**Task 8) Documentation for Electric Vehicle (EV) Charging Station Visualization Program**

**I. Program Structure**

The program is designed to process and visualize data related to electric vehicle charging stations and population distribution in Berlin. The final goal is to generate a **Streamlit application** to identify areas requiring additional EV charging stations.

The program's workflow is organized into the following steps:

1. **Load Data**:
   * Berlin geodata by postal codes (PLZ).
   * EV charging station data.
   * Population/resident data.
2. **Data Preprocessing**:
   * Process and merge data to create meaningful relationships.
   * Count charging stations by postal code (PLZ).
3. **Visualization**:
   * Generate an interactive Streamlit app for visual exploration of data.

**II. Key Components**

1. **Imports**:
   * pandas: For data manipulation.
   * core.methods: A custom module containing data processing methods.
   * core.HelperTools: A utility module providing helper functions, such as a timer.
   * config.pdict: Configuration dictionary storing filenames and other parameters.
2. **Functions**:
   * @ht.timer: A decorator from HelperTools used to measure execution time for the main() function.
   * main(): The main driver function that performs data loading, preprocessing, and app generation.
3. **Dataset Requirements**:
   * **Geodata**: Provides spatial data for Berlin postal codes.
   * **Charging Stations Data**: Information about existing EV charging stations.
   * **Residents Data**: Population/resident data by postal code.
4. **Error Handling**:
   * Each step includes try-except blocks to handle and report errors during execution.

#### III. Code Walkthrough

1. **Load Geodata**:
   * The geodata file (file\_geodat\_plz) is read using pd.read\_csv().
   * Purpose: Define Berlin's postal code regions for mapping and data aggregation.
2. **Load Charging Stations Data**:
   * Reads the charging stations dataset (file\_lstations) using pd.read\_csv().
   * This dataset contains the locations of EV charging stations.
3. **Preprocess Charging Stations Data**:
   * Uses m1.preprop\_lstat() to clean, transform, and integrate charging station data with geodata.
   * Prepares data for aggregation and visualization.
4. **Count Charging Stations per PLZ**:
   * Uses m1.count\_plz\_occurrences() to count the number of charging stations in each postal code.
5. **Load Residents Data**:
   * Reads the residents' dataset (file\_residents) using pd.read\_csv().
   * Contains information about population distribution by postal code.
6. **Preprocess Residents Data**:
   * Uses m1.preprop\_resid() to clean and align population data with geodata.
7. **Generate Streamlit App**:
   * Combines processed charging stations and population data to create a visual and interactive Streamlit app.
   * Uses m1.make\_streamlit\_electric\_Charging\_resid() to generate the app.

#### IV. Key Methods and Their Roles

* **m1.preprop\_lstat(df\_lstat, df\_geodat\_plz, pdict)**:
  + Cleans and integrates charging stations data with Berlin's geodata.
* **m1.count\_plz\_occurrences(df\_lstat2)**:
  + Aggregates the number of charging stations for each postal code.
* **m1.preprop\_resid(df\_residents, df\_geodat\_plz, pdict)**:
  + Cleans and aligns residents data with Berlin's postal codes.
* **m1.make\_streamlit\_electric\_Charging\_resid(gdf\_lstat3, gdf\_residents2)**:
  + Generates a Streamlit-based visualization app combining charging stations and population data.
* **ht.timer**:
  + A utility decorator that logs the execution time of the main() function.

#### V. Configuration (config.pdict)

The pdict dictionary stores file paths and other configuration parameters:

* file\_geodat\_plz: Filename for Berlin's postal code geodata.
* file\_lstations: Filename for charging stations data.
* file\_residents: Filename for population data.

#### VI. Output

The program outputs:

1. **Command-line Messages**:
   * Status updates for data loading, preprocessing, and app generation.
   * Error messages in case of failures.
2. **Streamlit App**:
   * Visualizes postal code regions, charging station density, and population distribution.
   * Provides insights into areas requiring additional EV charging stations.

### **Documentation for Utility Functions and Framework**

This script defines a series of utility functions and methods for tasks such as data processing, serialization, mathematical computations, and randomization. The goal is to provide reusable tools for managing and analyzing datasets, including working with DataFrames, dictionaries, and lists.

**I. Overview of Modules Imported**

1. **Standard Libraries**:
   * math: For mathematical operations such as factorials.
   * random: For random value generation.
   * pickle: For serialization and deserialization of objects.
   * time and functools: For timing function execution and working with decorators.
   * collections:
     + Counter: For counting occurrences of elements in a collection.
     + OrderedDict: For maintaining the order of dictionary items.
2. **Third-Party Libraries**:
   * pandas: For handling tabular data in DataFrames.

**II. Core Functions and Features**

#### ****a. Timing Decorator****

def timer(func):

"""Print the runtime of the decorated function"""

* **Purpose**: Measures and prints the execution time of the decorated function.
* **Usage**: Wrap any function with @timer to log its runtime.

