Final Project

COMP 250 Winter 2023

posted: Friday, April 14, 2023
due: Friday, April 28, 2023 at 11:59pm

General instructions

- Submission instructions
  - Please note that the submission deadline for the final project is very strict. No submissions will be accepted after the deadline.
  - As always you can submit your code multiple times but only the latest submission will be kept. We encourage you to submit the first version a few days before the deadline (computer crashes do happen and code posts may be overloaded during rush hours).
- These are the files you should be submitting on Ed:

  ```
  MyHashMap.java
  DataAnalyzer.java
  RatingDistributionByProf.java
  RatingDistributionBySchool.java
  GenderByKeyword.java
  RatingByKeyword.java
  RatingByGender.java
  ```

  Do not submit any other files, especially .class files. Any deviation from these requirements may lead to lost marks

- Please note that the classes you submit should be part of a package called *finalproject*.
- Starter code is provided for this project. Do not change any of the class names, file names, or method headers. You can add helper methods (`private` or `protected`) if you wish. Note also that for this project, you are NOT allowed to import any other class (all import statements other than the one provided in the starter code will be removed). Any failure to comply with these rules will give you an automatic 0.
- Whenever you submit your files to Ed, **you will see the results of some exposed tests**. If you do not see the results, your assignment is not submitted correctly. **If your assignment is not submitted correctly, you will get an automatic 0. If your submission does not compile on ED, you will get an automatic 0.**

- The assignment shall be graded automatically on ED. **Requests to evaluate the assignment manually shall not be entertained, and it might result in your final marks being lower than the results from the auto-tests.** Please make sure that you follow the instruction closely or your code may fail to pass the automatic tests.
- The exposed tests on ED are a mini version of the tests we will be using to grade your work. If your code fails those tests, it means that there is a mistake somewhere. Even if your code passes those tests, it may still contain some errors. Please note that these tests are only a subset of what we will be running on your submissions, we will test your code on a more challenging set of examples. Passing the exposed tests assures you that your submission will not receive a grade lower than 40/100. We highly encourage you to test your code thoroughly before submitting your final version.
- Next week, a mini-tester will also be posted. The mini-tester contains tests that are equivalent to those exposed on Ed. We encourage you to modify and expand it. ***You are welcome to share your tester code with other students on Ed.*** Try to identify tricky cases. Do **not** hand in your tester code.
- **Failure to comply with any of these rules will be penalized**. If anything is unclear, it is up to you to clarify it by asking either directly a TA during office hours, or on the discussion board on Ed.

Project set up

For this project, you would be using a GUI that is programmed in JavaFX, so you need to set up JavaFX in your IDE properly.

- **For Intellij user (recommended):**
  - **Windows user:** It should be already included in the SDK if you are using Java 1.8 or higher.
  - **Mac user:** By default you laptop might be using Amazom Correto distribution, you need to change it to Liberica distribution to support media.
    1. open File → Project Structure → SDKs → Add → Download new SDKs → Select Liberica and install it
    2. In your run configuration, select Liberica as your build SDK and build the project
- **For Eclipse user:**
  - **Windows user:** You need to install JavaFX library manually
    1. In Help menu, in Install new software wizzard you should add the new site location to find proper software. Use ”Add” button, then in ”name” section type ”e(fx)clipse (or anything you want, it does not matter). In ”location” section type: `https://download.eclipse.org/efxclipse/updates-nightly/site/`
    2. Search downloadable package by applying a filter ”e(fx)clipse” you should see a list of options (such as JavaFX SDK)
    3. Install them all, after that Eclipse will restart
    4. In Eclipse select the project, run Project → Preferences → Java Build Path → Add Library → Select JavaFX SDK, then rebuild the project, all errors should go away
  - **Mac user:** switch to Intellij

Learning Objectives

The main purpose of this project is to get you familiar with Hashmaps, their implementation, and their usage. You will also learn how to implement simple tools that allow you to do some basic data analysis. Finally, one of the goals of this project is to enable you to build a piece of software that helps you investigate and reason about how humans use language in gendered (and potentially biased) ways.

Introduction

This final project takes inspiration from an assignment created by Juliette Woodrow, Kathleen Creel, and Nick Bowman from Stanford University.

