Statistical Satire: A Data Analytics Approach to Stand-Up Comedy

Chris Heffernan

Sunday 10th December, 2023

Contents

Abstract and Introduction

With the modern intersection of stand-up comedy and digital mediums, stand-up comedy is undergoing its own renaissance. Initially experiencing stand-up comedy feels like witnessing magic unfold on stage. However, delving behind the curtain reveals an appreciation for the artistry involved, showing how comedians skillfully construct sets – molding words and timing to craft moments that elicit genuine laughter from their audience. This project leverages analytical tools and concepts to unravel the underlying patterns and trends within comedic content.

Motivated by the desire to understand the dynamics of humor and audience engagement, the report poses questions about prevalent humor styles, linguistic nuances, and the impact of audience reactions. Leveraging sentiment analysis, word frequency analysis, and clustering techniques, the analysis unveils distinct humor clusters, identifies frequently used comedic elements, and explores the interplay between language and laughter.

Results reinforce the complexity of analyzing stand-up comedy by its words. Sentiment analysis exposes the predominantly positive nature of comedic language, while word frequency analysis highlights the significance of specific words in crafting relatable and resonant humor. Clustering techniques unveil the nuanced topics discussed in stand-up specials, emphasizing the individuality of each comedian's style.

This project not only pays tribute to the field of data analytics but also provides insights for working comedians. The findings showcase the potential of data-driven approaches in decoding the intricate art of stand-up comedy and in doing so, provide a level of understanding that makes the seemingly intimidating nature of the craft into an accessible endeavor.

Data Description and Exploratory Data Analytics

While non-verbal cues offer great value in eliciting laughter, the essence of humor lies fundamentally in a comedian's choice of words. A stand-up special serves as a compilation of a comedian's most successful jokes within a specified time frame, so the dataset is exclusively centered around the transcript of stand-up specials to conduct essential statistical analyses. Renowned comedians traditionally release their specials on platforms like Netflix, HBO, and Comedy Central, attracted by the substantial payout. However, a notable shift has occurred, where comedians are increasingly opting for YouTube as a platform for special releases, providing cost-free access and in turn reaching a broader audience. Exploiting YouTube's open-source API capabilities, this project used closed-captioning functionalities to extract content from various comedy sets. This extraction facilitated the content and timing analyses.

The overall dataset was comprised of 50 specials featuring 50 different comedians and nearly 50 hours of content. Each individual special had its own csv file containing text information, the time with which it took to deliver those words, and its timestamp in the video. These columns were labeled *text*, *start*, and *duration* as directly sourced from the transcript API. The text column not only featured the comic's words, but also behaviors from the special including applause, music, and most importantly laughter. This facilitated manipulating individual specials and creating dataframes in R.

Initial EDA was performed on individual comedy sets. This process involved an in-depth examination of each set's unique characteristics, including audience behavior, sentiment analysis, word frequency patterns, and cadence. Building on the insights gained from individual set analyses, a comprehensive EDA was conducted on all comedy sets combined. This allowed for the aggregation of patterns and trends across the entire dataset. Preprocessing the text consisted of removing any unrecognizable characters and handling YouTube's approach to cuss words. YouTube transcripts don't provide text for certain profanities and instead replace the word with [__] in its closed captions. Consequently, this symbol was separated from the text, and allowed for some interesting analysis. Some datasets also did not feature behaviors like laughter, applause, and music, so this was taken into account when developing some of the graphics and statistics. Another area for preprocessing was that the captions sometimes confused laughter with music. Given the unlikely event for a stand-up special to feature music in the set anywhere besides the beginning and end, music detected outside of the beginning and end was assumed to be laughter.

Analysis

Sentiment analysis was conducted in the EDA to understand the tone and mood expressed in the text. The sentiment analysis was conducted after removing explicits. This was because there was no way to automate replacing the profanity symbol with the actual word. However, stand-up often undergoes controversy as some comics can be characterized as vulgar. In removing the explicits and focusing on all other words, the sampled comedy sets proved to be much showcase little negative content. The SentimentAnalysis library in R served as the primary tool for sentiment analysis. This library provides functionalities to assess sentiment in textual data and is particularly well-suited for applications in the field of natural language processing.

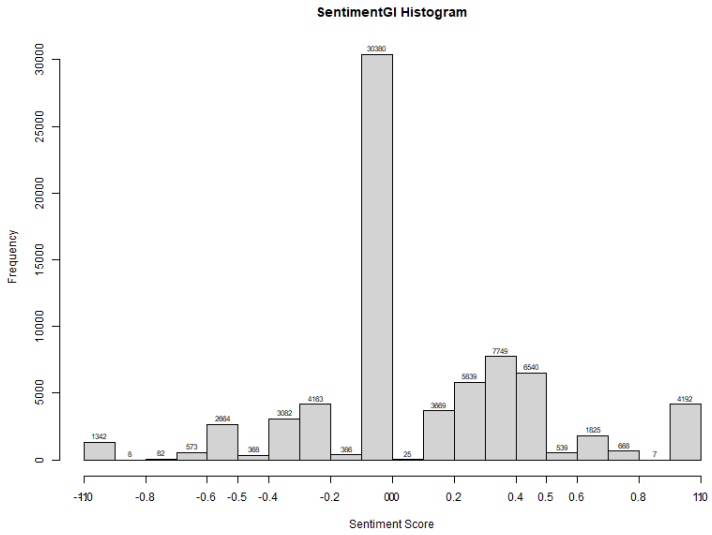
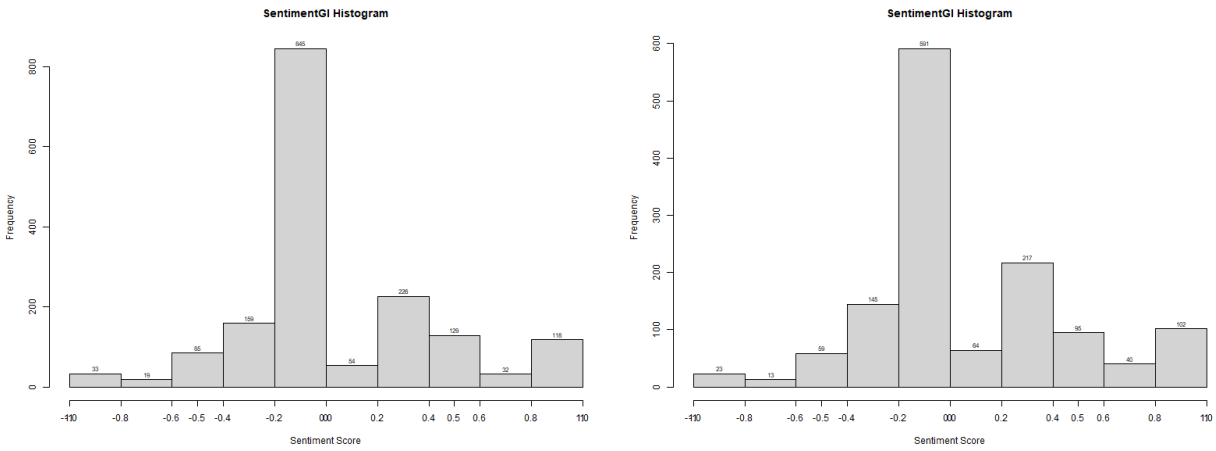


