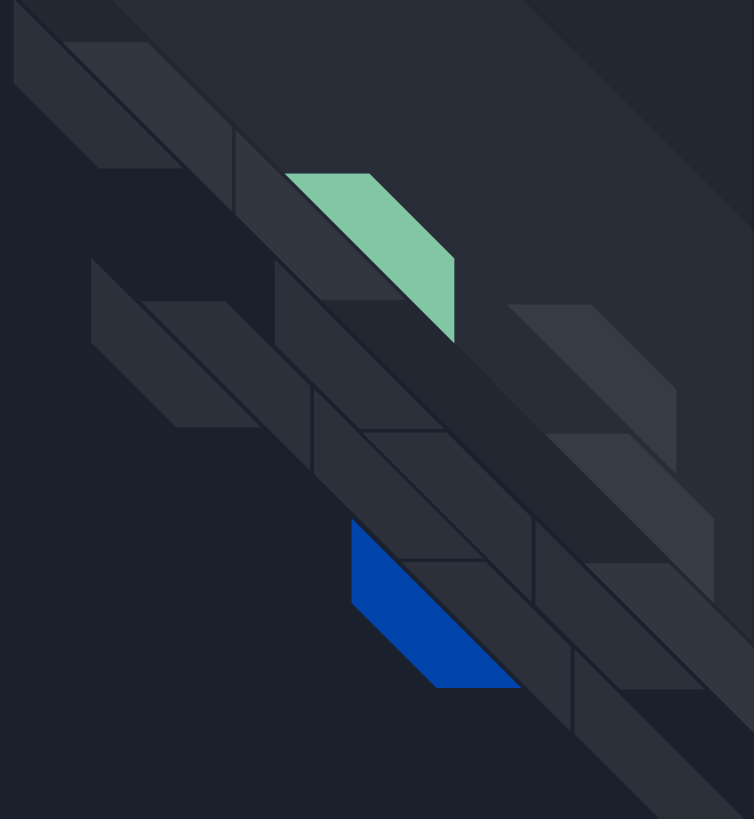


Introducción al lenguaje Python 3º clase

Técnicas de aprendizaje automático - Machine Learning

Temas a ver hoy

- ❖ Lectura y escritura de datos
- ❖ Introducción a la librería Matplotlib
 - Gráfico básico
 - Dispersión
 - Barras
 - Histograma
 - Torta
 - Boxplot





Lectura y escritura de datos

En ciencia de datos y Machine Learning es común que los datos se transfieran en un tipo de archivo desprovisto de formato (a diferencia, por ejemplo, de un archivo Excel) para mejorar la compatibilidad entre librerías y programas.

Un formato de archivo de este tipo es el de extensión CSV (comma separated values). Es parecido a un archivo .XLSX en el cual la información tabular está contenida en varias filas pero en una sola columna, donde se utiliza una coma (,) para significar la separación de datos (similar a los elementos en una lista de Python).

Por lo general, en la primer fila del archivo, se tiene una descripción del contenido de las columnas, lo que se conoce como header.



Lectura y escritura de datos

Las librerías de Python destinadas al tratamiento de datos (NumPy, Pandas) permiten guardar y leer información en formato CSV, para utilizarlos como objetos de Python (ndarray, data frame).

Vamos a ver la implementación de su uso en NumPy.

Nd-Array a CSV

fmt= '%f'
'%d'

para floats
para enteros

```
import numpy as np

# Create a sample NumPy array
data = np.array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9.8]])

# Specify the filename and save the array to a CSV file
filename = "data.csv"
header = 'col1,col2,col3'
np.savetxt(filename, data, delimiter=',', fmt='%f', header = header)
```



CSV a nd-Array

```
# Specify the filename of the CSV file
filename = "data.csv"

# Read the CSV file into a NumPy array
data = np.genfromtxt(filename, delimiter=',', skip_header=1, dtype='int32')
```



Aclaración

Al declarar un nombre para el archivo, se supone que el mismo se encuentra en la misma dirección que el directorio de trabajo. Si no es así, se debe especificar la dirección completa del archivo.

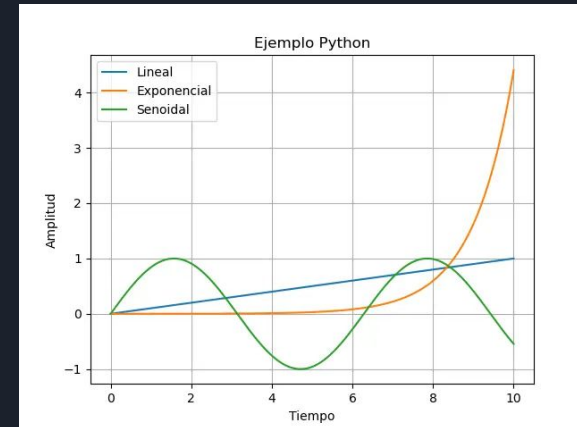
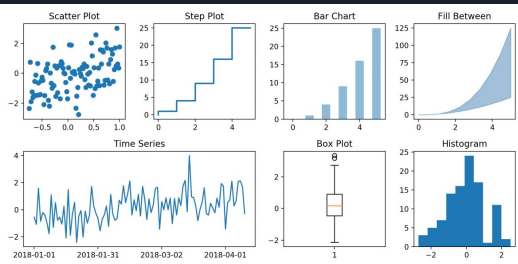
El directorio de trabajo (CWD: current working directory) cambia si se trabaja desde una terminal o desde un editor de código.

En el siguiente enlace se explica cómo declarar las direcciones de los archivos, tanto en Windows como en Linux, y como conocer y cambiar el CWD.

[File Path and CWD](#)

Introducción a la librería Matplotlib

matplotlib





Introducción a la librería Matplotlib

Matplotlib es una biblioteca de visualización de datos en el lenguaje de programación Python. Es ampliamente utilizada para crear gráficos estáticos, interactivos y animaciones en Python. Esta biblioteca proporciona una amplia variedad de funciones y herramientas para crear diferentes tipos de gráficos y visualizaciones.

[Documentación oficial de Matplotlib](#)

```
matplotlib.pyplot
```

`matplotlib.pyplot` is a state-based interface to matplotlib. It provides an implicit, MATLAB-like, way of plotting. It also opens figures on your screen, and acts as the figure GUI manager.

pyplot is mainly intended for interactive plots and simple cases of programmatic plot generation:



Matplotlib

matplotlib.pyplot.plot

```
matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)
```

Parameters:

x, y : *array-like or scalar*

The horizontal / vertical coordinates of the data points. *x* values are optional and default to `range(len(y))`.

Commonly, these parameters are 1D arrays.

They can also be scalars, or two-dimensional (in that case, the columns represent separate data sets).

These arguments cannot be passed as keywords.

fmt : *str, optional*

A format string, e.g. 'ro' for red circles. See the *Notes* section for a full description of the format strings.

Format strings are just an abbreviation for quickly setting basic line properties. All of these and more can also be controlled by keyword arguments.

This argument cannot be passed as keyword.

data : *indexable object, optional*

An object with labelled data. If given, provide the label names to plot in *x* and *y*.