For this project you will write several classes to help you parse and store a complex dataset of college professor reviews. You will then use this structured data together with the graphical tool provided in order to plot word frequency data across review quality and professor gender to reveal interesting trends about biases in language usage.

At the heart of this assignment is a dataset collected from the popular website RateMyProfessors.com, where university students can publish anonymous textual reviews of their professors along with numerical ratings on both the perceived quality and difficulty of a course. While this website contains an incredibly rich collection of data, we chose to focus on reviews from a subset of universities across a time span of 15 years, totaling 19685 reviews overall. Teacher ratings are a common and impactful facet of life in university. Students use the results of these reviews to help them choose their classes and plan their academic futures. However, teaching evaluations are not an objective evaluation of the quality of a professor's teaching. Recent research has shown that teaching evaluations often demonstrate harmful biases, including gender bias. The bias in teaching evaluations is a problem because the scores are often used in decisions about who to hire, fire, tenure, and promote.

In the first part of the project, you focus on implementing the data structure that you will need to process and analyze the dataset. In the second part, you will write tools to help you visualize the breakdown of word frequencies by numerical rating and professor gender. The data visualization in this project is quite simple, if you'd like to have some inspiration of how to extend it, or if you'd simply like to have fun exploring more aspects for this type of data you can check out this: http://benschmidt.org/profGender/.

The Dataset

Together with this pdf you have received two files called *RateMyProf_Data_Gendered.csv* and *RateMyProf_Data_Gendered_Sample.csv*. The latter only contains a subset of the former and can be used to help you with testing and debugging your code.[1] You will be tested on the bigger dataset. The two files provided contain some of the data collected from RateMyProfessors.com. There are 8 columns with the following labels:

- `professor_name`: the name of the prof receiving a review.
- `school_name`: the name of the school where the prof is from.
- `department_name`: the department where the prof teaches.

[1] I highly suggest you to create a very small csv file containing only 10 lines of data to use as a first step to debug your code!

- `post_date`: the date on which the review was posted.
- `student_star`: how the student rated the prof/course in terms of quality.
- `student_difficult`: how the student rated the prof/course in terms of difficulty.
- `comments`: the comment posted by the student.
- `gender`: the perceived gender of the prof.

Note that to help you analyse whether or not the language used in the reviews is potentially gender-biased, we added the last column to the dataset we collected. In this column we indicate the perceived gender of the professor at hand. Note that the gender is deduced by the pronouns used by the students in their comments. We identified the professor as perceived to be male ("M") if male pronouns were used (he/him/his/..), female ("F") if female pronouns were used (she/her/hers/..), and as non-binary ("X") otherwise (please note that this includes situations in which non-gendered/nonbinary pronouns were used, but most often we simply could not decide on a specific gender because either no pronouns were used in the comment or the comment was written in a language other than English).

Your Tasks

Make sure you implement all required methods according to the instructions given in this pdf. In addition to the required methods, you are free to add as many other `private` or `protected` methods as you want. Note that `null` keys are not allowed in the hash table. Remember that in most of scenarios objects comparison does not use '=='. We highly suggest you read through the entire instructions before you start coding. You do not necessarily have to implement the methods in the order in which they are presented. Note also that this project is meant for you to practice first hand how to implement a hash table as well as learning when to use it and how to exploit its properties. For this project you want to focus on optimizing the time efficiency of your code. We do not care about space efficiency.

Please note that the *Controller* and *Parser* classes have been already fully implemented for you. This means that you should NOT modify these classes and you do not need to fully understand them. However, you are strongly encouraged to take a look at these classes to get a general idea of how they work!

PART - 1: Your Own Hash Table

Let's start by implementing your own Hash table. To do so, you need to complete the file *MyHashTable.java* which was provided to you. To do so, you should first familiarize yourself with the provided `MyPair` class. An object of type `MyPair<K,V>` represents a key-value pair, where the key is of type `K` and the value is of type `V`.

The class `MyHashTable` has the following fields:

- An `int` called `size` that represents the number of entries stored inside the table.
- An `int` called `capacity` that represents the number of buckets the table has. The default capacity is 16. (Note that this value is not fixed, it could change after the creation of a hashmap if the number of entries increases enough).

- A `final double` representing the maximum load factor for the hash table.
- An `ArrayList` of buckets used to store the entries to the table. Where each bucket is a LinkedList of MyPairs.

