Announcements

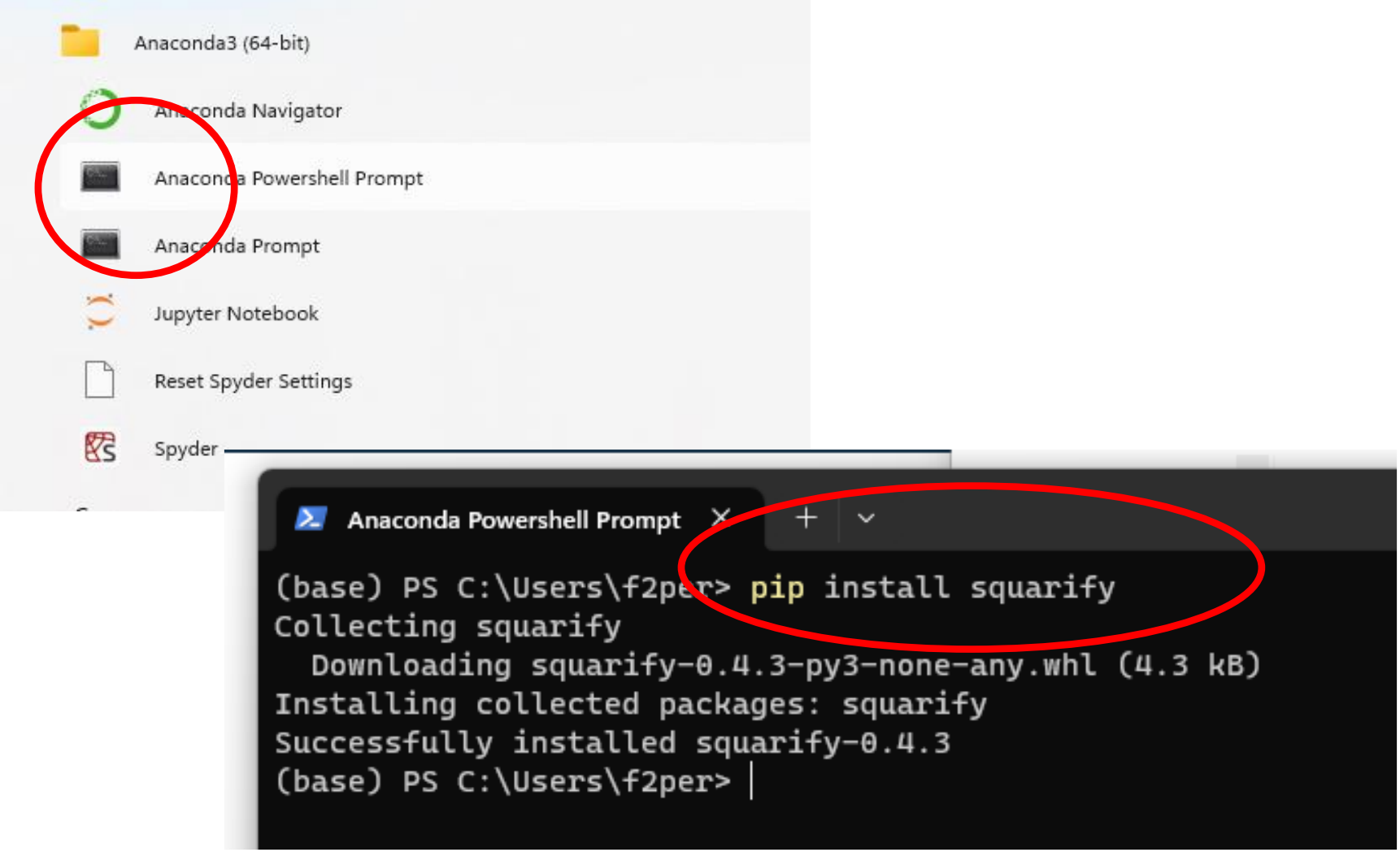
HW#1 Due 9/8 at 11:59 PM
Office Hours: 9/6 @ 12-1 pm Zoom

Agenda Today

Complete Lab2 - Statistical measures & Histogram

Lab3 - EDA

Installing Squarify: Lab3



Review

```
# With cross tabs, we can normalize our table using the normalize argument:

# If passed 'all' or True, will normalize overall values.
# If passed 'index' will normalize over each row.
# If passed 'columns' will normalize over each column.
# If margins is True, will also normalize margin values.

# normalize by total
pd.crosstab(store_sales.p1_promo, store_sales.country, normalize= 'all', margins= True)
```

country	AUS	BRA	CHN	DEU	GBR	JPN	USA	All
p1_promo								
0	0.042788	0.091827	0.093269	0.21875	0.136058	0.180769	0.136058	0.899519
1	0.007212	0.008173	0.006731	0.03125	0.013942	0.019231	0.013942	0.100481
All	0.050000	0.100000	0.100000	0.25000	0.150000	0.200000	0.150000	1.000000

Review

25]:
```
# normalize by column

# Conditional probability (conditioned on country here)
pd.crosstab(store_sales.p1_promo, store_sales.country, normalize= 'columns')

#P(promotion| Japan) = 0.096154
```

25]:

country	AUS	BRA	CHN	DEU	GBR	JPN	USA
p1_promo							
0	0.855769	0.918269	0.932692	0.875	0.907051	0.903846	0.907051
1	0.144231	0.081731	0.067308	0.125	0.092949	0.096154	0.092949

In [47]:
```
# normalize by row
# Conditional probability (conditioned on promo here)
pd.crosstab(store_sales.p1_promo, store_sales.country, normalize= 'index')

# P(USA | promotion) = 0.138756
```

Out[47]:

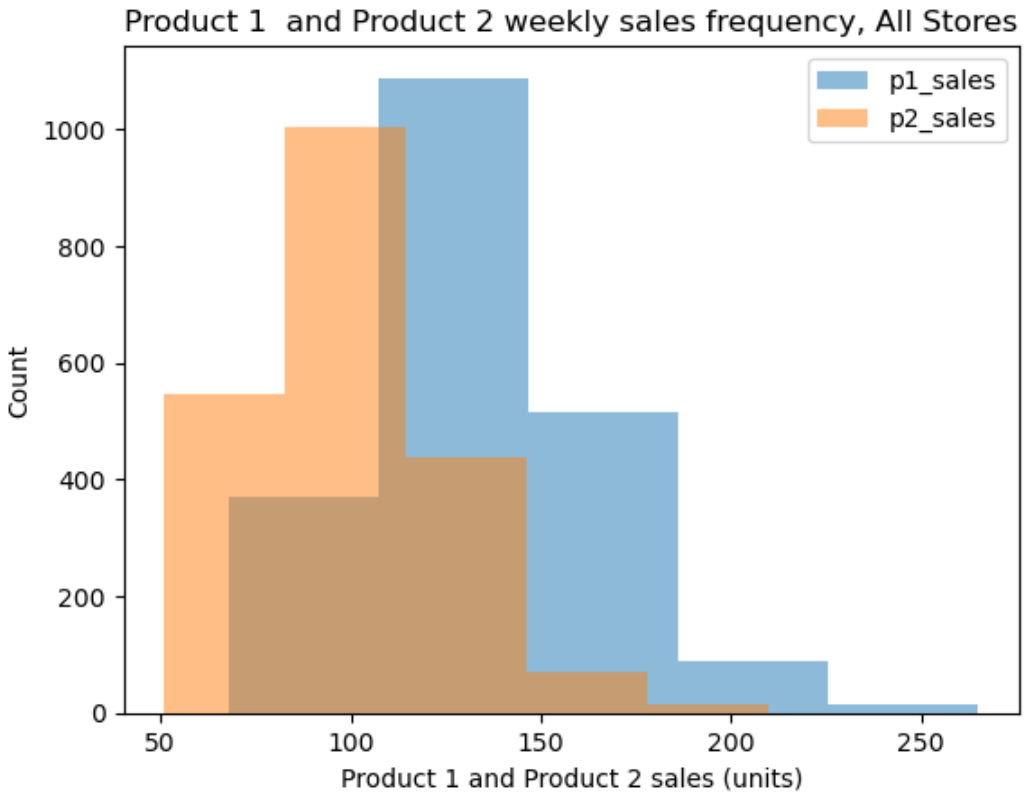
country	AUS	BRA	CHN	DEU	GBR	JPN	USA
p1_promo							
0	0.047568	0.102084	0.103688	0.243185	0.151256	0.200962	0.151256
1	0.071770	0.081340	0.066986	0.311005	0.138756	0.191388	0.138756

Runtime rc settings

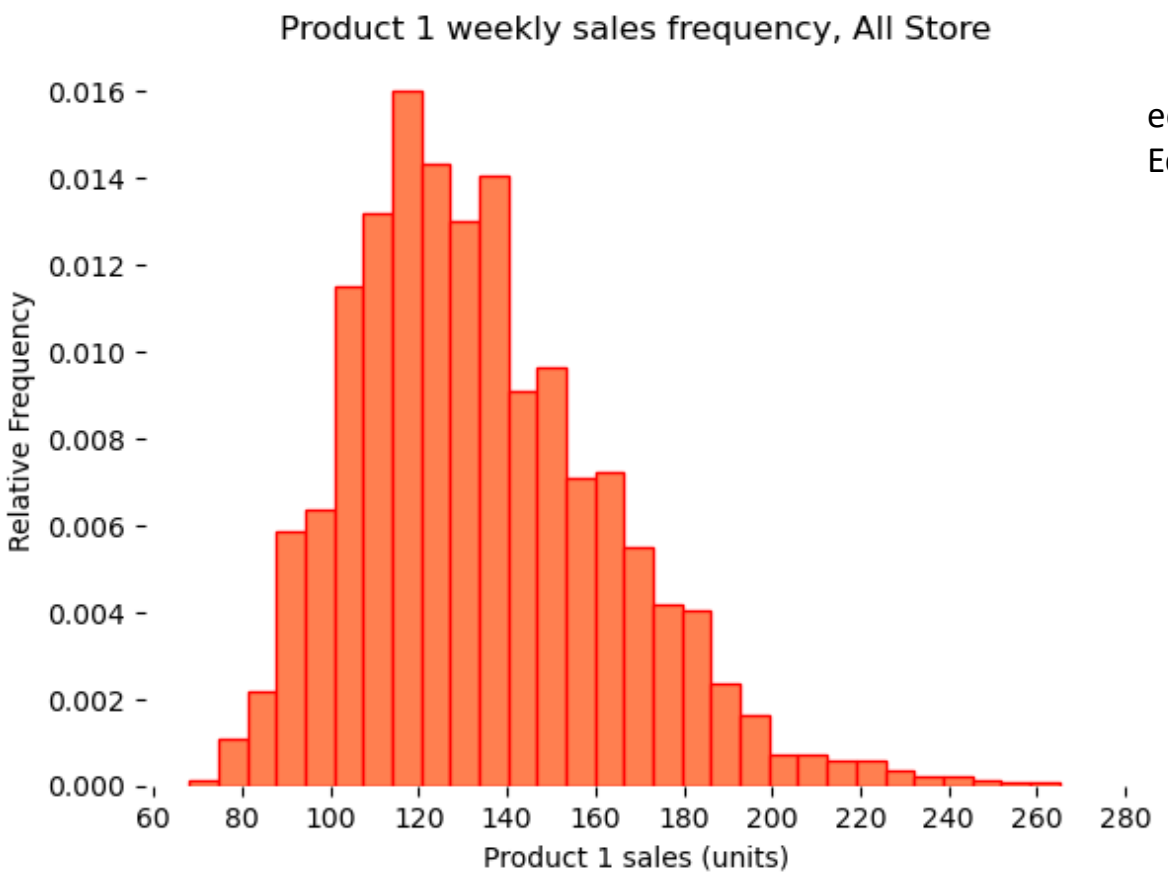
You can dynamically change the default rc (runtime configuration) settings in a python script or interactively from the python shell. All rc settings are stored in a dictionary-like variable called `matplotlib.rcParams`, which is global to the matplotlib package. See `matplotlib.rcParams` for a full list of configurable rcParams. rcParams can be modified directly, for example:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from cycler import cycler
mpl.rcParams['lines.linewidth'] = 2
mpl.rcParams['lines.linestyle'] = '--'
data = np.random.randn(50)
plt.plot(data)
```

In [22]:
```
plt.hist(store_sales['p1_sales'], bins=5, alpha=0.5,label='p1_sales')
plt.hist(store_sales['p2_sales'], bins=5, alpha=0.5,label='p2_sales')
plt.title('Product 1  and Product 2 weekly sales frequency, All Stores')
plt.xlabel('Product 1 and Product 2 sales (units)')
plt.ylabel ('Count')
sales=['p1_sales', 'p2_sales']
plt.legend(sales)
plt.show()
```



