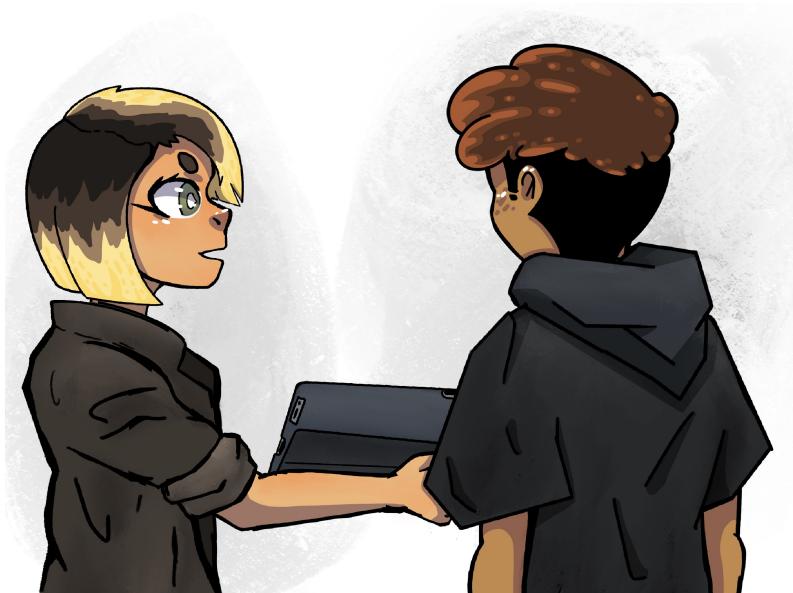


Librerías de programación de Aprendizaje Automático con Python.

Caso práctico



[@casfatesvano](https://twitter.com/casfatesvano) ([CC BY-SA](http://creativecommons.org/licenses/?lang=es))

Lorena lleva un par de meses haciendo las prácticas de sus estudios en la empresa Pick&Deliver, que ofrece servicios de logística, distribución y paquetería para empresas principalmente de e-commerce. Está aprendiendo a integrar técnicas de aprendizaje automático en el proceso de toma de decisiones del negocio.

Su mentor en la empresa, Miguel, le ha propuesto el siguiente reto: "Uno de los objetivos principales de la empresa es que se cumplan los plazos de entrega que nos piden las empresas cliente de cara a las entregas a su cliente final de los productos que gestionamos para ellos. ¿Crees que podrías entrenar un modelo que hiciese esa predicción en función de los parámetros que se controlan a partir de que el pedido pasa a manos de Pick&Deliver?"

El gran avance que ha experimentado el campo del aprendizaje automático, es debido, en gran medida, por la contribución de diferentes miembros de la comunidad que han compartido sus hallazgos y desarrollos, con licencias de código abierto. Una contribución clave, han sido las librerías que se han creado, para simplificar el diseño, creación, entrenamiento y evaluación de modelos de aprendizaje automático.

En esta unidad vamos a hacer un repaso de las librerías que constituyen la base de una gran mayoría de los desarrollos actuales de aprendizaje automático. En concreto, conoceremos y probaremos:

- ✓ Pandas.
- ✓ Matplotlib.
- ✓ Scikit learn.
- ✓ Tensorflow y Keras
- ✓ Pytorch

Trabajaremos un ejemplo de cada librería, aplicando funciones de éstas para resolver diferentes tareas habitualmente presentes en procesos de ciencia de datos, como pueden ser:

- ✓ Carga de los datos
- ✓ EDA o Análisis Exploratorio de los Datos
- ✓ Visualización gráfica de los datos
- ✓ Preparación y tratamiento previo de los datos
- ✓ Generación del modelo de aprendizaje automático
- ✓ Entrenamiento del modelo de aprendizaje automático a partir de los datos
- ✓ Definición del modelo predictivo
- ✓ Evaluación del modelo entrenado con datos reservados



Ministerio de Educación y Formación Profesional
(<https://www.educacionyfp.gob.es/portada.html>) (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](http://www.educacionyfp.gob.es/fpadistancia/comunes/aviso-legal.html) (<http://www.educacionyfp.gob.es/fpadistancia/comunes/aviso-legal.html>)

1.- Pandas.

Caso práctico



@casfatesvano (<https://twitter.com/casfatesvano>) (CC BY-SA (<http://creativecommons.org/licenses/?lang=es>))

conoce de Python que le pueden ayudar en esta tarea, y en seguida recuerda que Pandas es de las más efectivas para esto, así que la importa en la primera celda del notebook y se abre una pestaña en el navegador con la documentación para ir consultando las funciones que necesita.

Lorena está decidida a conseguir un buen modelo de predicción que les ayude a mejorar los plazos de entrega a los clientes finales. Ha obtenido un conjunto de datos con los que hacer una primera aproximación y se dispone a trabajar con ellos en colab, el entorno tipo notebook online.

Pero no tiene ni idea de cómo son los valores ni qué variables entran en juego, así que necesita empezar por obtener información sobre los datos en sí.

Revisa mentalmente las librerías que



Tenor (<https://tenor.com/view/pandas-dancing-wtf-gif-3428091>) (Dominio público)

La librería Pandas fue creada en 2008, y se liberó en 2009 como proyecto de código abierto. Más tarde, en 2015, pasó a estar gestionado por NumFOCUS. El principal logro de Pandas es la generación de un objeto, de la clase DataFrame, que va a permitir gran versatilidad y capacidad de manipulación, más allá de los métodos de los que se disponía. Desde la carga de un fichero csv, hasta las funcionalidades de preparación de los datos, es una librería que facilita enormemente el trabajo de los científicos de datos.

Para comprobar todo el potencial de esta librería, abre un cuaderno en colab y ponte a practicar con estos ejemplos (puedes acceder al cuaderno editable [aquí](https://colab.research.google.com/drive/1k3VjXgkyWdtPpZxQnr1NWkU47QoYFpsj?usp=sharing)):

Ejemplos utilizando librería Pandas

Empezamos importando la librería Pandas y numpy

In [2]:

```
import pandas as pd  
import numpy as np
```

Para entender cómo generamos un dataset con Pandas, lo comparamos con un vector convencional

In [3]:

```
myvector = np.array([1,2,3,4,5])
print("myvector:")
print(myvector)
print()
mydf1 = pd.DataFrame(myvector)
print("mydf1:")
print(mydf1)
```

myvector:

```
[1 2 3 4 5]
```

mydf1:

```
    0
0  1
1  2
2  3
3  4
4  5
```

El nuevo dataframe creado, tiene índices de filas y de columnas. Probamos a generar uno desde un array de dos dimensiones (una matriz)

In [4]:

```
myarray = np.array([[10,30,20], [50,40,60],[1000,2000,3000]])
print("myarray:")
print(myarray)
print()
mydf2 = pd.DataFrame(myarray)
print("mydf2:")
print(mydf2)
```

myarray:

```
[[ 10   30   20]
 [ 50   40   60]
 [1000 2000 3000]]
```

mydf2:

```
      0      1      2
0    10     30     20
1    50     40     60
2  1000   2000   3000
```

En los elementos anteriores, se aprecia que los dataframes de pandas ponen claves de filas y columnas por defecto, que serán números si no damos otro tipo de claves. Vamos a crear un df ahora en forma de diccionario:

In [5]:

```
names = pd.Series(['SF', 'San Jose', 'Sacramento'])
sizes = pd.Series([852469, 1015785, 485199])
df = pd.DataFrame({ 'Cities': names, 'Size': sizes })
print(df)
```

```
Cities      Size
0          SF    852469
1    San Jose   1015785
2  Sacramento    485199
```

Los índices de las columnas se han tomado a partir de las claves del diccionario. Pero también podemos crear un dataframe desde un array y asignar después los valores de índices de las filas y de las columnas

In [47]:

```
myarray = np.array([[10,30,20], [50,40,60],[1000,2000,3000]])
rownames = ['lentejas', 'espinacas', 'cerveza']
colnames = ['enero', 'febrero', 'marzo']
mydf = pd.DataFrame(myarray, index=rownames, columns=colnames)
print("Compras:")
print(mydf)
```

```
Compras:
        enero  febrero  marzo
lentejas      10        30     20
espinacas      50        40     60
cerveza     1000      2000    3000
```

Si solo queremos ver los datos de enero

In [48]:

```
print("Compras de enero:")
print(mydf['enero'])
```

```
Compras de enero:
lentejas      10
espinacas      50
cerveza     1000
Name: enero, dtype: int64
```

Si queremos saber la cantidad de elementos en cada dimensión:

In [49]:

```
print("Número de filas:")
print(mydf.shape[0])
print()
print("Número de columnas:")
print(mydf.shape[1])
```

```
Número de filas:
3
```

```
Número de columnas:
3
```

O ver las dimensiones en conjunto:

In [50]:

```
print("Número de filas y columnas:")
print(mydf.shape)
```

Número de filas y columnas:
(3, 3)

También podemos consultar los nombres o etiquetas de filas o columnas, que son los valores que habíamos asignado a los índices, así como el tipo de valor que son.