Figure 1: Histogram of Sentiment Analysis

The language seems to be remarkably neutral, but if one were to polarize between positive and negative, the histogram showcases that the majority of words chosen across all specials are positive. The sentiment analysis in this context is more concerned with capturing the humorous and positive aspects rather than negative sentiments. Profanities are often associated with negative emotions, but in the realm of comedy, they can be used playfully or satirically to enhance the humor without necessarily conveying a negative sentiment. The special that featured the most profanity was Andrew Schulz's special titled "Infamous" with 169 documented cases of profanity. Ali Siddiq's special "The Domino Effect" showed zero cases of profanity, and their histograms prove how similar their sentiments are.



(a) The Domino Effect Sentiment Analysis (b) Infamous Sentiment Analysis

Figure 2: Comparitive Sentiment Analysis

Next, analysis was conducted to determine which words were most prevalent in a stand-up comedy special. Identifying the most common words helps to gain a quick overview of the main themes or topics present in stand-up specials. These words are likely to be key elements in understanding a comedian's style, recurring themes, or the overall tone of the performance.

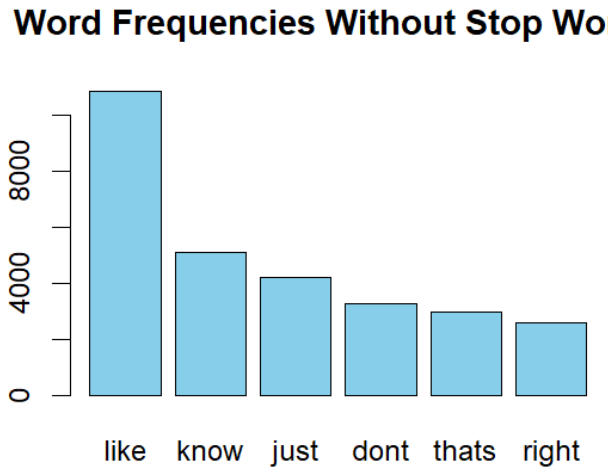


Figure 3: Word Frequency

The significance of "like" being the most common word in stand-up specials could be attributed to a couple of different factors. Stand-up comedians often use comparisons and similes as a humorous device. The word "like" is commonly used to introduce such comparisons, making it a frequent occurrence. In spoken language, especially in casual and

conversational settings like stand-up comedy, people tend to use filler words or phrases. "Like" is one such filler word that is often used to provide emphasis or convey a sense of approximation. If "like" is a frequently used word, it suggests that the audience may respond positively to comparisons and relatable content, adding a layer to why comics might choose to incorporate this in their act.

Next, the EDA considered a comic's timing and how cadence impacts the nature of stand-up. Timing and cadence play a pivotal role in stand-up comedy, as the way a comedian paces their speech, the emphasis on certain words, or the use of pauses can significantly contribute to the overall comedic effect. The transcript data was initially cleaned to ensure uniformity and eliminate potential discrepancies. Speech rate, measured in words per second, was calculated for each segment of the stand-up specials. This metric serves as an indicator of how quickly or slowly a comedian delivers their lines. The cubic spline interpolation technique was applied to individual stand-up specials. Pacing throughout the sets was sporadic and almost sinusoidal, so linear interpolation was not effective in capturing cadence, but the locally estimated scatterplot smoothing (LOESS) method was employed to also offer a simplified visualization of speech rate changes over time. Cubic spline interpolation helped in smoothing out the speech rate fluctuations, providing a clearer view of the underlying trends. The individual spline points from all stand-up specials were then aggregated into one comprehensive dataset. Here is a sample of the cadence featured in Jim Bruer's "Country Boy Will Survive":

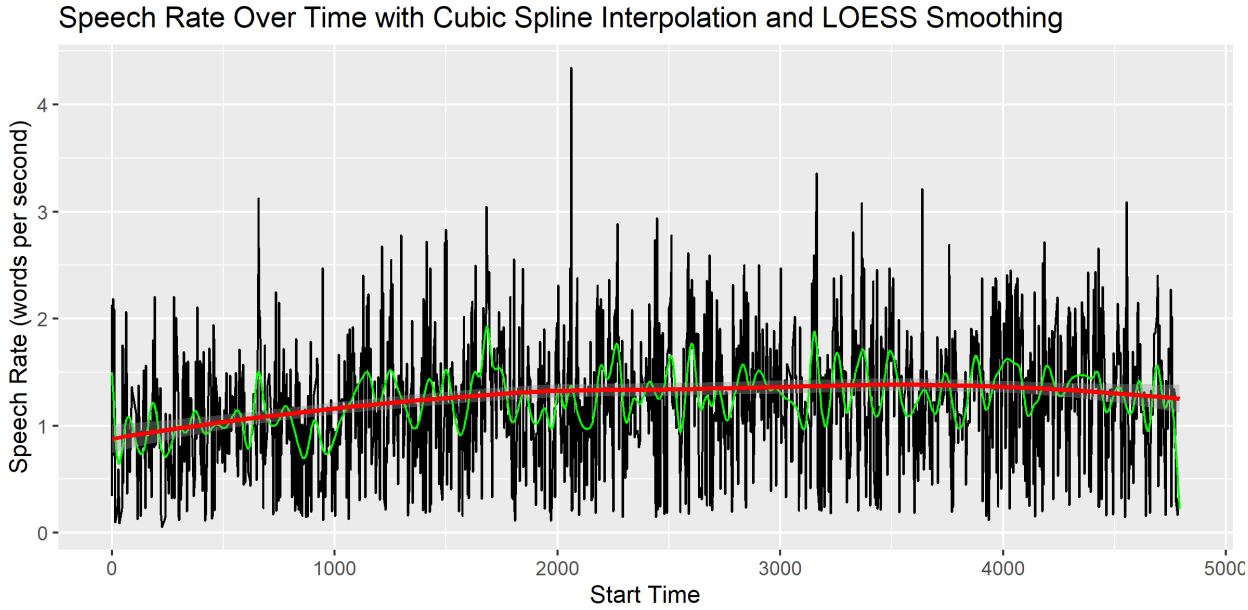


Figure 4: "Country Boy WIll Survive" Cadence

The aggregate cadence interpolation shows how comics tend to oscillate around 2 words per second, but that rate accounts for the audience's laughing and applause. The end of the special intuitively shows how speech rate decreases, but it appears that speech rate increases around 75% of the way through the special.

