Analyzing GitHub Commit Patterns: A Time Series and Natural Language Processing Approach

William Zhang

2023-04-27

Introduction

The increasing popularity and practice of collaborative software development has led to a burst of open-source repositories on GitHub, which is an internet hosting service for software development and version control using Git. Many state-of-art technology and open-source projects are hosted on Github, notable ones include Python, Linux, chromium, and more. It is with no exaggeration to say that the success of the programming community now lies heavily on the success of Github. Furthermore, being able to use Github and understanding the concept of version control is classified as an important skill for most programmers nowadays.

One interesting aspect of version control is the commit messages that developers leave when making changes to the code. These messages document the changes made and why they were made. A well-written commit message can provide insight into the evolution of code over time and can be analyzed to understand the development process and contributors' behavior.

In this paper, we aims to study the behavior and patterns of commit messages available on GitHub, consisting of two parts. The first part will analyze the overall trends and patterns of commit messages by examining the 20 most popular repositories on GitHub. This part will include key variables, such as the time of the commit, the repository committed to, and most importantly, the actual commit message in string format. This part's objective is to establish a baseline for commit message behavior.

The second part of the project will involve a more focused analysis on specific individuals. We will manually scrape all repositories of a selected individual using the GitHub API and analyze their commit messages using similar methods in part 1. The objective of this part is to compare the selected individual's commit messages to the overall trends and patterns found in part 1, providing insights into individual development practices and potential differences and similarities in commit message behavior between individuals and the wider programming community. We also aim to add more engagement to the project by analyzing interesting individuals, such as friends and colleagues.

In short, our project seeks to answer a simple question: What and when do people write in commit messages? By examining GitHub's most popular repositories and specific individuals' commit messages, we hope to gain a deeper understanding of commit message behavior and its impact on the programming community.

Method

The method section will include several subsections related to data wrangling, specifically how the data is obtained (for both parts), how the data-frame is evaluated and cleaned, as well as how the dataframe is modified so that the corresponding plots and tables can be produced.

Data Collection

We will describe the data collection separately for both parts too, since they are acquired quite differently.

Table 1: Top 7 rows of the raw Github Commit Messages dataset

commit	author	date	message	repo
692bba578efb5e305c	Mortada Mehyar <mor-tada@users.noreply.g	Wed Apr 21 12:27:07 2021 +0800	DOC: add example for plotting asymmetrical error bars (#41035)	pandas-dev/pandas
855696cde0ef5d80a7	Patrick Hoefler <61934744+phofl@u	Wed Apr 21 01:23:07 2021 +0200	Add keyword sort to pivot_table (#40954)	pandas-dev/pandas
eaaefd140289a51036	attack68 <24256554+at-tack68@users.norepl	Wed Apr 21 01:21:22 2021 +0200	ENH: ‘Styler.highlight_quantile method (#40926)	pandas-dev/pandas
aab87997058f3c74ba	attack68 <24256554+at-tack68@users.norepl	Wed Apr 21 01:01:03 2021 +0200	ENH: add ‘decimal‘ and ‘thousands‘ args to ‘Styler.format()‘ (#40596)	pandas-dev/pandas
9c43cd7675d961740	Simon Hawkins <simonjay-hawkins@gmail.com	Tue Apr 20 23:58:18 2021 +0100	[ArrowStringArray] Use utf8_upper and utf8_lower functions from Apache Arrow (#41056)	pandas-dev/pandas
9e091427a26aa313bb	jbrockmendel <jbrock-mendel@gmail.com>	Tue Apr 20 15:56:41 2021 -0700	BUG: groupby.rank with MaskedArray incorrect casting (#41010)	pandas-dev/pandas
1b95e1bdb566242b3	jbrockmendel <jbrock-mendel@gmail.com>	Tue Apr 20 15:55:13 2021 -0700	DOC: more accurate wording in roadmap (#41057)	pandas-dev/pandas

For **part 1**, the dataset is directly acquired from the website Kaggle, Github Commit Messages Dataset. Although the dataset is from Kaggle, we made sure that this one does not have much attention (e.g. there’s only 5 related analysis/code for this dataset, which is a pretty small number for all datasets on Kaggle). The full dataset has 4,336,299 observations with 5 variables: “commit”, “author”, “date”, “message”, “Repo”. Table 1 below shows the first 7 entries of the full dataset, directly read from the csv file using `read.csv`.

Looking at Table 1, the variable `commit` is an unique identifier of the specific commit, `author` is the Github user who made this commit, `date` specifies the time of the commit, where the last five characters indicates the timezone, `message` is the string provided to each commit, and is also the main variable we want to anaylze, lastly, `repo` is the repository the commit was made to.

Now, we proceed to the data collection for **part 2**. This part is a lot more complicated when it comes to getting the data. We used the Github API to manually scrap the user that we want to look at. The first thing is to obtain a personal access token from the Github website, which serves as a key to using the API. Then, we proceed by sending a `GET` request to the API, parse the `JSON` response, create an empty dataframe for storage, then lastly loop through the `JSON` response to extract the details of each repositories.

