class6

August 6, 2018

```
In [1]: %pylab inline

Populating the interactive namespace from numpy and matplotlib
```

1 Clase 6 - Análisis y ciencia de datos

- La ciencia necesita datos.
- El método científico: hacer hipótesis, adquirir **datos**, analizar **datos**, refutar/apoyar hip'otesis.
- En astronomía, meteorología, física, la cantidad de datos disponible (hoy en día) es enorme. ¿Cómo podemos analizarlos?

1.1 El propósito del análisis de datos

- Buscamos **información** dentro de los datos para apoyar o refutar una hipótesis científica.
- Este es la actividad fundamental de cualquier científico trabajando en un area empírica: e.g. astronomía, meteorología, física experimental, física computacional, etc.
- En la física teoríca se extrae "información" de las ecuaciones y teorías, en vez de datos.

1.2 ¿Qué son los datos?

Depende del 'area de trabajo...

- En un sentido general, los datos están compuestos de parámetros o cantidades que describen un sistema, o un aspecto del sistema que estamos estudiando.
- Ejemplo: para una estrella variable, podrían incluir la luminosidad de la estrella, su período de variabilidad, su tipo espectral, etc.
- Ejemplo: para un experimento de la física de partículas, podrían incluir la taza de producción de una partícula específica.
- Ejemplo: para una simulación computacional de la atmósfera, podrían incluir la presión, la temperatura, y la densidad del aire, como calculado en la simulación en ciertos puntos en la superficie de la Tierra, y en ciertos momentos.
- Con un conjunto de datos, podemos comenzar con el análisis.
- En esta clase, seguiremos un ejemplo en climatología, con datos de la temperatura global promedio mensual (dataset: HadCRUT4, del UK Met Office, https://crudata.uea.ac.uk/cru/data/temperature/).

1.3 Pasos principales del análisis de datos:

1.3.1 1. Explorar y procesar los datos.

1.3.2 2. Aplicar análisis

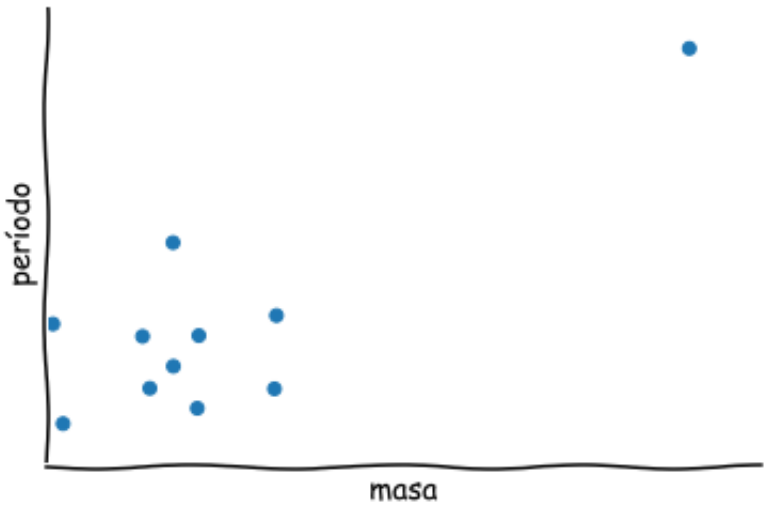
1.3.3 3. Visualizar el resultado.

1.3.4 1. Exploración de los datos

Primero, simplemente "mirando" a los datos (i.e. verificando el contenido del archivo de datos, haciendo un gráfico simple) es un paso importante, y frecuentemente olvidado...

Nuestros cerebros son muy buenos en procesar información visual, por lo tanto una imágen visual de los datos puede otorgar información útil inmediatemente. Podemos ver inmediatemente si hay distintos grupos de datos, o valores que son muy diferentes de los demás.

Por ejemplo, si tenemos observaciones de estrellas variables, y graficamos sus masas y sus períodos, quizás veremos uno o dos puntos que están muy lejos del resto del grupo:



Puntos como eso se llaman *outliers* en inglés.

Podría ser que los datos tienen un error, o podría ser algo nuevo y interesante...

En nuestro ejemplo, tenemos un archivo de texto que contiene información del promedio global de la temperatura. Podemos leer las primeras 10 líneas del archivo con "head":

```
In [2]: !head HadCRUT.4.6.0.0.monthly_ns_avg_mod.txt
```

```
1850 01 -0.700
1850 02 -0.286
1850 03 -0.732
1850 04 -0.563
1850 05 -0.327
1850 06 -0.213
```

```
1850 07 -0.125
1850 08 -0.237
1850 09 -0.439
1850 10 -0.451
```

Yo he procesado un poco el archivo de datos que viene de la página web.

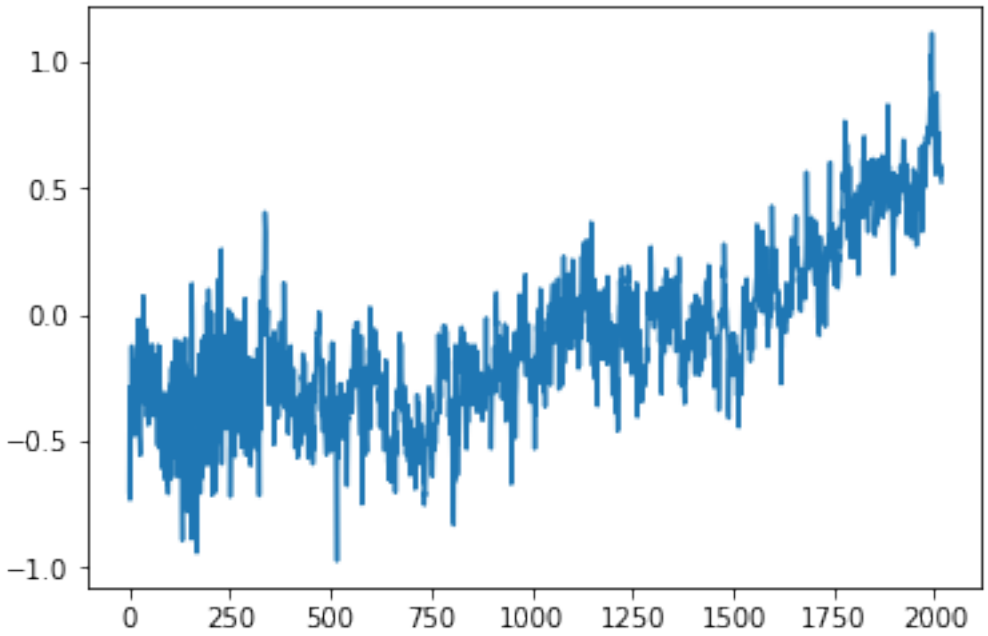
Aquí tenemos 3 columnas, la primera es el año, la segunda es el mes, y la tercera es la *anomalía en la tempertura promedia* (diferencia comparada con el promedio de los años 1961-1990).

```
In [70]: # !cat HadCRUT.4.6.0.0.monthly_ns_avg_mod.txt
```

Parece que no hay datos faltantes, ni valores muy grandes ni muy pequeños: los datos se ven "normal"...