```
In [30]: store_sales.p1_sales.hist(bins=30,edgecolor='r',facecolor='coral',density=True)
         plt.title('Product 1 weekly sales frequency, All Store')
         plt.xlabel('Product 1 sales (units)')
         plt.ylabel ('Relative Frequency')
         plt.xticks(range(60,300,20))
         plt.grid(False)
         plt.box(False)
```



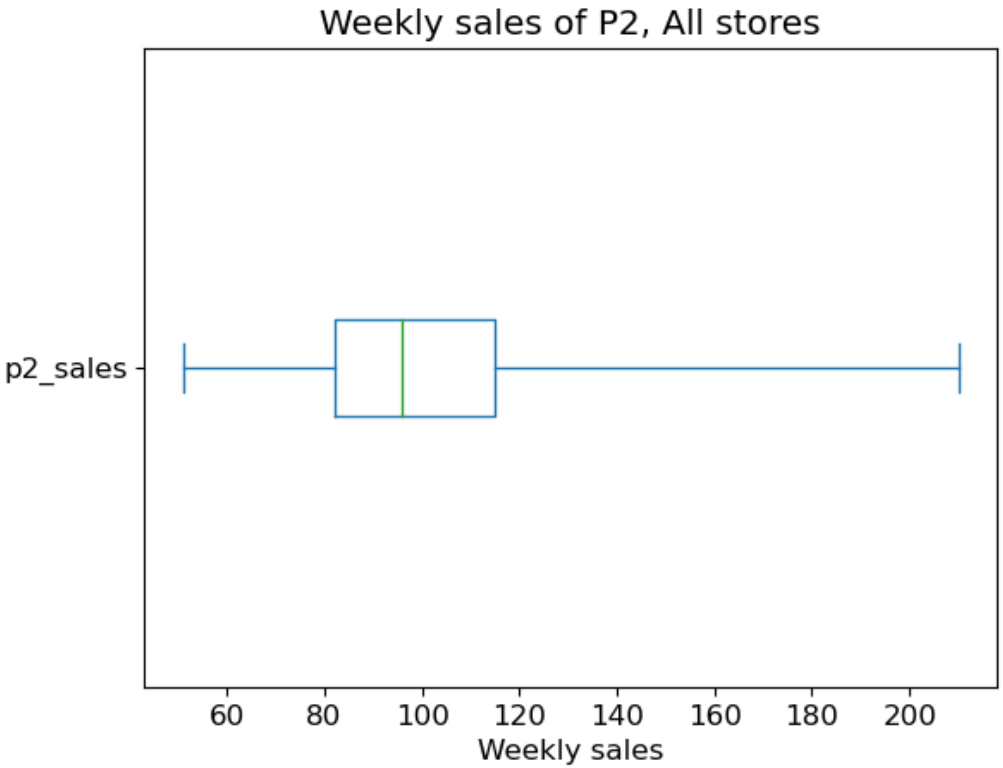
edgecolor = ‘r’ (red)
Edgecolor= ‘g’(green)

Box-Plots

Changing the whiskers using `whis`

- By default, the whiskers extend to 1.5 * the interquartile range
- Make them extend to 2.0 * IQR: `whis=2.0`