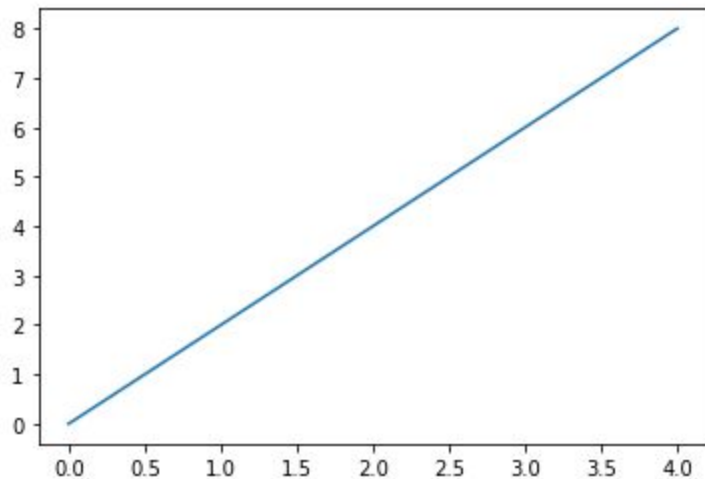
```
import matplotlib.pyplot as plt
import numpy as np
```

✓ 1.7s

```
x = [0,1,2,3,4]
y = [0,2,4,6,8]
```

```
plt.plot(x,y)
plt.show()
```

✓ 0.1s



Matplotlib

****kwargs :** `Line2D` *properties, optional*

kwargs are used to specify properties like a line label (for auto legends), linewidth, antialiasing, marker face color. Example:

```
>>> plot([1, 2, 3], [1, 2, 3], 'go-', label='line 1', linewidth=2)
>>> plot([1, 2, 3], [1, 4, 9], 'rs', label='line 2')
```

If you specify multiple lines with one plot call, the *kwargs* apply to all those lines. In case the label object is iterable, each element is used as labels for each set of data. Here is a list of available `Line2D` properties:

Property	Description
<code>agg_filter</code>	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array and two offsets from the bottom left corner of the image
<code>alpha</code>	scalar or None
<code>animated</code>	bool
<code>antialiased</code> or <code>aa</code>	bool
<code>clip_box</code>	<code>Bbox</code>
<code>clip_on</code>	bool

`markeredgecolor` or color

`mec`

`markeredgewidth` or float

`mew`

`markerfacecolor` or color

`mfc`

`markerfacecoloralt` color

or `mfcalt`

`markersize` or `ms` float

`markevery`

None or int or (int, int) or slice or list[int] or float or (float, float) or list[bool]

`mouseover`

bool

`path_effects`

`AbstractPathEffect`

`picker`

float or callable[[Artist, Event], tuple[bool, dict]]

`pickradius`

unknown

`rasterized`

bool

Matplotlib

Markers

character	description
'.'	point marker
'.'	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker

's'	square marker
'p'	pentagon marker
'p'	plus (filled) marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'X'	x (filled) marker
'D'	diamond marker
'd'	thin_diamond marker
' '	vline marker

Matplotlib

Line Styles

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
'...'	dotted line style

Colors

The supported color abbreviations are the single letter codes

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white



Matplotlib

Format Strings

A format string consists of a part for color, marker and line:

```
fmt = '[marker][line][color]'
```

Each of them is optional. If not provided, the value from the style cycle is used. Exception: If `line` is given, but no `marker`, the data will be a line without markers.

Other combinations such as `[color][marker][line]` are also supported, but note that their parsing may be ambiguous.

If the color is the only part of the format string, you can additionally use any `matplotlib.colors` spec, e.g. full names (`'green'`) or hex strings (`'#008000'`).

Matplotlib

```
x = [0,1,2,3,4]
```

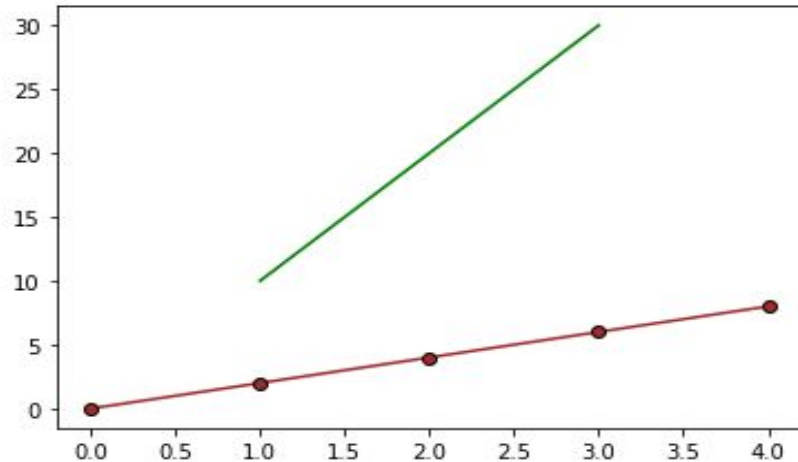
```
y = [0,2,4,6,8]
```

```
plt.plot([1,2,3],[10,20,30], color='green')
```

```
plt.plot(x,y, color = 'brown', marker = 'o', mec='black')
```

```
plt.show()
```

✓ 0.1s





Matplotlib

matplotlib.pyplot.title

```
matplotlib.pyplot.title(Label, fontdict=None, loc=None, pad=None, *, y=None,  
**kwargs) \[source\]
```

Set a title for the Axes.

Set one of the three available Axes titles. The available titles are positioned above the Axes in the center, flush with the left edge, and flush with the right edge.

Parameters:

label : str

Text to use for the title

fontdict : dict

A dictionary controlling the appearance of the title text, the default *fontdict* is:

```
{'fontsize': rcParams['axes.titlesize'],  
'fontweight': rcParams['axes.titleweight'],  
'color': rcParams['axes.titlecolor'],  
'verticalalignment': 'baseline',  
'horizontalalignment': loc}
```



Matplotlib

loc : {*center*, *left*, *right*}, default: `rcParams["axes.titlelocation"]` (default: `'center'`)

Which title to set.

y : float, default: `rcParams["axes.titlesize"]` (default: `None`)

Vertical Axes location for the title (1.0 is the top). If `None` (the default) and `rcParams["axes.titlesize"]` (default: `None`) is also `None`, `y` is determined automatically to avoid decorators on the Axes.

pad : float, default: `rcParams["axes.titlepad"]` (default: `6.0`)

The offset of the title from the top of the Axes, in points.

```
x = [0,1,2,3,4]
```

```
y = [0,2,4,6,8]
```

```
plt.plot([1,2,3],[10,20,30], color='green')
```

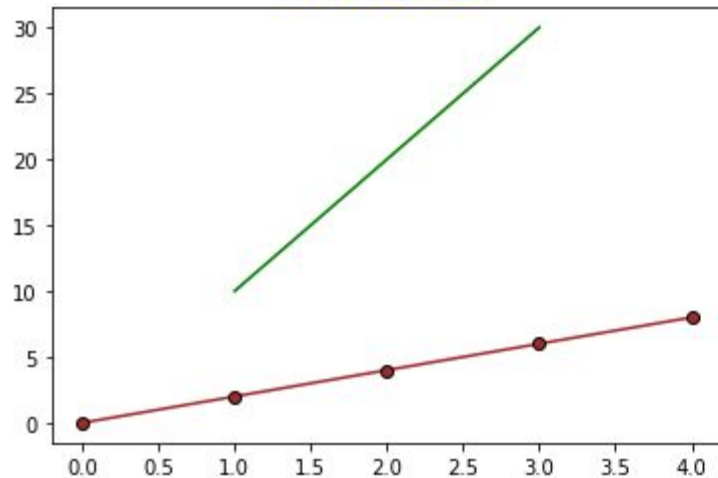
```
plt.plot(x,y, color = 'brown', marker = 'o', mec='black')
```

```
plt.title('Titulo', fontdict={'fontsize':40, 'fontname': 'Lucida Sans', 'color': 'blue'})
```

```
plt.show()
```

