Scraping, Storing, Visualizing, and Exploring Data: Preparing Emergency Warning Apps for Text Classification

Timoteo Kelly

Department of Health Informatics, Nova Southeastern University

Dr. Dario Bonaretti

30 November 2022

Table of Contents

Introduction

The purpose of this paper is to detail the process taken for the collection, storage, processing, and analysis of Google Play Store review data using various tools such as Python and R. This task was undertaken to facilitate the research project, Theory of Effective Use in the design of Emergency Warning apps by analyzing reviews with text classification and machine learning.

Process

The working environment for the first stages was Visual Studio Code where I used JupyterNotebook with Python.

To begin, the initial data needed to be collected. That is, the reviews from the Google Play Store for the over one hundred identified Emergency Warning apps needed to be scraped and stored for later use. Surface data from Emergency Warning applications that fit the parameters from the Google Play Store was recorded in an excel spreadsheet. This data contained basic information such as application title, URL, application date et cetera. The object-oriented programming language, Python was used to read through the list of Emergency Warning applications on the spreadsheet and iterated over each application.

This was done by building a web scraper using Python with Google Play Scraper API, and OpenPyxl packages. The reviews were then stored in both MongoDB an object-oriented database and SQLite3 a built-in relational database package coming standard for Python3. Other native Python packages used were Datetime for manipulating dates and time, Pprint a module for printing data structures and Pandas a tool for data structure and manipulation.

Once the list of identified Emergency Warning applications were called into the Python shell, they were loaded into a Pandas data frame, like an excel spreadsheet to be able to view the data

easily. This was to ensure that all the data was entered correctly. At this step, it's important to make sure that the applications added to the list actually contain reviews or are still functional otherwise it will cause the loop to stop.

The next step was to setup the database that would be used to store the reviews. Several options were experimented with. First, we experimented with using MongoDB an objected oriented software as a service database that is easily interactable and callable into Python and can be queried within using a NoSQL based language. It stores data as documents similar to a json file. Other services Mongo provides includes easy data sharing with data clusters and repositories, data lakes, data analysis and data visualizations all within their web-based interface. Their desktop GUI provides many of those features as well as makes it easy to view your data stored in the database. It is easy to clean and process the data within the Mongo database and to export data to CSV or a json file. The drawback to MongoDB is that there are limitations to how much data can be stored without a purchase and an account setup is required to begin being able to store data or create a database. One can store up to 950,000 documents before beginning to hit the storage threshold.

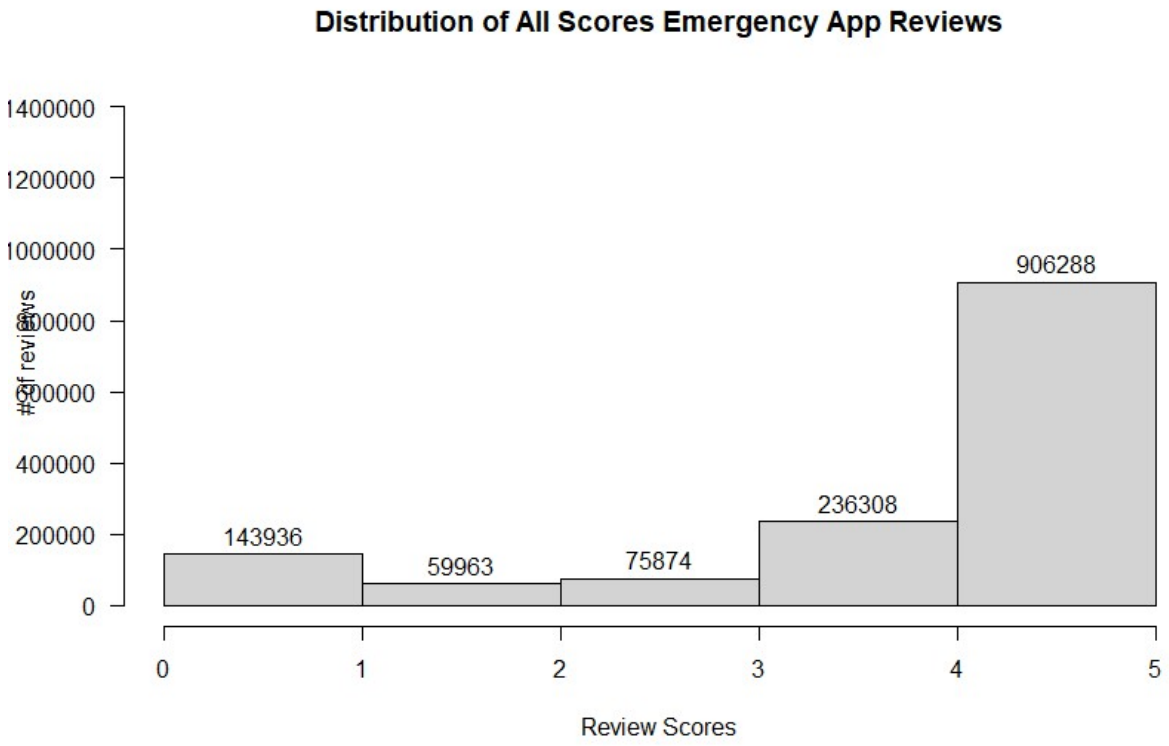
The other alternative that was to use the native Python package, SQLite3. SQLite3 is a relational database package embedded in Python library. It "provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language." (Python Software Foundation, n.d.). It can store up to 281 terabytes of data and is mostly limited by the system. One can use a function in Python from the SQlite3 package that allows one to execute SQL script within Python.

