Gráficos de barras y circular

Los gráficos de barras y circular son utilizados frecuentemente para visualizar la distribución de valores de una variable categórica.

En el caso de una variable contínua también se utiliza una versión especial de gráfico de barras, dónde los valores de una variable son agrupandos en intervalos. Este tipo de gráfico se denomina histograma.

Configuración del entorno.

Lo primero es configurar el entorno de ejecución. Véase el cuaderno "Primeros pasos con Matplotlib" para más detalles.

In [1]:
```
%matplotlib notebook
import matplotlib as mpl
import matplotlib.pyplot as plt
print('Matplotlib version: {}'.format(mpl.__version__))
import numpy as np
np.set_printoptions(floatmode='fixed', precision=3)
import pandas as pd
from io import StringIO
```

```
Matplotlib version: 3.5.2
```

Gráfico de barras.

Los gráficos de barras se crean con la función `Axes.bar()`.

La función toma dos argumentos: el argumento `x` puede indicar las posiciones de las barras en el eje X o una lista de etiquetas para las barras. El parámetro `height` indica las alturas de las barras.

Ejercicio 01: Cargar la tabla de datos resultante de la Elección del Parlamento Europeo en 2004 y en 1999.

El resultado mostrado debería parecerse a lo siguiente:

Grupo	Asientos (2004)	Asientos (1999)
EUL	39	49
PES	200	210
EFA	42	56
EDD	15	19
ELDR	67	60

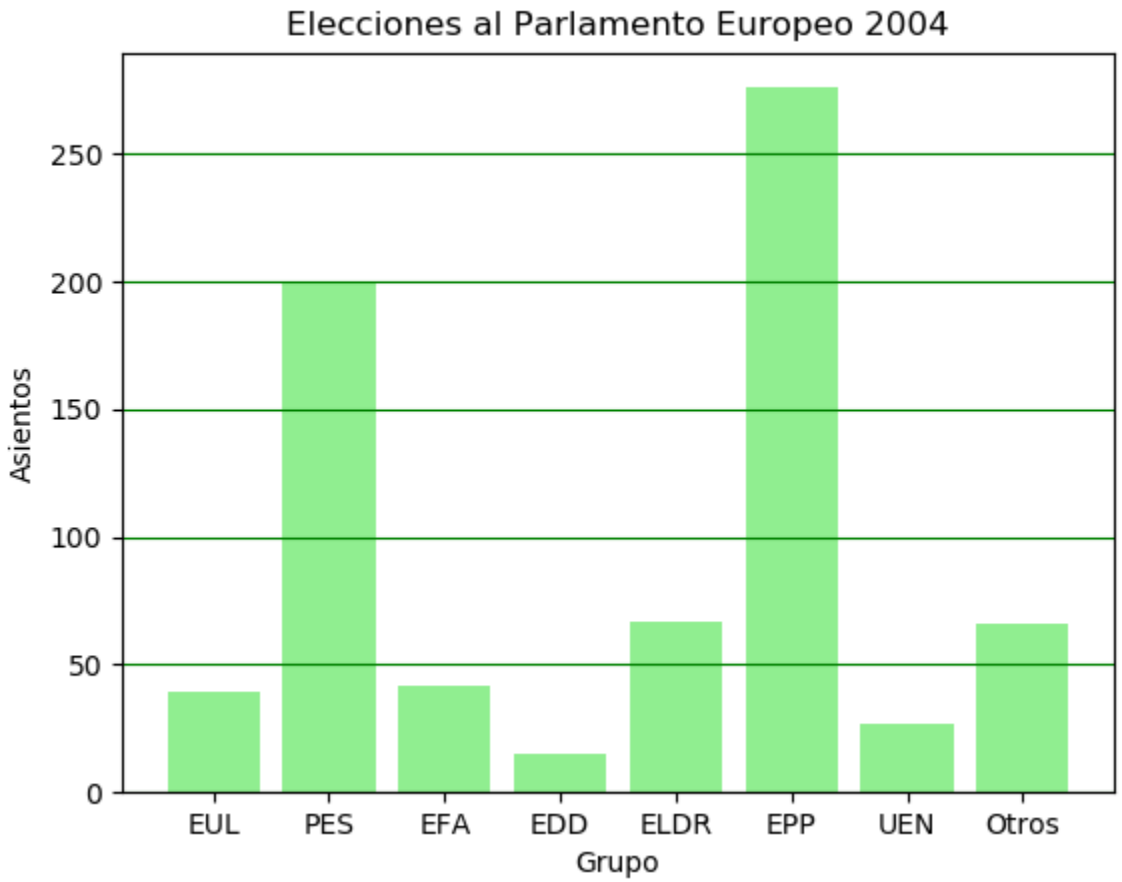
In [2]:
```
#Datos tomados de aqui: es.wikipedia.org/wiki/Diagrama_de_barras
file = StringIO(
'''Grupo        Asientos (2004) Asientos (1999)
EUL     39      49
```

```
PES     200     210
EFA     42      56
EDD     15      19
ELDR    67      60
EPP     276     272
UEN     27      36
Otros   66      29
''')

df = pd.DataFrame()
#Pon tu código aquí.
#Sugerencia: observa que que se trata de un CSV usando el caracter
#   tabulador '\t' como separador.

#

print(df.head())
```

```
Empty DataFrame
Columns: []
Index: []
```

Ejercicio 02: Dibujar un diagrama de barras con la distribución de asientos de la elección del año 2004.

