

**Prototyping a Literary Analysis Tool for Creatives that Doesn't Use Cloud Computing and
Develops New Analysis Metrics**

By

John Chmielowiec

Bachelor of Science in Computer Science

A thesis submitted to the Graduate Committee of
Ramapo College of New Jersey in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Fall, 2023

Committee Members:

Scott Frees, Advisor

Kathleen Walsh, Reader

Lawrence D'Antonio, Reader

COPYRIGHT

© John Chmielowiec

2023

Table of Contents

Table of Contents	iii
List of Tables	v
List of Figures.....	vi
Abstract	1
Introduction	2
Background	6
Previous + Current Research	6
Personal Tools	11
Standalone Natural Language Tools	14
Version Control.....	15
Methodology	17
General Design Decisions	17
Use of the .NET MAUI Framework for the User Interface and Model	17
Model-View-Viewmodel-Services	18
SQLite3 as the Database	19
Data Used	19
Open Source Projects	20
Natural Language Processing Libraries and Models.....	22
spaCy and Spacy Syllables.....	22
Emotion Classification Models	23
Revision Tracking.....	24
Data Cleaning Tools for Users.....	25
Interactive Visualizations	29
Chart Control	30
User Filters	30
Data Analytics for Users	38
Entity Presence Scores.....	38
Emotion Classification Probability Scores	39
Named Entity Tonality	40
Flesch Reading Ease	41
Flesch-Kincaid Grade Level	41
Paragraph Count.....	42

Part of Speech Usage	43
Sentence Detection and Count	43
Word Count.....	44
Word Count Change	44
Word Frequency	45
Data Summary	45
Data Export for Data Scientists.....	46
Analysis and Discussion.....	48
Interactive Visualizations	48
Basic Statistical Metrics	49
Paragraph Count.....	49
User Cleaning of Named Entity Recognition Errors	50
Entity Presence Scores	55
Emotion Classification Models	61
Emotion Classification Probability Scores.....	74
Entity Tonality.....	77
Revision Tracking.....	82
Ethical Implications.....	83
Conclusions	86
Goals.....	86
Contributions	86
Future Work.....	87
Entity Presence.....	87
Emotion Classification.....	87
Entity Tonality	87
Revision Tracking	88
Other Analytics	88
References.....	90
Data	90
Open Source Software	91
Writing Tools	93
Research.....	95
Miscellaneous.....	96
Appendices	97
Appendix A: User Interface Screenshots	97

List of Tables

Table 1 Tool Feature Comparison (input category not included)	14
Table 2 spaCy en_core_web_lg pipeline evaluation metrics	23
Table 3 simple mean of labels.....	62
Table 4 weighted average of labels (using support)	62
Table 5 Per-label metrics (maximizing f1) for the roberta-base-go_emotions model	63
Table 6 Per-label metrics (maximizing f1) for the EmoRoBERTa model.....	64

List of Figures

Figure 1 Possible Qualitative Analysis for Search Syntheses (Onwuegbuzie et al., 2012)	7
Figure 2 Screengrab From a Recording of Kurt Vonnegut in 1985 Talking About The Shape of Stories (David Comberg, 2010).....	9
Figure 3 The "Shape" of Harry Potter and the Deathly Hallows by J.K. Rowling (Jones, 2016) .	10
Figure 4 Named Entity Corrections Interface.....	28
Figure 5 Named Entity Correction Interface - Alias Selection	29
Figure 6 Entity Presence, No Smoothing.....	33
Figure 7 Entity Presence, Smoothing (Centered Rolling Average: 2 items)	33
Figure 8 Entity Presence, Smoothing (Centered Rolling Average: 5 items)	34
Figure 9 Entity Presence, Smoothing (Lookaround: 2 Back, 5 Forward).....	36
Figure 10 Entity Presence, Smoothing (Lookaround: 2 Back, 3 Forward).....	37
Figure 11 Entity Presence, Smoothing (Lookaround: 5 Back, 3 Forward).....	37
Figure 12 Emotion Probability – Multiple Emotions For A Single Item	39
Figure 13 Emotion Probabilities – Tooltip	40
Figure 14 Sentence Detection – Completely Blank Line.....	42
Figure 15 Sentence Detection – Newline Character At End Of Line.....	43
Figure 16 Data Summary Popup.....	45
Figure 17 Data Export Into Excel	47
Figure 18 Entity Presence Utilizing Entity Corrections (New Entity).....	50
Figure 19 Entity Presence Utilizing Entity Corrections (No Aliasing).....	51
Figure 20 Entity Presence Utilizing Entity Corrections (Successful Alias).....	52
Figure 21 Entity Presence Utilizing Entity Corrections (Incorrect Entity Combination)	53

Figure 22 Named Entity Corrections - Database Query of "pig"	54
Figure 23 Named Entity Correction - non-entity detection	54
Figure 24 Entity Presence - Title of Poem Alters Results	56
Figure 25 Entity Presence - Entity Grouping	57
Figure 26 Entity Presence - No Smoothing Function	58
Figure 27 Entity Presence - No Smoothing Function (Zoomed).....	59
Figure 28 Entity Presence - Lookaround Smoothing Function.....	59
Figure 29 Entity Presence - Centered Rolling Average Smoothing Function	60
Figure 30 Emotion Classification Comparison (Paragraphs, Neutral, Different Models)	65
Figure 31 Emotion Classification Comparison (Sentences, Neutral, Different Models)	65
Figure 32 Emotion Classification Comparison (Different Models, w/o Outliers)	67
Figure 33 Emotion Classification Comparison (Different Models, w/ Outliers)	68
Figure 34 Emotion Classification Comparison (Different Item Grouping, w/o Outliers)	69
Figure 35 Emotion Classification Comparison (Different Item Grouping, w/ Outliers)	70
Figure 36 Emotion Probability - Multiple Emotions	74
Figure 37 Emotion Probability - Low Chance	76
Figure 38 Emotion Probability - Not Present	77
Figure 39 Entity Tonality - Finding Corresponding Emotions for an Entity	78
Figure 40 Entity Tonality - Finding Alternating Emotions for an Entity	79
Figure 41 Entity Tonality – Finding A Scene by Entity Presence (No Emotions Selected)	80
Figure 42 Finding A Scene by Entity Presence (No Emotions Selected, Zoomed In)	80
Figure 43 Entity Tonality – Multiple Characters with Different Tonality Scores For The Same Emotion.....	81
Figure 44 Data Summary Popup.....	82

Figure 45 User Interface - People View.....	97
Figure 46 User Interface - People View, Filtering	98
Figure 47 User Interface - Sorting by Works	99
Figure 48 User Interface - Expanding a Work to View Revisions.....	100
Figure 49 User Interface - Sorting by Revisions	101
Figure 50 User Interface - Selecting a Revision	102
Figure 51 User Interface - Hiding The People List.....	103
Figure 52 User Interface - Works View.....	104
Figure 53 User Interface - Works View - Data Processing.....	105
Figure 54 User Interface - Visualization - Word Frequencies.....	106
Figure 55 User Interface - User Filter - First Item	107
Figure 56 User Interface - User Filter - Max Items.....	108
Figure 57 User Interface - User Filter - Last Item	109
Figure 58 User Interface - User Filter - Foreground Color	110
Figure 59 User Interface - User Filter - Background Color.....	111
Figure 60 User Interface - Visualization – Part of Speech Frequencies.....	112
Figure 61 User Interface - Visualization - Word Count Total shown by Revision Date	113
Figure 62 User Interface - Visualization - Word Count Change shown by Revision Date	114
Figure 63 User Interface – Part of Speech Usage Comparison Popup.....	115
Figure 64 User Interface - Visualization – Part of Speech Comparison Between Two Different Works.	116
Figure 65 User Interface - Visualization – Sentence Lengths	117
Figure 66 User Interface – Chart Interactivity – Zoom Window (Pre Zooming)	118
Figure 67 User Interface – Chart Interactivity – Zoom Window (Post Zooming).....	119

Figure 68 User Interface – Visualization – Entity Presence Scores (Single Entities)	120
Figure 69 User Interface – Visualization – Entity Presence Scores (Groups with Any Modifier)	121
Figure 70 User Interface – Visualization – Entity Presence Scores (Groups with All Modifier).	122
Figure 71 User Interface – Visualization – Entity Presence Scores (Single Entities and Groups with All Modifier).....	123
Figure 72 User Interface – User Filter – Fill Color (Lower Transparency).....	124
Figure 73 User Interface – User Filter – Fill Color (Higher Transparency)	125
Figure 74 User Interface – User Filter – Smoothing Function Selection and Parameters	126
Figure 75 User Interface – Visualization – Emotion Probability Scores (Single Emotions).....	127
Figure 76 User Interface – Visualization – Emotion Probability Scores (Groups with Any Modifier).....	128
Figure 77 User Interface – Visualization – Entity Tonality (Single Entity, Multiple Emotions) ...	129
Figure 78 User Interface – Visualization – Entity Tonality (Multiple Entities, Single Emotions)	130
Figure 79 User Interface – Visualization – Entity Tonality (Multiple Entities, Entity Grouping, Single Emotions).....	131
Figure 80 User Interface – Database Page (Loaded People and Works)	132
Figure 81 User Interface – Database Page (File Tracking and Unprocessed Revisions).....	133
Figure 82 User Interface – About Page	134
Figure 83 Figure 81 User Interface – Settings Page.....	135

Abstract

The creative process for composing large literary works takes a lot of time. The writing itself is the largest time sink, but the editing and following revising process also takes a lot of time. This is primarily due to the fact that humans are limited in terms of the speed that they are able to read a work and then subsequently communicate their feedback to the author.

Natural language processing (NLP) is a computer science and data science topic that has garnered a lot of public interest over 2023 due to the release of tools such as ChatGPT. One of the features of NLP is the ability to perform analysis on literature. Some of the current capabilities include sentiment analysis (is a work positive, neutral, or negative), named entity recognition (what people, places, organizations, etc. are mentioned), and detecting whether something is sarcastic or not. These tools can be used to provide more immediate feedback to a writer and may even be able to notice details that experienced editors might miss. However, these tools are often only available online and require the submission of data that a user may want to keep confidential.

The aim of this thesis is to show there can be a viable application that runs locally a user's computer that can perform some natural language processing tasks. This program needs to be simple to use, perform effective analysis, and enable the end user to explore the data in meaningful ways. The target audience for this product would include writers, editors, publishers, and data scientists.

To this end, existing software libraries were examined for functionality, accuracy, and speed. Suitable components were selected and incorporated into a single tool. Existing concepts such as version control for software development and known NLP functionality were combined in new ways. That proof-of-concept tool was then used to produce sample results to show the viability of the product.

Introduction

Literary analysis comes in many forms. Simple analysis can take the form of plot summaries or character profiles. More complex analysis can involve complex comparisons of a story's themes, verification of historical events, and attempting to read between the lines to find hidden meanings. Whatever the study is, they all have a common element: the text must first be read by a human before any conclusions can be drawn.

In recent years, as computers have grown more powerful and algorithms have grown more complex, it has become possible for computers to perform complex literary analysis. Some examples of this sentiment analysis (determining whether a work is positive, neutral, or negative), sarcasm detection, detecting hate speech, and named entity recognition (identifying characters, organizations, places, name of products, etc.)

The goal of this research is to show a proof-of-concept tool that allows an end user to perform literary analysis. This tool will not replace any person in the creation of a literary work, but it should increase the productivity of the humans in that process. With that in mind, this design of this program has a few key concepts that should be adhered to.

Firstly, this tool should work offline. This is for the prime reason that many people are rightfully concerned about submitting content to some online artificial intelligence. How exactly artificial intelligences and machine learning algorithms use and disseminate data they receive as input is often obfuscated, either intentionally by the organization controlling the algorithms or unintentionally by the developers due to the complex nature of models that may be used in the development of the tool, such as a neural network. Having this as a requirement also makes the application more reliable. Internet connections are ubiquitous, but no connection is guaranteed to work one hundred percent of the time; it would not be acceptable for a large section of the program to stop working without being connected.

Additionally, the program should showcase three different types of metrics to measure a work by. Some of these metrics are easier to implement than others. The purpose of including easier metrics is to showcase what a fully developed application may look like. As many metrics as possible should have interactive data displays so that the user can find the data that they need.

The first type of metric comprises of basic statistics that are simple to compute. The purpose for including these metrics is to give the user a more complete experience; the user should have to use the fewest amount of programs to get the data they need. Some of those statistics include word counts, word frequencies, and words written per revision.

The next type of metric is slightly more advanced and not as easy to implement, but can often be found in other programs. Examples of this include the Flesch Reading Ease, Flesch-Kincaid Grade Level, number of sentences, and part of speech usage. Some natural language processing libraries offer this functionality, but are typically harder to implement with access to the said libraries.

The last type of metric should be something completely new or something that combines existing analysis in a new and interesting way. The avenues explored here will be attempting to score a named entities “presence” within a work, breaking down the emotional tone of a work on a per-paragraph or per-sentence basis, and combining entity presence scores with emotional scores to get a metric of how a story feels when specific entities or groups of entities are present.

The second key concept is that it can be difficult or impossible to change the content of a literary work when attempting to analyze it. As such, errors can occur that will degrade the effectiveness of any applied metrics. This is especially true for rough drafts where there could be many typos or incomplete ideas. Consequently, a way needs to be developed for the user to process a work and then clean the input without having to directly modify the original work. To

this end, this application will allow the user to clean the results of the named entity recognition process as an example of this functionality.

The next major concept is that the user should be able to explore the resulting data in meaningful ways. This will be accomplished by presenting the data in an interactive dashboard. Interactivity will be defined as having a chart that can zoom in and out on key data points, filter data from the display, and have other miscellaneous user options such as changing the colors or transparency on a chart. In case the user is not able to examine the data in the way that they want to, an option to export any data creates will also need to be created.

The last major concept will be applying some of the concepts of software version control software to literary works. Other applications already allow a user to save versions of their documents. This program should allow a user to apply some of the aforementioned metrics to multiple revisions of a literary work and display the differences. Furthermore, a separate windows service will be developed that will run invisibly in the background on a user's system. This service will track changes to files of interest. When the user next opens the application, they will be presented with a list of revisions that they can process.

Finally, the program should be easy enough to use for the average person. The program should also run fairly quickly. This is important as not many users are willing to wait around for hours for analysis to be done on their work. The interface should not be overly complicated.

The above mentioned tasks constitute a large problem to be solved. The work will be limited with a focus on showing all of the above concepts functioning. When possible, existing libraries will be used to speed up development time.

Metrics created should show why and how they're useful. Polishing the analytics to handle every edge case should not be the primary concern. Relatedly, machine learning algorithms are never entirely accurate in their predictions. It is possible to tune algorithms and models, but the focus should be on getting tangible results, not near-perfect results.

Tasks in the sphere of natural language processing can be boundless in scope. Feature creep is a real possibility with software like this. Any additional ideas that may add the impact of this software will have to be weighed against the cost in time it would take to add those features.

Therefore, this thesis should show mastery in programming fundamentals, software design, user interface design, database design, design and application of appropriate algorithms, and data cleaning and analysis/visualization.

To demonstrate these points, an application was developed that allows literary writers, editors, publishers, and data scientists to analyze literary works in new ways. The novel aspects of this application involve applying existing concepts of software version control to literary works, allowing an end user to easily clean data which generally can't be modified before it is input into the system, and devising new metrics that can be used to gain insight into literary works that usually require a human touch to generate. The combination of all these things produces a result that fits into a niche in software design that isn't well explored as of the writing of this document.

The results were a program that shows aspects of each objective being met. In the future, the program could be enhanced with other metrics and additional capabilities to make it even more useful to the target audience.

Background

Previous + Current Research

Computer assisted literary analysis is not a new concept. Automated analysis has been happening since the first time a programmer created a word count on a document or measured the frequency of the number of times each word appeared in a document. As with many areas of research, the level at which this analysis can be performed has increased vastly in the last decade or two. Advancements with new techniques and new models, along with greater access to increased processing power, storage, and data, has lead to a rapid rise in the number of topics in literary analysis.

“With recent leaps in Computer Science, computational literary criticism has taken on new forms and moved beyond the world of word frequency analysis. For instance, Martin Paul Eve uses information retrieval techniques to identify anachronistic language in *Cloud Atlas*, a novel in which chapters are set in different time periods. By writing a Python script to query and cross-check word origins, Eve is able to explore the relationship between historical fiction and historical accuracy in Mitchell’s novel. At the Literary Lab, the Microgenres project uses machine learning to “identify points at which authors incorporate the language and style of other contemporary disciplines into their narratives.” (What Can Computer Algorithms Tell Us about Literature?, n.d.)

A group of researches further expands what is possible by creating a list of qualitative analysis techniques. That list can be seen in Figure 1.

Type of Analysis	Short Description of Analysis
Constant comparison analysis	Systematically reducing source(s) to codes inductively, then developing themes from the codes. These themes may become headings and subheadings in the literature review section.
Classical content analysis	Systematically reducing source(s) to codes deductively or inductively, then counting the number of codes.
Word count	Counting the total number of (key)words used or the number of times a particular word is used either during a within-study or between-study literature analysis.
Keywords-in-context	Identifying keywords and utilizing the surrounding words to understand the underlying meaning of the keyword in a source or across sources.
Domain analysis	Utilizing the relationships between symbols and referents to identify domains in a source(s).
Taxonomic analysis	Creating a classification system that categorizes the domains in a pictorial representation (e.g., flowchart) to help the literature reviewer understand the relationships among the domains.
Componential analysis	Using matrices and/or tables to discover the differences among the subcomponents of domains.
Theme analysis	Involves a search for relationships among domains, as well as a search for how these relationships are linked to the overall cultural context.
Discourse analysis	Selecting representative or unique segments of language use, such as several lines of an interview transcript involving a researcher, and then examining the selected lines in detail for rhetorical organization, variability, accountability, and positioning. This analysis is particularly useful when reviewing literature review sections of empirical articles, literature review articles, theoretical/conceptual articles, and methodological articles.
Secondary data analysis	Analyzing pre-existing sources or artifacts.
Membership categorization analysis	Examining how authors/researchers communicate research terms, concepts, findings, and categories in their works.
Semiotics	Using talk and text as systems of signs under the assumption that no meaning can be attached to a single term. This form of analysis shows how signs are interrelated for the purpose of creating and excluding specific meanings.
Manifest content analysis	Describing observed (i.e., manifest) aspects of communication via objective, systematic, and empirical means.
Qualitative comparative analysis	Systematically analyzing similarities and differences across sources, typically being used as a theory-building approach, allowing the reviewer to make connections among previously built categories, as well as to test and to develop the categories further. This analysis is particularly useful for assessing causality in findings across sources.
Narrative analysis	Considering the potential of stories to give meaning to research findings, and treating data as stories, enabling reviewers to reduce data to a summary.
Text mining	Analyzing naturally occurring text within multiple sources in order to discover and capture semantic information.
Micro-interlocutor analysis	Analyzing information stemming from one or more focus groups of researchers, scholars, or practitioners about which participant(s) responds to each question, the order that each participant responds, the characteristics of the response, the nonverbal communication used, and the like.

Figure 1 Possible Qualitative Analysis for Search Syntheses (Onwuegbuzie et al., 2012)

To perform computer-based literary analysis, a few steps must be taken. First, literary works must be selected, then selectively edited, and finally analyzed. The selective editing process can be summarized as:

Some typographic elements may need to be addressed. For example, to prevent dashes from being treated as hyphens by some text-analysis software, spaces may need to be added before and after them. In most electronic texts, apostrophes and opening and closing single quotation marks are identical; this is especially problematic for dialect forms, scare quotes, quotation within quotation, and dialogue marked with single quotation marks. It may be necessary to examine every apostrophe and single quotation mark and perhaps delete each one that is not an apostrophe or replace it with a double quotation mark or acute accent. Literature written before about 1800 presents additional problems, such as variant spellings, frequent and variable editorial intervention, and a high proportion of anonymous texts and texts of doubtful authorship. (Hoover, 2013)

Data scientists routinely have to “clean data” as part of their job. This process, while nothing out of the ordinary for a traditional researcher, may be too many steps for the average person who wants to use a computer to gain insight into a work. This project will aim to create a data cleaning tool for non-data scientists to clean the data needed to perform analysis.

Kurt Vonnegut, famed author, attempted two master theses of his own. The thesis of note here is where he describes stories as having a “shape” to them.

Vonnegut submitted two Master’s theses that were both, unfortunately, rejected. In one thesis, he presented the idea that stories have a shape, based on the happiness of characters over the span of the work. For instance, a character can start the story contented, then hit a low point, only to rally at the end. Think of Cinderella or Star Wars. Or a character can start at a low place, then end up happy or sad at the end, and so on. (abonato99, 2016)

Figure 2, drawn by Vonnegut himself, illustrates some of the shapes he talks about.



Figure 2 Screenshot From a Recording of Kurt Vonnegut in 1985 Talking About The Shape of Stories (David Comberg, 2010)

The shape of three different stories as described by Vonnegut, represented by three lines on a graph.

At the times Vonnegut defended his thesis and drew this chart, computers were not able to perform the type of analysis he described. In the intervening decades, processing power has dramatically increased and many new techniques have been developed. These advances have given way for literary analysis as done by a computer to prove Vonnegut's thesis.

A team of researchers at the University of Vermont developed a technique to computationally create the shape of a story by measuring its "happiness ranking." A short summary of how this was done follows:

How does the UVM team turn the happiness of a story, or parts of a story, into numerical values? By analyzing it at the word level. To begin, they decided to create an emotional

ranking system for a large quantity of words. They arranged for 10,000 of the most frequently used words, in 10 different languages, to be rated by 50 people on a happiness scale of 1 to 9.

Why not a more conventional 1-to-10 scale? For calculation purposes, Danforth said, there had to be a neutral middle value. The words with the happiest ranking, Reagan noted, are “love,” “laughter” and “happiness.”

To plot out what Reagan called a story's “emotional signature,” the researchers calculate the happiness average of a story segment using the rankings and occurrence of particular words. For statistical viability, segments are approximately 10,000 words long. So the happiness average of pages 1 through 20 of a novel is plotted beside the average for pages 2 through 21, and so on — overlapping to provide the richest analysis possible. As Danforth put it, “the instrument is about trying to rigorously quantify differences in word usage.” (Jones, 2016)

The results of this process can be seen in Figure 3.

Harry Potter and the Deathly Hallows

by J.K. Rowling

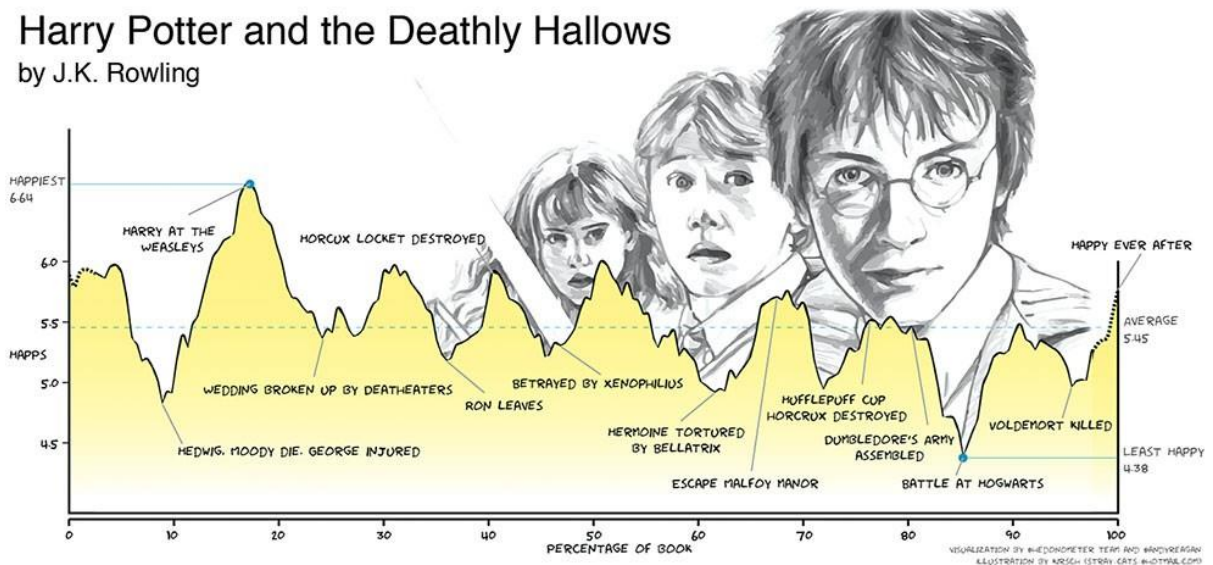


Figure 3 The "Shape" of Harry Potter and the Deathly Hallows by J.K. Rowling (Jones, 2016)

The team behind this image analyzed one thousand, seven hundred and thirty-seven stories. From the accumulated analysis, they came to the conclusion: “One of the most prominent findings of the group's research is that the emotional arc of most human narratives — defined as the trajectory of overall positive and negative feelings — fits one of six distinct shapes.” (Jones, 2016). The discrete number of shapes is tied to a specific evaluation metric. If there are other metrics that can be used to evaluate a story, does that mean other shapes are possible?

Decades ago, Vonnegut himself stated: “There’s no reason the simple shapes of stories can’t be fed into computers. They are beautiful shapes.” (David Comberg, 2010) This project aims to do just that. Others have already proven Vonnegut’s thesis, but this project does so not by producing a single shape for a story, but by producing multiple shapes for each character in the story.

Personal Tools

There are a number of publicly available tools that can assist writers with creating a literary work. Broadly speaking, the features present in these tools will be organized into six categories. Many available programs fall into multiple categories. The categories are input, planning, editing, formatting, statistical, and analysis. It could be argued that artificial intelligence assisted writing tools could be a seventh category, but seeing as these tools actually write content for the user instead of helping the user write content for themselves, this topic will not be considered as an important feature for comparison purposes.

Input tools have functionality that allow the user to actually write content. The most obvious example of this type of feature is Microsoft Word. Nearly every available writing tool has some kind of input functionality. At the basic level, this includes typing text and saving it to a file on a user’s file system. A more advanced features is saving the file to some kind of cloud-based

storage solution so that the user can access the file from multiple devices or even share the file with other users to enable collaboration.

Planning features are those that help the user plan how their work might look when it is completed. Specific examples of this category include plot diagrams, character biographies, and personal wikis. Three programs with these features are bibisco, Scrivener, and Milanote.

Editing components take text as input and output possible corrections and suggest changes to the text. A good example of this type of feature is, again, Microsoft Word with its spellchecker and basic grammar correction abilities. One of the most widely advertised editing solutions is Grammarly, which provides the user with real-time suggestions on grammar corrections, improved word choice, and tone detection.

Formatting tools allow the user to specify how the text is displayed. Bold and italic fonts, underlines, use of color, inclusion of images, writing in multiple columns, and others all transform a plain text experience into a rich text experience. Many input tools also function as formatting tools. Specialty formatting tools exist to export a work to given set of specifications. A few examples of why this would be done would be to submit a specific format needed for a publisher to physically print a book, output format for viewing on the internet, or produce a specific file format for e-book publication. Applications with this capability are Reedsy and Microsoft Word.

Statistical functions display mathematical summaries of a work. The most well known of these in terms of literature are word page, page count, word frequencies, etc. This type of data is often easy for a computer to produce. Many of the available tools have some of this type of capability.

Analysis operations require complex mathematical models and/or data structures to generate results. This is the primary area that artificial intelligence operates in. Since the release of ChatGPT, many companies are racing to add artificial intelligence capabilities to their products. This holds true for programs that have a target audience of writers.

Analysis tools assist someone in the revision of content they have already written. These tools aim to augment or replace some of the functionality of a human editor. These are the newest type of literary tool available. In recent years, many existing applications have been rushing to add this type of functionality to their products. It is quite possible that many of the programs mentioned here will contain some type of artificial intelligence backed analysis tools.

Statistical functions and analysis operations are similar enough in concept that they can be hard to differentiate. Two examples of this are the Flesch Reading Ease and the Flesch Kincaid Reading Level. Both of these metrics count the number of words, the number of sentences, and the number of syllables to give a numerical representation of the difficulty of reading said work. If a person or computer has these counts, then the math behind the calculations is basic. If this information is not available, then counting the number of syllables in a word or the number of sentences in a work is a non-trivial task. Approximations can be made, but they often use complex sets of rules or some kind of machine learning algorithms to compute the required inputs.

Program Planning Editing Formatting Statistical Analysis

	Program	Planning	Editing	Formatting	Statistical	Analysis
<i>Hemingway Editor</i>	-	X	-	X	-	
<i>Scrivener</i>	X	X	X	X	-	
<i>Writers Desk</i>	X	X	X	X	X	
<i>Milanote</i>	X	-	-	-	-	
<i>reedsy</i>	-	X	X	?	-	
<i>bibisco</i>	X	X	X	X	X	
<i>Word</i>	-	X	X	X	-	
<i>OneNote</i>	-	X	X	-	-	
<i>Evernote</i>	-	X	X	?	-	
<i>Grammarly</i>	-	X	-	?	X	
<i>Writing Analytics</i>	?	X	X	X	X	
<i>Mellel</i>	-	X	X	?	-	
<i>Atticus</i>	-	X	X	X	-	
<i>Quoll Writer</i>	X	X	-	X	-	
<i>LivingWriter</i>	X	X	X	X	-	

Table 1 Tool Feature Comparison (input category not included)

Nearly all the personal tools available are focused on editing as their primary concern.

Standalone Natural Language Tools

ChatGPT, released on November 30, 2022, is usually the first tool that comes to mind when people think of natural language tools. This is relevant when discussing computer-assisted literary analysis as people will associate and confuse the capabilities of ChatGPT with

the capabilities of a dedicated analysis tool. There are many branches of research contained within the natural language processing umbrella.

Natural language processing is “a branch of AI that allows more natural human-to-computer communication by linking human and machine language.” (What Is the Difference between NLP, NLU, and NLG?, n.d.)

Machine translation deals with the task of translating input from one language into a different language.

Natural language understanding “processes input data and can make sense of natural language sentences.” (What Is the Difference between NLP, NLU, and NLG?, n.d.)

Natural language generation “builds sentences and creates text responses understood by humans.” (What Is the Difference between NLP, NLU, and NLG?, n.d.)

ChatGPT has capabilities from a number of different NLP subfields. ChatGPT is also easy enough to use that guides appear on popular websites.

However, there are many trust issues with such a large artificial intelligence, including what the AI and the company behind the AI does with all the information that people are constantly submitting to it. The newer tools are often designed to be easy to use, and thus can potentially collect a lot of information. For example, one guide describes an easy to replicate process to:

1. Search for articles written by a specific author
2. Summarize those articles
3. Critique the author's writing style (Westley, 2023)

Version Control

In the software development world, there exists a type of software called reversion control. The purpose of version control is to allow a group of people, ranging in size from a

whole organization or the general public to just a single individual, to contribute to a project. Some of the primary features include rolling back existing files to previous version, identifying who made changes to specific parts of the project, providing secure file storage, allow (or restricting) access to the files, and allowing multiple people to work on the same file at the same time.

Aspects of version control can easily be applied to literary works. The ability to go back and see what a previous version of a work can be very helpful. However, not many analytics are able to compare between revisions.

In total, this program aims to tackle multiple problems, with the exact combination of problems being one of the unique aspects of the difficulty.

Methodology

This section will describe the data used to produce results, open source projects that the application utilizes, the models behind the data analytics along with their corresponding sources of data, and techniques that were developed to enable the desired functionality set forth in the goals. Additionally, decisions made about the design of the tool will be discussed.

General Design Decisions

Use of the .NET MAUI Framework for the User Interface and Model

Even though a core part of the data analysis would be performed using the Python programming language, creating a modern application with a user interface that would appeal to users in the current day would be difficult using just Python. As such, the decision was made to use the .NET MAUI Framework to handle the user interface as well as the model. The .NET MAUI framework, while a newer solution, has a number of tools that greatly simplify user interface programming. The primary purpose of MAUI is to allow developers to create a single codebase from which they can create an application that runs on the Windows, Android, iOS, and Mac operating systems. If that framework was not sufficient in its capabilities, there was also the ability to fallback on the similar framework provided by WinUI 3.

Additionally, programs made with .NET often execute much faster than those written purely in Python. As this application is designed to be heavily interacted with by users, the speed of the program was one of the top concerns.

The making of this decision was aided by the availability of the .NET MAUI Community Toolkit's implementation of the Model-View-Viewmodel (MVVM) architecture pattern. The MVVM pattern allows for a decoupling of the user interface from the rest of the application. The

pattern sped up development time as well as helped organize the code of the program. The pattern was also extended to the Model-View-Viewmodel-Services (MVVMS) as detailed by Microsoft employee James Montemagno.

The other alternative to .NET MAUI would have been to use WinUI3. WinUI3 has been around for a longer period of time than .NET MAUI and has more resources available to support it. Being newer, .NET MAUI also has a lot of issues that are typically associated with newer software packages. One of the original “wish list” items for this project was to be able to access the data from multiple platforms, such as an Android phone. Early on in the build process, it was discovered that this would be too difficult to do in a short period of time while still implementing the amount of features needed. If the intention was to completely stop work on this project after the thesis is presented, hindsight says that the better decision most likely would have been to go with WinUI3 as the application only works on Windows. However, the application being built on .NET MAUI leaves open a lot of doors for future development and may reduce work in the long run.

Model-View-Viewmodel-Services

The major alteration to the MVVM pattern that MVVMS makes is the addition of Services portion. Services take on the role of a lot of functionality that was previously located in a Viewmodel or Model. An example of this might be a database service. Instead of having database access code placed throughout the program, all the database code can be centralized in one location that other parts of the program can call into. The migration of these functions results in the Model portion being reduced mostly to mimicking the structure of the program's data. Similarly, a lot of the code that would have previously been included in the Viewmodel can now be placed into a Service. This can potentially increase the amount of code that is reused and reduce the overall complexity of the codebase.

SQLite3 as the Database

One of the earliest design choices made about the structure of the application was to use a database for storage purposes instead of reading files every time the user wanted to access something. SQLite3 was chosen as the database due to how portable and lightweight it is.

Data Used

Sample data for this project was gathered from Project Gutenberg, a website that provides access to literary works for which their copyright in the United States has expired. Most of the content available from the website is in the public domain, and all the literary works used as input data are in the public domain.

Works were chosen to illustrate the ability to operate on different types of literature.

Specifically, the following works were used:

- The Divine Comedy by Dante Alighieri
 - Reason: A narrative poem written a long time ago.
- The Wonderful Wizard of Oz by L. Frank Baum
 - Reason: Prose written for children.
- Alice's Adventures in Wonderland by Lewis Carroll
 - Reason: A revision of Alice's Adventures Under Ground.
- Alice's Adventures Under Ground by Lewis Carroll
 - Reason: Precursor work to Alice's Adventures in Wonderland.
- The Hunting of the Snark: An Agony in Eight Fits by Lewis Carroll
 - Reason: A nonsense poem.
- Through the Looking-Glass by Lewis Carroll
 - Reason: Related to Alice's Adventures in Wonderland.
- A Christmas Carol in Prose; Being a Ghost Story of Christmas by Charles Dickens

- Reason: A novella.
- Grimms' Fairy Tales by Jacob Grimm and Wilhelm Grimm
 - Reason: A collection of short stories.
- The Scarlet Letter by Nathaniel Hawthorne
 - Reason: A novel.
- Winnie-the-Pooh by A. A. Milne
- The Cask of Amontillado by Edgar Allan Poe
 - Reason: A short story.
- The Fall of the House of Usher by Edgar Allan Poe
- The Raven by Edgar Allan Poe
 - Reason: A poem.
- The Tell-Tale Heart by Edgar Allan Poe
- Hamlet, Prince of Denmark by William Shakespeare
 - Reason: A play.

The content of each work was extracted from the available download links as there was no need to perform analysis on the Project Gutenberg License or “Also Written By The Author ...” sections.

Open Source Projects

A number of open source projects were utilized to add functionality not available to the base installation of the .net MAUI Framework. Unless otherwise noted, all projects utilize the MIT license and were used unmodified.

- .NET MAUI Community Toolkit
 - Purpose: Built in support for additional MAUI controls, behaviors, converters, and the Model View Viewmodel pattern.

- SQLite-net
 - Purpose: Enable support for using a SQLite3 database.
- SQLite Portable Class Library
 - Purpose: Additional support features for SQLite3.
 - License : Apache-2.0
- LiveCharts2
 - Purpose: Chart control for displaying data.
- SpacyDotNet
 - Purpose: Access to the Python spaCy natural language processing library.
This library uses Python.NET which enables interoperability between the C# and Python programming languages.
 - Modified to work with a different version of Python, with the .NET MAUI Framework, and with additional spaCy pipelines.
- Json.NET
 - Purpose: Provides access to serialization and deserialization of C# objects to the JSON format.
- Maui.DataGrid
 - Purpose: Provides a control that can display tabular data.
- ColorPicker.Maui
 - Purpose: Provides a control that allows the user to select from a range of colors.

Natural Language Processing Libraries and Models

spaCy and Spacy Syllables

The spaCy library was chosen to handle the majority of the application's natural language processing needs. As spaCy is written in Python and the thesis project is written in C#, the open source project SpacyDotNet was used so that C# objects could be created to access the corresponding objects in Python.

spaCy is an established library (having been first released in 2015), has gone through multiple versions, and is considered to be a stable software package. Out of the box, spaCy is capable of handling many of the tasks that this project requires. These features include, but are not limited to, tokenization of input text, sentence segmentation, named entity recognition, part of speech tagging. This is all done invisibly to the end user via the use of convolutional neural networks.

Additionally, the spaCy library has access to multiple, prebuilt language models on which the above functionality can run. This is vitally important as building a new model is far beyond the scope of this project.

The functionality of the spaCy library can also be extended. This project required the detection of syllables in words which is not a built in feature. The Spacy Syllables pipeline was selected to provide this functionality. The SpacyDotNet package was extended by the author of this thesis to handle this pipeline, giving direct access to that data and functionality from within C# code.

In this application, spaCy runs the `en_core_web_lg` pipeline. The form of the data source is written text (blogs, news, and comments.) This pipeline has data sources of OntoNotes 5, ClearNLP Constituent-to-Dependency Conversion, WordNet 3.0, and Explosion Vectors (OSCAR 2109 + Wikipedia + OpenSubtitles + WMT News Crawl.)

In terms of evaluating the this pipeline, the following evaluation metrics are given on the spaCy website:

<i>metric</i>	Tokenization	Part of Speech Tagging	Sentence Segmentation	Named Entities
<i>Accuracy</i>	1.00	0.97	-	-
<i>Precision</i>	1.00	-	0.92	0.85
<i>Recall</i>	1.00	-	0.89	0.86
<i>F-score</i>	1.00	-	0.91	0.85

Table 2 spaCy en_core_web_lg pipeline evaluation metrics

The features of sentence segmentation and named entities will especially important to this project. Thankfully, the scores are relatively high. No model is perfect, but this should provide sufficient accuracy. Another point to consider when looking at these scores is that spaCy is actively being developed. This means that there is a good chance that these scores go up even further in future as new advancements are made.

Emotion Classification Models

Creating a model that will determine what the emotional tone is in a given text is beyond the scope of this work. Existing models were evaluated for accuracy, ease of use, and execution speed. Eventually, two models were used in the program, with one being discarded.

Both models use the GoEmotions dataset. This datasets consists of just over fifty-eight thousand comments from the website Reddit and was collected by Google. These comments were then annotated to say which of twenty-seven different emotional categories the comment embodies. An additional twenty-eight category of “neutral” was also included. The twenty-seven emotions annotated in the dataset are: admiration, amusement, anger, annoyance, approval, caring, confusion, curiosity, desire, disappointment, disapproval, disgust, embarrassment, excitement, fear, gratitude, grief, joy, love, nervousness, optimism, pride, realization, relief, remorse, sadness, and surprise.

The two models examined were EmoRoBERTa and roberta-base-go_emotions. Both models can be used in the same way (down to identical function calls in Python, albeit with different parameters used.) Both models take the same type of input data, and the format of their output data is identical. This made it easy to compare the models. Another commonality is that both models seem to use neural networks to help them come to their decisions.

Before the models could be used, the data needed to be changed into a format that they could process. The models have an upper token limit of five hundred and twelve tokens. This means that long passages of text cannot be processed all at once. Neither model can be used directly to get the overall feeling of a literary work if significant length.