SQLite3 can be used to create relational databases that are stored on the disk, but one of the additional benefits offered by SQLite3 and that was utilized during this project was the creation of in-memory databases. In-memory databases are excellent for creating temporary databases that are only stored on the RAM, making storing and querying data faster however, as the name implies, they are only temporary and as soon as the session is closed the data will be deleted. The use of an in-memory database was perfect for this project given that the data did not need to be stored in database, but a json file. This file would later then be imported to RStudio for visualizations and analysis. The reason the data needed to be stored somewhere temporarily is because the loop in the script is one iteration of data being extracted from the Google Play page. The Google Play store page only shows up to 200 reviews per load, (if you look at the reviews, when you scroll to the bottom one must click to continue, this continue only shows 200 reviews at a time). Given this, at the end of each loop the script pauses 5 seconds to wait for additional reviews to load before continuing to scrape. Between each iteration the already scraped reviews must be stored somewhere otherwise the data will be overwritten. This was experimented on by placing reviews as a Python list object and in a Pandas data frame, but data was overwritten each time indicating it was necessary to utilize a database of some sort, however temporary.

Once all the Emergency Warning Google reviews were scraped and stored in the temporary SQL database, the data was loaded into a Pandas data frame to review and perform any data cleaning such as removing duplicates and empty rows. Once that was completed, the data was exported to both CSV and json file. The data was exported to both because while the CSV is not able to contain all of the data for this project, it allows one to view it quickly and easily. The json file is able to contain all of the data.

