NumberAnalyzer

The **NumberAnalyzer** is a Python-based tool designed to analyze numbers within a specified range using configurable rules. It processes numbers between a start and end value (inclusive) and classifies them into categories based on rules defined in a JSON configuration file. The tool supports built-in checks (e.g., prime, even, odd) as well as custom rules defined as Python functions or lambda expressions.

Features

- Analyze numbers within a user-defined range.
- Classify numbers using:
  - Built-in rules: prime, even, odd.
  - Custom rules: Python functions or lambda expressions.
- Configurable through a JSON file.
- Debugging support with detailed output for each number.

Requirements

- Python 3.6 or higher
- Required modules: `argparse`, `json`, `math`, `os`, `re`, `pytest`, `pytest-html`

Installation

1. Clone the repository or download the script.
2. Ensure Python is installed on your system.
3. Place the configuration file (e.g., `default.json`) in the `config` directory.

Usage

Command-Line Execution

Run the script using the command line:

```
python number_analyzer.py --start <start_number> --end <end_number> --config_file <path_to_config_file>
```

If start and end is not passed as an argument, then the program will ask user to enter the input in the runtime.

Arguments:

- `--start` : The starting number of the range.
- `--end` : The ending number of the range.
- `--config_file` : Path to the JSON configuration file (default: `default.json`).

Example:

```
python number_analyzer.py --start 1 --end 10 --config_file config/rules.json
```

Configuration File Format

The configuration file must be in JSON format and define categories with their corresponding rules. Each rule can be:

1. A built-in rule (`prime`, `even`, `odd`).
2. A lambda expression (e.g., `"lambda x: x % 3 == 0"`).
3. A custom Python function (e.g., `"def custom_rule(x): return x > 5 and x % 2 == 0"`).

Example Configuration File (`rules.json`):

```
{
  "categories": [
    {"label": "prime", "rule": "prime"},
    {"label": "even", "rule": "even"},
    {"label": "odd", "rule": "odd"},
    {"label": "divisible_by_3", "rule": "lambda x: x % 3 == 0"},
    {
      "label": "custom_rule",
      "rule": "def custom_rule(x): return x > 5 and x % 2 == 0"
    }
  ]
}
```

Output

The results of the analysis are categorized and printed to the console. Depending on the debug flag, you can view:

1. **Detailed Output** (`debug=True`): Displays each number and its categories.
    - Example:

      ```
      1: odd
      2: prime, even
      3: prime, odd
      ```

2. **Simplified Output** (`debug=False`): Displays only categories as strings.
    - Example:

      ```
      odd
      prime, even
      prime, odd
      ```

Code Overview

Main Components:

1. **Initialization (`__init__`)**:
    - Validates input parameters.
    - Loads and parses the configuration file.
2. **Validation (`_validate_params`)**:
    - Ensures valid input values and checks for the existence of configuration files.
3. **Configuration Parsing (`_parse_config`)**:
    - Maps rules from the JSON file to corresponding functions.
4. **Number Analysis (`_get_results`)**:
    - Applies rules to each number in the range and stores results.
5. **Prime Check (`_check_prime`)**:
    - Determines whether a number is prime.
6. **Odd/Even Checks (`_check_odd` , `_check_even`)**:
    - Classifies numbers as odd or even.
7. **Result Printing (`print_results`)**:
    - Outputs results in either detailed or simplified format.

Testing

The NumberAnalyzer includes comprehensive tests to verify its functionality using pytest. These tests ensure that the program handles various scenarios correctly, including valid and invalid configurations, parameter validation, and rule execution.

Running Tests

```
pytest --html=report.html --self-contained-html
```

This will generate an HTML report (report.html) summarizing the test results.

Test coverage

The following types of tests are included:

Configuration-Based Tests

- Valid Initialization: Ensures that valid configuration files are loaded correctly.
- Invalid Configurations:
- Missing configuration directory or file.
- Syntax errors in rules or JSON structure.
- Custom Configurations: Verifies that configurations at custom paths or with full method definitions work as expected.

Parameter Validation Tests

- Ensures proper handling of invalid input parameters such as:
- Start greater than end.
- Non-integer start or end values.

Functional Tests

- Verifies that individual rules (prime, even, odd) work as expected.
- Ensures correct behavior for large ranges, negative ranges, and edge cases like single-number ranges.

Output-Based Tests

- Validates that results are printed in both detailed and simplified formats.

Fixtures Used

The test suite uses several pytest fixtures for reusable setup logic:

- setup: Creates necessary configuration files for testing and cleans up after tests.
- valid_config_file, invalid_config_file, etc.: Provide paths to specific test configurations.
- default_na: Initializes a default instance of NumberAnalyzer.

Error Handling

The program raises a custom exception, `NumberAnalyzerException`, for various errors such as:

- Invalid input parameters (e.g., non-integer start/end values).
- Missing configuration directory or file.
- Syntax errors in custom rules.