The solution to length problem was to break down the work into smaller chunks. Given the nature of the subject, two groupings were immediately apparent: sentences and paragraphs. Most paragraphs and almost every sentence would meet the length requirement. If an item is too large for the model to handle, the model has been instructed to truncate the input to the first 512 tokens.

A distinction needs to be made between words and tokens. The tokens of a sentence comprise more than just the simple words. They also may contain punctuation and excess whitespace. Tokens from the spaCy library also contain other information, such as a lemma. A lemma is basically the base form of a given word. For example, the words do, does, did, and doing all share the same lemma which is the word do. Due to this complexity of the English language, a sentence will often contain more tokens than it does words.

Revision Tracking

Revision tracking of documents isn't a new idea. Scrivener and OneNote both allow a user to go back and view previous versions of documents. Version control in those programs are limited to files created in that program. This program takes an approach that aligns more

with the software development idea where it can track changes to files created by other programs.

The main purpose behind revision tracking is that many users would not want to manually load in a new file to the application whenever they have completed a new revision. Also, many people do not create different files for ever revision they make. The inclusion of revision tracking will reduce the cognitive load required to use the program.

Revision tracking was implemented as a separate windows service. The service operates invisibly to the user. The service connects to the same database as the main program.

In the main program, any time a new revision is manually added by a user, the file path and file name are tracked. The user has the ability to toggle the tracking for and file on or off from within the main program on a separate database page.

While the service is running, it watches for any content changes to activate files. If a change is detected, it saves a copy of the contents of the file in the database for later processing. This is called an unprocessed revision.

The Works page and the database page of the main program both allow the user to see unprocessed revisions. On the works page, the user has the ability to process one revision at a time, or they can also process all revisions for a specific work. On the database page, the user is able to see a list of all revisions for all works and can process them one at a time in any order they wish. They can also remove an unprocessed revision

Data Cleaning Tools for Users

When data scientists perform analysis they often spend a lot of time cleaning the data before they can actually perform their analysis. As this program aims to be a tool for data analysis, any input data should technically be cleaned before it is used. However, the use case presented here prevents users from making changes to the input data. A writer wants to analyze

the data as written, not their next revision. An editor's job is to revise the data, but they cannot change the original data that will be input into this program.

The solution to this problem is to allow the end user to perform some type of data cleaning after the data is input but before it is analyzed. As most of the people this program is targeted at probably won't have data science, computer science, or mathematics degrees, this poses a large problem. A well-designed interface should be intuitive to use. User interfaces that are easy to use are usually simple. A simple interface may not provide enough functionality to the user. The objective here is to design a simple, easy to use interface, that is powerful enough to be effective.

A single method of data cleaning was developed to illustrate the above concepts. The metrics targeted are among the most important in this project, those being named entity presence scores and named entity tonality.

This feature is needed because the named entity recognition capability of the spaCy library, like most machine learning implementations, is not perfect. It can fail to detect named entities and (false negatives) it can improperly classify things as entities when they shouldn't be (false positives.) The named entity data is gathered every time the user attempts to show a visualization with entity presence or entity tonality. Any entities are added to a named entity correction table in the database if they are not already present.

The user needs to be able to correct these mistakes in order for any metric that depends on named entity detection to work properly. A user is able to summon a popup on any work to display the named entity correction popup as seen in Figure 4. This interface gives four key abilities to the user.

First, the user is able to ignore any entity that has been falsely identified as a named entity. They do this by simply toggling the ignore button next to an entity on or off.

The user can also just remove any named entity from the list completely. While this list is regenerated every time, this can be useful in situations where an entity was included in an early

draft of a work that the user does not want to run any metrics on again. Removing the item will not prevent any functionality of the program from working in the future, but it does allow the user to reduce the amount of entities displayed to them.

Similarly, the user can just create a new entity. A new entry in the database is created with a default name “?” that the user must change. It functions identically to any correction created by the program.

Lastly, the user able to select an entity, click the drop in the Alias For column, and select another entity correction. Any presence score attributed to the entity in that row will then be given to the item pointed to in the Alias For column. For example, entities named “Bob” and “Bobby” may both alias into an entity named “Robert.” Any item that aliases into another item this way itself becomes intelligible to be the target of an alias for. Any item that is ignored also becomes ineligible. Figure 5 shows the popup that is displayed when a user changes the “alias for” selection.

Name	Alias For	Ignore	Remove
Amontillado	-	Ignore	Remove
Luchesi	-	Ignore	Remove
Sherry	-	Ignore	Remove
impune laccessit_	-	Ignore	Remove
puncheons	-	Ignore	Remove
De Grâve	-	Ignore	Remove
he!—yes	-	Ignore	Remove
Lady Fortunato	-	Ignore	Remove
Montessor	-	Ignore	Remove
Fortunato	-	Ignore	Remove

Figure 4 Named Entity Corrections Interface

This shows the Named Entity Corrections Interface being used on *The Christmas Carol* by Charles Dickens.

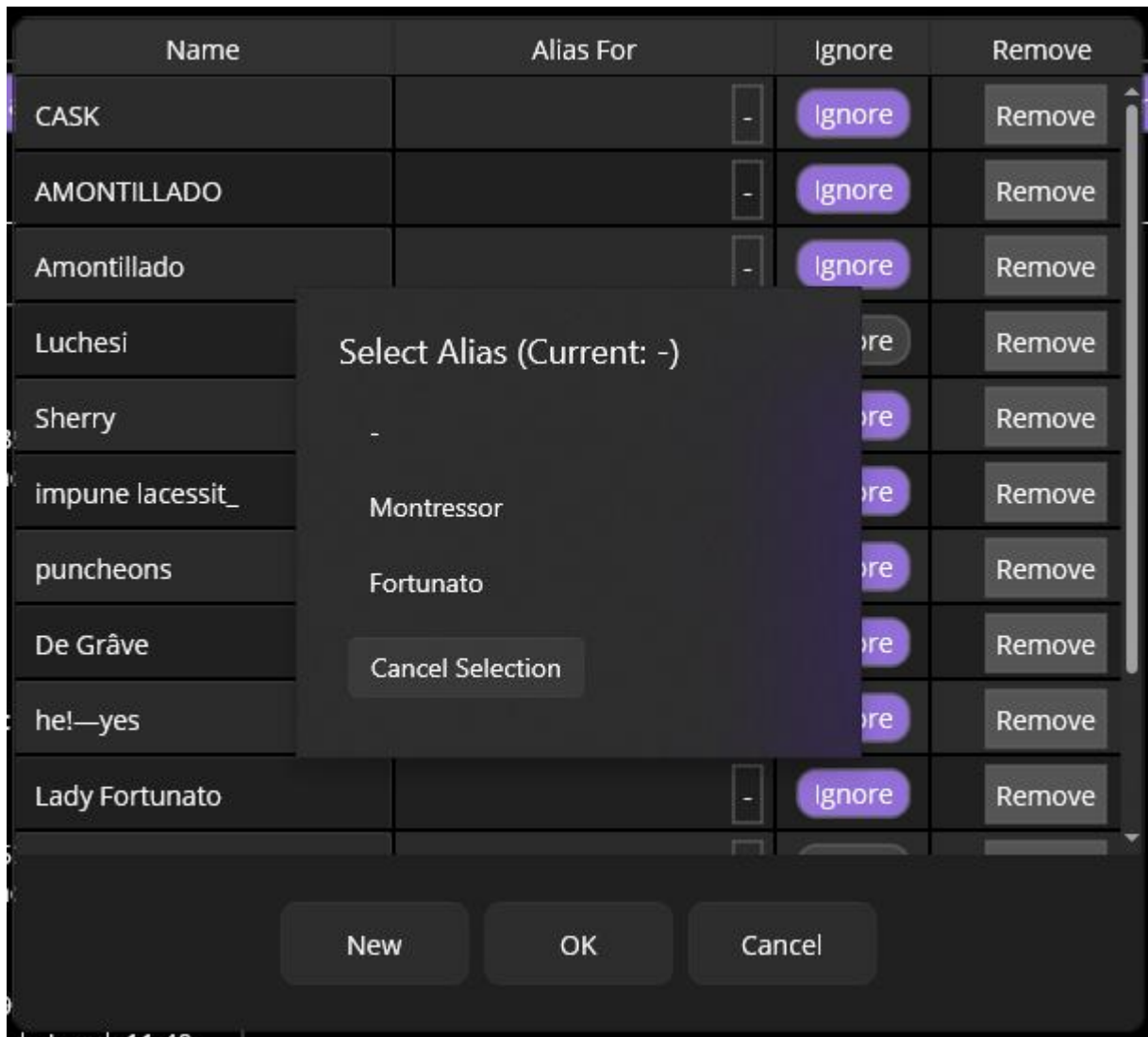


Figure 5 Named Entity Correction Interface - Alias Selection

The user is selecting an alias for the entity *Luchesi*. The only entities that are not ignored are “*Montressor*”, “*Fortunato*”, and “*I*.” However, “*I*” is an alias for “*Montressor*” and thus is not eligible to be aliased into. The no selected alias option of “-” is always available.

Interactive Visualizations

One of the core features of this project is to allow an end user to explore data in a meaningful way. This was addressed in two specific ways.

Chart Control

The most important part in displaying the results of the analysis to the end user will be through the use of a chart. As such, the chart control used should be robust in features and easy to use for the user.

Just because a chart control has good features doesn't mean the final charts displayed will be good quality. Designing good-looking charts is the responsibility of the programmer combining the data and the chart control. With this in mind, a chart control with a lot of features will greatly aid a programmer. Desired features of a good chart control include being able to display multiple types of charts (line charts, bar charts, scatter plots etc.), being able to display multiple data series, easy editing of the x-axis, y-axis, legend, and title, the ability to customize the chart with color, the ability for the end user to interactively pan and zoom the chart to explore the data with greater precision, being able to view the exact values for items on a chart through a tooltip at the mouse's position, having the feature to save a chart to a file for viewing at some point in the future, and the ability to update a chart with new data.

The LiveCharts2 project meets all of these requirements, and is also an open source project, and thus was selected to be the chart control for this project.

User Filters

In addition to the chart described above, the end user is able to add or remove filters to the displayed chart. This allows the user to display the chart that is most useful to them.

This functionality is often part of what is called a data dashboard. The included user filters include: the ability to set the first and last item that will be displayed, changing the maximum number of items displayed, enabling or disabling any functions that might alter the data (e.g. a smoothing function), hiding or showing the legend, changing the foreground and

background colors, and adjusting the fill color (i.e. whether the area under a line chart is a solid color, blank, or somewhere in between.)

The LiveCharts control can be databound to an object to be used as a data series. Whenever the user updates a filter, an update function will be called and the databound values will be updated.

To make the program responsive to the user's requests, any data produced by an analytic was stored but will not displayed to the user. Instead, updates to the filters generate a new series that is displayed.

As a new array of series gets created every time the user adjusts a filter, the user's current zoom and pan settings must be reapplied when the new series is databound. This is as simple as saving the current values before the data is changed and reapplying them after the data is changed.

When the user selects a new color to be used for either the foreground color (the lines on a line chart, the bars on a bar chart) or the background color, multiple things can happen depending on the visualization type.

For visualization that will only ever display a single data series (e.g. the word count change for all revisions belonging to a work), the exact color the user picked will be used.

For visualizations where only a few data series will be shown (e.g. part of speech comparison between the latest revisions of two different works), the user selected color is shown for the work that originates the visualization. For the data gathered from the second work, another shade of the same color is chosen. If the chosen color is closer to white than black, then a darker shade is shown, otherwise a lighter shade is shown.

For visualizations where multiple data series can be shown (entity presence, emotion probabilities, entity tonality) a predefined color palette is used. (e.g. the first color shown is blue, the second color shown is orange, etc.)

Any visualization that has more than one data series or a single data series that needs a description contains a legend. The ability to toggle the legend on and off involves changing the value of a single variable and was simple to implement.

The first item, last item, and max number of displayed items simply take the adjusted data and select the appropriate range to transferred to the displayed values.

Since the range on the data can change drastically from item to item, the user has the option of adding a smoothing function to the data before it is displayed. The user currently has two choices of smoothing function, though not ever visualization or data series has access to both functions.

The centered rolling average smoothing function adjusts the value for a given item by looking at a parameterized number of adjacent items, averaging all their values, and sets the value to the average. The user is able to adjust the parameter to any integer in the range one to fifty (inclusive of the endpoints) and the default value is 5.

If the examined values are at the beginning or end of the item group, dummy values are created so the number of displayed items stay the same. This was done by creating an interpolation using the first or last item and zero. The resulting dummy values are in the range of (0, first item) or (last item, 0). For example, if the number of items is set to 4, and the first value is 1, the numbers 0.2, 0.4, 0.6, and 0.8 would be added to the beginning of the list.

The following visualizations help show this algorithm in action. The data for this example consists of a number of paragraphs comprising a single sentence each. The sentences are either “this is a sentence” or “This is a sentence with Bob in it.” Figure 6 shows the base values. Bob is present in items 3, 13, 19, 22, and 23. Each time Bob is present, a score of 1 is given. Figure 7 shows what the centered rolling average looks like with a parameter value of 2. Figure 8 shows what the centered rolling average looks like with a parameter value of 5.

Entity Presence (Revision, Paragraph): Entity Presence Demo by Thesis Author



Figure 6 Entity Presence, No Smoothing

In this demo of entity presence, Bob is present in items 3, 13, 19, 22, and 23.

Entity Presence (Revision, Paragraph): Entity Presence Demo by Thesis Author

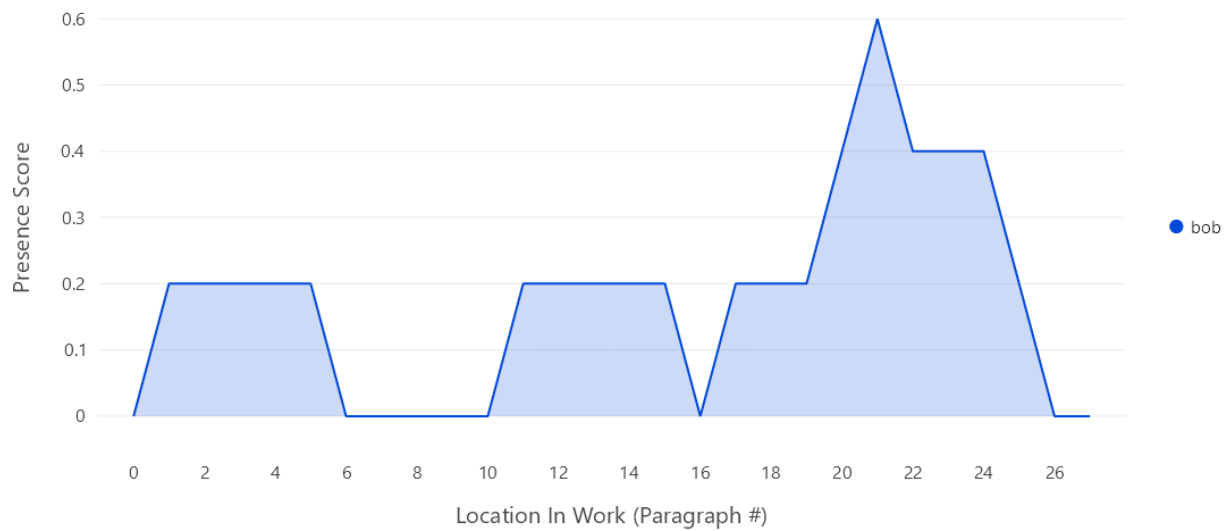


Figure 7 Entity Presence, Smoothing (Centered Rolling Average: 2 items)

In this demo of entity presence, Bob is present in items 3, 13, 19, 22, and 23.

Entity Presence (Revision, Paragraph): Entity Presence Demo by Thesis Author



Figure 8 Entity Presence, Smoothing (Centered Rolling Average: 5 items)

In this demo of entity presence, Bob is present in items 3, 13, 19, 22, and 23. A large value for the Centered Rolling Average algorithm shows the character Bob to be present throughout the entire story.

The lookahead smoothing function adjusts surrounding values based on the center item and previous known values. Starting at the first item and moving toward the last item, two steps are taken. There are two parameters for this function: the number of lookback steps and the number of lookahead steps. The user is able to adjust the parameter to any integer in the range one to twenty (inclusive of the endpoints.) The default value for the lookback parameter is two for paragraphs and five for sentences. The default value for the lookahead parameter is five for paragraphs and twenty for sentences. The lookahead smoothing function can only be applied to entity presence scores.

The first step is the lookback step. From the current value, the algorithm attempts to move backward a number of steps equal to the lookback parameter. The item the algorithm stops on will be the lookback item. If an entity is found at a position, the lookback stops one item short of the entity. If the start of the items is found, the lookback stops at the first item. Then any items between the current item and lookback item are overwritten. The new values will be evenly distributed to create a constant increase from the value of the lookback item to the value

of the current item. For example, if the lookback item has a value of 0.7, and the current item has a value of 1, with two intermediate values, the final set of values from the lookback item to the current item will be 0.7, 0.8, 0.9, and 1.

The second step is the lookahead step. From the current value, the algorithm attempts to move forward a number of steps equal to the lookahead parameter. The item the algorithm stops on will be the lookahead item. If an entity is found at a position, the lookahead stops one item short of the entity. If the end of the items is found, the lookahead stops at the last item. Then any items between the current item and lookahead item (inclusive of the lookahead item) are overwritten. The new values will be evenly distributed to create a constant decrease from the value of the current item to the value of 0. For example, if the current item has a value of 1, and there are 5 lookahead steps, the final set of values from the current item to the lookahead item will be 1, 0.8, 0.6, 0.4, 0.2, and 0.

Additionally, if an entity was found, the algorithm proceeds as normal up to the point where the entity was found. The value of entity is then increased by what would have been the new value. For example, if the current item has a value of 1, and there are 5 lookahead steps, but an entity with a value of 1 was discovered on the third lookahead step, the final set of values from the current item to the lookahead item will be 1, 0.8, 0.6, $1+0.4=1.4$, ignored (would have been 0.2), and ignored (would have been 0).

The following visualizations help show this algorithm in action. The data for this visualization is the same as for the previous example illustrating the centered rolling average smoothing function. Figure 6 (shown previously) illustrated the base values.

Figure 9 shows the default values for the lookaround function. The entity at position 3 has a value of 1. The lookback parameter of 2 sets the values of items in range (1, 3) to be in between the values of the items at positions 1 and 3. The lookahead parameter of 5 overwrites any values in the range (3, 8] and decreases them to 0.

The same process happens for the entity detected at position 13. Nothing different happens here.

The process happens again for the entity found at position 19. This time, the lookback portion of the algorithm finds a value of 0.2 at position 17. The item in position 18 is then set to 0.6, halfway between the values of 0.2 for position 17 and 1 for position 19.

When the lookahead portion of the algorithm happens, it finds an entity at position 22. The lookahead step would have set this to a value of 0.4, but it instead increases the current value of 1 by 0.4 for a total presence score of 1.4.

The last point to illustrate is when the algorithm starts from position 23. The lookback portion finds an entity in the previous position. Since the range of (22, 23) produces no numbers, the lookback step has no effect. The lookahead step hits the end of the item range, and simply stops at that point.

Figure 10 and Figure 11 illustrate the same process on the same data, but with different values for the lookback and lookahead parameters.

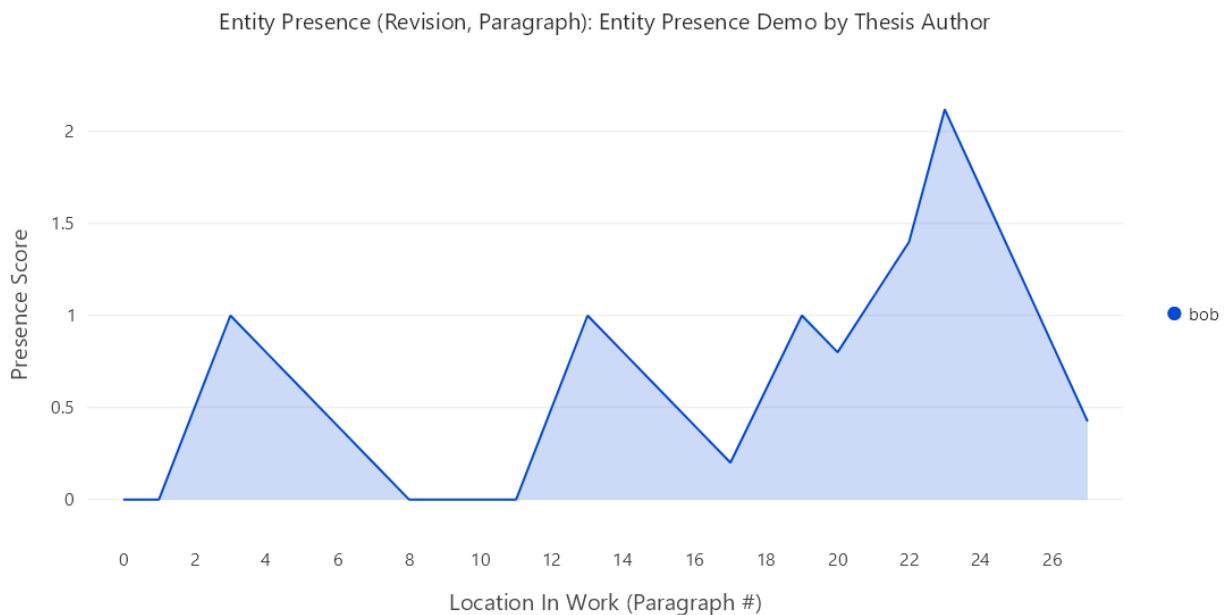


Figure 9 Entity Presence, Smoothing (Lookaround: 2 Back, 5 Forward)
In this demo of entity presence, Bob is present in items 3, 13, 19, 22, and 23.

Entity Presence (Revision, Paragraph): Entity Presence Demo by Thesis Author

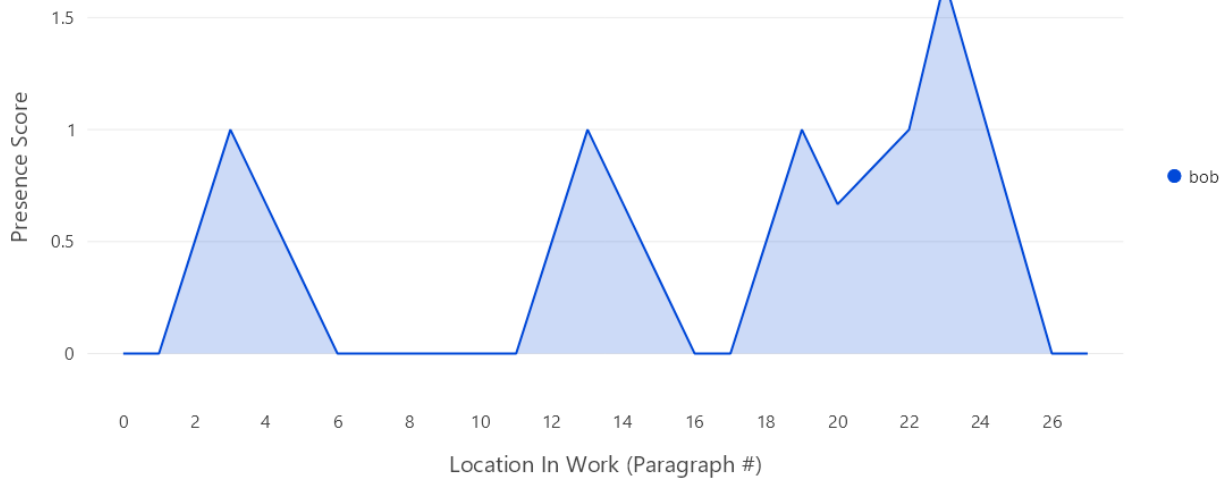


Figure 10 Entity Presence, Smoothing (Lookaround: 2 Back, 3 Forward)

In this demo of entity presence, Bob is present in items 3, 13, 19, 22, and 23.

Entity Presence (Revision, Paragraph): Entity Presence Demo by Thesis Author

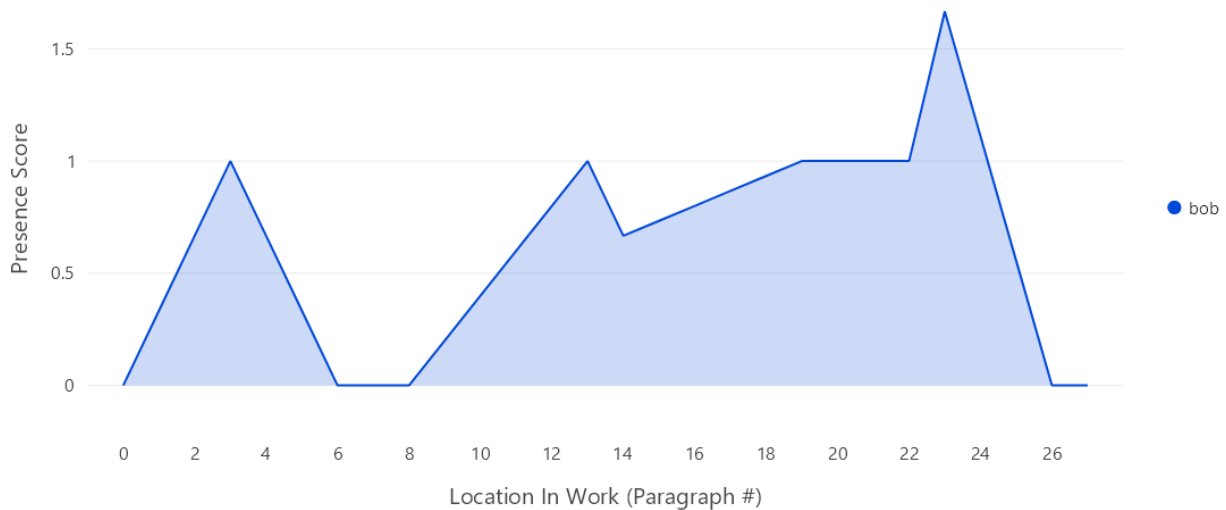


Figure 11 Entity Presence, Smoothing (Lookaround: 5 Back, 3 Forward)

In this demo of entity presence, Bob is present in items 3, 13, 19, 22, and 23.

This method allows multiple mentions of an entity to boost the score over a value of 1.

Data Analytics for Users

This section describes the algorithms used to compute the values shown to the user. Basic metrics, while simple to understand and implement, have been included for completeness. The simpler metrics are included toward the end of this section.

Entity Presence Scores

The entity presence value is a measure of how confident the program is that a given named entity is present in a story. This means the character is in the current scene. They could be talking to another entity, thinking to themselves, performing actions, etcetera. The entity does not actually have to be present with others for the algorithm to detect them though. If two entities are talking about a third, then that third entity is considered to be present. No distinct is made between the two types distinct types of presence.

To calculate the entity presence scores for a given work, first the work is split into item groups where the items are either paragraphs or sentences. Then, a list of entities is created. Named entity recognition is run on the document to produce the initial list of entities. This list is then modified by any named entity corrections the user has put in place. Ignored entities are removed from the list. The item groups are then searched to see if they contain the names of any entities. If an entity is present in an item, it is given a score of 1, otherwise it is given a score of 0. Entities that are an alias for another entity simply combine their score with the item they're aliasing into. Finally, before the scores are displayed to the user, any smoothing functions the user has selected are applied to the data.

Additionally, groups of entities can be created. This is done by simply averaging the presence score of each member in the groups. One parameter for grouping include the size of the group (which has been limited to 6 for efficiency and readability.) Another parameter specifies if the final group must contain all of the selected members or if any group that contains

at least the specified number of selected entities is valid. In either case, to prevent potentially dozens or hundreds of groupings from being created, non-selected entities are excluded from the created groups.

Entity Presence Scores are displayed as a line chart to the user. The user has the option to toggle the display of any entity in the final data series on or off.

Emotion Classification Probability Scores

The emotion probability score is a measure of how confident a given block of text conveys the emotion described by its label. A block of text is fed as an input to the model and a dictionary of twenty-eight emotions (actually twenty-seven emotions and a neutral label) and their corresponding values is returned.

Note that the probability scores do not need to add up to a value of one. It is possible that the input text can contain more than one emotion. Figure 12 shows a small selection of the available emotions in a small range for The Cask of Amontillado by Edgar Allan Poe.

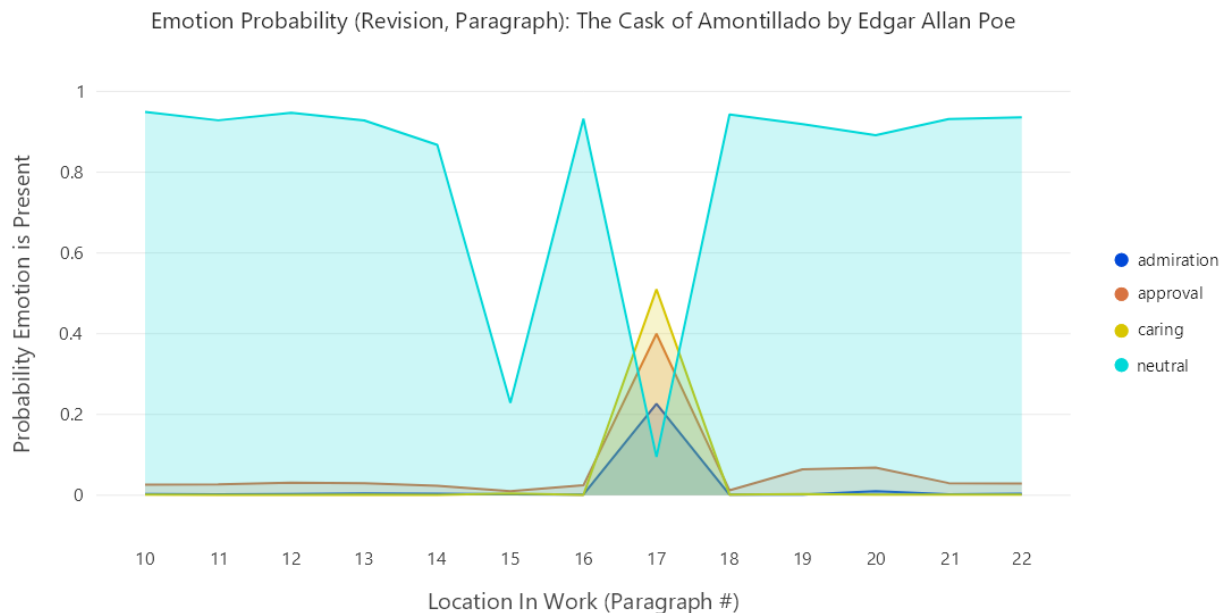
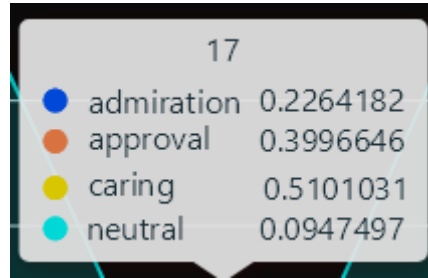


Figure 12 Emotion Probability – Multiple Emotions For A Single Item Showing just three of the emotions and the neutral classification of a particular point in The Cask of Amontillado by Edgar Allan Poe

This shows that there are multiple emotions present for item group seventeen. A further examination using the interactivity of the chart by showing the tooltip for that item groups (as shown in Figure 13) reveals that the sum of these three items exceed the value of 1.



*Figure 13 Emotion Probabilities – Tooltip
Emotion probabilities for a Specific Moment in The Cask of Amontillado by Edgar Allan Poe*

Emotion Classification Probability Scores are displayed as a line chart to the user. The user has the option to toggle the display of any emotion in the final data series on or off.

Named Entity Tonality

Named entity tonality is the combination of named entity presence scores with emotion classification probabilities. The calculation for a named entity tonality score is fairly basic. The algorithm take in as input two lists of data of equal length, an entity presence score and an emotional probability score. A minimum function is applied to every element the entity presence score so that each element in the score is in the range [0, 1]. This can be interpreted as the algorithm being fully confident that an entity is present at a score of 1 or higher, and only partially confident if has a value less than 1. This directly impacts the next step where each element of the presence values is multiplied by the corresponding element in the emotion probability score. As both sets of numbers have a minimum of 0 and a maximum of 1, this yields another number in the range [0, 1].

Named Entity Tonality is displayed as a line chart to the user.

Flesch Reading Ease

The Flesch Reading Ease score, developed by Rudolf Flesch, is a measure of how difficult a given work is to understand. The work must be written in English as the values in the formula are specific to the English language. The formula for the Flesch Reading Ease score is:

$$\text{Flesch Reading Ease} = 206.835 - 1.015 \left(\frac{\text{total words}}{\text{total sentences}} \right) - 84.6 \left(\frac{\text{total syllables}}{\text{total words}} \right)$$

Typical scores for the Flesch Reading Ease range from 0 to 100 but can go as high as 121.22 (calculated from a work with a single word, sentence, and syllable.) Scores much lower than 0 are possible with highly complex sentences. Thus, a higher score indicates the text is easier to read than another text with a lower score.

The spaCy Syllables pipeline for spaCy library is capable of adding syllable counts to the tokens generated by spaCy. Retrieving this information becomes a simple task, and the computation of the Flesch Reading Ease score becomes simple.

The Flesch Reading Ease is displayed as an inline statistic when selecting a revision and needs no visualization.

Flesch-Kincaid Grade Level

The Flesch-Kincaid Grade Level score, developed by Rudolf Flesch and J. Peter Kincaid, is another measure of how difficult a given work is to read. This metric uses the same input parameters as the Flesch Reading Ease test, but calculates the score using a different formula. The formula for the Flesch-Kincaid Grade Level is:

$$\text{Flesch - Kincaid Grade Level} = 0.39 \left(\frac{\text{total words}}{\text{total sentences}} \right) + 11.8 \left(\frac{\text{total syllables}}{\text{total words}} \right) - 15.59$$

The lowest score possible is 3.4 (calculated from a work with a single word, sentence, and syllable.) Scores represent the average grade level needed in the United States to readily

understand the text; lower scores indicate the text is easier to read than another text with a higher score.

As this metric depends on the same input as the Flesch Reading Ease, this metric can be calculated as soon as the Flesch Reading Ease metric can be calculated.

The Flesch-Kincaid Grade Level is displayed as an inline statistic when selecting a revision and needs no visualization.

Paragraph Count

Computing the paragraph count requires the user to specify how paragraphs are delimited. The two options available are detecting a newline character at the end of a sentence and detecting a completely blank line (where multiple blank lines in a row are considered to be a single blank line.) This step must be performed as different works have different ways of storing their data.

If the first option is used and the document also uses extra blank lines between paragraphs, the extra blank lines will be condensed into a single newline character.

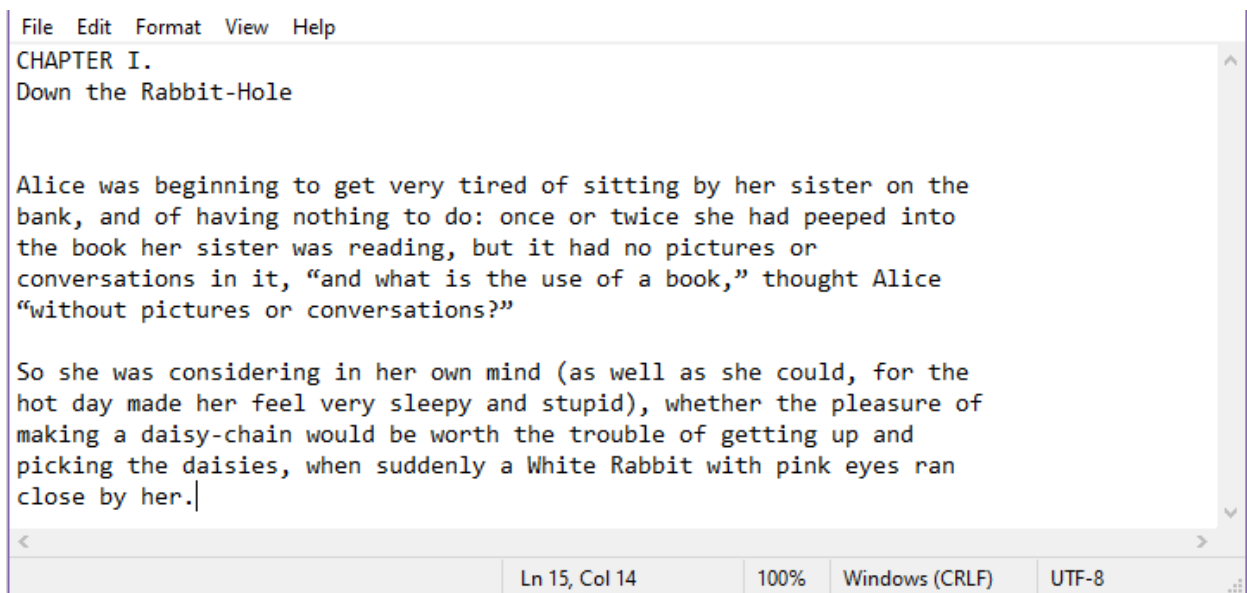
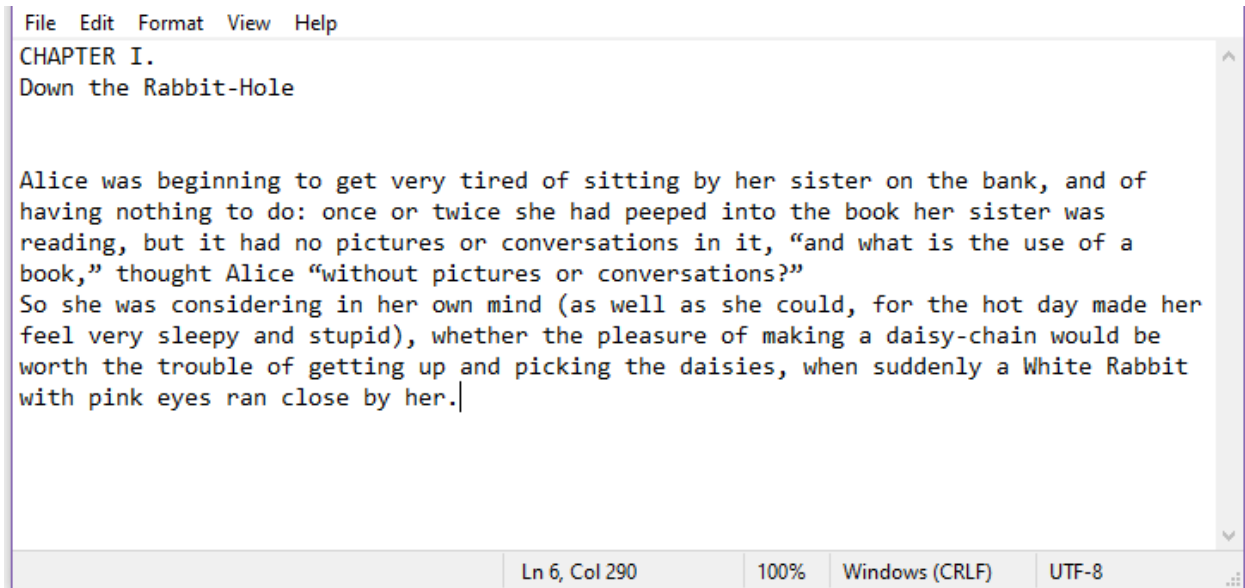


Figure 14 Sentence Detection – Completely Blank Line

The beginning of *Alice in Wonderland* by Lewis Carroll. The text is formatted to have a maximum number of characters on each line and a newline character is used to move text from line to line. A completely blank line is used to delimit the paragraph.



```
File Edit Format View Help
CHAPTER I.
Down the Rabbit-Hole

Alice was beginning to get very tired of sitting by her sister on the bank, and of
having nothing to do: once or twice she had peeped into the book her sister was
reading, but it had no pictures or conversations in it, "and what is the use of a
book," thought Alice "without pictures or conversations?"
So she was considering in her own mind (as well as she could, for the hot day made her
feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be
worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit
with pink eyes ran close by her.

Ln 6, Col 290 100% Windows (CRLF) UTF-8
```

Figure 15 Sentence Detection – Newline Character At End Of Line

The beginning of *Alice in Wonderland* by Lewis Carroll. Word wrap is turned on in Windows Notepad so the user can see all the text without having to scroll horizontally. The text is formatted to use a newline character to delimit the end of a paragraph.