Exploratory Data Analysis

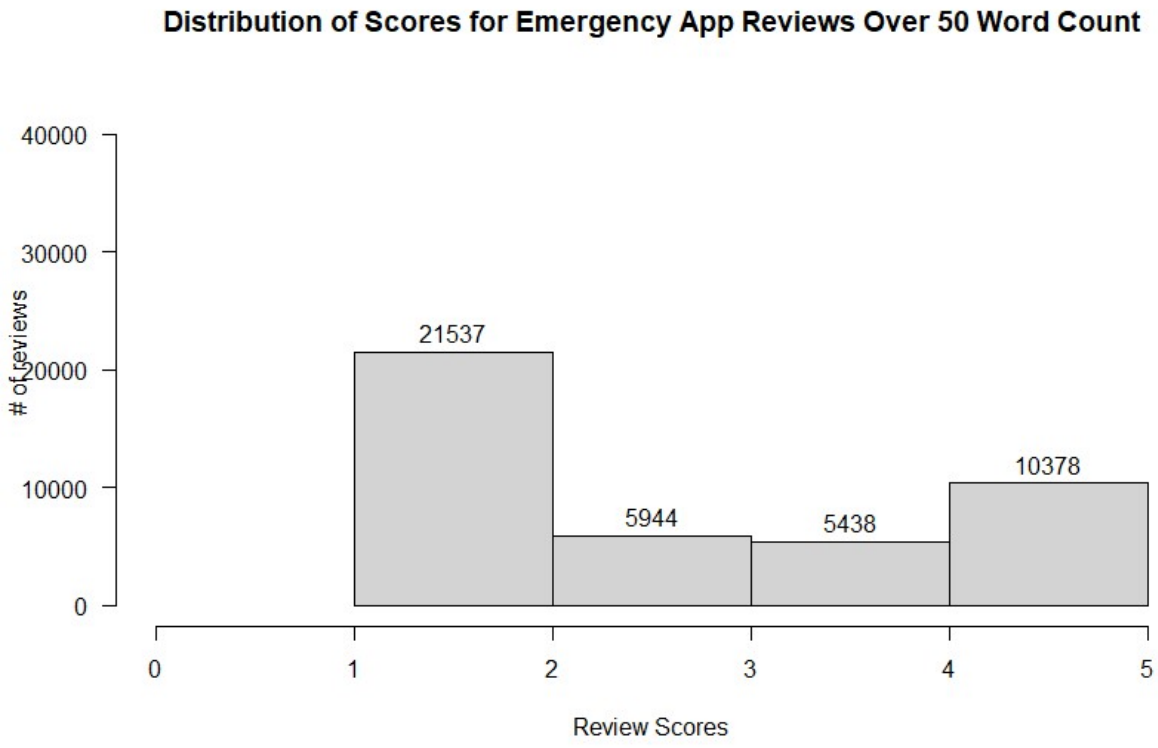
To perform exploratory data analysis the process was performed in RStudio. The json file containing the review data from earlier was imported into RStudio. Next, the packages gglpot2 from tidy verse was used to create a simple histogram for the distribution of the review scores.



As can be seen from the figure above, the largest group of reviews are 5, which is also the highest score. There are 906,288 reviews scored 5. The next highest group and highest rated is 4, with 236,308 reviews. The next largest group, but lowest rated is score 1, with 143,936 reviews. The next is score 3 with 75,874 reviews and score 2 at 59,963 reviews.

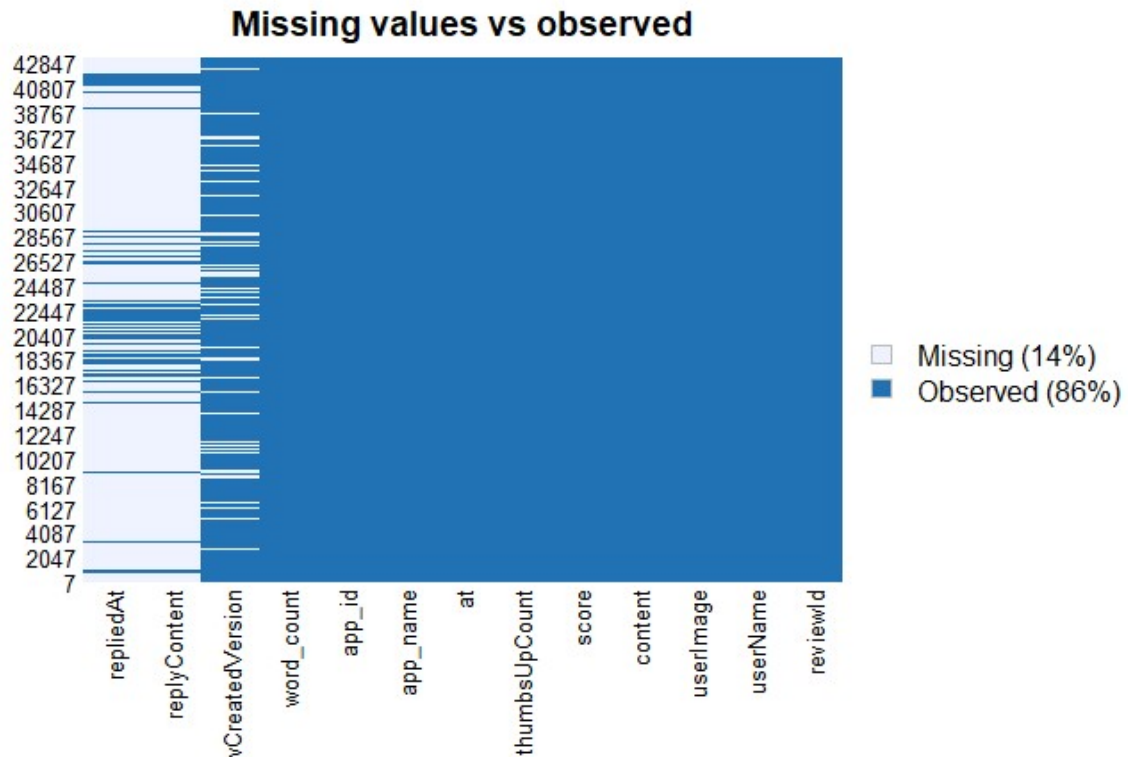
The next step used both ggplot2, here, and tidy verse packages to isolate reviews based on their word count. First, each word in each review needed to be counted, then a new column created

