```python
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats

def read_dataset(path):
    df = pd.read_csv(path, sep=';', encoding='cp1252')
    return df


def remove_typo(df):
    df.rename(columns={"Populatiion": "Population"}, inplace=True)
    return df

def clean_up(df):
    df['Area'] = df['Area'].str.replace(',', '.').astype(float)
    df["GDP per capita"] = df["GDP per capita"].str.replace(',', '.').astype(float)
    df["CO2 emission"] = df["CO2 emission"].str.replace(',', '.').astype(float)
    return df

def fix_negative(df):
    fix_gdp = df[df['GDP per capita'] < 0]
    area_gdp = df[df['Area'] < 0]
    fix_gdp['GDP per capita'] *= -1
    area_gdp['Area'] *= -1
    df[df['GDP per capita'] < 0] = fix_gdp
    df[df['Area'] < 0] = area_gdp
    return df

def fix_NaN(df):
    df = df.fillna(df.mean())
    return df


def print_exploring(df):
    print('Data frame info:')
    df.info()

    pd.set_option("display.max_columns", None)
    print('\nFirst 5 rows:')
    print(df.head())
```

```
    print('\nDescriptive statistics of the dataframe:')
    print(df.describe())


def normally_visual_test(df):
    fig, axs = plt.subplots(1, 4, figsize=(16, 4))

    fig.suptitle('Histograms', fontsize=16)

    axs[0].set_title('GDP per capita')
    axs[0].hist(df['GDP per capita'])

    axs[1].set_title('Population')
    axs[1].hist(df['Population'])

    axs[2].set_title('CO2 emission')
    axs[2].hist(df['CO2 emission'])

    axs[3].set_title('Area')
    axs[3].hist(df['Area'])
    plt.show()

def shapiro_test(df, columns=0, alpha=0.05):
    print("\nShapiro–Wilk test:")
    if columns==0:
        data = df
        columns = [1]
    for column in columns:
        if column != 1:
            data = df[column]
        stat, p = stats.shapiro(data)
        print('Statistics=%.3f, p=%.3f' % (stat, p))
        if p > alpha:
            print('The data correspond to a normal distribution')
        else:
            print('The data do not correspond to a normal distribution')

def ks_test(df, columns=0, alpha=0.05):
    print("\nKolmogorov–Smirnov test:")
```

```
    if columns==0:
        data = df
        columns = [1]
    for column in columns:
        if column != 1:
            data = df[column]
        stat, p = stats.kstest(data, 'norm')
        print('Statistics=%.3f, p=%.3f' % (stat, p))
        if p > alpha:
            print('The data correspond to a normal distribution')
        else:
            print('The data do not correspond to a normal distribution')

def dagostino_test(df, columns=0, alpha=0.05):
    print("\nD'Agostino's test:")
    if columns==0:
        data = df
        columns = [1]
    for column in columns:
        if column != 1:
            data = df[column]
        stat, p = stats.normaltest(data)
        print('Statistics=%.3f, p=%.3f' % (stat, p))
        if p > alpha:
            print('The data correspond to a normal distribution')
        else:
            print('The data do not correspond to a normal distribution')


def mean_median(df, columns):
    for column in columns:
        print(f'\n{column}:')
        mean_gdp = df[column].mean()
        median_gdp = df[column].median()
        print(f'Mean {column}', ' - ' ,mean_gdp)
        print(f'Median {column} ',  ' - ' , median_gdp, '\n')
        if mean_gdp == median_gdp:
            print(f"The mean and median {column} are the same: {mean_gdp}")

# def mean_median(df, columns):
```

```
#     for column in columns:
#         print(f'\n{column}:')
#         for region in df['Region'].unique():
#             region_data = df[df['Region'] == region]
#             mean_gdp = region_data[column].mean()
#             median_gdp = region_data[column].median()
#             print(f'Mean {column} in', region, ' - ' ,mean_gdp)
#             print(f'Median {column} in', region, ' - ' , median_gdp, '\n')
#             if mean_gdp == median_gdp:
#                 print(f"In the {region} region, the mean and median {column} are the same: {mean_gdp}")


def closest_co2(df):
    df['CO2 emission'].hist(by=df['Region'], layout=(4, 2), figsize=(10, 20))
    plt.show()

    for region in df['Region'].unique():
        region_emissions = df[df['Region'] == region]['CO2 emission']

        print(f'\nCheck for the region {region}:')
        try:
            shapiro_test(region_emissions)
        except ValueError as e:
            print(str(e))
        try:
            ks_test(region_emissions)
        except ValueError as e:
            print(str(e))
        try:
            dagostino_test(region_emissions)
        except ValueError as e:
            print(str(e))

def pie_chart(df):
    fig, ax = plt.subplots(figsize=(8, 8))
    fig.suptitle('Pie chart', fontsize=16)
    labels = pd.unique(df['Region'])
    wedges, texts, autotexts = ax.pie(df.groupby('Region').sum()['Population'], labels=labels,
                                      autopct='%1.1f%%', textprops=dict(color='w'))
```

```
    ax.set_title('Population by region')
    ax.legend(wedges, labels,
              title='Regions',
              loc='center left',
              bbox_to_anchor=(1, 0, 0, 1))

    plt.setp(autotexts, size=12, weight='bold')

    plt.show()


if __name__ == "__main__":
    #Основне завдання
    data_path = 'Data2.csv'
    #Читання файлу
    df = read_dataset(data_path)
    #Виправляємо дані
    df = remove_typo(df)
    df = clean_up(df)
    df = fix_negative(df)
    df = fix_NaN(df)
    #Проаналізуємо структуру
    print_exploring(df)
    #Перевіримо, чи є параметри, що розподілені за нормальним законом
    normally_visual_test(df)
    column = ['GDP per capita', 'Population', 'CO2 emission', 'Area']
    shapiro_test(df,column)
    ks_test(df, column)
    dagostino_test(df,column)
    #Перевіримо гіпотезу про рівність середнього і медіани для одного з параметрів
    mean_median(df, column)
    #Перевіримо, в якому регіоні розподіл викидів CO2 найбільш близький до нормального
    closest_co2(df)
    #Побудуємо кругову діаграму населення по регіонам
    pie_chart(df)


    #Додатове завдання
```