In [51]:

```
print("Columnas:")
print(mydf.columns)
print()
print("Column types:")
print(mydf.dtypes)
```

Columnas:
Index(['enero', 'febrero', 'marzo'], dtype='object')

Column types:
enero int64
febrero int64
marzo int64
dtype: object

La función "describe" es muy útil, pues nos devuelve los datos estadísticos principales de la distribución de datos

In [52]:

```
print("Valores estadísticos de los datos:")
print(mydf.describe())
```

Valores estadísticos de los datos:

	enero	febrero	marzo
count	3.000000	3.000000	3.000000
mean	353.333333	690.000000	1026.666667
std	560.386771	1134.504297	1709.073823
min	10.000000	30.000000	20.000000
25%	30.000000	35.000000	40.000000
50%	50.000000	40.000000	60.000000
75%	525.000000	1020.000000	1530.000000
max	1000.000000	2000.000000	3000.000000

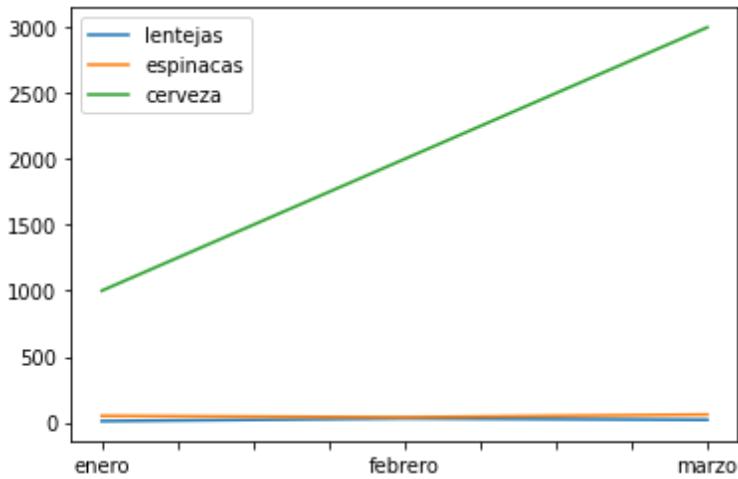
También podemos hacer una representación gráfica básica de los datos

In [53]:

```
mydf.plot()
```

Out[53]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8bb5cc0bd0>
```



A veces, vamos a necesitar trasponer un dataframe, esto es, cambiar las filas por las columnas, y lo hacemos con la función T

In [13]:

```
df1 = pd.DataFrame({'a': [1, 0, 1], 'b': [0, 1, 1]},  
dtype=int)  
print(df1)  
print("df1.T:")  
print(df1.T)
```

```
   a   b  
0  1   0  
1  0   1  
2  1   1  
df1.T:  
   0   1   2  
a  1   0   1  
b  0   1   1
```

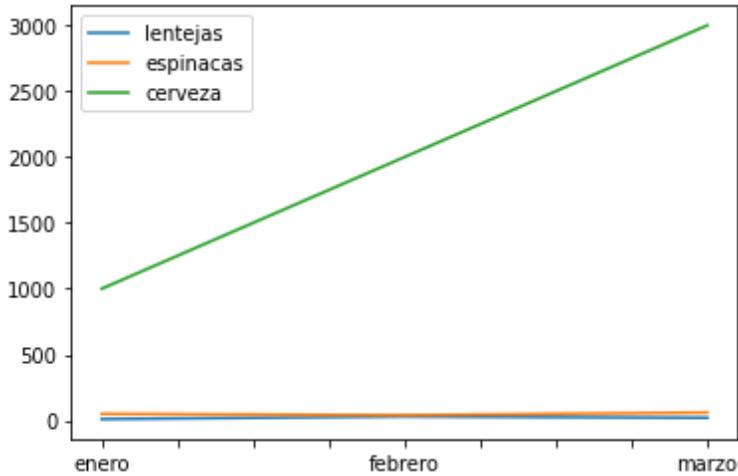
Si lo aplicamos a nuestro dataframe de las compras:

In [14]:

```
df = mydf.T  
df.plot()
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8bc5311110>
```



Viendo el gráfico, podemos concluir que vamos a tener que incentivar de alguna manera el consumo de lentejas y espinacas frente al de cerveza 😅

Probemos ahora a sumar dataframes valor a valor

In [15]:

```
df1 = pd.DataFrame({'a' : [1, 0, 1], 'b' : [0, 1, 1] },  
dtype=int)  
df2 = pd.DataFrame({'a' : [3, 3, 3], 'b' : [5, 5, 5] },  
dtype=int)  
print("df1 + df2:")  
print(df1 + df2)
```

```
df1 + df2:  
   a  b  
0  4  5  
1  3  6  
2  4  6
```

Algo que vamos a necesitar más de una vez, será convertir datos de tipo categórico a numérico. Esta opción es la más básica, y es adecuada para casos de valores binarios. Puedes descargar el archivo 'spam_no_spam.tsv' necesario para este ejemplo, [aquí](#) (<https://drive.google.com/file/d/1o3zlsYxTIU7JyJTzdJF9CIF-AF2-QAGZ/view?usp=sharing>).

In [18]:

```
df = pd.read_csv('/content/spam_no_spam.tsv', delimiter='\t')
print("Primeras cinco líneas:")
print(df.head(5))
print("-----")
# mapeamos no_spam/spam a valores 0/1 respectivamente:
df['type'] = df['type'].map( {'no_spam':0 , 'spam':1} )
print("Primeras cinco líneas despues del cambio:")
print(df.head(5))
print("-----")
```

Primeras cinco líneas:

```
    type          text
0  no_spam  Ya esta disponible su pedido
1  no_spam      Revise sus mensajes
2    spam        Un iPhone gratis!!
3  no_spam      Un 20% de descuento
4  no_spam  Tienes un nuevo comentario
-----
```

Primeras cinco líneas despues del cambio:

```
    type          text
0     0  Ya esta disponible su pedido
1     0      Revise sus mensajes
2     1        Un iPhone gratis!!
3     0      Un 20% de descuento
4     0  Tienes un nuevo comentario
-----
```

En este otro ejemplo, utilizamos replace para sustituir las categorías de talla de camisetas por el valor numérico. El archivo para este ejemplo, puedes descargarlo [aquí](#)

(<https://drive.google.com/file/d/1Q2696bRiBilF3DOpmPyFSOBcz2EiJunx/view?usp=sharing>). El parámetro regex corresponde a "regular expresions", y aunque suele ser de tipo booleano, en casos como éste, toma como valor una cadena. Si quieres saber más, puedes consultar la [documentación de Pandas sobre la función replace](#) (<https://pandas.pydata.org/docs/reference/api/pandas.Series.replace.html>).

In [21]:

```
shirts = pd.read_csv("shirts.csv")
print("shirts before:")
print(shirts)
print()
shirts['size'] = shirts['size'].replace(regex='xlarge',
value=4)
shirts['size'] = shirts['size'].replace(regex='large',
value=3)
shirts['size'] = shirts['size'].replace(regex='medium',
value=2)
shirts['size'] = shirts['size'].replace(regex='small',
value=1)
print("shirts after:")
print(shirts)
```

```
shirts before:
   type    size
0  shirt  xxlarge
1  shirt  xxlarge
2  shirt   xlarge
3  shirt   xlarge
4  shirt   xlarge
5  shirt     large
6  shirt   medium
7  shirt     small
8  shirt     small
9  shirt   xspecial
10 shirt   xspecial
```

```
shirts after:
   type  size
0  shirt    4
1  shirt    4
2  shirt    4
3  shirt    4
4  shirt    4
5  shirt    3
6  shirt    2
7  shirt    1
8  shirt    1
9  shirt    1
10 shirt    1
```

Otras funciones muy útiles son startswith, split, iloc... Probamos

In [28]:

```
shirts = pd.read_csv("shirts.csv")
print("shirts:")
print(shirts)
print()
print("shirts starting with xl:")
print(shirts[shirts['size'].str.startswith('xl')])
print()
print("Exclude 'xlarge' shirts:")
print(shirts[shirts['size'] != 'xlarge'])
print()
print("first three letters:")
shirts['sub1'] = shirts['size'].str[:3]
print(shirts)
print()
print("split ssize on letter 'a':")
shirts['sub2'] = shirts['size'].str.split('a')
print(shirts)
print()
print("Rows 3 through 5 and column 2:")
print(shirts.iloc[2:5, 2])
print()
```

```
shirts:  
      type    size  
0   shirt  xxlarge  
1   shirt  xxlarge  
2   shirt  xlarge  
3   shirt  xlarge  
4   shirt  xlarge  
5   shirt    large  
6   shirt   medium  
7   shirt     small  
8   shirt     small  
9   shirt  xsmall  
10  shirt  xsmall
```

shirts starting with xl:

```
      type    size  
2   shirt  xlarge  
3   shirt  xlarge  
4   shirt  xlarge
```

Exclude 'xlarge' shirts:

```
      type    size  
0   shirt  xxlarge  
1   shirt  xxlarge  
5   shirt    large  
6   shirt   medium  
7   shirt     small  
8   shirt     small  
9   shirt  xsmall  
10  shirt  xsmall
```

first three letters:

```
      type    size sub1  
0   shirt  xxlarge  xxl  
1   shirt  xxlarge  xxl  
2   shirt  xlarge   xla  
3   shirt  xlarge   xla  
4   shirt  xlarge   xla  
5   shirt    large  lar  
6   shirt   medium  med  
7   shirt     small  sma  
8   shirt     small  sma  
9   shirt  xsmall   xsm  
10  shirt  xsmall   xsm
```

split ssize on letter 'a':

```
      type    size sub1          sub2  
0   shirt  xxlarge  xxl  [xxl, rge]  
1   shirt  xxlarge  xxl  [xxl, rge]  
2   shirt  xlarge   xla  [xl, rge]  
3   shirt  xlarge   xla  [xl, rge]  
4   shirt  xlarge   xla  [xl, rge]  
5   shirt    large  lar  [l, rge]  
6   shirt   medium  med  [medium]  
7   shirt     small  sma  [sm, ll]  
8   shirt     small  sma  [sm, ll]  
9   shirt  xsmall   xsm  [xsm, ll]  
10  shirt  xsmall   xsm  [xsm, ll]
```

Rows 3 through 5 and column 2:

```
2   xla
```

```
3     xla
4     xla
Name: sub1, dtype: object
```

Se pueden combinar dataframes con concat

In [29]:

```
import pandas as pd
can_weather = pd.DataFrame({
    "city": ["Vancouver", "Toronto", "Montreal"],
    "temperature": [72, 65, 50],
    "humidity": [40, 20, 25]
})
us_weather = pd.DataFrame({
    "city": ["SF", "Chicago", "LA"],
    "temperature": [60, 40, 85],
    "humidity": [30, 15, 55]
})
df = pd.concat([can_weather, us_weather])
print(df)
```

	city	temperature	humidity
0	Vancouver	72	40
1	Toronto	65	20
2	Montreal	50	25
0	SF	60	30
1	Chicago	40	15
2	LA	85	55

En el siguiente ejemplo, creamos una nueva columna 'Total' que tendrá los valores resultado de sumar los de las otras dos columnas anteriores

In [34]:

```
summary = {
    'Quarter': ['Q1', 'Q2', 'Q3', 'Q4'],
    'Cost': [-23500, -34000, -57000, -32000],
    'Revenue': [40000, 40000, 40000, 40000]
}
df = pd.DataFrame(summary)
print("First Dataset:\n", df)
df['Total'] = df.sum(axis=1)
print("Second Dataset:\n", df)
```

First Dataset:

	Quarter	Cost	Revenue
0	Q1	-23500	40000
1	Q2	-34000	40000
2	Q3	-57000	40000
3	Q4	-32000	40000

Second Dataset:

	Quarter	Cost	Revenue	Total
0	Q1	-23500	40000	16500
1	Q2	-34000	40000	6000
2	Q3	-57000	40000	-17000
3	Q4	-32000	40000	8000

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=True') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.
```

