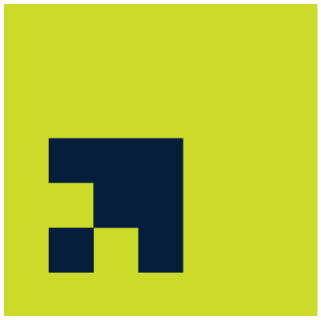


Gisma University of Applied Sciences

Data Viewer for Structure and Characteristics of Urban Areas in Rostock City

Project Documentation

Antonious Shehata - GH1028661

Mehmet Akif özkoç - GH1026518

Semester: SS0324

M604A Advanced Programming

Submission Date: July 2024

Table of Contents

Abstract

The Data Viewer for Structure and Characteristics of Urban Areas in Rostock City is a specialized tool designed to analyze and visualize demographic data. It provides insights into various aspects such as age distribution, gender demographics, household structures, and more, focusing specifically on urban areas within Rostock. This tool supports informed decision-making for policymakers and urban planners, aiming to enhance resource allocation and community development strategies.

Introduction

In contemporary governance and urban planning, data-driven insights are essential. The Data Viewer for Structure and Characteristics of Urban Areas in Rostock City serves as a critical instrument for understanding demographic trends within specific urban zones of Rostock. By examining key indicators like age groups, gender ratios, and household compositions, this tool aids in identifying priority areas for targeted interventions and service improvements. It empowers decision-makers to formulate effective policies that address the unique needs of urban residents, fostering sustainable urban development and improving overall quality of life.

Rostock and Its Urban Areas

Rostock, located on the northern coast of Germany along the Baltic Sea, is known for its rich history, vibrant culture, and economic significance. The city comprises diverse urban areas, each with unique demographic profiles shaped by factors such as historical development, economic activities, and social dynamics. The Data Viewer focuses on these urban areas, providing a detailed analysis of their demographic structure and characteristics to support local governance and urban planning initiatives.

Project Overview

Components and Features

Data Source:

The tool accesses demographic data from an Excel file named "Rostock.xlsx," which categorizes information into structured topics and subtopics.

User Interface (UI):

Graphical Interface (GUI):

Developed using Python's Tkinter library, the UI features:

Dropdown menus for selecting urban areas, main topics (e.g., Age, Gender), and subtopics (specific data points within topics).

A text area to display detailed comparative data between selected urban areas.

Graphical representations like bar graphs for visualizing trends and comparisons based on selected data.

Features

Urban Area Selection: Users can choose specific urban areas within Rostock for comparative demographic analysis.

Topic and Subtopic Selection: Dropdown menus enable selection of main topics (e.g., Age, Gender) and specific subtopics (e.g., Median Age, Household Structure).

Data Display: Upon selection, the tool generates a comparative table within the text area, providing a breakdown of data points for the selected urban areas across chosen subtopics.

Graphical Representation: Data is visually presented through interactive bar graphs, facilitating easy interpretation of trends and disparities between urban areas over specified periods.

Brief Description:

Project Name:

Data Viewer for Structure and Characteristics of Urban Areas in Rostock City

Developing Language and consul used:

Python, visual studio code

Python Version to be installed:

python 3.11.8 64 bit

Libraries to be installed:

tkinder, PIL, matplotlib, openpyxl,numpy, os

Import from tkinder:

ttk

Import from PIL:

Image, ImageTK

Import from matplotlib:

FigureCanvasTkAgg, matloplotlib.pyplot

Import from matplotlib.figure:

Figure

Project Link:

https://github.com/AntoniousShehata/Rostock.git

Main Details:

Main important files:

Rostock.xlsx as a dataset, Reports folder with Rostock.xlsx

Number of pages:

Three Pages

Name of pages:

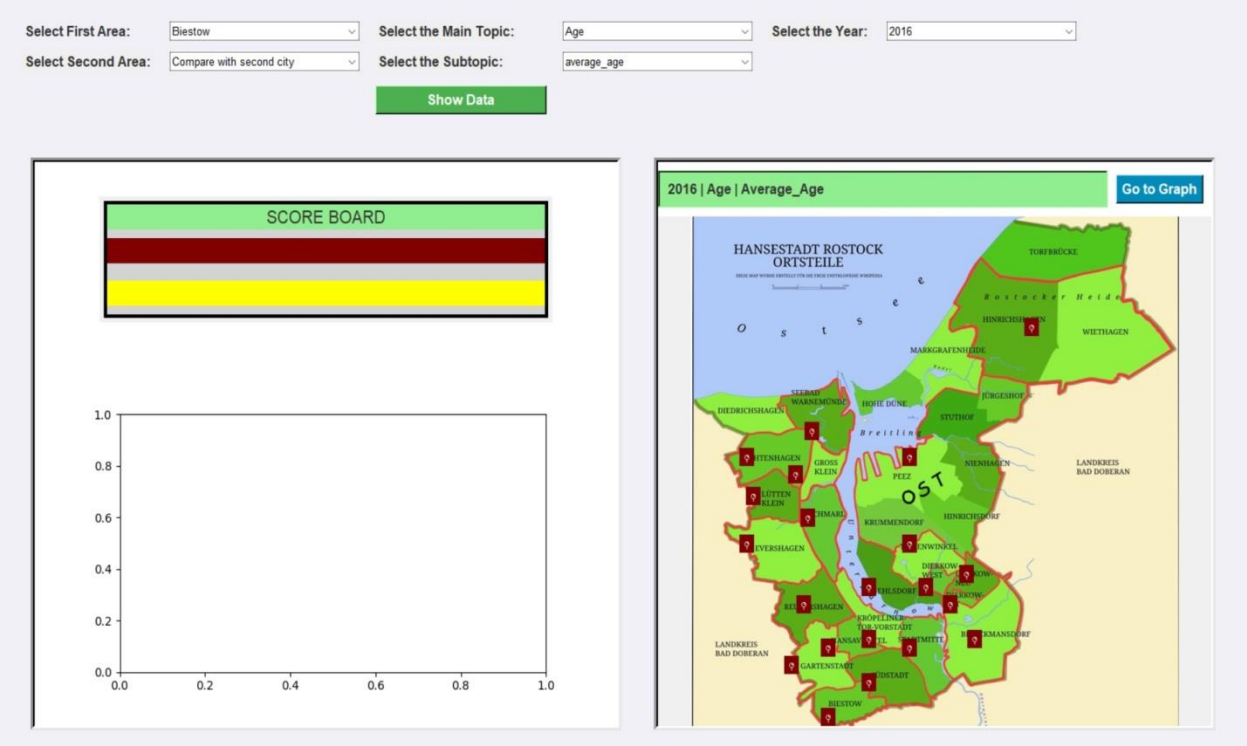
Home.py, dataload.py, styles.py, mergeprocess.py

Interface description:

Main screen:

Frame named **"Data Viewer for Structure and Characteristics of Urban Areas in Rostock City"**

Screen Shots for interface:



Consists of:

7 Labels, 5 dropdowns, 2 buttons, Score board, interactive map.

7 Labels:

"Select First Area" label points to its drowndown to choose first area (needed).

"Select Second Area" label points to its drowndown to choose Second area (Optional).

"Select The Main Topic" label points to its drowndown to choose Main Topic (needed).

"Select The Subtopic" label points to its drowndown to choose Subtopic (needed).

"Select The Year" label points to its drowndown to choose Year (map filter).

"Score Board" Green label points to data displayed highest and lowest values (statistics).

"2016 | Age | Average_Age" label above the map it is a dynamic title for the displayed data.

5 Dropdowns:

"Select First Area" drowndown to choose first area 1 of 21 items (needed).

"Select Second Area" drowndown to choose Second area 1 of 21 items (Optional).

- default value "Compare with second city" means no second city chosen.

"Select The Main Topic" drowndown to choose Main Topic 1 of 9 items (needed).

"Select The Subtopic" drowndown to choose Subtopic with respect to the main topic (needed).

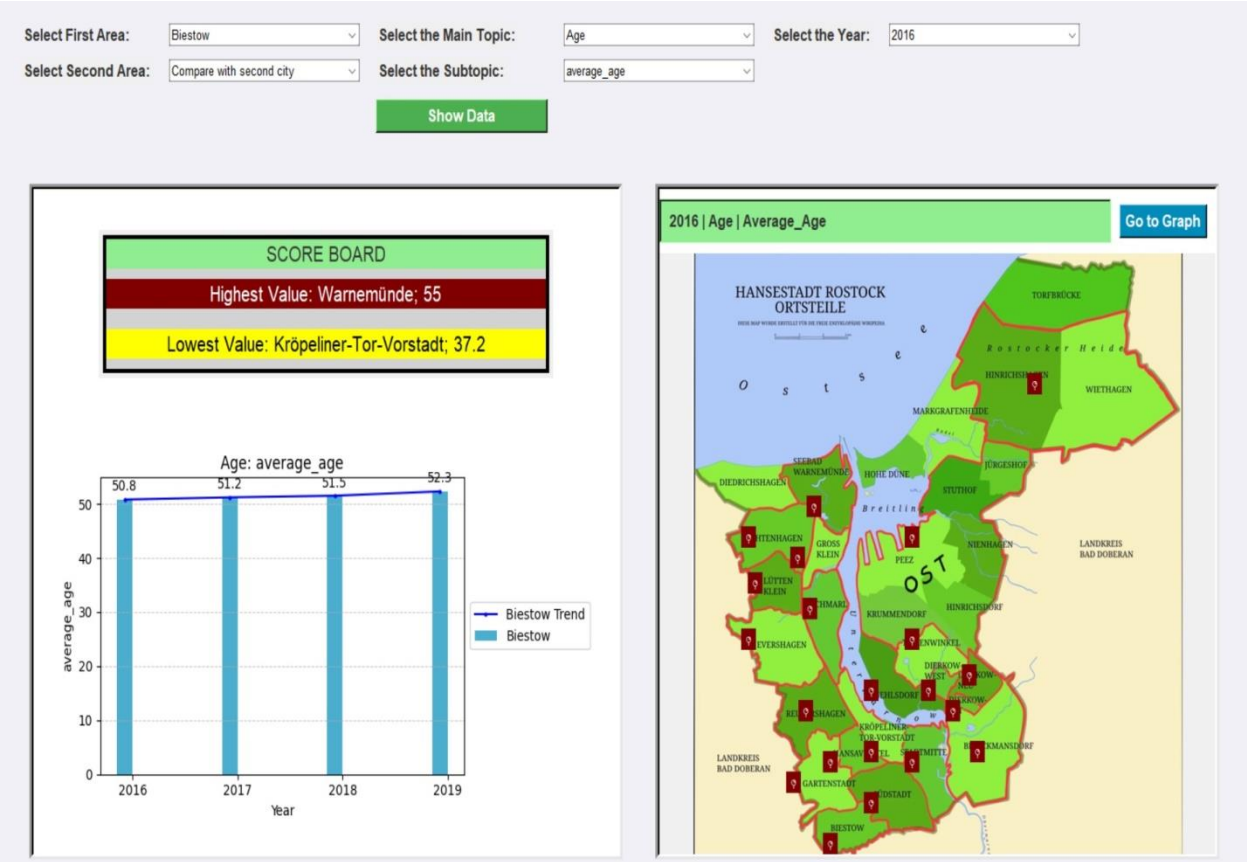
"Select the year" drowndown to choose a year to display its data on the map (Optional).