Ahora cargamos los datos en Python, y hacemos un gráfico simple:

```
In [4]: datos = loadtxt("HadCRUT.4.6.0.0.monthly_ns_avg_mod.txt")
        plot(datos[:,2])

Out[4]: [<matplotlib.lines.Line2D at 0x7f7a458879e8>]
```



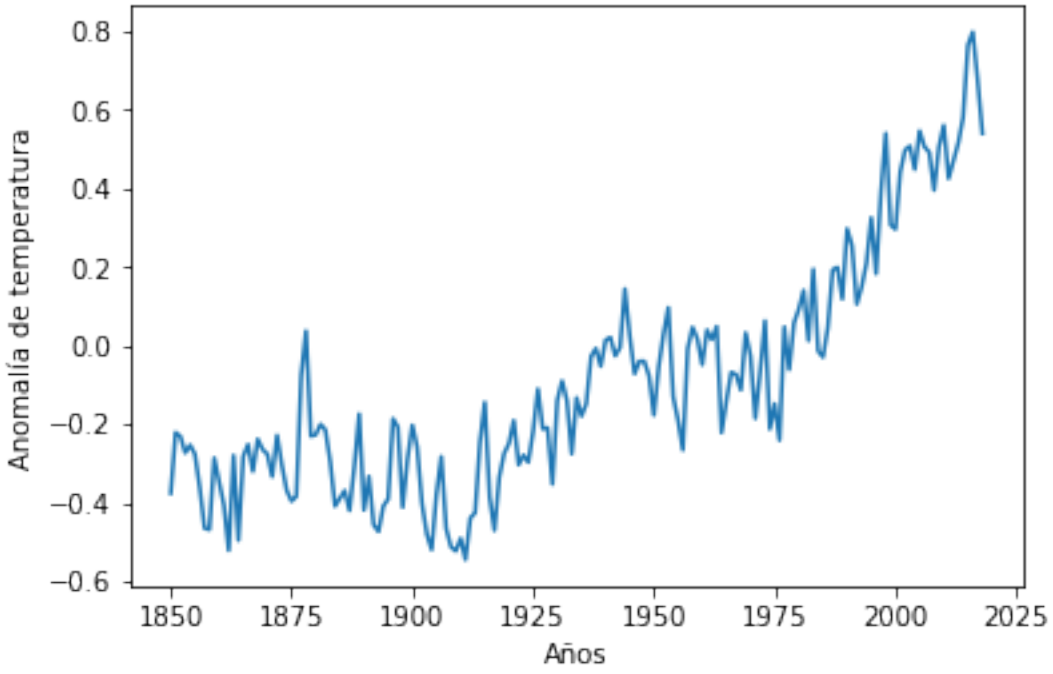
Hay una tendencia creciente, pero los datos todavía no están en el formato que necesitamos. Queremos eliminar las variaciones estacionales ya que los datos son mensuales. Así que queremos ver los promedios anuales, y necesitamos procesar los datos más.

```
In [5]: # Seleccionamos los años únicos con la función "unique":
        a = unique(datos[:,0])
```

```
In [6]: promedio = []
        for an in a:
            seleccion = (datos[:,0] == an)
            promedio.append( mean(datos[seleccion,2]) )
        promedio = array(promedio)
```

```
In [7]: plot(a,promedio)
        xlabel("Años")
        ylabel("Anomalía de temperatura")
```

```
Out[7]: Text(0,0.5,'Anomalía de temperatura')
```



1.3.5 2. Aplicar análisis

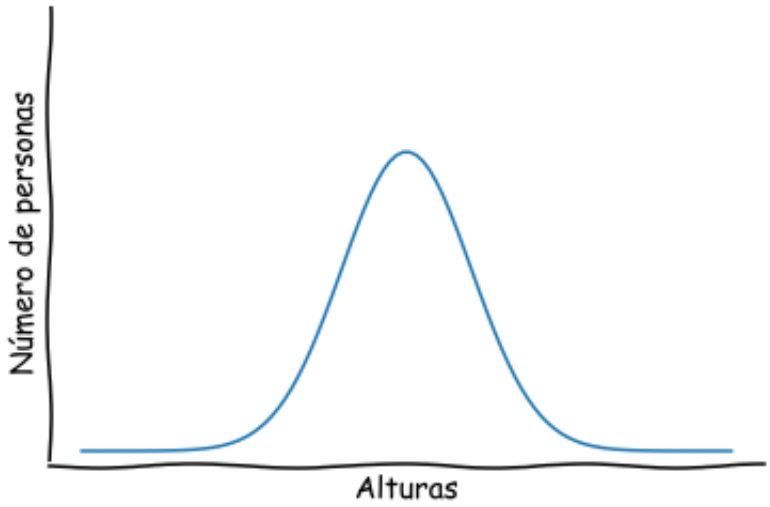
Análisis de datos muchas veces significa un análisis *estadística*. Ejemplos simples son el valor promedio de un parámetro y algo que se llama la *desviación estandar* de un parámetro.

Es muy importante tener conocimiento de la estadística para trabajar con datos! Hay un curso en el Semestre VI sobre la estadística.

La desviación estandar: es una medición de cuánto varía una variable. Si vemos a una parte del gráfico de temperatura donde los valores son más o menos "estables", vemos que hay desviaciones pequeñas de los valores.

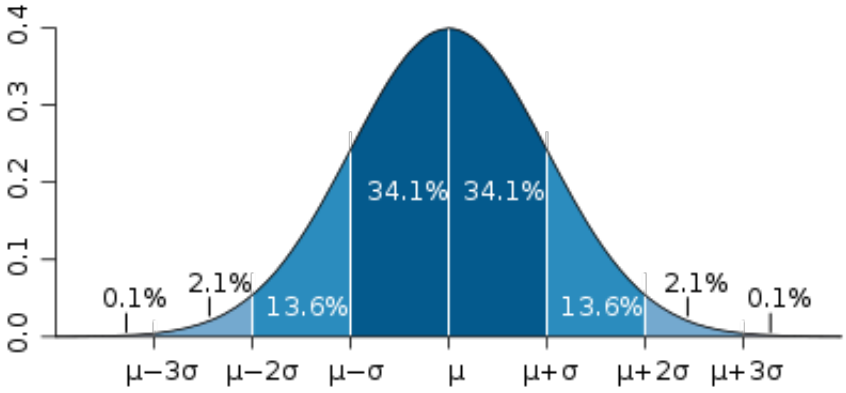
La desviación estandar es una cuantificación del tamaño de las desviaciones del valor promedio.

La variación de un valor está asociada a la "distribución" del valor. Para muchas variables en la naturaleza la distribución es "gausiana" o "normal". Por ejemplo, la distribución de las alturas en una población de personas es típicamente una gausiana:



Este gráfico dice que la mayoría de la población tiene una altura promedia (obviamente), y muy pocas personas son extremadamente alto o extremadamente corto.

Este tipo de función también se llama una *función de densidad de probabilidad* (PDF, por sus siglas en inglés). Esta función da la probabilidad que una variable toma valores dentro de un cierto rango. La PDF gausiana tiene dos parámetros: el promedio de la distribución, y la *desviación estandar*.



La desviación estandar determina el "ancho" de la distribución. Específicamente, incluye aproximadamente 68% del área bajo la curva. Este significa que 68% de los valores medidos deberían estar dentro de 1 desviación estandar del valor promedio.

Para las temperaturas, en la época de temperaturas estables, podemos aproximar que las anomalías están distribuidas según la distribución gausiana.

Es un hecho matemático que la combinación de errores en mediciones, variabilidad natural,

etc. tenderán a causar que las variables tengan una distribución gausiana.

Una desviación mayor que 1 desviación estandar puede occurir en 32% de los casos (para una distribución gausiana). Una anomalía mayor que 2 desviaciones estandares puede occurrir en sólo 5% de los casos. Algo mayor que 3 desviaciones estandares tiene una probabilidad de solamente 0.4%.

