¿Cómo estimar una distribución?

En la clase pasada se vio un ejercicio numérico con Python para entender como una distribución Gaussiana o normal es el patrón natural de distribución de probabilidades de un conjunto de datos real.

Esto nos llevo hasta cierto punto a ajustar una función de probabilidad a un conjunto de datos. Ya se ha mencionado en clases anteriores que básicamente esa es la filosofía del Machine Learning, en resumen ajustar una distribución a un conjunto de datos para que con esa distribución podamos hacer predicciones. Así que es muy importante hacer ese procedimiento en general

Entonces en esta clase vamos a hablar más en detalle sobre como estimar como estimar una distribución de probabilidad.

In [ ]:
```
#Importando librerías
import numpy as np
from matplotlib import pyplot
#Este es un generador aleatorio de números pero basado
#en la distribución normal
from numpy.random import normal
#Distribución normal
from scipy.stats import norm
```

Lo que voy a hacer con el generador aleatorio, es de manera similar como lo hicimos con la distribución binomial. Es crear un conjunto de datos a partir de varios muestreos sobre esta distribución.

Entonces digamos que quiero extraer 10 000 datos generados aleatoriamente, recordemos que este generador está basado en la distribución normal.

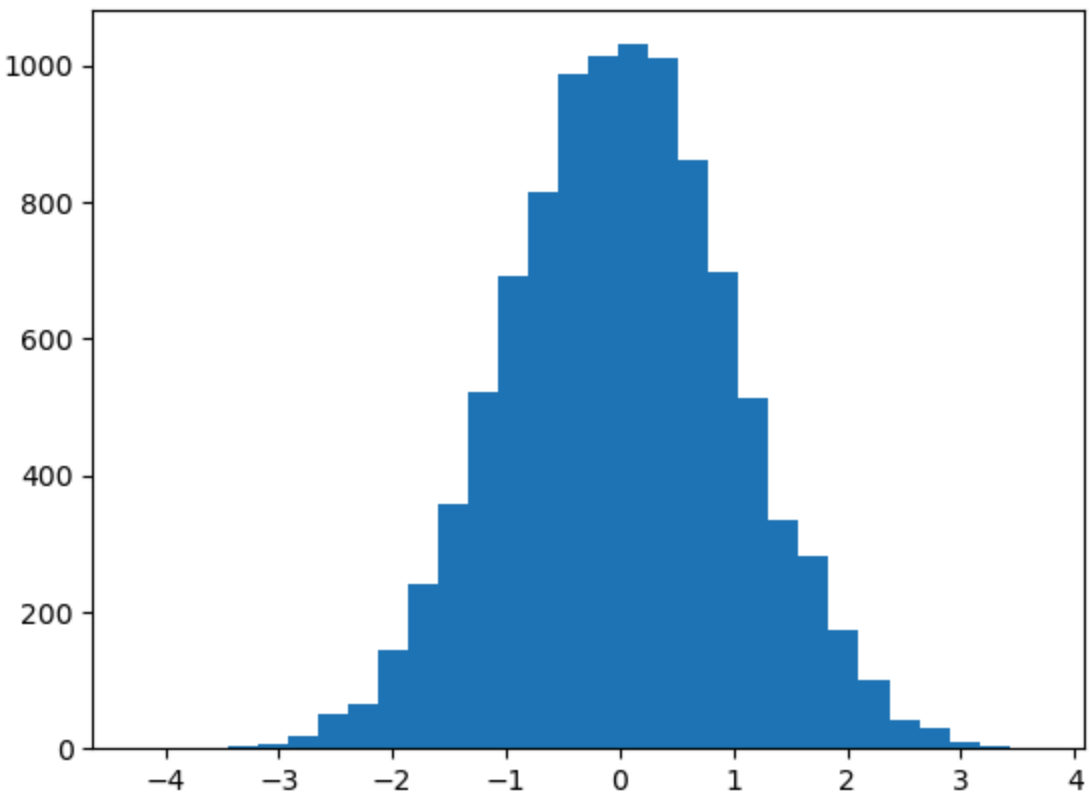
Lo que quiere decir que cuando yo grafique estos datos con un pyplot, en esta caso voy a usar un `hist()` sobre la muestra de datos que ya genere y que le de 30 intervalos para las barras del histograma.

Lo que se debería mostrar es algo que se asemeje a una distribución normal.

Verifiquemos la visualización

In [ ]:
```
#Generador de muestras
sample=normal(size=10000)

#Graficando
pyplot.hist(sample,bins=30)
pyplot.show()
```



Como se puede observar en la gráfica tenemos un conjunto de datos que es muy similar a una distribución normal, entonces ¿esto por qué lo estamos haciendo? Porque en este caso no estoy tomando datos reales como en el caso de la clase pasada, sino que estoy generando datos aleatorios con el uso del generador

```
from numpy.random import normal
```
y estos datos van a ser la base que usaremos en el ejercicio de **Estimación paramétrica** y **Estimación no paramétrica**.