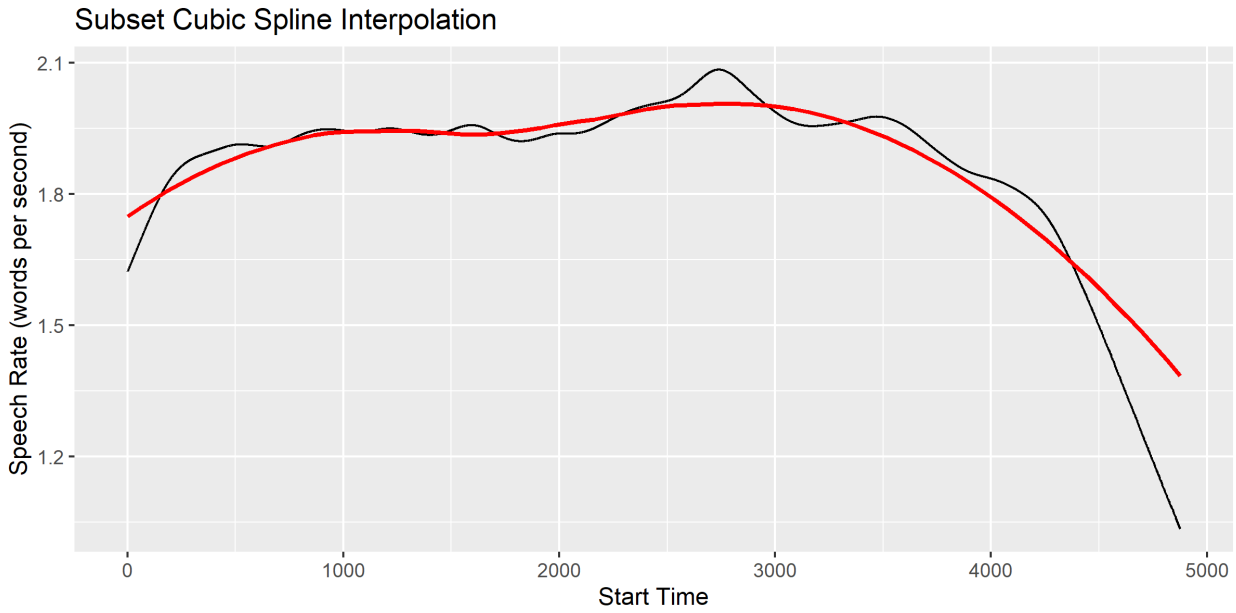


Figure 5: Aggregate Cadence Plot

The final component of the EDA was dedicated to determining the topics discussed in stand-up sets. In order to do so, the text information was input to a topic modeling method known as Latent Dirichlet Allocation (LDA). LDA is a type of statistical model used for discovering abstract topics within a collection of documents. The LDA model assumes that each document is a mixture of a small number of topics and that each word in the document is attributable to one of the document's topics, which seemed perfect for the various stand-up sets.

Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	Topic 9	Topic 10
"right"	"just"	"like"	"like"	"thats"	" "	"know"	"applause"	"people"	"get"
"got"	"make"	"youre"	"hes"	"one"	"going"	"dont"	"never"	"think"	"yeah"
"now"	"around"	"gonna"	"well"	"time"	"man"	"want"	"ive"	"theyre"	"can"
"back"	"cant"	"guys"	"guy"	"good"	"audience"	"didnt"	"music"	"theres"	"see"
"little"	"trying"	"much"	"shes"	"thing"	"said"	"say"	"ever"	"love"	"ill"
"come"	"still"	"way"	"look"	"first"	"laughing"	"even"	"two"	"really"	"take"
"show"	"saying"	"god"	"feel"	"day"	"dude"	"mean"	"years"	"lot"	"tell"
"okay"	"put"	"thank"	"kind"	"every"	"something"	"need"	"life"	"always"	"okay"
"new"	"give"	"getting"	"also"	"went"	"hey"	"talk"	"thought"	"white"	"laugh"
"talking"	"away"	"coming"	"pretty"	"whole"	"whats"	"doesnt"	"old"	"women"	"let"

Surprisingly, the LDA did not reveal any distinct and easily interpretable topics. The lack of coherent topics indicates that each of the specials makes a unique imprint. Analysis on individual specials could provide more valuable insights into topics discussed, but that would not contribute to the current mission to understand the comedy landscape as a whole.

Model Development and Application of Model(s)

The 2 models constructed for this project are used to get a more robust understanding of the topics discussed in a comedy set and the cadence. Initial LDA anaysis for the all-encompassing dataset proved to be ineffective, but through clustering techniques, it's possible to how topics are interconnected in individual sets and the stand-up landscape as a whole. The model built to understand cadence utilizes regression to correlate the duration of laughter and the sentiment of the words leading up to the laughter. With this we could potentially identify if people tend to laugh harder at raunchy humor or not.

For clustering topics using Latent Dirichlet Allocation results, hierarchical clustering is the most suitable method. It creates a hierarchy of clusters, providing a visual representation of topic relationships. The code takes a list of LDA models where one document represents one comedy set, and performs hierarchical clustering on the topics using the beta matrix of each LDA model. The beta matrix represents the probability distribution of terms in each topic. Each row in the beta matrix corresponds to a topic, and each column corresponds to a unique term in the vocabulary. The values in the matrix represent the probability of a term occurring in a particular topic. The beta matrix is particularly useful for clustering topics because it captures the underlying structure of topics in terms of their word distributions. By using the beta matrix for hierarchical clustering, the code is essentially comparing topics based on how similar their word distributions are. This can help identify topics that share common terms or have similar themes. The clustering is executed using Ward's method (method = "ward.D2") based on the Euclidean distance between topics. Ward's method is known for minimizing the total within-cluster variance during the merging of clusters. In the context of stand-up comedy transcripts, where topics may represent common themes or recurring jokes, minimizing within-cluster variance is desirable.