Paragraph counts are displayed as an inline statistic when selecting a revision and need no visualization.

Part of Speech Usage

Detecting what part of speech any given word in a sentence is another topic that is beyond the scope of this project. Luckily, the spaCy library has this ability as one of its core features. The tokens produced by spaCy can be queried for their part of speech, and a count of each is created and then displayed as a bar chart to the user.

Sentence Detection and Count

Without the use of a library, sentence detection would involve a bit work. An example of why the task is more difficult than it may first seem involves considering how sentences are ended. The most common sentence ending is a period. The logical next step is to attempt to

split a work into sentences by using end punctuation like periods, exclamation marks, and question marks. However, doing this would also split the text apart whenever an abbreviated title like “Mrs.” or “Dr.” is used. Modern texts that mention websites would also fail to parse properly as the period in “amazon.com” would cause the same issue.

One also has to consider the possibility that a sentence of dialogue is interrupted by another person. This is usually indicated by the use of the em dash. Does the interrupted line count as one line or two?

Luckily, these issues can be ignored as the spaCy library is able to parse sentences. Getting the sentence count in a work is as simple as processing the document and looking at the value of a single property on the processed document. As this property is a list, getting the number of sentences is also simple.

Sentence counts are displayed as an inline statistic when selecting a revision and need no visualization. Sentence lengths are displayed as a bar chart to the user.

Word Count

Getting the word count for a revision is simple. A string containing the entire document was split on an empty character / whitespace. This results in an array of strings where each item in the array is a word. Empty strings were pruned from this list. A simple count on the final array length results in the word count. Word counts are displayed as an inline statistic when selecting a revision and need no visualization.

Word Count Change

Getting the number of words written in one revision compared to another is also simple. The word count stored in the older revision is subtracted from the word count of the more recent revision. Word count changes are displayed as a line graph to the user.

Note that negative word change counts are possible when the editing process makes the new revision shorter in length.

Word Frequency

Computation of Word Frequencies is another simple task. A list of words is generated from a revision in a similar way that the Word Count metric uses. Additional steps are taken to remove all punctuation and convert the entire document to lowercase. (This ensures the word “However” is considered to be the same as the word “however.”) A dictionary is created where the key will be a word and the value will be the number of times that word appears in the document. The program then enumerates through the list of words. If the word being looked at is not in the dictionary, it gets added to the dictionary with a value of 1. If the word is already in the dictionary, its value is increased by 1. The final word frequency is then displayed as a bar chart to the user.

Data Summary

Some visualizations have the option to display some summary statistics about the data used to construct the chart.

Emotion	Avg	Avg (Present)	Sum	Max
curiosity	0.0257	0.0257	0.4875	0.2822
desire	0.0423	0.0423	0.8042	0.7395
fear	0.0310	0.0310	0.5881	0.2936
sadness	0.1082	0.1082	2.0563	0.6190

Figure 16 Data Summary Popup
A summary of the emotion probabilities for curiosity, desire, fear, and sadness in The Raven by Edgar Allan Poe.

Data Export for Data Scientists

The ideal use case scenario for this program is that it would be the only analytic software a person would need to use. This is a lofty goal, and probably not achievable even with a large, dedicated, and knowledgeable team aiding in the development of the program. This is especially true given the limited nature of the program as of the time this document was written.

To account for the scenarios that this program is not capable of handling, but a user would like to explore the data even further, a data export option was added. Now, whenever the user would like to use the underlying data in any metric they use, all they have to do is click a single button to have raw data copied to their computer's clipboard.

The copied data adheres to a specification. The first line of the data will be the header data for the primary series of data copied. Each line after that will contain the data for a single item in the primary series. An example of a data series and the items it contains is the entity presence scores for all the named entities within a work, with each entity being a single item.

Some of the analytics use more than one data series. Combining the aforementioned entities with emotion classification is the primary reason the option for a secondary data series was created. If copied data has a secondary data series, a blank line will be inserted after the primary data series, followed by an output for the secondary data series that mirrors the format of the primary data series.

Every entry on each line is separated by a tab character. This enables pasting the data directly into a spreadsheet program. Figure 17 shows what the data looks like when pasted directly into Microsoft Excel. This functionality was utilized to produce some of the graphs contained in this document.

	A	B	C	D	E	F	G	H
1	Emotion	0	1	2	3	4	5	6
2	admiration	0.003628	0.001836	0.006121	0.028805	0.004278	0.003497	0.003756
3	amusement	0.002644	0.004766	0.003106	0.008478	0.000944	0.004617	0.002471
4	anger	0.004263	0.001197	0.00425	0.002342	0.001128	0.002583	0.001092
5	annoyance	0.00756	0.017583	0.019189	0.005573	0.007828	0.007488	0.004852
6	approval	0.010659	0.014315	0.00816	0.008599	0.07932	0.011857	0.124221
7	caring	0.000931	0.007913	0.013316	0.003417	0.007303	0.00156	0.00123
8	confusion	0.003106	0.021461	0.001708	0.00399	0.001454	0.006639	0.015688
9	curiosity	0.001869	0.028609	0.004115	0.01263	0.000519	0.007926	0.027912
10	desire	0.001113	0.003847	0.739546	0.005426	0.001902	0.005336	0.00271
11	disappointment	0.002988	0.062557	0.089434	0.039363	0.012659	0.005922	0.002115
12	disapproval	0.003329	0.003905	0.00862	0.003027	0.003283	0.003386	0.001371
13	disgust	0.002842	0.003298	0.009506	0.00233	0.002275	0.006589	0.001139
14	embarrassment	0.000897	0.0081	0.002313	0.004607	0.002171	0.002919	0.000702

Figure 17 Data Export Into Excel

The above data was put on the clipboard by clicking the Copy All Data button for an Emotional Probability visualization of *The Raven* by Edgar Allan Poe. Excel was then opened, and the data pasted directly into Excel without any manipulations of the data. (The screenshot was truncated for display purposes. It contains 28 data rows and 19 data columns in addition to the header column and header row.)

Another export feature was added in the ability to save the currently displayed visualization to a file. The majority of the figures in this document were created this way.

Analysis and Discussion

This section will assess the effectiveness of the methods described above. Limitations of the implementation will also be acknowledged.

Interactive Visualizations

The purpose for the inclusion of interactive visualizations is to allow the user to explore data in a more meaningful way than simply looking at a summary statistic, an exported table of raw data, or even a static chart.

The legend portion of the LiveCharts control has a bug where the background color of the legend cannot be changed. This leads to the situation where the legend is not all that visually distinct from the rest of the chart. Nothing needs to be done about this though, as this bug is undergoing testing related to a fix in the next version of the control.

Additional features that would make the visualizations even more useful would being able to select the colors used in visualizations where a color palette is required. Comparing data can sometimes be difficult when the color of a displayed series changes whenever the user adds or removes selected items.

An alternative to customizing the color predefined color palette would to allow the user to manually specify the color for a displayed series by clicking on the label associated with the series in the legend and then selecting a new color to use.

Both of the color selection improvements could be implemented, giving even finer control to the user.

As it stands now, the user is not able to see the underlying text for an item group. Providing a way for the user to see that data would greatly help give context to the visualization. This could be done by creating a tooltip for the x-axis label that contains the item's text.

The interactive chart control the user has access to, combined with a number of data filters, provides the user with a lot of freedom in exploring the data this program creates.

It could also be useful to have a simple way to view the base entity presence score or emotion probability score when the user utilizes some of the selection and combination features the charts provide.

Basic Statistical Metrics

The Word Count, Word Count Change, and Word Frequency metrics are simple enough to compute (i.e. they're just simple counts) that no analysis is needed. Furthermore, since the Sentence Detection, Sentence Count, Part of Speech Usage, Flesch Reading Ease, and Flesch-Kincaid Grade Level metrics are computed easily via using libraries to get the required variables, no additional analysis is needed on these metrics beyond what the authors of the relevant libraries have already done.

There are a few changes that could improve the usability of some of these statistics. The Flesch Reading Ease and Flesch-Kincaid Grade Level metrics in particular could use an explanation and show a scale of common numbers. This was attempted via a tooltip when the user hovers their mouse over a number, but the built in tooltips don't allow for displaying of newline characters.

The inclusion of these basic statistics increases the usefulness of the program and can add content to other, more advanced metrics.

Paragraph Count

The paragraph splitting and counting features proved to be accurate by comparing the generated data by actually counting the number of paragraphs for a number of the shorter works used as sample data.

The implementation is currently limited to two different types of delimiters for splitting paragraphs. It is entirely possible the other ways of denoting a new paragraph in a literary work that this algorithm does not account for.

User Cleaning of Named Entity Recognition Errors

Allowing the user to create new entity corrections allows the user to correct for the situation where the perspective character is almost never named. In the case of the Divine Comedy by Dante Alighieri, the narrator is the main character, also named Dante, but is only named once in the entire translation.

This approach is not without errors of its own. It is possible that other characters may use the word “we”, thereby reducing the accuracy of the results. Another possible flaw in this approach is that if the perspective shifts characters but stays in the first person point of view, a similar situation will occur where the word “we” refers to multiple entities.

A possible solution to this issue is to further refine the entity correction process so that the user can specify item ranges where the entity correction applies.

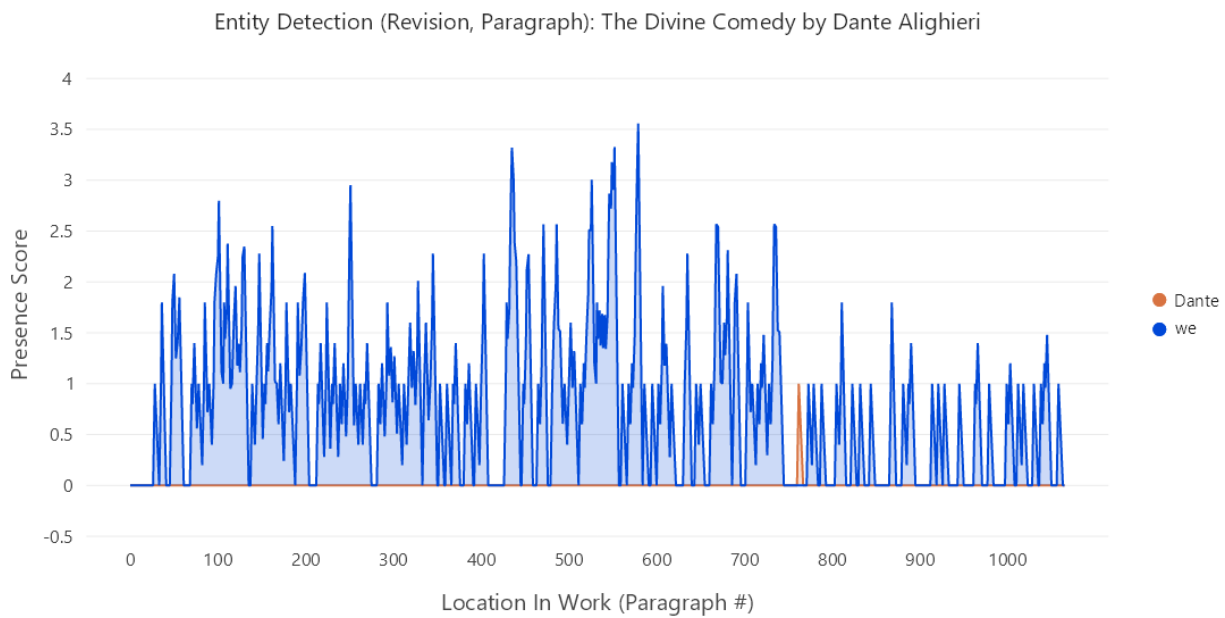


Figure 18 Entity Presence Utilizing Entity Corrections (New Entity)

Manual addition of the label "we" vs mentions of the main character "Dante" to visualize the presence of the main character in *The Divine Comedy* by Dante Alighieri

The ability to have one entity be an alias for another entity was also successful. Figure 19 shows the entity presence of the three main characters. Montessor is only mentioned once, even though they are the narrator and provide the perspective the story is told from. Figure 20 then shows a successful aliasing of the word "I" to the character of Montessor.

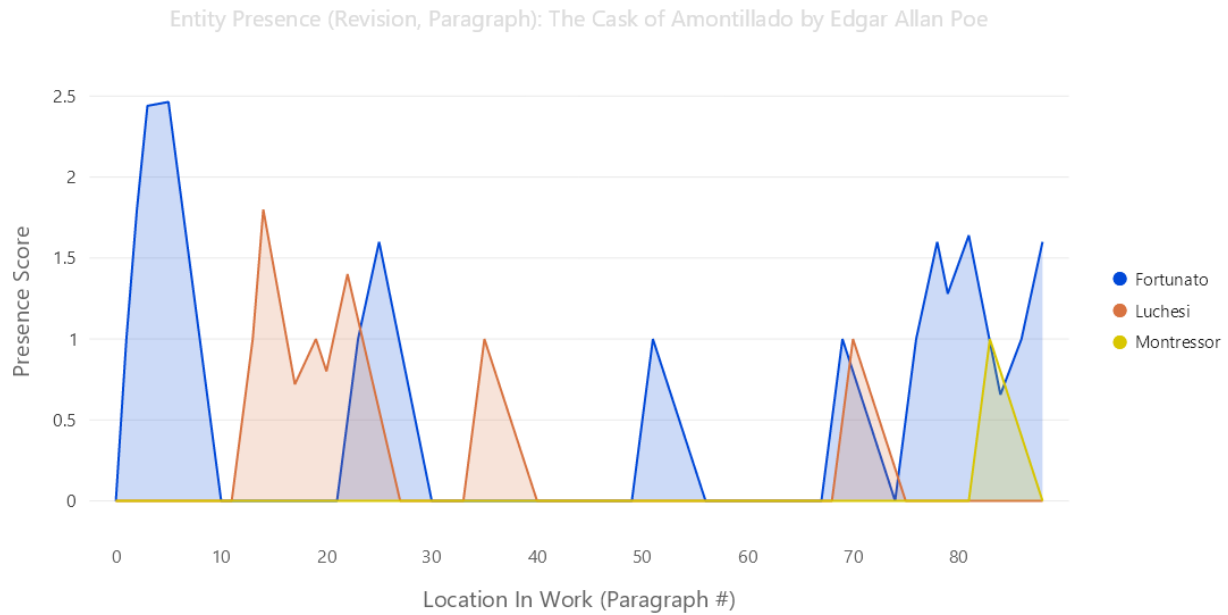


Figure 19 Entity Presence Utilizing Entity Corrections (No Aliasing)

Entity presence for the main characters in *The Cask of Amontillado* by Edgar Allan Poe. Montessor is only mentioned once by name.

Entity Presence (Revision, Paragraph): The Cask of Amontillado by Edgar Allan Poe

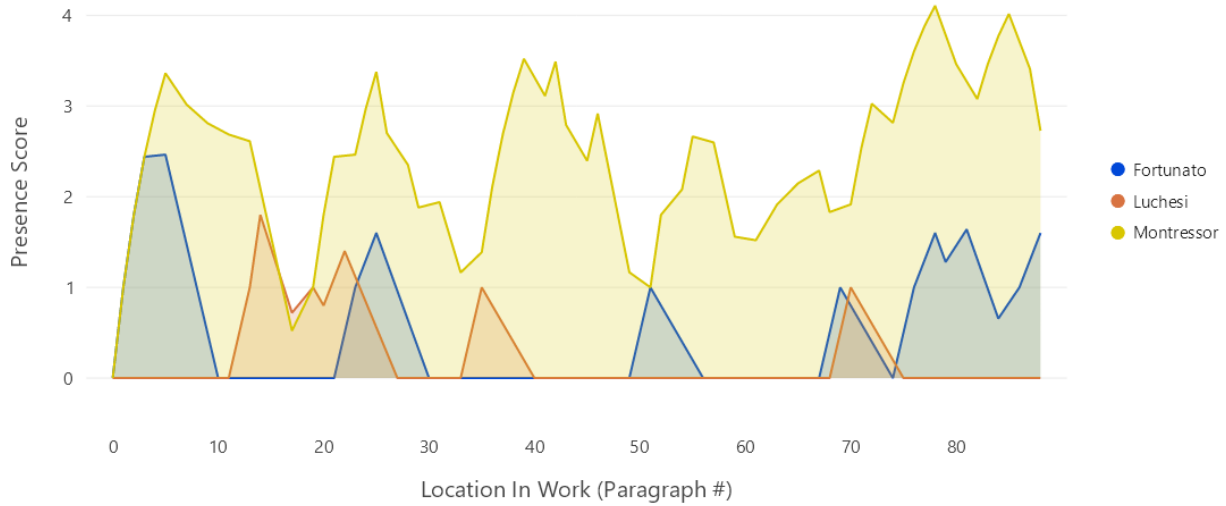


Figure 20 Entity Presence Utilizing Entity Corrections (Successful Alias)

Entity presence showing a successful use of an alias “I” for the character of Montresor in *The Cask of Amontillado* by Edgar Allan Poe.

Figure 21 also shows a problem with named entity corrections. Multiple characters can be combined into a single entity when they shouldn't be combined. The entities displayed are Queen, Rumpelstiltskin, and Snowdrop. However, the Queen in this case is two different characters from two different short stories contained in the larger work.

The solution to this problem is to extend the named entity correction so that the correction only applies to a range of items in the text. As the base named entity corrections operate on the Work level, not the Revision level, another table will have to be added to the database to extend this feature.

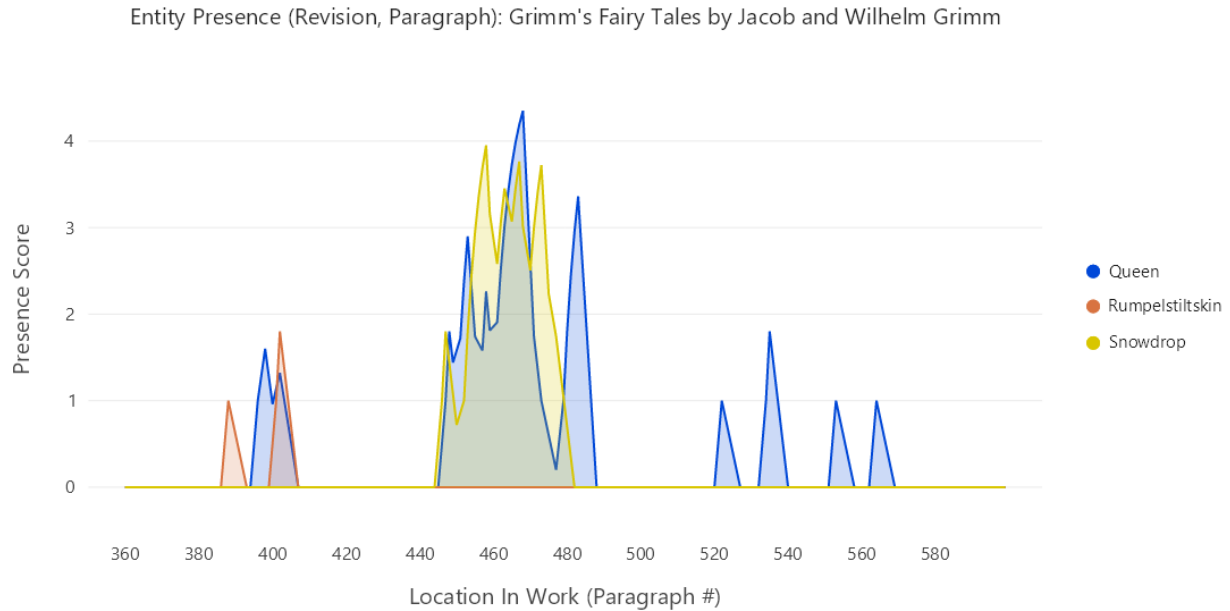


Figure 21 Entity Presence Utilizing Entity Corrections (Incorrect Entity Combination)
Entity presence showing a an unsuccessful combination of two different Queen characters in Grimm's Fairy Tales by Jacob and Wilhelm Grimm.

The named entity correction feature of spaCy is not perfect. It can often miss entities entirely. When examining Winnie the Pooh by Alan Alexander Milne, the character of Piglet was detected, but only as "Little Piglet" and not simply "Piglet." Figure 22 shows the EntityCorrections table for the database utilized by the program. The corrections have been filtered by work and by name. The entities detected by spaCy with the word "pig" include the non-entity phrase "Fond of Pigs", the nickname "Little Piglet", and the name of another character "Henry Pootel Piglet."

The named entity recognition feature of spaCy does have multiple labels. For simplicity, this program only looks at named entities with the "PERSON" label. It is possible that one of the other labels, such as "ORG" or "LOC" may have picked the correct entity. However, it is also possible that none of the other labels correctly identified the character. As this is an issue with the library being used, the best solution is to allow the user to manually add a named entity correction to identify missed entities.

EntityCorrectionID	WorkID	Name	Ignore	Alias_EntityCorrectionID
Filter	10 <input type="checkbox"/>	pig <input type="checkbox"/>	Filter	Filter
789	10	Fond of Pigs	1	NULL
805	10	Little Piglet	1	NULL
817	10	Henry Pootel Piglet	1	NULL

Figure 22 Named Entity Corrections - Database Query of "pig"

The database filtered by WorkID (10 == Winnie The Pooh by Alan Alexander Milne) and Name (the phrase "pig"). The character of Piglet was not directly detected by the spaCy named entity recognition feature.

Another issue with named entity recognition is the amount of entities that are created that are not actually named entities. Figure 23 shows many non-entities being detected as entities. Databinding in XAML can be slow, and large amounts of entities can slow the program down.

Name	Alias For	Ignore	Remove
thews	-	Ignore	Remove
cautel	-	Ignore	Remove
rede	-	Ignore	Remove
Bear't	-	Ignore	Remove
Pooh	-	Ignore	Remove
springes	-	Ignore	Remove
bawds	-	Ignore	Remove
the Nemean lion's	-	Ignore	Remove
harrow	-	Ignore	Remove
Rankly	-	Ignore	Remove

Figure 23 Named Entity Correction - non-entity detection

The named entity correction interface of the program, shown for Hamlet by William Shakespeare.

There is a bug with one of the MAUI function calls that displays a popup window. This bug only occurs intermittently. As this is the function that displays a list of aliases, this can break the ability to assign aliases for corrections. An alternative could be programmed in, but many of the alternatives that were attempted dramatically lagged the program. This bug will hopefully be fixed by Microsoft in the future. If this bug happens now, the user will be forced to restart the program, possibly multiple times.

Overall, this is an effective addition to the capabilities of this program. This functionality provides the necessary data so that characters can be identified.

Entity Presence Scores

Entity presence scores perform well but are not without issues. Assuming the user has made the necessary adjustments using the named entity correction tool, the entity presence algorithm reliably matches the positions and scores them accordingly.

The Raven by Edgar Allan Poe will be analyzed to illustrate these points as The Raven is a short work and the results are easily verifiable by comparing the graphs below with a cursory reading of the poem.

The first issue is that the entity presence algorithm looks for any instances of a word that matches a named entity. In the case where the name of entity is in the title of a work but the entity is not immediately present, this results in a false positive detection of the entity. Figure 24 shows this problem.

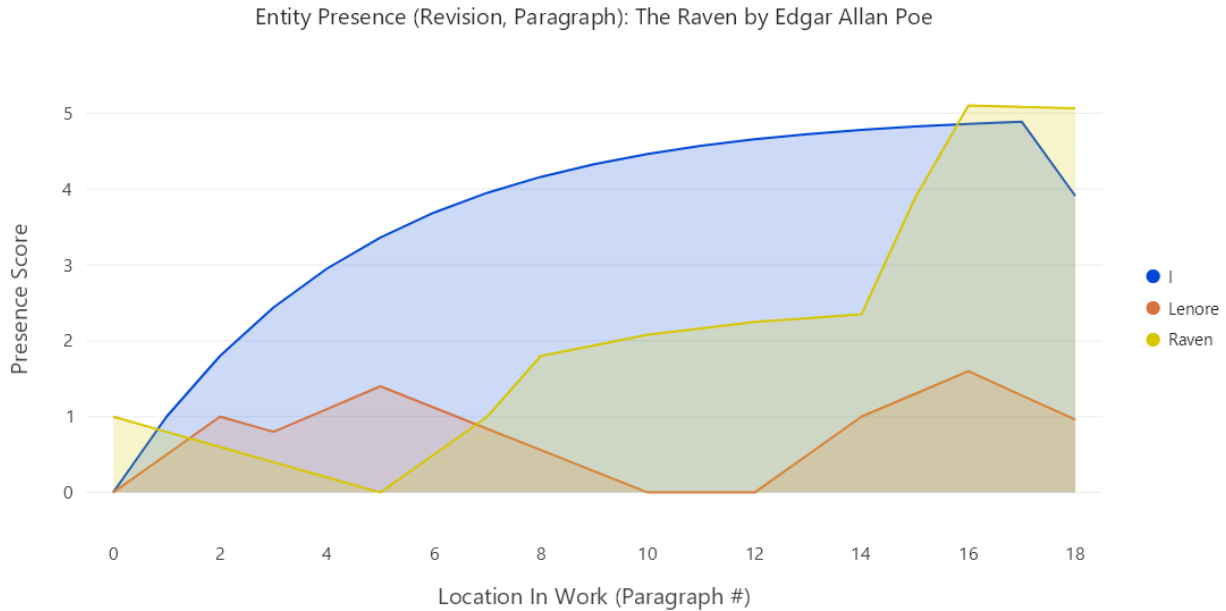


Figure 24 Entity Presence - Title of Poem Alters Results

Subject: The Raven by Edgar Allan Poe. The presence of the word Raven in the title of the poem (Paragraph 0) alters the graph so that it appears the entity of the Raven is present before its first actual appearance in the 6th stanza.

That issue aside, the rest of the visualization aligns with what is written in the poem. The narrator is detected immediately in the first stanza. As the poem is written in first person perspective, the narrator continues to refer to himself as “I” and his presence score continually climbs and approaches a stable score. As the narrator spaces out references to themselves fairly evenly, this stable score makes sense.

The narrator spends the first few stanzas reminiscing about the character Lenore. As Lenore is mentioned in stanzas two and five, her presence score spikes at those times, along with the algorithm’s predictions as to whether or not she is present in the periods immediately surrounding those mentions.

Stanza six sees the arrival of the titular raven. The narrator’s focus switches from thinking about Lenore to interacting with the raven. As expected, the presence score for the raven increases with the presence score for Lenore decreases.

The narrator once again mentions Lenore in stanza fourteen, which is responsible for making her presence score rise above zero. At the same time, the raven itself is mentioned repeatedly alongside dialogue tags in the poem’s most famous line, “Quoth the Raven, ‘Nevermore.’”

The final large change in the presence scores is a drop-off in the narrator’s presence in the last stanza. As the narrator does not refer to himself as “I” in that stanza, this drop-off is expected.

Groupings of entities are simply the average of the presence scores for every entity in the group. Figure 25 clearly shows the two entities and their average as a group.

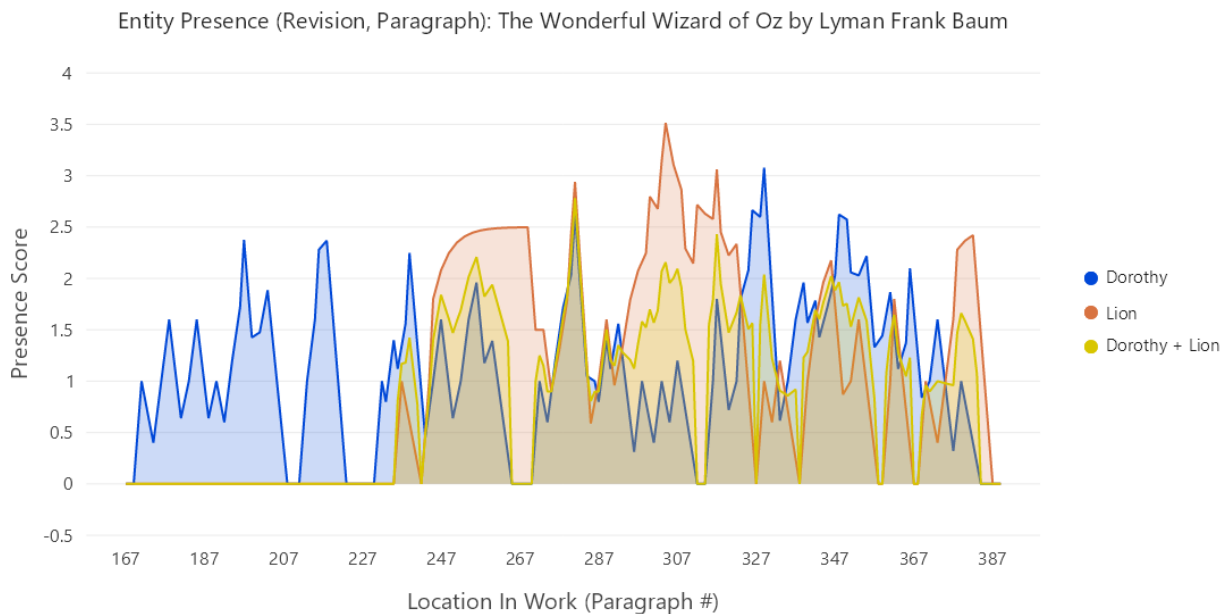


Figure 25 Entity Presence - Entity Grouping

Combining entity presence scores for multiple named entities to create a unique grouping. Subject: The Wizard of Oz by Lyman Frank Baum.

Another issue with entity presence scores is that the base data is not very readable without the use of a smoothing function. Figure 26 shows some of the entity presence scores for A Christmas Carol by Charles Dickens. Without a smoothing functions, all entities are scored as a 0 or 1.

This also illustrates two problems already discussed. One problem is a group of entities related by aliasing are being summed instead of averaged. (Scrooge is an alias of Ebenezer Scrooge. Ebenezer Scrooge thus gets counted twice.) The other problem is one entity picking up the presence scores of multiple characters. (Ghost, and it's alias Spirit, represent The Ghost of Christmas Past, The Ghost of Christmas Present, and the Ghost of Christmas Yet To Come.)

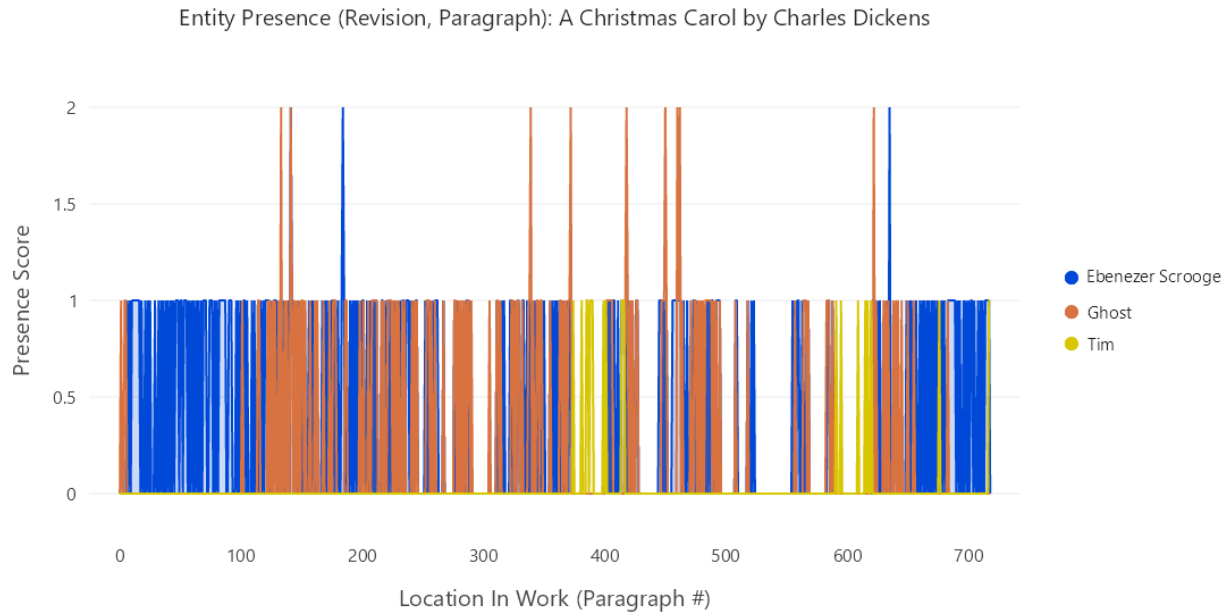


Figure 26 Entity Presence - No Smoothing Function

Subject: A Christmas Carol by Charles Dickens.

Even zooming in for more detail, as shown in Figure 27, doesn't provide much insight.

Entity Presence (Revision, Paragraph): A Christmas Carol by Charles Dickens

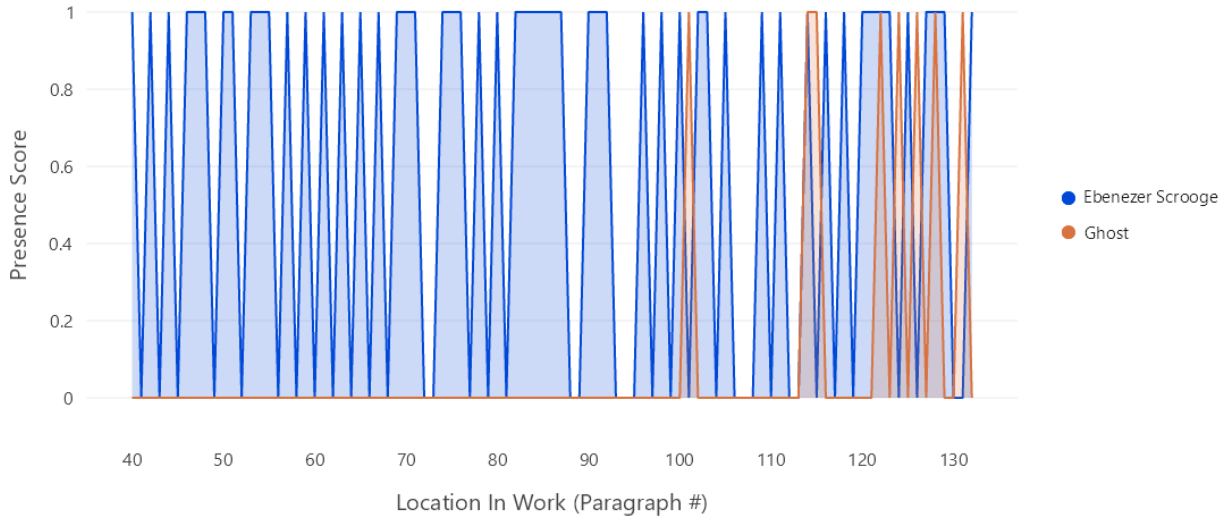


Figure 27 Entity Presence - No Smoothing Function (Zoomed)

Subject: A Christmas Carol by Charles Dickens. View from Figure 26 has been zoomed in for more detail.

The solution to this problem was the application of a smoothing function. Not only does a smoothing function approximate an entity fading from narrative after they stop being mentioned, it also makes it easier for the user to interpret.

Entity Presence (Revision, Paragraph): A Christmas Carol by Charles Dickens

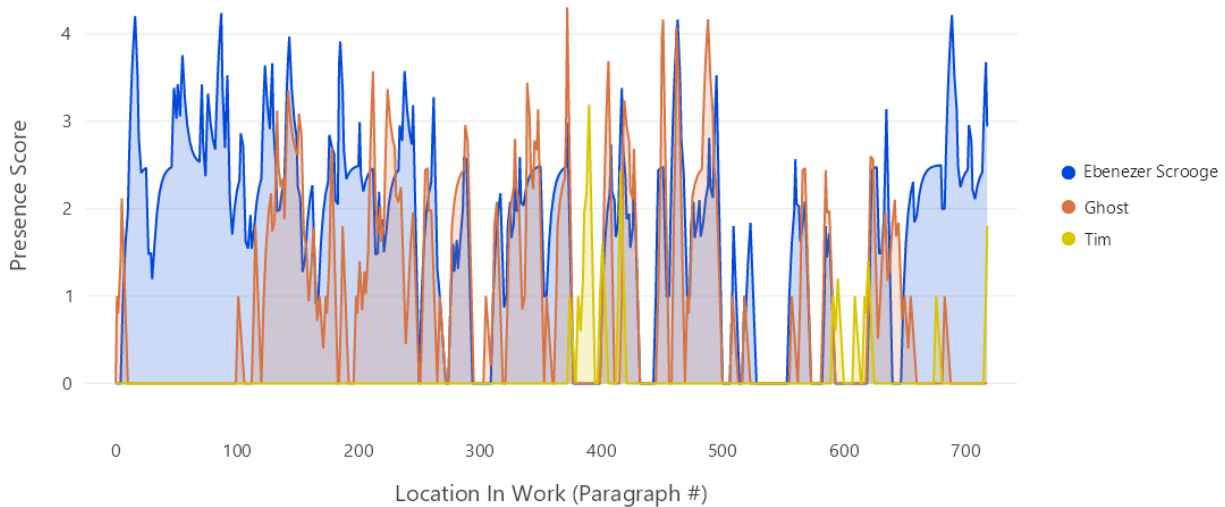
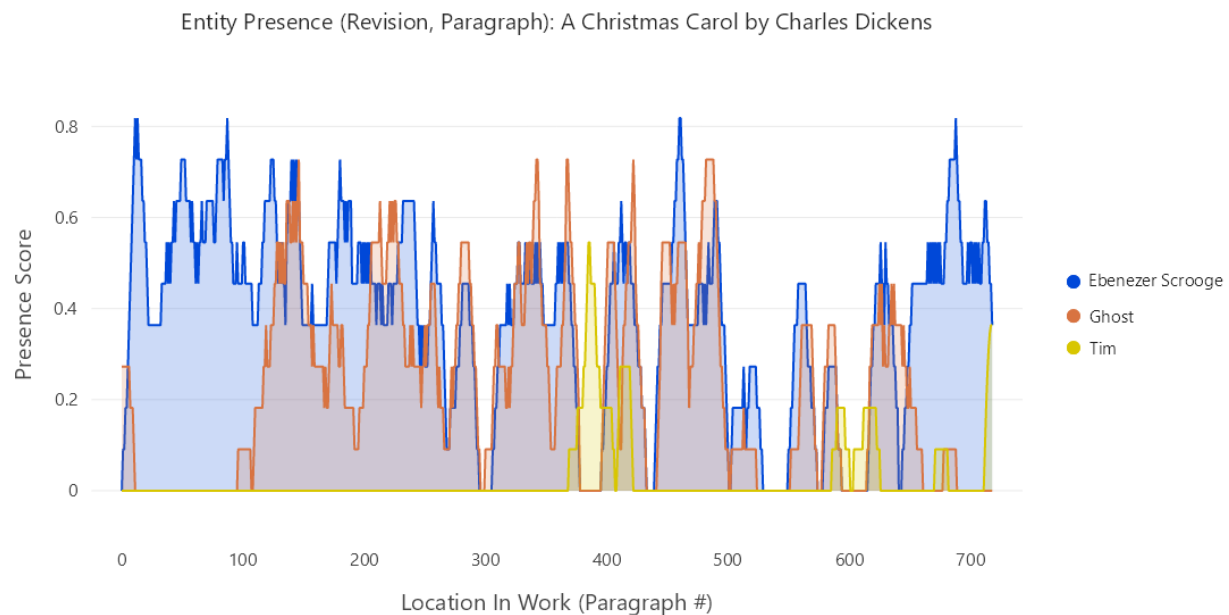


Figure 28 Entity Presence - Lookaround Smoothing Function

Subject: A Christmas Carol by Charles Dickens.

Once the centered rolling average smoothing function had been developed for emotion classification probability scores, the option was given to the user to use that smoothing function instead of the lookaround smoothing function. The rolling average is a decently well known algorithm, and is easy to interpret just from viewing the results. How the lookaround function smoothing function is more complicated. The rolling average also makes the differences from item to item more apparent as the curve generated by the data points is less smooth.



*Figure 29 Entity Presence - Centered Rolling Average Smoothing Function
Subject: A Christmas Carol by Charles Dickens.*

Improvements to the algorithm can be made. One easy to implement option would be exposing the value of a positive entity detection to the user. Perhaps the user might get better results if they were able to set the score to +0.5 for a match instead of +1. If this were done, the entity tonality function might also have to take this factor into account.