Inside the loop, a lot of sophisticated code was used to obtain the information we need. We first need to extract the repository name, the username (own of repo), then use `paste` to create another endpoint to send to `GET` to retrieve the actual commit history of each repository. Then, again, we parse the `JSON` response, turn it to character, and loop through each commits. Here, we are only storing the commit message, author, repo name and username, and we store them into the empty dataframe we created earlier. Sometimes we encounter a repository with no files (newly created repos), and we need to use `is.null(commits$message)` detect such repos and skip those repos so that our loop won’t run into problems.

Below in Table 2 shows the first 5 commits of repositorys under my Github account: `WilliamQD`.

Table 2: Raw dataset scraped from Github under the account WilliamQD

repo	username	commit_author	message
Chinese-Idioms-Translation-with-Neural-Network	WilliamQD	WilliamQD	Update README.md
Chinese-Idioms-Translation-with-Neural-Network	WilliamQD	Kexuan Zhang	Add files via upload
Chinese-Idioms-Translation-with-Neural-Network	WilliamQD	Kexuan Zhang	Delete Assessing_Fine_tuned_LLMs_for_Chinese_Idiom_Translation (1).pdf
Chinese-Idioms-Translation-with-Neural-Network	WilliamQD	Kexuan Zhang	Add files via upload
Chinese-Idioms-Translation-with-Neural-Network	WilliamQD	WilliamQD	Update README.md

It is worth noticing that even though the all repositories scraped are under my account, we still chose to include commits made by other users / contributors as well. For example the second row is a commit made by my friend Alex in first year. Furthermore, the raw dataset contains a lot of `NA` entries (row 21 for example), with no commit messages; this is unfortunate a very bad practice, made by me in the naive days of just started programming.

Data Cleaning

Now, with both raw dataset ready, the next step of data wrangling is data cleaning. We will follow the previous practice of first presenting the result of part 1, then part 2.

The data which was collected from Kaggle is well-formatted, and does not require any immediate cleaning in order for the information in the table to be readable and usable. However, since the full dataset contained more than 4 million observations, it would be too computationally expensive for us to work with a dataset of that size, therefore we decided to "size down" the dataset by ten times, to achieve that we selected every 10th row of the dataframe (e.g 1, 11, 21, . . .), the smaller dataframe is now with nrow of 433,630. The next step is to remove certain unnecessary columns, specifically `author` and `commit` since those variables won't be the interest of our analysis, which means `message`, `date`, and `repo` were kept.

Now, it remains to check for extraneous observations as well as `NA` entries. A quick `colSum(is.na())` returns a total of 0 entires, suggesting no `NA` entires, however, we suspect that there could be **empty** strings in the `message` column, which does not count as `NA`, and indeed we found **32** entries of such, they were removed as a consequences. Finally, the cleaned dataframe has dimension of (433598, 3).

Now, we switch gear to data cleaning for part 2. As seen in Table 2, the dataframe is "clean" in the sense that it does not require addition column pruning, nor it should not contain any extraneous observations. There are only two things to be taken care of: remove all non-English language characters (e.g. chinese characters) and any `NA` or empty string entries. We utilize the techniques we learned in class and use `string_replace_all` and the regular expression `"[^\x01-\x7F]+"`, which matches on only English characters (including punctuation and spaces), to turn all non-English characters into " ".

Then, we use `filter` to first remove all the `NA` rows. Lastly, we apply a similar method to remove the empty strings using the regular expression `^[[:space:]]*$`, which matches to any length of whitespaces. **The**

cleaned dataframe can be found in the interactive plots section on the project's website. With both dataset cleaned, we're ready to move onto the next step of the data processing procedures.

Data wrangling

We define the process of data wrangling to be steps we took, and the creations of intermediate dataframes for the usage visualization and further analysis. This means we will be describing and showing the transformed dataframes that we made to produce the plots in the later section (results).

Essentially, to answer the question we proposed in the beginning of the report, we will use two category of methods: **Time series analysis** and **Natural language processing**.

Time series analysis

Time series analysis is the study of a sequence of data collected over an interval of time, where time is a significant variable of the data, and almost always one of the predictors. The data we've collected falls under time series since it includes a variable `date` that captures the exact time which the commit was made, for over a long-period of time – the earliest record is in the year 1970 and the latest is 2021. This means we can use date to investigate the relationship of commit behavior with respect to various time factors, like year, day of week, and hours, which helps us to answer "when do people usually make commits", and also questions like "is there a particular time interval that is favored by the majority programmer to work".

However, the raw format of the `date` variable doesn't allow us to do so, we must extract the various component of the variable using functions. The original `date` variable has the form "Wed Apr 21 12:27:07 2021 +0800", which translates to "dayofweek month day time year timezone".