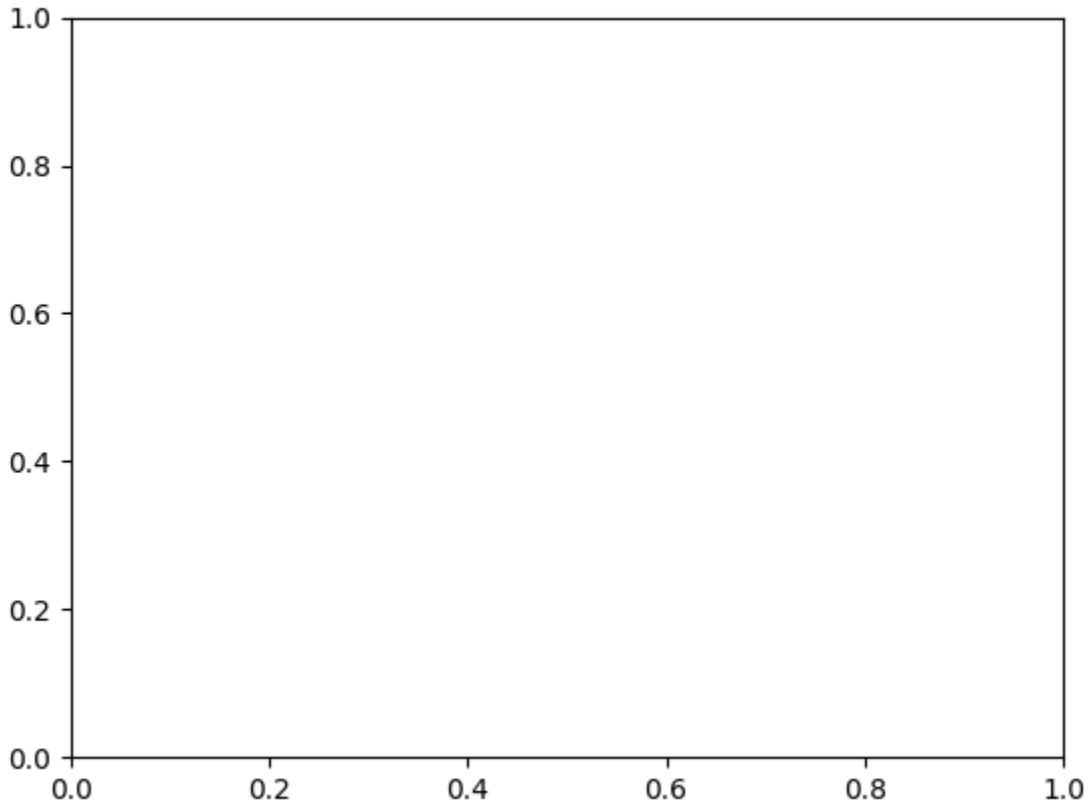
Intenta ajustar los parámetros para que el resultado sea lo más parecido posible a la siguiente figura.



In [3]:

```
fig=plt.figure()
ax = plt.axes()
#Pon tu código aquí.
#Sugerencia: establece el título set_title() y las etiquetas
#de los ejes set_xlabel(), set_ylabel().

#

#Pon tu código aquí.
```

```
#Sugerencia: activar la rejilla con el grid()
#   Usa los parámetros 'axis' y 'color'.

#

#Pon tu código aquí.
#Sugerencia: dibuja el diagrama de barras con bar().
#Recuerda: Queremos la serie con los datos del año 2004.
#   Usa el atributo index como valores X y el método
#   to_numpy() para obtener alturas de las barras.

#
```

