**1) An overview of the function of the code (i.e., what it does and what it can be used for).**
Our code scrapes authors and paper titles from the DBLP dataset. We extract patterns using a Python wrapper for the SPMF library. From these extracted patterns, our code then removes redundant patterns -- we implemented both hierarchical microclustering and one-pass microclustering. Finally, we extract strongest context indicators, representative transactions, and semantically similar patterns.

We can use this code to reproduce the DBLP experiment as described in section 5.1 of the paper. That is, we can take in an author or title pattern and find its strongest context indicators, most representative titles and authors / co-authors, and the most semantically similar title / author patterns.

**2) Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.**
**Prerequisites: the libraries mentioned below (see answer #3), Python3, Java, shell**

**Web Scraping (utils/build_data_from_web.py)**

- Build a CSV file called data/data.csv where each line is a list of comma-separated authors and a single title (aka all the information needed from a single paper).
- These papers are pulled from the following text mining/database-related conferences: (aciids, icdm, sdm, dba, balt, dbsec, dbcrowd, pkdd, kdd, trec, cikm, sigir).
- For each conference, we pull all papers from the most recent 10 events (workshops, talks, submissions, etc)
- For each paper, we stem the titles using the nltk.stem.porter stemmer, remove commas and periods (because we are writing the data as a .csv file), and make all author names lowercase in order to avoid interpreting the same author names with different capitalizations as different authors. We also remove all spaces within a single author name and replace them with underscores (firstname_lastname)
- Usage: python3 utils/build_data_from_web.py. Note that you must run this from CourseProject/, the data/ directory must already exist, and you must be using Python3.

**Utility file: Python wrapper for spmf.jar (utils/frequent_pattern_mining/spmf_python_wrapper.py)**

- Motivation: spmf is a Java library. To make everything run smoothly and to minimize the amount of work the user has to do to run the pattern mining algorithms and set everything up, we wrote a Python wrapper for spmf that runs Java as a subprocess.
    - Java -jar libs/spmf.jar run [alg_name] [input_file] [output_file]

**Frequent pattern mining: (utils/frequent_pattern_mining/build_frequent_patterns.py)**

- Builds frequent pattern files for authors and title terms and caches them to a file.
- Maps all unique author names to non-negative integers and all unique title terms to non-negative integer IDs as well -- this is because of 2 reasons: first of all, it's more efficient to work with numbers rather than strings of a theoretically arbitrary length; also, spmf is a Java library that works with numbers rather than strings. Note that the two

mappings are completely disjoint; in other words, an author name with an ID of 0 has nothing to do with a title term with an ID of 0.
- Runs our Python wrapper for spmf.jar for FP-Close (to mine frequent patterns from author terms) and Clo-Span (to mine sequential patterns from title terms) to generate intermediate patterns we would then parse.
- Files generated
    - data/frequent_author_patterns.txt: Frequent author patterns (via FP-Close), where each line is a pattern. Format: 11391 11393 11392 14928 #SUP: 9
    - data/frequent_title_term_patterns.txt: Frequent title term patterns (via Clo-Span), where each line is a pattern. Format: 4 -1 226 -1 240 -1 #SUP: 32
    - data/author_id_mappings.txt: Mapping from author ID to author name. Format: 0 helun_bu
    - data/title_term_id_mappings.txt: Mapping from title term ID to title term. Format: 0 toward

**Redundant pattern removal: (utils/remove_redundant_patterns.py)**
- Two utility methods for removing redundant patterns:
    - Eliminate redundancy using one-pass microclustering
    - Eliminate redundancy using hierarchical microclustering
- The main method is currently written to support one-pass microclustering -- a threshold can be passed in or the default threshold can be used. The cleaned title term patterns are then written to a file.
- Usage:
    - title_patterns = parse_author_file_into_patterns(FrequentPatternBuilder.TITLE_TERMS_OUTPUT_FILE_PATH)
    - minimal_patterns = find_one_pass_microclustering_patterns(title_patterns, 0.6) # To use one-pass microclustering
    - minimal_patterns = find_hierarchical_microclustering_patterns(title_patterns, 0.6) # To use hierarchical microclustering
    - write_patterns_to_file(MINIMAL_TITLE_TERMS_FILENAME, minimal_patterns)

**Utility file: Cosine similarity (utils/cosine_similarity.py)**
- Compute cosine similarity given two context vectors of the same length. Used as a utility method
- Usage: consine_sim = compute_cosine_similarity(context_vec_1, context_vec_2)

**Utility file: Mutual information computation/caching manager (utils/mutual_information_manager.py)**
- Computes mutual information for two given ordered collections of patterns (which can be author-author patterns, author-title patterns, or title-title patterns) and caches the mutual information values to a file for each (pattern 1 ID, pattern 2 ID) pair, where each pattern ID just corresponds to its index in the ordered collection.
    - Each file begins with a value [0, 3], which corresponds to a mapping: 0 = author-author, 1 = author-title, 2 = title-author, 3 = title-title