Análisis estadística de las temperaturas Calculamos el valor de la desviación estandar para la época $1961 - 1990$.

```
In [8]: # Convertimos el arreglo de años de "float" a "int"
        a = array(a,int)

In [9]: # Creamos un arreglo para elegir solamente las anomalías anuales para el
        # período 1961-1990
        seleccion = ((a <= 1990) & (a >= 1961))
```

La función **std** de NumPy calcula la desviación estandar de un conjunto de datos.

```
In [10]: std(promedio[seleccion])

Out[10]: 0.13215956816792848
```

Entonces, una anomalía anual de valor absoluto mayor que ~ 0.132 sería una anomalía mayor que 1 desviación estandar.

Ahora calculamos los valores absolutos de las anomalías a partir del año 2000.

```
In [11]: seleccion = (a >= 2000)
         abs(promedio[seleccion])/0.132

Out[11]: array([2.24179293, 3.34722222, 3.76893939, 3.84532828, 3.39393939,
                4.13699495, 3.83143939, 3.72474747, 2.99305556, 3.8270202 ,
                4.24558081, 3.21843434, 3.56691919, 3.90088384, 4.39457071,
                5.78977273, 6.04356061, 5.12815657, 4.08712121])
```

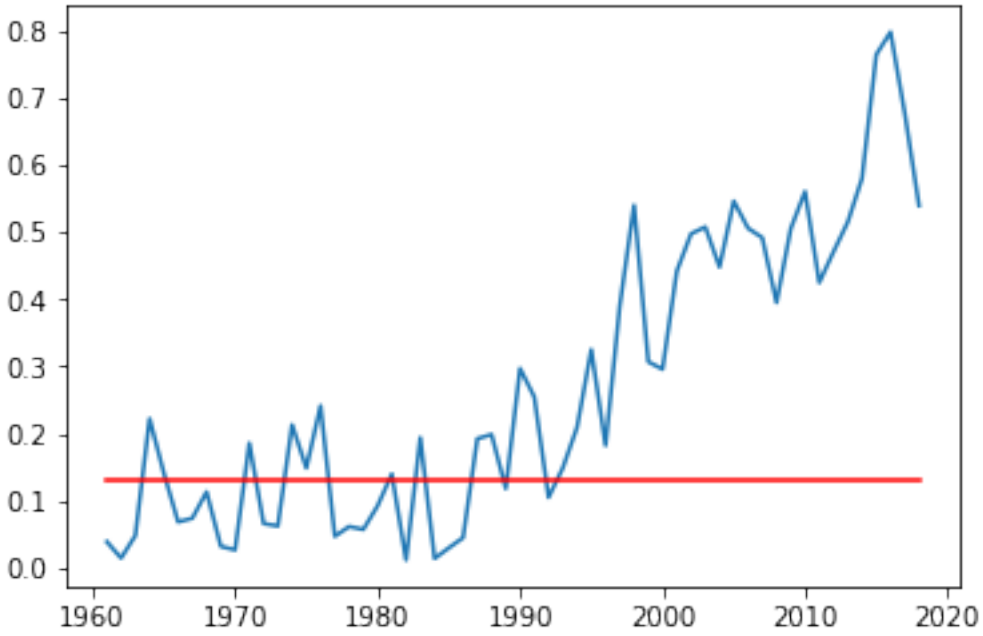
Las anomalías a partir del año 2000 son mayor que 2 desviaciones estandares, y algunas son mayor que 5!

Estas anomalías son tan improbable que no pueden ser variabilidad natural, aunque tenemos que tener en mente las suposiciones de nuestro análisis (por ejemplo, que el período $1961 - 1990$ es "estable").

1.3.6 3. Visualizar los resultados

Graficamos el valor absoluto de las anomalías a partir del año 1961, con una línea que indica 1 desviación estandar del valor promedio de las anomalías en el período $1961 - 1990$.

```
In [12]: seleccion = (a >= 1961)
         plot(a[seleccion],abs(promedio[seleccion]))
         plot([1961,2018],[0.132,0.132],'r-')

Out[12]: [<matplotlib.lines.Line2D at 0x7f7a45849710>]
```



Con este gráfico podemos ver muy claramente que las anomalías de temperatura en los últimos 20 años son mucho mayor que 1 desviación estandar.

La visualización del resultado es una parte muy importante del análisis de datos. Ayuda con la comunicación del resultado del análisis.

Así, otros investigadores pueden entender rápidamente nuestras conclusiones y sobre que base llegamos a ellas, para que puedan verificar nuestros resultados.

1.4 Herramientas para análisis de datos

El trabajo de análisis de datos es lo que los astrónomos, los físicos experimentales, los meteorólogos, hacen *todo el tiempo*.

Hoy en día, tenemos muchas herramientas computacionales para este tipo de análisis.

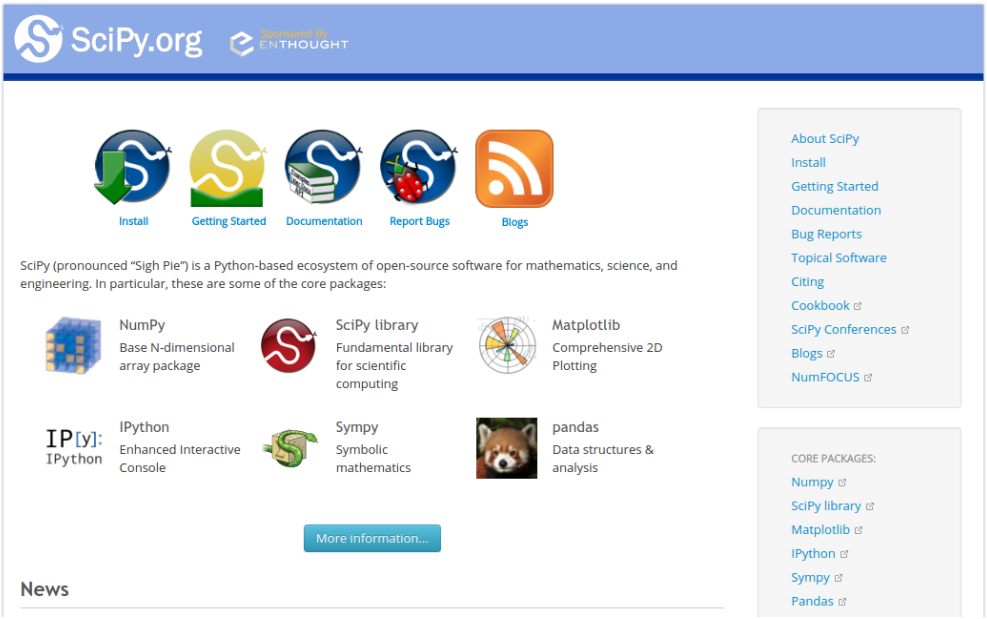
En Python hay varios módulos útiles para análisis de datos:

- NumPy - Numerical Python (Python numérico)
- SciPy - Scientific Python (Python científico)
- matplotlib - Gráficos en Python
- pandas - Módulo para análisis de datos
- scikit-learn - Módulo para *machine learning* en Python
- Theano - Otro módulo que se usa en *machine learning*

SciPy https://www.scipy.org/

Sympy

```
In [13]: from sympy import *
         init_printing()
```




```
In [14]: x, y, z = symbols('x y z')
```

```
In [15]: diff(cos(x),x)
```

Out[15]:

$$-\sin(x)$$

```
In [16]: expr = x/(1+x**2)
         expr
```

Out[16]:

$$\frac{x}{x^2+1}$$

```
In [17]: integrate(expr,x)
```

Out[17]:

$$\frac{1}{2}\log\left(x^2+1\right)$$

```
In [40]: init_printing(pretty_print=False)
```

pandas

```
In [41]: import pandas as pd
```

```
In [42]: !head class6_images/hubble_data.csv
```

```
distance,recession_velocity
.032,170
.034,290
.214,-130
.263,-70
.275,-185
.275,-220
.45,200
.5,290
.5,270
```

```
In [43]: datos = pd.read_csv("class6_images/hubble_data.csv")
         datos.head()
```