Implement the following `public` methods in the `MyHashTable` class:

- A constructor `MyHashTable()` which takes no inputs and initialiazes the fields so that this object represent a hash table with no entries and initial capacity equal to 16.
- Another constructor `MyHashTable()` which takes an `int` as input representing the initial capacity of the table.[2] Using the input, the constructor initializes all the fields.
- A `put()` method that takes a *key* and a *value* as input. The method adds a `MyPair` of the *key* and *value* to the hash table. If a `MyPair` with the *key* already exists in the hash table, then you should overwrite the old *value* associated with the *key* with the new one. This method should run in $O(1)$ on average. If in this hash table there was a previous value associated to the given key, then the method overwrites it with the new value and returns the old one. If there was no value associated to the given key, then the method returns `null`. Remember that the load factor of the table should never be greater than the maximum load factor stored in the appropriate field.
- A `get()` method which takes a *key* as input and returns the *value* associated with it. If there is no such key in the hash table, then the method should return `null`. This method should run in $O(1)$ on average.
- A `remove()` method that takes a *key* as input and removes from the table the entry (i.e. the `MyPair`) associated to this *key*. The method should return the *value* associated to the *key*. If the *key* is not found, then the method returns `null`. This method should run in $O(1)$ on average.
- A `rehash()` method which takes no input and modifies the table so that it contains double the number of buckets. This method should be $O(m)$ where m is the number of buckets in the table.[3] Generally this method would be `private`, but for testing purposes we made it `public`.
- A `getKeySet()` method which takes no input and returns an `ArrayList` containing all the keys in the table. The `keys` in the returned `ArrayList` may be in any order. This method should be $O(m)$ where m is the number of buckets in the table.
- A `getValueSet()` method which takes no input and returns an `ArrayList` containing all the ***unique*** values in the table. The returned `ArrayList` of *unique* `values` may be in any order. This method should be $O(m)$ where m is the number of buckets in the table.
- A `getEntries()` method which takes no input and returns an `ArrayList` of `MyPair<K, V>`s containing all the entries in the table. The order does not matter. This method should be $O(m)$ where m is the number of buckets in the table.

[2]The capacity is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created.

[3]In the slides we mention that these methods run in $O(n + m)$ where n is the number of entries, and m is the number of buckets. Note that if you have a good hash function and a max load factor of 0.75, then this is equivalent to say that the method runs in $O(m)$.

Finally, implement the following methods from the `private` nested class `MyHashIterator`[4]:

- The constructor which should be $O(m)$, where m is the number of buckets in the table.
- A `hasNext()` method which should be $O(1)$ and returns `true` if the hash table has a next `MyPair`.
- A `next()` method which is also $O(1)$ and returns the next `MyPair`.

PART - 2: Let's Analyze Some Data!

One of the most common applications of hashmaps is probably data analysis. For this project, we want you to build some handy tools to help us visualize the data collected.

As mentioned before, we have provided you with two CSV files storing data obtained from *RateMyProfessors.com*. In the starter code we also provide a parser that reads all the data from the CSV files to the program when launched. The data is stored in the field called `data` from the `Parser` class, and each data entry in the CSV file is represented by an array of strings, where each slot corresponds to a column in the CSV file. For example, when parsing data from the CSV files provided, the first string in the array corresponds to the professor's name. To help you quickly access the index for each field, we used another field variable called `fields` which is a `HashMap` mapping the field name (i.e. the column labels) to an integer representing the corresponding index in the arrays stored inside `data`. You can read the parser class and get yourself familiar with the setup.

For the data visualization of our results, we will use JavaFX to build a simple GUI to draw charts based on our results. The code has all been provided to you, once you have set up the environment correctly you should be able to launch the application right away. Once you implement the classes below you will be able to use the application by first pressing on the button describing the analysis you are interest in, then entering the keyword of your choice in the text box and pressing enter.

The `DataAnalyzer` class

Now let's look in more detail in what your task consists of. First, let's open the abstract class `DataAnalyzer` that has been provided to you. At the moment the class contains the following field:

- A `parser` of type `Parser` used to store the parsed data waiting to be processed further.

The class also has the following public methods:

- `DataAnalyzer(Parser)`: A constructor that initializes the parser field with the provided input and calls the method `extractInformation()` (see below for what the purpose of this method is).

