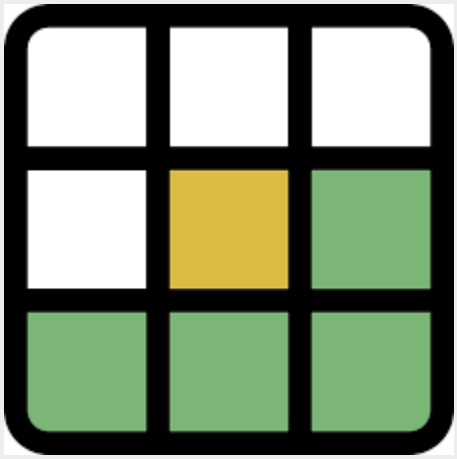
Analyzing Wordle Simulator



What is wordle simulator?

The wordle simulator is a simple project that uses the wordle rules to simulate a round of wordle and outputs a report.
This report contains detailed information about each round, enabling further analysis. By running the simulator multiple times, we can gather sufficient data to conduct statistical analysis and gain insights into the game mechanics.
In this document we try to draw some problems and find answers for them by doing analytics on our simulator's generated data.

How does it work?

The wordle simulator consists of two main functions: *check* and *guess_word*
The *check* function takes a guess and compares it with the answer, then outputs a feedback about it.
The *guess_word* function optionally takes the feedback from the *check* function and generates a new word as an output.

Feedbacks are consisted of three components:
False letters: This is a list of letters that are not present in the answer.
True letters: It is a dictionary that includes letters present in the answer along with their corresponding positions.
Misplaced letters: This dictionary contains letters that are present in the answer, but they are not located in their correct positions.

The *run(n)* function executes the game n times and generates a dataframe containing the details of each round. The dataframe includes the following information:
n_tries: The number of attempts the simulator took to find the answer.
n_choices: A list of the numbers of words that satisfied the criteria in each round.
guesses: A list of the guesses made in each attempt.
feedback: The last state of feedback before the final guess.

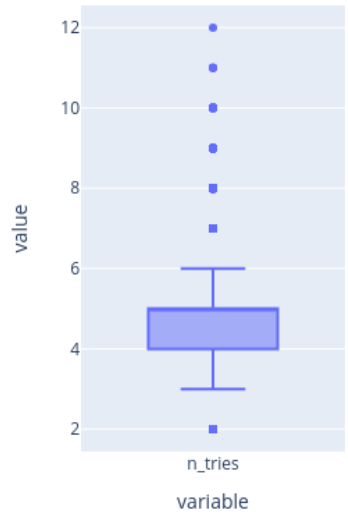
As an example, the 'feedback' for a round could be represented like this: {'false_letters': ['e', 's', 'g', 'd', 'u', 'n', 'i', 'r', 'a', 'l'], 'true_letters': {1: 'o', 0: 'h', 2: 'b', 3: 'b', 4: 'y'}, 'misplaced_letters': {0: ['b'], 2: ['o']}}.
won: It determines whether the simulator found the correct answer in 6 or fewer tries. (a value of 1 indicates success, while 0 indicates failure).

To address the first problem, **we will analyze the random behavior of the game**. Two random variables are involved in each game: the selection of the answer and the guessing of the answer. In order to simulate a more realistic game, we assign weights to these random occurrences. To determine the weights, we utilized the Project Gutenberg Corpus to calculate word frequencies. Subsequently, we created a score metric for the frequencies of each word by dividing them into bins. This metric ranges from 1 to 19 and can be used as input for the random functions we use to simulate a more human-like behavior.

