Using Linear Programing Techniques to Determine The Next Best Wordle Guess

David Ricci
Aidan Dilsavor
Nathan Johnson

ISE 3230 Section 10198

November 29th, 2023

Introduction/ Motivation:
Wordle is a popular game online in which the player has to guess a hidden 5-letter word in 6 guesses. 3Blue1Brown, a famous math youtuber, created a solver for the game and created a video on his findings. In our project, we are hoping to do something similar using linear optimization techniques discussed in class. To understand how it works, we would recommend playing the game.

Problem Formulation:
We used a data set consisting of valid wordle guesses, and all possible wordle solutions (provided on kaggle). Using these data sets, we created a table of frequency values.

(A Frequency Value is the number of times a letter appears in each valid solution or guess at a particular position. For example, the letter “h” appears 402 times as the second letter.)

When playing wordle, there are three outcomes when a letter is typed in:
Green - The letter is in the word and in the right position.
Yellow - The letter is in the word, but in the wrong position.
Grey - The letter is not in the word.

We will use the green, yellow, and grey letters after submitting a guess to constrain/ narrow down our dataset, and determine the set of possible “next best guesses”. Below is a description of how we constrained our dataset:

Green:
If there is a green A in the first position, we will constrain our dataset to only include words where A is in the first position.

Yellow:
If there is a yellow B in the second position, the possible set of words must include a B. However, that B must not be in the second position. Therefore, we will eliminate words that have a B in the second position, and increase the frequency scores of words that have more yellow letters in it by multiplying the frequency score of each word by a multiplier value. The multiplier value will increase if there are multiple yellow values present in the word.

Grey:
If there is a grey C in the third position, then we can take the data set and remove any words that have the letter C in them. So if a letter is grey, then any word with that letter is removed. (An exception is made if there is a green value and gray or green value and

yellow value value for the same letter- in which case the gray value will be filtered out of the appropriate positions)

Yellow ordering:
We now have words that could be used for the next guess. Our goal now is to choose the best words. To do this, we should choose a word with the most number of letters guessed in it already. So whichever words have the most number of yellows will have their frequency scores weighed higher. For example:

- If a word has 0 yellow values, the frequency score will not change
- If a word has 1 yellow value, the frequency score will be multiplied by 10
- If a word has 2 yellow values, the frequency score will be multiplied by 20
- So on and so forth…

Objective function:
To choose the best word from our new data frame that went through all the constraints above, the linear program finds the maximum frequency score given all of the constraints placed on the dataset. Frequency score is the amount of times each letter in their position appears in the data set summed up.

For example, A in the first position appears in the data set 596 times. So it has a frequency score of 596. An entire word's frequency score would be the sum of the letters in each of their respective positions (f[1] + f[2] + f[3] + f[4] + f[5])

Decision Variables:

F_i: The non-negative integer frequency scores associated with the letters of our guess
Example: F[1] = Frequency score of the letter at position 1 for any word

J_z: The decision to choose the word with the highest score
J[1] + J[2] + …. + J[Length of Constrained Data Set] = 1
(= 1 because we are only picking one word)

i = 1, 2, 3, 4, 5
j = 0, 1
Z = 1, 2, ….., n
n = (the number of viable guesses in the constrained dataset)

Constraints:

f[1] <= Valid_Guesses[1] * 10^(# of yellows) + ((1-j[z]) * M)
f[1] >= Valid_Guesses[1] * 10^(# of yellows) + ((1-j[z]) * M)
f[2] <= Valid_Guesses[2] * 10^(# of yellows) + ((1-j[z]) * M)
f[2] >= Valid_Guesses[2] * 10^(# of yellows) + ((1-j[z]) * M)
f[3] <= Valid_Guesses[3] * 10^(# of yellows) + ((1-j[z]) * M)
f[3] >= Valid_Guesses[3] * 10^(# of yellows) + ((1-j[z]) * M)
f[4] <= Valid_Guesses[4] * 10^(# of yellows) + ((1-j[z]) * M)
f[4] >= Valid_Guesses[4] * 10^(# of yellows) + ((1-j[z]) * M)
f[5] <= Valid_Guesses[5] * 10^(# of yellows) + ((1-j[z]) * M)
f[5] <= Valid_Guesses[5] * 10^(# of yellows) + ((1-j[z]) * M)
(Frequency score at a given position is the word's frequency score, multiplied by 10^(# of yellows) if the word has yellow letters in it)

J[1] + j[2] + ….. + j[n] = 1
(Must choose only one word as our guess)

F[1] …. F[5] >= 0
F[1] … f[5] a valid integer
(Nonnegativity constraint, integer constraint)

Objective Function: Max f[1] + f[2] + f[3] + f[4] + f[5]

Valid_Guesses:
Valid_Guesses refers to a subsetted data frame. Valid_Guesses begins as a data frame consisting of a word (which is a valid guess or solution), and the associated frequency score for each letter. This data frame was then narrowed down to account for green and grey letters. The decision was made to implement green and grey constraints outside of CVXPY to reduce the number of constraints added to the "constraints" array (and thus, significantly decrease the program runtime).

Results:
In all cases tested thus far, a solution to the Wordle puzzle has been achieved in 6 or less guesses using the "Ideal Guess" that the solver recommends.

Note that an initial guess must be provided by the user and imported into Wordle first. Only then can the program be used. Additionally, it is recommended to restart the kernel/ clear outputs after each time the program is run for ease of use.

Additionally, after Wordle comes up with an initial guess, the user should import the guess into Wordle, get the result, and run the program again. This time, when prompted “How many words have you previously guessed”, the user should tell the program how many previous words they have guessed.

Finally, please note that users may experience a delay in getting an “ideal guess” from the program on the initial guesses. This is because the dataset of possible words to guess (consisting of over 10,000 words) will not be significantly narrowed down, and thus, the solver will need to examine a significant number of words to find the optimal frequency score.

(In the extreme case that an optimal frequency score cannot be reached on the first try because of a tie in the frequency scores (the program would show no ideal guess), guess another word in the same Wordle. Run the program again, and import both words into the program. This will solve the problem and create an ideal guess. Out of the many cases we tested, only one has produced this issue, and was solved by importing another word).

Although our program provides the highest frequency score as the “ideal guess” that the solver recommends, it also provides a list of all other possible solutions given the constraints we imported. Additionally, this list also notes which valid guesses are also valid solutions. The program will typically narrow down this list to 4-10 possible solutions after multiple guesses. Picking the ideal word from here can prove difficult.

Initial consideration was given to figuring out a way to calculate distance between each letter of the word in the recommended dataset, and the recommended guess from the solver. However, our group decided against this, as we concluded that providing distance between each letter would not provide a substantial advantage in guessing the next word. Additionally, consideration was given to picking the word with the highest frequency score regardless of yellows (but we again concluded that it would not provide a significant advantage in picking the final word).

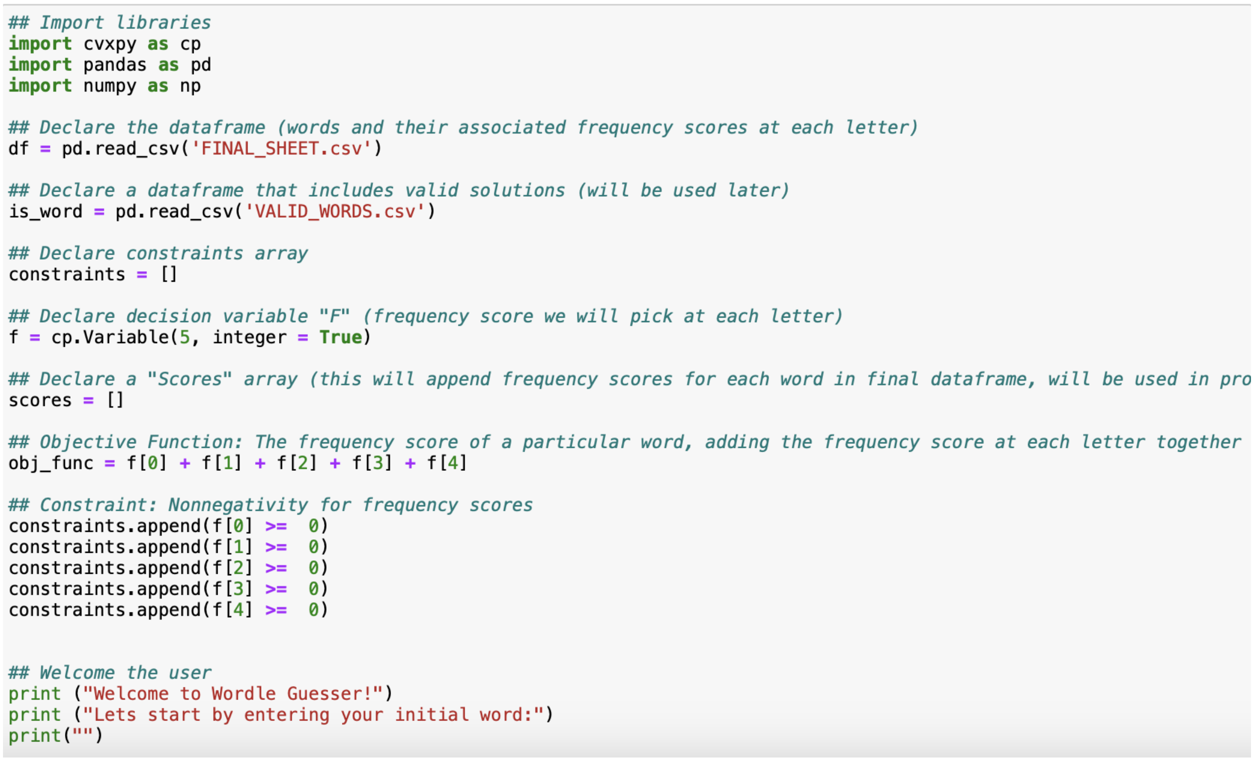
Therefore, when using this program, our group recommends two ideas when a small subset of words are left:

1) Observe the list of potential solutions when there are only a few left, and logically reason to think what the next “best guess” may be. Mathematically differentiating between only a few extremely similar words will likely provide the same benefit as picking a word randomly/ guessing by inspection.

2) Run the code snip at the bottom of the project file titled “Yellow Elimination”. This will eliminate all words that do not have the “yellow” letter present, and eliminate all words that are not valid solutions but are valid guesses, therefore making the guessing significantly easier. Only do this step when there are only a few words left and picking a word randomly seems beneficial (as running this step earlier will eliminate the goal of optimizing words that have a high count of yellow letters)

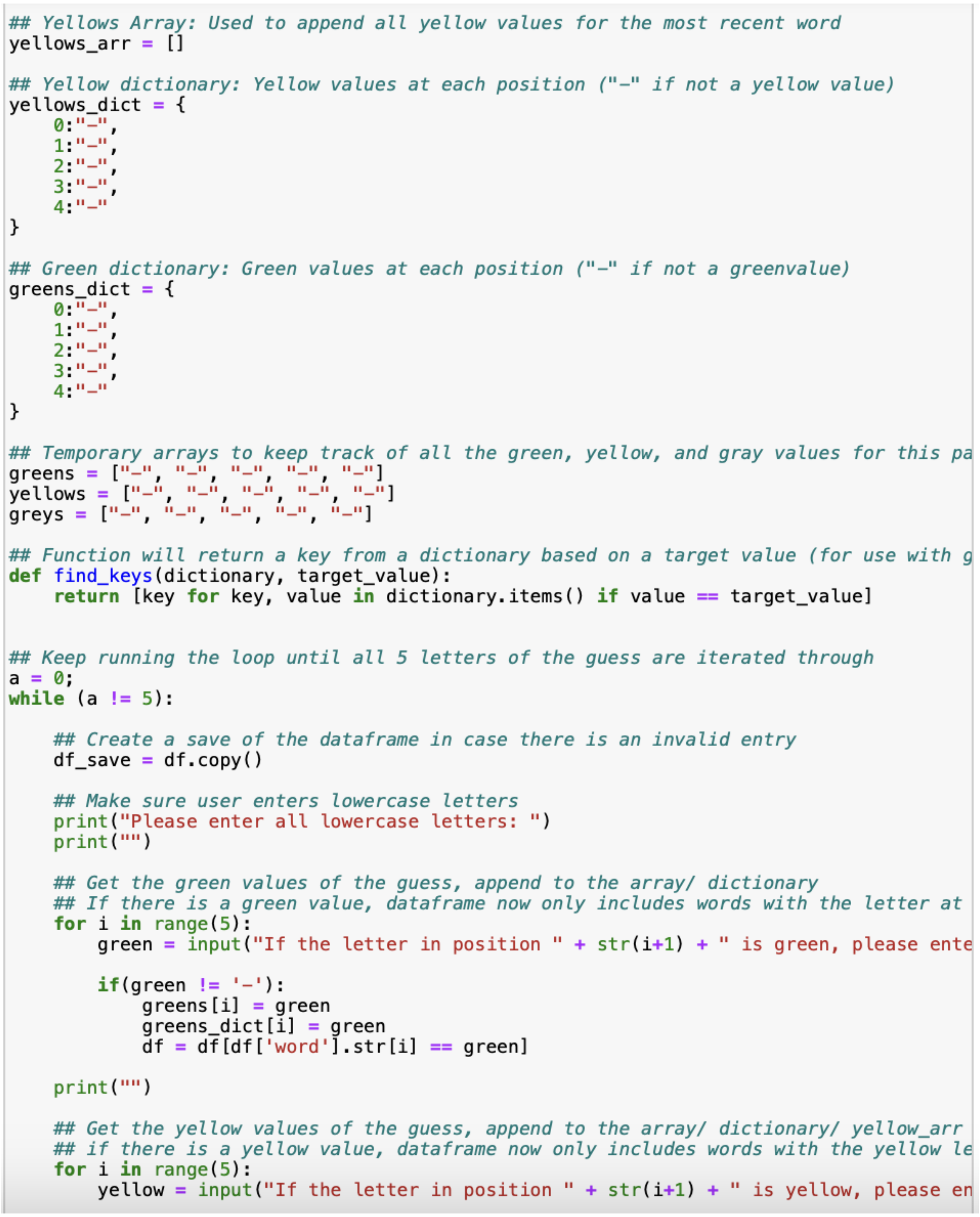
Code Analysis:
(Specific descriptions of what each line/ section does can be found in the code comments. General descriptions with corresponding images of sections can be found below):

1) Declare Objective Function/ Frequency Variables



```
## Import libraries
import cvxpy as cp
import pandas as pd
import numpy as np

## Declare the dataframe (words and their associated frequency scores at each letter)
df = pd.read_csv('FINAL_SHEET.csv')

## Declare a dataframe that includes valid solutions (will be used later)
is_word = pd.read_csv('VALID_WORDS.csv')

## Declare constraints array
constraints = []

## Declare decision variable "F" (frequency score we will pick at each letter)
f = cp.Variable(5, integer = True)

## Declare a "Scores" array (this will append frequency scores for each word in final dataframe, will be used in pro
scores = []

## Objective Function: The frequency score of a particular word, adding the frequency score at each letter together
obj_func = f[0] + f[1] + f[2] + f[3] + f[4]

## Constraint: Nonnegativity for frequency scores
constraints.append(f[0] >=  0)
constraints.append(f[1] >=  0)
constraints.append(f[2] >=  0)
constraints.append(f[3] >=  0)
constraints.append(f[4] >=  0)


## Welcome the user
print ("Welcome to Wordle Guesser!")
print ("Lets start by entering your initial word:")
print("")
```

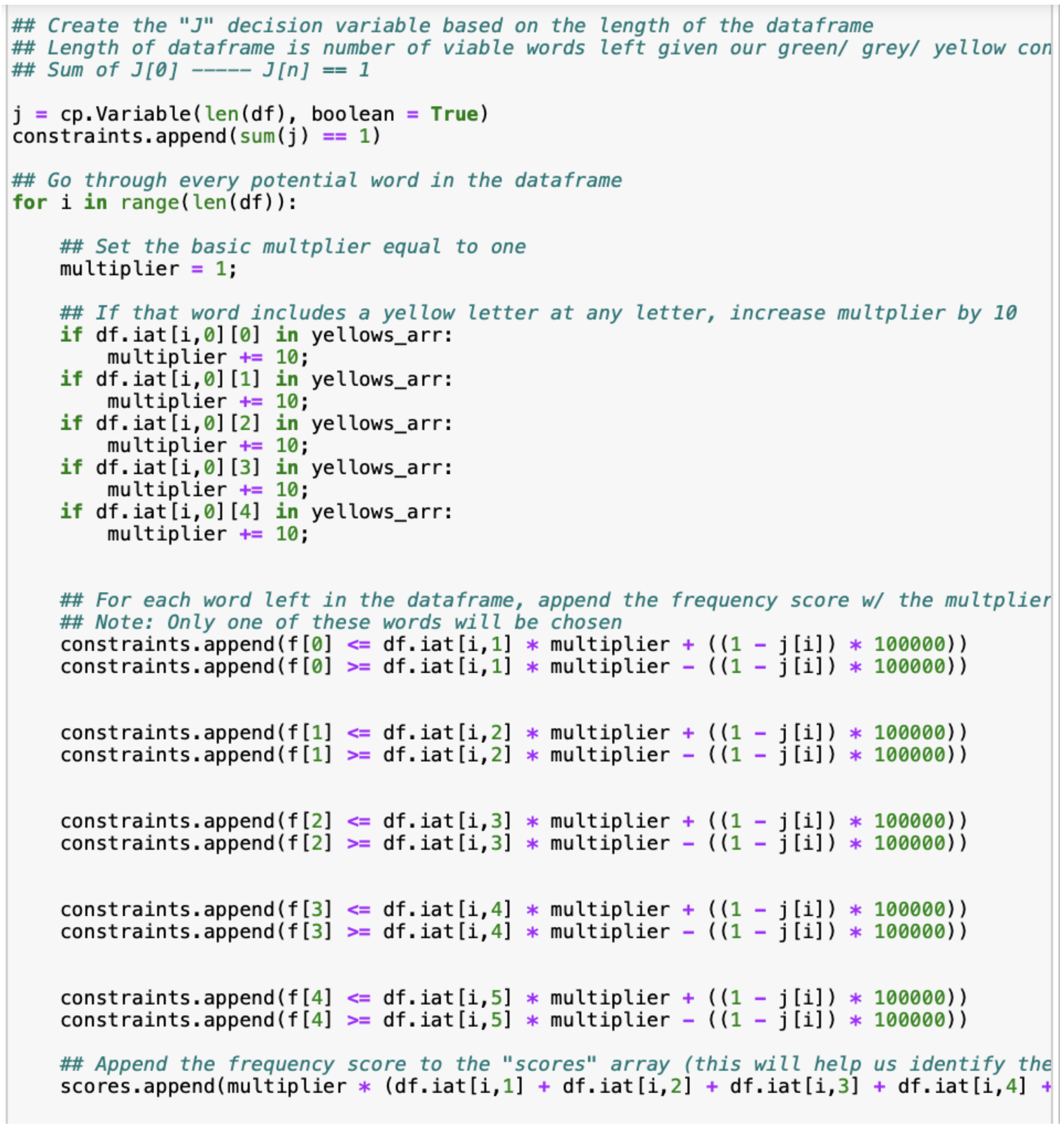
2) Loop to go through the dataframe and append green/ yellow/ grey constraints (A similar loop will run for previous words as well)



```
## Yellows Array: Used to append all yellow values for the most recent word
yellows_arr = []

## Yellow dictionary: Yellow values at each position ("-" if not a yellow value)
yellows_dict = {
    0:"-",
    1:"-",
    2:"-",
    3:"-",
    4:"-"
}

## Green dictionary: Green values at each position ("-" if not a greenvalue)
greens_dict = {
    0:"-",
    1:"-",
    2:"-",
    3:"-",
    4:"-"
}

## Temporary arrays to keep track of all the green, yellow, and gray values for this pa
greens = ["-", "-", "-", "-", "-"]
yellows = ["-", "-", "-", "-", "-"]
greys = ["-", "-", "-", "-", "-"]

## Function will return a key from a dictionary based on a target value (for use with g
def find_keys(dictionary, target_value):
    return [key for key, value in dictionary.items() if value == target_value]


## Keep running the loop until all 5 letters of the guess are iterated through
a = 0;
while (a != 5):

    ## Create a save of the dataframe in case there is an invalid entry
    df_save = df.copy()

    ## Make sure user enters lowercase letters
    print("Please enter all lowercase letters: ")
    print("")

    ## Get the green values of the guess, append to the array/ dictionary
    ## If there is a green value, dataframe now only includes words with the letter at
    for i in range(5):
        green = input("If the letter in position " + str(i+1) + " is green, please ente

        if(green != '-'):
            greens[i] = green
            greens_dict[i] = green
            df = df[df['word'].str[i] == green]

    print("")

    ## Get the yellow values of the guess, append to the array/ dictionary/ yellow_arr
    ## if there is a yellow value, dataframe now only includes words with the yellow le
    for i in range(5):
        yellow = input("If the letter in position " + str(i+1) + " is yellow, please en
```

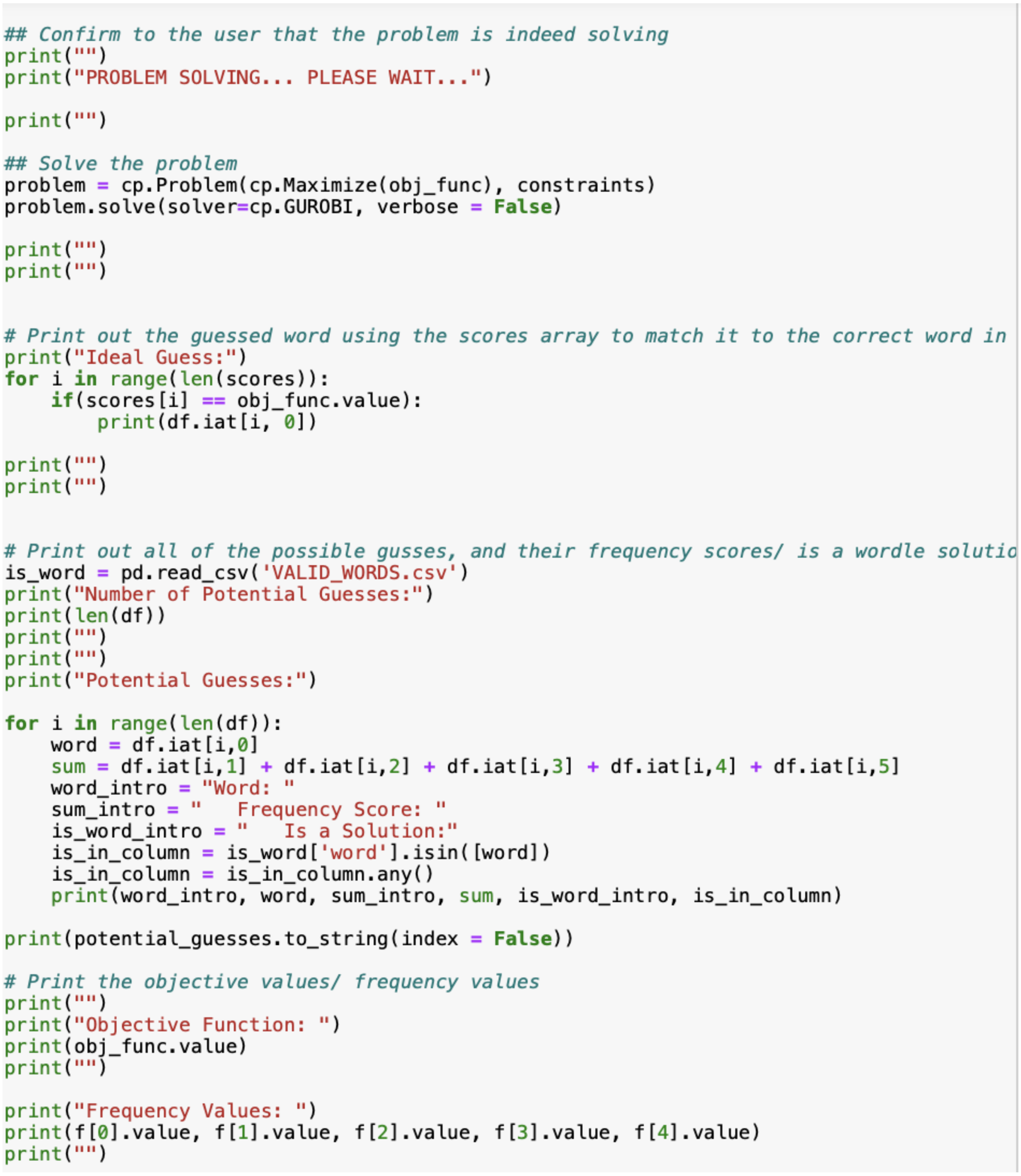
```python
    if(yellow != '-'):
        yellows[i] = yellow
        yellows_dict[i] = yellow
        yellows_arr.append(yellow)
        df = df[df['word'].apply(lambda x: len(x) > 1 and x[i] != yellow)]

print("")

## Get the grey values of the guess, append to the array
## If there is a grey value, filter the dataframe for words that do not have the gr
## Note: Special cases for grey/ green and grey/yellow combinations of the same let
for i in range(5):

    grey = input("If the letter in position " + str(i+1) + " is grey, please enter

    if(grey != '-'):
        greys[i] = grey

        ## If there is a green letter that is the same as the grey letter, remove a
        ## from every character except the character the green is at
        ## This is where the green_dict will come into play
        if grey in greens_dict.values():
            temp = find_keys(greens_dict,grey)
            temp2 = [0,1,2,3,4]
            filtered_array = [element for element in temp2 if element not in temp]

            for d in filtered_array:
                df = df[df['word'].str[d] != grey]

        ## If there is a yellow letter that is the same as the grey letter,
        ## Just say the grey letter cannot be in that particular position
        elif grey in yellows_dict.values():
            df = df[df['word'].apply(lambda x: len(x) > 1 and x[i] != grey)]

        ## If there is neither a duplicate yellow or green value that is the same a
        ## Words that don't have the grey letter
        else:
            df = df[~df['word'].apply(lambda x: grey in x)]


print("")

## Make sure user entered correct data (aka, user entered a green/ yellow/ grey val
for i in range(5):
    if(greens[i] != "-" or yellows[i] != "-" or greys[i] != "-"):
        a = a + 1

## Print an error and retry if the user did not enter the correct input
if (a != 5):
    print("ERROR- Each character must be green, yellow, or grey. Please reinput all
    print("")
    print("")

    ## Redefine the dataset and restart the loop
    df = df_save
    a = 0
```

3) Constrain the potential words to the constraints array, and choose only one word (binary “J” variable)



```
## Create the "J" decision variable based on the length of the dataframe
## Length of dataframe is number of viable words left given our green/ grey/ yellow con
## Sum of J[0] ----- J[n] == 1

j = cp.Variable(len(df), boolean = True)
constraints.append(sum(j) == 1)

## Go through every potential word in the dataframe
for i in range(len(df)):

    ## Set the basic multplier equal to one
    multiplier = 1;

    ## If that word includes a yellow letter at any letter, increase multplier by 10
    if df.iat[i,0][0] in yellows_arr:
        multiplier += 10;
    if df.iat[i,0][1] in yellows_arr:
        multiplier += 10;
    if df.iat[i,0][2] in yellows_arr:
        multiplier += 10;
    if df.iat[i,0][3] in yellows_arr:
        multiplier += 10;
    if df.iat[i,0][4] in yellows_arr:
        multiplier += 10;


    ## For each word left in the dataframe, append the frequency score w/ the multplier
    ## Note: Only one of these words will be chosen
    constraints.append(f[0] <= df.iat[i,1] * multiplier + ((1 - j[i]) * 100000))
    constraints.append(f[0] >= df.iat[i,1] * multiplier - ((1 - j[i]) * 100000))


    constraints.append(f[1] <= df.iat[i,2] * multiplier + ((1 - j[i]) * 100000))
    constraints.append(f[1] >= df.iat[i,2] * multiplier - ((1 - j[i]) * 100000))


    constraints.append(f[2] <= df.iat[i,3] * multiplier + ((1 - j[i]) * 100000))
    constraints.append(f[2] >= df.iat[i,3] * multiplier - ((1 - j[i]) * 100000))


    constraints.append(f[3] <= df.iat[i,4] * multiplier + ((1 - j[i]) * 100000))
    constraints.append(f[3] >= df.iat[i,4] * multiplier - ((1 - j[i]) * 100000))


    constraints.append(f[4] <= df.iat[i,5] * multiplier + ((1 - j[i]) * 100000))
    constraints.append(f[4] >= df.iat[i,5] * multiplier - ((1 - j[i]) * 100000))

    ## Append the frequency score to the "scores" array (this will help us identify the
    scores.append(multiplier * (df.iat[i,1] + df.iat[i,2] + df.iat[i,3] + df.iat[i,4] +
```

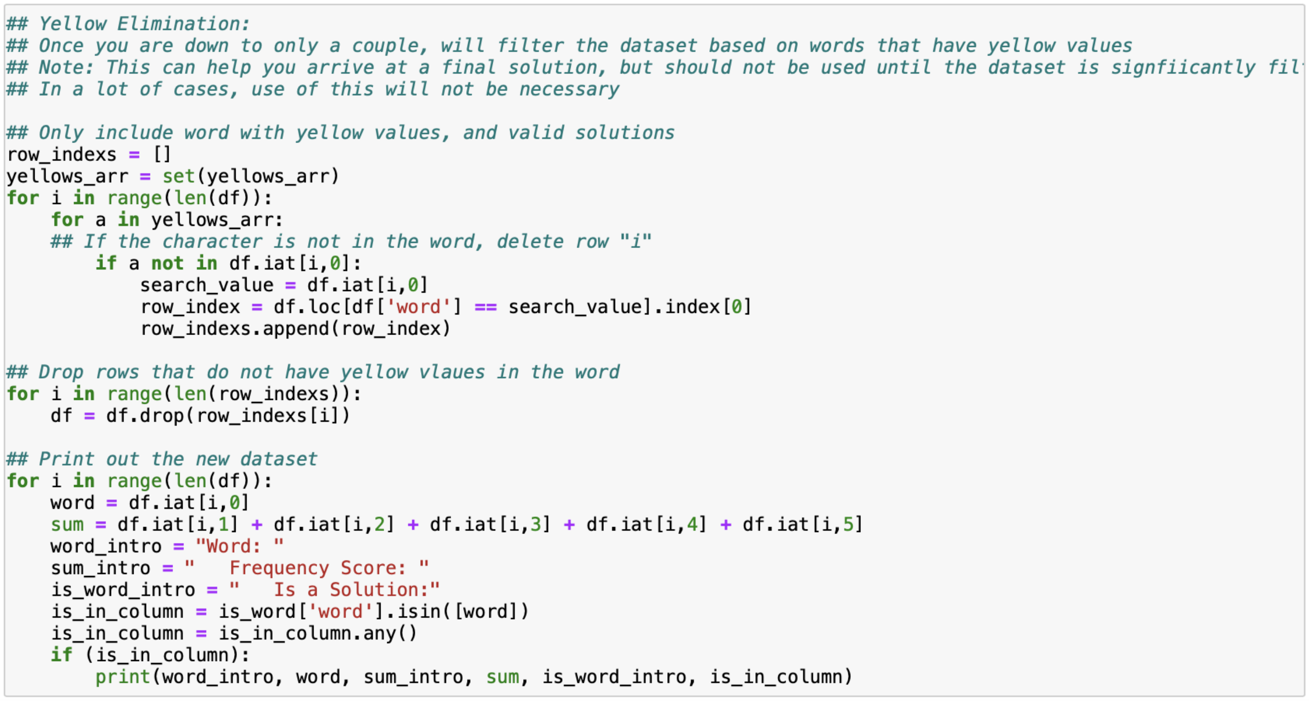
4) Solve the problem and print the results



```python
## Confirm to the user that the problem is indeed solving
print("")
print("PROBLEM SOLVING... PLEASE WAIT...")

print("")

## Solve the problem
problem = cp.Problem(cp.Maximize(obj_func), constraints)
problem.solve(solver=cp.GUROBI, verbose = False)

print("")
print("")


# Print out the guessed word using the scores array to match it to the correct word in
print("Ideal Guess:")
for i in range(len(scores)):
    if(scores[i] == obj_func.value):
        print(df.iat[i, 0])

print("")
print("")


# Print out all of the possible gusses, and their frequency scores/ is a wordle solutio
is_word = pd.read_csv('VALID_WORDS.csv')
print("Number of Potential Guesses:")
print(len(df))
print("")
print("")
print("Potential Guesses:")

for i in range(len(df)):
    word = df.iat[i,0]
    sum = df.iat[i,1] + df.iat[i,2] + df.iat[i,3] + df.iat[i,4] + df.iat[i,5]
    word_intro = "Word: "
    sum_intro = "   Frequency Score: "
    is_word_intro = "   Is a Solution:"
    is_in_column = is_word['word'].isin([word])
    is_in_column = is_in_column.any()
    print(word_intro, word, sum_intro, sum, is_word_intro, is_in_column)

print(potential_guesses.to_string(index = False))

# Print the objective values/ frequency values
print("")
print("Objective Function: ")
print(obj_func.value)
print("")

print("Frequency Values: ")
print(f[0].value, f[1].value, f[2].value, f[3].value, f[4].value)
print("")
```

5) Yellow Elimination (for very last step)- will eliminate words from the final dataframe that have no yellows in them

Yellow Elimination



```
## Yellow Elimination:
## Once you are down to only a couple, will filter the dataset based on words that have yellow values
## Note: This can help you arrive at a final solution, but should not be used until the dataset is signfiicantly fil
## In a lot of cases, use of this will not be necessary

## Only include word with yellow values, and valid solutions
row_indexs = []
yellows_arr = set(yellows_arr)
for i in range(len(df)):
    for a in yellows_arr:
    ## If the character is not in the word, delete row "i"
        if a not in df.iat[i,0]:
            search_value = df.iat[i,0]
            row_index = df.loc[df['word'] == search_value].index[0]
            row_indexs.append(row_index)

## Drop rows that do not have yellow vlaues in the word
for i in range(len(row_indexs)):
    df = df.drop(row_indexs[i])

## Print out the new dataset
for i in range(len(df)):
    word = df.iat[i,0]
    sum = df.iat[i,1] + df.iat[i,2] + df.iat[i,3] + df.iat[i,4] + df.iat[i,5]
    word_intro = "Word: "
    sum_intro = "   Frequency Score: "
    is_word_intro = "   Is a Solution:"
    is_in_column = is_word['word'].isin([word])
    is_in_column = is_in_column.any()
    if (is_in_column):
        print(word_intro, word, sum_intro, sum, is_word_intro, is_in_column)
```

Result/ Post Optimality Analysis Example:

An Example Test Case: Wordle #62
(https://wordlearchive.com/ includes an archive of all possible wordles, and therefore, provides great test cases. In this example, we used Wordle #62)

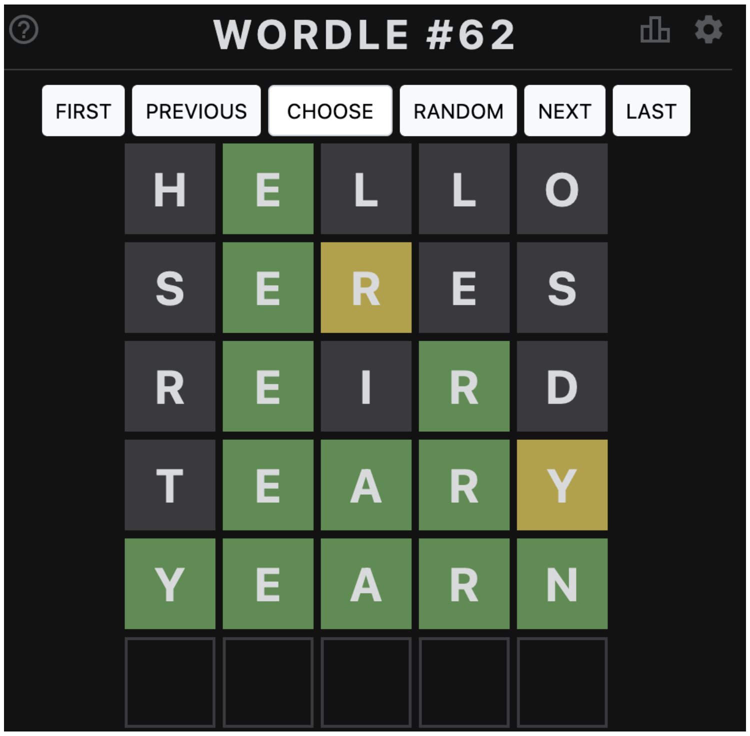
(G = Green, Y = yellow, B = Gray)

First Guess: “HELLO” (User created guess)
    Outcome: B,G,B,B,B, Obj_Func = N/A, num_potential_guesses = N/A
Second Guess: “SERES” (Program Ideal Guess)
    Outcome: B,G,Y,B,B, Obj_Func = 9551.0 , num_potential_guesses = 1003
Third Guess: “REIRD” (Program Ideal Guess)
    Outcome: B,G,B,G,B, Obj_Func = 83,286, num_potential_guesses = 239
Fourth Guess: “TEARY” (Program Ideal Guess)
    Outcome: B,G,G,G,Y, Obj_Func = 4485.0, num_potential_guesses = 7
Fifth Guess: “YEARN” (Program Ideal Guess)
    Outcome: G,G,G,G,G, Obj_Func = 38,027, num_potential_guesses = 1

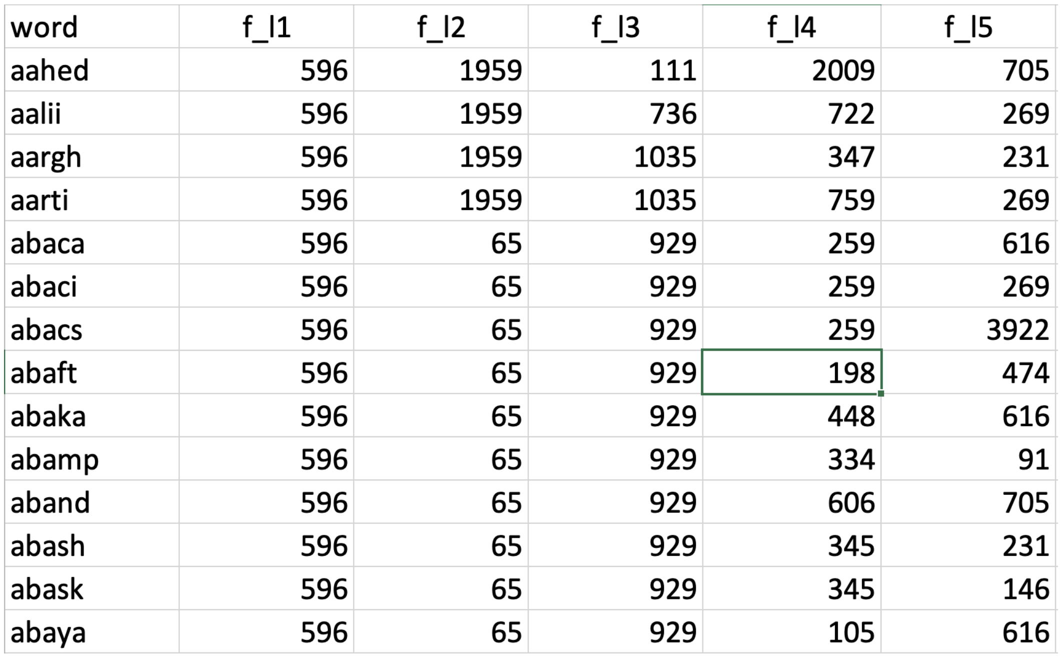
Code Snapshot from Last Guess:

```
Please enter all lowercase letters:

If the letter in position 1 is green, please enter it below (enter '-' if it isn't)-
If the letter in position 2 is green, please enter it below (enter '-' if it isn't)e
If the letter in position 3 is green, please enter it below (enter '-' if it isn't)a
If the letter in position 4 is green, please enter it below (enter '-' if it isn't)r
If the letter in position 5 is green, please enter it below (enter '-' if it isn't)-

If the letter in position 1 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 2 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 3 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 4 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 5 is yellow, please enter it below (enter '-' if it isn't)y

If the letter in position 1 is grey, please enter it below (enter '-' if it isn't)t
If the letter in position 2 is grey, please enter it below (enter '-' if it isn't)-
If the letter in position 3 is grey, please enter it below (enter '-' if it isn't)-
If the letter in position 4 is grey, please enter it below (enter '-' if it isn't)-
If the letter in position 5 is grey, please enter it below (enter '-' if it isn't)-


How many words have you previously guessed? 3


Lets get some info for word number  1


Please enter all lowercase letters:

If the letter in position 1 is green, please enter it below (enter '-' if it isn't)-
If the letter in position 2 is green, please enter it below (enter '-' if it isn't)e
If the letter in position 3 is green, please enter it below (enter '-' if it isn't)-
If the letter in position 4 is green, please enter it below (enter '-' if it isn't)-
If the letter in position 5 is green, please enter it below (enter '-' if it isn't)-

If the letter in position 1 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 2 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 3 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 4 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 5 is yellow, please enter it below (enter '-' if it isn't)-

If the letter in position 1 is grey, please enter it below (enter '-' if it isn't)h
If the letter in position 2 is grey, please enter it below (enter '-' if it isn't)-
If the letter in position 3 is grey, please enter it below (enter '-' if it isn't)l
If the letter in position 4 is grey, please enter it below (enter '-' if it isn't)l
If the letter in position 5 is grey, please enter it below (enter '-' if it isn't)o

Lets get some info for word number  2


Please enter all lowercase letters:

If the letter in position 1 is green, please enter it below (enter '-' if it isn't)-
If the letter in position 2 is green, please enter it below (enter '-' if it isn't)e
If the letter in position 3 is green, please enter it below (enter '-' if it isn't)-
If the letter in position 4 is green, please enter it below (enter '-' if it isn't)-
If the letter in position 5 is green, please enter it below (enter '-' if it isn't)-

If the letter in position 1 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 2 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 3 is yellow, please enter it below (enter '-' if it isn't)r
If the letter in position 4 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 5 is yellow, please enter it below (enter '-' if it isn't)-

If the letter in position 1 is grey, please enter it below (enter '-' if it isn't)s
If the letter in position 2 is grey, please enter it below (enter '-' if it isn't)-
If the letter in position 3 is grey, please enter it below (enter '-' if it isn't)-
If the letter in position 4 is grey, please enter it below (enter '-' if it isn't)e
If the letter in position 5 is grey, please enter it below (enter '-' if it isn't)s

Lets get some info for word number  3


Please enter all lowercase letters:

If the letter in position 1 is green, please enter it below (enter '-' if it isn't)-
If the letter in position 2 is green, please enter it below (enter '-' if it isn't)e
If the letter in position 3 is green, please enter it below (enter '-' if it isn't)-
If the letter in position 4 is green, please enter it below (enter '-' if it isn't)r
If the letter in position 5 is green, please enter it below (enter '-' if it isn't)-

If the letter in position 1 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 2 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 3 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 4 is yellow, please enter it below (enter '-' if it isn't)-
If the letter in position 5 is yellow, please enter it below (enter '-' if it isn't)-

If the letter in position 1 is grey, please enter it below (enter '-' if it isn't)r
If the letter in position 2 is grey, please enter it below (enter '-' if it isn't)-
If the letter in position 3 is grey, please enter it below (enter '-' if it isn't)i
If the letter in position 4 is grey, please enter it below (enter '-' if it isn't)-
If the letter in position 5 is grey, please enter it below (enter '-' if it isn't)d


PROBLEM SOLVING... PLEASE WAIT...

Set parameter Username
Academic license - for non-commercial use only - expires 2024-08-30


Ideal Guess:
yearn


Number of Potential Guesses:
1


Potential Guesses:
Word:  yearn    Frequency Score:  3457    Is a Solution: True
Empty DataFrame
Columns: [Word, Is_Word, Frequency_Score]
Index: []

Objective Function:
38027.0

Frequency Values:
1925.0 15246.0 10219.0 6237.0 4400.0
```

Wordle Outcome:



Words and their Frequency Scores (note- this is not filtered. Figure meant to show what the dataset referenced throughout the paper looks like):



word	f_l1	f_l2	f_l3	f_l4	f_l5
aahed	596	1959	111	2009	705
aalii	596	1959	736	722	269
aargh	596	1959	1035	347	231
aarti	596	1959	1035	759	269
abaca	596	65	929	259	616
abaci	596	65	929	259	269
abacs	596	65	929	259	3922
abaft	596	65	929	198	474
abaka	596	65	929	448	616
abamp	596	65	929	334	91
aband	596	65	929	606	705
abash	596	65	929	345	231
abask	596	65	929	345	146
abaya	596	65	929	105	616

Appendix (Video):
https://youtu.be/7l5xh_VxM10

Appendix (GitHub):
https://github.com/RealDavidRicci/ISE-3230

Appendix (Task Distribution):
For the majority of the project, we were in zoom calls all working on the same problem whether that be formulating the problem or the code.

Nathan: Lead the problem formulation section by researching past attempts at creating programs to solve Wordle (an example of this is mentioned in the "Problem Formulation" section of this paper), and coming up with the idea to use frequency scores to differentiate the next best guesses.

Aidan: Aidan created much of the initial code by creating the first working program draft. For most of the project, Aidan typed the code on his computer, while Nathan and David joined multiple Zoom calls to assist Aidan in writing the project code (writing a majority of the code was a collaborative effort)
David: David put many of the finishing touches on the program (transforming it into so that it could take multiple words into account, and coding many of the "user prompts"). David also did much of the Project Report drafting and editing, as well as created the video. Finally, David also created the CSV files that included words and their associated frequency scores.

Overall, each group member would conclude that the workload was divided evenly, as major project stories were made over the course of 5 Zoom calls lasting between 3-4 hours each.