***Individual Project Report***



***CMPE256: Large Scale Analytics***

***Job Recommendation System***

**Submitted By:**

**Ruchika Hazariwal**

**SJSU ID: 013728923**

**GitHub Link: https://github.com/ruchikahazariwal/Job\_Recommendation\_System**

**Introduction**:

Recommendation systems usually involve exploiting the relations among known features and content that describe items (content-based filtering) or the overlap of similar users who interacted with or rated the target item (collaborative filtering). To combine these two filtering approaches, current model-based hybrid recommendation systems typically require extensive feature engineering to construct a user profile and recommend job to the user. I am planning to implement some of the suggestions implemented in the latest research paper Yang, S., Korayem, M., AlJadda, K., Grainger, T., & Natarajan, S. (2017). Combining content-based and collaborative filtering for job recommendation system: A cost-sensitive Statistical Relational Learning approach. *Knowledge-Based Systems*, *136*, 37-45.

**Data Scraping:**

The job descriptions have been scraped from Linkedin.com The data has been scraped by providing various keywords for technologies and title. However, there is some redundant data, rows with missing values etc. This data has been preprocessed to remove redundancies and missing values to get more reliable output. Unnecessary english stopwords and characters have also been removed to get better recommendations. The job description is also converted to comma separated keywords in order to perform content-based recommendation against user resume. Our recommender system will follow prescriptive data analytics which involves high volume of data and advanced/complex analytical techniques to make correct recommendation.

**Data Preprocessing:**

* Redundancy- We make sure redundant data is not present in our dataset while web scraping unique jobs from the website.
* Missing Values- Few documents have missing attributes, we fill in title in the abstract attribute to make sure data is complete.
* Attribute Selection- We have selected the following columns after preprocessing of data.
* Stemming- Generating variants of root/base words.

**Data Analysis:**

I have followed following step in order to

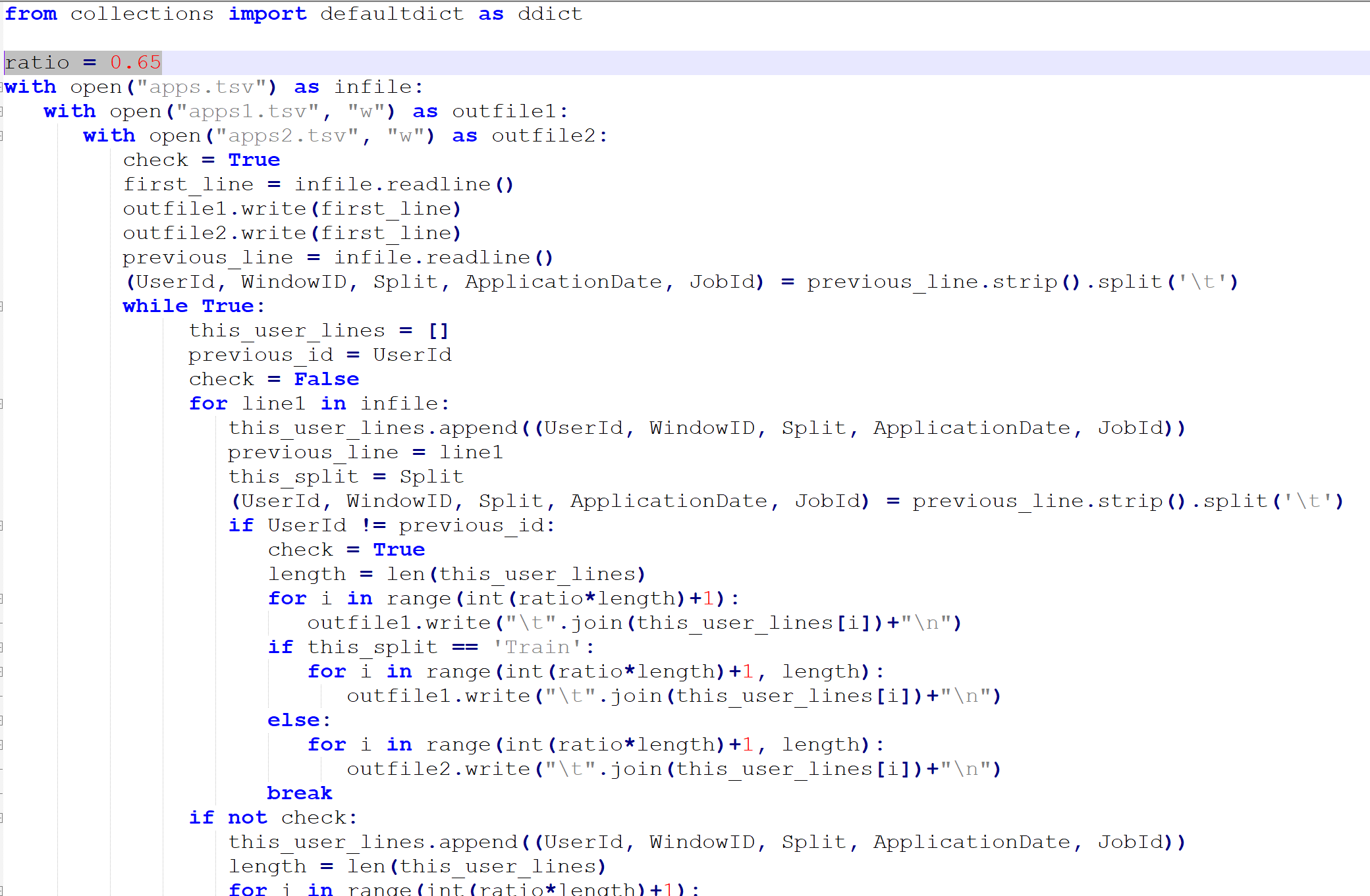
* First, all the skills will be fetched from the LinkedIn’s summary, experience and skills section.
* After, huge volume of jobs is fetched from indeed which covers most of the skills included in the skill set.
* Data is refined by removing stop words such as moreover, of, the etc.
* TF-IDF content-based recommendation system is used to calculate job title similarity,

Skills similarity and degree similarity

* Item based recommendation system is also used to find similarity among the jobs which needs similar skills
* After combining the result of two recommendation system, we can get desirable result and avoid all the shortcomings from our results.

