A ‘Numbers Guy’ Playing Wordle

Documentation

AUG
2022

JACOB
LINK

ENJOY !

Overview

A few months ago, a good mate of mine tried convincing me and another friend to start playing this game; he was claiming it would be a great addition to our morning coffee. Little did he know how much joy it would bring to our morning routine.

I am a ‘numbers guy’. No idea until today why my friend thought I would like this ‘word’ game. On the one hand, Wordle brought back a daily communication to a great friendship, by each of us sharing our daily score through Wordle’s amazing built-in feature. On the other hand, it took me, a Data Science and Engineering student no more than 2 days to decide – I will crack this game.

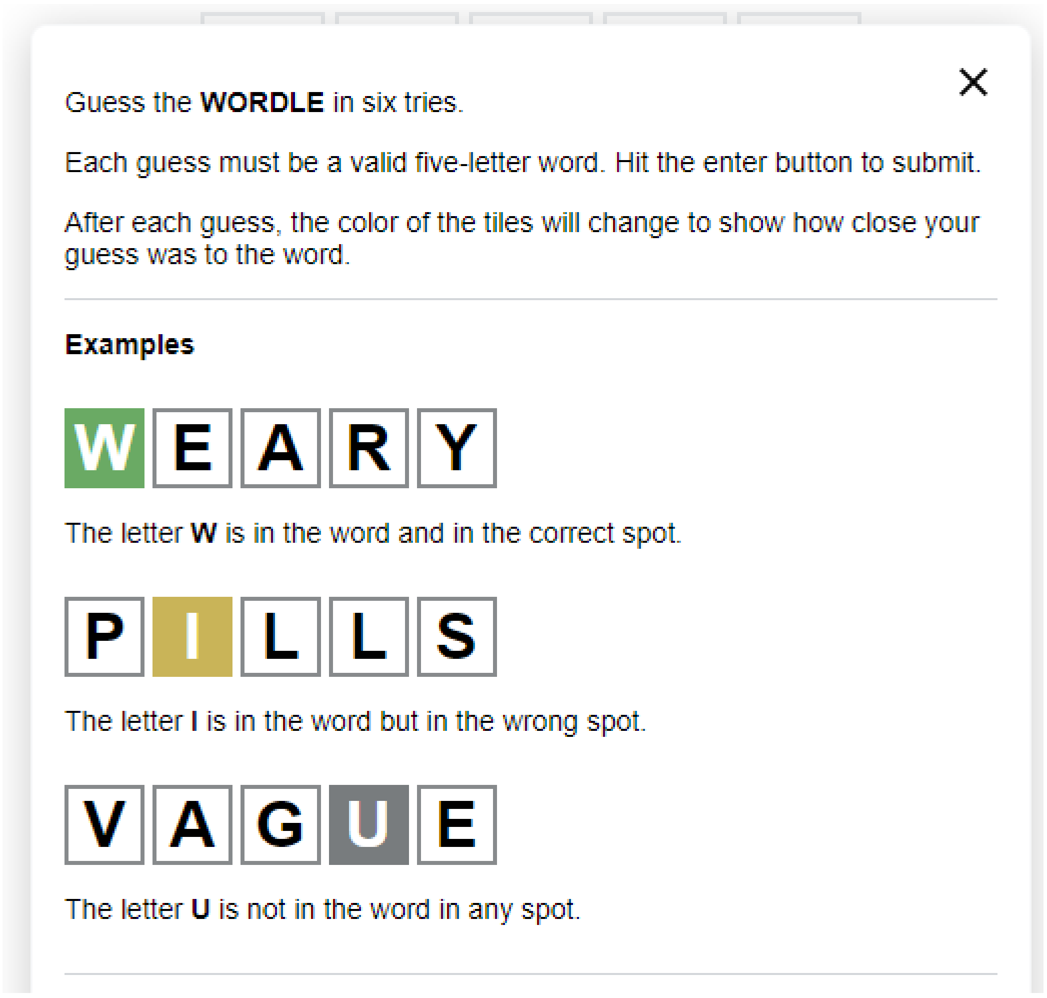
I decided to share the documentation of my thought process of the analysis and code for three main reasons. First, I believe projects built and worked on in one’s free time, shed a lot of light on the people we really are. Second, projects can compensate the “5 years of experience” many jobs require in one’s field of expertise (got to start somewhere...). Third, to all the competitive people out there – if you are competing every morning to beat someone dear to you in Wordle – here are a couple of useful insights that may assist.

If you are only interested in the final strategy, skip to the last chapter. (Page 19)

Let’s begin.

Wordle

The game instructions are:



Guess the **WORDLE** in six tries.

Each guess must be a valid five-letter word. Hit the enter button to submit.

After each guess, the color of the tiles will change to show how close your guess was to the word.

Examples

W E A R Y

The letter **W** is in the word and in the correct spot.

P I L L S

The letter **I** is in the word but in the wrong spot.

V A G U E

The letter **U** is not in the word in any spot.

Wikipedia simply states:

"Players have six attempts to guess a five-letter word, with feedback given for each guess in the form of colored tiles indicating when letters match or occupy the correct position."

Yet, the simplest way to understand it, is by just playing once. It's free and addictive when you start sharing your results.

Purpose: Guess the daily word in the least number of attempts.

Scoring Words

To start, my focus was to find the most useful first 5 letter word. One which will cover the most possible groups of words depending on the information gained from the colouring of the fist attempt.

For example, **ARISE**, is thought to be a more useful first word than the word **NYMPH**. Why is that? Is it solely due to the three vowels vs. no vowels? I wanted to try and quantify the 'usefulness' of using any particular word at any stage of the game.

First, I needed a reliable data set of all the 5 letter dictionary words to score them by certain traits. Finding a data set of approximately 5,000 words made me think I am in good hands until the word **FORAY** came along in the game, and my data set did not contain it. Luckily another friend of mine, while we were chatting about the theoretical ability to write a code that would crack the game in 2 attempts, told me he had found a list of 5 letter words in the source code of the game. Web coding is not part of my skill set, although, knowing that there is an official list of possible words was enough to get me scraping through the unfamiliar code. Finding the list of only 2,309 possible words was a great step forward. (I assume to make the game suitable for being owned by the New York Times, some words were removed.)

My first method to quantify the usefulness of any word was to use a histogram for all the letters in the data set, then sum each word's letters by the value in the histogram to get a score.

This score was named **'Sum Score'**.

Figure 1 illustrates the histogram of the current data set.

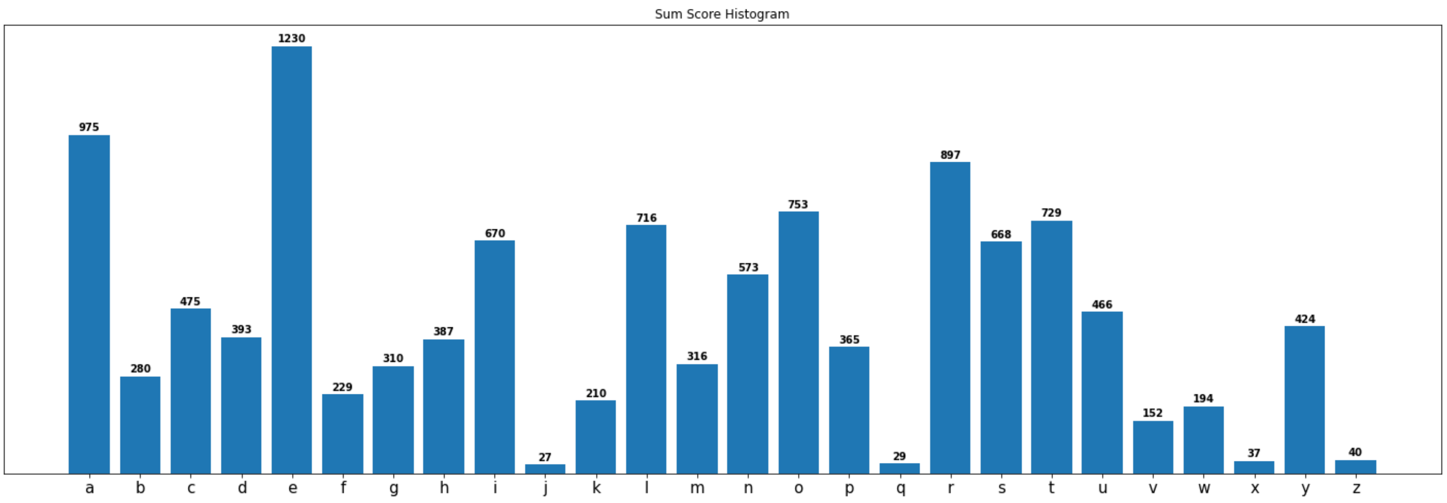


Figure 1. Sum Score Histogram, Example: 'a' appears 975 times in the data set

Based on the histogram above, **ARISE** would be given a 'Sum Score' of 4486,

(A + R + I + S + E = 975 + 897 + 716 + 668 + 1230 = 4486)

While **NYMPH'**s 'Sum Score' would be only 2065.

(N + Y + M + P + H = 573 + 424 + 316 + 365 + 387 = 2065)

The intuition is – the more a letter appears, the more information gained by using it in the guessed word. We all want the letters chosen to be in the final word, but the more 'useful' the word we chose, we gain also from being told the letter is **not** in the word – eliminating the maximum number of words possible for the next attempt.

Interesting Insight:

The letters 'I' and 'U', although vowels and thought to contain useful information, appear less than 'R' and 'T'.

‘Position Score’

The ‘Sum Score’ has its disadvantages. First, it would benefit words with double and triple high occurring letters as the word: **‘EERIE’.** Second, the information we are looking for is not solely the appearance of a letter in the word, we are also looking to place the letter in the correct position. Therefore, we should seek to identify the amount of times ‘A’ appears in each position of the 5-letter word and not only the total of ‘A’ appearing 975 times throughout the data set.

‘Position Score’ calculates 5 histograms, one for each position in the 5-letter word. The score is given to each word by the sum of the value of the letter by the corresponding histogram to its position.

See figures 2.1 – 2.5 with example below.

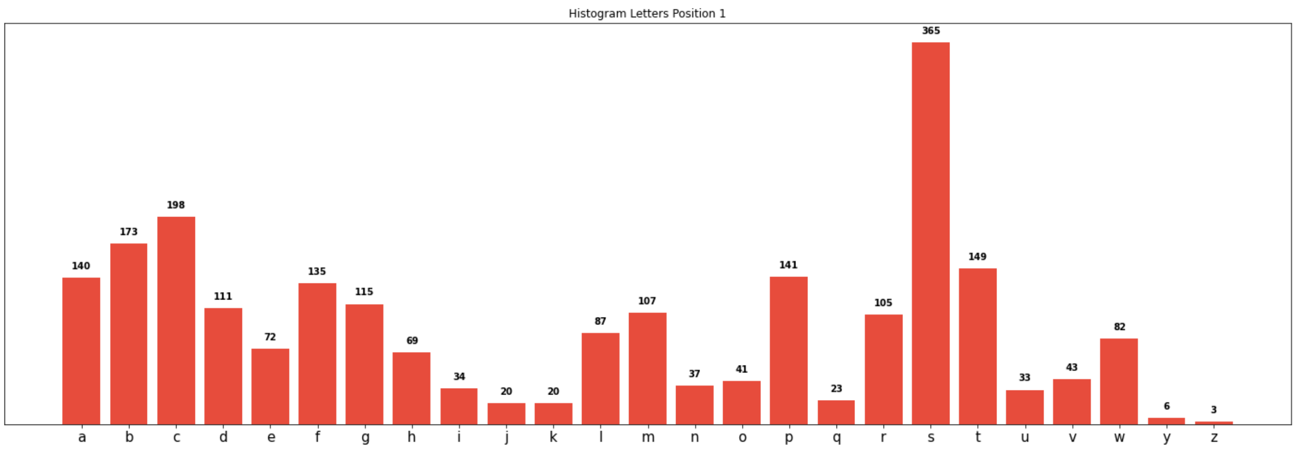


Figure 2.1 Histogram of the first letters in each word in the data set

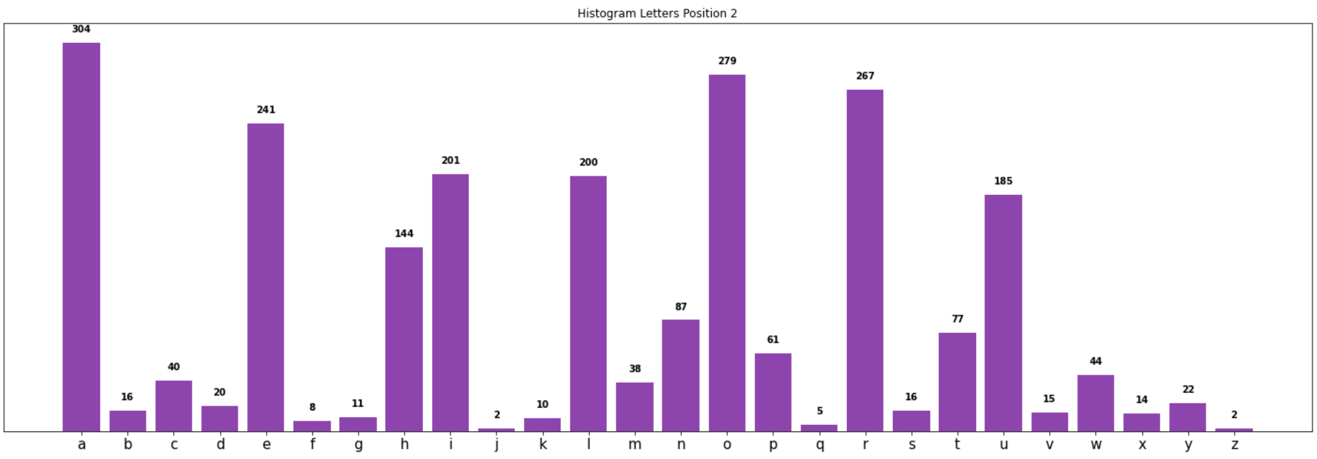


Figure 2.2 Histogram of the second letters in each word in the data set

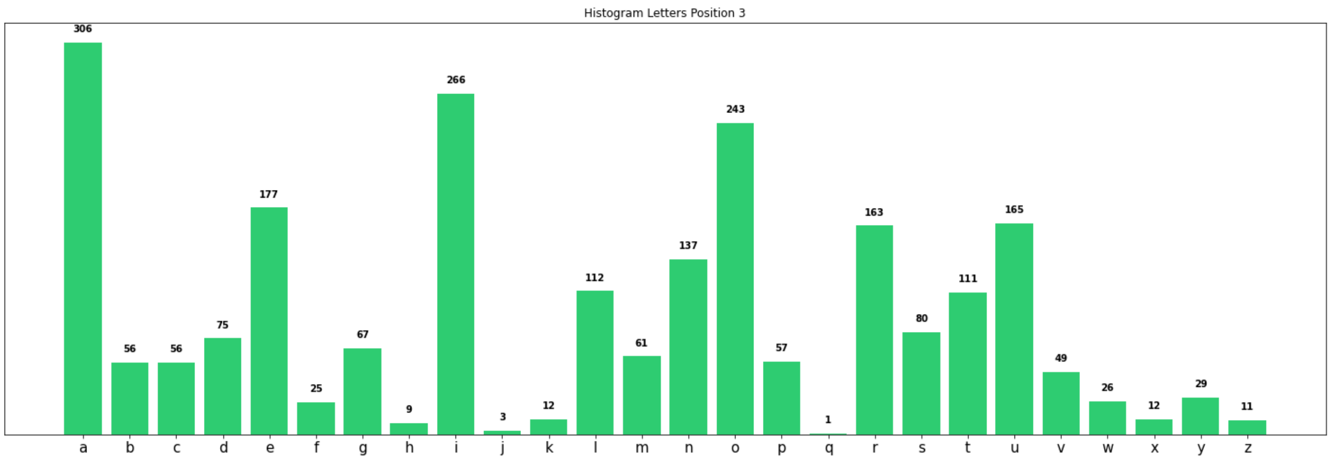


Figure 2.3 Histogram of the third letters in each word in the data set

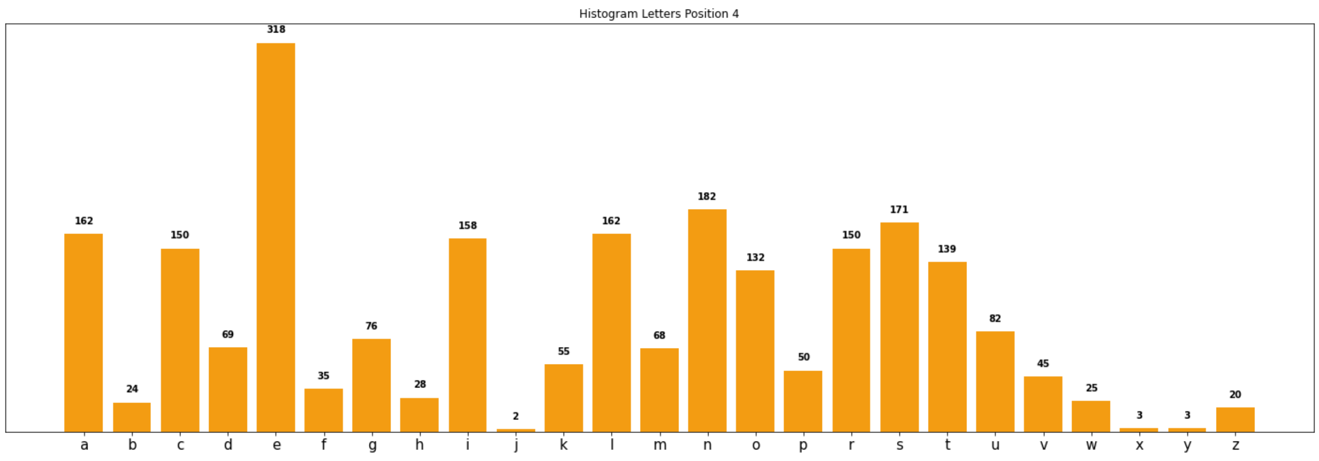


Figure 2.4 Histogram of the fourth letters in each word in the data set

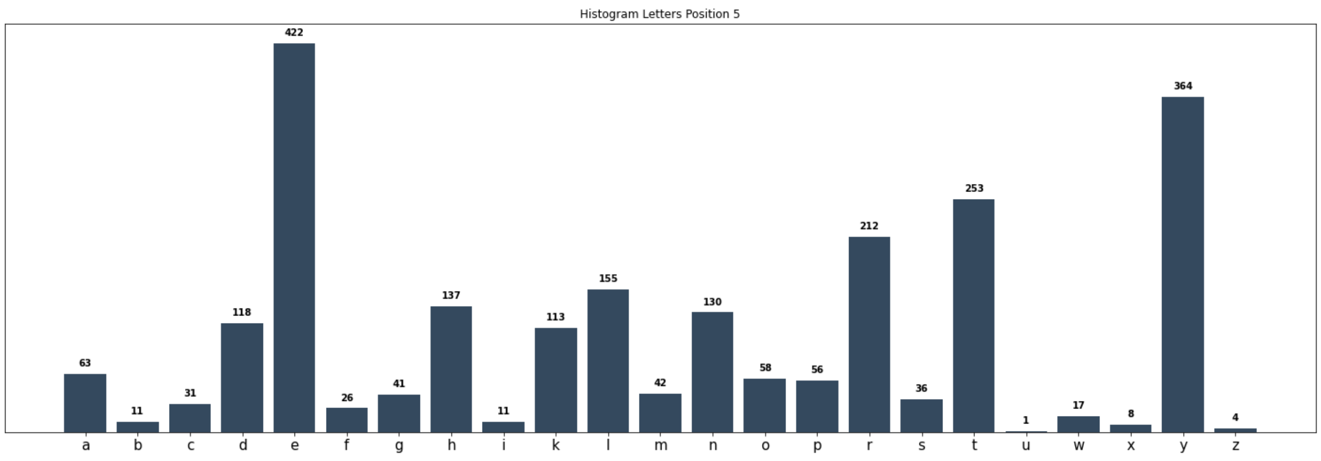
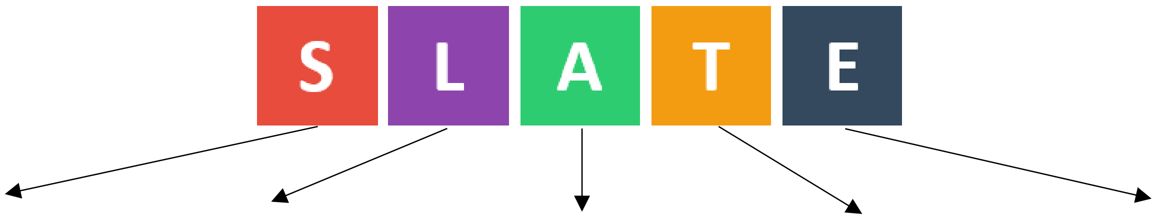


Figure 2.5 Histogram of the five letters in each word in the data set

Example:

SLATE

Position Score = 1432



‘S’ appears 365 times as the first letter throughout the data set.

See figure 2.1 above.

‘L’ appears 200 times as the second letter throughout the data set.

See figure 2.2 above.

‘A’ appears 306 times as the third letter throughout the data set.

See figure 2.3 above.

‘T’ appears 139 times as the fourth letter throughout the data set.

See figure 2.4 above.

‘E’ appears 422 times as the fifth letter throughout the data set.

See figure 2.5 above.

(S + L + A + T + E = 365 + 200 + 306 + 139 + 422 = 1432)

Interesting Insight:

SLATE is the highest ranking 'Position Score' for the first word in the data set. Meaning, if you decide to use the 'Position Score' scoring strategy, this is the best word to start with.

While playing every morning using the different scoring methods, I was finding the 'Position Score' was more useful than the 'Sum Score'. I started noticing that although a word had a higher score, my friends were still choosing the right word before me.

For instance, the word **BONEY** is ranked higher than **MONEY** by the 'Position Score' (they contain the same letters except for the first letter). Although most of you, with only the first letter missing, would most likely choose **MONEY** over **BONEY**. Why is that?

I believe it is the affect of the use of certain words in language. How many of us use the word **BONEY** in our daily speech?

How can we quantify how common a word is in a language (English)?

All I needed was a large enough data set of English text; extract all the 5 letter words from the text; and count the appearance of each of the 2,309 possible words within the text.

I had 2 options, either download free E-Books or use code which scrapes all Wikipedia. The Wikipedia option was a little overkill, so I decided I would use it if I find that this scoring method is superior to the others. I downloaded 30 E-Books and built a **'Book Appearance Score'**.

BONEY appeared **once** in the books (containing over 300,000 five letter words), while **MONEY** appeared **710** times.

The 'Book Appearance Score' brought with it a very interesting debate. Are the words in Wordle chosen at random each day, or is there some sort of human review and supervision of the daily words to come?

If the words were chosen randomly from all the possible words, the same probability is given to **BONEY** as to **MONEY.** This example was relevant only when the word was **MONEY**, but it was not the daily word because of it being more common in the language. On the other hand, if there was some sort of reasoning behind the word chosen, would we be able to say that there would not be a certain letter in the word today because the same letter appeared in yesterday's word? Is there some kind of characteristic to the words chosen? Caused by the bias of the people in charge approving the words for the game?

I was hoping to find some sort of pattern in the words of the game, even though those around me were sure the daily word was random from within the bag of words. Afterall, Alan Turing was able to break the Enigma by finding human patterns in the most complicated cypher at the time.

To further check if there is some connection between the chosen word each day, I decided to use a technique used in unsupervised machine learning models. Simply explained, each previous day Wordle contains information regarding the preference of words. By collecting the previous words and certain features of these words, we can rank the possible words remaining each day by the distance to the 'centroid' (centre of mass in the vector space created by these features). See figures 3.1 and 3.2 for example.

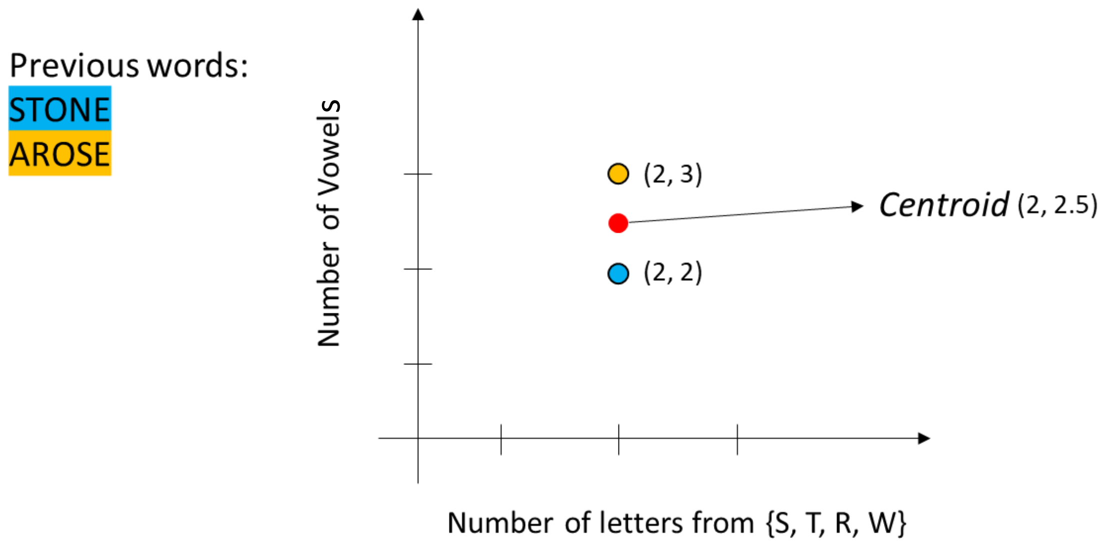


Figure 3.1 2-D representation of chosen features for 2 previous Wordles. Centroid calculated

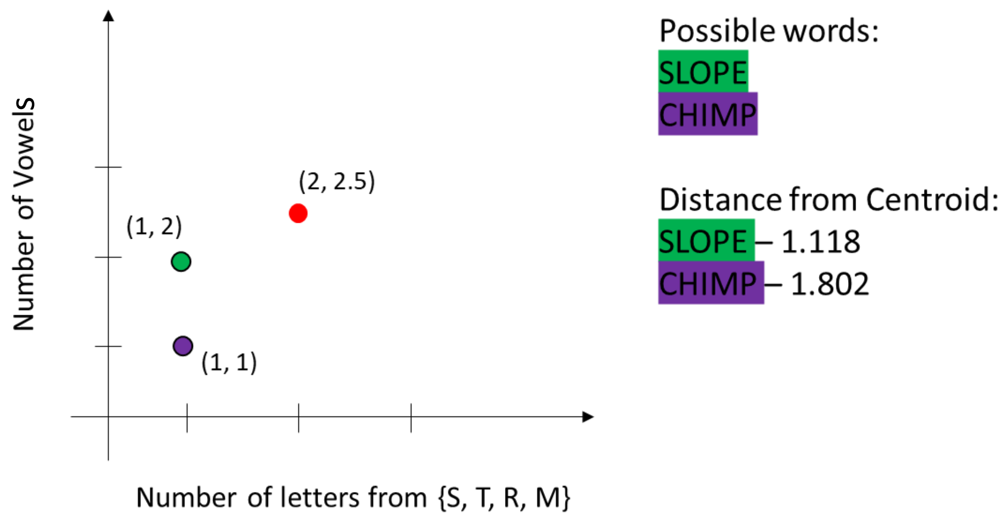


Figure 3.2, 'SLOPE' ranked higher because of the lower distance to the centroid of the previous words

Figures 3.1 and 3.2 are shown to simplify the method used. I chose to use the 3 scoring methods created until now as the 3 features for the representation of each word.

Example: **STORY,** represented by the vector (3471, 1199, 388) means the 'Sum Score' equals 3471, 'Position Score' equals 1199 and the 'Book Appearance Score' equals 388. Each day I logged the daily word and the centroid would be updated by the new word added to the previous words. (Note: this is the 'learning' part of machine learning models, which are known to perform better with the more data from which they can learn.) At each attempt the code calculated the distance from the remaining possible words' vectors to the centroid. The distance was named **'Distance from Prev Centroid Score'**, sorting the words by the closest to the furthest.

If the assumption is that there exists a connection between the words chosen each day, and that the combination of the scores I chose as the vector representation represent the words in a unique way, theoretically the closer a word is to the centroid represents the likelihood of the word being the word of the day.

‘Sum Score’ and ‘Position Sum Score’ Normalized

The next two scoring methods developed were based on a ‘normalization’ of the ‘Sum Score’ and the ‘Position Score’. By summing the value of each letter in a word we are exposing the score to the large gap created by very popular letters such as ‘A’ and ‘E’. Once this gap is too large, if the word contains one of these letters, no other word which does not contain these letters stands a chance when summing the letter values, even if all the letters are in the top 10 popular letters.

To avoid this flaw, the value given to each letter was calculated by subtracting from 26 the index of the letter after sorting the histogram. Figure 4.1 illustrates the ‘Sum Score Norm’ in a simple manner.

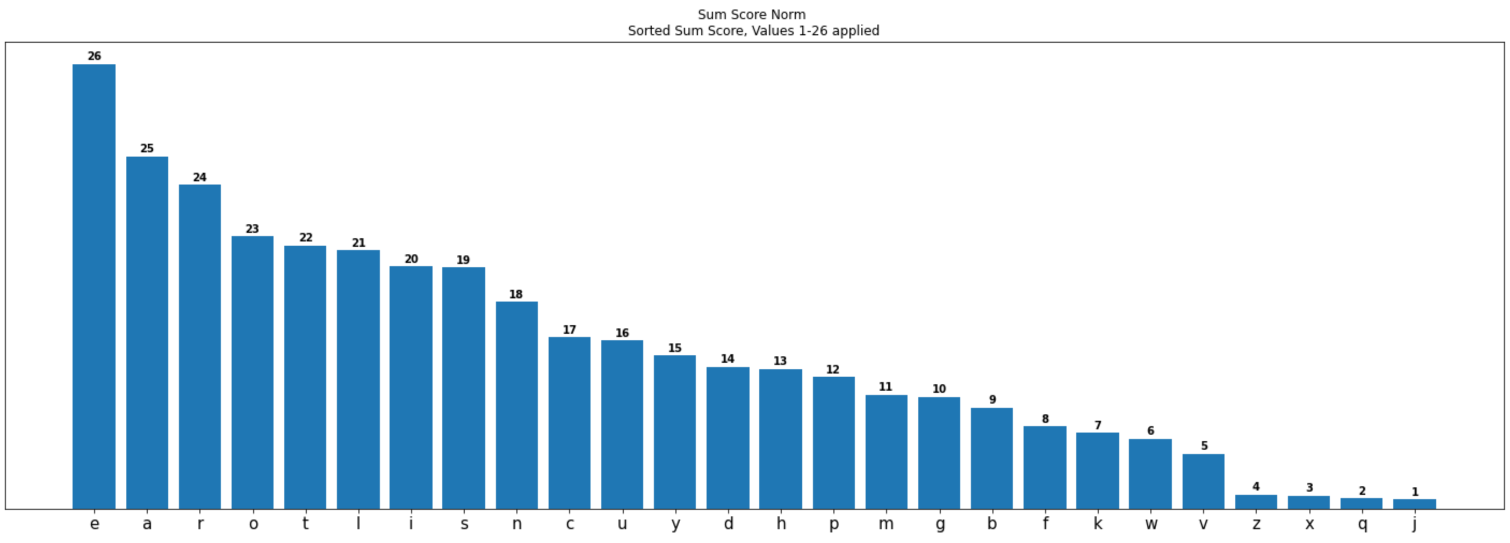


Figure 4.1, figure 1 sorted and the value given is based on the index of the sort

Example:

ARISE would be given a ‘Sum Score Norm’ of 114, (A + R + I + S + E = 25 + 24 + 20 + 19 + 26 = 114), While **NYMPH**’s ‘Sum Score Norm’ would be 69, (N + Y + M + P + H = 18 + 15 + 11 + 12 + 13 = 69). See how the gap between these scores has been reduced from approximately 2,400 points on page 5, to only 45 points here.

Interesting Insight:

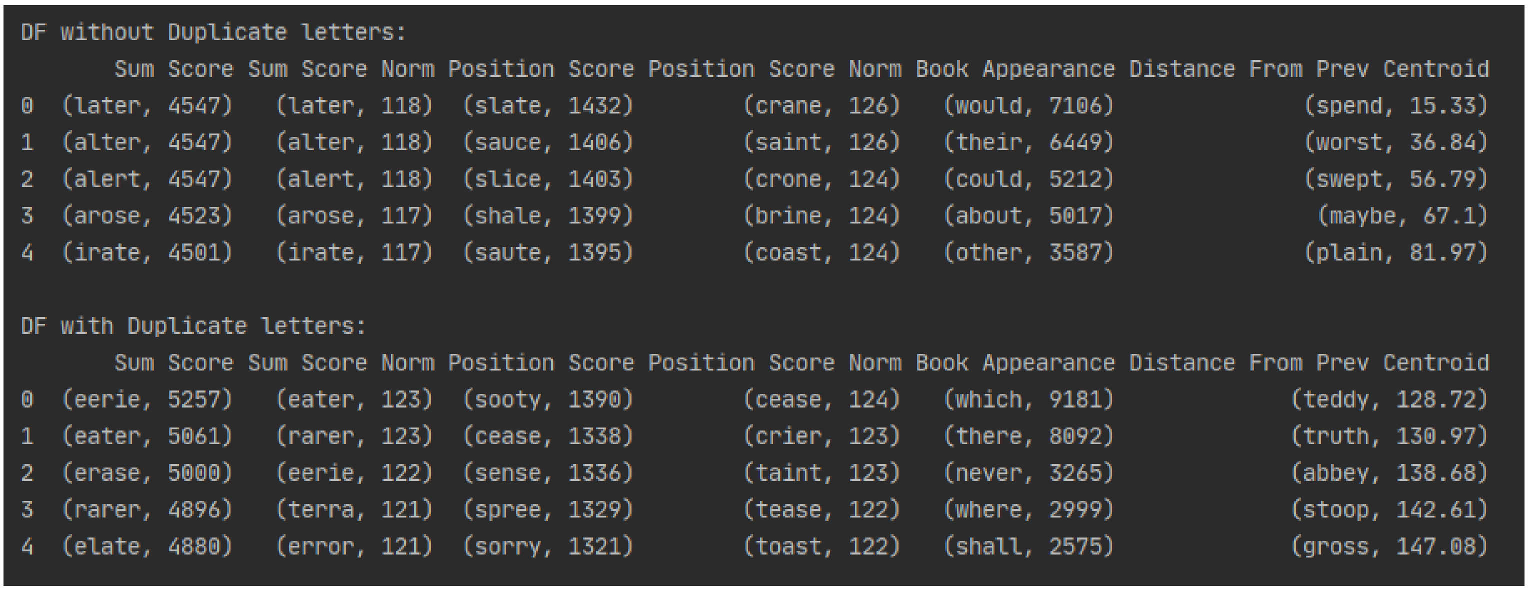
LATER, ALTER and **ALERT** are the highest ranking ‘Sum Score Norm’ for the first word in the data set. They score 118 points from a maximum possible 130 points (26 x 5 = 130).

To achieve the 'Position Score Norm', the same process was performed on each of the five histograms created for the 'Position Score' as explained above regarding the 'Sum Score'.

Interesting Insight:

CRANE and **SAINT** are the highest ranking 'Position Score Norm' for the first word in the data set. They score 126 points from a maximum possible 130 points.

The output of the code after each attempt (after inputting the output of the game information from each attempt) would be:



```
DF without Duplicate letters:
       Sum Score Sum Score Norm Position Score Position Score Norm Book Appearance Distance From Prev Centroid
0  (later, 4547)   (later, 118)  (slate, 1432)        (crane, 126)   (would, 7106)              (spend, 15.33)
1  (alter, 4547)   (alter, 118)  (sauce, 1406)        (saint, 126)   (their, 6449)              (worst, 36.84)
2  (alert, 4547)   (alert, 118)  (slice, 1403)        (crone, 124)   (could, 5212)              (swept, 56.79)
3  (arose, 4523)   (arose, 117)  (shale, 1399)        (brine, 124)   (about, 5017)               (maybe, 67.1)
4  (irate, 4501)   (irate, 117)  (saute, 1395)        (coast, 124)   (other, 3587)              (plain, 81.97)

DF with Duplicate letters:
       Sum Score Sum Score Norm Position Score Position Score Norm Book Appearance Distance From Prev Centroid
0  (eerie, 5257)   (eater, 123)  (sooty, 1390)        (cease, 124)   (which, 9181)             (teddy, 128.72)
1  (eater, 5061)   (rarer, 123)  (cease, 1338)        (crier, 123)   (there, 8092)             (truth, 130.97)
2  (erase, 5000)   (eerie, 122)  (sense, 1336)        (taint, 123)   (never, 3265)             (abbey, 138.68)
3  (rarer, 4896)   (terra, 121)  (spree, 1329)        (tease, 122)   (where, 2999)             (stoop, 142.61)
4  (elate, 4880)   (error, 121)  (sorry, 1321)        (toast, 122)   (shall, 2575)             (gross, 147.08)
```

Presenting the top 5 rows from each Data Frame (DF), the top DF is the words not containing any duplicate letter, and the bottom DF is the words with duplicate letters.

Each Column is the possible words left to choose from, sorted by the column name scoring method as described above.

Strategy 1

Now that I have defined 6 different scoring methods, each usually recommends a different word. Which one do I choose?

At first, this was based on intuition. Remember, it was just my ‘morning coffee, trying to beat my friends in a game’ project.

So, I was happy using the ‘Position Score’ for the first attempt, sometimes also the second if the same word would be recommended by a few scores and for the third and fourth attempt I was looking to use the ‘Book Appearance Score’. This was since the other scoring methods were recommending very ‘general’ words (like **WOULD, COULD…**).

As a numbers guy intuition was not going to be enough to convince me that I am playing the best strategy. I started seeing news headings such as:

“Canberra, Australia, is the best city in the world at Wordle, solving the daily puzzle in an average of 3.58.”

“Sweden is the world’s best country in Wordle, with an average score of 3.72.”

At first, before starting to write the code to crack the game, I was thinking about the possibility of consistently winning the game in 2 attempts. Later I understood this was not possible, due to the number of words in the data set. So, is consistently winning in 3 attempts possible?

My friends were pretty good at hitting within 3 to 4 attempts, with an average of 3.87 with approximately 3 unsuccessful days (not guessing the word after 6 attempts).

If Canberra and Sweden are considered best in the world with 3.58 and 3.72 as their overall average, surely I should use this as a benchmark to beat.

I decided to code a simulation of the game, input a strategy, and run it using all the possible words in the data set, outputting the true distribution of winnings for each strategy.

When I say strategy, I mean a sequence of scoring methods by which the code will choose the next guessing word. Is it by the ‘Sum Score’ or ‘Book Appearance’ method? If there are 6 possible attempts, so we need to give the simulation the instruction as to which scoring method to choose on each possible attempt.

For each possible strategy the code will calculate the average score, and will look to minimize the average score.

In a fancy equation notation: (for some, this makes more sense than words…)

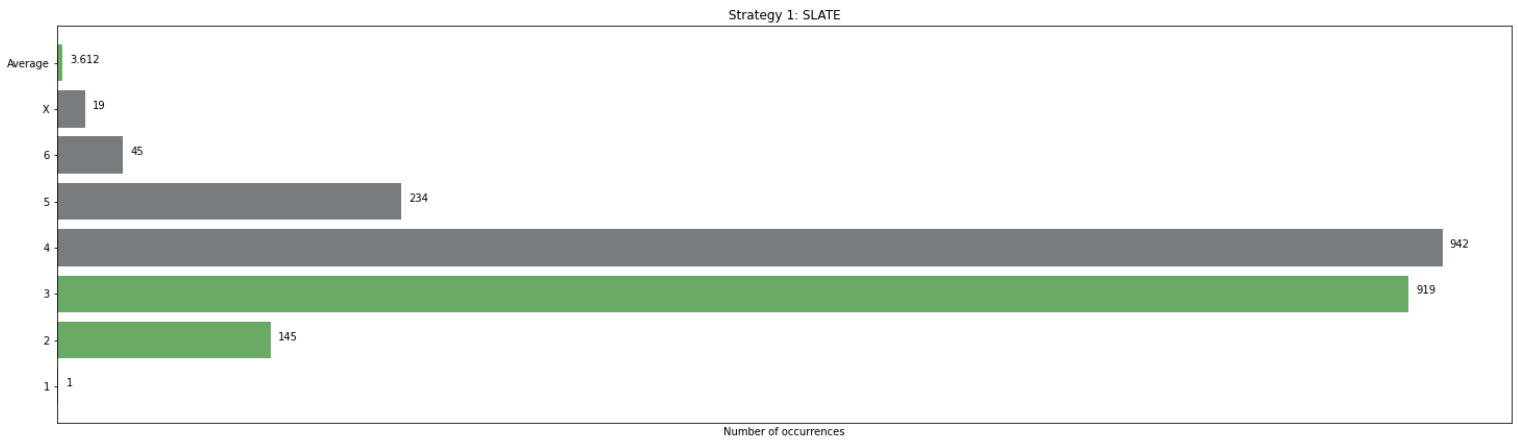
$$\underset{Strategy}{\operatorname{argmin}}\{\, E(number\ of\ attempts\ to\ win)\ \}$$

By trying to solve the problem straight forward, we have 6^6 (i.e. 46,656) possible strategies. Each strategy was taking approximately 25 seconds to compute. Each was running through 2309 words and playing the game, for each round the code calculates the scores and sorts 6 columns. This would cost 325 hours, meaning 13.5 days and nights running constantly.

To reduce the computational time, I assumed that to find the optimal strategy with a chance of scoring an average below 4, would require the best strategy over the first 3 steps in the game. Therefore, I should look for the optimal prefix of the strategy from 6^3 (i.e. 216) strategies and once found, use the prefix to find the optimal suffix of the optimal strategy. This would only cost: 216 x 25 / 60 / 60 = 1.5 hours for prefix and an additional 1.5 hours for the suffix.

The result was clear, using only the 'Position Score' was resulting in the lowest overall average. Achieving a 3.612 average.

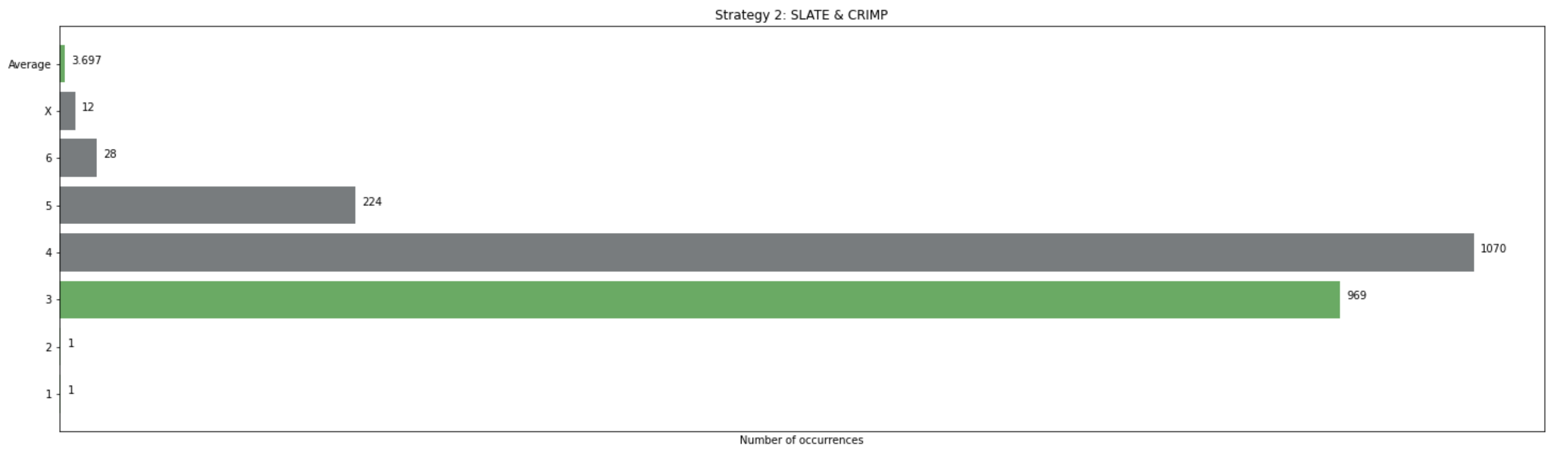
I was not happy with this result. I was sure that there was more to be found and was sure that it is possible to achieve a better outcome.



Strategy 2

After brainstorming with a good friend, we came up with the idea of finding a second word, which does not contain any of the letters in the first word chosen, but used together would ensure that we would win more consistently on the third attempt. This was counter-intuitive, due to the nature of the game. Each attempt is too important to decide not to try and choose a word which might be able to win. We were essentially giving up on winning on the second attempt, to increase our chances of winning on the third attempt.

To find such a word, we ran the simulation at first as if the first word attempt (‘SLATE’) yielded only black tiles. We took the top scored 200 words by the ‘Position Score’ and ran the simulation on all the words available – as if we were forcing the first word to be ‘SLATE’ and the second word to be one of the 200 words. The second word yielding the best winning average was the word ‘CRIMP’, with a score of 3.697. Meaning, by using the combination of ‘SLATE’ and ‘CRIMP’ and trying to maximize winning in 3 attempts, we were still achieving inferior results in comparison with using ‘SLATE’ alone as shown in Strategy 1 above. Not the result we were expecting.



Strategy 3

Each strategy above has its advantages and disadvantages, and this brought me to think of using the upside from each strategy – based on the output of the coloured tiles from the first attempt.

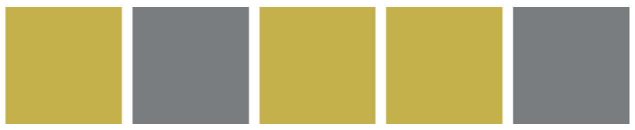
For instance: using Strategy 1, we would win on the second attempt 145 times. These must have something in common based on the information output from the first attempt – when we recognize this common characteristic, we would prefer to choose Strategy 1 over Strategy 2. On the other hand, for the common characteristics of winning on the 4th, 5th or 6th attempt using Strategy 1, we might find it better using Strategy 2 in these cases.

The way I decided to put this theory to test was:

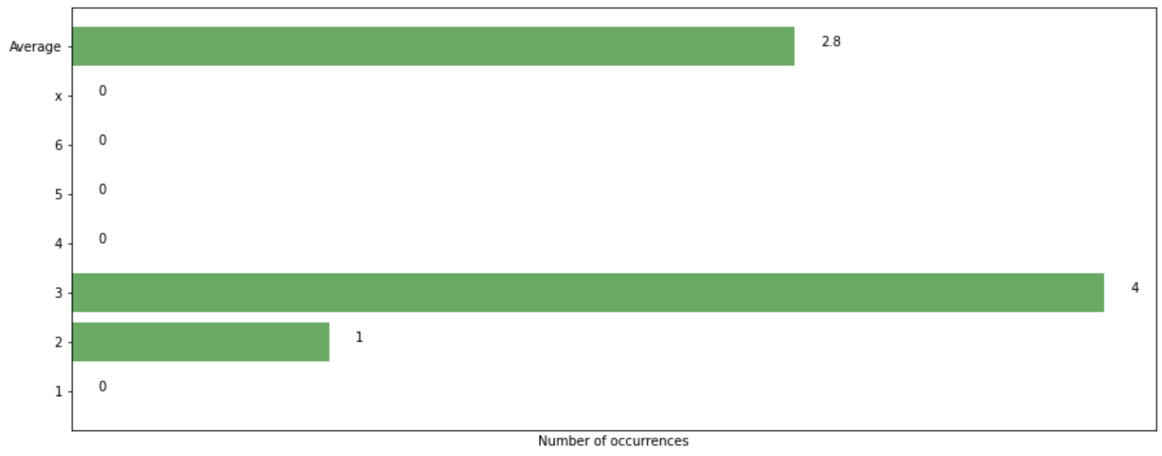
For each unique colouring of the tiles in the first attempt, I created a result profile.

Example:

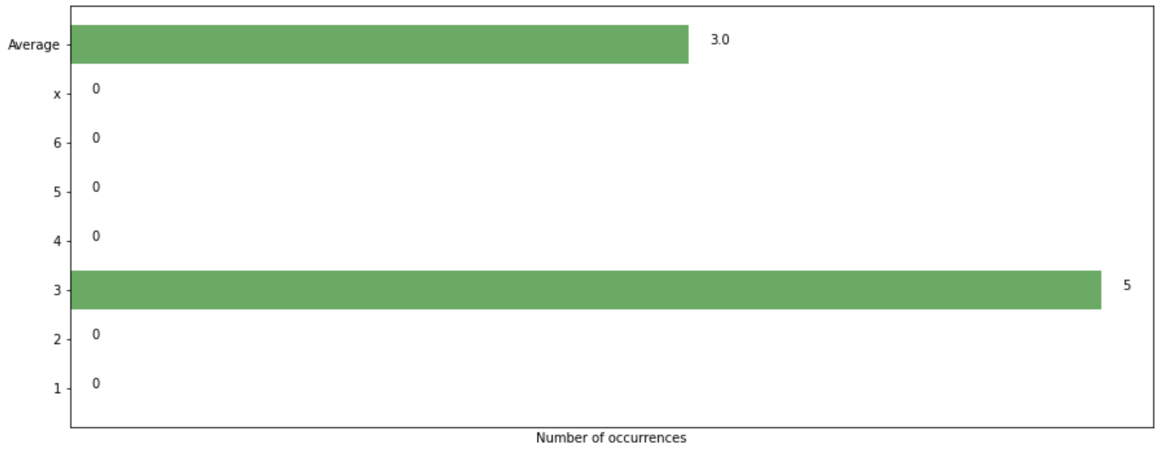
Using Strategy 1, the following unique colouring:



Yielded the result profile:



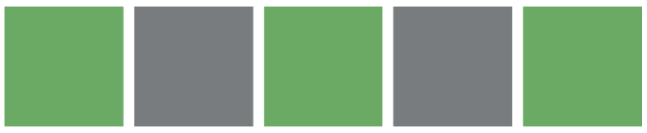
For the same unique colouring, using Strategy 2, the result profile yielded is:



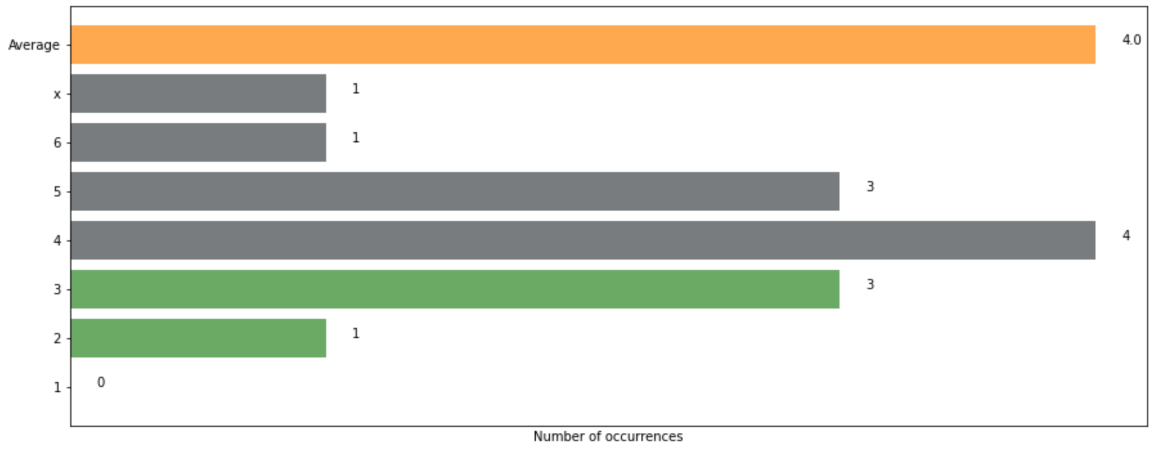
It is clear that we would prefer Strategy 1 in this situation, due to a lower average.

Another example showing Strategy 2 being the superior strategy.

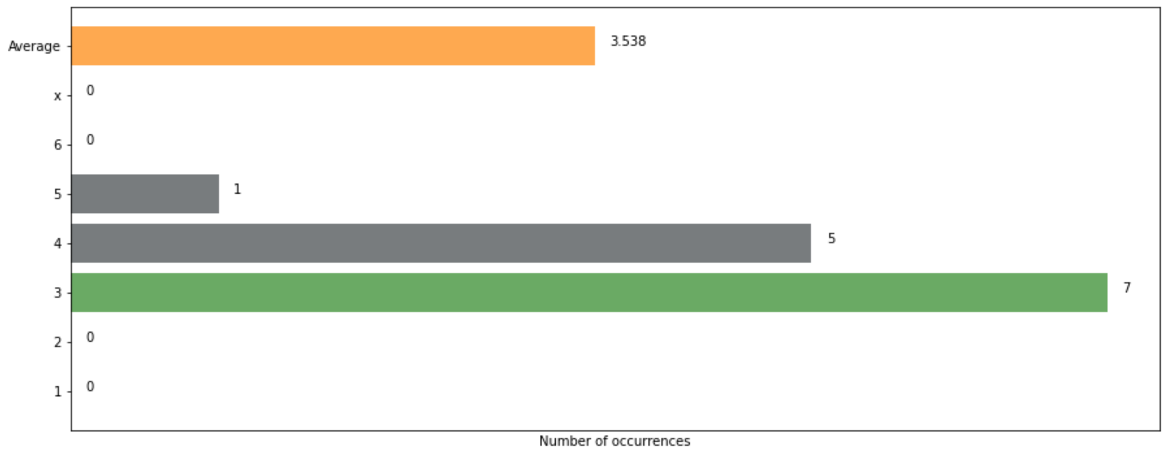
For the following colouring:



Strategy 1 result profile:



While Strategy 2 result profile:



In this case, Strategy 2 results in a much better result than Strategy 1!

