PyWaffle Documentation

Guangyang Li

Jun 08, 2022

CONTENTS:

1 Installation **3**

2 Examples **5**
- 2.1 Basic Examples and Formats of Values . . . 5
- 2.2 Figure and Axis Manipulation . . . 6
- 2.3 Value Scaling and Auto-sizing . . . 7
- 2.4 Title, Label and Legend . . . 8
- 2.5 Block Colors . . . 9
- 2.6 Plot with Characters or Icons . . . 9
- 2.7 Block Shape, Spacing, Location, Direction and Arranging Style . . . 11
- 2.8 Subplots . . . 13
- 2.9 Adjust Figures . . . 14
- 2.10 Add Other Elements . . . 15

3 Font Awesome Integration **17**

4 API Reference **19**

Python Module Index **23**

Index **25**

PyWaffle is an open source, MIT-licensed Python package for plotting waffle charts.

A Figure constructor class *Waffle* is provided, which could be passed to matplotlib.pyplot.figure and generate a matplotlib Figure object.

Visit PyWaffle on Github and PyPI.

CHAPTER

ONE

INSTALLATION

The last stable release is available on PyPI and can be installed with `pip`:

```
$ pip install pywaffle
```

Requirements

- Python 3.5+
- Matplotlib

CHAPTER

TWO

EXAMPLES

Several examples are shown below by topics, which covers all parameters that PyWaffle has, and their most common use cases. If you are looking for the detailed definition and description of parameters, please check out API Reference.

2.1 Basic Examples and Formats of Values

This is a simple example to plot a 5-row, 10-column waffle chart. The three values are plotted as blocks directly, and the blocks number matches the numbers in `values`, because the sum of `values` equals to total block number (`rows * columns`).

Parameter `values` accept numbers in multiple format, including list, dict, and pandas.DataFrame.

```
import matplotlib.pyplot as plt
from pywaffle import Waffle
```

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    columns=10,  # Either rows or columns could be omitted
    values=[30, 16, 4]  # Pass a list of integers to values
)
fig.savefig("plot.png", bbox_inches="tight")
```

> ***NOTE:*** One of the parameter `rows` and `columns` is redundant in this case, since both of the chart size and value sum are 50. So, either one of `rows` and `columns` could be omitted, and it can still be calculated through value sum automatically. See *Auto-sizing* for more details.

When a dictionary is passed to `values`, the key will be used as labels and shown in legends.

```
plt.figure(
    FigureClass=Waffle,
    rows=5,
    columns=10,
    values={'Cat1': 30, 'Cat2': 16, 'Cat3': 4},
    legend={'loc': 'upper left', 'bbox_to_anchor': (1, 1)}
)
```

However, unlike values in a dictionary that can generate labels and legend automatically, when the `values` is a DataFrame, `Waffle` does not use the row index of the DataFrame as label by default. So you have to pass the index to parameter `labels` manually, if you would like to use column index as label.

```
import pandas as pd

data = [30, 16, 4]
df = pd.DataFrame(data, columns=['Value'], index=['Cat1', 'Cat2', 'Cat3'])


plt.figure(
    FigureClass=Waffle,
    rows=5,
    columns=10,
    values=df['Value'],
    labels=list(df.index),  # Legend would not be created without this line
    legend={'loc': 'upper left', 'bbox_to_anchor': (1, 1)}
)
```

2.2 Figure and Axis Manipulation

2.2.1 Update plot with returned Figure and Axes

You are free to modify the figure generated by `Waffle` class. You may get all axes in the figure through property Figure.axes, and modify them through the list.

For example, if you would like to remove the auto-generated legend, and save the figure into a file, you can update the axis through `axes` property and then call `savefig()` with the returned figure:

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    columns=10,
    values={'Cat1': 30, 'Cat2': 16, 'Cat3': 4},
)

# Remove legend
fig.axes[0].get_legend().remove()

# Save the figure into a file
fig.savefig("my_plot.png")
```

2.2.2 Make Waffle on Existed Axis

Sometimes you might have figure and axis created already, and you would like to plot waffle chart on top of it, without creating a new figure. In this case, you may use `Waffle.make_waffle()` function, which accepts axis `ax` and other arguments available in `Waffle`.