- Show the 5th and 95th percentiles: `whis=[5, 95]`
- Show min and max values: `whis=[0, 100]`

In [22]:
```
p = store_sales.p2_sales.plot.box(vert=False, whis=[0,100])
plt.title('Weekly sales of P2, All stores')
plt.xlabel('Weekly sales')
p.set_facecolor('w')
```

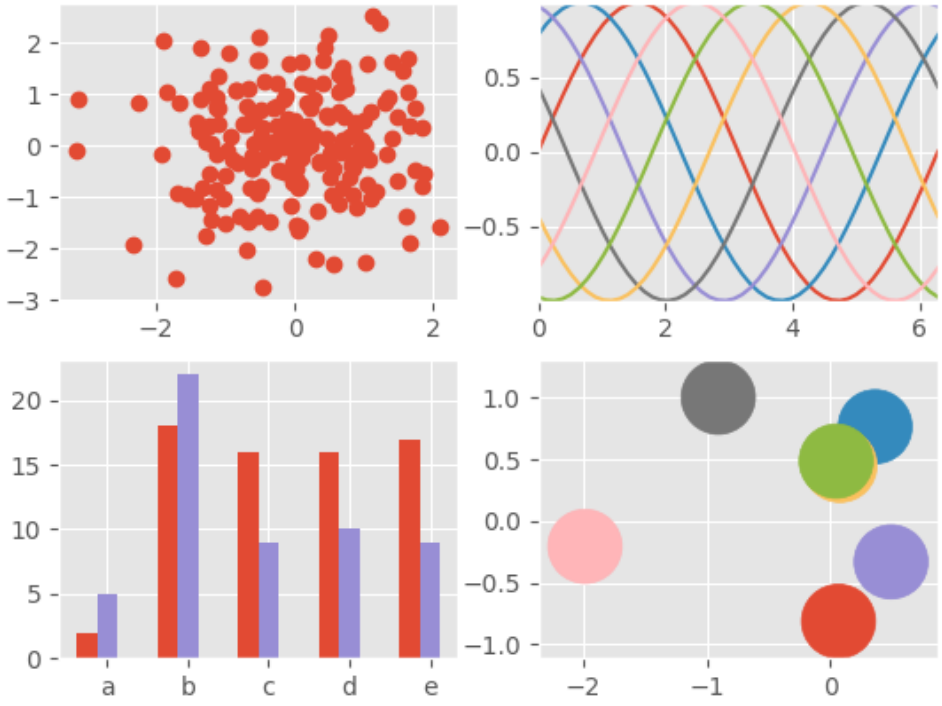


ggplot style sheet

This example demonstrates the "ggplot" style, which adjusts the style to emulate ggplot (a popular plotting package for R).

These settings were shamelessly stolen from [1] (with permission).

[1] https://everyhue.me/posts/sane-color-scheme-for-matplotlib/



Announcements