An important "engineering choice" that we're making here is that we will "disregard" the timezone part of the `date`, this is because currently the date is recorded in local time, and we want to preserve that local time property since it makes more sense to analyze with that; it wouldn't make sense, for instance, that when someone made a commit in Asia at 4pm local time, but we instead convert it to 4am Europe and conclude that a lot people commit at 4am. Thus, we rather choose to work in local time and abondon the timezone part of the `date` variable.

Now, all there's left is to convert the original `date` into a more standardized datetime format. To do so, we first truncate the last 6 characters to get rid of the timezone, then use the method `as_datetime` which converts the `date` variable to the format `"%a %b %d %H:%M:%S %Y"`, equivalently to something like "2021-04-21 12:27:07".

Then, we simply need to use the corresponding function `hour()`, `wday()`, `month()`, and lastly `year()` to get each component of the modified date variable and make them into new variables. We shown below in Table 4 the first 5 rows of the new dataframe that will be used for time series analysis:

Table 3: Commit message dataset with time variables

date	message	repo	hour	day_of_week	month	year
2021-04-21 12:27:07	DOC: add example for plotting asymmetrical error bars (#41035)	pandas-dev/pandas	12	Wed	Apr	2021
2021-04-21 00:49:55	Bug in to_datetime raising ValueError with None and NaT and more than 50 elements (#41006)	pandas-dev/pandas	0	Wed	Apr	2021

Table 3: Commit message dataset with time variables *(continued)*

date	message	repo	hour	day_of_week	month	year
2021-04-19 22:52:28	fix: function _take_with_is_copy was defined final but overriden (#41004) * fix: function _take_with_is_copy was defined final but overriden * fix: add return type	pandas-dev/pandas	22	Mon	Apr	2021
2021-04-19 06:41:50	REF: pass arguments to Index._foo_indexer correctly (#41024)	pandas-dev/pandas	6	Mon	Apr	2021
2021-04-16 11:57:20	TST: use expected_html for to_html tests (#40981)	pandas-dev/pandas	11	Fri	Apr	2021

Natural Language Processing

Natural language processing is an integral part of computer science nowadays, particularly in the field of artificial intelligence. It studies how to program computers to process and analyze large amounts of natural language data, according to the definition on Wikipedia.

Natural language processing comes in many applications: speech-to-text, semantic analysis, machine translation, and more. For our project, natural language processing comes as a *nature* technique to implement, since all the commit messages are strings. Furthermore, through NLP, we can explore the most common occurence of words in commit messages across different repositories, investigate their relative frequencies, and create visually appealing plots like wordclouds to show the distribution of words in commit messages, which would help us to answer the question: “what do people write in their commit messages”, as well as questions like “is there a common structue / theme in commit messages”, “how do people concisely represent a change to the codebase”.

To achieve the aforementioned, we will be following a common pipeline for natural language processing: tokenization, then stopword removal. Tokenization is the process of turning sentences into individual “tokens”, which is usually a unit of word that can more easily be assigned meanings.

Before tokenization, we pre-removed all the digits in the `message` variable, this is because numbers aren’t as meaningful to analyze for our project, since individual digits don’t carry much information related to what the commit message is about, we would hardly infer anything from them. Another reason is that most commit messages consist of a reference number to the commit in 5 digits, and we would like those to be gone. With the numbers removed, we used `unnest_token` and with the parameter `to_lower` set as `TRUE` so that all words are considered regardless of cases, then we filtered out all the stopwords using `stopwords("english")`.

We also created another dataframe of tokens that is group by each repo so that it is easier to compare across different repositories, Table 5 shows a snippet of it:

Table 4: Tokenized commit message grouped by repo

repo	tokens	n
torvalds/linux	signed	185380
chromium/chromium	https	123450
chromium/chromium	chromium.org	123191

Table 4: Tokenized commit message grouped by repo *(continued)*

repo	tokens	n
chromium/chromium	chromium	120562
chromium/chromium	reviewed	110995

This table will be used to create the **animated wordcloud** in the following section of preliminary results and visualization. However, there remains one problem with this dataframe. That is, since n is calculated globally avaliable for all wordclouds, repositories with less commits will have smaller tokens counts (smaller n) naturally, this causes the word size in those repo's wordclouds to be very small, if we set n as the parameter for size.

That motivates us to have a relative tokens count for each repo, a locally percentage that shows how much the token accounts for in all tokens of that repository specifically. To do so, we need to first extract the total number of token counts for each repo, which we've done so in `msg_token_byrepo_count`, then **merge** that with the dataframe shown in Table 5 to get the percentage of each token compared to all tokens of that repo only.