> ***NOTE:*** `Waffle.make_waffle()` is a classmethod, and you should call it without creating a new `Waffle` instance.
>
> ***NOTE:*** The only argument that is available in Waffle but unsupported in `Waffle.make_waffle()` is `plots`, since it only accept one axis. Thus, this function can only generate waffle chart in a single plot.

Below is an example showing that you may create and modify the figure and axis first, and then pass the axis to `Waffle.make_waffle()` for waffle chart plotting.

```
fig = plt.figure()
ax = fig.add_subplot(111)

# Modify existed axis
ax.set_title("Axis Title")
ax.set_aspect(aspect="equal")

Waffle.make_waffle(
    ax=ax,  # pass axis to make_waffle
    rows=5,
    columns=10,
    values=[30, 16, 4],
    title={"label": "Waffle Title", "loc": "left"}
)
```

2.3 Value Scaling and Auto-sizing

2.3.1 Value Scaling

It is a common case that the chart size does not equal to the total number of values. Then the values have to be scaled to fit the chart size.

Change argument `rounding_rule` to set a preferred rounding rule when scaling. It accepts `floor` or `ceil` or `nearest`.

> ***NOTE:*** When `rounding_rule` is `ceil` or `nearest`, sum of scaled values might be greater than chart size. If so, the blocks of last category would not be shown completely in the chart. Therefore, although `nearest` is the default rounding rule, `floor` is actually the most consistent rule as it avoids the block overflowing.

In the following example, values are scaled to 24, 23, 1 as block numbers with `rounding_rule=floor`.

```
plt.figure(
    FigureClass=Waffle,
    rows=5,
    columns=10,
    values=[48, 46, 3],
    rounding_rule='floor'
)
```

2.3.2 Auto-sizing

If you would like to avoid values scaling, pass an integer to either one of `rows` or `columns` parameter only. Then the absolute number of `values` would be used as block number directly and the other parameter would be calculated automatically.

In the following example, we set `rows=5`, `values=[48, 46, 3]` and leave `columns` empty. Then the block numbers would be the same to values. Since the sum of values is 97, the column number has to be 20 to fit all blocks.

```
plt.figure(
    FigureClass=Waffle,
    rows=5,
    values=[48, 46, 3]
)
```

2.4 Title, Label and Legend

Parameter `title` accepts parameters of matplotlib.pyplot.title in a dict.

Parameter `labels` accepts string labels in a list. If it is not specified, key of `values` would be used as labels.

Parameter `legend` accepts parameters of matplotlib.pyplot.legend in a dict.

> ***NOTE:*** Labels could also be specified in parameter `legend` under key `labels` instead.

```
data = {'Cat1': 30, 'Cat2': 16, 'Cat3': 4}
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    columns=10,
    values=data,
    title={
        'label': 'Example plot',
        'loc': 'left',
        'fontdict': {
            'fontsize': 20
        }
    },
    labels=[f"{k} ({int(v / sum(data.values()) * 100)}%)" for k, v in data.items()],
    legend={
        # 'labels': [f"{k} ({v}%)" for k, v in data.items()],  # lebels could also be␣
↪under legend instead
        'loc': 'lower left',
        'bbox_to_anchor': (0, -0.4),
        'ncol': len(data),
        'framealpha': 0,
        'fontsize': 12
    }
)
```

2.5 Block Colors

Parameter `colors` accepts colors in a list or tuple. The length must be same to `values`, and the acceptable color format includes case-insensitive hex RGB or RGBA, RGB or RGBA tuple, single character notation, case-insensitive X11/CSS4 color name, and more, as long as Matplotlib can recognize. See Matplotlib Colors for the full list.