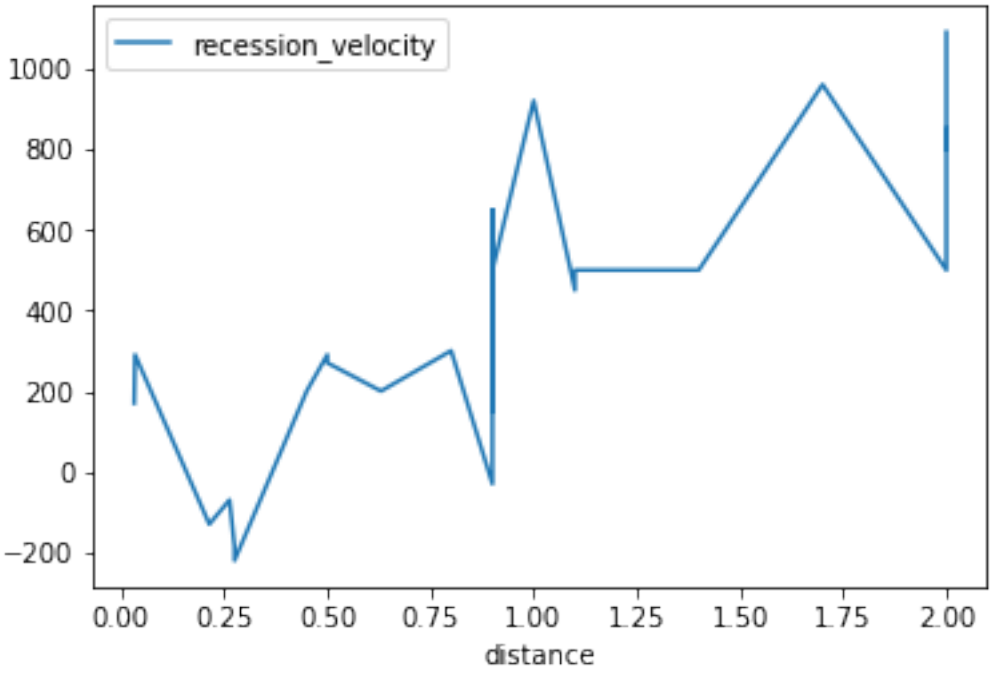
```
Out[43]:    distance  recession_velocity
         0     0.032                 170
         1     0.034                 290
         2     0.214                -130
         3     0.263                 -70
         4     0.275                -185
```

```
In [44]: datos.set_index("distance", inplace= True)
         datos.head()
```

```
Out[44]:           recession_velocity
         distance
         0.032                   170
         0.034                   290
         0.214                  -130
         0.263                   -70
         0.275                  -185
```

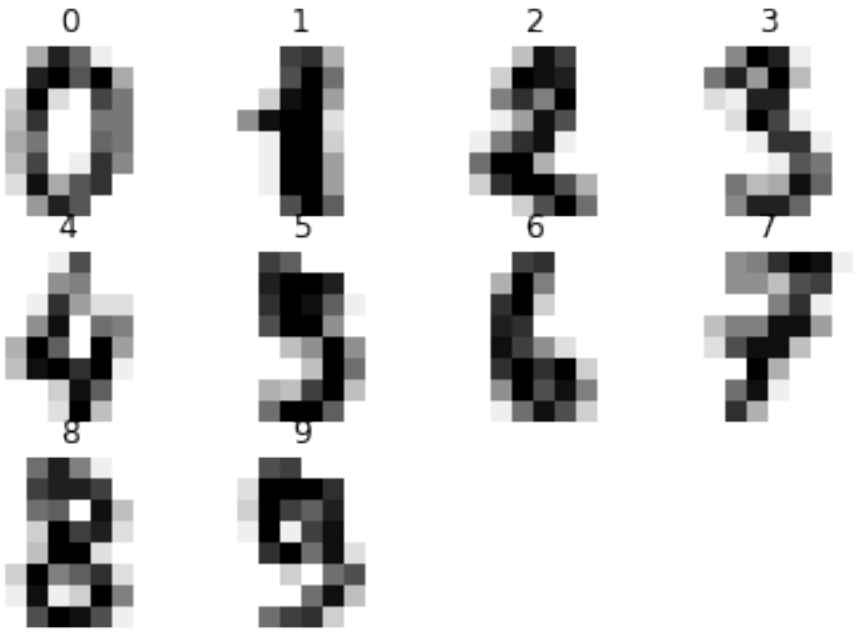
```
In [45]: datos.plot()
```

```
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7a0c634898>
```

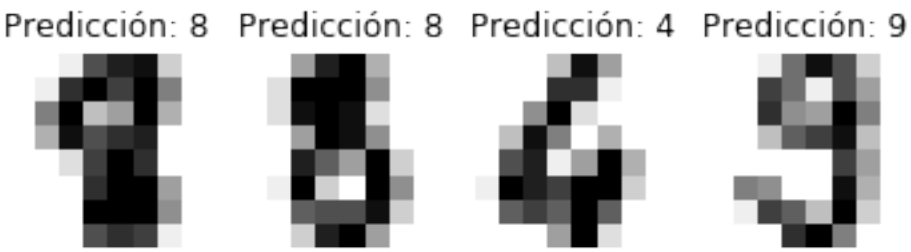


Scikit-learn

```
In [46]: from sklearn import datasets, svm, metrics

In [47]: digitos = datasets.load_digits()

In [48]: imagenes_y_etiquetas = list(zip(digitos.images, digitos.target))
         for index, (image, label) in enumerate(imagenes_y_etiquetas[:10]):
             subplot(3, 4, index + 1)
             axis('off')
             imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
             title('%i' % label)
```




```
In [49]: n_muestras = len(digitos.images)
         datos = digitos.images.reshape((n_muestras, -1))

In [50]: # Usamos un algoritmo de machine learning que se llama
         # "support vector machine"
         clasificador = svm.SVC(gamma=0.001)

In [51]: # Usamos la mitad de los digitos para "entrenar" el algoritmo
         clasificador.fit(datos[:n_muestras // 2], digitos.target[:n_muestras // 2])

Out[51]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

In [52]: # Predecimos los valores de la otra mitad de los digitos
         esperado = digitos.target[n_muestras // 2:]
         predicho = clasificador.predict(datos[n_muestras // 2:])

In [53]: imagenes_y_predicciones = list(zip(digitos.images[n_muestras // 2:], predicho))
         for index, (image, prediction) in enumerate(imagenes_y_predicciones[:4]):
             subplot(2, 4, index + 5)
             axis('off')
             imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
             title('Predicción: %i' % prediction)
```



Theano Diseñado para optimizar y evaluar expresiones matemáticas con arreglos. También se puede usarlo con GPUs para acelerar el algoritmo.