¿Que quiere decir?
Ambos son tipos de estimación de densidades, es decir la estimación paramétrica; como fue el ejercicio anterior, esta basada en **forzar los parámetros de la distribución**.

¿Y la estimación no paramétrica? Ya la veremos.

Estimación paramétrica

Para el caso de esta estimación, voy a comenzar a generar datos como lo hice anteriormente y en esta ocasión voy a darle unos parámetros adicionales.

`normal(loc,scale,size)`

- loc: corresponde al promedio
- scale: sigma o desviación estándar
- size: numero de datos Un vez hecho esto, tengo mis datos simulados que reflejarían un experimento real.

Mientras que por otro lado lo que voy a hacer es ajustar una función con sus parámetros, una distribución Gaussiana, forzando de la misma manera que hicimos la vez pasada.

Entonces mi hipótesis de modelado es que mis datos deben seguir una distribución Gaussiana, donde el promedio es el **promedio de los datos** y la desviación es la **desviación de los datos**. Entonces lo hago de la misma forma que la vez pasada.

Luego de eso puedo crear un objeto llamado `distribucion` que esta basado en la función `norm()` de Scipy, que contiene los parámetros antes obtenidos (promedio y desviación estándar).

Hecho esto, ahora solo tengo que construir los valores numéricos sobre los cuales voy a evaluar mi función teórica de **Densidad de probabilidad**; esto lo hago creando valores con un ciclo for que recorra un rango de número de entre 30 y 70 de uno en uno.

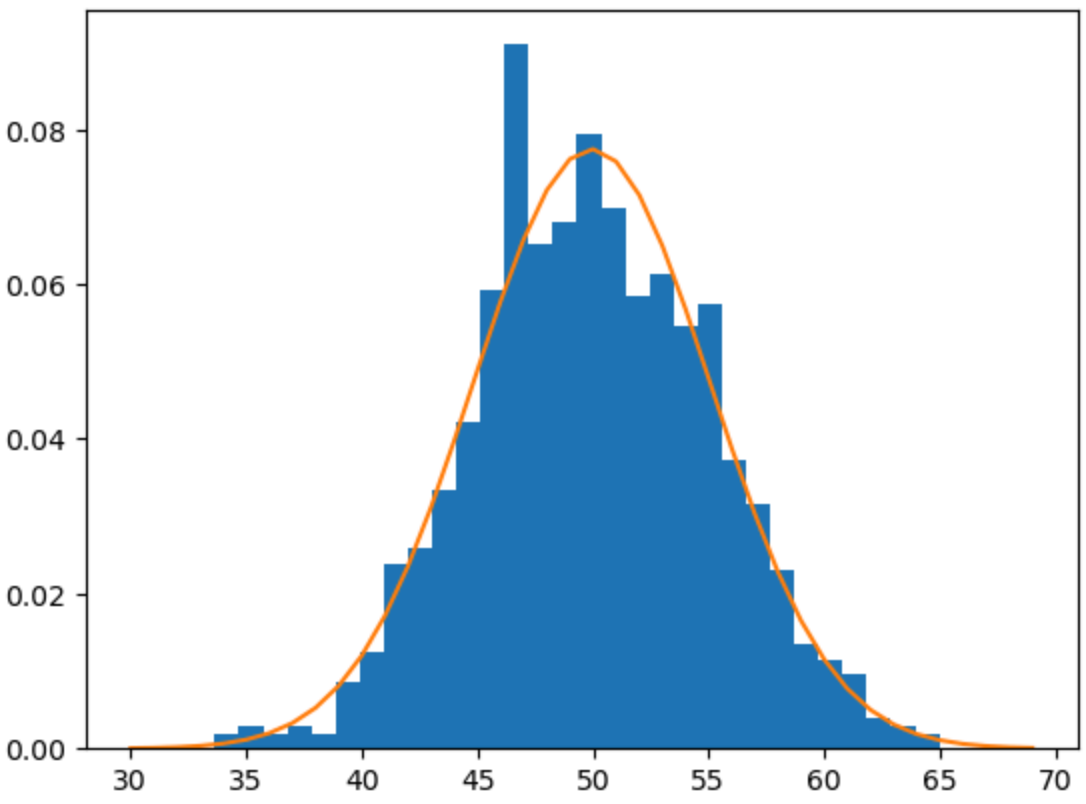
Finalmente voy a calcular las probabilidades de cada uno de esos valores, con otra lista que voy llenando de manera recursiva pero utilizando mi objeto `distribucion` y el método `pdf()` que es la **Densidad de Probabilidad** evaluada en cada uno de los valores de la lista `values`.