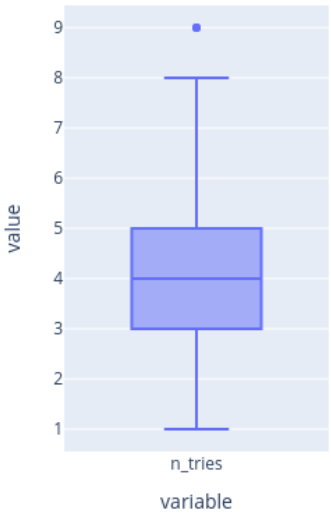
We can compare some analytics by simulating the game both with and without the inclusion of weights. Here are the results after simulating 3000 rounds:

non weighted		weighted	
win_rate	0.9	win_rate	0.98
mean_n_tries	4.73	mean_n_tries	4.01
mean_last_n_choices	3.65	mean_last_n_choices	10.23

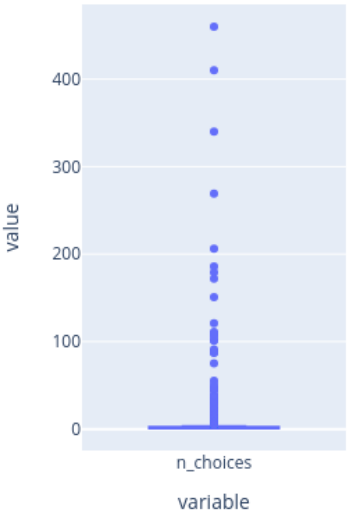
non_weighted_n_tries



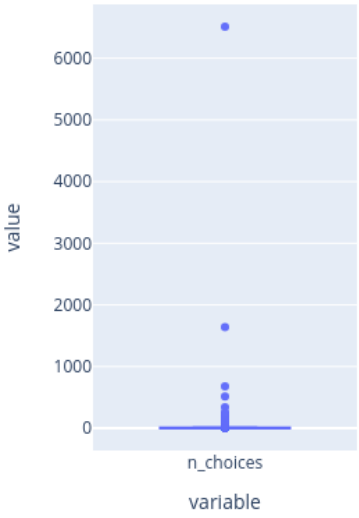
wieghted_n_tries



n_last_choices_non_weighted



n_last_choices_weighted



Using word frequency scores as weights for random choices on words has a significant impact on the win rate. It introduces a fairer and more winnable game experience, reducing the average number of tries by 0.7. Additionally, it reduces the number of remaining choices to approximately seven words on the final guess. This means that, on average, the player has to select from only 3.5 possible words when guessing the final word utilizing the weighted model.

We know that the real game is also similar to the weighted model. This is because the answers primarily consist of common words, and users tend to guess from words commonly used in their everyday language.

To address the second problem, we can **determine the optimal starting words** for the game. By simulating multiple games for each possible word and calculating the mean number of tries required to find the answer, we can compare these values and sort the words from best to worst, thus identifying the most advantageous starting words.

To obtain the results, I utilized the 'run_with_first_guess' function to simulate 100 games for each possible word as the initial guess. By comparing three metrics, we can rank the first guess words from best to worst. These metrics include:

n_tries: The average number of rounds required to find the answer, indicating the difficulty level of the game.
win_rate: The percentage of games that were completed in 6 rounds or less.
last_n_choices: The number of possible words to choose from based on the last feedback received.

By evaluating and comparing these metrics, we can effectively sort the first guess words and identify the best options:

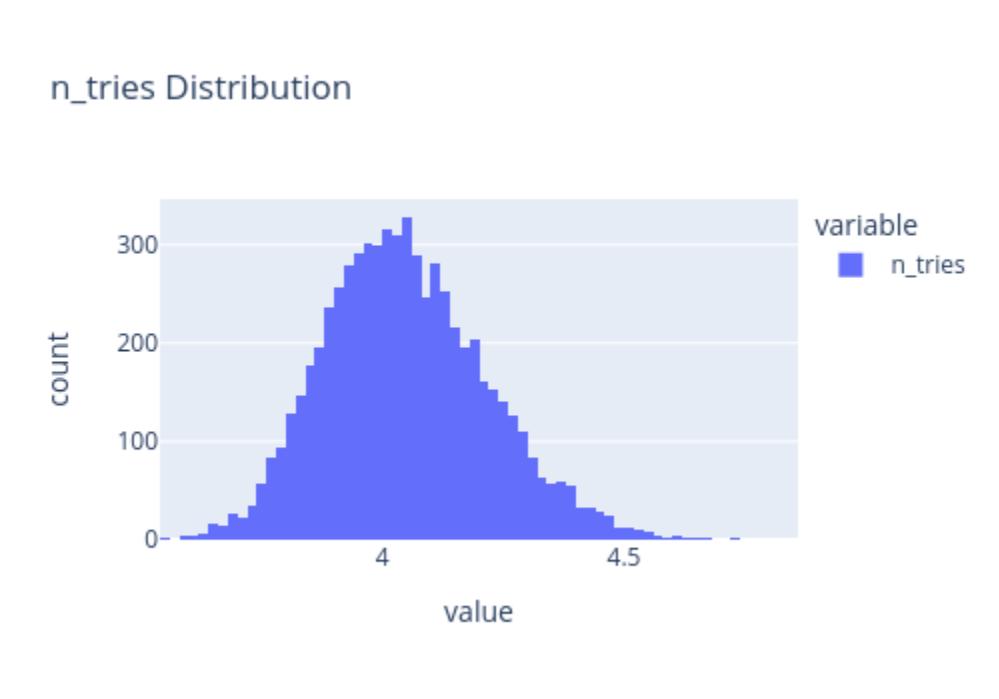
Min n_tries:

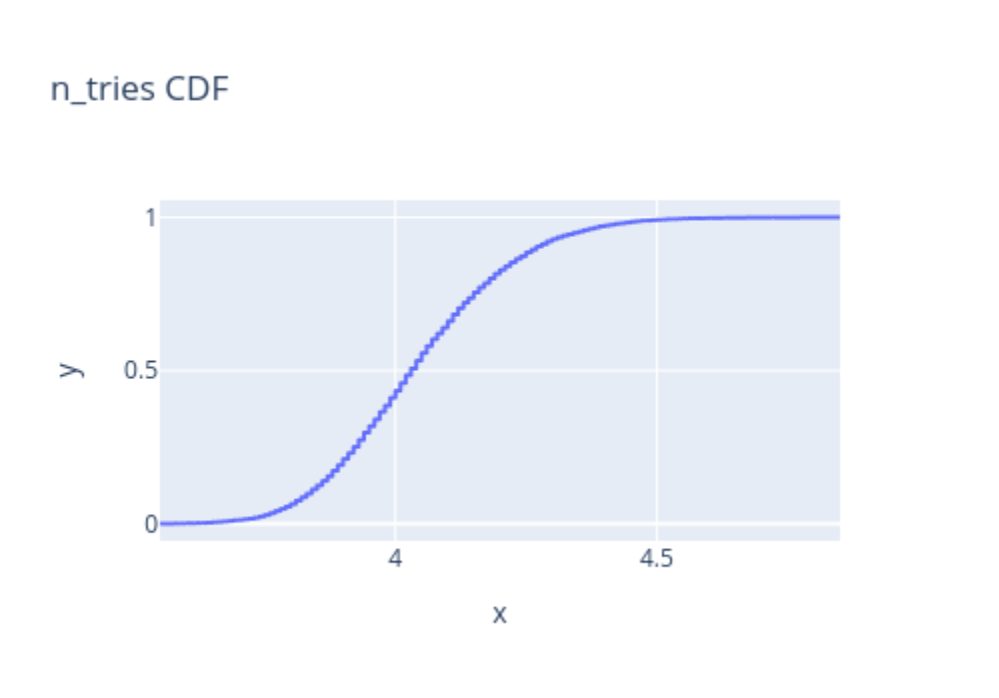
first_guess	n_tries	last_n_choices	won
tamer	3.55	13.97	0.99
tared	3.55	15.74	0.99
dante	3.57	14.36	1
crate	3.58	7.09	1
tesla	3.59	13.3	0.99
trend	3.59	7.71	1
share	3.59	8.37	1
sharp	3.6	7.97	1
eulas	3.6	6.23	1
hilts	3.6	5.72	0.99

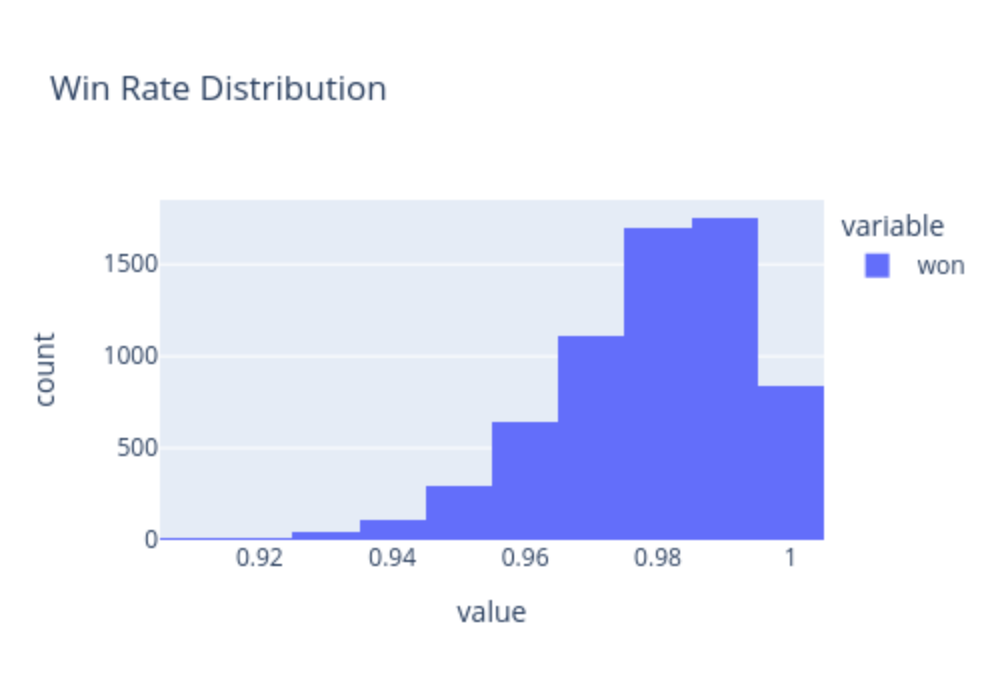
Max Win Rate:

first_guess	n_tries	last_n_choices	won
dante	3.57	14.36	1
crate	3.58	7.09	1
share	3.59	8.37	1
trend	3.59	7.71	1
eulas	3.6	6.23	1
sharp	3.6	7.97	1
waits	3.61	12.65	1
hales	3.62	9.31	1
plant	3.62	14.04	1
rites	3.63	7.24	1

We can also take a look at some plots obtained from the simulation results:







Considering the limited number of vowels in the English language and the spelling rules governing their usage in forming correct words, we hypothesize that there might be a correlation between the number of vowels in the first guess and the outcome of the game. To

validate this hypothesis, we can analyze our data and examine the relationship between the number of unique vowels in the initial guess and the game's outcome.

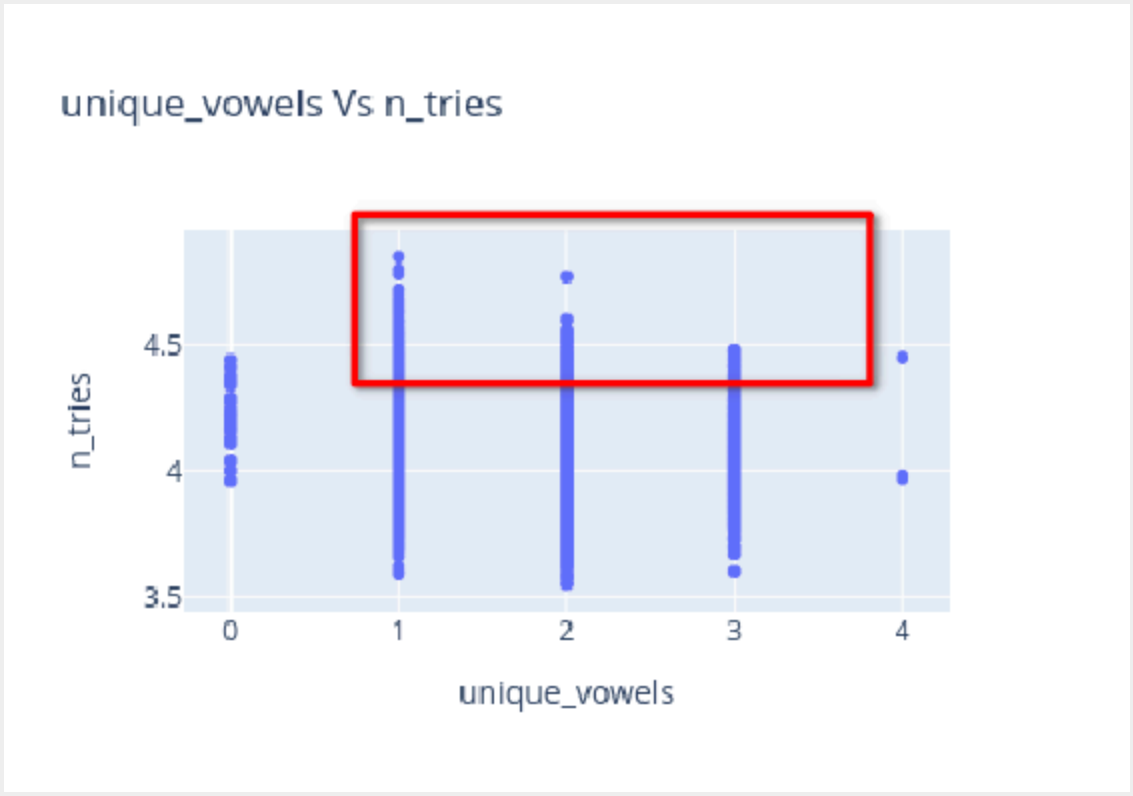
First of all, we can check the correlations:

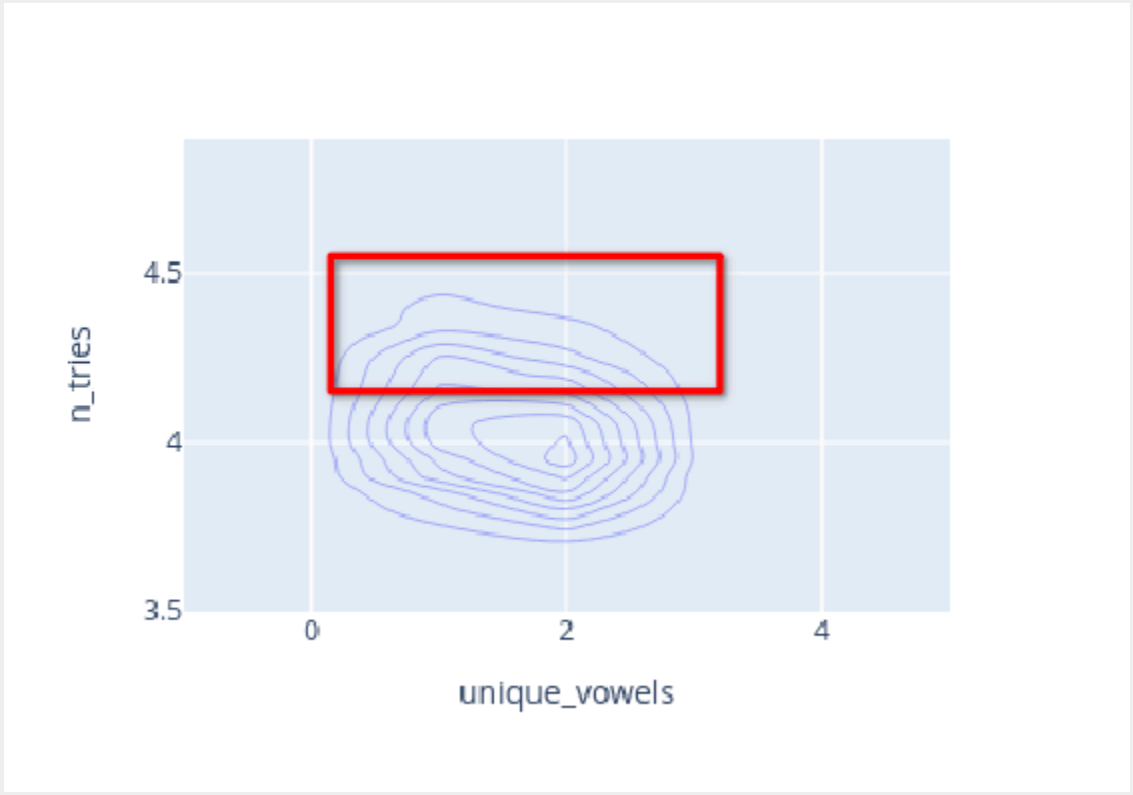
	unique_vowels
n_tries	-18.86%
last_n_choices	-1.70%
won	-3.69%
unique_vowels	100.00%

As we can see, there is a noticeable correlation between n_tries and the number of unique vowels in the first guess.
To analyze further, we can also draw some plots. Before proceeding, it is essential to determine the count of words in each group to avoid inaccurate interpretation.

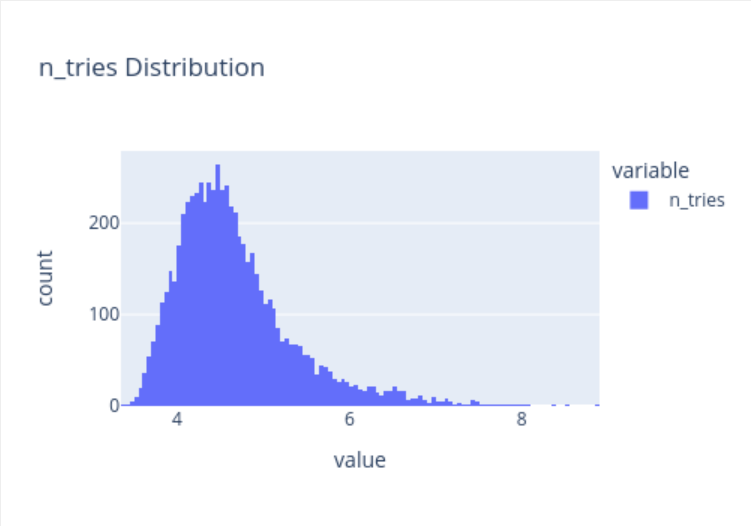
unique_vowels	count	n_tries	won
0	25	4.2072	97.92%
1	2815	4.08469627	97.94%
2	3331	4.010351246	97.91%
3	337	4.03388724	97.58%
4	4	4.0925	97.00%

Given that the distribution of the number of unique vowels in words is not uniform, we cannot draw conclusions for all word groups. Most words tend to have either 1 or 2 vowels, making it feasible to draw meaningful conclusions only for these specific groups.

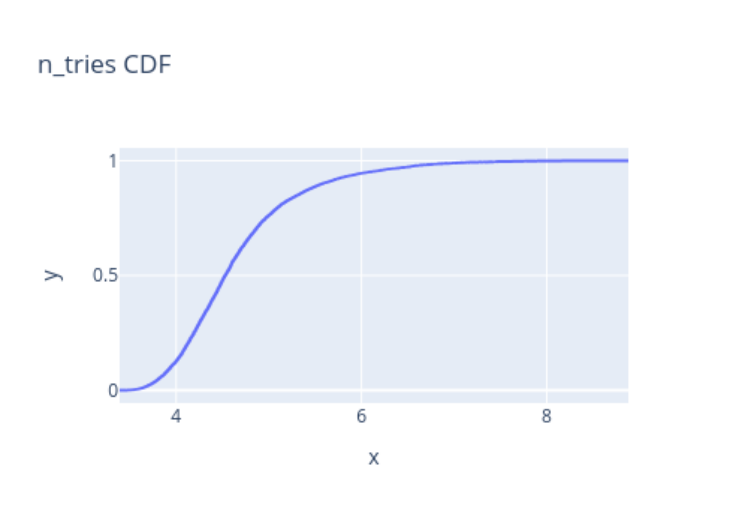

unique_vowels Vs n_tries
n_tries
4.5
4
3.5
0
1
2
3
4
unique_vowels


n_tries
4.5
4
3.5
0
2
4
unique_vowels

For the third part of the analysis we can categorize the words based on their difficulty level by running the simulator for each possible answer.
I chose n = 100 for simulation, which performs the 100 iterations for each possible answer. The metrics used to determine the difficulty level are n_tries and win rate.
Let's examine the results. The histogram below illustrates the distribution of the mean n_tries for each word after the simulation:



To further establish difficulty levels, we can plot the Cumulative Density Function (CDF). This will help us set the boundaries for each category (easy, medium, hard):



Based on the CDF plot, we can categorize the words as follows:
Words in the 0 - 30 percentile (n_tries: 0 - 4.25) are classified as 'easy'.
Words in the 30 - 80 percentile (n_tries: 4.25 - 5.11) are classified as 'medium'.
Words in the 80 - 100 percentile (n_tries: 5.11 - n_tries max) are classified as 'hard'.