**Data Splitting:**

Data has been divided on the basis of 65:35 split with the first 65% examples in training.



**Approaches:**

I have used two different approaches. First is item-based collaborative filtering and other one is Jaccard similarity to compare.

1. **Item based collaborative filtering:**

In this algorithm, jobs are represented as two vectors that contain the user IDE and jobs ID. The similarity between user ID and job ID is calculated by the cosine of the angle between the two vectors. Matrix of vectors is generated with rows and columns as User ID and job ID. Number represented in a row is matched to the job ID.

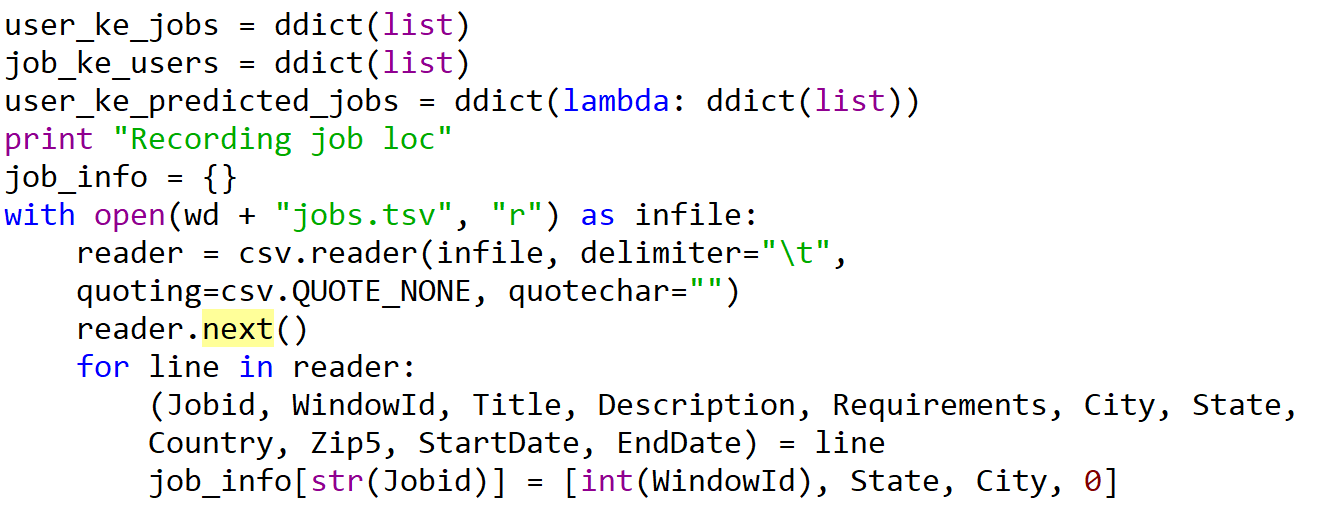
Earlier collaborative filtering systems based on [rating](https://en.wikipedia.org/wiki/Star_(classification)) similarity between users (known as [user-user collaborative filtering](https://en.wikipedia.org/w/index.php?title=User-user_collaborative_filtering&action=edit&redlink=1)) had several problems:

* systems performed poorly when they had many items but comparatively few ratings
* computing similarities between all pairs of users was expensive
* user profiles changed quickly and the entire system model had to be recomputed

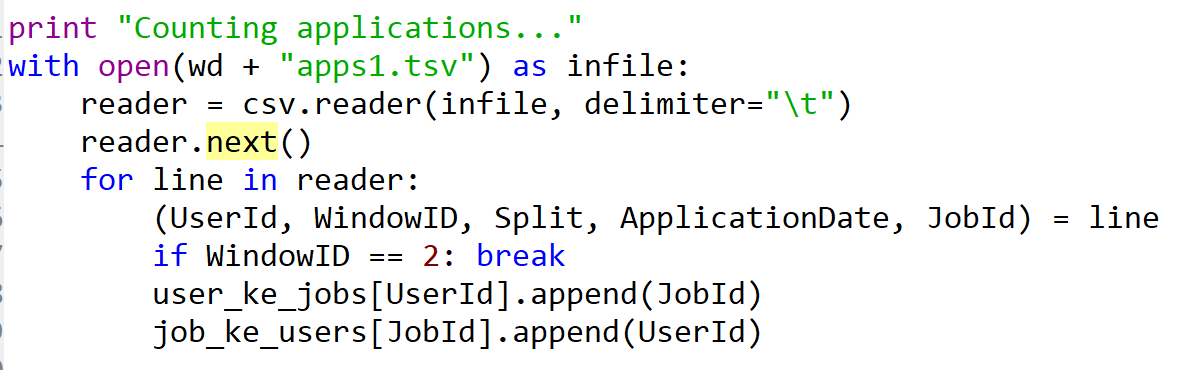
Item-item models resolve these problems in systems that have more users than items. Item-item models use rating distributions *per item*, not *per user*. With more users than items, each item tends to have more ratings than each user, so an item's average rating usually doesn't change quickly. This leads to more stable rating distributions in the model, so the model doesn't have to be rebuilt as often. When users consume and then rate an item, that item's similar items are picked from the existing system model and added to the user's recommendations.