in the data frame displaying the word count. After this was executed, the following figure was created.



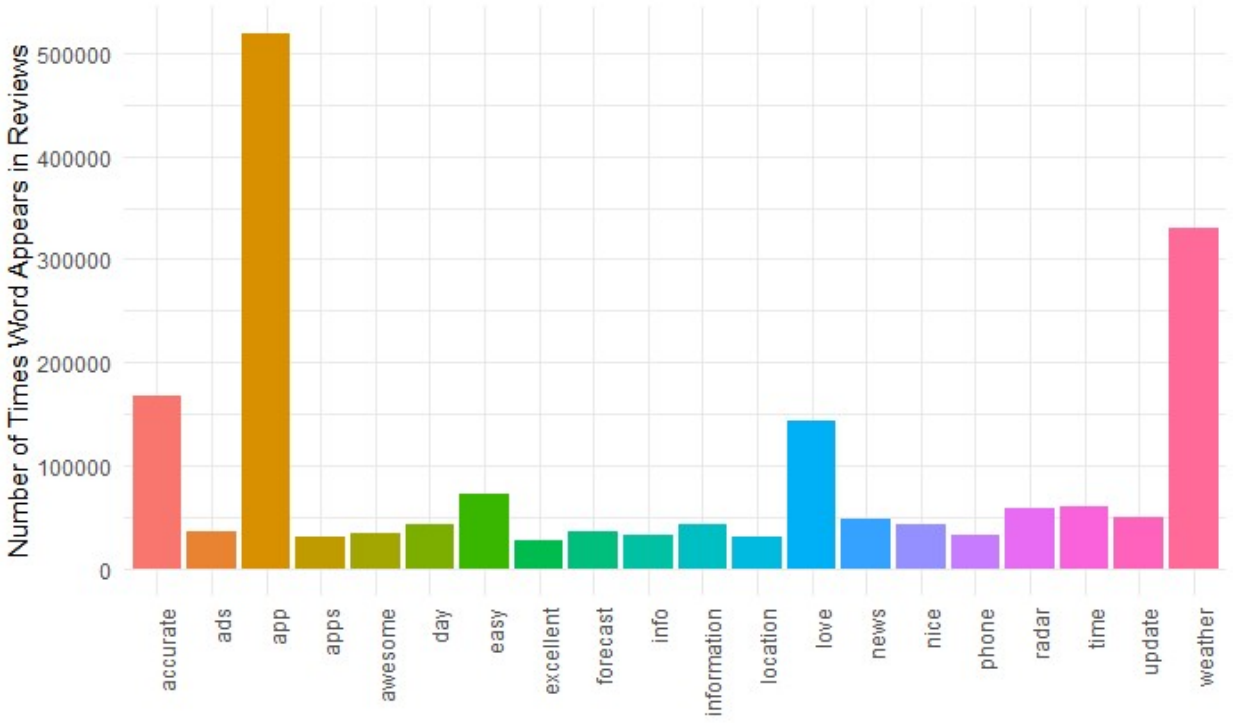
The figure above shows the distribution across review scores for all the reviews that were over a 50-word count. Based on this visualization, it appears as though the longer a review would be, the more likely it is to be negative based on the fact that 21,537 the largest group and more the double the next largest group sits at score 1.

The next step taken was using the package Amelia for creating a visual representation of any missing or NA rows in our dataset as shown below.