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy.spatial import distance


def map_downloading(img_path):
    map_img = mpimg.imread(img_path)
    return map_img

def bubles(map_img, coords, population):
    fig, ax = plt.subplots(figsize=(15, 15))
    fig.suptitle('Ukraine', fontsize=16)
    ax.imshow(map_img)
    ax.scatter(
        coords[:, 0],
        coords[:, 1],
        s=population * 2,
        c='green',
        alpha=0.5,
        linewidth=2
    )
    ax.axis('off')

    plt.show()

def greatest_distance(map_img, cities, coords):
    distances = distance.cdist(coords, coords, 'euclidean')

    city_A, city_B = np.unravel_index(distances.argmax(), distances.shape)

    pixel_distance = distances[city_A, city_B]

    ukraine_width_km = 1316
    km_per_pixel = ukraine_width_km / map_img.shape[1]
    km_distance = distances[city_A, city_B] * km_per_pixel
    print(f'Найбільша відстань - між містами {cities[city_A]} та {cities[city_B]}.')
    print(f'Відстань у пікселях: {pixel_distance:.2f} пікселів')
    print(f'Відстань у кілометрах: {km_distance:.2f} км')
```

```python
if __name__ == "__main__":
    #Додаткове завдання 1
    img_path = 'Ukraine.jpg'
    #Завантажемо карту
    map_img = map_downloading(img_path)
    #Розмістити бульбашки, що відповідають їх населенню, на довільних 5 містах
    cities = ['Київ', 'Краматорськ', 'Харків', 'Львів', 'Миколаїв']
    cities_coords = np.array([(387, 146), (715, 256), (647, 176), (91, 188), (452, 381)])
    cities_population = np.array([2884, 185, 1419, 721, 486])
    bubles(map_img, cities_coords, cities_population)
    #Знайти найбільшу відстань між містами в пікселях та кілометрах
    greatest_distance(map_img, cities, cities_coords)
```

```python
import geoviews as gv
from geoviews import dim
import pandas as pd
from geoviews.tile_sources import CartoDark
from geoviews.tile_sources import StamenTerrain


def read_dataset(path):
    df = pd.read_csv(path, sep=';', encoding='cp1252', decimal=',')
    print("Data:")
    # print(df)
    return df

def grouping(conflicts):
    grouped_conflicts = conflicts.groupby(['country', 'longitude', 'latitude']).size().to_frame('quantity').reset_index()
    print("\nGrouped data:")
    grouped_conflicts
    print(grouped_conflicts)
    return grouped_conflicts

def visualization_europe( g_conflicts):
    # Europe
    points = gv.Points(g_conflicts, ['longitude', 'latitude'])
    tiles = gv.tile_sources.StamenToner
    gv.output(tiles * points.opts(
```

```
        title='Spatial distribution of conflicts in Europe',
        color='quantity', size=dim('quantity') ** (1/2) * 5,
        cmap='Oranges', tools=['hover'], width=1000, height=700,
        show_legend=False, alpha=0.5, colorbar=True
    ))


def visualization_ukraine(conflicts):
    # Ukraine
    ukr_conflicts = conflicts[conflicts['country'] == 'Ukraine']
    ukr_grouped_conflicts = ukr_conflicts.groupby(['year', 'longitude',
'latitude']).size().to_frame('quantity').reset_index()
    print(ukr_grouped_conflicts)

    ukr_grouped_conflicts['year'] = ukr_grouped_conflicts['year'].apply(str)
    points = gv.Points(ukr_grouped_conflicts, ['longitude', 'latitude'])
    tiles = gv.tile_sources.CartoDark

    gv.output(tiles * points.opts(
    title='Spatial distribution of conflicts in Ukraine',
    color='year', size=dim('quantity') ** (1/2) * 10,
    cmap='Category10', tools=['hover'], width=1000, height=700,
    show_legend=False, alpha=0.8
    ))
    # Number of conflicts in Ukraine by year
    print('\nNumber of conflicts in Ukraine by year:')
    print(ukr_conflicts.groupby(['year']).size())


if __name__ == "__main__":
    gv.extension('bokeh', 'matplotlib')
    data_path = 'conflicts.csv'
    conflicts = read_dataset(data_path)
    g_conflicts = grouping(conflicts)
    visualization_europe(g_conflicts)
    visualization_ukraine(conflicts)
```