Coding process works as:

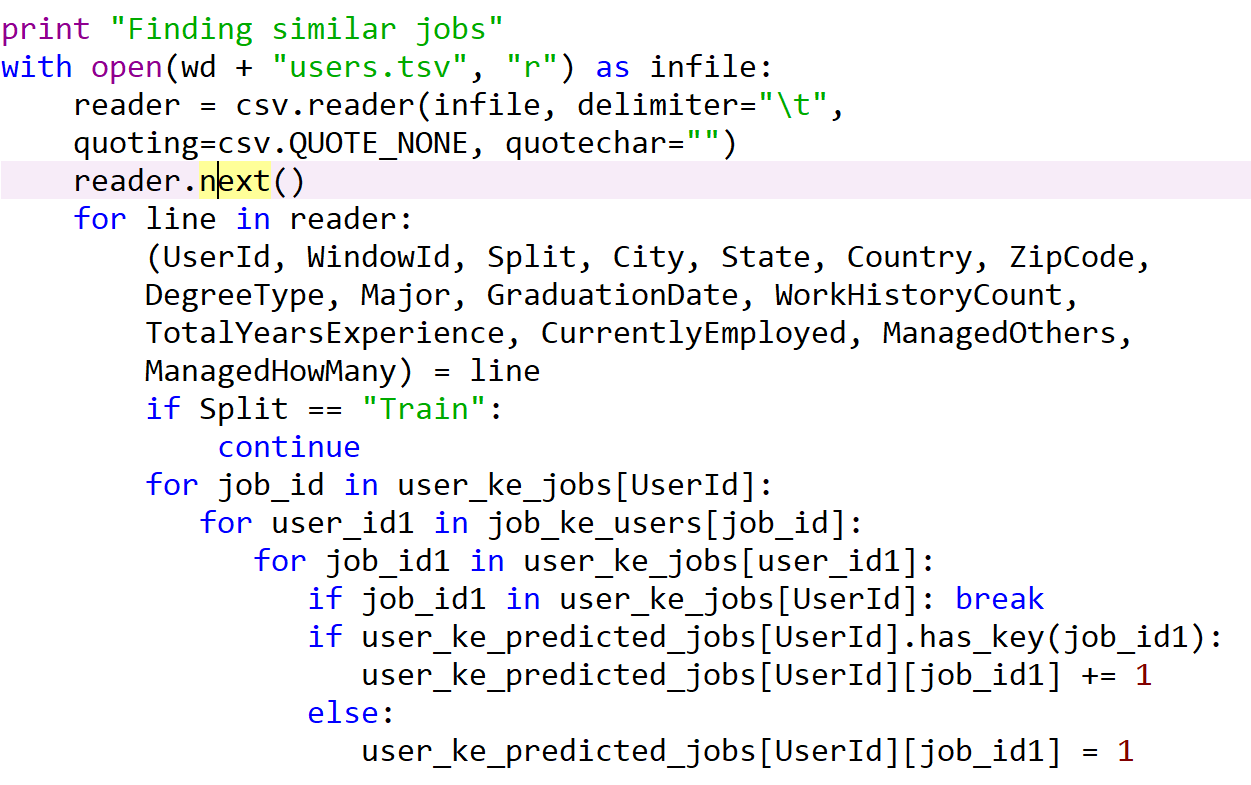
**Step 1:** Record all the job locations



