Before running any of the scripts, the first step is to ensure that the **TPTP dataset** is downloaded and placed inside a folder named **TPTP** within the **myproject_with_data** directory. This dataset contains the problem and axiom files required for extraction. Once the data is in place, the process begins with **extract_formulas.py**, which scans the **TPTP** folder, extracts FOF and CNF formulas, and saves them in a structured format inside a newly created **formulas** directory. After this, both **extract_data.py** and **extract_data_with_id.py** should be executed to extract all predicates, constants, and variables from the formulas and store them in a separate directory. Once the data is extracted, various analysis scripts can be run: **statistics.py** analyzes word segmentation patterns, **symbols.py** identifies unique special symbols, **symbols_in_name.py** finds occurrences of symbols within names, and **symbols_in_problem.py** examines special symbols linked to specific problems. This sequence ensures that the data is fully processed and ready for further analysis.

The **extract_data_with_id.py** script is responsible for extracting predicates, constants, and variables from JSON files that contain logical formulas. It processes each formula using regular expressions to identify different types of symbols, ensuring that quoted constants are handled separately. Additionally, it skips the first two constants to refine the data output. The extracted data is then stored in a new directory with modified filenames for further processing.

The **extract_formulas.py** script scans directories containing problem and axiom files and extracts formulas in First-Order Form (FOF) and Conjunctive Normal Form (CNF). Using regex, it identifies these formulas within `.p` and `.ax` files. It organizes them by domain and stores the results in JSON format. The script ensures that only relevant files are processed and logs errors when necessary.

The **statistics.py** script is designed to analyze word segmentation within extracted symbols such as predicates, constants, and variables. It categorizes words based on structural patterns like underscores, camelCase, and concatenation. The script generates statistical insights into these structures and visualizes them using pie charts and bar graphs. Additionally, the categorized results are saved in structured JSON files, along with text reports detailing word classifications.

The **symbols.py** script focuses on identifying special symbols (non-alphanumeric characters) found in extracted data files. It scans through JSON files, applies regex to detect these symbols, and stores them in a set to ensure uniqueness. The script then outputs a sorted list of these symbols for further analysis.

The **symbols_in_name.py** script is an extension of the symbol analysis process. It examines occurrences of special symbols within predicates, constants, and variables. The script maintains a dictionary that tracks where each special symbol appears in different files. It decodes escape sequences to ensure accurate interpretation of names and prints detailed occurrences of symbols for further review.

The **symbols_in_problem.py** script identifies special symbols found in problems stored in structured JSON files. It scans each problem's predicates, constants, and variables to locate

symbols that do not fit standard alphanumeric patterns. The extracted information is then organized into a CSV file, making it easier to analyze occurrences across different problems.

The **test_statistics.py** script serves as a test version of the main statistics analysis. It processes a predefined list of words to evaluate segmentation patterns, categorizing words based on their structure. The script generates statistical insights and visualizations using pie charts and bar graphs. Additionally, it prints any words that do not fit into predefined categories for further investigation.

The **extract_data.py** script, similar to extract_data_with_id.py, processes JSON files containing logical formulas and extracts predicates, constants, and variables using regular expressions. It ensures that quoted constants are handled separately and skips a small portion of unquoted constants to refine the results. The extracted data is then saved in JSON format for further processing in the workflow.