✓ 0.1s

Titulo





matplotlib.pyplot.figure

```
matplotlib.pyplot.figure(num=None, figsize=None, dpi=None, *, facecolor=None,
    edgecolor=None, frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>,
    clear=False, **kwargs) [source]
```

Create a new figure, or activate an existing figure.

matplotlib.pyplot.savefig

```
matplotlib.pyplot.savefig(*args, **kwargs) [source]
```

Save the current figure.

Call signature:

```
savefig(fname, *, dpi='figure', format=None, metadata=None,
    bbox_inches=None, pad_inches=0.1,
    facecolor='auto', edgecolor='auto',
    backend=None, **kwargs
)
```

The available output formats depend on the backend being used.

Parameters:

fname : *str or path-like or binary file-like*

A path, or a Python file-like object, or possibly some backend-dependent object such

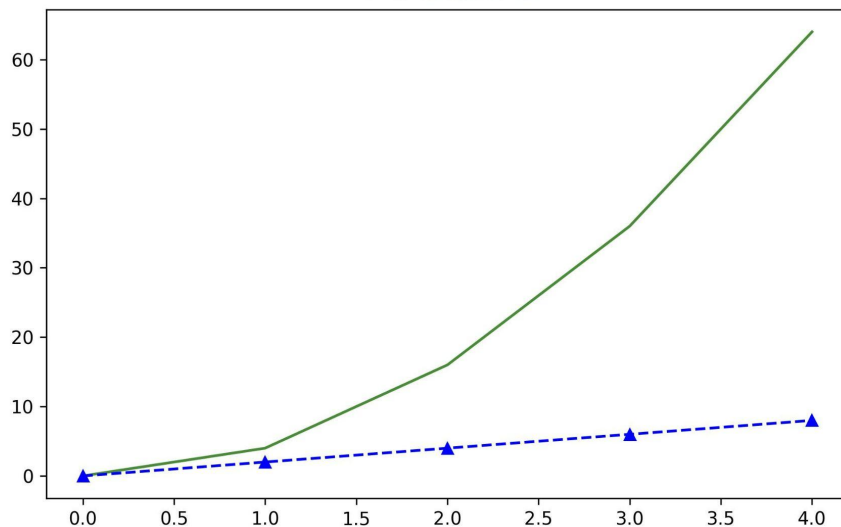
as `matplotlib.backends.backend_pdf.PdfPages`.

```
x = np.array([0,1,2,3,4])
y = np.array([0,2,4,6,8])
plt.figure(figsize=(8,5), dpi=100)
plt.plot(x,y**2, color='#4c8a3b') #Buscar en Google "color picker"
plt.plot(x,y, 'b^--', label='2x')
plt.title('Titulo', fontdict={'fontsize':40, 'fontname': 'Lucida Sans', 'color': '#a84232'})
plt.savefig('grafico.jpg', dpi=300)
plt.show()
```

Savefig debe ir antes de show()

✓ 0.2s

Titulo





```
# select interval we want to plot points at
x2 = np.arange(0,4.5,0.5)

# Plot part of the graph as line
plt.plot(x2[:6], x2[:6]**2, 'r', label='X^2')

# Plot remainder of graph as a dot
plt.plot(x2[5:], x2[5:]**2, 'r--')

# Add a title (specify font parameters with fontdict)
plt.title('Título del gráfico', fontdict={'fontname': 'Georgia', 'fontsize': 20})

# X and Y labels
plt.xlabel('X Axis')
plt.ylabel('Y Axis')

# X, Y axis Tickmarks (scale of your graph)
plt.xticks([0,1,2,3,4,])
plt.yticks([0,2,4,6,8,10])

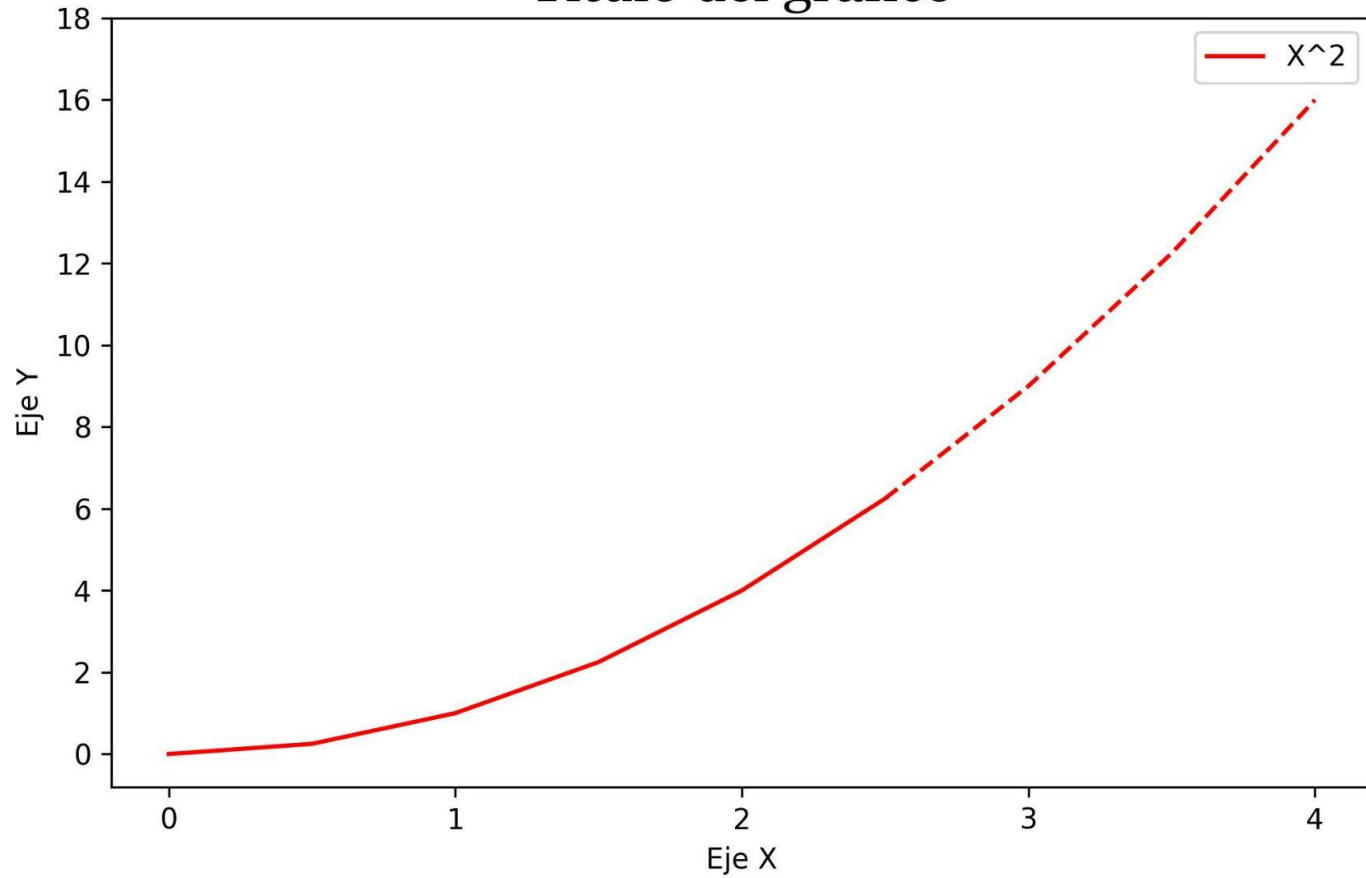
# Add a legend
plt.legend()

# Save figure (dpi 300 is good when saving so graph has high resolution)
plt.savefig('mygraph.png', dpi=300)

# Show plot
plt.show()
```