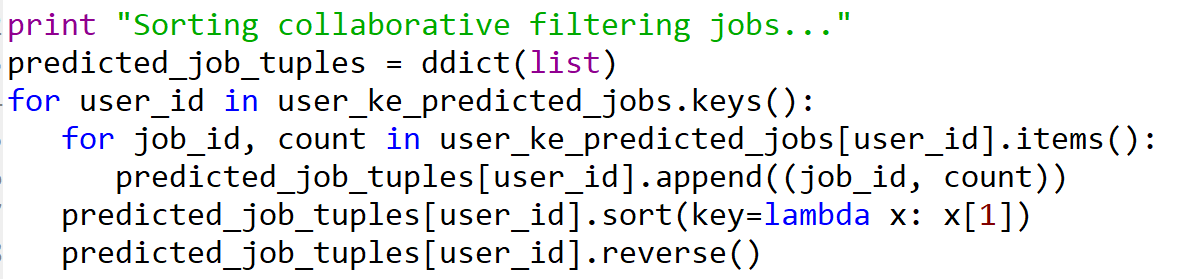
**Step 2:** Count all the applications



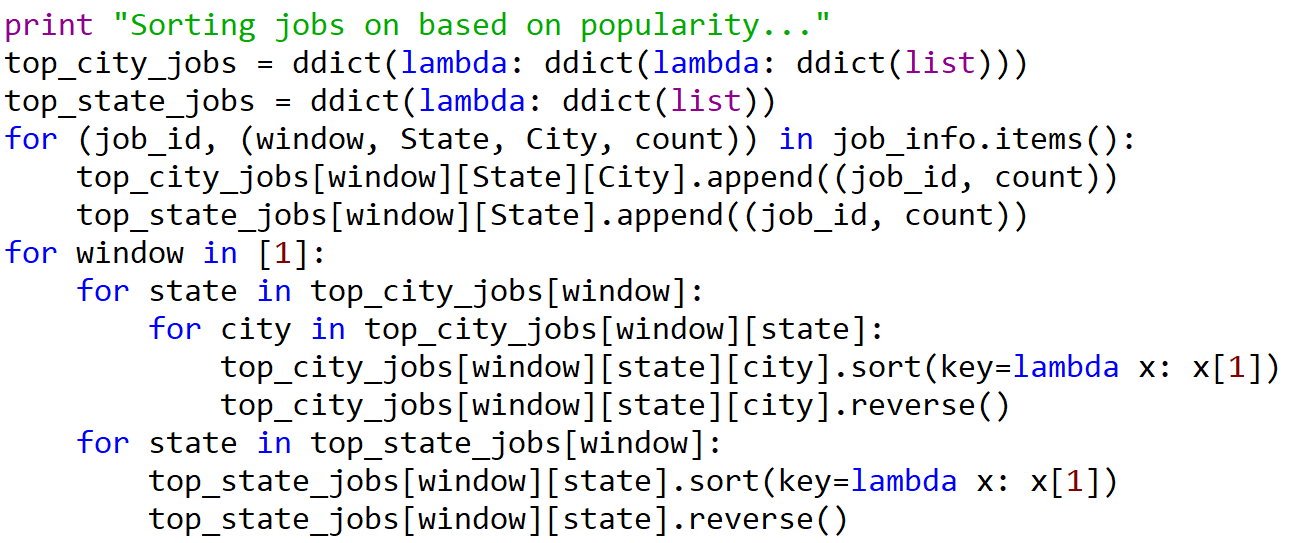
Step 3: Find similar jobs for each user



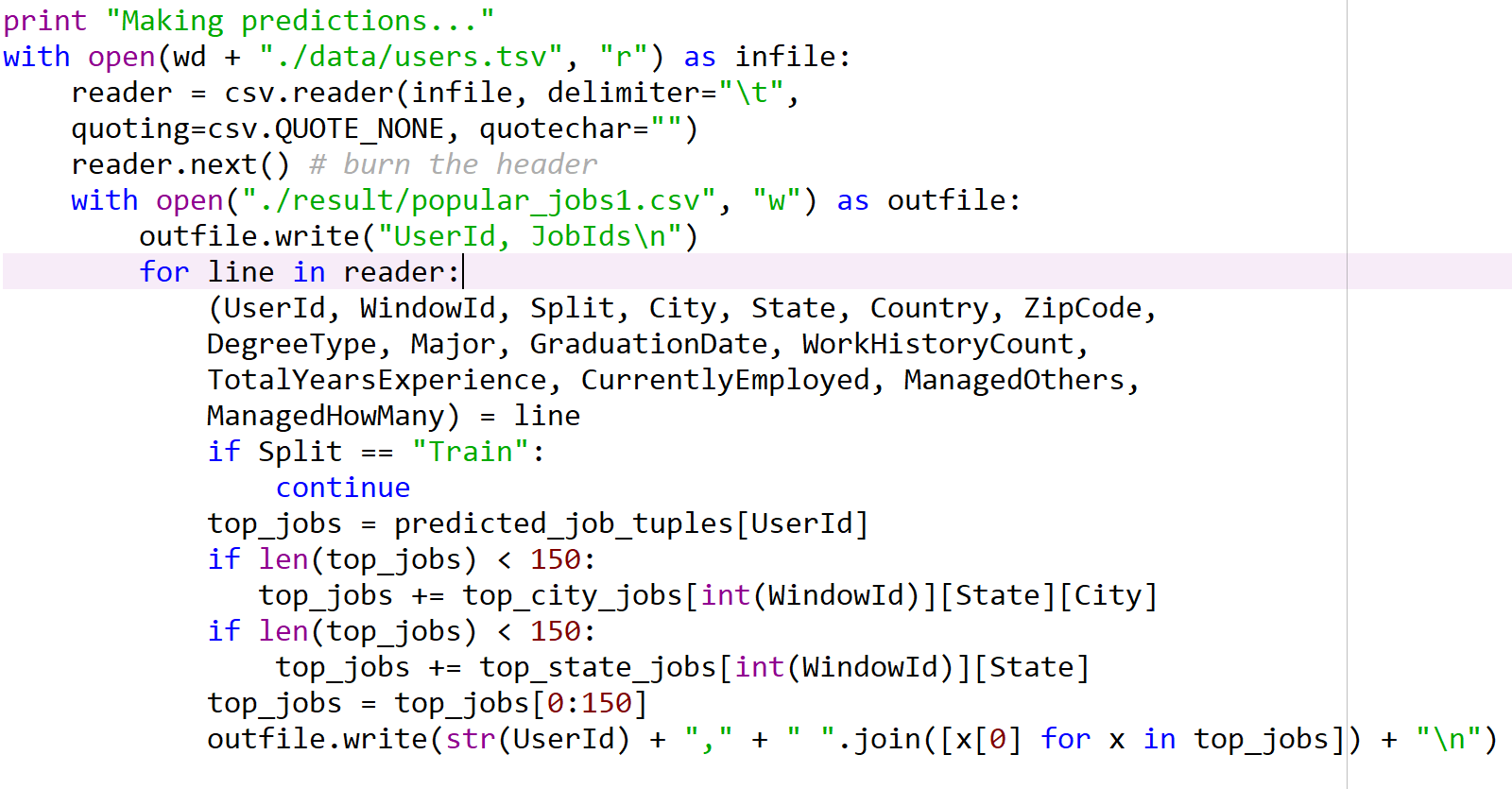
**Step 4:** Sorting collaborative filtered jobs