En las siguientes celdas, vemos cómo operar, añadir y borrar columnas

In [35]:

```
df = pd.DataFrame.from_dict({'A':[1,2,3], 'B':[4,5,6]}, orient='index', columns=['one', 'two', 'three'])
print(df)
```

	one	two	three
A	1	2	3
B	4	5	6

In [36]:

```
df['four'] = df['one'] * df['two']
print(df)
```

	one	two	three	four
A	1	2	3	2
B	4	5	6	20

In [37]:

```
df['three'] = df['two'] * df['two']
print(df)
```

	one	two	three	four
A	1	2	4	2
B	4	5	25	20

In [38]:

```
import numpy as np
rand = np.random.randn(2)
df.insert(1, 'random', rand)
print(df)
```

```
   one    random  two  three  four
A    1    0.431789    2      4      2
B    4   -0.126975    5     25     20
```

In [39]:

```
df['flag'] = df['one'] > 2
print(df)
```

```
   one    random  two  three  four  flag
A    1    0.431789    2      4      2  False
B    4   -0.126975    5     25     20   True
```

In [40]:

```
del df['two']
print(df)
```

```
   one    random  three  four  flag
A    1    0.431789    4      2  False
B    4   -0.126975    25     20   True
```

In [41]:

```
three = df.pop('three')
print(df)
```

```
   one    random  four  flag
A    1    0.431789    2  False
B    4   -0.126975    20   True
```

In [42]:

```
df['foo'] = 'bar'
print(df)
```

```
   one    random  four  flag  foo
A    1    0.431789    2  False  bar
B    4   -0.126975    20   True  bar
```

Para saber más

Puedes consultar todas las posibilidades de Pandas en su [documentación](https://pandas.pydata.org/docs/) (<https://pandas.pydata.org/docs/>). También te recomendamos su [tutorial inicial de 10 minutos](https://pandas.pydata.org/docs/user_guide/10min.html) (https://pandas.pydata.org/docs/user_guide/10min.html).

Autoevaluación

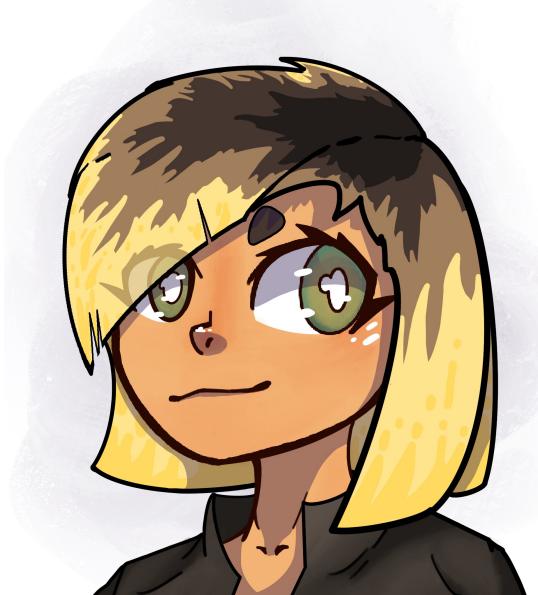
Introduce la parte de código que falta en la siguiente instrucción para cargar datos con la librería Pandas en un proyecto de ciencia de datos:

```
import pandas as pd  
data = pd.  ('datos.csv')
```

El método para la carga de los datos desde un csv es *read_csv*

2.- Matplotlib.

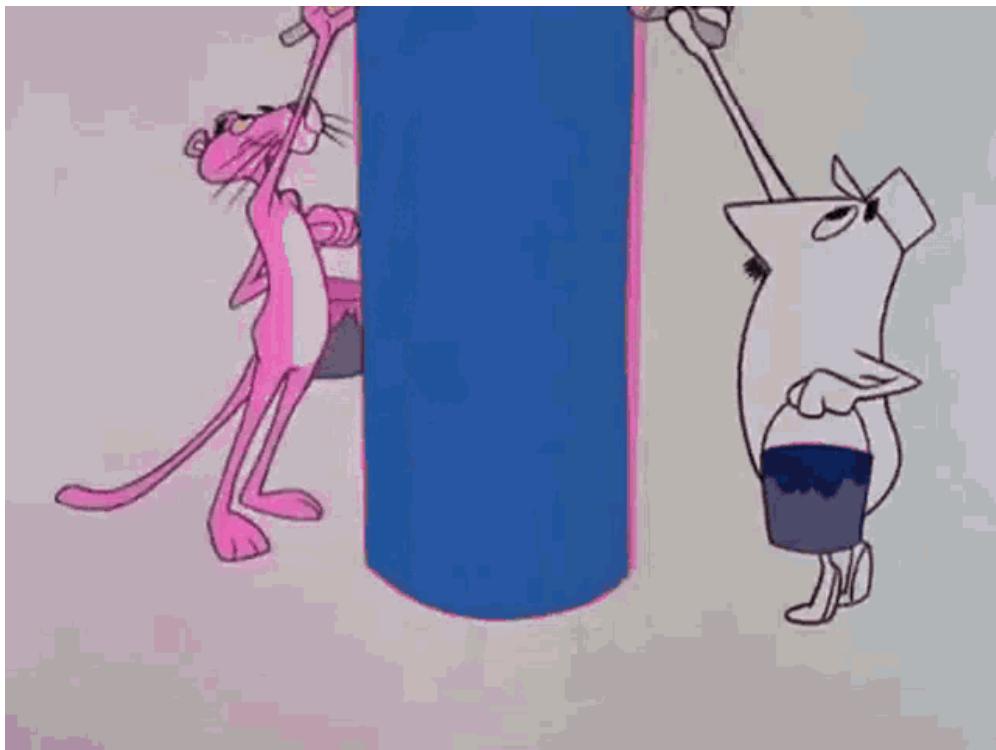
Caso práctico



@casfatesvano (<https://twitter.com/casfatesvano>) (CC BY-SA (<http://creativecommons.org/licenses/?lang=es>))

Lorena ha comenzado a explorar y tratar los datos que tiene, pero echa de menos poder "verlos". Necesita sacar algunas gráficas que, de un golpe de vista, le permitan sacar conclusiones sobre lo que representan y las relaciones entre ellos.

Decide importar la librería matplotlib, y en concreto el subpaquete pyplot, pues ha consultado la documentación disponible y parece que las funciones que le proporciona van a ser más de suficientes.



[luisgibi7](https://tenor.com/view/pintar-pait-pintura-work-trabalhar-gif-15327873) (<https://tenor.com/view/pintar-pait-pintura-work-trabalhar-gif-15327873>) (CC BY-SA (<http://creativecommons.org/licenses/?lang=es>))

[Matplotlib](https://matplotlib.org/) (<https://matplotlib.org/>) es la librería para Python que recoge un catálogo de lo más variado para hacer representaciones gráficas y de imágenes. Permite, incluso, hacer gráficas interactivas, con pocas líneas de código. Está basada en la gestión de arrays de NumPy y fue creada por John Hunter en 2002 como parche de IPython para poder ver los gráficos del formato Matlab generados con gnuplot. Como con otros proyectos de código abierto, la comunidad que se adhiere al mismo lo ha hecho evolucionar y crecer hasta convertir esta librería en una herramienta muy valiosa en cualquier desarrollo de exploración o investigación.

Aquí tienes varios ejemplos de formas de representación habituales. Si quieres, puedes [acceder al notebook original en colab](#) (<https://colab.research.google.com/drive/1LbSRZogB5iWqqMpvHpZlqht58cGib01c?usp=sharing>), crear una copia desde el menú archivo y ya puedes hacer tus propias versiones.

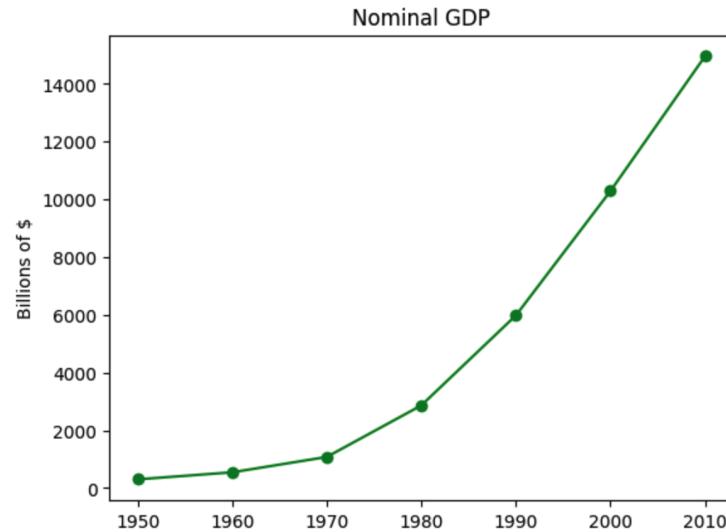
Empezamos por importar el subpaquete pyplot, que es el que realmente vamos a utilizar, desde matplotlib

```
✓  from matplotlib import pyplot as plt
```

Empezamos por algo sencillo, la figura que representa los valores de una función y (Producto Interior Bruto) para una sucesión de datos en x (años)

```
✓  [71] years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
  gdp = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]
```

```
✓  [72] plt.plot(years,gdp,color='green',marker='o',linestyle='solid')
  plt.title("Nominal GDP")
  plt.ylabel("Billions of $")
```