Título del gráfico



matplotlib.style

Styles are predefined sets of `rcParams` that define the visual appearance of a plot.

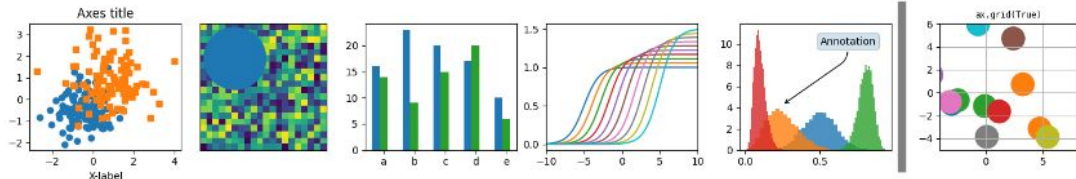
Customizing Matplotlib with style sheets and rcParams describes the mechanism and usage of styles.

The Style sheets reference gives an overview of the builtin styles.

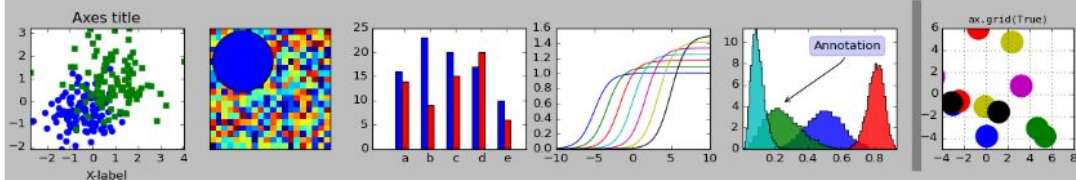
Style sheets reference

This script demonstrates the different available style sheets on a common set of example plots: scatter plot, image, bar graph, patches, line plot and histogram,

default



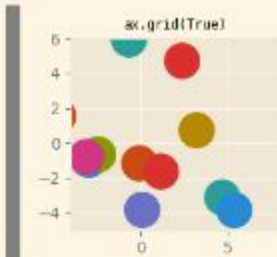
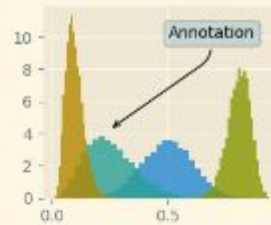
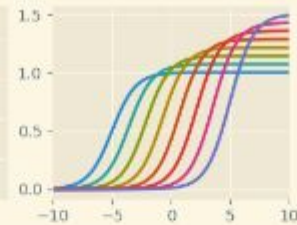
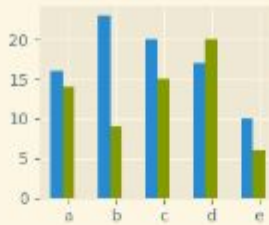
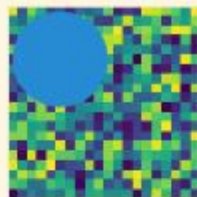
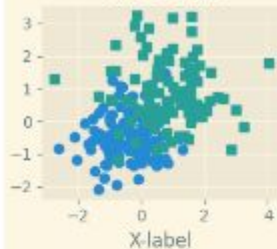
classic



Style sheets reference

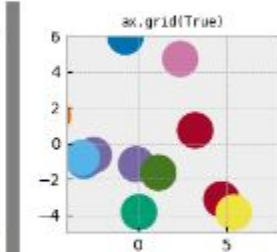
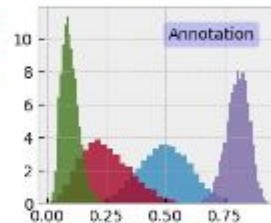
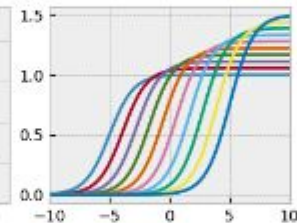
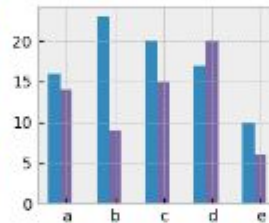
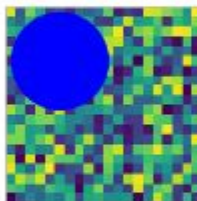
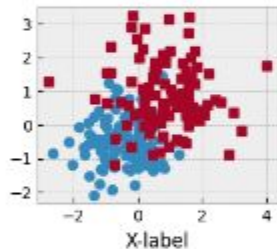
Solarize_Light2

Axes title



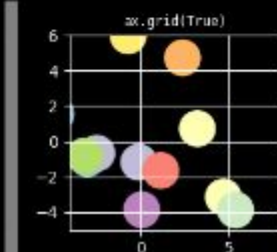
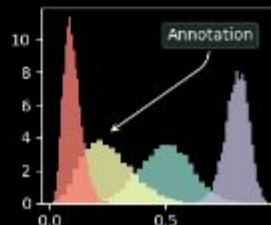
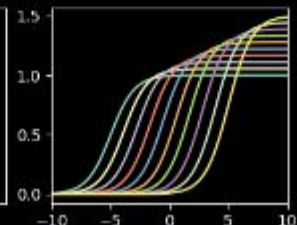
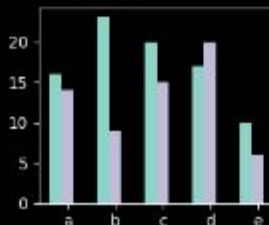
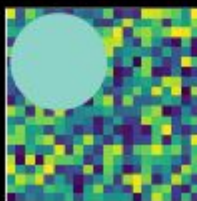
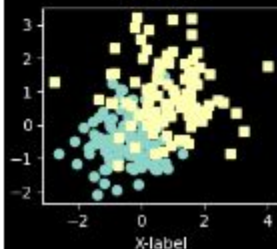
bmh