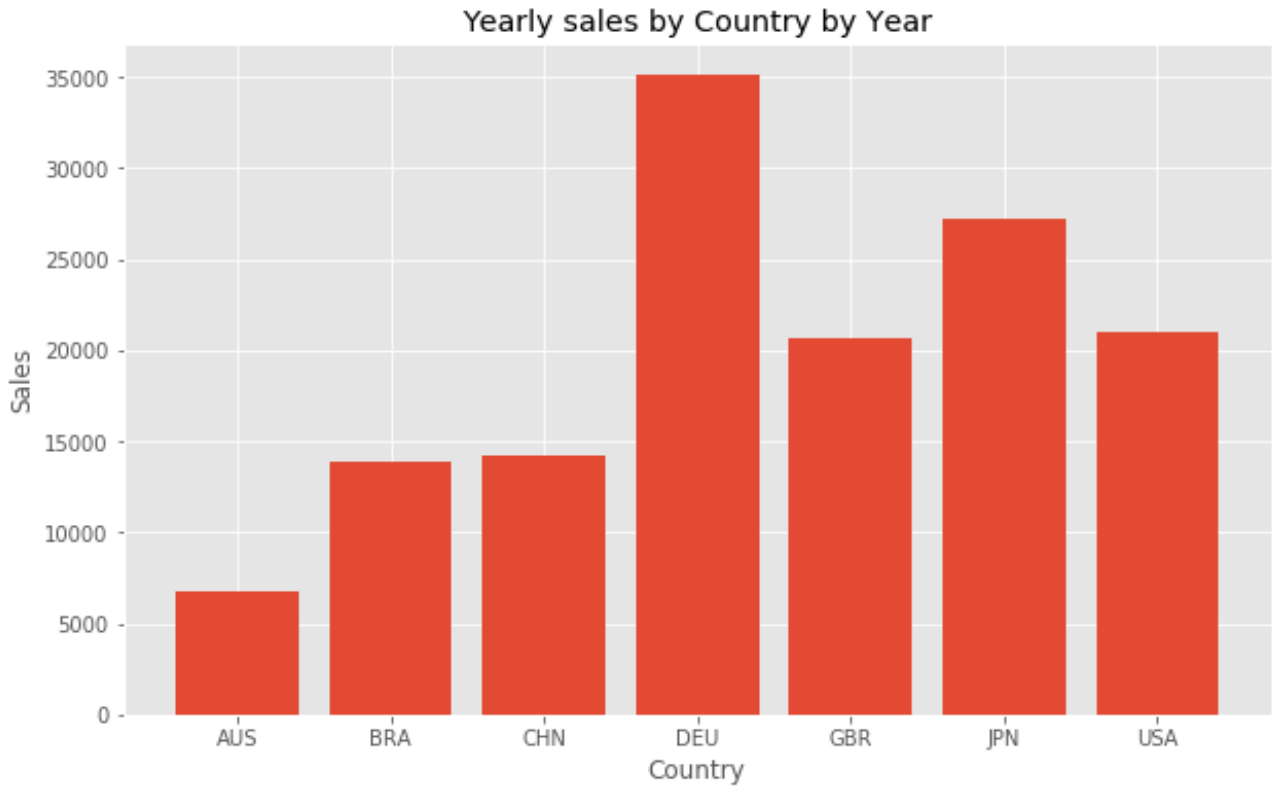
HW#1 Due 9/8 at 11:59 PM
HW#2 Online – Due 9/19 @ 11:59 PM

Agenda Today

Lab3 - EDA

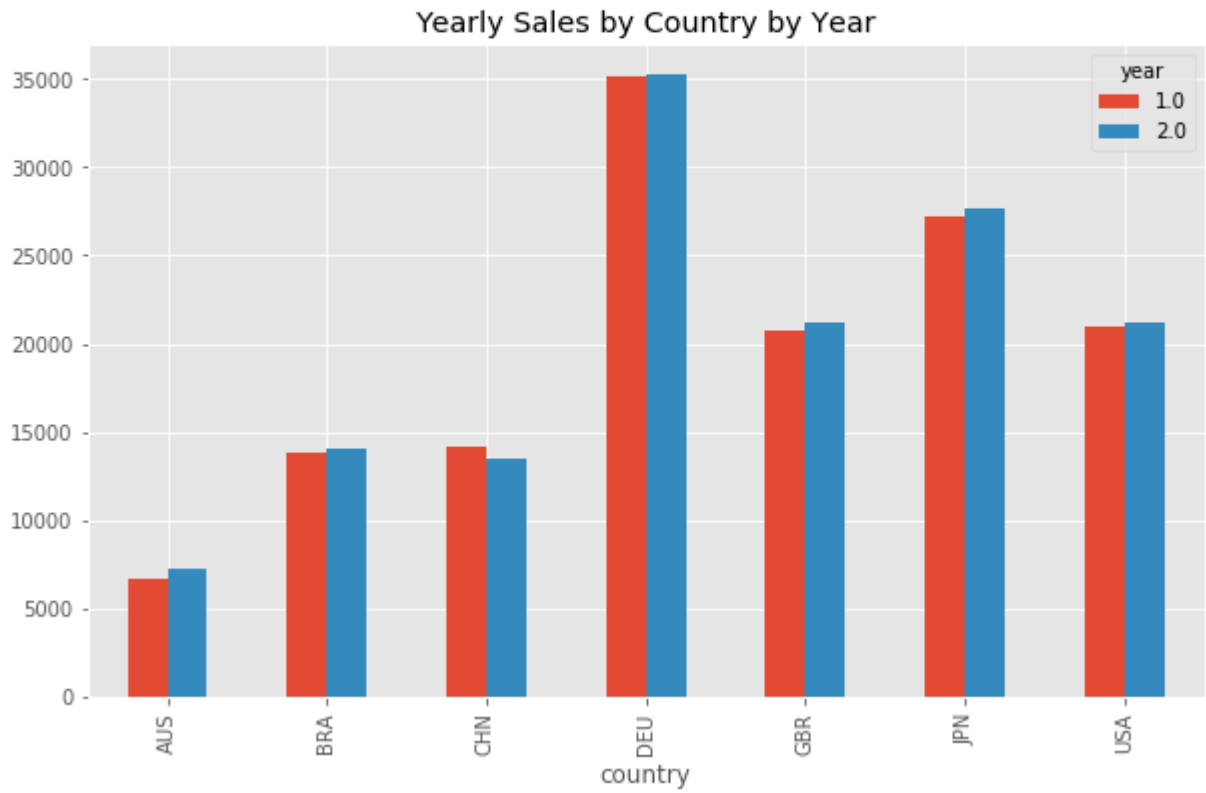
In [7]:
```
plt.bar(yearly_sales.index,yearly_sales[1.0])
plt.xlabel('Country')
plt.ylabel('Sales')
plt.title('Yearly sales by Country by Year')
```

Out[7]:
```
Text(0.5, 1.0, 'Yearly sales by Country by Year')
```

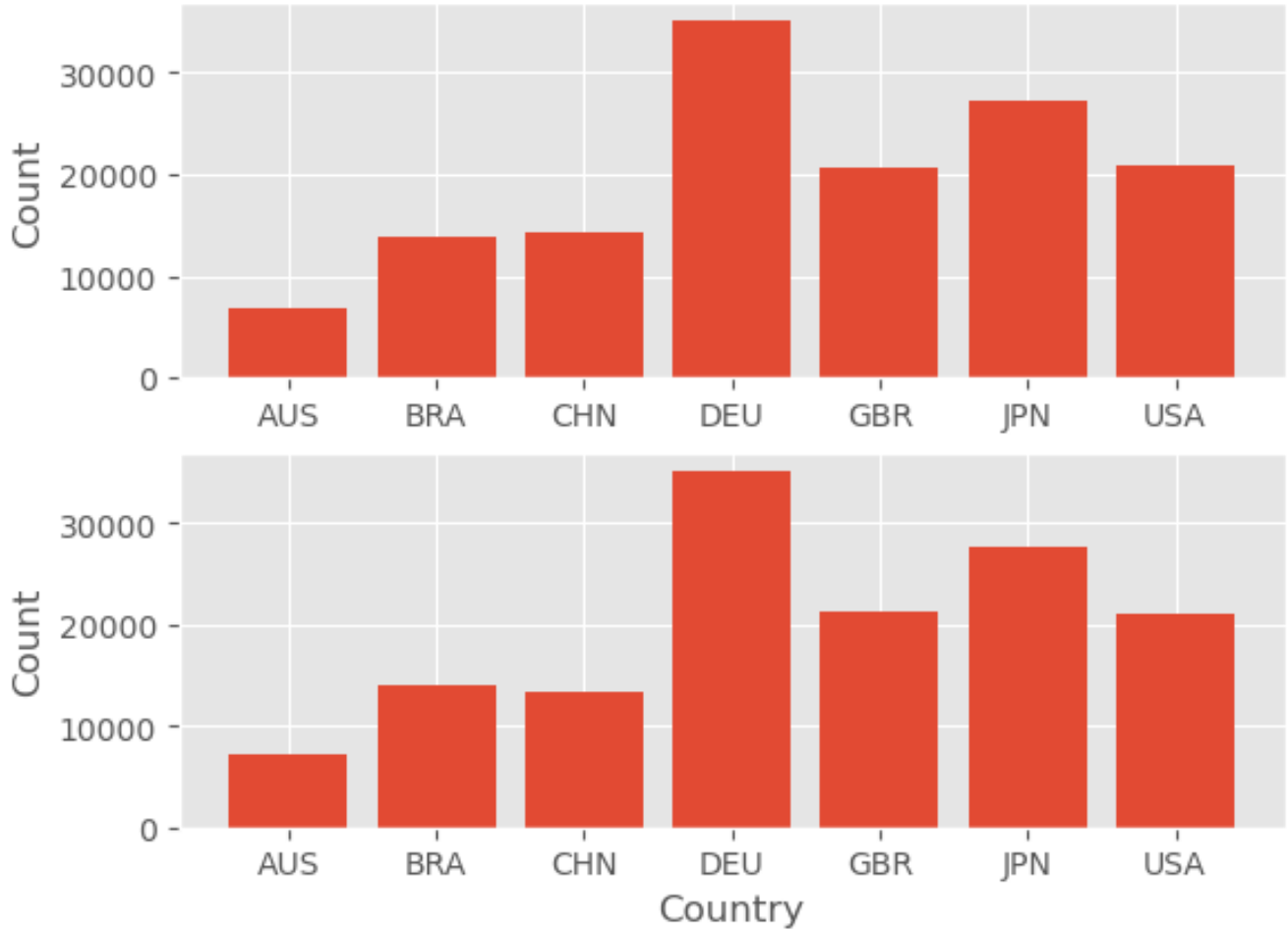


In [8]:
```
#another example
yearly_sales.plot.bar()
plt.title('Yearly Sales by Country by Year')
```

Out[8]:
```
Text(0.5, 1.0, 'Yearly Sales by Country by Year')
```



