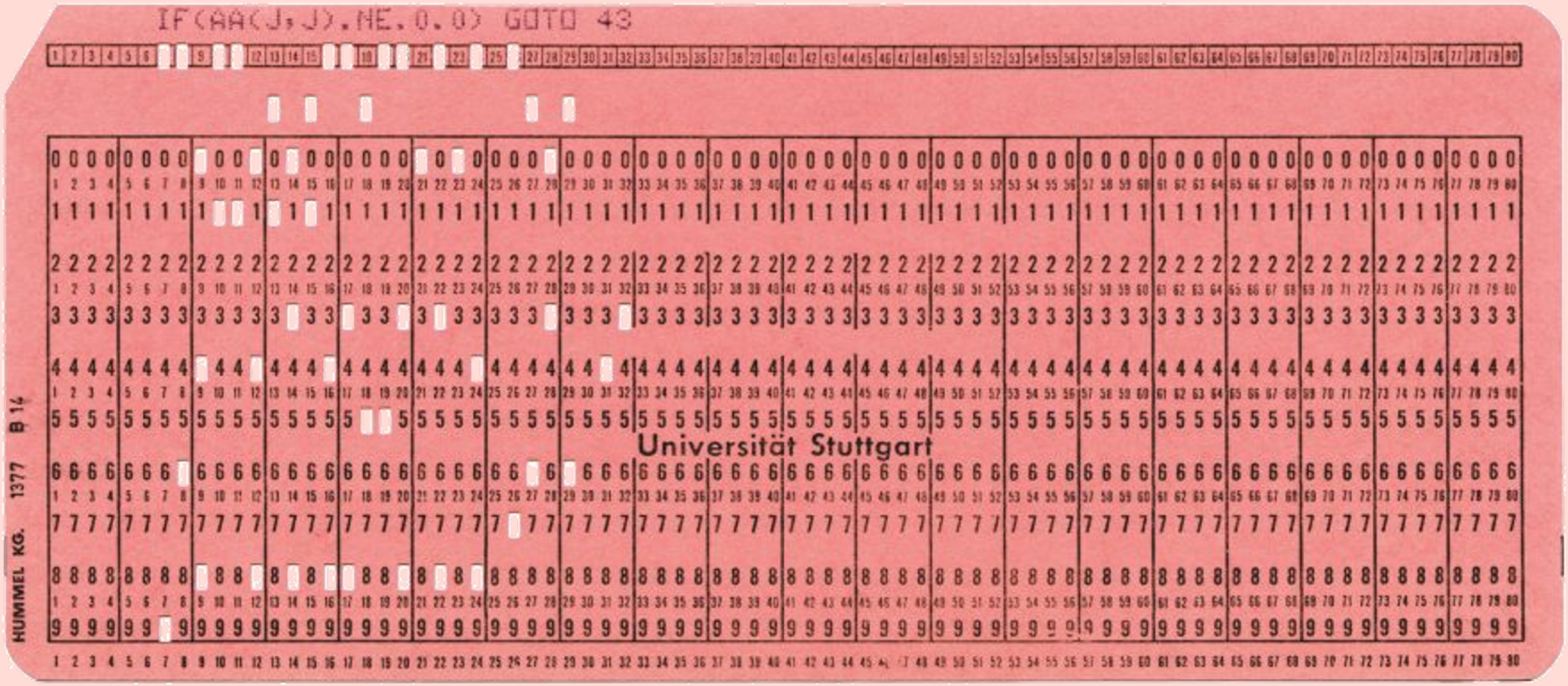
ML Fundamentals

Instituto Balseiro

Instituto Balseiro
19/08/2022



Lecture 2
Code

ML Fundamentals – lecture 2

- Why Python
- Modules& tools
  - Anaconda
  - Jupyter
  - Python modules
  - R
  - Keras/Tensorflow
- End-2-end WF example with Scikit-learn
- Sesión especial
  - Jupyter Notebooks
  - Caso de estudio end-to-end
  - (Google Colab)

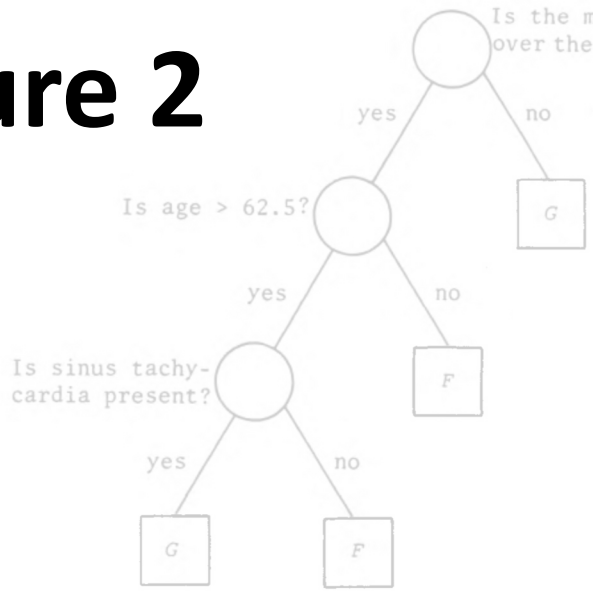
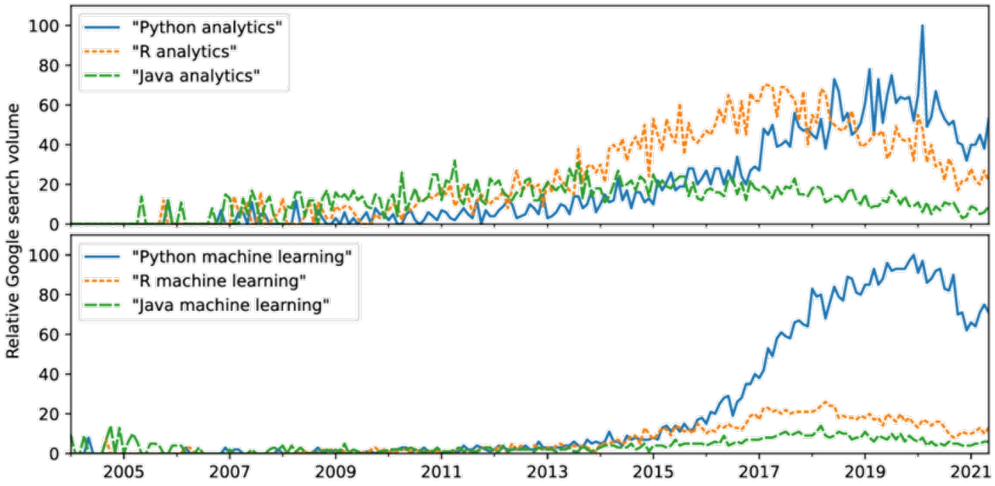


FIGURE 1.1

Why Python?

- Python se distingue por ser uno de los lenguajes con una comunidad grande y activa
- Gran colección de modulos disponibles
- Fácil de aprender
- Fácil de comunicar
- Código eficiente
- Código abierto, gratis
- Flexible



https://www.python.org
http://scipy-lectures.org/intro/intro.html
http://scipy-lectures.org/intro/language/python_language.html

Python modules extend Python

1

Shell

```
$ conda install scipy
$ conda update scipy
```

(2)

Python >>>

```
import scipy
print(scipy.__file__)
```

3

Python

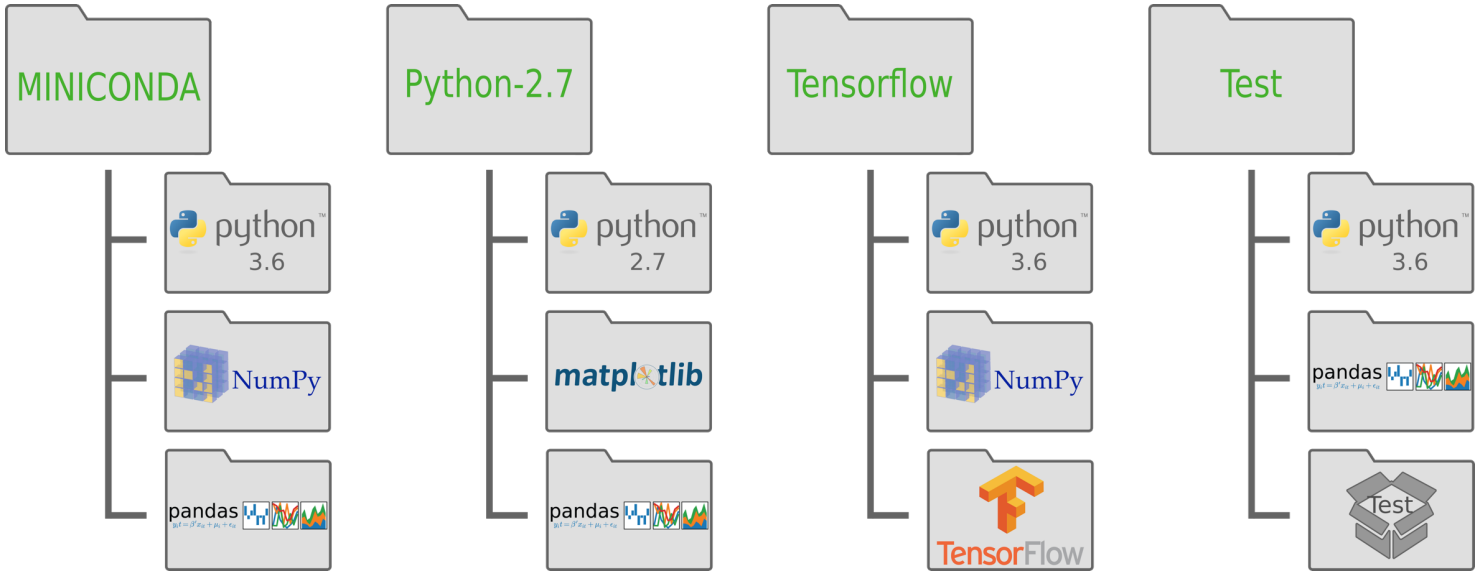
```
1 from pathlib import Path
2 import numpy as np
3 from scipy.cluster.vq import whiten, kmeans, vq
```