Axes title

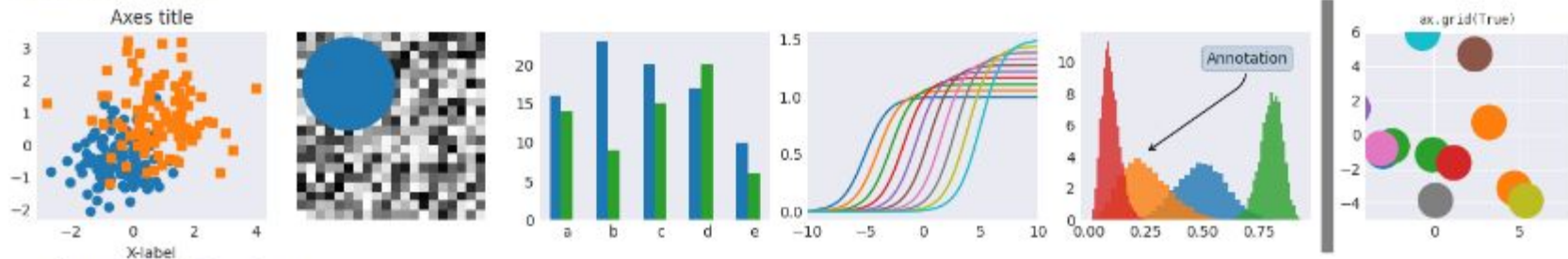


dark_background

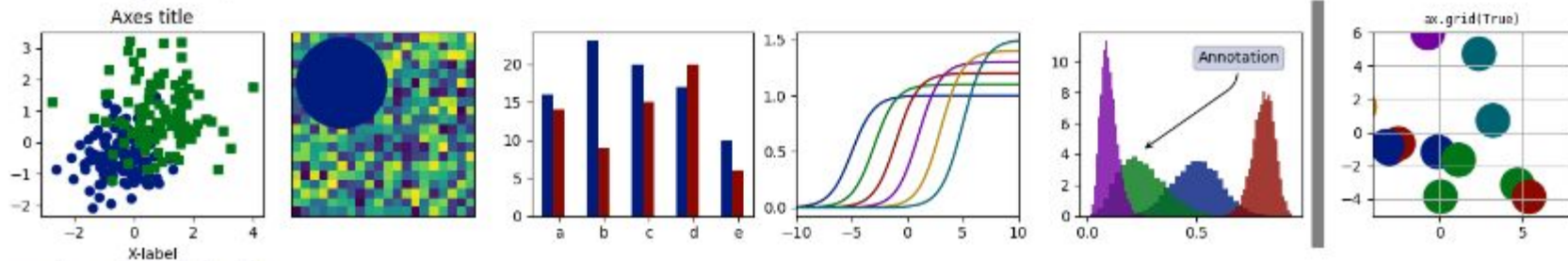
Axes title



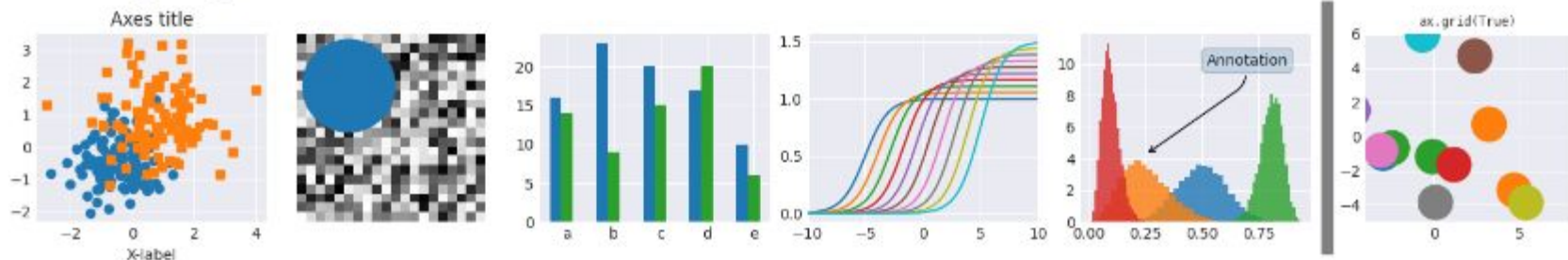
seaborn-v0_8-dark



seaborn-v0_8-dark-palette



seaborn-v0_8-darkgrid



```
import matplotlib
matplotlib.style.use('Solarize_Light2')

x = [0,1,2,3,4]
y = [0,2,4,6,8]
plt.title('Titulo')
plt.plot([0,1,2,3,4],[3,10,30,84,92])
plt.plot(x,y)

plt.show()
```

✓ 0.1s



Gas Prices over Time (in USD)

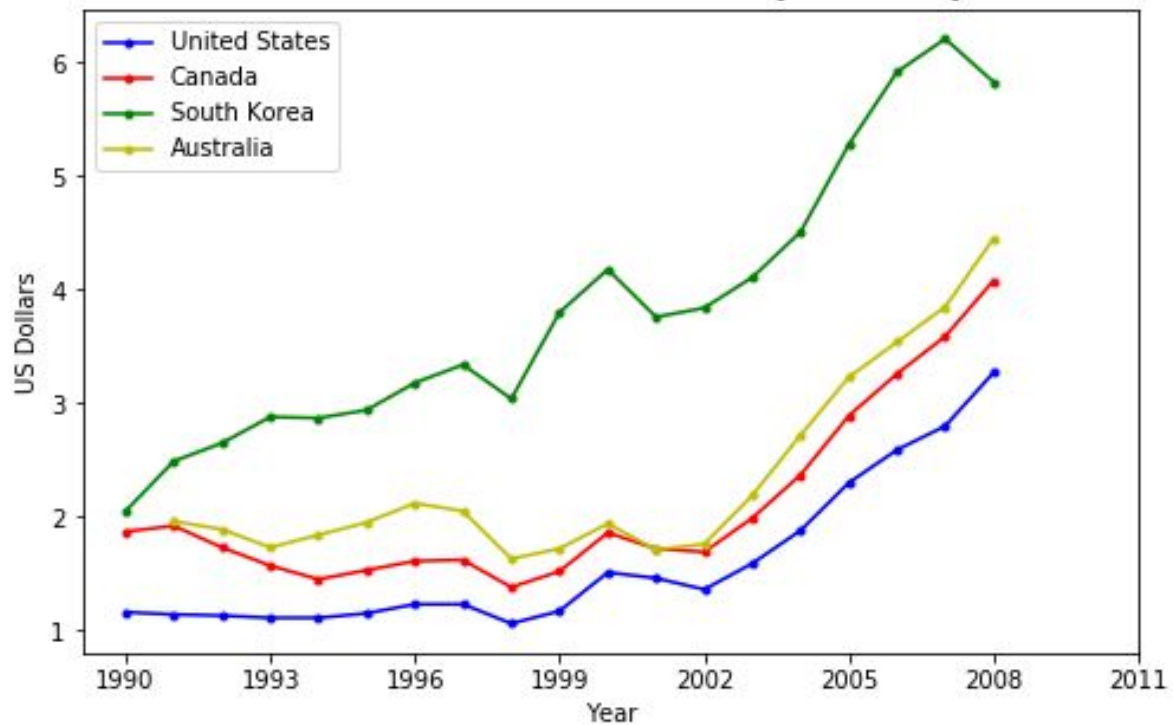


Gráfico de dispersión

matplotlib.pyplot.scatter

```
matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None,  
norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, *, edgecolors=None,  
plotnonfinite=False, data=None, **kwargs) \[source\]
```

A scatter plot of y vs. x with varying marker size and/or color.