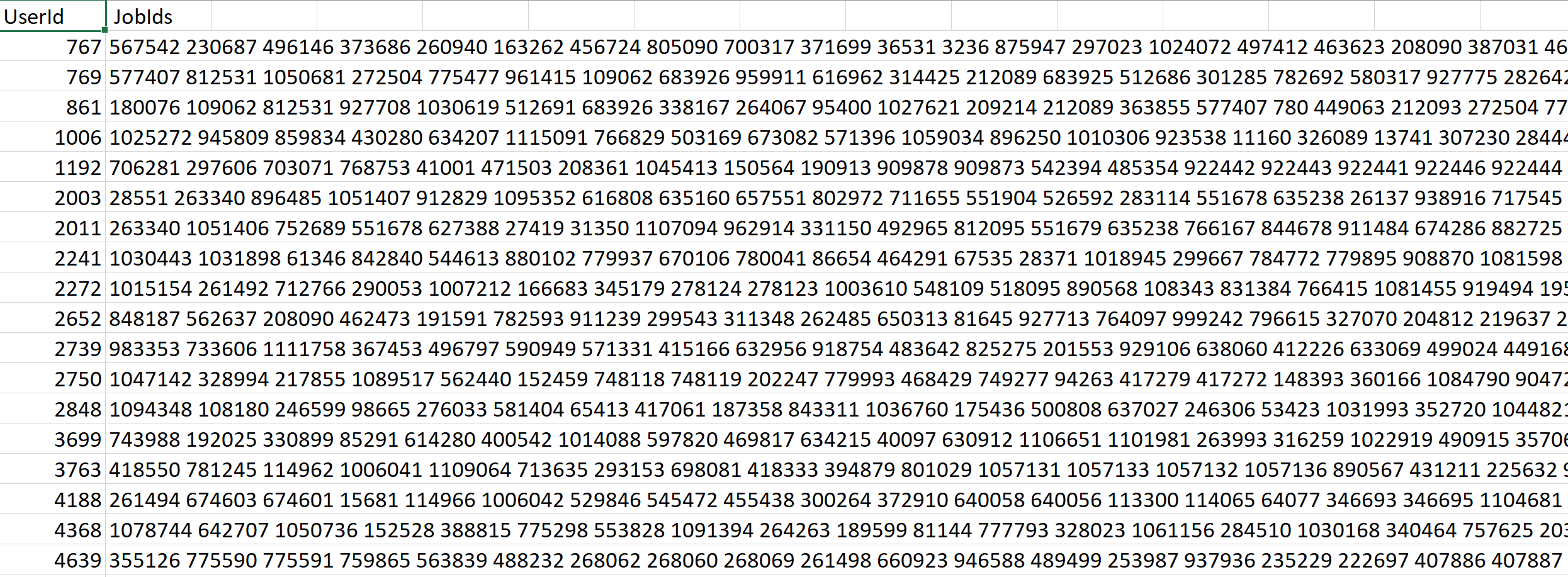
**Step 5**: Sorting job based on popularity

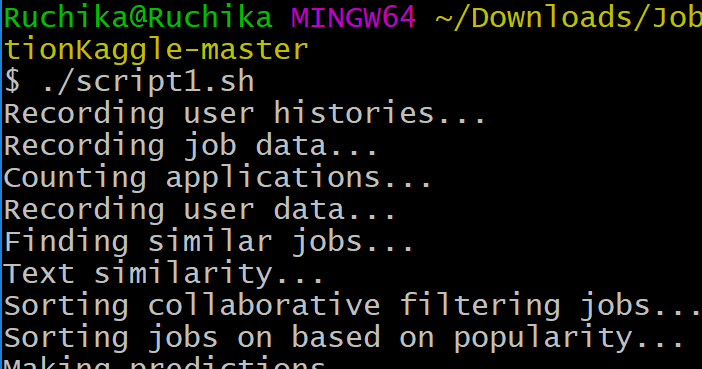


**Step 6:** Making predictions for the test data users



**Step 6:** Open popular\_jobs1.csv to see the result for every





1. **Jaccard Similarity**

The Jaccard similarity index (sometimes called the Jaccard similarity coefficient) compares members for two sets to see which members are shared and which are distinct. It’s a measure of similarity for the two sets of data, with a range from 0% to 100%. The higher the percentage, the more similar the two populations. Although it’s easy to interpret, it is extremely sensitive to small samples sizes and may give erroneous results, especially with very small samples or data sets with missing observations.

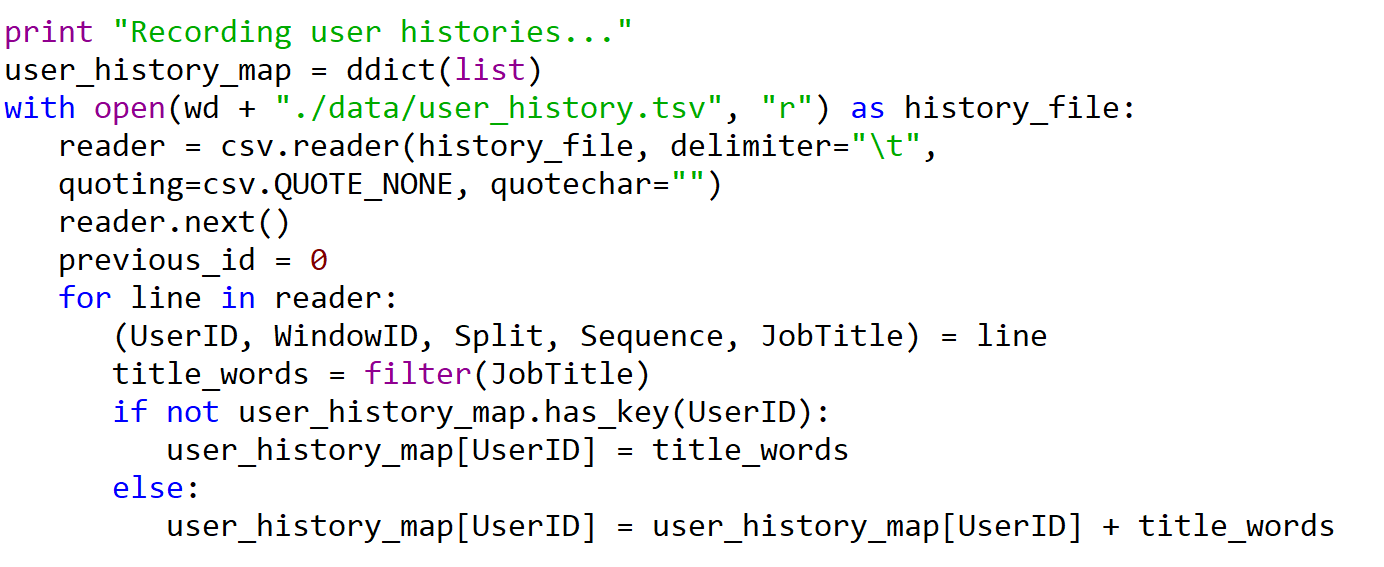
Jaccard coefficient and cosine similarity are two of best known techniques for finding the similarity between two documents, time required for cluster generation by using Cosine Similarity measure takes less amount of time as compare to Jaccard Coefficient measure because of using mathematical formula for calculating the similarity measure between the documents.