4

Python

```
15 whitened_counts = whiten(unique_counts)
16 codebook, _ = kmeans(whitened_counts, 3)
```

Environments




```
conda create -n python2 python=2.7 matplotlib pandas
```

```
conda create -n Tensorflow python=3.6 numpy
```

```
pip install tensorflow
```

```
conda activate Tensorflow
```

Anaconda ecosystem

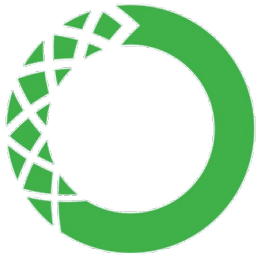
Anaconda

www.anaconda.com

A downloadable, free, open-source, high-performance, and optimized Python and R ***distribution****. Anaconda includes conda, conda-build, Python, and 250+ automatically installed, open-source scientific packages and their dependencies*

CONDA

https://docs.conda.io/projects/conda/en/stable/



Package, dependency and environment management *for any language---Python, R, Ruby, Lua, Scala, Java, JavaScript, C/C++, FORTRAN*

Anaconda vs. pip/virtualenv

https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/

Anaconda cli

- Algunos comando útiles (https://docs.conda.io/projects/conda/en/latest/commands.html):
  - **conda list**
  - **conda search**
  - **conda install**
  - **conda clean --all**
  - **conda update --all** o **conda update nombre_paquete**
  - **conda info --envs**
  - **conda create -n myenv python=3.6 numpy ….**
  - **conda remove —name myenv —all**
  - **conda -h** o **conda list -h** o **conda update -h ….**

Miniconda

https://docs.conda.io/en/latest/miniconda.html

Miniconda is a free minimal installer for conda. It is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib and a few others. Use the `conda install command` to install 720+ additional conda packages from the Anaconda repository.

Choose Anaconda if you:

- Are new to conda or Python.
- Like the convenience of having Python and over 1,500 scientific packages automatically installed at once.
- Have the time and disk space---a few minutes and 3 GB.
- Do not want to individually install each of the packages you want to use.

Choose Miniconda if you:

- Do not mind installing each of the packages you want to use individually.
- Do not have time or disk space to install over 1,500 packages at once.
- Want fast access to Python and the conda commands and you wish to sort out the other programs later.

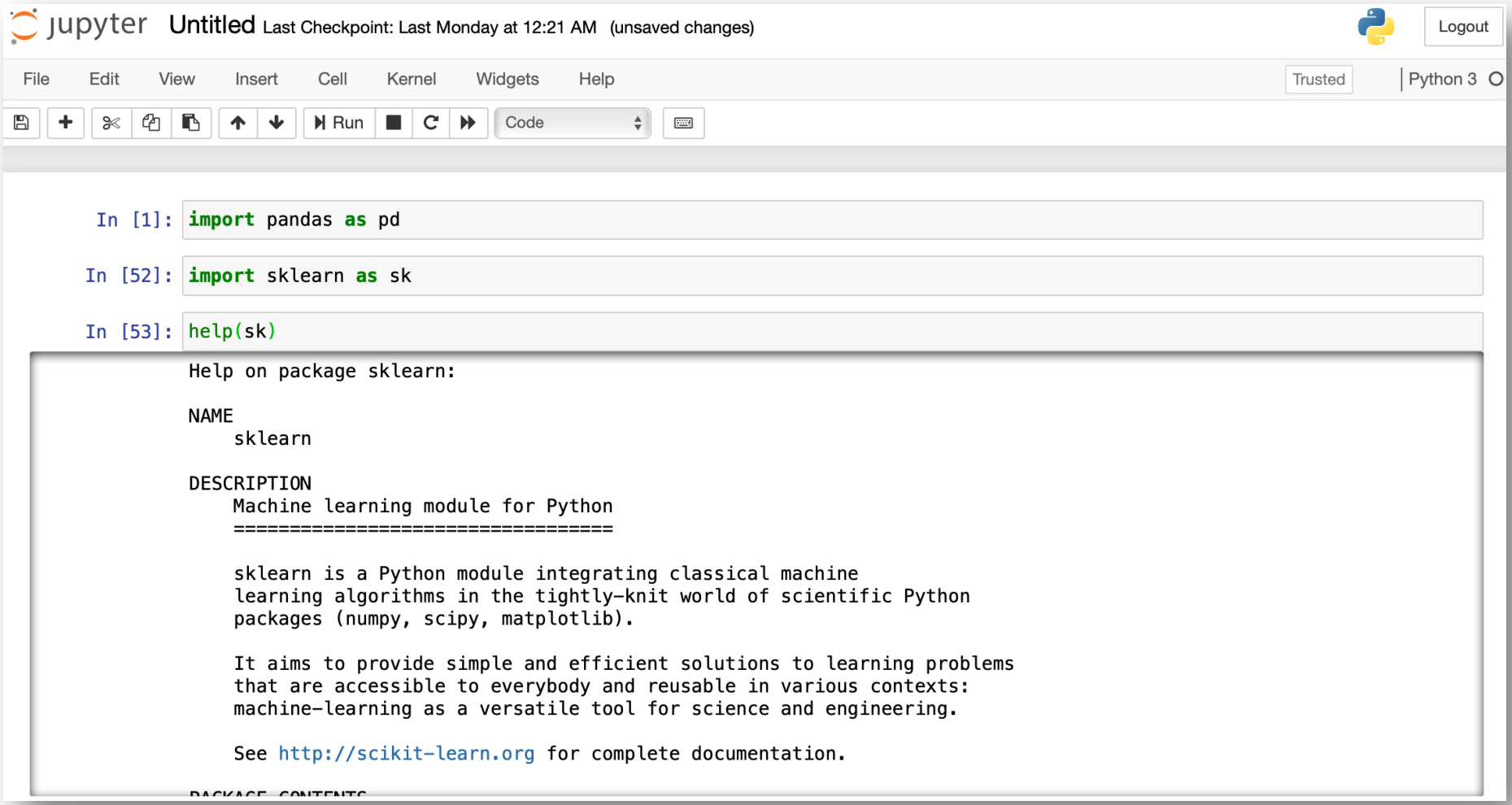
Tip

If you are unsure which option to download, choose the most recent version of Anaconda3, which includes Python 3.7. If you are on Windows or macOS, choose the version with the GUI installer.

Jupyter is an open-source project created to support interactive data science and scientific computing across programming languages. Jupyter offers a web-based environment for working with notebooks containing code, data, and text. Jupyter notebooks are the standard workspace for most Python data scientists.

- The Jupyter Notebook is an interactive environment for running code in the browser.
- It is a great tool for exploratory data analysis and is widely used by data scientists.
- Supports many programming languages, we only need the Python support.
- Easy to incorporate code, text, and images

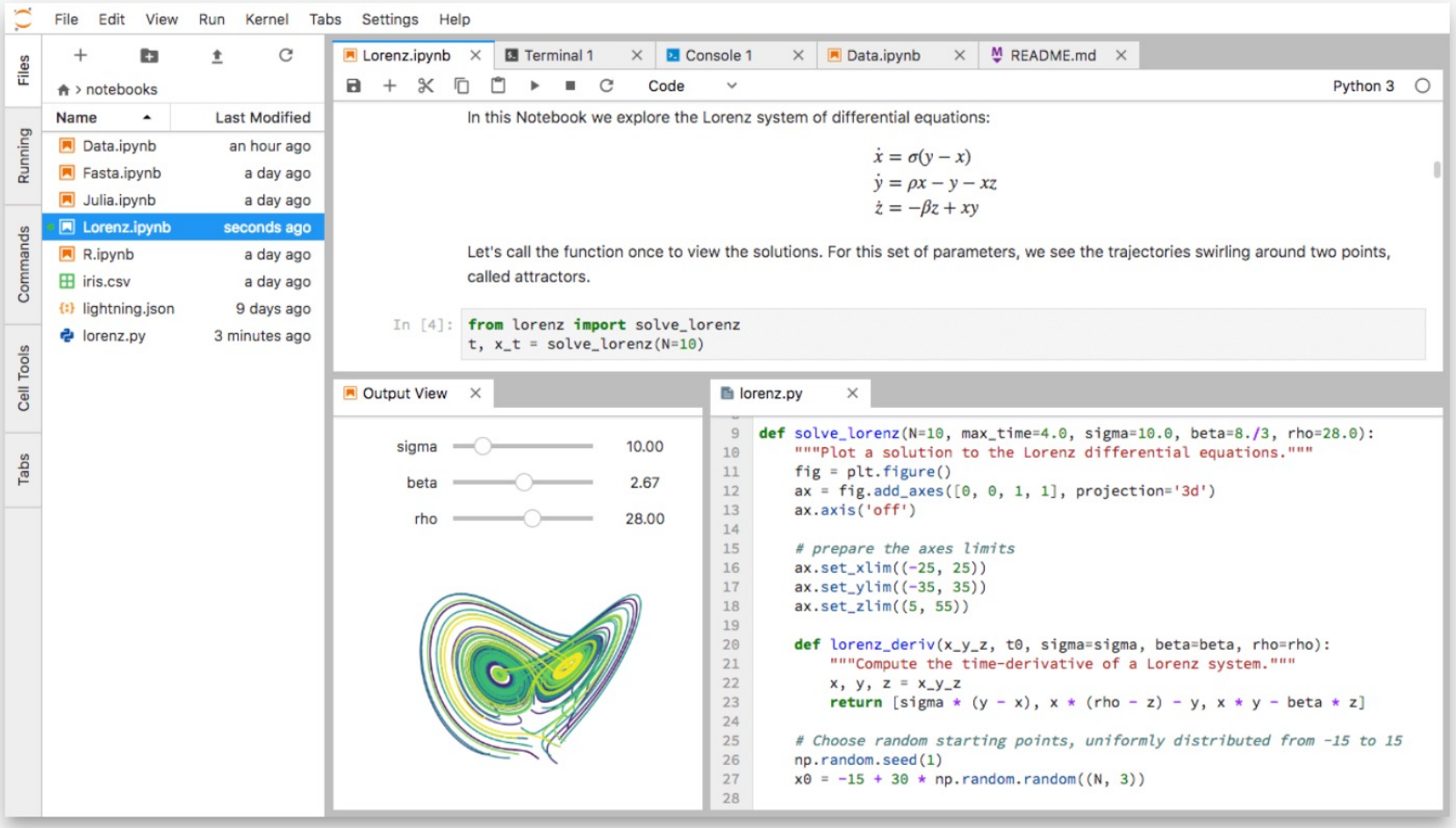
Jupyter Notebook



JupyterLab Documentation

https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html

JupyterLab is the next-generation web-based user interface for Project Jupyter. Try it on Binder. JupyterLab follows the Jupyter Community Guides.