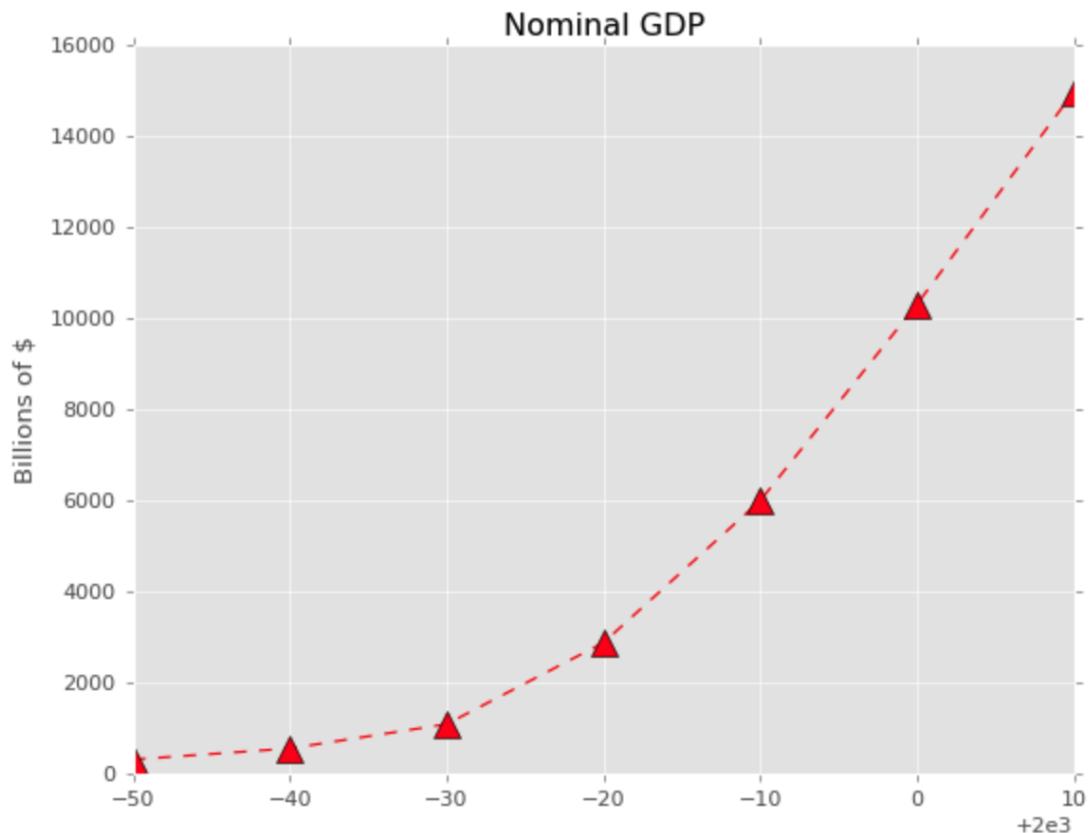
Carmen Bartolomé - Captura de pantalla (Dominio público)

Podemos utilizar distintos temas para los estilos de los gráficos

```
✓ [74] plt.style.use('ggplot')

plt.plot(years,gdp,color='red',marker='^', markersize=12,linestyle='dashed')
plt.title("Nominal GDP")

plt.ylabel("Billions of $")
plt.show()
```



```
✓ [75] plt.style.available
```

```
['Solarize_Light2',
'_classic_test_patch',
'bmh',
'classic',
'dark_background',
'fast',
'fivethirtyeight',
'ggplot',
'grayscale',
```

Carmen Bartolomé - Captura de pantalla (Dominio público)

Si queremos dibujar varias líneas, tenemos que llamar a la función plot por cada representación

```
✓ 0 s ➔ variance = [1, 2, 4, 8, 16, 32, 64, 128, 256]
    bias_squared = [256, 128, 64, 32, 16, 8, 4, 2, 1]
    total_error = [x + y for x, y in zip(variance, bias_squared)]

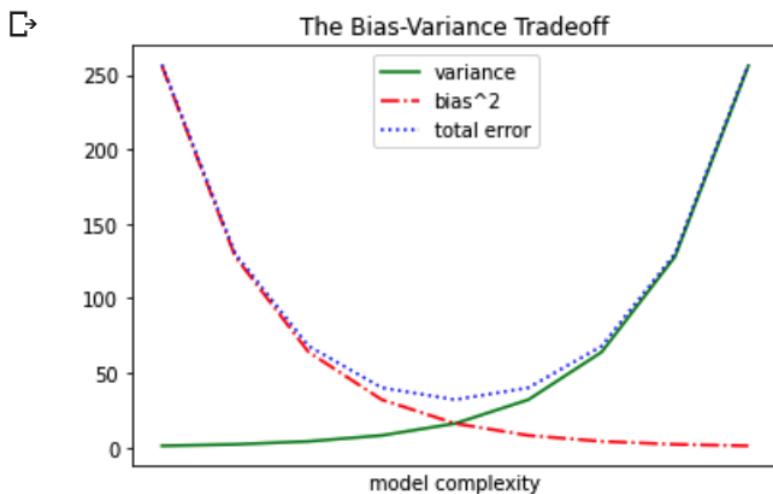
    xs = [i for i, _ in enumerate(variance)]

    # We can make multiple calls to plt.plot
    # to show multiple series on the same chart

    plt.plot(xs, variance, 'g-', label='variance') # green solid line
    plt.plot(xs, bias_squared, 'r-.', label='bias^2') # red dot-dashed line
    plt.plot(xs, total_error, 'b:', label='total error') # blue dotted line

    # Because we've assigned labels to each series,
    # we can get a legend for free (loc=9 means "top center")

    plt.legend(loc=9)
    plt.xlabel("model complexity")
    plt.xticks([])
    plt.title("The Bias-Variance Tradeoff")
    plt.show()
```



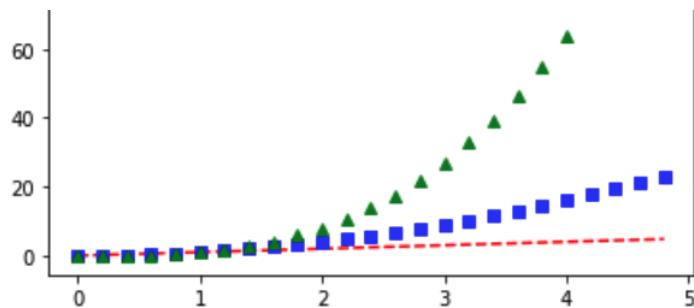
Otra opción para representar varias líneas en el mismo gráfico:

```
✓ 0 s [79] import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```





Carmen Bartolomé - Captura de pantalla (Dominio público)

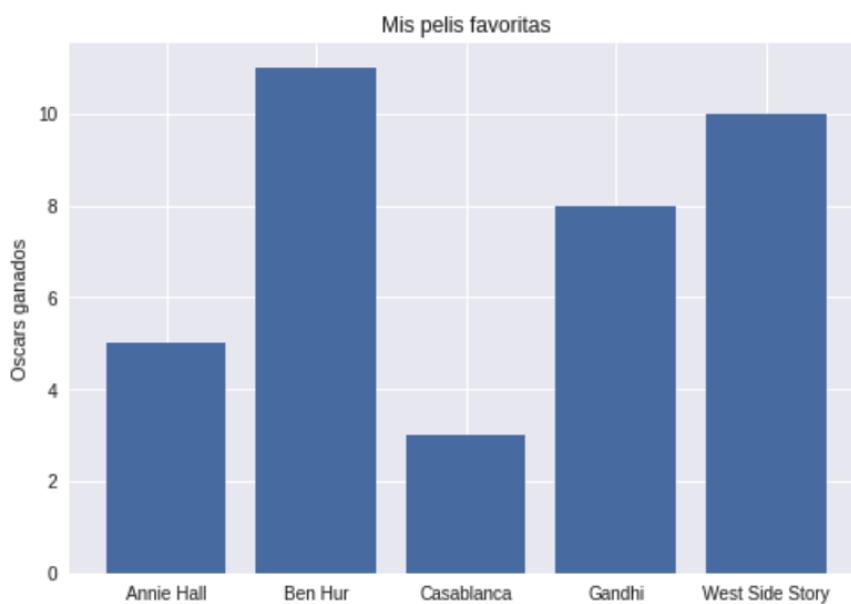
Otra función muy utilizada es la que permite representar diagramas de barras

```
[82] movies = ["Annie Hall", "Ben Hur", "Casablanca", "Gandhi", "West Side Story"]
     num_oscars = [5, 11, 3, 8, 10]

[83] plt.bar(range(len(movies)), num_oscars)
     plt.title("Mis pelis favoritas")
     plt.ylabel("Oscars ganados")

     plt.xticks(range(len(movies)), movies)

     plt.show()
```



Carmen Bartolomé - Captura de pantalla (Dominio público)

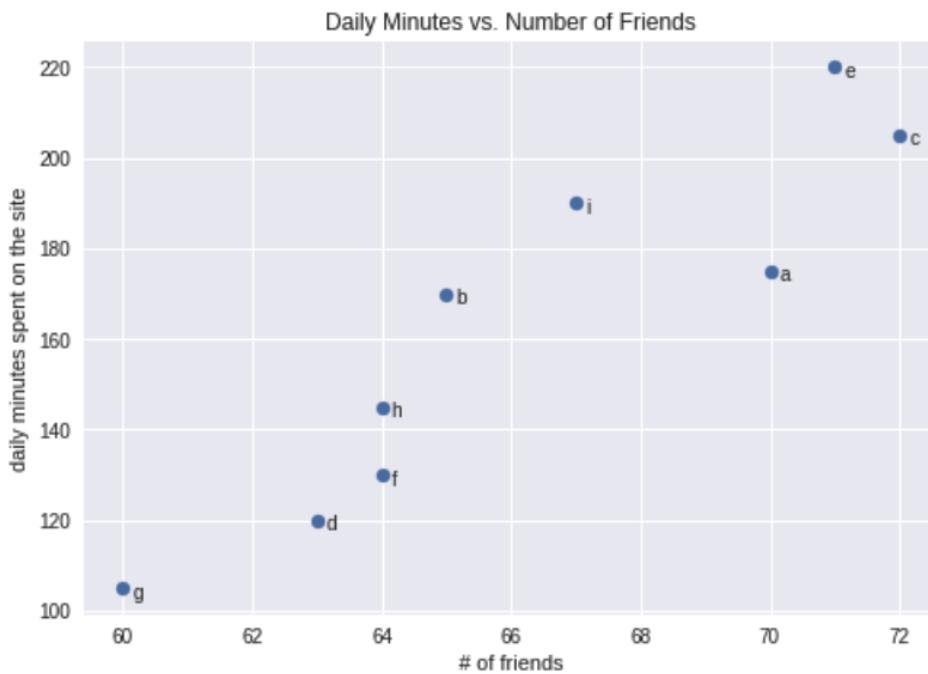
Probamos ahora la representación de diagrama de puntos (Scatterplot)

```
✓ 0 s [88] friends = [ 70, 65, 72, 63, 71, 64, 60, 64, 67]
     minutes = [175, 170, 205, 120, 220, 130, 105, 145, 190]
     labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

     plt.scatter(friends, minutes)

     # Ponemos una etiqueta a cada punto
     for label, friend_count, minute_count in zip(labels, friends, minutes):
         plt.annotate(label,
                     xy=(friend_count, minute_count), # cada etiqueta con su punto
                     xytext=(5, -5), # sepreamos el texto del punto
                     textcoords='offset points')

     plt.title("Daily Minutes vs. Number of Friends")
     plt.xlabel("# of friends")
     plt.ylabel("daily minutes spent on the site")
     plt.show()
```



Carmen Bartolomé - Captura de pantalla (Dominio público)

Una función muy interesante, es Scattergeo, que nos permite posicionar puntos en función de sus coordenadas sobre la representación de un mapa y permite cierto grado de interactividad.

