```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib.colors import LogNorm
from mpl_toolkits.basemap import Basemap


class Map:
    """
    Class to generate maps

    Notes
    -----
    How to use this class code example.
        code snippet::
            bin_size_y = 2
            bin_size_x = 2
            matrix_name = 'D0'
            lat_name = 'GeocLat'
            long_name = 'GeocLong'

            bins, matrix = Map.generate_2d_histogram(df, lat_name, long_name,
matrix_name, bin_size_y, bin_size_x)
            Map.show_map(bins[0], bins[1], matrix[2], "Map title")

    """


    @staticmethod
    def show_map(lon_bins, lat_bins, matrix, title, save=None,isLog=False,
vmax=None, vmin=None): #
        """
        Show a Cylindrical Projection map with a pseudo-color plot over the map

        Parameters
        ----------
        lon_bins: ndarray
            1-D array representing the coordinates of long bins provided by
generate_2d_histogram
        lat_bins: : ndarray
            1-D array representing the coordinates of lat bins provided by
generate_2d_histogram
        matrix: ndarray
            2-D array representing the values that needs to be shown on top of
the map
        title: string
            String representing the title of the plot
        save: string, optional
            String representing location/path where you want to save the output
map, if None it will not be saved
        isLog: bool, optional
            if True logarithmic scale is applied. Default is False
        vmax: int, optional
            int representing the max value that the scale will show, usually
used to give a same scale to different plots
```

```
        """
        m = Basemap(projection='cyl', llcrnrlat=-82, urcrnrlat=82,
llcrnrlon=-180, urcrnrlon=180, resolution='l')
        m.drawcoastlines()
        m.drawcountries()

        m.drawmeridians(np.arange(-180., 180., 10.), labels=[0, 0, 0, 1])  #
draw meridians
        m.drawparallels(np.arange(-80., 80., 5.), labels=[1, 0, 0, 0])  # draw
parallels

        # get the mesh for the lat and lon
        lon_bins_2d, lat_bins_2d = np.meshgrid(lon_bins, lat_bins)

        # convert the bin mesh to map coordinates:
        xs, ys = m(lon_bins_2d, lat_bins_2d)  # will be plotted using pcolormesh

        if np.any(xs > 1e20) or np.any(ys > 1e20):
            xs, ys, matrix = xs[1:-1, 1:-1], ys[1:-1, 1:-1], matrix[1:-1, 1:-1]

        if vmax is not None:
            m.pcolormesh(xs, ys, matrix, cmap="jet", shading='flat', vmax=vmax,
vmin=vmin)

        # if isLog and vmax is not None:
        #     m.pcolormesh(xs, ys, matrix, cmap="jet", shading='flat',
norm=LogNorm(), vmax=vmax, vmin=vmin)
        # elif isLog:
        #     m.pcolormesh(xs, ys, matrix, cmap="jet", shading='flat',
norm=LogNorm(), vmax=vmax, vmin=vmin)
        # elif vmax is not None:
        #     m.pcolormesh(xs, ys, matrix, cmap="jet", shading='flat',
vmax=vmax, vmin=vmin)
        # else:
        #     m.pcolormesh(xs, ys, matrix, cmap="jet", shading='flat',
vmax=vmax, vmin=vmin)

        cbar = plt.colorbar(orientation='horizontal', shrink=0.625, aspect=20,
fraction=0.2, pad=0.02)
        cbar.set_label('dB', size=18)
        cbar.set_ticklabels([vmin,vmax,20])
        plt.title(title)
        # make image bigger:
        plt.gcf().set_size_inches(20, 20)
        if save is not None:
            plt.savefig(save, bbox_inches='tight', pad_inches=1)
            plt.close()

    @staticmethod
    def is_nan(num):
        """
        Utils function to check if a number is nan
        """
        return num != num
```

```
    @staticmethod
    def generate_2d_histogram(df: pd.DataFrame, lat_name: str, long_name: str,
matrix_name: str, bin_size_y: float,
                              bin_size_x: float):
        """
        Return two tuples, the first with the bins and the second with the
different matrix types

        Generate all the items needed for the map

        Parameters
        ----------
        df: DataFrame
            DataFrame representing the information that will be aggregated
        lat_name: : string