```
fig, ax=plt.subplots(2,1)
ax[0].bar(yearly_sales.index,yearly_sales[1.0])
ax[1].bar(yearly_sales.index,yearly_sales[2.0])
ax[0].set_ylabel('Count')
ax[1].set_ylabel('Count')
ax[1].set_xlabel("Country")
plt.show()
```

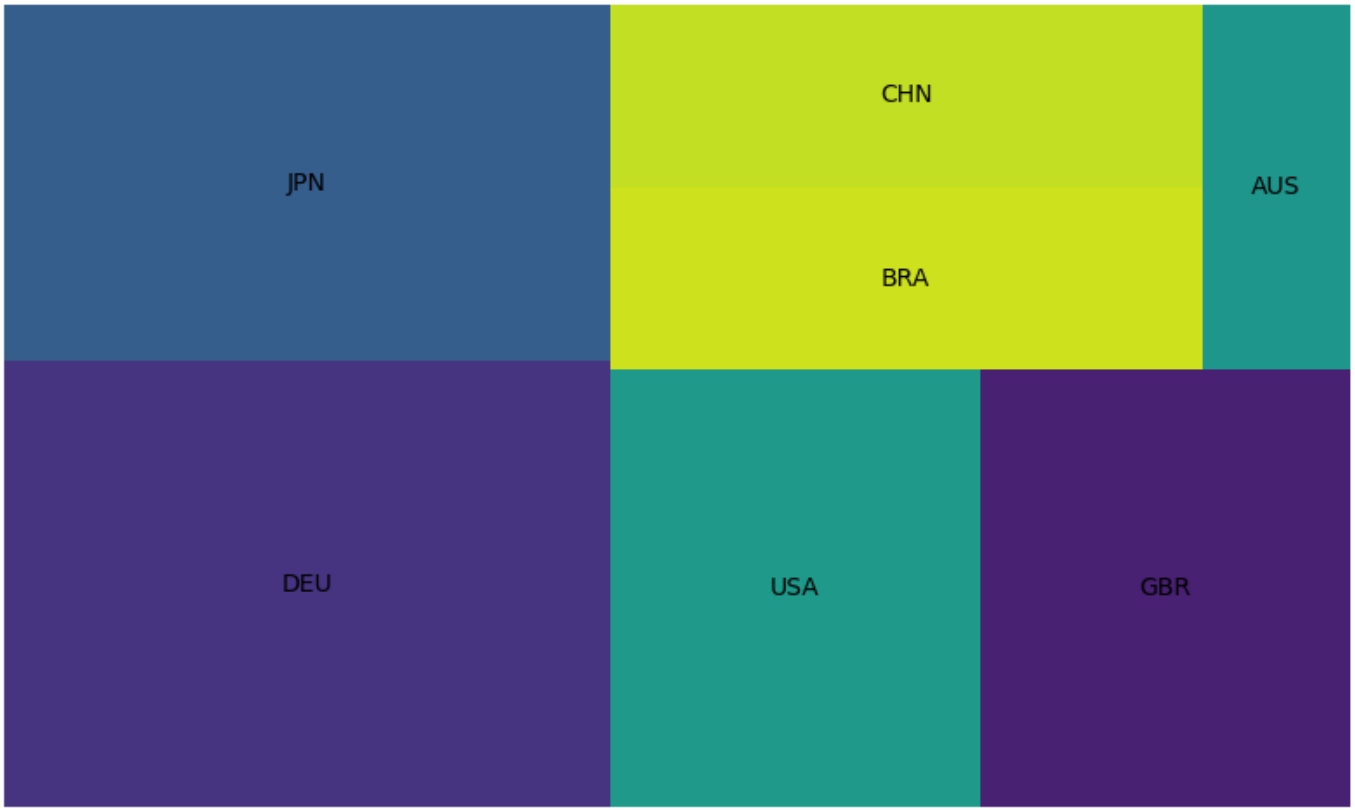


Tree-Map

Default colors

In [16]:
```
import squarify
import matplotlib
squarify.plot(sizes = country_sales, label=country_sales.index)