Another feature that would be helpful would be live corrections of the data. If a user were able to select item position 0 as displayed in Figure 24 and change the entity presence score of "Raven" from 1 to 0, it would provide a much more accurate result.

Additionally, if there was a way to view each item, it would help provide extra context to the displayed scores without having to manually check the source document.

This feature meets the requirements of the program. Entities can now be detected and their presence in a story plotted in meaningful ways.

Emotion Classification Models

Before the selected emotion classification models can be used, they have to be evaluated for accuracy, ease of use, and execution speed. EmoRoBERTa was the first model chosen. The roberta-base-go_emotions model was found as an alternative to EmoRoBERTa. Ultimately, the decision was made to utilize the roberta-base-go_emotions model over EmoRoBERTa.

The creator of the roberta-base-go_emotions model, Sam Lowe, also did some statistical analysis of the model. Not only were those results made available, but the Jupyter notebook that performed the analysis was also made available. EmoRoBERTa was provided without the same analysis. However, since both models are created in the same way, used in the same way, and output data in the same format, it was possible to alter the parameter in the provided Jupyter notebook to perform the exact same analysis on EmoRoBERTa.

Table 3 shows the overall results of both models when taking the mean values of all 28 possible labels produced by the model. Precision, recall, and f1 scores were reported. Sam Lowe had this to say about the accuracy statistic of the analysis:

Note, should probably ignore accuracy metric again, but in this case at the per-label level a multi-label dataset has a huge number of true negatives which make the accuracy figure pretty meaningless. E.g. in a situation where there are 10 positive items and 990 negative items, if a model simply predicts negative for everything, its accuracy figure still appears very high (0.99) even though its clearly not performing to a useful level. (Go_emotions-Dataset/Eval-Roberta-Base-Go_emotions.ipynb at Main · Samlowe/Go_emotions-Dataset, n.d.)

Due to that, accuracy scores have been omitted from overall the results. The results show that Roberta-base-go_emotions outperforms EmoRoBERTa in all three statistics.

	EmoRoBERTa	roberta-base-go_emotions
<i>precision</i>	0.482	0.542
<i>recall</i>	0.5	0.577
<i>f1</i>	0.482	0.541

*Table 3 simple mean of labels
Evaluation metrics for the EmoRoBERTa and roberta-base-go_emotions models.*

Table 4 shows the same analysis can be performed, but instead of using a simple mean to compute the values, a weighted mean (based on support level) was used. The results again show that roberta-base-go_emotions outperforms EmoRoBERTa in all three statistics.

	EmoRoBERTa	roberta-base-go_emotions
<i>precision</i>	0.517	0.572
<i>recall</i>	0.585	0.677
<i>f1</i>	0.544	0.611

*Table 4 weighted average of labels (using support)
Evaluation metrics for the EmoRoBERTa and roberta-base-go_emotions models.*

A more detailed comparison of the models can be done by examining statistics for each individual label.

Table 5 (roberta-base-go_emotions) and

Table 6 (EmoRoBERTa) show the results of this operation.

emotion	accuracy	precision	recall	f1	mcc	support	threshold
<i>admiration</i>	0.94	0.651	0.776	0.708	0.678	504	0.25
<i>amusement</i>	0.982	0.781	0.89	0.832	0.825	264	0.45
<i>anger</i>	0.959	0.454	0.601	0.517	0.502	198	0.15
<i>annoyance</i>	0.864	0.243	0.619	0.349	0.328	320	0.1
<i>approval</i>	0.926	0.432	0.442	0.437	0.397	351	0.3
<i>caring</i>	0.972	0.426	0.385	0.405	0.391	135	0.4
<i>confusion</i>	0.974	0.548	0.412	0.47	0.462	153	0.55
<i>curiosity</i>	0.943	0.473	0.711	0.568	0.552	284	0.25
<i>desire</i>	0.985	0.518	0.53	0.524	0.516	83	0.25
<i>disappointment</i>	0.974	0.562	0.298	0.39	0.398	151	0.4
<i>disapproval</i>	0.941	0.414	0.468	0.439	0.409	267	0.3
<i>disgust</i>	0.978	0.523	0.463	0.491	0.481	123	0.2
<i>embarrassment</i>	0.994	0.567	0.459	0.507	0.507	37	0.1
<i>excitement</i>	0.981	0.5	0.417	0.455	0.447	103	0.35
<i>fear</i>	0.991	0.712	0.667	0.689	0.685	78	0.4
<i>gratitude</i>	0.99	0.957	0.889	0.922	0.917	352	0.45
<i>grief</i>	0.999	0.333	0.333	0.333	0.333	6	0.05
<i>joy</i>	0.978	0.623	0.646	0.634	0.623	161	0.4
<i>love</i>	0.982	0.74	0.899	0.812	0.807	238	0.25
<i>nervousness</i>	0.996	0.571	0.348	0.432	0.444	23	0.25
<i>optimism</i>	0.971	0.58	0.565	0.572	0.557	186	0.2
<i>pride</i>	0.998	0.875	0.438	0.583	0.618	16	0.1
<i>realization</i>	0.961	0.27	0.262	0.266	0.246	145	0.15
<i>relief</i>	0.992	0.152	0.636	0.246	0.309	11	0.05
<i>remorse</i>	0.991	0.541	0.946	0.688	0.712	56	0.1
<i>sadness</i>	0.977	0.599	0.583	0.591	0.579	156	0.4
<i>surprise</i>	0.977	0.543	0.674	0.601	0.593	141	0.15
<i>neutral</i>	0.758	0.598	0.81	0.688	0.513	1787	0.25

Table 5 Per-label metrics (maximizing f1) for the roberta-base-go_emotions model
Evaluation metrics for the EmoRoBERTa and roberta-base-go_emotions models.

emotion	accuracy	precision	recall	f1	mcc	support	threshold
<i>admiration</i>	0.936	0.657	0.645	0.651	0.615	504	0.15
<i>amusement</i>	0.975	0.71	0.833	0.767	0.756	264	0.35
<i>anger</i>	0.954	0.41	0.561	0.473	0.456	198	0.15
<i>annoyance</i>	0.914	0.301	0.353	0.325	0.28	320	0.1
<i>approval</i>	0.921	0.377	0.33	0.352	0.311	351	0.45
<i>caring</i>	0.974	0.48	0.452	0.466	0.453	135	0.5
<i>confusion</i>	0.968	0.432	0.412	0.421	0.405	153	0.45
<i>curiosity</i>	0.932	0.411	0.694	0.516	0.501	284	0.05
<i>desire</i>	0.982	0.429	0.434	0.431	0.422	83	0.05
<i>disappointment</i>	0.944	0.2	0.338	0.251	0.232	151	0.05
<i>disapproval</i>	0.926	0.319	0.453	0.375	0.342	267	0.1
<i>disgust</i>	0.977	0.491	0.439	0.464	0.453	123	0.65
<i>embarrassment</i>	0.993	0.524	0.297	0.379	0.392	37	0.3
<i>excitement</i>	0.974	0.363	0.476	0.412	0.403	103	0.35
<i>fear</i>	0.987	0.529	0.59	0.558	0.552	78	0.65
<i>gratitude</i>	0.977	0.788	0.878	0.831	0.82	352	0.05
<i>grief</i>	1	1	0.667	0.8	0.816	6	0.3
<i>joy</i>	0.975	0.603	0.491	0.541	0.532	161	0.65
<i>love</i>	0.971	0.634	0.815	0.713	0.704	238	0.05
<i>nervousness</i>	0.995	0.4	0.261	0.316	0.321	23	0.4
<i>optimism</i>	0.964	0.467	0.425	0.445	0.427	186	0.1
<i>pride</i>	0.996	0.333	0.25	0.286	0.287	16	0.2
<i>realization</i>	0.967	0.27	0.138	0.183	0.178	145	0.9
<i>relief</i>	0.996	0.263	0.455	0.333	0.344	11	0.1
<i>remorse</i>	0.991	0.543	0.679	0.603	0.602	56	0.05
<i>sadness</i>	0.974	0.552	0.442	0.491	0.481	156	0.35
<i>surprise</i>	0.971	0.446	0.496	0.47	0.456	141	0.25
<i>neutral</i>	0.732	0.577	0.7	0.632	0.43	1787	0.05

Table 6 Per-label metrics (maximizing f1) for the EmoRoBERTa model
Evaluation metrics for the EmoRoBERTa and roberta-base-go_emotions models.

Both models were then used to analyze *The Scarlet Letter* by Nathaniel Hawthorne. This was done in order to evaluate the outcomes when the specific type of data being used (literary works) were fed into the model.

The model was applied on specific grouping types, either paragraphs or sentences, for the entirety of the work. Then the results were used to create boxplots to examine the model's output.

Figure 30 and Figure 31 shows just the neutral label. The neutral label was separated from the other twenty-seven labels since the values present on the neutral label are vastly different from the other labels. Figure 30 uses paragraph groupings and Figure 31 uses sentence groupings.

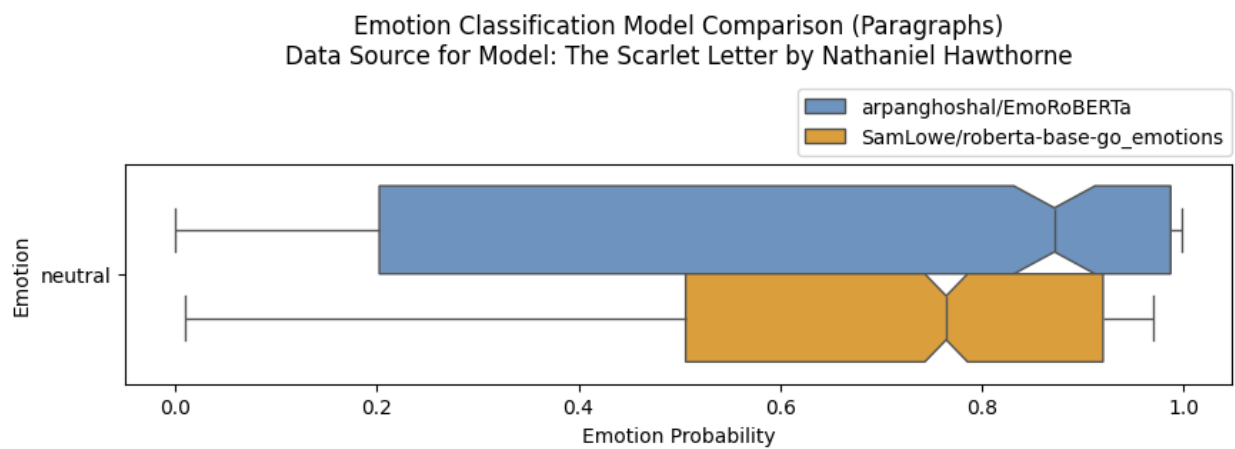


Figure 30 Emotion Classification Comparison (Paragraphs, Neutral, Different Models)
 Displaying the difference in detecting the "neutral" classification between the EmoRoBERTa and roberta-base-go_emotions models (outliers were not detected.)

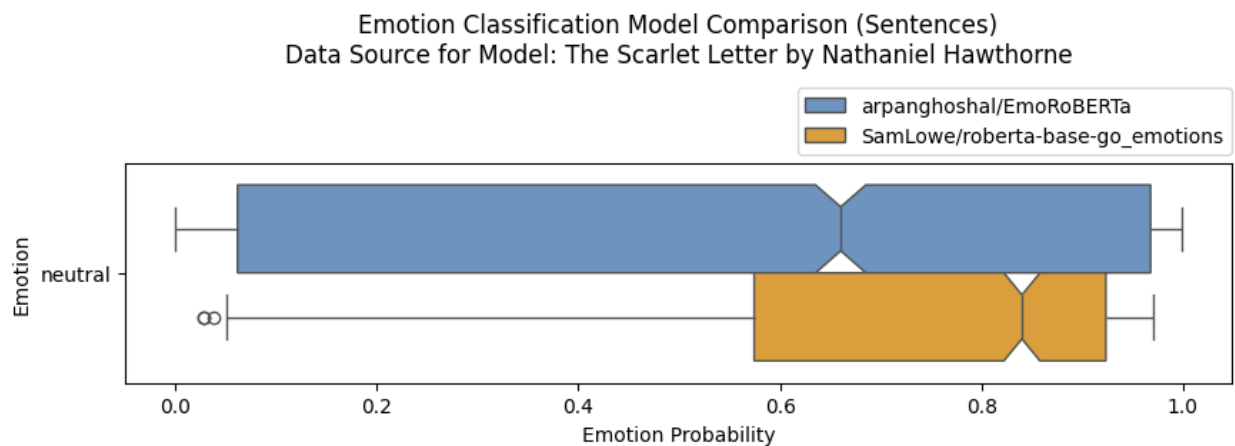


Figure 31 Emotion Classification Comparison (Sentences, Neutral, Different Models)
 Displaying the difference in detecting the "neutral" classification between the EmoRoBERTa and roberta-base-go_emotions models.

Figure 32 and Figure 33 then show the other twenty-seven labels for the paragraph grouping. Figure 32 has the outliers removed from the graph so that the viewer can make a better examination of the box and whiskers of the plot. Figure 33 is the same chart but the outliers are included. The combination of the two charts allows for a fuller examination of the data.

Figure 34 and Figure 35 are composed identically to Figure 32 and Figure 33 but use a sentence grouping instead of the paragraph grouping.

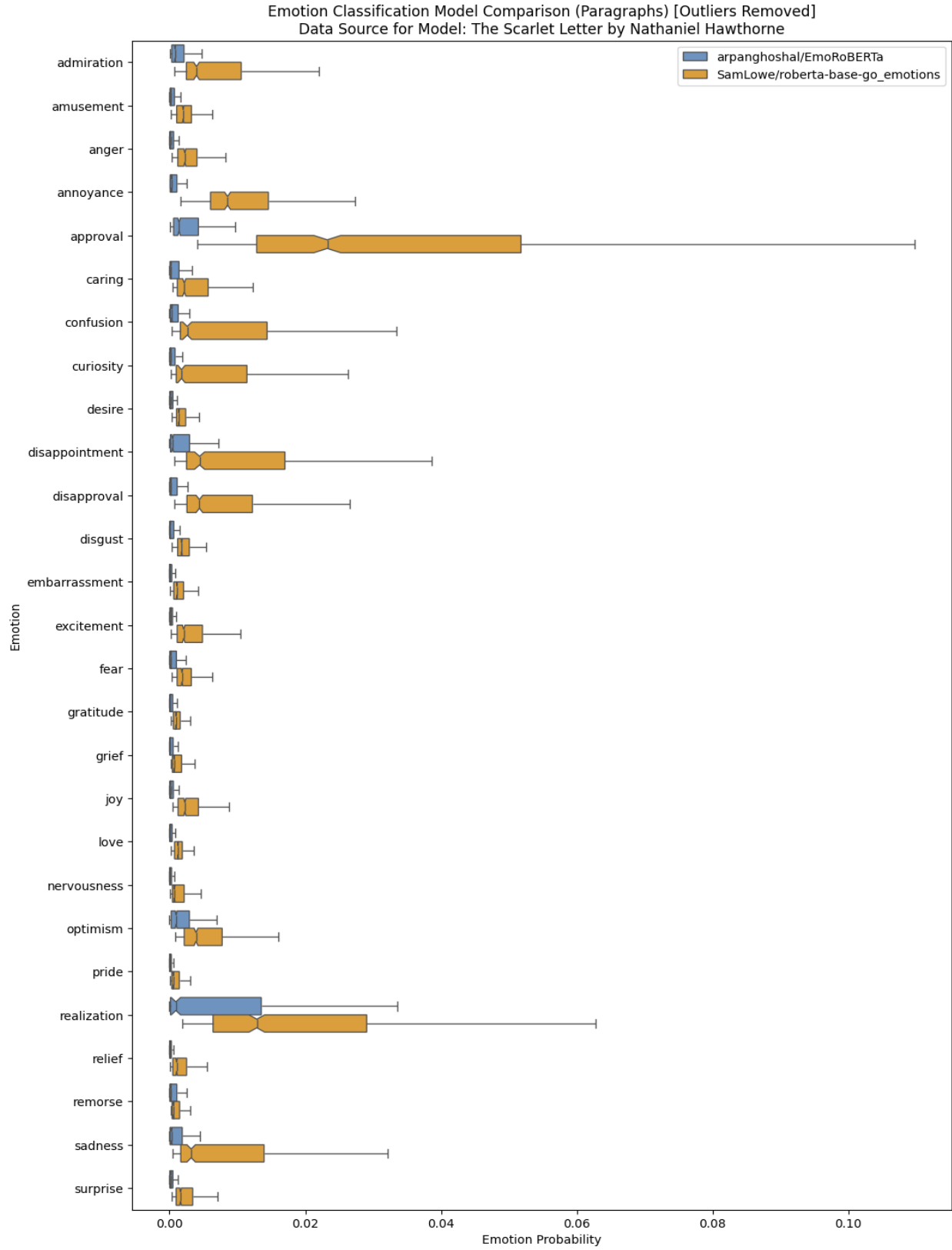


Figure 32 Emotion Classification Comparison (Different Models, w/o Outliers)
Comparison of EmoRoBERTa and roberta-base-go_emotions models. Outliers removed for clarity on box plots.

Emotion Classification Model Comparison (Paragraphs)
 Data Source for Model: The Scarlet Letter by Nathaniel Hawthorne

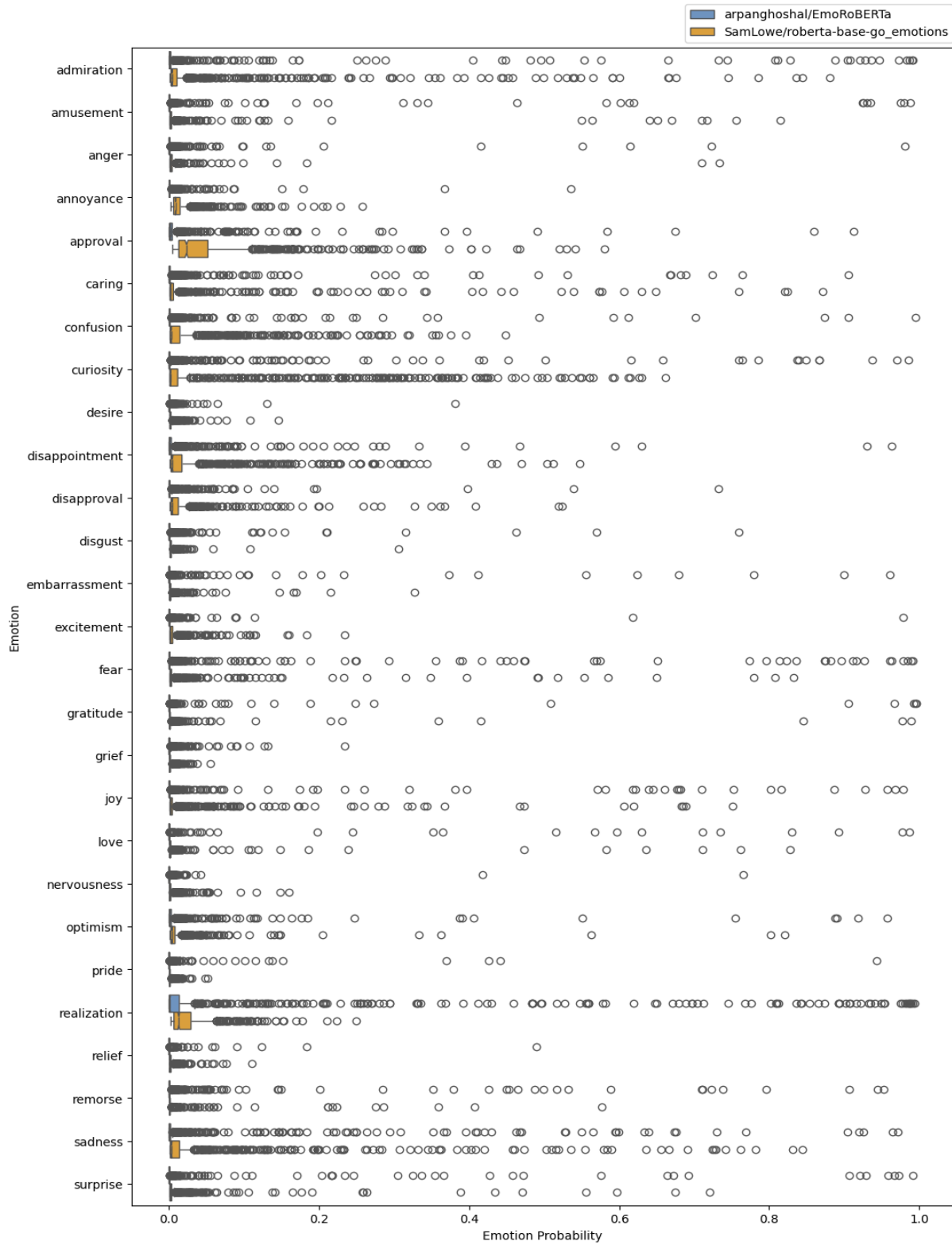


Figure 33 Emotion Classification Comparison (Different Models, w/ Outliers)
 Comparison of EmoRoBERTa and roberta-base-go_emotions models.

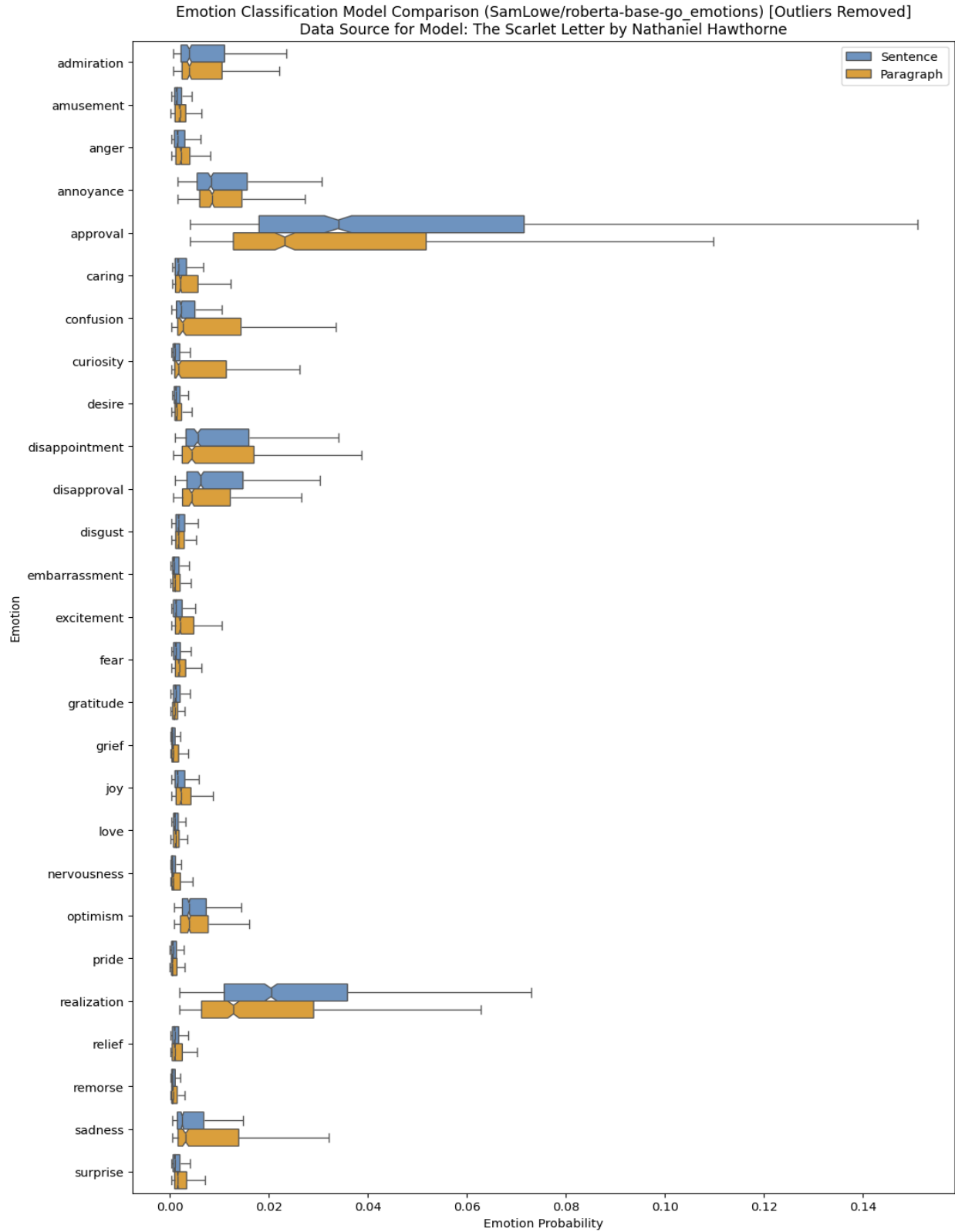


Figure 34 Emotion Classification Comparison (Different Item Grouping, w/o Outliers)
Comparison of EmoRoBERTa and roberta-base-go_emotions models. Outliers removed for clarity on box plots.

Emotion Classification Model Comparison (SamLowe/roberta-base-go_emotions)
 Data Source for Model: The Scarlet Letter by Nathaniel Hawthorne

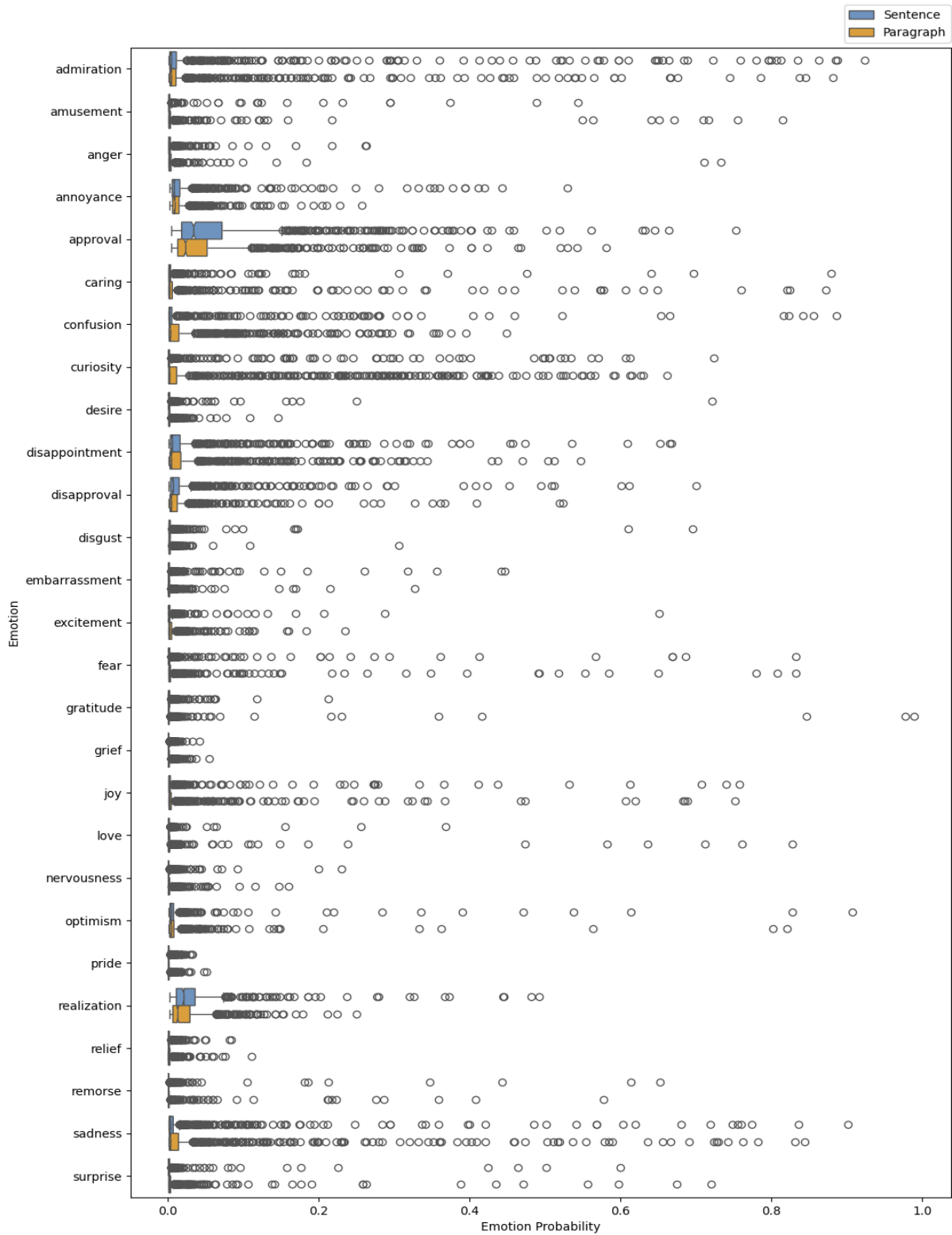


Figure 35 Emotion Classification Comparison (Different Item Grouping, w/ Outliers)
 Comparison of EmoRoBERTa and roberta-base-go_emotions models.

These results show that roberta-base-go_emotions outperforms EmoRoBERTa.

Interestingly, these graphs show the results can vary depending on the grouping type. When taken individually, two sentences being used to convey two ideas may present two different meanings or emotions. When those two sentences are combined into a single paragraph, a completely new third idea can be conveyed. The overall meaning and emotion can change to reflect the third idea.

The results being different for both paragraphs and sentences is a feature of the program. Users are allowed to perform analysis on both types of groupings.

Ease of use was both simple and hard. Once all the requirements were put in place, using the model in Python required a single import statement, a single line to instantiate the object that represents the model, and a single call to a method of the model's object. The two models are so similar that the only thing that needed to be done to switch between the two models is changing two parameters on the instantiating call. All things considered, using the model once the requirements were in place was easy.

The initial setup of the environment to run the models was both simple and complicated. Adding a library to the environment was simple. However, after the execution speed of the first implemented model was tested and found to be very slow, adjustments to the environment were attempted.

Both models are fully capable of being run on the CPU alone. Both models use neural networks and are capable of being run on the GPU as well. GPU execution of the models can greatly speed up execution time. This type of execution requires the installation of specific versions of some Python libraries, as well as drivers for the end user's GPU.

As a large portion of the program was already built, the models had to be integrated with a specific setup. The program requires a specific version of Python. Python 3.11.5 was selected as it was the latest stable version available when this project was started. It was not known at the time that the GPU version of TensorFlow, one of the aforementioned libraries, is no longer

supporting on the Windows operating system. To work around this, an older version of TensorFlow can be installed. However, the older versions of TensorFlow are not built against the newest versions of Python. A downgrade to Python 3.9.x was needed. This obviously caused a lot of issues.

Before major changes to the program were made, a separate test environment was created just for the models in question. Different combinations of Python versions, models, and CPU and GPU bound computations were tested to find an acceptable combination. The test data was an unpublished book penned by the author. The test data had approximately 53,000 words, 3,700 sentences, and 1,100 paragraphs. Unless otherwise noted, execution times are approximate averages based off multiple runs of each setup.

The first test was used as a baseline and involved the first model selected, EmoRoBERTa, in using Python 3.11.5, running only the CPU. Execution time was 14 minutes for sentences.

To test what effect downgrading Python would have, the same setup was used except Python was downgraded to 3.9.18. Execution time doubled to 28 minutes.

To test what effect enabling GPU acceleration would have, the same setup as the last test was used except now GPU enabled versions of TensorFlow were substituted in for the CPU-only versions. Execution time decreased to 11.9 minutes.

As the last test only provided a 15% savings in execution time, the roberta-base-go_emotions model was chosen as the second model for evaluation. The first test for this model still utilized Python 3.9.18 but changed back to only using the CPU. Execution time decreased to just 2.5 minutes.

The next step was to test the effect running on a GPU would have on the roberta-base-go_emotions model. Execution time decreased to 50 seconds.

Now possessing a fast model, attempts were made to optimize the GPU-only parameters. The addition of a single parameter to modify the batch size of processed items was able to further reduce the total execution time to just 12 seconds.

This was a massive improvement over the first few execution times and warranted changing the environment for the program. However, this was going to be a lot of work as it required modifying getting a lot of open-source software configured. As this speed savings was discovered late in the process of creating the program, one last test was run to find an easier solution.

The roberta-base-go_emotions model was then run in Python 3.11.5 using just the CPU. The execution time was only 55 seconds. While not as fast as running the same model on the GPU in a compatible version of Python, this was an acceptable increase in speed for almost no increase in complexity. Thus, the final decision was to use the roberta-base-go_emotions model as the environment was already set up, with future plans to modify the environment to enable GPU processing for users who have that capability.

The models examined are very good at what they do given the complex nature of their task, but their evaluation metrics (e.g. precision, recall) could be better.

The data set used to train the models consists of reddit.com comments. The way people communicate on reddit is vastly different than the majority of literary works. As such, the performance of the model on literary works could suffer.

Relatedly, the source timeframe of the reddit comments is within the last two decades. Languages are not static. Languages change and adapt to the needs of its speakers and writers over time. Using this model to operate on older texts may reduce the overall accuracy. This is particularly important as all of the sample texts utilized here are works from the public domain and were written at least ninety-five years ago. Using modern literary works, as well as time period appropriate literary works might help improve the results the models generate.

Some of the emotion labels were underrepresented in the reddit comments, leading to poorer performance on specific labels (e.g. grief.) Training the models on additional data that shows the underrepresented labels should decrease the error rate on these labels.

The models also have an upper limit on the number of tokens they can process. The solution for this project was to ignore any tokens past that limit, but that means some data was ignore. The frequency that this happen was very low so it didn't have a large impact, but the ideal situation would throw away no data.

The ability to detect emotions present was a core requirement of the program. Without this feature, the program would not be successful. This feature works and can be built upon by the other features.

Emotion Classification Probability Scores

The Raven by Edgar Allan Poe will once again be used as the analysis subject for the same reasons described above. Figure 37 shows the probabilities for curiosity, desire, fear, and sadness to be present.

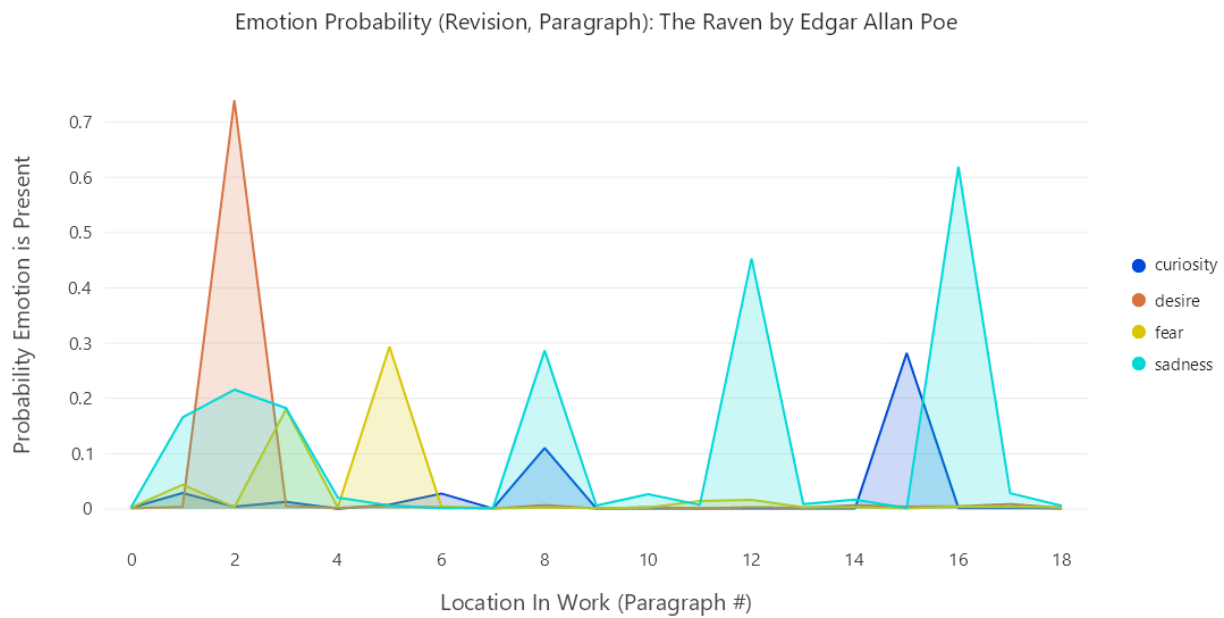


Figure 36 Emotion Probability - Multiple Emotions

Subject: The Raven by Edgar Allen Poe. This shows the emotion probabilities for curiosity, desire, fear, and sadness.

Stanza two has a large spike in desire. When examining the poem, the following partial line is found: "Eagerly I wished the morrow" This is a really clear instance of desire being expressed.

Corresponding with that spike in desire is a smaller, but still noticeable, increase in sadness. The rest of that stanza deals the narrator expressing his feelings of loss relating to the character of Lenore. He also describes the month of December as bleak. The other three major detections of sadness come in stanzas eight, twelve, and sixteen. The eighth stanza talks about the raven brought a smile to the narrator's "sad fancy." The twelfth stanza has a similar phrase in "still beguiling all my sad soul into smiling," The sixteenth stanza has the narrator instructing the raven to give him information: "Tell this soul with sorrow laden" In every instance, the narrator is directly referring to himself as being very sad.

Stanza five sees the narrator attempting to communicate with an unknown person that is later revealed to be the raven. The narrator is looking into darkness, "wondering, fearing, / Doubting" as he tries to understand who has come to visit him. As the narrator directly describes being fearful, the increase in the probability of fear is expected.

In stanza eight, the narrator attempts to communicate with the raven for the first time. Curious as to if the raven was the creature that had echoed the name Lenore back to him a few stanzas earlier, the narrator implores the raven to identify itself with a name. This act of inquisition does show curiosity, but not as strongly as elements of sadness and desire have been seen before.

Figure 37 shows anger separately from the previously discussed emotions as the probably score for anger is much lower than the other emotions and would hard to discern to the values for anger on the previous graph. Examining stanza seventeen reveals that the narrator is shrieking at the raven and every sentence with an exclamation mark. This could be interpreted

as anger, though it is not the most clearcut of assumptions, which explains the low score and the spike in value for anger in this instance.

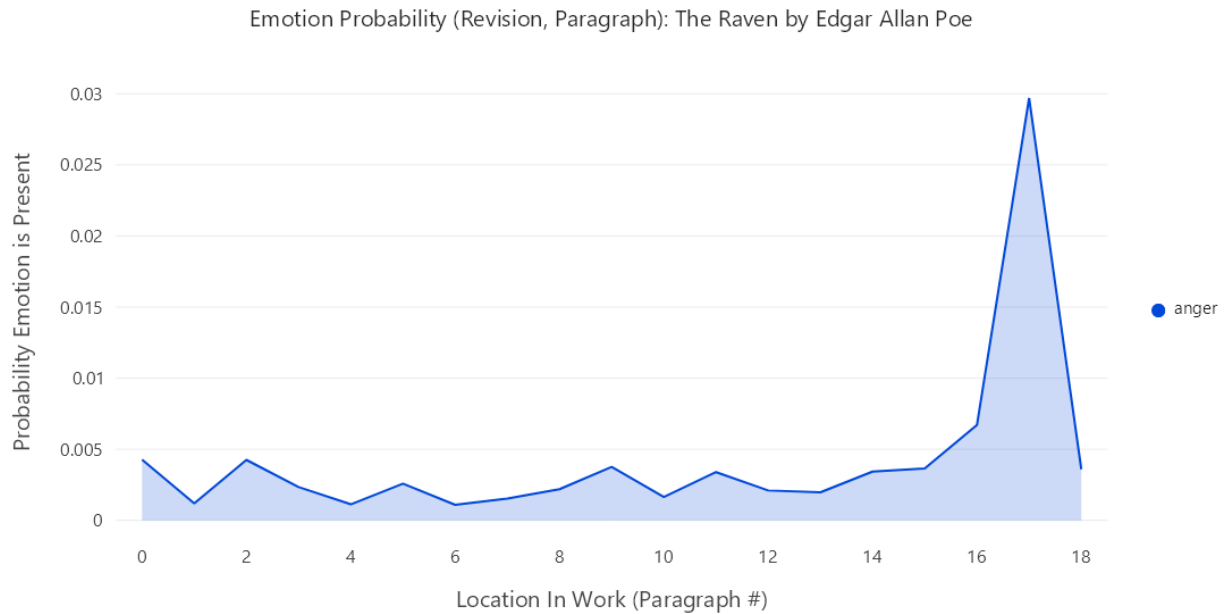


Figure 37 Emotion Probability - Low Chance

Subject: The Raven by Edgar Allen Poe. This shows the emotion probabilities for anger.