2 Buttons:

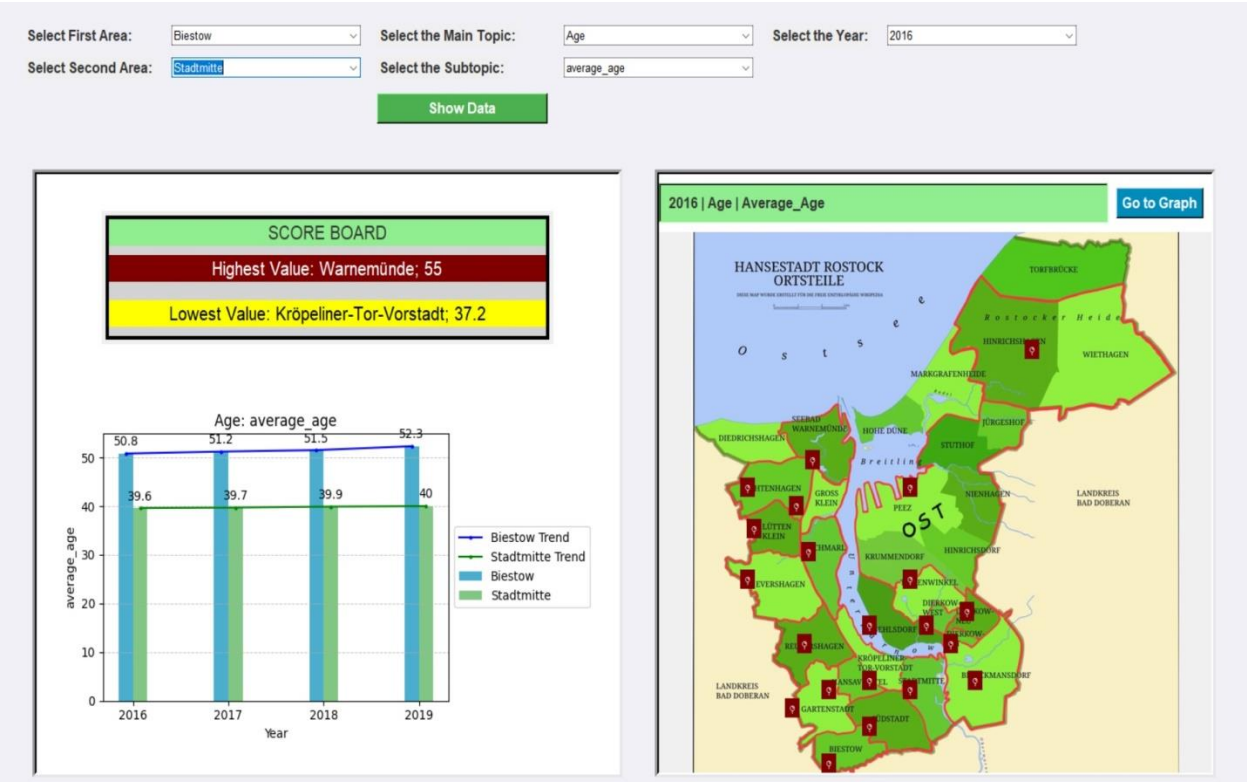
"Show Data" display data selected from three/four dropdowns to the graph and the text area.

"Go to Graph" display data selected: subtopic for 21 areas at a selected year.