The resulting merged dataframe in shown in Table 6, **where `n.x` represents the count of that token in that repository only, and `n.y` represents the total count of tokens in that repo, and percentage is calcualted as 'n.x / n.y**:

Table 5: Tokenized commit message with percentage based on each repo

repo	tokens	n.x	n.y	percentage
apache/httpd	httpd	6560	77478	0.0846692
jupyterlab/jupyterlab	jupyterlab	1427	19596	0.0728210
apple/swift	swift	16070	235507	0.0682358
nodejs/node	reviewed	7885	122197	0.0645270
jupyterlab/jupyterlab	alpha	888	19596	0.0453154

Notice that for the same repository (Tensorflow for example), `n.y` is the same, which conforms to our specification. With this dataframe ready, we can create individual wordclouds that reflect the distribution of tokens in each repos locally. This also concludes the NLP steps for Part 1, meaning that they're ready for visualization and interpretation now.

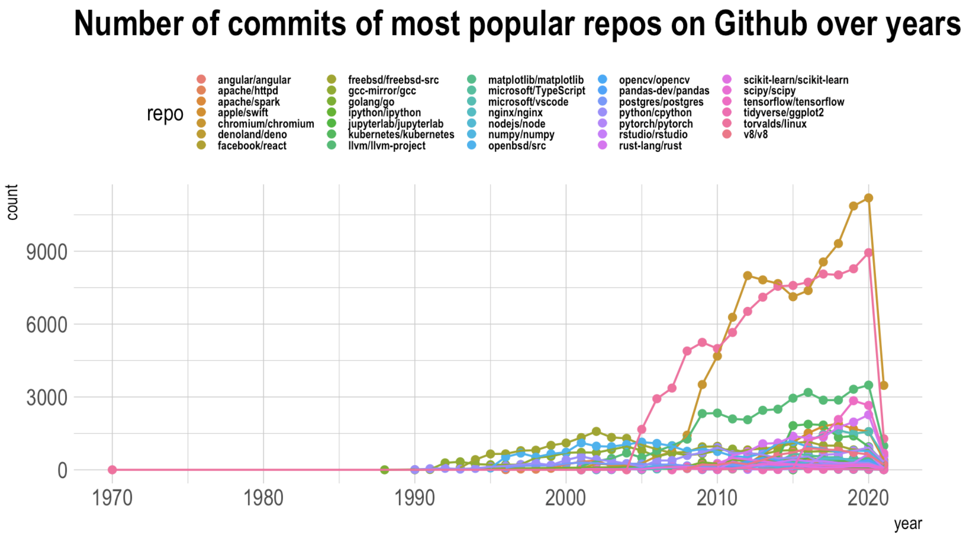
For part 2, the same procedure applies, thus we will omit the details of transforming the messages and present the processed dataframe in Table 7 below:

Table 6: Tokenized commit message of repo 'WilliamQD' with counts

tokens	n
main	19
update	17
merge	16
branch	11
readme.md	10

Preliminary Results and Visualizations:

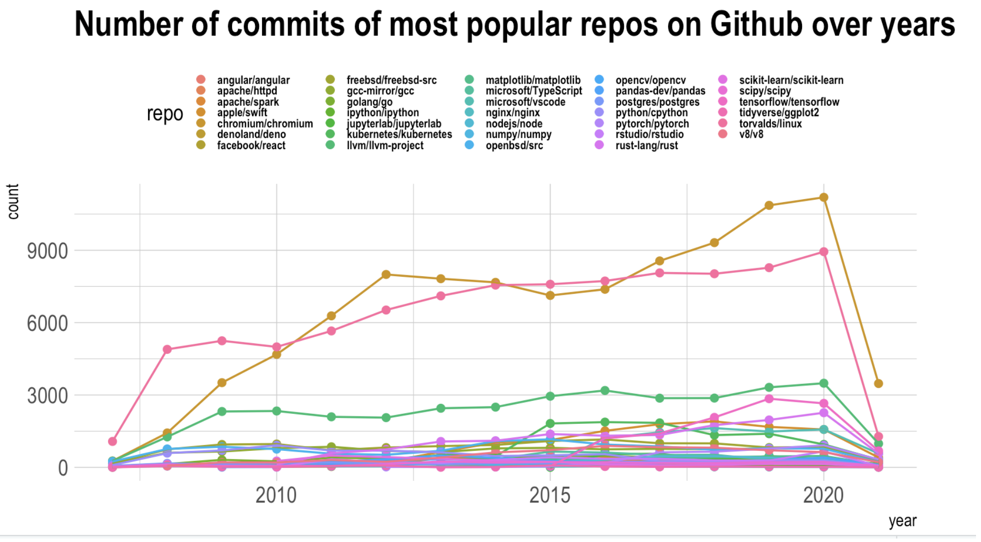
Coming back to our research question: “What and when do people write in commit messages?” To answer this, we will create several time series plots using `year`, `day of week` to answer the when part of the question; then we switch gear to wordclouds to answer the what part of the question. Let’s begin with the first plot that shows the transition in years and the number of commits for each repo over the years:



The animated version can be found on the website’s animated plot section

We see that the data does indeed begin in 1970, and continues all the way to 2021. Although it starts in 1970, there’s only a few records from 1970 to 1990, and all the commits comes from one repository “torvalds/linux”. We suspect that this could be due to an error in date where the “default” date for a lot of programs is “1970/1/1”, also known as the “Unix time”. If we disregard this fact, we see that the second repository that started having commits is “freebsd”, which is also an Unix-like operating system, developed a long time ago back in the 90s.