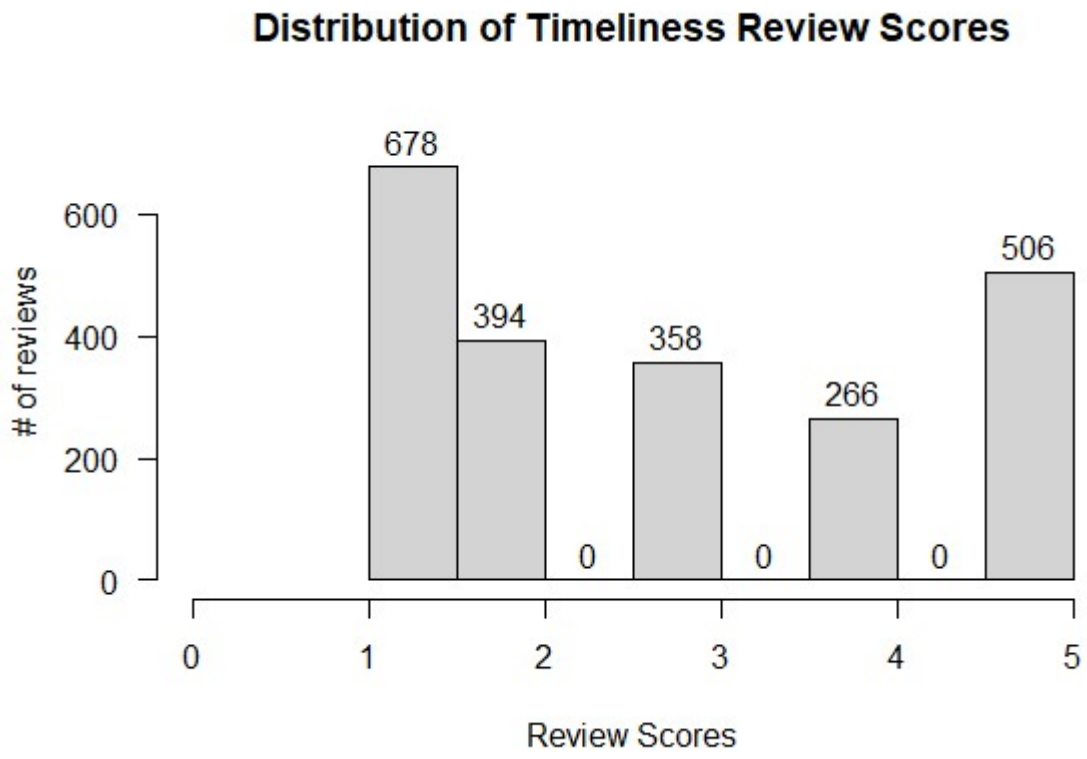
This gives us a visual representation of any data that may be missing. It also gives us the chance to determine if the data that is missing should be or not. From the figure above, it can be seen that it is the repliedAt, replyContent and CreatedVersion columns that have some missing data. This is to be expected, as not every review will have a reply and the repliedAt contains the date for the reply and the replyContent contains the text. CreatedVersion is not always available for every application in the Google Play store.

Next package used was tidytext to search for the most common used words in the review data and then plotted using ggplot2.