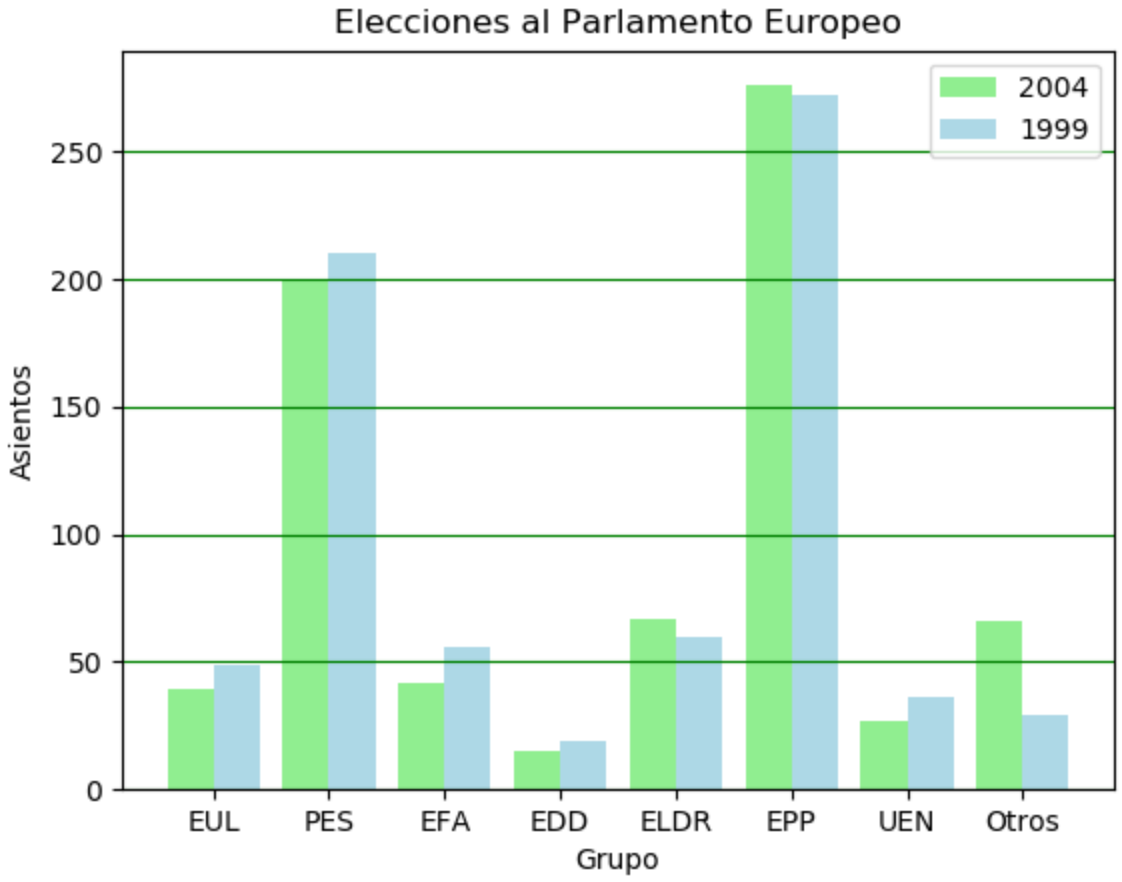


Si tenemos más de una serie para el mismo conjunto de etiquetas (por ejemplo las elecciones en dos años distintos) es muy común mostrar grupos de barras agrupadas.

Para ello usaremos valores x en el rango entero `[0, <num-etiquetas>-1]` y desplazamos para cada serie las barras un offset según el ancho de las barras. Para disponer las etiquetas de las barras usaremos las funciones `Axes.set_xticks()` y `Axes.set_xticklabels()`.

Ejercicio 03: Dibujar un diagrama de barras con la distribución de asientos de las elecciones del año 2004 y 1999.

Intenta ajustar los parámetros para que el resultado sea los más parecido posible a la siguiente figura.



In [4]:
```
fig=plt.figure()
ax = plt.axes()
#Pon tu código aquí.
#Sugerencia: establece el título set_title() y las etiquetas
#de los ejes set_xlabel(), set_ylabel().

#

#Pon tu código aquí.
#Sugerencia: activar la rejilla con el grid()
#usa los parámetros axis y color.

#

x = []
#Pon tu código aquí.
#Sugerencia: genera un rango [0, num-etiquetas)

#

#Pon tu código aquí.
#Sugerencia: usa Axes.set_xticks() para indicar que queremos
#ticks en cada valor del rango generado.

#
#Pon tu código aquí.
#Sugerencia: usa Axes.set_xticklabels() para indicar una etiqueta
# (el índice del dataframe).

#

#Pon tu código aquí.
```

```
#Sugerencia: dibuja el diagrama de barras con bar() para la serie del año 2004.
#Usa un ancho de barra de 0.4, y desplaza con -0.2 los valores x.
#Utiliza alienación centerada. Asigna etiqueta '2004' al gráfico para la leyenda.

#

#Pon tu código aquí.
#Sugerencia: dibuja el diagrama de barras con bar() para la serie del año 1999.
#Usa un ancho de 0.4, y desplaza con +0.2 los valores x.
#Utiliza alienación centerada. Asigna etiqueta '1999' al gráfico para la leyenda.

#

#Pon tu código aquí.
#Activa la caja de leyendas con Axes.legend().

#
```