Parameters:

x, y : float or array-like, shape (n,)

The data positions.

s : float or array-like, shape (n,), optional

The marker size in points**2 (typographic points are 1/72 in.). Default is

```
rcParams['lines.markersize'] ** 2.
```

c : array-like or list of colors or color, optional

The marker colors. Possible values:

- A scalar or sequence of n numbers to be mapped to colors using *cmap* and *norm*.
- A 2D array in which the rows are RGB or RGBA.
- A sequence of colors of length n .
- A single color format string.

```
# Sample data for GDP per capita and life expectancy
countries = ['USA', 'China', 'Germany', 'Brazil']
gdp_per_capita = np.array([1995,1996,1997,1998,1999,2000,2001]) # Año
life_expectancy = np.array([
    [1,2,3,4,5,6,7],
    [20.9,39.3,49.9,23.8,58.8,62.8,74.7],
    [8,14,15,16,16,17,14],
    [10,27,25,27,24,25,33]
])

# Different colors for each country
colors = ['blue', 'green', 'red', 'purple']

# Create a scatter plot
plt.figure(figsize=(10, 6))

for i in range(len(countries)):
    plt.scatter(gdp_per_capita, life_expectancy[i], color=colors[i], label=countries[i], s=100)

plt.title('GDP per Capita vs Life Expectancy')
plt.xlabel('GDP per Capita (USD)')
plt.ylabel('Life Expectancy (Years)')
plt.grid(True)
plt.legend()

plt.show()
```



GDP per Capita vs Life Expectancy

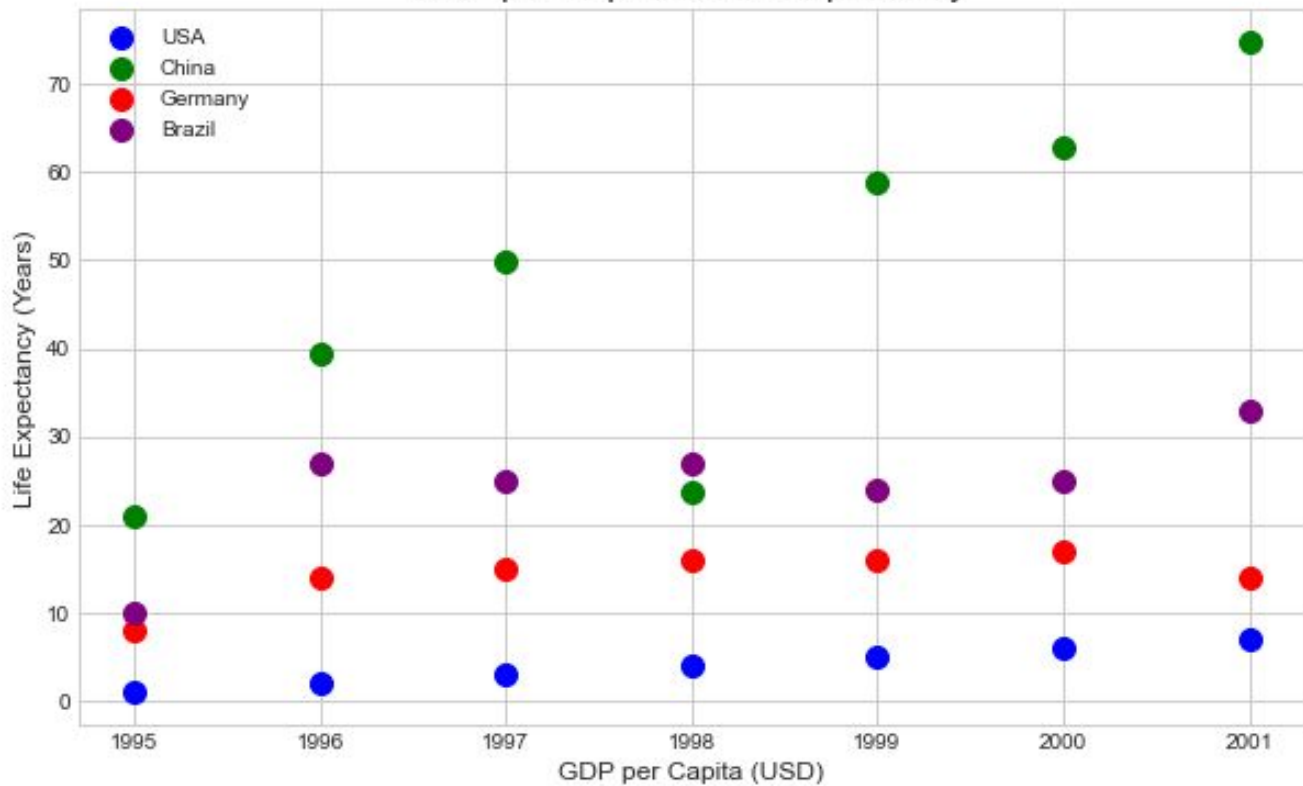




Gráfico de barras

matplotlib.pyplot.bar

```
matplotlib.pyplot.bar(x, height, width=0.8, bottom=None, *, align='center',  
data=None, **kwargs) \[source\]
```

Make a bar plot.

The bars are positioned at *x* with the given *alignment*. Their dimensions are given by *height* and *width*. The vertical baseline is *bottom* (default 0).

Many parameters can take either a single value applying to all bars or a sequence of values, one for each bar.



Gráfico de barras

Parameters:

x : *float or array-like*

The x coordinates of the bars. See also *align* for the alignment of the bars to the coordinates.

height : *float or array-like*

The height(s) of the bars.

width : *float or array-like, default: 0.8*

The width(s) of the bars.

bottom : *float or array-like, default: 0*

The y coordinate(s) of the bottom side(s) of the bars.

align : *{'center', 'edge'}, default: 'center'*

Alignment of the bars to the x coordinates:

- 'center': Center the base on the x positions.
- 'edge': Align the left edges of the bars with the x positions.

To align the bars on the right edge pass a negative *width* and `align='edge'`.



Gráfico de barras

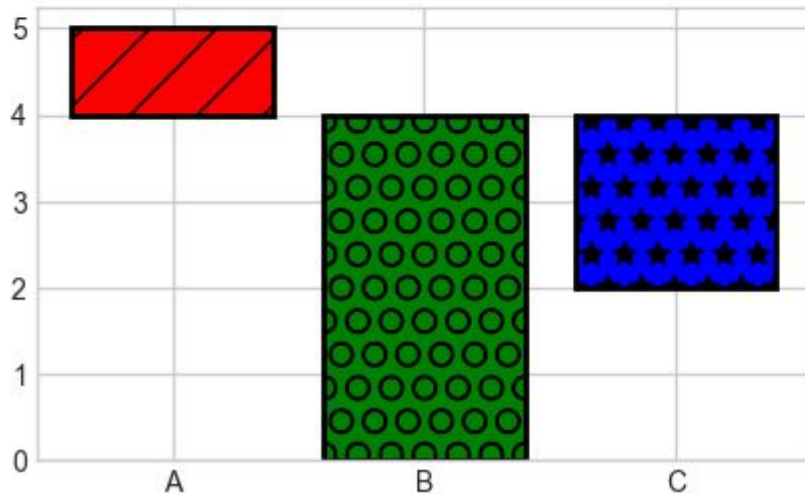
<code>color</code>	color
<code>edgecolor</code> or <code>ec</code>	color or None
<code>facecolor</code> or <code>fc</code>	color or None
<code>figure</code>	<code>Figure</code>
<code>fill</code>	bool
<code>gid</code>	str
<code>hatch</code>	{ '/', '\', ' ', '-', '+', 'x', 'o', 'O', '.', '*' }
<code>height</code>	unknown
<code>in_layout</code>	bool
<code>joinstyle</code>	<code>JoinStyle</code> or { 'miter', 'round', 'bevel' }
<code>label</code>	object
<code>linestyle</code> or <code>ls</code>	{ '-', '--', '-.', ':', ' ', (offset, on-off-seq), ... }
<code>linewidth</code> or <code>lw</code>	float or None
<code>mouseover</code>	bool

```
labels = ['A', 'B', 'C']
values = [1,4,2]
plt.figure(figsize=(5,3), dpi=100)
```

```
bars = plt.bar(labels, values, 0.8, [4,0,2],edgecolor='black', color=['red', 'green', 'blue'],
hatch = [ '/', 'O', '*'], linewidth=2)
```

```
plt.show()
```

✓ 0.0s

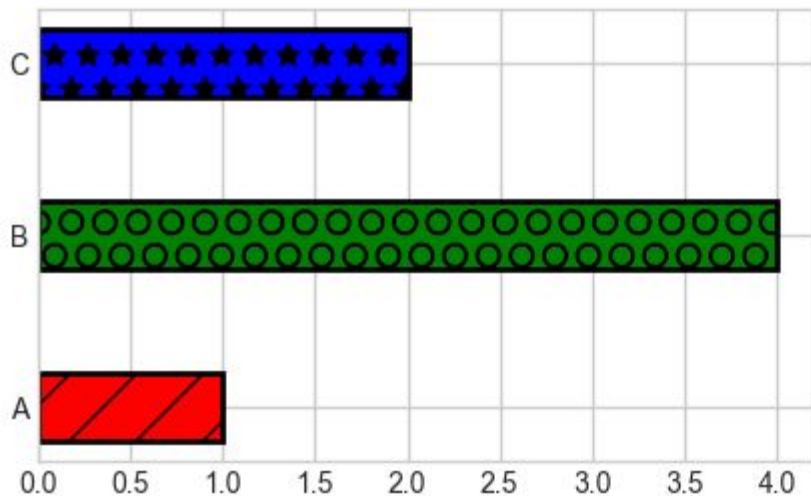


```
labels = ['A', 'B', 'C']
values = [1,4,2]
plt.figure(figsize=(5,3), dpi=100)

plt.barh(labels, values, 0.4,edgecolor='black', color=['red', 'green', 'blue'],
hatch = ['/', 'O', '*'], linewidth=2)

plt.show()
```

✓ 0.1s





Histograma

matplotlib.pyplot.hist

```
matplotlib.pyplot.hist(x, bins=None, range=None, density=False, weights=None,  
cumulative=False, bottom=None, histtype='bar', align='mid',  
orientation='vertical', rwidth=None, log=False, color=None, label=None,  
stacked=False, *, data=None, **kwargs) \[source\]
```

Compute and plot a histogram.

This method uses `numpy.histogram` to bin the data in `x` and count the number of values in each bin, then draws the distribution either as a `BarContainer` or `Polygon`. The `bins`, `range`, `density`, and `weights` parameters are forwarded to `numpy.histogram`.



Histograma

```
fifa = np.genfromtxt('fifa_data.csv', delimiter=',', skip_header=1, dtype='str', encoding='UTF-8', invalid_raise=False, missing_values='', usemask=False, filling_values=0.0)
```

```
fifa = fifa.T
```

✓ 1.2s

```
skills=[]  
for i in range(len(fifa[7])):  
    elem = float(fifa[7,i])  
    skills.append(elem)
```

✓ 0.0s



```
bins = [40,50,60,70,80,90,100]

plt.figure(figsize=(8,5))

plt.hist(skills, bins=bins, color='#abcdef')

plt.xticks(bins)

plt.ylabel('Number of Players')
plt.xlabel('Skill Level')
plt.title('Distribution of Player Skills in FIFA 2018')

plt.savefig('histogram.png', dpi=300)

plt.show()
```

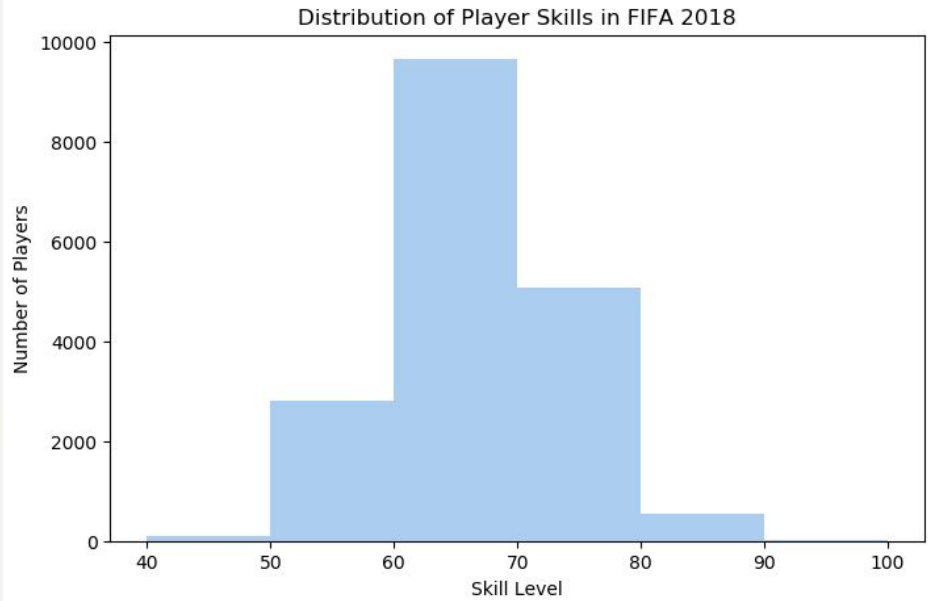




Gráfico de torta

matplotlib.pyplot.pie

```
matplotlib.pyplot.pie(x, explode=None, labels=None, colors=None, autopct=None,  
pctdistance=0.6, shadow=False, labeldistance=1.1, startangle=0, radius=1,  
counterclock=True, wedgeprops=None, textprops=None, center=(0, 0), frame=False,  
rotatelabels=False, *, normalize=True, hatch=None, data=None) \[source\]
```

Plot a pie chart.

Make a pie chart of array x . The fractional area of each wedge is given by $x/\text{sum}(x)$.

The wedges are plotted counterclockwise, by default starting from the x-axis.