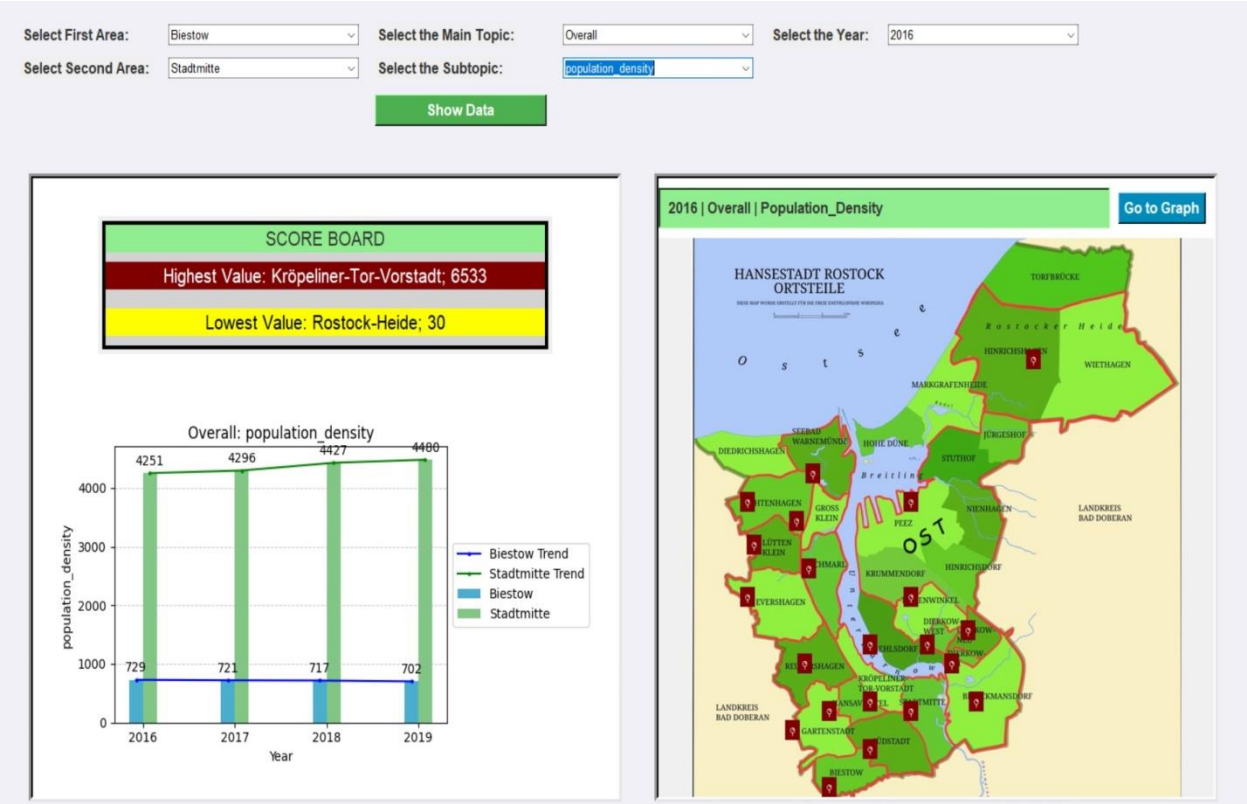
Change First Area



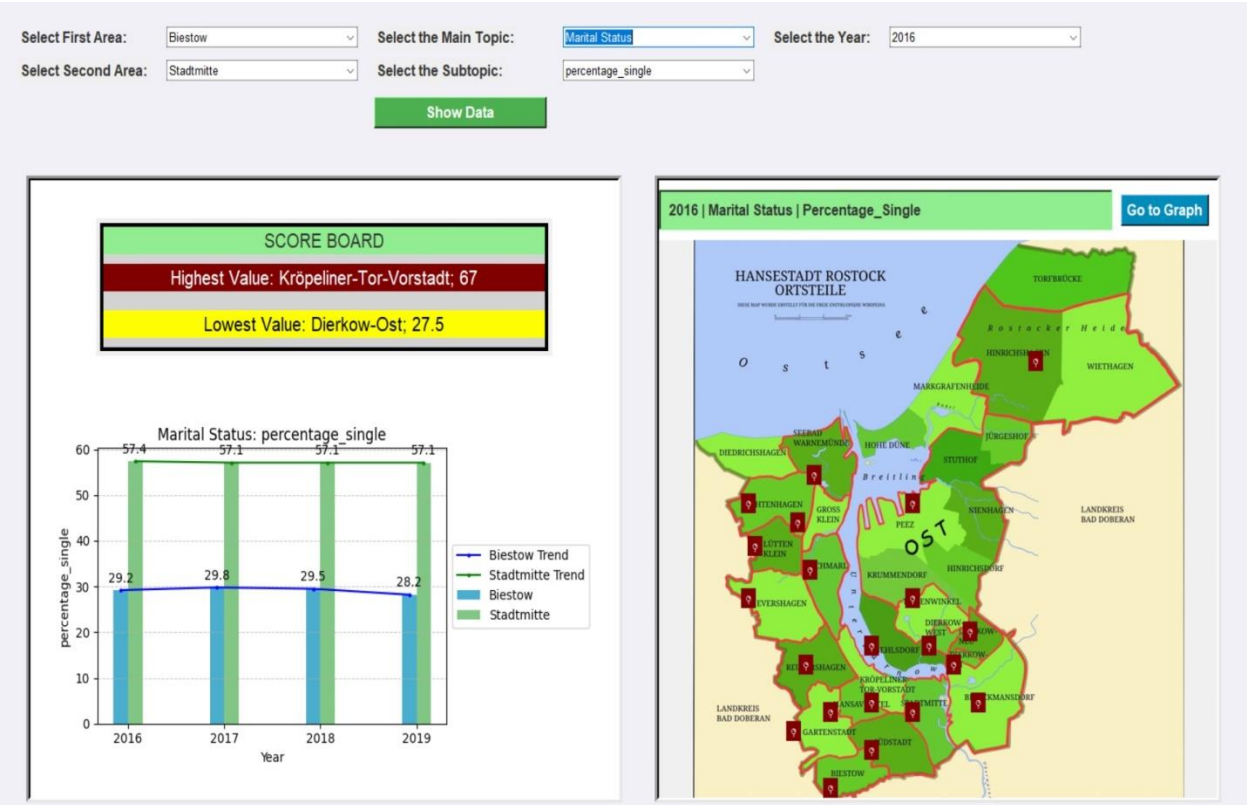
Change Second Area



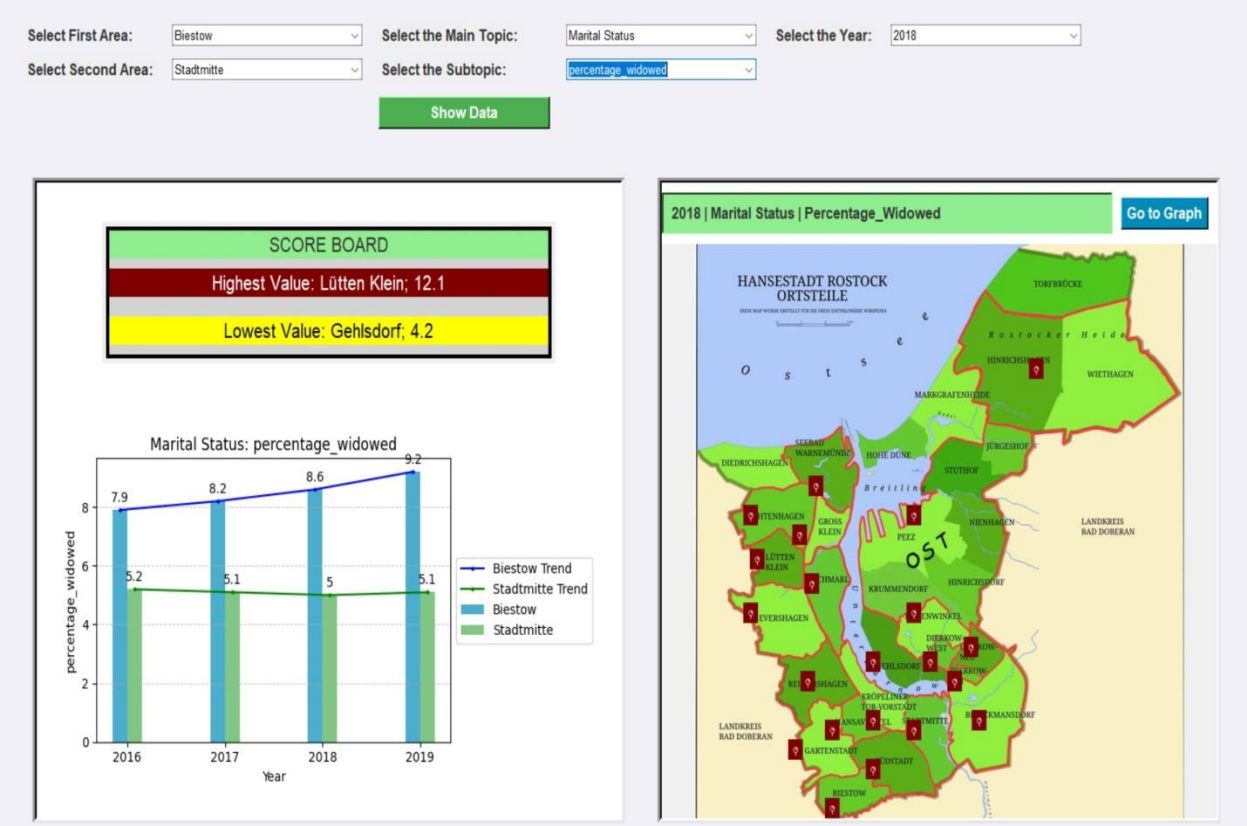
Change the main topic



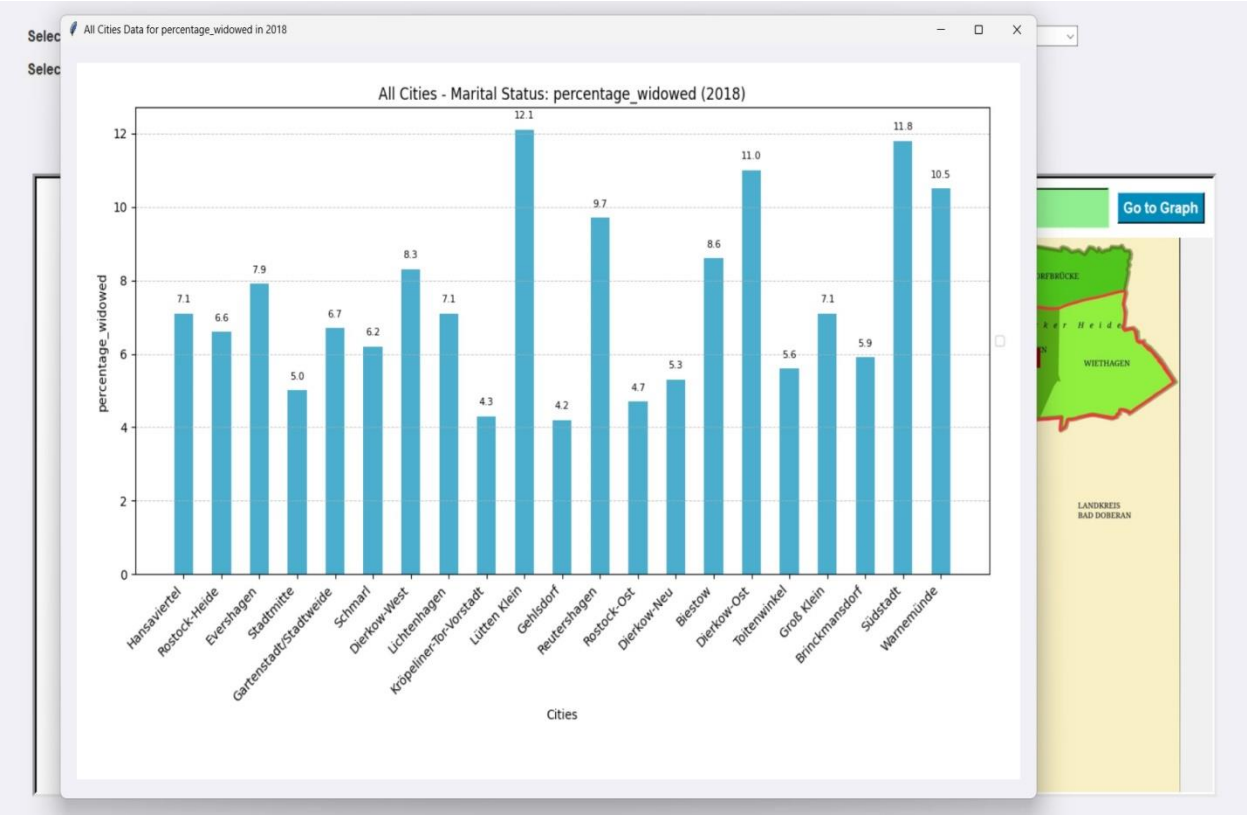
Change the subtopic



Change the year



Click “Got to graph”



Dataset integration process: (See Appendix A: Project Source Code Figure 1-4)

Purpose:

The code reads data from multiple Excel files stored in different directories corresponding to different years (2016 to 2019), processes this data, combines it into a single dataset, renames columns according to a specified mapping, and finally saves the combined and processed data into a new Excel file named 'Rostock.xlsx'.

Steps and Explanation:

1. **Imports**:
   - `import os`: Used for interacting with the operating system (e.g., file paths).
   - `import pandas as pd`: Used for data manipulation and analysis.
2. **Directories and File Names**:
   - `directories`: List of directory paths where data files for each year are stored.
   - `file_names`: List of specific Excel file names to be read from each directory.
3. **Function `read_and_prepare_data`**:
   - Reads an Excel file from a given `file_path`.
   - Removes any unnamed columns that may arise from merged cells.
   - Drops duplicate rows based on specified columns (`stadtbereich_code` and `stadtbereich_bezeichnung`).
4. **Processing Loop**:
   - Iterates through each directory (`directories`).

- Reads each specified Excel file (`file_names`) using `read_and_prepare_data`.
- Concatenates the cleaned data frames (`all_dfs`) horizontally (`axis=1`).
- Adds a `year` column based on the directory name to each concatenated dataset.
- Inserts `stadtbereich_code` and `stadtbereich_bezeichnung` columns from the first sheet into desired positions.

5. **Combining Data Across Years**:
   - Stores each processed dataset (`final_data`) in a dictionary (`all_years_data`) with the year as the key.
   - Concatenates all datasets from `all_years_data` into a single dataframe (`combined_data`).
6. **Column Renaming**:
   - Renames columns in `combined_data` according to a predefined mapping (`column_rename_map`).
7. **Saving Data**:
   - Saves the final processed dataset (`combined_data`) to an Excel file 'Rostock.xlsx'

Summary of integration:

This script effectively gathers data from multiple Excel files across several years, processes and cleans the data, combines it into a unified dataset, standardizes column names, and exports the consolidated data to a new Excel file. It utilizes pandas for data manipulation and “os” for file handling, ensuring that the final output meets the specified format and content requirements.

Code functionality:

main.py: **(See Appendix A: Project Source Code Figure 5)**

1. City_names_ordered (See Appendix A: Project Source Code Figure 6)

```python
city_names_ordered = list(city_points.values())
```

This list contains city names ordered by city points. The points can be used to rank or sort cities based on specific criteria. This get its data from “dataload.py”

2. Function: `read_subtopics(main_topic)` (See Appendix A: Project Source Code Figure 7)

```python
def read_subtopics(main_topic): ...
```

- **Purpose:** Reads and returns subtopics related to the selected main topic.
- **Parameters:** `main_topic` (str) - The main topic for which subtopics are to be read.
- **Returns:** List of subtopics related to the main topic.

3. Function: `read_data(city_code, main_topic, subtopic)`

(See Appendix A: Project Source Code Figure 8)

```python
def read_data(city_code, main_topic, subtopic): ...
```

- **Purpose:** Reads data for a specific city and subtopic.
- **Parameters:**
  - `city_code` (str) - The code representing the specific city.
  - `main_topic` (str) - The main topic of interest.
  - `subtopic` (str) - The subtopic under the main topic.
- **Returns:** Data related to the specified city, main topic, and subtopic.

4. Function: `read_data_for_all_cities(main_topic, subtopic)`

(See Appendix A: Project Source Code Figure 9)

```python
def read_data_for_all_cities(main_topic, subtopic): ...
```

- **Purpose:** Reads data for all cities for a specific subtopic.
- **Parameters:**
  - `main_topic` (str) - The main topic of interest.
  - `subtopic` (str) - The subtopic under the main topic.
- **Returns:** Data related to all cities for the specified main topic and subtopic.

5. Function: `get_years()`

(See Appendix A: Project Source Code Figure 10)

```python
def get_years(): ...
```

- **Purpose:** Retrieves all unique years from the dataset.
- **Parameters:** None.
- **Returns:** List of unique years available in the dataset.

6. Function: `update_subtopics(event=None)`

(See Appendix A: Project Source Code Figure 11)

```python
def update_subtopics(event=None): ...
```

- **Purpose:** Updates the subtopic dropdown based on the selected main topic.
- **Parameters:** `event` (optional) - The event that triggers the update.
- **Returns:** None.

7. Function: `update_info_text(event=None)`

(See Appendix A: Project Source Code Figure 12)

```python
def update_info_text(event=None): ...
```

- **Purpose:** Updates the information text displayed based on user interaction.
- **Parameters:** `event` (optional) - The event that triggers the update.
- **Returns:** None.

8. Function: `on_city_hover(event)`

(See Appendix A: Project Source Code Figure 13)

```python
def on_city_hover(event): ...
```

- **Purpose:** Handles the city button hover event to provide interactive feedback.
- **Parameters:** `event` - The hover event.
- **Returns:** None.

9. Function: `on_hover(event, city)`

(See Appendix A: Project Source Code Figure 14)

- **Purpose:** Handles the mouse hover event over a city button to highlight or provide additional information.
- **Parameters:**
  - `event` - The hover event.
  - `city` (str) - The city being hovered over.
- **Returns:** None.