Al final llega la hora de graficar y observar como es la distribución de los datos. Para observar si mi hipótesis es correcta, porque yo debería ver una campana de Gauss que encierra perfectamente mis datos.

Esto lo hago, dibujando primero la muestra de datos experimentales `sample` en un histograma. Le digo que aplique `density=True` es decir normaliza los conteos, para que los conteos me den en un rango de 0 a 1 como probabilidades, y no que me den números enteros (frecuencias). Luego dibujare la función teórica con un plot, donde los valores en **X** son `values` y los valores en **Y** son las `probabilidades`. Si todo esta perfecto, cuando haga un pyplot entonces debería mostrar ambas cosas.

In [ ]:
```
#Obtener datos con el generador
sample = normal(loc=50,scale=5,size=1000)
#Calcular el promedio de los datos
promedio = sample.mean()
#Calcular la desviación estándar de los datos
desviacion = sample.std()
#Creando un objeto basado en la distribución normal
distribucion = norm(promedio,desviacion)
#Obtener valores numéricos sobre los cuales evaluar la función
#teórica de la Densidad de Probabilidad
values = [value for value in range(30,70)]
#Calculando las probabilidades de cada uno de esos datos
#de manera recursiva y usando el objeto distribucion
probabilidades = [distribucion.pdf(value) for value in values]

#Graficando los datos generados o simulados
pyplot.hist(sample,bins=30,density=True)
#Graficando la función de densidad de probabilidad
pyplot.plot(values,probabilidades)
pyplot.show()
```



Recuerda que el ajuste no es perfecto, porque cuando sea perfecto es cuando tengamos **infinitos datos**, y de manera computacional no podemos obtener eso.

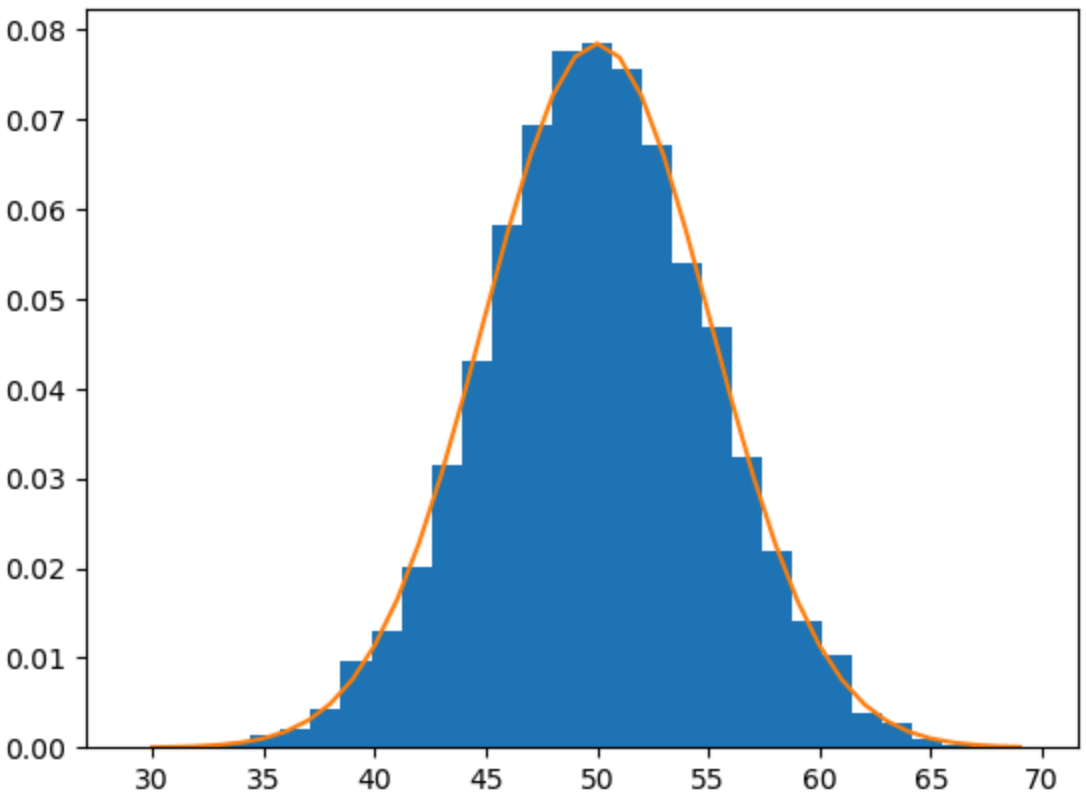
En la medida en que aumentemos los datos en el generador, el ajuste será mas preciso

```
#Obtener datos con el generador
sample = normal(loc=50,scale=5,size=1000)
```