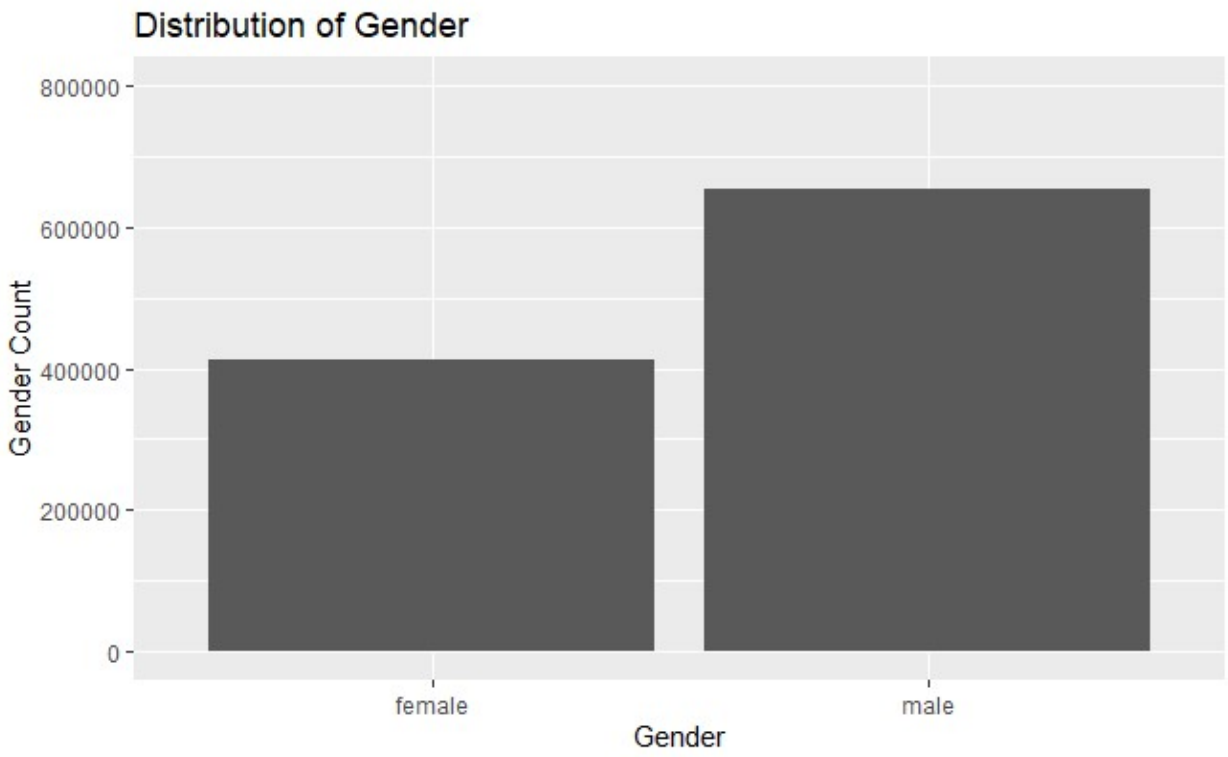
The above figure shows the top 20 most common words in the review data based on how many times they appear in the text.

Next, a search was performed based on a dictionary created for “time”. The words "time", "delay", "update", "limit", "current", "prompt", "alert", "rarely", "usually", "sometimes", "often", "never" ,"early", "late", "short", "fast", "slow", "brief", "old", "ancient", "elapse", "swift", "occasionally", "always", "minute", and “hour” were all added to the definition. This was done so that only reviews that contained these parameters would be pulled. It is reviews discussing their use of the application in reference to time.



As seen by the figure above, the distribution of the reviews based on the parameters of time as defined above, most of the reviews skew towards score 1 or negative.

Next, the genderdata package was used to determine gender of the review users. It is a very rough estimate, but the package works but utilizing data collected from Social Security from 1930 to 2012 based on gender and name to determine the gender of a name given. Since the review data contains usernames from Google accounts which typically use an individual's real name (but often do not), it was not a bad set of data to try this tool on. Based on running the data through this tool, the distribution of gender of the reviews is shown below.



Based on the Social Security data estimate, there are more males leaving reviews on Emergency application data than females.

Multiple regression analysis was also performed in Microsoft Excel using the add-on SigmaXL package to examine possible correlation between the review scores and review word count to see if what was being seen in the histogram would be supported by further analysis. Low R-Square of .04% showed low correlation between word count and score, although this with performed on reviews over 50 word count, had lower count been included in the sample data the correlation might have been higher.

Multiple Regression Model: score = (2.607) + (-0.00149) * Word.Count

Model Summary:	
R-Square	0.04%
R-Square Adjusted	0.00%
S (Root Mean Square Error)	1.566

Parameter Estimates:

Predictor Term	Coefficient	SE Coefficient	T	P	VIF	Tolerance
Constant	2.607	0.125926	20.703	0.0000		
Word.Count	-0.001485466	0.002456997	-0.604586	0.5456	1	1

Analysis of Variance for Model:

Source	DF	SS	MS	F	P
Model	1	0.896500	0.896500	0.365524	0.5456
Error	998	2447.7	2.453		
Lack of Fit	78	176.28	2.260	0.915362	0.6830
Pure Error	920	2271.5	2.469		
Total (Model + Error)	999	2448.6	2.451		

Durbin-Watson Test for Autocorrelation in Residuals:

DW Statistic	1.958
P-Value Positive Autocorrelation	0.2522
P-Value Negative Autocorrelation	0.7478