10. Function: `on_leave(event)` **(See Appendix A: Project Source Code Figure 15)**

- **Purpose:** Handles the mouse leave event from a city button to remove any highlights or additional information.
- **Parameters:** `event` - The leave event.
- **Returns:** None.

11. Function: `create_main_window()` **(See Appendix A: Project Source Code Figure 16 - 23)**

Description: Creates the main window and UI components of the application, including dropdowns, buttons, labels, and frames for displaying data and graphs.

Details:

- Sets up the main window with a full-screen attribute.
- Creates control, content, display, and image frames for organizing UI components.
- Configures dropdowns for selecting cities, main topics, subtopics, and years.
- Displays data and graphs based on user selections.
- Loads an image of Rostock and places markers for different city points.
- Sets up event bindings for hover and click events.

Usage: This function is called when the script is executed as the main program.

Main Execution

```python
if __name__ == "__main__":
    create_main_window()
```

Description: Checks if the script is run directly (not imported as a module) and creates the main window.

dataload.py:

1. Reading the Excel File (See Appendix A: Project Source Code Figure 24)

file_path: Specifies the path to the Excel file.

wb: Loads the workbook from the specified file path.

sheet: Sets the active sheet in the workbook for data extraction.

2. Defining Column Ranges (See Appendix A: Project Source Code Figure 25)

column_ranges: A dictionary that maps main topics to their corresponding column ranges in the Excel sheet. Each topic is associated with a range of columns.

3. Defining map Points (See Appendix A: Project Source Code Figure 26)

map_points: A list of tuples representing geographic coordinates (x, y) for different districts in the city.

4. City Points Dictionary (See Appendix A: Project Source Code Figure 27)

city_points: A dictionary mapping city codes to their corresponding city names.

5. Reading Cities from Excel (See Appendix A: Project Source Code Figure 28)

read_cities_from_excel(): A function that reads city codes and names from the Excel sheet starting from the second row.

- **cities:** A dictionary to store city codes and names.
- **sorted_cities:** A dictionary of cities sorted by their names.

6. Storing City Names Globally (See Appendix A: Project Source Code Figure 29)

- **city_names:** A global variable to store the sorted dictionary of city names.

Conclusion:

The Data Viewer for Structure and Characteristics of Urban Areas in Rostock City supports evidence-based decision-making and strategic planning for urban development. By harnessing demographic insights, the tool empowers stakeholders to allocate resources effectively, address community needs, and foster inclusive growth across Rostock's urban landscape.

References:

GitHub: Let's build from here. (n.d.). GitHub. https://github.com/

3.12.2 Documentation. (n.d.). https://docs.python.org/

Stack Overflow - Where Developers Learn, Share, & Build Careers. (n.d.). Stack Overflow. https://stackoverflow.com/

PyPI · The Python Package Index. (n.d.). PyPI. https://pypi.org/

data.europa.eu. (n.d.). https://data.europa.eu/data/datasets/821ea501-d5a9-46b8-aa21-e01f4f9dbd22?locale=en

data.europa.eu. (n.d.-b). https://data.europa.eu/data/datasets/b879e3cf-7671-4989-aa3a-5c8cbc7442bd?locale=en

data.europa.eu. (n.d.-c). https://data.europa.eu/data/datasets/e0f296a7-adbe-4a0f-a0f6-6a464b3e6dc7?locale=en

https://data.europa.eu/data/datasets/38cc66ec-be3c-45e3-bef3-c781ed9c1e2d?locale=en

tkinter — Python interface to Tcl/Tk. (n.d.). Python Documentation. https://docs.python.org/3/library/tkinter.html#threading-model

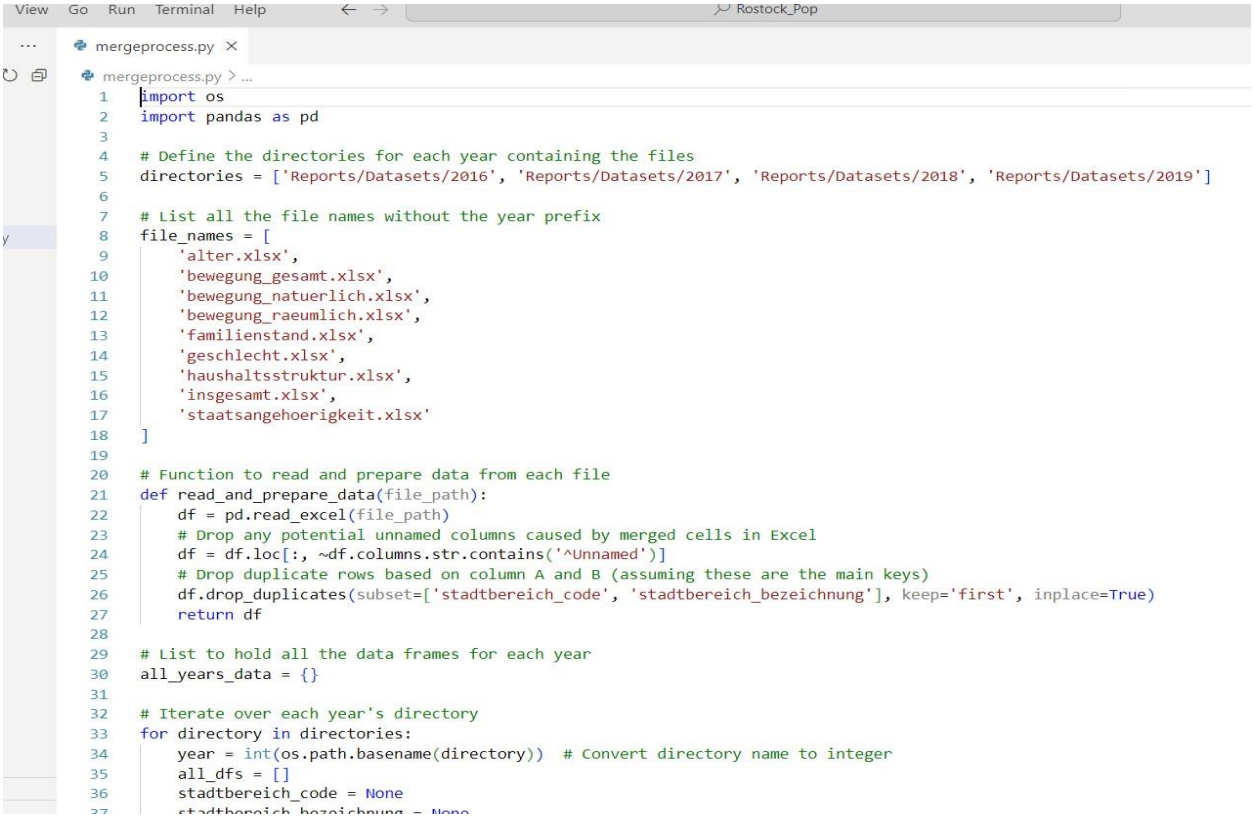
Pillow. (n.d.). Pillow (PIL Fork). https://pillow.readthedocs.io/en/stable/

Plot types — Matplotlib 3.9.0 documentation. (n.d.). https://matplotlib.org/stable/plot_types/index.html