The distance matrix quantifies the level of similarity between the various topics in the dataset. When initially building the model, distance matrix was calculated using the default Euclidean distance which measures a straight line between to topics in the multidimensional space. The dendrogram resulting from hierarchical clustering shows the hierarchy of topic similarities. The branches of the dendrogram represent the clusters formed at different levels of similarity. Similar topics are joined together at lower levels of the dendrogram, while more dissimilar topics are joined at higher levels. Featured here is the dendrogram produced from Anjelah Johnson-Reyes' special "Say I Won't":

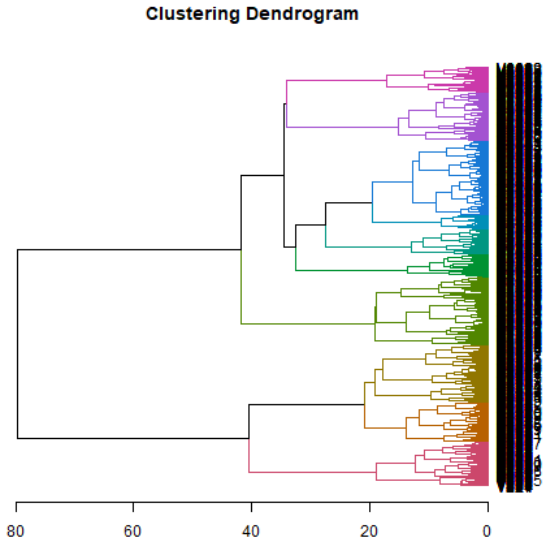


Figure 6: "Say I Won't" Dendrogram

There are 10 distinct colors specifying the partition of the set into 10 clusters based on the results. There are so many words to choose from and the variability in topics can be so diverse. Comics like Mark Normand deliver one-line jokes where each joke is completely unrelated to the next, but a comic like Mike Birbiglia is notable for building his set around an overarching theme. The dataset included a transcript from Mark Normand's special "Don't Be Yourself", which showed interesting results when compared to Anjelah Johnson's special.

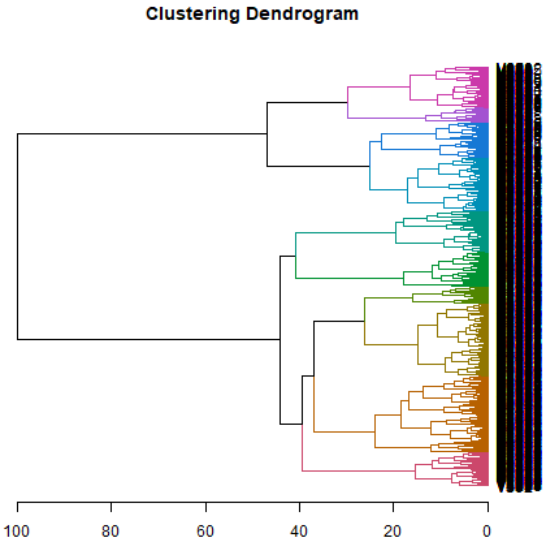


Figure 7: "Don't Be Yourself" Dendrogram

	Say I Won't	Don't Be Yourself
merge	2642	3236
height	1321	1618
order	1322	1619
labels	1322	1619

Using the summary function, the observed difference in the number of merges between 'Say I Won't' and 'Don't Be Yourself' suggests intriguing distinctions in the structure of these comedy sets. The higher number of merges in 'Don't Be Yourself,' particularly in comparison to 'Say I Won't,' implies a more intricate hierarchy of topics. This metric was discovered unintentionally, derived from the hierarchical clustering process, effectively quantifying the range and diversity of topics discussed in a stand-up set. In essence, a higher number of merges reflects a broader spectrum of themes and subjects covered within the same time duration. This novel metric serves as a valuable quantitative measure for assessing the richness and variety of content in stand-up comedy specials, offering a nuanced perspective beyond traditional topic modeling techniques. As comedy specials typically share a similar time frame, this metric becomes especially pertinent in gauging the efficiency with which comedians navigate through diverse topics, providing a unique lens through which to evaluate the distinct comedic styles of performers like Mark Normand and Anjelah Johnson-Reyes.

When generating the cluster for all combined sets, each bta matrix was stacked into 1 matrix and transposed which showcased the total different types of topics discussed based on the similarity in word choices. The intention was to determine if there were some overarching theme similarities across all the data. Finding a massive similarity would expose that comedians tend to drift to certain topics when producing specials as they could be classified as the funniest. As it turns out, and an homage to the art, there are extremely low similarities in the topics discussed in comedy specials. This indicates that comedians don't rely on the subject matter in order to produce humor. This information does not make it clear as to what a comic needs to say in order to be funny, but in fact eliminates any misconception that what a comedian talks about needs to be inherently funny.

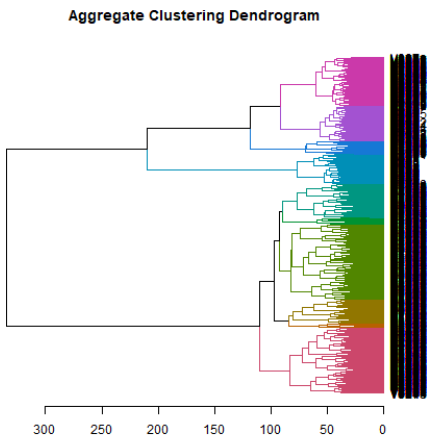


Figure 8: Aggregated Specials Dendrogram

The next model utilizes regression to create different plots showcasing the relationship between the sentiment of the vernacular leading up to a laugh and the duration of laughter. This relationship will in theory demonstrate if the sentiment of language can dictate how hard the audience will laugh. This regression will also showcase the sentiment of the text leading up to applause. Comedians have expressed their disdain for applause during their sets as they believe the audience is compensating for their lack of laughter. Laughter can be characterized as an involuntary and uncontrollable reaction to a given situation, whereas applause tends to be a conscious decision. Of course, it is not appropriate to generalize applause as a negative, but it begs the question, do certain reactions from the audience stem from the sentiment of the jokes being told?

The stereotypical structure of a joke is set-up and punchline; however, the structure of jokes vary wildly for different comedians. Comedians may even employ various styles within their own sets. This paper unfortunately does not suggest any groundbreaking techniques that can digitize a special into jokes. Instead, the regression will extract the text before the audience reaction and split it in half denoting the set-up and punchline respectively.

The regression analysis revealed intriguing insights. For laughter, the model indicates a minimal effect of LaughTime on the Average Sentiment, with a non-significant coefficient (t value = -0.144, p-value = 0.886). The overall model's explanatory power is low, as indicated by the low Multiple R-squared (5.918e-05) and Adjusted R-squared (-0.002814). This suggests that the sentiment of the language leading up to a laugh might not strongly predict the intensity or duration of laughter.

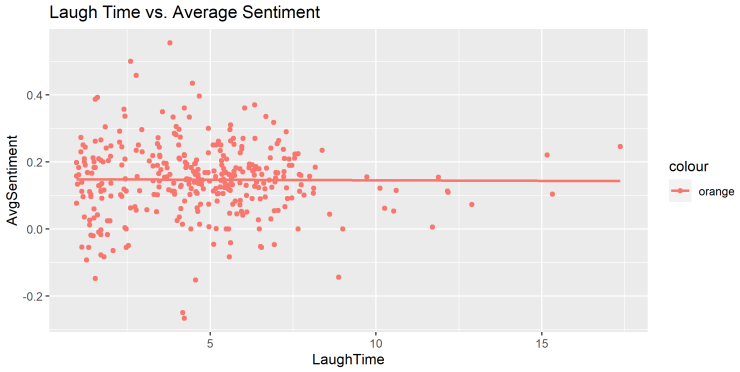


Figure 9: Aggregate Laughter Regression

Similarly, the regression for applause showed that the ApplauseTime has a non-significant effect on the Average Sentiment (coefficient = 0.002023, t value = 1.238, p-value = 0.216). The Multiple R-squared (0.001569) and Adjusted R-squared (0.0005447) also indicate a low overall explanatory power of the model. These findings suggest that the sentiment of the text before applause might not significantly influence the audience's conscious decision to applaud.

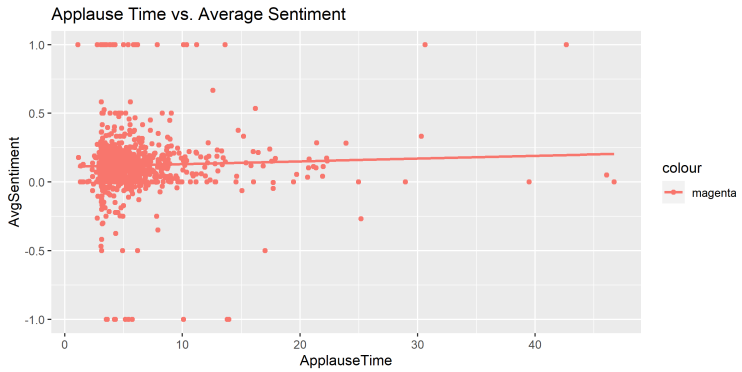


Figure 10: Aggregate Applause Regression

It's important to acknowledge the complexity of comedy dynamics since laughter is often considered an involuntary response, while applause is more deliberate. The stereotypical joke structure of set-up and punchline introduces another layer of nuance, although the regression approach simplifies this by extracting and analyzing the text before audience reactions. The outliers in the applause regression suggest a line different from laughter, but the data points show extremely similar values. This makes it complicated to distinguish between what types of jokes would lead to laughter as opposed to applause. This adds another layer of complexity to the art form and makes the art form that much more unique.

Conclusions and Discussion

This project has shed light on various aspects of the comedic landscape. Through sentiment analysis, word frequency analysis, clustering techniques, and regression models, the report explored the dynamics of humor, linguistic patterns, and audience reactions across a diverse range of stand-up specials. Sentiment analysis revealed the predominantly positive nature of comedic language, emphasizing the role of humor in fostering a positive and enjoyable atmosphere. Word frequency analysis unveiled the significance of certain words, such as "like," providing insights into the stylistic choices and relatability that resonate with audiences. Hierarchical clustering and Latent Dirichlet Allocation (LDA) brought forth a nuanced understanding of the diverse topics covered by comedians. The lack of substantial similarity across topics emphasized the individuality and uniqueness of each comedian's comedic style, challenging preconceptions about the necessity of specific subjects for humor. The analysis of laughter and applause dynamics, while demonstrating limited predictive power, showcased the complexity of audience responses and the multifaceted nature of comedic delivery. Aspiring comedians can take solace in the data-driven revelation that there is no predetermined path to success in stand-up; instead, the diverse landscape of comedic expression welcomes experimentation and authenticity of any kind.

This project offered an interesting opportunity to explore an avenue that seems to be untouched by professionals. The analytical and language processing skills necessary to fully realize quantifiable models for stand-up are far above my ability. When you watch a lot of stand-up you start to recognize patterns, so I thought if I could as a human then surely I could train a computer to do so. Next time I would do my best to look more into language processing since that's what the paper was mostly concerned about. I was hoping to use OpenAI's ChatGPT to extract topics, and that seems like it would be a valuable tool for future work. This project was a motivating and exciting way to apply what we learned in class to a core passion of mine. In the time between graduation and finding a job, I plan to continue working on finding reliable models that can characterize stand-up.

Bibliography

[1] Google for Developers. "Channels: list — YouTube Data API." Available: https://developers-dot-devsite-v2-prod.appspot.com/youtube/v3/docs/channels/list. [Accessed: Dec. 10, 2023]

[2] Udacity Team. "Natural Language Processing With R." Udacity, Oct. 08, 2020. Available: https://www.udacity.com/blog/2020/10/natural-language-processing-with-r.html

[3] X. Wang, Y. Ming, T. Wu, H. Zeng, Y. Wang, and H. Qu. "DeHumor: Visual Analytics for Decomposing Humor." Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9488285

[4] R. Kulshrestha. "Latent Dirichlet Allocation(LDA)." Medium, Jul. 03, 2020. Available: https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2

[5] sicss.io. "Basic Text Analysis in R." Available: https://sicss.io/2020/materials/day3-text-analysis/basic-text-analysis/rmarkdown/Basic_Text_Analysis_in_R.html. [Accessed: Dec. 10, 2023]

[6] J. Depoix. "youtube-transcript-api: This is a Python API which allows you to get the transcripts/subtitles for a given YouTube video. It also works for automatically generated subtitles, supports translating subtitles and it does not require a headless browser, like other Selenium based solutions do!" PyPI. Available: https://pypi.org/project/youtube-transcript-api/

[7] T. Munasinghe. "Labs: Trees, Hierarchical Clustering, Heatmaps." Rensselaer Polytechnic Institute, Oct. 2023.

Code Appendix

Python Code: Transcript API

```python
from youtube_transcript_api import YouTubeTranscriptApi
import csv

# READS THROUGH THE LIST OF COMICS, GETS THEIR YOUTUBE SPECIAL INFO
def getSpecials(id_list):
    # Define the path to your CSV file
    csv_file_path = 'ComicsList.csv'

    # Specify the target column name (e.g., 'SPECIAL')
    target_column_name = 'SPECIAL'

    # Open the CSV file for reading
    with open(csv_file_path, 'r', newline='') as csv_file:
        # Create a CSV reader
        csv_reader = csv.DictReader(csv_file)

        # Iterate through the rows in the CSV file
        for row in csv_reader:
            # Get the content of the 'SPECIAL' column
            specials_cell_content = row[target_column_name]
            # Check if the 'SPECIAL' cell is not empty
            if specials_cell_content:
                # Parse the pseudo-JSON content within the 'SPECIAL' cell
                # Extract the 'id' value using string manipulation
                id_start = specials_cell_content.find('id:')  # Find the position of 'id:'
                if id_start != -1:
                    id_start += len('id:')  # Move the starting position after 'id:'
                    id_end = specials_cell_content.find('}', id_start)  # Find the end of the 'id' value
                    if id_end != -1:
                        id_value = specials_cell_content[id_start:id_end]
                        id_list.append(id_value)

# PROCESS AND SAVE THE TRANSCRIPT TO A CSV WHERE THE FILENAME IS THE VIDEO ID
def saveTranscriptToCSV(id):
    # Retrieve the available transcripts
    transcript_list = YouTubeTranscriptApi.list_transcripts(id)

    # Create a list to store the transcript data
    transcript_data = []

    # Iterate over all available transcripts
    for transcript in transcript_list:
        # Fetch the transcript data
        transcript_text = transcript.fetch()

        # Append each entry to the transcript_data list
        transcript_data.extend(transcript_text)

    # Define the CSV file name
    csv_file_name = id+'.csv'

    # Open the CSV file in write mode
    with open(csv_file_name, 'w', newline='', encoding='utf-8') as csv_file:
        # Create a CSV writer
        csv_writer = csv.writer(csv_file)

        # Write the header row
        csv_writer.writerow(['text', 'start', 'duration'])

        # Write data from the transcript data to the CSV
        for entry in transcript_data:
            csv_writer.writerow([entry['text'], entry['start'], entry['duration']])

    print(f'CSV data has been saved to {csv_file_name}')

id_list = []
getSpecials(id_list)
for id in id_list:
    saveTranscriptToCSV(id)
```

R Code: Text EDA

```r
# Load the ggplot2 library
library(ggplot2)
library(SentimentAnalysis)
library(tm)
library(NLP)
library(gutenbergr)
library(dplyr)
library(quanteda)
library(readr)
library(wordcloud)

```

```r
fileNames <- list()
allData <- list()
wordDict <- c()
cuss <- "\\[ __ \\]"
soundEffectsAll <- c("[applause]"=0, "[music]"=0, "[laughter]"=0, "[cuss]"=0)

folder <- "C:/Users/heffec/Desktop/Semester 8/Data Analytics/FinalProject"
# Get a list of CSV files in the directory
csv_files <- list.files(
  path = "C:/Users/heffec/Desktop/Semester 8/Data Analytics/FinalProject",
  pattern = "\\.csv$",
  full.names = TRUE,
)
csv_files <- setdiff(csv_files, "C:/Users/heffec/Desktop/Semester 8/Data Analytics/FinalProject/ComicsList.csv")
# ----- Data Preprocessing ----- #

# Iterate over each CSV file
for (file_path in csv_files) {
  print(file_path)

  soundEffects <- c("[applause]"=0, "[music]"=0, "[laughter]"=0, "[cuss]"=0)


  # Read and clean CSV text
  #print(file_path)
  data <- read.csv(file_path)
  data$text <- tolower(data$text)
  data <- na.omit(data, cols = "text")

  # Clean up file names to extract the specific dataset name
  split_fileName <- strsplit(file_path, '/')
  elements <- unlist(split_fileName)
  last = length(elements)
  modified_string <- substr(elements[last], 1, nchar(elements[last]) - 4)


  # --- Audience Behavior ---
  # This will record laughter, applause, and music from the set
  # Saved on an individual basis and in aggregate
  explicitUTFCode <- c(91,160,95,95,160,93) # [ __ ] expressed as explicit
  for (line in data$text) {
    for (words in line) {
      for (words in strsplit(words, " ")){
        for (word in words) {
          if (word == "[applause]") {
            soundEffects[word] = soundEffects[word] + 1
            soundEffectsAll[word] = soundEffectsAll[word] + 1
          }
          # music is only present in the beginning and end of a set
          # maybe once in between. Visual inspection of these sets proved that
          # music is often confused with laughter in YouTubes captioning system
          else if (word == "[music]") {
            if (!is.na(soundEffects["music"]) &&soundEffects["music"] > 3) {
              soundEffectsAll["[laughter]"] <- soundEffectsAll["[laughter]"] + 1
            }
          }
          else if (word == "[laughter]") {
            soundEffects[word] = soundEffects[word] + 1
            soundEffectsAll[word] = soundEffectsAll[word] + 1

          }
          else {
            utf_codes <- utf8ToInt(word)
            if (length(utf_codes) == 6 && all(utf_codes == explicitUTFCode)) {
              soundEffects["[cuss]"] <- soundEffects["[cuss]"] + 1
              soundEffectsAll["[cuss]"] <- soundEffectsAll["[cuss]"] + 1
            }
          }
        }
      }
    }
  }

  print(soundEffects["[applause]"])
  print(soundEffects["[laughter]"])
  print(soundEffects["[cuss]"])
  print("")

  data$text <- gsub("[[:punct:]]", "", data$text)
  data$text <- removeWords(data$text, stopwords("english"))
  wordDict <- c(wordDict, data$text)

  # --- Sentiment Analysis --- #
  # Assuming file_path is a variable containing the file path
  hist_path <- paste0(folder, modified_string, "_hist.png")
  # Obtain SentimentGI scores
  sentiment <- analyzeSentiment(data$text)
  sentimentGI_scores <- sentiment$SentimentGI
  # Determine breaks for the histogram
  breaks <- "sturges"
  # Set the width and height of the plotting device to make the image smaller
  png(file = hist_path, width = 800, height = 600)
```

```r
  # Create a histogram using the SentimentGI scores
  h <- hist(sentimentGI_scores,
            breaks = breaks,
            main = "SentimentGI Histogram",
            xlab = "Sentiment Score",
            ylab = "Frequency")

  # Customize x-axis ticks and labels
  axis(side = 1, at = pretty(sentimentGI_scores, n = 10), labels = pretty(sentimentGI_scores, n = 10))
  # Adjust text labels to be smaller and inside the bars
  text(h$mids, h$counts, labels = h$counts, adj = c(0.5, -0.5), cex = 0.7)
  # Close the plotting device
  dev.off()
}

allDtm <- DocumentTermMatrix(Corpus(VectorSource(wordDict)))
# Convert the document-term matrix to a data frame
allDtm_df <- as.data.frame(as.matrix(allDtm))

# Visualize word frequencies
word_freq_all <- colSums(allDtm_df)
word_freq_all <- sort(word_freq_all, decreasing = TRUE)

barplot(word_freq_all[1:5], col = "skyblue", main = "Word Frequencies Without Stop Words")
all_wordFreq_path = paste0(folder, "/all_wordFreq.png")
ggsave(all_wordFreq_path, plot = last_plot(), units = "px", device = "png", width = 800, height = 600)

# Generate a word cloud
# wordcloud(words = names(word_freq_all), freq = word_freq_all, scale=c(3,0.5), min.freq = 1, colors = brewer.pal(8, "
    Dark2"))

# --- Overall Sentiment Analysis --- #
# Assuming file_path is a variable containing the file path
hist_path <- paste0(folder, "/all_hist.png")
# Obtain SentimentGI scores
sentiment <- analyzeSentiment(wordDict)
sentimentGI_scores <- sentiment$SentimentGI
# Determine breaks for the histogram
breaks <- "sturges"
# Set the width and height of the plotting device to make the image smaller
png(file = hist_path, width = 800, height = 600)
# Create a histogram using the SentimentGI scores
h <- hist(sentimentGI_scores,
          breaks = breaks,
          main = "SentimentGI Histogram",
          xlab = "Sentiment Score",
          ylab = "Frequency")

# Customize x-axis ticks and labels
axis(side = 1, at = pretty(sentimentGI_scores, n = 10), labels = pretty(sentimentGI_scores, n = 10))
# Adjust text labels to be smaller and inside the bars
text(h$mids, h$counts, labels = h$counts, adj = c(0.5, -0.5), cex = 0.7)
# Close the plotting device
dev.off()

# ---- Topic Modeling --- #
allDtm <- allDtm[rowSums(as.matrix(allDtm)) > 0, ]
lda <- LDA(allDtm, method = "Gibbs", k=10)
terms_df <- as.data.frame(terms(lda, 10))

# Save the list of data frames for further analysis if needed
save(allData, file = "allData.RData")

as.data.frame(wordDict)
```

R Code: Timing EDA

```r
library(ggplot2)

# List to store individual plots
individual_plots <- list()

# Get a list of CSV files in the directory
csv_files <- list.files(
  path = "C:/Users/heffec/Desktop/Semester 8/Data Analytics/FinalProject",
  pattern = "\\.csv$",
  full.names = TRUE,
)
csv_files <- setdiff(csv_files, "C:/Users/heffec/Desktop/Semester 8/Data Analytics/FinalProject/ComicsList.csv")

# List to store spline points and outlier points for each file
all_spline_points <- list()
outlier_indices_list <- list()
folder = "C:/Users/heffec/Desktop/Semester 8/Data Analytics/FinalProject"

for (file_path in csv_files) {
  print(file_path)

  # Read and clean CSV text
  df <- read.csv(file_path)
```

```r
  df$text <- tolower(df$text)

  # Check for missing values
  if (any(is.na(df$start)) || any(is.na(df$speech_rate))) {
    next()  # Skip to the next iteration
  }

  # Calculate speech rate (words per second)
  df$word_count <- sapply(strsplit(df$text, "\\s+"), length)
  df$speech_rate <- df$word_count / df$duration

  # Weighted average speech rate based on duration
  weighted_avg_rate <- weighted.mean(df$speech_rate, df$duration)

  # Cubic spline interpolation for the cleaned data
  spline_points <- smooth.spline(df$start, df$speech_rate)
  spline_df <- data.frame(x = spline_points$x, y = spline_points$y)
  all_spline_points[[file_path]] <- spline_df

  # Create a time series plot with cubic spline interpolation and LOESS smoothing
  plot <- ggplot(df, aes(x = start, y = speech_rate)) +
    geom_line() +
    geom_line(data = spline_df, aes(x, y), color = "green") +
    geom_smooth(method = "loess", se = TRUE, color = "red") +
    labs(x = "Start Time", y = "Speech Rate (words per second)", title = "Speech Rate Over Time with Cubic Spline
         Interpolation and LOESS Smoothing") +
    scale_color_manual(values = c("Actual Data", "Cubic Spline Interpolation", "Locally Estimated Scatterplot Smoothing"))

    # Save individual plot as PNG
    individual_plot_path <- paste0(file_path, "_spline_plot.png")
  ggsave(individual_plot_path, plot, width = 2400, height = 1200, units = "px")

  individual_plots[[file_path]] <- plot
}

# Combine all individual spline points into one big spline
all_spline_points_df_t <- data.table::rbindlist(all_spline_points)

# Get the first 80% of the data
subset_index <- 1:round(0.8 * nrow(all_spline_points_df_t))
subset_spline_points_df_t <- all_spline_points_df_t[subset_index, ]

# Fit a smooth spline to the subset data
spline_points_big_subset <- smooth.spline(subset_spline_points_df_t$x, subset_spline_points_df_t$y, spar = 0.8)

# Create a data frame from the spline points
spline_df_big_subset <- data.frame(x = spline_points_big_subset$x, y = spline_points_big_subset$y)

# Create a big plot with the subset spline
subset_plot <- ggplot(data = spline_df_big_subset, aes(x = x, y = y)) +
  geom_line() +
  geom_smooth(method = "loess", se = FALSE, color = "red") +
  labs(x = "Start Time", y = "Speech Rate (words per second)", title = "Subset Cubic Spline Interpolation")

# Save the subset spline plot as PNG
subset_plot_path <- paste0(folder, "/subset_splinePlot.png")
ggsave(subset_plot_path, subset_plot, width = 2400, height = 1200, units = "px")
```

R Code: Clustering Model

```r
library(topicmodels)
library(tm)
library(dendextend)
library(dplyr)
library(cluster)

explicitUTFCode <- c(91, 160, 95, 95, 160, 93) # [ __ ] expressed as explicit
folder <- "C:/Users/heffec/Desktop/Semester 8/Data Analytics/FinalProject"

# Function to clean text
clean_text <- function(text) {
  # Remove profanity placeholders
  text <- gsub(paste0("\\", paste(explicitUTFCode, collapse = "\\s*")), NA, text, fixed = TRUE)

  # Remove audience behavior annotations
  text <- gsub("\\[laughter\\]", NA, text)
  text <- gsub("\\[applause\\]", NA, text)
  text <- gsub("\\[music\\]", NA, text)

  # Convert to lowercase
  text <- tolower(text)

  return(text)
}

perform_LDA <- function(text, num_topics = 5) {
  dtm <- DocumentTermMatrix(Corpus(VectorSource(text)))
  dtm_matrix <- as.matrix(dtm)
  dtmCleaned <- dtm_matrix[rowSums(dtm_matrix) > 0, ]