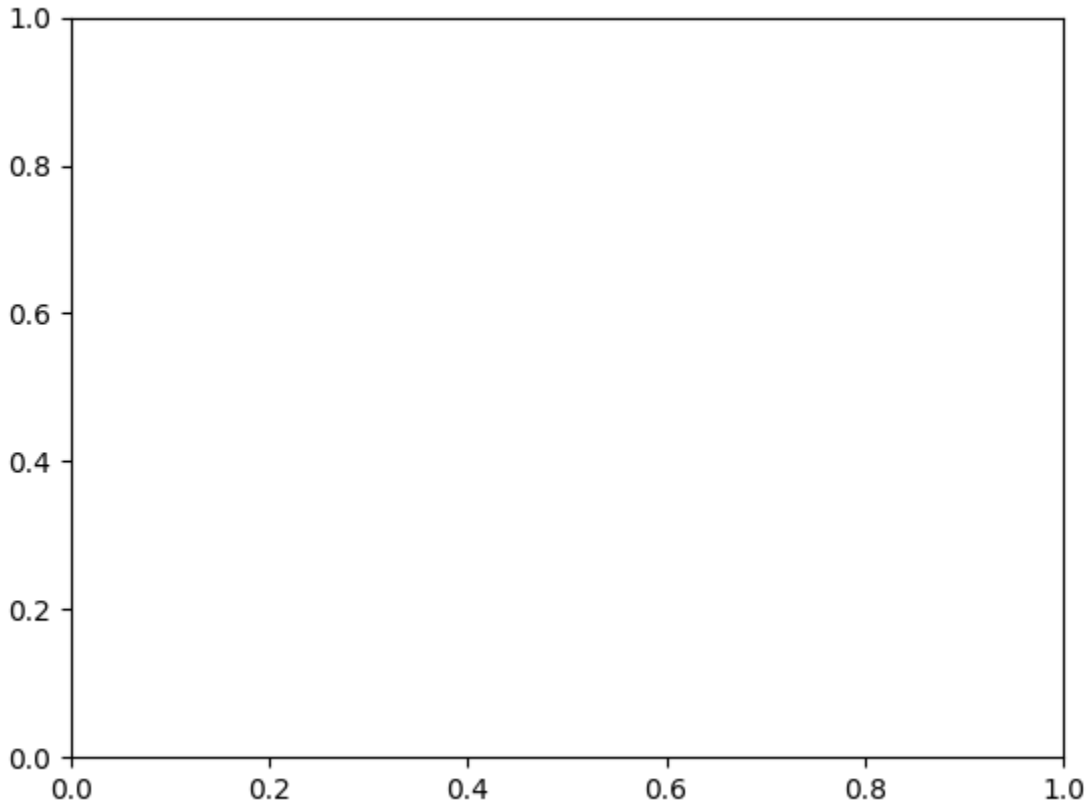


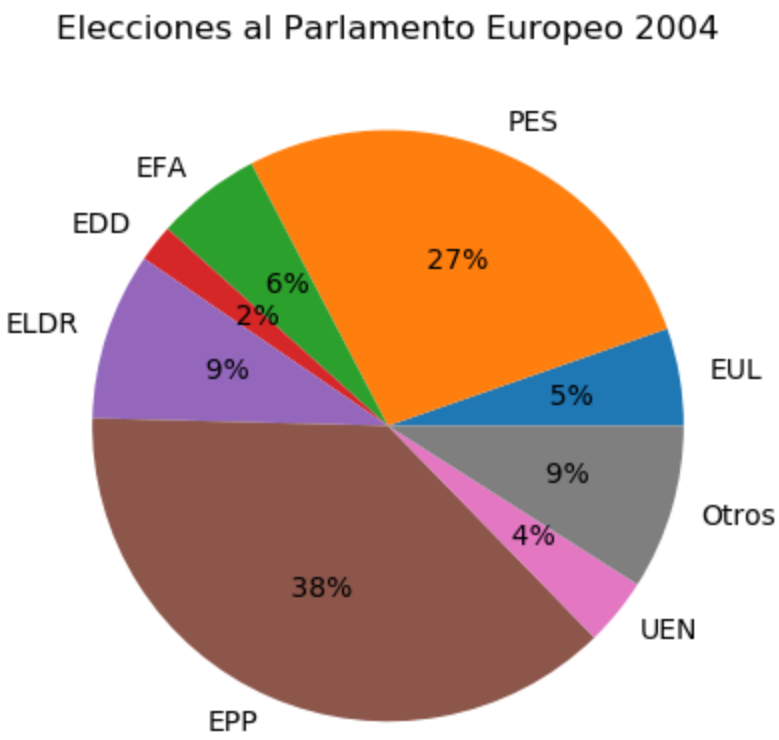
Gráfico circular.

Un gráfico circular (también conocido como gráfico de pastel) se crea con la función `Axes.pie()`.

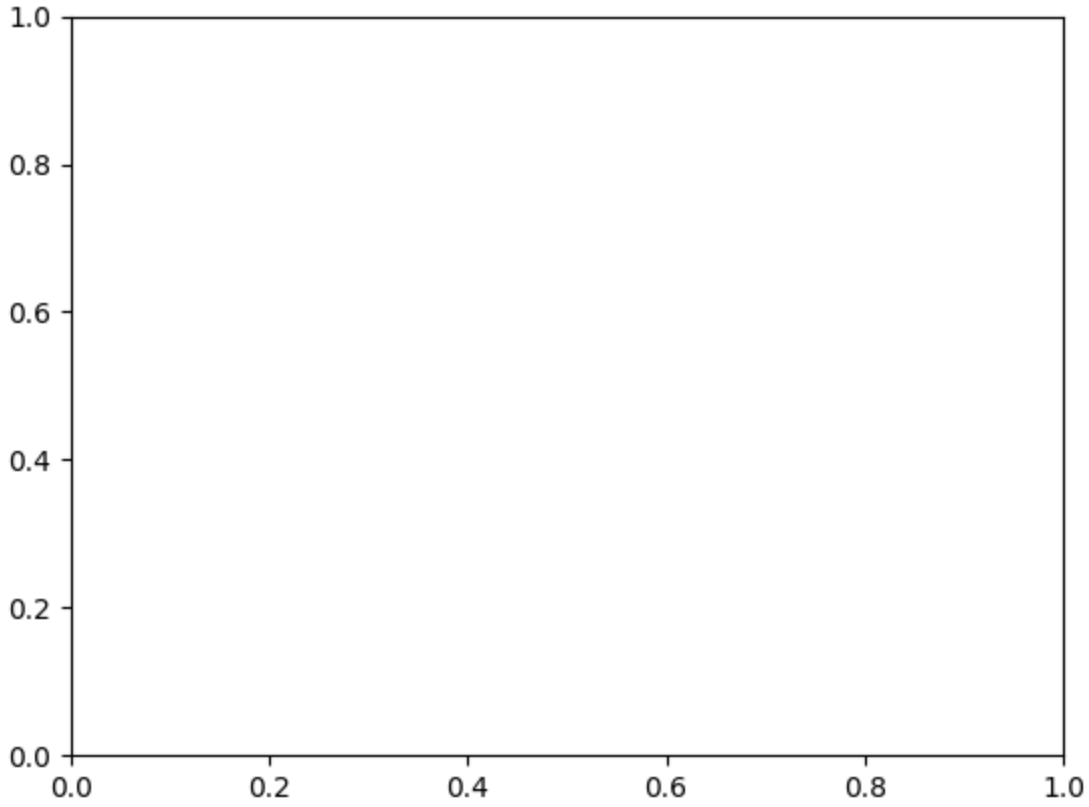
Estos gráficos están indicados para mostrar proporciones o porcentajes entre categorías, ya que cada categoría se representa con un sector con una amplitud proporcional a la frecuencia de ocurrencia de dicha categoría.

Ejercicio 04: Representar mediante un gráfico circular la distribución de asientos en las elecciones del año 2004.

Intenta ajustar los parámetros para que el resultado sea lo más parecido posible a la siguiente figura.



In [5]:
```
fig=plt.figure()
ax = plt.axes()
#Pon tu código aquí.
#Sugerencia: establece el título set_title() y las etiquetas
#de los ejes set_xlabel(), set_ylabel().

#

#Pon tu código aquí.
#Sugerencia: Utiliza la función Axes.pie()
#   Usa los valores de la serie como argumento 'x'.
#   Usa el índice del dataframe como el argumento 'labels'.
#   Usa el parámetro autopct para indicar que se muestre el
#porcentaje de cada etiqueta con tres dígitos enteros y un
#decimal.

#
```



Histogramas.