```
✓ 0 s
import plotly.graph_objects as go

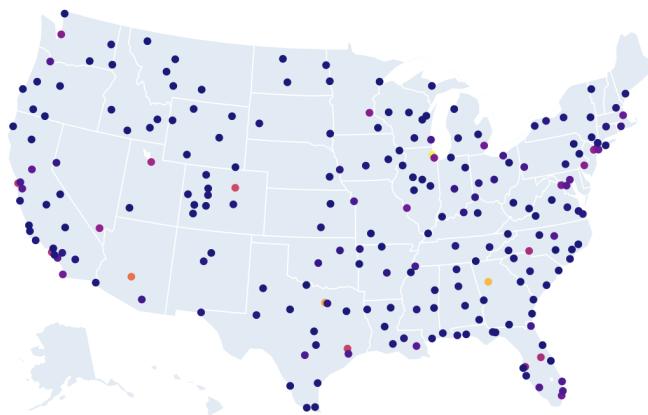
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/2011_february_us_airport_traffic.csv')
df['text'] = df['airport'] + ' ' + df['city'] + ', ' + df['state'] + ' ' + 'Arrivals: ' + df['cnt'].astype(str)

fig = go.Figure(data=go.Scattergeo(
    lon = df['long'],
    lat = df['lat'],
    text = df['text'],
    mode = 'markers',
    marker_color = df['cnt'],
))

fig.update_layout(
    title = 'Most trafficked US airports<br>(Hover for airport names)',
    geo_scope='usa',
)
fig.show()
```

Most trafficked US airports
(Hover for airport names)



Carmen Bartolomé - Captura de pantalla (Dominio público)

Para saber más

Matplotlib es una librería muy extensa, y es difícil llegar a controlar todas sus posibilidades sin recurrir a menudo a su [documentación](https://matplotlib.org/stable/api/index.html) (<https://matplotlib.org/stable/api/index.html>), que es muy clara y descriptiva. Te recomendamos el [tutorial de iniciación rápida](https://matplotlib.org/stable/tutorials/introductory/usage.html) (<https://matplotlib.org/stable/tutorials/introductory/usage.html>) para perderle el miedo y que elijas una chuleta de la [colección de "cheatsheets"](https://matplotlib.org/cheatsheets/) (<https://matplotlib.org/cheatsheets/>) que ofrecen, para tener siempre a mano y utilizar las funciones más frecuentes sin tener que ir a la documentación.



Version 3.5.0

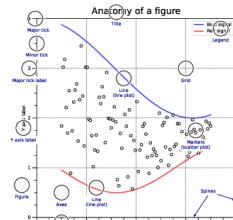
Quick start

```
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')
fig.show()
```

Anatomy of a figure



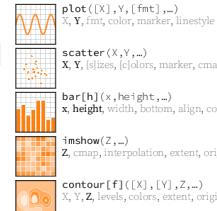
Subplots layout

```
subplot(s, rows, cols, ...)  
fig, ax = 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%')
```

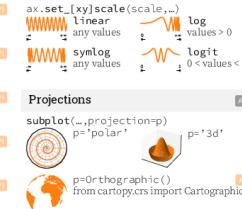
Getting help

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

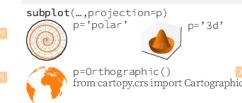
Basic plots



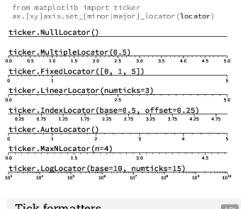
Scales



Projections



Tick locators

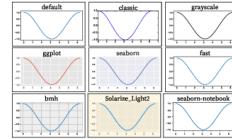


Animation

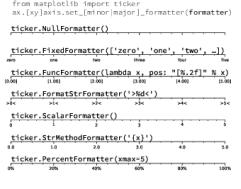
```
T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpl.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

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



Tick formatters



Ornaments



Quick reminder

```
ax.grid()  
ax.set_xlim(xmin, xmax)  
ax.set_xlabel(label)  
ax.set_xticks(ticks, [labels])  
ax.set_xticklabels(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.rcParams['axes', linewidth=1, ...]  
[fig|ax].patch.set_alpha(0)  
text=r'$\frac{1}{n} \prod_{i=1}^n p_i^{a_i}$'
```

```
Keyboard shortcuts
```

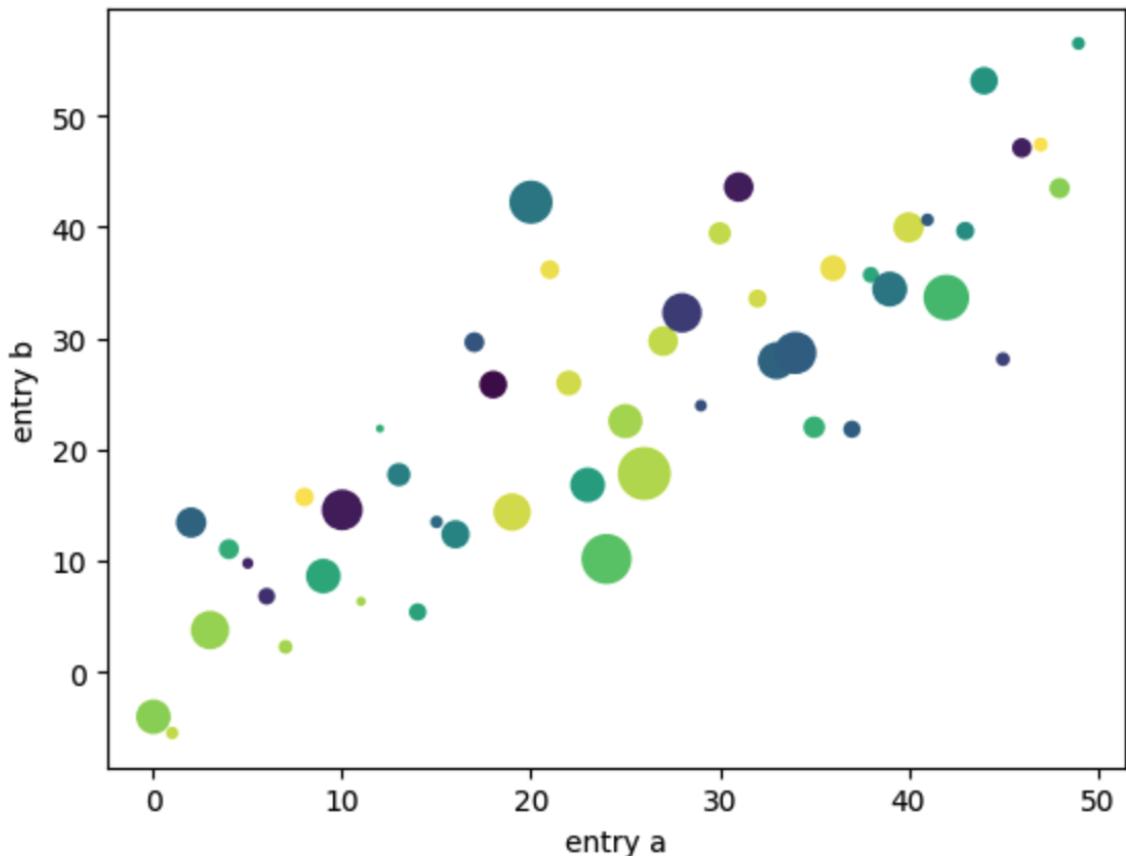
ctrl + s	Save	ctrl + w	Close plot
ctrl + r	Reset view	ctrl + f	Fullscreen 0/1
ctrl + f	View forward	ctrl + b	View back
p	Pan view	ctrl + z	Zoom to rect
x	X pan/zoom	y	Y pan/zoom
g	Minor grid 0/1	g	Major grid 0/1
x	X axis log/linear	y	Y axis log/linear

Ten simple rules

1. Know Your Audience
2. Identify Your Message
3. Adapt the Figure
4. Captions Are Not Optional
5. Do Not Throw the Defaults
6. Use the Right Tools
7. Do Not Mislead the Reader
8. Avoid "Chartjunk"
9. Message Trumps Beauty
10. Get the Right Tool

Autoevaluación

Introduce el código necesario en el espacio en blanco para poder hacer una representación de tipo dispersión de puntos como la de esta imagen



Carmen Bartolomé - Elaboración propia (Dominio público)

```

data = {'a': np.arange(50),'c': np.random.randint(0, 50, 50),'d': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100
plt._____ ('a', 'b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()

```

El método para poder hacer una representación típica de dispersión de puntos es *scatter*.

3.- Scikit learn.

Caso práctico



[@casfatesvano](https://twitter.com/casfatesvano) ([CC BY-SA](https://creativecommons.org/licenses/?lang=es))
<http://creativecommons.org/licenses/?lang=es>)

Lorena ha podido hacer todo un EDA (Exploratory Data Analysis) de los datos que tenía. Ahora que ya sabe cuáles son las diferentes variables de entrada y también tiene claro los que quiere predecir, es el momento de crear un modelo de aprendizaje automático. Decide empezar por los modelos más básicos, y, según el grado de acierto de cada uno, ir orientando su búsqueda de un buen modelo.

En primer lugar, partiendo de la base de que tiene datos "etiquetados" y va a aplicar aprendizaje supervisado, debe decidir si está ante un problema de regresión o de clasificación. De momento, va a estudiar si los pedidos llegarán o no en el plazo máximo de entrega que les dan a los clientes, pero, más adelante, es posible que sea bueno poder predecir el plazo de entrega en horas desde que el pedido es realizado. El primero, es un problema de clasificación, mientras que el segundo lo sería de regresión.

 Install User Guide API Examples Community More ▾

scikit-learn

Machine Learning in Python

Getting Started Release Highlights for 1.1 GitHub

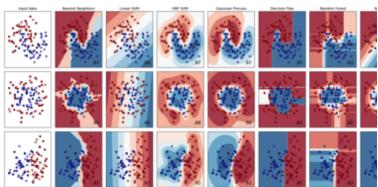
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: SVM, nearest neighbors, random forest, and more...



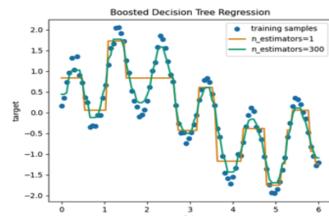
Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, nearest neighbors, random forest, and more...



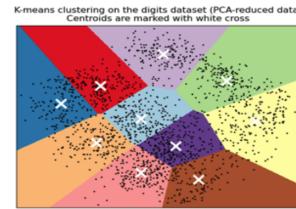
Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, and more...