```

```
  if (nrow(dtmCleaned) == 0) {
    warning("Empty document-term matrix after cleaning. Skipping LDA.")
    return(NULL)
  }

  lda_model <- LDA(dtmCleaned, k = num_topics)
  return(lda_model)
}

extract_topics <- function(lda_model, threshold = 0.1) {
  topic_terms <- terms(lda_model, 10)
  terms_df <- as.data.frame(topic_terms)
  return(terms_df)
}

allTopics <- list()

# Iterate over each CSV file
for (file_path in csv_files) {
  print(file_path)

  data <- read.csv(file_path)
  data$text <- clean_text(data$text)
  data <- na.omit(data, cols = "text")
  data$text <- removeWords(data$text, stopwords("english"))
  data$text <- gsub("[[:punct:]]", "", data$text)
  data <- na.omit(data, cols = "text")

  individualLDA <- perform_LDA(data$text)

  # Check if LDA was successful
  if (!is.null(individualLDA)) {
    extractedTopics <- extract_topics(individualLDA)
    allTopics[[file_path]] <- individualLDA

    # Transpose the matrix so that rows represent topics
    topic_matrix_transposed <- t(as.data.frame(individualLDA@beta))

    # Calculate distances
    dist_matrix <- dist(topic_matrix_transposed)

    # Perform hierarchical clustering
    hclust_result <- hclust(dist_matrix, method = "ward.D2")
    print(summary(hclust_result))

    # Assign cluster labels to topics
    cluster_labels <- cutree(hclust_result, k = 10)

    # Set the output PNG file path
    file_name <- basename(file_path)
    png_file_path <- file.path(folder, paste0(file_name, "_dendrogramManhattan", ".png"))

    # Open PNG file for plotting
    png(png_file_path)

    # Plot the dendrogram for each document
    dendrogram_ind <- as.dendrogram(hclust_result)
    dendrogram_ind <- color_branches(dendrogram_ind, k = length(unique(cluster_labels)))

    # Plot the individual dendrogram
    plot(dendrogram_ind, main = "Clustering Dendrogram", horiz = TRUE)

    # Save the PNG file
    dev.off()
  }
}


# Combine topic matrices of all documents
all_topic_matrices <- lapply(allTopics, function(lda_model) as.data.frame(lda_model@beta))
combined_topic_matrix <- dplyr::bind_rows(all_topic_matrices)
combined_topic_matrix_transposed <- t(combined_topic_matrix)
dist_matrix <- dist(combined_topic_matrix_transposed)
hclust_result_all <- hclust(dist_matrix, method = "ward.D2")
cluster_labels_all <- cutree(hclust_result_all, k = 10)

# Plot the dendrogram for all documents
dendrogram_all <- as.dendrogram(hclust_result_all)
dendrogram_all <- color_branches(dendrogram_all, k = length(unique(cluster_labels_all)))

# Plot the combined dendrogram
png_file_path <- file.path(folder, paste0("all_dendrogram", ".png"))
png(png_file_path)
plot(dendrogram_all, main = "Aggregate Clustering Dendrogram", horiz = TRUE)
dev.off()
```

R Code: Regression Model

```
# Load the required libraries
library(SentimentAnalysis)
```

```r
library(ggplot2)

sets_with_laughter <- list()
sets_with_applause <- list()

# Iterate over each CSV file
for (file_path in csv_files) {

  print(file_path)

  soundEffects <- c("[applause]"=0, "[music]"=0, "[laughter]"=0, "[cuss]"=0)

  # Read and clean CSV text
  data <- read.csv(file_path)
  data$text <- tolower(data$text)
  explicitUTFCode <- c(91, 160, 95, 95, 160, 93) # [ __ ] expressed as explicit
  data$text <- gsub(paste0("\\s*", paste(explicitUTFCode, collapse = "\\s*")), "", data$text, fixed = TRUE)
  data$text <- removeWords(data$text, stopwords("english"))
  data$text <- na.omit(data$text, cols = "text")

  # Find indices where text is equal to "\\[laughter\\]"
  laughter_indices <- grep("\\[laughter\\]", data$text)
  applause_indices <- grep("\\[applause\\]", data$text)

  data$text <- gsub("[[:punct:]]", "", data$text)

  # Check if there are more than 0 laughter indices
  if (length(laughter_indices) > 1) {
    prev <- 1  # Start from the first index
    avg_sentiments <- numeric()  # Vector to store average sentiments
    laugh_times <- numeric()  # Vector to store laugh times

    for (index in laughter_indices) {
      joke <- data$text[prev:(index - 1)]
      sentiment <- analyzeSentiment(joke)
      sentimentGI_scores <- sentiment$SentimentGI
      sentimentGI_scores <- na.omit(sentimentGI_scores)
      avg_sentiment <- mean(sentimentGI_scores)
      avg_sentiments <- c(avg_sentiments, avg_sentiment)

      laugh_time <- data$duration[index]
      laugh_times <- c(laugh_times, laugh_time)
      # Check if sentiment analysis result is valid
      prev <- index + 1
    }

    # Create a data frame for each set with laughter
    set_data <- data.frame(LaughTime = laugh_times, AvgSentiment = avg_sentiments)
    sets_with_laughter[[file_path]] <- set_data
  }

  # Check if there are more than 0 laughter indices
  if (length(applause_indices) > 1) {
    prev <- 1  # Start from the first index
    avg_sentiments <- numeric()  # Vector to store average sentiments
    applause_times <- numeric()  # Vector to store laugh times

    for (index in applause_indices) {
      joke <- data$text[prev:(index - 1)]
      sentiment <- analyzeSentiment(joke)
      sentimentGI_scores <- sentiment$SentimentGI
      sentimentGI_scores <- na.omit(sentimentGI_scores)
      avg_sentiment <- mean(sentimentGI_scores)
      avg_sentiments <- c(avg_sentiments, avg_sentiment)

      applause_time <- data$duration[index]
      applause_times <- c(applause_times, applause_time)
      # Check if sentiment analysis result is valid
      prev <- index + 1
    }

    # Create a data frame for each set with laughter
    set_data <- data.frame(ApplauseTime = applause_times, AvgSentiment = avg_sentiments)
    sets_with_applause[[file_path]] <- set_data
  }
}

combined_data <- bind_rows(sets_with_laughter)
combined_data_applause <- bind_rows(sets_with_applause)

# Fit linear regression model - laugh
lm_model_laugh <- lm(AvgSentiment ~ LaughTime, data = combined_data)
print(summary(lm_model_laugh))
laugh_graph <- ggplot(combined_data, aes(x = LaughTime, y = AvgSentiment, color = "orange")) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(title = "Laugh Time vs. Average Sentiment")

# Fit linear regression model - applause
lm_model_applause <- lm(AvgSentiment ~ ApplauseTime, data = combined_data_applause)
print(summary(lm_model_applause))
# Corrected applause regression plot
```

```
app_graph <- ggplot(combined_data_applause, aes(x = ApplauseTime, y = AvgSentiment, color = "magenta")) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(title = "Applause Time vs. Average Sentiment")

# Save the corrected plot
path_applause <- paste0(folder, "/all_applauseRegression.png")
ggsave(path_applause, app_graph, width = 2400, height = 1200, units = "px")
```