Para dibujar histogramas usaremos la función `Axes.hist()`.

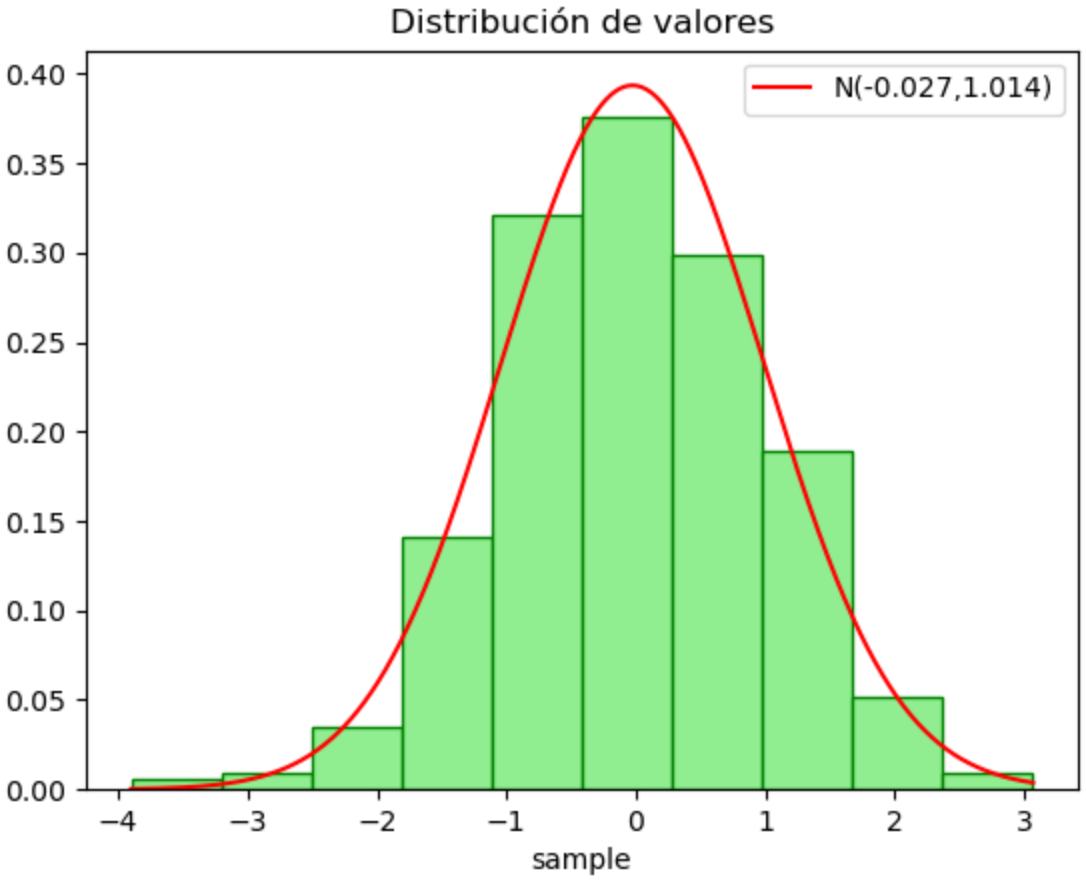
Además de dibujar el gráfico, la función devuelve las frecuencias, las posiciones de los bins y los intervalos, datos que pueden ser útiles para inferir nuevos datos.

Lo primero que vamos a hacer es generar una muestra de datos generados usando un proceso aleatorio con distribución gaussiana estándar.

In [6]:
```
gen = np.random.default_rng(0)
muestra = gen.standard_normal(500)
```

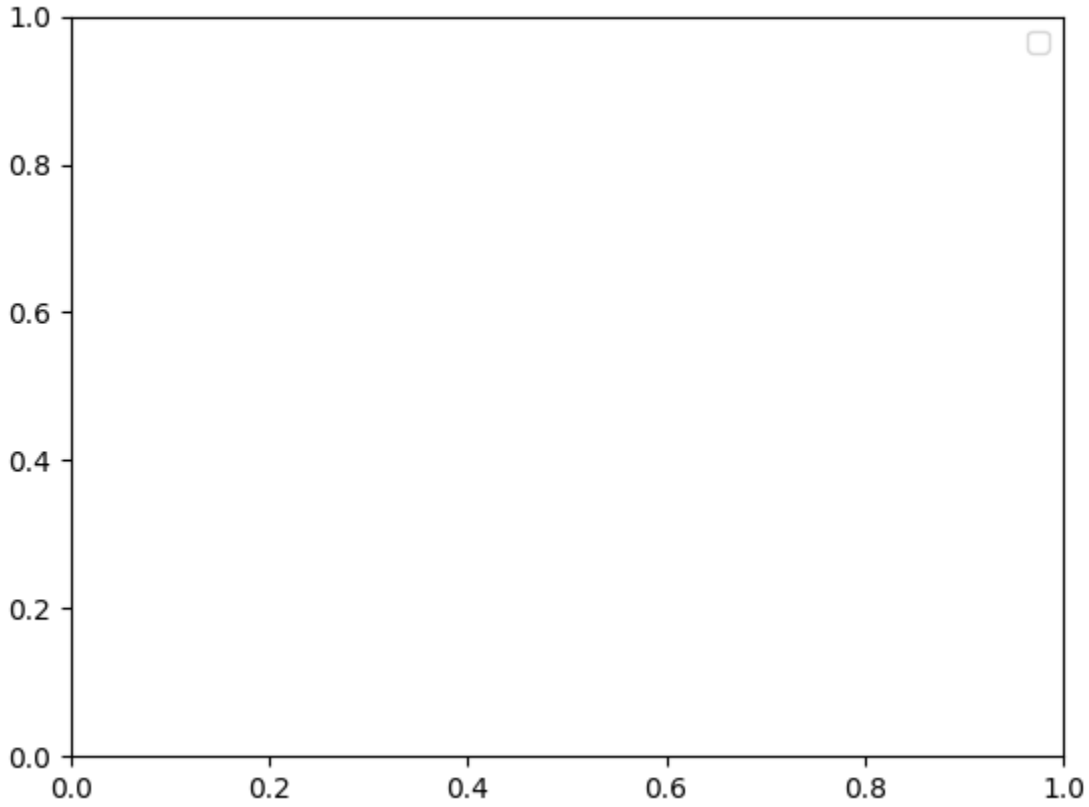
Ejercicio 05: Generar gráfico con el histograma de la variable sample usando 10 usando 10 clases (bins). Dado que pensamos que la distribución sigue una Normal, estimar los parámetros de la distribución y dibujar el modelo estimado sobre el histograma. Se desea mostrar las frecuencias normalizadas $[0, 1]$.