        - Example file format:
            - 2
            - 0 0 0.000016
            - 0 1 0.000016
            - 0 2 0.000016
            - 0 3 0.000016
            - 0 4 0.000016
            - 0 5 0.000016
            - 0 6 0.000016
            - 0 7 0.000016
            - 0 8 0.000016
            - 0 9 0.000016
        - Files generated
            - data/author_author_mutual_info_patterns.txt = Mutual information values for each author-author pattern pair, where index_1 <= index_2
            - data/author_title_mutual_info_patterns.txt = Mutual information values for each author-title/title-author pattern pair
            - data/title_title_mutual_info_patterns.txt = Mutual information values for each title-title pattern pair, where index_1 <= index_2
        - Usage:
            - transactions = TransactionsManager("data/data.csv", "data/author_id_mappings.txt", "data/title_term_id_mappings.txt")
            - mutual_info = MutualInformationManager(transactions, True)
            - mutual_info.compute_mutual_information(author_patterns)
- Reads in the mutual information from one of the files described above, depending on what the type of manager (author-author, author-title, or title-title manager), and caches all the mutual information values in a large matrix -- which is triangular for the author-author/title-title cases and rectangular for the author-title case
    - Usage:
    - mutual_info = MutualInformationManager(MutualInformationManager.PatternType.X)
    - mutual_info.read_mutual_information_from_file()
    - mutual_info.get_mutual_information(1, 2) # to get mutual info for patterns 1 and 2
- Used to compute mutual information one time (which is expensive) and to abstract how mutual information is stored and managed into a class

**Utility file: Transaction/paper data parser/manager (utils/transactions_manager.py)**
- Parses and stores author-id mappings, title-term-id mappings, and a list of all papers, which are basically (author set, title-term sequence patterns)
- Utility methods for
    - Compute context models for each paper’s title terms given a list of frequent title patterns
    - Compute context models for each paper’s authors given a list of frequent author patterns

    - Find all transaction IDs, which are represented as paper indices from the list of all papers parsed, that have a title pattern as a sequential subset
    - Find all transaction IDs, which are represented as paper indices from the list of all papers parsed, that have an author pattern as a non-sequential subset
    - Get author name from author ID, title term from author ID, get a paper's author from the paper ID, get a paper's title terms from the paper ID, and get the number of transactions (number of papers)
    - Usage: transactions = TransactionsManager("data/data.csv", "data/author_id_mappings.txt", "data/title_term_id_mappings.txt")

**Pattern annotator: Representative transaction extraction (pattern_annotators/representative_transaction_extractor.py)**

- Extracts representative transactions given a pattern ID, a threshold for the number of representative transactions to extract, and a boolean value describing whether the pattern ID refers to an author pattern or a title-term pattern
    - Usage: python pattern_annotators/representative_transaction_extractor.py [target_id] [k] [is author experiment]
    - Example: python pattern_annotators/representative_transaction_extractor.py 0 3 True
- For an author experiment, the top k most representative titles from the transaction manager are displayed
- For a title experiment, the top k most representative titles and the top k most representative authors from the transaction manager are displayed

**Pattern annotator: Semantically similar pattern extraction (pattern_annotators/semantically_similar_pattern_extractor.py)**

- Extracts semantically similar patterns given a pattern ID, a threshold for the number of representative transactions to extract, and a boolean value describing whether the pattern ID refers to an author pattern or a title-term pattern
    - Usage: python pattern_annotators/semantically_similar_pattern_extractor.py [target_id] [k] [is author experiment]
    - Example: python pattern_annotators/representative_transaction_extractor.py 1 10 False
- For an author experiment, the top k most representative author patterns are displayed
- For a title experiment, the top k most representative title term patterns are displayed

**Pattern annotator: Strongest context indicator extraction (pattern_annotators/strongest_context_indicator_extractor.py)**

- Extracts strongest context indicators given a pattern ID, a threshold for the number of representative transactions to extract, and a boolean value describing whether the pattern ID refers to an author pattern or a title-term pattern
    - Usage: python pattern_annotators/strongest_context_indicator_extractor.py [target_id] [k] [is author experiment]
    - Example: python pattern_annotators/strongest_context_indicator_extractor.py 2 15 True

- For an author experiment, the given pattern is annotated with the top k strongest context indicators -- the top k author patterns and the top k title patterns
- For a title experiment, the given pattern is annotated with the top k strongest context indicators -- the top k title patterns and the top k author patterns

**3) Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run a software, whichever is applicable.**
**Libraries Used**
- urlllib (web scraping from DBLP)
- bs4 (parsing scraped data from DBLP)
- nltk (stemming)
- spmf (Java library to mine frequent patterns -- will be installed as a part of the setup step, see the next section)
- Installation: pip install urllib/bs4/nltk

**Setup and Installation**
NOTE: Our data files are quite large; thus, we put the data/ directory in our .gitignore and we're subsequently not committing any of our data files or our library files to our repository, which is a standard in good software design.
- git clone https://github.com/ElizWang/CourseProject.git # Clones repository
- cd CourseProject # Navigate to repository
- sh setup.sh # Sets up the data/ and libs/ directories and runs all scripts from the preprocessing step. Detailed description below:
    - Creates a data/ directory if one does not already exist
    - Builds a csv file called data/data.csv by scraping DBLP paper submissions from the web
    - Creates a libs/ directory if one does not already exist and pulls spmf as libs/spmf.jar
    - Mines frequent author patterns using FP-Close
    - Mines sequential title term patterns using Clo-Span
    - Removes redundancies from sequential title term patterns using microclustering
    - Computes and caches all mutual-information values between (author pattern, author pattern), (title term pattern, author pattern), and (title term pattern, title term pattern)

**4) Brief description of contribution of each team member in case of a multi-person team.**
We worked together in understanding the paper and formulating pseudocode that we used as our overarching plan. We split the paper into two steps -- preprocessing (which includes data scraping, pattern mining, and similar pattern pruning) and pattern annotation.

We worked as a team by calling each other on Zoom and pair programming. Elizabeth typed, while Steven frequently took remote control in implementing preprocessing, extracting representative transactions, and the extraction of strongest context indicators and semantically similar patterns.

**5) Software demo link:** https://www.youtube.com/watch?v=3v8M0sW3xHc