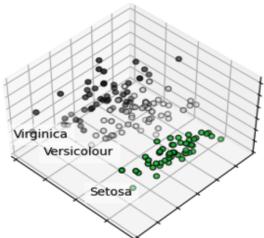
Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: PCA, feature selection, non-negative matrix factorization, and more...

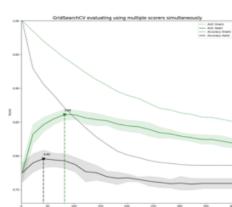


Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning

Algorithms: grid search, cross validation, metrics, and more...

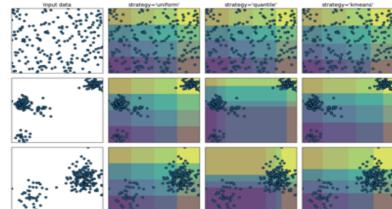


Preprocessing

Feature extraction and normalization.

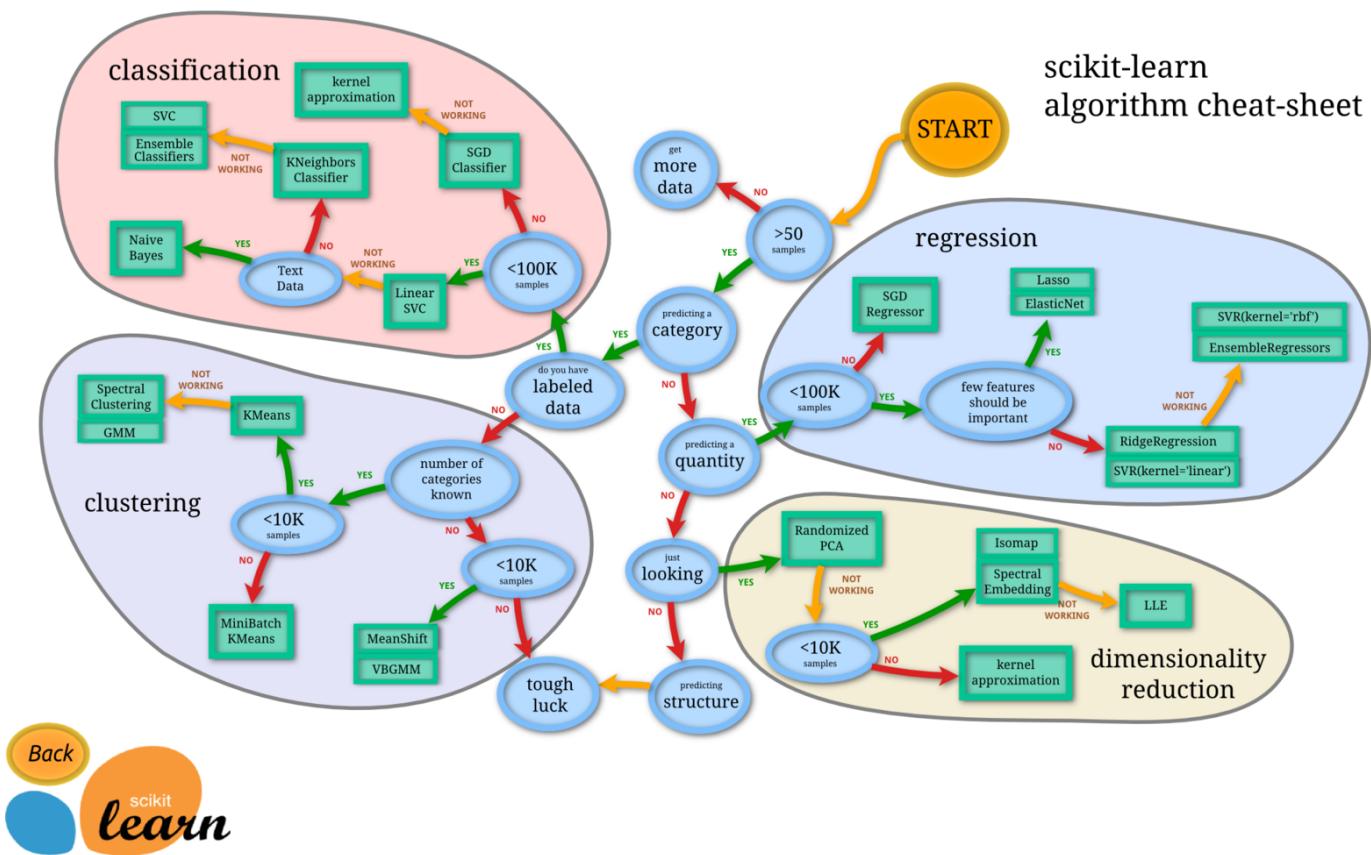
Applications: Transforming input data such as text for use with machine learning algorithms.

Algorithms: preprocessing, feature extraction, and more...



[Scikit-learn \(<https://scikit-learn.org/stable>\)](https://scikit-learn.org/stable) (Dominio público)

Scikit learn (<https://scikit-learn.org/stable/>) es una de las librerías de aprendizaje automático más utilizadas en todo el mundo. Surgió como un proyecto de Google Summer of Code de la mano de David Cournapeau. Su concepto inicial era ser una extensión auxiliar a Scipy, de la que provenía. La comunidad se esforzó en contribuir para que Scikit-learn constituyese una distribución independiente de SciPy, y lo lograron en 2010, con la primera distribución pública. Dada su naturaleza para fines científicos, algunos algoritmos de su núcleo están escritos en Cython para conseguir mayor rendimiento en cálculos muy pesados, pero la mayoría está escrito en Python. Es muy utilizada para aplicar regresión lineal, regresión logística, random forest y máquinas de vector soporte. Aunque permite crear modelos de redes neuronales, no se suele utilizar para esa técnica tras la aparición de Tensorflow. Destaca, entre su documentación, este esquema para elegir el algoritmo más adecuado para el problema que queramos tratar:



[Wikipedia \(<https://es.wikipedia.org/wiki/Scikit-learn>\)](https://es.wikipedia.org/wiki/Scikit-learn) (CC BY-SA (<http://creativecommons.org/licenses/?lang=es>)))

Veamos un ejemplo de aplicación a un [dataset muy apreciado de Kaggle sobre entrega en plazo de pedidos](https://www.kaggle.com/datasets/prachi13/customer-analytics) (<https://www.kaggle.com/datasets/prachi13/customer-analytics>) en el que se aplican varias técnicas y se compara la métrica de acierto al final, pudiendo deducir qué técnica era la más interesante para seguir trabajando sobre ella. Puedes acceder al [notebook original](https://colab.research.google.com/drive/1zixmkzgYpLIQC8O72r62R7v4Vh4L1CjI?usp=sharing) (<https://colab.research.google.com/drive/1zixmkzgYpLIQC8O72r62R7v4Vh4L1CjI?usp=sharing>) para hacer una copia en el menú archivo y poder editarla a tu gusto.

Para saber más

Te recomendamos que profundices más en las posibilidades de esta librería [consultando su documentación](https://scikit-learn.org/stable/modules/classes.html) (<https://scikit-learn.org/stable/modules/classes.html>), así como practicar con los [ejemplos](https://scikit-learn.org/stable/auto_examples/index.html) que ofrecen en su web (https://scikit-learn.org/stable/auto_examples/index.html).

Autoevaluación

Rellena los espacios en blanco con el código necesario para crear y entrenar un modelo de regresión logística utilizando la librería Scikit learn

```
from sklearn.linear_model import LogisticRegression
```

```
data = pd.read_csv('data.csv')
y = data['result']
X = data.drop('result', axis=1)
model = ()
model.  (X, y)
```

La clase para el modelo es LogisticRegression().

El método para hacer el entrenamiento es la función fit.

4.- Tensorflow y Keras.

Caso práctico



@casfatesvano (<https://twitter.com/casfatesvano>) (CC BY-SA (<http://creativecommons.org/licenses/?lang=es>))

Lorena ha conseguido llegar, utilizando la librería Scikit learn, a varios modelos que logran una tasa de acierto en las predicciones de un 70%. Realmente no ha tenido que dedicar demasiado tiempo ni recursos para llegar a ésto, y ha automatizado un sistema de alerta que le envía un email al Director de Logística de Pick&Deliver con los pedidos del día que presentan riesgo de no cumplir con el plazo de entrega. Al recibir la alerta temprana, el Director de Logística, ahora, puede sacar esos pedidos del flujo normal y pasarlos a envío preferente, de forma que evitan el retraso antes de que sea demasiado tarde.

Pero Lorena sabe que el modelo solo acierta en un 30%, y de hecho, algunos pedidos siguen llegando fuera de plazo. Es necesario hacerlo mejor, y, por eso, Lorena, en cuanto ha dejado listo el sistema de alerta, se ha puesto a explorar un posible desarrollo de deep learning, que es una técnica mucho más precisa.