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    columns=10,
    values=[30, 16, 4],
    colors=["#232066", "#983D3D", "#DCB732"]
)
```

Another method to change block colors is passing Colormap to parameter `cmap_name`, which sets colors in batch.

> ***NOTE:*** Sequential colormaps does not work with PyWaffle. Only Qualitative colormaps are supported, including `Pastel1`, `Pastel2`, `Paired`, `Accent`, `Dark2`, `Set1`, `Set2`, `Set3`, `tab10`, `tab20`, `tab20b`, `tab20c`. See the list and color examples in Colormaps in Matplotlib.

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    columns=10,
    values=[30, 16, 4],
    cmap_name="tab10"
)
```

2.6 Plot with Characters or Icons

2.6.1 Characters

Blocks could be Unicode characters instead of rectangles by passing the desired character to `characters`.

Categories could have different character for each category, by passing a list or tuple of characters to parameter `characters` . The length must be the same as `values`.

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    values=[30, 16, 4],
    colors=["#4C8CB5", "#B7CBD7", "#C0C0C0"],
    characters='',
    font_size=24
)
```

To specify the font, pass the absolute path to a .ttf or .otf file to parameter `font_file`.

2.6.2 Icons

Waffle Chart with icons is also known as Pictogram Chart.

PyWaffle supports plotting with icons through Font Awesome. See page Font Awesome Integration for how Font Awesome is integrated into PyWaffle.

For searching available icon name in Font Awesome, please visit https://fontawesome.com/search.

Icon Size

When using icons, the parameters for setting block size would be ignored, including `interval_ratio_x`, `interval_ratio_y` and `block_aspect_ratio`. Instead, use `font_size` to set the size of icons. For allowed sizes, see FontProperties.set_size.

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    values=[30, 16, 4],
    colors=["#232066", "#983D3D", "#DCB732"],
    icons='star',
    font_size=24
)
```

Icons in Legend

Each category could have a different icon, by passing a list or tuple of icon names to parameter `icons`. The length must be the same as `values`.

In Font Awesome Icons, there are different icon sets in different styles, including Solid, Regular and Brands. It can be specified through parameter `icon_style`. By default it searches icon from `solid` style.

With `icon_legend=True`, the symbol in legend would be the icon. Otherwise, it would be a color bar.

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    values=[30, 16, 4],
    colors=["#FFA500", "#4384FF", "#C0C0C0"],
    icons=['sun', 'cloud-showers-heavy', 'snowflake'],
    font_size=20,
    icon_style='solid',
    icon_legend=True,
    legend={
        'labels': ['Sun', 'Shower', 'Snow'],
        'loc': 'upper left',
        'bbox_to_anchor': (1, 1)
    }
)
```

Specify Icon Style for Each Category

Font Awesome locates icons by both of icon name and style. Thus, icon style might not be the same for all icons, and you have to specify icon style for all icons separately. Therefore, `icon_style` also accepts a list or a tuple of styles in string.

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    values=[30, 16, 4],
    colors=["#FFA500", "#4384FF", "#C0C0C0"],
    icons=['sun', 'cloud-showers-heavy', 'font-awesome'],
    icon_size=20,
    icon_style=['regular', 'solid', 'brands'],
    icon_legend=False,
    legend={
        'labels': ['Sun', 'Shower', 'Flag'],
        'loc': 'upper left',
        'bbox_to_anchor': (1, 1)
    }
)
```

2.7 Block Shape, Spacing, Location, Direction and Arranging Style

PyWaffle provides several parameters to control how and where to plot blocks.

2.7.1 Block Shape

Parameter `block_aspect_ratio` controls the shape of blocks by change the ratio of block's width to block's height. By default it is 1, so that the blocks are squares.