On the other hand, Jaccard Coefficient between the two documents by matching all the terms of one document to another which take much more amount of time. By implementing the model for both Jaccard Coefficient and Cosine.

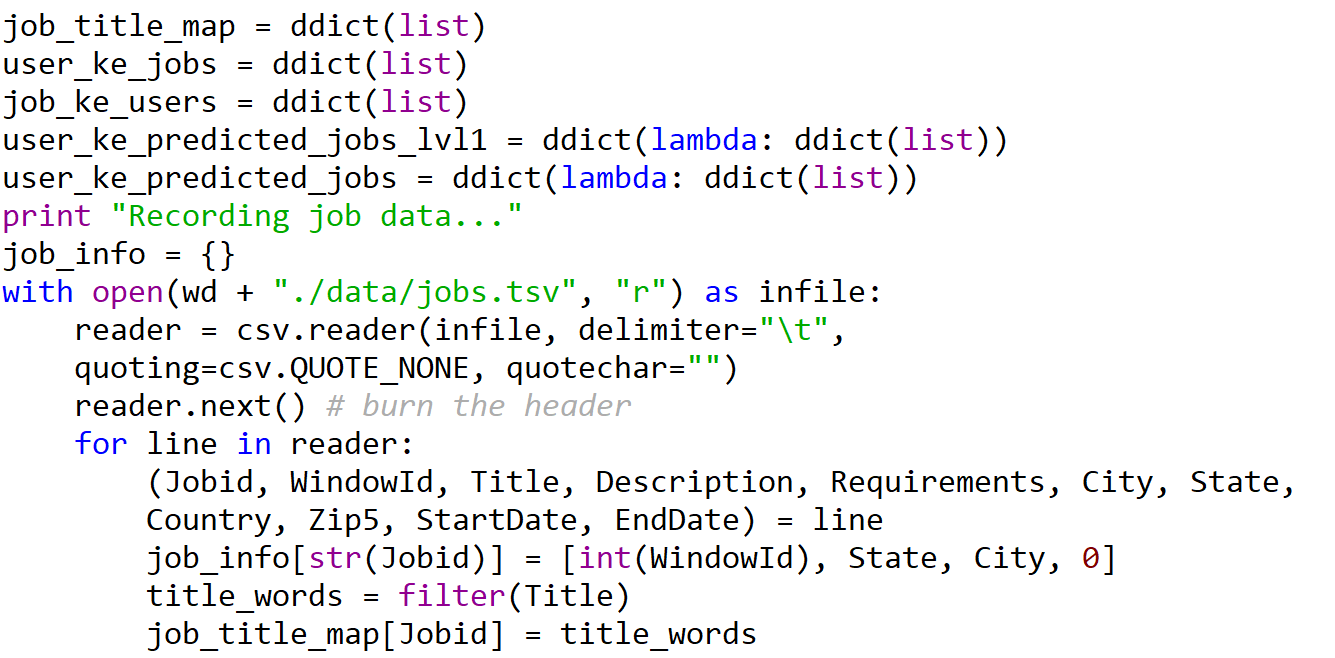
Jaccard Index = (the number in both sets) / (the number in either set) \* 100

J(X,Y) = |X∩Y| / |X∪Y|

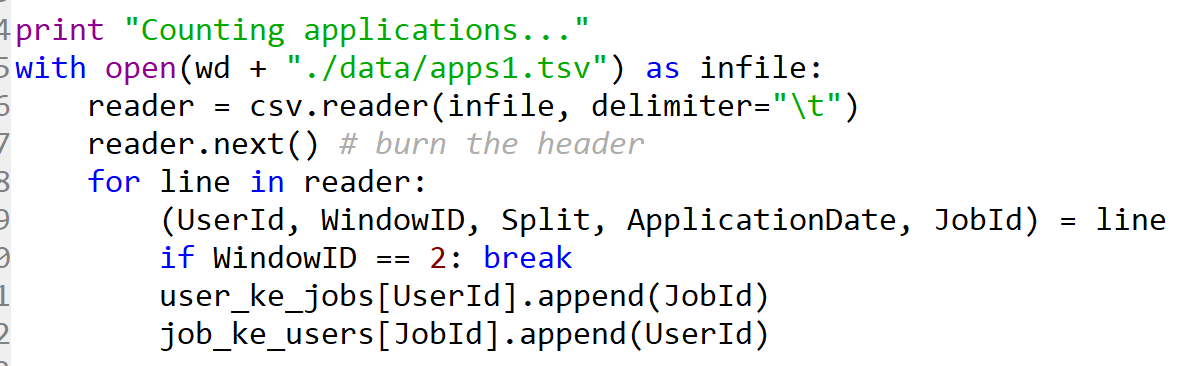
**Step 1**: Getting all the information of job title from the user history