The results from the unique tile colouring showed that Strategy 3 would have an advantage over the previous strategies.

How much of an advantage?

To find out, as shown in the example above, I simply compared each unique first attempt colour output between the two strategies. The strategy resulting in the lowest average for the unique output was the strategy chosen for the second attempt.

Coding Strategy 3 in the simulation resulted in a final average of: 3.579

This was amusing, due to the fact I presented on page 13,

“Canberra, Australia, is the best city in the world at Wordle, solving the daily puzzle in an average of 3.58.”

While using this benchmark to beat, even though I did not beat it by much,

I decided it was time to start wrapping up the project…

*It is important to point out, I was comparing my strategy on the overall average of solving the game over all the possible words with the **daily** average of Canberra… Let’s see if they will be able to maintain their awesome result and maybe even beat Strategy 3’s results.*

Strategy 4

After much thought about the combination of strategies used in Strategy 3, I thought about how to ‘overfit’ the predictions on the list of words. For those unfamiliar with the term, ‘overfitting’ is used to characterise a machine learning (ML) model, which is fitted to predict very low error rates on a training set, causing it to perform poorly on new data, the test set, which was not seen in the training stage. In ML we would not want to overfit a model because the purpose of the model is to predict unseen data. In Wordle, where the list of words from which the word can be selected is known, there are no new words. Therefore, if we could reach a 0-error rate on the given list, we would be satisfied.

How could we overfit our predictions?

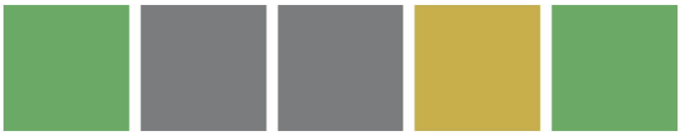
The information gained from the colouring of the tiles after the first guess can be theoretically one in 3^5 (i.e. 243) options. In our case, using ‘SLATE’ as the first guess, there are only 145 unique colouring sequences. If we would find the list of words from all the words for each colouring, and based on these words find the most useful second word, tailored to minimise the steps to win, we would probably be better off than choosing a single second word as we did in Strategy 2.

For every unique output from the first guess, we would find the optimal second word, based on the words which fit the unique colouring.

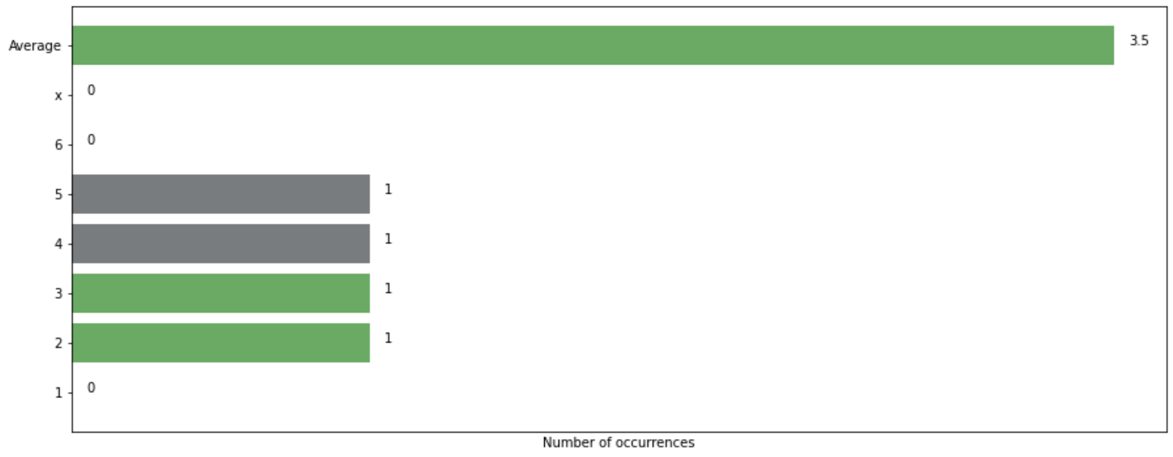
I took all the colourings which resulted in an average score of more than 3, of which there were 70, and I started manually looking into the process needed to find the optimal second word.

For instance:

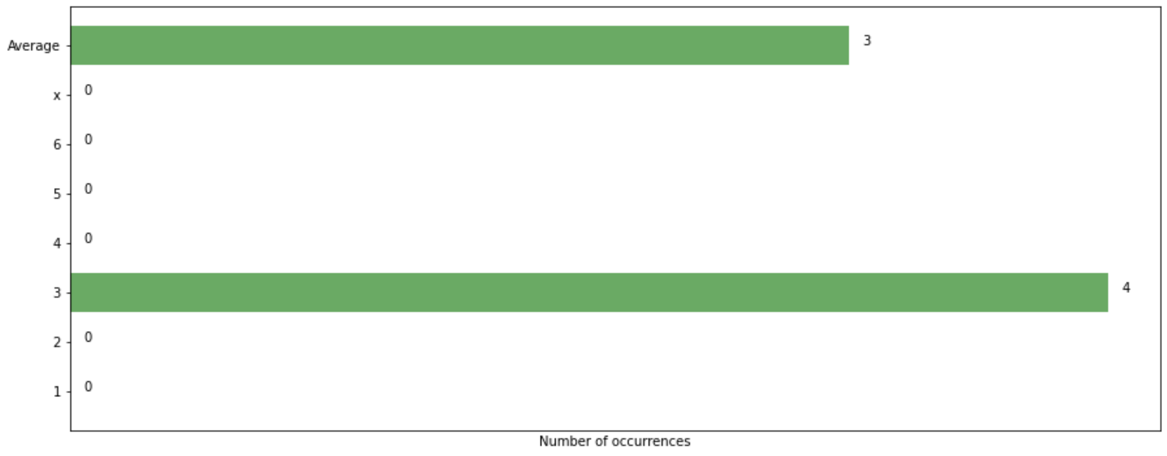
The unique colouring:



Fitted the result profile:



The four words fitting this sequence are: store, stove, stoke, stone. By using the word 'krunk' we can achieve the result profile:



Reducing the score for the unique colouring from 3.5 to a solid 3.

The reason this is possible, is because we are trying to create a tailored response to each of the 70 colourings possible from the first attempt.

After comparing all 70 colourings with the optimal second word found, 55 out of the 70 were achieving better scores than prior to the tailored approach.

Running the simulation on all the words with this new approach resulted in an overall score of 3.5.

I believe there is more to be discovered and many more ways to overfit our choices to improve the overall score. But, at this point, I decided to wrap up the project and enjoy my morning coffee with the insights found. Or…maybe even dive into a new challenge.

Wrap up

Pretty awesome to see how far a little curiosity can take us.

By thinking about a simple theoretical question, “Is it possible to consistently win within 2 attempts?”, to wanting to win every day’s friendly challenge, through to wanting to create a strategy which achieves better results than the “Best city in the world at Wordle”.

I am sure there are many more scoring metrics, which probably lead to better strategies than those I explained throughout this paper, and I hope to come across them one day.

If any of you reading this have an idea of such a strategy, I would love to chat about it.

Uploaded code to https://github.com/Jacob-Link under the ‘Wordle’ directory.

The code is built to recommend the next word while playing Wordle, and implements the final strategy explained. For more information about how to run the code – read the READ-ME file in the GitHub link under the ‘Wordle’ directory.

Until the next challenge,

August 2022,

Jacob Link