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    values=[30, 16, 4],
    block_aspect_ratio=1.618
)
```

2.7.2 Block Spacing

Parameter `interval_ratio_x` and `interval_ratio_y` control the horizontal and vertical distance between blocks. `interval_ratio_x` is the ratio of horizontal distance between blocks to block's width and `interval_ratio_y` is the ratio of vertical distance between blocks to block's height.

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    values=[30, 16, 4],
```

(continues on next page)

(continued from previous page)

```
    interval_ratio_x=1,
    interval_ratio_y=0.5
)
```

2.7.3 Where to Start First Block

Use parameter `starting_location` to set the location of starting block. It accepts locations in string like `NW`, `SW`, `NE` and `SE` representing four corners. By default, it is `SW`, meaning PyWaffle starts drawing blocks from lower-left corner.

Here is an example that start plotting from lower-right corner (`SE`).

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    values=[30, 16, 4],
    starting_location='SE'
)
```

2.7.4 Plotting Direction

By default, PyWaffle draws blocks column by column, thus categories are plotted horizontally. To make it vertical, set parameter `vertical` to `True`.

In the following example, it draws blocks from lower-left corner to lower-right corner row by row up to the top.

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    values=[30, 16, 4],
    vertical=True
)
```

2.7.5 Where to Start Each Category

Use parameter `block_arranging_style` to set how to arrange blocks for each category. By default it is `'normal'`, which draws block of new category from where last category ends.

When it is `snake`, it draws with snake pattern, starting a new line from an opposite direction every time. This style is useful if you would like to keep blocks of each category together.

In the below example, since the default starting location is lower-left and default direction is not vertical, it draws blocks from lower-left to top-left for the first line, and then from the top block of second column down to the bottom one of this column, and then go on with snake-like pattern.

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    values=[12, 22, 20, 4],
    block_arranging_style='snake'
)
```

When it is `new-line`, it forces every category starting from a new line. Note that it only works when only `columns` is passed and `vertical=True`, or `rows` is passes and `vertical=False`. It will be ignored if both of `columns` and `rows` are passed.

```
fig = plt.figure(
    FigureClass=Waffle,
    columns=10,
    values=[30, 16, 4],
    block_arranging_style='new-line',
    vertical=True
)
```

2.8 Subplots

Sometimes we would like to show multiple waffle charts in one figure. This is supported through adding subplots. It also helps avoiding duplicated legends, titles and other components.

Let's say we have sample data as shown below:

```
import pandas as pd
data = pd.DataFrame(
    {
        'labels': ['Car', 'Truck', 'Motorcycle'],
        'Factory A': [32384, 13354, 5245],
        'Factory B': [22147, 6678, 2156],
        'Factory C': [8932, 3879, 896],
    },
).set_index('labels')

# A glance of the data:
#             Factory A  Factory B  Factory C
# labels
# Car             27384      22147       8932
# Truck            7354       6678       3879
# Motorcycle       3245       2156       1196
```

To convert the Vehicle Production data above into reasonable block numbers, we can simply pass values like `data['Factory A'] / 1000`, and PyWaffle will handle the rounding. Note that parameter `values` also accepts column data in `pandas.Series`. However, unlike values in dict, pandas.Series value does not support auto labeling yet.

To plot multiple subplots in one figure, merge the parameters for each plot to parameter `plots` as dict values. The keys are integers describing the position of the subplot. It accepts tuple, int and string. If position is tuple, the format should be like `(nrows, ncols, index)`; if it is int or string, it should be a 3-digit integer like `312`, standing for nrows, ncols, and index in order. Note that all integers must be less than 10 for the later form to work. See arguments of matplotlib.pyplot.subplot for more detail.

NOTE: Parameters which are passed outside of `plots` would be applied to all subplots, if they are not specified in `plots`. Otherwise, settings in `plots` have higher priority.