NumPy Documentation. (n.d.). https://numpy.org/doc/

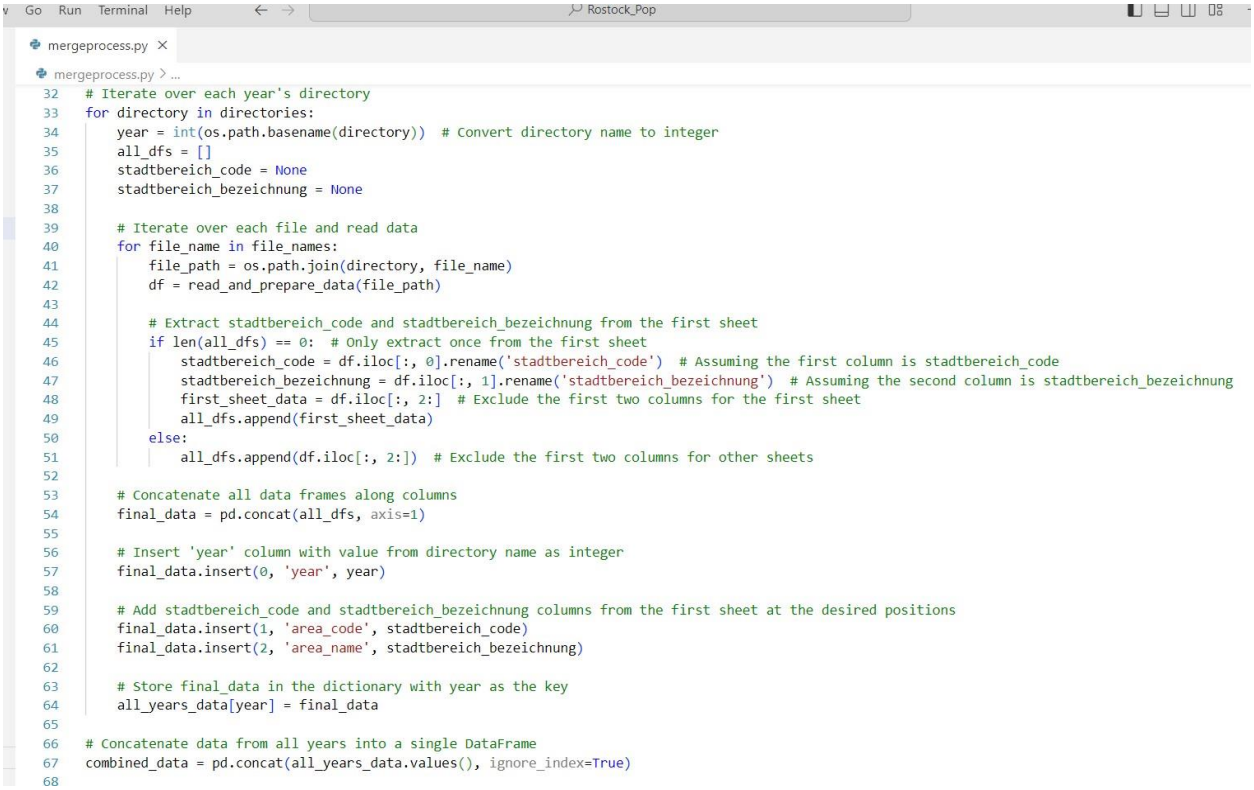
Appendix A: Project Source Code

Figure 1



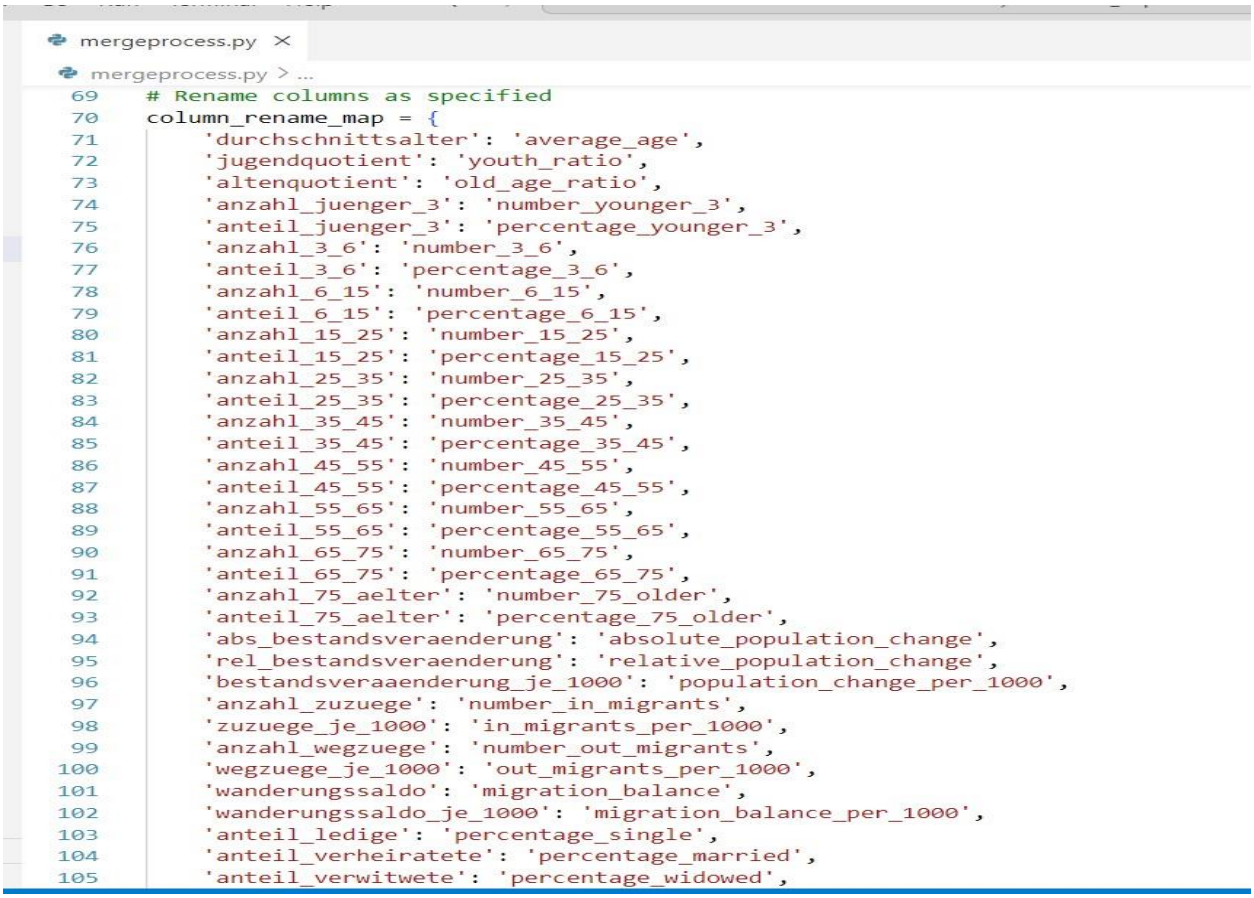
```python
import os
import pandas as pd

# Define the directories for each year containing the files
directories = ['Reports/Datasets/2016', 'Reports/Datasets/2017', 'Reports/Datasets/2018', 'Reports/Datasets/2019']

# List all the file names without the year prefix
file_names = [
    'alter.xlsx',
    'bewegung_gesamt.xlsx',
    'bewegung_natuerlich.xlsx',
    'bewegung_raeumlich.xlsx',
    'familienstand.xlsx',
    'geschlecht.xlsx',
    'haushaltsstruktur.xlsx',
    'insgesamt.xlsx',
    'staatsangehoerigkeit.xlsx'
]

# Function to read and prepare data from each file
def read_and_prepare_data(file_path):
    df = pd.read_excel(file_path)
    # Drop any potential unnamed columns caused by merged cells in Excel
    df = df.loc[:, ~df.columns.str.contains('^Unnamed')]
    # Drop duplicate rows based on column A and B (assuming these are the main keys)
    df.drop_duplicates(subset=['stadtbereich_code', 'stadtbereich_bezeichnung'], keep='first', inplace=True)
    return df

# List to hold all the data frames for each year
all_years_data = {}

# Iterate over each year's directory
for directory in directories:
    year = int(os.path.basename(directory))  # Convert directory name to integer
    all_dfs = []
    stadtbereich_code = None
```

Figure 2



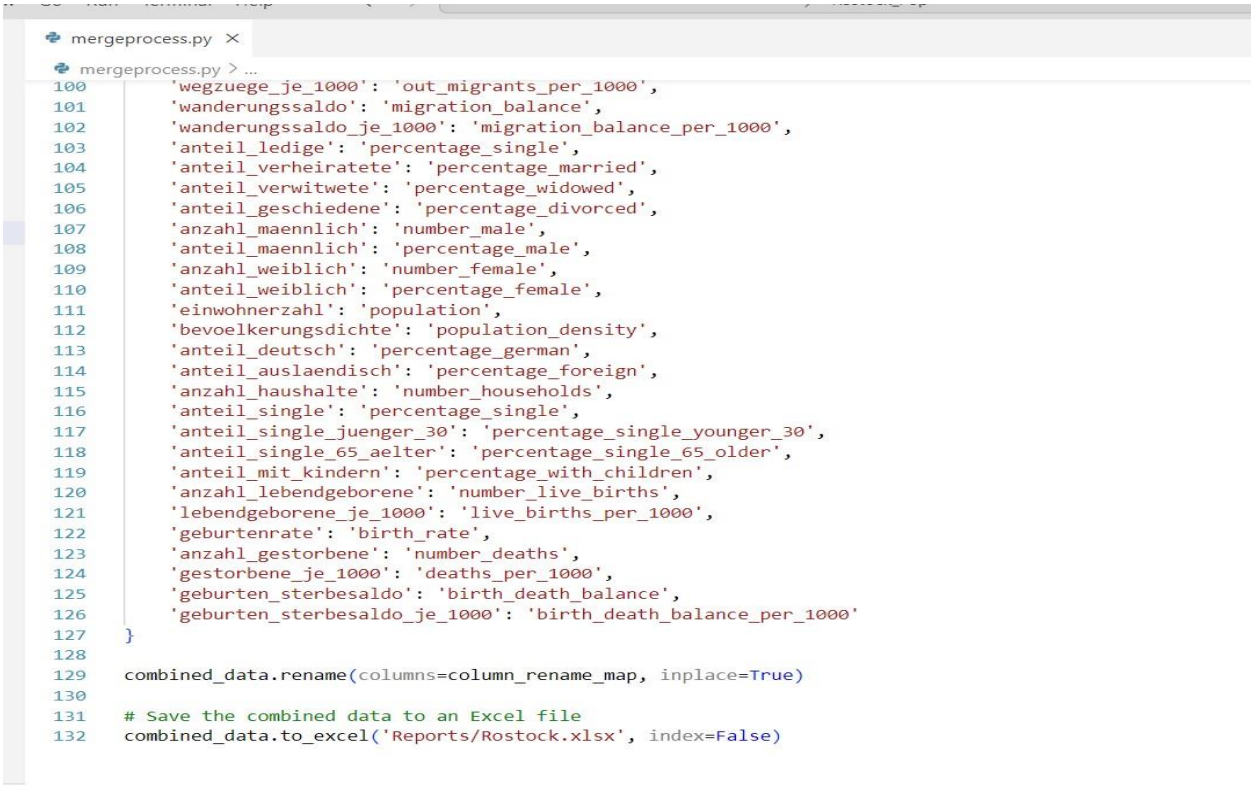
```python
# Iterate over each year's directory
for directory in directories:
    year = int(os.path.basename(directory))  # Convert directory name to integer
    all_dfs = []
    stadtbereich_code = None
    stadtbereich_bezeichnung = None

    # Iterate over each file and read data
    for file_name in file_names:
        file_path = os.path.join(directory, file_name)
        df = read_and_prepare_data(file_path)

        # Extract stadtbereich_code and stadtbereich_bezeichnung from the first sheet
        if len(all_dfs) == 0:  # Only extract once from the first sheet
            stadtbereich_code = df.iloc[:, 0].rename('stadtbereich_code')  # Assuming the first column is stadtbereich_code
            stadtbereich_bezeichnung = df.iloc[:, 1].rename('stadtbereich_bezeichnung')  # Assuming the second column is stadtbereich_bezeichnung
            first_sheet_data = df.iloc[:, 2:]  # Exclude the first two columns for the first sheet
            all_dfs.append(first_sheet_data)
        else:
            all_dfs.append(df.iloc[:, 2:])  # Exclude the first two columns for other sheets

    # Concatenate all data frames along columns
    final_data = pd.concat(all_dfs, axis=1)

    # Insert 'year' column with value from directory name as integer
    final_data.insert(0, 'year', year)

    # Add stadtbereich_code and stadtbereich_bezeichnung columns from the first sheet at the desired positions
    final_data.insert(1, 'area_code', stadtbereich_code)
    final_data.insert(2, 'area_name', stadtbereich_bezeichnung)

    # Store final_data in the dictionary with year as the key
    all_years_data[year] = final_data

# Concatenate data from all years into a single DataFrame
combined_data = pd.concat(all_years_data.values(), ignore_index=True)
```

