README file for project 1

1. Name: Ruochen Li, Zeqing Wang
   Uni: rl3292, zw2856

2. List of files: pj1.py, pj1_stop_words.py, stopwords.txt

3. Commands required to run the program:

   ```
   cd ..
   cd zeqing/cs6111/proj1
   python3 -m pip install --upgrade pip
   python3 install numpy
   pip3 install -U scikit-learn
   python3 pj1.py
   ```

4. The general structure of our code and explanation of our major functions are listed below:
   a. Class ‘item’: an object we created to keep track of the required information from a search result (url, title, snippet, relevance)
   b. main(): Prompt the user for the input query, with a while loop which continues to expand the query if the precision score is below 0.9. Inside the loop, it first calls the convert_resp function to get the list of class ‘item’ and precision score and then calls the query_expansion function.
   c. convert_resp(): a function that convert the json response from our request into class ‘item’, and returns the list of ‘item’ and precision score
   d. Query_expansion(): the high-level function that expands the query. The main idea is to use a vectorizer from sklearn library, and fit transform the corpus, then split the corpus into relevant and irrelevant document sets based on the user feedback, and then vectorize both document sets and get the tf-idf vector, then it calculates the query vector and apply rocchio’s algorithm to calculate the new query vector. Finally, it expands the query based on the new query vector
   e. rocchio_algorithm(): The function that calculates the new query vector based on the input relevant vector, irrelevant vector and query vector. The function for the algorithm is the same as we talked about in class, and the Alpha, Beta, Gamma value are set to 1, 0.75, 0.25 based on heuristics (source cited in code comments).
   f. refine_query(): The function that converts the tf-idf vector into a sorted array of terms, and then choose the first two terms to add to our query, the general idea is to use a TfidfTransformer from sklearn, and convert the our new query vector into normalized tf-idf vector matrix. Once we have that, we loop through the entire matrix to get each word’s tf-idf score, and sort them from the highest to the lowest score. Finally, we get the first two words that do not appear in the input query from the sorted word array, and add them to our query

g. External library used:
    i. Numpy: https://numpy.org/doc/stable/
    ii. TfidfVectorizer from sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
    iii. TfidfTransformer from sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html

5. Query_expansion(): The main techniques we have used is tf-idf and rocchio's algorithm, both are talked in class so no need to explain how they work here in detail. The specific implementation is: use the vectorizer from sklearn library, and fit transform the corpus, then split the corpus into relevant and irrelevant document sets based on the user feedback, and then vectorize both document sets and get the tf-idf vector, then it calculates the query vector and apply rocchio's algorithm to calculate the new query vector. Once we have the new query vector, our programs converts it back to words using the tf-idf transformer and use each word's tf-idf score to sort them into a sorted list. Finally, we add the words into the query, the order of the added words is based on the tf-idf score.

6. **Engine ID**: 'd23523bff22df70a8'.
   Google Custom Search Engine JSON API Key:
   'AIzaSyDPE-QprcLMrjwQAktSOGyatbLrJNWi8Gw'

7. How to deal with non-html files: We recognized that from the http response json returned from google, all the non-html files has "mime" in the dictionary to mark their content type, while html files do not have that key, therefore, we just skipped all the results with "mime" as a key, and only counted the html files for precision score.