In [ ]:
```
#Obtener datos con el generador
sample = normal(loc=50,scale=5,size=10000)
#Calcular el promedio de los datos
```

```
promedio = sample.mean()
#Calcular la desviación estándar de los datos
desviacion = sample.std()
#Creando un objeto basado en la distribución normal
distribucion = norm(promedio,desviacion)
#Obtener valores numéricos sobre los cuales evaluar la función
#teórica de la Densidad de Probabilidad
values = [value for value in range(30,70)]
#Calculando las probabilidades de cada uno de esos datos
#de manera recursiva y usando el objeto distribucion
probabilidades = [distribucion.pdf(value) for value in values]

#Graficando los datos generados o simulados
pyplot.hist(sample,bins=30,density=True)
#Graficando la función de densidad de probabilidad
pyplot.plot(values,probabilidades)
pyplot.show()
```



Aquí aumentamos la cantidad de datos a `10 000`, así que esto satisface nuestra hipótesis de la filosofía frecuentista.

Esto fue un ejemplo de estimación paramétrica.

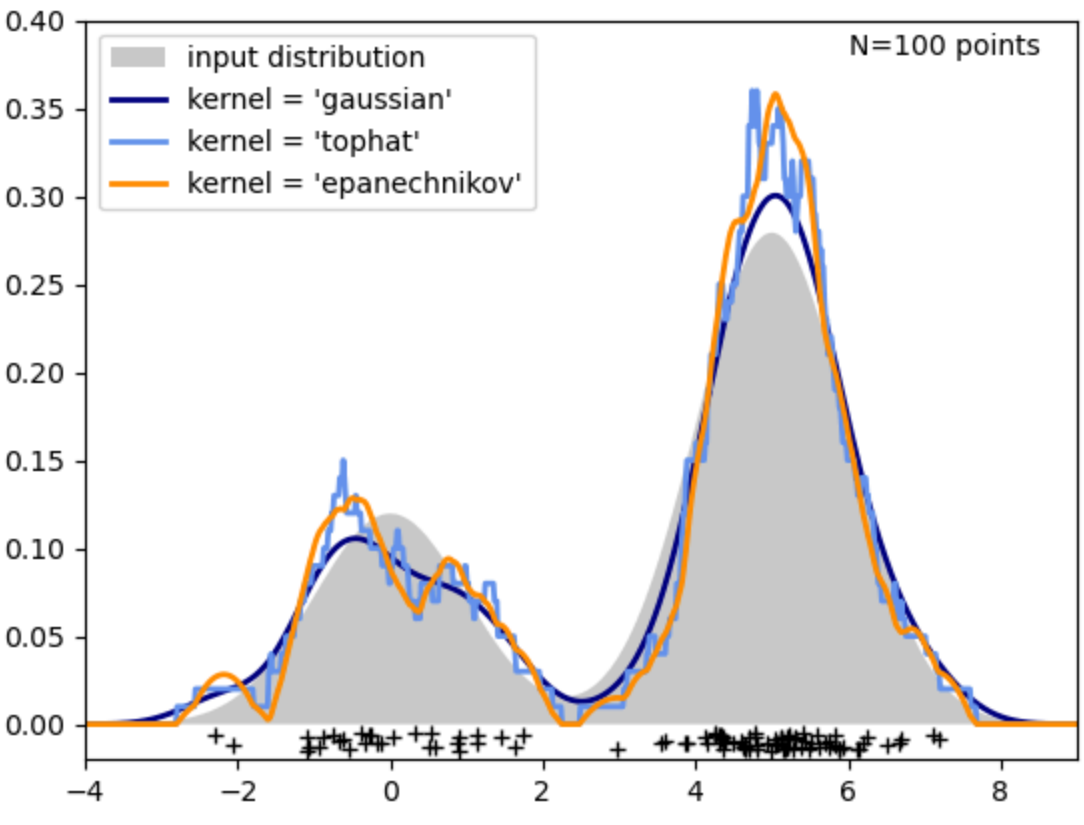
¿Cómo sería una estimación no paramétrica?

Estimación No Paramétrica

Cuando yo hago una estimación no paramétrica, quiere decir que no estoy forzando forzando parámetros dentro de una distribución única. Sino lo que voy a hacer es una combinación de varias distribuciones.

Rápidamente podemos hacer este ejercicio con una función de una librería de Scikit Learn y que es sobre el método de **Estimación de Densidad** Aquí el enlace Density Estimation.