```
fig = plt.figure(
    FigureClass=Waffle,
    plots={
        311: {
            'values': data['Factory A'] / 1000,  # Convert actual number to a reasonable␣
↪block number
            'labels': [f"{k} ({v})" for k, v in data['Factory A'].items()],
            'legend': {'loc': 'upper left', 'bbox_to_anchor': (1.05, 1), 'fontsize': 8},
            'title': {'label': 'Vehicle Production of Factory A', 'loc': 'left',
↪'fontsize': 12}
        },
        312: {
            'values': data['Factory B'] / 1000,
            'labels': [f"{k} ({v})" for k, v in data['Factory B'].items()],
            'legend': {'loc': 'upper left', 'bbox_to_anchor': (1.2, 1), 'fontsize': 8},
            'title': {'label': 'Vehicle Production of Factory B', 'loc': 'left',
↪'fontsize': 12}
        },
        313: {
            'values': data['Factory C'] / 1000,
            'labels': [f"{k} ({v})" for k, v in data['Factory C'].items()],
            'legend': {'loc': 'upper left', 'bbox_to_anchor': (1.3, 1), 'fontsize': 8},
            'title': {'label': 'Vehicle Production of Factory C', 'loc': 'left',
↪'fontsize': 12}
        },
    },
    rows=5,  # Outside parameter applied to all subplots, same as below
    cmap_name="Accent",  # Change color with cmap
    rounding_rule='ceil',  # Change rounding rule, so value less than 1000 will still␣
↪have at least 1 block
    figsize=(5, 5)
)

fig.suptitle('Vehicle Production by Vehicle Type', fontsize=14, fontweight='bold')
fig.supxlabel('1 block = 1000 vehicles', fontsize=8, ha='right')
```

2.9 Adjust Figures

2.9.1 Figure Size, Background Color, DPI, etc.

Built-in parameters of matplotlib.pyplot.figure could be passed alone with `Waffle` as `FigureClass` and PyWaffle parameters. Some of them are commonly used to adjust figure, like `figsize`, `dpi`, `facecolor`, `tight_layout`, etc. Check matplotlib.pyplot.figure for the details.

In the following example, we change the background color of figure by passing color value to `facecolor`.

```
fig = plt.figure(
    FigureClass=Waffle,
```

(continues on next page)

(continued from previous page)

```
    rows=5,
    values=[30, 16, 4],
    colors=["#232066", "#983D3D", "#DCB732"],
    facecolor='#DDDDDD'  # facecolor is a parameter of matplotlib.pyplot.figure
)
```

2.9.2 Plot Location

Use parameter `plot_anchor` to change the location of plot in the figure.

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    values=[30, 16, 4],
    plot_anchor='S'
)
fig.set_facecolor('#DDDDDD')
```

Parameter `tight` controls whether `tight_layout` in matplotlib is called when drawing.

2.9.3 Tight Layout

By default, PyWaffle sets the figure with tight layout. Thus, when showing the plot, the following warning might pop up:

```
UserWarning: This figure includes Axes that are not compatible with tight_layout, so its␣
↪results might be incorrect
```

It is usually not an issue when saving the plot to a file, so it could ignored. If you still want to remove the moving, you can either suppress warnings through Python's `warnings` module, or set PyWaffle parameter `tight` to False.

2.10 Add Other Elements

Although PyWaffle provides parameters like `title` and `legend` to add elements, it is only for plotting waffle charts after all. To add other elements and make a fancy plot, you can update the figure directly or use the methods provided by Figure class.

In the following example, we use `text()` method to add a custom watermark to the figure.

```
fig = plt.figure(
    FigureClass=Waffle,
    rows=5,
    values=[30, 16, 4]
)
fig.text(
    x=0.5,
```

(continues on next page)

(continued from previous page)

```
    y=0.5,
    s="Sample",
    ha="center",
    va="center",
    rotation=30,
    fontsize=40,
    color='gray',
    alpha=0.3,
    bbox={
        'boxstyle': 'square',
        'lw': 3,
        'ec': 'gray',
        'fc': (0.9, 0.9, 0.9, 0.5),
        'alpha': 0.3
    }
)
```

You may also use method like `add_artist` to add custom objects, which is not a method of PyWaffle, but its parent class `matplotlib.figure.Figure`. See matplotlib.figure.Figure for all available methods.

CHAPTER

THREE

FONT AWESOME INTEGRATION

PyWaffle installs Font Awesome free version automatically as a dependent package. The package it is trying to install is the latest version of fontawesomefree.