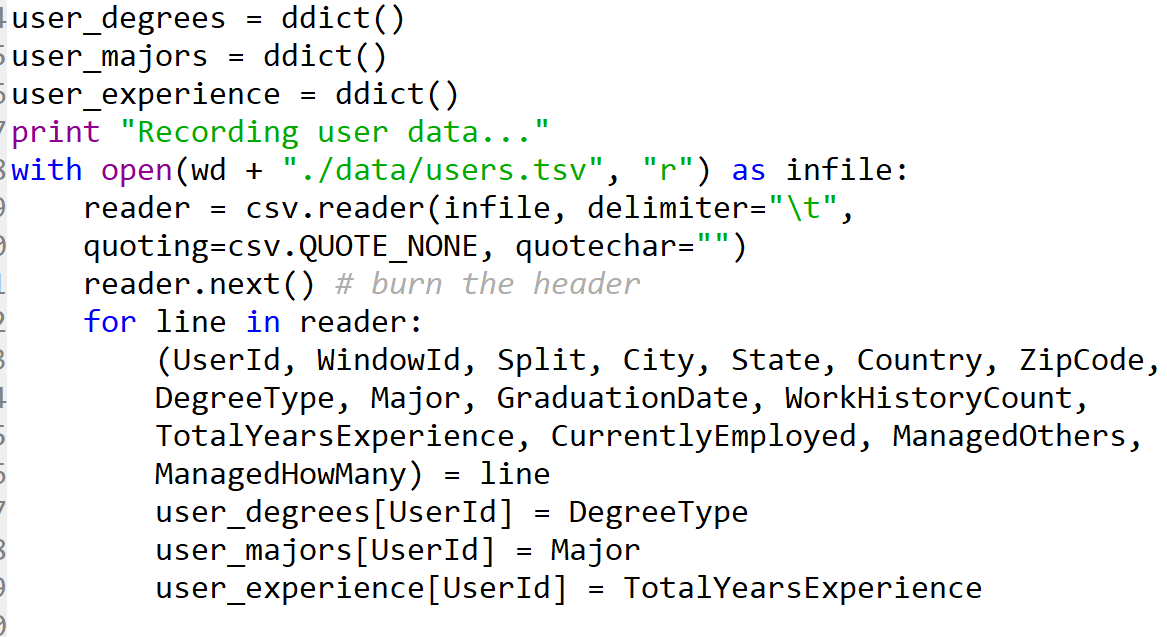
**Step 2**: Getting all the job data



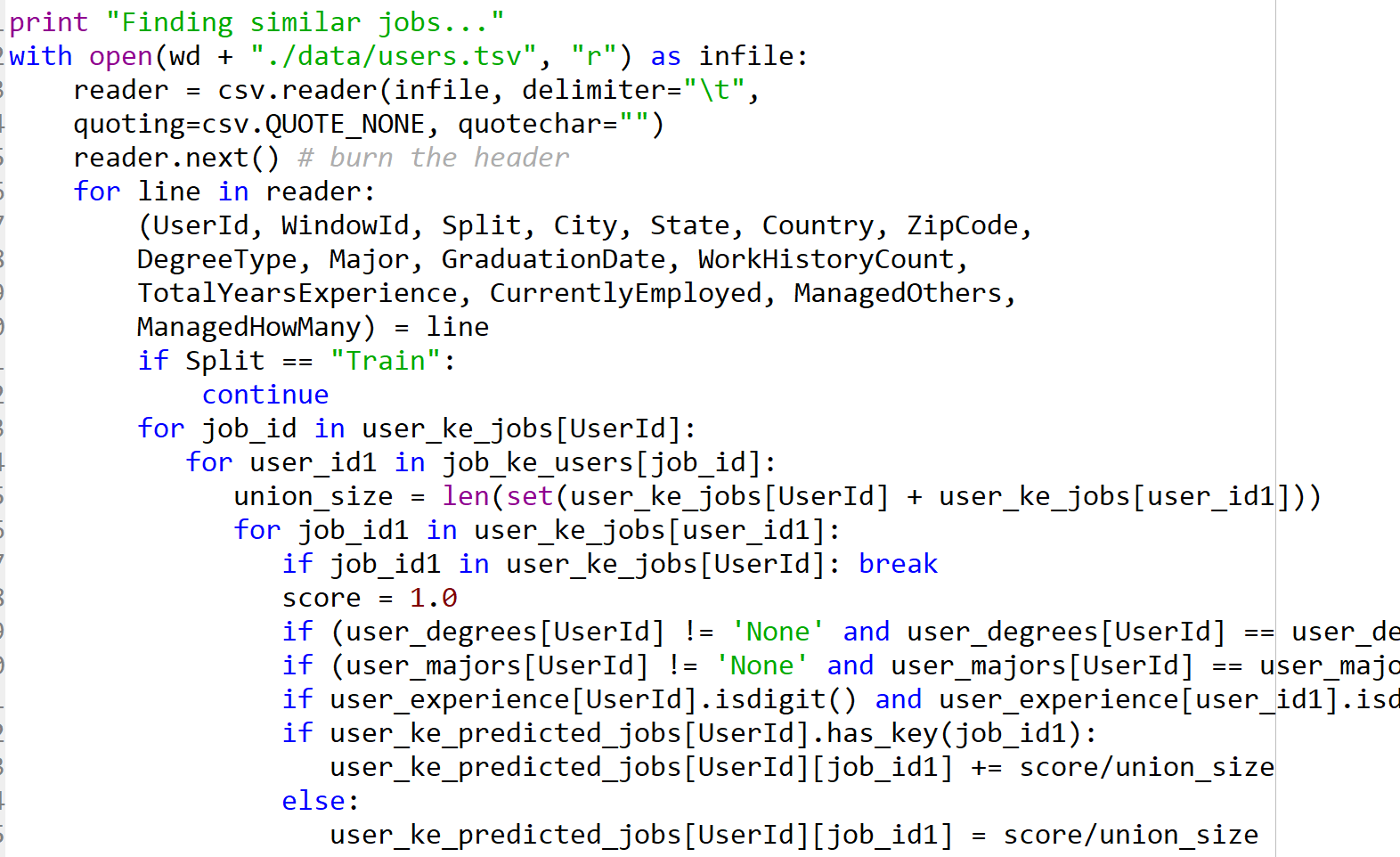
**Step 3**: Counting application



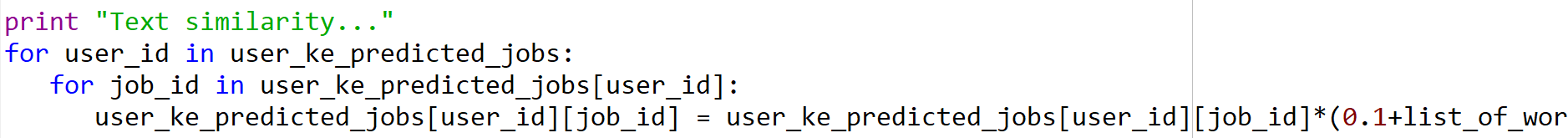
**Step 4**: Recording user data



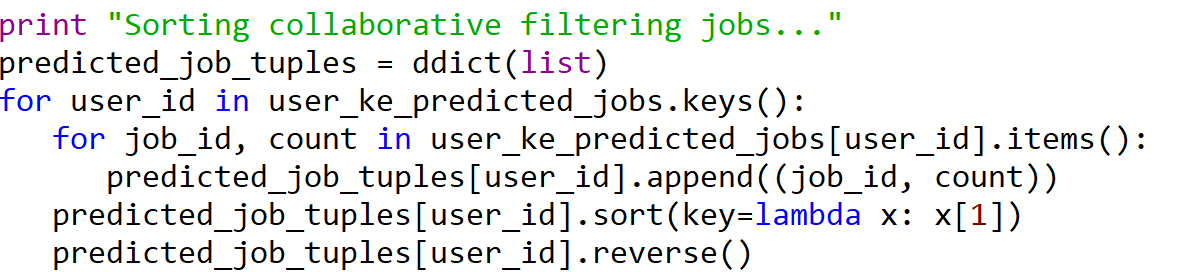
