**A logo with blue text

Description automatically generated**

**AC50002**

**PROGRAMMING LANGUAGES FOR DATA ENGINEERING**

**PYTHON ASSIGNMENT**

**STUDENT ID: 2546338**

**NAME: KINGSLEY ABU IGBANOI**

**https://github.com/kbanz55/Generating-Abbreviation-in-Python.git**

**Comprehensive Report of Abbreviation Generation in Python**

**Introduction**

This report is the Python script that reads a text file, generates abbreviations and has a scoring system used to assess the resulting abbreviations. By comprehending the script's methodology, I aim to uncover insights into how it operates and its possible applications. This script has some set of functions that make it possible to achieve this goal. Using the GitHub link, you can access the code <https://github.com/kbanz55/Generating-Abbreviation-in-Python.git>

**Methodology**

1. **File Reading**

The first step in the script is all about having solid ‘**file\_reading**’ functions. The function reads the file and grabs the content from a specific file (trees.txt). It's got an error output in place to deal with situations where the (trees.txt) file can't be found or if some unexpected issue pops up during the reading file. A second function is also defined in the Python script, **‘read\_score\_card’**. This function pulls out scoring values from a different file (values.txt), making it a modular way to handle scoring systems.

Thirdly, the ‘**read\_lines\_for\_abbrevs**’ function as defined in the Python script, which is crucial for gathering the initial batch of words to make abbreviations. This function goes through all the lines in the text file, making sure the script has a tidy and well-organized dataset for the next steps.

1. **Text Cleaning Function**

Before generating abbreviations, the script in the function makes sure the dataset is all tidy, with no extra characters in the text file. The **‘remove\_non\_alphabets’** function cleans out any characters that are not letters from each word. And then, every alphabet gets turned into uppercase using the ‘**convert\_to\_uppercase’** function to keep things uniform.

For challenging instances involving apostrophes, the apostrophe removal function was used. This function not only removes apostrophes but also ensures that the resulting words are converted into uppercase, maintaining a clean and uniform dataset.

1. **Generating Of Abbreviation**

The core of the script lies in its ability to generate three-letter abbreviations for each word. The **‘generate\_abbreviation’** function creates a list of abbreviations based on the initial letter of the word and subsequent characters. This function employs nested loops to cover all possible combinations while avoiding abbreviations with spaces and preventing the iteration from going all over after it runs through the words at the last alphabet.

1. **Cleaning the Name:**

The input name is further refined by removing non-alphabetic characters and changing the whole thing to uppercase. By ensuring that the name only consists of capital alphabet characters, this purification procedure helps to create a standardized and consistent format for upcoming processing or analysis.

1. **Detailed Abbreviation Generation:**

The input name is further refined by removing non-alphabetic characters and changing the whole thing to uppercase. By ensuring that the name only consists of capital alphabet characters, this purification procedure helps to create a standardized and consistent format for upcoming processing or analysis.

1. **Scoring Abbreviations**

Giving an abbreviation a score entails assessing how closely it resembles the original term. The positions of individual letters are taken into consideration, and predefined values for particular letters are created. Every abbreviation is evaluated according to how closely it follows the structure of the original name, taking into account the letter positions as well as the previously established significance of individual letters. The ensuing scores offer a numerical representation of how well each abbreviation represents the main idea of the original name.

1. **Selection of Best Abbreviation:**

The script finds the best abbreviation for the full name by calculating the score of each generated abbreviation. By using this method, it weighs the scores to determine which abbreviation best represents the full name, enabling an informed choice to be made.

1. **Duplicate Handling:**

Repeated abbreviations are eliminated from a given name as well as from all names combined. By following this process, it is ensured that any abbreviations used for a specific name are specific to that context and that there is no global duplication across names. The methodical approach seeks to increase understanding and guard against misunderstandings caused by abbreviations.

**Output File Generation**

Making an output file to store the results is the last step. The output file contains the original name along with each name's best abbreviation and score. Additionally, duplicate abbreviations are dealt with and reported.

**Conclusion**

In conclusion, the Python script presented shows a comprehensive and systematic approach to abbreviation generation. The methodology encompasses key steps, starting with reading file functions, followed by text cleaning processes to ensure a uniform and tidy dataset. The core of the script lies in its ability to generate three-letter abbreviations for words, leveraging nested loops to cover all possible combinations while avoiding undesirable outcomes.

The scoring system adds a layer of competence to the abbreviation generation process. By calculating relative indexes and employing a scoring algorithm based on a predefined scorecard, the script evaluates each abbreviation intelligently. This approach allows for flexibility and customization, making it adaptable to various scenarios and applications.

The result analysis phase ensures the cleanliness and uniqueness of the generated abbreviations. Detection and removal of duplicates, along with the assignment of scores to each abbreviation, contribute to a well-organized output. The script not only generates abbreviations but also provides insights into the quality and relevance of these abbreviations through the scoring mechanism.

**References**

Lutz, M., 2013. *Learning python: Powerful object-oriented programming*. " O'Reilly Media, Inc.".

Mekterović, I., Brkić, L., Milašinović, B. and Baranović, M., 2020. Building a comprehensive automated programming assessment system. *IEEE Access*, *8*, pp.81154-81172.

Pelli, D.G., Burns, C.W., Farell, B. and Moore, D.C., 2006. Identifying letters. *Vision Research*, *46*(28), pp.4646-4674.

Cokelaer, T., Bansal, M., Bare, C., Bilal, E., Bot, B.M., Neto, E.C., Eduati, F., de la Fuente, A., Gönen, M., Hill, S.M. and Hoff, B., 2015. DREAMTools: a Python package for scoring collaborative challenges. *F1000Research*, *4*.

Summerfield, M., 2010. *Programming in Python 3: a complete introduction to the Python language*. Addison-Wesley Professional.