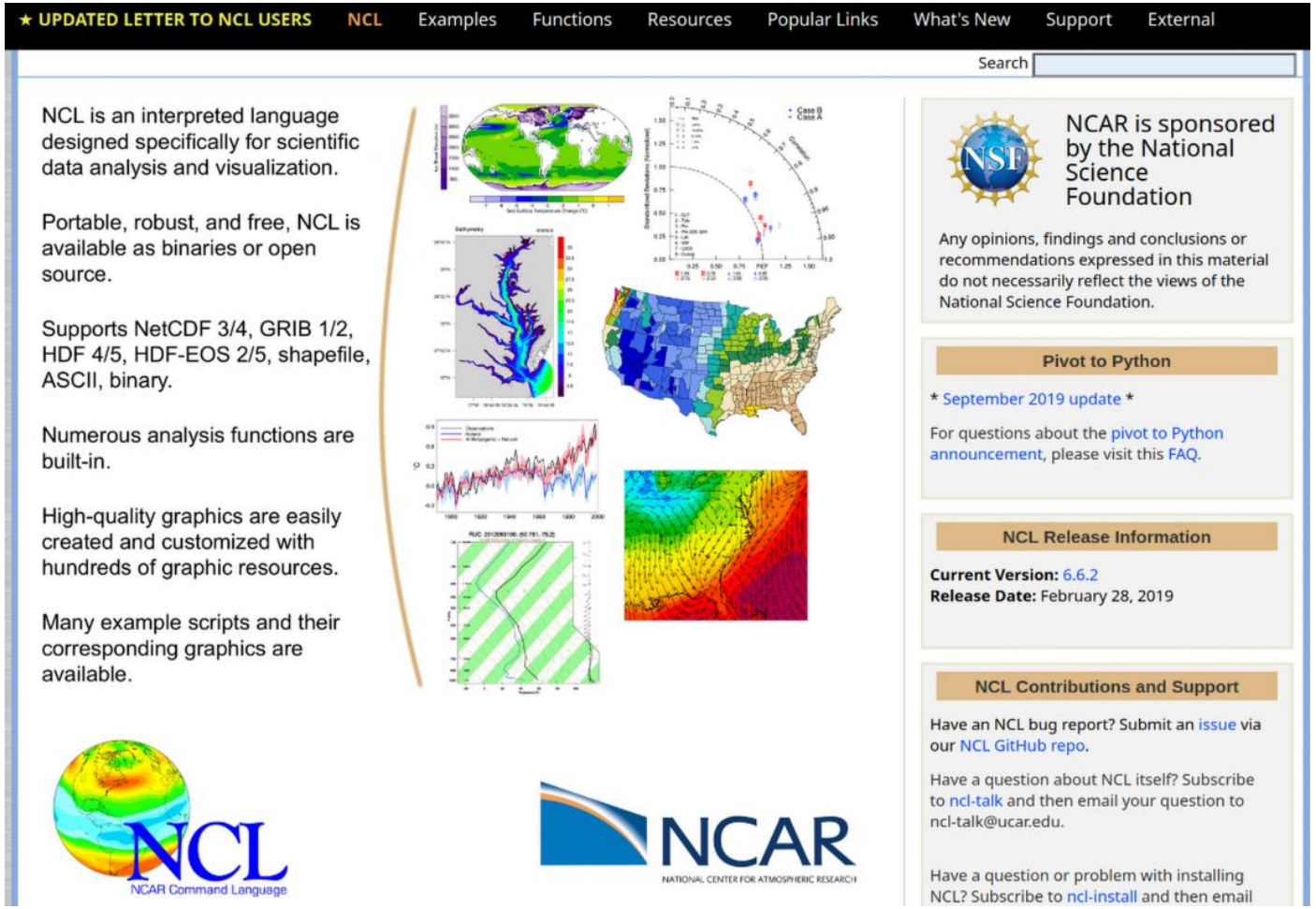
Panoply



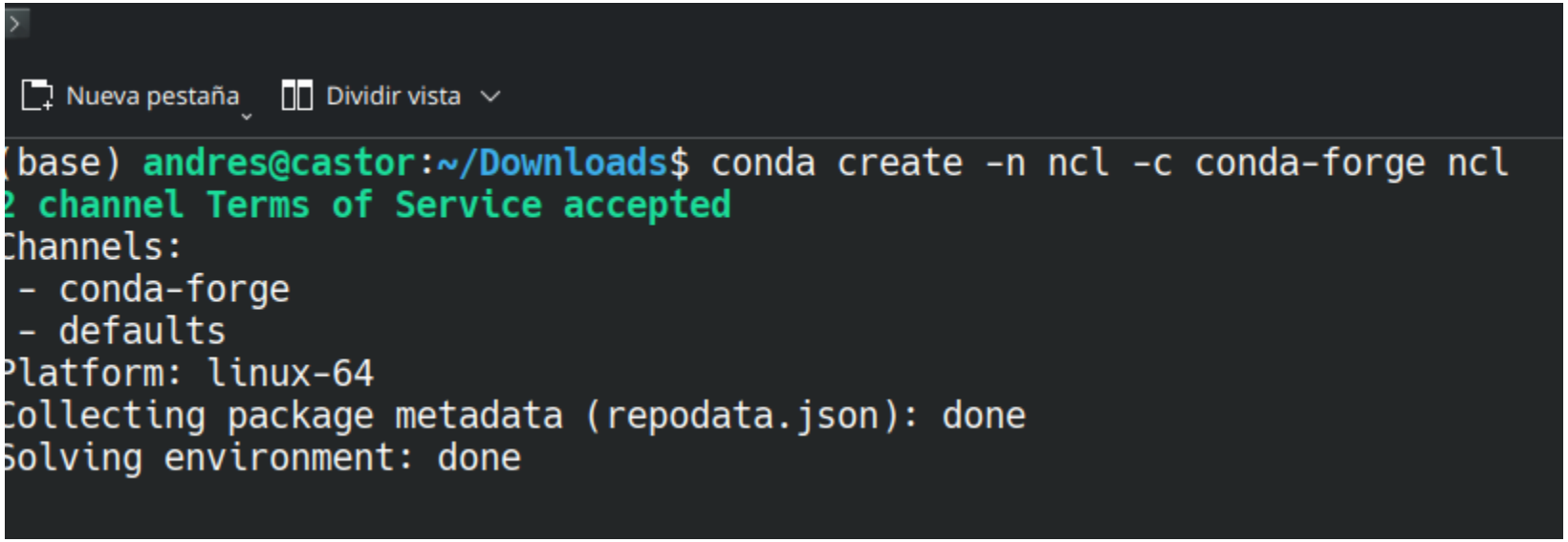
https://www.giss.nasa.gov/tools/panoply/

NCAR Command Language

https://www.ncl.ucar.edu/



NCAR Command Language



conda create -n *ncl_stable* -c conda-forge ncl