La idea es cuando tu tienes distribuciones que no son ni Gaussianas o alguna otra conocida como sucede aquí:



Los datos anteriores describen una función que nosotros llamaríamos **Bimodal**, aquí tu no puedes ajustar ninguna **distribución Gaussiana**, porque las 2 montañas no se van a ajustar de forma perfecta nunca.

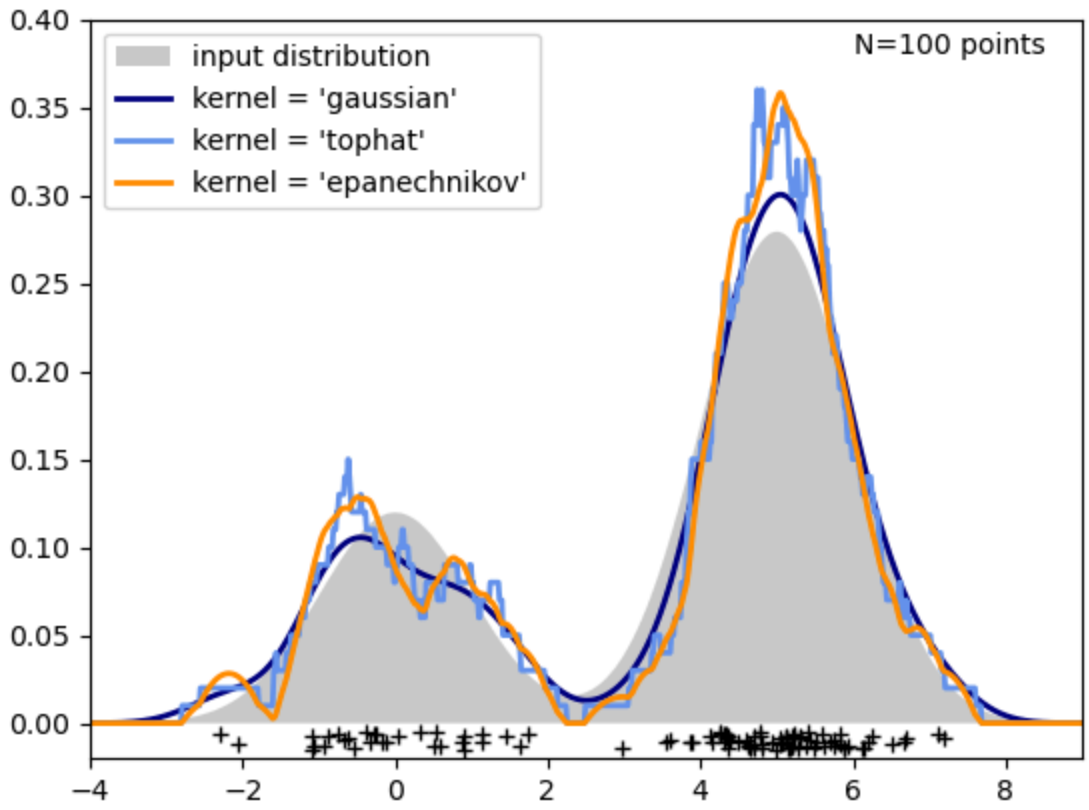
Entonces la idea consiste en combinar varias funciones y por eso nuestro método tiene esas 2 opciones.

- Primero un parámetro de suavizado llamados `smoothing parameter`
- Y otro que es la función base o `basis function`

¿Que quiere decir esto?

Recuerda la imagen de las montañas que pusimos anteriormente que están en la documentación de Scikit, esas 2 montañas no son los datos como tal sino que son un ajuste.

Si vemos en nuestro histograma los datos son saltos y no son continuos, entonces yo lo que tengo que crear es un parámetro de suavizado que contornee la envolvente de los datos (que se ajuste a los datos de forma suave). Y por otro lado la **función base** es una función que yo voy a usar para sumar varias de ellas. Analizando la forma de los datos