Finally, one emotion that is not that present will be examine. Figure 38 shows the probability of pride being present. The models do detect some small level of pride, and the level does fluctuate over the course of the work, but the maximum value is slightly less than half a percent. Comparing this result to the poem confirms that there no paragraphs that give the impression of pride.

Given the higher scores seen in the above analysis, it could be said that a person using this tool could reasonable put a lower threshold on the scores to be evaluated, perhaps one percent or half a percent, depending on the level of detail the analyst is pursuing.

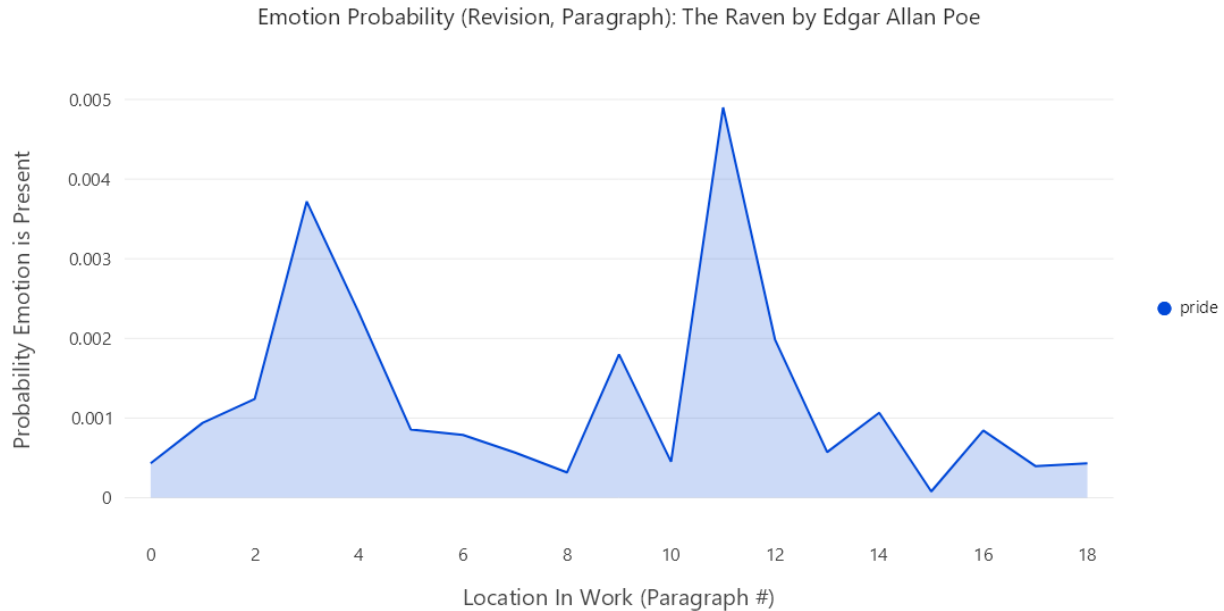


Figure 38 Emotion Probability - Not Present

Subject: The Raven by Edgar Allen Poe. This shows the emotion probabilities for pride, which is practically not present.

The above analysis was performed only by looking for interesting points in the visualizations created by this program. This shows, while not perfect in its implementation, that applying an emotion classifier to small sections of a larger literary work can produce interesting and actionable results.

The ability to detect emotions present at different points within a story was a key requirement of the program. This feature works and is demonstratable.

Entity Tonality

The entity tonality scores are highly effected by any imperfections in the generation of either component part of the entity tonality score (those being the entity presence score and the emotion probability score.)

One of the most important things about the calculation of the entity tonality score is that the entity presence scores are restricted to a range of [0, 1]. This is a very important step to take

as successive mentions of an entity can drastically increase their presence score. The measure of an entities presence being between 0 (not present) and 1 (fully present) are easy to conceptualize. How would a presence score of 4.5 be interpreted? When looking at entity presence alone, this high score allows the entity to still be fully present (score greater than or equal to 1), even after the entity gets mentioned. This can be interpreted as the character having been so central to the story for a few paragraphs/items that the reader doesn't quickly forget that they are present. If the algorithm were to multiple the emotion presence score against an entity presence score greater than 1, there is the potential the final entity tonality is above 1. (e.g. an entity presence of 4.5 times an emotion probably of 0.25 yields an entity tonality score of 112.5.) This reads as "there is a 112.5% chance of the specified emotion when the entity is present.

In the story Alice in Wonderland by Lewis Carroll, the character of Alice often feels anger and annoyance at the same time. The entity tonality scores on display in Figure 39 confirm this claim.

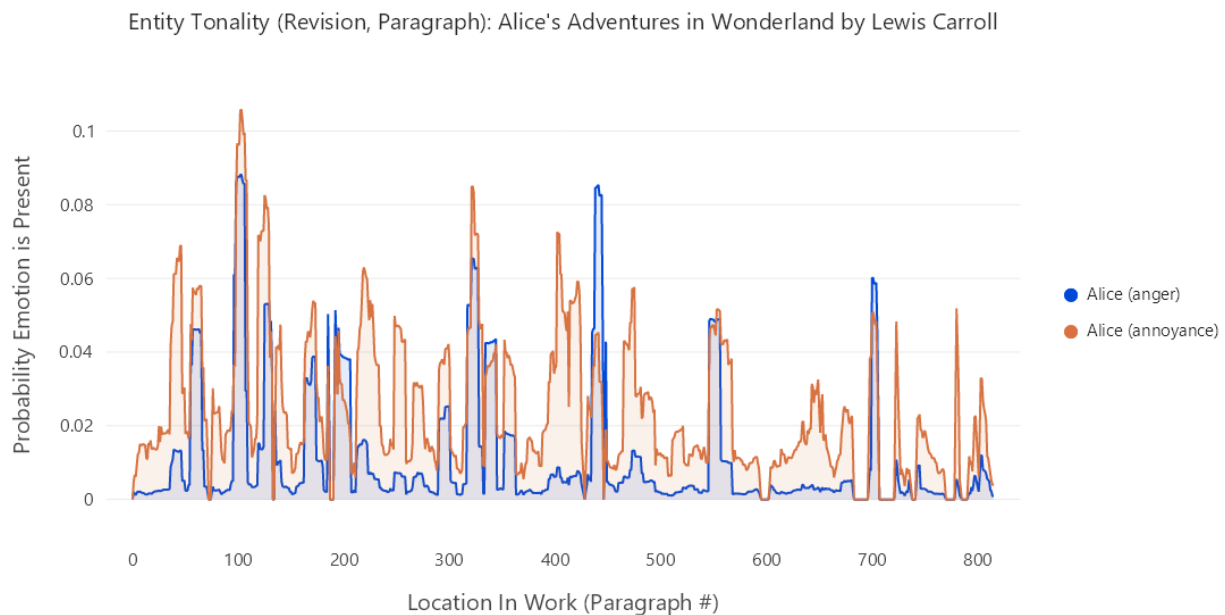


Figure 39 Entity Tonality - Finding Corresponding Emotions for an Entity
Subject: Alice in Wonderland by Lewis Carroll.

Similarly, Alice often alternates between being confused by the strange sights of Wonderland to having flashes of insight when she understands how the world works. Figure 45 shows these alternating moments.

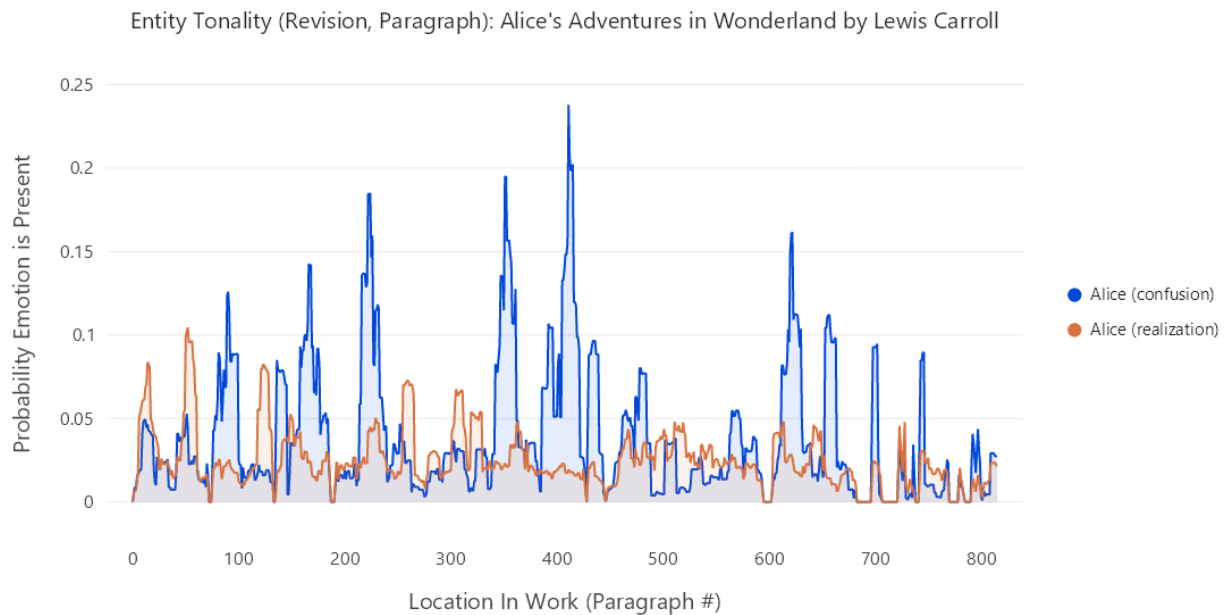


Figure 40 Entity Tonality - Finding Alternating Emotions for an Entity
Subject: Alice in Wonderland by Lewis Carroll.

The user wants to analyze the tea party scene. They know that Alice, the Mad Hatter, the Dormouse, and the March Hare are all present. They identify the scene in the book by filtering by those entities as shown in Figure 41 and Figure 42.

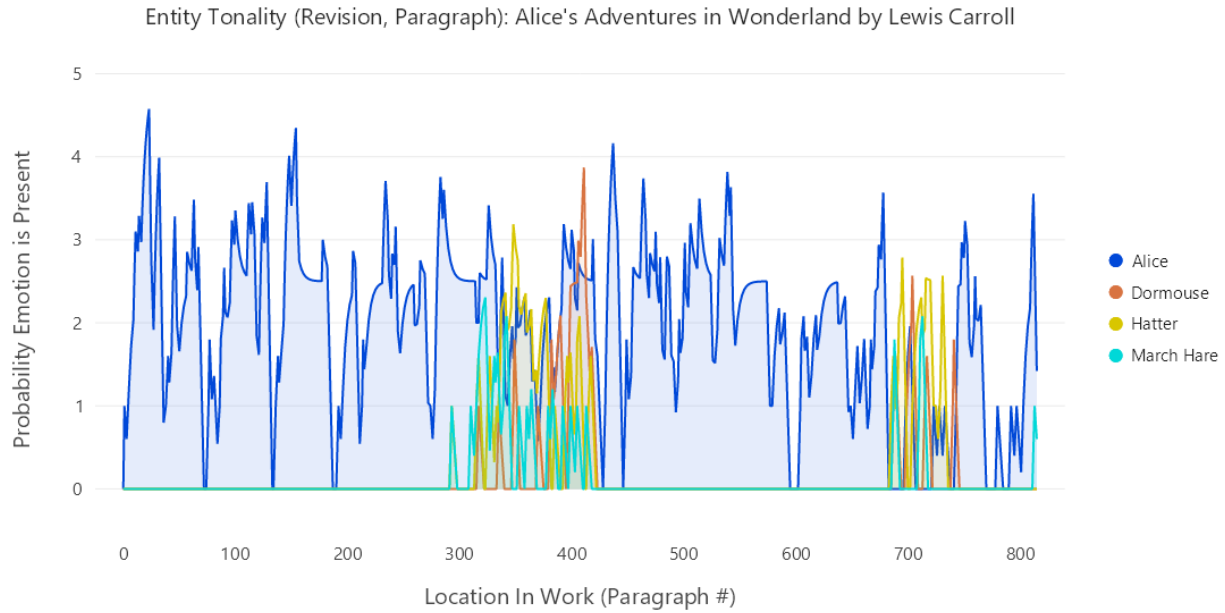


Figure 41 Entity Tonalities – Finding A Scene by Entity Presence (No Emotions Selected)
 Subject: Alice in Wonderland by Lewis Carroll.

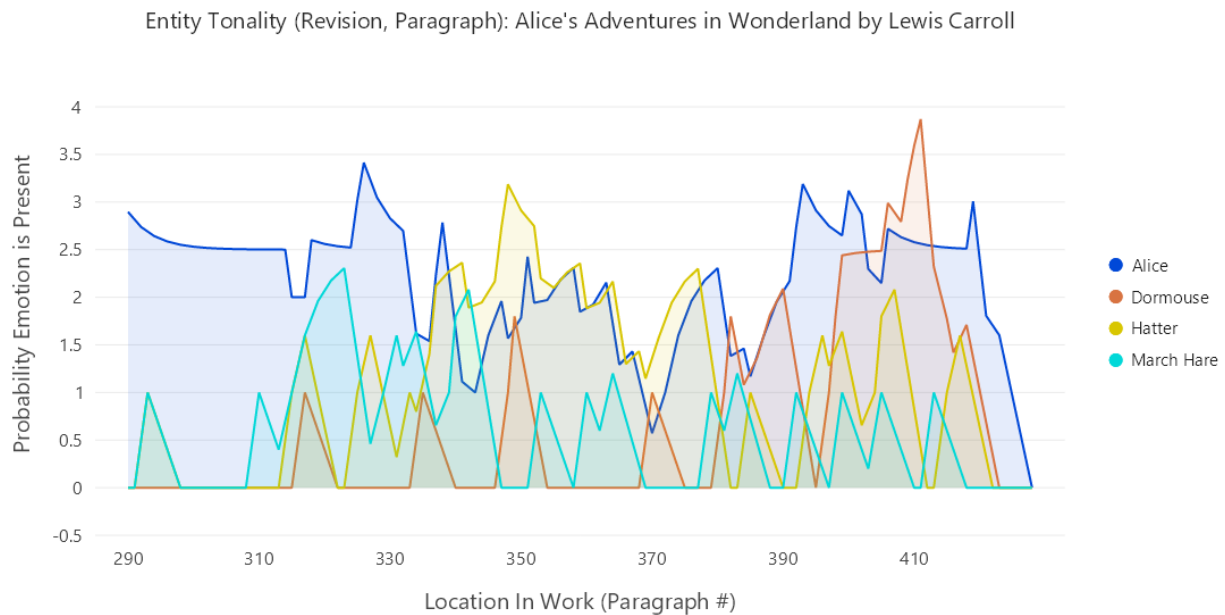


Figure 42 Finding A Scene by Entity Presence (No Emotions Selected, Zoomed In)
 Subject: Alice in Wonderland by Lewis Carroll.

Having found the scene they want to analyze, and thinking that many of the characters felt annoyed during the tea party, the user adds the emotion annoyance to display as shown in Figure 43. This figure shows that even though all four characters are present for the tea party,

they each have slightly different levels of annoyance. This is evidence that a less present character is attributed a smaller amount of the emotion probabilities at that item.

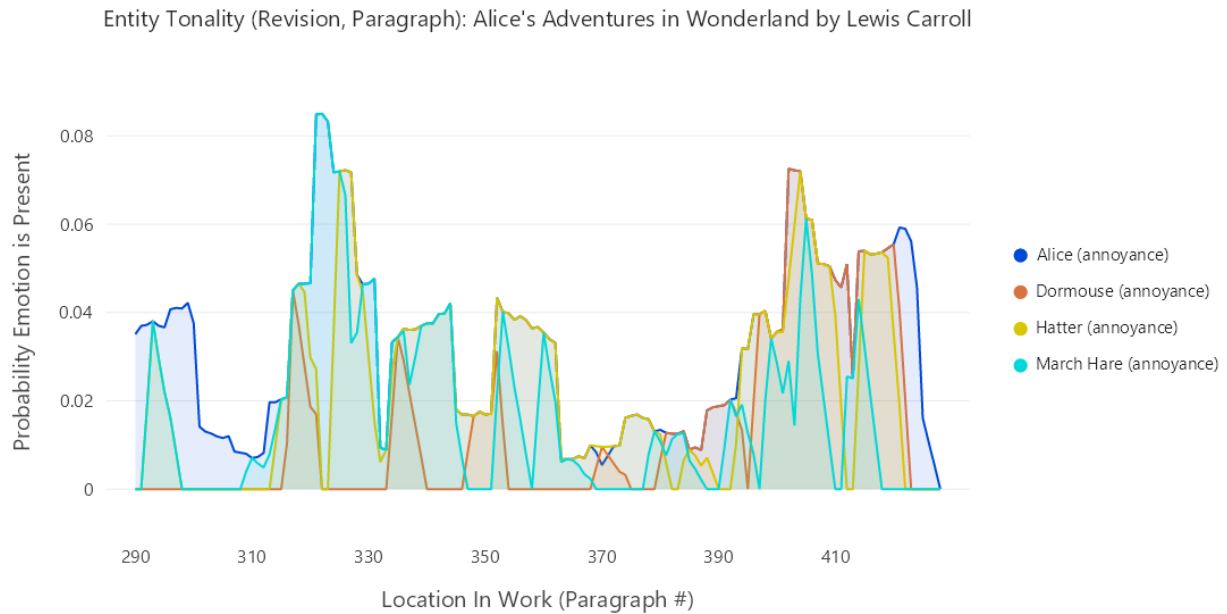


Figure 43 Entity Tonicity – Multiple Characters with Different Tonicity Scores For The Same Emotion
Subject: Alice in Wonderland by Lewis Carroll.

Finally, the user wants to evaluate some summary statistics of the displayed series and brings up the data summary popup as shown in Figure 44. The statistics show the values for the entire series. The sum shows that, overall, Alice was present for more items in the story that were classified as annoying. The max shows that Alice was also present for the highest emotion probability of being annoyed. The average shows the overall emotion probability for the whole series. The average (present) statistic takes the average only when an entity is present (has a score greater than 0.)

Entity	Emotion	Avg	Avg (Present)	Sum	Max
Alice	annoyance	0.0244	0.0266	19.8788	0.1059
Hatter	annoyance	0.0054	0.0286	4.4105	0.0722
Queen	annoyance	0.0045	0.0178	3.6981	0.0590
White Rabbit	annoyance	0.0018	0.0141	1.4781	0.0688

Figure 44 Data Summary Popup

Subject: Alice in Wonderland by Lewis Carroll.

One improvement that could be made to the data summary popup is taking into account the user filters for first item, last item, and max number of items. Another option would be to just display the summary statistics for the window the user has zoomed in on.

These are easy to understand statistics that also reaffirm the user's domain knowledge of the subject matter and show that this metric and associated visualizations are effective.

Revision Tracking

The revision tracking service works as expected. Changes made to tracked files get saved and the user can process them at any time when the main program is running.

There are a few limitations with the revision tracking service as it currently stands. The main reason for most of these limitations is that file tracking features beyond the basics usually requires complex win32 programming. It was decided that spending time on other features was more important than adding to revision tracking capabilities.

First, if the user saves the same file multiple times but doesn't change any content, a new unprocessed revision will still be created. An improvement would be to detect when the file is opened and close and only save the last revision. This makes the most sense as some

people can write or edit for hours at a time without closing the file, but may have dozens of manual saves to make sure they don't lose their work.

Another limitation is the service cannot track when a file is moved to a different location on the file system. It is smart enough to determine if a file still exists though.

A limitation in the main program related to revision tracking is that the data on the works page and database page that displays the unprocessed revisions does not update in real time. It is currently possible to refresh the page by switching to another page and then returning to the original page. A solution to this is to have a simple timer that queries the database for new items and adds them to the collections that are already databound to the view.

One of the main limitations of the revision tracking is that only plain text files are supported as input. Other file types could be easily supported though, especially if there are open source libraries that will allow easy raw text extraction.

Despite the limitations, the goal of the revision tracking feature was met. It easily tracks files that are input to the main program. Users do not have to remember to load each individual revision into the main program; all the user has to do is click a button to process unprocessed revisions whenever they are present.

Ethical Implications

The ethical implications of the creation of any tool need to be considered, especially when that tool makes use of artificial intelligence.

One of the core aspects of this tool is that any data that is put into it stays on the user's system. The app is not connected and will not send data to anyone (though some of the Python emotion classification models may require a small, automated download before they are used.) This is very important given the large, justified concerns about data privacy in the modern world.

This becomes more relevant if this tool gets released to the public. Very few users actually read the terms of service or the end user license agreements for software they install. Many of these terms see the end user giving full rights to their content to the company who owns the program. These terms often appear similar to the following passage: You grant us a non-exclusive, worldwide, royalty-free, perpetual, irrevocable, sublicensable, and transferable, and all others rights required to license to use, publish, access, use, store, preserve, transmit, reproduce, distribute, modify, translate, create derivative works from, and display your content in for the purpose of ...

Adding to the often overreaching terms, if an otherwise well-behaving company gets bought out by another company, the new owners usually can modify the terms of service and the user may find that a large company is profiting off their hard work.

There are also both positive and negative outcomes that could potentially happen if a tool like this was made available to the public.

Positive Outcomes:

- Writers can use this tool to speed up their creative process to see if what they think they are writing aligns with what they are actually writing.
- Writers can discover new ways of thinking about their writing that they may not have previously thought about.
- Editors can use this tool to add to their feedback to their clients.
- A publishing house who may not have read a manuscript for many reasons (lack of staff, a policy of not accepting unsolicited manuscripts, etc.) may run the manuscript through the tool, find something interesting, and decide to spend the time to actually read the manuscript.
- A literary agent may use some of the visualizations generated by this tool to help sell a work to a publisher.

- Readers can learn new things about a favorite literary work without harming anybody.

Negative Outcomes:

- A publishing house may use this tool to reject a manuscript before they read it by creating an additional requirement that manuscripts must meet. (Many larger publishing houses don't accept unsolicited manuscripts however.)
- Students may use this tool to do some of their work for them instead of doing it themselves.

Conclusions

Goals

The goals of this project were to create an application that works offline, manage historical versions of a literary work, allow the user to perform data cleaning without leaving the application, and is able to analyze literary works in new and interesting ways. Every goal was achieved.

Contributions

The final application demonstrates that there is untapped potential in version control for literary works. The implementation in this program is limited, but inspiration should be drawn from existing software version control systems.

Demonstrates that effective machine learning algorithms targeting literary analysis (a complex task) can be deployed to an individual's computer, can run efficiently, produce usable results, and does not need the infrastructure of cloud computing to function.

Gives the ability for end users who are not data scientists the ability to clean their data in a fairly straightforward way.

Measuring multiple emotion probabilities and groups of emotions can produce graphs with forms similar to Kurt Vonnegut's "shapes of stories", but with more than the few shapes he and others have come up with.

Entity tonality combines most of the above features and provides a new way of analyzing literary works.

Future Work

There are a number of areas where the ideas presented here could be further developed. Some of these ideas are focused on improving the functionality of this application while others could find use in a more general sense.

Entity Presence

The entity presence feature developed here is quite simple. The algorithm simply searches a body of text for strings that match the name of named entity as detected by the spaCy library. A more robust solution would analyze the type of presence of a character (e.g. are they present and interacting with other characters, are they not present but being thought about, are they present but not directly participating in the story, etc.)

Emotion Classification

The model used here is built using a dataset comprised of comments on the website Reddit. The writing style used for internet discussion is often very different from written works of literature. Training the model on a dataset of literary works could potentially increase the accuracy of the results.

Furthermore, the data used in the training of the model comes from the last 18 years (the time since Reddit was founded.) Languages change over time, and trying to analyze older literary works with a modern dataset may be problematic. Datasets from time periods could increase the efficacy of the model.

Entity Tonality

Entity tonality is the largest success of this project.

Revision Tracking

Many users today expect their apps to be connected to some kind of data cloud where they can share or save data without having to worry about local files. Enabling some kind of cloud saving would make collaboration with others easier. (e.g. if a writer and their editor both have access to the same source file, all new additions and edits could be made to a single file, which could then be tracked and distributed to any user who has access to that file.

Another desirable feature would be the ability to track how long each revision took to produce. This could be used in other analysis. (This is further discussed below.)

Other Analytics

The Flesch Reading Ease and Flesch-Kincaid Grade Level metrics were included in the hopes that they could be used in conjunction with dialogue detection. On they're own, they're useful statistics, but there is also the potential to combine them with dialogue detection to try to detect any differences between dialogue and narrative text.

The reading level metrics could also be applied on a per-entity basis to see if a writer purposely alters the complexity level to match specific entities.

Other dialogue related metrics could be measuring the count of dialogue tags, the frequency that dialogue is written as one large block versus being interspaced with narrative text, and measuring how much each entity speaks.

Frequently used phrases or combinations of words could also be helpful to a creative type to check if they're overusing certain items.

Additional basic statistics could help authors identify patterns about how they write. Possessing the start time in addition to the end time, and thus the overall time it took to produce a revision, could help writers identify which parts of the day they are the most productive and least productive. The same concept could be applied to days of the week.

Producing a difference file for two revisions (i.e. seeing what words are new, changed, or deleted) could be utilized in a few ways as well. Differences, when combined with writing time information, and performing entity detection on new text might show that writing certain entities is easier than writing other entities.

Comparing the emotion probabilities for different works could help identify trends for a writer or a series of books.

The interactive dashboards could be improved via the inclusion of automatic chapter/scene start markers.

A user might better be able to interpret their visualizations if they had the ability to create annotations for the graphs.

References

Data

Alighieri, Dante. (2005). *The Divine Comedy*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/8800>

Baum, L. Frank. (1993). *The Wonderful Wizard of Oz*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/55>

Carroll, Lewis. (2008). *Alice's Adventures in Wonderland*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/11>

Carroll, Lewis. (2006). *Alice's Adventures Under Ground*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/19002>

Carroll, Lewis. (2008). *The Hunting of the Snark: An Agony in Eight Fits*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/13>

Carroll, Lewis. (2008). *Through the Looking-Glass*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/12>

Dickens, Charles. (2004). *A Christmas Carol in Prose; Being a Ghost Story of Christmas*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/46>

Jacob and Wilhelm Grimm. (2001). *Grimms' Fairy Tales*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/2591>

Hawthorne, Nathaniel. (2008). *The Scarlet Letter*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/25344>

Milne, A. A.. (2022). *Winnie-the-Pooh*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/67098>

Poe, Edgar Allan. (2000). *The Works of Edgar Allan Poe — Volume 2*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/2148>

Poe, Edgar Allan. (2000). *The Works of Edgar Allan Poe — Volume 5*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/2151>

Shakespeare, William. (1998). *Hamlet, Prince of Denmark*. Urbana, Illinois: Project Gutenberg. Retrieved November 27, 2023, from <https://www.gutenberg.org/ebooks/1524>

Open Source Software

.NET MAUI Community Toolkit. (2023, November 27, 2023). GitHub.

<https://github.com/CommunityToolkit/Maui>

Krueger, F. A. (2021, May 3). *praeclarum/sqlite-net*. GitHub.

<https://github.com/praeclarum/sqlite-net>

Sink, E. (2023, November 30). *SQLitePCLRaw*. GitHub.

<https://github.com/ericsink/SQLitePCL.raw/>

Rodríguez, A. (2023, December 2). *LiveCharts2*. GitHub. <https://github.com/beto-rodriguez/LiveCharts2>

Miras, A. (2023, October 21). *SpacyDotNet*. GitHub.

<https://github.com/AMARostegui/SpacyDotNet>

Newton-King, J. (2021, May 3). *JamesNK/Newtonsoft.Json*. GitHub.

<https://github.com/JamesNK/Newtonsoft.Json>

Akgul, E. (2023, December 2). *Maui.DataGrid*. GitHub.

<https://github.com/akgulebubekir/Maui.DataGrid>

Nguyen, T. (2023, October 19). *ColorPicker.Maui*. GitHub.

<https://github.com/trungnt2910/ColorPicker.Maui>

Valbjørn, sloev / J. (2023, November 24). *Spacy Syllables*. GitHub.

<https://github.com/sloev/spacy-syllables>

google-research/goemotions at master · google-research/google-research. (n.d.). GitHub.

<https://github.com/google-research/google-research/tree/master/goemotions>

SamLowe/roberta-base-go_emotions · Hugging Face. (2023, June 1). Huggingface.co.

https://huggingface.co/SamLowe/roberta-base-go_emotions

go_emotions-dataset/eval-roberta-base-go_emotions.ipynb at main · samlowe/go_emotions-

dataset. (n.d.). GitHub. [https://github.com/samlowe/go_emotions-dataset/blob/main/eval-](https://github.com/samlowe/go_emotions-dataset/blob/main/eval-roberta-base-go_emotions.ipynb)

[roberta-base-go_emotions.ipynb](https://github.com/samlowe/go_emotions-dataset/blob/main/eval-roberta-base-go_emotions.ipynb)

arpanghoshal/EmoRoBERTa · Hugging Face. (n.d.). Huggingface.co.

<https://huggingface.co/arpanghoshal/EmoRoBERTa>

Writing Tools

Microsoft. (n.d.). Introducing OneNote. [https://support.microsoft.com/en-us/office/introducing-](https://support.microsoft.com/en-us/office/introducing-onenote-38be036d-5b5a-49ad-83be-292fe53ad7b3)

[onenote-38be036d-5b5a-49ad-83be-292fe53ad7b3](https://support.microsoft.com/en-us/office/introducing-onenote-38be036d-5b5a-49ad-83be-292fe53ad7b3)

Evernote. (2019). Best Note Taking App | Organize Your Notes with Evernote. Evernote.

<https://evernote.com/>

Grammarly. (2023). Write your best with Grammarly. Grammarly.com.

<https://www.grammarly.com>

Hemingway Editor. (n.d.). Hemingwayapp.com. <https://hemingwayapp.com/>

Writer's Desk. (n.d.). AutoCrit Online Editing. <https://www.autocrit.com/writers-desk/>

Writing Analytics: The Ultimate Productivity Tool for Writers. (n.d.). www.writinganalytics.co.
<https://www.writinganalytics.co/>

Scrivener | Literature & Latte. (2019). Literatureandlatte.com.
<https://www.literatureandlatte.com/scrivener/overview>

Writing App - Creative Writing Software. (n.d.). Milanote. <https://milanote.com/product/writing-software>

The Reedsy Book Editor: A Powerful Writing Tool. (n.d.). Reedsy. <https://reedsy.com/write-a-book>

bibisco: Best Novel Writing Software for Writers. (n.d.). Bibisco.com. <https://bibisco.com/>

Atticus - An Author's Best Friend. (n.d.). Atticus. <https://www.atticus.io/>

Quoll Writer - focus on your words. (n.d.). Quollwriter.com. <http://quollwriter.com/>

LivingWriter - The Best Writing App for Authors, Novelists, and Beginners | Novel Writing Software, Book Writing Apps. (n.d.). Livingwriter.com. <https://livingwriter.com/>

Mellel. (n.d.). Www.mellel.com. <https://www.mellel.com/>

Research

David Comberg. (2010, October 30). Kurt Vonnegut on the Shapes of Stories [Video]. YouTube.

<https://www.youtube.com/watch?v=oP3c1h8v2ZQ>

abonato99. (2016, July 13). *The shapes of stories: from Vonnegut to big data*. The Intrepid Mathematician. <https://anthonybonato.com/2016/07/13/the-shapes-of-stories-from-vonnegut-to-big-data/>

Jones, R. E. (2016, December 21). *UVM Researchers Measure Emotion in Stories*. Seven Days. <https://www.sevendaysvt.com/news/uvm-researchers-measure-emotion-in-stories-3899808>

Onwuegbuzie, A., Leech, N., & Collins, K. (2012). Qualitative Analysis Techniques for the Review of the Literature. *The Qualitative Report*, 17(56), 1–28.

<https://files.eric.ed.gov/fulltext/EJ981457.pdf>

What Can Computer Algorithms Tell Us About Literature? (n.d.). Stanfordrewired.com.

<https://stanfordrewired.com/post/computer-algorithms-and-literature>

Hoover, D. L. (2013). Textual Analysis. *Literary Studies in the Digital Age*.

<https://doi.org/10.1632/lstda.2013.3>

Antons, D., Breidbach, C. F., Joshi, A. M., & Salge, T. O. (2021). Computational Literature Reviews: Method, Algorithms, and Roadmap. *Organizational Research Methods*. <https://sci-hub.se/10.1177/1094428121991230>

What is the Difference Between NLP, NLU, and NLG? (n.d.). Slator.

<https://slator.com/resources/what-is-the-difference-between-nlp-nlu-nlg/>

Westley, A. (2023, April 29). *A Modern Take on Literary Analysis: Using ChatGPT-4 and a Web*

Browser. Medium. [https://medium.com/@allen.westley/a-modern-take-on-literary-analysis-](https://medium.com/@allen.westley/a-modern-take-on-literary-analysis-using-chatgpt-4-and-a-web-browser-6625f3e1374e)

[using-chatgpt-4-and-a-web-browser-6625f3e1374e](https://medium.com/@allen.westley/a-modern-take-on-literary-analysis-using-chatgpt-4-and-a-web-browser-6625f3e1374e)

Miscellaneous

James Montemagno. (2022, July 7). MVVMS... A Better MVVM? Model-View-ViewModel-

Services Explained [Video]. YouTube. <https://www.youtube.com/watch?v=ve0DFu-arD8>

Icons8. (2019). *Download free icons, music, stock photos, vectors*. Icons8.com.

<https://icons8.com/>

Appendices

Appendix A: User Interface Screenshots

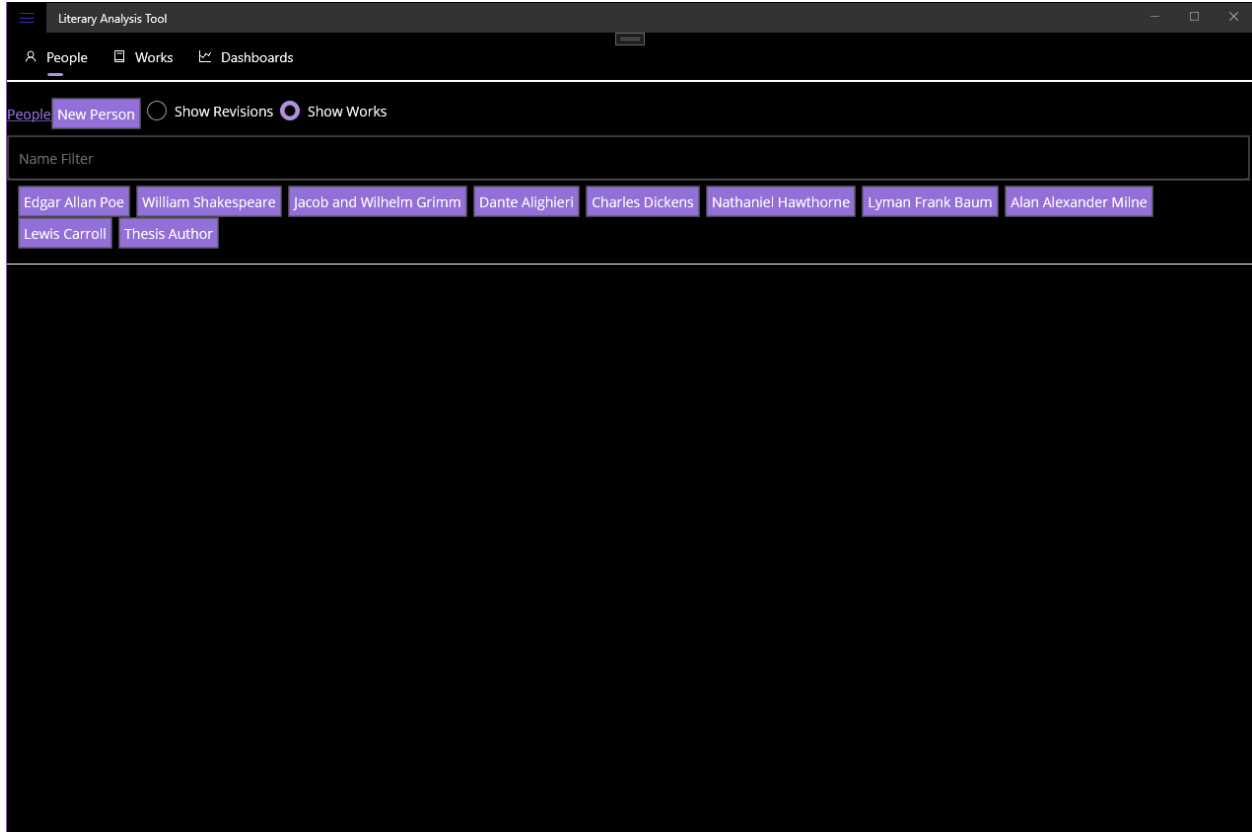


Figure 45 User Interface - People View

The view the user gets when they start the program. This displays the People in the database.