[4]Make sure to read the documentation of `java.util.Iterator` to know exactly how each of the following methods are expected to behave.

- `getDistByKeyword(String keyword)`: A method that returns the analyzed results based on the input `keyword` as a `MyHashTable<String, Integer>` object. This is the data that will be plotted using either a bar graph or a line graph. The key, which is of type `String`, represents the entry of the chart (what will appear on the x-axis), and the value, of type `Integer`, represents that entry's data (the value on the y-axis). The method is `abstract` for its implementation changes according to the type of analysis we are interested in.
- `extractInformation()`: The purpose of this method is to extract information from the parsed data. Which information to extract, and how to store it depends on the type of analysis we are interested in doing. This is why the method is `abstract` and the implementation is left to each specific subclass. Note that the method is called by the constructor, which means that whenever an object of type `DataAnalyzer` is created, the relevant information is extracted from the parsed data and stored where can be accessed by `getDistByKeyword()` which will in turn select what should be returned.

The rest of the project consists in implementing classes that extends `DataAnalyzer`, each of which focuses on a specific way of analyzing the parsed data. For all of these subclasses, you will need to implement `getDistByKeyword()` and `extractInformation()` based on the analysis required. How you do so, it's up to you! These are the only constraints you have:

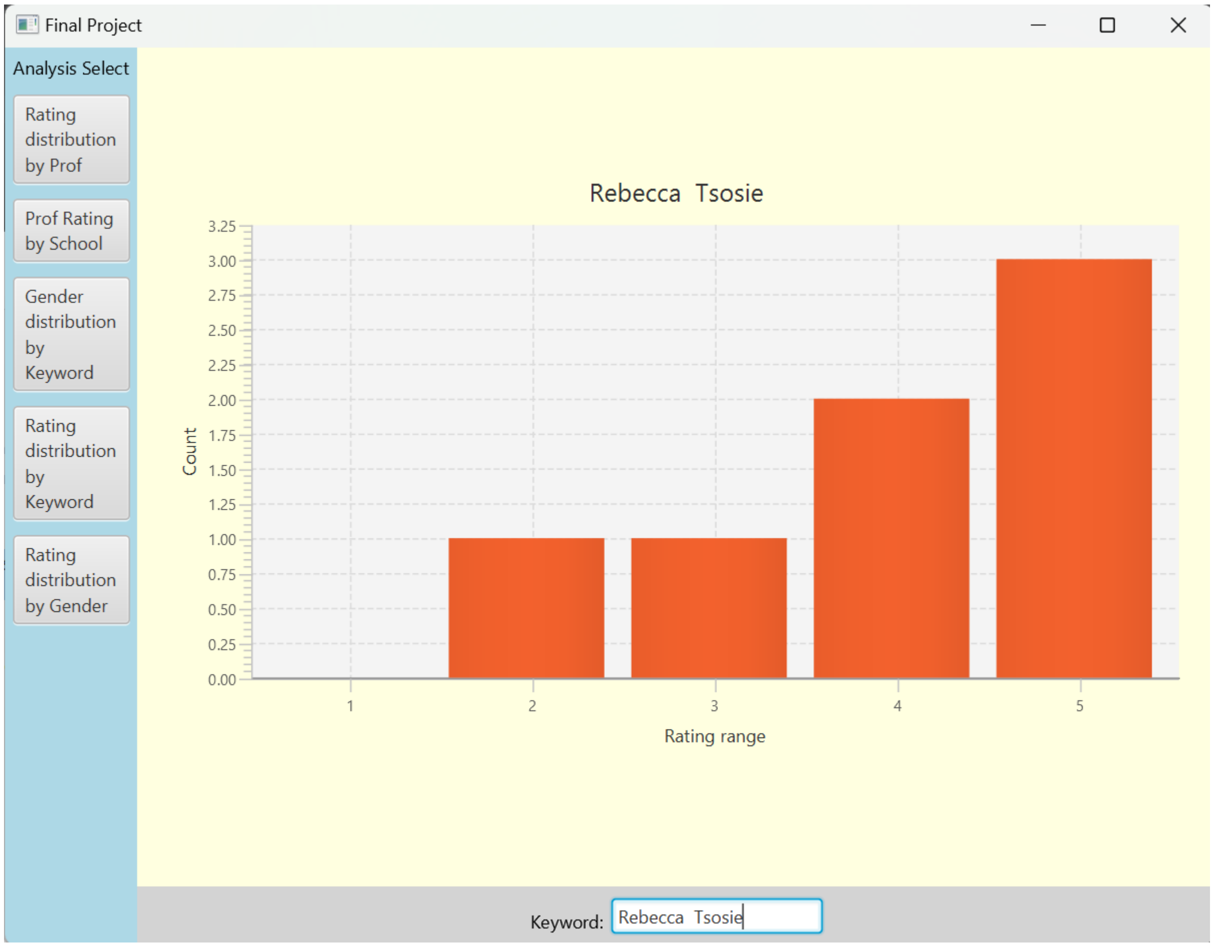
- `getDistByKeyword()` should run in $O(1)$ on average.
- `extractInformation()` should run in $O(n)$ on average, where n is the number of data entries, i.e. the size of `data` from `Parser`.
- You can add as many `private`/`protected` fields/methods as you like.