1.5 Datos masivos

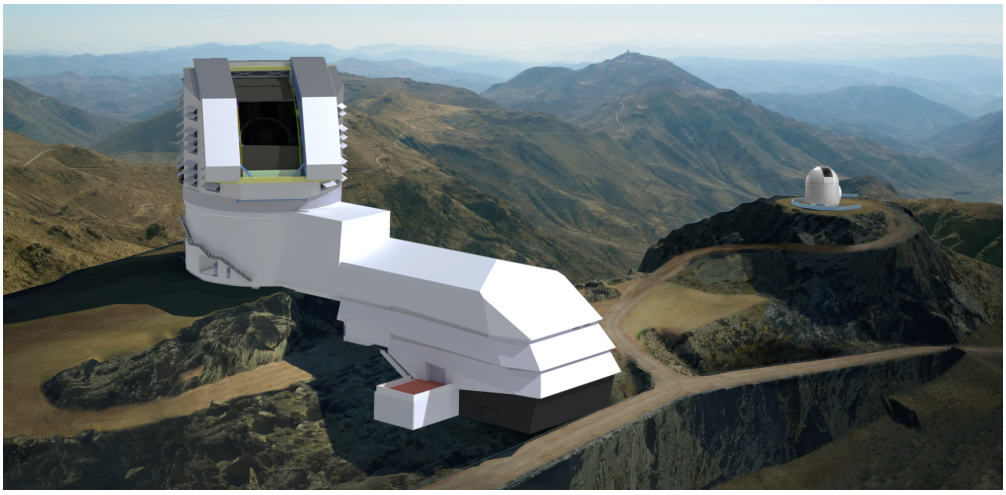
Vimos antes que hoy en día tenemos muchos datos disponibles, pero demasiado para hacer un análisis "tradicional", como lo que hicimos arriba.

Necesitamos nuevas técnicas de extracción, almacenamiento y análisis de datos. El desarrollo y aplicación de estas técnicas es una ciencia nueva que se llama *ciencia de datos*.

1.5.1 Bases de datos

Uno de los desafíos grandes en ciencia de datos es cómo almacenar y procesar los datos.

Un ejemplo es el nuevo telescopio el LSST.



- $\sim$ 15 TB de datos por noche.
- No es factible tener todos estos datos en formato *raw* en un servidor.
- Por ejemplo, si quiero usar solamente los datos de estrellas variables, quiero buscar y descargar solamente esos datos, y no todos.

Por lo tanto, es muy importante construir una *base de datos* que guarda los datos en una forma estructurada y accesible.

Hay equipos que trabajan en la colaboración del LSST que están diseñando esta base de datos, y la infraestructura necesaria para facilitarla (servidores, conexiones, etc.)

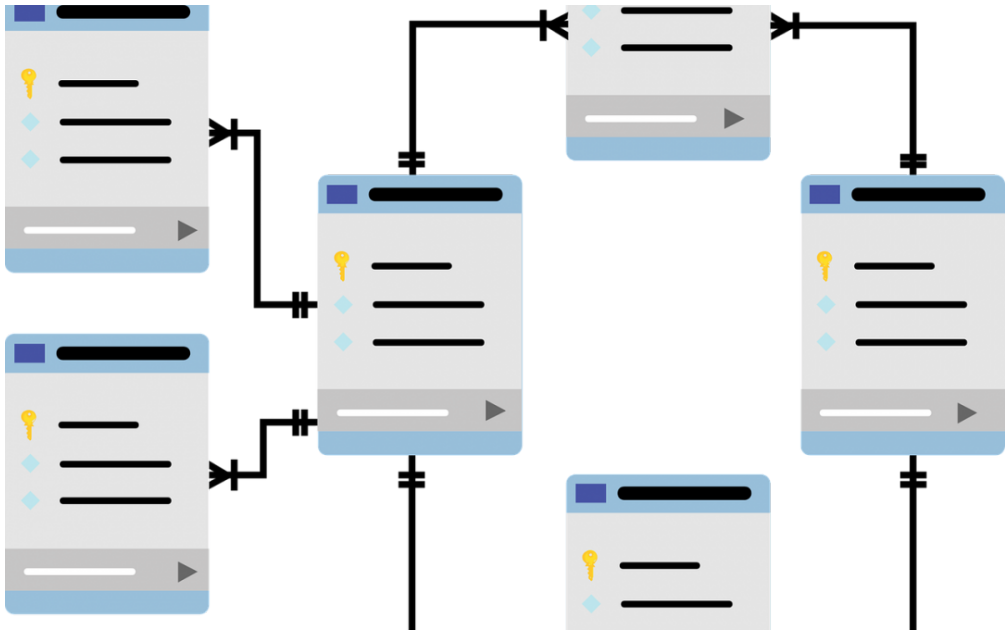
Hay otros ejemplos del uso de bases de datos:

- Simulaciones cosmológicas modernas generan muchos datos que necesitan organización.
- Experimentos en la física de partículas (por ejemplo, el LHC) también generan muchos datos.

Por lo tanto, para los científicos modernos es muy importante tener conocimiento del diseño y operación de las bases de datos.

1.5.2 Los tipos de bases de datos

- Relacionales: los datos están organizados en tablas, que tiene relaciones entre ellas. Este tipo es el más común y tradicional.



- No-relacional: bases de datos distribuidas, almacen de documentos, bases de datos gráficas, pares de valores-claves

Bases de datos relacionales Se puede "interactuar" con muchas bases de datos utilizando un lenguaje que se llama *SQL* (Structured Query Language). **No** es un lenguaje de programación, sino un lenguaje especializado para el uso de bases de datos.

- Los datos en una base de datos relacional están organizados en tablas (que a veces se llaman "relaciones") con columnas y filas.

- Las columnas corresponden a las atributas de los datos, mientra las filas son los registros.

Ejemplo: un catálogo de estrellas (filas), con coordenadas, tipo espectral, luminosidad, etc. (columnas):

Tabla: estrellas

Rank	Star Name	Bayer Name	Visual Mag.	Abs. Mag.	Dist. (ly)	Spectral Type	Lum. (Sol)	Mass (Sol)	Diam. (Sol)	RA (hr_min)	Dec (deg)
1	Sirius	Alp CMa	-1.44	1.45	8.6	A1V	25	2.0	1.7	06h 45m	-16.7°
2	Canopus	Alp Car	-0.62	-5.53	310	F0Ib	13600	8.5	65	06h 24m	-52.7°
3	Rigil Kent.	Alp Cen	-0.28c	4.34	4.4	G2V+K1V	1.5	1.1	1.2	14h 40m	-60.8°
4	Arcturus	Alp Boo	-0.05v	-0.31	36.7	K2III	170	1.1	26	14h 16m	+19.2°
5	Vega	Alp Lyr	0.03v	0.58	25.3	A0V	37	2.1	2.3	18h 37m	+38.8°
6	Capella	Alp Aur	0.08v	-0.48	42.2	G5III+G0II	79	2.7	12	05h 17m	+46.0°
7	Rigel	Bet Ori	0.18v	-6.69	770	B8Ia	66000	17	78	05h 15m	-8.2°
8	Procyon	Alp CMi	0.40	2.68	11.4	F5IV-V	7.7	1.5	2.0	07h 39m	+5.2°
9	Betelgeuse	Alp Ori	0.45v	-5.14	430	M2Ib	105000	18	936	05h 55m	+7.4°
10	Achernar	Alp Eri	0.45v	-2.77	144	B3V	3300	6-8	10	01h 38m	-57.2°
11	Hadar	Bet Cen	0.61v	-5.42	525	B1III	16000	10.7	8	14h 04m	-60.4°
12	Altair	Alp Aql	0.76v	2.20	16.8	A7V	10.6	1.8	1.8	19h 51m	+8.9°
13	Acrux	Alp Cru	0.77c	-4.19	320	B0.5IV+B1V	25000	14	?	12h 27m	-63.1°
14	Aldebaran	Alp Tau	0.87	-0.63	65.1	K5III	425	1.7	44.2	04h 36m	+16.5°
15	Spica	Alp Vir	0.98v	-3.55	260	B1V+B2V	13400	11	7.8	13h 25m	-11.2°
16	Antares	Alp Sco	1.06v	-5.28	605	M1Ib+B4V	65000	15.5	800	16h 29m	-26.4°
17	Pollux	Bet Gem	1.16	1.09	33.7	K0III	32	1.9	8	07h 45m	+28.0°
18	Fomalhaut	Alp PsA	1.17	1.74	25.1	A3V	17.7	2.1	1.8	22h 58m	-29.6°
19	Deneb	Alp Cyg	1.25v	-8.73	3200	A2Ia	54000	20	110	20h 41m	+45.3°
20	Mimosa	Bet Cru	1.25v	-3.92	350	B0.5III	34000	14	8	12h 48m	-59.7°

El punto importante es que cada fila tiene un número de identificación, o una clave.