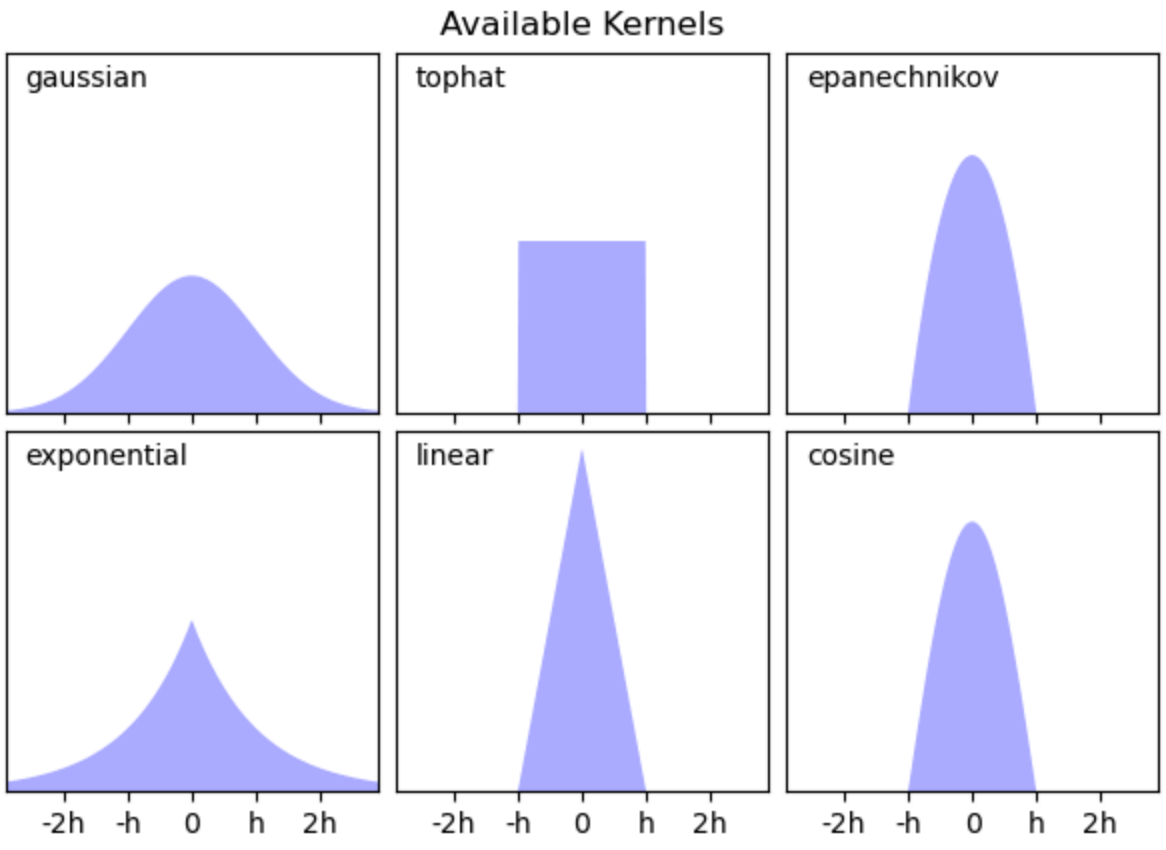
Por ejemplo puedo usar 2 Gaussianas, la de la derecha desplazada con una probabilidad mas grande y una desviación estándar mas pequeña en comparación con la de la izquierda.

Entonces en general los de Scikit usan sumas de funciones (kernels) para poder ajustar sus curvas.

> Here we have used kernel='gaussian', as seen above. Mathematically, a kernel is a positive function $K(x;h)$ which is controlled by the bandwidth parameter h . Given this kernel form, the density estimate at a point y within a group of points $x_i; i = 1...N$ is given by:

$$\rho\mathcal{K}(y) = \sum_{i=1}^{N} K(y - x_i; h)$$

Kernels



Dependiendo de la forma de los datos es como lo voy a usar.

¿Entonces como sería esto? Voy a construir una distribución bimodal de forma artificial.

In [ ]:
```
from numpy import hstack
from sklearn.neighbors import KernelDensity

#construimos una distribución bimodal
sample1 = normal(loc=20, scale=5, size=300)
sample2 = normal(loc=40, scale=5, size=700)
sample = hstack((sample1, sample2))
```

Explicación

Importo de numpy una function llamada `hstack()` y me ofrece juntar varios arreglos.

Y de la librería SkLearn voy a importar KernelDensity, que es el método anterior que se explicó. Los dos arreglos crean diferentes distribuciones Gaussianas con diferentes parametros; recuerda que el `loc=promedio=desplazamiento en el eje x` y el `scale=desviación estándar=ancho de la curva de la distribución`. Finalmente con `hstack` me junta los 2 arreglos dentro de uno solo.

Aplicando Kernel Density

Lo primero es crear el modelo que es `KernelDensity` y aquí yo le tengo que pasar unos argumentos:

- **bandwidth**: parámetro de suavizado
- **kernel**: aquí es el tipo de distribución que usaremos para ajustar la distribución.

Luego haremos un `reshape` para arreglar la estructura de datos de manera correcta para que después se pueda trabajar adecuadamente.

Luego al final vamos a ajustar los datos con el método `fit()`

In [ ]:
```
model = KernelDensity(bandwidth=2, kernel='gaussian')
sample = sample.reshape((len(sample), 1))
model.fit(sample)
```

Out[ ]:
```
KernelDensity(bandwidth=2)
```

Luego de manera similar a como ajustamos las probabilidades para los datos experimentales y comparar eso con los datos teóricos de una distribución.