plt.axis('off')
plt.show()
```



Tree-Map

Squarify uses by default random colors from the viridis color palette, so each time you call the plot you will get a different result. If you want to set custom colors you can make use of the `color` argument and input an array of colors. Note that you can control the transparency of the colors with `alpha`.

```
import matplotlib.pyplot as plt
import squarify

# Sample data
values = [250, 120, 280, 320, 140, 95]
colors = ['#91DCEA', '#64CDCC', '#5FBB68',
          '#F9D23C', '#F9A729', '#FD6F30']

squarify.plot(sizes = values,
              color = colors, alpha = 0.7)

# Remove the axis:
plt.axis("off")

# plt.show()
```

https://python-charts.com/part-whole/treemap-matplotlib/

You can also use a **predefined color palette** instead of specifying an array of colors. In the example below we are using the `'magma'` color palette from `seaborn`.

```
import matplotlib.pyplot as plt
import squarify
import seaborn as sb

# Sample data
values = [250, 120, 280, 320, 140, 95]

# Treemap
squarify.plot(sizes = values,
              color = sb.color_palette("magma",
                                       len(values)), alpha =

# Remove the axis:
plt.axis("off")
```

Python Color Codes

The colors are shown in the table below and the full code is shown below that.

Color Name	Hex Value	RGB Value	Sample
aliceblue	#F0F8FF	RGB(240,248,255)	
antiquewhite	#FAEBD7	RGB(250,235,215)	
antiquewhite1	#FFEFDB	RGB(255,239,219)	
antiquewhite2	#EEDFCC	RGB(238,223,204)	
antiquewhite3	#CDC0B0	RGB(205,192,176)	
antiquewhite4	#8B8378	RGB(139,131,120)	
aqua	#00FFFF	RGB(0,255,255)	
aquamarine1	#7FFFD4	RGB(127,255,212)	
aquamarine2	#76EEC6	RGB(118,238,198)	
aquamarine3	#66CDAA	RGB(102,205,170)	
aquamarine4	#458B74	RGB(69,139,116)	
azure1	#F0FFFF	RGB(240,255,255)	
azure2	#E0EEEE	RGB(224,238,238)	
azure3	#C1CDCD	RGB(193,205,205)	
azure4	#838B8B	RGB(131,139,139)	

https://www.webucator.com/article/python-color-constants-module/

Python Color Codes

X11/CSS4	xkcd	
#00FFFF	#13EAC9	aqua
#7FFFD4	#04D8B2	aquamarine
#F0FFFF	#069AF3	azure
#F5F5DC	#E6DAA6	beige
#000000	#000000	**black**
#0000FF	#0343DF	blue
#A52A2A	#653700	brown
#7FFF00	#C1F80A	chartreuse
#D2691E	#3D1C02	chocolate
#FF7F50	#FC5A50	coral
#DC143C	#8C000F	crimson
#00FFFF	#00FFFF	**cyan**
#00008B	#030764	darkblue
#006400	#054907	darkgreen
#FF00FF	#ED0DD9	fuchsia
#FFD700	#DBB40C	gold
#DAA520	#FAC205	goldenrod

X11/CSS4	xkcd	
#008000	#15B01A	green
#808080	#929591	grey
#4B0082	#380282	indigo
#FFFFF0	#FFFFCB	ivory
#F0E68C	#AAA662	khaki
#E6E6FA	#C79FEF	lavender
#ADD8E6	#7BC8F6	lightblue
#90EE90	#76FF7B	lightgreen
#00FF00	#AAFF32	lime
#FF00FF	#C20078	magenta
#800000	#650021	maroon
#000080	#01153E	navy
#808000	#6E750E	olive
#FFA500	#F97306	orange
#FF4500	#FE420F	orangered
#DA70D6	#C875C4	orchid
#FFC0CB	#FF81C0	pink

X11/CSS4	xkcd	
#DDA0DD	#580F41	plum
#800080	#7E1E9C	purple
#FF0000	#E50000	red
#FA8072	#FF796C	salmon
#A0522D	#A9561E	sienna
#C0C0C0	#C5C9C7	silver
#D2B48C	#D1B26F	tan
#008080	#029386	teal
#FF6347	#EF4026	tomato
#40E0D0	#06C2AC	turquoise
#EE82EE	#9A0EEA	violet
#F5DEB3	#FBDD7E	wheat
#FFFFFF	#FFFFFF	**white**
#FFFF00	#FFFF14	yellow
#9ACD32	#BBF90F	yellowgreen

Axis scaling

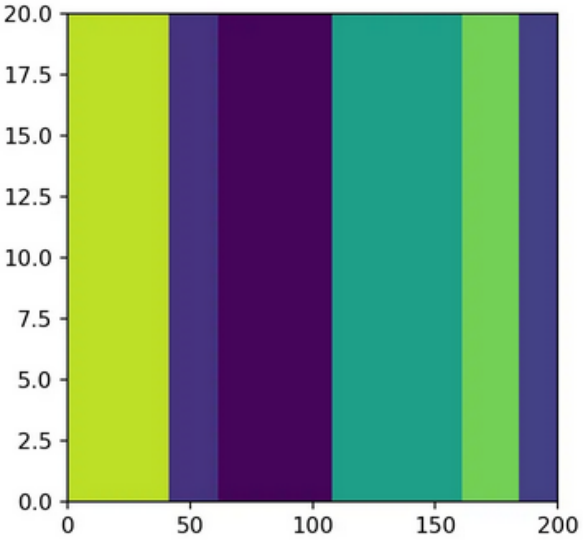
By default, treemaps made with `squarify` are of size 100x100, but with the `norm_x` and `norm_y` arguments you can override the dimensions if needed of the X and Y-axis, respectively.