Entonces, quizás tenemos otra tabla sobre estrellas variables, con atributas (columnas) acerca de la variación en magnitud, tipo de estrella variable, etc. Cada estrella en esa tabla también tendrá un número de identificación:

Tabla: estrellas_variables

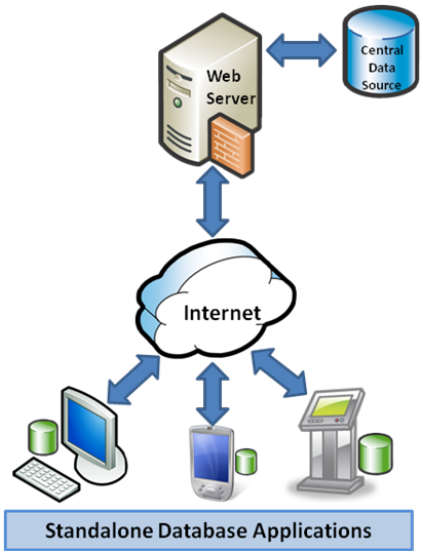
ID	Nombre	Mag. max.	Mag. min.	Tipo
9	Betelgeuse	0.0	1.3	Supergigante rojo
60	Algol	2.1	3.4	Binaria eclipsante
61	Eta Carinae	-0.8	7.9	Variable LBV

Así que podemos relacionar las dos tablas por coincidencia de los números de identificación.

En esta manera podemos combinar la información de muchas tablas para encontrar lo que necesitamos.

Acceso remoto En el caso de los datos masivos del LSST, por ejemplo, las tablas que contienen la información de las observaciones van a estar en un servidor (o varios). Un astrónomo que quiere usar los datos puede enviar una solicitud al servidor (escrito en *SQL*) y el servidor combinará las tablas según lo que está pidiendo. Finalmente enviará la tabla resultante al usuario:

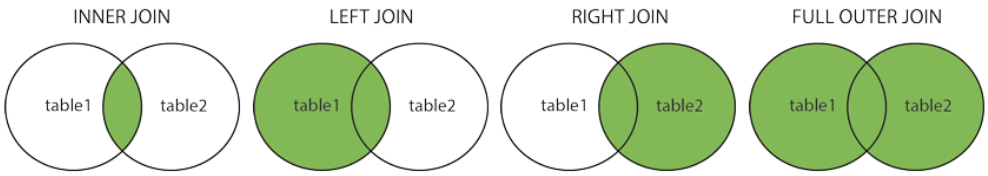
SQL Ahora veremos algunos ejemplos del uso de SQL para acceder información en una base de datos relacional. Primero, veremos algunos de los comandos básicos:



- SELECT (seleccionar): para elegir datos de una tabla
- WHERE (donde): para aplicar un filtro
- JOIN (unir): para combinar las filas de varias tablas

Hay distintos tipos de JOIN:

- INNER JOIN (unir internamente): solamente las filas que coinciden entre las tablas
- LEFT JOIN (unir a la izquierda): todas las filas de la tabla a la izquierda, y solamente las que coinciden en la tabla a la derecha
- RIGHT JOIN (unir a la derecha): opuesto de LEFT JOIN
- OUTER JOIN (unir externamente): usar todas las filas de las dos tablas si hay coincidencia



Un ejemplo con las tablas de estrellas:

```
SELECT estrellas.nombre, estrellas.mass, estrellas_variables.tipo FROM estrellas
INNER JOIN estrellas_variables ON estrellas.Rank=estrellas_variables.ID
```

Estos comandos crearán otra tabla que contiene solamente los nombre, tipos y masas de las estrellas que coinciden en su valor de "rank" de la tabla *estrellas* y de su valor de "ID" de la tabla *estrellas_variables*:

Nombre	Mass	Tipo
Betelgeuse	18	Supergigante rojo

Nombre	Mass	Tipo
Algol	3.2 (primaria)	Binaria eclipsante
Eta Carinae	100-200	Variable LBV

Si la tabla arriba se llama *estrellas_var_masas*, podemos aplicar un filtro con WHERE:

```
SELECT estrellas_var_masas.nombre, estrellas_var_masas.tipo FROM
estrellas_var_masas WHERE estrellas_var_masas > 10
```

Con este comando tenemos la siguiente tabla con sólo una fila:

Nombre	Tipo
Betelgeuse	Supergigante rojo

Bases de datos no-relacionales

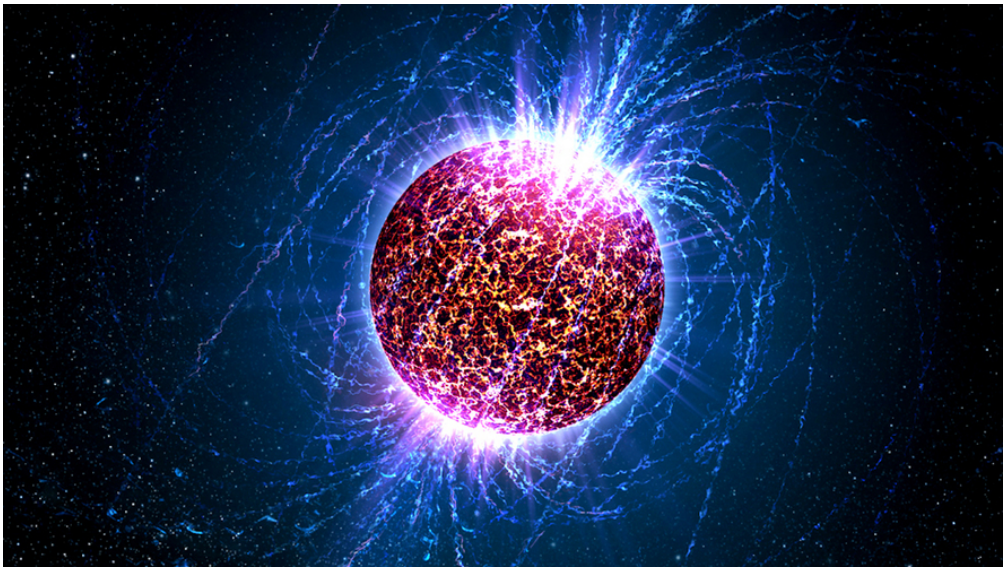
- Bases de datos sin mucha estructura
- No hay una forma "estandar" de estos tipos de bases de datos
- Para aplicaciones en el Internet, este tipo es más común ya que la información es muy dinámica, y cambia rápidamente
- También se usa este tipo si la información está distribuida en la red

Bases de datos en la astronomía Se puede acceder a muchos bases de datos astronómicas directamente de Python por el uso del módulo **astroquery** que es parte de **astropy**, un módulo de funciones útiles para la astronomía.

http://www.astropy.org/
https://astroquery.readthedocs.io/en/latest/

1.6 Análisis de datos masivos

Obtener los datos es el primer paso. El segundo es hacer el análisis! Vimos el ejemplo antes de temperaturas promedias. Ahora veremos un ejemplo que muestra las dificultades en trabajar con datos **masivos**.