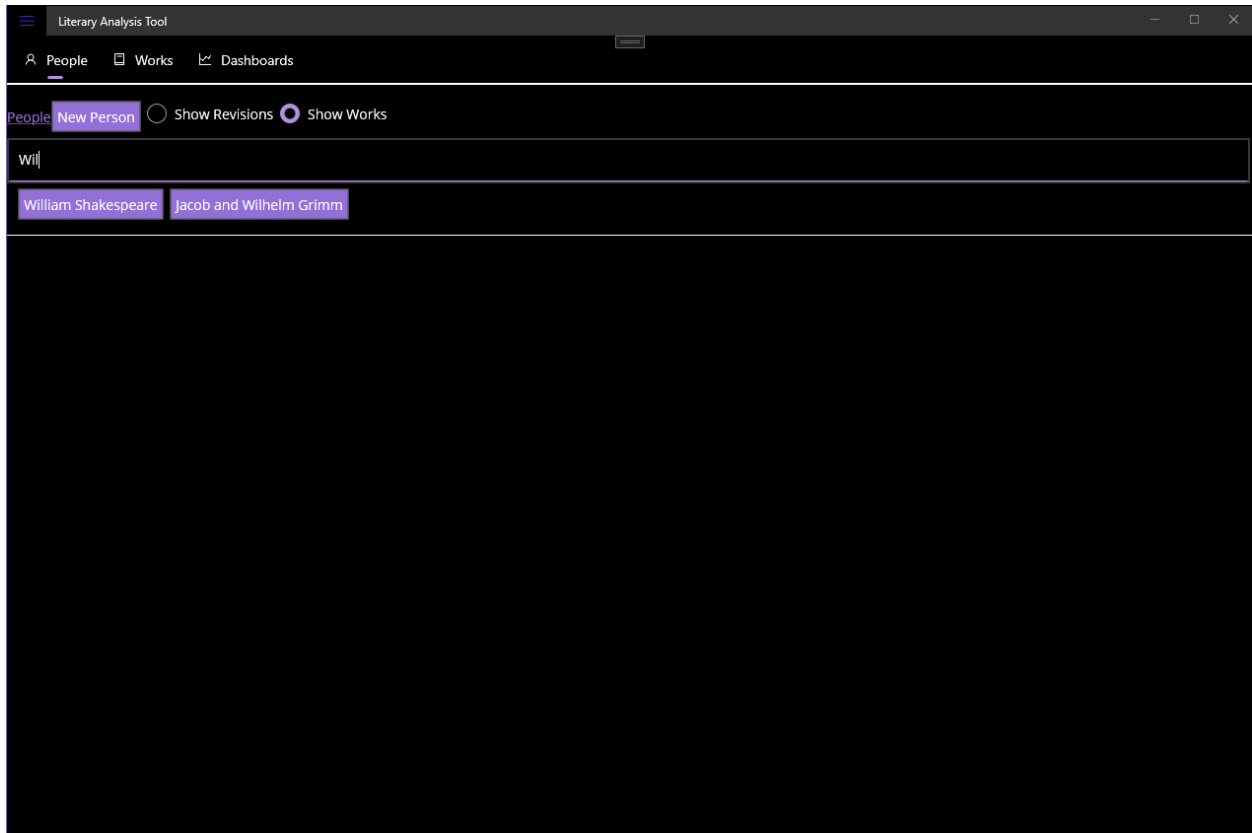


Figure 46 User Interface - People View, Filtering

The user can filter the shown people by typing into the text field. In this case the user typed "Wil" which matches "William Shakespeare" and "Jacob and Wilhelm Grimm"

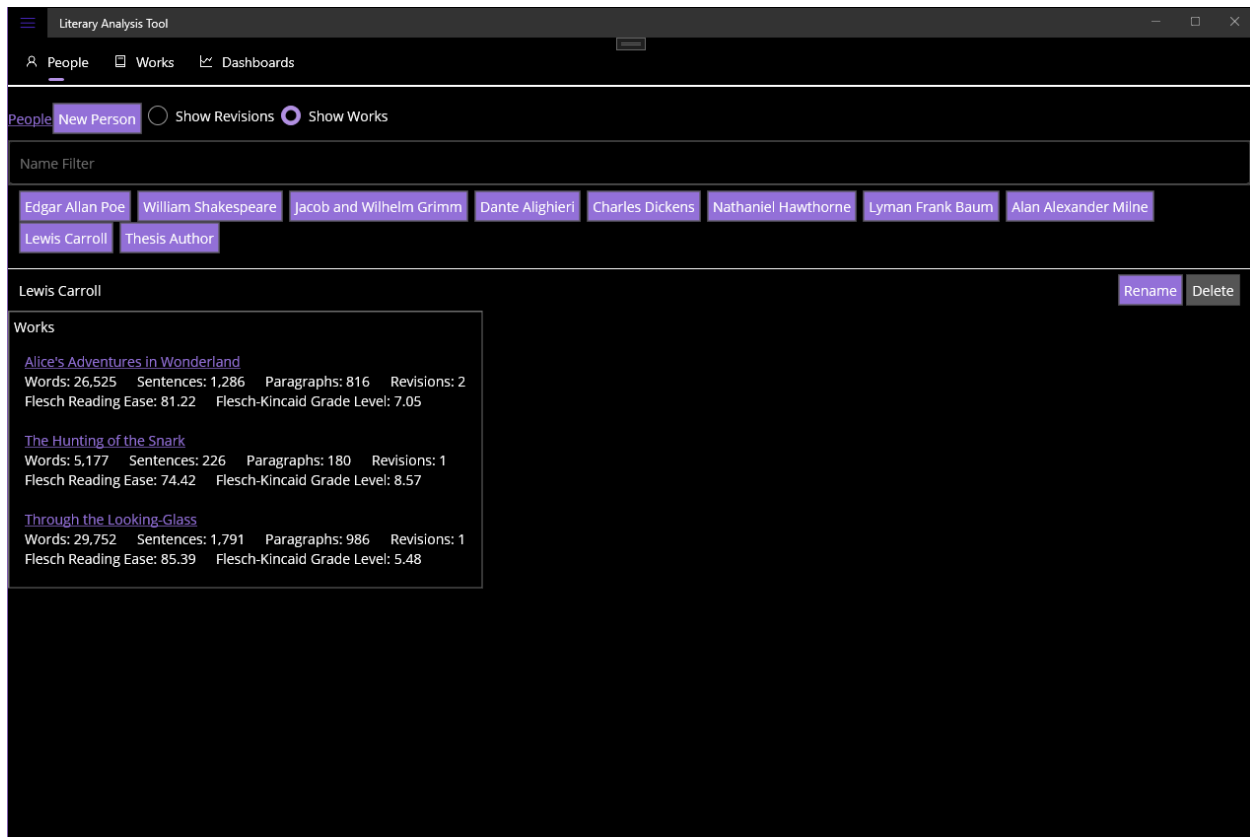


Figure 47 User Interface - Sorting by Works

Clicking on a person in the People view shows either the works they've contributed to or the revisions that are attributed to them. By default works are shown. Each work displays summary statistics about itself.

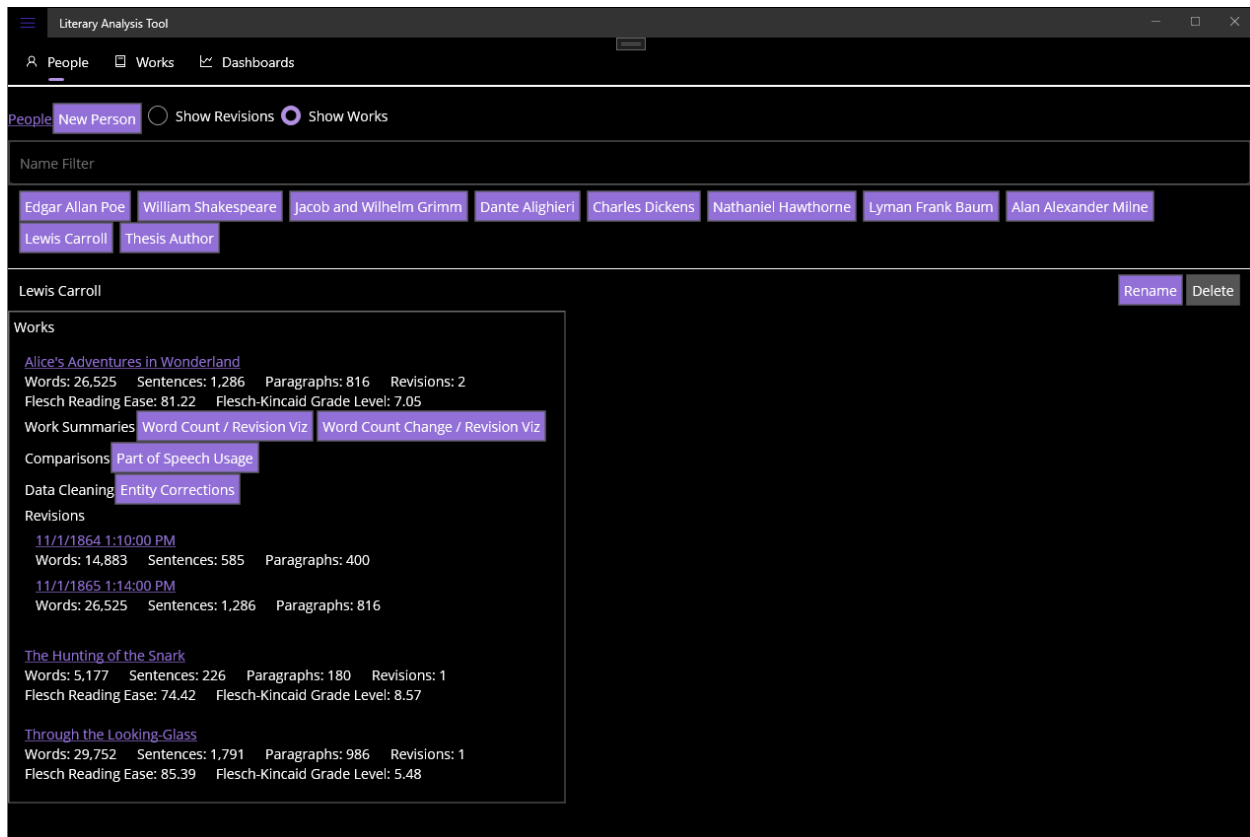


Figure 48 User Interface - Expanding a Work to View Revisions

Clicking on a work in expands that work to show all revisions in that work that are attributed to the selected person. Select visualization options that are valid only on works are shown. These buttons will not appear if the certain conditions are not met. (e.g. If there is only one revision in a given work, there is no need to display "Word Count / Revision" and "Word Count Change / Revision.")

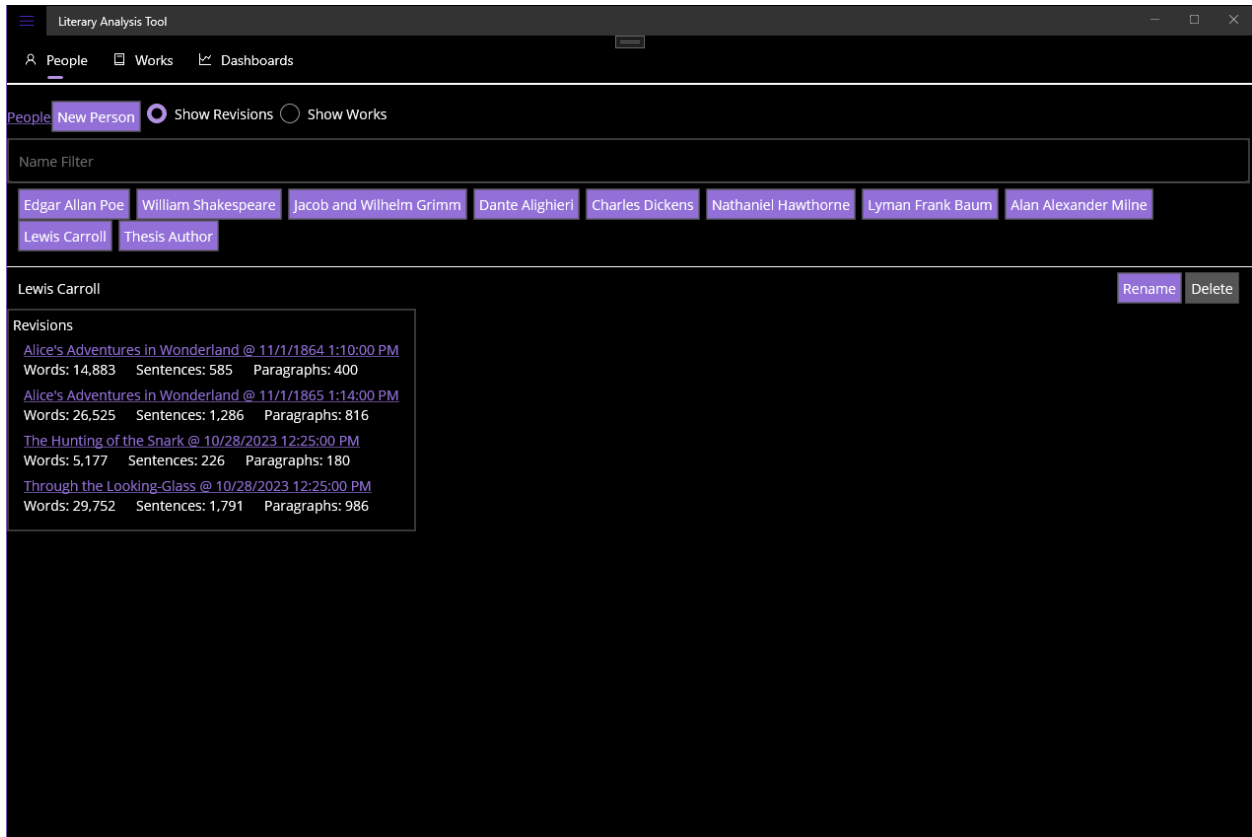


Figure 49 User Interface - Sorting by Revisions

Clicking on a person in People view with the Show Revisions option toggled to on shows all the Revisions attributed to that person.

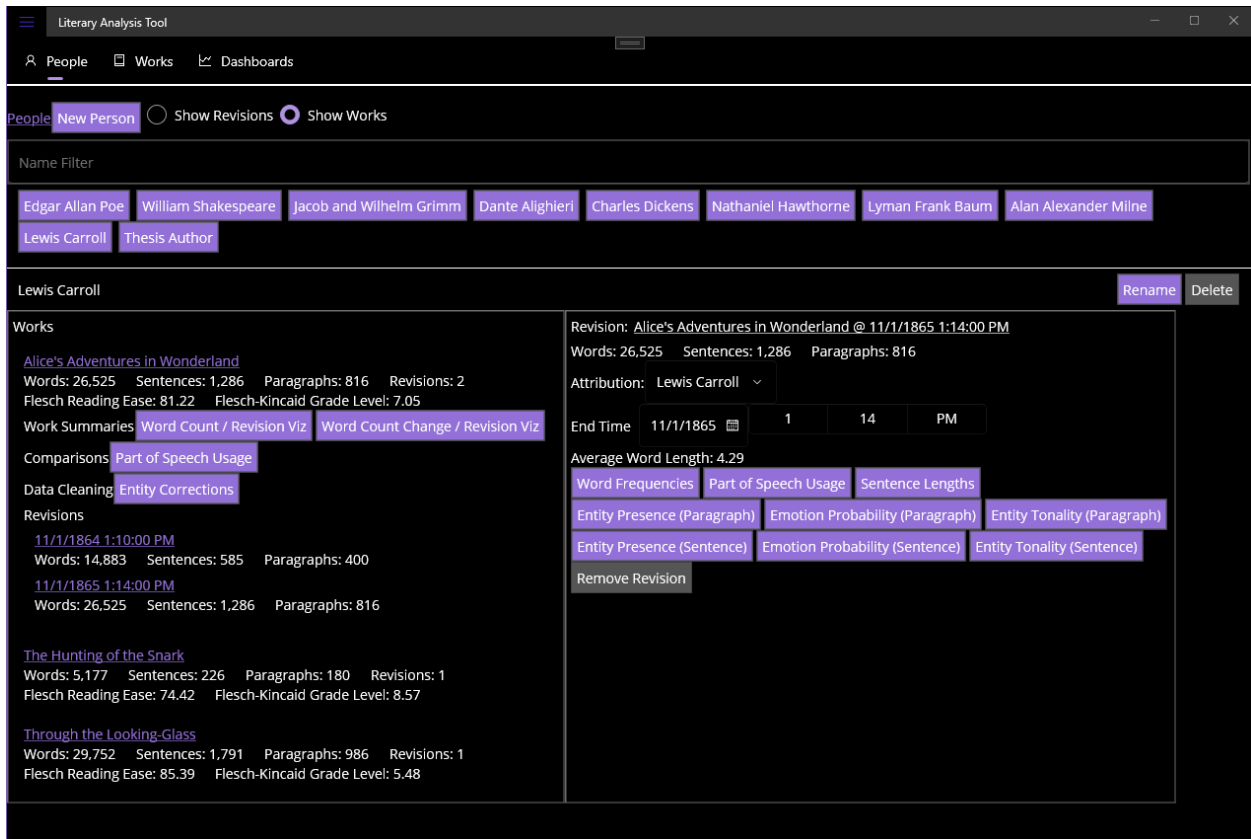


Figure 50 User Interface - Selecting a Revision

Clicking on any revision shows the details of that revision. Summary statistics for the revision are shown. This is also where the revision attribution to a person can be changed. Other information, such as the time the revision was completed can also be changed.

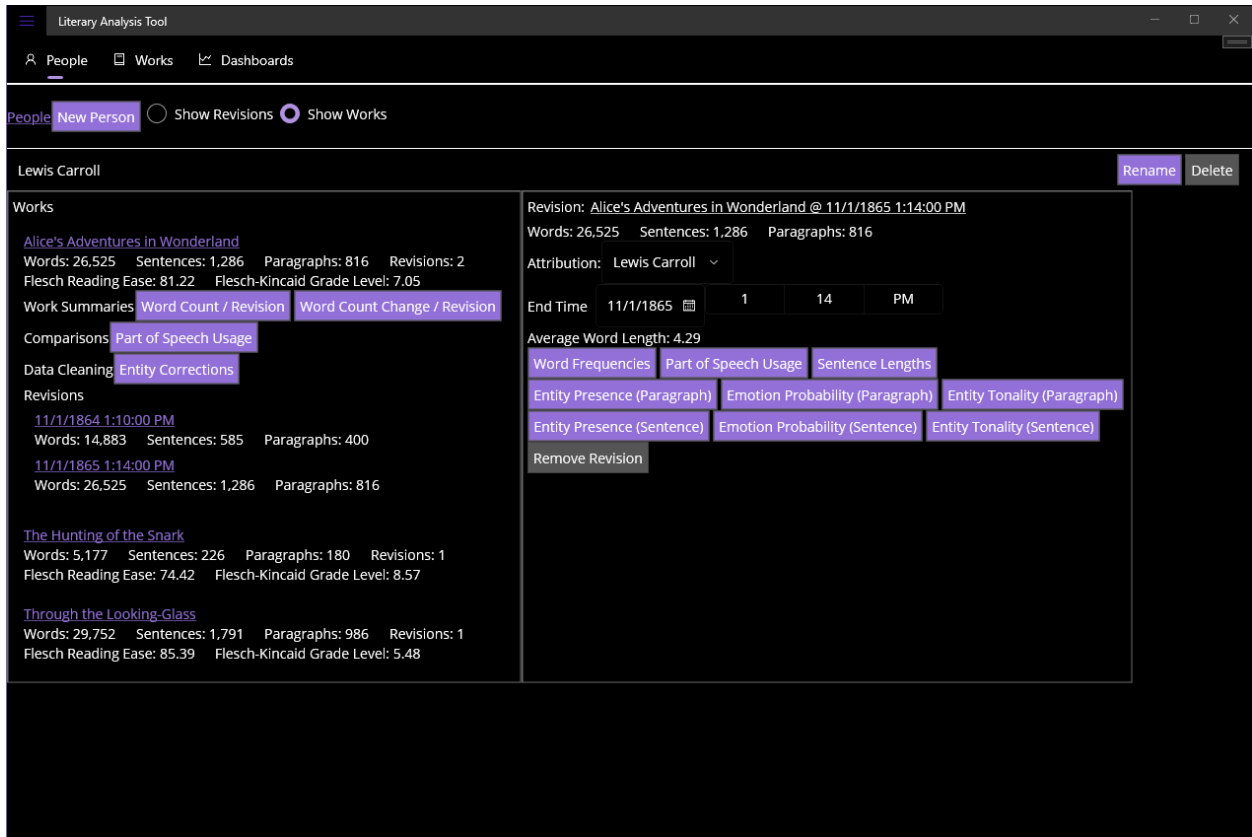


Figure 51 User Interface - Hiding The People List

If the visual display of all the People gets too cluttered for the user, they can hide all the people by clicking on the "People" label in the top left corner.

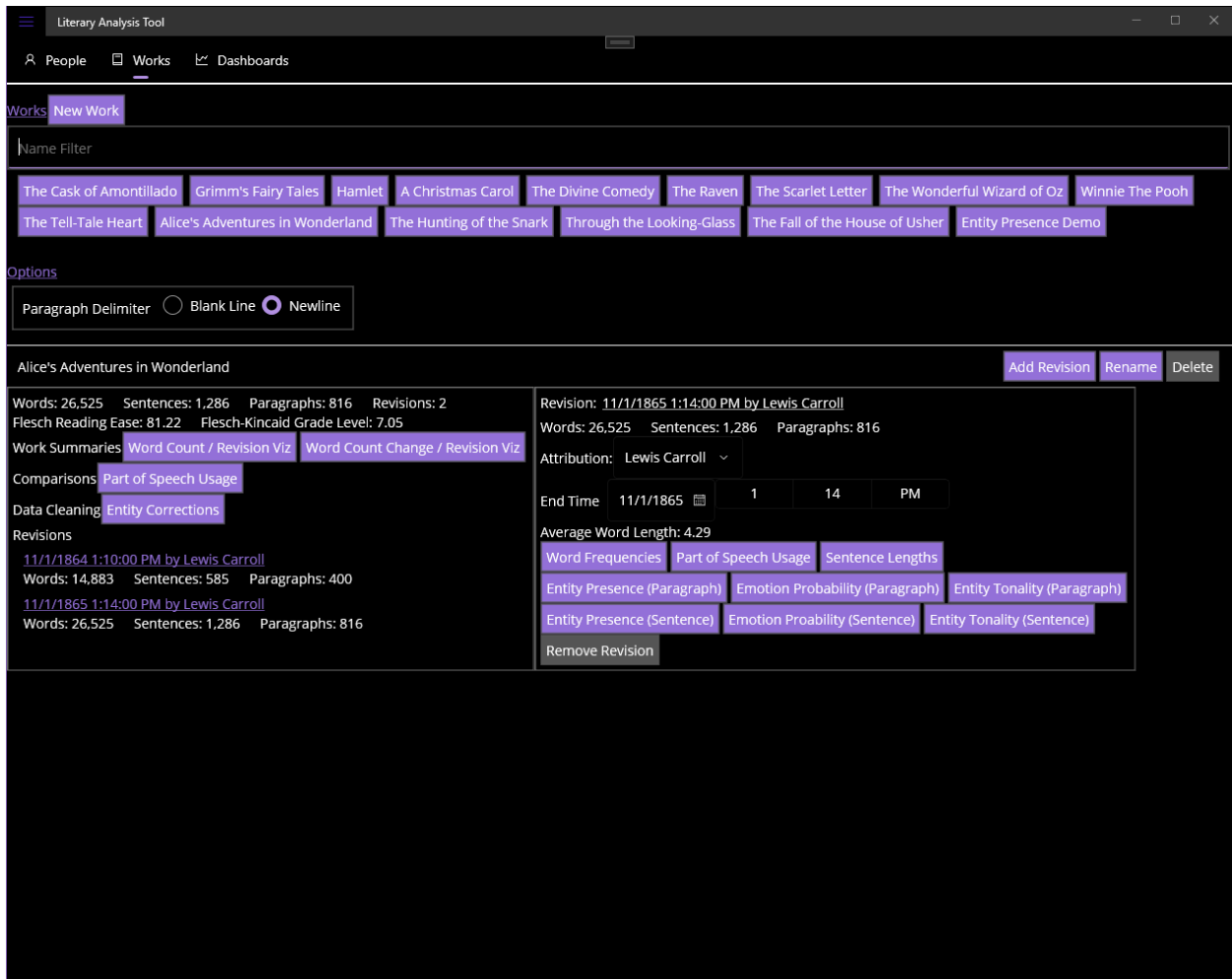


Figure 52 User Interface - Works View

The Works view works mostly the same as the People view. Instead of selecting a person and viewing any works and revisions they've contributed to, the user selects a work and can see all revisions for that work, no matter who made the Revision. This is also where the user can add new revisions to the selected work. When a revision is added by selecting a local file to read, the location of the file also gets added to the database. If the version control service is running, it uses these file locations to watch for new revisions. When processing a new revision, either by manually loading a file or by using data stored by the windows service, the program will use the chosen Paragraph Delimiter option.

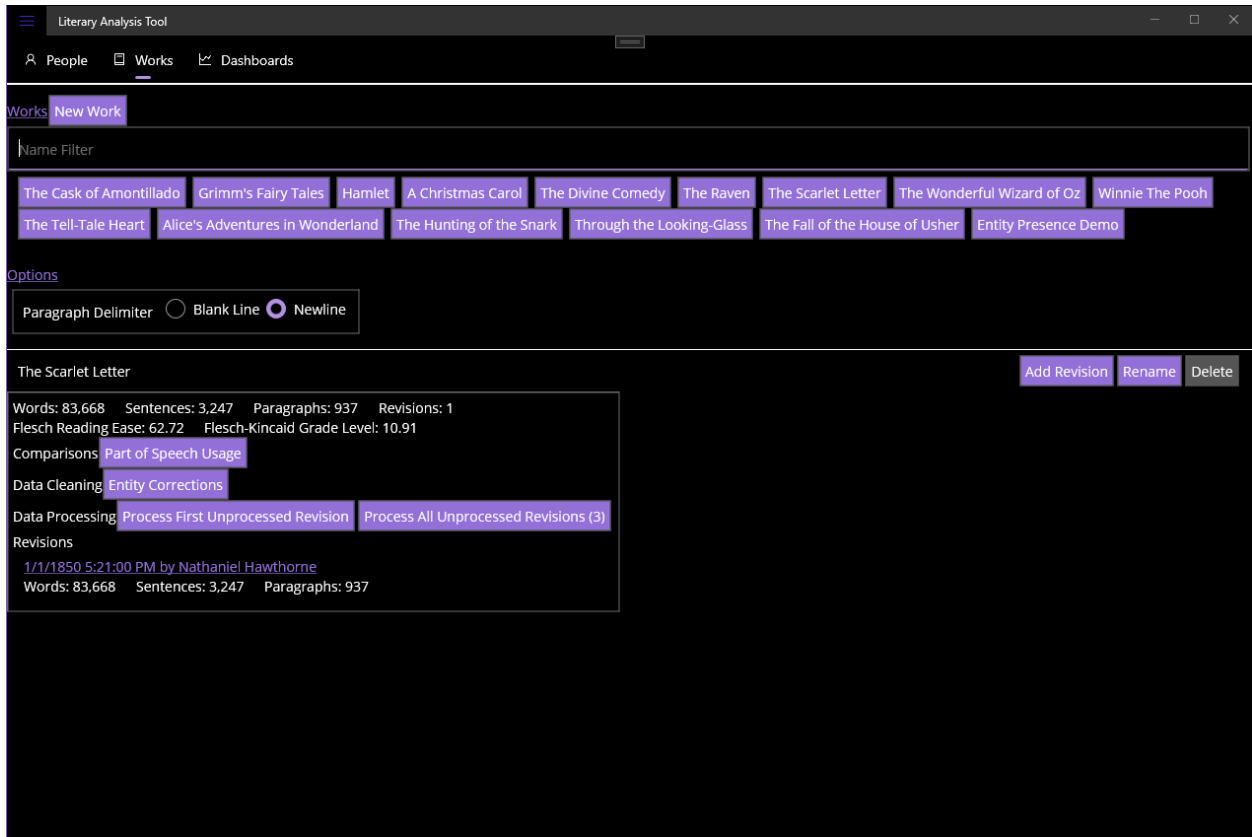


Figure 53 User Interface - Works View - Data Processing

If the windows service has stored data about a revision that was made to a tracked file, the user can process that data from Works page. The user can either process the revisions one by one, or all at once.

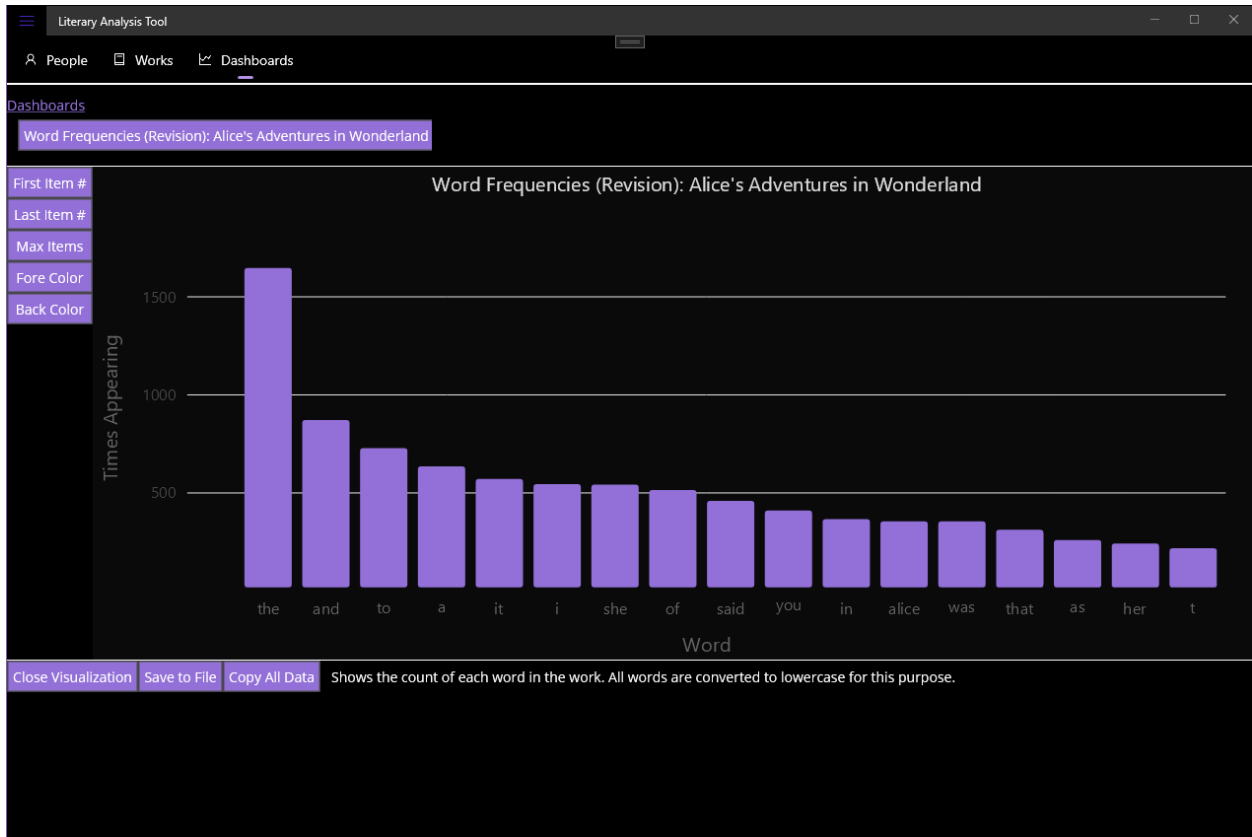


Figure 54 User Interface - Visualization - Word Frequencies

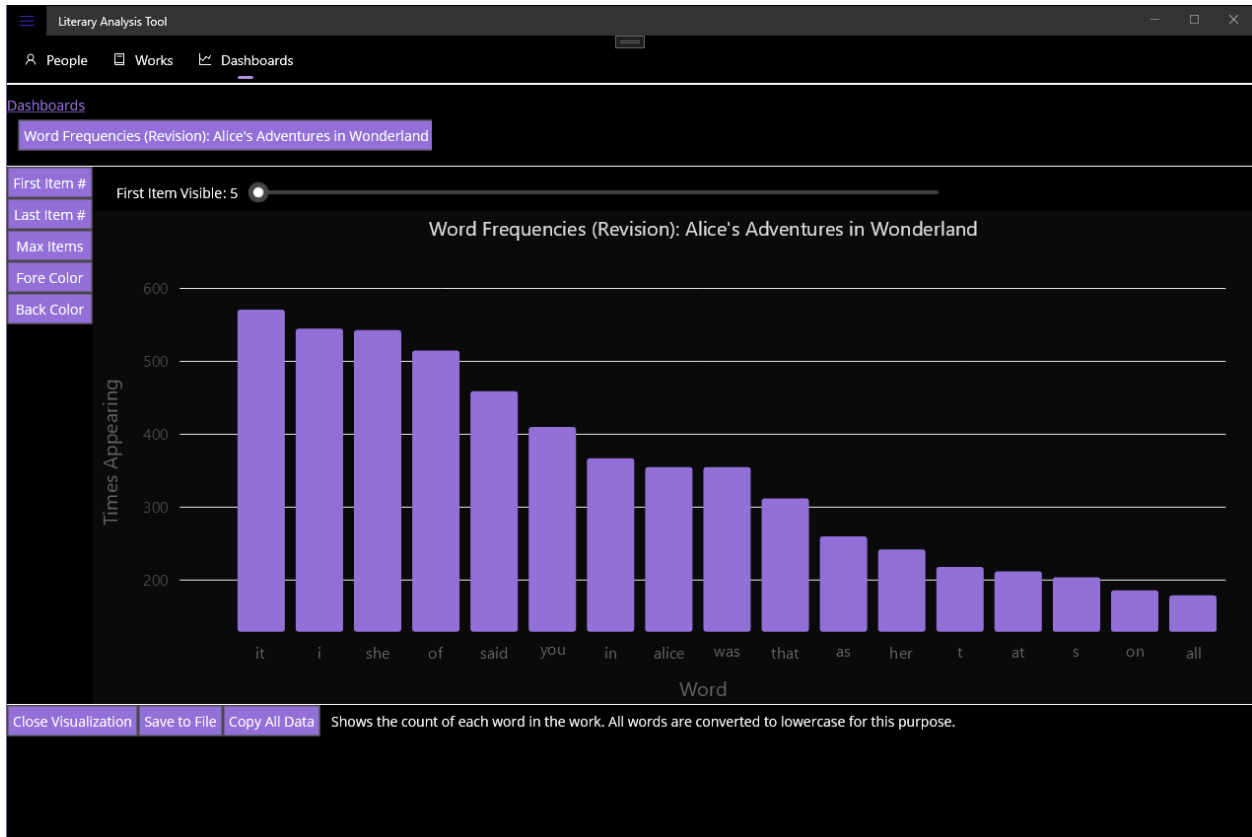


Figure 55 User Interface - User Filter - First Item

This is the same visualization shown in Figure 54, except the first item visible filter has been set to 5.

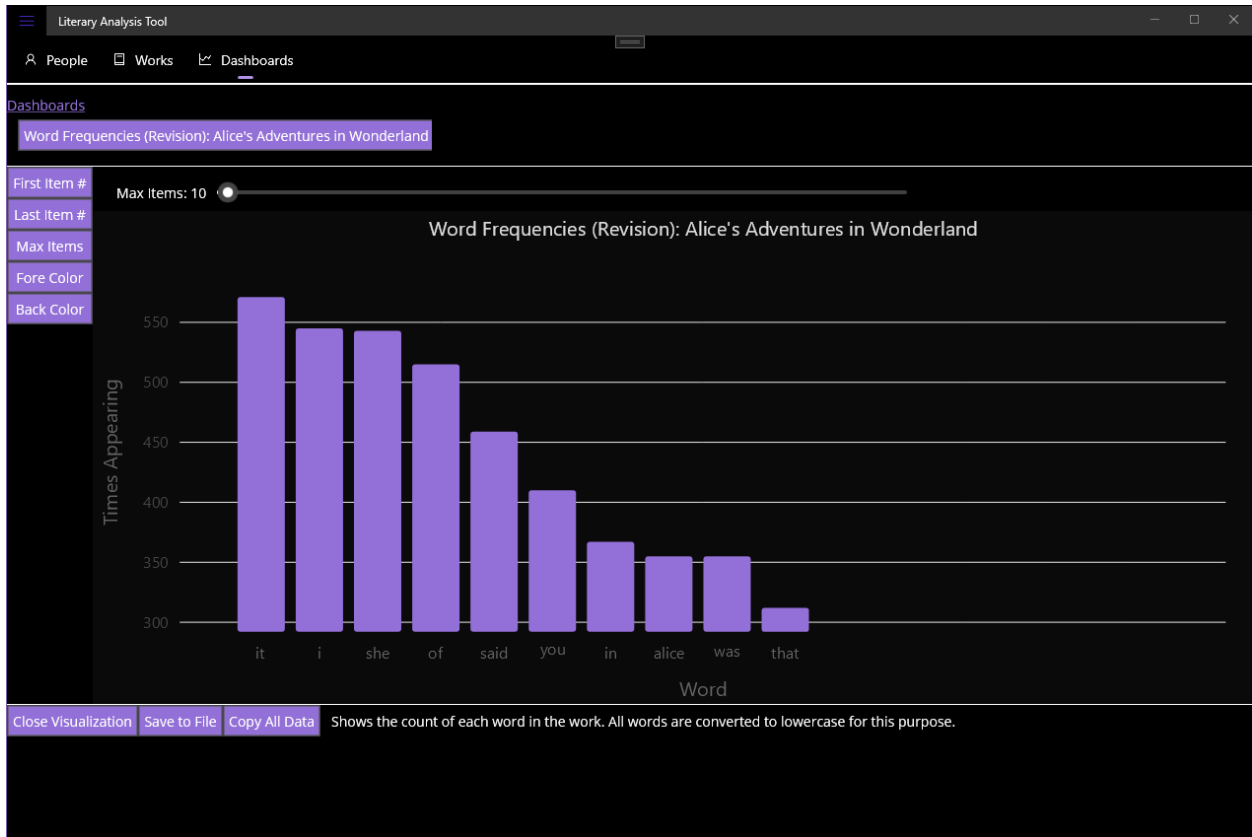


Figure 56 User Interface - User Filter - Max Items

This is the same visualization shown in Figure 55, except the first item visible filter has been set to 5.

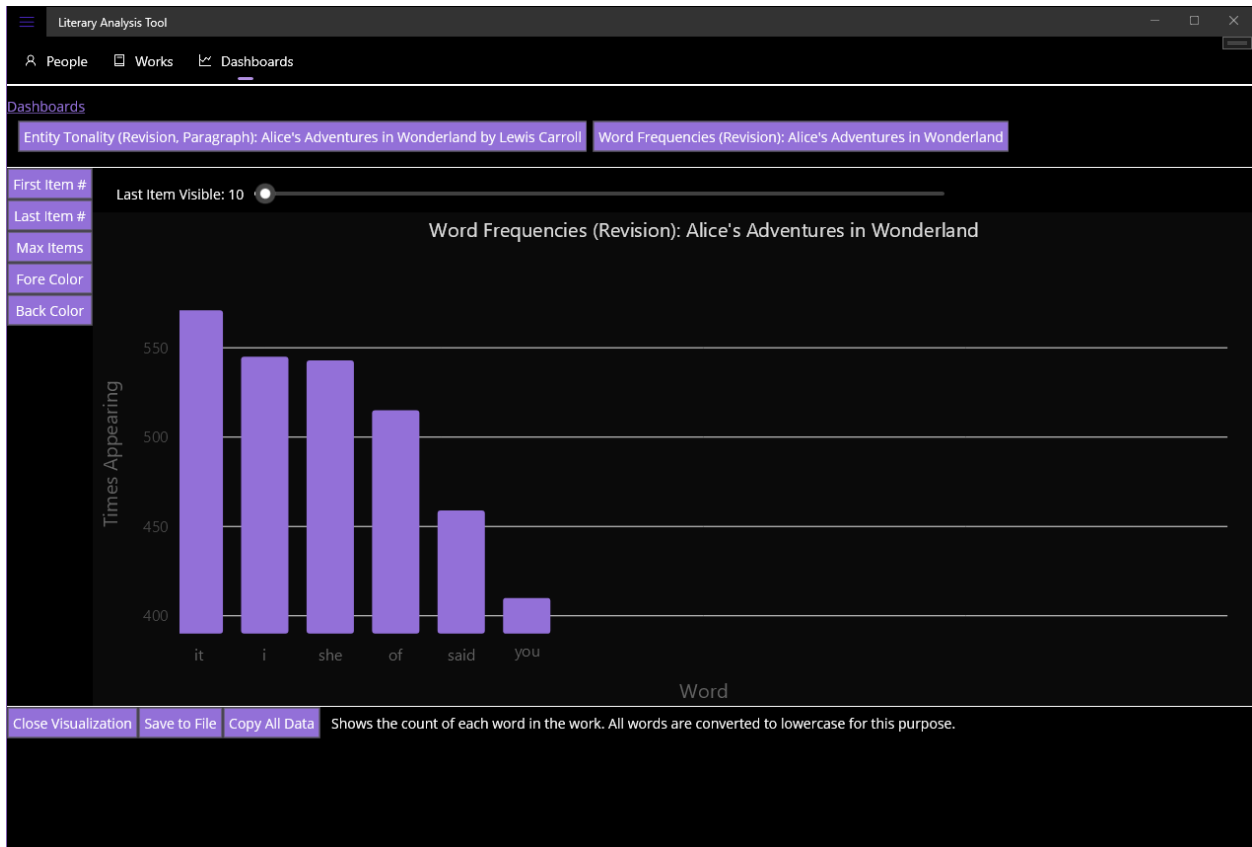


Figure 57 User Interface - User Filter - Last Item

This is the same visualization shown in Figure 56, except the first item visible filter has been set to 10.