Given the main topic of this project, you might have correctly guessed that to adhere to the runtime constraints you will have to find the right way to use the `MyHashTable` data type created in the previous part of this project. Once again, it is up to you to figure out how and where to store the extracted data, whether using one big nested hashmap or multiple hashmaps, just note that the value returned by `getDistByKeyword()` needs to be correct in order to get full marks. Below, you can find a description of the type of data analysis each of the subclasses focuses on. Note that this might not be the most efficient way to analyze the data, but we had to give up on some of the efficiency so that it is possible for us to assign marks for each of the tasks separately.

Rating distribution by prof

Sometimes you might wonder, how ratings are distributed for a given professor you are interested in. This is what the class `RatingDistributionByProfessor` is for! The method `getDistByKeyword()` will receive as input the name of a prof. It should then return a table mapping five rating categories (`"1"`, `"2"`, `"3"`, `"4"`, `"5"`) to the number of reviews received with quality ratings that falls within each of those category. Note that, we count as category `"1"` any review with a quality rating that falls between 1 and 2, including the 1 and excluding the 2. Similarly for all other categories.

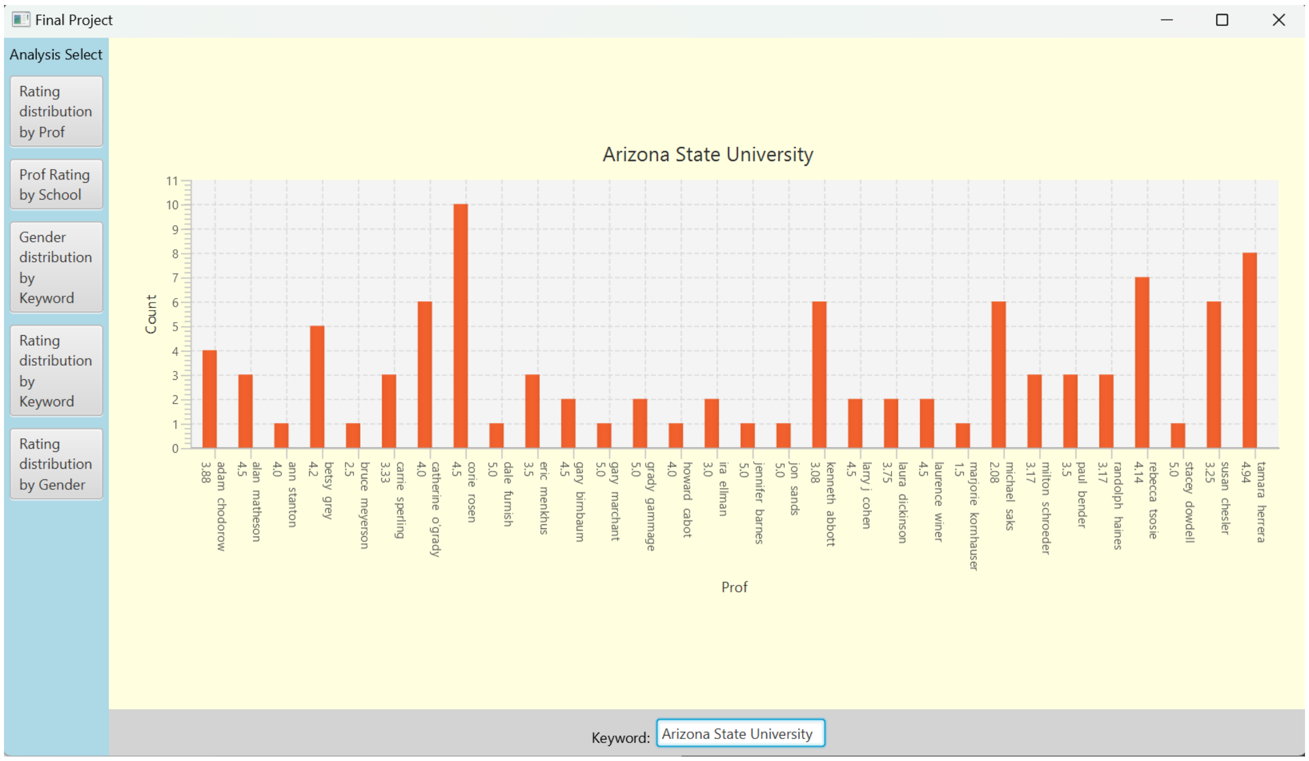
When processing the strings containing the names of the profs make sure remove all leading and trailing whitespaces, and convert the string to lower case. Below you can see what would be displayed when running the GUI with the *RateMyProf_Data_Gendered_Sample.csv* data set and asking for the rating distribution of *Rebecca Tsosie*. Note that you can control which dataset is used by modifying line 35 in the *Controller.java* file.