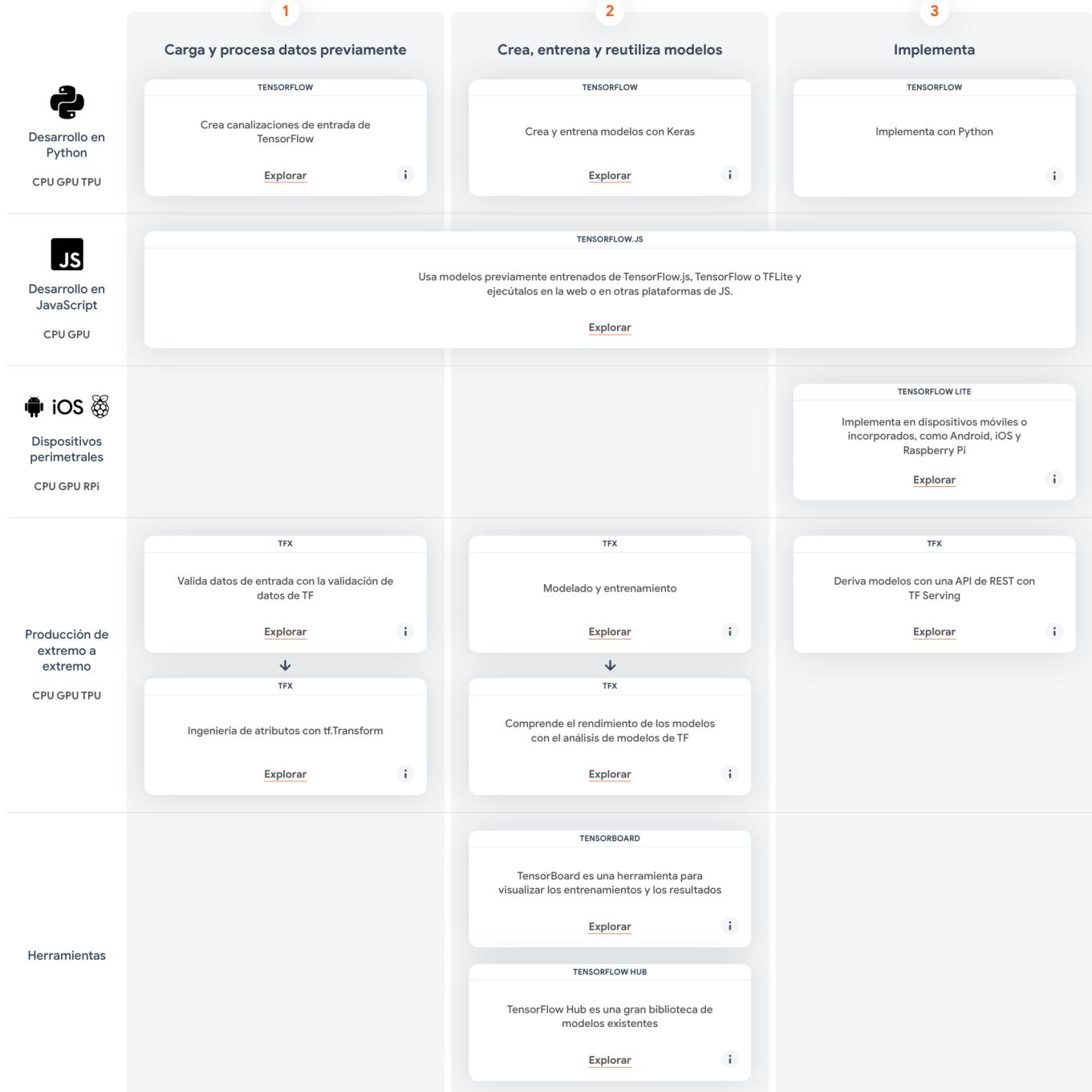
TensorFlow

[Tensorflow \(https://www.tensorflow.org/\)](https://www.tensorflow.org/) (Dominio público)

[Tensorflow \(https://www.tensorflow.org\)](https://www.tensorflow.org/) ha sido la librería que ha marcado un antes y un después en el desarrollo de inteligencia artificial a nivel mundial. Fue desarrollado originalmente por el equipo de Google Brain para su uso interno en Google hasta que fue publicado con licencia de código abierto Apache en 2015. Antes de que pasase un año desde su liberalización, ya aparecía en cerca de 1500 repositorios de GitHub, de los cuales, solo 5 eran de Google. Fue rápidamente absorbido por la comunidad en torno al aprendizaje automático, que ya contaba con otros incentivos, como las competiciones de Kaggle, y las nuevas opciones de hardware.

En 2019 sale la versión alfa de TensorFlow 2, que incluye Keras, desarrollada por François Chollet específicamente para Tensorflow, con módulos de alto nivel que simplifican la programación de redes neuronales profundas lo suficiente como para dar un segundo impulso a la comunidad y que se produzca un avance exponencial en la aparición de nuevas arquitecturas de algoritmos, y nuevas aplicaciones basadas en ellos.

El ecosistema actual de Tensorflow incluye herramientas complementarias como Tensorboard, una interfaz para visualizar entrenamientos y resultados, o Tensorflow Lite, para integrar los modelos en dispositivos móviles en modo de cómputo en el perímetro (edge computing).



[tensorflow.org \(https://www.tensorflow.org/learn\)](https://www.tensorflow.org/learn) (Dominio público)

La implementación de [Keras \(https://keras.io/\)](https://keras.io/) fue todo un acierto, y ya se utiliza en la mayoría de desarrollos de modelos habituales con los parámetros más comunes. Según su creador, está totalmente orientada a los seres humanos, a que programen inteligencia artificial de forma cómoda y sin necesidad de saturarse mentalmente.



Keras

Simple. Flexible. Powerful.

keras.io (<https://keras.io/>) (Dominio público)

Reduce las acciones necesarias para la programación de la mayoría de casos, y sus mensajes de error son más claros y efectivos. Finalmente, se ha convertido en el núcleo central de Tensorflow, siendo imposible considerar ambas por separado. Y el factor determinante que ha extendido su uso, es que proporciona una experiencia de aprendizaje mucho más amigable a todos los que se están introduciendo en este campo. Por lo tanto, te animamos a que aproveches al máximo todo el potencial de esta librería y practiques mucho. A modo de introducción, te presentamos un ejemplo muy sencillo de aplicación de Keras al problema de reconocimiento de dígitos manuscritos. ¡Abre una sesión en colab y ponte manos a la obra! Si lo necesitas, puedes acceder al [cuaderno original](https://colab.research.google.com/drive/1yAi3PfntYflw2Gp2GRiaZG-80eAI_BQt?usp=sharing) (https://colab.research.google.com/drive/1yAi3PfntYflw2Gp2GRiaZG-80eAI_BQt?usp=sharing) y hacer una copia en tu Drive desde el menú Archivo.

Utilización de keras para reconocimiento de dígitos manuscritos

Datos de partida: utilizaremos el [dataset MNIST, recopilado y compartido por Yann LeCun](http://yann.lecun.com/exdb/mnist/) (<http://yann.lecun.com/exdb/mnist/>) y que podemos cargar desde la propia librería de Keras.

1. Carga el dataset MNIST de Keras

Empezamos por importar el dataset desde la librería Keras y cargarlo con el método `load_data()`

In []:

```
from keras.datasets import mnist  
  
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz  
11493376/11490434 [=====] - 0s 0us/step  
11501568/11490434 [=====] - 0s 0us/step
```

2. Construcción del modelo

Importamos los módulos para crear un modelo y para utilizar la arquitectura por capas típica del deep learning. Creamos el objeto network, que será nuestro modelo de la clase Sequential(), que es la que admite varias capas de profundidad. Después, con el método add, iremos apilando capas, en este caso, de la clase Dense, que las de la arquitectura típica de red neuronal (densely connected). Una primera capa tendrá el atributo de 512 neuronas y función de activación "relu" y la siguiente capa tendrá 10 neuronas y función de activación "softmax"

In []:

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation = 'relu', input_shape = (28*28,)))
network.add(layers.Dense(10, activation = 'softmax'))
```

3. Fijar los parámetros del entrenamiento

Con el método compile(), indicamos el optimizador a utilizar, el tipo de función de coste (la relacionada con el error entre el resultado del modelo y las etiquetas dadas) y la métrica que queremos ir monitorizando durante el entrenamiento, que será 'accuracy' (precisión o casos correctamente clasificados).

In []:

```
network.compile(optimizer = 'rmsprop',
                 loss = 'categorical_crossentropy',
                 metrics = ['accuracy'])
```

4. Preparando las imágenes y las etiquetas

Antes de pasar al entrenamiento, necesitamos hacer algunos ajustes. Por un lado, tenemos que cambiar las dimensiones del array que entrará en el modelo, pasando a forma vector. Por otro, adimensionalizamos los valores de intensidad de los pixeles de las imágenes, para que estén entre 0 y 1 (antes estaban entre 0 y 255) y los convertimos en tipo float32.

In []:

```
train_images = train_images.reshape((60000, 28*28))
train_images = train_images.astype('float32')/255

test_images = test_images.reshape((10000, 28*28))
test_images = test_images.astype('float32')/255
```

Tambien va a ser necesario pasar las etiquetas a tipo "categórico"

In []:

```
from tensorflow.keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

5. Entrenamiento del modelo

Llegamos al paso culminante: el entrenamiento del modelo. Se hace utilizando el método fit(), con los parámetros imprescindibles del conjunto de datos de entrada, el número de iteraciones de cálculo, y el tamaño del lote de datos a utilizar en cada iteración.

In []:

```
network.fit(train_images, train_labels, epochs = 5, batch_size = 128)
```

```
Epoch 1/5  
469/469 [=====] - 6s 11ms/step - loss: 0.2581 - accuracy: 0.9253  
Epoch 2/5  
469/469 [=====] - 5s 11ms/step - loss: 0.1039 - accuracy: 0.9693  
Epoch 3/5  
469/469 [=====] - 5s 11ms/step - loss: 0.0686 - accuracy: 0.9793  
Epoch 4/5  
469/469 [=====] - 5s 11ms/step - loss: 0.0509 - accuracy: 0.9844  
Epoch 5/5  
469/469 [=====] - 6s 12ms/step - loss: 0.0375 - accuracy: 0.9885
```

Out[]:

```
<keras.callbacks.History at 0x7f8a554365d0>
```

6. Evaluación del modelo entrenado

Ahora probamos el modelo entrenado utilizando el método evaluate() y los datos de test.

In []:

```
test_loss,test_acc = network.evaluate(test_images, test_labels)  
print("test_acc: ", test_acc)
```

```
313/313 [=====] - 1s 3ms/step - loss: 0.0716 - accuracy: 0.9794  
test_acc: 0.9793999791145325
```

Podemos ver que se obtiene una precisión del 97% con los nuevos datos de test.

Para saber más

Tienes a tu disposición la [documentación de Tensorflow](https://www.tensorflow.org/api_docs/python/tf) (https://www.tensorflow.org/api_docs/python/tf), pero es mucho mejor explorar los recursos que comparten públicamente, como una gran [colección de datasets](#)

(<https://www.tensorflow.org/resources/models-datasets>), o el [catálogo de herramientas](https://www.tensorflow.org/resources/tools) (<https://www.tensorflow.org/resources/tools>). de apoyo que complementarán tus procesos de desarrollo de aprendizaje automático con Tensorflow.

Como hemos comentado, Keras es un subpaquete integrado dentro de Tensorflow. La [documentación de Keras](https://keras.io/api/) (<https://keras.io/api/>) es muy extensa y completa, y cuenta también con un [conjunto de datasets](https://keras.io/api/datasets/) (<https://keras.io/api/datasets/>) integrados en la propia librería y un [gran repositorio de ejemplos](https://keras.io/examples/) (<https://keras.io/examples/>) más que recomendable.