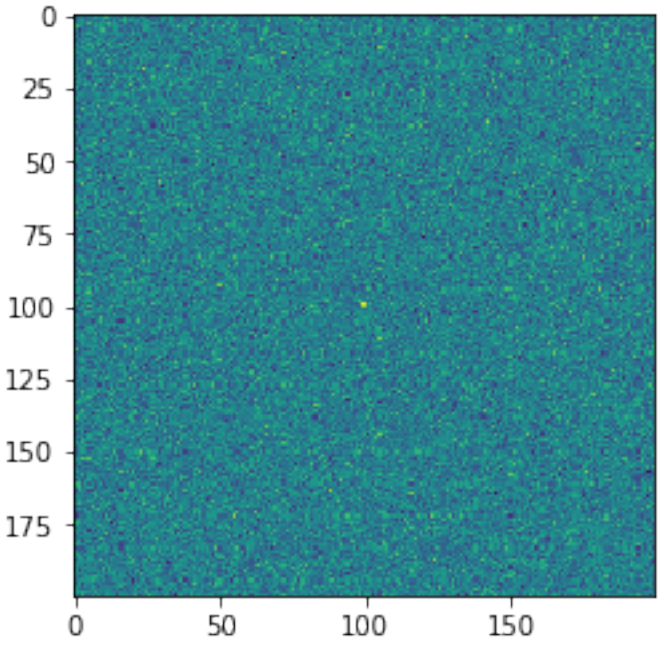


Ejemplo: datos (imágenes) de pulsares Un púlsar es una estrella de neutrones con una rotación muy rápida que emite en la longitud de onda de radio.

Suponemos que tenemos varias imagenes de pulsares.

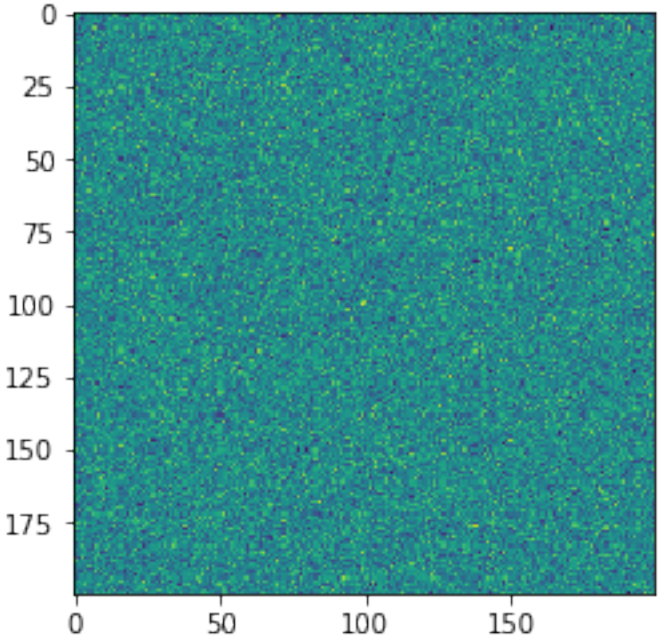
- Podemos cargar la imágen a un arreglo, como vimos antes.
- Así que, tendremos los arreglos `imagen1`, `imagen2`, etc.

```
In [54]: %pylab inline

Populating the interactive namespace from numpy and matplotlib


In [55]: imagen1 = load("class6_images/pulsars/image1.npy")
         imagen6 = load("class6_images/pulsars/image6.npy")

In [56]: imshow(imagen1)

Out[56]: <matplotlib.image.AxesImage at 0x7f7a44c76470>
```




```
In [57]: imshow(imagen6)
```

```
Out[57]: <matplotlib.image.AxesImage at 0x7f7a0cf59748>
```



```
In [58]: type(imagen1)

Out[58]: numpy.ndarray
```

Los datos de cada imágen están en arreglos de NumPy. Si las imágenes tienen una resolución de 200×200, vamos a tener arreglos de $200 \times 200 = 40000$ elementos.

```
In [59]: imagen1.shape

Out[59]: (200, 200)
```

Tenemos 10 imagenes, así que cargamos todas las imagenes en un arreglos de 3 dimensiones: $200 \times 200 \times 10 = 400000$ elementos.

```
In [60]: imagenes = zeros((200,200,10))
         for i in range(1,10+1):
             imagenes[:,:,i-1] = load("class6_images/pulsars/image"+str(i)+".npy")

In [61]: imagenes.shape

Out[61]: (200, 200, 10)
```

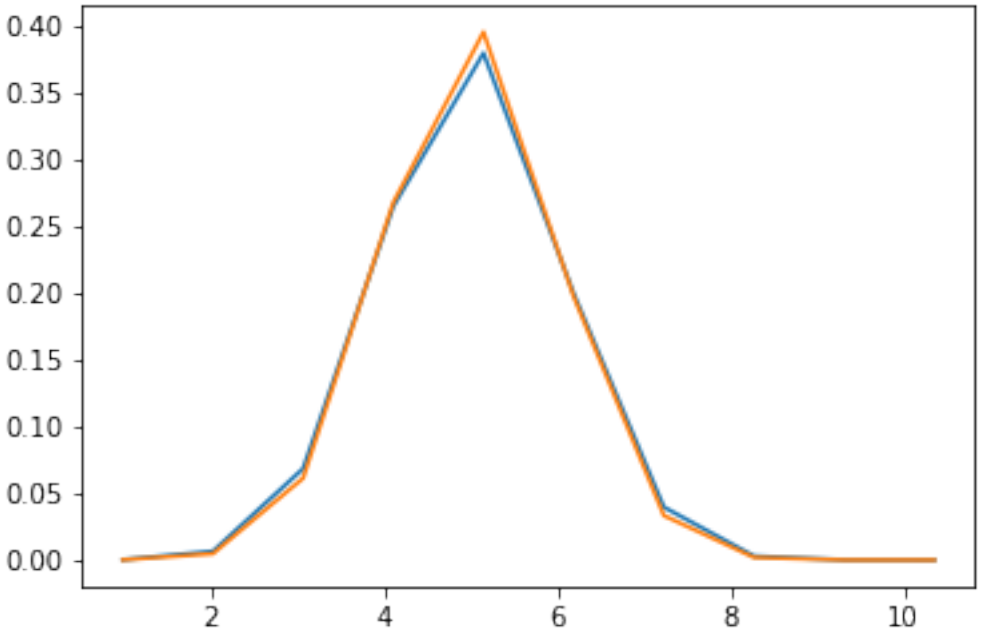
Combinando las imagenes No se va nada en las imagenes! Solo "ruido"!

En cada pixel hay un "error" (ruido térmico en el detector, etc.). Hay un señal, pero está muy cerca a los valores de ruido en las imagenes, así que no vemos nada.

Suponemos que los errores tienen una distribución gausiana, que es muy probable.