Rating count per prof for selected school

Another feature we want to implement is to see the rating distribution within a selected school provided by the keyword. For example, given a school name, we want to know how many reviews have been posted for each professor and the average quality rating for this professor. In this case, `getDistByKeyword()` will receive as input the name of a school. It should return a table mapping strings containing prof names and their average rating (on a newline) to the number of reviews they have received.

Once again, when processing the strings containing the names of the profs make sure remove all leading and trailing whitespaces, and convert the string to lower case. Below you can see what would be displayed when running the GUI with the *RateMyProf_Data_Gendered_Sample.csv* data set and asking for the ratings of the profs from *Arizona State University*.



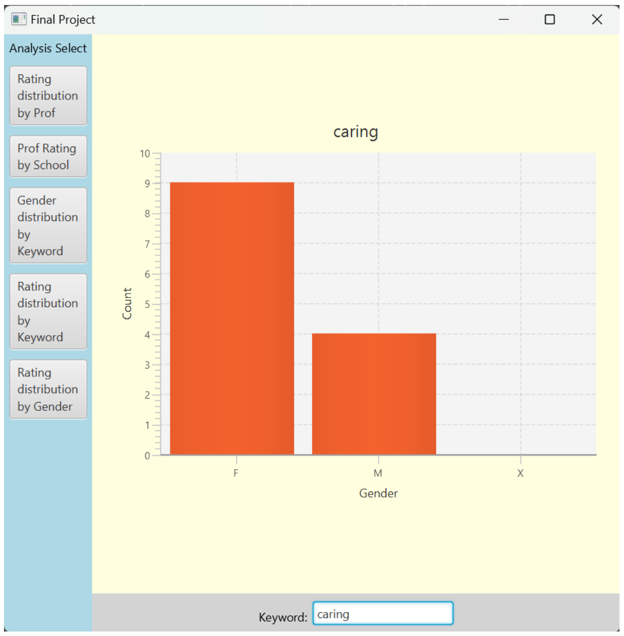
Gender by keyword

Next, we would like to evaluate the gender distribution based on keywords that appear in the students' comments. For example, we would like to see whether the word "caring" is used equally when describing professors across all genders. In this case, `getDistByKeyword()` will receive as input a word. It should return a table mapping genders (`"M"`, `"W"`, and `"X"`) to ***the number of times*** the input word appears in reviews posted for a prof with the corresponding gender.

To do so, while going through all the data you need to make sure to process all the comments and extract all the tokens. Do so by converting the comments to lower case characters, replacing all characters that aren't letters from the English alphabet or the apostrophe (`'`) with a space, and then extract words that are separated by spaces. For example, from `"It's fun and clear."` you should get 4 words: `"it's"`, `"fun"`, `"and"`, `"clear"`. Please note that empty strings should not be considered as words. In Figure 1, you can see what would be displayed when running the GUI with the *RateMyProf_Data_Gendered_Sample.csv* data set and querying the program with *caring* or *smart*.