Autoevaluación

Rellena el hueco en blanco para que el código a continuación genere un modelo de tipo Sequential (a capas) con dos capas, una de 128 neuronas y otra de 10 neuronas.

```
from keras import models  
from keras import layers  
model = models._____()  
model.add(layers.Dense(_____, activation='relu')  
model.add(layers.Dense(_____, activation='softmax'))
```

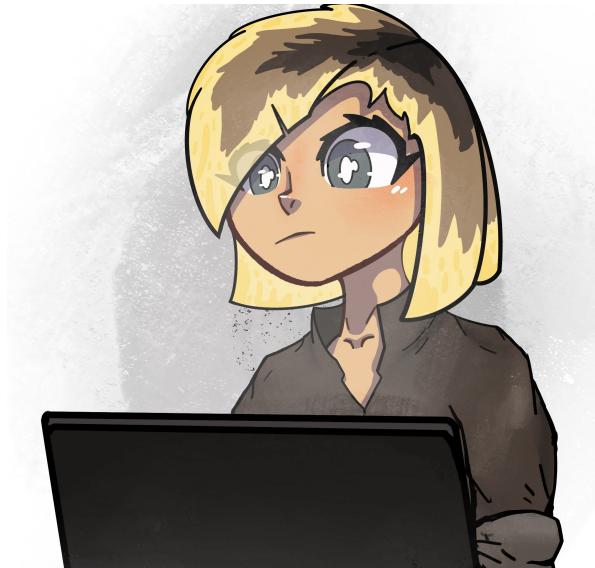
Enviar

La clase correspondiente de un modelo de capas es Sequential()

En la primera capa, al introducir 128, estás indicando que esa capa debe tener 128 neuronas, y, análogamente, en la segunda capa, al introducir 10, estás indicando que esa capa debe tener 10 neuronas.

5.- Pytorch.

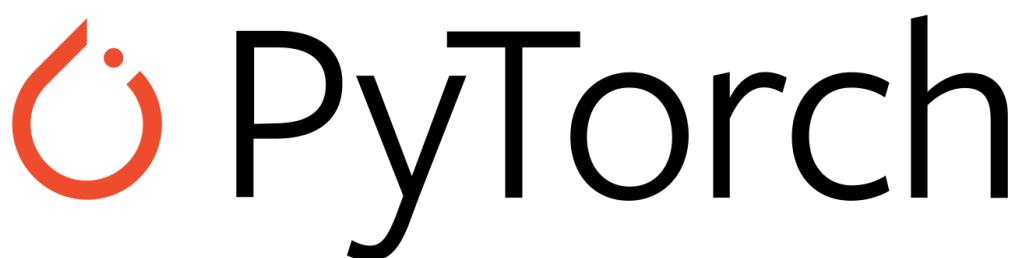
Caso práctico



[@casfatesvano](https://twitter.com/casfatesvano) ([CC BY-SA](https://creativecommons.org/licenses/?lang=es))
<http://creativecommons.org/licenses/?lang=es>)

Lorena ya ha conseguido desarrollar sus primeros modelos de deep learning. Está bastante satisfecha con la precisión que dan estos modelos basados en redes neuronales profundas. En su búsqueda de ejemplos y modelos propuestos por la comunidad, descubre que muchos profesionales alaban una librería llamada Pytorch. Elena empieza a indagar sobre ella, y la prueba con algunos de los datasets con los que había probado Keras. ¿Será esta librería la mejor? Con sus primeras pruebas, descubre que suele dar una precisión algo mayor, pero es más compleja a la hora de programar los algoritmos.

Tras algunas consultas a miembros de la comunidad, deduce que es una librería muy potente, pero que requiere un grado de especialización y un tiempo de aprendizaje mayores. ¡Habrá que seguir aprendiendo a la vez que continua las pruebas con modelos de Tensorflow y Keras!



pytorch.org (<https://pytorch.org/>). (Dominio público)

[Pytorch](https://pytorch.org/) (<https://pytorch.org/>) es una librería de código abierto para aprendizaje automático basada en Torch para Python. Fue creada por desarrolladores del FAIR, el Laboratorio de Investigación de Inteligencia Artificial de Facebook. Está optimizada para el cálculo con tensores, característico de los modelos de redes neuronales profundas, y tiene bien integradas las herramientas que permiten hacer, incluso, la implementación del modelo en producción. Cabe desatacar la librería [fastai](https://docs.fast.ai/) (<https://docs.fast.ai/>), que simplifica bastante el desarrollo de inteligencia artificial, utilizando modelos pre-entrenados que ya vienen cargados en la librería. Puedes probar lo ágil que es el

desarrollo con fast ai con este breve cuaderno (<https://colab.research.google.com/drive/1vBLOneaywiwGuXrzvar7SSFI4HGzqIUz?usp=sharing>), que con muy pocas líneas de código, nos proporciona un modelo para reconocer el tipo de mascota.

En el siguiente cuaderno, puedes ver la creación y entrenamiento de un modelo de red neuronal de dos capas para la clasificación de dígitos manuscritos del dataset MNIST. Es el mismo problema que hemos visto en la sección anterior, en el ejemplo de keras.

Instalamos el módulo de visión artificial de Pytorch

In [23]:

```
!pip install torch  
!pip install torchvision
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/  
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (1.12.0+cu113)  
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch) (4.1.1)  
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/  
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-packages (0.13.0+cu113)  
Requirement already satisfied: torch==1.12.0 in /usr/local/lib/python3.7/dist-packages (from torchvision) (1.12.0+cu113)  
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torchvision) (4.1.1)  
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from torchvision) (2.23.0)  
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.7/dist-packages (from torchvision) (7.1.2)  
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torchvision) (1.21.6)  
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->torchvision) (3.0.4)  
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->torchvision) (1.24.3)  
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->torchvision) (2.10)  
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->torchvision) (2022.6.15)
```

Importamos los paquetes necesarios

In [24]:

```
import torch  
import torch.nn as nn  
import torchvision.datasets as dsets  
import torchvision.transforms as transforms  
from torch.autograd import Variable
```

Definimos los parámetros para el modelo y el entrenamiento

In [25]:

```
input_size = 784 # img_size = (28,28) ---> 28*28=784 in total
hidden_size = 500 # number of nodes at hidden layer
num_classes = 10 # number of output classes discrete range [0,9]
num_epochs = 20 # number of times which the entire dataset is passed throughout the mode
batch_size = 100 # the size of input data took for one iteration
lr = 1e-3 # size of step
```

Descargamos los datos desde el repositorio de Yann Lecun y los cargamos

In [26]:

```
train_data = dsets.MNIST(root = './data', train = True,
                         transform = transforms.ToTensor(), download = True)

test_data = dsets.MNIST(root = './data', train = False,
                        transform = transforms.ToTensor())
```

In [27]:

```
train_gen = torch.utils.data.DataLoader(dataset = train_data,
                                         batch_size = batch_size,
                                         shuffle = True)

test_gen = torch.utils.data.DataLoader(dataset = test_data,
                                       batch_size = batch_size,
                                       shuffle = False)
```

Definimos la clase para el modelo y lo generamos

In [28]:

```
class Net(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

In [29]:

```
net = Net(input_size, hidden_size, num_classes)
if torch.cuda.is_available():
    net.cuda()
```

Definimos la función de coste (Loss) y el optimizador

In [30]:

```
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam( net.parameters(), lr=lr)
```

Entrenamos el modelo

In [31]:

```
for epoch in range(num_epochs):
    for i ,(images,labels) in enumerate(train_gen):
        images = Variable(images.view(-1,28*28)).cuda()
        labels = Variable(labels).cuda()

        optimizer.zero_grad()
        outputs = net(images)
        loss = loss_function(outputs, labels)
        loss.backward()
        optimizer.step()

#    if (i+1) % 100 == 0:
#        print('Epoch [%d/%d], Step [%d/%d], Loss: %.4f'
#              %(epoch+1, num_epochs, i+1, len(train_data)//batch_size, loss.data))

    print('Epoch [%d/%d], Loss: %.4f'
          %(epoch+1, num_epochs, loss.data))
```

```
Epoch [1/20], Loss: 0.1017
Epoch [2/20], Loss: 0.1060
Epoch [3/20], Loss: 0.0460
Epoch [4/20], Loss: 0.0091
Epoch [5/20], Loss: 0.0695
Epoch [6/20], Loss: 0.0298
Epoch [7/20], Loss: 0.0057
Epoch [8/20], Loss: 0.0208
Epoch [9/20], Loss: 0.0094
Epoch [10/20], Loss: 0.0068
Epoch [11/20], Loss: 0.0017
Epoch [12/20], Loss: 0.0017
Epoch [13/20], Loss: 0.0109
Epoch [14/20], Loss: 0.0022
Epoch [15/20], Loss: 0.0008
Epoch [16/20], Loss: 0.0004
Epoch [17/20], Loss: 0.0070
Epoch [18/20], Loss: 0.0441
Epoch [19/20], Loss: 0.0007
Epoch [20/20], Loss: 0.0030
```

Evaluamos el modelo

In [32]:

```
correct = 0
total = 0
for images,labels in test_gen:
    images = Variable(images.view(-1,28*28)).cuda()
    labels = labels.cuda()

    output = net(images)
    _, predicted = torch.max(output,1)
    correct += (predicted == labels).sum()
    total += labels.size(0)

print('Accuracy of the model: %.3f %%' %((100*correct)/(total+1)))
```

Accuracy of the model: 98.280 %

Para saber más

Existe bastante material para profundizar en Pytorch. Cuenta con una [documentación muy descriptiva para la API Torch](#) (<https://pytorch.org/docs/stable/index.html>), y una [guía](#) (<https://pytorch.org/get-started/locally/>) para poder empezar a trabajar tanto localmente, como en la nube. Es especialmente interesante recurrir a su [hub de modelos pre-entrenados](#) (<https://pytorch.org/hub/>), entre los que está su conocido YOLO. Cuenta, también, con un [catálogo de tutoriales](#) (<https://pytorch.org/tutorials/>) para ir familiarizándose con los módulos de Pytorch.

Te recomendamos especialmente el [MOOC](#) (<https://course.fast.ai/>) que ofrece gratuitamente el equipo de fast.ai, basado en gran parte en el uso de Pytorch, y que tiene una gran aceptación en la comunidad.

Autoevaluación

Dado el siguiente código que define la clase a utilizar a la hora de generar un modelo de red neuronal profunda con Pytorch, ¿Cuál es la llamada correcta para crear el modelo?

```
class Net(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```



```
net = Net(input_size, hidden_size, num_classes)
```



```
net = NN(input_size, hidden_size, num_classes)
```



```
model = keras.model.Sequential(input_size, hidden_size, num_classes)
```

Opción correcta

La clase se ha definido con el nombre Net, no con NN

Estamos usando la librería Pytorch, no keras

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto