Kai Wilson

1 August 2024

Data Practicum

Understanding Customer Feedback With Natural Language Processing

Summary

The goal of this research paper is to use machine learning techniques to analyze user review and complaint data to learn what are the most common keywords and complaints people have with Carvana, using the company's Better Business Bureau (BBB) profile. I will also try to find the overall sentiment of Carvana reviews. Are they, as expected, mostly negative or is there a good variety of reviews within the data. This question is primarily important to a business because it will allow them to get a better understanding of not only how the business is viewed by the public but also if they notice a spike or dip in sales.

The data was collected by web scraping the company homepage, company details, reviews and complaints pages from the BBB site. Then, once I have the data collected, I will then use machine learning libraries to extract the most prevalent keywords and key phrases. In addition, I will create and train my own personalized machine learning model to get the overall sentiment of the customer reviews. Then I will visualize my findings in a python shiny dashboard that is then deployed using shinyapps.io.

When reviewing the data, I found that the overall sentiment across the years was mostly negative. However, in 2021 there was a much larger amount of positive and neutral reviews compared to other years. Another interesting finding was the number of complaints and reviews Carvana received has declined since 2021. A possible reason for this decline could be from a slow in car sales, or buyers interested in buying their cars in person.

Overall, the methods and analysis proved to be successful as I could determine the overall sentiment and gain insights about the customer feedback by reviewing the most popular keywords and key phrases.

Introduction

Better Business Bureau (BBB) is a nonprofit organization that provides information about business and charities. The purpose of BBB is to help consumers by exposing bad business practices so consumers can make informed decisions before they purchase or use a service from the business. How BBB achieves this transparency is by allowing consumers to file complaints and reviews on the site to inform other consumers about their experience with the company. While reviews do not affect a company's rating, complaints can affect the BBB letter grade.

BBB only handles disputes that are strictly related to marketplace issues a consumer has with a business service and products a business provides. Complaints are available for 36 months and processed in approximately 30 days or fewer on average, and most consumers desire a resolution when they make a complaint. Consumers can file a complaint in one of two ways, either through the BBB website, or by submitting a letter to your nearest BBB location. However, BBB encourages the consumer to contact the business at least once to resolve the issue, before making a complaint. For a complaint to but published on the site, it must meet the BBB Complaint Acceptance Guidelines.

All complaints submitted by the user cannot change, edit, or delete their complaint after it is submitted. A copy of the complaint is then submitted to the business for their review. BBB encourages all users to submit a complaint that is a truthful account of their experience with the business and not to include any personal information or inappropriate language. BBB will attempt to remove this information, but it is the responsibility of the user to remove the information. All complaints that are submitted will be available to read on the business's profile for at least 3 years (complaints). Below is the timeline that BBB has set when it applies to processing complaints.

1. A consumer files a complaint.
2. BBB will process the complaints within two days.
3. Once the complaint is processed the business will be asked to respond withing 14 days. If there is no response received a follow-up letter will be sent.
4. depending on the Business's response one of two things will happen:
    a. If a response is received, then BBB will notify the consumer and ask if they want to respond.
    b. If no response is received, then the consumer will be notified there was no response.

Once a complaint is closed, BBB will assign one of the following closing statuses:

- Resolved = The complainant verified the issue was resolved to their satisfaction.
- Answered = The business addressed the issues within the complaint, but the consumer either (a) did not accept the response, OR (b) did not notify BBB as to their satisfaction.
- Unresolved = The business responded to the dispute but failed to make a good faith effort to resolve it.
- Unanswered = The business failed to respond to the dispute.
- Unpursuable = BBB is unable to locate the business.

BBB handles reviews a bit differently. Similarly to complaints, A reviewer will give their feedback, then BBB gives the business an opportunity to respond to the review. However, it is nearly impossible to guarantee that each response received is from a real consumer. If a business receives a review that they disagree with, they can reach out to BBB to express their concerns. With that said, BBB will take the following steps to ensure their system is not being misused:

- Validating the email address or phone number of reviewers.

- Allowing the business to confirm an interaction with a particular consumer and respond to customer comments.
- Allowing the reviewer to also submit comments on the customer review. Comments from third parties are not allowed.
- Scrubbing reviews to remove any inappropriate language or personal information before the review is published.
- Not allowing anonymous reviews or reviews in which the consumer was compensated for the review, either directly or indirectly.
- Publishing the customer review of the company's BBB Business Profile.

I will focus on Carvana for this review. This online car retailer has been in business for the last 12 years and is still operational to this day. Most people know them for their car vending machines that can be found in certain cities. Currently, they have a total of 7,178 complaints, with an average of 2,649 customer reviews giving them a rating of 1.58 /5 stars. with these factors BBB has rated them as a C+ company (customer reviews).

The goal of this research paper is to use machine learning techniques to analyze user review and complaint data to learn what are the most common keywords and complaints people have with Carvana. I will also try to find the overall sentiment of Carvana reviews. Are they, as expected, mostly negative or is there a good variety of good, bad and neutral reviews? This question is important to a business because it will allow them to get a better understanding of not only how the business is viewed by the public but also if they notice a spike or dip in sales.

Customer reviews and complaints can also play a huge role in the Reputation of a company. Reputation marketing is an approach where a business uses their brand's reputation to build trust with customers. A positive reputation shows customers that your brand is dependable and trustworthy. A business can leverage this positive perception to show off how reliable they are to new customers

(Oyinloye). This makes monitoring customer reviews very crucial to a business's success because the best people to know your business flaws are the people using the products and services.

Unfortunately, many researchers have found that consumers are more likely to leave negative reviews than positive reviews. However, these negative reviews in their own way can be beneficial to a business. Even though not all negative reviews are legitimate, some of the reviews can reveal the shortcomings of the business and different pain points that customers may face (Clark).

Everyone knows no business is perfect, so when they see that there are negative reviews, it can lead people to trust a business more because they can understand some of the common problems people have working with the business. These negative reviews also allow businesses to assist disappointed customers and win them back, which may lead them to leave a positive review that could bring in more customers (Garfinkel).

Methods and Techniques

To collect the data from the Better Business Bureau website, I needed to create a python script that could grab a large amount of data from a website automatically, this is commonly known as web scraping. By extracting website content with a script, I can extract a large amount of data automatically and that saves a ton of time and will allow me to spend more time to processing the data for analysis. But not every site is the same, every site is structured differently from the next. The front end of Walmart's website looks vastly different from the front end of Amazon. So, in order to pull the information effectively from a site, you will need to create a script that is tailored to the website you are scraping.

There are various things that you will need to keep in mind when you are web scraping data, first you will need to determine whether it is legal. In every case, scraping data that is not publicly available is illegal. This means you need a login to access the information, you are not allowed to scrape the information. With that said, if you are scraping data that is public, you will need to read the site's 'Terms of Use' document to determine if this site allows their data to be scrapped. BBB allows users to collect

data from their site, however under section 2 of their 'Terms of Use' it states that "When using the Sites under the foregoing license, you shall not directly or indirectly (a) use the Sites to create, supplement, or compile data for any service, website, application, or documentation that performs substantially the same functionality as the Sites" (Terms of use). This means that we can scrape the information, but we cannot use it to make a replica of the BBB site or its various pages, which we do not intend to do.

The most important aspect of web scraping is having the correct urls. To extract the information, I needed to have the correct URL to receive the HTML I plan to parse through. Apart from my internship, I was required to grab the information from a variety of sites instead of just Carvana, which this paper focuses on. So, to accomplish this I needed to find each company's specific company id and then inject that into the one standard url.

For example:

Original Link

https://www.bbb.org/us/nj/princeton/profile/credit-union/healthcare-employees-federal-credit-union-0221-4001371

Code to change the link

company id = 0704-96088173

url = f'https://www.bbb.org/us/nj/princeton/profile/credit-union/healthcare-employees-federal-credit-union-{company id}'

The example above shows the original link that can be found on the BBB site. The highlighted portion of the link is the company's unique id. We can swap out the id for another company id and it will change the entire link redirecting to the new company.

When creating a web scraping script in python, you will use first import the library '*request'*. This will make a request to the site to for the entire front end of the website (HTML, CSS Styling, and

JavaScript elements.). If the request is granted, it will send back the code 200. If a page is not found, you will receive a code 404. If the server becomes unavailable, it will throw a code of 503. (HTTP) The most common codes that I received when scraping the data were code 200, 404 and 429, meaning that I was being rate limited.

To manage the traffic a site receives sites, it will limit the number of actions that a user can perform to prevent excessive or abusive use of the network, server or resources. This practice is called rate limiting, and it's used to ensure that the resource is available to all users. This technique also serves to prevent malicious actors from overburdening the system and cause attacks like Denial of Service (DOS). To overcome the hurdle of being rate limited created, I created the following function in figure 1 below:

```python
def get_page_with_retry(self, company, url, max_retries=5, initial_delay=30):
    results_list = []
    for attempt in range(max_retries):
        try:
            r = requests.get(url, headers= self.header, proxies= self.proxy, timeout=30)
            if r.status_code == 200:
                self.log_to_db(company, "accepted", "good")
                return r
            elif r.status_code == 429:
                if attempt == 3:
                    delay = initial_delay * (2 ** attempt) + random.uniform(40,60)
                    self.log_to_db(company, f"Received 429. Retrying after {delay:.2f} seconds. Attempt {attempt + 1}/{max_retries}", "error")
                    time.sleep(delay)
                    print(f'{time.sleep(30)}')
                else:
                    delay = initial_delay * (2 ** attempt) + random.uniform(40,60)
                    self.log_to_db(company, f"Received 429. Retrying after {delay:.2f} seconds. Attempt {attempt + 1}/{max_retries}", "error")
                    time.sleep(delay)
            elif r.status_code == 404:
                self.log_to_db(company, f"404 error: page does not exist", "error")
                return "SKIP"
            else:
                r.raise_for_status()
                results_list.append({
                    'company': company,
                    'error results' : f"Failed to retrieve data due to [ {r.status_code} ] code",
                    'url error' : f"Failed to retrieve data due to [ {r.status_code} ] code",
                })
                return results_list
        except RequestException as e:
            self.log_to_db(company, f"Request failed: {str(e)}. Retrying...", "retrying to get page")
            time.sleep(random.uniform(5,15))
    raise Exception(company, f"Failed to retrieve page after {max_retries} attempts", "error")
```

Figure 1: get page with retry function

The get page function will check the status of the request it receives. If it the request is 200 then it will pass the HTML it received. The request receives a code 404 then it will return skip and the next URL will be passed through. Lastly if the function receives 429 then it will retry a total of five times. Each

attempt will have an initial delay of 30 seconds * by 2 with an exponent of the attempt number then it will be added to a random number between 40 seconds and 60 seconds. This method proved to be the best approach when taking breaks in making requests as it would reach the 3rd attempt and then continue scraping the companies.

The urls that were created were for the homepage that had a general overview of the company's BBB profile, the businesses detail page for more specific information regarding the business, the complaints page and reviews page. With the information I received from the request, I needed to use the python library 'Beautiful Soup' to parse through the HTML and grab the information. We will store the HTML inside this variable like the following 'soup = BeautifulSoup(request.text, 'html.parser')'. We can then use that soup variable with the HTML inside to find all the information that we need.

For example, the HTML string:

Average of 2,646 Customer Reviews

The code to grab the text:

Average rating = soup.find('p', class_="bds-body text-size-5")

The homepage was fairly simple to scrape as all the elements were present on the page and could be easily grabbed using the above technique. For the other pages, I needed to use more creative methods to grab the information I needed.

The details page was structured similarly to the homepage, but there were a few company sites with a *'read more'* button and when the button is clicked, it would reveal more items. For example, if a site has ten phone numbers, only three would be shown at first. Then when the *'read more'* button is clicked, it will reveal all ten phone numbers. To overcome this, I created a function that will go through and click the 'read more' button, as shown in figure 2 below:

```
try:
    self.log_to_db(company, "Waiting for 'Read More' button", "good")
    # Wait for the "Read More" button to appear
    read_more_button = WebDriverWait(driver, 20).until(
        EC.presence_of_element_located((By.XPATH, "//button[contains(@class, 'bds-button-unstyled') and text()='Read More']"))
    )
    driver.execute_script("arguments[0].scrollIntoView();", read_more_button)
    time.sleep(1)
    read_more_button = WebDriverWait(driver, 20).until(
        EC.element_to_be_clickable((By.XPATH, "//button[contains(@class, 'bds-button-unstyled') and text()='Read More']"))
    )
    read_more_button.click()
    self.log_to_db(company, "'Read More' clicked", "good")
    time.sleep(randint(1,3))
    return driver.page_source
except Exception as e:
    self.log_to_db(company, f"Unexpected error waiting for 'Read More' button: {str(e)}", "error")
```

Figure 2: click read more button function

The function will first wait for the button element to be present. When the page is loaded, and the element is clickable, it will scroll down to click the button. After the button is clicked, it will return the expanded content.

For the reviews page, all the reviews are contained within an unordered list with a few list items presented at first. When the button is clicked, the page will reload with more list items and if there is more content, the load of more buttons will reappear at the bottom. The load more buttons will keep appearing as long as there are more items and when it is finished, it will show all the items. Figure 3 shows the function that will perform this action.

```
        try:
            load_more_button = WebDriverWait(driver, 30).until(
                                    EC.element_to_be_clickable((By.XPATH, "//button[contains(@class, 'bds-button') and text()='Load More']"))
                                )
            driver.execute_script("arguments[0].scrollIntoView(true);", load_more_button)
            time.sleep(2)
            self.log_to_db(company, "'load more' button found", "good")
            while load_more_button:
                try:
                    driver.execute_script("arguments[0].scrollIntoView(true);", load_more_button)
                    load_more_button.click()
                    time.sleep(3)
                    self.log_to_db(company, 'load more button clicked', "good")
                except NoSuchElementException:
                    print(f"No 'Load More' button found, exiting loop")
                    break
                except StaleElementReferenceException:
                    print(f"encountered stale element")
                    break
            self.log_to_db(company, 'finished clicking load more', "good")
            return driver.page_source
        except Exception as e:
            self.log_to_db(company, f"Unexpected error waiting for 'load more' button: {str(e)}", "error")
    except Exception as e:
        self.log_to_db(company, f"Attempt {attempt + 1} failed: {type(e).__name__}", "retrying to get page")
        self.log_to_db(company, f"Error details: {e}", "retrying to get page")
        if attempt < max_retries - 1:
            delay = intial_delay * (2 ** attempt) + random.uniform(0,5)
            self.log_to_db(company, f"Retrying in {delay:.2f} secs...", "retrying to get page")
            driver.refresh()
            time.sleep(delay)
        else:
            self.log_to_db(company, f"Max retries reached Failed to click 'Read More' button: {type(e).__name__} grabbing available content", "error")
            self.log_to_db(company, f"Error details: {e}", "error")
            driver.quit()
            return False
    driver.quit()
    return driver.page_source
```

Figure 3: Load more function

While the load more button is present, it will scroll down to the page then click the button. It will continue to do this until the load more button is not found. Then it will return the updated url with the entire unordered list revealed. An issue I ran into getting this section of the code to work is when I encountered a stale element error. This error means that the element is no longer found on the DOM and has been removed or the document has changed. I needed to place this properly within the loop to overcome this issue. The hurdle with the complaints page was slightly simpler. Unlike the reviews page, the complaints page has the information listed page by page. So, I would just need to update the link in with the next page number:

f'https://www.bbb.org/us/ca/laguna-hills/profile/leasing-services/american-capital-group {company_id}/complaints?page={page_number}'

At first it seemed simple I would make a request to grab the last page and then I would update the {page_number} to grab all the pages. However, when I tested the script, the browser only updates the url shown below:

f'https://www.bbb.org/us/ca/laguna-hills/profile/leasing-services/american-capital-group-{company_id}/complaints'

Any other link that is provided will not update the link and throw a 404 error. To overcome this, I needed to manipulate the link differently from the others, as shown in figure 4 below:

```
url = row['complaints_url']
current_page = 1
last_page = self.grab_page_number(company, url)
self.log_to_db(company, f"Getting ready to scrape {company}'s complaints page", "good")
self.log_to_db(company, url,"good")
r = self.get_page_with_retry(company, url)
new_url = r.url
print(new_url)
while current_page < last_page:
    updated_url = new_url + f"?page={current_page}"
    r = self.get_page_with_retry(company, updated_url)
    print(updated_url)
```

Figure 4: grab last page and update url

The code will take the previously created url and send out a request. When the url changes, the updated url will be stored in the variable new_url and then reconstructed to make the updated_url.

With this updated url, I can then add the page number and scrape all the pages. Since this script will be running automatically and will be used for a large volume of text, I created a function to log all the console information into a SQLite database. This will be useful because I can go back and find all the companies that threw an error I didn't find during the creation of the web scrapper and then address them at a later. I found this to be easier than other logging functions because it gets to the point, and it's structured in a way that I could easily review.

```
def log_to_db(self, company_name, message, message_type):
    self.current_table = self.create_daily_table()
    timestamp = datetime.now().isoformat()
    if self.current_table != self.create_daily_table():
        self.current_table = self.create_daily_table()
    self.cursor.execute(f"INSERT INTO bbb_log (timestamp, company_name, message, message_type) VALUES (?,?,?,?)",
                        (timestamp, company_name, message, message_type))
    self.conn.commit()
    print(f"{company_name}: {message}")
```

Figure 5: log to console to database

The function will take in the company name, the predefined message, and the type of message it is (either good or error). Then in the console it will print the company name and the message that was logged in.For each page, every item is stored inside a dictionary, that is appended into a list and returned as a data frame. The data frame is then sent to the Leadx postgres database to be stored to be and analyzed later.

My task for analyzing data was to find keywords/key phrases in from the consumers reviews and complaints. To complete this task, I needed to create a function to grab the keywords and a function to grab the key phrases.

Keywords and key phrases are important for a variety of reasons. If you are using them in your marketing strategy, you would want to know the best words that will get customers to convert. Keywords and key phrases are also important for search engine optimization (SEO) because the more you use them, the higher your site will rank on social media platforms. For the purpose of this research, we will be looking at the keywords and phrases to determine what are the most important items that customers bring up when they are to leave a complaint or review.

To find the key phrases, I created the function below with the python library *'KeyBERT'* and *'Keyphrase_vecotorizer'*. KeyBERT is a popular keyword extraction library that leverages BERT-based language models to create keywords and key phrases. KeyBERT uses BERT embeddings to find the words and phrases that are most semantically similar to the document as a whole. One of the advantages of using KeyBERT is that you do not need to train the model because you can work with pre-trained BERT models. On the other hand, Keyphrase_vectorizer is a tool used in natural language processing for extracting keyphrases from text and converting them into numerical vectors.

For the purpose of this research, I used a KeyphraseCountVectorizer that will extract keyphrases from text, then count the occurrences of the keyphrases and create a sparse matrix of these counts. An advantage of using these together is that you can receive both semantic and statistical perspectives on the

provided text. It also allows improved keyword feature extraction. KeyBERT can identify semantically important keywords that might be missed by frequency-based methods. While KeyphraseCountVectorizer can capture the frequency of important phrases, including multi-word expressions. This approach also provides a balance of quality and speed, where KeyBERT provides high quality contextual keywords and KeyphraseCountVectorizer provides speed for analyzing keywords and can handle large volumes of text efficiently. The function I used is shown in figure 6 below:

```python
# multi word
def keyphrase(dataframe: pd.Series) -> pd.DataFrame:
    '''
    Generates keywords of multiple lengths from given text input
        Args:
            column (pd.Series): DataFrame column containing text
        Returns:
            pd.DataFrame: DataFrame of keyword phrases and their importance

        print("Keywords, Phrases")
    '''
    if not isinstance(dataframe, pd.DataFrame):
        raise TypeError("train_data must be a pandas DataFrame")

    # Check if required columns exist
    if 'text' not in dataframe.columns:
        raise ValueError("train_data must have column labels 'text'")

    kw_model = KeyBERT()
    # Convert all values to strings and replace NaN with empty string
    dataframe['text'] = dataframe['text'].astype(str).replace('nan', '')

    vectorizer = KeyphraseCountVectorizer(
        pos_pattern='<N.*>+<N.*>|<N.*>|<J.*>+<N.*>+',
        stop_words='english'
    )


    try:
        kw = kw_model.extract_keywords(
                                docs= dataframe['text'].to_list(),
                                vectorizer= vectorizer)
    except ValueError as e:
        print(f"Error occurred: {e}")
        return pd.DataFrame()



    keyword_phrases_list = []
    for kw_tuple in kw:
        try:
            if kw_tuple:  # Check if kw_tuple is not empty
                keyword_list, keyword_importance = map(list, zip(*kw_tuple))
                keyword_phrases_list.append({'keyword': keyword_list,
                                             "importance": keyword_importance})
            else:
                keyword_phrases_list.append({'keyword': [], "importance": []})
        except Exception as e:
            print(f"Error processing keyword tuple: {e}")
            keyword_phrases_list.append({'keyword': [], "importance": []})

    keyword_phrases = pd.DataFrame(keyword_phrases_list)

    if not keyword_phrases.empty:
        keyword_phrases['keyword'] = keyword_phrases['keyword'].apply(lambda x: ', '.join(x) if x else '')
        keyword_phrases['total_importance'] = keyword_phrases['importance'].apply(lambda x: sum(x) if x else 0).round(2)
        keyword_phrases['importance'] = keyword_phrases['importance'].apply(lambda x: '-'.join(map(str, x)) if x else '')

    return keyword_phrases
```

Figure 6: Key phrase function

The function will take in a data frame of text. I set up some error catching to make sure that only a data frame can be entered, and it has to have a column named text. It then takes the data frame column text. When the extract key phrase function is applied, it will return a tuple of the keyword and their importance. From there, I unpack all the key phrases into a list of dictionaries and then transform it into a

data frame. Once it is inside the data frame, I manipulate the strings to clean up the output. The original output will show the importance of each word individually. I created another column for the total importance of the entire phrase.

When I wanted to find the overall keywords and their importance, the function operates similarly to the key phrase function. KeyphraseCountVectorizer is used for the rapid generation of keywords and their importance. How this was accomplished is shown in the function below:

```
def overall_keywords_and_importance(dataframe):
    if not isinstance(dataframe, pd.DataFrame):
        raise TypeError("train_data must be a pandas DataFrame")
    # Check if required columns exist
    if 'text' not in dataframe.columns:
        raise ValueError("train_data must have column labels 'text'")
    try:
        # In case some items are not strings
        dataframe['text'] = dataframe['text'].astype(str)
        vectorizer = KeyphraseCountVectorizer()
        list_matrix = vectorizer.fit_transform(dataframe['text'])
        feature_names = vectorizer.get_feature_names_out()
        importance = list_matrix.sum(axis=0).A1
        keyword_list = []
        for keyword, importance  in zip(feature_names, importance):
            keyword_list.append({
                'keyword': keyword,
                'importance': importance})
        unsorted_keywords = pd.DataFrame(keyword_list)
        keywords = unsorted_keywords.sort_values('importance', ascending=False)
    except Exception as e:
            keyword_list.append([])
            keywords = pd.DataFrame(keyword_list)
    return keywords
'''
```

Figure 7: Overall keywords and their importance function

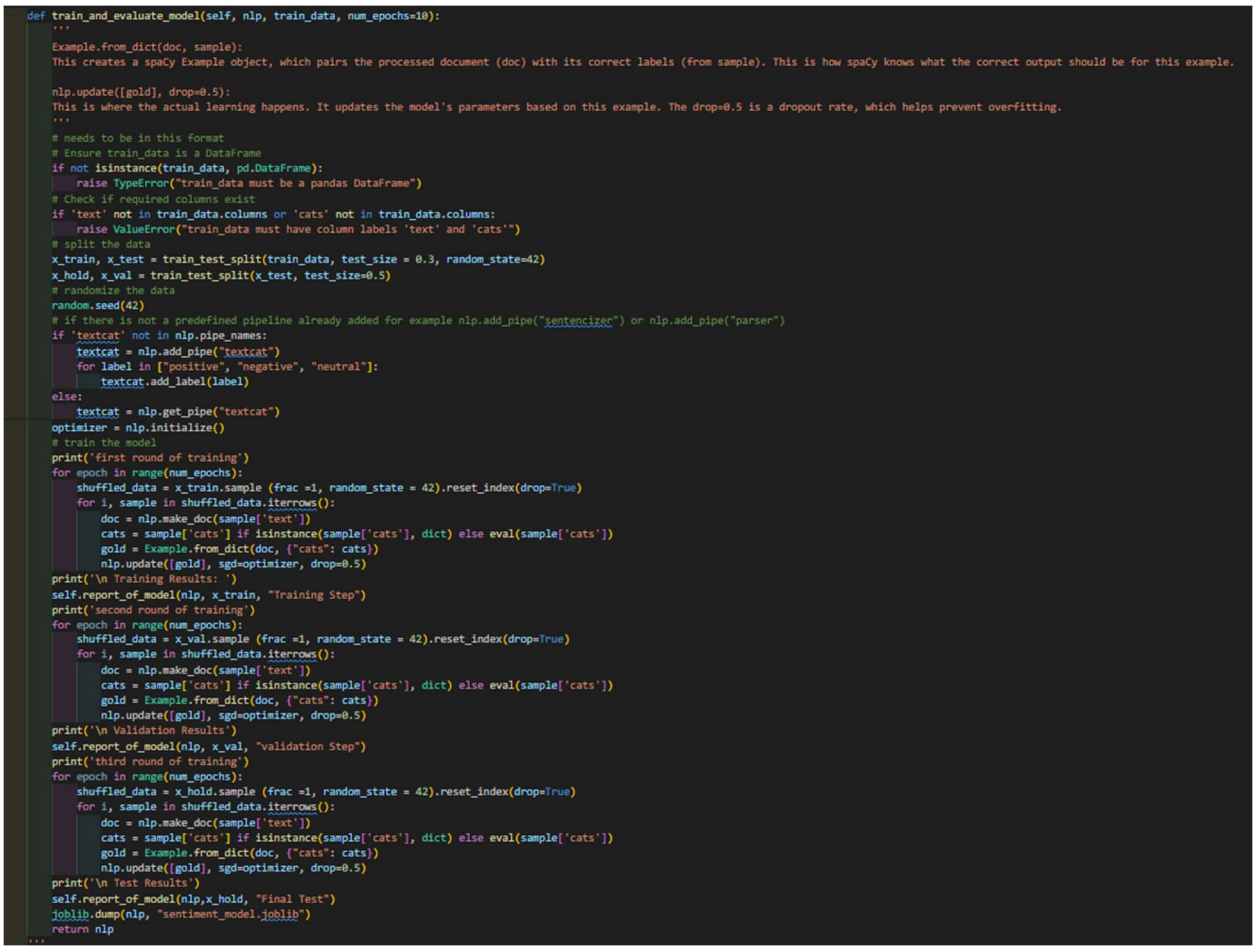
An issue I ran into when extracting the overall keywords, I didn't account for non-letters that would appear or stop phrases common stop words. For this I created a function in figure 8 below to remove all the non-letters, single letters and stop words.

```python
def remove_all_non_letters(dataframe):
    if not isinstance(dataframe, pd.DataFrame):
        raise TypeError("train_data must be a pandas DataFrame")

    # Check if required columns exist
    if 'keyword' not in dataframe.columns:
        raise ValueError("train_data must have column labels keyword")


    common_stop_words = [
    "a", "about", "above", "after", "again", "against", "all", "am", "an", "and", "any", "are", "as", "at",
    "be", "because", "been", "before", "being", "below", "between", "both", "but", "by",
    "could",
    "did", "do", "does", "doing", "down", "during",
    "each",
    "few", "for", "from", "further",
    "had", "has", "have", "having", "he", "he'd", "he'll", "he's", "her", "here", "here's", "hers", "herself",
    "him", "himself", "his", "how", "how's",
    "i", "i'd", "i'll", "i'm", "i've", "if", "in", "into", "is", "it", "it's", "its", "itself",
    "let's",
    "me", "more", "most", "my", "myself",
    "no", "nor", "not",
    "of", "off", "on", "once", "only", "or", "other", "ought", "our", "ours", "ourselves", "out", "over", "own",
    "same", "she", "she'd", "she'll", "she's", "should", "so", "some", "such",
    "than", "that", "that's", "the", "their", "theirs", "them", "themselves", "then", "there", "there's",
    "these", "they", "they'd", "they'll", "they're", "they've", "this", "those", "through", "to", "too",
    "under", "until", "up",
    "very",
    "was", "we", "we'd", "we'll", "we're", "we've", "were", "what", "what's", "when", "when's", "where",
    "where's", "which", "while", "who", "who's", "whom", "why", "why's", "with", "would",
    "you", "you'd", "you'll", "you're", "you've", "your", "yours", "yourself", "yourselves", 'get', 'back','get'
    ]

    # Identify the columns with the non letters
    cleaned_df = dataframe.copy()

    # cleaning of unnecessary string data
    cleaned_df['keyword'] = cleaned_df['keyword'].astype(str)
    cleaned_df['keyword']  = cleaned_df['keyword'].str.replace(r'[^a-zA-Z\s]', '', regex=True)
    cleaned_df['keyword'] = cleaned_df['keyword'].str.replace(r'\b[a-zA-Z]\b','',regex=True)
    cleaned_df['keyword']  = cleaned_df['keyword'].str.replace(r'\s+', '', regex=True).str.strip()

    for i, row in cleaned_df.iterrows():
        if row['keyword'] == common_stop_words:
            cleaned_df.at[i, 'keyword'] = ' '

    # drop empty rows
    cleaned_df = cleaned_df[cleaned_df['keyword'] != '']

    #print results
    original_columns = len(dataframe)
    cleaned_columns = len(cleaned_df)

    print(f'Original columns: {original_columns}')
    print(f'Columns after cleaning: {cleaned_columns}')
    return cleaned_df
```

Figure 8: Remove stop words and non-letters

My next task was to get the overall sentiment of the review data with sentiment analysis, which is analyzing digital text to determine if the tone of the text is positive, negative, or neutral. By analyzing customer reviews, we can get an overall idea of customers' attitudes toward the company. From there, a business can look into how to either improve or keep providing what the customers want.

To complete the task of determining a model's sentiment, I used the library *'spacy',* a popular open-source library, for advanced natural language processing. This specific library has a wide range of NLP functionalities and can process many texts efficiently. I specifically chose this library because I can load in a black instance of the model for the language I want and then train it with the text I want so that it is specifically trained on the data I want it to analyze. Figure 9 below shows the complete model:



```
def train_and_evaluate_model(self, nlp, train_data, num_epochs=10):
    '''
    Example.from_dict(doc, sample):
    This creates a spaCy Example object, which pairs the processed document (doc) with its correct labels (from sample). This is how spaCy knows what the correct output should be for this example.

    nlp.update([gold], drop=0.5):
    This is where the actual learning happens. It updates the model's parameters based on this example. The drop=0.5 is a dropout rate, which helps prevent overfitting.
    '''
    # needs to be in this format
    # Ensure train_data is a DataFrame
    if not isinstance(train_data, pd.DataFrame):
        raise TypeError("train_data must be a pandas DataFrame")
    # Check if required columns exist
    if 'text' not in train_data.columns or 'cats' not in train_data.columns:
        raise ValueError("train_data must have column labels 'text' and 'cats'")
    # split the data
    x_train, x_test = train_test_split(train_data, test_size = 0.3, random_state=42)
    x_hold, x_val = train_test_split(x_test, test_size=0.5)
    # randomize the data
    random.seed(42)
    # if there is not a predefined pipeline already added for example nlp.add_pipe("sentencizer") or nlp.add_pipe("parser")
    if 'textcat' not in nlp.pipe_names:
        textcat = nlp.add_pipe("textcat")
        for label in ["positive", "negative", "neutral"]:
            textcat.add_label(label)
    else:
        textcat = nlp.get_pipe("textcat")
    optimizer = nlp.initialize()
    # train the model
    print('first round of training')
    for epoch in range(num_epochs):
        shuffled_data = x_train.sample (frac =1, random_state = 42).reset_index(drop=True)
        for i, sample in shuffled_data.iterrows():
            doc = nlp.make_doc(sample['text'])
            cats = sample['cats'] if isinstance(sample['cats'], dict) else eval(sample['cats'])
            gold = Example.from_dict(doc, {"cats": cats})
            nlp.update([gold], sgd=optimizer, drop=0.5)
    print('\n Training Results: ')
    self.report_of_model(nlp, x_train, "Training Step")
    print('second round of training')
    for epoch in range(num_epochs):
        shuffled_data = x_val.sample (frac =1, random_state = 42).reset_index(drop=True)
        for i, sample in shuffled_data.iterrows():
            doc = nlp.make_doc(sample['text'])
            cats = sample['cats'] if isinstance(sample['cats'], dict) else eval(sample['cats'])
            gold = Example.from_dict(doc, {"cats": cats})
            nlp.update([gold], sgd=optimizer, drop=0.5)
    print('\n Validation Results')
    self.report_of_model(nlp, x_val, "validation Step")
    print('third round of training')
    for epoch in range(num_epochs):
        shuffled_data = x_hold.sample (frac =1, random_state = 42).reset_index(drop=True)
        for i, sample in shuffled_data.iterrows():
            doc = nlp.make_doc(sample['text'])
            cats = sample['cats'] if isinstance(sample['cats'], dict) else eval(sample['cats'])
            gold = Example.from_dict(doc, {"cats": cats})
            nlp.update([gold], sgd=optimizer, drop=0.5)
    print('\n Test Results')
    self.report_of_model(nlp,x_hold, "Final Test")
    joblib.dump(nlp, "sentiment_model.joblib")
    return nlp
'''
```

Figure 9: Train the model

To ensure that the model is reuseable, make sure to set up some error catching in the beginning. The function will only accept the training data if it is in the format of a data frame. It will also check if there is a column named text. This will ensure the data is properly formatted before continuing. First, the function will take in the training data and split it into three different sets: a training set, a validation set, and a final testing set. The next part of the function will go through and find if there is a text categorization component in the NLP pipeline. If there is not a pipeline, then it will add a new text categorization component to the pipeline. It will add positive, negative, and neutral text categorizers. If the text categorization already exists, it will retrieve the existing text categorization component from the pipeline.

With the training sets prepared and text categorizers pipelines created or added, we will now prepare the Spacy model by setting up an optimizer and initializing the weights of all the components in the NLP pipeline. This is a crucial step because it will ensure that all parts of the model are properly initialized and ready to be updated during the training iterations.

```
for epoch in range(num_epochs):
    shuffled_data = x_train.sample (frac =1, random_state = 42).reset_index(drop=True)
    for i, sample in shuffled_data.iterrows():
        doc = nlp.make_doc(sample['text'])
        cats = sample['cats'] if isinstance(sample['cats'], dict) else eval(sample['cats'])
        gold = Example.from_dict(doc, {"cats": cats})
        nlp.update([gold], sgd=optimizer, drop=0.5)
```

Figure 10: Training of the model

The code will iterate over the epoch for a specified number of times that is defined in the code above. Then, for each epoch, it will shuffle the training data randomly and prepare the data so that it is in the correct format. After those steps, the function will create the gold standard for training. The gold standard in machine learning refers to high-quality manually labeled data that serves as a ground truth for training and evaluation.

This is used to provide the correct answers that the model should learn to predict. The gold standard is used to calculate the loss (error) during training, which guides the model's learning process. The gold standard is structured in this format using the Example.from_dict() it will take the created input text, then it will take the category labels. It will then combine the input doc with the expected output cats.

The gold standard will allow the Spacy Model to compare the model's predictions with the correct answers during training. From there, nlp.update() is called, and it will compare the models' predictions with the gold standard. The difference between the gold standard and the prediction will calculate the loss and update the model's parameters.

The algorithm will go through these steps for each training set and print out a classification report using the function below:

```
def report_of_model(self, nlp, data, stage):
    y_true = []
    y_pred = []
    for i, sample in data.iterrows():
        doc = nlp(sample['text'])
        true_cats = sample['cats'] if isinstance(sample['cats'], dict) else eval(sample['cats'])
        pred_cats = doc.cats
        true_label = max(true_cats, key=true_cats.get)
        pred_label = max(pred_cats, key=pred_cats.get)
        y_true.append(true_label)
        y_pred.append(pred_label)
    print(f"\n Classification Report for {stage} Data:")
    print(classification_report(y_true, y_pred))
```

Figure 11: report of the model

To report the progress of the function, we will create a new function. It will create a document from the text and then it will extract the true category labels, handling the different formats. Then it will get the predicted categories from the model.

By finding the category with the highest score it will separate them by a true or predicted label. The function will then append the labels to the respective list. As a result, you will have a list of y_true and y_pred. These variables are then placed into a classification report and the results are printed out to the console.

Now that the model was built, I needed to devise a comprehensive training strategy. I scraped a few companies from the BBB website, so the model is familiar with some responses that can be found on the BBB site as they structure reviews a bit differently.

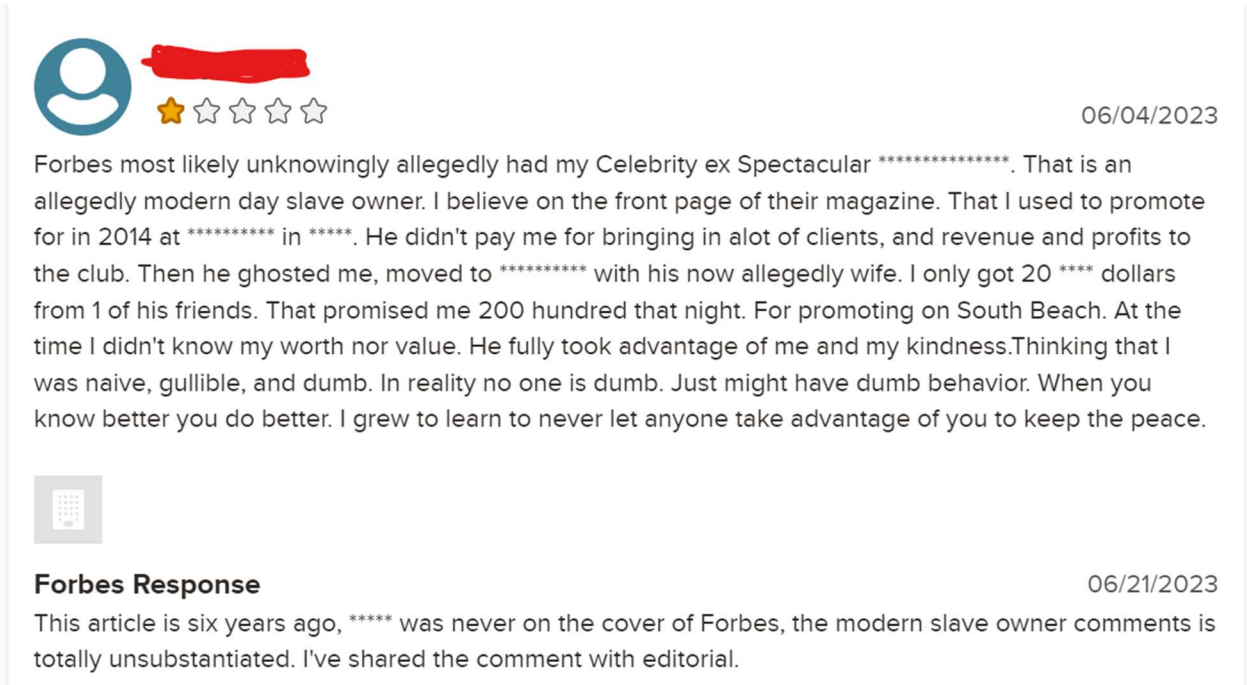


Figure 12: Example BBB review

The way Forbes will format their reviews is by making all personal information or profanity into stars. So, I needed to expose the training model to this type of text as I will review many reviews from the site. However, to make the model more robust and able to handle different parts of speech and slang, I went on to Kaggle and found a 5 GB JSON file I could use to extract review data and transform it into training data. I use the following code in figure 13 to extract the information.

The code will open the JSON file and then it will go line by line and pull out a predefined number of positive, negative, and neutral reviews. I categorized the sentiment of a review in the following method below:

- A review with a score of 4 or 5 stars was considered to be positive
- A review with a score of 3 stars is considered to be neutral
- A review with a score of 2 or 1 stars is considered to be negative

Once the scores are extracted, I created new row with the corresponding rating structured in a dictionary format set the example below:

{'positive': 1.0, 'negative': 0.0, 'neutral': 0.0}

```
keys_of_intrest = ['stars', 'text']
data_list = []

with open('yelp_academic_dataset_review.json', 'r', encoding='utf-8') as file:
    neg_item_counter = 7000
    neu_item_counter = 7000
    pos_item_counter = 7000

    for line in file:
        if neg_item_counter != 0:
            # json 'loads' if for string
            # json 'load' is for json file like object
            item = json.loads(line)
            if item['stars'] == 3:
                if neg_item_counter <= 0:
                    break
                else:
                    filterd_item ={key: item[key] for key in keys_of_intrest if key in item}
                    data_list.append(filterd_item)
                    neg_item_counter -= 1
                    print(f'grabbed item negative Item {neg_item_counter} left to grab')
        else:
            continue
        if neu_item_counter != 0:
            if item['stars'] < 3:
                if neu_item_counter <= 0:
                    break
                else:
                    filterd_item ={key: item[key] for key in keys_of_intrest if key in item}
                    data_list.append(filterd_item)
                    neu_item_counter -= 1
                    print(f'grabbed item neutral {neu_item_counter} left to grab')
        else:
            continue
        if pos_item_counter != 0:
            if item['stars'] > 3:
                if pos_item_counter <= 0:
                    break
                else:
                    filterd_item ={key: item[key] for key in keys_of_intrest if key in item}
                    data_list.append(filterd_item)
                    pos_item_counter -= 1
                    print(f'grabbed item positive {pos_item_counter} left to grab')
        else:
            continue

json_data =pd.DataFrame(data_list)
# clean the data for training

for index, row in json_data.iterrows():
    if row['stars'] > 3:
        json_data.at[index, 'rating'] = "{'positive': 1.0, 'negative': 0.0, 'neutral': 0.0}"
    elif row['stars'] < 3:
        json_data.at[index, 'rating'] = "{'positive': 0.0, 'negative': 1.0, 'neutral': 0.0}"
    else:
        json_data.at[index, 'rating'] = "{'positive': 0.0, 'negative': 0.0, 'neutral': 1.0}"

json_data = json_data.drop(columns='stars')

json_data.to_csv("yelp_reveiw_data.csv", index=False)
```

Figure 13: Yelp review data extraction

Initially, I trained the model using 13,000 reviews from both yelp and BBB. I received poor ratings for the negative and positive categories as shown below in figure 14.

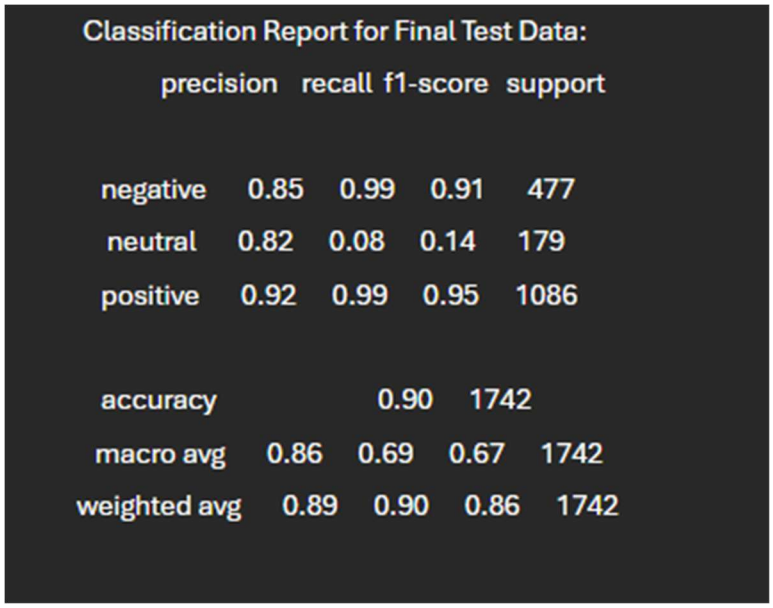


Figure 14: Initial testing of the trained model.

To overcome this hurdle, removed the yelp reviews and then ran the yelp review extraction script again to get an even the amount of negative, positive and neutral reviews for 21,000. This resulted in my CSV training data growing to 22,778 reviews for training. As result, I figure 15 shows the run with the highest scores.

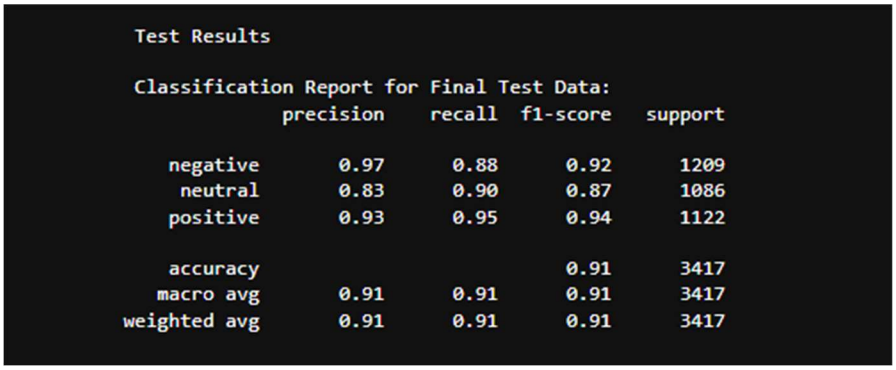


Figure 15: Testing of the model with the highest scores.

To complete the process and use the model I trained, I created one last function. As shown in figure 16. The model will accept a data frame and the path for a joblib file. It will flag the user if they enter an item that is not a data frame and if the joblib file path is incorrect. It will also flag the data user if the data frame does not contain a column called text. The function will then apply the text to the trained model and then append the results into a list and return a data frame with the list of results.

```python
# dataframe input, dataframe output
def predict_sentiment_df(joblib_file_path , dataframe):
    if not isinstance(dataframe, pd.DataFrame):
        raise TypeError("train_data must be a pandas DataFrame")
    if not isinstance(dataframe, pd.DataFrame):
        raise TypeError("train_data must be a pandas DataFrame")

    try:
        trained_model = joblib.load(joblib_file_path)
    except Exception as e:
        raise ValueError(f'Error loading joblib file please make sure you have the correct filepath')

    # Check if required columns exist
    if 'text' not in dataframe.columns:
        raise ValueError("train_data must have column labels 'text'")

    results = []
    texts = dataframe['text']
    docs = list(trained_model.pipe(texts))
    for docs in docs:
        scores = docs.cats
        positive, negative, neutral = scores.values()
        predicted_class = max(scores, key=scores.get)
        results.append({
                'text': docs.text,
                'predicted_class': predicted_class,
                'positive': positive,
                'negative': negative,
                'neutral': neutral
        })
    results_df = pd.DataFrame(results)
    return results_df
```

Figure 16: Function to use the model.

After I completed the training and collection of the data, I visualized my results in a python shiny dashboard and deployed it using shinyapps.io. A link to the dashboard can be found in Appendix A.

Findings

To visualize my results, I first looked at the historical data overtime. Figure 17 shows the data all the data collected from the complaints from 2021 to 2024.

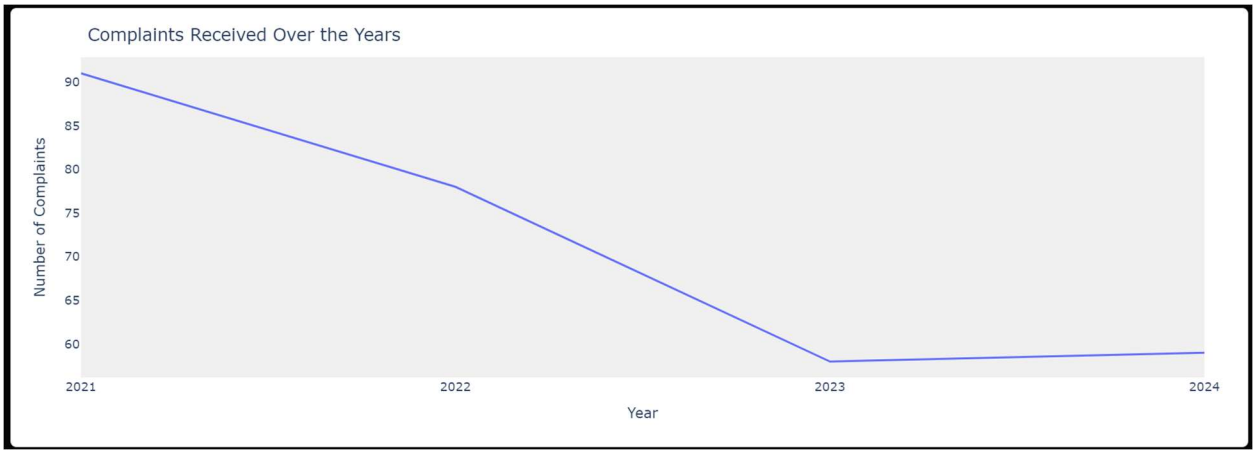


Figure 17: Complaints received over time

There could be a variety of reasons for the sharp decline in reviews. The demand for pre-owned cars may have declined because it has become unaffordable. High car interest rates could be another factor for drivers to keep their current rates, or they could be more interested in purchasing their car in person at their local dealership.

When reviewing the complaint types, I found the complaint types remained consistent, with little improvement over the years. The top three complaints were:

1. Service or Repair Issues
2. Sales and Advertising Issues
3. Product Issues

Figure 18: Complaint type count for 2024

Figure 19: Complaint type count for 2021

A possible reason for the decline in complaints across the category after 2022 is reflected in the first graph shown where there is a decline in complaints. That said, it's important that even though there is a decline in complaints, the overall complaints that the customers are facing remain the same.

Much like the complaints, the reviews have also taken a sharp decline over the years, as shown in Figure 20.

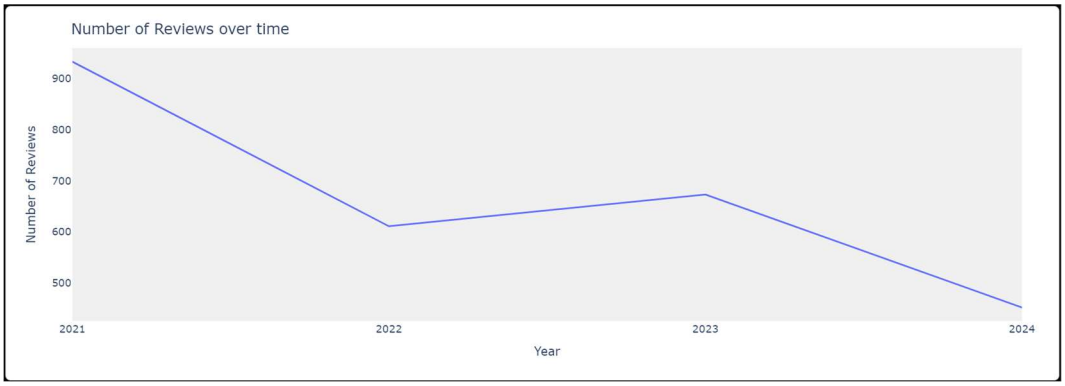


Figure 20: Number of reviews over time.

It would make sense if there were some correlations between complaints, both showing a decrease. If there were fewer cars being sold, it would make sense that both the number of complaints and reviews would decrease.

When reviewing the key phrases, I sorted them by their importance. Below are the top five, but the top 50 can be viewed on the dashboard.

1. different temporary license plates, temporary licenses plate, temporary license plates, temporary license plate, last temporary license plate
2. 5) deceptive business practice, 3) deceptive business practice, 1) deceptive business practice, 4) deceptive business practice, deceptive practices
3. consumer credit transaction, consumer credit, consumer credit transaction period, credit payments, credit transaction
4. title lien, lien title, lien date, incorrect lien, lien
5. purchase carvana, car carvana, carvana vehicle, carvanas possession, carvana

Judging from these key phrases in the complaints most of the customers pain points have to do with issues with license plates, credit, bad business practices and purchasing a car from the company. Figure 21 displays a word cloud of the top 500 most used keywords. The size of the keyword indicates its frequency of use, and the shade of red shows its level of importance.

Figure 21: Top 500 Complaints Keywords

It would make sense that the most important keyword for a car dealership is vehicle, car, miles and the company's name would be popular keywords. However, it is interesting that people frequently used the words day, time, payment, and call. There could be issues with the time of delivery or service, as well as there could be an issue with completing payments and calling the company for help.

Judging from the top 5 key phrases for the reviews section, they appear to carry a negative sentiment.

1. broken air bags, car seller, lawyer, corrupt business, air bags
2. warranty mileage, warranty provider, mileage discrepancy, warranty work, warranty
3. appointment, call, different car, phone call, tuesday
4. business carvana, real customer reviews, carvana, refund, carvanas website
5. transaction, financing, car purchase, purchase process, refund

The reviews seem to resemble some insights that were gathered from reviewing the complaints. Customers are facing issues getting in touch with the company, they are having problems with payments and appear to be requesting refunds. There also appears to be an issue with the vehicle and the warranty's offered.

Figure 22: Top 500 Review Keywords

The keywords for the reviews and complaints appear to be similar. However, there are more words that were ranked with high importance. For example, if delivery appears to be a bright red would mean there are many customers talking about the delivery of the product. Another interesting finding is that experience is a darker read. This could mean there are many customers talking about their experience, which is valuable information for the company.

Given the reviews, it should be no surprise that when viewing the sentiment analysis, it is overall negative.

Figure 23: Sentiment analysis results

What is interesting about the graph is that in 2021 there were many positive and neutral reviews, but they drastically dropped off after 10/2021. Meanwhile, the negative reviews remained consistently high across the years with a large spike at the end of 2021 and then it

drops over the years. When looking at the years individually, 2022–2024 are very similar, with 2021 looking drastically different from the rest.