#### ****b. Predicates****

* **isElFilled(el, liste)**: Checks if an element exists in a list and is not None.
* **validateIndex (lambda)**: Checks if a DataFrame has no duplicate rows.
  + Returns True if no duplicates exist, otherwise False.

#### ****c. Serialization****

@timer

def pickle\_out(objName, dateiName):

"""Serialization"""

* **pickle\_out(objName, dateiName)**: Saves a Python object to a file using pickle.
* **pickle\_in(dateiName)**: Loads a Python object from a file using pickle.

#### ****d. Data Processing****

1. **col\_base\_features(col, pattern)**:
   * Splits a column's string values by a pattern and extracts the first part.
2. **determine\_dyn\_colorder(colvals, colorder\_fixedpart, pdict)**:
   * Dynamically adjusts column order by removing unnecessary elements and appending the fixed part to the remaining columns.
3. **cleanse\_colnames(dfcn, zeichen)**:
   * Removes specified characters (zeichen) from DataFrame column names.
4. **countFreqs(arr)**:
   * Counts frequencies of elements in a list and returns an OrderedDict.

#### ****e. Data Cleaning****

1. **List Cleaning**:
   * **remNanFromListFloat(x)**: Removes NaN values from a list.
   * **remNullItemsFromList(x)**: Removes None values from a list.
2. **Dictionary Cleaning:**
   * **remNanFromDict(d)**: Removes keys with NaN values from a dictionary.
   * **remNullItemsFromDict(d)**: Removes keys with None values from a dictionary.

#### ****f. Math and Combinatorics****

1. **Set Operations**:
   * **intersect(x, y)**: Finds the intersection of two lists.
2. **Binomial Coefficient**:
   * **binom(n, k)**: Calculates the binomial coefficient C(n,k)=n!k!(n−k)!C(n, k)

#### ****g. Randomization****

1. **getRandomColor(\_)**:
   * Generates a random hex color code.

#### ****h. DataFrame Operations****

1. **popRowFromDF(dframe, indexVal)**:
   * Removes a row from a DataFrame by index and returns the row as a list and the updated DataFrame.
2. **sortDF(dframe, col, asc)**:
   * Sorts a DataFrame by a specified column in ascending (asc=True) or descending (asc=False) order. Returns a new DataFrame with sorted rows.

#### ****i. Column Operations****

1. **df\_cols\_assign\_alias(x, y)**:
   * Renames DataFrame columns based on a mapping provided in pdict (parameter dictionary).
   * Mimics the SQL "AS" aliasing functionality.

**III. Key Lambda Functions**

1. **Splitting and Formatting**:
   * **lam\_split**: Extracts the second part of a string split by $.
   * **tupToStr**: Converts a tuple to a formatted string, e.g., (1, "A") -> "1. A".
2. **Sorting Dictionaries**:
   * **sortDictReverseOrderIntKey**: Sorts a dictionary in reverse order by integer keys.
3. **Feature Extraction**:
   * **ohlist\_To\_FeaturesList**: Extracts the unique prefix of each string in a list, split by $.

### **Documentation: Geospatial Data Preprocessing and Visualization Functions**

This following explains the functions provided for geospatial data processing, preprocessing, and visualization of datasets. These functions focus on electric charging stations and residential data with postal codes (PLZ) in Germany.

### **I. Imports**

* **geopandas (gpd)**: For geospatial data handling.
* **core.HelperTools (ht)**: Utility tools (e.g., @ht.timer for runtime logging).
* **folium**: For creating interactive maps.
* **streamlit (st)**: To build interactive web applications.
* **streamlit\_folium (folium\_static)**: To embed Folium maps into Streamlit apps.
* **branca.colormap (LinearColormap)**: For custom color mapping in visualizations.

### **II. Core Functions**

#### 1. sort\_by\_plz\_add\_geometry

Prepares and merges dataframes by postal codes (PLZ) and adds geospatial geometry.

**Parameters:**

* dfr: Pandas DataFrame containing the primary dataset.
* dfg: Geopandas DataFrame containing geospatial data.
* pdict: Dictionary containing column mappings (e.g., pdict["geocode"] for the key to merge on).

**Returns:**

* A GeoDataFrame with sorted data and valid geometries.

#### 2. preprop\_lstat

Preprocesses a dataset of charging stations and filters rows based on geographic and postal code constraints.

**Parameters:**

* dfr: Pandas DataFrame from Ladesaeulenregister.csv.
* dfg: Geospatial DataFrame.
* pdict: Column mapping dictionary.

**Returns:**

* GeoDataFrame of filtered and geospatially processed charging station data.

**Steps:**

1. Filters relevant columns (PLZ, Bundesland, Breitengrad, Längengrad, KW).
2. Converts latitude and longitude values to a proper decimal format.
3. Filters rows for Berlin with postal codes between 10115 and 14200.
4. Merges with geospatial data and ensures valid geometries.