Figure 3



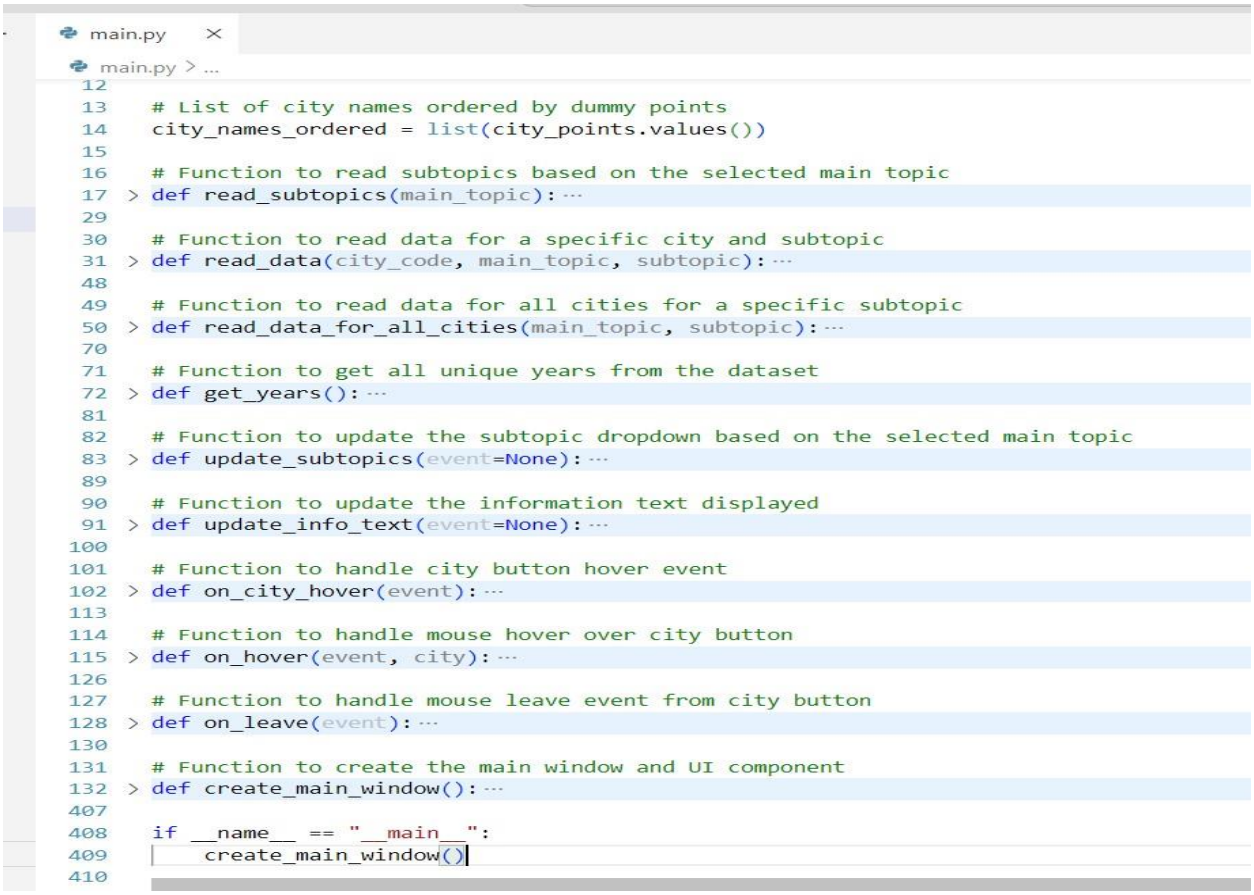
```
mergeprocess.py
# Rename columns as specified
column_rename_map = {
    'durchschnittsalter': 'average_age',
    'jugendquotient': 'youth_ratio',
    'altenquotient': 'old_age_ratio',
    'anzahl_juenger_3': 'number_younger_3',
    'anteil_juenger_3': 'percentage_younger_3',
    'anzahl_3_6': 'number_3_6',
    'anteil_3_6': 'percentage_3_6',
    'anzahl_6_15': 'number_6_15',
    'anteil_6_15': 'percentage_6_15',
    'anzahl_15_25': 'number_15_25',
    'anteil_15_25': 'percentage_15_25',
    'anzahl_25_35': 'number_25_35',
    'anteil_25_35': 'percentage_25_35',
    'anzahl_35_45': 'number_35_45',
    'anteil_35_45': 'percentage_35_45',
    'anzahl_45_55': 'number_45_55',
    'anteil_45_55': 'percentage_45_55',
    'anzahl_55_65': 'number_55_65',
    'anteil_55_65': 'percentage_55_65',
    'anzahl_65_75': 'number_65_75',
    'anteil_65_75': 'percentage_65_75',
    'anzahl_75_aelter': 'number_75_older',
    'anteil_75_aelter': 'percentage_75_older',
    'abs_bestandsveraenderung': 'absolute_population_change',
    'rel_bestandsveraenderung': 'relative_population_change',
    'bestandsveraaenderung_je_1000': 'population_change_per_1000',
    'anzahl_zuzuege': 'number_in_migrants',
    'zuzuege_je_1000': 'in_migrants_per_1000',
    'anzahl_wegzuege': 'number_out_migrants',
    'wegzuege_je_1000': 'out_migrants_per_1000',
    'wanderungssaldo': 'migration_balance',
    'wanderungssaldo_je_1000': 'migration_balance_per_1000',
    'anteil_ledige': 'percentage_single',
    'anteil_verheiratete': 'percentage_married',
    'anteil_verwitwete': 'percentage_widowed',
```

Figure 4



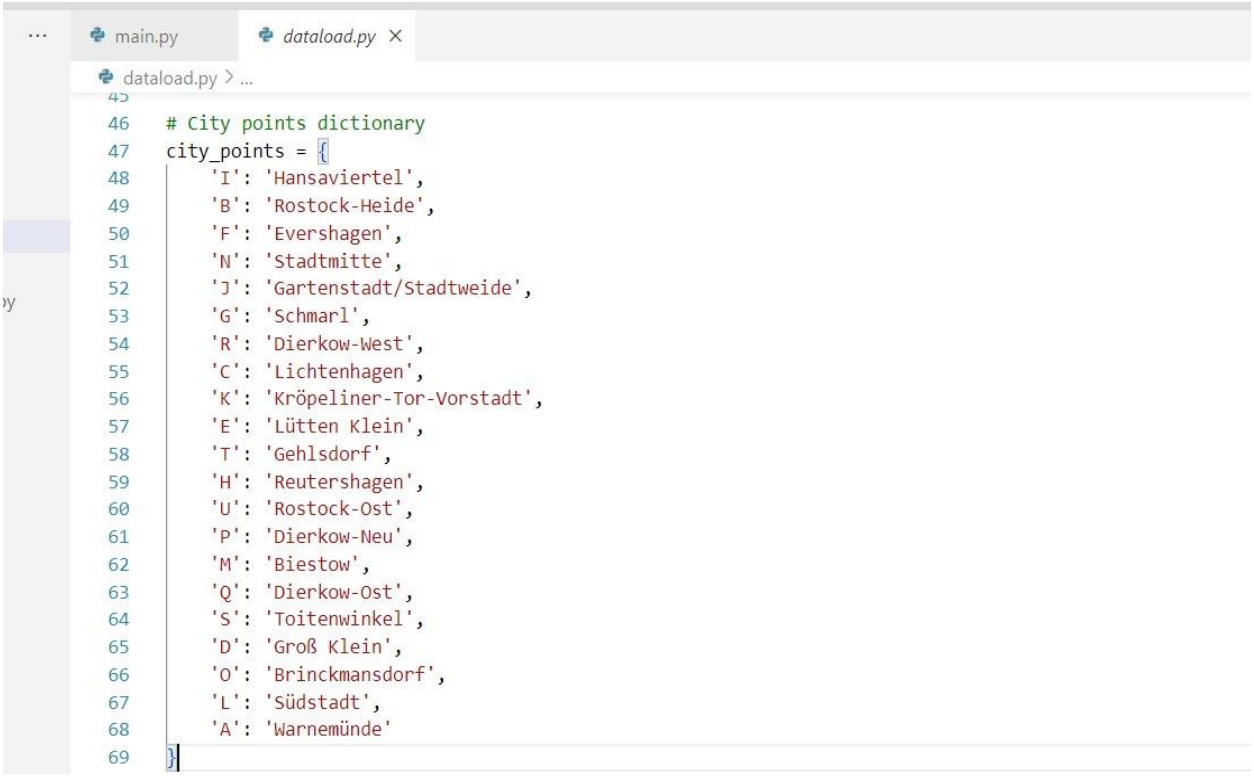
```
mergeprocess.py
    'wegzuege_je_1000': 'out_migrants_per_1000',
    'wanderungssaldo': 'migration_balance',
    'wanderungssaldo_je_1000': 'migration_balance_per_1000',
    'anteil_ledige': 'percentage_single',
    'anteil_verheiratete': 'percentage_married',
    'anteil_verwitwete': 'percentage_widowed',
    'anteil_geschiedene': 'percentage_divorced',
    'anzahl_maennlich': 'number_male',
    'anteil_maennlich': 'percentage_male',
    'anzahl_weiblich': 'number_female',
    'anteil_weiblich': 'percentage_female',
    'einwohnerzahl': 'population',
    'bevoelkerungsdichte': 'population_density',
    'anteil_deutsch': 'percentage_german',
    'anteil_auslaendisch': 'percentage_foreign',
    'anzahl_haushalte': 'number_households',
    'anteil_single': 'percentage_single',
    'anteil_single_juenger_30': 'percentage_single_younger_30',
    'anteil_single_65_aelter': 'percentage_single_65_older',
    'anteil_mit_kindern': 'percentage_with_children',
    'anzahl_lebendgeborene': 'number_live_births',
    'lebendgeborene_je_1000': 'live_births_per_1000',
    'geburtenrate': 'birth_rate',
    'anzahl_gestorbene': 'number_deaths',
    'gestorbene_je_1000': 'deaths_per_1000',
    'geburten_sterbesaldo': 'birth_death_balance',
    'geburten_sterbesaldo_je_1000': 'birth_death_balance_per_1000'
}

combined_data.rename(columns=column_rename_map, inplace=True)

# Save the combined data to an Excel file
combined_data.to_excel('Reports/Rostock.xlsx', index=False)
```

Figure 5



```
main.py
main.py > ...
12
13  # List of city names ordered by dummy points
14  city_names_ordered = list(city_points.values())
15
16  # Function to read subtopics based on the selected main topic
17  > def read_subtopics(main_topic): ···
29
30  # Function to read data for a specific city and subtopic
31  > def read_data(city_code, main_topic, subtopic): ···
48
49  # Function to read data for all cities for a specific subtopic
50  > def read_data_for_all_cities(main_topic, subtopic): ···
70
71  # Function to get all unique years from the dataset
72  > def get_years(): ···
81
82  # Function to update the subtopic dropdown based on the selected main topic
83  > def update_subtopics(event=None): ···
89
90  # Function to update the information text displayed
91  > def update_info_text(event=None): ···
100
101 # Function to handle city button hover event
102 > def on_city_hover(event): ···
113
114 # Function to handle mouse hover over city button
115 > def on_hover(event, city): ···
126
127 # Function to handle mouse leave event from city button
128 > def on_leave(event): ···
130
131 # Function to create the main window and UI component
132 > def create_main_window(): ···
407
408 if __name__ == "__main__":
409     create_main_window()
410
```

Figure 6



```
main.py   dataload.py
dataload.py > ...
45
46  # City points dictionary
47  city_points = {
48      'I': 'Hansaviertel',
49      'B': 'Rostock-Heide',
50      'F': 'Evershagen',
51      'N': 'Stadtmitte',
52      'J': 'Gartenstadt/Stadtweide',
53      'G': 'Schmarl',
54      'R': 'Dierkow-West',
55      'C': 'Lichtenhagen',
56      'K': 'Kröpeliner-Tor-Vorstadt',
57      'E': 'Lütten Klein',
58      'T': 'Gehlsdorf',
59      'H': 'Reutershagen',
60      'U': 'Rostock-Ost',
61      'P': 'Dierkow-Neu',
62      'M': 'Biestow',
63      'Q': 'Dierkow-Ost',
64      'S': 'Toitenwinkel',
65      'D': 'Groß Klein',
66      'O': 'Brinckmansdorf',
67      'L': 'Südstadt',
68      'A': 'Warnemünde'
69  }
```