Continued Research

Although the exploratory data analysis has been completed, the text classification based on the parameters set by Trustworthiness, Timeliness and Situational Awareness has yet to be completed and is beyond the scope of this project. Data annotation based on the aforementioned dimensions has almost been completed utilizing the software as a service, MonkeyLearn. Currently, the predictive model can efficiently tag reviews based on Trustworthiness, Timeliness and Situational Awareness at up to 87% accuracy. More data needs to be annotated to improve the classification model to above 95%.

```
from monkeylearn import MonkeyLearn

ml = MonkeyLearn('1e733b6a2d02ef8f16b73a4dcefe7d6beb5d38ff')
data = ["it could be set up with a little bit more user friendly inter
model_id = 'cl_7cSmAWxP'
result = ml.classifiers.classify(model_id, data)
print(result.body)
```

✓ 0.3s Python

```
[{'text': 'it could be set up with a little bit more user friendly
interface and start with the current/ future radar instead of being on
default past radar bit im happy with it', 'external_id': None, 'error':
False, 'classifications': [{'tag_name': 'Currency', 'tag_id': 124243233,
'confidence': 0.812}]}]
```

The above figure shows a sample from our review data being run through our text classification model in Visual Studio Code. The predictive model determined that the review was discussing "the present state of the real world and its representative data structures" (Bonaretti & Fischer-Preßler, 2021, p. 8) which is the definition of Currency determined in *Timeliness, Trustworthiness, and Situational Awareness: Three Design Goals for Warning with Emergency Apps*. The level of accuracy for this prediction is at 81.2%. With more data being annotated, results should yield >95%. MonkeyLearn is proving to be an essential tool for researchers who neither have the time nor skill to annotate their training data or build and train a model. MonkeyLearn combines the process in a no or low code format yet is versatile enough to be called into Python as shown above and used with a programming language. One person could annotate, build, train, run their model and produce results within a day or two. A task that would have previously taken one person at least a week.

Conclusion

The process for scraping, storing, transforming, and analyzing data previously detailed is a streamlined way for researchers and analysts to collect data from multiple apps from the Google Play store. In addition, the advancements software as a service has made, has given cost effective solutions to smaller research teams who may not have anyone who can perform data annotation or data entry tasks. All of the scripts performed in either R markdown file in RStudio or Jupyternotebooks in Visual Studio Code can be found in the Emergency Warning Apps GitHub repository referenced below.

References

Bonaretti, D., & Fischer-Preßler, D. (2021). Timeliness, Trustworthiness, and Situational Awareness: Three Design Goals for Warning with Emergency Apps. *Forty-Second International Conference on Information Systems, Austin 2021*, 17.

Gazoni, E., & Clark, C. (2022, May 24). *Openpyxl*. openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files — openpyxl 3.0.10 documentation. https://openpyxl.readthedocs.io/en/stable/

Honaker, J., King, G., & Blackwell, M. (2022, November 19). *Package Amelia: Program for Missing Data*. The Comprehensive R Archive Network. https://cran.r-project.org/web/packages/Amelia/index.html

JoMingyu. (2019, July 17). *Google-play-scraper*. PyPI. https://pypi.org/project/google-play-scraper/

Kelly, T., & Bonaretti, D. (2022, May 26). *Tkelly1107/emergency-warning-App*. GitHub. Retrieved November 30, 2022, from https://github.com/tkelly1107/Emergency-Warning-App

MonkeyLearn. (2022). *Text Analytics*. https://monkeylearn.com/

Mullen, L. (2022, January 23). *Lmullen/genderdata*. GitHub. Retrieved November 29, 2022, from https://github.com/lmullen/genderdata

Müller, K. (2013). *Here package*. A Simpler Way to Find Your Files. https://here.r-lib.org/

Python Software Foundation. (n.d.). *Sqlite3 — DB-API 2.0 interface for SQLite databases — Python 3.11.0 documentation*. Python.org. Retrieved November 27, 2022, from https://docs.python.org/3/library/sqlite3.html

Robinson, D., & Silge, J. (2022, August 20). *Package tidytext*. The Comprehensive R Archive Network. Retrieved November 29, 2022, from https://cran.r-project.org/web/packages/tidytext/index.html

Tidyverse. (n.d.). *Tidyverse packages*. Retrieved November 29, 2022, from https://www.tidyverse.org/packages/