If you would like to upgrade or downgrade `fontawesomefree`, and use the specific version within PyWaffle, you can upgrade `fontawesomefree` and then reinstall `pywaffle`. In commands, that is:

```
# Either upgrade to the latest, or specify a version number
pip install --upgrade fontawesomefree
# OR
pip install fontawesomefree==6.1.1

# Then reinstall pywaffle
pip install --force-reinstall --no-deps pywaffle
```

Option `--force-reinstal` ensures icon mapping file in the package would be regenerated, and option `--no-deps` avoid package `fontawesomefree` being upgraded to unexpected version.

To validate the upgrade, you may check the version number at the first line of the icon mapping file. The file path is `<PYTHON_LIB>/pywaffle/fontawesome_mapping.py`. The line should be something like:

```
# For Font Awesome version: 6.1.1
```

For how to use Font Awesome with PyWaffle, please visit Plot with Characters or Icons.

CHAPTER

FOUR

API REFERENCE

class pywaffle.waffle.**Waffle**(**args*, ***kwargs*)

A custom Figure class to make waffle charts.

Parameters

- **values** (*list|dict|pandas.Series*) – Numerical value of each category. If it is a dict, the keys would be used as labels.
- **rows** (*int*) – The number of lines of the waffle chart.
- **columns** (*int*) – The number of columns of the waffle chart.

  At least one of rows and columns is required.
  If either rows or columns is passed, the other parameter would be calculated automatically through the absolute value of values.
  If both of rows and columns are passed, the block number is fixed and block numbers are calculated from scaled values.

- **colors** (*list[str]|tuple[str], optional*) – A list of colors for each category. Its length should be the same as values.

  Default values are from Set2 colormap.

- **labels** (*list[str]|tuple[str], optional*) – The name of each category.

  If the values is a dict, this parameter would be replaced by the keys of values.

- **legend** (*dict, optional*) – Parameters of matplotlib.pyplot.legend in a dict.

  E.g. {'loc': '', 'bbox_to_anchor': (,), ... }
  See full parameter list in https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html

- **interval_ratio_x** (*float, optional*) – Ratio of horizontal distance between blocks to block's width. [Default 0.2]

- **interval_ratio_y** (*float, optional*) – Ratio of vertical distance between blocks to block's height. [Default 0.2]
- **block_aspect_ratio** (*float, optional*) – The ratio of block's width to height. [Default 1]
- **cmap_name** (*str, optional*) – Name of colormaps for default color, if colors is not assigned.

  See full list in https://matplotlib.org/examples/color/colormaps_reference.html
  [Default 'Set2']

- **title** (*dict, optional*) – Parameters of matplotlib.axes.Axes.set_title in a dict.

  E.g. {'label': '', 'fontdict': {}, 'loc': ''}
  See full parameter list in https://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html

- **characters** – A character in string or a list of characters for each category. [Default None]
- **font_size** – Font size of Font Awesome icons.

  The default size is not fixed and depends on the block size.
  Either an relative value of 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large' or an absolute font size.

- **font_file** – Path to custom font file.
- **icons** (*str|list[str]|tuple[str], optional*) – Icon name of Font Awesome.

  If it is a string, all categories use the same icon;
  If it's a list or tuple of icons, the length should be the same as values.
  See the full list of Font Awesome on https://fontawesome.com/icons?d=gallery&m=free
  [Default None]

- **icon_style** (*str|list[str]|tuple[str], optional*) – The style of icons to be used.

  Font Awesome Icons find an icon by style and icon name.

  The style could be 'brands', 'regular' and 'solid'. Visit https://fontawesome.com/cheatsheet for detail.

  If it is a string, it would search icons within given style.

If it is a list or a tuple, the length should be the same as values and it means the style for each icon.

[Default 'solid']

- **icon_size** (*int*|*str, optional*) – Font size of Font Awesome icons.