Figure 7

```
# Function to read subtopics based on the selected main topic
def read_subtopics(main_topic):
    subtopics = []
    if main_topic in column_ranges:
        column_range = column_ranges[main_topic]
        for col_idx in column_range:
            subtopic = sheet.cell(row=1, column=col_idx).value
            if subtopic:
                subtopics.append(subtopic)
        if main_topic == 'Age' and 'Average Age' in subtopics:
            subtopics.remove('Average Age')
            subtopics.insert(0, 'Average Age')
    return subtopics
```

Figure 8

```
# Function to read data for a specific city and subtopic
def read_data(city_code, main_topic, subtopic):
    data = []
    main_topic_columns = column_ranges.get(main_topic, [])
    if main_topic_columns:
        subtopic_index = None
        for col_idx in main_topic_columns:
            if sheet.cell(row=1, column=col_idx).value == subtopic:
                subtopic_index = col_idx
                break
        if subtopic_index is not None:
            for row in sheet.iter_rows(min_row=2, values_only=True):
                if row[1] == city_code:
                    year = row[0]
                    value = row[subtopic_index - 1]
                    if year in [2016, 2017, 2018, 2019]:
                        data.append((year, value))
    return data
```

Figure 9

```
48
49  # Function to read data for all cities for a specific subtopic
50  def read_data_for_all_cities(main_topic, subtopic):
51      data = {}
52      main_topic_columns = column_ranges.get(main_topic, [])
53      if main_topic_columns:
54          subtopic_index = None
55          for col_idx in main_topic_columns:
56              if sheet.cell(row=1, column=col_idx).value == subtopic:
57                  subtopic_index = col_idx
58                  break
59          if subtopic_index is not None:
60              for row in sheet.iter_rows(min_row=2, values_only=True):
61                  year = row[0]
62                  city_code = row[1]
63                  city_name = row[2]
64                  value = row[subtopic_index - 1]
65                  if year in [2016, 2017, 2018, 2019]:
66                      if city_name not in data:
67                          data[city_name] = []
68                      data[city_name].append((year, value))
69      return data
```

Figure 10

```
71  # Function to get all unique years from the dataset
72  def get_years():
73      years = set()
74      for main_topic in column_ranges.keys():
75          for subtopic in read_subtopics(main_topic):
76              data = read_data_for_all_cities(main_topic, subtopic)
77              for city_data in data.values():
78                  for year, _ in city_data:
79                      years.add(year)
80      return sorted(list(years))
```

Figure 11

```
82  # Function to update the subtopic dropdown based on the selected main topic
83  def update_subtopics(event=None):
84      selected_main_topic = topic_dropdown.get()
85      subtopics = read_subtopics(selected_main_topic)
86      subtopic_dropdown['values'] = subtopics
87      subtopic_dropdown.current(0)
88      update_info_text()
89
```

Figure 12

```
89
90  # Function to update the information text displayed
91  def update_info_text(event=None):
92      selected_year = year_dropdown.get()
93      main_topic = topic_dropdown.get().title()
94      subtopic = subtopic_dropdown.get().title()
95
96      info_text.config(state=tk.NORMAL)
97      info_text.delete(1.0, tk.END)
98      info_text.insert(tk.END, f"{selected_year} | {main_topic} | {subtopic}")
99      info_text.config(state=tk.DISABLED)
100
```

Figure 13

```
101 # Function to handle city button hover event
102 def on_city_hover(event):
103     city_name = event.widget.cget("text")
104     main_topic = topic_dropdown.get()
105     subtopic = subtopic_dropdown.get()
106     year = year_dropdown.get()
107
108     if city_name and main_topic and subtopic and year:
109         data = read_data_for_all_cities(main_topic, subtopic)
110         value = next((v for y, v in data.get(city_name, []) if y == int(year)), 'N/A')
111         tooltip = f"{subtopic} ({year}): {value}"
112         tooltip_label.config(text=tooltip)
113
```

Figure 14

```
# Function to handle mouse hover over city button
def on_hover(event, city):
    main_topic = topic_dropdown.get()
    subtopic = subtopic_dropdown.get()
    selected_year = int(year_dropdown.get())

    if main_topic and subtopic and selected_year:
        data = read_data_for_all_cities(main_topic, subtopic)
        value = next((value for y, value in data.get(city, []) if y == selected_year), 'N/A')
        tooltip = f"{city}: {value}"
        tooltip_label.place(x=event.x_root - window.winfo_rootx() + 10, y=event.y_root - window.winfo_rooty() + 10)
        tooltip_label.config(text=tooltip, bg='yellow', fg='black')
```

Figure 15

```
# Function to handle mouse leave event from city button
def on_leave(event):
    tooltip_label.place_forget()
```

Figure 16

```
# Function to create the main window and UI component
def create_main_window():
    global window, info_text, score_board
    window = tk.Tk()
    window.title("Data Viewer for Structure and Characteristics of Urban Areas in Rostock City")
    window.attributes("-fullscreen", True)  # Set full screen
    window.configure(bg=BG_COLOR)

    def exit_fullscreen(event=None):
        window.attributes("-fullscreen", False)

    window.bind("<Escape>", exit_fullscreen)  # Bind the escape key to exit full-screen mode

    # Control frame for dropdowns and info text
    control_frame = tk.Frame(window, bg=BG_COLOR, padx=20, pady=20)
    control_frame.pack(side=tk.TOP, fill=tk.X)

    # Content frame to hold display and image frames
    content_frame = tk.Frame(window, bg=BG_COLOR)
    content_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

    # Display frame for score board and graphs
    display_frame = tk.Frame(content_frame, bg=SPECIAL_BG_COLOR, bd=5, relief=tk.SUNKEN)
    display_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=20)

    # Image frame for the map
    image_frame = tk.Frame(content_frame, bg=SPECIAL_BG_COLOR, bd=5, relief=tk.SUNKEN)
    image_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=20)

    # Scoreboard for displaying high and low values
    score_board = tk.Frame(display_frame, bg='light grey', border=10, borderwidth=4, highlightthickness=5,  # Thickness of the border
    relief="solid" )

    score_board.pack(side=tk.TOP, fill=tk.BOTH, expand=True, pady=40, padx=80)
```

Figure 17

```
160    # Scoreboard for displaying high and low values
161    score_board = tk.Frame(display_frame, bg='light grey', border=10, borderwidth=4, highlightthickness=5,  # Thickness of the border
162    relief="solid" )
163
164    score_board.pack(side=tk.TOP, fill=tk.BOTH, expand=True, pady=40, padx=80)
165
166    # Scoreboard title label
167    title_label = tk.Label(score_board, text="SCORE BOARD", bg='light green', fg=TEXT_COLOR, font=('Helvetica', 15))
168    title_label.pack(fill=tk.X)
169
170    high_value_label = tk.Label(score_board, text="", bg='maroon', fg='white', font=(FONT_DEFAULT[0], 15))
171    high_value_label.pack(pady=10)
172    high_value_label.pack(fill=tk.X)
173
174    low_value_label = tk.Label(score_board, text="", bg='yellow', fg='black', font=(FONT_DEFAULT[0], 15))
175    low_value_label.pack(pady=10)
176    low_value_label.pack(fill=tk.X)
177
178    # Graph frame for data visualization
179    graph_frame = tk.Frame(display_frame, bg=SPECIAL_BG_COLOR)
180    graph_frame.pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)
181
182    # Labels and dropdowns for user controls
183    city_label = tk.Label(control_frame, text="Select First Area:", bg=BG_COLOR, fg=TEXT_COLOR, font=FONT_BOLD)
184    city_label.grid(row=0, column=0, padx=10, pady=5, sticky='w')
185
186    selected_city1 = tk.StringVar()
187    city_dropdown1 = ttk.Combobox(control_frame, textvariable=selected_city1, values=list(city_names.values()), width=30, font=FONT_DEFAULT)
188    city_dropdown1.grid(row=0, column=1, padx=10, pady=5)
189    city_dropdown1.current(0)
190
191    city_label2 = tk.Label(control_frame, text="Select Second Area:", bg=CONTROL_FRAME_COLOR, fg=TEXT_COLOR2, font=FONT_BOLD)
192    city_label2.grid(row=1, column=0, padx=10, pady=5, sticky='w')
193
```

Figure 18

```
194    selected_city2 = tk.StringVar()
195    city_dropdown2 = ttk.Combobox(control_frame, textvariable=selected_city2, values=['Compare with second city'] + list(city_names.values()), width=30, font=FONT_
196    city_dropdown2.grid(row=1, column=1, padx=10, pady=5)
197    city_dropdown2.current(0)
198
199    style = ttk.Style()
200    style.configure('TCombobox', foreground='black', background=TEXT_COLOR, fieldbackground=TEXT_COLOR, font=FONT_ITALIC)
201
202    main_topic_label = tk.Label(control_frame, text="Select the Main Topic:", bg=CONTROL_FRAME_COLOR, fg=TEXT_COLOR2, font=FONT_BOLD)
203    main_topic_label.grid(row=0, column=2, padx=10, pady=5, sticky='w')
204
205    selected_topic = tk.StringVar()
206    global topic_dropdown
207    topic_dropdown = ttk.Combobox(control_frame, textvariable=selected_topic, width=30, font=FONT_DEFAULT)
208    topic_dropdown.grid(row=0, column=3, padx=10, pady=5)
209    topic_dropdown.bind("<<ComboboxSelected>>", update_subtopics)
210
211    main_topics = list(column_ranges.keys())
212    topic_dropdown['values'] = main_topics
213    topic_dropdown.current(0)
214
215    year_label = tk.Label(control_frame, text="Select the Year:", bg=CONTROL_FRAME_COLOR, fg=TEXT_COLOR2, font=FONT_BOLD)
216    year_label.grid(row=0, column=4, padx=10, pady=5, sticky='w')
217
218    selected_year = tk.StringVar()
219    global year_dropdown
220    year_dropdown = ttk.Combobox(control_frame, textvariable=selected_year, width=30, font=FONT_DEFAULT)
221    year_dropdown.grid(row=0, column=5, padx=10, pady=5)
222    year_dropdown.bind("<<ComboboxSelected>>", update_info_text)
223
224    years = get_years()
225    year_dropdown['values'] = years
226    year_dropdown.current(0)
227
228    subtopic_label = tk.Label(control_frame, text="Select the Subtopic:", bg=CONTROL_FRAME_COLOR, fg=TEXT_COLOR2, font=FONT_BOLD)
229    subtopic_label.grid(row=1, column=2, padx=10, pady=5, sticky='w')
```