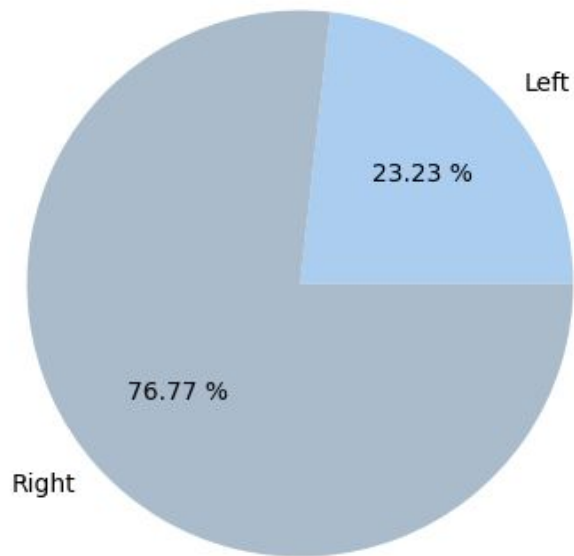


```
right=0
left=0

for i in range(len(fifa[14])):
    if fifa[14,i]=='Right':
        right+=1
    elif fifa[14,i]=='Left':
        left+=1

plt.figure(figsize=(8,5))
labels = ['Left', 'Right']
colors = ['#abcdef', '#aabbcc']
plt.pie([left, right], labels = labels, colors=colors, autopct='%.2f %%')
plt.title('Foot Preference of FIFA Players')
plt.show()
```

Foot Preference of FIFA Players



```
plt.figure(figsize=(8,5), dpi=100)
plt.style.use('ggplot')

light = 0
light_medium = 0
medium = 0
medium_heavy = 0
heavy = 0

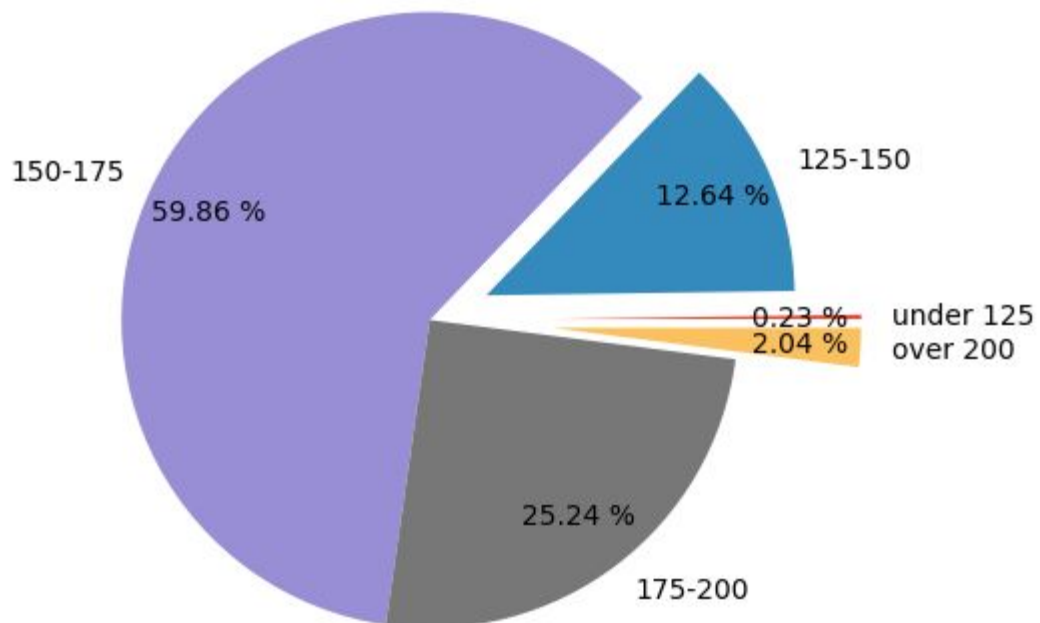
for i in range(len(fifa[28])):
    elem = int(fifa[28,i].strip('lbs'))
    if elem<125:
        light+=1
    elif elem>=125 and elem<150:
        light_medium+=1
    elif elem>=150 and elem<175:
        medium+=1
    elif elem>=175 and elem<200:
        medium_heavy+=1
    elif elem>200:
        heavy+=1

weights = [light,light_medium, medium, medium_heavy, heavy]
label = ['under 125', '125-150', '150-175', '175-200', 'over 200']
explode = (.4,.2,0,0,.4)

plt.title('Weight of Professional Soccer Players (lbs)')

plt.pie(weights, labels=label, explode=explode, pctdistance=0.8,autopct='%2f %%')
plt.show()
```

Weight of Professional Soccer Players (lbs)



Boxplot

matplotlib.pyplot.boxplot


```
matplotlib.pyplot.boxplot(x, notch=None, sym=None, vert=None, whis=None,  
positions=None, widths=None, patch_artist=None, bootstrap=None, usermedians=None,  
conf_intervals=None, meanline=None, showmeans=None, showcaps=None, showbox=None,  
showfliers=None, boxprops=None, labels=None, flierprops=None, medianprops=None,  
meanprops=None, capprops=None, whiskerprops=None, manage_ticks=True,  
autorange=False, zorder=None, capwidths=None, *, data=None) \[source\]
```

Draw a box and whisker plot.

The box extends from the first quartile (Q1) to the third quartile (Q3) of the data, with a line at the median. The whiskers extend from the box by 1.5x the inter-quartile range (IQR). Flier points are those past the end of the whiskers. See https://en.wikipedia.org/wiki/Box_plot for reference.

A diagram illustrating the components of a boxplot. The top row labels the key values: Q1-1.5IQR, Q1, median, Q3, and Q3+1.5IQR. Below these labels, a horizontal line represents the boxplot structure. The box is bounded by vertical lines at Q1 and Q3, with a vertical line at the median. Whiskers extend from the box edges to the points Q1-1.5IQR and Q3+1.5IQR. Small circles represent fliers. The IQR is indicated by a double-headed arrow below the box.

```
      Q1-1.5IQR  Q1  median  Q3  Q3+1.5IQR  
o  |-----|  :  |-----|  o o  
  |-----|  :  |-----|  
flier  <----->  fliers  
                IQR
```



```
barcelona = []
for i in range(len(fifa[9])):
    if fifa[9,i]=='FC Barcelona':
        barcelona.append(fifa[7,i])

madrid = []
for i in range(len(fifa[9])):
    if fifa[9,i]=='Real Madrid':
        madrid.append(fifa[7,i])

revs = []
for i in range(len(fifa[9])):
    if fifa[9,i]=='New England Revolution':
        revs.append(fifa[7,i])
```

✓ 0.0s

```
barcelona=np.array(barcelona, dtype='int')
barcelona=barcelona.T
madrid =np.array(madrid, dtype='int')
madrid = madrid.T
revs = np.array(revs, dtype='int')
revs= revs.T
```

✓ 0.0s

```
plt.figure(figsize=(5,8), dpi=100)

plt.style.use('default')
bp = plt.boxplot([barcelona, madrid, revs], labels=['FC Barcelona', 'Real Madrid', 'NE Revolution'],
patch_artist=True, medianprops={'linewidth': 2})

plt.title('Professional Soccer Team Comparison')
plt.ylabel('FIFA Overall Rating')

for box in bp['boxes']:
    # change outline color
    box.set(color='#4286f4', linewidth=2)
    # change fill color
    box.set(facecolor = '#e0e0e0' )

plt.show()
```