De hecho, podemos investigar si es así o no. Usamos la función **histogram** para calcular la frecuencia de ocurrencia de valores:

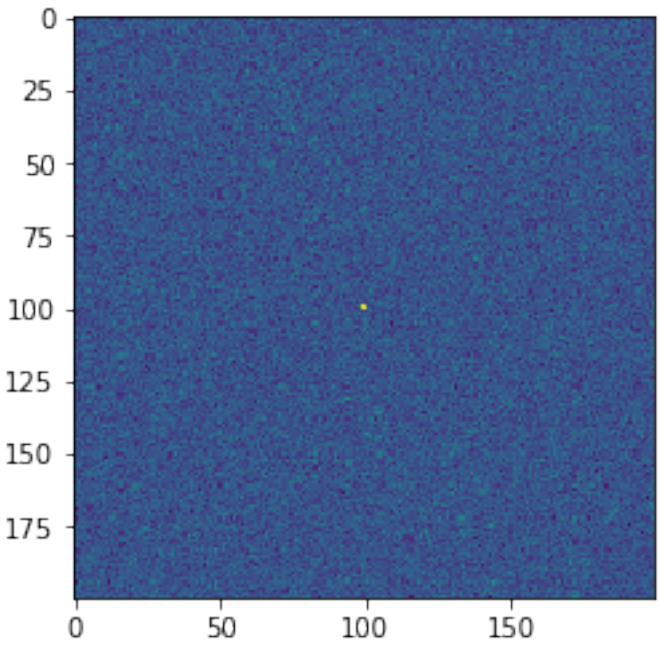
```
In [62]: from scipy import stats
         h, bins = histogram(imagenes.flatten(), normed=True)
         centros = 0.5*(bins[1:] + bins[:-1])

         pdf = stats.norm.pdf(centros,loc=5.0)
         plot(centros, h)
         plot(centros, pdf)

Out[62]: [<matplotlib.lines.Line2D at 0x7f7a0cf966d8>]
```



Del gráfico arriba, parece que los valores en los pixeles de las imagenes están distribuidas según la distribución gausiana, con valor promedio de 5. Este valor corresponde al valor de fondo del instrumento (lo que detecta el telescopio de un cielo oscuro).

¿Por qué es útil saber que los "errores" tienen una distribución gausiana? Porque ahora podemos sumar toda las imagenes, y los errores se van a *cancelar* (estadísticamente)! De hecho, calculamos el valor *promedio* de cada pixel.

```
In [63]: imagen_resultante = mean(imagenes,axis=2)

In [64]: imshow(imagen_resultante)

Out[64]: <matplotlib.image.AxesImage at 0x7f7a0d0c2e10>
```



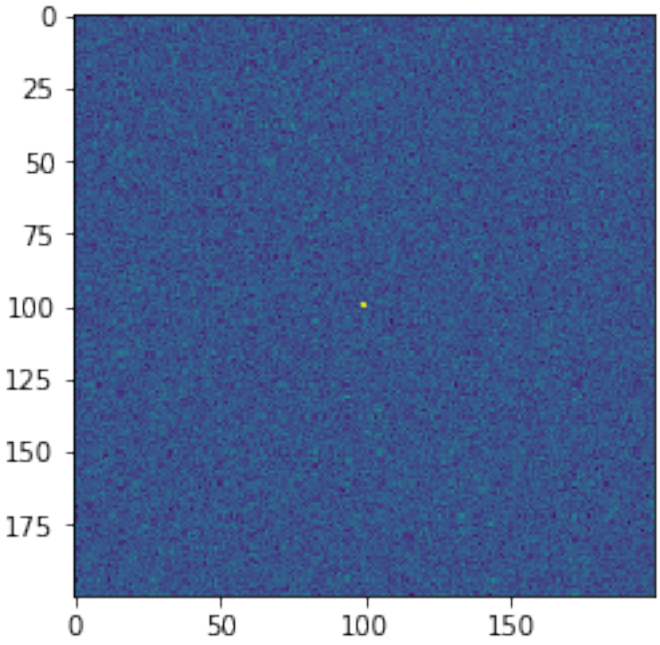
Ahora podemos ver un puntito en el medio que es la detección (estadística) de un púlsar!

Otro análisis: usando el mediano Esta operación de combinar imagenes es algo muy común en la astronomía. Se llama *stacking* (apilado).

Para calcular el valor promedio de los pixeles, no era necesario cargar *todas* las imagenes.

Otra manera de hacerlo es la siguiente:

```
In [65]: imagen_resultante = zeros((200,200))
         for i in range(1,10+1):
             imagen_resultante += load("class6_images/pulsars/image"+str(i)+".npy")
         imagen_resultante /= 10

In [66]: imshow(imagen_resultante)

Out[66]: <matplotlib.image.AxesImage at 0x7f7a0d0804a8>
```



Llegamos al mismo resultado, pero estamos guardando solamente 2 arreglos en la memoría a la vez: el arreglo `imagen_resultante` y el arreglo del imágen actual en cada iteración del ciclo.

Un problema con el promedio de un conjunto de variables es que es muy sensible a valores extremos (los llamados *outliers*) que vimos antes.

Por lo tanto, muchas veces es mejor usar la *mediana*. La mediana es el valor que separa la distribución de valores en dos partes iguales.

```
In [67]: mean(imagenes), median(imagenes)

Out[67]: (5.001326936528571, 5.00041743962621)
```

Por su definición, para calcular la mediana necesitamos tener *todos* los valores en la memoria a la vez, para organizar los datos en sus dos partes.

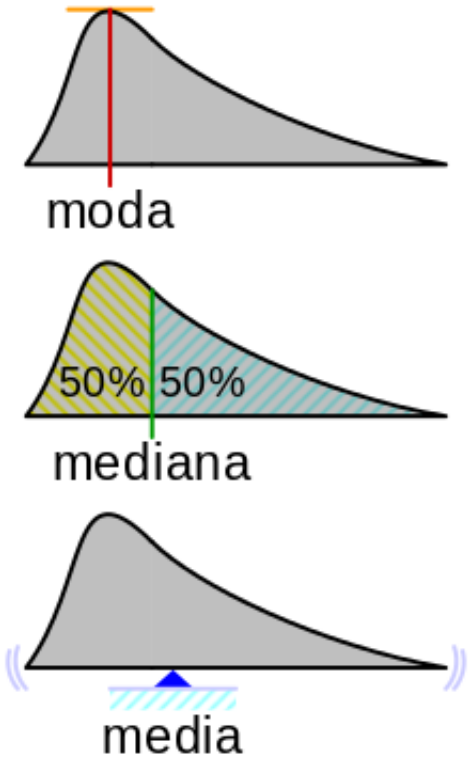
Este puede ser problemático si tenemos muchos datos.

```
In [68]: type(imagenes)

Out[68]: numpy.ndarray

In [69]: imagenes.dtype

Out[69]: dtype('float64')
```

Todos los valores del arreglo de imagenes son *floats* de 64 bits. Así que la memoria necesaria para cargar todos los datos es

$64 \times 400000 = 2.56 \times 10^7$ bits $\sim$ 3.2 MB.

Pero si tenemos 1000 imagenes, con resolución 1000×1000, ocupará


moda
50%
50%
mediana
media

$64 \times 1000 \times 1000 \times 1000 \approx 8$ GB!

Es poco práctico cargar 8 GB de datos en un computador para calcuar la mediana.

Una opción es usar un algorítmo que calcula una aproximación a la mediana que no necesita tener todos los arreglos en la memoria. Otra opción es... paralelizar el algorítmo!

Podríamos dividir las imágenes en sub-imágenes, calcular la mediana de cada sub-imágen, y determinar su promedio para la imágen total. Esta opción también es una aproximación al resultado.

- En el caso de analizar datos masivos, necesitamos nuevas formas de trabajar. El análisis "hecho a mano" que vimos al principio de la clase no sirve...

2 Resumen

- Análisis de datos es algo que los científicos hacen todo el tiempo.
- Las herramientas computacionales que tenemos hoy en día ayudan muchisimo en ese área.
- El lenguaje **Python** tiene muchos módulos útiles para trabajar con datos.
- Otro lenguaje útil para análisis estadística es **R**.
- Con datos masivos, necesitamos una forma de organizarlos, por ejemplo con *bases de datos*.
- También hay que procesar (limpiar) los datos de problemas, errores, etc.
- En el análisis de datos masivos es necesario el uso de métodos nuevos (automáticos).
- En la mención hay 2 cursos en el área de **ciencia de datos**. Las habilidades que uno aprende en este área sirven para muchos trabajos.