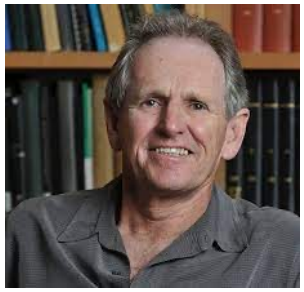


(O, pueden usar el editor de texto favorito (Atom, Emacs, Vim, ...) + un shell)

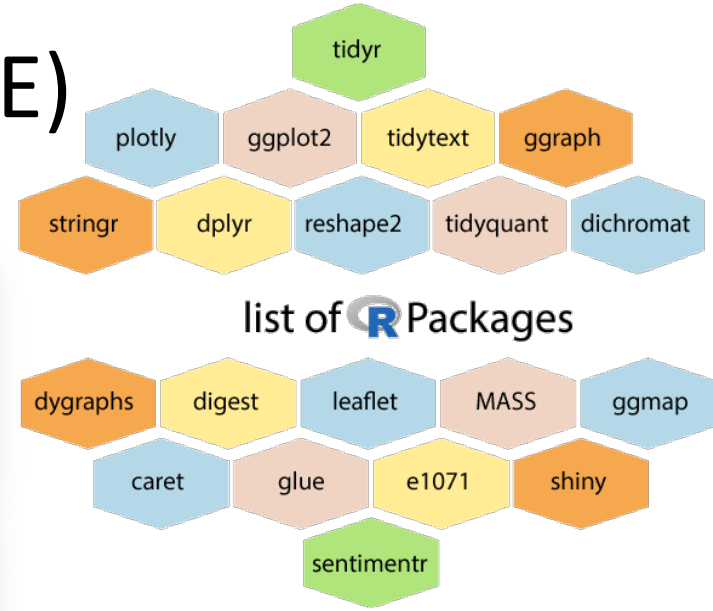
GNU R

- 1995-2000 onwards
- Statistical language, now DS
- Tidyverse: DS library
- Tidy models (caret): ML packets