Voy a generar una lista de valores que cada uno va de 1 a 60, por que los datos tienen promedios entre 20 y 40. Así que el rango de 1 a 60 va a permitir ver de manera efectiva el ajuste en un intervalo razonable.

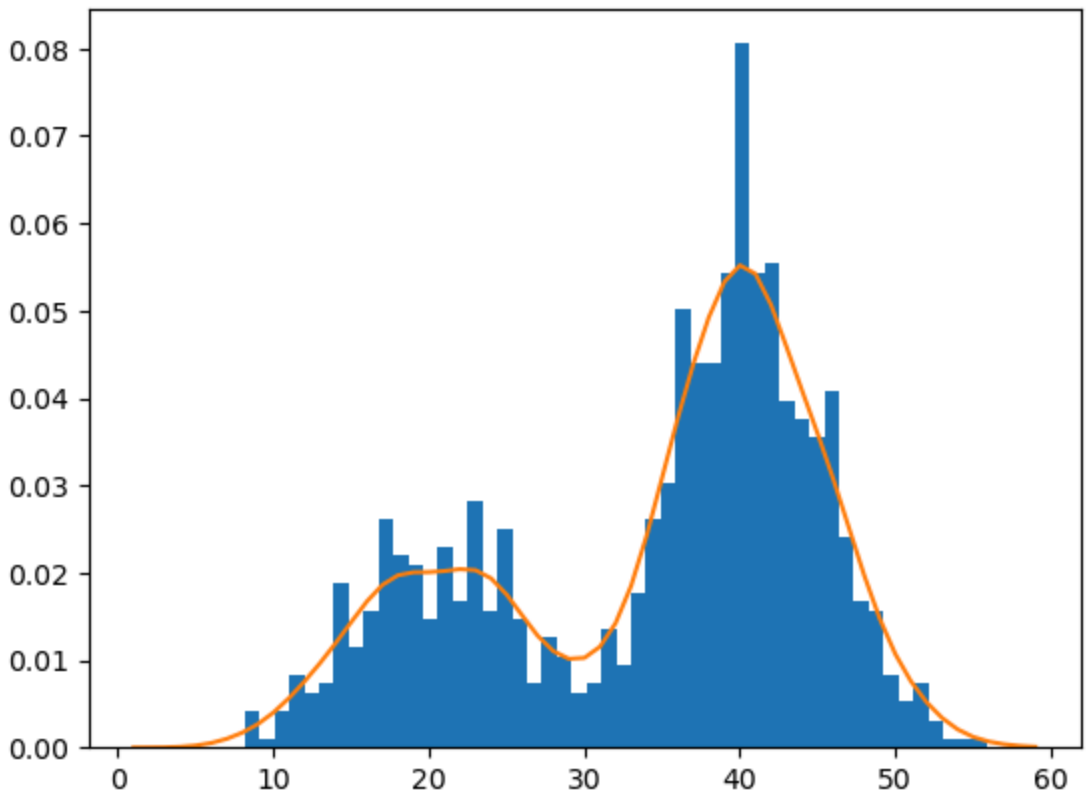
Luego voy a volver a hacer un **reshape**.

> OJO: TE VAS A DAR CUENTA QUE EN SCIKIT LEARN SIEMPRE HAY QUE HACER ESTE TIPO DE `reshape()` PARA PODER TRABAJAR CON LOS DATOS O USAR LOS MÉTODOS DE SCIKIT.

Luego calculo las probabilidades logarítmicas **¿por qué logarítmicas?** porque para la computadora es mas cómodo trabajar en logaritmos y que no exista el fenómeno de **underflow**, pero como necesito las probabilidades reales. Después de que hizo el calculo logarítmico, puedo regresar al espacio original de probabilidades, aplicando la exponencial que es la inversa. Esto es como una inversion a las probabilidades originales y después todo esta listo para hacer el histograma.

In [ ]:
```
#Obteniendo los valores del eje x
values = np.asarray([value for value in range(1,60)])
#Acondicionando los valores para usar en Scikit Learn
values = values.reshape((len(values),1))    #Lo va aguardar en forma de columna
#Obteniendo la probabilidad logarítmica
probabilidades = model.score_samples(values)
#Regresando a valores adecuados en la probabilidad
#Inversion de la probabilidad
probabilidades = np.exp(probabilidades)

#Graficando los datos simulados
pyplot.hist(sample,bins=50,density=True)
#Graficando la probabilidad con Estimación no paramétrica
pyplot.plot(values,probabilidades)
pyplot.show()
```



Como se puede observar los datos simulados están en azul y la curva naranja que obtenemos, es la distribución teórica que obtuve por medio de `DensityKernel` o el ajuste por `Densidad de Kernel`.