Figure 19

```
main.py > create_main_window > load_image
231     selected_subtopic = tk.StringVar()
232     global subtopic_dropdown
233     subtopic_dropdown = ttk.Combobox(control_frame, textvariable=selected_subtopic, values=[], width=30, font=FONT_DEFAULT)
234     subtopic_dropdown.grid(row=1, column=3, padx=10, pady=5)
235     subtopic_dropdown.bind("<<ComboboxSelected>>", update_info_text)
236
237     fig = Figure(figsize=(6, 4), dpi=100)
238     ax = fig.add_subplot(111)
239     canvas = FigureCanvasTkAgg(fig, master=graph_frame)
240     canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True, padx=20, pady=20)
241
242     # Create a frame for the info text and the button
243     info_and_button_frame = tk.Frame(image_frame, bg=SPECIAL_BG_COLOR)
244     info_and_button_frame.pack(side=tk.TOP, fill=tk.X, pady=10)
245
246     info_text = tk.Text(info_and_button_frame, height=1, width=45, wrap=tk.WORD, bg='light green', fg=TEXT_COLOR, font=FONT_BOLD, bd=2, relief=tk.SUNKEN, padx=1
247     info_text.pack(side=tk.LEFT, fill=tk.X, expand=True)
248     info_text.insert(tk.END, "")
249     info_text.config(state=tk.DISABLED)
250
251     go_to_graph_button = tk.Button(info_and_button_frame, text="Go to Graph", command=lambda: show_all_cities_data(), bg=SECONDARY_COLOR, fg="white", font=FONT_
252     go_to_graph_button.pack(side=tk.RIGHT, padx=10)
253
254     def load_image():
255         try:
256             image_path = "Rostock.jpg"
257             if os.path.exists(image_path):
258                 image = Image.open(image_path)
259                 image = image.resize((600, 600), Image.LANCZOS)
260                 tk_image = ImageTk.PhotoImage(image)
261                 image_label = tk.Label(image_frame, image=tk_image)
262                 image_label.image = tk_image
263                 image_label.pack()
264
265                 image_label.pack(expand=True, fill="both")
266
```

Figure 20

```
264
265                 image_label.pack(expand=True, fill="both")
266
267                 for point, city in zip(map_points, city_names_ordered):
268                     pin = tk.Label(image_frame, text='📍', bg='maroon', fg='white')
269                     pin.place(x=point[0], y=point[1])
270                     pin.bind("<Enter>", lambda event, c=city: on_hover(event, c))
271                     pin.bind("<Leave>", on_leave)
272
273             else:
274                 print(f"Error: Image file '{image_path}' not found.")
275         except Exception as e:
276             print(f"Error: {e}")
277
278     load_image()
279     update_subtopics()
280
281     def show_data():
282         city_code1 = next((key for key, value in city_names.items() if value == selected_city1.get()), None)
283         city_code2 = next((key for key, value in city_names.items() if value == selected_city2.get()), None) if selected_city2.get() != 'Compare with second ci
284         main_topic = topic_dropdown.get()
285         subtopic = subtopic_dropdown.get()
286         selected_year = int(year_dropdown.get())
287
288         high_value_label.config(text="")
289         low_value_label.config(text="")
290         ax.clear()
291
292         if city_code1 and main_topic and subtopic:
293             data1 = read_data(city_code1, main_topic, subtopic)
294             data2 = read_data(city_code2, main_topic, subtopic) if city_code2 else []
295
296             years1 = [year for year, value in data1]
297             values1 = [value for year, value in data1]
298             years2 = [year for year, value in data2] if city_code2 else []
299             values2 = [value for year, value in data2] if city_code2 else []
```

Figure 21

```
main.py > create_main_window > show_data
            data_all_cities = read_data_for_all_cities(main_topic, subtopic)
            values_for_year = {city: next((value for y, value in city_data if y == selected_year), None) for city, city_data in data_all_cities.items()}
            values_for_year = {city: value for city, value in values_for_year.items() if value is not None}

            if values_for_year:
                max_city = max(values_for_year, key=values_for_year.get)
                min_city = min(values_for_year, key=values_for_year.get)

                max_city_value = values_for_year[max_city]
                min_city_value = values_for_year[min_city]

                high_value_label.config(text=f"Highest Value: {max_city}; {max_city_value}")
                low_value_label.config(text=f"Lowest Value: {min_city}; {min_city_value}")

            bar_width = 0.3
            bar_positions1 = np.arange(len(years1)) * 2
            bar_positions2 = bar_positions1 + bar_width

            ax.bar(bar_positions1, values1, color=SECONDARY_COLOR, alpha=0.7, width=bar_width, label=city_names[city_code1])
            if city_code2:
                ax.bar(bar_positions2, values2, color=PRIMARY_COLOR, alpha=0.7, width=bar_width, label=city_names[city_code2])

            if years1 and values1:
                ax.plot(bar_positions1, values1, marker='o', color='b', linestyle='-', label=f"{city_names[city_code1]} Trend", markersize=2)
            if years2 and values2:
                ax.plot(bar_positions2, values2, marker='o', color='g', linestyle='-', label=f"{city_names[city_code2]} Trend", markersize=2)

            offset = 0.02 * max(max(values1, default=0), max(values2, default=0))
            for i, v in enumerate(values1):
                ax.text(bar_positions1[i], v + offset, str(v), ha='center', va='bottom', fontsize=10)
            if city_code2:
                for i, v in enumerate(values2):
                    ax.text(bar_positions2[i], v + offset, str(v), ha='center', va='bottom', fontsize=10)

            ax.yaxis.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)
```

Figure 22

```
                ax.set_title(f"{main_topic}: {subtopic}")
                ax.set_xlabel("Year")
                ax.set_ylabel(subtopic)
                ax.set_xticks(bar_positions1 + bar_width / 2)
                ax.set_xticklabels([str(year) for year in years1])
                ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))

                fig.tight_layout()

                canvas.draw()
            else:
                if not city_code1:
                    high_value_label.config(text="Please select City 1.")
                if not main_topic:
                    low_value_label.config(text="Please select a Main Topic.")
                if not subtopic:
                    low_value_label.config(text="Please select a Subtopic.")

        def show_all_cities_data():
            main_topic = topic_dropdown.get()
            subtopic = subtopic_dropdown.get()
            selected_year = int(year_dropdown.get())
            data = read_data_for_all_cities(main_topic, subtopic)

            if data:
                all_cities_window = tk.Toplevel(window)
                all_cities_window.title(f"All Cities Data for {subtopic} in {selected_year}")
                all_cities_window.geometry("1200x800")
                all_cities_window.configure(bg=BG_COLOR)

                fig_all = Figure(figsize=(10, 6), dpi=100)
                ax_all = fig_all.add_subplot(111)
                canvas_all = FigureCanvasTkAgg(fig_all, master=all_cities_window)
                canvas_all.get_tk_widget().pack(fill=tk.BOTH, expand=True, padx=20, pady=20)
```

Figure 23

```
            cities = list(data.keys())
            bar_width = 0.5
            bar_positions = np.arange(len(cities))

            values = [next((value for y, value in data[city_name] if y == selected_year), 'N/A') for city_name in cities]
            bars = ax_all.bar(bar_positions, values, width=bar_width, color=SECONDARY_COLOR, alpha=0.7)

            offset = 0.02 * max(max(values, default=0), 0)
            for bar in bars:
                height = bar.get_height()
                ax_all.text(bar.get_x() + bar.get_width() / 2, height + offset, str(height), ha='center', va='bottom', fontsize=8)

            ax_all.set_title(f"All Cities - {main_topic}: {subtopic} ({selected_year})")
            ax_all.set_xlabel("Cities")
            ax_all.set_ylabel(subtopic)
            ax_all.set_xticks(bar_positions)
            ax_all.set_xticklabels(cities, rotation=45, ha="right")
            ax_all.legend(loc='center left', bbox_to_anchor=(1, 0.5))

            ax_all.yaxis.grid(True, linestyle='--', linewidth=0.7, alpha=0.7)

            fig_all.tight_layout()

            canvas_all.draw()
        else:
            high_value_label.config(text="No data found for any city.")

    show_data_button = tk.Button(control_frame, text="Show Data", command=show_data, bg=PRIMARY_COLOR, fg="white", width=20, font=FONT_BOLD)
    show_data_button.grid(row=2, column=2, padx=10, pady=10, sticky='e')

    global tooltip_label
    tooltip_label = tk.Label(window, text="", bg=SPECIAL_BG_COLOR, fg=TEXT_COLOR, font=FONT_DEFAULT)
    tooltip_label.pack_forget()

    window.mainloop()
```

Figure 24

python Copy code

```
file_path = "Reports/Rostock.xlsx"
wb = openpyxl.load_workbook(file_path)
sheet = wb.active
```

Figure 25

python Copy code

```
column_ranges = {
    'Age': range(4, 27),
    'Overall Movement': range(27, 30),
    'Natural Movement': range(30, 37),
    'Spatial Movement': range(37, 43),
    'Marital Status': range(43, 47),
    'Gender': range(47, 51),
    'Household Structure': range(51, 56),
    'Overall': range(56, 58),
    'Nationality': range(58, 60)
}
```

Figure 26

```python
map_points = [
    (200, 550), # Hansaviertel
    (450, 180), # Rostock-Heide
    (100, 430), # Evershagen
    (300, 550), # Stadtmitte
    (155, 570), # Gartenstadt/Stadtweide
    (175, 400), # Schmarl
    (320, 480), # Dierkow-West
    (100, 330), # Lichtenhagen
    (250, 540), # Kröpeliner-Tor-Vorstadt
    (108, 375), # Lütten Klein
    (250, 480), # Gehlsdorf
    (170, 500), # Reutershagen
    (300, 330), # Rostock-Ost
    (370, 465), # Dierkow-Neu
    (200, 630), # Biestow
    (350, 500), # Dierkow-Ost
    (300, 430), # Toitenwinkel
    (160, 350), # Groß Klein
    (380, 540), # Brinckmansdorf
    (250, 590), # Südstadt
    (180, 300)  # Warnemünde
]
```

Figure 27

```python
city_points = {
    'I': 'Hansaviertel',
    'B': 'Rostock-Heide',
    'F': 'Evershagen',
    'N': 'Stadtmitte',
    'J': 'Gartenstadt/Stadtweide',
    'G': 'Schmarl',
    'R': 'Dierkow-West',
    'C': 'Lichtenhagen',
    'K': 'Kröpeliner-Tor-Vorstadt',
    'E': 'Lütten Klein',
    'T': 'Gehlsdorf',
    'H': 'Reutershagen',
    'U': 'Rostock-Ost',
    'P': 'Dierkow-Neu',
    'M': 'Biestow',
    'Q': 'Dierkow-Ost',
    'S': 'Toitenwinkel',
    'D': 'Groß Klein',
    'O': 'Brinckmansdorf',
    'L': 'Südstadt',
    'A': 'Warnemünde'
}
```

Figure 28

```python
def read_cities_from_excel():
    cities = {}
    for row in sheet.iter_rows(min_row=2, values_only=True):
        city_code = row[1]
        city_name = row[2]
        cities[city_code] = city_name
    sorted_cities = {k: v for k, v in sorted(cities.items(), key=lambda item: item[1])}
    return sorted_cities
```

Figure 29

```python
global city_names
city_names = read_cities_from_excel()
```