- CRAN and the R Foundation
- RStudio (IDE)
  - Posit



T. Hastie



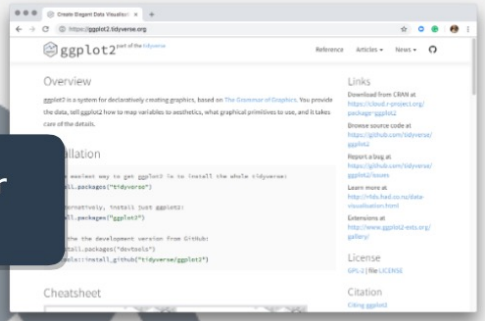
Hadley Wickham

Jenny Bryan

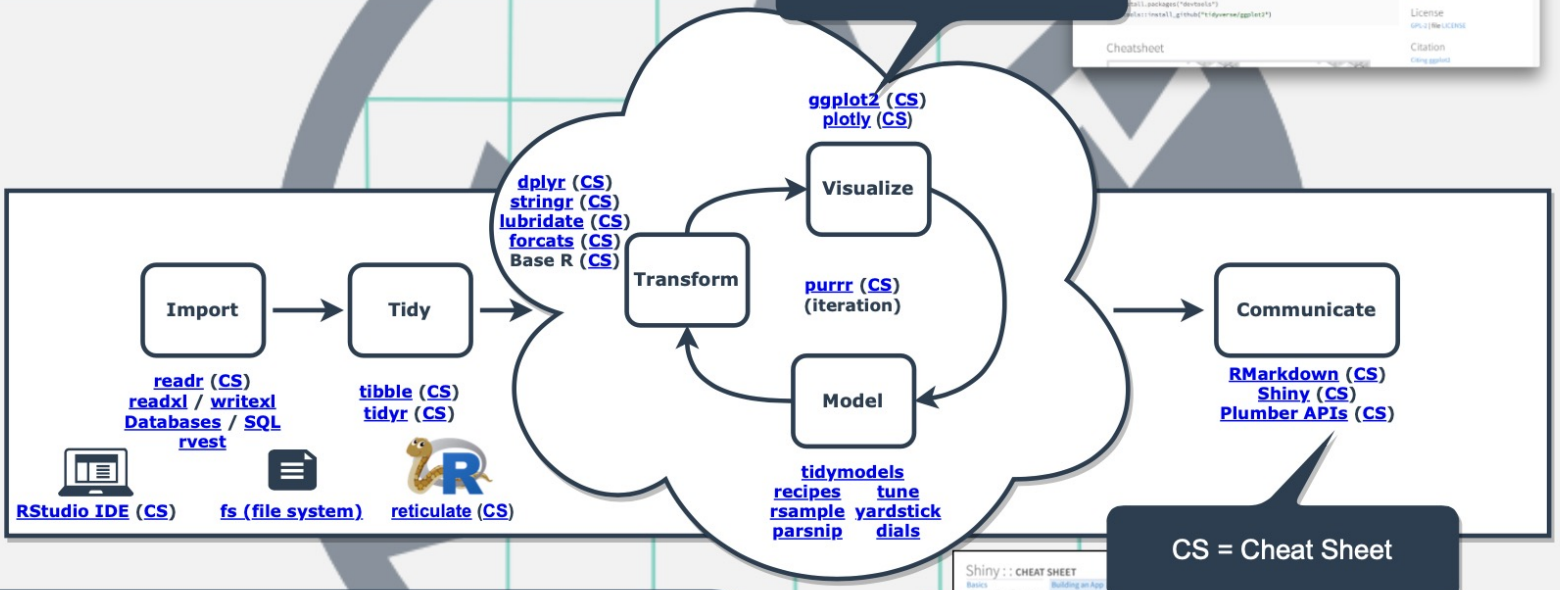
Max Kuhn

Julia Silge

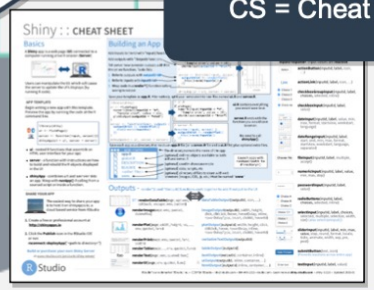
Data Science with R Workflow



Click the links for Documentation



CS = Cheat Sheet



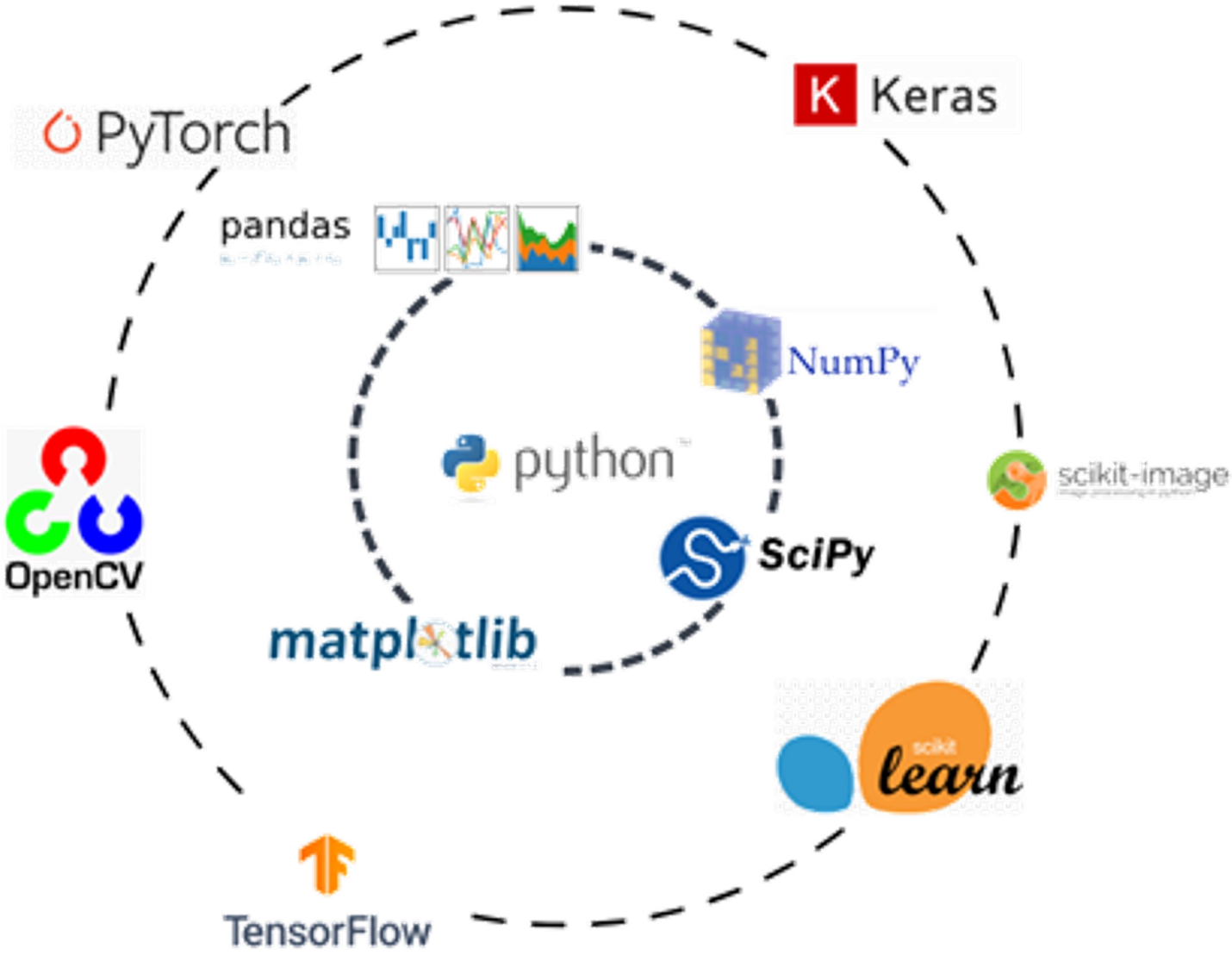
Important Resources

- **R For Data Science Book:** http://r4ds.had.co.nz/
- **Rmarkdown:** Book and Cookbook
- **More Cheatsheets:** https://www.rstudio.com/resources/cheatsheets/
- **tidyverse packages:** https://www.tidyverse.org/
- **Connecting to databases:** https://db.rstudio.com/
- **Reproducible Environments:** https://environments.rstudio.com/

Business Science University
university.business-science.io

version: 4.1

Python basic modules for ML



http://scipy-lectures.org/intro/numpy/index.html

A core package for scientific computing with Python. Numpy enables array formation and basic operations with arrays. Numpy is used for indexing and sorting but can also be used for linear algebra and other operations. Many other data-science libraries for Python are built on NumPy internally, including Pandas and SciPy.

- ndarray, a fast and space-efficient multidimensional array providing vectorized arithmetic operations
- Standard mathematical functions for fast operations on entire arrays of data without having to write loops
- Tools for reading / writing array data to disk and working with memory-mapped files
- Linear algebra, random number generation, and Fourier transform capabilities
- Tools for integrating code written in C, C++, and Fortran

Python For Data Science *Cheat Sheet*

NumPy Basics

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

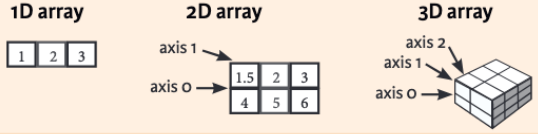
NumPy

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                          Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                   Subtraction
>>> b + a                              Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                        Addition
>>> a / b                              Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                     Division
>>> a * b                              Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                   Multiplication
>>> np.exp(b)                          Exponentiation
>>> np.sqrt(b)                         Square root
>>> np.sin(a)                          Print sines of an array
>>> np.cos(b)                          Element-wise cosine
>>> np.log(a)                          Element-wise natural logarithm
>>> e.dot(f)                           Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                             Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                              Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)               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]                                   Select the element at the 2nd index
  3
>>> b[1,2]                                 Select the element at row 1 column 2
  6.0                                      (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]                                 Select items at index 0 and 1
  array([1, 2])
>>> b[0:2,1]                               Select items at rows 0 and 1 in column 1
  array([ 2.,  5.])
>>> b[:1]                                  Select all items at row 0
  array([[1.5, 2., 3.]])                   (equivalent to b[0:1, :])
>>> c[1,...]                               Same as [1,:,:]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
>>> a[ : :-1]                              Reversed array a
  array([3, 2, 1])
```

Boolean Indexing

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

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]           Select elements (1,0),(0,1),(1,2) and (0,0)
  array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]           Select a subset of the matrix's rows
  array([[ 4. ,5. , 6. , 4. ],             and columns
         [ 1.5, 2. , 3. , 1.5],
         [ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5]])
```

Array Manipulation

Transposing Array

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

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)           Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                       Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                             Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                       Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))                 Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                             Create stacked column-wise arrays
```

Splitting Arrays

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

NumPy Arrays are 'Tensors'

```
import numpy as np

x = np.random.random((64, 3, 32, 10))
y = np.random.random((32, 10))

z = np.maximum(x, y)

z = np.dot(x, y)
```

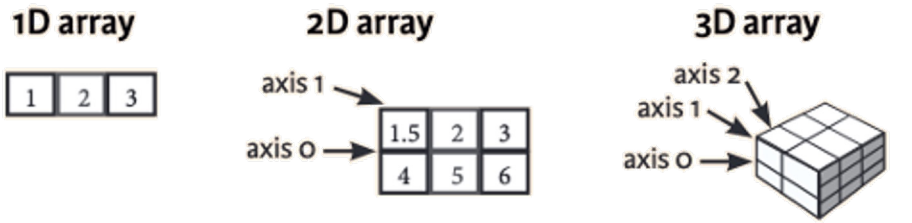
x is a random tensor with shape (64, 3, 32, 10).

y is a random tensor with shape (32, 10).

The output z has shape (64, 3, 32, 10) like x.

```
>>> x = x.reshape((6, 1))
>>> x
array([[ 0.],
       [ 1.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.]])
```

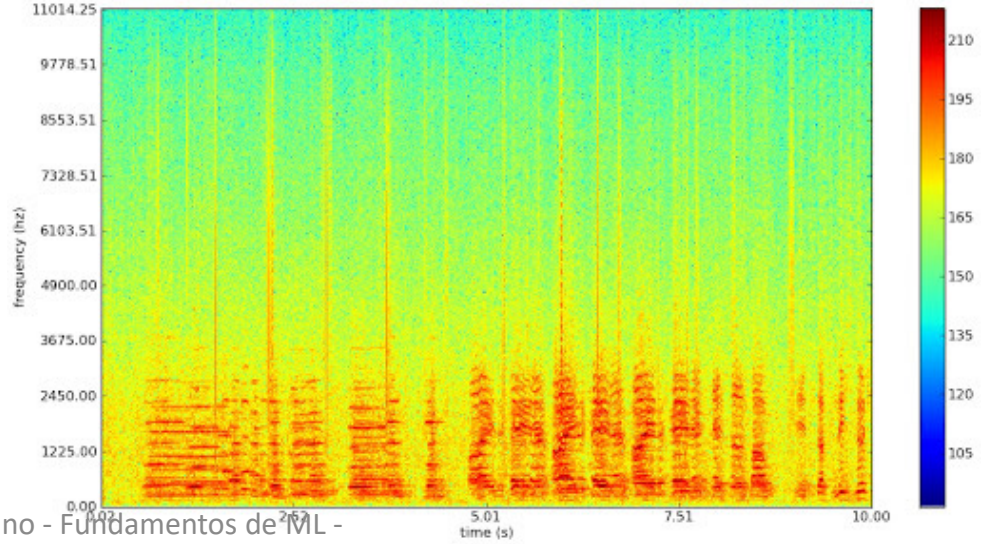
```
>>> x = x.reshape((2, 3))
>>> x
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.]])
```



SciPy

http://scipy-lectures.org/intro/scipy.html

The SciPy library consists of a specific set of fundamental scientific and numerical tools for Python that data scientists use to build their own tools and programs. It provides many user-friendly and efficient numerical routines, such as routines for numerical integration, interpolation, optimization, linear algebra, and statistics.



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.vpslit(d,2)             Split the array vertically at the 2nd index
```

Polynomials

```
>>> from numpy import poly1d
>>> p = poly1d([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(c)                 Return the real part of the array elements
>>> np.imag(c)                 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
>>> A.T                        Tranpose matrix
>>> A.H                        Conjugate transposition
>>> 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(D,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
```

Matrix Functions

Addition

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

Subtraction

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

Division

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

Multiplication

```
>>> 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 Tunctions

```
>>> 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.sigm(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
```

Asking For Help

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

SciPy example: optimization

```
from scipy.optimize import minimize

def eqn(x):
  return x**2 + x + 2

mymin = minimize(eqn, 0, method='BFGS')

print(mymin)
```

```
from scipy.optimize import fmin_l_bfgs_b
from scipy.misc import imsave
import time

result_prefix = 'my_result'
iterations = 20

x = preprocess_image(target_image_path)
x = x.flatten()
for i in range(iterations):
    print('Start of iteration', i)
    start_time = time.time()
    x, min_val, info = fmin_l_bfgs_b(evaluator.loss,
                                     x,
                                     fprime=evaluator.grads,
                                     maxfun=20)
    print('Current loss value:', min_val)
    img = x.copy().reshape((img_height, img_width, 3))
    img = deprocess_image(img)
    fname = result_prefix + '_at_iteration_%d.png' % i
    imsave(fname, img)
    print('Image saved as', fname)
    end_time = time.time()
    print('Iteration %d completed in %ds' % (i, end_time - start_time))
```

This is the initial state: the target image.

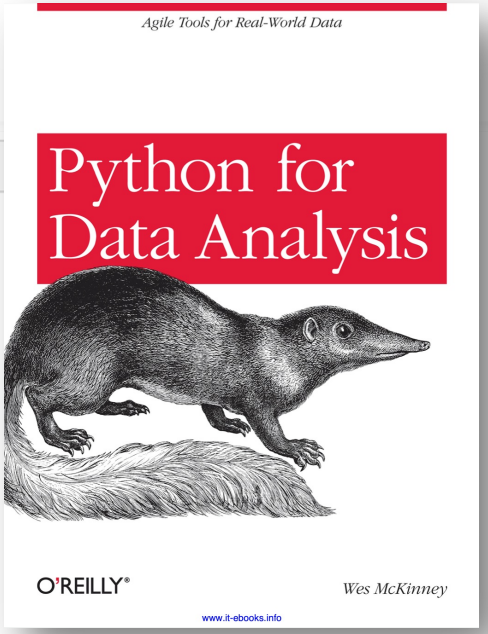
You flatten the image because scipy.optimize.fmin_l_bfgs_b can only process flat vectors.

Runs L-BFGS optimization over the pixels of the generated image to minimize the neural style loss. Note that you have to pass the function that computes the loss and the function that computes the gradients as two separate arguments.

Saves the current generated image.

pandas

A library for tabular data structures, data analysis, and data modeling tools, including built-in plotting using Matplotlib. pandas aims to be the fundamental high-level building block for doing practical, real world data analysis in Python



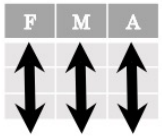

```
In [62]: help(pd)
Help on package pandas:

NAME
    pandas

DESCRIPTION
    pandas - a powerful data analysis and manipulation library for Python
    =====================================================================

    **pandas** is a Python package providing fast, flexible, and expressive data
    structures designed to make working with "relational" or "labeled" data both
    easy and intuitive. It aims to be the fundamental high-level building block for
    doing practical, **real world** data analysis in Python. Additionally, it has
    the broader goal of becoming **the most powerful and flexible open source data
    analysis / manipulation tool available in any language**. It is already well on
    its way toward this goal.

    Main Features
    -------------
    Here are just a few of the things that pandas does well:

      - Easy handling of missing data in floating point as well as non-floating
        point data.
```

Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

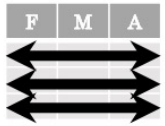
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



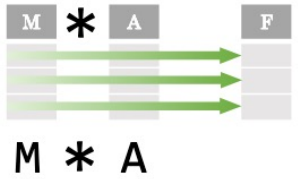
Each **variable** is saved in its own **column**

&



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.



Syntax – 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 the layout of a data set

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).

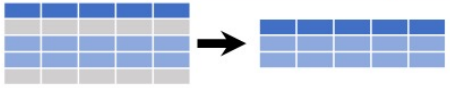
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

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.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

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

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

df.iloc[10:20]
Select rows by position.

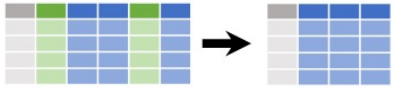
df.nlargest(n, 'value')
Select and order top n entries.

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

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

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*.

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'

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

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

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

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

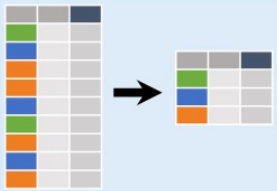
`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group.
Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

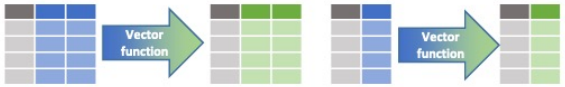
`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)`
Element-wise max.

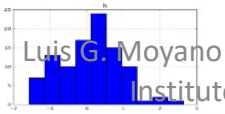
`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.

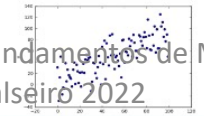
`abs()`
Absolute value.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

```
In [16]: data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]

In [17]: arr2 = np.array(data2)

In [18]: arr2
Out[18]:
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])

In [19]: arr2.ndim
Out[19]: 2

In [20]: arr2.shape
Out[20]: (2, 4)
```

ndarray

```
In [39]: DataFrame(data, columns=['year', 'state', 'pop'])
Out[39]:
   year   state  pop
0  2000    Ohio  1.5
1  2001    Ohio  1.7
2  2002    Ohio  3.6
3  2001  Nevada  2.4
4  2002  Nevada  2.9
```

DataFrame

Each entity or row here is known as a *sample* (or data point) in machine learning, while the columns—the properties that describe these entities—are called *features*.

Pandas example

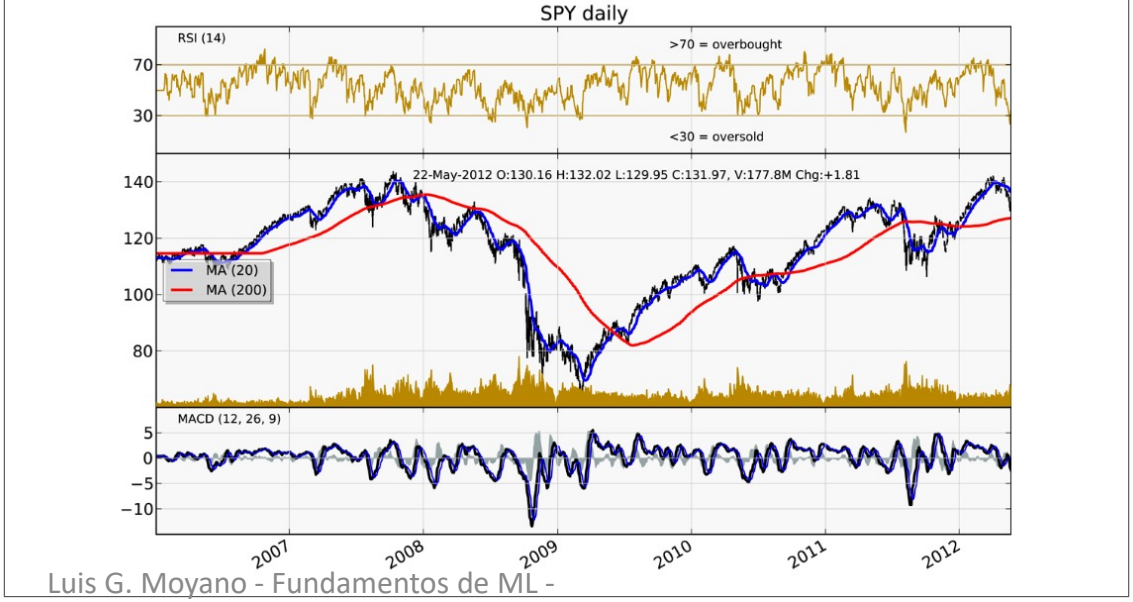
```
In [57]: df = pd.DataFrame(
   ....:     {
   ....:         "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
   ....:         "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
   ....:         "C": np.random.randn(8),
   ....:         "D": np.random.randn(8),
   ....:     }
   ....: )
   ....:

In [58]: df
Out[58]:
     A      B         C         D
0  foo    one -0.575247  1.346061
1  bar    one  0.254161  1.511763
2  foo    two -1.143704  1.627081
3  bar  three  0.215897 -0.990582
4  foo    two  1.193555 -0.441652
5  bar    two -0.077118  1.211526
6  foo    one -0.408530  0.268520
7  foo  three -0.862495  0.024580
```

```
In [82]: grouped = df.groupby("A")

In [83]: grouped["C"].agg([np.sum, np.mean, np.std])
Out[83]:
            sum      mean       std
A
bar  0.392940  0.130980  0.181231
foo -1.796421 -0.359284  0.912265
```

matplotlib

http://scipy-lectures.org/intro/matplotlib/index.html

Matplotlib is the most well-established Python data visualization tool, focusing primarily on two-dimensional plots (line charts, bar charts, scatter plots, histograms, and many others). It works with many GUI interfaces and file formats, but has relatively limited interactive support in web browsers.



Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

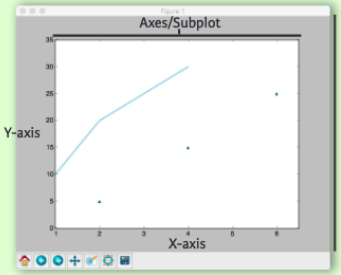
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                 Add padding to a plot
>>> ax.axis('equal')                        Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])   Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                     Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',         Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                   No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),          Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,        Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                      Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)    Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> lines = ax.plot(x,y)                    Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                         Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])          Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])     Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                 Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                 Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')               Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')     Fill between y-values and o
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)            Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                   Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)           Plot 2D vector fields
```

Data Distributions

```
>>> ax1.hist(y)                             Plot a histogram
>>> ax3.boxplot(y)                          Make a box and whisker plot
>>> ax3.violinplot(z)                       Make a violin plot
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                     Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                  Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)               Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)                 Plot contours
>>> axes2[2].contourf(data1)                Plot filled contours
>>> axes2[2]= ax.clabel(CS)                 Label a contour plot
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

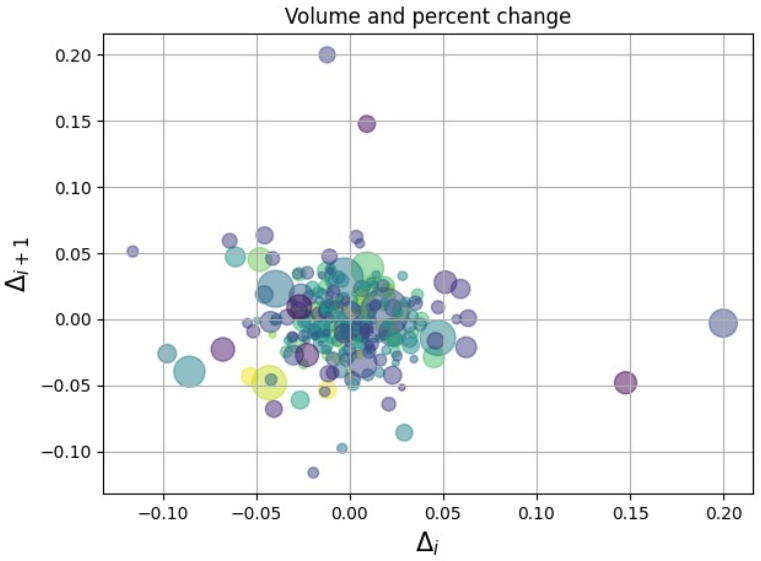
```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()                               Clear an axis
>>> plt.clf()                               Clear the entire figure
>>> plt.close()                             Close a window
```

DataCamp
Learn Python for Data Science Interactively

Matplotlib example




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

# Load a numpy record array from yahoo csv data with fields date, open, close,
# volume, adj_close from the mpl-data/example directory. The record array
# stores the date as an np.datetime64 with a day unit ('D') in the date column.
price_data = (cbook.get_sample_data('goog.npz', np_load=True)['price_data']
              .view(np.recarray))
price_data = price_data[-250:]  # get the most recent 250 trading days

delta1 = np.diff(price_data.adj_close) / price_data.adj_close[:-1]

# Marker size in units of points^2
volume = (15 * price_data.volume[:-2] / price_data.volume[0])**2
close = 0.003 * price_data.close[:-2] / 0.003 * price_data.open[:-2]

fig, ax = plt.subplots()
ax.scatter(delta1[:-1], delta1[1:], c=close, s=volume, alpha=0.5)

ax.set_xlabel(r'$\Delta_i$', fontsize=15)
ax.set_ylabel(r'$\Delta_{i+1}$', fontsize=15)
ax.set_title('Volume and percent change')

ax.grid(True)
fig.tight_layout()

plt.show()
```

A powerful and versatile machine learning library for machine learning basics like classification, regression, and clustering. It includes both supervised and unsupervised ML algorithms with important functions like cross-validation and feature extraction. Scikit-learn is the most frequently downloaded machine learning library.

```
In [53]: help(sk)
```

```
Help on package sklearn:

NAME
    sklearn

DESCRIPTION
    Machine learning module for Python
    ==================================

    sklearn is a Python module integrating classical machine
    learning algorithms in the tightly-knit world of scientific Python
    packages (numpy, scipy, matplotlib).

    It aims to provide simple and efficient solutions to learning problems
    that are accessible to everybody and reusable in various contexts:
    machine-learning as a versatile tool for science and engineering.

    See http://scikit-learn.org for complete documentation.

PACKAGE CONTENTS
```

scikit learn Install User Guide API Examples More Go

scikit-learn

Machine Learning in Python

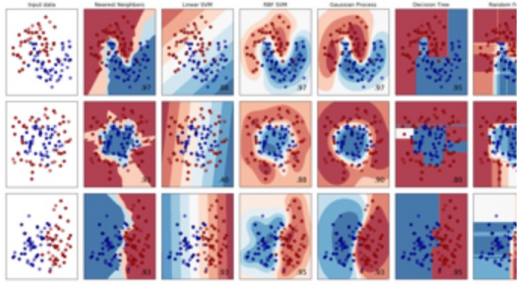
Getting Started | What's New in 0.22.2 | GitHub

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

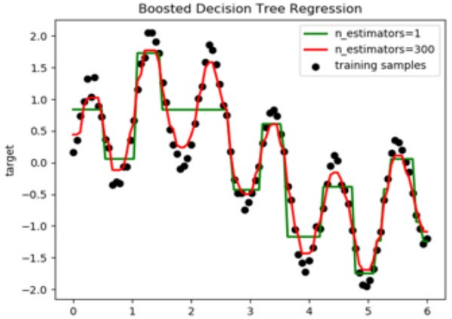
Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...



Regression

Predicting a continuous-valued attribute associated with an object.

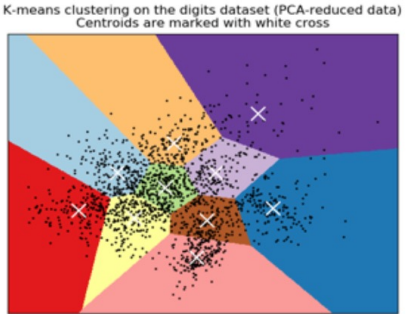
Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...



Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...



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

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_)
```

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

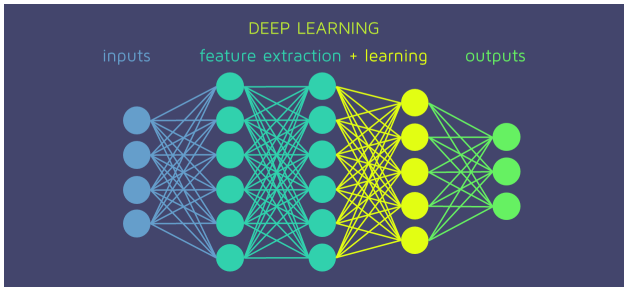
```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

DataCamp
Learn Python for Data Science Interactively

TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

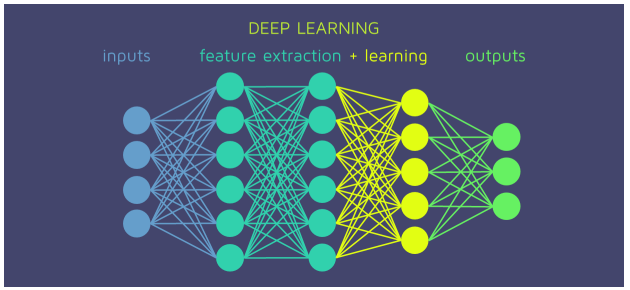
TensorFlow is a more complex library for distributed numerical computation. It makes it possible to train and run very large neural networks efficiently by distributing the computations across potentially hundreds of multi-GPU servers. TensorFlow was created at Google and supports many of their large-scale Machine Learning applications. It was open sourced in November 2015.



Keras

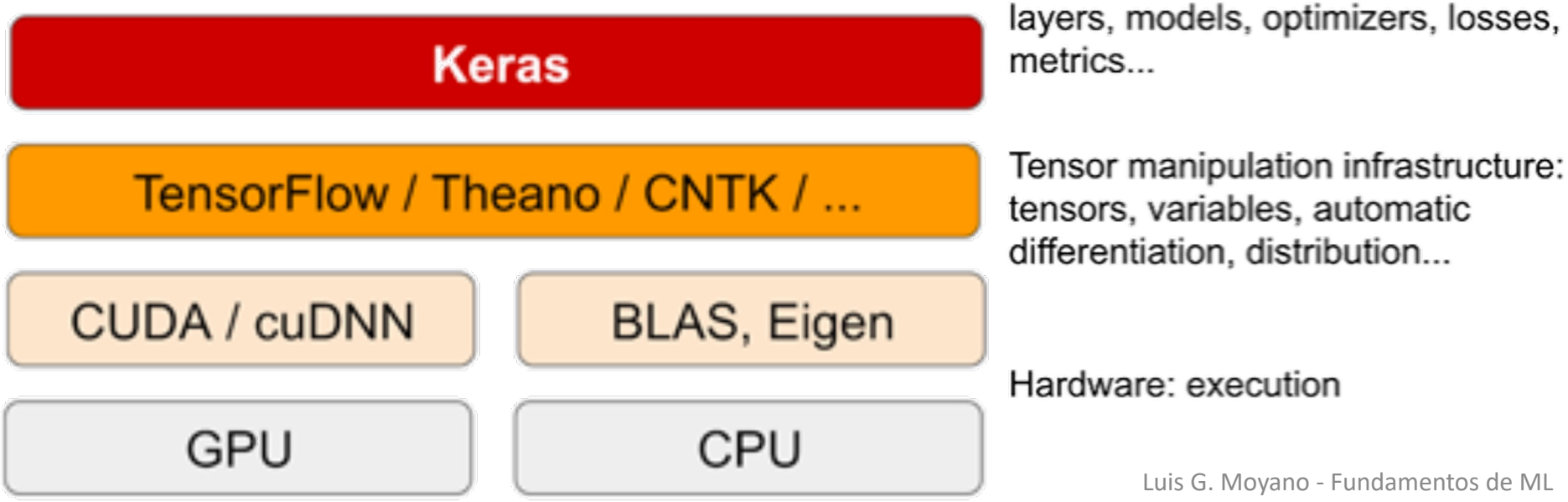
Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Keras is a high level Deep Learning API that makes it very simple to train and run neural networks. It can run on top of either TensorFlow, Theano or Microsoft Cognitive Toolkit (formerly known as CNTK). TensorFlow comes with its own implementation of this API, called *tf.keras*, which provides support for some advanced TensorFlow features (e.g., to efficiently load data).

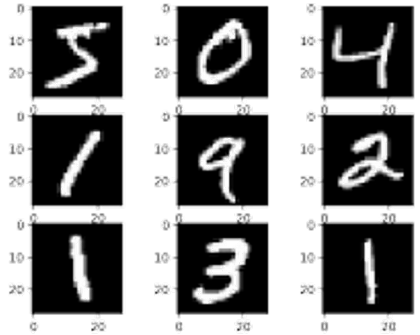


NNs & DL in Python - Keras and TF

- **TensorFlow** es una plataforma de ML hecha por Google en Python, abierta y gratis.
- **Keras** is una API en Python para deep learning, que funciona como wrapper sobre TensorFlow, y proporciona una manera conveniente de definir y entrenar cualquier modelo de deep learning.



Keras & TensorFlow



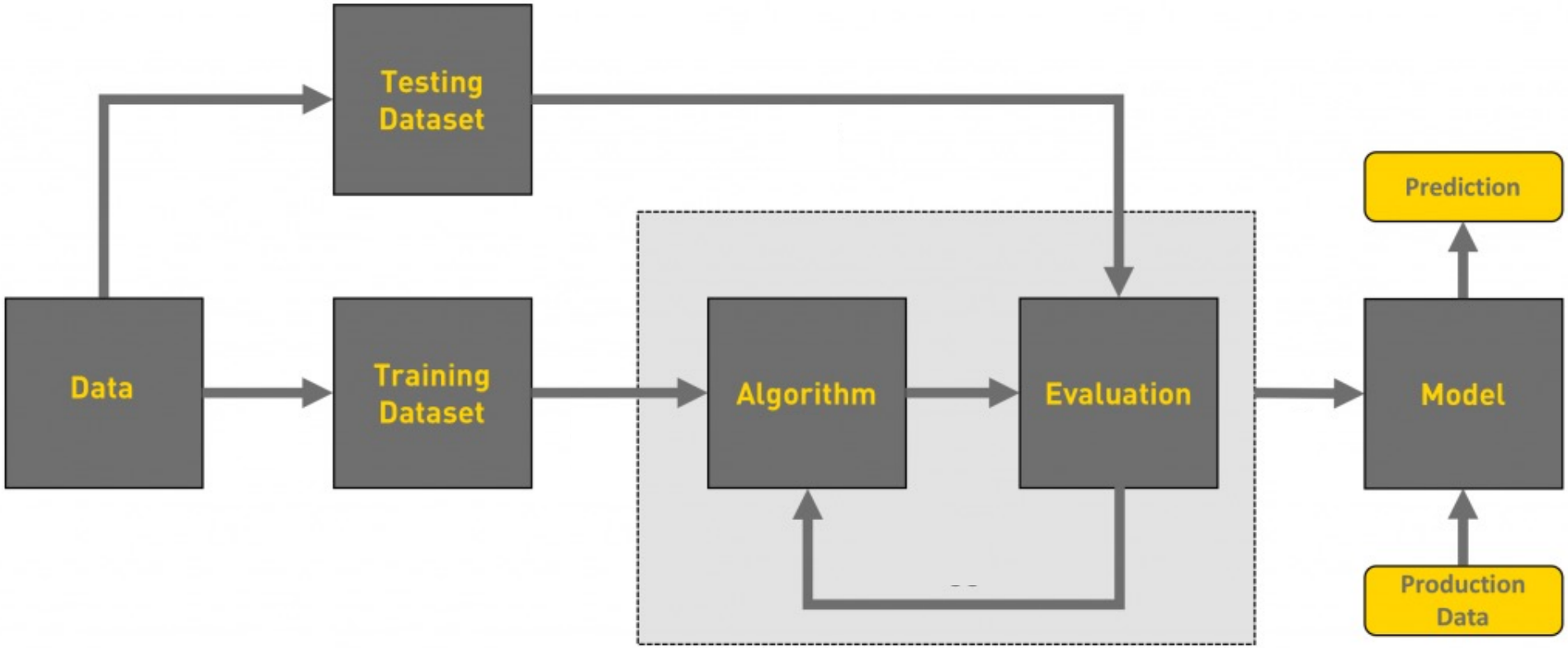
```
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([          # ANN model
tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(10)
])

predictions = model(x_test[:1]).numpy() # predict test image
predictions

tf.nn.softmax(predictions).numpy() # compute probabilities
```

https://www.tensorflow.org/tutorials/keras/classification

TensorFlow

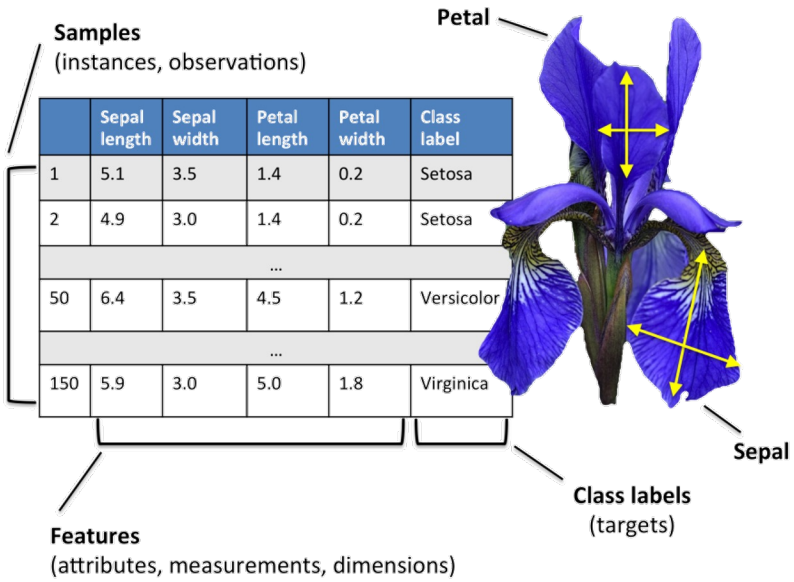
Run in Google Colab

ML canonical end-to-end workflow



Entrenamos un primer modelo de ML
Clasificador de especies de Iris

- Clásico set de datos
- Años ‘20 o ‘30
- Adjudicado a R. Fischer, eminencia frecuentista
- Es el ‘Hello world’ de los datos



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

https://github.com/amueller/introduction_to_ml_with_python/blob/master/01-introduction.ipynb

Cargamos los datos, los miramos un poco

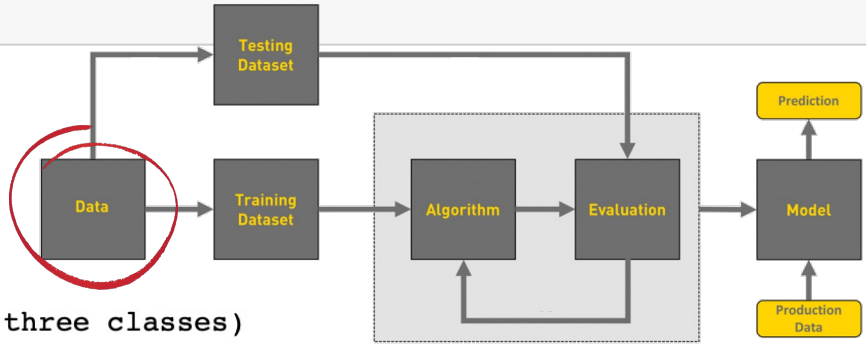
In [10]:
```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

In [11]:
```
print("Keys of iris_dataset:\n", iris_dataset.keys())
```

```
Keys of iris_dataset:
 dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

In [12]:
```
print(iris_dataset['DESCR'][:193] + "\n...")
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, pre
...
```



In [13]:
```
print("Target names:", iris_dataset['target_names'])
```

```
Target names: ['setosa' 'versicolor' 'virginica']
```

In [14]:
```
print("Feature names:\n", iris_dataset['feature_names'])
```

```
Feature names:
 ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

In [15]:
```python
print("Type of data:", type(iris_dataset['data']))
```

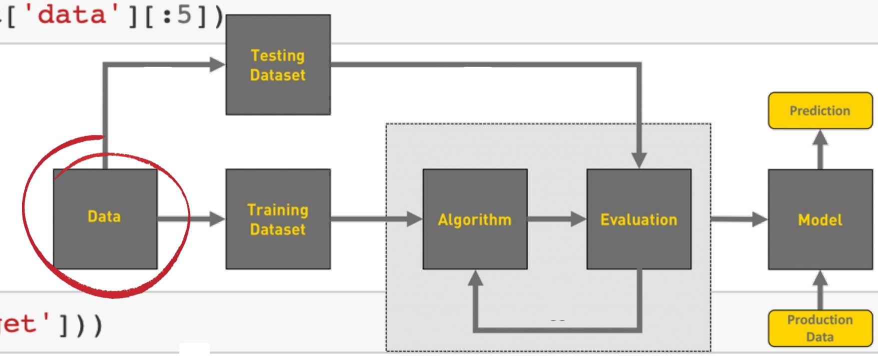
```
Type of data: <class 'numpy.ndarray'>
```

In [16]:
```python
print("Shape of data:", iris_dataset['data'].shape)
```

```
Shape of data: (150, 4)
```

In [17]:
```python
print("First five rows of data:\n", iris_dataset['data'][:5])
```

```
First five rows of data:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```



In [18]:
```python
print("Type of target:", type(iris_dataset['target']))
```

```
Type of target: <class 'numpy.ndarray'>
```

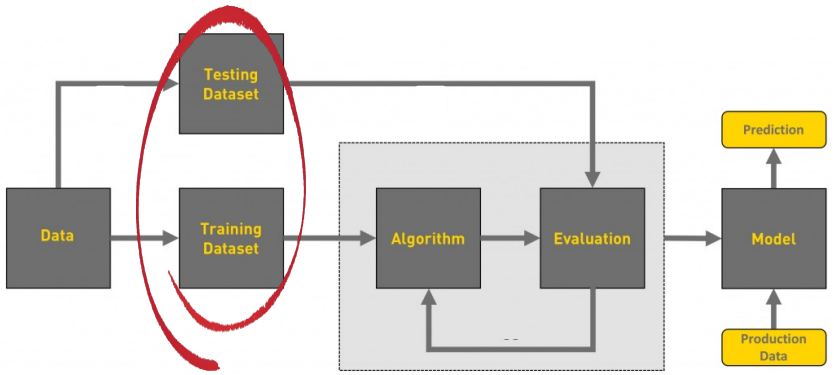
In [19]:
```python
print("Shape of target:", iris_dataset['target'].shape)
```

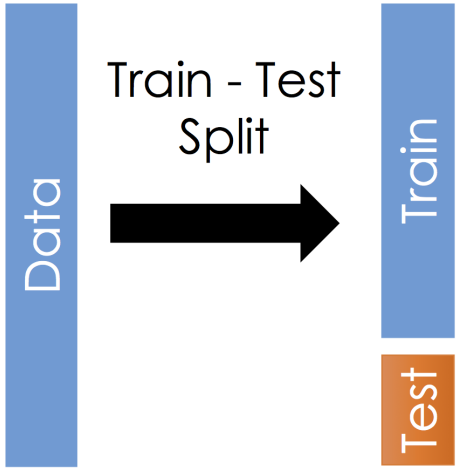
```
Shape of target: (150,)
```

In [20]:
```python
print("Target:\n", iris_dataset['target'])
```

```
Target:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Dividimos entre datos de entrenamiento y test





In [21]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)
```

In [22]:
```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

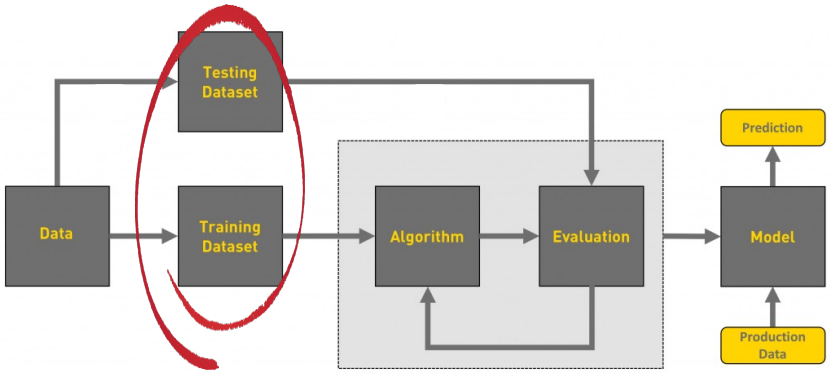
```
X_train shape: (112, 4)
y_train shape: (112,)
```

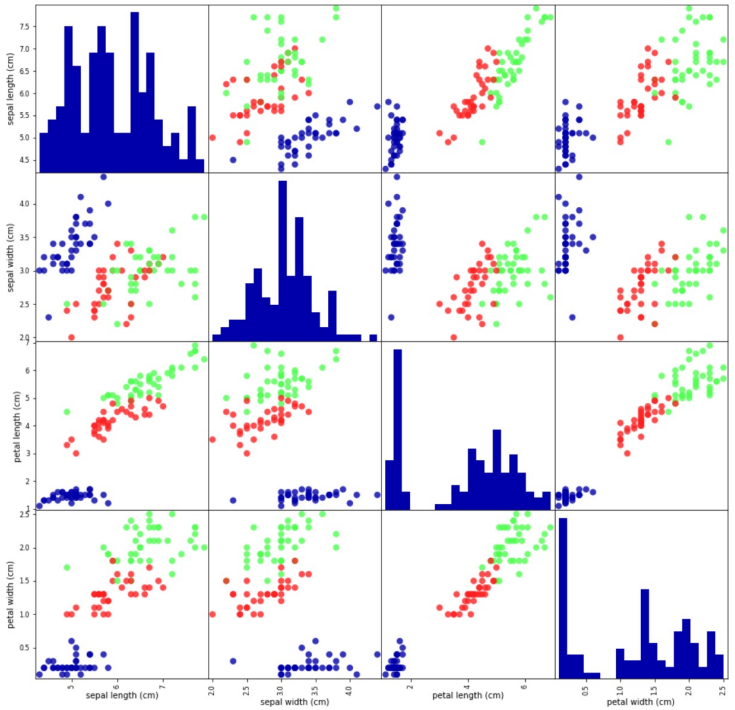
In [23]:
```
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_test shape: (38, 4)
y_test shape: (38,)
```

Exploramos los datos

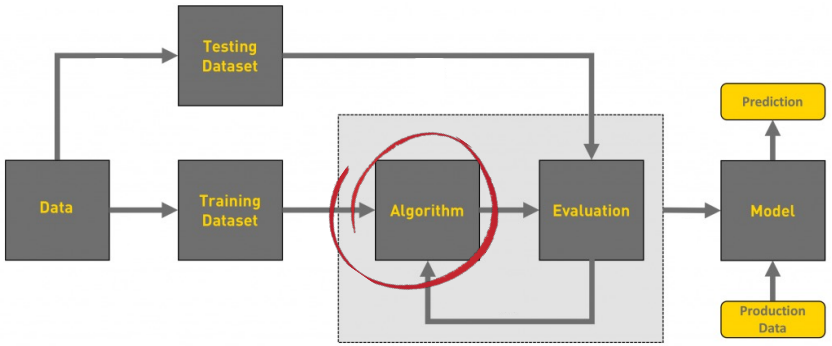
In [24]:
```
# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# create a scatter matrix from the dataframe, color by y_train
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15),
                           marker='o', hist_kwds={'bins': 20}, s=60,
                           alpha=.8, cmap=mglearn.cm3)
```

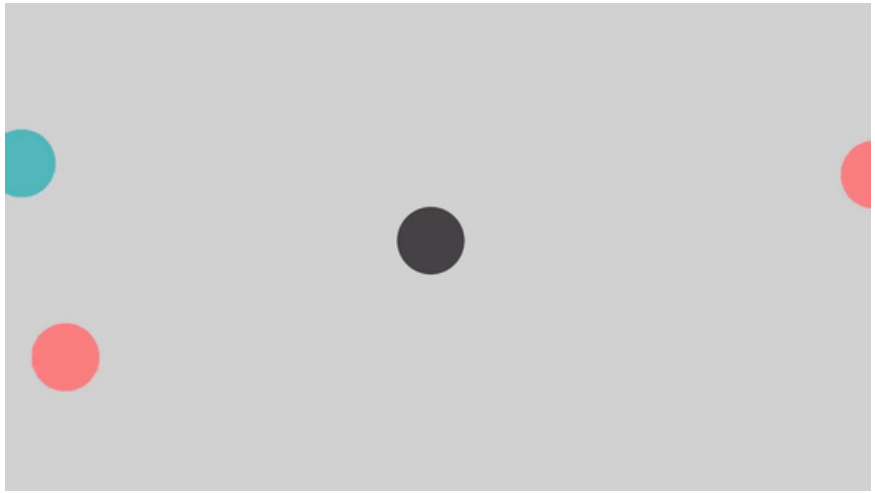




Primer modelo de ML: k vecinos más cercanos - KNN

```
In [25]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors=1)

In [26]: knn.fit(X_train, y_train)

Out[26]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                              weights='uniform')
```





Predecimos con los datos de test

In[27]:

```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape: {}".format(X_new.shape))
```

Out[27]:

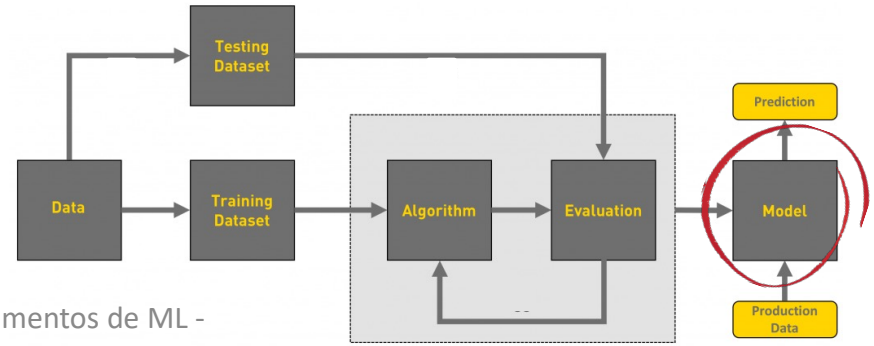
```
X_new.shape: (1, 4)
```

In[28]:

```
prediction = knn.predict(X_new)
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format(
       iris_dataset['target_names'][prediction]))
```

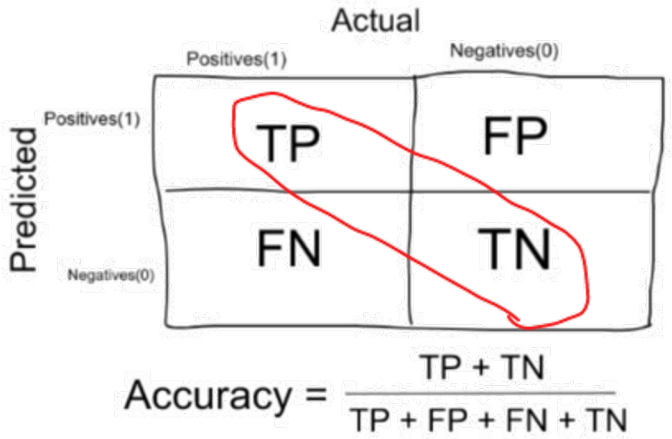
Out[28]:

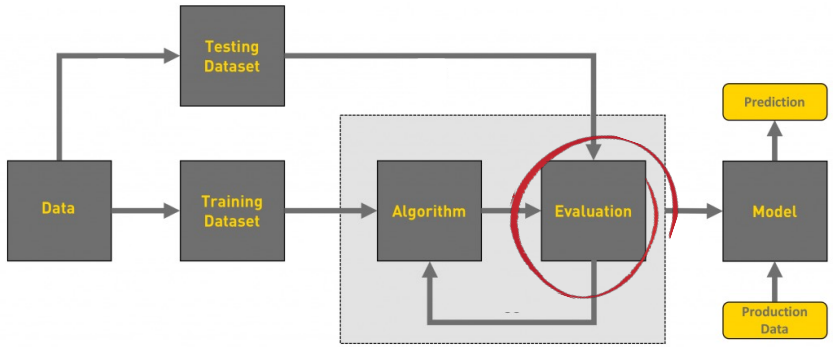
```
Prediction: [0]
Predicted target name: ['setosa']
```



Medimos el error

```
In [29]: y_pred = knn.predict(X_test)
         print("Test set predictions:\n", y_pred)

         Test set predictions:
          [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
          2]

In [30]: print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))

         Test set score: 0.97

In [31]: print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))

         Test set score: 0.97
```





Hay otras métricas como *precision* y *recall*

Next

- Anímense a hacer lo mismo, pero en vez de usar knn, usar random forest en su version clasificador (RandomForestClassifier)
- Intenten calcular que *precision* y *recall* tienen

Summary

- **Python** es uno de los lenguajes más extendidos y flexibles, ideal para aplicaciones de data science y ML
- **Herramientas y módulos**: Anaconda, Jupyter notebook, Pandas, Scikit-learn y otros, Git
- **Ejemplo de Workflow** end-to-end de ML

Próxima clase: **Error**