Rating by keyword

The fourth feature we are interested in is seeing the distribution of ratings associated with each keyword, that is, for example, how many comments that contains the word "nice" are associate with reviews with low/high quality ratings. In this case, `getDistByKeyword()` will receive as input a word. It should then return a table mapping five rating categories (`"1"`, `"2"`, `"3"`, `"4"`, `"5"`) to ***the number of reviews*** with comments containing the input keyword and with quality ratings that falls within each of those category. This means that a review that contains a comment such as "Fun fun fun!", should be counted only


Final Project
Analysis Select
Rating distribution by Prof
Prof Rating by School
Gender distribution by Keyword
Rating distribution by Keyword
Rating distribution by Gender
caring
Count
Gender
F
M
X
Keyword: caring

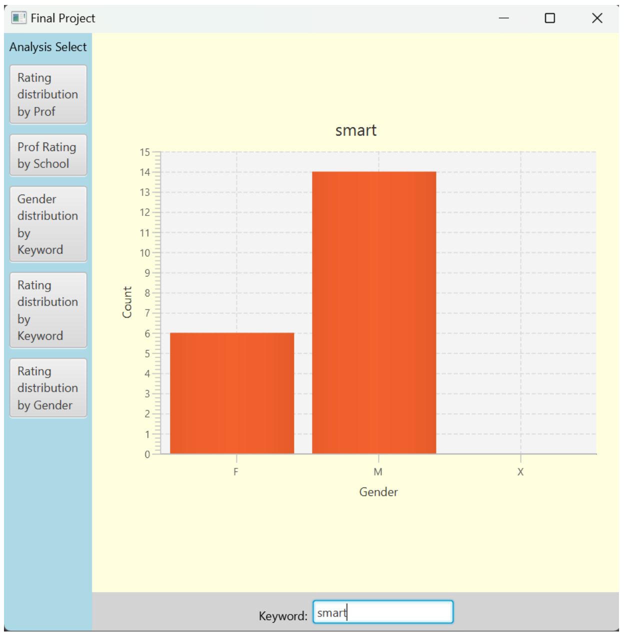

Final Project
Analysis Select
Rating distribution by Prof
Prof Rating by School
Gender distribution by Keyword
Rating distribution by Keyword
Rating distribution by Gender
smart
Count
Gender
F
M
X
Keyword: smart

Figure 1: Results for *caring* on the left, and for *smart* on the right.

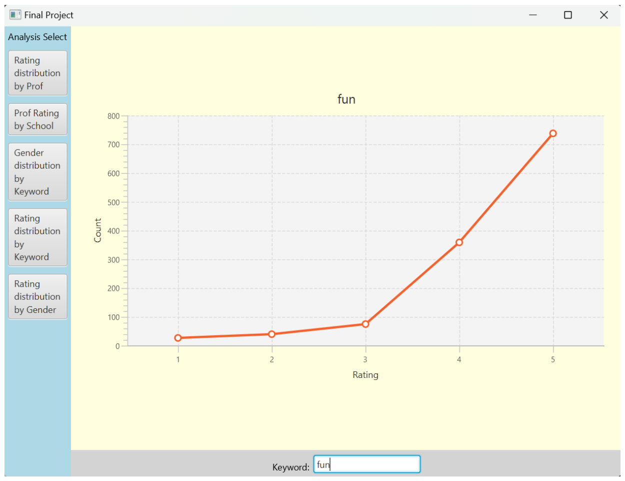
once when analysing the data using the keyword “fun”. As before, we count as category `"1"` any review with a quality rating that falls between 1 and 2, including the 1 and excluding the 2.

As for the previous class, while going through all the data you need to make sure to process all the comments and extract all the tokens. In Figure 2, you can see what would be displayed when running the GUI with the *RateMyProf_Data_Gendered.csv* data set and querying the program with *fun* or *horrible*.