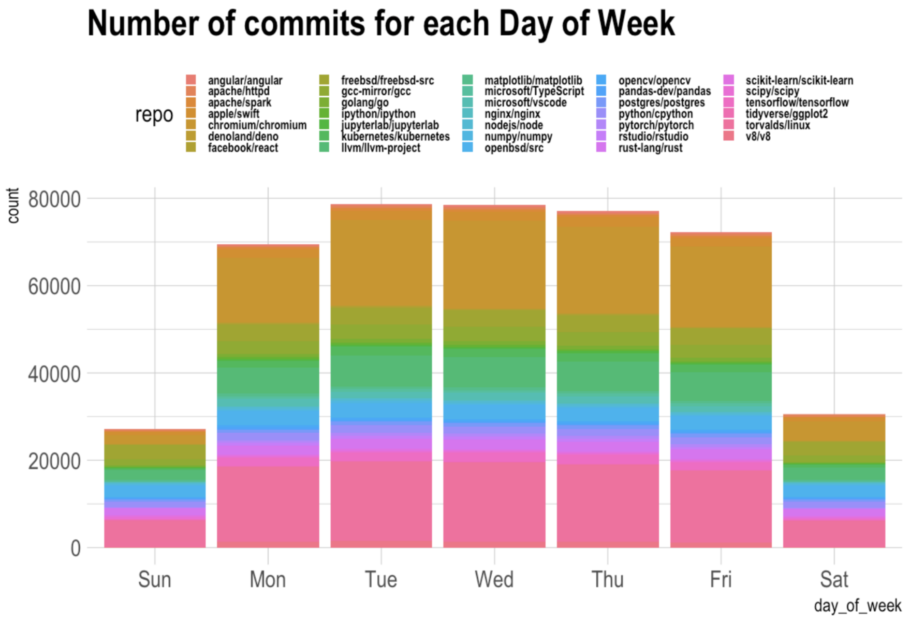
Thus, taken this into account, we fit a more `reasonable` plot that begins on the date when the development of Github actually began, **October 19, 2007**:



This presents a much better view for the number of Github commit messages. As time progresses, the number of commits skyrockets with some super popular repositories such as “chromium” (the one in brown), “linux” (the one in pink), and “llvm” (the one in green). The sudden decrease at the end is likely due to the fact that the data was collected during middle of 2021, thus for 2021, it has a lot less observations than the previous years. *Another important thing to note is that the displayed “count” for commits is actually roughly 1/10 of the actual number of commits, since we truncated the dataset to 1/10 of the original dataset at the beginning for faster computation*, this applies for the rest of the results if it involves number of counts for any commits, and should be kept in mind by the reader.

Coming back to the plot, we see a generall steady increase in the number of commits, signifying that collaborative programming has indeed became the mainstream of projects over the past decades. It is without a doubt that the practice of open-source helped many projects to develop and improve at a pace which could never been achieved before when done solely or in a small team.

Next, we investigate the relation between DOW (day of week) and number of commits:

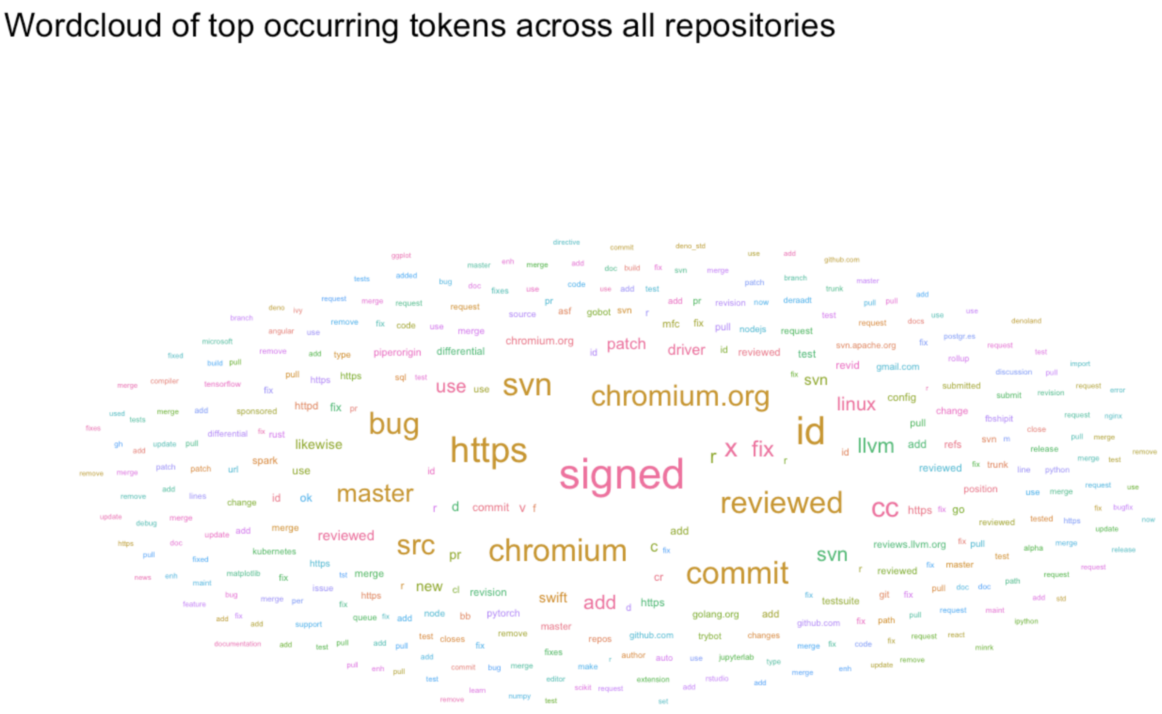


Based on this plot, we see that most commits are done in the weekdays, which is reasonable given that the majority of people work on weekdays. Within weekdays, Tuesday, Wednesday, and Thursday have similar number of commits, while Monday and Friday have less but still considerable amounts. Saturday and Sunday both have less than half compared to any of the weekday’s.

We can reasonably infer that the contributors to the projects are predominantly working professionals, as the highest number of commits are made on weekdays when most people are at work. The similar number of commits on Tuesday, Wednesday, and Thursday suggests that these are the most productive days for the contributors, while Monday and Friday may have fewer commits due to other work or personal commitments.

The lower number of commits on Saturday and Sunday suggests that many contributors take these days off from working on the project, which is not surprising given that weekends are typically reserved for leisure time and personal activities.

Next, we switch to natural language processing. The below wordcloud shows the most occurring tokens across all repositories included, where the size of the token indicates its frequency and the color indicates the repo it’s from. We chose not to include a legend since that would make the wordcloud look messy, but one can reasonably infer what color corresponds to what repo, given the number of commits plots from earlier, at least for the top contributed repositories:



The wordcloud contains some surprisingly results as well as some expected ones. For example, we see words like “commit”, “reviewed”, “bug”, “fix”, which are very common words used in commit message to document changes. However, we also see tokens like “signed” which we’re unsure of the reason and meaning of it. There are also a lot of “chromium”, “https”, which could be related to the structure of syntax of commit messages of specific repositories; for instance, there could be a rule when making a commit to chromium such that the word “chromium” must be included. Just a wild guess.

We also use the “relative frequency” dataframe shown in Table 6 to make a wordcloud for each repositories. This allows us to examine each repository in details and know the local distribution of tokens. The wordclouds are shown in an animation below, notice that we only took the top 20 tokens of each repository:

The animated wordcloud can be found on the website’s animated plot section

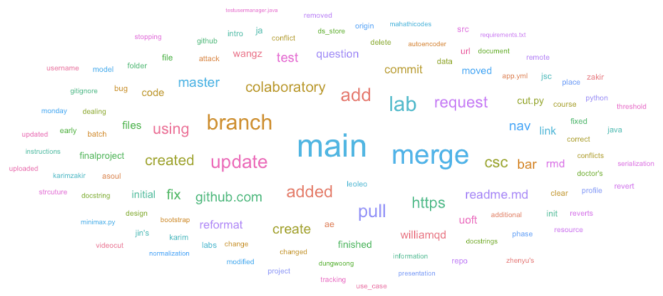
The wordclouds produced for each repository demonstrated that the tokens were more closely related to the specific repository they belong to. For instance, we noticed tokens such as “Rstudio” in the Rstudio repository, “Pytorch” in the pytorch repository, and “Go” in the go repository. These repository-specific tokens provided valuable information about the themes and topics related to each repository.

Moreover, we identified more common words related to commit messages in the generated wordclouds. The majority of the wordclouds contained words such as “Merge,” “Pull,” “Fix,” “Add,” “Test,” and “Bug,” which are frequently used in commit messages. The presence of these common commit message words reaffirms the fact that our analysis was focused on the right set of data and provides a better understanding of the development activities for each repository.

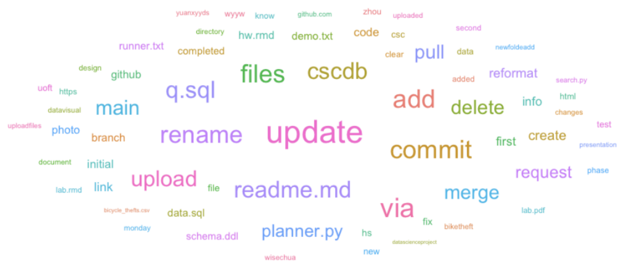
Overall, the wordclouds created using the “relative frequency” dataframe offered a more detailed and comprehensive view of each repository. The repository-specific tokens and common commit message words identified in the wordclouds provided us with valuable insights into the development activities and themes associated with each repository.