Entonces el resultado es un ajuste de datos que fueron generados artificialmente que describen una distribución **bimodal** y el ajuste por medio de 2 Gaussianas nos permiten tener ese grado de precisión en la comparación con los datos simulados.

Este es un método que si nos damos cuenta, no implica que yo tenga que forzar parámetros de la distribución, sino que depende de otros parámetros; como parámetro de suavizado y el tipo de función base que yo uso para el ajuste.

Sin embargo ambas técnicas; estimación paramétrica y estimación no paramétrica se usan para ajustar distribuciones de probabilidad.

¿En que casos uso uno u otra? Bueno eso depende del tipo de datos que tengamos, por ejemplo una bimodal nunca podrá ajustar con exclusivamente una Gaussiana, entonces ahí podemos usar una densidad estimada por medio de kernels podría funcionar mejor.

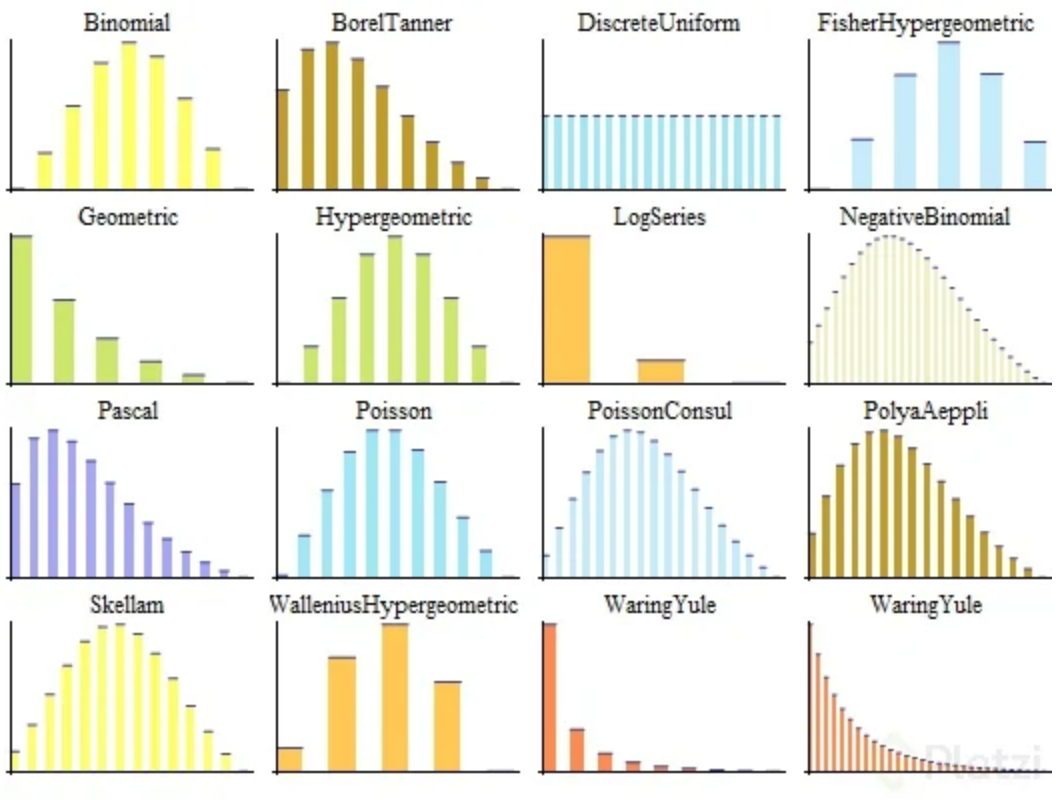
Resumen

Acabamos de ver con este ejercicio 2 maneras de ajustar Densidades de probabilidad teóricas a un conjunto de datos real. Este problema es fundamental, porque como ya hemos mencionado el Machine Learning en general y todo modelo que se haga sobre un conjunto de datos, consiste en que tu por trasfondo siempre estas intentando ajustar una densidad de probabilidad o una distribución de probabilidad a un conjunto de datos real y por eso este tema es muy importante.

En nuestra proxima clase vamos a ver un esquema de trabajo o un Framework que es muy popular a la hora de hacer este tipo de procesos, es decir a la hora de ajustar densidades de probabilidad a datos reales.

Extras:

- Density Estimation
- Numpy normal


Binomial
BorelTanner
DiscreteUniform
FisherHypergeometric
Geometric
Hypergeometric
LogSeries
NegativeBinomial
Pascal
Poisson
PoissonConsul
PolyaAeppli
Skellam
WalleniusHypergeometric
WaringYule
WaringYule