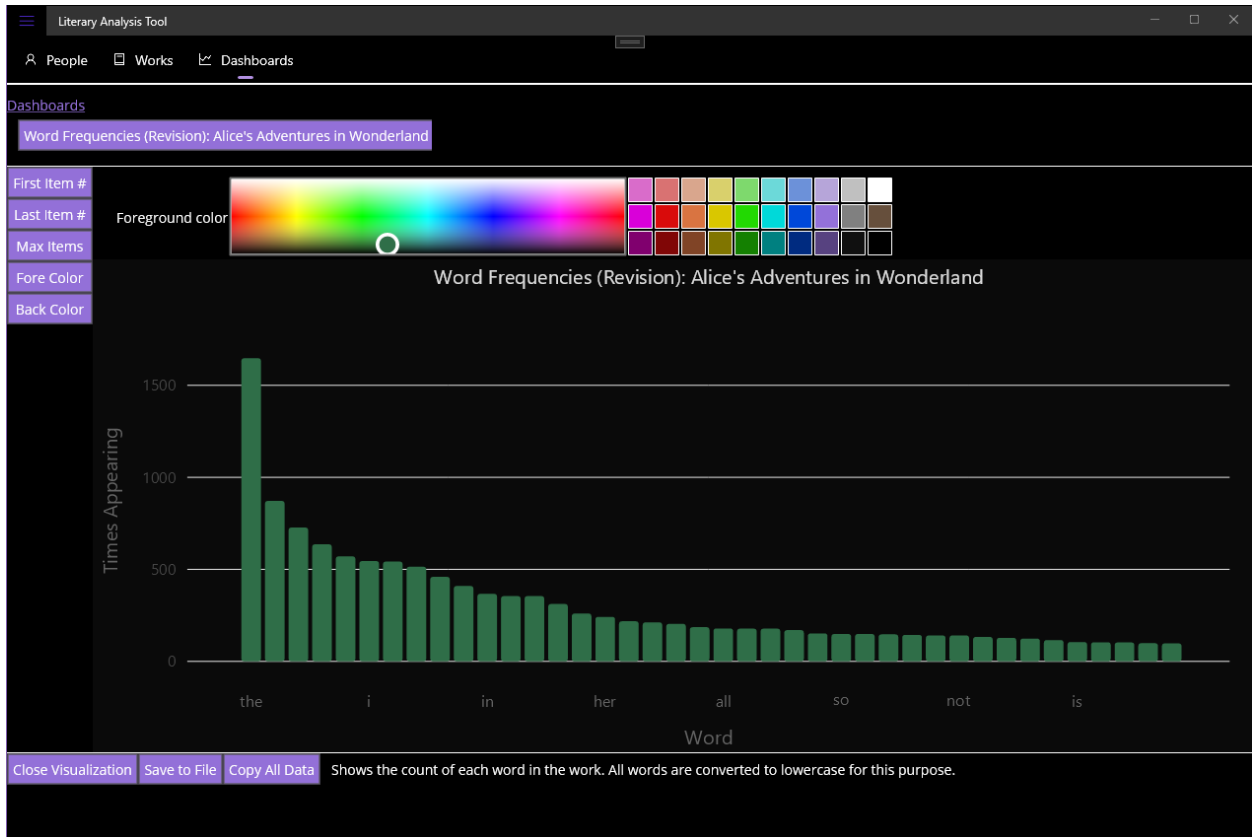


Figure 58 User Interface - User Filter - Foreground Color

The user is able to select any color they want for the foreground color in some charts. In other charts, the user can select a color and the nearest match on a predefined color palette will be used. Selections can be made from the color picker control on the left, or by clicking on any one of the predefined color boxes on the right.

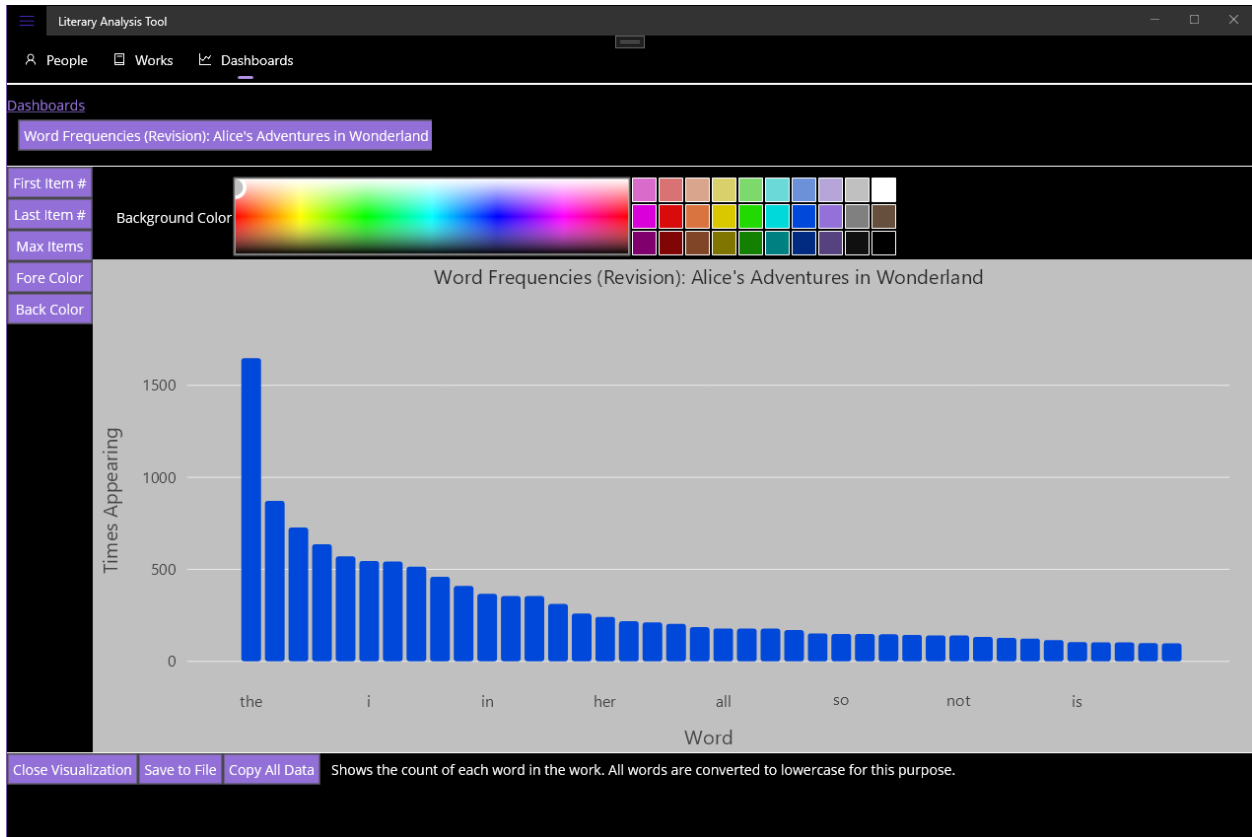


Figure 59 User Interface - User Filter - Background Color

The user is able to select the background color of a chart just as they are able to select the foreground color of a chart.

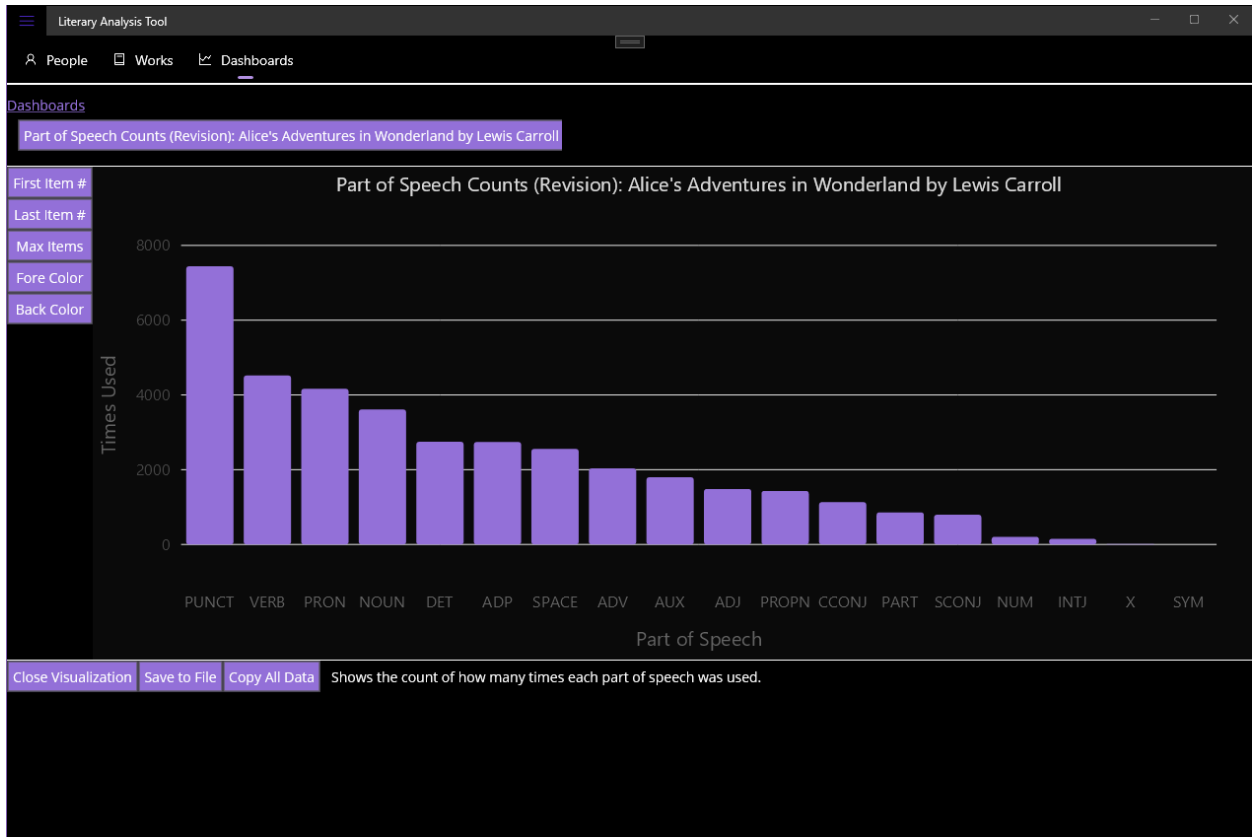


Figure 60 User Interface - Visualization – Part of Speech Frequencies

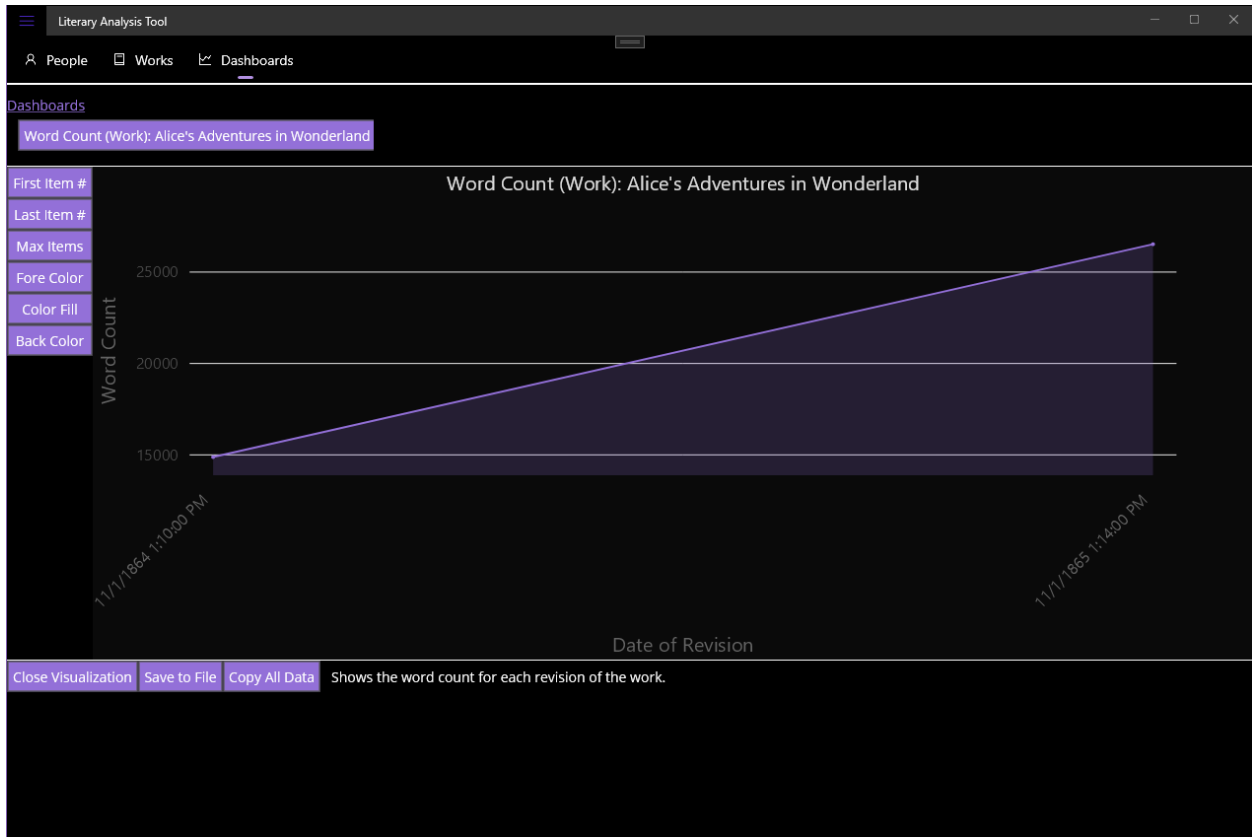


Figure 61 User Interface - Visualization - Word Count Total shown by Revision Date

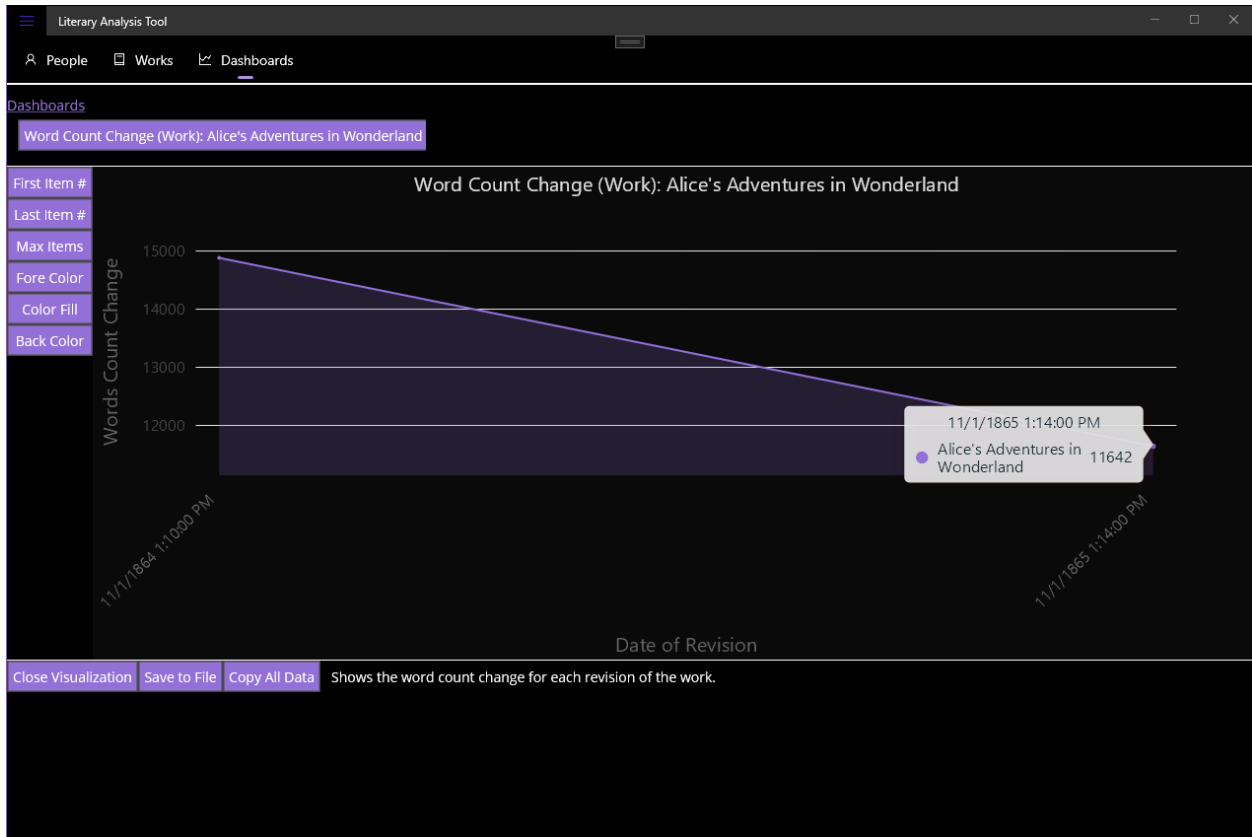


Figure 62 User Interface - Visualization - Word Count Change shown by Revision Date

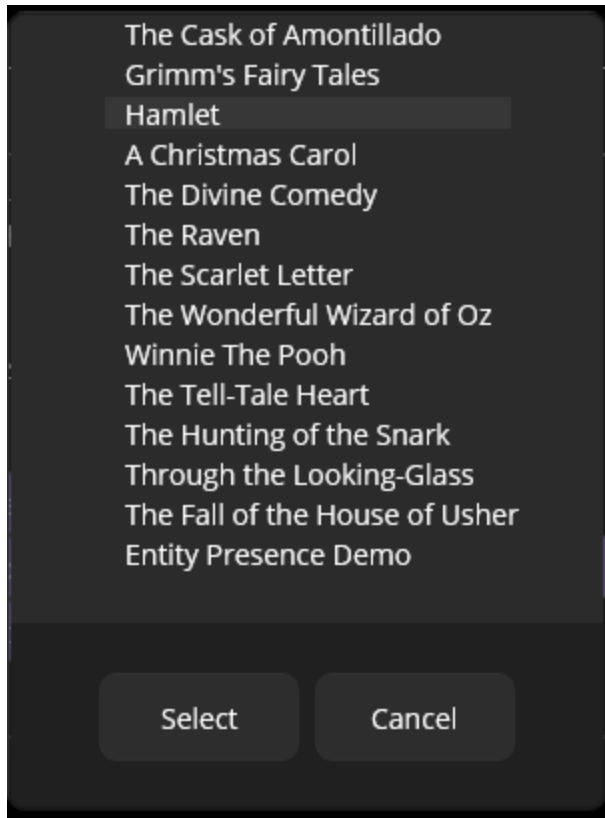


Figure 63 User Interface – Part of Speech Usage Comparison Popup
When the user selects the Part of Speech Usage visualization on a work, they must select a work to compare to.

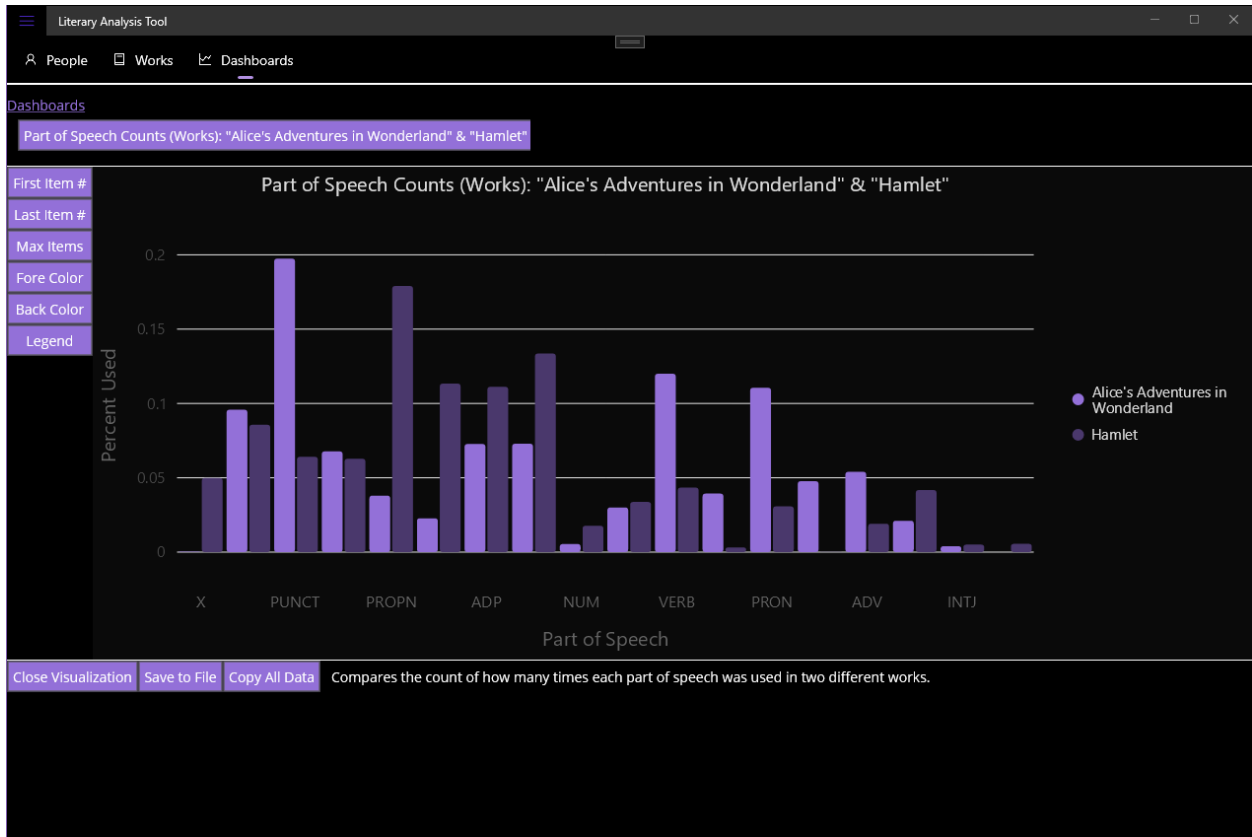


Figure 64 User Interface - Visualization – Part of Speech Comparison Between Two Different Works.

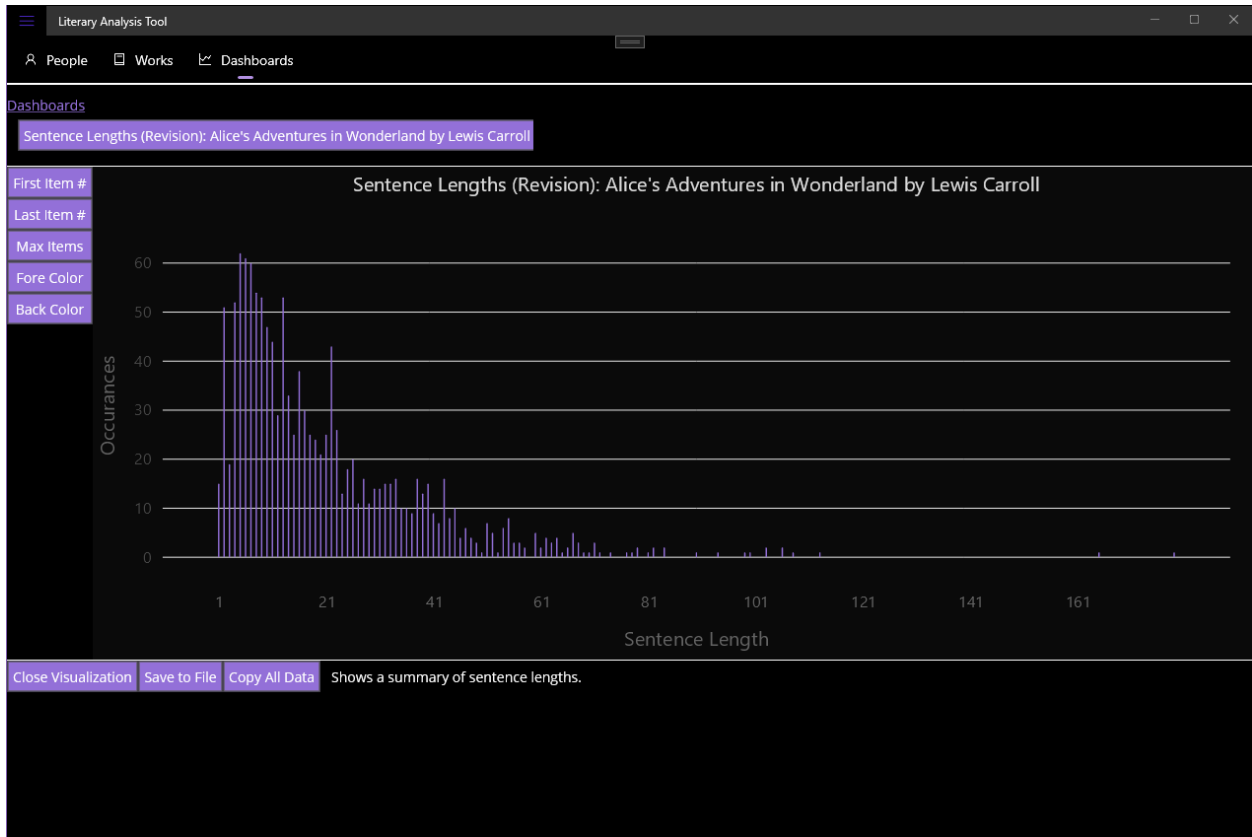


Figure 65 User Interface - Visualization – Sentence Lengths

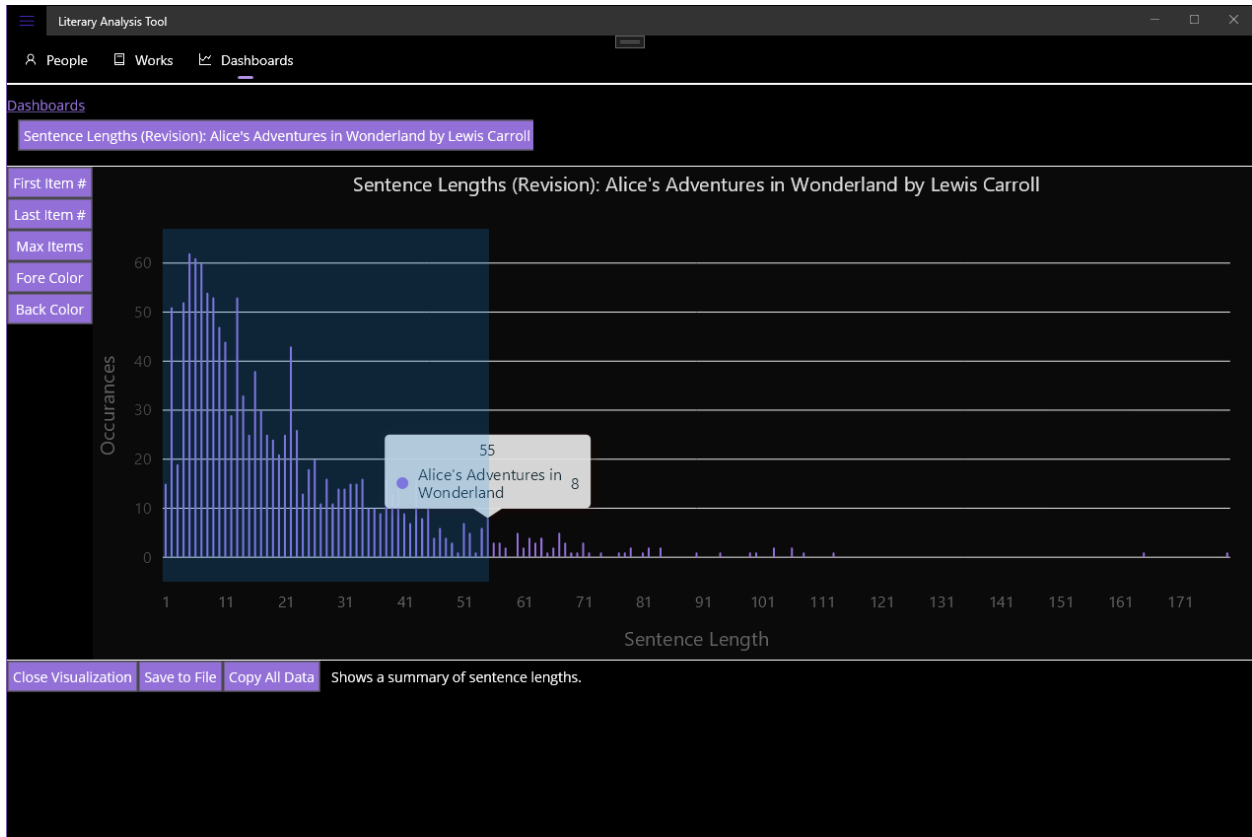


Figure 66 User Interface – Chart Interactivity – Zoom Window (Pre Zooming)

The user can right click on a chart and drag a window. When the user releases the mouse, the chart will be zoomed in to the selected window.

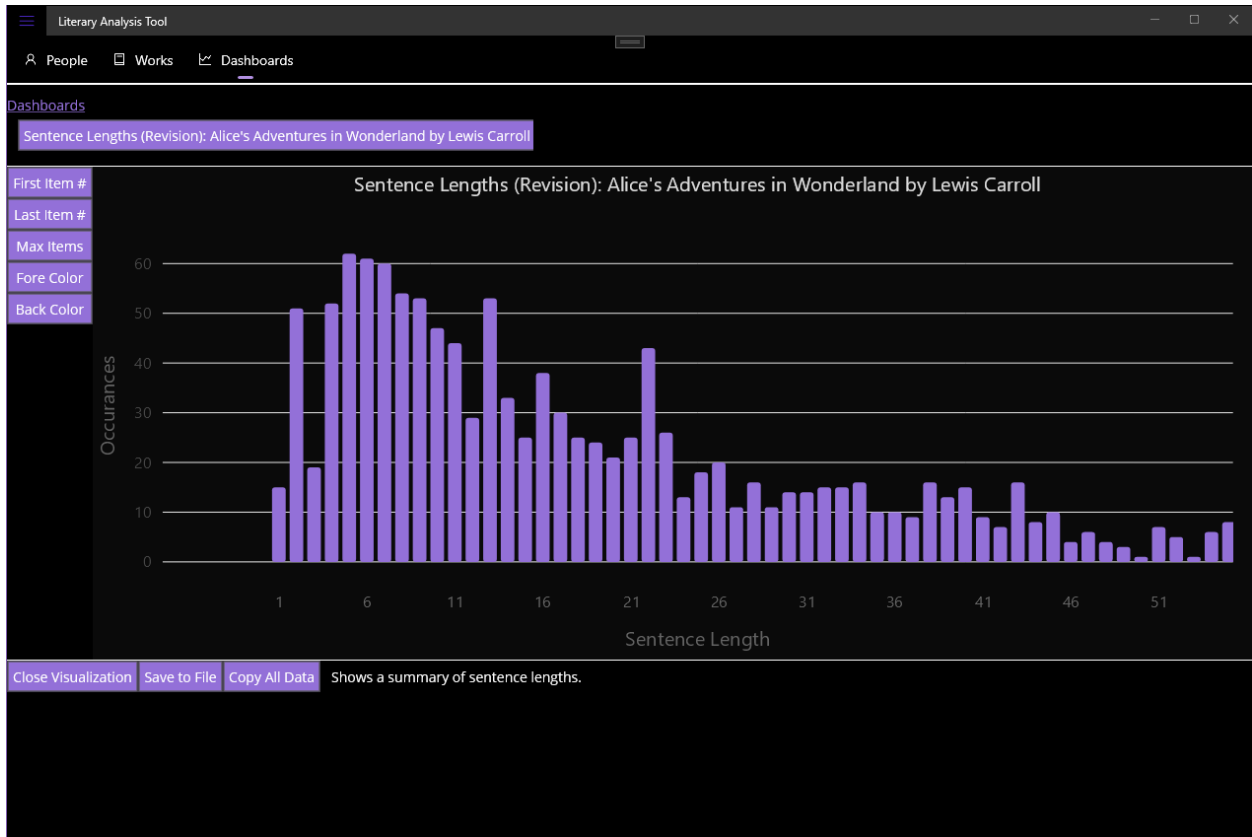


Figure 67 User Interface – Chart Interactivity – Zoom Window (Post Zooming)

This is the same visualization shown in Figure 66, except now the chart has been zoomed into the window selected by the user.

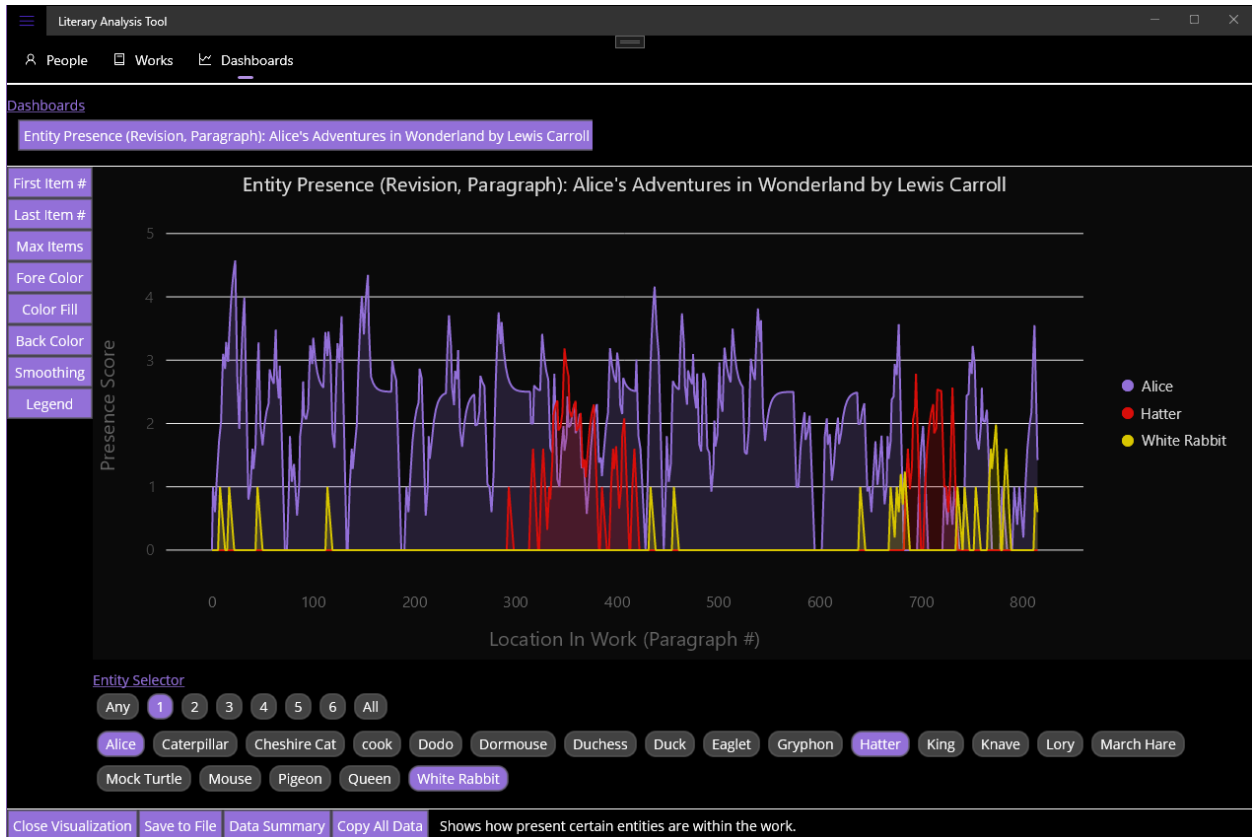


Figure 68 User Interface – Visualization – Entity Presence Scores (Single Entities)

The user is able to select multiple data series to display. The entity select is set to show single entities as denoted by the selected number 1. The first color used will be as close to the foreground color the user has chosen. Further colors will programmatically be determined. A color palette is created based on the number of data series that needs to be displayed. Then the colors are sorted by hue, value, then saturation using the HSV color space. The list of colors is made into a circularly linked list, and colors are chosen to be equidistant from each other.

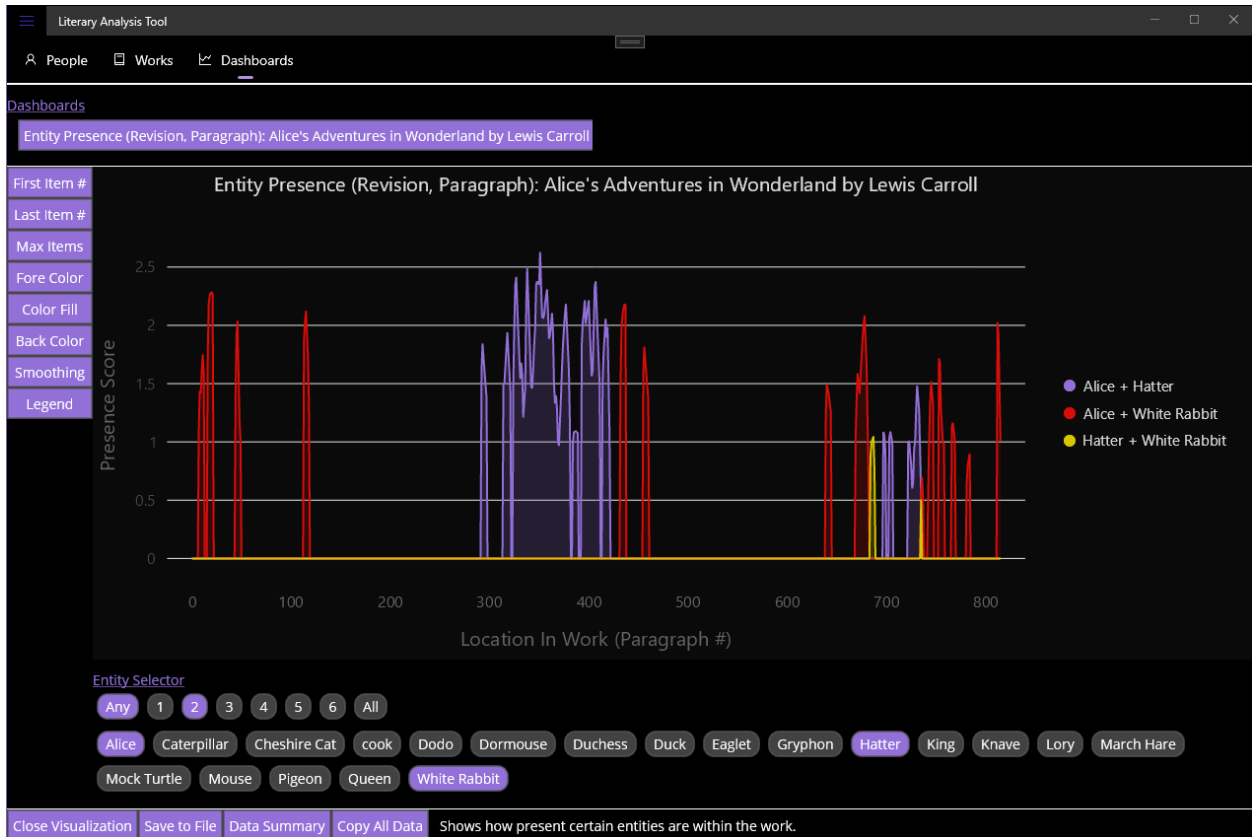


Figure 69 User Interface – Visualization – Entity Presence Scores (Groups with Any Modifier)
 This is the same visualization shown in Figure 68, except instead of showing single entities, the user has chosen to show groups of two that contain the any selected entity. This is denoted by the selected number 2 and the selected rule “Any.”

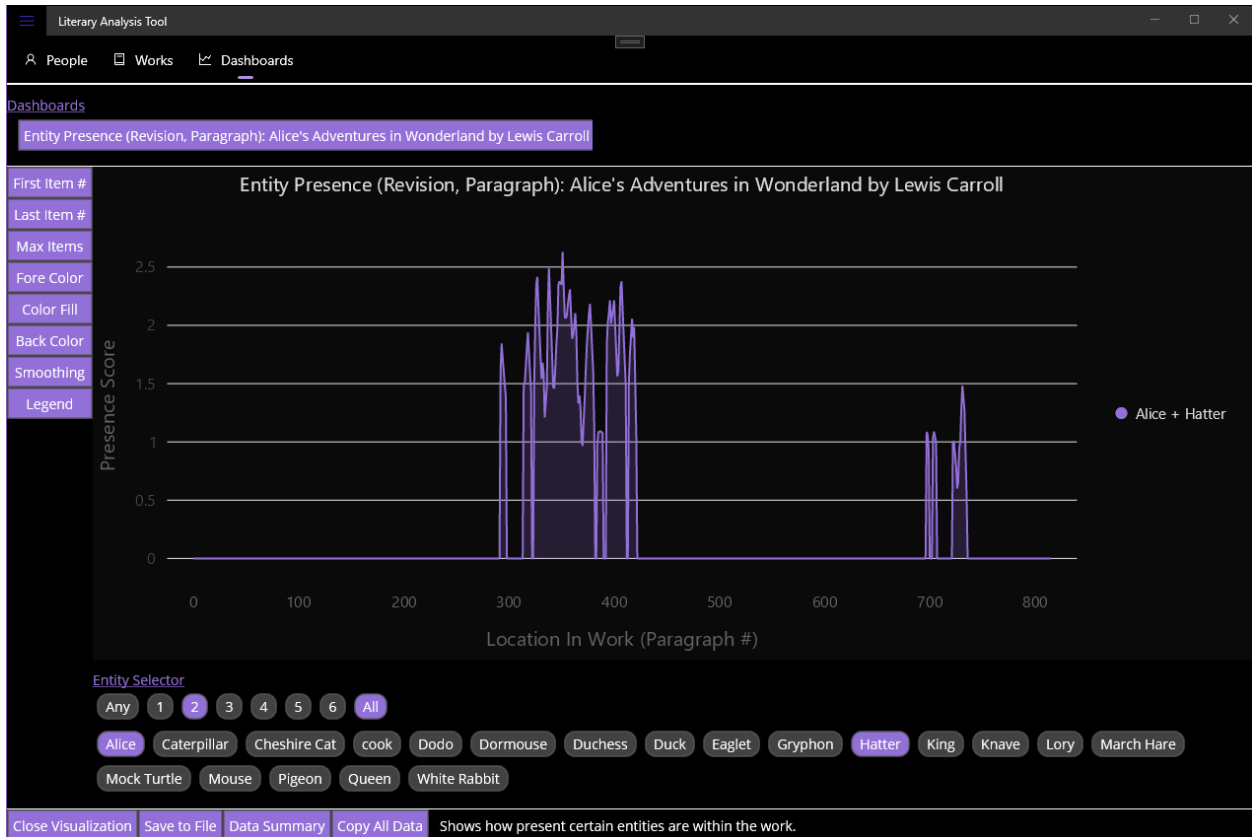


Figure 70 User Interface – Visualization – Entity Presence Scores (Groups with All Modifier)
 This is the same visualization shown in Figure 68, except instead of showing single entities, the user has chosen to show groups of two that contain all of the selected entities. This is denoted by the selected number 2 and the selected rule “All.”