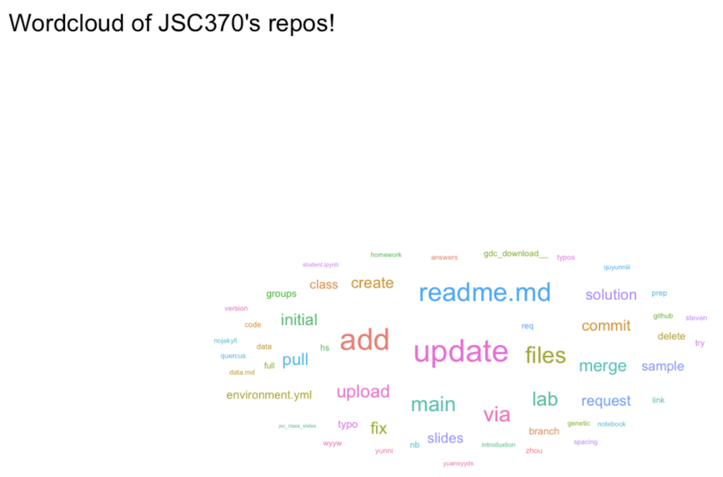
Lastly, we will look at the wordclouds for **Part 2**, specifically I’ve extracted data from three users: `WilliamQD`, which belongs to me, `Yuanxyyds`, my friend Steven liu who’s also in this class, and lastly `JSC370`, the user representing this class and its previous offerings.

Wordcloud of my repos!



Wordcloud of Steven's repos!





Interestingly, the most frequent token in both JSC370's and Steven's commit messages is "update," while "main" is the most commonly used token in mine. Furthermore, there is a significant overlap of tokens in the wordclouds, indicating that we tend to use similar words in our commit messages.

While the primary objective of this section is for entertainment purposes, it is worth noting that the words we use in our commit messages differ from those found in popular repositories. However, we must exercise caution in drawing any conclusions since our sample size is relatively small.

Time Series Modeling

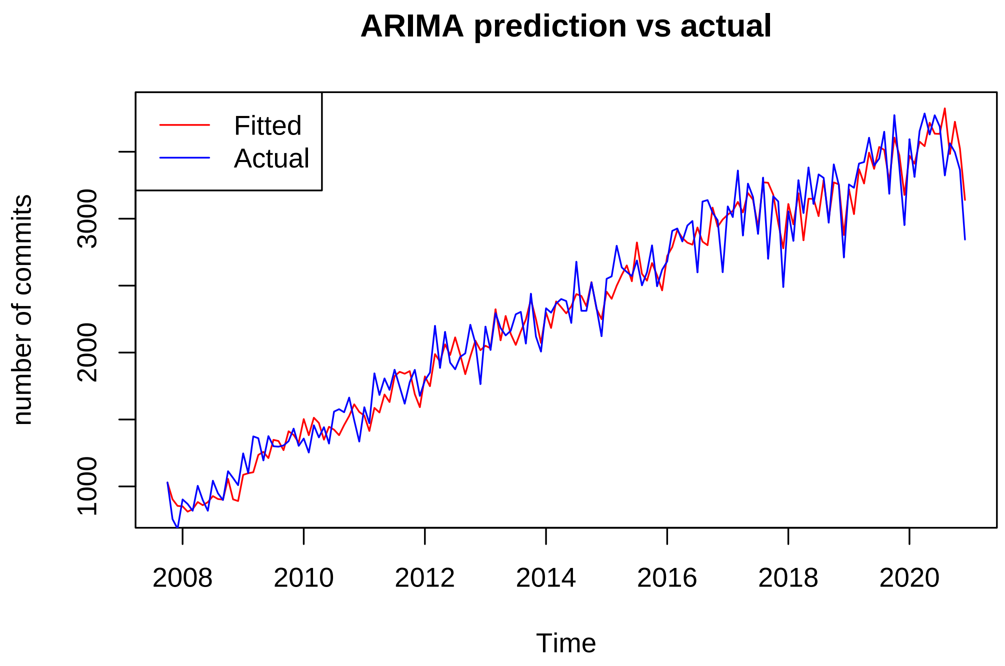
In addition to the visualizations, it comes natural to fit time series models on our data to investigate the patterns in the number of commits of the popular repositories. In this section, we will be fitting two models: an Autoregressive Integrated Moving Average (ARIMA) to investigate the inheret trend in the number of commits and a Seasonal Decomposition of Time Series by Loess (STL) to decomposite the seasonal, trend, and irregular components of the number of the commits.