            String representing the name of the column of the DataFrame for the
latitude
        long_name: string
            String representing the name of the column of the DataFrame for the
longitude
        matrix_name: string
            String representing the name of the column of the DataFrame for the
value that needs to be aggregate,
            this type must be an integer
        bin_size_y: int
            int representing the size of the bin for y
        bin_size_x: int
            int representing the size of the bin for x

        Returns
        -------
        (lon_bins, lat_bins): tuple
            lon_bins: ndarray
                1-D array representing the coordinates of long
            lat_bins: : ndarray
                1-D array representing the coordinates of lat
        (matrix_sum, matrix_mean, matrix_count): tuple
            matrix_sum: ndarray
                2-D array representing the values that needs to be shown on top
of the map with aggregation function sum,
                so all the value in the column the belong to the specific bin
will be sum each other
            matrix_mean: ndarray
                2-D array representing the values that needs to be shown on top
of the map with aggregation function mean,
                so all the value in the column the belong to the specific bin
will be sum each other and divided by the number of all values added
            matrix_count: ndarray
                2-D array representing the values that needs to be shown on top
of the map with aggregation function count,
                so all the value in the column the belong to the specific bin
will be counted
```

```
        Notes
        -----
        This function will modify the dataframe provided so in needed give a
copy instead.
        code snippet::
            generate_2d_histogram(df.copy(), lat_name, long_name, matrix_name,
bin_size_y, bin_size_x)

        If the type of your dataframe is not an integer but an object it can be
converted.
        code snippet::
            matrix_name = 'column_to_conver'
            df = df.assign(D0=pd.to_numeric(df[matrix_name], errors='coerce'))

        Make sure that the coordinates are 0-360 for the longitude and 0-180 for
the latitude.

        """

        def generate_index(bins, values):
            """
            helper function that generates the bins
            """
            round_to_hundreds = [round(num, 3) for num in bins]
            listp = []
            for value in values:
                if not Map.is_nan(value):
                    result = np.argwhere(round_to_hundreds == value.left)[0][0]
                    listp.append(result)
                else:
                    listp.append(value)

            return listp

        nx = int(360 / bin_size_x)
        ny = int(180 / bin_size_y)

        lon_bins = np.linspace(-180, 180, nx + 1).astype(np.float)
        lat_bins = np.linspace(-90, 90, ny + 1).astype(np.float)
        df['bin_long'] = pd.cut(df[long_name], bins=np.insert(lon_bins, 0,
(lon_bins[0] - 1)),duplicates='drop')
        df['bin_lat'] = pd.cut(df[lat_name], bins=np.insert(lat_bins, 0,
(lon_bins[0] - 1)),duplicates='drop')

        df['bin_long_index'] = generate_index(lon_bins, df['bin_long'].values)

        df['bin_lat_index'] = generate_index(lat_bins, df['bin_lat'].values)

        df_matrix = df.groupby(['bin_long_index',
'bin_lat_index'])[matrix_name].agg(
            ['sum', 'mean', 'count']).reset_index()

        count = df_matrix['bin_long_index'].size
```

```
matrix_sum = np.full([lat_bins.size, lon_bins.size], np.nan)
matrix_mean = np.full([lat_bins.size, lon_bins.size], np.nan)
matrix_count = np.full([lat_bins.size, lon_bins.size], np.nan)

for i in range(count):
    x = int(df_matrix.loc[i]['bin_long_index'])
    y = int(df_matrix.loc[i]['bin_lat_index'])
    sum_v = float(df_matrix.loc[i]['sum'])
    mean_v = float(df_matrix.loc[i]['mean'])
    count_v = float(df_matrix.loc[i]['count'])
    matrix_sum[y][x] = sum_v
    matrix_mean[y][x] = mean_v
    matrix_count[y][x] = count_v

return (lon_bins, lat_bins), (matrix_sum, matrix_mean, matrix_count)
```