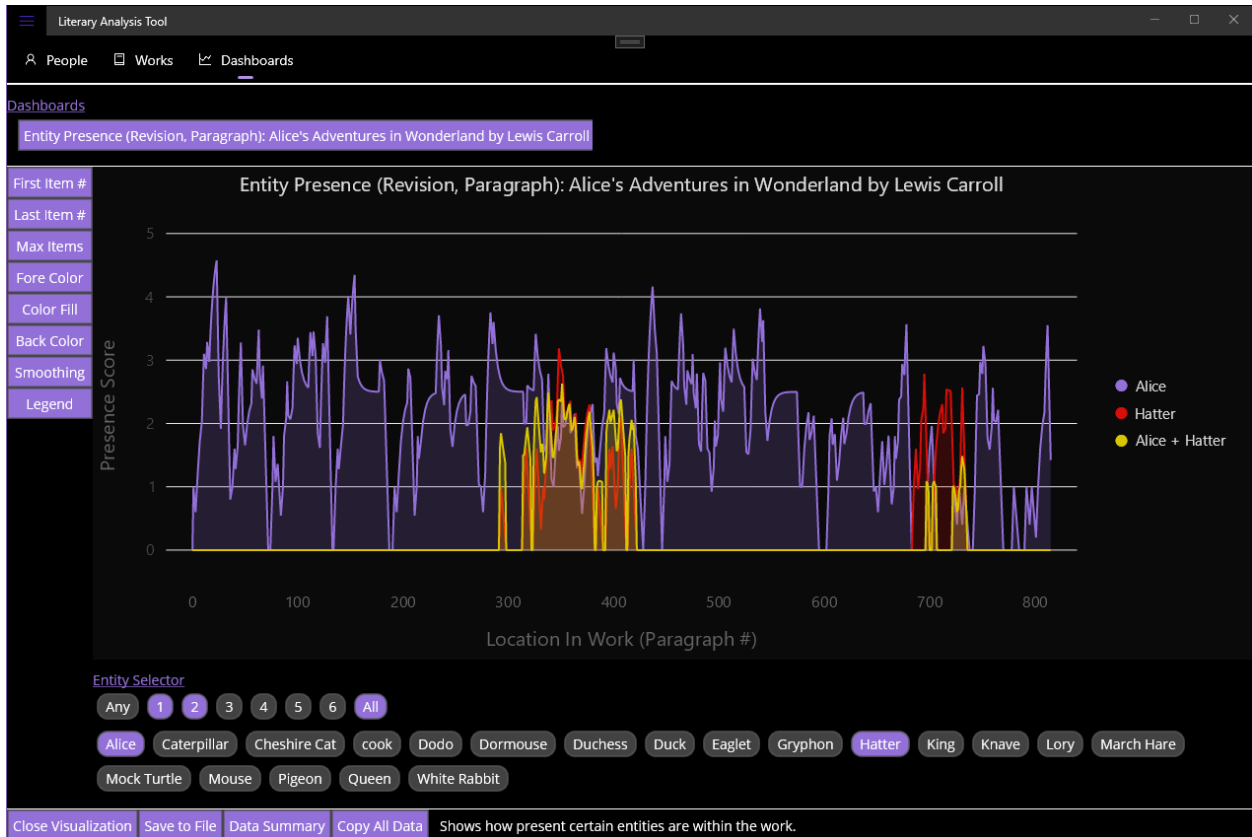


Figure 71 User Interface – Visualization – Entity Presence Scores (Single Entities and Groups with All Modifier)
 This is the same visualization shown in Figure 68, except instead of just showing single entities, the user has also chosen to show groups of two that contain all of the selected entities. This is denoted by the selected numbers 1 and 2 and the selected rule “All.”

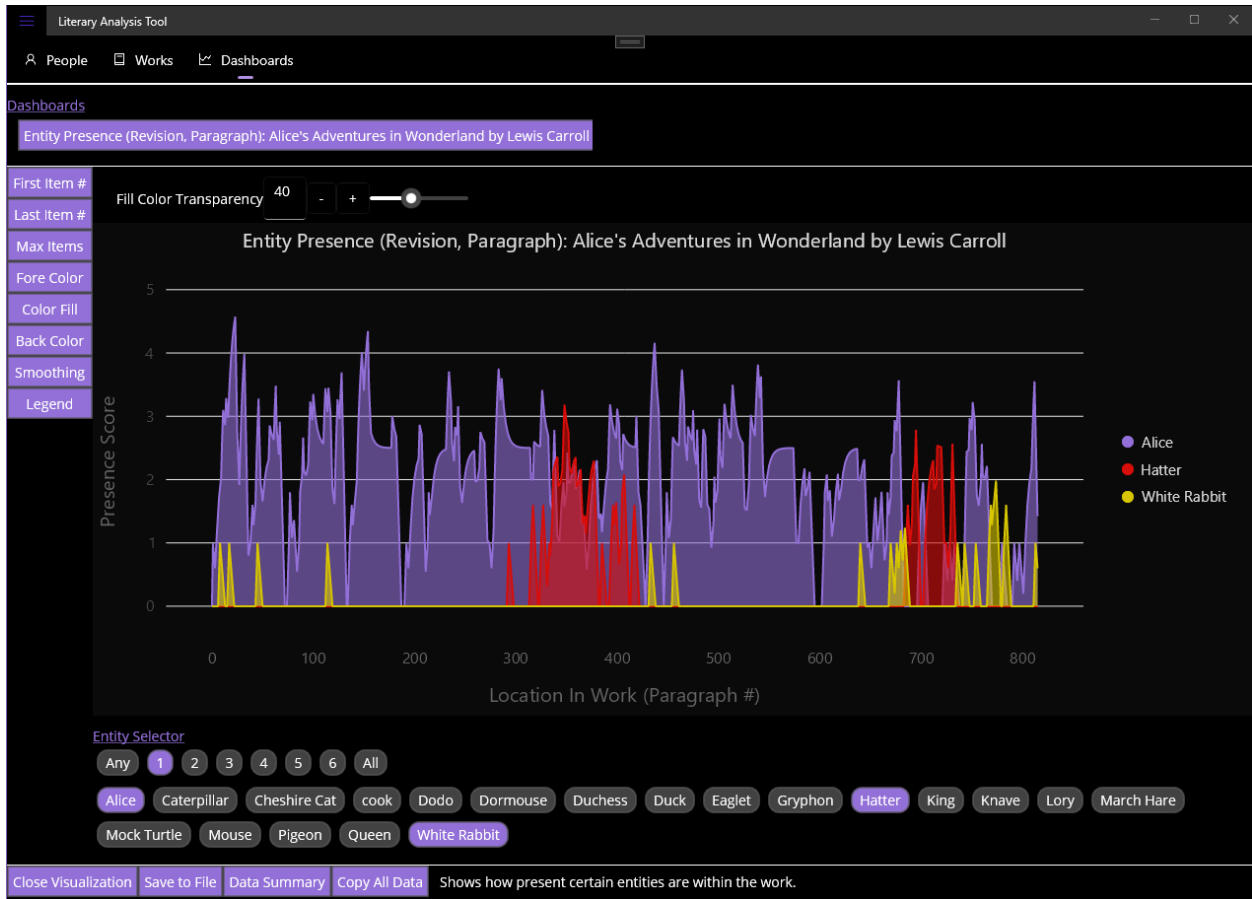


Figure 72 User Interface – User Filter – Fill Color (Lower Transparency)

This is the same visualization shown in Figure 68, except the user has changed the value of the Fill Color filter so the area under the lines is less transparent.



Figure 73 User Interface – User Filter – Fill Color (Higher Transparency)

This is the same visualization shown in Figure 68, except the user has changed the value of the Fill Color filter so the area under the lines is more (100%) transparent.

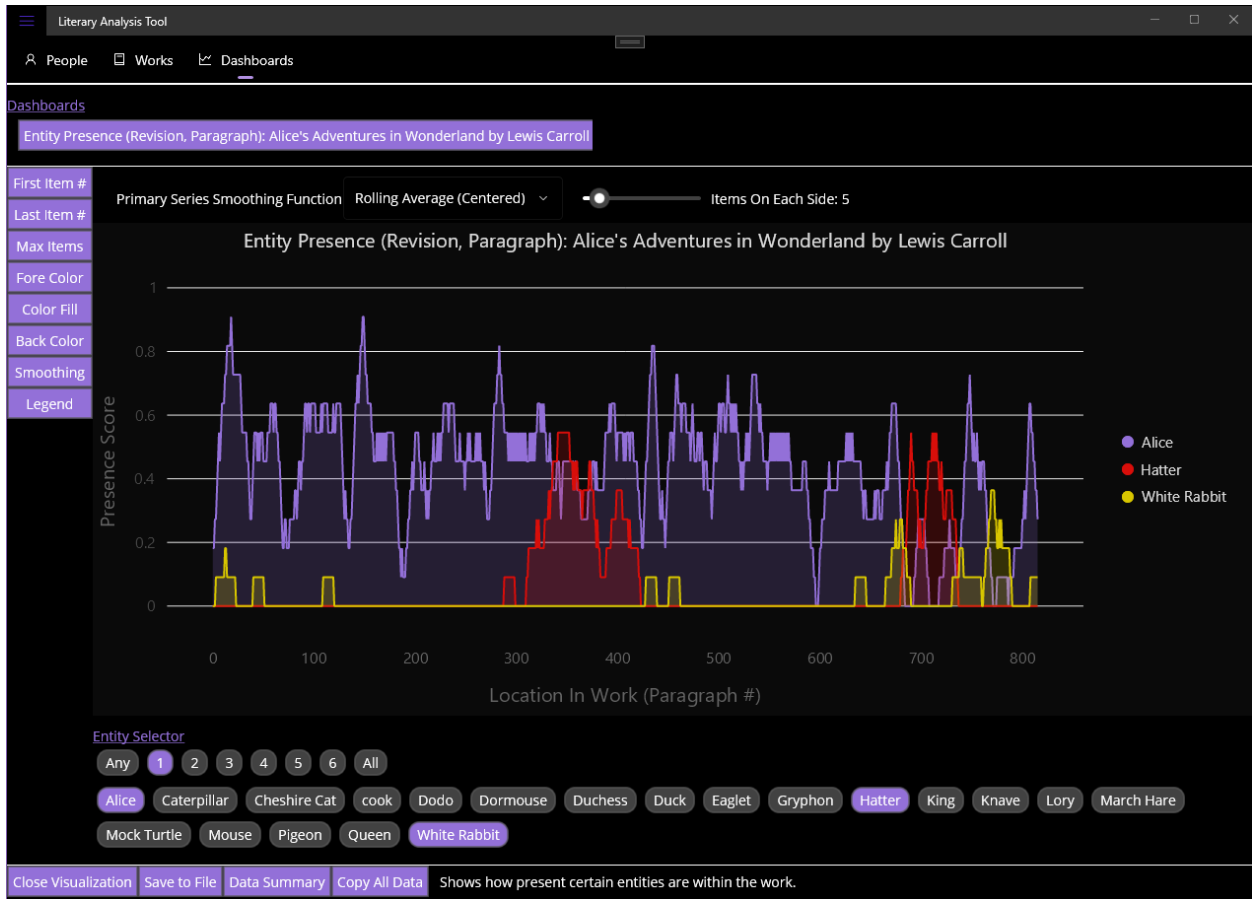


Figure 74 User Interface – User Filter – Smoothing Function Selection and Parameters

This is the same visualization shown in Figure 68, except the user has changed the value of the smoothing function to be a Rolling Average (Centered) instead of the default Lookaround smoothing function. The user can also change the parameter of the function by using the slider control.

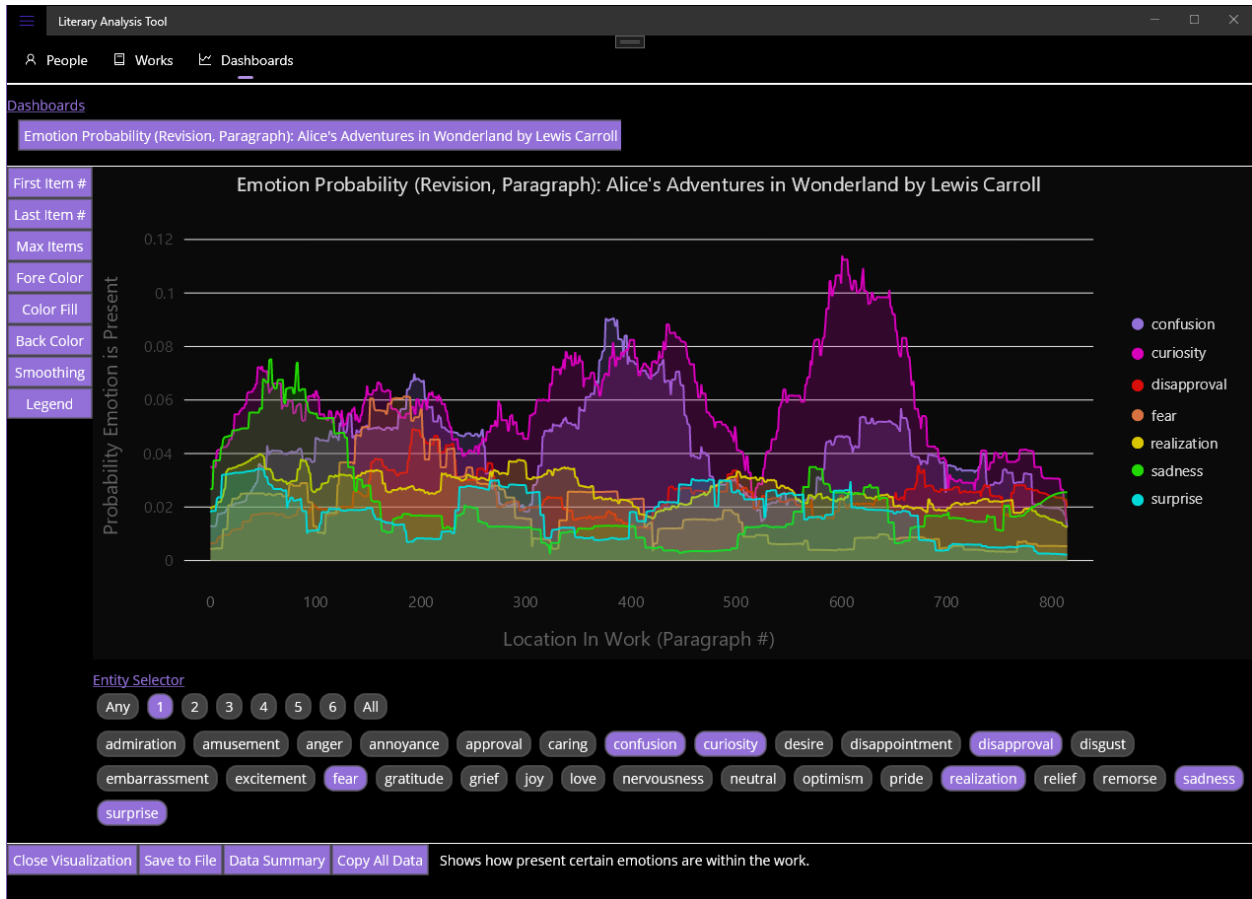


Figure 75 User Interface – Visualization – Emotion Probability Scores (Single Emotions)

The user has selected multiple emotions to display and has changed the smoothing function to the centered rolling average with a parameter value of 40.

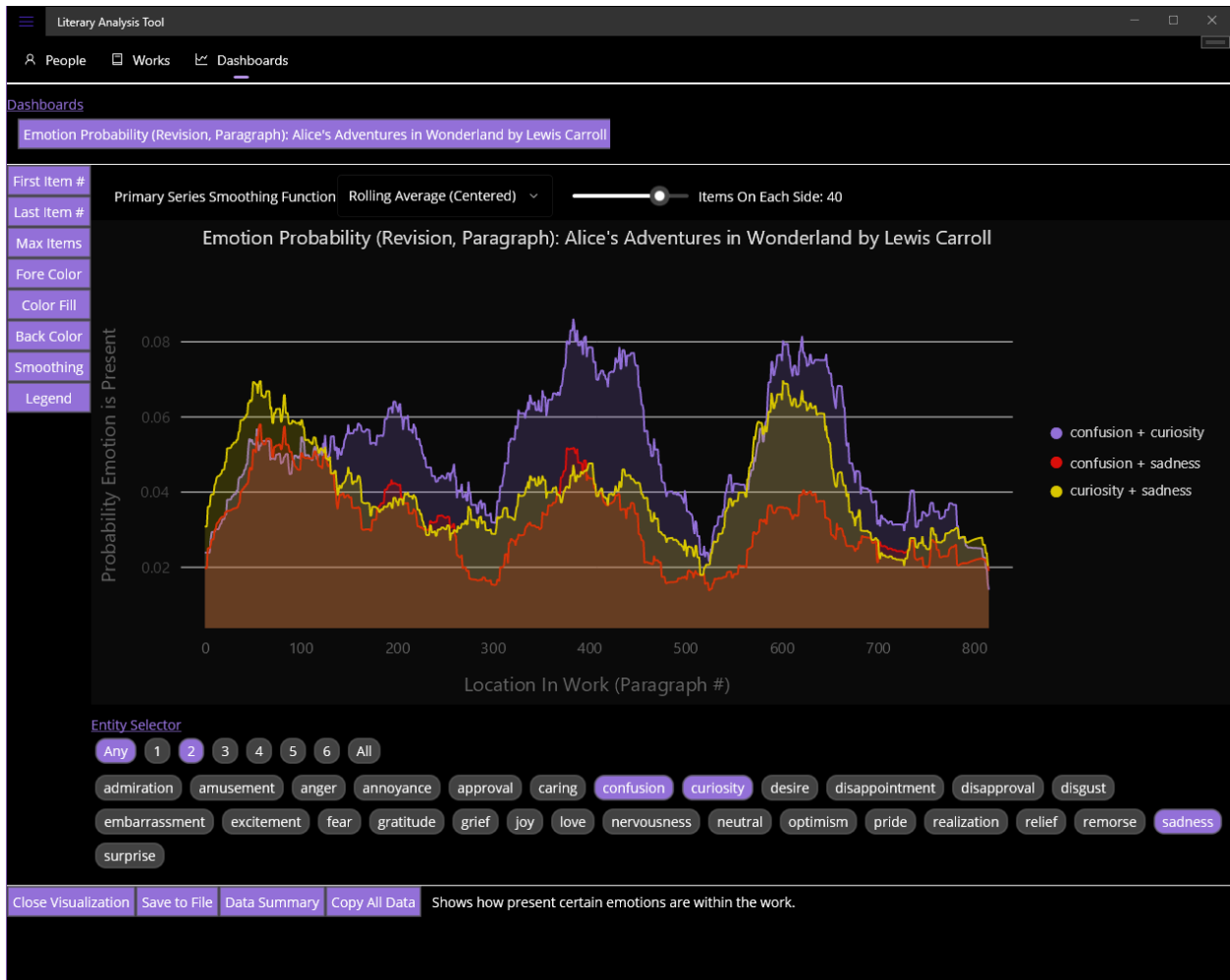


Figure 76 User Interface – Visualization – Emotion Probability Scores (Groups with Any Modifier)
 This is the same visualization shown in Figure 75, except the user has chosen to display combinations of 2 emotions, the same way they can display groups of entity presence scores.



Figure 77 User Interface – Visualization – Entity Tonality (Single Entity, Multiple Emotions)
 The user has chosen to display four different emotion probability scores combined with a single entity presence score.

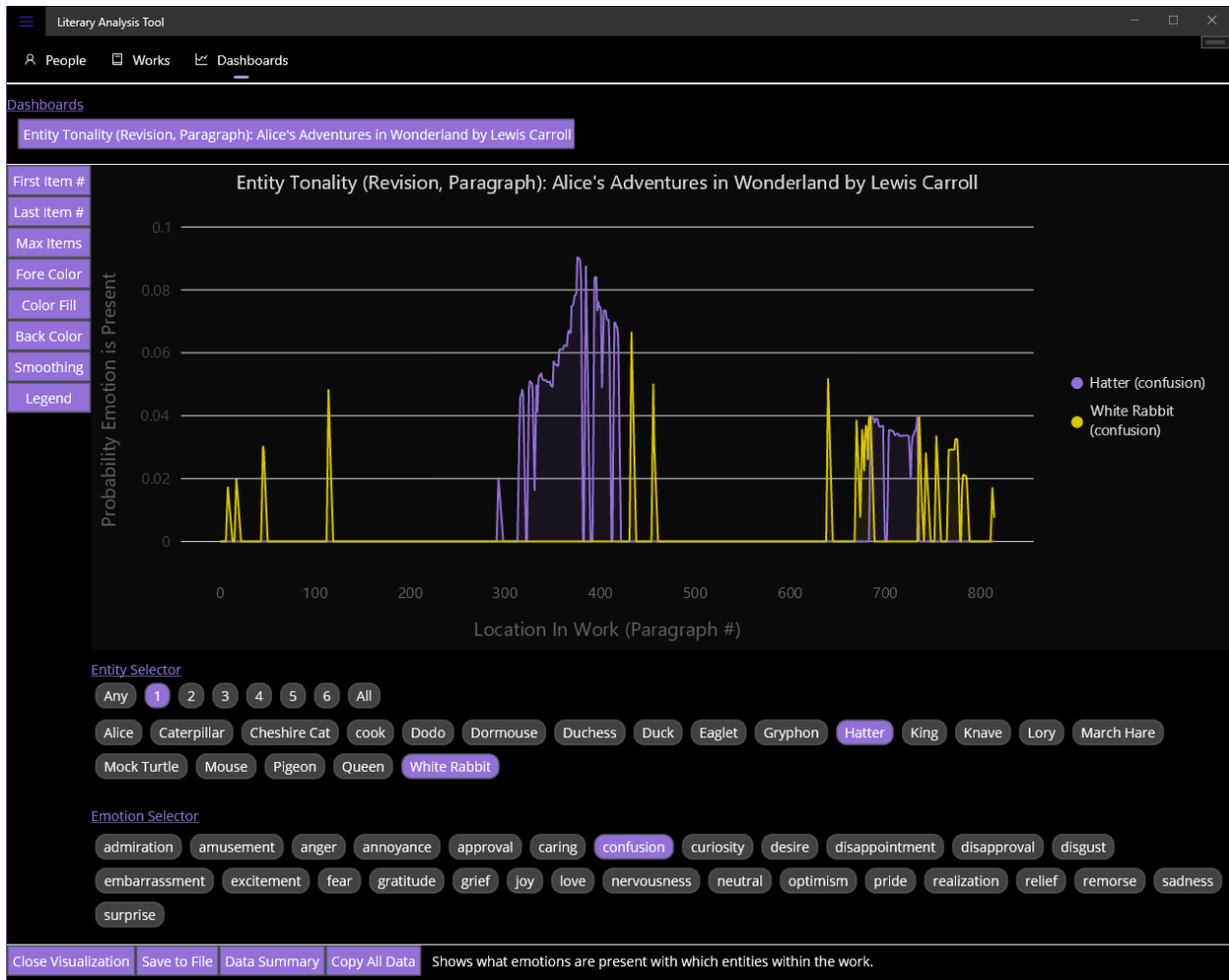


Figure 78 User Interface – Visualization – Entity Tonicity (Multiple Entities, Single Emotions)
 The user has chosen to display one emotion probability score combined with the presence score for two single entities.



Figure 79 User Interface – Visualization – Entity Tonality (Multiple Entities, Entity Grouping, Single Emotions)
 This is the same visualization shown in Figure 78, except the user has also decided to add in the grouping of both entities.

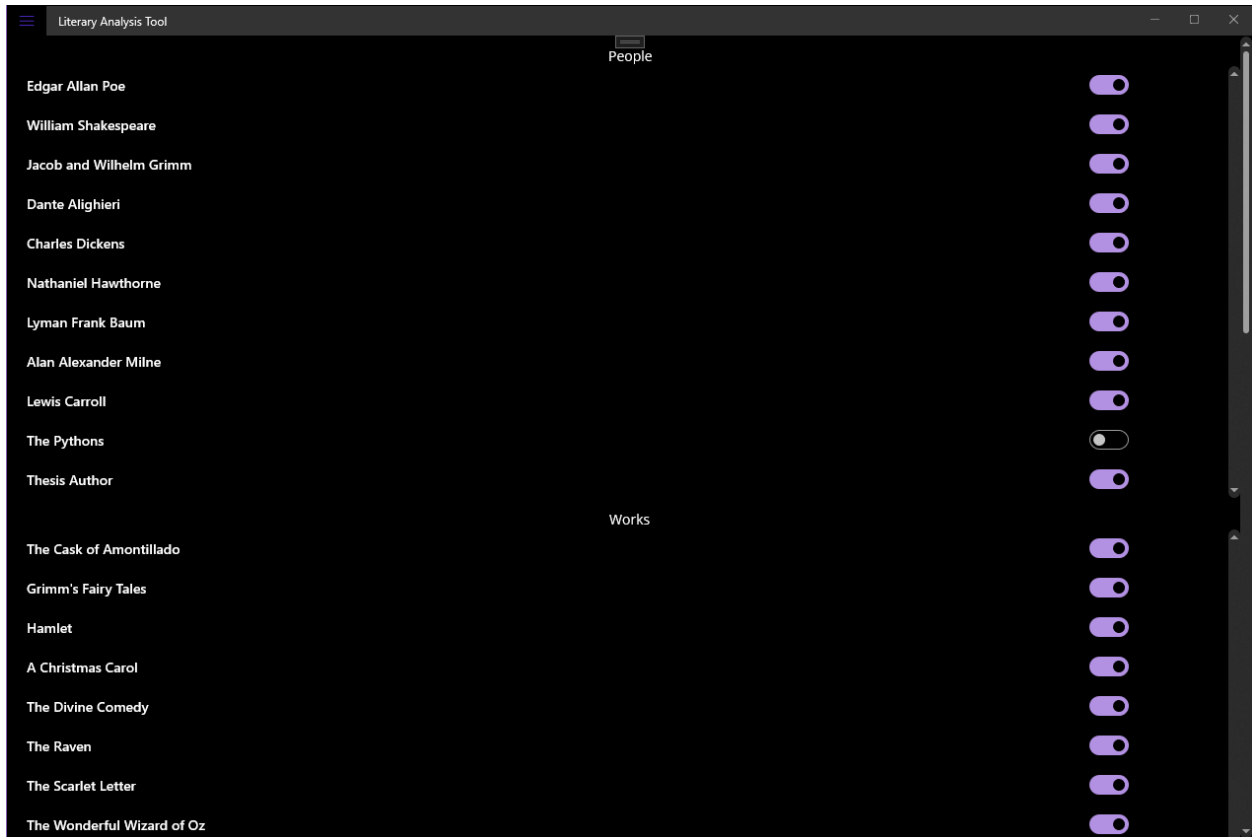


Figure 80 User Interface – Database Page (Loaded People and Works)

This page, which is accessed through the “hamburger” menu on the top left, allows the user to deselect specific people and works from being shown on the People and Works pages. With a lot of data in the database, selectively loading only the data a user needs could speed up the execution and responsiveness times of the program. The File Tracking and Unprocessed Revisions sections of this page are below the Works section.

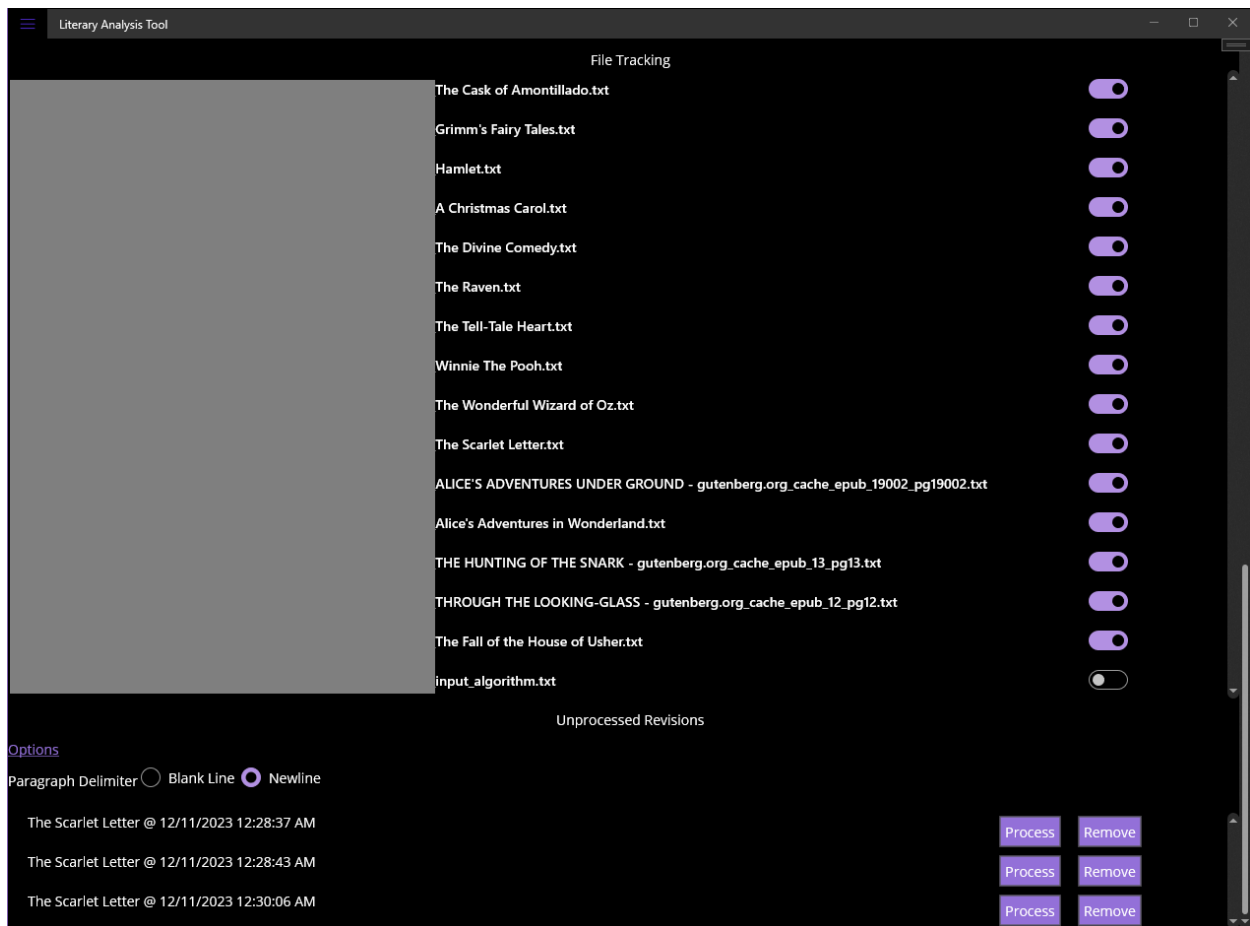


Figure 81 User Interface – Database Page (File Tracking and Unprocessed Revisions)

This portion of the database page allows the user to turn file tracking on or off for the background revision tracking service. Here, the file paths of the files have been redacted. Additionally, any unprocessed revisions are listed here in addition to being listed on the Works page. The user has the option of processing each individual unprocessed revision in any order that they want. They can also remove any unprocessed revision. (A user may want to do that if a file had been saved multiple times in quick succession, as was the case here.)

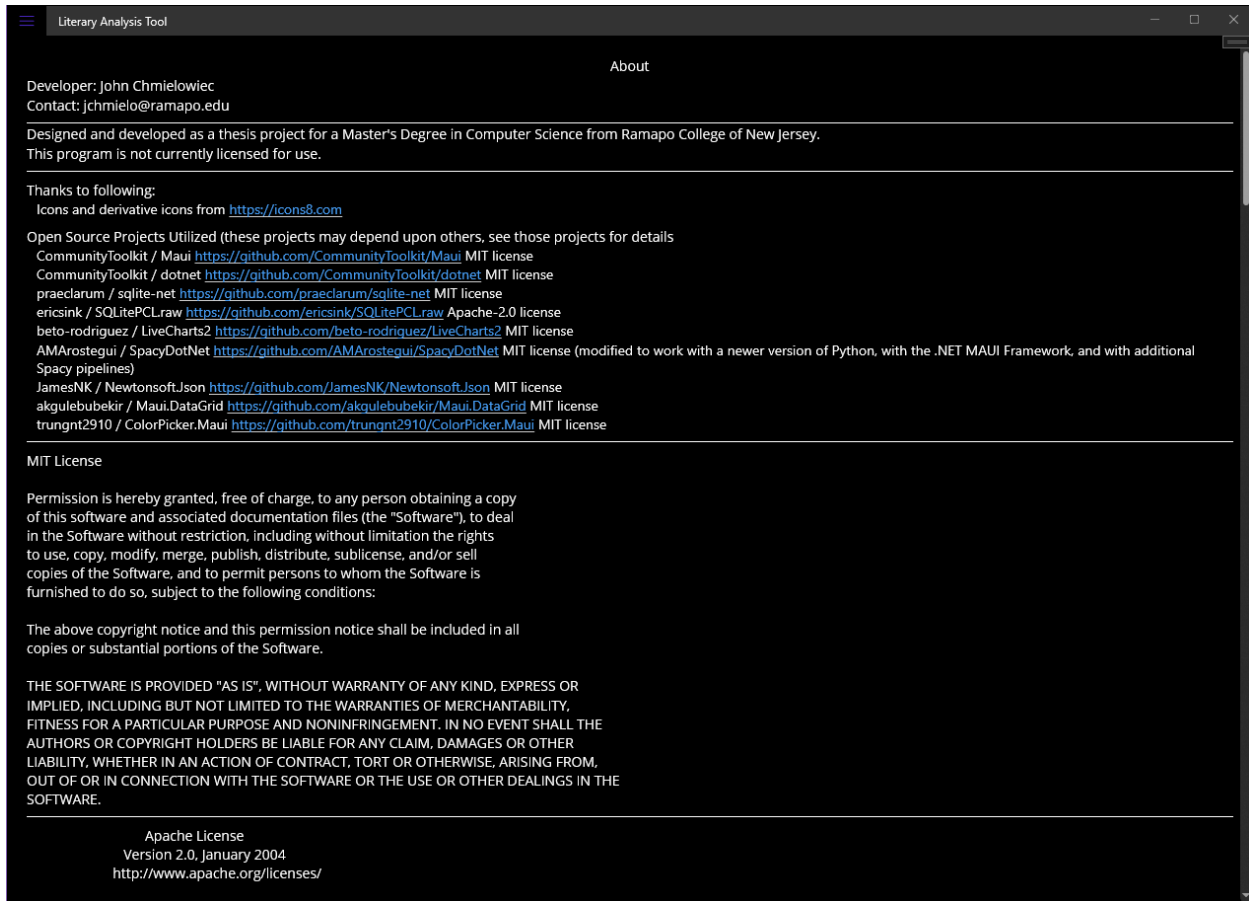


Figure 82 User Interface – About Page

This page, which is accessed through the “hamburger” menu on the top left, displays some information about the program to the user. All open-sourced projects are mentioned, along with links to their source code on github.com, and which license the code is used under. An acknowledgement to the icons8.com website is also placed here as a requirement for using some of their icons in the program. Finally, a copy of the MIT and Apache licenses are included here as all of the open-sourced projects use one of these two licenses.

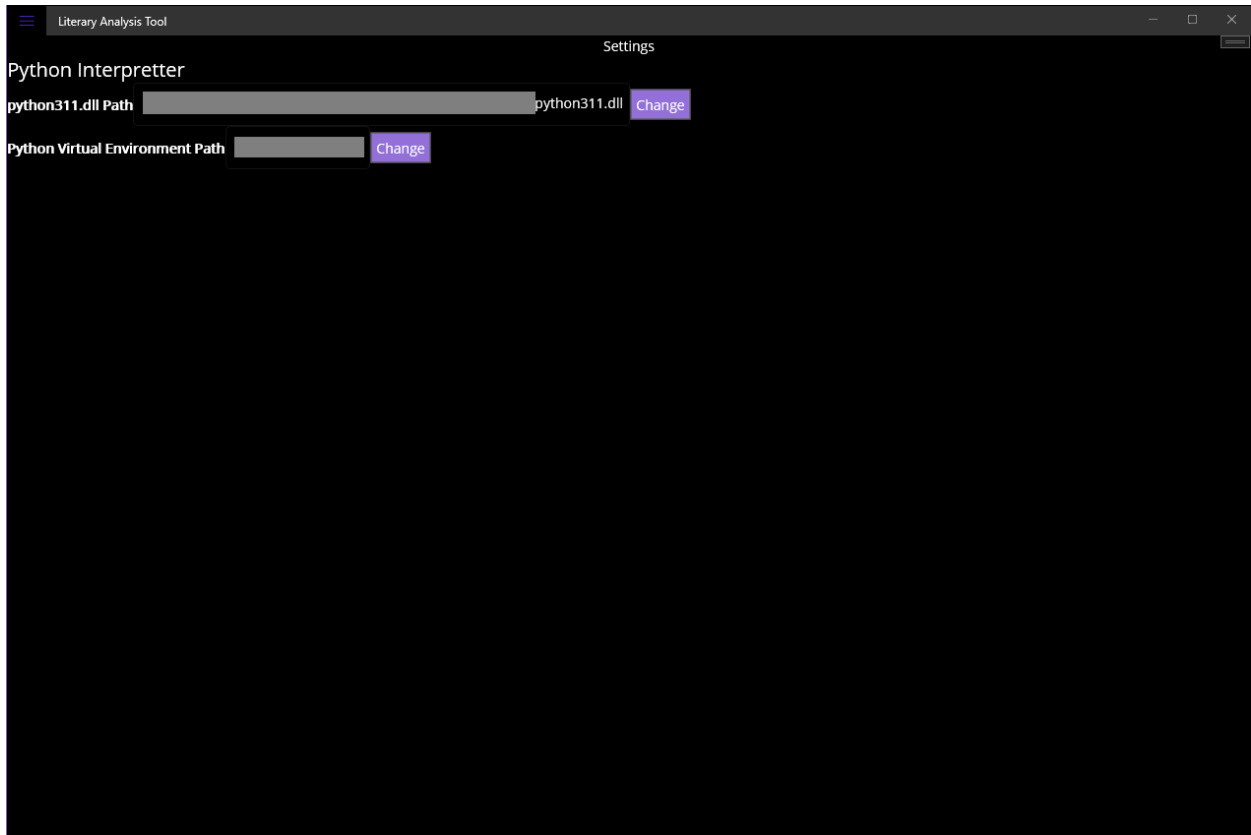


Figure 83 Figure 81 User Interface – Settings Page

The settings page is where the user saves the locations for the Python311.dll and the path to the Python virtual environment.