To do so, we need to first create a time-series object using the function `ts`. Like earlier, we've restricted our date to start from *2007-10-01*, which is when github started development, futhermore, we ended our data on *2020-12-31*, since the data for 2021 was incomplete. Then, we aggregate all the commits from all repositories and group them based on year and month, then we add a "-01" component for the "DD" component of a `Date` object to so that they can be converted to a `Date` object. The dataframe that is used to convert to the time-series object looks like the following:

Table 7: Time series table for number of commits

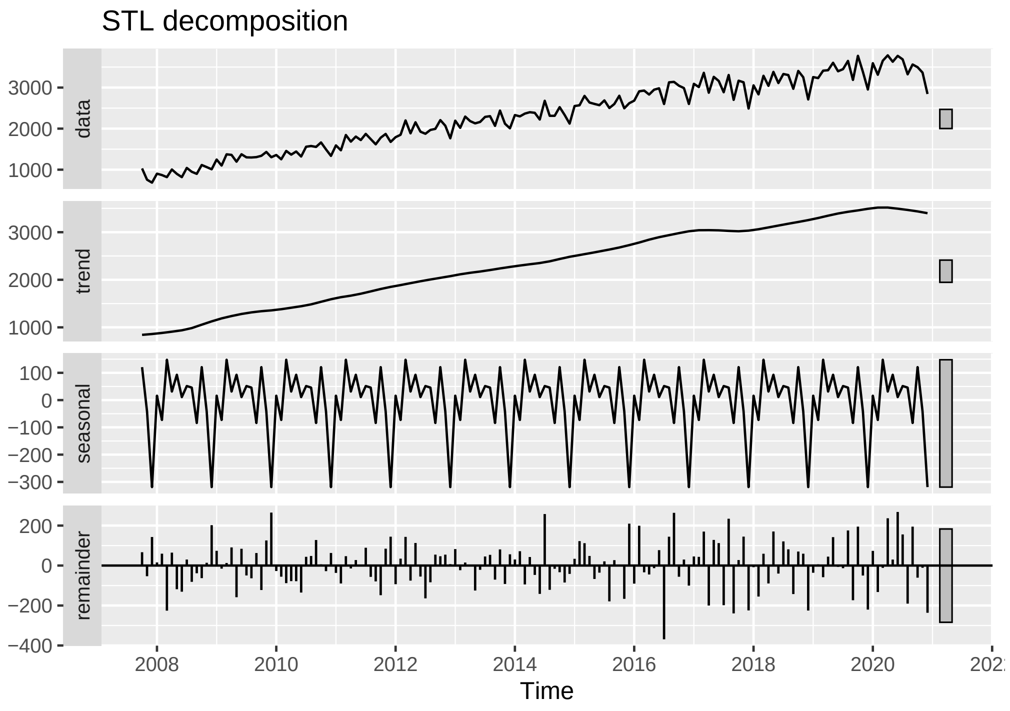
date	count
2007-10-01	1030
2007-11-01	756
2007-12-01	684
2008-01-01	903
2008-02-01	869

Then, we apply `ts` and fit the models. First one is the ARIMA model using `auto.arima` from the `forecast` package. A plot of fitted vs actual is shown below:



We see that the fitted (predicted) number of commits is quite close to the actual count. As the ARIMA model is able to capture the underlying patterns and trends in a time series data, such as seasonality and long-term trends, a close match of fitted vs actual suggests that the model is a good fit for the data. This is an indication that the model can be used to forecast future number of commits with a reasonable degree of accuracy.

Next, we decompose the data using STL, this can be simply done through `stl`. Then, using `autoplot` from ggplot we can visualize the different components of our time series data.



From the decomposition, the first thing we observe is the steady increase in the number of commits over the years from the "trend" component, which matches our earlier conclusions. Then, the oscillating seasonal component suggests that there exists some seasonal patterns to our data. More precisely, since our data is aggregated monthly and ends on December of 2020, **we see that every December, the number of commits experience sudden drops**, one reasonable explanation to this is that most people are on Holidays like Christmas, or decides to take a break on programming at the end of the year.

Conclusion

After conducting the methods and results, we were able to answer our research questions regarding both time and content. Our time series analysis revealed a significant increase in commit message counts as programming gained popularity over the years. Furthermore, by analyzing the distribution of commits across the days of the week, we were able to infer coding behavior and preferences.

In addition, natural language processing enabled us to gain a deeper understanding of programmers' commit messages. Using word clouds, we identified the keywords that composed their commit messages and compared them across various repositories. In part two, we compared our commit messages to the baselines of earlier commits.

Overall, our analysis contributed significantly to answering our research questions. We are thrilled with the findings and excited about the potential for future improvements and extensions to our project.

Limitation & Future steps

Regarding the research question, there is still room for improvement in data collection and merging. We could expand our data collection to a more extensive range of repositories and examine commit messages from repositories of various sizes. To improve our analysis, we could plot different variables against commits, such as time of day. Additionally, we could try different natural language processing techniques such as stemming and lemmatization for our token generation process, and examine n-grams rather than individual tokens.