Intenta ajustar los parámetros para que el resultado sea lo más parecido posible a la siguiente figura.



In [7]:
```
fig = plt.figure()
ax = plt.axes()

#Pon tu código aquí
#Sugerencia: usa las funciones np.mean() y np.std() para
#estimar los parámetros de la Normal.

#

#Pon tu código aquí.
#Sugerencia: establece el título set_title() y las etiquetas
#de los ejes set_xlabel(), set_ylabel().

#

frecs=[]
bins=[]
ints=[]

#Pon tu código aquí.
#Sugerencia: usa hist()
#Indica que deseamos 10 intervalos.
#Indica que queremos que se calcule una densidad y elige
#un tipo de histograma 'bar'. Usa los parámetros color y
#edgecolor para especificar los colores.
#No olvides recoger el resultado.

#

#Pon tu código aquí.
#Sugerencia: genera un intervalo lineal desde el primer hasta
#el último valor de bins con 1000 muestras.
```

```
#Calcula los valores y correpondientes a una Normal(media, sigma).
#Dibuja un gráfico de línea con los valores x,y.
#Usa el parámetro label para asignar la leyenda.

#

ax.legend()
```




```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
```

Out[7]:
```
<matplotlib.legend.Legend at 0x7f41ac78c220>
```

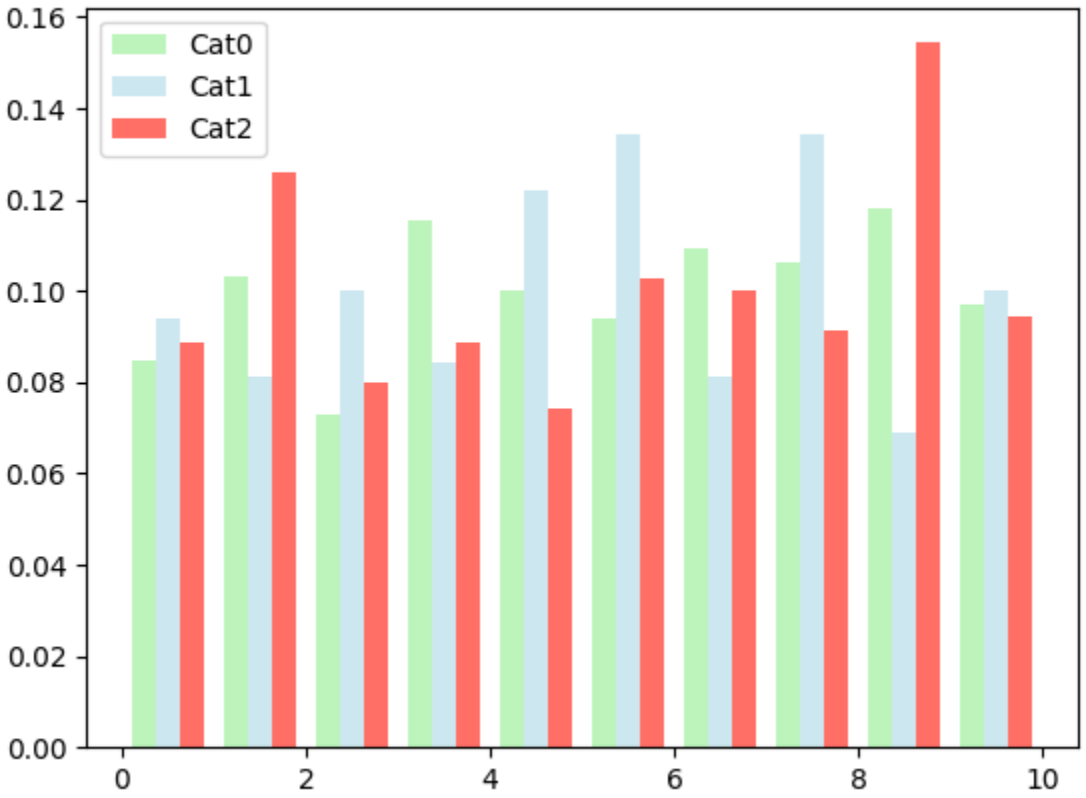
Multi histogramas.

Cuando queremos distinguir la distribución de los valores de una variable respecto a varias categorías, es útil mostrar cómo cada categoría contribuye a la distribución total.

Para ello, a partir de la muestra original, podemos obtener una submuestra por categoría y representar un multi histograma. Tenemos varias alternativas para mostrar tal histograma: dibujar una barra separada por categoría, o apilar (*"statcked"*) las barras.