We can also check the win_rate for each difficulty level:

difficulty_level	win rate
easy	99.52%
medium	95.63%
hard	69.81%

Based on the results, there appears to be a potential correlation between the presence of duplicated letters and the difficulty level of the words. This suggests that words with duplicated letters may be harder to guess. To investigate this hypothesis further, we need to establish metrics for measuring duplicated letters. The following metrics have been defined:

Duplicated Letters: The number of unique letters that are duplicated in the word.
Duplicated Letters Count: The sum of duplications for all letters in the word.
For example, in the word 'programming', there are 3 duplicated letters (r, p, m) and a total duplicated letters count of 6 (2+2+2).
To assess the relationship between these metrics and the difficulty level, we can begin by examining their correlation with n_tries and win rate.

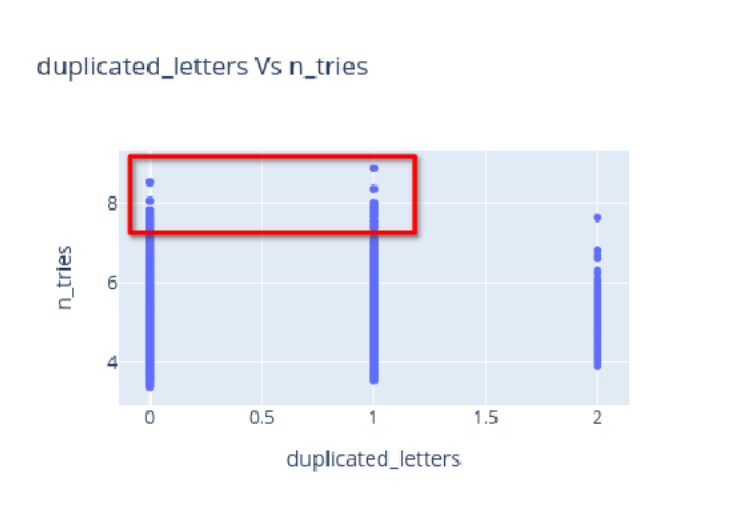
	n_tries
n_tries	100.00%
last_n_choices	-14.60%
won	-92.13%
duplicated_letters	14.57%
duplicated_letters_count	14.95%

There seems to be a potential correlation between the presence of duplicated letters and the difficulty level of the words. To further investigate this relationship, we can examine a scatter plot to determine if any observable correlation exists. However, before proceeding with the plot, it is essential to ensure that we have enough data for each group.

duplicated_letters	count
0	4271
1	2090
2	151

duplicated_letters_count	count
0	4271
2	2025
3	64
4	148
5	4

It appears that we have a sufficient number of words with 0 and 1 duplicated letters, as well as 0 and 2 duplicated_letters_count, in order to proceed with the analysis and the subsequent scatter plot.



The scatter plots provide some indication of the relationship between the variables. To further analyze this relationship, we can calculate the mean values of duplicated_letters and duplicated_letters_count for each difficulty group.

difficulty_level	mean_duplicated_letters	mean_duplicated_letters_count
easy	0.221074	0.443698
medium	0.416488	0.844928
hard	0.460777	0.942879

Based on the provided analytics, there is evidence supporting the hypothesis that words with duplicated letters tend to have a higher difficulty level. The mean values of duplicated_letters and duplicated_letters_count increase as the difficulty level progresses from easy to medium to hard. This suggests that the presence of duplicated letters in a word correlates with an increased level of difficulty in the game.

If you have any interest in this subject or if you have any ideas or recommendations, please feel free to contact me via my LinkedIn address at:
www.linkedin.com/in/alijavanbakht/.
Additionally, you can explore the code in my personal repository located at:
www.github.com/Ali-jb/Wordle-Simulator
Thank you for your interest.