#### 3. count\_plz\_occurrences

Counts the occurrences of charging stations per postal code (PLZ).

**Parameters:**

* df\_lstat2: GeoDataFrame of charging station data.

**Returns:**

* DataFrame with:
  + PLZ: Postal codes.
  + Number: Count of charging stations.
  + geometry: Geometry of the region.

#### 4. preprop\_resid

Preprocesses resident population data by postal codes and merges with geospatial data.

**Parameters:**

* dfr: Pandas DataFrame from plz\_einwohner.csv.
* dfg: Geospatial DataFrame.
* pdict: Column mapping dictionary.

**Returns:**

* GeoDataFrame of residents data with valid geometries.

**Steps:**

1. Filters relevant columns (PLZ, Einwohner, Breitengrad, Längengrad).
2. Converts latitude and longitude to decimal format.
3. Filters rows for postal codes between 10000 and 14200.
4. Merges with geospatial data.

#### 5. make\_streamlit\_electric\_Charging\_resid

Creates an interactive Streamlit app for visualizing heatmaps of:

* Charging stations.
* Resident population.

**Parameters:**

* dfr1: GeoDataFrame with charging station data.
* dfr2: GeoDataFrame with resident population data.

**Returns:**

* A Streamlit app rendering heatmaps.

**Features:**

1. **Map Layers:**
   * Heatmap for charging stations (based on Number).
   * Heatmap for residents (based on Einwohner).
2. **Color Maps:**
   * Gradation from yellow to red to depict density.
3. **Interactivity:**
   * Users can toggle between layers using radio buttons.
4. **Integration:**
   * Embeds Folium maps into Streamlit via folium\_static.

**Steps:**

1. Initializes a Folium map centered on Berlin.
2. Adds layers for residents and charging stations based on user selection.
3. Applies color maps for density visualization.
4. Displays the map in the Streamlit app.

**Interpretation of Data and Findings on EV Charging Stations in Berlin**

The analysis conducted for the distribution of residents and EV charging stations in Berlin highlights key insights regarding infrastructure adequacy and potential areas for development.

**1. Residents' Distribution**

The heatmap derived from the data reflects the population distribution across Berlin. The code utilizes Berlin's postal code regions and assigns population density data to each region using geospatial processing libraries such as GeoPandas.

The distribution of Berlin's population shows higher concentrations in districts like Pankow, Mitte, often exceeding 30,000 people in specific neighborhoods. On the other hand, areas in the southern and western parts of Berlin tend to have fewer residents, reflecting a more suburban or less populated character.

**Key Insight:** The variation in population density indicates areas where public services, including EV charging infrastructure, might see higher or lower usage demand.

#### ****2. EV Charging Stations Distribution****

The second heatmap visualizes the locations of EV charging stations across Berlin, generated by overlaying the geospatial points of existing stations.

The heatmap for EV charging stations reveals that they are primarily clustered in central Berlin, particularly in areas of higher population concentrations such as Mitte and its surrounding neighborhoods. The largest hubs host up to 95 charging stations, with a secondary concentration of 82 stations in east Berlin districts.

**Key Insight:** The current placement of charging stations focuses heavily on central areas, which aligns with higher population and traffic volumes. However, the disparity in station density across the city suggests gaps in infrastructure planning.

#### ****3. Gap Analysis and Demand for Additional Charging Stations****

By comparing the residents' distribution with the charging station heatmap, notable gaps are evident:

* **Northern Berlin:** While Pankow has the highest population, the availability of EV charging stations in this district does not seem proportionate to the demand implied by the population. Additional stations in Pankow's neighborhoods could address this discrepancy and improve accessibility.
* **Southern Berlin:** This region has a sparse distribution of charging stations and moderate population levels, indicating an underserved area that would benefit from more infrastructure.
* **Western Berlin:** Despite lower population levels, areas could benefit from a greater number of stations to encourage EV adoption and ensure equitable infrastructure distribution.
* Certain transit corridors or commercial zones might show moderate station density despite low population density. These locations could be strategically selected for their traffic patterns or utility for long-distance travelers.

#### 4. ****Recommendations****

* **Address High Population Gaps:** Increase station density in populous districts with relatively few stations to match resident needs and future EV adoption rates.
* **Equitable Distribution:** Address further underserved areas in west and southern Berlin to ensure EV owners in these regions have reasonable access to charging infrastructure.
* **Strategic Placement:** Ensure charging stations are not only placed in densely populated areas but also along transit routes and near commercial or public spaces to maximize utility.

#### 5. Conclusion

Berlin’s current distribution of EV charging stations shows significant progress in central areas but reveals gaps in coverage for densely populated or underserved regions. A targeted approach to expand charging infrastructure based on these insights can help Berlin advance toward a sustainable and accessible EV ecosystem for its residents.