Ejercicio 06: Dado un dataframe donde cada fila representa una muestra de una variable y la categoría asignada (de un total de 3 categorías). Se desea estudiar la distribución de los valores de la variable separando por categorías. Para ello generar un multi histograma separando las barras. Mostrar las frecuencias normalizadas.

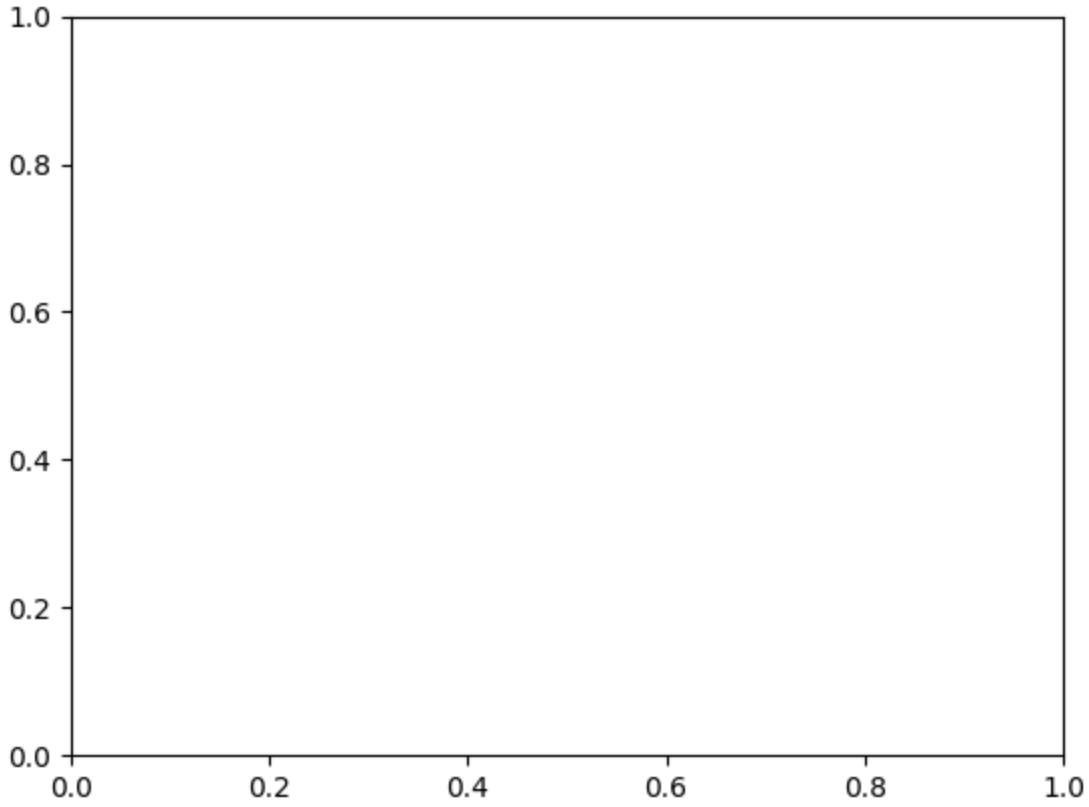
Intenta ajustar los parámetros para que el resultado sea lo más parecido posible a la siguiente figura.



In [8]:
```
gen = np.random.default_rng(0)
muestra = 10.0*gen.random(1000)
categorias = gen.integers(3, size=1000)
df = pd.DataFrame({'variable':muestra, 'categoria':categorias})
df.head()

m_0 = pd.Series()
m_1 = pd.Series()
m_2 = pd.Series()
#Pon tu código aquí.
#Sugerencia: utliza indixación con una máscara por categoría.

#

fig = plt.figure()
ax = plt.axes()

#Pon tu código aquí.
#Sugerencia: unsa axes.hist() para debes proporcionar una lista de distribuciones
#Como hay tres muestras, necesitamos indicar
#-tres colores en una lista.
#-tres etiquetas en una lista.

#
```

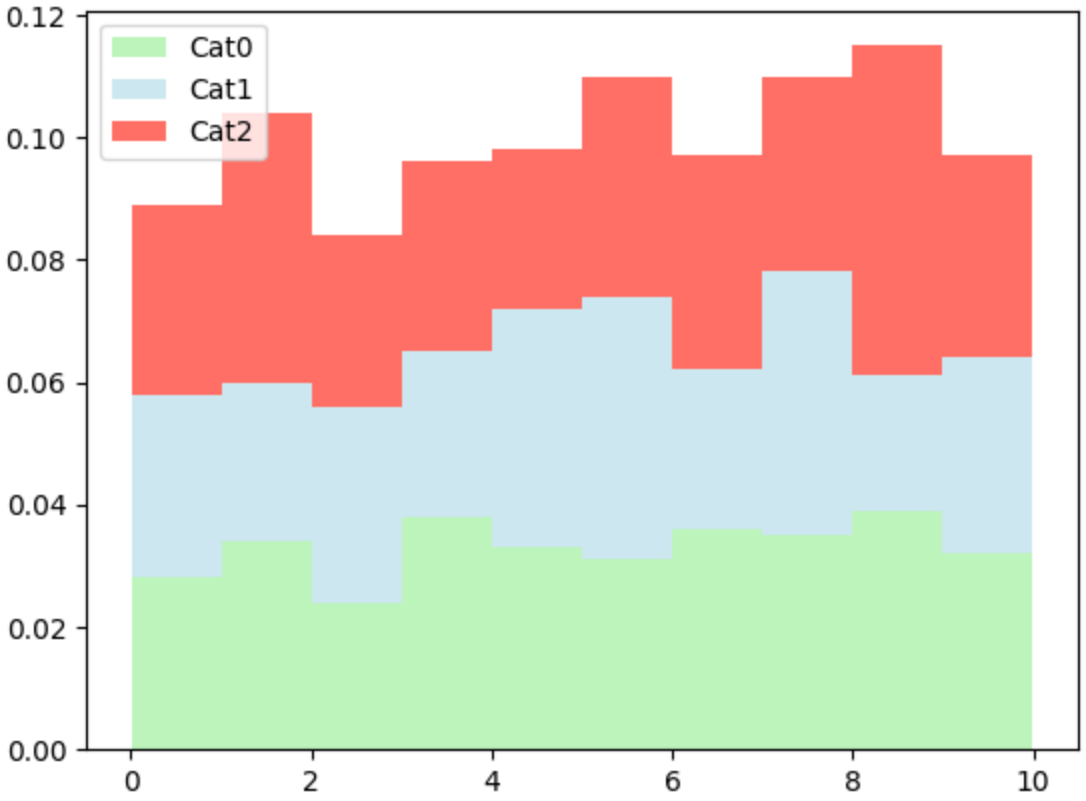
```
/tmp/ipykernel_38236/4032315787.py:7: FutureWarning: The default dtype for empty Series
will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to
silence this warning.
  m_0 = pd.Series()
/tmp/ipykernel_38236/4032315787.py:8: FutureWarning: The default dtype for empty Series
will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to
silence this warning.
```

```
  m_1 = pd.Series()
/tmp/ipykernel_38236/4032315787.py:9: FutureWarning: The default dtype for empty Series
will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to
silence this warning.
  m_2 = pd.Series()
```