  Deprecated! Use font_size instead.
  The default size is not fixed and depends on the block size.
  Either an relative value of 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large' or an absolute value of font size.

- **icon_legend** (*bool, optional*) – Whether to use icon but not color bar in legend. [Default False]
- **plot_anchor** (*str, optional*) – The alignment method of subplots. `{'C', 'SW', 'S', 'SE', 'E', 'NE', 'N', 'NW', 'W'}`

  See details in https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.set_anchor.html
  [Default 'W']

- **plots** (*dict, optional*) – Position and parameters of Waffle class for subplots in a dict, with format like {pos: {subplot_args: values, }, }.

  Pos could be a tuple of three integer, where the first digit is the number of rows, the second the number of columns, and the third the index of the subplot.
  Pos could also be a 3-digit number in int or string type. For example, it accept 235 or '235' standing for the Ith plot on a grid with J rows and K columns. Note that all integers must be less than 10 for this form to work.
  The parameters of subplots are the same as Waffle class parameters, excluding plots itself.
  If any parameter of subplots is not assigned, it use the same parameter in Waffle class as default value.

- **vertical** (*bool, optional*) – Whether to draw the plot vertically or horizontally. [Default False]
- **starting_location** (*str, optional*) – Change the starting location plotting the blocks. `{'NW', 'SW', 'NE', 'SE'}`

  When it's 'NW', it means plots start from upper-left; 'SW' means plots start from lower-left; 'NE' means plots start from upper-right; 'SE' means plots start from lower-right.
  [Default 'SW']

- **rounding_rule** (*str, optional*) – The rounding rule applied when adjusting values to fit the chart size. {'nearest', 'floor', 'ceil'}

  When it's 'nearest', it is "round to nearest, ties to even" rounding mode;
  When it's 'floor', it rounds to less of the two endpoints of the interval;
  When it's 'ceil', it rounds to greater of the two endpoints of the interval.
  [Default 'nearest']

- **tight** (*bool|dict, optional*) – Set whether and how *.tight_layout* is called when drawing.

  It could be bool or dict with keys "pad", "w_pad", "h_pad", "rect" or None
  If a bool, sets whether to call *.tight_layout* upon drawing.
  If None, use the figure.autolayout param instead.
  If a dict, pass it as kwargs to *.tight_layout*, overriding the default paddings.
  [Default True]

- **block_arranging_style** (*string, optional*) – Set how to arrange blocks. {'normal', 'snake', 'new-line'}

  If it is 'normal', it draws blocks line by line with same direction.
  If it is 'snake', it draws blocks with snake pattern.
  If it is 'new-line', it starts with a new line when drawing each category. This only works when only one of rows and columns is assigned, and vertical=False when rows is assigned or vertical=True when rows is assigned.
  [Default 'normal']

classmethod make_waffle(*ax: matplotlib.axes._axes.Axes*, ***kwargs*)

Plot waffle chart on given axis. Run it with codes like: Waffle.make_waffle(ax=ax, rows=5, values=[48, 46, 6])

Note that calling this method does not update attributes, including fig_args, plot_args, and values_len

Parameters

- **ax** (*matplotlib.axes.Axes*) – An instance of Matplotlib Axes
- ****kwargs** – Waffle properties

Attributes:

Waffle.**fig_args: Dict**

All Waffle-specific arguments with default values

Waffle.**plot_args: List**

Standardized arguments of all subplots

Waffle.**values_len: Optional[int]**

The length of values

PYTHON MODULE INDEX

p
pywaffle.waffle, 22

INDEX

F

fig_args (*pywaffle.waffle.Waffle attribute*), 22

M

make_waffle() (*pywaffle.waffle.Waffle class method*), 22
module
    pywaffle.waffle, 19, 22

P

plot_args (*pywaffle.waffle.Waffle attribute*), 22
pywaffle.waffle
    module, 19, 22

V

values_len (*pywaffle.waffle.Waffle attribute*), 22

W

Waffle (*class in pywaffle.waffle*), 19