```
"""
Library of functions that are useful for analyzing plain-text log files.
"""
import re
import sys
import os
import pandas as pd

def main():
    # Get the log file path from the command line
    log_path = get_file_path_from_cmd_line()

    # Investigate the gateway log
    #filtered_records, _ = filter_log_by_regex(log_file, 'sshd', print_summary=True, print_records=True)
    #filtered_records, _ = filter_log_by_regex(log_file, 'invalid user', print_summary=True, print_records=True)
    #filtered_records, _ = filter_log_by_regex(log_file, 'invalid user.*220.195.35.40', print_summary=True, print_records=True)
    #filtered_records, _ = filter_log_by_regex(log_file, 'error', print_summary=True, print_records=True)
    filtered_records, _ = filter_log_by_regex(log_path, 'pam', print_summary=True, print_records=True)

    # Extract data from the gateway log
    filtered_records, extracted_data = filter_log_by_regex(log_path, 'SRC=(.*?) DST=(.*?) LEN=(.*?) ')
    extracted_df = pd.DataFrame(extracted_data, columns=('Source IP', 'Desination IP', 'Length'))
    extracted_df.to_csv('data.csv', index=False)

    pass

def get_file_path_from_cmd_line(param_num=1):
    """Gets a file path from a command line parameter.

    Exits script execution if no file path is specified as a command
    line parameter or the specified path is not for an existing file.

    Args:
        param_num (int): Parameter number from which to look for file path. Defaults to 1.

    Returns:
        str: File path
    """
    # Check whether the command line parameter was provided
    if len(sys.argv) < param_num + 1:
        print(f'Error: Missing log file path expected as command line parameter {param_num}.')
        sys.exit('Script execution aborted')

    # Get the parmeter value and convert it to a full path
    log_path = os.path.abspath(sys.argv[param_num])

    # Check whether the file exists
    if not os.path.isfile(log_path):
        print(f'Error: "{log_path}" is not the path of an existing file.')
        sys.exit('Script execution aborted')

    return log_path

def filter_log_by_regex(log_path, regex, ignore_case=True, print_summary=False, print_records=False):
    """Gets a list of records in a log file that match a specified regex.

```

```
    Args:
        log_path (str): Path of the log file
        regex (str): Regex filter
        ignore_case (bool, optional): Enable case insensitive regex matching. Defaults to True.
        print_summary (bool, optional): Enable printing summary of results. Defaults to False.
        print_records (bool, optional): Enable printing all records that match the regex. Defaults to False.

    Returns:
        (list, list): List of records that match regex, List of tuples of captured data
    """
    # Initalize lists returned by function
    filtered_records = []
    captured_data = []

    # Set the regex search flag for case sensitivity
    search_flags = re.IGNORECASE if ignore_case else 0

    # Iterate the log file line by line
    with open(log_path, 'r') as file:
        for record in file:
            # Check each line for regex match
            match = re.search(regex, record, search_flags)
            if match:
                # Add lines that match to list of filtered records
                filtered_records.append(record[:-1])
                # Check if regex match contains any capture groups
                if match.lastindex:
                    # Add tuple of captured data to captured data list
                    captured_data.append(match.groups())

    # Print all records, if enabled
    if print_records is True:
        print(*filtered_records, sep='\n', end='\n')

    # Print print summary of results, if enabled
    if print_summary is True:
        print(f'The log file contains {len(filtered_records)} records that case-{"in" if ignore_case else ""}sensitive match the regex "{regex}".')

    return (filtered_records, captured_data)

if __name__ == '__main__':
    main()
```