

# Effects of Font Design on Highway Sign Legibility

Marta Perez Vidal-Ribas

Thesis submitted to the Faculty of the  
Virginia Polytechnic Institute and State University  
in partial fulfillment of the requirements of the degree of

Master of Science

In

Civil Engineering

Bryan J. Katz

Kathleen Hancock

Kevin P Heaslip

August 11, 2023

Blacksburg, Virginia

Keywords: Sign Legibility, Font Design, Typefaces, Transportation Safety, Standard Highway Gothic, Clearview

# Effects of Font Design on Highway Sign Legibility

Marta Perez Vidal-Ribas

## Abstract

The Manual on Uniform Traffic Control Devices (MUTCD) set Standard Highway Alphabet, or Highway Gothic, as the standard font for all American roadway signs in 1966. Since then, that standard has not changed, with all signs following the norm.

In the 1980s, new retro-reflective sheeting introduced on American roadways caused Highway Gothic to be more difficult to read, due to the light “halo” effect caused around the letters, or halation. Recently, more studies have been conducted to improve the overall legibility of Highway Gothic. One study found that its legibility could greatly improve if its size was increased by 20%. This, however, is extremely unlikely, since increasing the font size would also entail an increase in the physical signs lining roadways. In the 1990s, a new font was created, Clearview, to help combat the negative effects of Standard Highway Alphabet. This font received interim approval in 2004, which was removed in 2016 due to ambiguous results from studies as to whether it was more beneficial than Highway Gothic. It was reinstated two years later, in 2018.

Legibility has five different components: retro-reflectivity, irradiation, luminance, contrast, and font design. Understanding these five components, and the benefits of each, can lead to the betterment of the font design on highway signs.

This study consisted of two web-based tests. In the first test, the “Letters Test”, participants would see a character slowly increasing in size on the screen. Once they could decipher the character, they would click the screen and input the character shown. On the second test, the “Words Test”, participants would follow the same instructions, albeit with words in place of characters. There were four fonts tested, on both a positive and negative contrasts. The positive contrast consisted of a green background with a white font, and the negative contrast was a white background with a black font. The four tested fonts were E Modified Base, Alpha Two FHWA E Narrow, Alpha Two FHWA D, and Alpha Two FHWA C, named Base, Narrow, D-Altered, and C-Altered respectively.

Forty-two participants participated in both tests. For the “Letters Test”, the smallest average font size was the narrow font, followed by the base and D-altered. For the “Words Test”, the smallest average font size was the base font, followed by the narrow, D-altered, and C-altered fonts.

Overall, the base and narrow fonts took up more space than the D-altered and C-altered fonts. It is recommended that field tests are conducted with these fonts, taking into account the space that they take up, not the font size. This analysis could help to determine whether or not the altered fonts are as legible, or even more legible, than the base and narrow fonts when occupying the same space.

# Effects of Font Design on Highway Sign Legibility

Marta Perez Vidal-Ribas

## General Audience Abstract

The Manual on Uniform Traffic Control Devices (MUTCD) set Standard Highway Alphabet, or Highway Gothic, as the standard font for all American roadway signs in 1966. Since then, that standard has not changed, with all signs following the norm.

In the 1980s, new retro-reflective sheeting introduced on American roadways caused Highway Gothic to be more difficult to read, due to the light “halo” effect caused around the letters, or halation. Recently, more studies have been conducted to improve the overall legibility of Highway Gothic. One study found that its legibility could greatly improve if its size was increased by 20%. This, however, is extremely unlikely, since increasing the font size would also entail an increase in the physical signs lining roadways. In the 1990s, a new font was created, Clearview, to help combat the negative effects of Standard Highway Alphabet. This font received interim approval in 2004, which was removed in 2016 due to ambiguous results from studies as to whether it was more beneficial than Highway Gothic. It was reinstated two years later, in 2018.

Legibility has five different components: retro-reflectivity, irradiation, luminance, contrast, and font design. Understanding these five components, and the benefits of each, can lead to the betterment of the font design on highway signs.

This study consisted of two web-based tests. In the first test, the “Letters Test”, participants would see a character slowly increasing in size on the screen. Once they could decipher the character, they would click the screen and input the character shown. On the second test, the “Words Test”, participants would follow the same instructions, albeit with words in place of characters. There were four fonts tested, on both a positive and negative contrasts. The positive contrast consisted of a green background with a white font, and the negative contrast was a white background with a black font. The four tested fonts were E Modified Base, Alpha Two FHWA E Narrow, Alpha Two FHWA D, and Alpha Two FHWA C, named Base, Narrow, D-Altered, and C-Altered respectively.

Forty-two participants participated in both tests. For the “Letters Test”, the smallest average font size was the narrow font, followed by the base and D-altered. For the “Words Test”, the smallest average font size was the base font, followed by the narrow, D-altered, and C-altered fonts.

Overall, the base and narrow fonts took up more space than the D-altered and C-altered fonts. It is recommended that field tests are conducted with these fonts, taking into account the space that they take up, not the font size. This analysis could help to determine whether or not the altered fonts are as legible, or even more legible, than the base and narrow fonts when occupying the same space.

# Table of Contents

Introduction.....	1
Background .....	1
Necessity of research.....	1
Research question.....	1
General approach.....	2
Literature Review.....	2
Legibility .....	2
Existing Research .....	2
Retro-reflectivity .....	2
Irradiation .....	3
Luminance .....	4
Contrast.....	4
Font Design.....	4
Preliminary Research.....	9
Conclusion.....	9
Methods.....	10
Overview .....	10
Code Development .....	10
Contrast .....	10
Letters.....	11
Words .....	11
Setting.....	11
Participants .....	11
Procedures .....	12
Data Analysis .....	12
Overview .....	12
Techniques Used .....	13
Results.....	13
Overview .....	13
Letters Test.....	14
All Letters .....	14

Words Test .....	24
All Words .....	24
All Words in Positive Contrast .....	33
All Words in Negative Contrast .....	35
Discussion .....	37
Overview .....	37
Letters .....	37
Words .....	41
Summary .....	41
Recommendations .....	41
References .....	44
Appendix 1 – IRB Approval Letter .....	47
Appendix 2 – Informed Consent Form .....	49
Appendix 3 – Python Code .....	52
Letters Test .....	52
Words Test .....	67
Appendix 4 – Instructions Displayed Before Test .....	84
Appendix 5 – Data for Individual Letters .....	86
Appendix 6 – Data for Individual Words .....	93

# Introduction

## Background

The first roadway sign introduced in the United States was a STOP sign, making its debut in 1915 in Detroit, Michigan. Shortly after, experts came together to devise a manual in which different shaped signs would denote different hazards, and nine years later, the American Association of State Highway Officials (AASHO) released the first signing report. This report would serve as a blueprint for the *Manual and Specifications for the Manufacture, Display, and Erection of U.S. Standard Road Markers and Signs*. The goal of this manual was to set the standard design and shape for signs on roadways, to be able to create a standard system for all drivers to understand. The development of the manual then led to the creation of the MUTCD, and a country-wide implementation of its norms in 1935 [1].

The MUTCD classifies signs into three different types: regulatory, warning, and guide signs. All of these signs have different standard designs, shapes, and colors, but they all use the same font. The Standard Highway Alphabet font, commonly referred to as Highway Gothic, was a modification to the Gothic style alphabet. In the late 1940s, researchers at the California Department of Transportation conducted a variety of legibility studies to increase the legibility and readability of the Gothic style font. In 1966, the Federal Highway Administration re-released the Standard Alphabets for Traffic Control Devices with the modified font. Later versions would release more series of the font (Series B, C, D, E, E Modified, F) and spacing requirement when used with different letter combinations. However, the original font created by the California researchers did not change [2].

## Necessity of research

The legibility of the font used across all American highway signs is an important aspect of sign design that should be evaluated. If different fonts could make roadway signs easier to read, it could lead to an overall improvement of the driving experience. In the 1980s, when new reflective materials were introduced to American highways, Highway Gothic became more difficult to read due to halation, or the incident caused when light reflects off of characters, causing a “halo” effect around the letters [3]. Halation especially impacted older drivers, as age comes with slower reaction times and minimized visual perception. By 2050, 39% of drivers on American highways will be older than 55 years old, an 11% increase from what that number was in 2000. [4]. As vehicle environments have been continuously improving, year by year, engineers have to ensure that the driving environment is advancing as well to accommodate the general public.

There are multiple components in font design that can be altered to make words easier to read. For example, the internal spaces of letters such as *a* and *e* can be made wider to combat the effects of halation [3]. Understanding the different components, and their respective effects on legibility can aid designers in the development of updated fonts to create a more pleasant driving experience.

## Research question

Overall, the goal of this thesis is to evaluate a newly designed set of standard alphabets related to two main components of the font design and their effects on legibility: letter stroke width and spacing between characters. In other words, in what ways do these particular changes in these newly designed fonts allow for drivers to process information on signs at a faster and

more efficient rate? If so, what changes are beneficial, and which are not? The answers to these questions will aim to lead font designers to optimize new typefaces, and to be able to implement a new standard across American roadways.

## General approach

Much research has been done in the past to study the effects of different fonts on roadways. However, this research is outdated since there have been major advancements in reflective sign materials and vehicle headlights. This thesis aims to analyze past research, but also to conduct an updated study. This study will compare up to four fonts at once, all with different modifications, to help understand which changes cause the most differences, and whether those differences cause a positive or negative change in legibility. This thesis will contribute to the fields of transportation safety and font design through the analysis of the data collected in the study.

## Literature Review

This literature review will analyze the different components of font design and legibility. Past studies and projects will be analyzed to help understand the background of the research question. Areas where research has not been conducted will be identified, and this review will allow for the understanding of font design as it applies to American roadways. The components of this review include legibility, and an analysis of existing research as it applies to retro-reflectivity, irradiation, luminance, contrast, and font design.

## Legibility

Legibility, defined as the ability and ease of people to encode text, is arguably one of the most important aspects of traffic sign design [5]. Legibility in regard to roadway signs is impacted by a variety of factors: retro-reflectivity, irradiation, luminance, contrast, and font design [6]. With vehicular travel on the rise on U.S. roadways, it is important to understand and develop the legibility of signage fonts.

## Existing Research

As driving has increased in popularity, roadways, vehicles, and driving environments have adapted to make it safer for vehicle users. However, something that has remained constant since it was introduced in the 1940s has been the sign font required by the Federal Highway Administration. In the past 50 years, researchers have studied the effects of font design on roadway signs, but most of these studies have produced inconclusive results. Other studies done on major aspects of legibility have been able to yield definitive recommendations.

## Retro-reflectivity

According to the Manual on Uniform Traffic Control Devices for Streets and Highways, better known as the MUTCD, retro-reflectivity is an essential aspect of legibility, especially at nighttime. Retro-reflective sheeting is what allows a vehicle's headlamp light to reflect off of the sign, making the information visible to the driver [7]. In nighttime conditions, it is important for light to be able to reflect off of the sign itself, as these signs are usually not lighted. A material's retro-reflective capabilities determine how much light can be reflected back to the driver, and whether the words reflected are still legible.

The Texas Transportation Institute (TTI) conducted a study on the effects of retro-reflective sheeting on sign legibility, and it consisted entirely of older drivers with an average age of 71.

This study was conducted on the field, on three different sites in St. Paul, Minnesota. The three sites were of varying complexities, with factors such as roadway lighting, number of lanes, environment, and amount of traffic. The font selected for the study was Standard Highway Gothic Series C, and the six words chosen were similar to one another. Several types of reflective materials were chosen to determine if retro-reflectivity can impact legibility in nighttime conditions. Retro-reflective sheeting is classified by the FHWA, with Type I sheeting having the lowest intensity, and Type IX sheeting having the highest retroreflective intensity. The researchers at TTI found that Type VII and Type IX materials are best, followed by Type III material, and lastly Type I. However, arguably the most significant finding was the importance of being able to do legibility studies on the field [8]. While most studies are conducted under controlled, experimental conditions, it is hard to mimic the effects of driving in an open, uncontrolled environment. Being able to do this study on the field gave the researchers an insight into uncontrolled driving conditions, and how these materials could affect drivers on the roadway.

If a material is too reflective, it can lead to an occurrence known as irradiation. This happens when too much light is reflected off of the characters on the highway sign, and it causes the letters to have a “halo” effect. This “halo” effect can be combated through thinner line strokes and wider gaps within the letters.

#### Irradiation

As previously mentioned, irradiation is defined as the blurring that occurs around the edges of lettering under retro-reflective conditions. This effect, also known as halation, creates a fog or halo effect around each letter [9]. This effect can be visualized in Figure 1. The current standard font, Highway Gothic, has thick strokes and uses all capital letter displays for most roadway signs. Researchers at the Pennsylvania Transportation Institute (PTI) believed that these two characteristics caused irradiation, especially in older drivers [10]. Irradiation “becomes a problem when a stroke is so bright that it ... bleeds into the character’s open spaces, creating a blobbing effect that reduces character legibility” [10]. Clearview, developed by Meeker and Associates, was created based off of the Standard Highway Series E (Modified) font to combat the effects of irradiation, and to allow for the use of mixed-case legends. With thinner strokes, and more open spaces within characters, Clearview combated the effects of halation. Clearview was designed in both a normal and condensed version, and they were named accordingly.



Figure 1: Irradiation, or "Halo" Effect [11]

## Luminance

Luminance is the “intensity of light emitted from a surface, per unit area” [12]. Luminance and retro-reflectivity work together so drivers can read text on roadway signs, especially at night. A couple of studies have been conducted, with the objective of finding the optimal luminance value on roadway signs. Sivak and Olson synthesized 18 of these conducted studies to come up with the best values for luminance on nonreflective black legends and reflective light backgrounds. “Legibility is ... an inverted U-shaped function of luminance” [13] which means the optimum luminance for legibility is commonly located at the crest of the function. The differences come into play when factors such as complexity of environment, visual capabilities, age, legend to background contrast, and others are applied. Findings for a black legend on a light background show the average optimal luminance to be  $75 \text{ cd/m}^2$ , therefore establishing the recommended value for those types of signs. For signs that are considered to be fully reflectorized, the luminance of one color varies with the luminance of the contrasting color. For these scenarios, the average was a contrast of 12:1. For example, if the background luminance is  $2 \text{ cd/m}^2$ , the optimal luminance would equal  $24 \text{ cd/m}^2$ . Lastly, replacement luminance, or the point at which the sign fails retro-reflectivity at night, was analyzed. The minimum replacement luminance was found to be  $2.4 \text{ cd/m}^2$  for the lighter component. This finding should be applied for signs with black legends and light backgrounds and white legends and dark backgrounds with a luminance of  $0.4 \text{ cd/m}^2$  [13]. For these signs, it is clear that light-background signs need a higher minimum luminance than dark-background signs, to make the information easier to process for readers. Sivak and Olson were able to give specific recommendations on luminance values based on the synthesis of these studies.

## Contrast

Contrast remains an important factor of sign legibility. Researchers at Michigan State University conducted a study to analyze the visibility of high contrast signs of black letters on white backgrounds (negative contrast) and vice versa (positive contrast). Previously, only one or two studies had analyzed both scenarios. The results found that there were no major differences between negative and positive contrast signs [14]. For the purpose of this study, fonts will be tested on both types of contrast. Figure 2 shows a negative contrast sign on the right, and a positive contrast sign on the left.



Figure 2: Contrast on Signs

## Font Design

Standard Highway Alphabet, also known as Highway Gothic, has been the long-standing font used on American roadways for over 70 years [15]. When studies began to be conducted on the feasibility and readability of Highway Gothic, they observed that if font size was increased by 20% it would largely improve its own legibility. However, this would cause many issues, as this would require larger signs, and larger structures to hold those signs. In the 1990s, a new font, Clearview, was designed to battle the shortcomings of Highway Gothic. Clearview was designed

by the Pennsylvania Transportation Institute (PTI) to overcome the deficiencies of Highway Gothic and reduced irradiation [11]. In the first stage of studies, Clearview was found to be more legible than Highway Gothic, at the same size. The FHWA approved its use in 2004, but discontinued it in 2016, “citing the ambiguity of subsequent results” [15].

The studies that have been done on new fonts have been found to be inconclusive, which has raised criticism from the FHWA. A study from the Texas Transportation Institute (TTI), analyzed the performance of three different alphabets: Highway Gothic Series E (Modified), Clearview, and British Transport Medium. A visualization of these fonts can be seen in Figure 3.

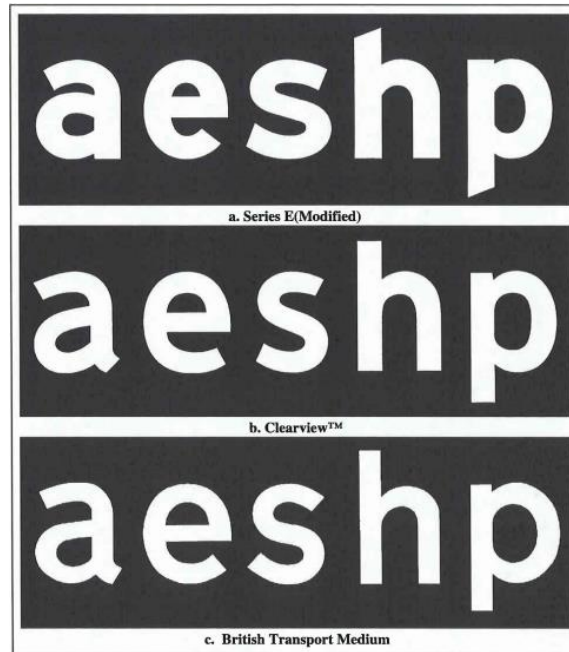


Figure 3: Fonts Used in TTI Study [11]

Researchers tested these fonts during daytime and nighttime conditions, in overhead and ground-mounted positions, and looked for legibility and word recognition. The results concluded that Clearview was, in fact, more legible than Highway Gothic Series E (M) in overhead signals. In ground-mounted signals, Series E (M) was more legible during the daytime, and the two fonts were equal at nighttime conditions. The third font, British Transport Medium, did not outperform Highway Gothic or Clearview in any of the categories [11]. This study shows the benefits of Clearview for participants; however, the authors did not recommend the implementation of Clearview as a national font.

Another factor of font design is the legibility index of each font, which is especially applicable to older drivers. A legibility index is defined as the “distance in feet at which a letter is legible per inch of letter height” [16] and is used to determine the relative legibility of different fonts. Standard Highway Alphabet Series E letters have a legibility index of 50 ft/in, while Series B, C, D have indexes of 33, 42.5, and 50 respectively [16]. Table 1 shows the relationship between Snellen Visual Acuity (a tool widely used to measure eyesight), visual angle of the letter, and the visibility index. A legibility index for “perfect vision”, or 20/20 on the Snellen test is 57.3 ft/in, and for 20/40 vision (the minimum for a driver’s license in the majority of states) is

29 ft/in [16]. This alludes to the fact that many drivers do not have the minimum visual acuity to read Standard Highway Alphabet Series E letters on highway signs.

Table 1: Legibility Index [11]

Snellen Visual Acuity	Visual Angle of Letter (minutes)	Legibility Index	
		m/mm	ft/in
20/10	2.5	1.38	114.6
20/20	5.0	0.69	57.3
20/30	7.5	0.46	38.2
20/40	10.0	0.34	28.7
20/50	12.5	0.28	22.9
20/60	15.0	0.23	19.1

The aforementioned study, led by PTI, also found that the Clearview font was beneficial to older drivers. The fonts tested in this study were Clearview, Clearview at a 12% increase in character height, Clearview Condensed, Clearview Condensed at a 12% increase in character height, Standard Highway Series E (Modified), Standard Highway Series D (all uppercase). It should be noted that the Clearview fonts take up 12% less space than the standard fonts. The Clearview fonts at a 12% increase were tested since they would be taking up just as much signage space as the currently used Standard Highway fonts.

The study found that there were no significant differences for the Clearview, Clearview at a 12% increase, or Standard Highway Series E (M) fonts. There were, however, notable differences between the Clearview, Clearview Condensed at 12%, and Standard Highway series D fonts. In the study, the Clearview fonts had larger legibility indexes, as they were recognized from further away. Figure 4 and Figure 5 below show the legibility indexes of all of the tested fonts, with Series D and Clearview Condensed having the lowest values and Clearview at a 12% increased height having the highest one [10].

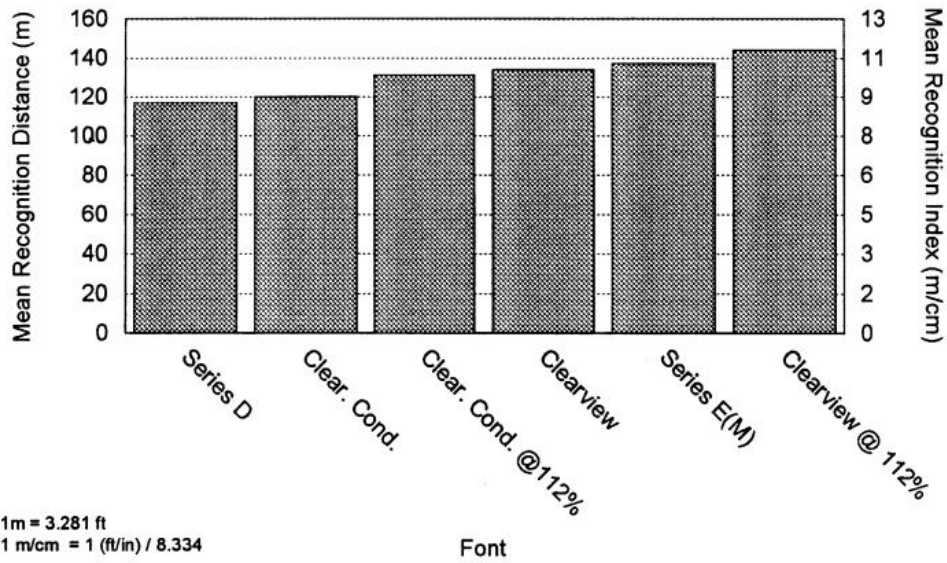


Figure 4: Font Legibility (Daytime) [10]

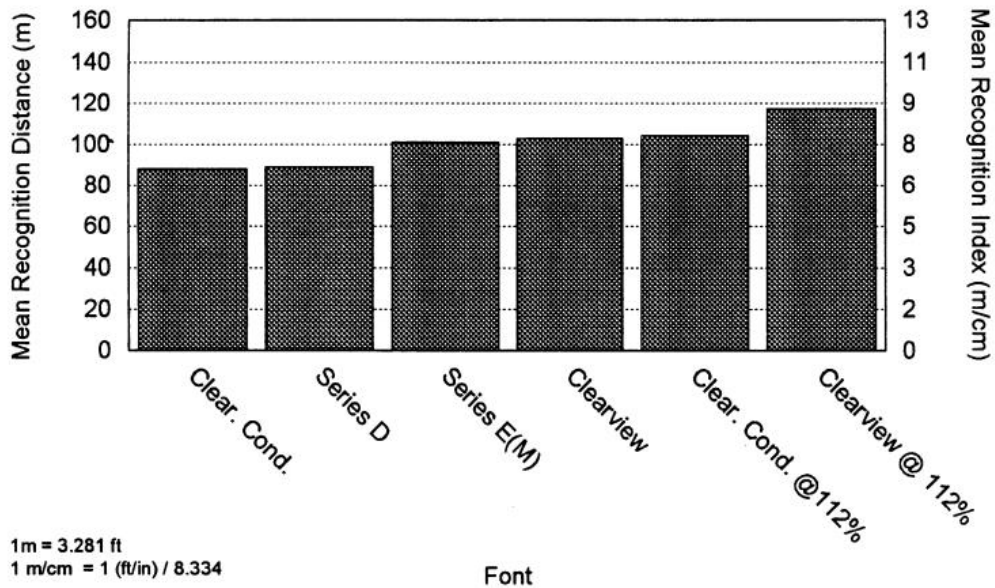


Figure 5: Font Legibility (Nighttime) [10]

Although the findings did show that Clearview was the most legible font, the findings were not significant enough to warrant a change during daytime conditions. During the night tests, however, all of the Clearview fonts had a higher legibility index than the Standard Highway

ones. This was due to the lack of halation and irradiation present in the Clearview letters due to the thinner strokes and wider open spaces of the font.

Another study, also led by PTI, took into account different fonts and contrasts. As previously discussed, researchers at Michigan State found little to no differences in the legibility of the same font using different contrasts. This study was done in the field, with the test subjects driving the vehicle. As found in the retro-reflectivity study led by TTI, it is beneficial to conduct these tests on the field, mimicking real driving conditions. Words on signs were randomly assigned, with each test subject having three separate combinations presented to them. The fonts tested in the study included Clearview (2-B, 2-B, and 4-B) and Standard Highway Alphabet (series C, D, and E). The Standard Highway Alphabet fonts were tested in a combination of mixed-case lettering and all-capital lettering. The researchers included many other variables, such as spacing in between letters, letter height (6” or 4.8”), time of day, and age group [17]. In the end, they tested 18 negative contrast conditions and 9 positive contrast conditions. These combinations can be seen in Figure 6.

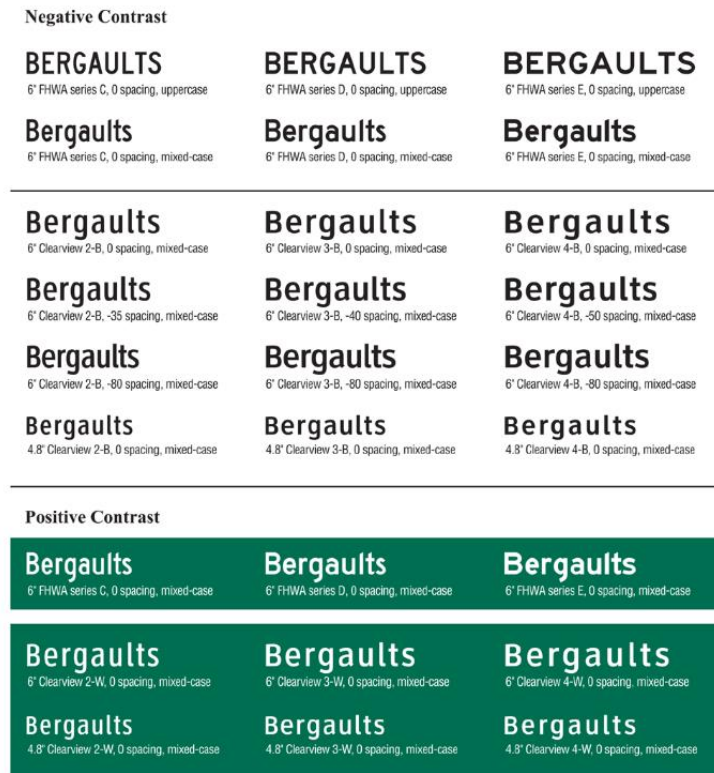


Figure 6: 27 Different Combinations Used in the Study [17]

The researchers ended up finding that mixed-case Clearview is just as legible as uppercase Standard Highway Alphabet and takes up significantly less sign space. Clearview mixed case outperformed Standard Highway by as much as 31 percent. It was also found that wide character spacing outperformed all other condensed spacing options, and the 4.8” Clearview font was just as successful as the 6” Standard Highway Alphabet fonts. Clearview was clearly the preferred font for participants of all ages, leading to more recognizable and readable signs.

Overall, most studies conducted have shown that Clearview is better, if not just as good, than the current Standard Highway Alphabet font. Through its use of thinner stroke lines, it diminishes irradiation and outperforms other fonts in nighttime conditions. However, it was concluded that font design could still improve.

### Preliminary Research

Research conducted at Virginia Tech by Andrea Ruano Duke had a similar experimental design, testing the legibility distances of seven fonts, as seen in Figure 7: Clearview 5W, Clearview 5WR, E Mod Base CVS, E Mod Max, Series E, Series E (Modified), and Sidelane 400 E [18]. The test was conducted on a computer, using Sign Sim, a simulation software developed by the Federal Highway Administration. This software allowed the researcher to test 140 different sign conditions, each with a different contrast, font, and word combination. The words used in this study were modeled after the words Ruano Duke used [18].



Figure 7: Fonts Used in Virginia Tech Study [18]

The study concluded that, out of the fonts provided, Series E had the lowest mean legibility distance, and Clearview 5W had the highest mean legibility distance across all age groups tested. Words displayed in positive contrast had a higher legibility index than those displayed on negative contrast, and these differences were found to be statistically significant. Overall, the study concluded that Clearview in positive contrast outperformed any other font and contrast combinations, making it the most legible [18].

### Conclusion

Overall, past studies show that the current typeface used by the Federal Highway Administration could be modified to be more legible. Albeit Clearview has given inconclusive results, some studies have shown some improvement over the current standard. By further altering the existing FHWA fonts, researchers could find a typeface that outperforms the current Highway Gothic style and could potentially become the new American standard.

## Methods

### Overview

This chapter gives a detailed overview of the experimental procedure, participant selection, experimental development, and testing methods used for this thesis. The research team received IRB approval (IRB 23-357, approval letter found in Appendix 1) before testing began to be able to use human participants. Participants were aware of this approval and were required to read and verbally agree to an informed consent form, seen in Appendix 2, ensuring that their participation in the study was voluntary.

### Code Development

To be able to carry out the experimental procedure, a code provided by ToXcel, a transportation research company, was modified. The modifications allowed the research team to make the results of the quiz directly relevant to the study. The changes to the code were made with the assistance of Stephen Taylor, who is also the original developer of the code. The full code for both tests is provided in Appendix 3.

### Font

There were four different fonts chosen for these tests, designed by Robert Wertz. The first test (which will be also referred to as the “Letters Test”) used three fonts, referred to as: *Base*, *Narrow*, and *D-altered*. The font referred to as *Base* is formally known as E Mod Base, *Narrow* is Alpha Two FHWA E Narrow, and *D-altered* is Alpha Two FHWA D. The second test (referred to as the “Words Test”) used the aforementioned three fonts, as well as a fourth font referred to as *C-altered*. This font is formally known as Alpha Two FHWA C. All of these fonts can be visualized in

Figure 8.

*Base* - E Mod Base

**The quick brown fox jumps over the lazy dog. 1234567890**

*Narrow* - Alpha Two FHWA E Narrow

**The quick brown fox jumps over the lazy dog. 1234567890**

*D-Altered* - Alpha Two FHWA D

**The quick brown fox jumps over the lazy dog. 1234567890**

*C-Altered* - Alpha Two FHWA C

**The quick brown fox jumps over the lazy dog. 1234567890**

Figure 8: Fonts Used in Study

### Contrast

Positive and negative contrast variations were added to the test, to be randomized within each iteration. The positive contrast chosen was a white font color on a green background, and the negative contrast was black font color on a yellow background. Factoring in contrast into the

test allowed the researchers to determine whether or not contrast had an effect on the legibility of each font, and whether it favored certain typefaces.

## Letters

The first test used a mix of letters and symbols. The symbols were introduced to preserve the integrity of the test, and to ensure the user could confidently see a letter instead of speculating. The letters used were chosen due to their inter-character spacing, or the space within the character itself. All of the letters were lowercase, except for “A” and “F”, which were provided in both uppercase and lowercase form. The letters and characters used can be seen in Table 2.

Table 2: Words and Characters Used in First Test (“Letters Test”)

A	F	a	c	e
f	g	k	s	t
w	x	z	>	<
%	^			

## Words

The second test was composed of mostly English words. *Bartey* was used as a “fake” word, due to its similarity to *Barley*, another word used in the study. This served a similar purpose to the symbols in the first test, and allowed the research team to see if the participants were confident in their answers. The full list of words used in the study can be found in Table 3.

Table 3: Words Used in Second Test (“Words Test”) [18]

Barley	Bartey
Eatery	Felony
Honors	Houses
Season	Senior
Sensor	Doting

## Setting

To maximize the number of participants, participants were allowed to take both tests remotely. The two requirements to do so were a portable computer and high-speed internet. Since the test compares the size of the letters and words shown to a uniform unit, a point (equivalent to 1/72 inches) the participant’s own screen sizes were largely insignificant. However, this was still taken into account in the analysis of the results.

## Participants

Participants selected were over the age of 18, held a valid driver’s license, and were fluent in English. There were 42 participants recruited to take part in this study. Participants could choose to withdraw from the test at any time while taking it, but once they finished the test, their personal data was no longer stored with their responses, so they could no longer withdraw their responses.

## Procedures

The study consisted of two separate website-based tests. Each participant was given a unique username and password, which allowed them to log on to the website to complete the tests. Participants were instructed to complete the first test, referred to as the “Letters Test” prior to completing the second test, referred to as the “Words Test”. They were instructed to use the same username and password for both tests.

Participants were provided with a copy of the Informed Consent Form, found in Appendix 2, to which they verbally consented to participate in the study. They were then given an overview of the study, and how the data collected would be used.

The study was to be completed on a personal laptop. Once the participant logged onto the web test, there was a set of instructions that the participant had to read. These instructions can be found in Appendix 4. Once the participant was ready, they could begin the test. Both tests began with three “examples” which did not count towards the final data analysis. This allowed the users to understand the environment and testing conditions, and to be prepared for the rest of the test.

In the “Letters Test”, participants would see a character appear on screen, slowly getting bigger to mimic the participant driving towards it. Once the participant could clearly read the character, they would click the green button at the top of their screen. This would stop the test, and the size at which the participant could see the character would be recorded. The participant would then have to select if it was a letter or a symbol, as some symbols were introduced to preserve the integrity of the test. Once the participant indicated which character they saw on screen, they were given the option to go to the next character. There were 43 different characters in the first test, representing three different fonts. The participant could choose to stop the test at any time.

The “Words Test” was similar to the first test but used 6-letter words in place of characters. The words chosen for this study were modeled after Ruano Duke’s choices [18]. Similar to the first test, a word would appear on the screen, slowly getting larger to mimic the effect of driving towards it. Once the participant could read the word, they would press the green button on the screen, pausing the test and recording the size of the word. They would then type out the word they had just seen on the screen. Once they had entered the word, they were given the option to go to the next word. There were 40 different words, representing 4 different fonts, and the participant could choose to stop the test at any time.

## Data Analysis

### Overview

A variety of different data points were collected throughout this test. Not only were different fonts analyzed against each other, but also the contrasting screens they were presented on. Each individual letter and word tested was also analyzed against each other, to see which font was complimentary to it. To be able to complete a thorough analyses, a variety of techniques were used.

## Techniques Used

For the analysis of this study, all of the alternative fonts were compared to the base font. This allowed for the use of a t-test, a statistical test used to compare the means of two different groups. The information used for the analysis of the data is as follows:

The average reduction in size. This is the average base size minus the average alternative size. A positive value means the alternative size improved; a negative value indicates the base font performed better.

The p-value of the t-test, which measures the likelihood of finding a similar difference within a different population. A lower p-value suggests a higher significance and attributes the results to the different fonts, while a higher p-value considers the results to be non-significant.

The average improvement of the alternative fonts over the base font, which is calculated as the base font divided by alternative font minus 1. This results in a percentage number, in which a positive number denotes an improvement, and a negative number denotes a setback. This calculation is done by analyzing each individual data point, not the final average, which explains why it sometimes is different from the average reduction in size.

The first graph analyzed is the alternative font size graphed against the base font size. This graph analyzes each user's data, comparing the same characters or words in the fonts provided. Through ordinary least squares, a linear regression is graphed to suggest the improvement of the alternative font over the base. Like before, a negative result suggests the base performed better, and a positive result suggests the alternative font performed better.

The second figure created graphs the alternative font's change score relative to the starting base font size. In other words, this graph shows the alternative font's size when compared to its base font counterpart. To indicate an advantage in the alternative font, data points have to be below 0%. Data points above this threshold indicate that the base font had a smaller size than its alternate counterparts.

The third figure is the frequency distribution of size between the two compared fonts. This allows for the easy visualization of the more legible font. If the curve drawn is more to the left than its counterpart, it means that it was the more legible font. The frequency on the y-axis indicates an average of the percentage of characters or words in each participant's test that could be correctly guessed at the size shown in the x-axis.

Lastly, the last figure shows the change score of the alternative fonts relative to the base. This histogram is derived from the data in the second figure and shows the normal distribution of the change score data. A left-skewed distribution has a left tail, and a right-skewed has a right tail. A left-skewed distribution indicates that the alternative outperformed the base font, and a right-skewed distribution indicates the opposite is true. In a normal distribution, both sides of the curve are symmetrical to one another, with most observations occurring at the peak.

## Results

### Overview

In total, 42 participants are included in the study. After collection, the raw data was filtered to exclude any outliers. Following an initial overview of the data for the first test, letters with a font size smaller than 5 and larger than 25 points were excluded from the final analysis.

This cleanup ensured that any data points in which the user just recognized the shape of a letter instead of fully visualizing or the user forgot to pause the test were deleted. In the second test, the parameters were set to 6 points and 25 points, to ensure the same “clean” data was being analyzed. The lower end of the outlier parameter was higher for this test to account for the longer time it would take for a user to be able to read and process a word. After cleaning the data, 1,534 data points were collected for the first test (“Letters Test”), and 1,632 data points for the second test (“Words Test”). Ideally, all of the fonts within the tests would have had the same number of observations each, however, the web-based test would glitch at times, causing the users to unknowingly repeat the same character and font. Due to this, some fonts have a larger amount number of observations than others.

## Letters Test

### All Letters

Table 4: Data for all of the Letters Used

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	529	10.89	10.50	3.38
D-altered Font	512	10.94	10.50	3.44
Narrow Font	493	10.76	10.33	3.36

Table 4 shows the major data points for the first test, including the observations, average size, median size, and standard deviation. The base font had the largest number of observations, followed by D-altered and then narrow fonts. The narrow font had the smallest average size, followed by the base, and D-altered fonts.

The standard deviation was highest for the D-altered font. For this font, the median size was 10.50 points, with a standard deviation of 3.44. The base font also had a median size of 10.50, with a standard deviation of 3.38. Lastly, the narrow font had a median size of 10.33, with the smallest standard deviation at 3.36.

In this test, there was a total of 32 incorrect letter guesses, or around 2.08%. The base and narrow fonts tied at 13 incorrect guesses each, and the D-altered font had 6 incorrect guesses. The symbols, or the characters used to preserve the integrity of the test, had 9 incorrect guesses, but they were omitted out of the overall analysis.

The data was also divided between the positive and negative contrast and analyzed within those parameters. For this test, there were 792 data points collected in a positive contrast and 742 collected in a negative contrast.

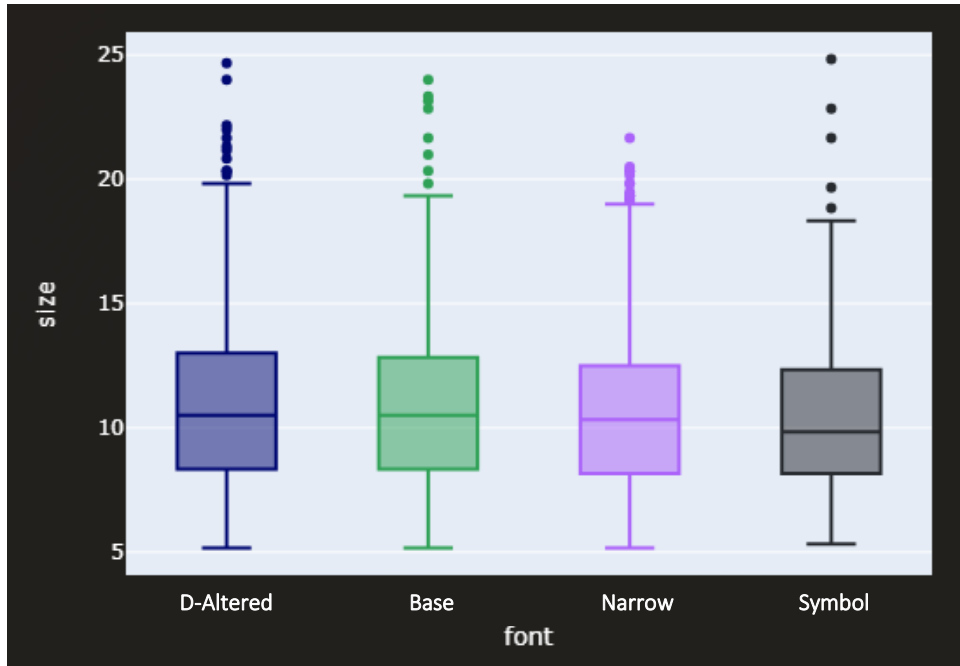


Figure 9: Size of Each Font Type for All Letters

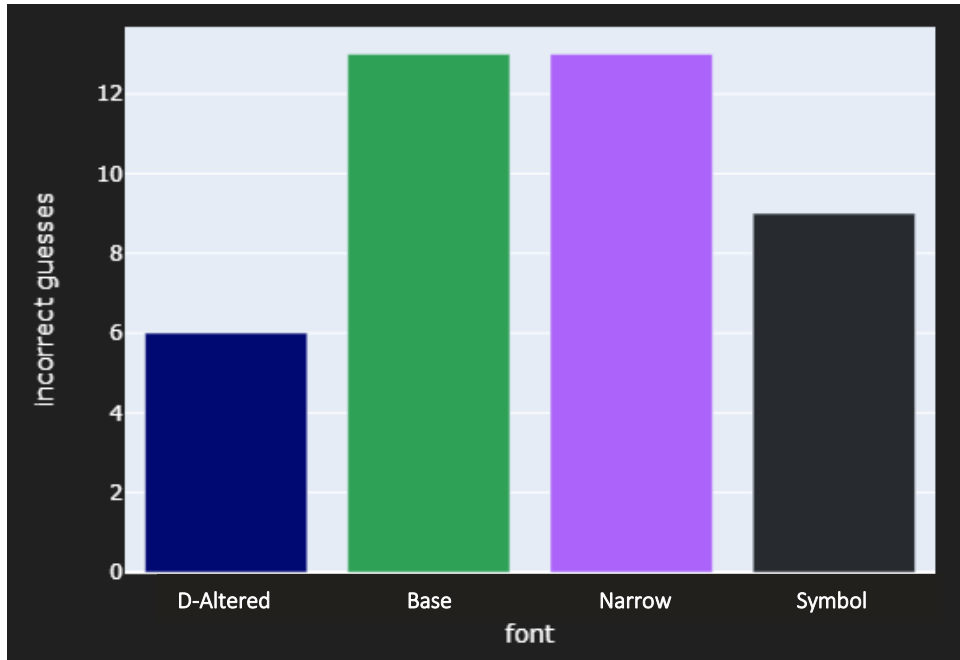


Figure 10: Incorrect Guesses of Each Font Type for All Letters

## Base v D-altered

The first comparison in this test was done between the base and D-altered fonts. The average reduction in size for these fonts was -0.05 points. This indicates that the base font performed slightly better than the D-altered font, and when analyzing it on an individual basis, this trendline indicated a negative improvement of 4.6% as seen in Figure 11.

The p-value of the t-test was found to be 0.600, which does not suggest significance. The change score for these fonts can be visualized in Figure 12, which shows some outliers in the top right quadrant. The frequency distribution, Figure 13, for both fonts was almost identical, with the base font peaking at a slightly higher frequency at 12%, and the D-altered font peaking slightly below that. The change score and frequency distribution, Figure 14, shows a right-skew to the data.

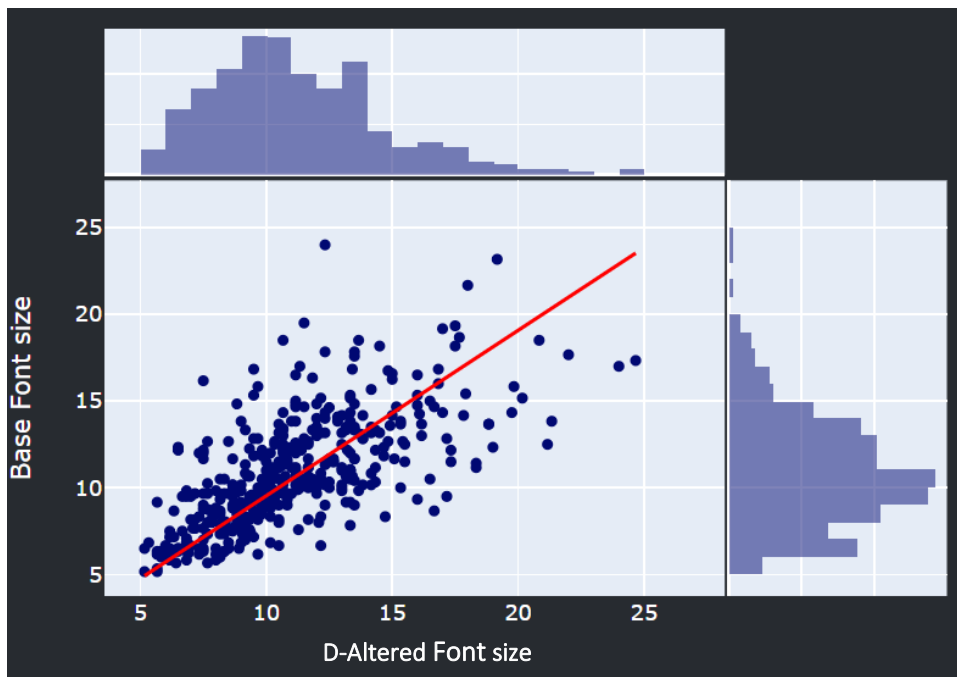


Figure 11: D-Altered Font Size Versus Base Font Size

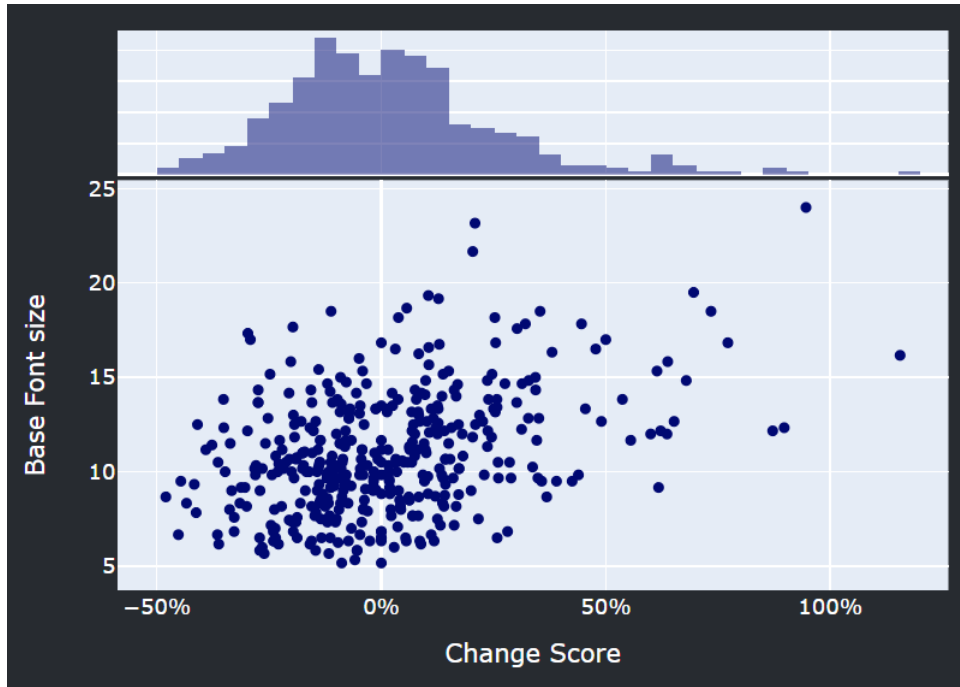


Figure 12: D-Altered Font Change Score Relative to Starting Base Font Size

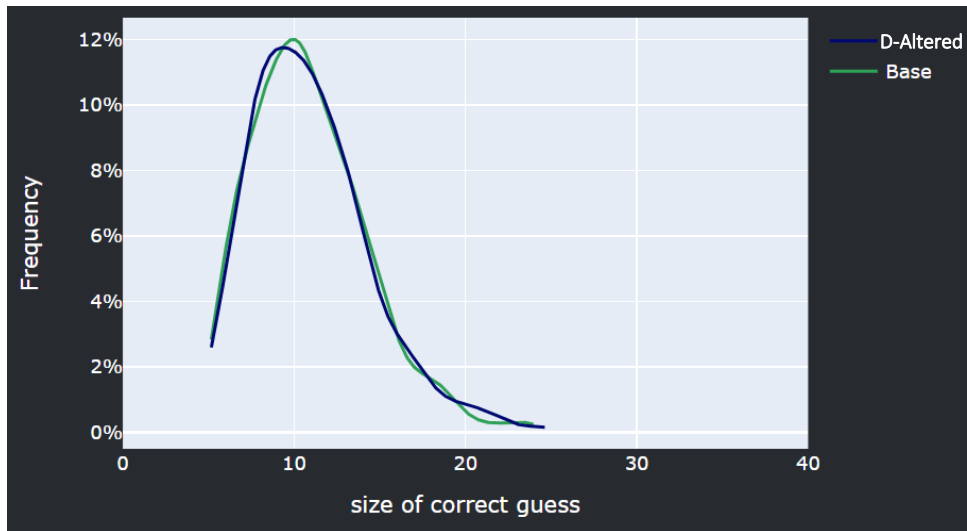


Figure 13: Frequency Distribution of Size for D-Altered and Base Fonts

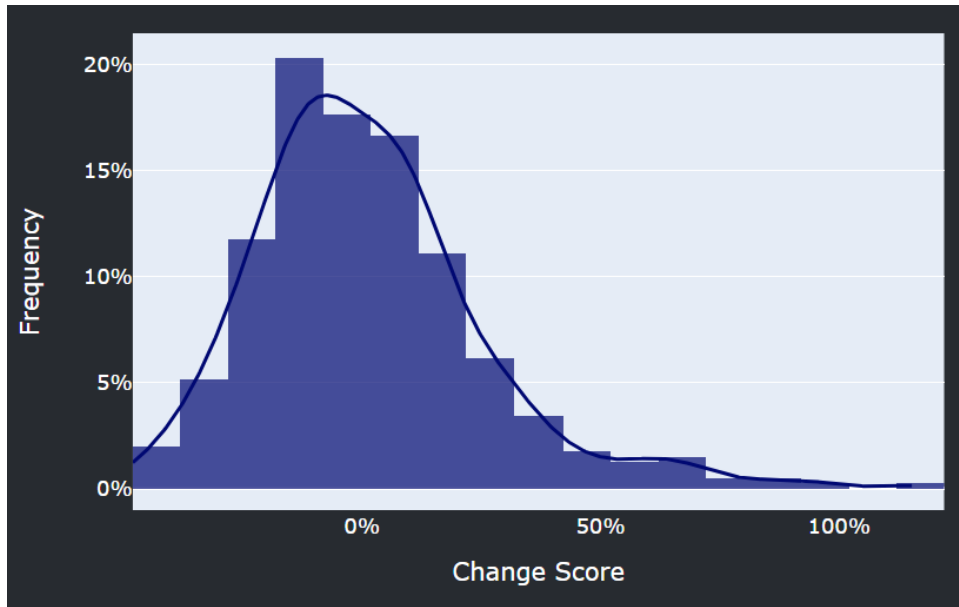


Figure 14: Change Score of D-Altered Font Relative to Base Font

#### Base v Narrow

The average reduction in size for the narrow versus base fonts was 0.13 points, which indicates the narrow font had a smaller average size than the base font. When analyzed on a graph, this trendline indicates a negative improvement of 2.2% for the narrow font over the base, as seen in Figure 15. The difference between the average reduction in size and the graphed improvement is due to the outliers in the ordinary least squares model, which can be seen in the graph.

The p-value of the t-test was found to be 0.267, which suggests a slightly larger significance than the comparison between the D-altered and base fonts. The change score shows some outliers in the data in Figure 16. The frequency distribution in Figure 17 shows that the narrow font performed better overall than the base. The change score and frequency in Figure 18 show a right-skewed data distribution.

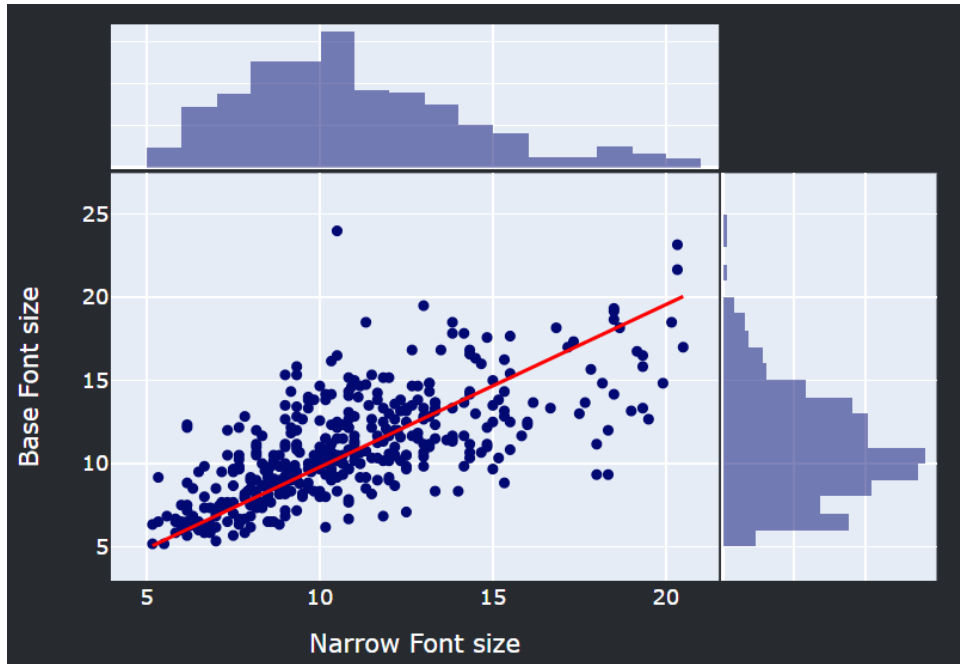


Figure 15: Narrow Font Size Versus Base Font Size

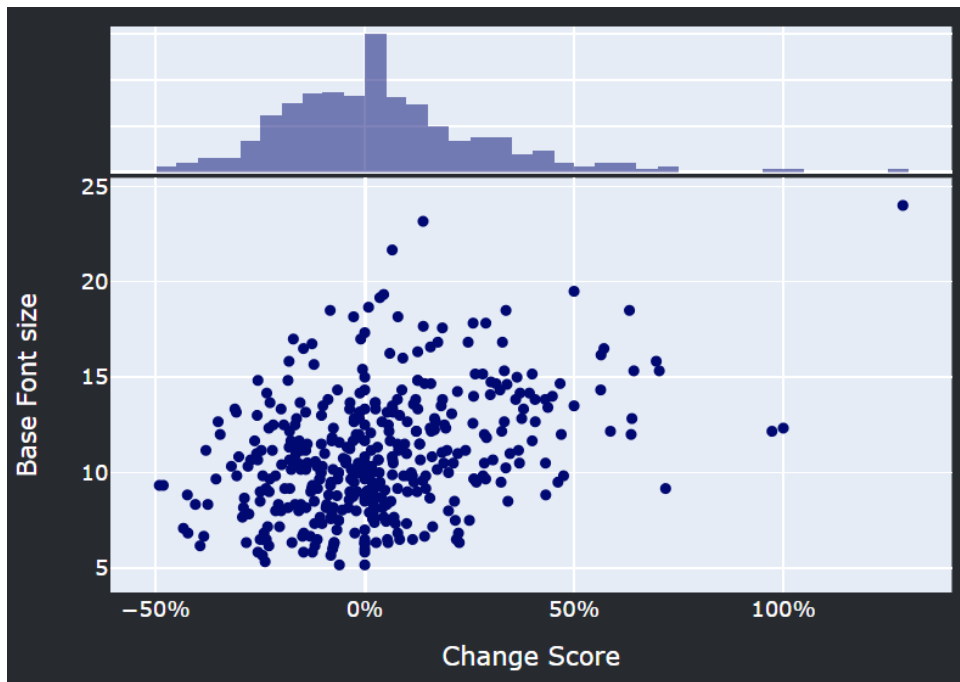


Figure 16: Narrow Font Change Score Relative to Starting Base Font Size

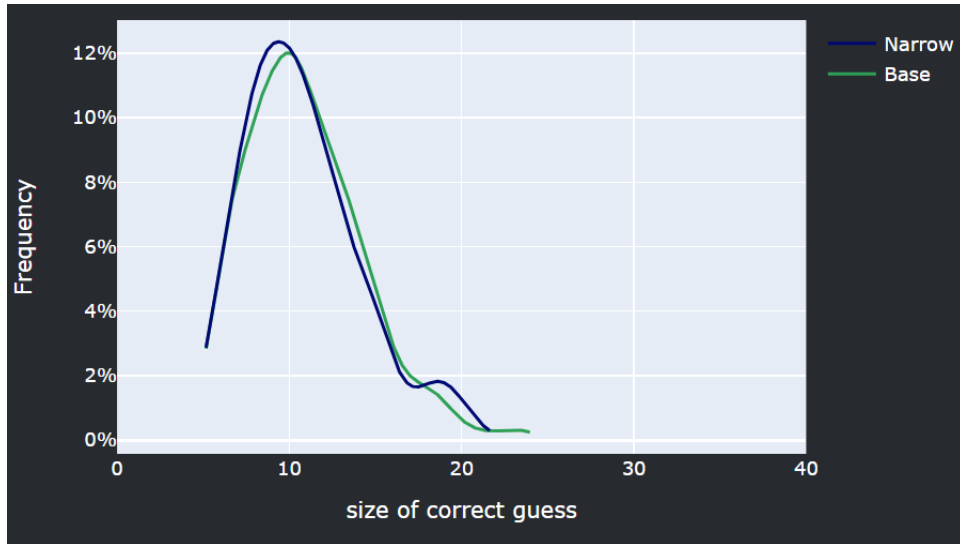


Figure 17: Frequency Distribution of Size for Narrow and Base Fonts

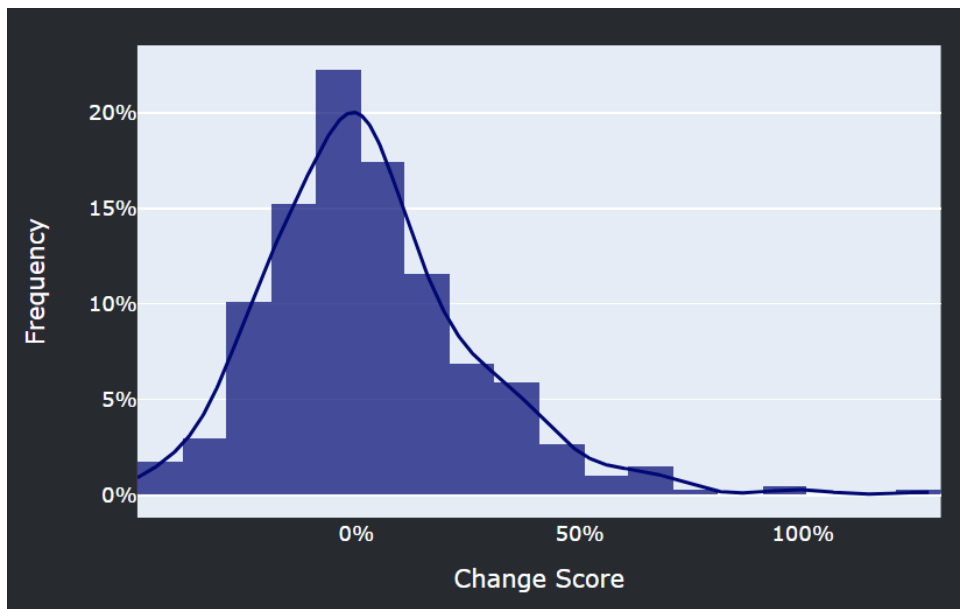


Figure 18: Change Score of Narrow Font Relative to Base Font

All Letters in Positive Contrast

Table 5: Data for all of the Letters Used in Positive Contrast

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	283	11.25	10.67	3.56
D-altered Font	266	11.37	11.00	3.64
Narrow Font	243	11.02	10.50	3.33

Table 5 shows all of the data points collected in a positive contrast. Out of the 792 data points, 283 were observed in the base font, 266 in the D-altered font, and 243 in the narrow font. These results were similar to all of the letters, with the narrow font having the smallest average size, followed by the base font and lastly the D-altered. The differences between these average sized were slightly larger than those of the initial analysis.

Similar to the analysis with both contrasts, the base outperformed the D-altered font by 0.12 points, and the narrow outperformed the base by 0.23 points. The p-value of the t-test for the D-altered and base fonts was 0.650 and the base and narrow fonts had a value of 0.224. s

The D-altered font had a median size of 11.00, with a standard deviation of 3.64 and the largest number of outliers. The median sizes for the base and narrow fonts were 10.67 and 10.50, respectively. The base font had 5 outliers, and the narrow font had just one. The median value for the symbols in a positive contrast was 9.83, and it had just 2 outliers.

The letters in positive contrast had a total of 10 incorrect guesses, or just 31.25% of the total amount of incorrect guesses in the test. The narrow font had the highest number of incorrect guesses, at 6, followed by the base and D-altered fonts at 2 each. The symbols in positive contrast had 5 incorrect guesses in total.

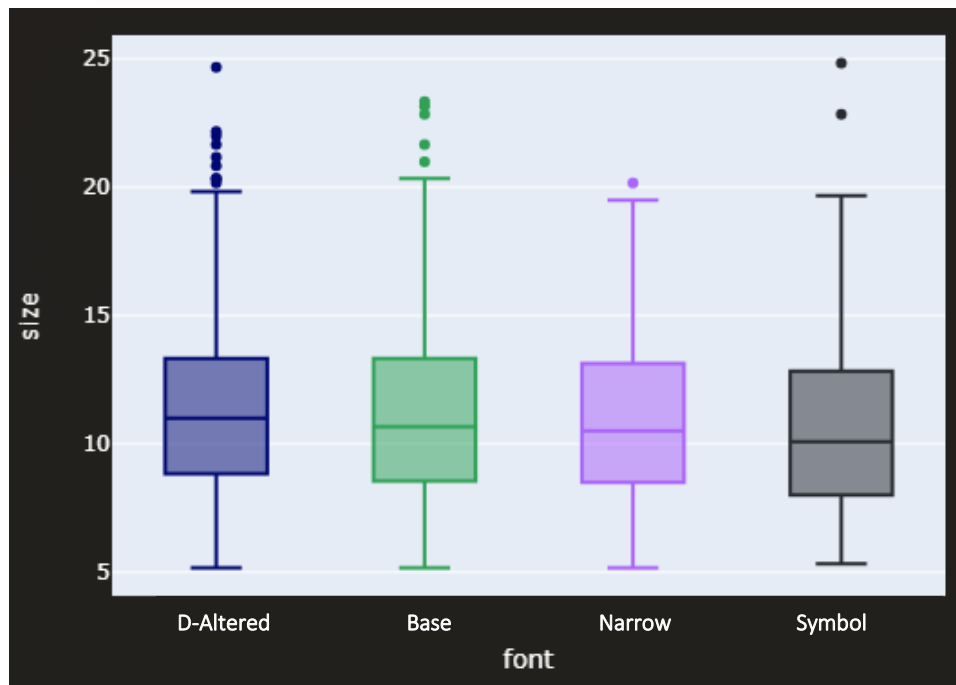


Figure 19: Size of Each Font Type for All Letters in Positive Contrast

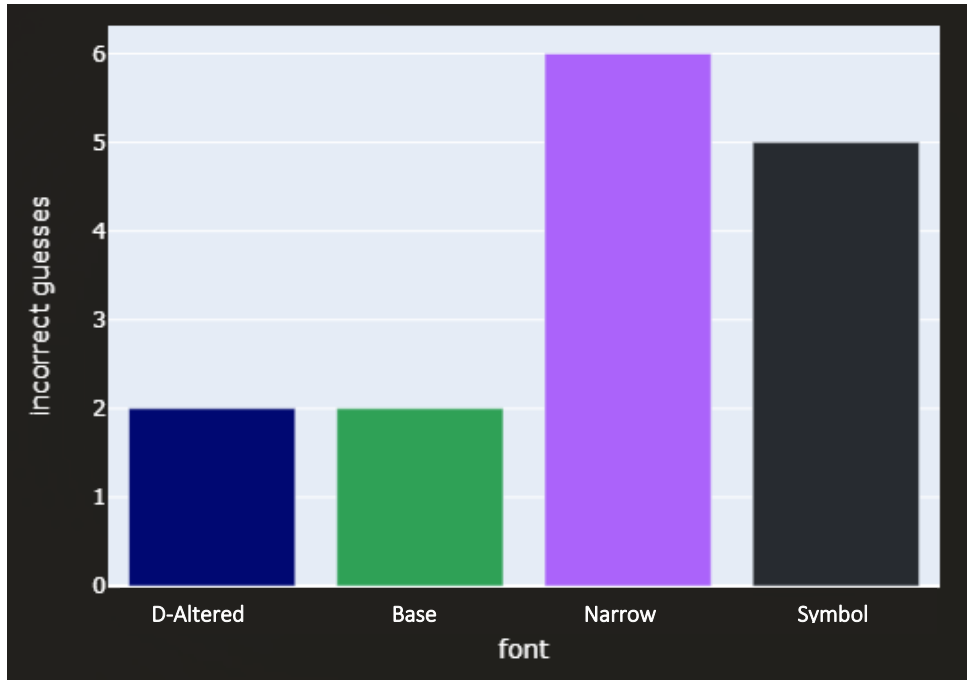


Figure 20: Incorrect Guesses of Each Font Type for All Letters in Positive Contrast

All Letters in Negative Contrast

Table 6: Data for all of the Letters Used in Negative Contrast

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	246	10.47	10.17	3.10
D-altered Font	246	10.48	10.00	3.15
Narrow Font	250	10.50	10.00	3.38

Table 6 shows all of the data collected in a negative contrast. Out of the 742 data points collected, 250 were in narrow font, and 246 in both the base and D-altered fonts. Unlike the previous two analyses, the base font was the best-performing, closely followed by the D-altered and narrow fonts. The average size for fonts in negative contrast was also much smaller than those in positive contrast, by an average of 0.73 points.

In negative contrast, the base did better than the D-altered font by just 0.01 points, and the narrow font by 0.03 points. The p-value of the t-test for the D-altered and base fonts was 0.512 and the base and narrow fonts had a value of 0.540. This indicates that neither of the two comparisons was statistically significant.

The D-altered and narrow fonts both had a median value of 10, with a standard deviation of 3.15 and 3.38 respectively. The narrow font had the most outliers, followed by the D-altered and then base fonts. The median for the base font was slightly higher than the others at 10.17

points, albeit the average size was the smallest. The standard deviation of this font was also the smallest, at 3.10.

The letters in negative contrast had a total of 22 incorrect guesses, or 68.75% of the total. The base font had 11 incorrect guesses, the narrow font had 7, and the D-altered font had 4. Although the average size of the letters on negative contrast was lower than those on positive contrast, the errors were much higher. The symbols in negative contrast had 4 incorrect guesses in total.

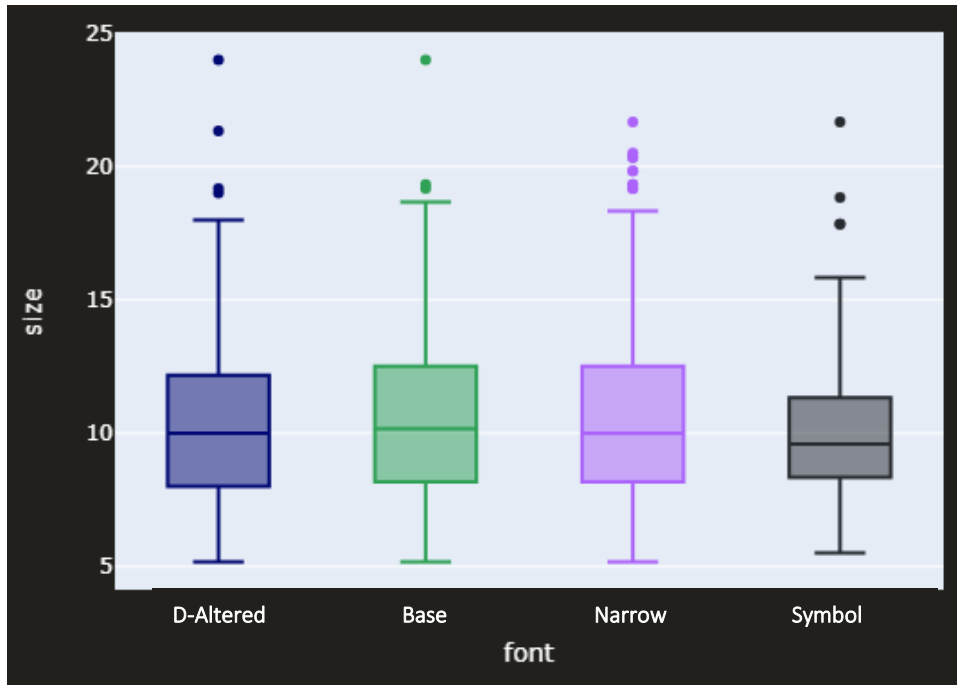


Figure 21: Size of Each Font Type for All Letters in Negative Contrast

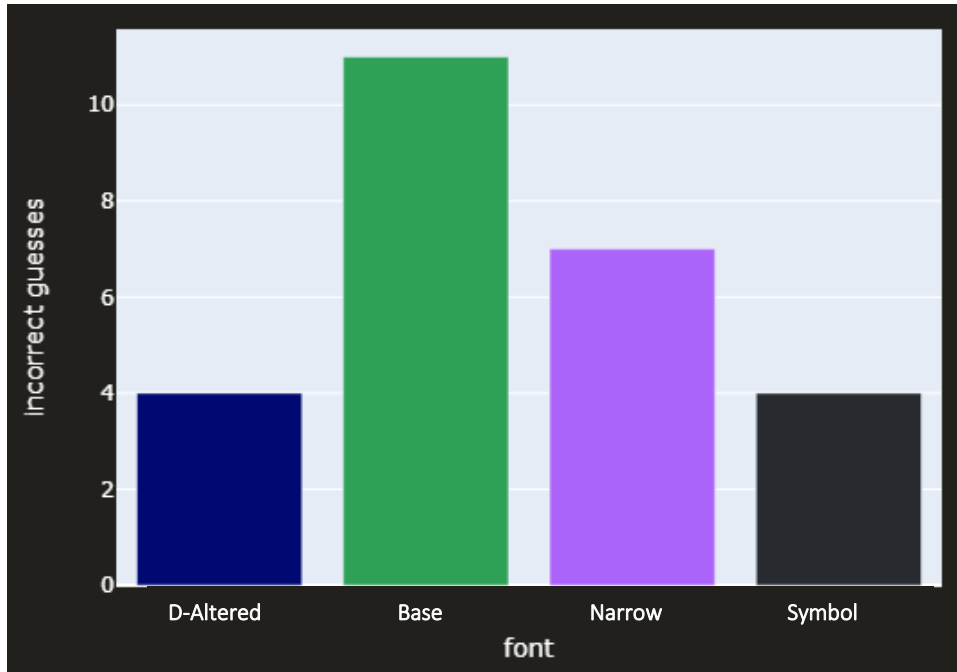


Figure 22: Incorrect Guesses of Each Font Type for All Letters in Negative Contrast

## Words Test

All Words

Table 7: Data for all of the Words Used in the Test

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	451	9.97	9.50	2.87
D-altered Font	418	10.56	10.00	2.99
Narrow Font	380	10.09	9.50	2.83
C-altered Font	383	11.64	11.00	3.31

In this test, 3 different fonts were compared against the base font. Similar to the letters test, the majority of the observations happened in the base font, followed by the base, D-altered, C-altered, and narrow fonts, and this data can be seen in Table 7.

For the words test, the base font performed the best, closely followed by the narrow font, and then the D-altered and C-altered fonts. The C-altered font performed 1.67 points worse than the base font, the biggest difference yet. The base and narrow fonts both had a median size of 9.50, followed by the D-altered and C-altered fonts, with a median size of 10 and 11 respectively. The narrow font had the smallest standard deviation, at 2.83, followed by the base, D-altered, and C-altered fonts at 2.87, 2.99, and 3.31 correspondingly.

There was a total of 73 incorrect word guesses, or around 4.47% of the total data points. The base had the largest number of incorrect guesses at 22, then the narrow font at 18, and lastly

the C-altered and D-altered fonts at 17 and 16 each. The D-altered, C-altered, and narrow fonts were compared against the base font for the study.

This data was also divided between positive and negative contrasts to be analyzed within the parameters. There were 876 data points collected in a positive contrast and 756 collected in a negative contrast. These data points would allow for the analysis of different fonts on each contrast, to see which led to a clearer output.

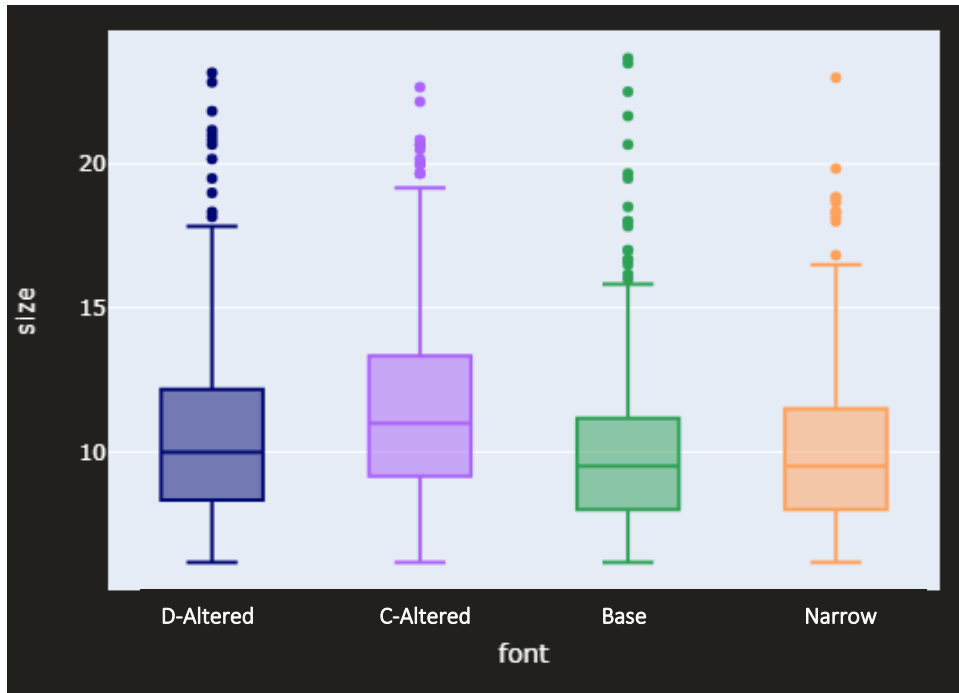


Figure 23: Size of Each Font Type for All Words

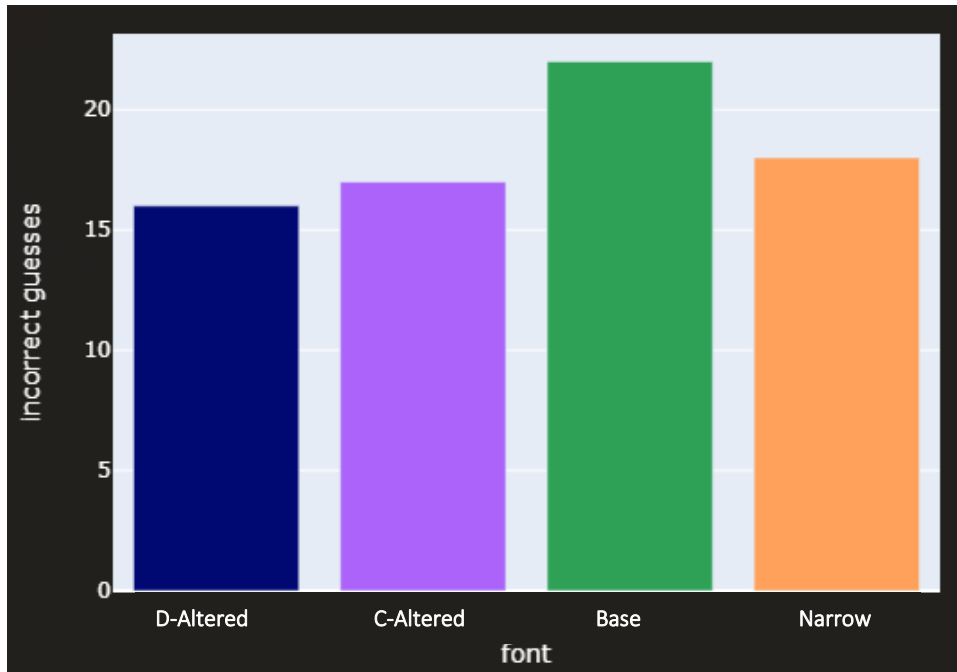


Figure 24: Incorrect Guesses of Each Font Type for All Words

#### Base v D-altered

The average reduction in size for the D-altered versus base fonts was -0.59 points. In this test, the base font performed better than the D-altered font. The linear regression model shows a negative 8.4% trendline as seen in Figure 25.

The p-value of the t-test was found to be 0.999, which shows that the difference between these fonts was insignificant. The change score for these fonts can be visualized in Figure 26, with most of the data points falling under the 0% mark. The frequency distribution for both fonts was similar, showing that the base font performed slightly better than the D-altered font overall. This graph can be seen in Figure 27. Lastly, the change score and frequency curve of the D-altered font relative to the base can be seen in Figure 28, and it shows a uniformly distributed curve with a slight right-skew.

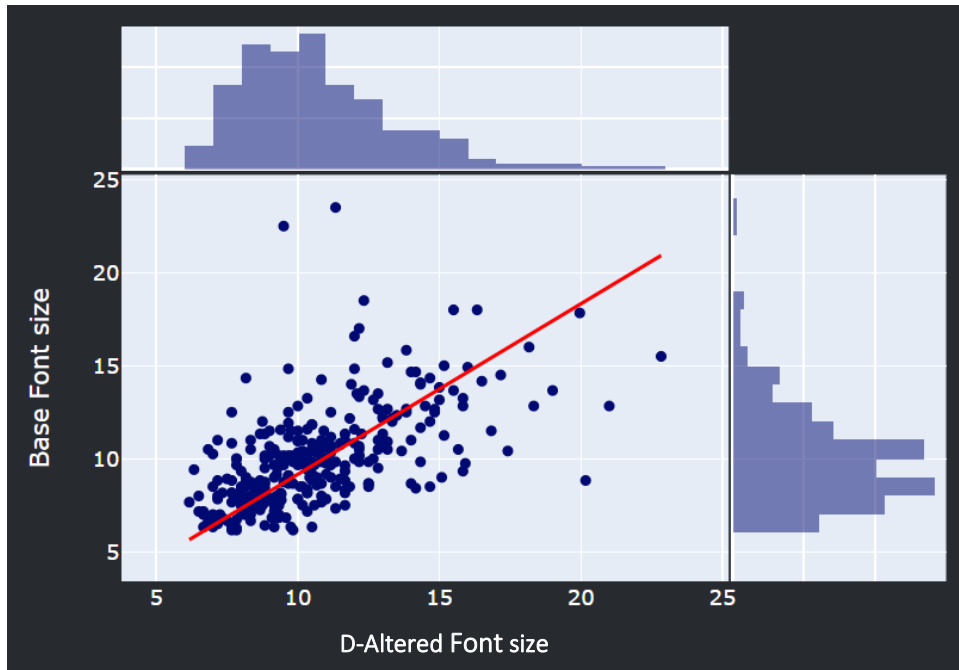


Figure 25: D-Altered Font Size Versus Base Font Size

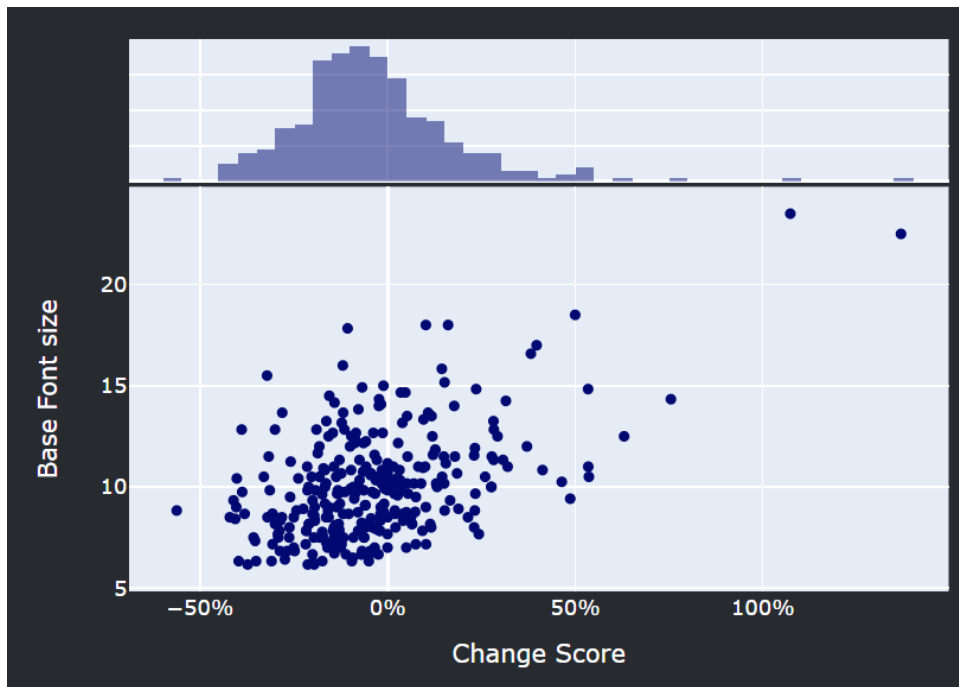


Figure 26: D-Altered Font Change Score Relative to Starting Base Font Size

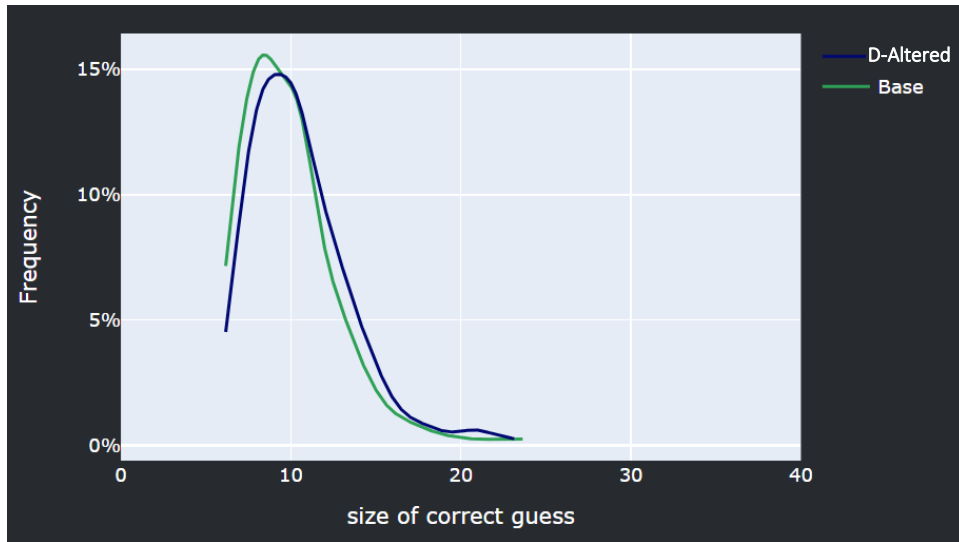


Figure 27: Frequency Distribution of Size for D-Altered and Base Fonts

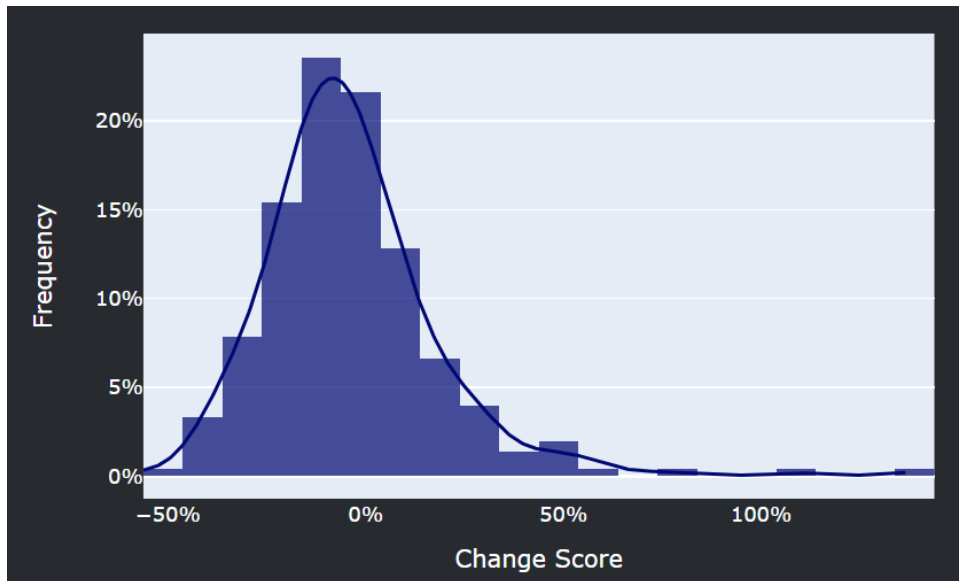


Figure 28: Change Score of D-Altered Font Relative to Base Font

### Base v Narrow

The average change in size for the base and narrow fonts was -0.13 points. Although very similar in values, the base font slightly outperformed the narrow font. When analyzed using a linear regression model, this indicated a negative 2.6% overall improvement, as seen in Figure 29. The p-value of the t-test is 0.737, which suggests this difference to be highly insignificant. The change score for the fonts can be seen in Figure 30, with the majority of the points hovering around 0%. Figure 31, the frequency distribution, shows both fonts being almost identical. The change score and frequency in Figure 32 shows a normally distributed curve with a slight right-skew.

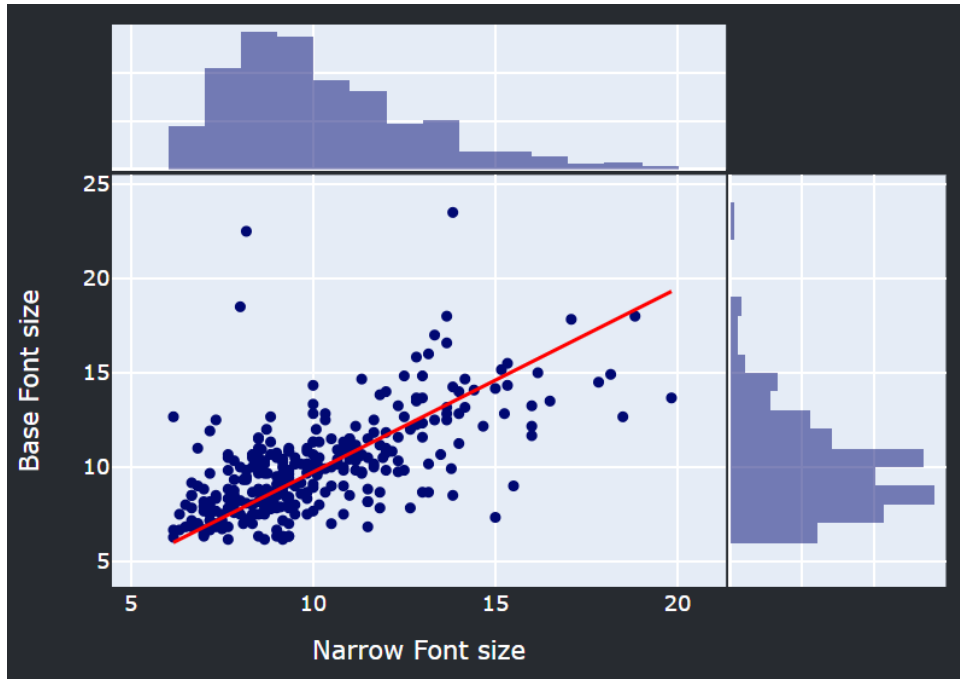


Figure 29: Narrow Font Size Versus Base Font Size

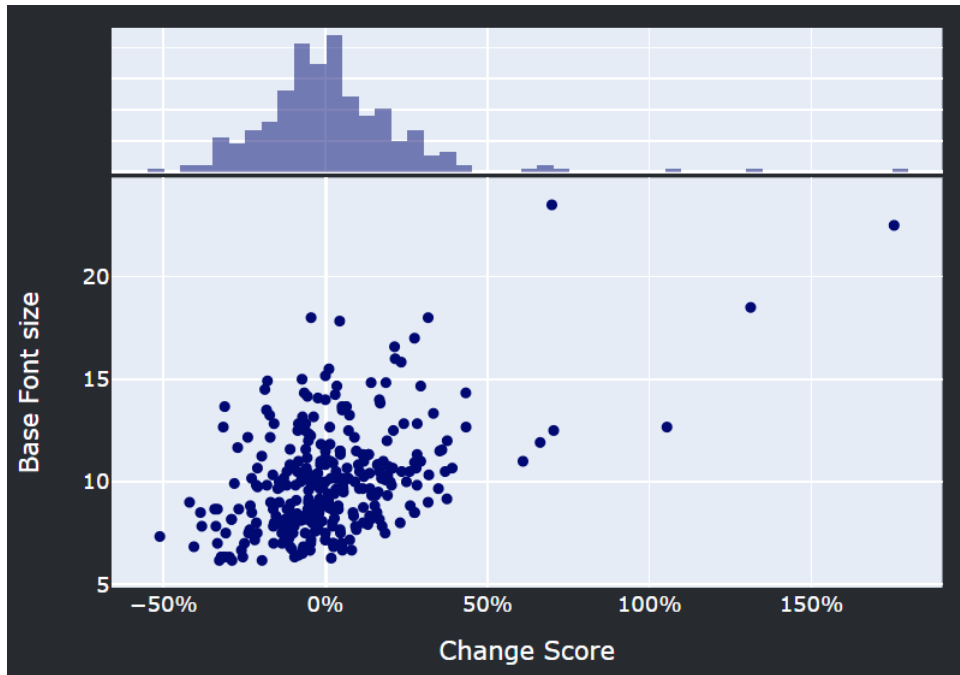


Figure 30: Narrow Font Change Score Relative to Starting Base Font Size

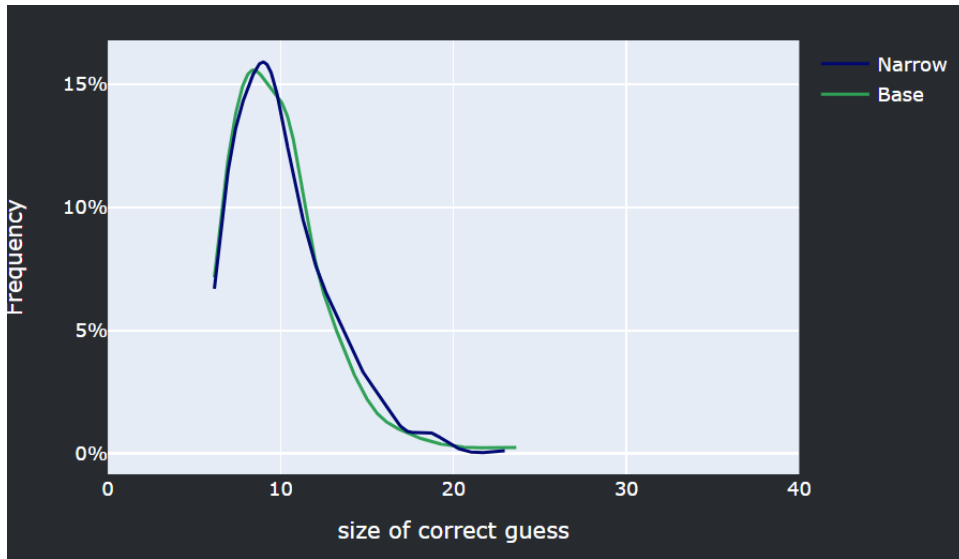


Figure 31: Frequency Distribution of Size for Narrow and Base Fonts

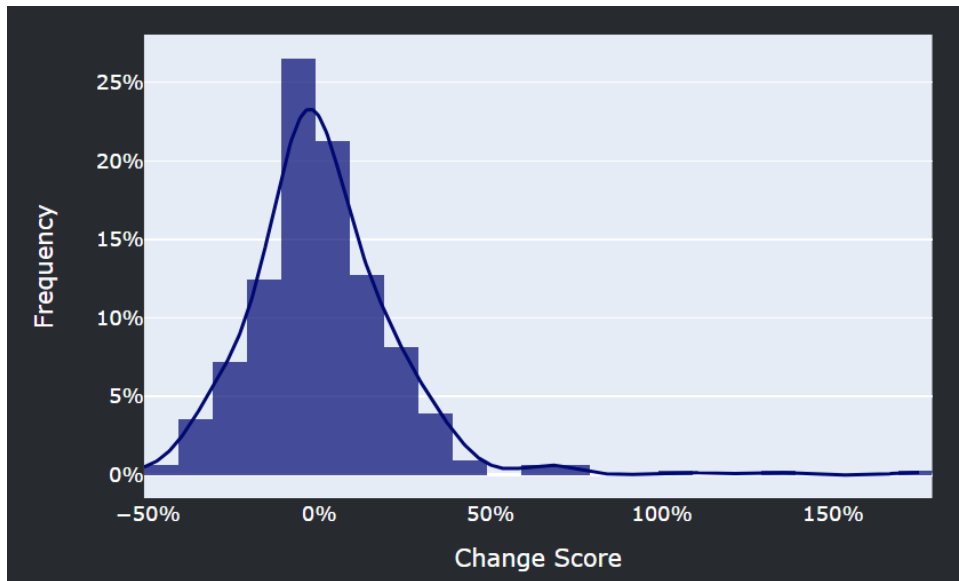


Figure 32: Change Score of Narrow Font Relative to Base Font

### Base v C-Altered

The average change in size for the C-altered and base fonts was -1.67 points with a p-value of 1.0. The p-value indicated this change to be insignificant. In this test, the base font outperformed the C-Altered font by the largest margin yet. When analyzed using the linear regression model, it yielded a negative improvement of 17.5%, visualized in Figure 33. Figure 34 shows the graph of the change score, in which most data points fall below 0%. The frequency distribution in Figure 35 also shows the dramatic improvement the base font has over C-altered, with the base font leading the left of the graph. Lastly, Figure 36 shows a normally distributed curve, with a slight left-skew.

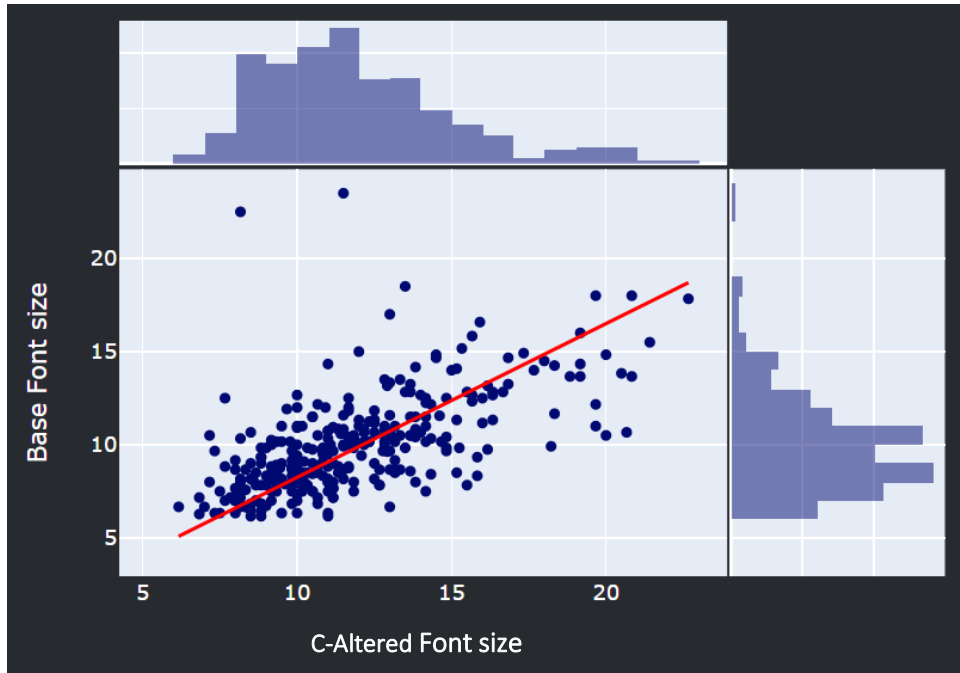


Figure 33: C-Altered Font Size Versus Base Font Size

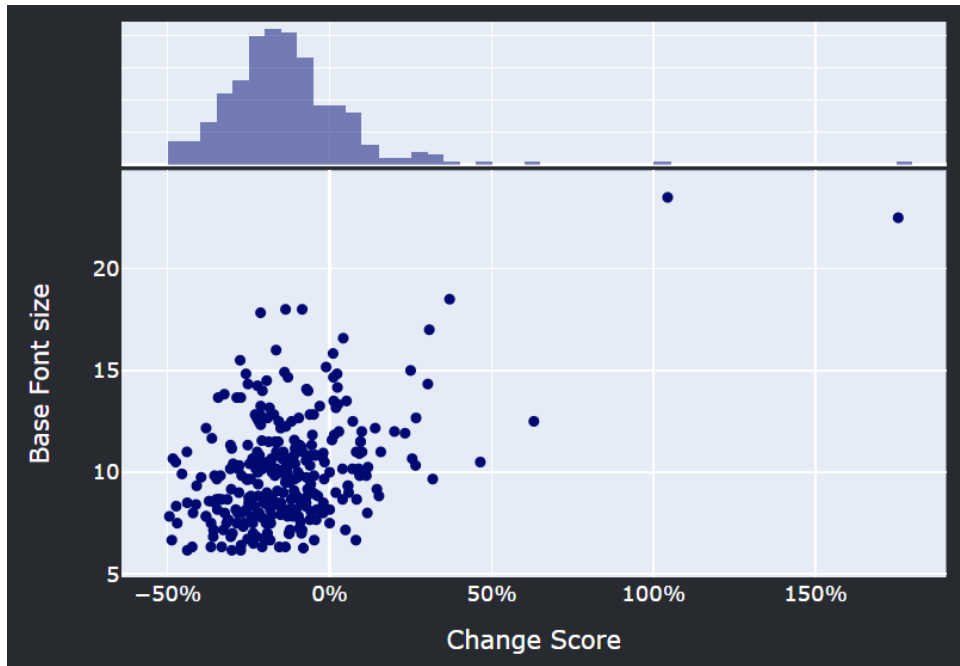


Figure 34: C-Altered Font Change Score Relative to Starting Base Font Size

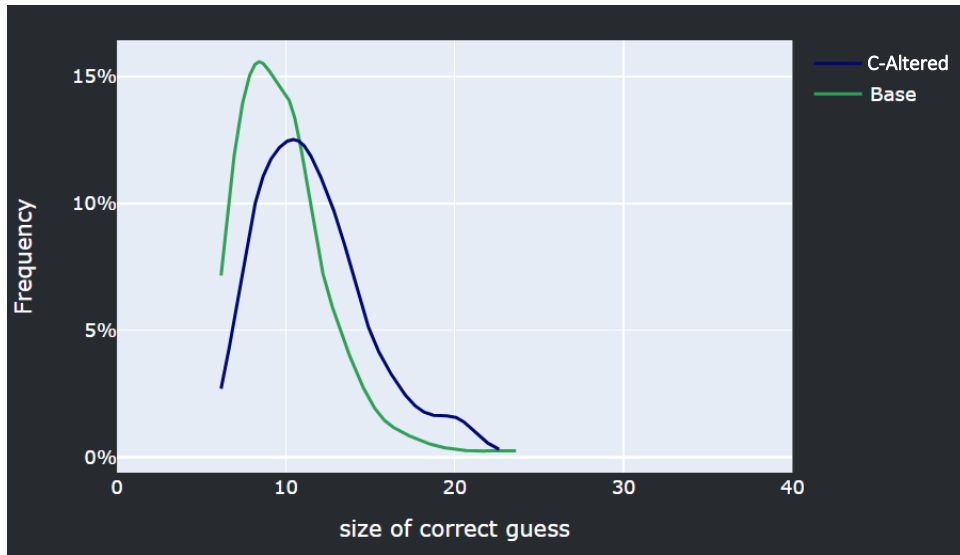


Figure 35: Frequency Distribution of Size for C-Altered and Base Fonts

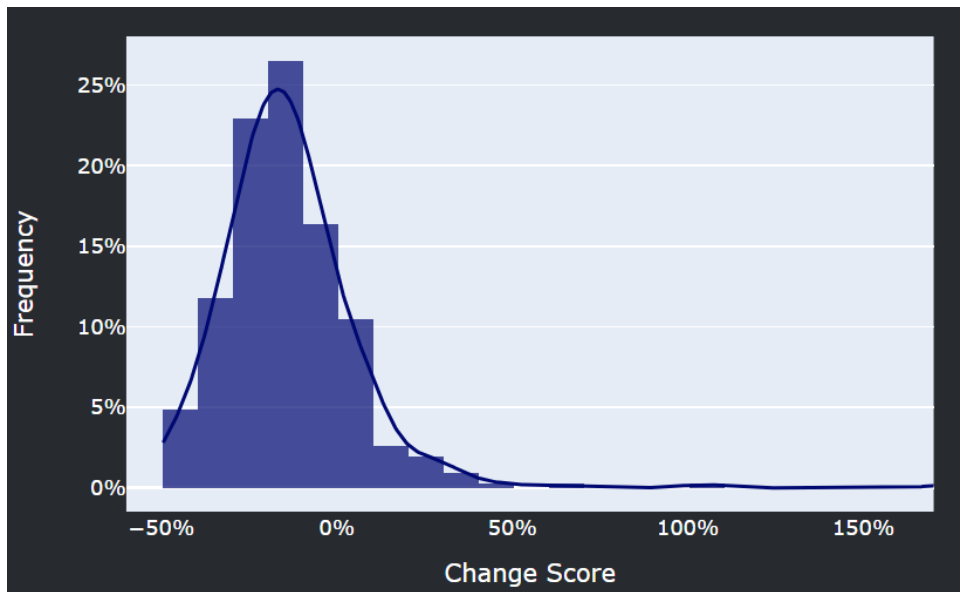


Figure 36: Change Score of C-Altered Font Relative to Base Font

## All Words in Positive Contrast

Table 8: Data for all of the Words Used in the Test in Positive Contrast

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	259	10.21	9.83	3.09
D-Altered Font	225	10.75	10.17	2.81
Narrow Font	205	10.42	9.83	2.96
C-Altered Font	187	12.05	11.33	3.59

Table 7 shows all of the data collected in a positive contrast in the second test. Out of the 876 data points collected, 159 were in base font, 225 in D-altered font, 205 in narrow font, and 187 in C-altered font. Out of this data, the best performing font was the base, closely followed by narrow, then D-altered, and lastly C-altered. C-altered trailed the previous 3 fonts by the largest margin yet.

In this contrast, the base outperformed the narrow font by 0.21 points, the D-altered font by 0.54 points, and the C-altered font by 1.84 points. The p-value of these tests was 0.769, 0.978, and 1.0 respectively, indicating the change in these was not significant.

The base and narrow fonts had the same median size value, at 9.83, followed by the D-altered font with a value of 10.17, and lastly the C-altered font at 11.33. The standard deviations were 2.81 for the D-altered font, 2.96 for the narrow font, 2.09 for the base font, and 3.59 for the C-altered font.

The words in negative contrast had 37 total incorrect guesses, or around 50.68% of the total for the second test. Out of these, the C-altered and base fonts had a total of 11 incorrect guesses each, followed by narrow with 8 and D-altered with 7.

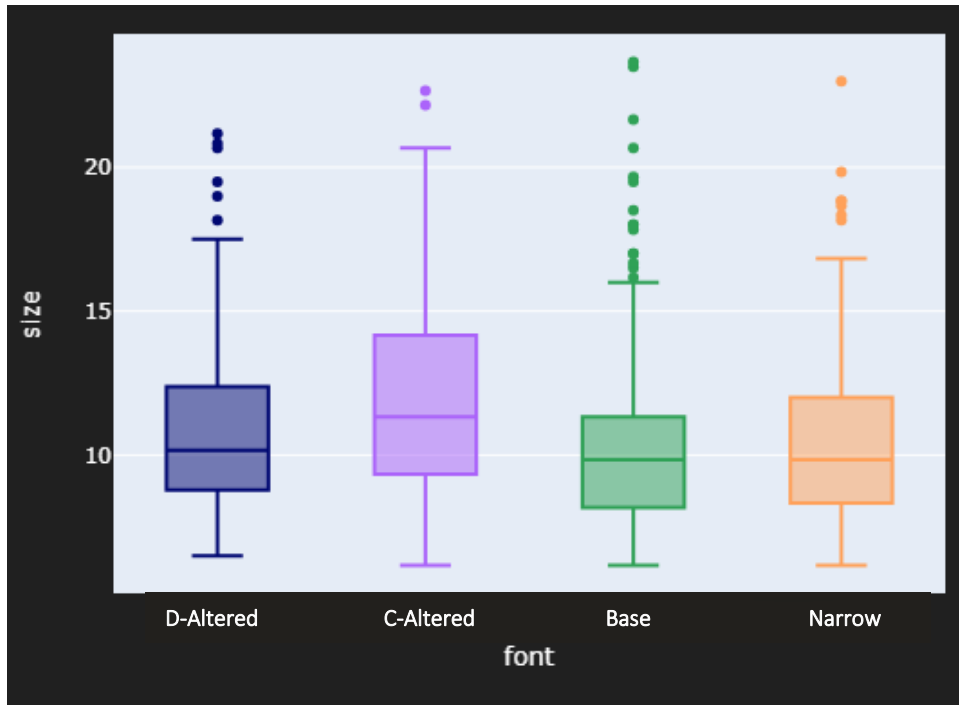


Figure 37: Size of Each Font Type for All Words in Positive Contrast

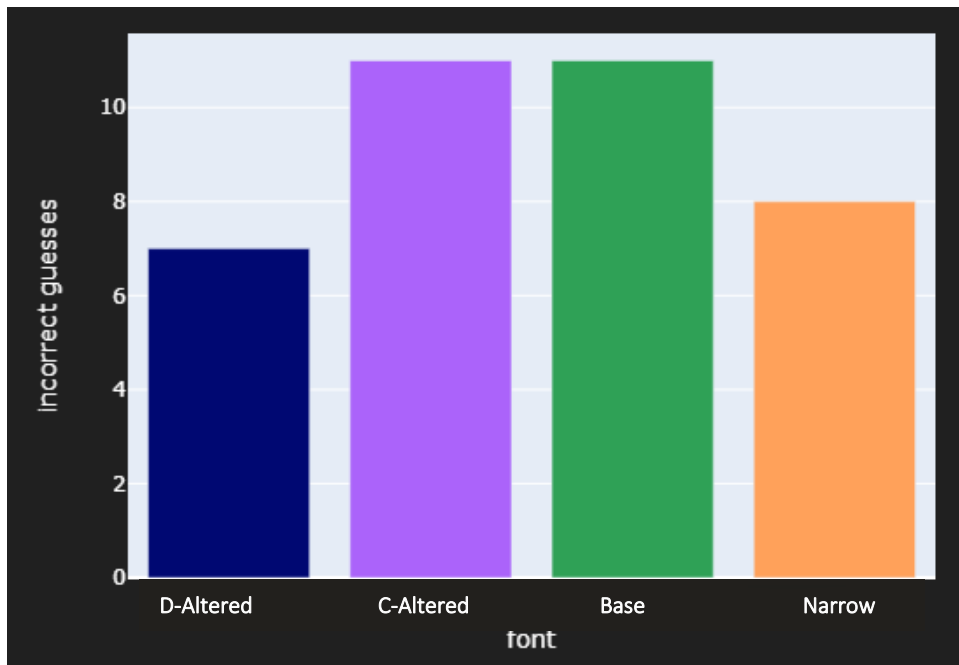


Figure 38: Incorrect Guesses of Each Font Type for All Words in Positive Contrast

## All Words in Negative Contrast

Table 9: Data for all of the Words Used in the Test in Negative Contrast

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	192	9.64	9.17	2.52
D-altered Font	193	10.34	9.83	3.17
Narrow Font	175	9.71	9.17	2.62
C-altered Font	196	11.24	11.00	2.97

Table 9 shows all of the data collected in a negative contrast in the words test. Out of the 756 collected data points, 196 were in the C-altered font, 193 in the D-altered font, 192 in the base font, and 175 in the narrow font. Similar to the data in a positive contrast, the base font was the best performer, closely followed by narrow, then D-altered and C-altered fonts.

In a negative contrast, the base outperformed the narrow font by 0.07 points, the D-altered font by 0.67 points, and the C-altered font by 1.6 points. The p-value of these tests was 0.605, 0.991, and 1.0 respectively. This shows that these results were not significant.

Like the positive contrast, the base and narrow fonts had the same median size value of 9.17, followed by the D-altered font with a value of 9.83. Lastly the C-altered font trailed with a value of 11.00. The standard deviations were 2.52 for the base font, 2.62 for the narrow font, 2.97 for the C-altered font, and 3.17 for the D-altered font.

The words in negative contrast had 36 total incorrect guesses, or around 49.32% of the total. Out of these, the base font had 11 incorrect guesses, followed by the narrow font with 10, D-altered with 9 and C-altered with 6.

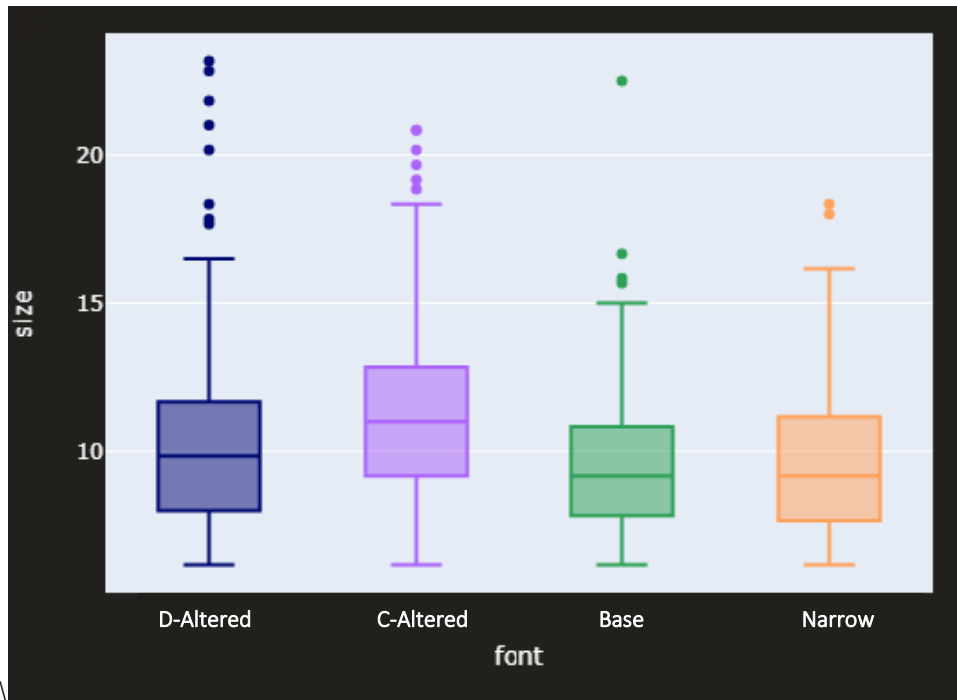


Figure 39: Size of Each Font Type for All Words in Negative Contrast

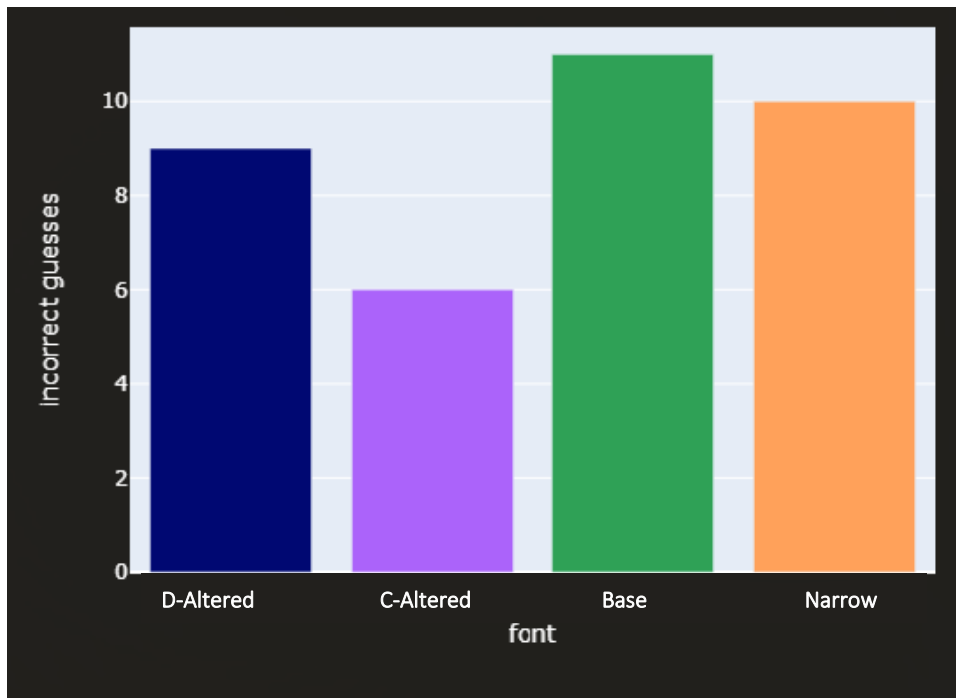


Figure 40: Incorrect Guesses of Each Font Type for All Words in Negative Contrast

## **Discussion**

### **Overview**

The results of both tests yielded a variety of different findings, which will be discussed in this section. To maximize the organization, the discussion will be covered in three different sections: letters, words, and key findings. Letters will dive into the findings of the first test, words will do the same with the second test, and key findings will synthesize both discussions to give a final recommendation.

### **Letters**

In the letters test, it was seen that, overall, the narrow font performed slightly better than the base and D-altered fonts. In general, the narrow font has thinner line strokes than its two competitors, making it more legible to participants. It also takes up less space, which would be beneficial for roadway signs and markings, as this font would take up less material than its counterparts. Overall, the average size for all three fonts were very similar to one another, so the letters were broken down into three categories for further analysis: capital letters (A, F), round letters (a, c, e, g, s), and stick letters (f, k, t, w, x, z).

## Capital Letters

Table 10 shows the data for just the capital letters of the first test. In this subgroup, the D-altered font outperformed all fonts. However, it was the narrow font that followed this, with the base font coming in last. The average sizes of the capital letters were over a point smaller than the rest, which could have led to a faster letter recognition than lowercase letters.

Figure 41 shows the boxplots of the capitals letters, in which the average sizes, median sizes, and standard deviations can be visualized. In this figure, it can be seen that the narrow font has the smallest standard deviation, with only 1 outlier. The D-altered font follows, with a slightly smaller median size and one outlier. Lastly, the base font has the least amount of consistency, hence the higher standard deviation and average size.

Table 10: Data for Capital Letters

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	111	9.74	9.00	3.51
D-altered Font	77	9.17	9.17	2.81
Narrow Font	76	9.19	9.00	2.57

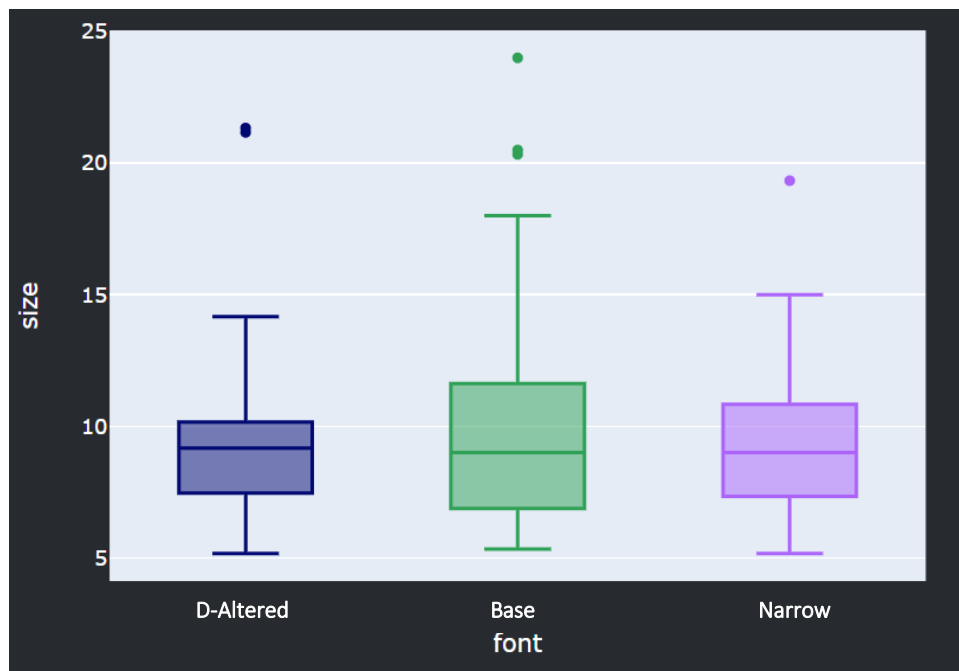


Figure 41: Size of Each Font Type for Capital Letters

## Round Letters

Table 11 shows the data for the round letters. Similar to the general data, the narrow form outperforms its counterparts. It is followed by the D-altered font, and then the base font. The median size of the narrow font is much smaller than the other two, by 1.17 and 2 points respectively.

Figure 42 shows that the narrow font has the smallest standard deviation and lowest median size, followed by the standard deviation of the base font. The D-altered font has the highest standard deviation, making it the least consistent out of the three fonts.

Within the 3 subgroups, the round letters seem to see the largest improvement when using the narrow font. This is likely due to the fact that the font's thinner strokes allow for more inter-character spacing, which, in turn, improved the overall legibility of the letter.

Table 11: Data for Round Letters

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	224	11.93	12.00	3.54
D-altered Font	190	11.58	11.17	3.64
Narrow Font	184	11.04	10.50	3.35

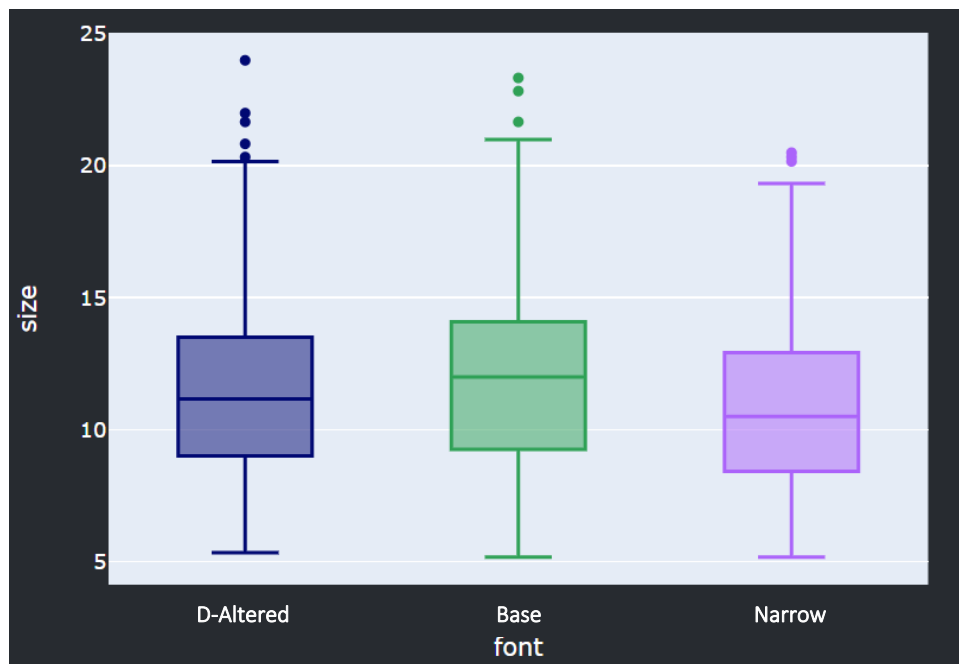


Figure 42: Size of Each Font Type for Round Letters

## Stick Letters

The last of the 3 subgroups is the stick letters. Like the capital letters, Table 12 shows that the D-altered font slightly outperformed the narrow font, and they both outdid the base font. And, like the capital letters, the average sizes were more similar to each other than the round letters. The similarities between the stick and capital letters is likely due to the fact that both of these sets of characters use straight lines, and do not have any inter-character spacing like the round letters.

Figure 43 shows the standard deviation and median sizes of the 3 fonts, with the base font being the most consistent with the smallest deviation. This is followed by the D-altered font, with the narrow font being the least consistent in this sub-category.

Table 12: Data for Stick Letters

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	226	10.48	10.17	2.90
D-altered Font	245	11.01	10.67	3.28
Narrow Font	233	11.05	10.33	3.47

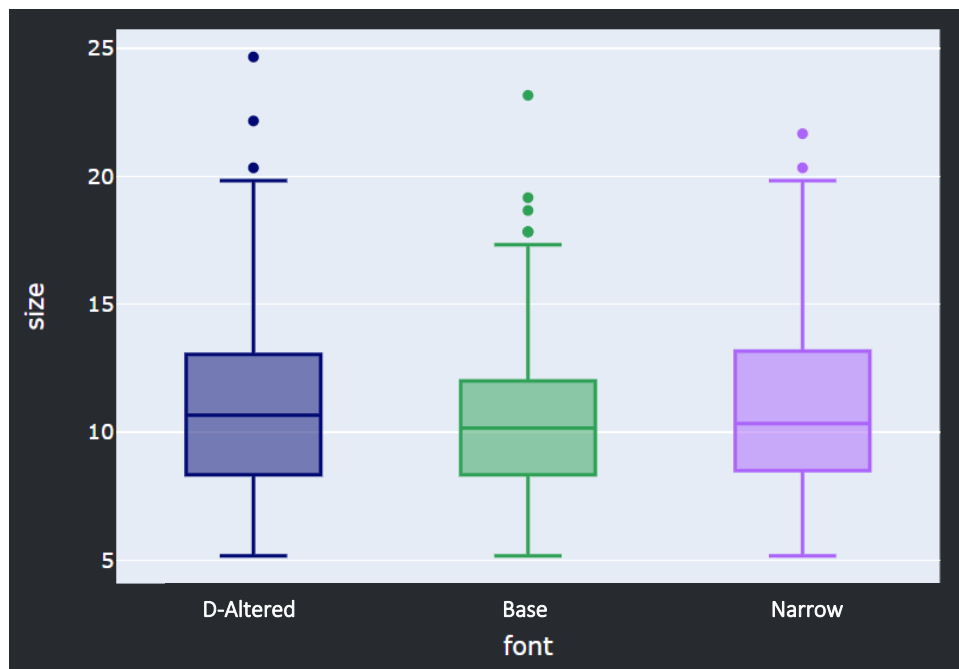


Figure 43: Size of Each Font Type for Stick Letters

Breaking the letters down into three subcategories based on their shape allowed for a deeper understanding into what typeface changes cause improvements in legibility. The round letters, all of which have some instances of inter-character spacing, benefitted the most from the

narrow font style, with thinner line strokes and more intercharacter spacing. The capital and stick letters, both of which have straight lines in their shape, benefited from the D-altered font, which includes thinner line strokes and a stockier character. Data with the individual analysis of each letter can be found in Appendix 5.

## Words

Unlike the test with the individual letters, each word has a mixture of capital letters, round letters, and stick letters. Due to the mixture of different letter shapes within each word, it was more difficult to see the effect of the studied fonts on the words.

In the words test, the base font outperformed all others, closely followed by the narrow font. This is a trend that was also observed within the letters test. With the same spacing, the narrow font, smaller than its base counterpart, was able to trail the base font by only 0.12 points. This shows that, overall, these two fonts are almost equally legible.

Through studying the analysis of the individual words in Appendix 7, there were some patterns within words that became apparent. Firstly, the C-altered font performed the worst in every word tested. This was likely due to the font being a more elongated and crowded version of the base font.

There were 6 words in which the base font was the top performer: Honors, Houses, Season, Senior, Sensor, and Doting. The other four words (Barley, Bartey, Eatery, and Felony) had the narrow font as its top performer. The only words to have the D-alternate font as its second best were Felony and Season. This data gives some insight into how typeface design can change when it is tested in individual letters versus in whole words. In the second test, the differences in between the narrow and base fonts were less noticeable than they were in the first test. Because the narrow font takes up less space, it allows the participant to process the information at a faster rate. This, in turn, helped to bridge the gap in the average size of the two fonts. The narrow font also takes up much less space than the base font but was found to be just as legible. This is extremely beneficial, as the signs can be made smaller to save material, or the font can be made larger to fill up the entirety of the existing roadway sign.

## Summary

### Recommendations

In both tests, the narrow and base fonts were found to yield very similar results, having the same effect on test participants. Because the narrow font takes up less space than the base font, it could be used to take up as much signage space as the base font, but with a higher font size. As previously mentioned, Standard Highway Alphabet at a 20% size increase was found to be much more legible than the standard size counterpart. “Smaller” fonts, such as the narrow, d-altered, and c-altered font could take advantage of their small typeface and be printed at a larger font size on highway signs. The narrow font option should be further explored as an alternative to the current Standard Highway Alphabet used on US highways. In the end, the fonts with more inter-character spacing and rounder letters were found to be the most legible by the test participants.

Some future edits to fonts could include incorporating more white space in round letters and using thinner lines in stick letters. Wider letters would also be beneficial to overall word legibility. Testing the fonts in terms of the space they take up, not just their font size, would lead to better results as to their functionality on highway signs.

There are also many ways in which to build on this research. Future research should include physical highway signs, in which different color combinations are tested to maximize compatibility and legibility. A variety of research participants should be sought out for this field test, ranging from those with perfect vision to differing levels of achromatopsia, commonly known as color blindness. Expanding on the effects of contrast and the compatibility of background and text colors, synthesized with font design, could lead to a major improvement in the legibility of highway signs.

As the Texas Transportation Institute retro-reflectivity study showed, it is beneficial to carry out these types of studies in uncontrolled roadway conditions. It is recommended that a similar study is carried out in the field, in both day and nighttime hours. This would also allow the researchers to understand the effects of reflective sheeting and vehicle headlights on the fonts studied. Future studies should focus on the existing size of highway signs, not just the size of the fonts themselves. Instead of altering the size of the physical sign, the size of the different fonts should be increased to the maximum size that can fit on the highway sign. This would allow researchers to understand the most legible font, as it applies to the existing and available “canvases”, or highway signs.

## Future Changes

As vehicle technologies evolve, drivers are slowly becoming less involved in driving, with more of the process becoming automated. Within the scope of autonomous driving there are 6 different levels, ranging from level 0 (or no driving automation) to level 6 (or full driving automation) [19]. Each level requires different levels of driver interaction and attention. The higher levels, especially 3, 4, and 5, do not require active involvement from the driver. In these levels, it is the vehicle’s job to understand and adapt to a variety of roadway conditions. This includes the ability to read and process roadway signs, whether they be permanent or temporary ones.

While permanent signs could be hardwired into GPS and similar driving systems, roadway conditions are ever changing, and technology is not always current. Google maps, one of the largest map providers, updates their information every 1-3 years, with longer intervals for rural areas. The ability of an assisted vehicle to read, process and understand the information provided on road signs leads to a safer riding experience and more accurate trips.

When considering the future of driving, it is important to not only look at the legibility of fonts, but also the compatibility of colors, contrast, and reflective materials. Although autonomous vehicles use laser based light detection and lidar systems for driving purposes, visible cameras allow the systems to recognize traffic signs. Through a process known as harmonization, highway sign data is collected for recognition algorithms to be able to process [20]. The more data collected, the easier it is for said systems to understand signs. However, that

means that the cameras have a hard time recognizing damaged or modified signs, or those whose data has not been collected prior, which can lead to a dangerous misinterpretation of the information provided.

Scientists have found that altering the material that the sign is composed of can make it much easier for the system to visualize and process the information presented. The reflective material used, a microscale concave interface, causes a reflection that changes depending on the distance and angle at which it is visualized. This ensures a variety of things: the signs do not need to be self-illuminated at night and can depend on a vehicle's headlights; if a pedestrian is walking towards a sign, the vehicle lights on the road would cause the sign to change color as the vehicle quickly approaches the sign, alerting the pedestrian; and drivers would always visualize the same color on the reflective sign, as the position relative to the light source is fixed [21]. This material also makes it possible for vehicles to use lidar systems, in place of cameras, to read these signs, giving more accurate results based on the infrared image, and makes the system capable of ignoring damaged or altered signs. This material, a retro-reflective sheeting, could also impact halation and pedestrian legibility. When testing its effect on self-driving systems, it is important to also consider its impact on human drivers.

When looking at the future of driving, road conditions have to be adapted to meet the needs of newer technologies. Through a combination of retro-reflective sheeting, contrast, and font design, highway signage could improve for both drivers and automated systems. The harmony of the success of signs is critical as more autonomous vehicles enter the market.

## References

- [1] “The Evolution of MUTCD - Knowledge - FHWA MUTCD.”  
<https://mutcd.fhwa.dot.gov/kno-history.htm> (accessed May 25, 2023).
- [2] “Standard Alphabets For Traffic Control Devices”, Accessed: May 26, 2023. [Online]. Available: <https://mutcd.fhwa.dot.gov/shsm/alphabets.pdf>
- [3] “Why the US has two different highway fonts - YouTube.”  
<https://www.youtube.com/watch?v=eky17clTEeQ> (accessed May 25, 2023).
- [4] D. Persson, “The Elderly Driver: Deciding When to Stop”, Accessed: Dec. 26, 2022. [Online]. Available: <https://academic.oup.com/gerontologist/article/33/1/88/610585>
- [5] T. J. Slattery and K. Rayner, “The influence of text legibility on eye movements during reading,” *Appl Cogn Psychol*, vol. 24, no. 8, pp. 1129–1148, Nov. 2010, doi: 10.1002/ACP.1623.
- [6] “Nighttime Visibility of Traffic Signs: Chapter 1 - Introduction.”  
[https://safety.fhwa.dot.gov/roadway\\_dept/night\\_visib/sign\\_visib/fhwasa03002/chapter1.htm](https://safety.fhwa.dot.gov/roadway_dept/night_visib/sign_visib/fhwasa03002/chapter1.htm) (accessed Jan. 29, 2023).
- [7] “Methods for Maintaining Traffic Sign Retroreflectivity - Safety | Federal Highway Administration.”  
[https://safety.fhwa.dot.gov/roadway\\_dept/night\\_visib/policy\\_guide/fhwahrt08026/chapter1.cfm](https://safety.fhwa.dot.gov/roadway_dept/night_visib/policy_guide/fhwahrt08026/chapter1.cfm) (accessed Apr. 06, 2023).
- [8] S. Chrysler, S. Stackhouse, D. Tranchida, and E. Arthur, “Improving street name sign legibility for older drivers,” *Proceedings of the Human Factors and Ergonomics Society*, pp. 1597–1601, 2001, doi: 10.1177/154193120104502308.
- [9] “Report on Highway Guide Sign Fonts Prepared by: U.S. Department of Transportation Federal Highway Administration Office of Transportation Operations - 2.0 Background - FHWA Office of Operations.”  
[https://mutcd.fhwa.dot.gov/resources/interim\\_approval/ia5rptcongress/ch2.htm](https://mutcd.fhwa.dot.gov/resources/interim_approval/ia5rptcongress/ch2.htm) (accessed Apr. 06, 2023).
- [10] P. M. Garvey, M. T. Pietrucha, D. Meeker, P. M. Garvey, and M. T. Pietrucha, “Effects of Font and Capitalization on Legibility of Guide Signs”.
- [11] H. G. Hawkins Jr, M. D. Woodridge, A. B. Kelly, D. L. Picha, and F. K. Greene, “Legibility Comparison of Three Freeway Guide Sign Alphabets,” May 1999.  
<https://static.tti.tamu.edu/tti.tamu.edu/documents/1276-1F.pdf> (accessed Feb. 15, 2023).
- [12] “Luminance - Oxford Reference.”  
<https://www.oxfordreference.com/display/10.1093/oi/authority.20110803100118635;jsessionid=72A7CBD84B12D0F8B0867C641D6E2977> (accessed May 21, 2023).

- [13] M. Sivak and P. L. Olson, “Optimal and Minimal Luminance Characteristics for Retroreflective Highway Signs.” <https://onlinepubs.trb.org/Onlinepubs/trr/1985/1027/1027-011.pdf> (accessed Feb. 13, 2023).
- [14] T. W. Forbes, B. B. Saari, W. H. Greenwood, J. G. Goldblatt, and T. E. Hill, “LUMINANCE AND CONTRAST REQUIREMENTS FOR LEGIBILITY AND VISIBILITY OF HIGHWAY SIGNS”.
- [15] J. Dobres, S. T. Chrysler, B. Wolfe, N. Chahine, and B. Reimer, “Empirical assessment of the legibility of the highway gothic and clearview signage fonts,” *Transp Res Rec*, vol. 2624, pp. 1–8, 2017, doi: 10.3141/2624-01.
- [16] D. J. Mace, “Transportation in an Aging Society: Improving Mobility and Safety for Older Persons,” 1998. <https://onlinepubs.trb.org/onlinepubs/sr/sr218v2.pdf> (accessed Feb. 14, 2023).
- [17] P. M. Garvey, M. J. Klena, W.-Y. Eie, D. Meeker, M. T. Pietrucha, and T. D. L. P. T. Institute, “The Legibility of the Clearview Typeface System Versus Standard Highway Alphabets on Negative- and Positive-Contrast Signs,” Feb. 2015, doi: 10.21949/1503647.
- [18] A. G. Ruano Duke, “Evaluation of Clearview, Highway Gothic, and Other Modified Fonts on Guide Signs,” Dec. 2019.
- [19] “What are the 6 levels of autonomous vehicles?” <https://www.faistgroup.com/news/autonomous-vehicles-levels/> (accessed Aug. 16, 2023).
- [20] T. Mihalj *et al.*, “Road Infrastructure Challenges Faced by Automated Driving: A Review,” *Applied Sciences* 2022, Vol. 12, Page 3477, vol. 12, no. 7, p. 3477, Mar. 2022, doi: 10.3390/APP12073477.
- [21] “Could microscale concave interfaces help self-driving cars read road signs? – Physics World.” <https://physicsworld.com/a/could-microscale-concave-interfaces-help-self-driving-cars-read-road-signs/> (accessed Aug. 15, 2023).



# Appendix 1 – IRB Approval Letter

**Division of Scholarly Integrity and  
Research Compliance**  
Institutional Review Board  
North End Center, Suite 4120 (MC 0497)  
300 Turner Street NW  
Blacksburg, Virginia 24061  
540/231-3732  
irb@vt.edu  
<http://www.research.vt.edu/sirc/hrpp>



## MEMORANDUM

**DATE:** April 7, 2023  
**TO:** Bryan J Katz, Marta Perez Vidal-Ribas  
**FROM:** Virginia Tech Institutional Review Board (FWA00000572)  
**PROTOCOL TITLE:** Assessing Design Improvements of Highway Sign Fonts  
**IRB NUMBER:** 23-357

Effective April 7, 2023, the Virginia Tech Human Research Protection Program (HRPP) determined that this protocol meets the criteria for exemption from IRB review under 45 CFR 46.104(d) category (ies) 2(ii).

Ongoing IRB review and approval by this organization is not required. This determination applies only to the activities described in the IRB submission and does not apply should any changes be made. If changes are made and there are questions about whether these activities impact the exempt determination, please submit an amendment to the HRPP for a determination.

This exempt determination does not apply to any collaborating institution(s). The Virginia Tech HRPP and IRB cannot provide an exemption that overrides the jurisdiction of a local IRB or other institutional mechanism for determining exemptions.

All investigators (listed above) are required to comply with the researcher requirements outlined at:

<https://secure.research.vt.edu/external/irb/responsibilities.htm>

(Please review responsibilities before beginning your research.)

## PROTOCOL INFORMATION:

Determined As: **Exempt, under 45 CFR 46.104(d) category(ies) 2(ii)**  
Protocol Determination Date: **April 7, 2023**

## ASSOCIATED FUNDING:

The table on the following page indicates whether grant proposals are related to this protocol, and which of the listed proposals, if any, have been compared to this protocol, if required.

*Invent the Future*

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY  
*An equal opportunity, affirmative action institution*

Date*	OSP Number	Sponsor	Grant Comparison Conducted?

\* Date this proposal number was compared, assessed as not requiring comparison, or comparison information was revised.

If this protocol is to cover any other grant proposals, please contact the HRPP office ([irb@vt.edu](mailto:irb@vt.edu)) immediately.

# **Appendix 2 – Informed Consent Form**

## **VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY**

### *Informed Consent for Participants*

#### **Title of Project:**

Assessing Design Improvements of Highway Sign Fonts

#### **Investigators:**

- Dr. Bryan Katz – Associate Professor of Practice within the Department of Civil and Environmental Engineering at Virginia Tech
- Marta Perez Vidal-Ribas – Master’s Candidate within the Department of Civil and Environmental Engineering at Virginia Tech

### **I. THE PURPOSE OF THIS RESEARCH PROJECT**

The Federal Highway Administration (FHWA) maintains the Manual on Uniform Traffic Control Devices (MUTCD), which serves as the standard for highway sign design and application in the United States. The MUTCD requires that signs are designed using standard fonts, known as alphabets. The alphabets were originally designed in the late 1940s and early 1950s and then incorporated into official alphabets released by FHWA in 1966 and 1977. While small changes have been made to the letters and spacing since that time, the alphabet has largely remained unchanged.

Since the alphabets were designed, there has been research to look at various fonts to replace the FHWA standard alphabets. The research has been inconclusive with some fonts working better than others, but not for all conditions. Practitioners have been interested in looking at potential alphabet improvements that would improve legibility without impacting sign cost.

By assessing different font designs, decisions can be made to improve legibility which can impact roadway safety. Since signs are the main method to relay information to drivers, research will help to define which letter design choices aid in overall sign legibility. The purpose of this research study is to determine the legibility of different, newly designed letters, and the spacing in between those letters. The study will also analyze the difference in spacing in words, and whether or not that affects legibility. The results will provide guidance on the effectiveness of the new designs.

### **II. PROCEDURES**

During the course of this experiment, you will be asked to perform the following tasks:

- 1) Read this Informed Consent Form and sign it if you agree to participate.
- 2) Show the experimenter your valid driver’s license.
- 3) Identify words and letters as they appear on a computer screen.

It is important for you to understand that we are not evaluating you or your performance in any way. You are helping us to evaluate the readability of various signs. The opinions you have will help us determine recommendations for guidelines for highway signs. The information and feedback that you provide is very important to this project. The total experiment time will last up to 30 minutes.

### **III. RISKS**

The tasks described here are believed to pose no more than minimal risk to your health or well-being. The risks of reading text on our computer screen are similar to that of using your own personal computer screen. You may become bored due to the length of the study.

The following precautions will be taken to ensure minimal risk to you:

1. You may take breaks or decide not to participate at any time.
2. The experimenter will be present while you are testing.

### **IV. BENEFITS**

While there are no direct benefits to you from this research, you may find the experiment interesting. No promise or guarantee of benefits is made to encourage you to participate. Participation in this study will contribute to the improvement of highway safety by providing insight into proper signing practices.

### **V. EXTENT OF ANONYMITY AND CONFIDENTIALITY**

The data gathered in this experiment, including name, age, and vision data, will be treated with confidentiality. Shortly after participation, your name will be separated from your data. A coding scheme will be employed to identify the data by participant number only (e.g., Participant No. 1). You may elect to have your data withdrawn from the study if you so desire, but you must inform the experimenters immediately of this decision so that the data may be promptly removed.

The data collected in this study may be used in future transportation research projects. IRB approval will be obtained prior to accessing the data for other projects. It is possible that the Institutional Review Board (IRB) may view this study's collected data for auditing purposes. The IRB is responsible for the oversight of the protection of human subjects involved in research.

### **VI. COMPENSATION**

You will receive no compensation for participating.

### **VII. FREEDOM TO WITHDRAW**

As a participant in this research, you are free to withdraw at any time without penalty. If you choose to withdraw, your data will be removed from the database. Furthermore, you are free not to answer any question or respond to experimental situations without penalty. If you choose to withdraw during the study session, please inform the experimenter of this decision and he/she will immediately end your experiment.

### **VIII. APPROVAL OF RESEARCH**

This research project has been approved, as required, by the Institutional Review Board for Research Involving Human Subjects at Virginia Polytechnic Institute and State University. This form is valid for the period listed at the bottom of the page.

## IX. PARTICIPANT'S RESPONSIBILITIES

If you voluntarily agree to participate in this study, you will have the following responsibilities:

1. To follow the experimental procedures as well as you can.

Check all that apply:

- I have informed the experimenter of any concerns/questions I have about this study.

Should I have any questions about this research or its conduct, I may contact:

Dr. Bryan Katz @ (540) 231-0290, or by email: [bkatz@vt.edu](mailto:bkatz@vt.edu)

Marta Perez Vidal-Ribas @ (214) 766-2130, or by email: [martap18@vt.edu](mailto:martap18@vt.edu)

If I should have any questions about the protection of human research participants regarding this study, I may contact: the Virginia Tech Institutional Review Board for the Protection of Human Subjects, telephone: (540) 231-3732; email: [irb@vt.edu](mailto:irb@vt.edu);

## X. PARTICIPANT'S PERMISSION

I have read and understand the Informed Consent and conditions of this project. I have had all my questions answered. I hereby acknowledge the above and give my voluntary consent for participation in this project. **If I participate, I may withdraw at any time without penalty. I agree to abide by the rules of this project.**

## Appendix 3 – Python Code

### Letters Test

```
"""
Created on Tue Feb 1 15:17:43 2022

@author: StephenTaylor
@editor: MartaPerez
"""
# Import necessary packages
from random import sample,randint # imported to
generate random numbers
import pandas as pd
import numpy as np
import plotly.figure_factory as ff
import plotly.express as px
from scipy.stats import ttest_ind, mannwhitneyu

import dash # Dash creates the website
from dash import html # Used to creat HTML
containers
from dash import dcc # Used to create dcc containers
from dash import dash_table
from dash.dependencies import Input, Output, State
# Import for callbacks
from dash.exceptions import PreventUpdate #
Prevents a callback to operate

from dash_extensions import Keyboard

import dash_bootstrap_components as dbc # Creates
grid layout for dash

from azure.storage.blob import BlobServiceClient #
Accesses Azure
from datetime import datetime # Easier work with
timestamps

# %%
"""This section creates a series of screens that can be
displayed based on
the results of a callback at the bottom of this script.
They are placed in
the script in the order they should be displayed in the
app."""

# This is the header row that is shown on all pages.
header_row = dbc.Row(dbc.Col([
    html.H1('Road Sign Font Study',
            style={'backgroundColor': '#000872', #
This is toXcel Blue
'color': '#DFDFDF', # This is the grey
font color
'margin-bottom': '0px'})),
    dcc.Markdown("""A quick app to test the
legibility of various
fonts powered by
[toXcel](https://toxcel.com/)""",
            style={'backgroundColor': '#000872',
'color': '#DFDFDF',
'font-size': 24}))
    )
    )

# %%
# This is the first page info wall complete with
instructions and the signin
info_wall = html.Div([
    html.Div(
        dcc.Markdown("""
Welcome to the font study. This app is still in Beta
testing.
If you have been asked to participate, please provide
your email and the
password you have been provided""")),
    html.Div(
        dcc.Input(id='input_email', # Email input used
in update_output
        type='email', # means it is expecting the
@ symbol
        placeholder='email', # Instructs user that
email goes there
        style={'width': '450px', # Formatting
changes
'height': '45px',
'padding': '10px',
'margin-top': '15px',
'font-size': '16px',
'border-width': '3px',
'border-color': '#a0a3a2'})),
        style={'display': 'flex',
'justifyContent': 'center'}
    ),
    html.Div(
        dcc.Input(id='input_password', # password
input used in update_output
        type='text',
        placeholder='password',
        style={'width': '450px',
'height': '45px',
'padding': '10px',
'margin-top': '15px',
'font-size': '16px',
'border-width': '3px',
'border-color': '#a0a3a2'})),
        style={'display': 'flex',
```

```

        'justifyContent': 'center'
    ),
    html.Div(
        html.Button('Verify', # Button for user to click
to login
            id='verify', # number of clicks used in
update_output
            n_clicks=0, # Starts counting at 0
            style={'border-width': '3px', # More
formatting
                'font-size': '14px'}),
        style={'display': 'flex',
            'justifyContent': 'center',
            'padding-top': '15px'
        },
        dbc.Row(
            html.Div(id='output1', # Output from
update_output. Text will change
            style={'display': 'flex',
                'justifyContent': 'center'})
        ))
# %%
# Second page is where the signs with letters will be
shown.
image_test = html.Div([
    # This is the actual sign to be displayed.
    Keyboard(id='keyboard'),
    dbc.Row(
        dbc.Button('I KNOW WHAT IT IS!', # Button
spread across top of screen
            id='confirm', # Input for time_elapsed
(stops timing)
            n_clicks=0, # Unnecessary
            color="success", # Color is green
            href='/yesorno'), # takes user to next
page
        style={'display': 'flex', # Formatting
            'justifyContent': 'center',
            'padding-top': '15px'}),

    dbc.Row(id='road_sign', # Output from
create_image
        align='center',
        justify='center'),

    # Counter. When the page is first displayed, it
starts counting at 0.
    html.Div(
        dcc.Interval(id="interval", # Input for
random_letters & create_image
            interval=120, # time between each
interval in ms
            max_intervals=500) # Maximum count.
    ),
    dbc.Row(

```

```

        dcc.Link('Log out', # logout button on every
page
            href='/', # takes user back to the info_wall
            style={'color': '#bed4c4', # More
formatting
                'font-family': 'serif',
                'font-weight': 'bold',
                "text-decoration": "none",
                'font-size': '20px'
            },
        style={'padding-left': '90%',
            'padding-top': '10px'
        })
# %%
# This is the third page where the user is asked if it
was a letter or not.
yes_or_no = html.Div(
    dbc.Row([
        dbc.Col(dbc.Button('Yes, that was a letter', #
First button for YES
            id='yes', # Input to initial_guess
            n_clicks=0, # Starts counting at 0
            href='/response', # Takes users to
next page
                color='success'), # Green
            width=6),

        dbc.Col(dbc.Button('No, that was not a letter', #
Second button NO
            id='no', # Input to initial_guess
            n_clicks=0,
            href='/response', # Takes users to
same place
                color='danger'), # Red
            width=6)
    ])),
# %% Fourth page
response = html.Div([ # This is set at the same level
as previous buttons.
    html.Div(id='button_wording'), # Output from
button_wording
    dbc.Row([
        html.Div(id='tracker'),
        dcc.Link('Log out', # Same logout button
            href='/',
            style={'color': '#bed4c4',
                'font-family': 'serif',
                'font-weight': 'bold',
                "text-decoration": "none",
                'font-size': '20px'
            },
        ),
        style={'padding-left': '80%',
            'padding-top': '10px'
        }
    ])
)

```

```

    ])
# %%
# Final page. Is shown once the participant is
finished.
thank_you = html.Div([
    dbc.Row(html.Div("Thank you for participating in
this study.
If you would like to participate again, close your
browser window and restart
the app.")),
    dbc.Row(
        dcc.Link('Log out',
            href='/',
            style={'color': '#bed4c4',
                'font-family': 'serif',
                'font-weight': 'bold',
                'text-decoration': "none",
                'font-size': '20px'}
        ),
        style={'padding-left': '90%',
            'padding-top': '10px'}
    ))

# %%
# Analysis page. Can only be found by those who
already know where it is
analysis = html.Div(children=[
    # Top row with all of the filters and pull button
    dbc.Row(children=[
        dbc.Col(children=[
            html.Div('Filter by Letter'),
            dcc.Dropdown(['All', 'A', 'F', 'a', 'c', 'e', 'f', 'g',
                'k',
                's', 't', 'w', 'x', 'z'],
                placeholder='Filter by letter shown',
                value='All',
                multi=True,
                id='letter_dd')]
        ),
        dbc.Col(children=[
            html.Div("Which Statistical Test?"),
            dcc.Dropdown(['t_test', 'mann_whitney_u'],
                placeholder='Pick a Distribution
Test',
                value='t_test',
                id='test_type_dd')]
        ),
        dbc.Col(children=[
            html.Div("Which Contrast?"),
            dcc.Dropdown(['All', 'diamond', 'rectangle'],
                placeholder='Pick a Contrast',
                value='All',
                id='contrast_dd')]
        ),
        dbc.Col(children=[

```

```

            html.Div("Which Metric?"),
            dcc.Dropdown(['size', 'time'],
                placeholder='Pick a Measure',
                value='size',
                id='measure_dd')]
        ),
        dbc.Col(
            dbc.Button('Pull Data',
                id='analysis_button',
                color='success',
                n_clicks=0)
        )
    ]),
    # First row Is the summary table
    dbc.Row(children=[
        html.Div("Select which filters above you would
like to see, then
click the \"Pull Data\" button to see summary statistics.
Note that each summary
statistic represents the moment that someone
recognized the letter."),
        html.Div(id='summary_tb')]
    ),

    # Final row has the scores of each model.
    dbc.Row(children=[
        dbc.Col(html.Div(id='avg_improve')),
        dbc.Col(html.Div(id='avg_improve2')),
        dbc.Col(html.Div(id='p_score')),
        dbc.Col(html.Div(id='score'))
    ]),

    # Second row includes The two scatter plots with
their fit lines
    dbc.Row(children=[
        # First Column - Base vs Altered Scores
        dbc.Col(children=[
            html.Div(id='linear_model'),
            html.Div(id='model_results'),
            html.Div(id='linear_model2'),
            html.Div(id='model_results2')
        ]),

        # Second Column - Change Score relative to
starting position
        dbc.Col(children=[
            html.Div(id='change_score'),
            html.Div(id='change_score2')
        ])
    ]),
    # Third Row includes the two frequency
distributions
    dbc.Row(children=[
        # First Column - Frequency Distribution Chart
        dbc.Col(children=[
            html.Div(id='freq_main'),
            html.Div(id='freq_main2')

```

```

    ),
    # Second Column Column - Relative Scores
    dbc.Col(children=[
        html.Div(id='rel_score_main'),
        #Adding base v narrow
        html.Div(id='rel_score_main2')
    ])
]),

# Fourth Row includes the box plots of each font
type as well as incorrect
dbc.Row(children=[
    # First Column - Boxplots
    dbc.Col(children=[
        html.Div(id='boxplot_main')
    ]),

    # Second Column - Incorrect Guesses
    dbc.Col(children=[
        html.Div(id='wrong_main')
    ]),
])

# %%
# Create the app
dash_app = dash.Dash(__name__,
    # this is the darker theme that toXcel
    uses in apps.

    external_stylesheets=[dbc.themes.SLATE],
    # Do not want to show callback
    exceptions
        suppress_callback_exceptions=True,
    # Do not want the page to look like its
    loading always
        update_title=None)

# A little code used to pull the css file in the assets
folder.
dash_app.css.config.serve_locally = True
dash_app.scripts.config.serve_locally = True

# Line added for GitHub/Azure integration.
app = dash_app.server

# Activate the dashboard with the necessary graphics
dash_app.layout = html.Div([header_row,
    dcc.Location(id='url',
refresh=False),
    html.Div(id='page-content',
        dcc.Store(id='size',
            storage_type='session'),
        dcc.Store(id='email',
            storage_type='session'),
        dcc.Store(id='sign_shown',
            storage_type='session'),
        dcc.Store(id='letter_shown',
            storage_type='session'),
        dcc.Store(id='font_shown',
            storage_type='session'),
        dcc.Store(id='test_count',
            storage_type='session'),
        dcc.Store(id='start_time',
            storage_type='session'),
        dcc.Store(id='time_elapsed',
            storage_type='session'),
        dcc.Store(id='initial_answer',
            storage_type='session'),
        dcc.Store(id='random_list',
            storage_type='session'),
        dcc.Store(id='keyed_time',
            storage_type='session'),
        dcc.Store(id='analysis_data',
            storage_type='session')
    ])

# %%
# The following section includes all of the callbacks.
@dash_app.callback(Output('keyed_time', 'data'),
    Input('keyed_time', 'data'),
    Input('keyboard', 'keydown'))
def keyed_event(keyed_time, event):
    if keyed_time is None:
        return {'value': 0}
    elif event is None:
        return {'value': keyed_time['value']}
    else:
        return {'value': datetime.now().timestamp()}

@dash_app.callback(Output('page-content',
'children'),
    inputs=[Input('url', 'pathname')])
def display_page(pathname):
    """This function takes the url provided in the app
    layout above and
    shows the correct page based on the url. The pages
    are defined in the
    first half of this script."""

    # This is an if statement. python checks the first if
    statement. If it is
    # true, then it does what it says to do (in this case
    returns image_test)
    # and moves on.

    # if pathname == '/image_test' and keyed['key'] ==
    '':
    # return yes_or_no
    if pathname == '/image_test':

```

```

    return image_test
elif pathname == '/response':
    return response
elif pathname == '/yesorno':
    return yes_or_no
elif pathname == '/thank_you':
    return thank_you
elif pathname == '/analysis':
    return analysis
else:
    return info_wall

@dash_app.callback(
    # Outputs the text to be shown after clicking
    verify.
    # Is input for the first page
    Output('output1', 'children'),
    # Outputs the user's email to be saved in the
    responses
    # Is input for restart_interval
    Output('email', 'data'),
    # Creates a random list used to pull letters and
    fonts later
    # Is input for random_letters
    Output('random_list', 'data'),
    # Input from verify button on first page.
    inputs=Input('verify', 'n_clicks'),
    # State is used to input values that may or may not
    exist.
    # Input from email text box on first page.
    state=[State('input_email', 'value'),
           # Input from password text box on first page.
           State('input_password', 'value')])
def update_output(n_clicks, uname, passw):
    """Inputs:
    n-clicks - from verify button
    uname - from email text box
    passw - from password text box

    If the verify button is clicked, this function
    begins automatically
    working. This function outputs:
    text - to be displayed on the first page and a
    button if the user
    provides the correct login information.
    email - stores their email for tracking
    purposes
    random_list - a list of random numbers
    between 3 and 32 (inclusive)
    which is used to ensure random showing of
    letters, but that each
    letter is shown only once with each font
    type."""

```

```

    # Prevents function from doing anything unless
    verify is clicked.
    if n_clicks < 1:
        raise PreventUpdate
    # Tries to split the email at the @ symbol. (Case
    does not matter)
    try:
        domain = uname.split('@', 1)[1].upper()
        # If the domain of the email contains these three
        options, it lets the
        # user into the app.
        if ((domain == 'TOXCEL.COM'
            or domain == 'DOT.GOV'
            or domain == 'BATTELLE.ORG')
            and passw == 'toxcel'):
            # Show the instructions and continue button
            # if username and password are correct.
            what_to_show = html.Div([
                dbc.Row(html.Pre(children=[
                    html.Strong('PLEASE READ THE
                    FOLLOWING INSTRUCTIONS:'),
                    html.Pre("
                    Thank you for volunteering to participate in this
                    legibility study. Before we
                    begin, please make sure that you are sitting normally
                    in your chair without
                    leaning forward or slouching. For the duration of the
                    study, it is important to
                    remain approximately the same distance from the
                    screen. Please place your hand
                    on your mouse.

```

Once the study begins, a colored rectangle will appear on the screen. A small character will appear in the center of the screen and increase in size as if you were driving towards it in your car. The character will eventually be large enough for you to see so please do not squint or strain your eyes. Please do not try and guess what the character is before it becomes legible.

Once the character becomes legible to you, click the green button above the sign with the mouse, or click any button on your keyboard. This will mark the distance at which you can see the character on the sign so it is important that you press it once you can clearly identify what the character is.

If you select a button on your keyboard, the test will stop and the distance at the time of the button press will be measured; however, you will still have to

click the green button to advance to the next page.

When you click the green button, the sign will disappear completely, and you will be given the opportunity to indicate if the character was a letter or not.

If you said that the character was a letter, you will be given the opportunity to indicate what the letter was. Type in the provided text box what letter you saw. It is important to note that the case of the letter matters. When you are ready for the next character, select the button in the top left corner.

You have the option to opt out of the test at any point by selecting the button that says "Log out." Note, the test records your answers in real time, in case you lose access to the test for any reason.

You will notice that some signs may repeat the same characters. That behavior is normal for the test, and not an indication of anything wrong with the program. The first three signs will be test images and will be the same across all tests. After the first three, the test will begin and the font, color of sign, and character displayed will be randomized. You will be shown a total of 43 signs. Thank you again for your participation.

```
    ""))),
    # Button to show at the bottom of the
    screen
    dbc.Row(dbc.Button('Click here when you
    are ready.',
                      href='/image_test', # Takes
                      style={ 'font-weight': 'bold',
                              "text-decoration": "none",
                              'font-size': '20px',
                              'font-family': 'Altered '},
                              color='success'),
            style={ 'padding-top': '15px'}))
    else:
        # If the user put in a valid email but it is
        incorrect or if the
        # password is incorrect, show this text.
        what_to_show = html.Div(
            children="Sorry, I do not have your
            information.
            Please contact the research team at toXcel to get
            access",
            style={ 'padding-top': '15px',
```

```
        'font-size': '16px'})
    # If we cannot split on the @ symbol for some
    reason, show this text.
    except IndexError:
        what_to_show = html.Div(
            children="Please input valid email address",
            style={ 'padding-top': '15px',
                    'font-size': '16px'})
    # Outputs of the function are: text to output, email,
    random list.
    return (what_to_show,
            {'value': uname},
            {'value': sample(range(3, 46), 43)}
            )
```

# If you are still reading this... you are dedicated!  
Almost there!

```
@dash_app.callback(
    # This one is weird. Its inputs (as states) and
    outputs are mostly the same
    # Output which sign should be shown (Green or
    Yellow)
    Output('sign_shown', 'data'),
    # Output which letter should be shown
    Output('letter_shown', 'data'),
    # Output which font should be used in this test
    Output('font_shown', 'data'),
    # Increments the test count
    Output('test_count', 'data'),
    # Starts a timer.
    Output('start_time', 'data'),

    # Input the current interval (this will change every
    120 ms)
    inputs=[Input('interval', 'n_intervals'),
            # Input what the test count is (This will
            change every test)
            Input('test_count', 'data')],
    # If they are available input the same outputs
    state=[State('sign_shown', 'data'),
           State('letter_shown', 'data'),
           State('font_shown', 'data'),
           State('start_time', 'data'),
           # Input the random list generated in the
           function above.
           State('random_list', 'data')]
    )
```

```
def random_letters(interval, test, sign, letter, font,
start_time, rand_list):
    """Randomly generates the letters and fonts for the
    test. Also starts
    timing the current test, and increments the test
    number as it goes.
    Inputs:
```

```

Interval - from image_test. Increments every
120 ms a sign is shown
test - Counts the number of total tests that have
started
sign - which sign is being be shown (green or
yellow)
letter - which letter is being shown
font - which font is being shown
start_time - what time did the current test start
rand_list - from update_output. List of random
numbers used to
randomize the order that letters and fonts are
shown, but ensures they
are all shown exactly one time.

Outputs stay the same as the inputs unless this is a
new test.
Outputs:
sign - which sign should be shown
letter - which letter should be shown
font - which font should the letter have
test - what test number are we on?
start_time - starts timing if the interval is 0,
otherwise is a
passthrough variable."
# If the interval is 0, then treat this as a new test.
Generate a new
# sign, letter, font, and start timing.
if interval == 0 or interval is None:
# Simple test to make sure we have a test
number. The only time this is
# not true is if it is the first run.
try:
test_num = test['value']
except TypeError:
test_num = 0
# For the first three tests, show the same test
images: A, a, %
# Note: python is 0 indexed, meaning counting
starts at 0, NOT at 1.
if test_num < 3:
sign_num = 0
letter_num = test_num

else:
# If this is not one of the first three tests, then
randomly pull a
# letter and font pair based on the random
number list generated.
sign_num = randint(0,1)
random_num = rand_list['value']
# This looks funky because counting is hard :)
letter_num = random_num[test_num-3]

# This is legacy code. We were originally asked
to look at two sign

```

```

# types, but since have been asked only to look
at white font on green.
# Performance hit is minimal, so I have chosen
to leave it for now bc
# I think we will want to add it back.
sign = ['rectangle', 'diamond'][sign_num]

# Pairs of letters and fonts. The length of this list
(43 currently) is
# crucial. If you add letter/font pairs to this list,
you need to
# change the length of the random numbers list
(output from
# update_output), the number of tests before
killing the program (
# output from restart_interval), and the
instructions.
ltrfnt = [['A', 'Base'], ['a', 'Base'], ['%', 'Altered '],
['A', 'Base'], ['F', 'Base'], ['a', 'Base'], ['c',
'Base'],
['e', 'Base'], ['f', 'Base'], ['g', 'Base'], ['k',
'Base'],
['s', 'Base'], ['t', 'Base'], ['w', 'Base'], ['x',
'Base'],
['z', 'Base'],
['A', 'Narrow'], ['F', 'Narrow'], ['a',
'Narrow'], ['c', 'Narrow'],
['e', 'Narrow'], ['f', 'Narrow'], ['g',
'Narrow'], ['k', 'Narrow'],
['s', 'Narrow'], ['t', 'Narrow'], ['w',
'Narrow'], ['x', 'Narrow'],
['z', 'Narrow'],
['A', 'Altered '], ['F', 'Altered '], ['a',
'Altered '],
['c', 'Altered '], ['e', 'Altered '], ['f', 'Altered
'],
['g', 'Altered '], ['k', 'Altered '], ['s', 'Altered
'],
['t', 'Altered '], ['w', 'Altered '], ['x',
'Altered '],
['z', 'Altered '], ['>', 'Altered '], ['<',
'Altered '],
['%', 'Altered '], ['^', 'Altered
']][letter_num]
# The letter is the first thing in the pair.
letter = ltrfnt[0]
# The font of the letter is the second thing in the
pair.
font = ltrfnt[1]
# Start timing.
start_time = datetime.now().timestamp()
return({'value': sign},
{'value': letter},
{'value': font},
{'value': test_num + 1},
{'value': start_time})

```

```

# If this is not the first interval in a test, just keep
the same values.
else:
    return(sign,
           letter,
           font,
           test,
           start_time)

# This outputs the actual letters/signs. Also records
the size of the letter.
@dash_app.callback(
    Output('road_sign', 'children'),
    Output('size', 'data'),
    # All inputs from random_letter
    inputs=[Input('sign_shown', 'data'),
            Input('letter_shown', 'data'),
            Input('font_shown', 'data')],
    # Inputs the interval number.
    state=[State('keyboard', 'keydown'),
           State('interval', 'n_intervals'),
           State('size', 'data')]
)
def create_image(random_sign, random_letter,
                random_font, keyed, interval, s):
    """Creates colored rectangle with a letter in the
    center. The rectangle
    remains the same size, but the appears to grow
    over time.
    Inputs:
        random_sign - output from random_letters,
        random_letter - output from random_letters
        random_font - output from random_letters
        interval - incremented every 120 ms.
    Outputs:
        road_sign - card to be displayed on image_test
        size - records the size of the letter shown."""
    # This only will run if you are on the correct page.
    if interval is None:
        raise PreventUpdate

    # Brings in the data dictionaries generated in the
    function above.
    if keyed is None:
        sign = random_sign['value']
        letter = random_letter['value']
        font = random_font['value']

    # Size is 1/6th of the current interval.
    size = interval/6

    # Keeps the letter mostly center. Allows for
    some "jumping" to mimic
    # More accurate driving.
    location = 200 - size

```

```

# Also legacy code. Left because we will likely
start testing on diff
# backgrounds. For now, it jumps straight to the
else statement (green)

if sign == 'diamond':
    card = dbc.Col(html.Div(letter,
                             style={'margin-top':
f'{location}px'}),
                   width=6,
                   style={'font-size': size,
                           'text-align': 'center',
                           'backgroundColor': '#FFFFFF',
                           'font-family': font,
                           'color': '#000000',
                           'height': '400px'},
                           align='center')

    else:
        card = dbc.Col(html.Div(letter,
                                 style={'margin-top':
f'{location}px'}),
                        width=6,
                        style={'font-size': size,
                                'text-align': 'center',
                                'backgroundColor': '#006747',
                                'font-family': font,
                                'color': '#FFFFFF',
                                'height': '400px'},
                                align='center')

    else:
        card = html.Pre("""Your time has stopped.
Select the green button to go to the next page""")
        size = s['value']

    return (card,
           {'value': size})

# %%
# This callback calculates how long it took for a user
to recognize the letter.
@dash_app.callback(
    Output('time_elapsed', 'data'),
    inputs=[Input('start_time', 'data'),
            Input('keyed_time', 'data'),
            Input('confirm', 'n_clicks')])
def time_elapsed(start_time, key_time, clicks):
    """Calculates the time passed for each test.
    Inputs:
        clicks - counts how many times the confirm
        button is clicked.
        start_time - imported from random_letters to
        calc time elapsed

```

```

Outputs:
    time_elapsed - How long it took for someone to
identify letter in s"
# Is added to prevent callback errors, but also as a
failsafe in case
# the timer does not start.
if clicks < 1:
    raise PreventUpdate

if start_time is None:
    return({'value': -1})
else:
    if key_time['value'] < start_time['value']:
        end_time = datetime.now().timestamp()
    else:
        end_time = key_time['value']
    return({'value': end_time-start_time['value']})

# %%
@dash_app.callback(
    Output('initial_answer', 'data'),
    Input('yes', 'n_clicks'))
def initial_guess(clicks):
    """Records and stores whether the user thought the
character was a letter
or not.
Inputs:
    yes - counts the number of times the yes button
is clicked. If it is
    not clicked, it assumes the no button was
clicked.
Outputs:
    initial_answer - records that the user recognized
that it either was
    or was not a letter. Used as an input in
restart_interval"
# Note, for performance sake, it is always better to
use booleans than
# strings like 'yes' or 'no'. Also typically better than
1 and 0.
if clicks == 1:
    return({'value': True})
else:
    return({'value': False})

@dash_app.callback(Output('button_wording',
'children'),
    Input('test_count', 'data'),
    Input('initial_answer', 'data'))
def button_wording(test, initial_guess):
    """This is a minor callback that could be removed
entirely. I just like the
    idea of letting the user know whether they are still
in a test or are

```

```

actually being evaluated.
Inputs:
    test_count - Which test number are we on?
Output from random_letters
Outputs:
    button_wording - outputs the button that should
be shown based on
    which test number the user is on."
try:
    test_num = test['value']
except TypeError:
    test_num = 0
initial_guess = initial_guess['value']
if initial_guess:
    input_box = dbc.Row(
        dcc.Input(id='guess', # Input for
restart_interval
                type='text',
                placeholder='What did you see?', #
Asks user what letter
                style={'width': '450px', # More
formatting
                    'height': '45px',
                    'padding': '10px',
                    'margin-top': '15px',
                    'font-size': '20px',
                    'border-width': '3px',
                    'border-color': '#a0a3a2',
                    'font-family': 'serif',
                    autoComplete='off'
                })
    else:
        input_box = html.Div(id='guess')
# If the user is still in the first three test images
show this in yellow
if test_num < 3:
    return (dbc.Button('I am ready for my next test
image',
                    id='back_to_test',
                    color='warning',
                    n_clicks=0),
            input_box)
# If the user has finished their last test image, show
this in red
elif test_num == 3:
    return (dbc.Button('I am ready to start',
                    id='back_to_test',
                    color='danger',
                    n_clicks=0),
            input_box)
# All others, show this in green.
else:
    return (dbc.Button('I am ready for the next one',
                    id='back_to_test',
                    color='success',
                    n_clicks=0),

```

```

input_box)

# This is the last one! You have almost made it.
@dash_app.callback(
    Output('url', 'pathname'),
    inputs=[Input('back_to_test', 'n_clicks'),
            Input('test_count', 'data')],
    state=[State('guess', 'value'),
           State('size', 'data'),
           State('sign_shown', 'data'),
           State('letter_shown', 'data'),
           State('font_shown', 'data'),
           State('email', 'data'),
           State('time_elapsed', 'data'),
           State('initial_answer', 'data')])
def restart_interval(clicks, test, guess, size, sign,
                    letter, font, email,
                    time_elapsed, initial_guess):
    """This function practically takes all of the previous
    outputs and saves them to a txt file. Answers are written to an append
    blob as the user provides them, so if a user does not finish, we can
    still record their results.
    Inputs:
        Clicks - Runs the function as soon as someone
        clicks back_to_test.
        test - which test count are we on? Output from
        random_letters
        guess - what value did the user put in the text
        box in response
        size - what was the final size of the letter shown
        in create_image
        sign - what color sign was shown? Output from
        random_letters
        letter - what letter was shown? Output from
        random_letters
        font - what font was shown? Output from
        random_letters
        email - what email did they provide? Output
        from update_output
        time_elapsed - How long did it take them to
        recognize the letter?
        output from time_elapsed
        initial_guess - Did they recognize it as a letter or
        not?"""
    if clicks < 1:
        raise PreventUpdate
    else:
        guess = 'symbol' if guess is None else guess
        size = size['value']
        sign = sign['value']
        letter = letter['value']
        font = font['value']

```

```

        email = email['value']
        int_guess = initial_guess['value']
        timed_time = time_elapsed['value']
        if int_guess:
            correct = letter == guess.strip() # removes
            whitespace if any
        else:
            correct = True if letter in ['>', '%', '^', '<'] else
            False

        # writes to a new line (\n) the answers for each
        test.
        data =
        """\n{0},{1},{2},{3},{4},{5},{6},{7},{8}""".format(
            email, sign, letter, int_guess, guess, font, size,
            timed_time, correct)

        # creates a connection with the Azure Blob
        Storage
        service = BlobServiceClient(
            # If these credentials get out, please notify
            Stephen at
            # stephen.taylor@toxcel.com as soon a
            practical. We can generate
            # new keys, so it is not a big deal.

            account_url='https://safecurves.blob.core.windows.ne
            t',

            credential='p6PwDNuoxvxMUgpVaybejqABBujZRI
            5Jkjq/WLba93ex/6NsH8JYsChfysqRzOG+Pz2R/+9u
            1pARBOMBTKFsJg==')

        client =
        service.get_blob_client(container='safecurves',
                                blob='Marta.txt',)

        # appends the data string to the blob.
        client.append_block(data)

    try:
        test_num = test['value']
    except TypeError:
        test_num = 0

    # Test 0-2 is test images. Then 43 letter/font pairs.
    Then kill the program
    if test_num < 46:
        return ('/image_test')
    else:
        return ('/thank_you')

@dash_app.callback(Output('tracker', 'children'),
                    Input('test_count', 'data'))
def tracker_update(test):

```

```

test = test['value']
if test > 3:
    return html.Pre('Character #{ }/43'.format(test-
3))
else:
    return html.Pre('Test Character
#{ }/3'.format(test))

# %%
# These are the callbacks for the analysis page
@dash_app.callback(Output('analysis_data', 'data'),
                    Input('analysis_button', 'n_clicks'),
                    Input('test_type_dd', 'value'),
                    Input('measure_dd', 'value'),
                    Input('letter_dd', 'value'),
                    Input('contrast_dd', 'value'))
def pull_clean_data(nclicks, test_type, measure,
letter_filter, contrast):
    if nclicks < 1:
        raise PreventUpdate
    elif test_type is None or measure is None:
        raise PreventUpdate

    guesses =
pd.read_csv('https://safecurves.blob.core.windows.ne
t/safecurves/Marta.txt',
            sep=',',
            on_bad_lines='skip').reset_index()
    if 'All' in letter_filter:
        pass
    else:
        guesses =
guesses[guesses['letter'].isin(letter_filter)]
        #defining the drop down in contrast value
        if 'All' in contrast:
            pass
        else:
            guesses = guesses[guesses['sign'] == contrast]

    # When I built this, I used test@toxcel to let me
know where I was testing
    # Let's also look out for speeders. Any time that is
less than 2 seconds
    # should raise a red flag.
    guesses = guesses.sort_values(by=['email', 'index'])

    guesses = guesses[guesses['email'] !=
'test@toxcel.com']

    guesses['speeder?'] = guesses['time'].apply(lambda
t:
        True if t < 2
        else False)

```

```

# The correct column is case specific, which is
causing problems
# with x, z, and w. this new feature ignores case.

guesses['correct'] =
np.where((guesses['letter'].str.upper() ==
        guesses['guess'].str.upper()) |
        ((guesses['letter'].isin(['%',
            '<',
            '>',
            '^']))
        & (guesses['guess'] ==
'symbol')),
        True, False)

    guesses.loc[guesses['letter'].isin(['%', '<', '>', '^']),
'font'] = np.nan
    guesses['font'].fillna('symbol', inplace=True)
    # Now lets highlight the test letters
    # First create a temporary column to be dropped
later that has combines the
    # letter and font. Eg. ABase, % Altered
    guesses['temp'] = guesses['letter'] + guesses['font']

    # Now shift that temporary column down once and
twice
    guesses['temp1'] = guesses['temp'].shift(-1,
fill_value=0)
    guesses['temp2'] = guesses['temp'].shift(-2,
fill_value=0)

    # Now highlight test
    guesses['test?'] = np.where((guesses['temp'] ==
'ABase')
        & (guesses['temp1'] == 'aBase')
        & (guesses['temp2'] ==
'%symbol'),
        True,
        False)

    guesses.drop(columns=['temp', 'temp1', 'temp2'],
inplace=True)

    guesses = guesses[guesses['test?'] == False]

    # Remove outliers for folks that fell asleep at their
desk
    guesses = guesses[guesses['time'] < 100]
    #remove outliers for quick guesses and large words
    guesses = guesses[guesses['size'] > 5]
    guesses = guesses[guesses['size'] < 25]

    guesses.sort_values(by=['font'], inplace=True)

    # separate traps from base from Altered

```

```

traps = guesses[guesses['font'] == 'symbol']
base = guesses[guesses['font'] == 'Base']
Altered = guesses[guesses['font'] == 'Altered']
narrow = guesses[guesses['font'] == 'Narrow']

# Run a ttest. The data is pretty normally
distributed. I am happy with it
if test_type == 't_test':
    res = ttest_ind(base[measure], Altered
[measure],
                    alternative='greater')
    res2 = ttest_ind(base[measure],
narrow[measure], alternative='greater')

    elif test_type == 'mann_whitney_u':
        res = mannwhitneyu(base[measure], Altered
[measure],
                            alternative='greater')
        res2 = mannwhitneyu(base[measure],
narrow[measure],
                            alternative='greater')

# Now lets look at incorrect guesses.
incorrect = guesses[['email', 'font', 'letter',
'correct']]
incorrect['incorrect guesses'] =
incorrect['correct'].apply(lambda c:
                            1 if c is False
                            else 0)

# Just count each time that each font type was
incorrect
incorrect = incorrect[['font',
                        'incorrect guesses']]
                        ].groupby('font').sum().reset_index()

# Now lets look at each individual's relationship
between Altered and base
# For now we will ignore symbols
rel_measure = guesses[guesses['font'] != 'symbol']
rel_measure = rel_measure[['email', 'font', 'letter',
measure]]
rel_measure = rel_measure.groupby(['email', 'font',
'letter']).mean()
rel_measure = rel_measure.unstack(
    level=-2).reset_index().droplevel(0,
axis=1).dropna()

#Looking at the relationship between narrow and
base
rel_measure2 = guesses[guesses['font'] != 'symbol']
rel_measure2 = rel_measure2[['email', 'font',
'letter', measure]]
rel_measure2 = rel_measure2.groupby(['email',
'font', 'letter']).mean()
rel_measure2 = rel_measure2.unstack(

```

```

    level=-2).reset_index().droplevel(0,
axis=1).dropna()

# >1 means that the new font is good. <1 means
the new font is not good
rel_measure['rel_measure'] =
rel_measure['Base']/rel_measure['Altered']-1
rel_measure2['rel_measure2'] =
rel_measure2['Base']/rel_measure2['Narrow']-1

return {'base': base.to_dict(),
        'Altered': Altered.to_dict(),
        'narrow': narrow.to_dict(),
        'traps': traps.to_dict(),
        'guesses': guesses.to_dict(),
        'incorrect': incorrect.to_dict(),
        'relat_measure': rel_measure.to_dict(),
        'relat_measure2': rel_measure2.to_dict(),
        'res': res,
        'res2': res2}

# %%
# Finally produce the pretty charts and metrics
@dash_app.callback(Output('summary_tb',
'children'),
                    Output('score', 'children'),
                    Output('p_score', 'children'),
                    Output('avg_improve', 'children'), #base v
Altered
                    Output('avg_improve2', 'children'), #base
v narrow
                    Output('freq_main',
'children'),#distribution (base v Altered )
                    Output('freq_main2',
'children'),#distribution2 (base v narrow)
                    Output('wrong_main', 'children'),
                    Output('rel_score_main',
'children'),#real_meas_dist (base v Altered )
                    Output('rel_score_main2',
'children'),#real_meas_dist2 (base v narrow)
                    Output('boxplot_main', 'children'),
                    Output('change_score', 'children'),#base v
Altered
                    Output('change_score2', 'children'),#base
v narrow
                    Output('linear_model', 'children'),#base v
Altered
                    Output('linear_model2', 'children'),#base
v narrow
                    Output('model_results', 'children'),#base v
Altered
                    Output('model_results2', 'children'),#base
v narrow
                    Input('analysis_data', 'data'),
                    Input('analysis_button', 'n_clicks'),

```

```

        Input('measure_dd', 'value'),
        Input('test_type_dd', 'value'))
def produce_graphs(data, n_clicks, measure,
test_type):
    if n_clicks < 1:
        raise PreventUpdate
    # First lets unpack the data
    base = pd.DataFrame(data['base'])
    Altered = pd.DataFrame(data['Altered'])
    narrow = pd.DataFrame(data['narrow'])
    guesses = pd.DataFrame(data['guesses'])
    incorrect = pd.DataFrame(data['incorrect'])
    rel_measure =
pd.DataFrame(data['relat_measure'])
    rel_measure2 =
pd.DataFrame(data['relat_measure2'])
    res = data['res']
    res2 = data['res2']

    # define the colors
    db = '#000872'
    lg = '#2FA156'
    gr = '#272B30'

    # distributions for base v Altered
    distributions = ff.create_distplot([
        base[measure],
        Altered [measure]],
        ['Base', 'Altered'],
        show_rug=False,
        show_hist=False,
        histnorm='probability',
        colors=[lg, db])

    distributions.update_layout(xaxis_range=[0, 40],
        paper_bgcolor='rgba(0,0,0,0)',
        font={'color': '#FFFFFF'},
        title=f'Frequency Distribution of
{measure}',
        xaxis_title=f'{measure} of
correct guess',
        yaxis_title='Frequency',
        yaxis={'tickformat': '.0%'})

    #Distributions for base v narrow
    distributions2 = ff.create_distplot([
        base[measure],
        narrow[measure]],
        ['Base', 'Narrow'],
        show_rug=False,
        show_hist=False,
        histnorm='probability',
        colors=[lg, db])

    distributions2.update_layout(xaxis_range=[0, 40],
        paper_bgcolor='rgba(0,0,0,0)',
        font={'color': '#FFFFFF'},

```

```

        title=f'Frequency Distribution of
{measure}',
        xaxis_title=f'{measure} of
correct guess',
        yaxis_title='Frequency',
        yaxis={'tickformat': '.0%'})

    # Bar chart showing incorrect answers
    incorrect_bars = px.bar(incorrect, x='font',
y='incorrect guesses',
        color='font',
        color_discrete_map={'Base': lg,
'Altered': db,
'symbol': gr})

    incorrect_bars.update_layout(showlegend=False,
        paper_bgcolor='rgba(0,0,0,0)',
        font={'color': '#FFFFFF'},
        title='Number of Incorrect
Guesses')

    # Relative Measure Distribution for base v Altered
    rel_meas_dist =
ff.create_distplot([rel_measure['rel_measure']],
        ['relationship'],
        histnorm='probability',
        bin_size=0.1,
        show_rug=False,
        colors=[db])

    rel_meas_dist.update_layout(showlegend=False,
        paper_bgcolor='rgba(0,0,0,0)',
        font={'color': '#FFFFFF'},
        title="Change Score of Altered
font
relative to base font",
        xaxis_title='Change Score',
        yaxis_title='Frequency',
        xaxis={'tickformat': '.0%'},
        yaxis={'tickformat': '.0%'})

    #Relative measure distribution for base v narrow
    rel_meas_dist2 =
ff.create_distplot([rel_measure2['rel_measure2']],
        ['relationship'],
        histnorm='probability',
        bin_size=0.1,
        show_rug=False,
        colors=[db])

    rel_meas_dist2.update_layout(showlegend=False,
        paper_bgcolor='rgba(0,0,0,0)',
        font={'color': '#FFFFFF'},
        title="Change Score of narrow
font
relative to base font",
        xaxis_title='Change Score',
        yaxis_title='Frequency',
        xaxis={'tickformat': '.0%'},

```

```

        yaxis={'tickformat': '.0%'})

# Box plots are helpful
box_plots = px.box(guesses.sort_values(by='font'),
                    x='font',
                    y=measure,
                    color='font',
                    color_discrete_map={'Base': lg,
                                        'Altered ': db,
                                        'symbol': gr})
box_plots.update_layout(showlegend=False,
                        paper_bgcolor='rgba(0,0,0,0)',
                        font={'color': '#FFFFFF'},
                        title=f'{measure} of each font type')

# Change Score Metrics (Base v Altered )
change_score = px.scatter(rel_measure,
                          'rel_measure',
                          'Base',
                          marginal_x='histogram',
                          color_discrete_sequence=[db])
change_score.update_layout(showlegend=False,
                           paper_bgcolor='rgba(0,0,0,0)',
                           font={'color': '#FFFFFF'},
                           title=f"Altered Font Change
Score Relative To Starting
Base Font {measure}"),
                    xaxis_title='Change Score',
                    yaxis_title=f'Base Font
{measure}',
                    xaxis={'tickformat': '.0%'})

# Change Score Metrics (Base v Narrow)
change_score2 = px.scatter(rel_measure2,
                            'rel_measure2',
                            'Base',
                            marginal_x='histogram',
                            color_discrete_sequence=[db])
change_score2.update_layout(showlegend=False,
                             paper_bgcolor='rgba(0,0,0,0)',
                             font={'color': '#FFFFFF'},
                             title=f"Narrow Font Change
Score Relative To Starting
Base Font {measure}"),
                    xaxis_title='Change Score',
                    yaxis_title=f'Base Font
{measure}',
                    xaxis={'tickformat': '.0%'})

# Base Size vs Altered Size
linear_model = px.scatter(rel_measure,
                          'Altered ',
                          'Base',
                          marginal_x='histogram',
                          marginal_y='histogram',
                          color_discrete_sequence=[db],
                          trendline='ols',
                          trendline_color_override='red',

```

```

        trendline_options={'add_constant':
0})

        linear_model.update_layout(showlegend=False,
                                   paper_bgcolor='rgba(0,0,0,0)',
                                   font={'color': '#FFFFFF'},
                                   title=f"Altered font {measure}
compared to
Base font {measure}"),
                    xaxis_title=f'Altered Font
{measure}',
                    yaxis_title=f'Base Font
{measure}',
                    xaxis_range=[0, 40],
                    yaxis_range=[0,40])

        results = px.get_trendline_results(linear_model)

        model_results =
results.px_fit_results.iloc[0].params
        model_results = np.round((model_results-1)*100,
1)[0]
        # Base Size vs Narrow Size
        linear_model2 = px.scatter(rel_measure2,
                                   'Narrow',
                                   'Base',
                                   marginal_x='histogram',
                                   marginal_y='histogram',
                                   color_discrete_sequence=[db],
                                   trendline='ols',
                                   trendline_color_override='red',
                                   trendline_options={'add_constant':
0})

        linear_model2.update_layout(showlegend=False,
                                    paper_bgcolor='rgba(0,0,0,0)',
                                    font={'color': '#FFFFFF'},
                                    title=f"Narrow Font {measure}
Compared To
Base Font {measure}"),
                    xaxis_title=f'Narrow Font
{measure}',
                    yaxis_title=f'Base Font
{measure}',
                    xaxis_range=[0, 40],
                    yaxis_range=[0,40])

        results2 = px.get_trendline_results(linear_model2)

        model_results2 =
results2.px_fit_results.iloc[0].params
        model_results2 = np.round((model_results2-
1)*100, 1)[0]
        # Pandas Dataframe for dash table
        summary_df = pd.DataFrame(
            {f'Base Font {measure}': [

```

```

base[measure].count(),
round(base[measure].mean(), 2),
round(base[measure].median(), 2),
round(base[measure].std(), 2)],
f'Altered Font {measure}': [
    Altered [measure].count(),
    round(Altered [measure].mean(), 2),
    round(Altered [measure].median(), 2),
    round(Altered [measure].std(), 2)],
f'Narrow Font {measure}': [
    narrow[measure].count(),
    round(narrow[measure].mean(), 2),
    round(narrow[measure].median(), 2),
    round(narrow[measure].std(), 2)]
    })
summary_df = summary_df.T.reset_index()
summary_df.rename(columns={'index': 'Font',
                           0: 'Observations',
                           1: f'Average {measure}',
                           2: f'Median {measure}',
                           3: 'Standard Deviation'},
inplace=True)

summary_tb = dash_table.DataTable(
    summary_df.to_dict('records'),
    [{'name': i, 'id': i} for i in summary_df.columns],
    style_header={'backgroundColor': 'rgb(39, 43,
48)',
                  'color': 'white'},
    style_data={'backgroundColor': 'rgb(39, 43, 48)',
                'color': 'white'})

# Relative score
rel_score =
str(round(rel_measure[rel_measure].mean()*100, 1))
+ ' %'
rel_score2 =
str(round(rel_measure2[rel_measure2].mean()*100,
1)) + ' %'
score = dbc.Card(
    dbc.CardBody(
        [html.H4('Average Improvement of Altered
Font over Base Font'),
         html.H2(rel_score),
         html.H4('Average Improvement of Narrow
Font over Base Font'),
         html.H2(rel_score2),
         html.P(f'Note: metric is Base Font
{measure}/Altered Font
{measure}-1, so >0 is an improvement and <0 is a
setback"')]
    ))

# distribution test scores
p_score = dbc.Card(
    dbc.CardBody(

```

```

[html.H4(f'P-Value of the {test_type} (Base v
Altered)'),
    html.H2(f' {round(res[1],3)}'),
    html.H4(f'P-Value of the {test_type} (Base v
Narrow)'),
    html.H2(f' {round(res2[1],3)}'),
    html.P("Note: The P-Value is basically
measuring what are the
odds that we would get these results by chance. A
lower p-value means it is
more likely that the change in font is the reason for
the results.
Typically, we look for a p-value below 0.05 to
suggest significance."))]
    ))

# distribution test scores Base v Altered
avg_improve = dbc.Card(
    dbc.CardBody(
        [html.H4(f'Base v Altered - Average
reduction in {measure}')],
    html.H2(round(base[measure].mean() -
                Altered [measure].mean(),
                2)),
        html.P(f'Calculation is Average
Base {measure} -
Average Altered {measure}. Positive means it
improved, negative means it did not"
        )))

# distribution test scores Base v Narrow
avg_improve2 = dbc.Card(
    dbc.CardBody(
        [html.H4(f'Base v Narrow - Average
reduction in {measure}')],
    html.H2(round(base[measure].mean() -
                narrow[measure].mean(),
                2)),
        html.P(f'Calculation is Average
Base {measure} -
Average Narrow {measure}. Positive means it
improved, negative means it did not"
        )))

return (summary_tb,
        score,
        p_score,
        avg_improve,
        avg_improve2,
        dcc.Graph(figure=distributions),
        dcc.Graph(figure=distributions2),
        dcc.Graph(figure=incorrect_bars),
        dcc.Graph(figure=rel_meas_dist),
        dcc.Graph(figure=rel_meas_dist2),
        dcc.Graph(figure=box_plots),
        dcc.Graph(figure=change_score),

```

```

        dcc.Graph(figure=change_score2),
        dcc.Graph(figure=linear_model),
        dcc.Graph(figure=linear_model2),
        html.Div(f'The model suggests a
{model_results}% improvement'),
        html.Div(f'The model suggests a
{model_results2}% improvement'))

```

## Words Test

```

# -*- coding: utf-8 -*-
"""

```

Created on Tue Feb 1 15:17:43 2022

@author: StephenTaylor

@editor: MartaPerez

```

"""

```

```

# Import necessary packages
from random import sample, randint # imported to
generate random numbers
import pandas as pd
import numpy as np
import plotly.figure_factory as ff
import plotly.express as px
from scipy.stats import ttest_ind, mannwhitneyu

import dash # Dash creates the website
from dash import html # Used to creat HTML
containers
from dash import dcc # Used to create dcc containers
from dash import dash_table
from dash.dependencies import Input, Output, State
# Import for callbacks
from dash.exceptions import PreventUpdate #
Prevents a callback to operate

from dash_extensions import Keyboard

import dash_bootstrap_components as dbc # Creates
grid layout for dash

from azure.storage.blob import BlobServiceClient #
Accesses Azure
from datetime import datetime # Easier work with
timestamps

# %%
"""This section creates a series of screens that can be
displayed based on
the results of a callback at the bottom of this script.
They are placed in
the script in the order they should be displayed in the
app."""

```

```

# %%
# This finally displays the final product. Helpful to
keep the debug on.
if __name__ == '__main__':
    dash_app.run_server(debug=True)

# This is the header row that is shown on all pages.
header_row = dbc.Row(dbc.Col([
    html.H1('Road Sign Font Study',
        style={'backgroundColor': '#000872', #
This is toXcel Blue
        'color': '#DFDFDF', # This is the grey
font color
        'margin-bottom': '0px'})),
    dcc.Markdown("A quick app to test the
legibility of various
fonts powered by
[toXcel](https://toxcel.com/)"),
        style={'backgroundColor': '#000872',
        'color': '#DFDFDF',
        'font-size': 24})]
    )
)

# %%
# This is the first page info wall complete with
instructions and the signin
info_wall = html.Div([
    html.Div(
        dcc.Markdown("
Welcome to the font study. This app is still in Beta
testing.
If you have been asked to participate, please provide
your email and the
password you have been provided")),
        html.Div(
            dcc.Input(id='input_email', # Email input used
in update_output
            type='email', # means it is expecting the
@ symbol
            placeholder='email', # Instructs user that
email goes there
            style={'width': '450px', # Formatting
changes
            'height': '45px',
            'padding': '10px',
            'margin-top': '15px',
            'font-size': '16px',
            'border-width': '3px',
            'border-color': '#a0a3a2'})),
            style={'display': 'flex',
            'justifyContent': 'center'}
        ),
        html.Div(

```

```

    dcc.Input(id='input_password', # password
input used in update_output
    type='text',
    placeholder='password',
    style={'width': '450px',
    'height': '45px',
    'padding': '10px',
    'margin-top': '15px',
    'font-size': '16px',
    'border-width': '3px',
    'border-color': '#a0a3a2'}),
    style={'display': 'flex',
    'justifyContent': 'center'}
    ),
    html.Div(
    html.Button('Verify', # Button for user to click
to login
        id='verify', # number of clicks used in
update_output
        n_clicks=0, # Starts counting at 0
        style={'border-width': '3px', # More
formatting
            'font-size': '14px'}),
    style={'display': 'flex',
    'justifyContent': 'center',
    'padding-top': '15px'}
    ),
    dbc.Row(
    html.Div(id='output1', # Output from
update_output. Text will change
        style={'display': 'flex',
        'justifyContent': 'center'})
    ))

# %%
# Second page is where the signs with letters will be
shown.
image_test = html.Div([
    # This is the actual sign to be displayed.
    Keyboard(id='keyboard'),
    dbc.Row(
    dbc.Button('I KNOW WHAT IT IS!', # Button
spread across top of screen
        id='confirm', # Input for time_elapsed
(stops timing)
        n_clicks=0, # Unnecessary
        color="success", # Color is green
        href='/yesorno'), # takes user to next
page
    style={'display': 'flex', # Formatting
    'justifyContent': 'center',
    'padding-top': '15px'}),

    dbc.Row(id='road_sign', # Output from
create_image
        align='center',

```

```

        justify='center'),

    # Counter. When the page is first displayed, it
starts counting at 0.
    html.Div(
    dcc.Interval(id="interval", # Input for
random_letters & create_image
        interval=120, # time between each
interval in ms
        max_intervals=500) # Maximum count.
    ),
    dbc.Row(
    dcc.Link('Log out', # logout button on every
page
        href='/', # takes user back to the info_wall
        style={'color': '#bed4c4', # More
formatting
            'font-family': 'serif',
            'font-weight': 'bold',
            "text-decoration": "none",
            'font-size': '20px'})
    ),
    style={'padding-left': '90%',
    'padding-top': '10px'}
    ))

# %%
# This is the third page where the user is asked if it
was a letter or not.
yes_or_no = html.Div(
    dbc.Row([
    dbc.Col(dbc.Button('Yes, I am ready to type the
word', # First button for YES
        id='yes', # Input to initial_guess
        n_clicks=0, # Starts counting at 0
        href='/response', # Takes users to
next page
        color='success'), # Green
        width=6),
    ])),

# %% Fourth page
response = html.Div([ # This is set at the same level
as previous buttons.
    html.Div(id='button_wording'), # Output from
button_wording
    dbc.Row([
    html.Div(id='tracker'),
    dcc.Link('Log out', # Same logout button

```

```

        href='/',
        style={ 'color': '#bed4c4',
                'font-family': 'serif',
                'font-weight': 'bold',
                "text-decoration": "none",
                'font-size': '20px'
              }
      ),
      style={ 'padding-left': '80%',
            'padding-top': '10px'
          }
    )
  ]
}

# %%
# Final page. Is shown once the participant is
finished.
thank_you = html.Div([
  dbc.Row(html.Div("Thank you for participating in
this study.
If you would like to participate again, close your
browser window and restart
the app.")),
  dbc.Row(
    dcc.Link('Log out',
             href='/',
             style={ 'color': '#bed4c4',
                    'font-family': 'serif',
                    'font-weight': 'bold',
                    "text-decoration": "none",
                    'font-size': '20px'
                  }
            ),
    style={ 'padding-left': '90%',
          'padding-top': '10px'
        }
    ))

# %%
# Analysis page. Can only be found by those who
already know where it is
analysis = html.Div(children=[
  # Top row with all of the filters and pull button
  dbc.Row(children=[
    dbc.Col(children=[
      html.Div('Filter by Word'),
      dcc.Dropdown(['All', 'Barley', 'Bartey',
'Eatery', 'Felony', 'Honors', 'Houses', 'Season',
'Senior', 'Sensor', 'Doting'],
                 placeholder='Filter by letter shown',
                 value='All',
                 multi=True,
                 id='letter_dd')]
    ),
    dbc.Col(children=[
      html.Div('Which Statistical Test?'),
      dcc.Dropdown(['t_test', 'mann_whitney_u'],
                  placeholder='Pick a Distribution
Test',
                  value='t_test',

```

```

        id='test_type_dd']]
    ),
    dbc.Col(children=[
      html.Div('Which Contrast?'),
      dcc.Dropdown(['All', 'diamond', 'rectangle'],
                  placeholder='Pick a Contrast',
                  value='All',
                  id='contrast_dd')]
    ),
    dbc.Col(children=[
      html.Div('Which Metric?'),
      dcc.Dropdown(['size', 'time'],
                  placeholder='Pick a Measure',
                  value='size',
                  id='measure_dd')]
    ),
    dbc.Col(
      dbc.Button('Pull Data',
                id='analysis_button',
                color='success',
                n_clicks=0)
    )
  ]),
  # First row Is the summary table
  dbc.Row(children=[
    html.Div("Select which filters above you would
like to see, then
click the \"Pull Data\" button to see summary statistics.
Note that each summary
statistic represents the moment that someone
recognized the letter."),
    html.Div(id='summary_tb')]),

# Final row has the scores of each model.
dbc.Row(children=[
  dbc.Col(html.Div(id='avg_improve')),
  dbc.Col(html.Div(id='avg_improve2')),
  dbc.Col(html.Div(id='avg_improve3')),
  dbc.Col(html.Div(id='p_score')),
  dbc.Col(html.Div(id='score'))
]),

# Second row includes The two scatter plots with
their fit lines
dbc.Row(children=[
  # First Column - Base vs Altered Scores
  dbc.Col(children=[
    html.Div(id='linear_model'),
    html.Div(id='model_results'),
    html.Div(id='linear_model2'),
    html.Div(id='model_results2'),
    html.Div(id='linear_model3'),
    html.Div(id='model_results3')
  ]),

```

```

    # Second Column - Change Score relative to
starting position
    dbc.Col(children=[
        html.Div(id='change_score'),
        html.Div(id='change_score2'),
        html.Div(id='change_score3')
    ])
]),

# Third Row includes the two frequency
distributions
dbc.Row(children=[
    # First Column - Frequency Distribution Chart
    dbc.Col(children=[
        html.Div(id='freq_main'),
        html.Div(id='freq_main2'),
        html.Div(id='freq_main3')
    ]),
    # Second Column Column - Relative Scores
    dbc.Col(children=[
        html.Div(id='rel_score_main'),
        html.Div(id='rel_score_main2'),
        html.Div(id='rel_score_main3')
    ])
]),
# Fourth Row includes the box plots of each font
type as well as incorrect
dbc.Row(children=[
    # First Column - Boxplots
    dbc.Col(children=[
        html.Div(id='boxplot_main')
    ]),

    # Second Column - Incorrect Guesses
    dbc.Col(children=[
        html.Div(id='wrong_main')
    ]),
])

# %%
# Create the app
dash_app = dash.Dash(__name__,
    # this is the darker theme that toXcel
uses in apps.

external_stylesheets=[dbc.themes.SLATE],
    # Do not want to show callback
exceptions
    suppress_callback_exceptions=True,
    # Do not want the page to look like its
loading always
    update_title=None)

# A little code used to pull the css file in the assets
folder.

```

```

dash_app.css.config.serve_locally = True
dash_app.scripts.config.serve_locally = True

# Line added for GitHub/Azure integration.
app = dash_app.server

# Activate the dashboard with the necessary graphics
dash_app.layout = html.Div([header_row,
    dcc.Location(id='url',
refresh=False),
    html.Div(id='page-content'),
    dcc.Store(id='size',
        storage_type='session'),
    dcc.Store(id='email',
        storage_type='session'),
    dcc.Store(id='sign_shown',
        storage_type='session'),
    dcc.Store(id='letter_shown',
        storage_type='session'),
    dcc.Store(id='font_shown',
        storage_type='session'),
    dcc.Store(id='test_count',
        storage_type='session'),
    dcc.Store(id='start_time',
        storage_type='session'),
    dcc.Store(id='time_elapsed',
        storage_type='session'),
    dcc.Store(id='initial_answer',
        storage_type='session'),
    dcc.Store(id='random_list',
        storage_type='session'),
    dcc.Store(id='keyed_time',
        storage_type='session'),
    dcc.Store(id='analysis_data',
        storage_type='session')
])

# %%
# The following section includes all of the callbacks.
@dash_app.callback(Output('keyed_time', 'data'),
    Input('keyed_time', 'data'),
    Input('keyboard', 'keydown'))
def keyed_event(keyed_time, event):
    if keyed_time is None:
        return {'value': 0}
    elif event is None:
        return {'value': keyed_time['value']}
    else:
        return {'value': datetime.now().timestamp()}

@dash_app.callback(Output('page-content',
'children'),
    inputs=[Input('url', 'pathname')])
def display_page(pathname):

```

"""This function takes the url provided in the app layout above and shows the correct page based on the url. The pages are defined in the first half of this script."""

```
# This is an if statement. python checks the first if statement. If it is # true, then it does what it says to do (in this case returns image_test) # and moves on.
```

```
# if pathname == '/image_test' and keyed['key'] == '':
#     return yes_or_no
if pathname == '/image_test':
    return image_test
elif pathname == '/response':
    return response
elif pathname == '/yesorno':
    return yes_or_no
elif pathname == '/thank_you':
    return thank_you
elif pathname == '/analysis':
    return analysis
else:
    return info_wall
```

```
@dash_app.callback(
    # Outputs the text to be shown after clicking verify.
    # Is input for the first page
    Output('output1', 'children'),
    # Outputs the user's email to be saved in the responses
    # Is input for restart_interval
    Output('email', 'data'),
    # Creates a random list used to pull letters and fonts later
    # Is input for random_letters
    Output('random_list', 'data'),
    # Input from verify button on first page.
    inputs=Input('verify', 'n_clicks'),
    # State is used to input values that may or may not exist.
    # Input from email text box on first page.
    state=[State('input_email', 'value'),
           # Input from password text box on first page.
           State('input_password', 'value')])
def update_output(n_clicks, uname, passw):
    """Inputs:
    n-clicks - from verify button
    uname - from email text box
    passw - from password text box
```

If the verify button is clicked, this function begins automatically working. This function outputs:  
 text - to be displayed on the first page and a button if the user provides the correct login information.  
 email - stores their email for tracking purposes  
 random\_list - a list of random numbers between 3 and 32 (inclusive) which is used to ensure random showing of letters, but that each letter is shown only once with each font type."""

```
# Prevents function from doing anything unless verify is clicked.
if n_clicks < 1:
    raise PreventUpdate
# Tries to split the email at the @ symbol. (Case does not matter)
try:
    domain = uname.split('@', 1)[1].upper()
    # If the domain of the email contains these three options, it lets the
    # user into the app.
    if ((domain == 'TOXCEL.COM'
        or domain == 'DOT.GOV'
        or domain == 'BATTELLE.ORG')
        and passw == 'toxcel'):
    # Show the instructions and continue button
    # if username and password are correct.
    what_to_show = html.Div([
        dbc.Row(html.Pre(children=[
            html.Strong('PLEASE READ THE FOLLOWING INSTRUCTIONS:'),
            html.Pre("""
```

Thank you for volunteering to participate in this legibility study. Before we begin, please make sure that you are sitting normally in your chair without leaning forward or slouching. For the duration of the study, it is important to remain approximately the same distance from the screen. Please place your hand on your mouse.

Once the study begins, a colored rectangle will appear on the screen. A small word will appear in the center of the screen and increase in size as if you were driving towards it in your car. The word will eventually be large enough for you to see so please do not squint or strain your eyes. Please do not try and guess what the word is before it becomes legible.

Once the word becomes legible to you, click the green button above the sign with the mouse, or click any button on your keyboard. This will mark the distance at which you can see the word on the sign so it is important that you press it once you can clearly identify what the word is.

If you select a button on your keyboard, the test will stop and the distance at the time of the button press will be measured; however, you will still have to click the green button to advance to the next page.

When you click the green button, the sign will disappear completely, and you will be given the opportunity to indicate if you are ready to type out the word.

You will be given the opportunity to indicate what the word was. Type in the provided text box what word you saw. It is important to note that the case of the letters matters. When you are ready for the next word, select the button in the top left corner.

You have the option to opt out of the test at any point by selecting the button that says “Log out.” Note, the test records your answers in real time, in case you lose access to the test for any reason.

You will notice that some signs may repeat the same words. That behavior is normal for the test, and not an indication of anything wrong with the program. The first three signs will be test images and will be the same across all tests. After the first three, the test will begin and the font, color of sign, and character displayed will be randomized. You will be shown a total of 46 signs. Thank you again for your participation.

```

    ""))],
      # Button to show at the bottom of the
screen
      dbc.Row(dbc.Button('Click here when you
are ready.',
                        href='/image_test', # Takes
user to test.
                        style={'font-weight': 'bold',

```

```

        "text-decoration": "none",
        'font-size': '20px',
        'font-family': 'Altered'},
        color='success'),
        style={'padding-top': '15px'}}))
    else:
        # If the user put in a valid email but it is
incorrect or if the
        # password is incorrect, show this text.
        what_to_show = html.Div(
            children="Sorry, I do not have your
information.
Please contact the research team at toXcel to get
access",
            style={'padding-top': '15px',
                  'font-size': '16px'})
        # If we cannot split on the @ symbol for some
reason, show this text.
    except IndexError:
        what_to_show = html.Div(
            children="Please input valid email address",
            style={'padding-top': '15px',
                  'font-size': '16px'})
        # Outputs of the function are: text to output, email,
random list.
    return (what_to_show,
            {'value': uname},
            {'value': sample(range(3, 43), 40)}
            )

# If you are still reading this... you are dedicated!
Almost there!
@dash_app.callback(
    # This one is weird. Its inputs (as states) and
outputs are mostly the same
    # Output which sign should be shown (Green or
Yellow)
    Output('sign_shown', 'data'),
    # Output which letter should be shown
    Output('letter_shown', 'data'),
    # Output which font should be used in this test
    Output('font_shown', 'data'),
    # Increments the test count
    Output('test_count', 'data'),
    # Starts a timer.
    Output('start_time', 'data'),

    # Input the current interval (this will change every
120 ms)
    inputs=[Input('interval', 'n_intervals'),
            # Input what the test count is (This will
change every test)
            Input('test_count', 'data')],
    # If they are available input the same outputs
state=[State('sign_shown', 'data'),

```

```

        State('letter_shown', 'data'),
        State('font_shown', 'data'),
        State('start_time', 'data'),
        # Input the random list generated in the
function above.
        State('random_list', 'data')]
    )
def random_letters(interval, test, sign, letter, font,
start_time, rand_list):
    """Randomly generates the letters and fonts for the
test. Also starts
    timing the current test, and increments the test
number as it goes.
    Inputs:
        Interval - from image_test. Increments every
120 ms a sign is shown
        test - Counts the number of total tests that have
started
        sign - which sign is being be shown (green or
yellow)
        letter - which letter is being shown
        font - which font is being shown
        start_time - what time did the current test start
        rand_list - from update_output. List of random
numbers used to
        randomize the order that letters and fonts are
shown, but ensures they
        are all shown exactly one time.

    Outputs stay the same as the inputs unless this is a
new test.
    Outputs:
        sign - which sign should be shown
        letter - which letter should be shown
        font - which font should the letter have
        test - what test number are we on?
        start_time - starts timing if the interval is 0,
otherwise is a
        passthrough variable."""
    # If the interval is 0, then treat this as a new test.
Generate a new
    # sign, letter, font, and start timing.
    if interval == 0 or interval is None:
        # Simple test to make sure we have a test
number. The only time this is
        # not true is if it is the first run.
        try:
            test_num = test['value']
        except TypeError:
            test_num = 0
        # For the first three tests, show the same test
images: A, a, %
        # Note: python is 0 indexed, meaning counting
starts at 0, NOT at 1.
        if test_num < 3:
            sign_num = 0

```

```

        letter_num = test_num

    else:
        # If this is not one of the first three tests, then
randomly pull a
        # letter and font pair based on the random
number list generated.
        sign_num = randint(0,1)
        random_num = rand_list['value']
        # This looks funky because counting is hard :)
        letter_num = random_num[test_num-3]

        # This is legacy code. We were originally asked
to look at two sign
        # types, but since have been asked only to look
at white font on green.
        # Performance hit is minimal, so I have chosen
to leave it for now bc
        # I think we will want to add it back.
        sign = ['rectangle', 'diamond'][sign_num]

        # Pairs of letters and fonts. The length of this list
(43 currently) is
        # crucial. If you add letter/font pairs to this list,
you need to
        # change the length of the random numbers list
(output from
        # update_output), the number of tests before
killing the program (
        # output from restart_interval), and the
instructions.
        Itrfmt = [['Barley', 'Base'], ['Season', 'Base'],
['Honors', 'Altered'],
            ['Barley', 'Base'], ['Bartey', 'Base'],
['Eatery', 'Base'], ['Felony', 'Base'],
            ['Honors', 'Base'], ['Houses', 'Base'],
['Season', 'Base'], ['Senior', 'Base'],
            ['Sensor', 'Base'], ['Doting', 'Base'],
            ['Barley', 'Narrow'], ['Bartey', 'Narrow'],
['Eatery', 'Narrow'], ['Felony', 'Narrow'],
            ['Honors', 'Narrow'], ['Houses', 'Narrow'],
['Season', 'Narrow'], ['Senior', 'Narrow'],
            ['Sensor', 'Narrow'], ['Doting', 'Narrow'],
            ['Barley', 'Altered'], ['Bartey', 'Altered'],
['Eatery', 'Altered'], ['Felony', 'Altered'],
            ['Honors', 'Altered'], ['Houses', 'Altered'],
['Season', 'Altered'], ['Senior', 'Altered'],
            ['Sensor', 'Altered'], ['Doting', 'Altered'],
            ['Barley', 'AlteredC'], ['Bartey',
'AlteredC'], ['Eatery', 'AlteredC'], ['Felony',
'AlteredC'],
            ['Honors', 'AlteredC'], ['Houses',
'AlteredC'], ['Season', 'AlteredC'], ['Senior',
'AlteredC'],
            ['Sensor', 'AlteredC'], ['Doting',
'AlteredC']] [letter_num]

```

```

# The letter is the first thing in the pair.
letter = ltrfnt[0]
# The font of the letter is the second thing in the
pair.
font = ltrfnt[1]
# Start timing.
start_time = datetime.now().timestamp()
return({'value': sign},
        {'value': letter},
        {'value': font},
        {'value': test_num + 1},
        {'value': start_time})
# If this is not the first interval in a test, just keep
the same values.
else:
    return(sign,
           letter,
           font,
           test,
           start_time)

# This outputs the actual letters/signs. Also records
the size of the letter.
@dash_app.callback(
    Output('road_sign', 'children'),
    Output('size', 'data'),
    # All inputs from random_letter
    inputs=[Input('sign_shown', 'data'),
            Input('letter_shown', 'data'),
            Input('font_shown', 'data')],
    # Inputs the interval number.
    state=[State('keyboard', 'keydown'),
           State('interval', 'n_intervals'),
           State('size', 'data')]
)
def create_image(random_sign, random_letter,
                random_font, keyed, interval, s):
    """Creates colored rectangle with a letter in the
    center. The rectangle
    remains the same size, but the appears to grow
    over time.
    Inputs:
        random_sign - output from random_letters,
        random_letter - output from random_letters
        random_font - output from random_letters
        interval - incremented every 120 ms.
    Outputs:
        road_sign - card to be displayed on image_test
        size - records the size of the letter shown."""
    # This only will run if you are on the correct page.
    if interval is None:
        raise PreventUpdate

    # Brings in the data dictionaries generated in the
    function above.

```

```

if keyed is None:
    sign = random_sign['value']
    letter = random_letter['value']
    font = random_font['value']

# Size is 1/6th of the current interval.
size = interval/6

# Keeps the letter mostly center. Allows for
some "jumping" to mimic
# More accurate driving.
location = 200 - size

# Also legacy code. Left because we will likely
start testing on diff
# backgrounds. For now, it jumps straight to the
else statement (green)

if sign == 'diamond':
    card = dbc.Col(html.Div(letter,
                             style={'margin-top':
                                     f'{location}px'}),
                   width=6,
                   style={'font-size': size,
                           'text-align': 'center',
                           'backgroundColor': '#FFFFFF',
                           'font-family': font,
                           'color': '#000000',
                           'height': '400px'},
                           align='center')
    else:
        card = dbc.Col(html.Div(letter,
                                 style={'margin-top':
                                         f'{location}px'}),
                        width=6,
                        style={'font-size': size,
                                'text-align': 'center',
                                'backgroundColor': '#006747',
                                'font-family': font,
                                'color': '#FFFFFF',
                                'height': '400px'},
                                align='center')
        else:
            card = html.Pre("Your time has stopped.
            Select the green button to go to the next page")
            size = s['value']

    return (card,
            {'value': size})

# %%
# This callback calculates how long it took for a user
to recognize the letter.

```

```

@dash_app.callback(
    Output('time_elapsed', 'data'),
    inputs=[Input('start_time', 'data'),
            Input('keyed_time', 'data'),
            Input('confirm', 'n_clicks')])
def time_elapsed(start_time, key_time, clicks):
    """Calculates the time passed for each test.
    Inputs:
        clicks - counts how many times the confirm
        button is clicked.
        start_time - imported from random_letters to
        calc time elapsed
    Outputs:
        time_elapsed - How long it took for someone to
        identify letter in s"""
    # Is added to prevent callback errors, but also as a
    failsafe in case
    # the timer does not start.
    if clicks < 1:
        raise PreventUpdate

    if start_time is None:
        return({'value': -1})
    else:
        if key_time['value'] < start_time['value']:
            end_time = datetime.now().timestamp()
        else:
            end_time = key_time['value']
        return({'value': end_time-start_time['value']})

# %%
@dash_app.callback(
    Output('initial_answer', 'data'),
    Input('yes', 'n_clicks'))
def initial_guess(clicks):
    """Records and stores whether the user thought the
    character was a letter
    or not.
    Inputs:
        yes - counts the number of times the yes button
        is clicked. If it is
        not clicked, it assumes the no button was
        clicked.
    Outputs:
        initial_answer - records that the user recognized
        that it either was
        or was not a letter. Used as an input in
        restart_interval"""
    # Note, for performance sake, it is always better to
    use booleans than
    # strings like 'yes' or 'no'. Also typically better than
    1 and 0.
    if clicks == 1:
        return({'value': True})
    else:

```

```

        return({'value': False})

@dash_app.callback(Output('button_wording',
'children'),
    Input('test_count', 'data'),
    Input('initial_answer', 'data'))
def button_wording(test, initial_guess):
    """This is a minor callback that could be removed
    entirely. I just like the
    idea of letting the user know whether they are still
    in a test or are
    actually being evaluated.
    Inputs:
        test_count - Which test number are we on?
    Output from random_letters
    Outputs:
        button_wording - outputs the button that should
        be shown based on
        which test number the user is on."""
    try:
        test_num = test['value']
    except TypeError:
        test_num = 0
    initial_guess = initial_guess['value']
    if initial_guess:
        input_box = dbc.Row(
            dcc.Input(id='guess', # Input for
            restart_interval
                type='text',
                placeholder='What did you see?', #
                Asks user what letter
                style={'width': '450px', # More
                formatting
                    'height': '45px',
                    'padding': '10px',
                    'margin-top': '15px',
                    'font-size': '20px',
                    'border-width': '3px',
                    'border-color': '#a0a3a2',
                    'font-family': 'serif'},
                    autoComplete='off')
            )
    else:
        input_box = html.Div(id='guess')
        # If the user is still in the first three test images
        show this in yellow
        if test_num < 3:
            return (dbc.Button('I am ready for my next test
            image',
                id='back_to_test',
                color='warning',
                n_clicks=0),
                input_box)
        # If the user has finished their last test image, show
        this in red

```

```

elif test_num == 3:
    return (dbc.Button('I am ready to start',
        id='back_to_test',
        color='danger',
        n_clicks=0),
        input_box)
# All others, show this in green.
else:
    return (dbc.Button('I am ready for the next one',
        id='back_to_test',
        color='success',
        n_clicks=0),
        input_box)

# This is the last one! You have almost made it.
@dash_app.callback(
    Output('url', 'pathname'),
    inputs=[Input('back_to_test', 'n_clicks'),
        Input('test_count', 'data')],
    state=[State('guess', 'value'),
        State('size', 'data'),
        State('sign_shown', 'data'),
        State('letter_shown', 'data'),
        State('font_shown', 'data'),
        State('email', 'data'),
        State('time_elapsed', 'data'),
        State('initial_answer', 'data')])
def restart_interval(clicks, test, guess, size, sign,
    letter, font, email,
        time_elapsed, initial_guess):
    """This function practically takes all of the previous
    outputs and saves
    them to a txt file. Answers are written to an append
    blob as the user
    provides them, so if a user does not finish, we can
    still record their
    results.
    Inputs:
    Clicks - Runs the function as soon as someone
    clicks back_to_test.
    test - which test count are we on? Output from
    random_letters
    guess - what value did the user put in the text
    box in response
    size - what was the final size of the letter shown
    in create_image
    sign - what color sign was shown? Output from
    random_letters
    letter - what letter was shown? Output from
    random_letters
    font - what font was shown? Output from
    random_letters
    email - what email did they provide? Output
    from update_output

```

```

        time_elapsed - How long did it take them to
        recognize the letter?
        output from time_elapsed
        initial_guess - Did they recognize it as a letter or
        not?"""
    if clicks < 1:
        raise PreventUpdate
    else:
        guess = 'symbol' if guess is None else guess
        size = size['value']
        sign = sign['value']
        letter = letter['value']
        font = font['value']
        email = email['value']
        int_guess = initial_guess['value']
        timed_time = time_elapsed['value']
        if int_guess:
            correct = letter == guess.strip() # removes
            whitespace if any
        else:
            correct = True if letter in ['>', '%', '^', '<'] else
            False

        # writes to a new line (\n) the answers for each
        test.
        data =
        ""\n{0},{1},{2},{3},{4},{5},{6},{7},{8}"".format(
            email, sign, letter, int_guess, guess, font, size,
            timed_time, correct)

        # creates a connection with the Azure Blob
        Storage
        service = BlobServiceClient(
            # If these credentials get out, please notify
            Stephen at
            # stephen.taylor@toxcel.com as soon a
            practical. We can generate
            # new keys, so it is not a big deal.

            account_url='https://safecurves.blob.core.windows.ne
            t',

            credential='p6PwDNuoxvxMUgpVaybejqABBujZRI
            5Jkjq/WLba93ex/6NsH8JYsChfysqRzOG+Pz2R/+9u
            1pARBOMBTKFsJg==')

        client =
        service.get_blob_client(container='safecurves',
            blob='Words.txt',)

        # appends the data string to the blob.
        client.append_block(data)

    try:
        test_num = test['value']
    except TypeError:

```

```

test_num = 0

# Test 0-2 is test images. Then 40 letter/font pairs.
Then kill the program
if test_num < 43:
    return ('/image_test')
else:
    return ('/thank_you')

@dash_app.callback(Output('tracker', 'children'),
                    Input('test_count', 'data'))
def tracker_update(test):
    test = test['value']
    if test > 3:
        return html.Pre('Character #{}/40'.format(test-
3))
    else:
        return html.Pre('Test Character
#{}/3'.format(test))

# %%
# These are the callbacks for the analysis page
@dash_app.callback(Output('analysis_data', 'data'),
                    Input('analysis_button', 'n_clicks'),
                    Input('test_type_dd', 'value'),
                    Input('measure_dd', 'value'),
                    Input('letter_dd', 'value'),
                    Input('contrast_dd', 'value'))
def pull_clean_data(nclicks, test_type, measure,
letter_filter, contrast):
    if nclicks < 1:
        raise PreventUpdate
    elif test_type is None or measure is None:
        raise PreventUpdate

    guesses =
pd.read_csv('https://safecurves.blob.core.windows.ne
t/safecurves/Words.txt',
            sep=',',
            on_bad_lines='skip').reset_index()
    if 'All' in letter_filter:
        pass
    else:
        guesses =
guesses[guesses['letter'].isin(letter_filter)]
    #defining the drop down in contrast value
    if 'All' in contrast:
        pass
    else:
        guesses = guesses[guesses['sign'] == contrast]

# When I built this, I used test@toxcel to let me
know where I was testing

```

```

# Let's also look out for speeders. Any time that is
less than 2 seconds
# should raise a red flag.
guesses = guesses.sort_values(by=['email', 'index'])

guesses = guesses[guesses['email'] !=
'test@toxcel.com']

guesses['speeder?'] = guesses['time'].apply(lambda
t:
    True if t < 2
    else False)

# The correct column is case specific, which is
causing problems
# with x, z, and w. this new feature ignores case.

guesses['correct'] =
np.where((guesses['letter'].str.upper() ==
        guesses['guess'].str.upper()) |
        ((guesses['letter'].isin(['%',
            '<',
            '>',
            '^']))
        & (guesses['guess'] ==
'symbol')),
        True, False)

guesses.loc[guesses['letter'].isin(['%', '<', '>', '^']),
'font'] = np.nan
guesses['font'].fillna('symbol', inplace=True)
# Now lets highlight the test letters
# First create a temporary column to be dropped
later that has combines the
# letter and font. Eg. ABase, %Altered
guesses['temp'] = guesses['letter'] + guesses['font']

# Now shift that temporary column down once and
twice
guesses['temp1'] = guesses['temp'].shift(-1,
fill_value=0)
guesses['temp2'] = guesses['temp'].shift(-2,
fill_value=0)

# Now highlight test
guesses['test?'] = np.where((guesses['temp'] ==
'ABase')
        & (guesses['temp1'] == 'aBase')
        & (guesses['temp2'] ==
'%symbol'),
        True,
        False)

guesses.drop(columns=['temp', 'temp1', 'temp2'],
inplace=True)

```

```

guesses = guesses[guesses['test?'] == False]

# Remove outliers for folks that fell asleep at their
desk
guesses = guesses[guesses['time'] < 100]
#remove outliers for quick guesses
guesses = guesses[guesses['size'] > 6]
guesses = guesses[guesses['size'] < 25]

guesses.sort_values(by=['font'], inplace=True)

# separate traps from base from Altered
traps = guesses[guesses['font'] == 'symbol']
base = guesses[guesses['font'] == 'Base']
Altered = guesses[guesses['font'] == 'Altered']
narrow = guesses[guesses['font'] == 'Narrow']
AlteredC = guesses[guesses['font'] == 'AlteredC']

# Run a ttest. The data is pretty normally
distributed. I am happy with it
if test_type == 't_test':
    res = ttest_ind(base[measure], Altered[measure],
                    alternative='greater')
    res2 = ttest_ind(base[measure],
narrow[measure], alternative='greater')
    res3 = ttest_ind(base[measure],
AlteredC[measure], alternative='greater')

    elif test_type == 'mann_whitney_u':
        res = mannwhitneyu(base[measure],
Altered[measure],
                        alternative='greater')
        res2 = mannwhitneyu(base[measure],
narrow[measure],
                        alternative='greater')
        res3 = mannwhitneyu(base[measure],
AlteredC[measure],
                        alternative='greater')

# Now lets look at incorrect guesses.
incorrect = guesses[['email', 'font', 'letter',
'correct']]
incorrect['incorrect guesses'] =
incorrect['correct'].apply(lambda c:
                            1 if c is False
                            else 0)

# Just count each time that each font type was
incorrect
incorrect = incorrect[['font',
'incorrect guesses']
                    ].groupby('font').sum().reset_index()

# Now lets look at each individual's relationship
between Altered and base
# For now we will ignore symbols

```

```

rel_measure = guesses[guesses['font'] != 'symbol']
rel_measure = rel_measure[['email', 'font', 'letter',
measure]]
rel_measure = rel_measure.groupby(['email', 'font',
'letter']).mean()
rel_measure = rel_measure.unstack(
    level=-2).reset_index().droplevel(0,
axis=1).dropna()

#Looking at the relationship between narrow and
base
rel_measure2 = guesses[guesses['font'] != 'symbol']
rel_measure2 = rel_measure2[['email', 'font',
'letter', measure]]
rel_measure2 = rel_measure2.groupby(['email',
'font', 'letter']).mean()
rel_measure2 = rel_measure2.unstack(
    level=-2).reset_index().droplevel(0,
axis=1).dropna()

#Looking at the relationship between AlteredC and
base
rel_measure3 = guesses[guesses['font'] != 'symbol']
rel_measure3 = rel_measure3[['email', 'font',
'letter', measure]]
rel_measure3 = rel_measure3.groupby(['email',
'font', 'letter']).mean()
rel_measure3 = rel_measure3.unstack(
    level=-2).reset_index().droplevel(0,
axis=1).dropna()

# >1 means that the new font is good. <1 means
the new font is not good
rel_measure['rel_measure'] =
rel_measure['Base']/rel_measure['Altered']-1
rel_measure2['rel_measure2'] =
rel_measure2['Base']/rel_measure2['Narrow']-1
rel_measure3['rel_measure3'] =
rel_measure3['Base']/rel_measure2['AlteredC']-1

return {'base': base.to_dict(),
        'Altered': Altered.to_dict(),
        'narrow': narrow.to_dict(),
        'AlteredC': AlteredC.to_dict(),
        'traps': traps.to_dict(),
        'guesses': guesses.to_dict(),
        'incorrect': incorrect.to_dict(),
        'relat_measure': rel_measure.to_dict(),
        'relat_measure2': rel_measure2.to_dict(),
        'relat_measure3': rel_measure3.to_dict(),
        'res': res,
        'res2': res2,
        'res3': res3}

# %%

```

```

# Finally produce the pretty charts and metrics
@dash_app.callback(Output('summary_tb',
'children'),
                    Output('score', 'children'),
                    Output('p_score', 'children'),
                    Output('avg_improve', 'children'), #base v
Altered
                    Output('avg_improve2', 'children'), #base
v narrow
                    Output('avg_improve3', 'children'), #base
v AlteredC
                    Output('freq_main',
'children'),#distribution (base v Altered)
                    Output('freq_main2',
'children'),#distribution2 (base v narrow)
                    Output('freq_main3',
'children'),#distribution3 (base v AlteredC)
                    Output('wrong_main', 'children'),
                    Output('rel_score_main',
'children'),#real_meas_dist (base v Altered)
                    Output('rel_score_main2',
'children'),#real_meas_dist2 (base v narrow)
                    Output('rel_score_main3',
'children'),#real_meas_dist3 (base v AlteredC)
                    Output('boxplot_main', 'children'),
                    Output('change_score', 'children'),#base v
Altered
                    Output('change_score2', 'children'),#base
v narrow
                    Output('change_score3', 'children'),#base
v AlteredC
                    Output('linear_model', 'children'),#base v
Altered
                    Output('linear_model2', 'children'),#base
v narrow
                    Output('linear_model3', 'children'),#base
v AlteredC
                    Output('model_results', 'children'),#base v
Altered
                    Output('model_results2', 'children'),#base
v narrow
                    Output('model_results3', 'children'),#base
v AlteredC
                    Input('analysis_data', 'data'),
                    Input('analysis_button', 'n_clicks'),
                    Input('measure_dd', 'value'),
                    Input('test_type_dd', 'value'))
def produce_graphs(data, n_clicks, measure,
test_type):
    if n_clicks < 1:
        raise PreventUpdate
    # First lets unpack the data
    base = pd.DataFrame(data['base'])
    Altered = pd.DataFrame(data['Altered'])
    narrow = pd.DataFrame(data['narrow'])
    AlteredC = pd.DataFrame(data['AlteredC'])

```

```

guesses = pd.DataFrame(data['guesses'])
incorrect = pd.DataFrame(data['incorrect'])
rel_measure =
pd.DataFrame(data['relat_measure'])
rel_measure2 =
pd.DataFrame(data['relat_measure2'])
rel_measure3 =
pd.DataFrame(data['relat_measure3'])
res = data['res']
res2 = data['res2']
res3 = data['res3']

# define the colors
db = '#000872'
lg = '#2FA156'
gr = '#272B30'

# Show off those beautiful distributions!
distributions = ff.create_distplot([
                                base[measure],
                                Altered[measure]],
                                ['Base', 'Altered'],
                                show_rug=False,
                                show_hist=False,
                                histnorm='probability',
                                colors=[lg, db])

distributions.update_layout(xaxis_range=[0, 40],
                            paper_bgcolor='rgba(0,0,0,0)',
                            font={'color': '#FFFFFF'},
                            title=f'Frequency Distribution of
{measure}',
                            xaxis_title=f'{measure} of
correct guess',
                            yaxis_title='Frequency',
                            yaxis={'tickformat': '.0%'})
#Distributions for base v narrow
distributions2 = ff.create_distplot([
                                base[measure],
                                narrow[measure]],
                                ['Base', 'Narrow'],
                                show_rug=False,
                                show_hist=False,
                                histnorm='probability',
                                colors=[lg, db])

distributions2.update_layout(xaxis_range=[0, 40],
                             paper_bgcolor='rgba(0,0,0,0)',
                             font={'color': '#FFFFFF'},
                             title=f'Frequency Distribution of
{measure}',
                             xaxis_title=f'{measure} of
correct guess',
                             yaxis_title='Frequency',
                             yaxis={'tickformat': '.0%'})
#Distributions for base v AlteredC

```

```

distributions3 = ff.create_distplot([
    base[measure],
    AlteredC[measure]],
    ['Base', 'AlteredC'],
    show_rug=False,
    show_hist=False,
    histnorm='probability',
    colors=[lg, db])

distributions3.update_layout(xaxis_range=[0, 40],
    paper_bgcolor='rgba(0,0,0,0)',
    font={'color': '#FFFFFF'},
    title=f'Frequency Distribution of
{measure}',
    xaxis_title=f'{measure} of
correct guess',
    yaxis_title='Frequency',
    yaxis={'tickformat': '.0%'})

# Bar chart showing incorrect answers
incorrect_bars = px.bar(incorrect, x='font',
y='incorrect guesses',
    color='font',
    color_discrete_map={'Base': lg,
    'Altered': db,
    'symbol': gr})

incorrect_bars.update_layout(showlegend=False,
    paper_bgcolor='rgba(0,0,0,0)',
    font={'color': '#FFFFFF'},
    title='Number of Incorrect
Guesses')

# Relative Measure Distribution
rel_meas_dist =
ff.create_distplot([rel_measure[rel_measure]],
    ['relationship'],
    histnorm='probability',
    bin_size=0.1,
    show_rug=False,
    colors=[db])
rel_meas_dist.update_layout(showlegend=False,
    paper_bgcolor='rgba(0,0,0,0)',
    font={'color': '#FFFFFF'},
    title="Change Score of Altered
font
relative to base font",
    xaxis_title='Change Score',
    yaxis_title='Frequency',
    xaxis={'tickformat': '.0%'},
    yaxis={'tickformat': '.0%'})

#Relative measure distribution for base v narrow
rel_meas_dist2 =
ff.create_distplot([rel_measure2[rel_measure2]],
    ['relationship'],
    histnorm='probability',
    bin_size=0.1,
    show_rug=False,
    colors=[db])
rel_meas_dist2.update_layout(showlegend=False,
    paper_bgcolor='rgba(0,0,0,0)',
    font={'color': '#FFFFFF'},
    title="Change Score of narrow
font
relative to base font",
    xaxis_title='Change Score',
    yaxis_title='Frequency',
    xaxis={'tickformat': '.0%'},
    yaxis={'tickformat': '.0%'})

#Relative measure distribution for base v AlteredC
rel_meas_dist3 =
ff.create_distplot([rel_measure3[rel_measure3]],
    ['relationship'],
    histnorm='probability',
    bin_size=0.1,
    show_rug=False,
    colors=[db])
rel_meas_dist3.update_layout(showlegend=False,
    paper_bgcolor='rgba(0,0,0,0)',
    font={'color': '#FFFFFF'},
    title="Change Score of AlteredC
font
relative to base font",
    xaxis_title='Change Score',
    yaxis_title='Frequency',
    xaxis={'tickformat': '.0%'},
    yaxis={'tickformat': '.0%'})

# Box plots are helpful
box_plots = px.box(guesses.sort_values(by='font'),
    x='font',
    y=measure,
    color='font',
    color_discrete_map={'Base': lg,
    'Altered': db,
    'symbol': gr})
box_plots.update_layout(showlegend=False,
    paper_bgcolor='rgba(0,0,0,0)',
    font={'color': '#FFFFFF'},
    title=f'{measure} of each font type')

# Change Score Metrics
change_score = px.scatter(rel_measure,
    'rel_measure',
    'Base',
    marginal_x='histogram',
    color_discrete_sequence=[db])
change_score.update_layout(showlegend=False,
    paper_bgcolor='rgba(0,0,0,0)',
    font={'color': '#FFFFFF'},
    title=f"Altered Font Change
Score Relative To Starting

```

```

Base font {measure}''',
    xaxis_title='Change Score',
    yaxis_title=f'Base Font
{measure}'),
    xaxis={'tickformat': '.0%'})
# Change Score Metrics (Base v Narrow)
change_score2 = px.scatter(rel_measure2,
    'rel_measure2',
    'Base',
    marginal_x='histogram',
    color_discrete_sequence=[db])
change_score2.update_layout(showlegend=False,
    paper_bgcolor='rgba(0,0,0,0)',
    font={'color': '#FFFFFF'},
    title=f'"Narrow Font Change
Score Relative To Starting
Base Font {measure}''',
    xaxis_title='Change Score',
    yaxis_title=f'Base Font
{measure}'),
    xaxis={'tickformat': '.0%'})
# Change Score Metrics (Base v AlteredC)
change_score3 = px.scatter(rel_measure3,
    'rel_measure3',
    'Base',
    marginal_x='histogram',
    color_discrete_sequence=[db])
change_score3.update_layout(showlegend=False,
    paper_bgcolor='rgba(0,0,0,0)',
    font={'color': '#FFFFFF'},
    title=f'"AlteredC Font Change
Score Relative To Starting
Base Font {measure}''',
    xaxis_title='Change Score',
    yaxis_title=f'Base Font
{measure}'),
    xaxis={'tickformat': '.0%'})
# Base Size vs Altered Size
linear_model = px.scatter(rel_measure,
    'Altered',
    'Base',
    marginal_x='histogram',
    marginal_y='histogram',
    color_discrete_sequence=[db],
    trendline='ols',
    trendline_color_override='red',
    trendline_options={'add_constant':
0})

linear_model.update_layout(showlegend=False,
    paper_bgcolor='rgba(0,0,0,0)',
    font={'color': '#FFFFFF'},
    title=f'"Altered font {measure}
compared to
Base font {measure}''',
    xaxis_title=f'Altered Font
{measure}',
    yaxis_title=f'Base Font
{measure}',
    xaxis_range=[0, 40],
    yaxis_range=[0,40])

results = px.get_trendline_results(linear_model)

model_results =
results.px_fit_results.iloc[0].params
model_results = np.round((model_results-1)*100,
1)[0]
# Base Size vs Narrow Size
linear_model2 = px.scatter(rel_measure2,
    'Narrow',
    'Base',
    marginal_x='histogram',
    marginal_y='histogram',
    color_discrete_sequence=[db],
    trendline='ols',
    trendline_color_override='red',
    trendline_options={'add_constant':
0})

linear_model2.update_layout(showlegend=False,
    paper_bgcolor='rgba(0,0,0,0)',
    font={'color': '#FFFFFF'},
    title=f'"Narrow Font {measure}
Compared To
Base Font {measure}''',
    xaxis_title=f'Narrow Font
{measure}',
    yaxis_title=f'Base Font
{measure}',
    xaxis_range=[0, 40],
    yaxis_range=[0,40])

results2 = px.get_trendline_results(linear_model2)

model_results2 =
results2.px_fit_results.iloc[0].params
model_results2 = np.round((model_results2-
1)*100, 1)[0]
# Base Size vs AlteredC Size
linear_model3 = px.scatter(rel_measure3,
    'AlteredC',
    'Base',
    marginal_x='histogram',
    marginal_y='histogram',
    color_discrete_sequence=[db],
    trendline='ols',
    trendline_color_override='red',
    trendline_options={'add_constant':
0})

```

```

linear_model3.update_layout(showlegend=False,
                             paper_bgcolor='rgba(0,0,0,0)',
                             font={'color': '#FFFFFF'},
                             title=f"AlteredC Font {measure}")
Compared To
Base Font {measure}''',
    xaxis_title=f'AlteredC Font
{measure}',
    yaxis_title=f'Base Font
{measure}',
    xaxis_range=[0, 40],
    yaxis_range=[0,40])

results3 = px.get_trendline_results(linear_model3)

model_results3 =
results3.px_fit_results.iloc[0].params
model_results3 = np.round((model_results3-
1)*100, 1)[0]

# Pandas Dataframe for dash table
summary_df = pd.DataFrame(
    {f'Base Font {measure}': [
        base[measure].count(),
        round(base[measure].mean(), 2),
        round(base[measure].median(), 2),
        round(base[measure].std(), 2)],
    f'Altered Font {measure}': [
        Altered[measure].count(),
        round(Altered[measure].mean(), 2),
        round(Altered[measure].median(), 2),
        round(Altered[measure].std(), 2)],
    f'Narrow Font {measure}': [
        narrow[measure].count(),
        round(narrow[measure].mean(), 2),
        round(narrow[measure].median(), 2),
        round(narrow[measure].std(), 2)],
    f'AlteredC Font {measure}': [
        AlteredC[measure].count(),
        round(AlteredC[measure].mean(), 2),
        round(AlteredC[measure].median(), 2),
        round(AlteredC[measure].std(), 2)]
    })
summary_df = summary_df.T.reset_index()
summary_df.rename(columns={'index': 'Font',
                           0: 'Observations',
                           1: f'Average {measure}',
                           2: f'Median {measure}',
                           3: 'Standard Deviation'},
                  inplace=True)
summary_df[f'Average
{measure}']=summary_df[f'Average
{measure}'].map('{:,.2f}'.format)

summary_tb = dash_table.DataTable(
    summary_df.to_dict('records'),

```

```

[{'name': i, 'id': i} for i in summary_df.columns],
style_header={'backgroundColor': 'rgb(39, 43,
48)',
              'color': 'white'},
style_data={'backgroundColor': 'rgb(39, 43, 48)',
            'color': 'white'})

# Relative score
rel_score =
str(round(rel_measure[rel_measure].mean()*100, 1))
+ '%'
rel_score2 =
str(round(rel_measure2[rel_measure2].mean()*100,
1)) + '%'
rel_score3 =
str(round(rel_measure3[rel_measure3].mean()*100,
1)) + '%'
score = dbc.Card(
    dbc.CardBody(
        [html.H4('Average Improvement of Altered
Font over Base Font'),
        html.H2(rel_score),
        html.H4('Average Improvement of Narrow
Font over Base Font'),
        html.H2(rel_score2),
        html.H4('Average Improvement of AlteredC
Font over Base Font'),
        html.H2(rel_score3),
        html.P(f"Note: metric is Base Font
{measure}/Altered Font
{measure}-1, so >0 is an improvement and <0 is a
setback"")]
    ))

# distribution test scores
p_score = dbc.Card(
    dbc.CardBody(
        [html.H4(f'Base v Altered: P-Value of the
{test_type}'),
        html.H2(f" {round(res[1],3)}"),
        html.H4(f'Base v Narrow: P-Value of the
{test_type}'),
        html.H2(f" {round(res2[1],3)}"),
        html.H4(f'Base v AlteredC: P-Value of the
{test_type}'),
        html.H2(f" {round(res3[1],3)}"),
        html.P("Note: The P-Value is basically
measuring what are the
odds that we would get these results by chance. A
lower p-value means it is
more likely that the change in font is the reason for
the results.
Typically, we look for a p-value below 0.05 to
suggest significance."))]
    ))

```

```

# distribution test scores
avg_improve = dbc.Card(
    dbc.CardBody(
        [html.H4(f'Base v Altered - Average
reduction in {measure}')],

html.H2(round(base[measure].mean() -
            Altered[measure].mean(),
            2)),
        html.P(f'"Calculation is Average
Base {measure} -
Average Altered {measure}. Positive means it
improved, negative means it did not"'
            )))
# distribution test scores Base v Narrow
avg_improve2 = dbc.Card(
    dbc.CardBody(
        [html.H4(f'Base v Narrow - Average
reduction in {measure}')],

html.H2(round(base[measure].mean() -
            narrow[measure].mean(),
            2)),
        html.P(f'"Calculation is Average
Base {measure} -
Average Narrow {measure}. Positive means it
improved, negative means it did not"'
            )))

# distribution test scores Base v AlteredC
avg_improve3 = dbc.Card(
    dbc.CardBody(
        [html.H4(f'Base v AlteredC - Average
reduction in {measure}')],

html.H2(round(base[measure].mean() -
            AlteredC[measure].mean(),
            2)),

```

```

        html.P(f'"Calculation is Average
Base {measure} -
Average AlteredC {measure}. Positive means it
improved, negative means it did not"'
            )))

```

```

return (summary_tb,
        score,
        p_score,
        avg_improve,
        avg_improve2,
        avg_improve3,
        dcc.Graph(figure=distributions),
        dcc.Graph(figure=distributions2),
        dcc.Graph(figure=distributions3),
        dcc.Graph(figure=incorrect_bars),
        dcc.Graph(figure=rel_meas_dist),
        dcc.Graph(figure=rel_meas_dist2),
        dcc.Graph(figure=rel_meas_dist3),
        dcc.Graph(figure=box_plots),
        dcc.Graph(figure=change_score),
        dcc.Graph(figure=change_score2),
        dcc.Graph(figure=change_score3),
        dcc.Graph(figure=linear_model),
        dcc.Graph(figure=linear_model2),
        dcc.Graph(figure=linear_model3),
        html.Div(f'The model suggests a
{model_results}% improvement'),
        html.Div(f'The model suggests a
{model_results2}% improvement'),
        html.Div(f'The model suggests a
{model_results3}% improvement'))

# %%
# This finally displays the final product. Helpful to
keep the debug on.
if __name__ == '__main__':
    dash_app.run_server(debug=True)

```

## **Appendix 4 – Instructions Displayed Before Test**

### **Test 1 (“Letters Test”)**

#### **PLEASE READ THE FOLLOWING INSTRUCTIONS:**

Thank you for volunteering to participate in this legibility study. Before we begin, please make sure that you are sitting normally in your chair without leaning forward or slouching. For the duration of the study, it is important to remain approximately the same distance from the screen. Please place your hand on your mouse.

Once the study begins, a colored rectangle will appear on the screen. A small character will appear in the center of the screen and increase in size as if you were driving towards it in your car. The character will eventually be large enough for you to see so please do not squint or strain your eyes. Please do not try and guess what the character is before it becomes legible.

Once the character becomes legible to you, click the green button above the sign with the mouse, or click any button on your keyboard. This will mark the distance at which you can see the character on the sign so it is important that you press it once you can clearly identify what the character is.

If you select a button on your keyboard, the test will stop and the distance at the time of the button press will be measured; however, you will still have to click the green button to advance to the next page.

When you click the green button, the sign will disappear completely, and you will be given the opportunity to indicate if the character was a letter or not.

If you said that the character was a letter, you will be given the opportunity to indicate what the letter was. Type in the provided text box what letter you saw. It is important to note that the case of the letter matters. When you are ready for the next character, select the button in the top left corner.

You have the option to opt out of the test at any point by selecting the button that says “Log out.” Note, the test records your answers in real time, in case you lose access to the test for any reason.

You will notice that some signs may repeat the same characters. That behavior is normal for the test, and not an indication of anything wrong with the program. The first three signs will be test images and will be the same across all tests. After the first three, the test will begin and the font, color of sign, and character displayed will be randomized. You will be shown a total of 40 signs. Thank you again for your participation.

## Test 2 (“Words Test”)

### **PLEASE READ THE FOLLOWING INSTRUCTIONS:**

Thank you for volunteering to participate in this legibility study. Before we begin, please make sure that you are sitting normally in your chair without leaning forward or slouching. For the duration of the study, it is important to remain approximately the same distance from the screen. Please place your hand on your mouse.

Once the study begins, a colored rectangle will appear on the screen. A small word will appear in the center of the screen and increase in size as if you were driving towards it in your car. The word will eventually be large enough for you to see so please do not squint or strain your eyes. Please do not try and guess what the word is before it becomes legible.

Once the word becomes legible to you, click the green button above the sign with the mouse, or click any button on your keyboard. This will mark the distance at which you can see the word on the sign so it is important that you press it once you can clearly identify what the word is.

If you select a button on your keyboard, the test will stop and the distance at the time of the button press will be measured; however, you will still have to click the green button to advance to the next page.

When you click the green button, the sign will disappear completely, and you will be given the opportunity to indicate if you are ready to type out the word.

You will be given the opportunity to indicate what the word was. Type in the provided text box what word you saw. It is important to note that the case of the letters matters. When you are ready for the next word, select the button in the top left corner.

You have the option to opt out of the test at any point by selecting the button that says “Log out.” Note, the test records your answers in real time, in case you lose access to the test for any reason.

You will notice that some signs may repeat the same words. That behavior is normal for the test, and not an indication of anything wrong with the program. The first three signs will be test images and will be the same across all tests. After the first three, the test will begin and the font, color of sign, and character displayed will be randomized. You will be shown a total of 46 signs. Thank you again for your participation.

## Appendix 5 – Data for Individual Letters

A

Table 13: Data for 'A'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	74	10.00	9.0	3.84
D-altered Font	39	8.84	8.5	2.95
Narrow Font	36	9.15	9.0	2.32

Base v D-altered

- Average reduction in size: 1.16
- P-value of the t-test: 0.051

Base v Narrow

- Average reduction in size: 0.85
- P-value of the t-test: 0.112

F

Table 14: Data for 'F'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	37	9.22	9.50	2.71
D-altered Font	38	9.51	9.42	2.66
Narrow Font	40	9.22	8.92	2.80

Base v D-altered

- Average reduction in size: -0.29
- P-value of the t-test: 0.681

Base v Narrow

- Average reduction in size: -0.01
- P-value of the t-test: 0.506

a

Table 15: Data for 'a'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	72	12.75	12.50	3.99
D-altered Font	38	10.82	10.67	3.28
Narrow Font	37	10.86	10.50	2.87

Base v D-altered

- Average reduction in size: 1.93
- P-value of the t-test: 0.006

Base v Narrow

- Average reduction in size: 1.89
- P-value of the t-test: 0.006

c

Table 16: Data for 'c'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	36	11.21	10.75	2.99
D-altered Font	37	11.46	11.50	3.89
Narrow Font	36	10.75	10.58	3.02

Base v D-altered

- Average reduction in size: -0.25
- P-value of the t-test: 0.621

Base v Narrow

- Average reduction in size: 0.46
- P-value of the t-test: 0.260

e

Table 17: Data for 'e'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	38	12.05	12.25	3.49
D-altered Font	37	12.73	12.67	4.08
Narrow Font	39	11.54	10.67	3.65

Base v D-altered

- Average reduction in size: -0.67
- P-value of the t-test: 0.777

Base v Narrow

- Average reduction in size: 0.51
- P-value of the t-test: 0.265

f

Table 18: Data for 'f'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	37	11.36	11.17	2.91
D-altered Font	36	12.88	12.17	4.28
Narrow Font	36	13.07	12.50	3.62

Base v D-altered

- Average reduction in size: -1.52
- P-value of the t-test: 0.960

Base v Narrow

- Average reduction in size: -1.71
- P-value of the t-test: 0.985

g

Table 19: Data for 'g'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	40	11.22	10.42	3.46
D-altered Font	40	11.17	10.83	3.32
Narrow Font	36	10.63	9.75	3.43

Base v D-altered

- Average reduction in size: 0.05
- P-value of the t-test: 0.476

Base v Narrow

- Average reduction in size: 0.58
- P-value of the t-test: 0.232

k

Table 20: Data for 'k'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	37	10.15	10.00	2.80
D-altered Font	38	10.42	10.42	2.73
Narrow Font	39	10.09	9.17	3.25

Base v D-altered

- Average reduction in size: -0.27
- P-value of the t-test: 0.662

Base v Narrow

- Average reduction in size: 0.07
- P-value of the t-test: 0.461

s

Table 21: Data for 's'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	38	11.67	12.08	3.04
D-altered Font	38	11.76	11.42	3.50
Narrow Font	36	11.38	11.50	3.74

Base v D-altered

- Average reduction in size: -0.1
- P-value of the t-test: 0.551

Base v Narrow

- Average reduction in size: 0.28
- P-value of the t-test: 0.361

t

Table 22: Data for 't'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	39	9.85	9.17	2.75
D-altered Font	51	10.32	9.33	2.83
Narrow Font	40	11.35	11.00	3.69

Base v D-altered

- Average reduction in size: -0.47
- P-value of the t-test: 0.785

Base v Narrow

- Average reduction in size: -1.5
- P-value of the t-test: 0.978

W

Table 23: Data for 'w'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	37	10.41	10.67	2.68
D-altered Font	42	11.08	11.08	2.92
Narrow Font	43	10.61	10.00	3.30

Base v D-altered

- Average reduction in size: -0.67
- P-value of the t-test: 0.854

Base v Narrow

- Average reduction in size: -0.2
- P-value of the t-test: 0.615

X

Table 24: Data for 'x'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	38	10.90	10.17	3.60
D-altered Font	39	10.78	10.50	3.18
Narrow Font	36	11.13	10.42	3.36

Base v D-altered

- Average reduction in size: 0.12
- P-value of the t-test: 0.440

Base v Narrow

- Average reduction in size: -0.23
- P-value of the t-test: 0.612

Z

Table 25: Data for 'z'

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	38	10.21	10.25	2.45
D-altered Font	39	10.91	10.50	3.27
Narrow Font	39	10.25	10.17	2.95

Base v D-altered

- Average reduction in size: -0.7t
- P-value of the t-test: 0.854

Base v Narrow

- Average reduction in size: -0.05
- P-value of the t-test: 0.530

## Appendix 6 – Data for Individual Words

### Barley

Table 26: Data for Barley

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	77	10.06	9.67	2.95
D-altered Font	39	11.30	10.33	3.61
Narrow Font	38	9.92	9.08	2.60
C-altered Font	37	12.71	11.67	3.95

#### Base v D-altered

- Average reduction in size: -1.24
- P-value of the t-test: 0.975

#### Base v Narrow

- Average reduction in size: 0.14
- P-value of the t-test: 0.405

#### Base v C-altered

- Average reduction in size: -2.66
- P-value of the t-test: 1.0

### Bartey

Table 27: Data for Bartey

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	39	11.48	11.00	3.76
D-altered Font	36	11.77	11.67	3.22
Narrow Font	39	11.08	10.50	3.27
C-altered Font	39	13.99	13.17	3.97

#### Base v D-altered

- Average reduction in size: -0.29
- P-value of the t-test: 0.639

#### Base v Narrow

- Average reduction in size: 0.41
- P-value of the t-test: 0.306

#### Base v C-altered

- Average reduction in size: -2.51
- P-value of the t-test: 0.997

## Eatery

Table 28: Data for Eatery

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	35	9.98	10.00	2.50
D-altered Font	38	10.71	9.83	3.24
Narrow Font	39	9.77	9.17	3.11
C-altered Font	37	11.45	10.00	3.45

### Base v D-altered

- Average reduction in size: -0.73
- P-value of the t-test: 0.857

### Base v Narrow

- Average reduction in size: 0.21
- P-value of the t-test: 0.375

### Base v C-altered

- Average reduction in size: -1.47
- P-value of the t-test: 0.979

## Felony

Table 29: Data for Felony

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	35	10.02	9.50	2.90
D-altered Font	39	9.97	9.33	2.74
Narrow Font	38	9.74	9.08	2.82
C-altered Font	38	11.29	11.00	2.77

### Base v D-altered

- Average reduction in size: 0.05
- P-value of the t-test: 0.467

### Base v Narrow

- Average reduction in size: 0.28
- P-value of the t-test: 0.337

### Base v C-altered

- Average reduction in size: -1.27
- P-value of the t-test: 0.970

## Honors

Table 30: Data for Honors

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	43	9.00	8.00	2.17
D-altered Font	76	10.00	9.83	2.01
Narrow Font	37	9.34	9.33	1.79
C-altered Font	41	10.85	10.50	2.53

### Base v D-altered

- Average reduction in size: -1.00
- P-value of the t-test: 0.994

### Base v Narrow

- Average reduction in size: -0.34
- P-value of the t-test: 0.773

### Base v C-altered

- Average reduction in size: -1.85
- P-value of the t-test: 1.0

## Houses

Table 31: Data for Houses

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	37	9.58	9.50	3.15
D-altered Font	38	9.88	9.17	3.08
Narrow Font	38	9.83	9.33	2.59
C-altered Font	36	11.18	11.00	2.80

### Base v D-altered

- Average reduction in size: -0.31
- P-value of the t-test: 0.663

### Base v Narrow

- Average reduction in size: -0.26
- P-value of the t-test: 0.650

### Base v C-altered

- Average reduction in size: -1.60
- P-value of the t-test: 0.988

## Season

Table 32: Data for Season

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	73	9.68	9.33	2.37
D-altered Font	37	9.87	9.83	2.53
Narrow Font	37	10.25	9.67	2.66
C-altered Font	40	11.59	11.08	3.23

### Base v D-altered

- Average reduction in size: -0.19
- P-value of the t-test: 0.652

### Base v Narrow

- Average reduction in size: -0.57
- P-value of the t-test: 0.870

### Base v C-altered

- Average reduction in size: -1.91
- P-value of the t-test: 1.0

## Senior

Table 33: Data for Senior

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	35	9.50	8.67	3.04
D-altered Font	35	10.34	10.00	2.33
Narrow Font	35	9.80	8.67	3.01
C-altered Font	40	10.46	10.00	2.28

### Base v D-altered

- Average reduction in size: -0.84
- P-value of the t-test: 0.901

### Base v Narrow

- Average reduction in size: -0.3
- P-value of the t-test: 0.663

### Base v C-altered

- Average reduction in size: -0.96
- P-value of the t-test: 0.939

## Sensor

Table 34: Data for Sensor

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	37	10.00	9.67	2.42
D-altered Font	40	10.34	10.00	2.78
Narrow Font	36	10.33	9.83	2.54
C-altered Font	37	10.69	10.17	2.56

### Base v D-altered

- Average reduction in size: -0.34
- P-value of the t-test: 0.713

### Base v Narrow

- Average reduction in size: -0.33
- P-value of the t-test: 0.714

### Base v C-altered

- Average reduction in size: -0.69
- P-value of the t-test: 0.883

## Doting

Table 35: Data for Doting

Font	Observations	Average Size	Median Size	Standard Deviation
Base Font	40	10.54	9.92	3.06
D-altered Font	40	11.93	11.50	3.88
Narrow Font	43	10.73	10.00	3.33
C-altered Font	38	12.20	11.75	3.78

### Base v D-altered

- Average reduction in size: -1.39
- P-value of the t-test: 0.961

### Base v Narrow

- Average reduction in size: -0.19
- P-value of the t-test: 0.607

### Base v C-altered

- Average reduction in size: -1.66
- P-value of the t-test: 0.982