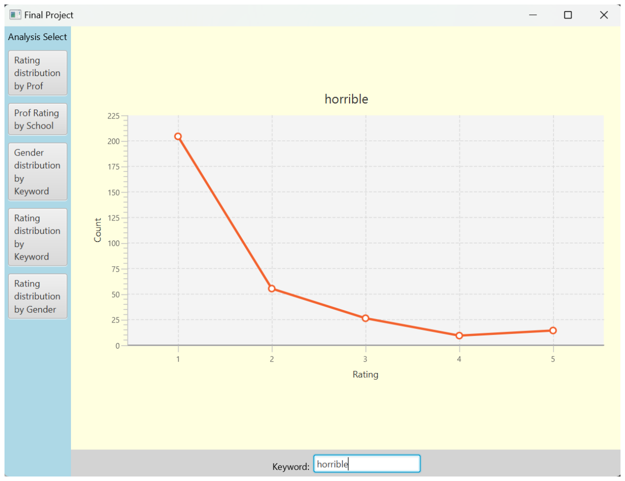
Rating by Gender

Finally, we would like to look at the distribution of the quality and difficulty ratings based on gender. For this part we will focus only on binary genders. In this case, `getDistByKeyword()` will receive as input a keyword containing the gender we want to analyzes followed by the type of rating we are interested in (`quality` or `difficulty`). The two will be separated by a comma. So, a possible input could be `"F, difficulty"`. It should then return a table mapping five rating categories (`"1"`, `"2"`, `"3"`, `"4"`, `"5"`) to the number of reviews in the data with the given rating for a prof with the given gender.

In Figure 3, you can see what would be displayed when running the GUI with the data set from *RateMyProf_Data_Gendered.csv* and querying the program with *quality* or *difficulty*.

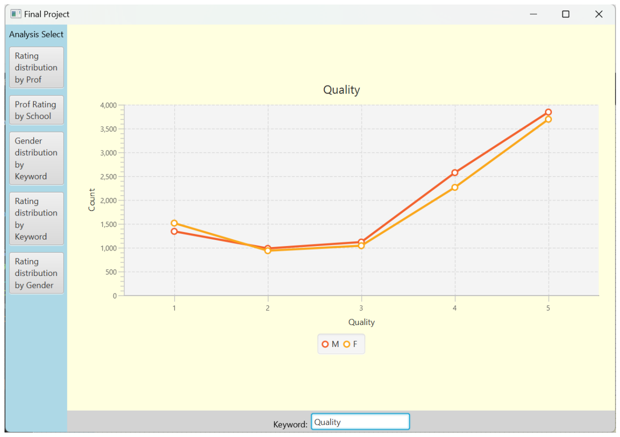

Final Project
Analysis Select
Rating distribution by Prof
Prof Rating by School
Gender distribution by Keyword
Rating distribution by Keyword
Rating distribution by Gender
fun
Count
0
100
200
300
400
500
600
700
800
1
2
3
4
5
Rating

Keyword: fun

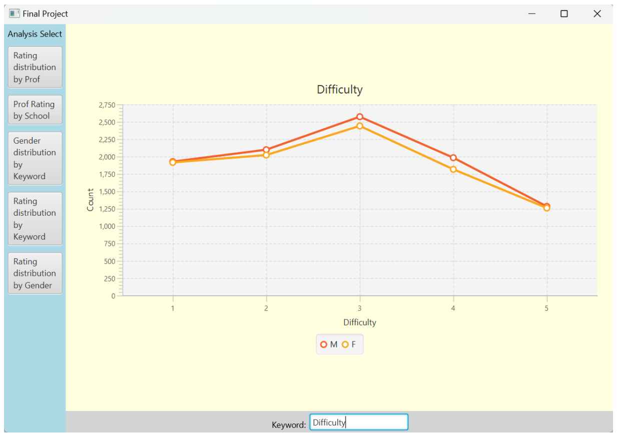

Final Project
Analysis Select
Rating distribution by Prof
Prof Rating by School
Gender distribution by Keyword
Rating distribution by Keyword
Rating distribution by Gender
horrible
Count
0
25
50
75
100
125
150
175
200
225
1
2
3
4
5
Rating

Keyword: horrible

Figure 2: Results for *fun* on the left, and for *horrible* on the right.


Final Project
Analysis Select
Rating distribution by Prof
Prof Rating by School
Gender distribution by Keyword
Rating distribution by Keyword
Rating distribution by Gender
Quality
Count
0
500
1,000
1,500
2,000
2,500
3,000
3,500
4,000
1
2
3
4
5
Quality
M
F

Keyword: Quality



Keyword: Difficulty

Figure 3: Results for *quality* on the left, and for *difficulty* on the right.