**Step 5**: Finding similarity



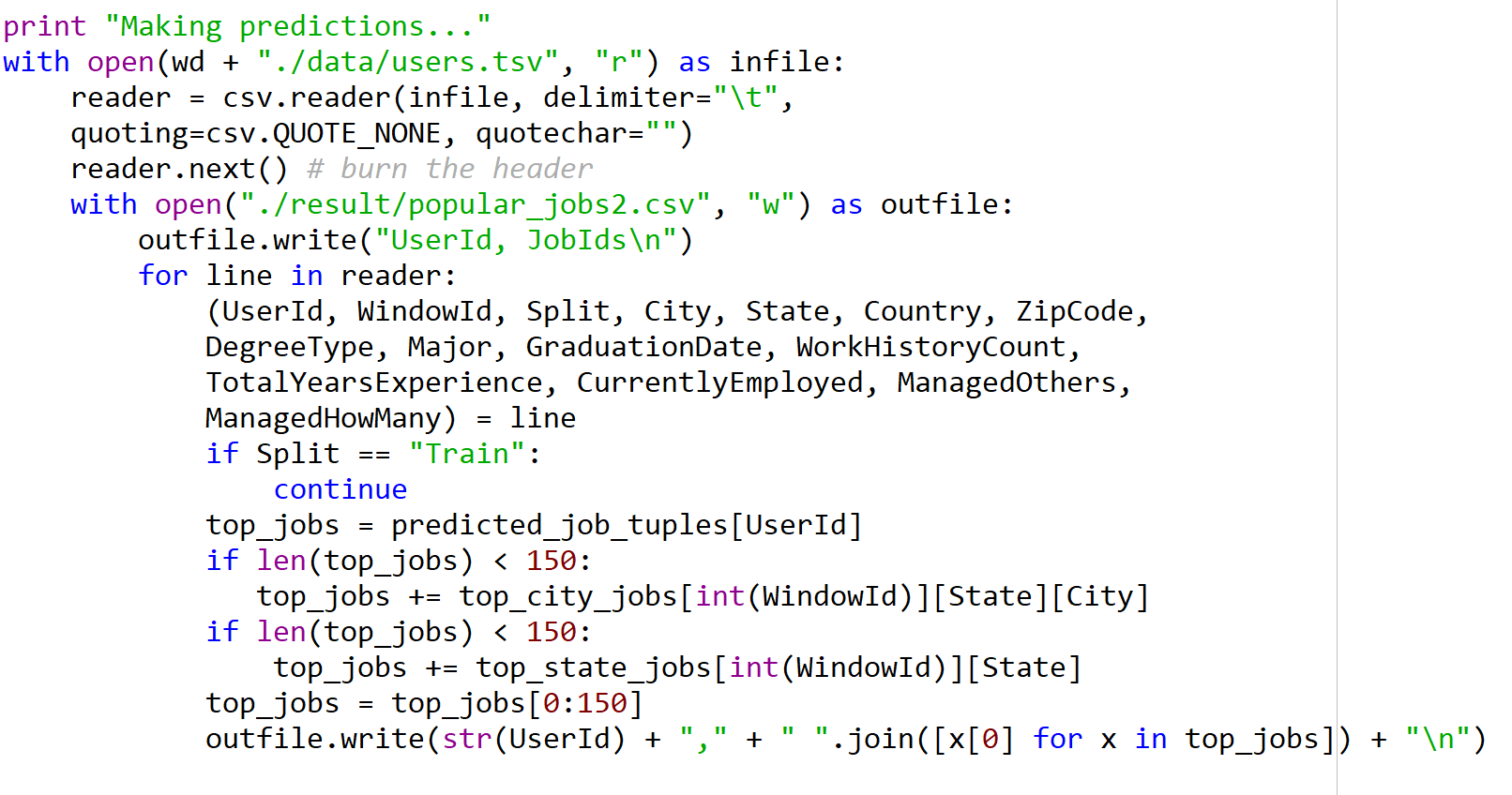
**Step 6:** Find the text similarity



Step 7: Sorting jobs



Step 8: Making predictions



Step 9: Check the popular jobs in the result folder