```
import matplotlib.pyplot as plt
import squarify

# Sample data
values = [250, 120, 280, 320, 140, 95]

# Treemap
squarify.plot(sizes = values,
              norm_x = 200,
              norm_y = 20)

# plt.show()
```



https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html

matplotlib

Plot types Examples Tutorials Reference User guide Develop Releases

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 *align*ment. 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.

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:

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html

matplotlib.pyplot.yticks

`matplotlib.pyplot.yticks(ticks=None, labels=None, *, minor=False, **kwargs)` [source]

Get or set the current tick locations and labels of the y-axis.

Pass no arguments to return the current values without modifying them.

Parameters:

ticks : ***array-like, optional***

The list of ytick locations. Passing an empty list removes all yticks.

labels : ***array-like, optional***

The labels to place at the given *ticks* locations. This argument can only be passed if *ticks* is passed as well.

minor : ***bool, default: False***

If `False`, get/set the major ticks/labels; if `True`, the minor ticks/labels.

****kwargs**

`Text` properties can be used to control the appearance of the labels.

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.yticks.html

matplotlib.pyplot.axhline

`matplotlib.pyplot.axhline(y=0, xmin=0, xmax=1, **kwargs)` [source]

Add a horizontal line across the Axes.

Parameters:

y : *float, default: 0*

y position in data coordinates of the horizontal line.

xmin : *float, default: 0*

Should be between 0 and 1, 0 being the far left of the plot, 1 the far right of the plot.

xmax : *float, default: 1*

Should be between 0 and 1, 0 being the far left of the plot, 1 the far right of the plot.

Returns:

`Line2D`

Other Parameters:

****kwargs**

`color` or c	color
`dash_capstyle`	`CapStyle` or {'butt', 'projecting', 'round'}
`dash_joinstyle`	`JoinStyle` or {'miter', 'round', 'bevel'}
`dashes`	sequence of floats (on/off ink in points) or (None, None)
`data`	(2, N) array or two 1D arrays
`drawstyle` or ds	{'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'}, default: 'default'
`figure`	`Figure`
`fillstyle`	{'full', 'left', 'right', 'bottom', 'top', 'none'}
`gapcolor`	color or None
`gid`	str
`in_layout`	bool
`label`	object
`linestyle` or ls	{'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
`linewidth` or lw	float

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.axhline.html

matplotlib.pyplot.yticks

`matplotlib.pyplot.yticks(ticks=None, labels=None, *, minor=False, **kwargs)` [source]

Get or set the current tick locations and labels of the y-axis.

Pass no arguments to return the current values without modifying them.

Parameters:

ticks : ***array-like, optional***

The list of ytick locations. Passing an empty list removes all yticks.

labels : ***array-like, optional***

The labels to place at the given *ticks* locations. This argument can only be passed if *ticks* is passed as well.

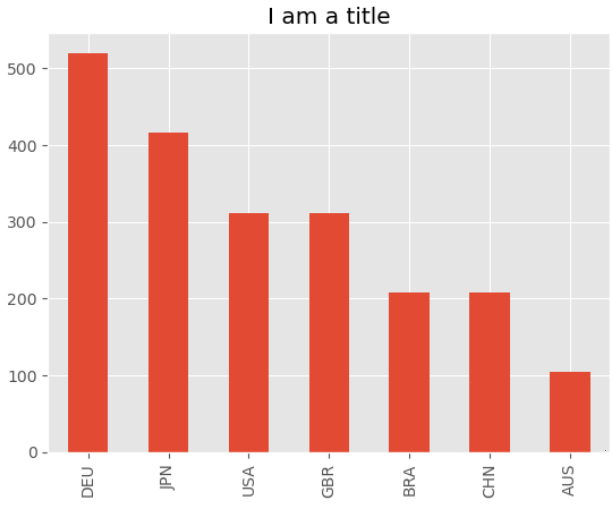
minor : ***bool, default: False***

If `False`, get/set the major ticks/labels; if `True`, the minor ticks/labels.

****kwargs**

`Text` properties can be used to control the appearance of the labels.

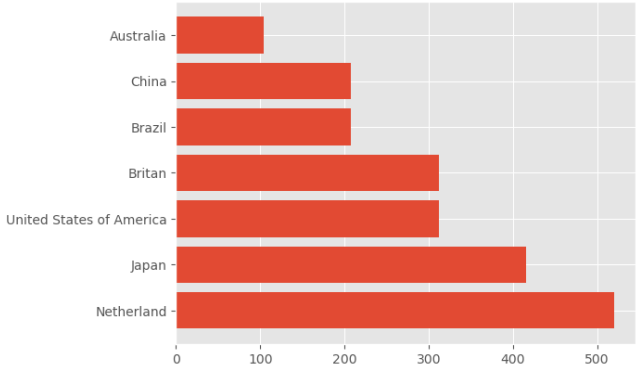
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.yticks.html




```
y_ticklabels = ['Netherland', 'Japan', 'United States of America', 'Britan', 'Brazil',
                'China', 'Australia']

plt.yticks(range(7), y_ticklabels)

plt.show()
```



matplotlib.axes.Axes.text

`Axes.text(x, y, s, fontdict=None, **kwargs)` [source]

Add text to the Axes.

Add the text *s* to the Axes at location *x*, *y* in data coordinates.

Parameters:

x, y : *float*

The position to place the text. By default, this is in data coordinates. The coordinate system can be changed using the *transform* parameter.

s : *str*

The text.

fontdict : *dict, default: None*

matplotlib.pyplot.axhline

`matplotlib.pyplot.axhline(y=0, xmin=0, xmax=1, **kwargs)` [source]

Add a horizontal line across the Axes.

Parameters:

y : *float, default: 0*

y position in data coordinates of the horizontal line.

xmin : *float, default: 0*

Should be between 0 and 1, 0 being the far left of the plot, 1 the far right of the plot.

xmax : *float, default: 1*

Should be between 0 and 1, 0 being the far left of the plot, 1 the far right of the plot.

Returns:

`Line2D`

Other Parameters:

****kwargs**

`color` or c	color
`dash_capstyle`	`CapStyle` or {'butt', 'projecting', 'round'}
`dash_joinstyle`	`JoinStyle` or {'miter', 'round', 'bevel'}
`dashes`	sequence of floats (on/off ink in points) or (None, None)
`data`	(2, N) array or two 1D arrays
`drawstyle` or ds	{'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'}, default: 'default'
`figure`	`Figure`
`fillstyle`	{'full', 'left', 'right', 'bottom', 'top', 'none'}
`gapcolor`	color or None
`gid`	str
`in_layout`	bool
`label`	object
`linestyle` or ls	{'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
`linewidth` or lw	float

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.axhline.html