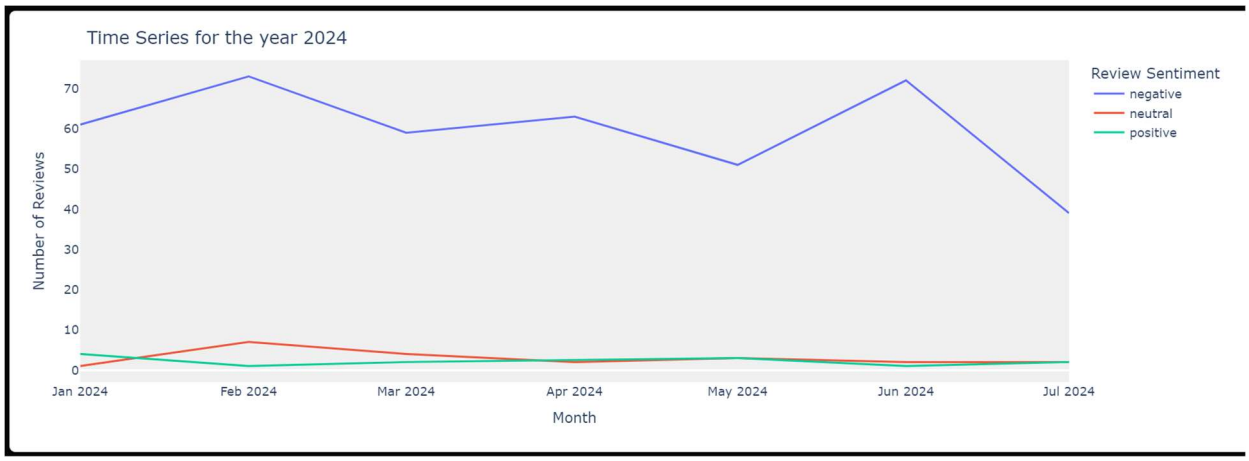


Figure 24: Number of reviews by sentiment 2024

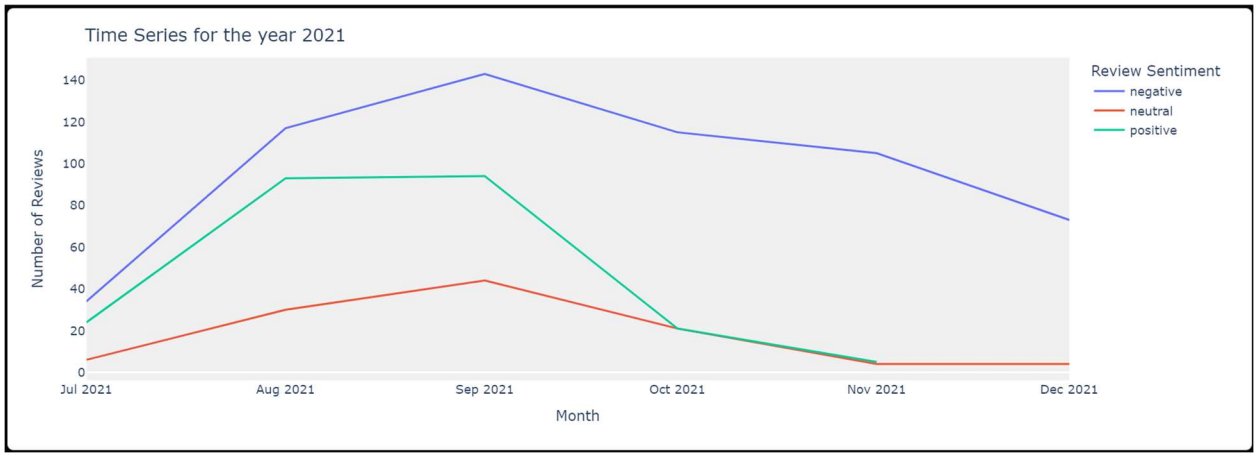


Figure 25: Number of reviews by sentiment 2021

Conclusion

In conclusion, the method I choose proved to be successful. By using the customers feedback I was able to get a better understanding of the company's performance based on their customer feedback. I discovered over time the number of reviews and complaints has decreased over the years. This could be because of many factors such as lower amount of car sales, more people choosing to buy their car at a local dealership. What was interesting was that with the decline in sales, the complaints received, with service and repairs being the number one complaint over the years.

When viewing the keywords and complaints, there were similarities to what customers were saying in their post. A reason for them being similar is that if a customer did not have their complaint answered, they would leave a negative review to get the business's attention. A way to determine if this assumption is true is by viewing the dates of the complaints and reviews and seeing if there are any trends to when a complaint is made and when a review is made.

When viewing the sentiment, reflected the average review rating of 1.58/5 stars, as most of the reviews were predicted to be negative. This means that the training of the model was successful and reflects the known average user rating. However, I found it interesting that most of the positive and neutral reviews were given in 2021, then after that there was a steep drop off while the negative reviews remained consistently high.

One way to improve this research is to view the overall sentiment of the top performers in an industry. A weakness of this research is that it only focuses on one company, so we can only draw conclusions from one source. However, with more than one company in an industry, we can draw better conclusions of a company's performance when looking at a company's customer feedback.

Appendix A

- https://kai-wilson.shinyapps.io/data-prac-dashboard/
- Where the code can be found
    - https://github.com/Kwilso3412/Understanding_Customer_Feedback_with_Natural_Language_Processing

Appendix B

- Where the data can be found
    - https://github.com/Kwilso3412/Understanding_Customer_Feedback_with_Natural_Language_Processing/tree/main/csv%20files

Works Cited

Clark, S. (2023, August 16). Turn customer reviews into gold. CMSWire.com. https://www.cmswire.com/customer-experience/how-customer-reviews-can-make-or-break-your-business/

Clay, B. (2023, August 12). What are keywords? why are keywords important to seo?. Bruce Clay, Inc. https://www.bruceclay.com/blog/what-are-keywords/

Garfinkel, J. (2023, September 13). 7 reasons why negative reviews are valuable. 7 Reasons Why Negative Reviews Are Valuable. https://www.widewail.com/blog/2018/10/16/why-negative-reviews-are-valuable-to-your-business

How BBB complaints are handled: Better Business Bureau®. BBB. (n.d.). https://www.bbb.org/process-of-complaints-and-reviews/complaints

How BBB customer reviews are handled. International Association of Better Business Bureaus. (n.d.-a). https://www.bbb.org/all/customer-reviews/reviews

HTTP Status Codes [SEO 2021]. (n.d.). Moz. https://moz.com/learn/seo/http-status-codes

Oyinloye, M. (n.d.). Why customer reviews are important and how to get more. NiceJob. https://get.nicejob.com/resources/why-customer-reviews-are-important-and-how-to-get-more#:~:text=Reviews%20Establish%20Your%20Business%20as%20a%20Force%20in%20Your%20Industry,-Want%20to%20know&text=Reputation%20marketing%20is%20a%20marketing,you're%20a%20dependable%20brand.

Process of complaints and reviews. International Association of Better Business Bureaus. (n.d.). https://www.bbb.org/all/customer-reviews/complaint-reviews-process

Sentiment Analysis Guide. MonkeyLearn. (n.d.). https://monkeylearn.com/sentiment-analysis/

Terms of use: Better Business Bureau®. BBB. (n.d.-b). https://www.bbb.org/terms-of-use

What is rate limiting: Types & Algorithms: Imperva. Learning Center. (2024, January 8). https://www.imperva.com/learn/application-security/rate-limiting/#:~:text=Rate%20limiting%20is%20a%20technique,Denial%20of%20Service%20(DoS).

Webpages that were scrapped:

Homepage: https://www.bbb.org/us/az/tempe/profile/online-car-dealers/carvana-llc-1126-1000037076

Business Detail page: https://www.bbb.org/us/az/tempe/profile/online-car-dealers/carvana-llc-1126-1000037076/details

Customer Complaints: https://www.bbb.org/us/az/tempe/profile/online-car-dealers/carvana-llc-1126-1000037076/complaints

Customer Reviews: https://www.bbb.org/us/az/tempe/profile/online-car-dealers/carvana-llc-1126-1000037076/customer-reviews