Ejercicio 07: Repetir el ejercicio anterior pero usando un multi histograma apilado ("stacked").

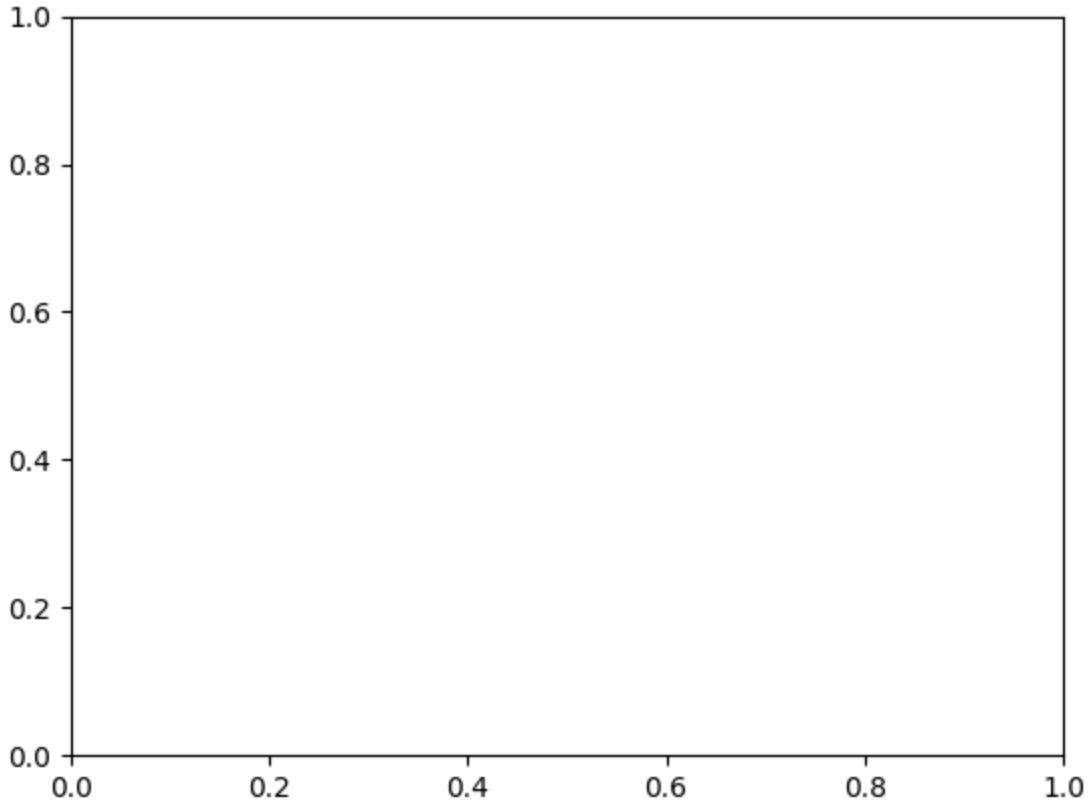
Intenta ajustar los parámetros para que el resultado sea lo más parecido posible a la siguiente figura.



In [9]:
```
gen = np.random.default_rng(0)
muestra = 10.0*gen.random(1000)
categorias = gen.integers(3, size=1000)
df = pd.DataFrame({'variable':muestra, 'categoria':categorias})
df.head()

m_0 = pd.Series()
m_1 = pd.Series()
m_2 = pd.Series()
#Pon tu código aquí.
#Sugerencia: utliza indixación con una máscara por categoría.

#

fig = plt.figure()
ax = plt.axes()

#Pon tu código aquí.
#Sugerencia: unsa axes.hist() para debes proporcionar una lista de distribuciones
#Como hay tres muestras, necesitamos indicar
#-tres colores en una lista.
#-tres etiquetas en una lista.

#
```

```
/tmp/ipykernel_38236/4032315787.py:7: FutureWarning: The default dtype for empty Series
will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to
silence this warning.
  m_0 = pd.Series()
/tmp/ipykernel_38236/4032315787.py:8: FutureWarning: The default dtype for empty Series
will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to
silence this warning.
```

```
  m_1 = pd.Series()
/tmp/ipykernel_38236/4032315787.py:9: FutureWarning: The default dtype for empty Series
will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to
silence this warning.
  m_2 = pd.Series()
```



In [ ]: