Understanding Customer sentiment for sustainable rail: An analysis on GWR reviews

Introduction:

In recent years, many governments across the globe have no choice but to adopt policies based on sustainability in recent years, thanks to concerns about the climate change crisis. The UK government has promised to reach **net-zero emissions** by **2050** and cut emissions by **68%** by 2030 (Burnett et al., 2024). To help achieve this ambitious goal, promoting and advancing public transportation and using renewable energy is essential for both environmental benefits and economic development opportunities.

This project focuses on analyzing customer reviews of Great Western Railway (GWR), a major train operator in England and Wales, using data collected from Trustpilot between **2019** and **2024**. Understanding customer sentiment is an important aspect to gauge service quality and improve public perception (Shah and Rai, 2022). Since customer satisfaction is key to establishing customer loyalty, the analysis also contributes toward GWR's long-term business resilience and its Environmental, Social, and Governance (ESG) objectives.

To perform this analysis, a combination of Natural Language Processing (NLP) and machine learning methods has been employed. Sentiment analysis using the VADER lexicon evaluates the polarity of customer reviews. Topic detection occurs through the implementation of LDA and BERTopic while common themes and complaints are assessed. Furthermore, classification algorithms, Naive Bayes and logistic regression are also trained on the review dataset with parameter tuning to predict sentiment classifications, which give the company the ability to assess and more rapidly, more effectively, respond to consumer concerns. The project insights and findings will enable GWR to identify shortcomings and thus enhancing the customer experience and to increase overall productivity of railways, which has been declining over the years (Discussion paper on rail industry productivity, n.d.). Improving the services will foster greater public confidence and usage of trains. Encouraging more people to use public transport over their

private transportation directly contributing to achieving the national carbon reduction targets and supporting the UK government's broader sustainability goals.

Literature Review

As digital channels become increasingly dominant, customer feedback represented through online reviews is becoming an ever more important source of business intelligence. Rail operators, such as Great Western Railway (GWR) are being increasingly impacted by user-generated content on platforms like Trustpilot, where customers can publish their experiences about punctuality, ticketing, staff service, and journey comfort. Social media reviews can not only shape perceptions of the public but, when analyzed correctly, can also be valuable planning assets for service improvements **(He et al., 2013)**. Despite their importance, many companies do not take full advantage of these datasets to turn them into actionable insights. Our project proposes to overcome this by applying natural language processing (NLP) to analyze review data gathered from Trustpilot on GWR passengers from 2019 to 2024. This analysis is important in terms of uncovering service pain points and trends related to customer satisfaction, and especially important to aid the UK transport sector in recovering from the COVID-19 pandemic.

To understand customer sentiment and classify opinions, we apply sentiment analysis methods including both lexicon-based and machine learning approaches. VADER (Valence Aware Dictionary and sentiment Reasoner), a rule-based model specifically designed for social media text, is used to compute sentiment polarity scores **(Hutto and Gilbert, 2014)**. While lexicon-based methods are fast and interpretable, they struggle with nuances such as sarcasm or domain-specific expressions. Therefore, we complement this with Naive Bayes and Logistic Regression classifiers trained on labeled review data to improve accuracy. Logistic Regression is optimized using gradient descent to enhance model convergence **(Jurafsky and Martin, 2021)**. These traditional classifiers have shown strong performance in text classification tasks, especially with short user reviews **(Ali et al., 2019)**. Beyond sentiment polarity, our project also uncovers the underlying themes in customer feedback using topic modeling. Latent Dirichlet Allocation (LDA) is used as a probabilistic model to detect co-occurring terms and assign topics, offering interpretable clusters of recurring issues such as “ticket refunds” or “train delays” **(Blei et al., 2003)**. We also implement BERTopic, a more modern, transformer-based approach that uses document embeddings and

clustering to generate semantically rich topics **(Grootendorst, 2022)**. The comparison between LDA and BERTopic allows us to evaluate not just which topics emerge, but how coherent and distinct they are. We further validate topic quality using coherence scores and visualize topic distances to assess overlap. A key part of our methodology is advanced text preprocessing: removing stopwords, lemmatization, constructing bigrams/trigrams, and filtering domain-specific terms such as "train", "GWR", and "service" that dominate all reviews without offering thematic distinction. This careful cleaning enhances both sentiment and topic modeling performance and addresses a common limitation in prior literature **(Wang et al., 2022)**. We also account for imbalanced data issues by exploring class weighting in our classifiers.

In conclusion, our study illustrates how using multiple NLP methods can offer a holistic view of public perceptions of sentiment with regard to service-related concerns. By using analysis of actual customer reviews with sentiment analysis and topic modeling, this project generates recommendations for how to utilize sentiment analysis for service design, service communication and crisis management. The spectrum of both traditional and modern NLP techniques places this study as a practical model for other service-based organizations wishing to triangulate structured information from unstructured data.

Data collection and Preprocessing:

This project used web scraping techniques to collect customer reviews **(2115 reviews**) of Greate Western Railway (GWR) for sentiment classification and topic modelling from Trustpilot. The aim was to extract user-generated content that provides insights such as customer satisfaction, pain points, and service feedback. Trustpilot, a popular online review platform, utilizes JavaScript-enhanced front-end architecture to render content dynamically. However, the underlying review content and metadata can still be reached using the static HTML structure through python-based tools.

The reviews were extracted using the requests and **BeautifulSoup** libraries. requests were employed to send HTTP GET requests to Trustpilot's paginated review pages, while **BeautifulSoup** parsed the HTML content to extract specific elements such as username, rating, title, review text, and date of experience. To mimic typical browser behavior and avoid IP blocking,

custom headers (including User-Agent) were used, along with randomized sleep intervals between requests to prevent detection by anti-bot systems. This iterative page-by-page extraction process ensured comprehensive retrieval of publicly available reviews from the GWR Trustpilot page. When the scraping was completed, the review data was exported as a CSV file preserving the natural state of the data in a transparent, accessible, and replicable format. CSV means transparency and accessibility in the future to process even further or assess the legitimizing of the dataset without the need to re-scrape. In addition, it was easy to load into any subsequent step of natural language processing or machine learning. The first thing that happened to the dataset after collection was preprocessing of the text data, as was necessary. Proceeding was done according to natural language processing best practices. For instance, everything was made lower case since the computer had to understand that "Train" and "train" are the same. In addition, special characters, digits, punctuation, and hyperlinks were deleted via regular expressions so that only letters remained for proper semantic analysis.

Stopwords—common words such as “the,” “and,” and “is”—were removed using NLTK’s built-in stopword list. These words, while frequent, contribute little to the meaning of a sentence and can introduce noise in sentiment and topic models. A custom stopword list was also developed to exclude frequently occurring but domain-specific non-informative words such as “gwr,” “get,” “would,” and “still.” This ensured that models focused on more contextually significant terms. Additionally, a tokenization process ensued to take every review and separate it word by word. Following this, lemmatization occurred utilizing **NLTK's WordNet lemmatizer** to better standardize words to their base or root form (i.e., "delays" became "delay," "better" became "good"), which reduced the vocabulary and created a better semantic relationship between words. Following lemmatization, **bigrams and trigrams** were established for critical two or three-word phrases that needed to stay together throughout the process (i.e., "train station," "customer service"). Finally, Term Frequency-Inverse Document Frequency **(TF-IDF)** vectorization was completed to convert the lemmatized and tokenized word features into numerical feature vectors for machine learning use. TF-IDF serves as a weight evaluator for rare but significant words in a review to promote greater accuracy within classification and topic modeling efforts. Therefore, this comprehensive preprocessing of data was necessary to establish a quality foundation for

sentiment classification algorithms (**Naive Bayes and Logistic Regression**) and topic modeling efforts (**LDA and BERTopic**).

Methodology and Analysis:

VADER sentiment analysis

This analysis uses VADER model on GWR's Trustpilot reviews and shows polarity in customer feedback. As shown in Figure 1, the histogram displays many reviews clustered around strong positive and negative VADER sentiment scores, with fewer reviews showing neutral sentiment. From the total dataset, **1008** reviews are marked negative**, 997** as positive and only **110** are flagged as neutral showing that the customers have strong polarized experiences. figure 2 shows a bar chart comparing the sentiment scores by start rating further supporting the reliability of VADER, there is a consistent increase in average sentiment from 1 star (**-0.28**) to 5-start review (**+0.71**) suggesting that there is strong alignment between textual sentiment and numerical ratings.

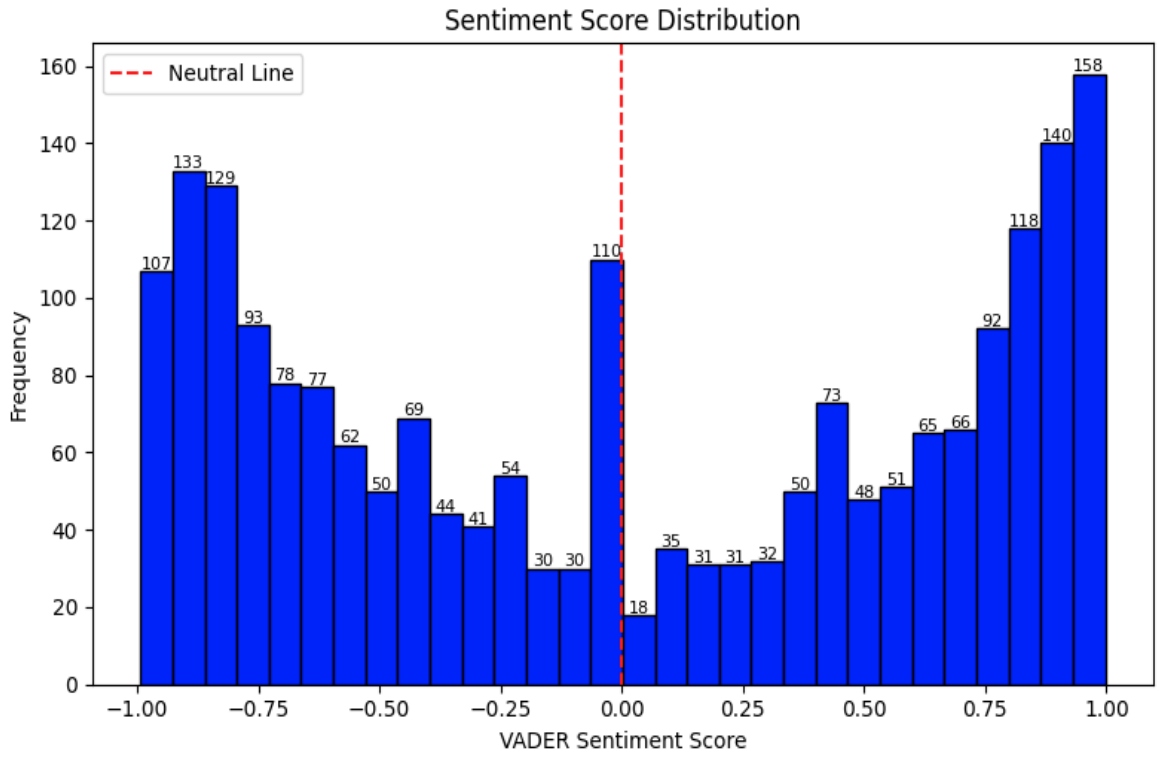


Figure 1: Histogram of sentiment score distribution

Figure 2: Bar chart comparing sentiment score by rating

Figure 3 shows positive and negative word cloud, highlighting the recurring themes, Positive reviews often mention words such as **"helpful," "staff," "refund," and "ticket,**", showing that the customers are satisfied with their service and resolution. While the negative review shows words such as **"delay," "cancelled," "refund," and "poor,"** indicating widespread frustration with punctuality and service disruptions.

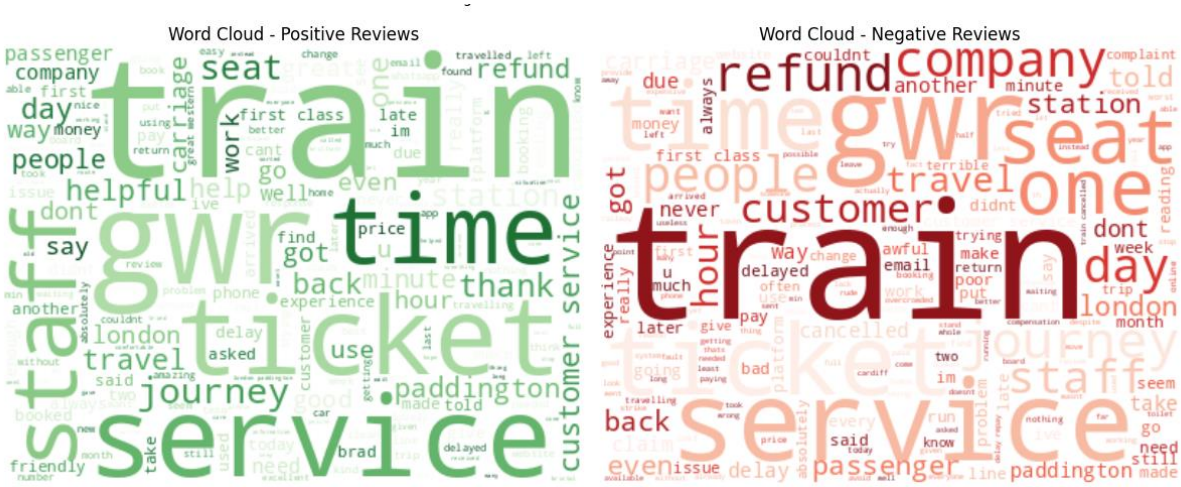
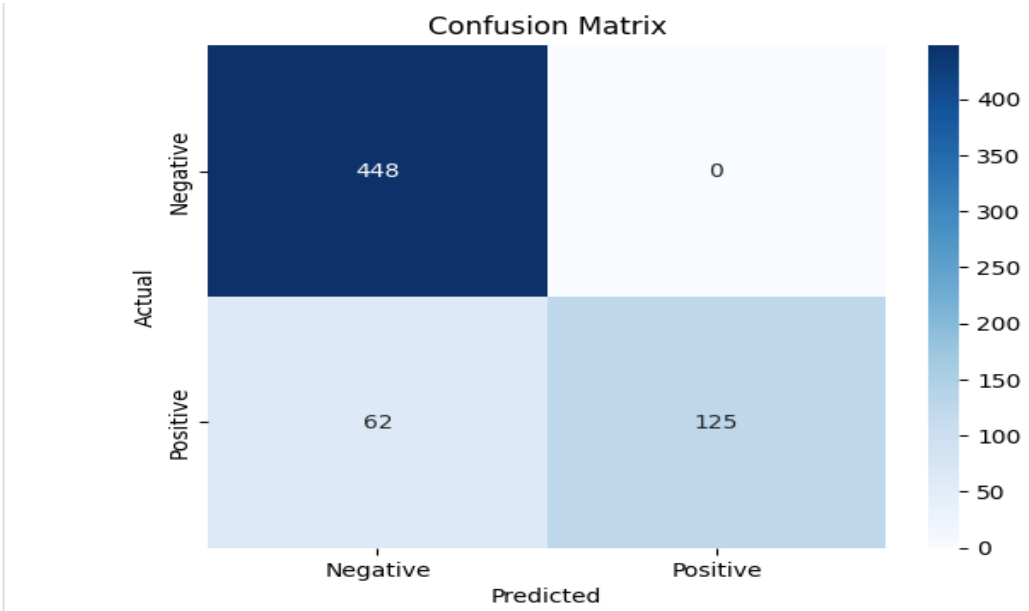


Figure 3: Positive and negative reviews word cloud

Naive bayes Classifier

Implemented Naïve Bayes classification model to predict review sentiment (Positive vs. Negative) based on textual content from customer reviews collected from Trustpilot. After preprocessing the text using lemmatization, stopword removal, and TF-IDF vectorization (with unigrams, bigrams, and trigrams), the model was trained on **70%** of the data and tested on the remaining **30%**. Since there are **1538** negative reviews and **577** positive reviews, class weights were applied to address the class imbalance and to avoid bias.

As shown in Figure 4, the confusion matrix reveals the model's ability to distinguish between the two sentiment classes accurately. Of the **448** actual negative reviews in the test set, all were correctly classified, resulting in a recall of 1.00 for the Negative class. However, while **125** were correctly classified for Positive reviews, **62** were misclassified as Negative, indicating a recall of **0.67** for the Positive class.



Classification Report:

	precision	recall	f1-score	support
Negative	0.88	1.00	0.94	448
Positive	1.00	0.67	0.80	187
accuracy			0.90	635
macro avg	0.94	0.83	0.87	635
weighted avg	0.91	0.90	0.90	635

Figure 4: Confusion matrix and classification report for naive bayes classification model

The model's overall performance is good, with **90%** accuracy. The primary objective of the model is to evaluate customer satisfaction towards GWR services; the model's high accuracy in identifying negative reviews is specifically valuable as it ensures that customer complaints and dissatisfaction are detected allowing to take actionable business decisions. However, the comparatively lower recall for positive reviews suggests that some satisfied customers may be incorrectly classified, which could underrepresent positive sentiment in strategic analysis.

Logistic Classifier

The logistic regression model using gradient descent was used to classify sentiments through customer feedback. As shown in figure 5, Various learning rates were tested to understand convergence behaviour. It was observed that **$\alpha = 0.1$** provided the most efficient performance, achieving rapid cost reduction without overshooting or instability. Meanwhile, higher rates such as $\alpha = 1.0$ caused oscillations, while smaller values like $\alpha = 0.001$ resulted in slow convergence. This confirms that 0.1 is a suitable learning rate for this task.

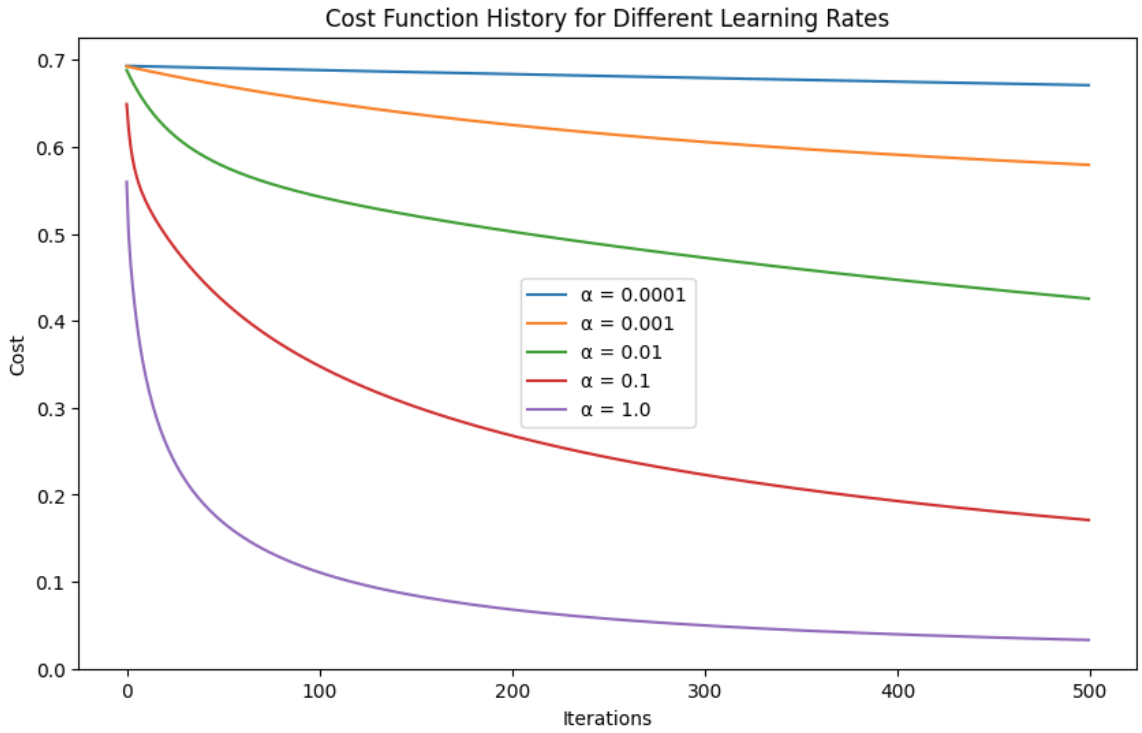
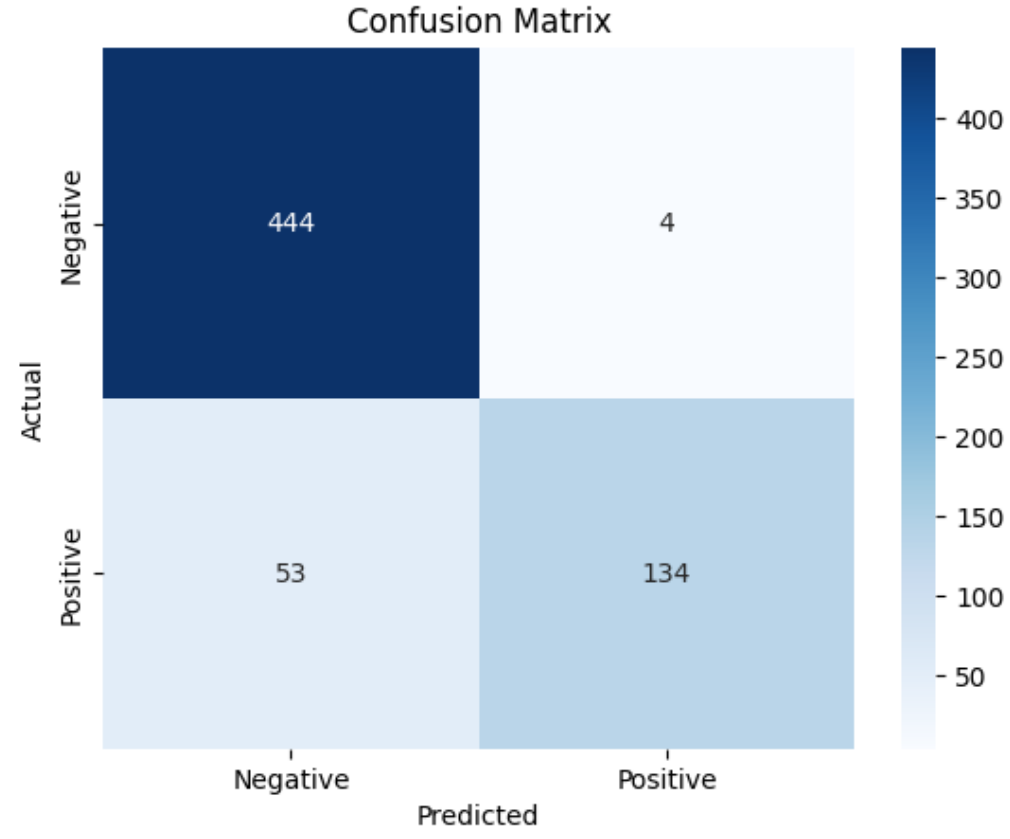


Figure 5: Graph of cost function for different learning rates

The model's performance was evaluated through a confusion matrix and classification report as shown in figure 6. It achieved an **overall accuracy of 91%**, correctly identifying 444 out of 448 negative reviews and 134 out of 187 positive ones. While misclassification of positive reviews (false negatives) was more frequent, the model maintained high reliability in detecting negative sentiment—an essential requirement for complaint monitoring.



Classification Report:

	precision	recall	f1-score	support
Negative	0.89	0.99	0.94	448
Positive	0.97	0.72	0.82	187
accuracy			0.91	635
macro avg	0.93	0.85	0.88	635
weighted avg	0.92	0.91	0.91	635

Figure 6: Confusion matrix and classification report for logistic classification model

The outcomes indicate that the classifier is particularly capable of detecting negative reviews, with less sensitivity towards positive sentiment. This is suitable for services where negative sentiment detection is more important for managing a business's reputation. Also confirming the overall competence of the model to learn steadily and effectively over iterations, are the "cost function" plots. Overall, the logistic regression model with gradient descent, and tuned learning rate, provided an interpretable and useful model for sentiment analysis to inform broader business insights from user-generated content.

Model comparison (Naive bayes vs Logistic classifier)

To assess the effectiveness of sentiment classification on customer reviews, both **Naïve Bayes** and **Logistic Regression** models were implemented and evaluated using standard performance metrics. The classification reports for each model were examined across key indicators such as precision, recall, and overall accuracy.

Metric	Model	Negative Precision	Negative Recall	Positive Precision	Positive Recall	Accuracy
Naïve Bayes	MultinomialNB	0.88	1	1	0.67	0.9
Logistic Regression	Gradient Descent Model	0.89	0.99	0.97	0.72	0.91

Table 1: Comparison of Performance between Naïve Bayes and Logistic Regression Models

According to the results, both models have good accuracy for identifying negative sentiment, with recall values near 1.00. However, when classifying positive reviews, logistic regression had better recall than Naïve Bayes as it had a higher sensitivity to identifying the positive instances (0.72 vs. 0.67). Moreover, logistic regression had good precision (0.97) and has the highest overall accuracy of 91%, just above Naïve Bayes' accuracy level (90%). The cost function history further validates the learning effectiveness of the logistic model, with the curve displaying a stable and smooth decline over 1000 iterations. Experimentation with various learning rates provided the best trade-off between convergence speed and stability.

In conclusion, logistic regression came out as the strongest and most balanced model for sentiment classification in this study. Its ability to robustly handle class imbalance and to properly tune weights using gradient descent made it much more appropriate for real-world business decision-making scenarios, where high precision and recall is very important.

Topic modelling – LDA

LDA was applied as a topic modelling approach to understand recurring patterns and thematic clusters within the customer reviews. LDA functions assuming that documents are mixtures of topics, where each topic is distributed over words. Its use helps businesses to extract key insights about user experiences, issues, and service expectations from textual data. Prior to applying LDA, the data underwent text processing to improve the model coherence and reduce semantic noise, such as the removal of custom stopwords and transformed tokens into a weighted numerical representation using TF-IDF (Term Frequency-Inverse Document Frequency), which emphasizes important and distinctive words.

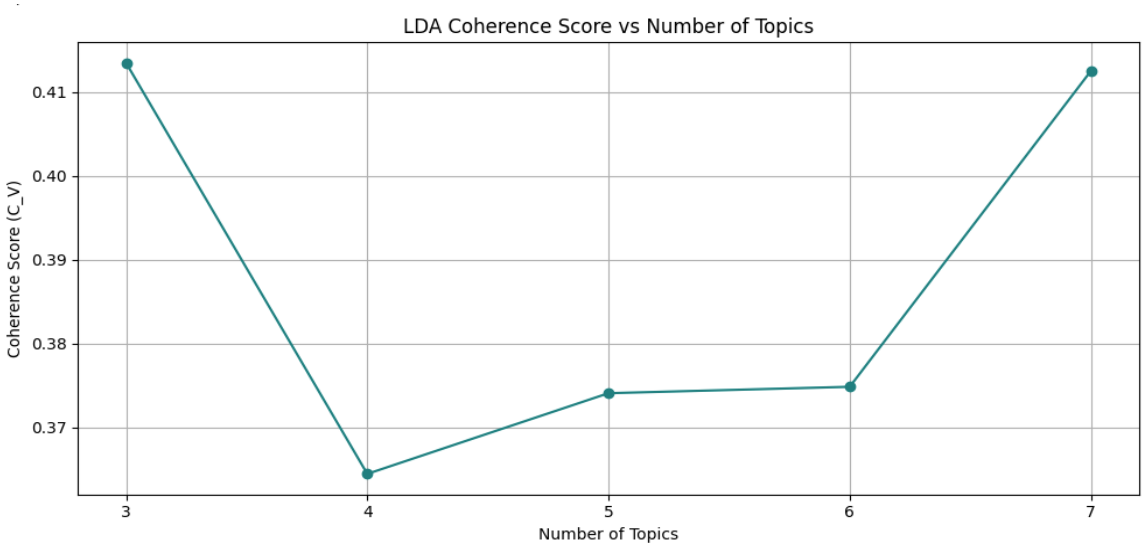


Figure 7: Graph of LDA coherence score vs number of topics

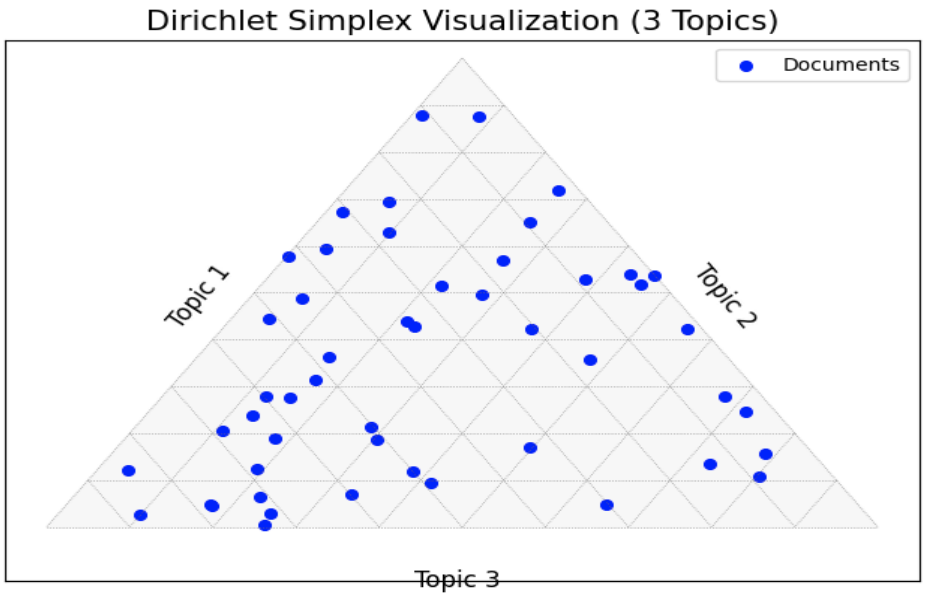


Figure 8: Dirichlet simplex algorithm for the three topics

To determine the optimal number of topics, as shown in figure 7, model was tested with topic count ranging from 3 to 7 and selected 3 since it had the highest coherence score, which measures the semantic similarity among the top topic words. The highest coherence score (C_V = 0.4224) was obtained for **three topics**, making it the most semantically coherent configuration for interpretation.

The table below outlines the final three topics generated by the LDA model, including their top keywords and interpreted themes:

Topic	Top Keywords	Interpretation
Topic 1	September, utterly, high, shocking, service, using, bother, seem, reservation, seat	Strong emotional dissatisfaction, likely related to customer service or incidents that shocked or upset users. Suggests severe disruptions or complaints.
Topic 2	website, refund, ticket, app, worst, perfect, number, page, assistant, week	Frustration with digital services—website usability, refund delays, app failures. Suggests system inefficiencies in online platforms and lack of support responsiveness.
Topic 3	train, time, ticket, journey, seat, service, people, minute, delay, hour	Operational and punctuality concerns—delays, time management, seat allocation. Captures logistical issues affecting passenger journeys.

Table 2: Result of LDA-Derived Topics with Top Keywords and Interpretation

BERTopic

The BERTopic model produced three distinctive and interpretable themes from the reviews with a coherence score of **0.47**. The topics show strong separation and internal consistency. As shown in the table below, the **first topic** primarily relates to core **service experiences** such as seat availability, travel time, and interactions with staff—indicative of recurring operational feedback. The **second** theme revolves around **bike reservation services**, suggesting either a positive or negative sentiment compared to transportation for those with bikes. The inclusion of "bike space" and "reservation" as keywords here shows how powerful BERTopic is at getting into the specifics. The **third** theme is sentiment around **customer service**, involving employees and thank yous sent out over Twitter, which implies that digital customer support helps bolster reputation.

Topic	Top Keywords	Interpretation
Topic 1	September, utterly, high, shocking, service, using, bother, seem, reservation, seat	Strong emotional dissatisfaction, likely related to customer service or incidents that shocked or upset users. Suggests severe disruptions or complaints.
Topic 2	website, refund, ticket, app, worst, perfect, number, page, assistant, week	Frustration with digital services—website usability, refund delays, app failures. Suggests system inefficiencies in online platforms and lack of support responsiveness.
Topic 3	train, time, ticket, journey, seat, service, people, minute, delay, hour	Operational and punctuality concerns—delays, time management, seat allocation. Captures logistical issues affecting passenger journeys.

Table 3: Result of BERTopic-Derived Topics with Top Keywords and Interpretation

This analysis reinforces BERTopic's utility in recognizing nuanced, context-rich feedback when supported by domain-specific stopword removal and n-gram modeling. The findings can assist GWR in prioritizing improvement areas (e.g., punctuality and seating) while maintaining strengths in service personalization and digital support. The ability to extract named entities (e.g., "Michaela," "Celia") also reveals how personal interactions influence review sentiment, making BERTopic an effective tool for reputation monitoring.

Model comparison (LDA vs BERTopic)

Both **Latent Dirichlet Allocation (LDA)** and **BERTopic** were applied to extract underlying themes from GWR's Trustpilot reviews. While LDA provided interpretable topics with a coherence score of **0.42**, the model tended to group generic terms (e.g., "train," "service," "ticket") across multiple topics, limiting distinctiveness. In contrast, **BERTopic** produced a higher coherence score of **0.47** and demonstrated **greater topic separation**, identifying more granular and context-specific themes—such as feedback on **bike reservations**, **journey experience**, and **individual staff interactions**. BERTopic's use of contextual embeddings and n-gram support enabled it to detect subtle differences in customer concerns that LDA's bag-of-words approach could not capture. Additionally, BERTopic's ability to associate topics with named entities and digital interactions (e.g., "Twitter," staff names) makes it more aligned with modern customer service environments.

Taking into account both the nature of the data set and the intention behind the project in attempting to derive useful information for business improvement, the second topic modeling approach, **BERTopic**, is the better model. It produces more relevant and nuanced topics that translate to greater service improvements and more effective customer interactions.

Managerial recommendations:

- **Implement higher occupancy and reservations**:
  Many complaints involve overcrowded trains and not implementing reservations on certain routes during peak hours
- **Compensate with tickets and delays:** The digital refund portal is inordinately frustrating, and many suggestions reported the inability to reach someone via phone to troubleshoot delays—let alone receive a refund—when the issue was never communicated appropriately.
- **Improve with delays and cancellations:** The more trains are on time, the better the anticipated experience and trust in timetables.
- I**mprove Digital Experience:** Optimize website and mobile app usability for smoother ticket booking, refund claims, and customer support.

- **Leverage Staff Excellence:** Recognise and retain high-performing staff members frequently praised in reviews and ensure consistent training across teams.
- **Maintain and Expand Bike Reservation Services:** Continue supporting and promoting this well-received service to retain cyclists and eco-conscious travelers.
- **Strengthen Social Media Engagement:** Build on positive feedback around WhatsApp and Twitter responsiveness by expanding real-time support and issue resolution.
- **Use Feedback for Continuous Improvement:** Regularly monitor online reviews to identify emerging issues and implement data-driven service improvements.

Conclusion and limitations:

This project aimed to explore customer sentiment about Great Western Railway (GWR) through the use of various Natural Language Processing (NLP) techniques on Trustpilot reviews. By implementing sentiment analysis through VADER, Naïve Bayes, and Logistic Regression, customer sentiment was classified and revealed that the vast majority of reviews are negative—many citing delays, difficulties in obtaining refunds, and standing room only trains. By implementing topic modelling through LDA and BERTopic, the most popular themes included bike reservations, train staff interactions, and the effectiveness of customer service. While both LDA and BERTopic generated successful topics, BERTopic generated more on point with a higher topic variety, making it the most suitable for generating actionable insights for this project. The two types of analyses are merged to create a basis for managerial recommendations for proper service improvements and future proactive efforts.

Limitations

Despite its effectiveness, the study faced multiple limitations. First, the dataset was limited to reviews posted on Trustpilot, which may not represent the full spectrum of customer feedback. Second, the class imbalance in sentiment categories, especially neutral reviews; reduced model performance in some cases. Additionally, sarcasm, slang, and domain-specific language posed challenges to sentiment classification models. Finally, topic interpretability in LDA and BERTopic was partly subjective and depended on manual review, which can introduce bias. Future work could benefit from larger datasets, multi-platform review integration, and more advanced contextual models like BERT-based classifiers.

References

Ali, A., Siddiqui, S. and Adnan, M., 2019. *Performance analysis of machine learning algorithms for sentiment classification*. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 5(1), pp.19–27.

Blei, D.M., Ng, A.Y. and Jordan, M.I., 2003. Latent Dirichlet Allocation. Journal of Machine Learning Research, 3(Jan), pp.993–1022.

Burnett, N., Stewart, I., Hinson, S., Tyers, R., Hutton, G. and Malik, X. (2024). *The UK's Plans and Progress to Reach Net Zero by 2050*. [online] House of Commons Library. Available at: https://commonslibrary.parliament.uk/research-briefings/cbp-9888/. [Accessed 30 Mar. 2025].

Discussion paper on rail industry productivity. (n.d.). Available at: https://www.orr.gov.uk/sites/default/files/2024-04/rail-industry-productivity-report-april-2024.pdf. [Accessed 30 Mar. 2025].

Grootendorst, M. (2022). *BERTopic: Neural topic modeling with class-based TF-IDF*. Available at: https://maartengr.github.io/BERTopic/ [Accessed 30 Mar. 2025]

Güner, S., Taskin, K., Ibrahim, C.H. and Aydemir, E. (2024). Service Quality in Rail Systems: Listen to the Voice of Social Media. *Transportation Research Record: Journal of the Transportation Research Board*, [online] 2678(6). Available at: https://trid.trb.org/View/2264414 [Accessed 30 Mar. 2025].

He, W., Zha, S., & Li, L. (2013). Social media competitive analysis and text mining: A case study in the pizza industry. International Journal of Information Management, 33(3), 464-472.

Hutto, C. J., & Gilbert, E. (2014). VADER: A parsimonious rule-based model for sentiment analysis of social media text. In Eighth International Conference on Weblogs and Social Media (ICWSM-14).

Jurafsky, D. and Martin, J.H. (2021). *Speech and Language Processing* (3rd ed. draft). Stanford University. Available at: https://web.stanford.edu/~jurafsky/slp3/ [Accessed 30 Mar. 2025].

Shah, Z. and Rai, S. (2022). *A Research Paper on the Effects of Customer Feedback on Business*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/361916247_A_Research_Paper_on_the_Effects_of_Customer_Feedback_on_Business. [Accessed 30 Mar. 2025].

Wang, S., Paul, M.J. and Dredze, M., 2022. Social media and public health: Detecting COVID-19 symptoms from tweets. PLoS ONE, 17(4), p.e0264260.

Appendix

The following link has source code for all the analysis and visualizations that have been included in this report

Google collab:
https://colab.research.google.com/drive/1kJnJzK_xJEqpKrDWJ3gHGteczfxmBv_C?usp=sharing

GWR reviews (Trustpilot) – Data:

https://drive.google.com/file/d/1yILzv4c7voditPFhTUnbg93VljaadfD7A/view?usp=sharing

Code (Text format):

Web scraping (GWR reviews from Trustpilot)

```
import requests

from bs4 import BeautifulSoup
import pandas as pd
import time
import random
import re

def scrape_trustpilot_reviews(company_name, num_pages=1):
    """
    Scrape reviews from Trustpilot for a specific company, including Date
of Experience.

    Args:
        company_name (str): The company name as it appears in Trustpilot
URL
        num_pages (int): Number of pages to scrape

    Returns:
        DataFrame: A pandas DataFrame containing the scraped reviews
    """
    reviews_data = []

    # Define headers to mimic a real browser
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36',
        'Accept-Language': 'en-US,en;q=0.9',
```

```
        'Accept':
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
8',
        'Referer': 'https://www.google.com/'
    }

    for page in range(1, num_pages + 1):
        url =
f"https://www.trustpilot.com/review/{company_name}?page={page}"
        print(f"Scraping page {page} of {num_pages}: {url}")

        try:
            response = requests.get(url, headers=headers)

            if response.status_code == 200:
                soup = BeautifulSoup(response.text, 'html.parser')

                # Debugging: Print a sample of HTML
                print(f"Response received. HTML sample:
{response.text[:500]}...")

                # Find all review containers
                review_containers = soup.find_all('article', {'class':
re.compile('paper_paper.*')}) or \
                                    soup.find_all('div', {'data-service-
review-card-paper': True}) or \
                                    soup.find_all('div', {'class':
re.compile('styles_cardWrapper.*')})

                print(f"Found {len(review_containers)} review containers")

                for container in review_containers:
                    try:
                        # Extract username
                        username_element = container.find('span',
{'class': re.compile('typography_heading-.*')}) or \
                                           container.find('span', {'class':
re.compile('styles_consumerName.*')}) or \
                                           container.find('div', {'class':
re.compile('styles_consumerDetails.*')})
                        username = username_element.text.strip() if
username_element else "Anonymous"

                        # Extract rating
                        rating = "N/A"
```

```
                        rating_div = container.find('div', {'data-service-
review-rating': True})
                        if rating_div:
                            rating_value = rating_div.get('data-service-
review-rating')
                            if rating_value:
                                rating = rating_value

                        # If rating not found, try to find it in the img
alt text
                        if rating == "N/A":
                            rating_img = container.find('img', {'alt':
re.compile('.*star.*')})
                            if rating_img and 'alt' in rating_img.attrs:
                                rating = rating_img['alt'].split()[0]

                        # Extract review title
                        title_element = container.find('h2', {'class':
re.compile('styles_reviewTitle.*')}) or \
                                        container.find('h2', {'class':
re.compile('typography_heading-.*')})
                        title = title_element.text.strip() if
title_element else "No Title"

                        # Extract review content
                        content_element = container.find('p', {'class':
re.compile('styles_reviewContent.*')}) or \
                                          container.find('p', {'class':
re.compile('typography_body-.*')})
                        content = content_element.text.strip() if
content_element else "No Content"

                        # Extract Date of Experience
                        date_experience_element = container.find('time',
{'class': re.compile('typography_body-.*')}) or \
                                                  container.find('p',
{'class': re.compile('styles_reviewDate.*')})
                        date_experience =
date_experience_element.text.strip() if date_experience_element else "No
Date"

                        # Debugging output
                        print(f"Extracted review - Username: {username},
Rating: {rating}, Date: {date_experience}, Title: {title[:20]}...")
```

```
                        # Append to list
                        reviews_data.append({
                            'username': username,
                            'rating': rating,
                            'title': title,
                            'content': content,
                            'date_of_experience': date_experience
                        })

                    except Exception as e:
                        print(f"Error extracting review data: {e}")

            else:
                print(f"Failed to fetch page {page}. Status code:
{response.status_code}")

        except Exception as e:
            print(f"Error fetching page {page}: {e}")

        # Add a random delay to avoid being blocked
        if page < num_pages:
            delay = random.uniform(3, 7)
            print(f"Waiting for {delay:.2f} seconds before next
request...")
            time.sleep(delay)

    # Convert the data to a DataFrame
    df = pd.DataFrame(reviews_data)
    return df

def save_to_csv(df, filename="trustpilot_reviews.csv"):
    """Save the DataFrame to a CSV file"""
    if df.empty:
        print("Warning: DataFrame is empty! No data to save.")
    else:
        df.to_csv(filename, index=False, encoding='utf-8')
        print(f"Data saved to {filename} with {len(df)} reviews.")

def main():
    print(f"Starting to scrape reviews for
'https://www.trustpilot.com/review/www.gwr.com'...")
    df = scrape_trustpilot_reviews("www.gwr.com", 106)

    if df.empty:
```

```
        print("No reviews were scraped. Check the HTML structure and try
again.")
    else:
        print(f"Scraped {len(df)} reviews.")
        save_to_csv(df, "gwr_reviews.csv")

if __name__ == "__main__":
    main()
```

Vader sentiment Analysis:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
import re
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from wordcloud import WordCloud
from textblob import TextBlob

# Download NLTK data
nltk.download('vader_lexicon')
nltk.download('stopwords')

# Initialize tools
sia = SentimentIntensityAnalyzer()
stop_words = set(stopwords.words("english"))

# Text cleaning function
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'http\S+|www\S+', '', text)
    text = re.sub(r'[^a-z\s]', '', text)
    tokens = [word for word in text.split() if word not in stop_words]
    return " ".join(tokens)

# Load dataset
file_path = "/content/GWR with date.csv"
df = pd.read_csv(file_path)
df['content'] = df['content'].fillna('').astype(str)

# Preprocess and analyze sentiment
df['cleaned_content'] = df['content'].apply(preprocess_text)
```

```
df['vader_score'] = df['cleaned_content'].apply(lambda x:
sia.polarity_scores(x)['compound'])
df['textblob_polarity'] = df['cleaned_content'].apply(lambda x:
TextBlob(x).sentiment.polarity)
df['sentiment'] = df['vader_score'].apply(lambda x: 'Positive' if x > 0.05
else ('Negative' if x < -0.05 else 'Neutral'))

# --- Visualization 1: Sentiment Score Distribution with Value Labels ---
plt.figure(figsize=(8, 5))
counts, bins, patches = plt.hist(df['vader_score'], bins=30, color='blue',
edgecolor='black')
plt.axvline(0, color='red', linestyle='dashed', label='Neutral Line')
for i in range(len(patches)):
    height = counts[i]
    if height > 0:
        plt.text(patches[i].get_x() + patches[i].get_width()/2, height +
0.5,
                 f'{int(height)}', ha='center', fontsize=8)
plt.title("Sentiment Score Distribution")
plt.xlabel("VADER Sentiment Score")
plt.ylabel("Frequency")
plt.legend()
plt.tight_layout()
plt.show()

# --- Visualization 2: Sentiment vs. Ratings with Labels ---
if 'rating' in df.columns:
    df['rating'] = pd.to_numeric(df['rating'], errors='coerce')
    rating_sentiment =
df.groupby('rating')['vader_score'].mean().reset_index()

    plt.figure(figsize=(8, 5))
    ax = sns.barplot(x='rating', y='vader_score', data=rating_sentiment,
palette="coolwarm")

    # Add labels
    for p in ax.patches:
        height = p.get_height()
        ax.annotate(f'{height:.2f}', (p.get_x() + p.get_width() / 2,
height),
                    ha='center', va='bottom', fontsize=9, color='black',
fontweight='bold')

    plt.title("Average Sentiment Score by Rating")
    plt.xlabel("Review Rating")
```

```
    plt.ylabel("Average Sentiment Score")
    plt.tight_layout()
    plt.show()

# --- Visualization 3: Word Clouds ---
positive_text = " ".join(df[df['sentiment'] ==
'Positive']['cleaned_content'].dropna())
negative_text = " ".join(df[df['sentiment'] ==
'Negative']['cleaned_content'].dropna())

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
wordcloud_pos = WordCloud(width=400, height=300, background_color='white',
colormap='Greens').generate(positive_text)
plt.imshow(wordcloud_pos, interpolation="bilinear")
plt.axis("off")
plt.title("Word Cloud - Positive Reviews")

plt.subplot(1, 2, 2)
wordcloud_neg = WordCloud(width=400, height=300, background_color='white',
colormap='Reds').generate(negative_text)
plt.imshow(wordcloud_neg, interpolation="bilinear")
plt.axis("off")
plt.title("Word Cloud - Negative Reviews")
plt.tight_layout()
plt.show()

# Print sentiment counts
print("Sentiment Distribution:")
print(df['sentiment'].value_counts())
```

Naïve Bayes Classification:

```
import pandas as pd
import numpy as np
import re
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
```

```
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Download NLTK resources
nltk.download("stopwords")
nltk.download("wordnet")

# Load the dataset
df = pd.read_csv("/content/GWR with date.csv")

# Drop missing values in 'content' and 'rating'
df = df.dropna(subset=['content', 'rating'])

# Map ratings to sentiment categories
def map_sentiment(rating):
    if rating >= 3:
        return "Positive"
    else:
        return "Negative"

df["sentiment"] = df["rating"].apply(map_sentiment)

# Text Cleaning Function
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words("english"))

def clean_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\W+', ' ', text)  # Remove special characters
    words = text.split()
    words = [word for word in words if word not in stop_words]  # Remove
stopwords
    words = [lemmatizer.lemmatize(word) for word in words]  #
Lemmatization
    return " ".join(words)

# Apply text cleaning
df["cleaned_content"] = df["content"].apply(clean_text)

# Define features and labels
X = df["cleaned_content"]
y = df["sentiment"]

# Convert labels to numerical format
```

```
# Updated label_mapping to only include 'Negative' and 'Positive'
label_mapping = {"Negative": 0, "Positive": 1}
y = y.map(label_mapping)

# TF-IDF vectorization with bigrams & trigrams
vectorizer = TfidfVectorizer(ngram_range=(1, 3), max_features=5000)  #
Limit features to avoid overfitting
X_tfidf = vectorizer.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y,
test_size=0.3, random_state=42, stratify=y)

# Compute class weights to balance the dataset
class_weights = compute_class_weight("balanced",
classes=np.unique(y_train), y=y_train)
class_weight_dict = {i: class_weights[i] for i in
range(len(class_weights))}

# Train Naïve Bayes Model
model = MultinomialNB()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Confusion Matrix Visualization
conf_matrix = confusion_matrix(y_test, y_pred)
# Updated heatmap to use label_mapping.keys() for labels
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=label_mapping.keys(), yticklabels=label_mapping.keys())
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

# Classification Report
print("\nClassification Report:")
# Updated classification_report to use label_mapping.keys() for
target_names
print(classification_report(y_test, y_pred,
target_names=label_mapping.keys()))
```

Logistic Classifier with Gradient Decent:

learning rate:

```
import numpy as np
import pandas as pd
import re
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix, classification_report

# Download stopwords if not already present
nltk.download("stopwords")
stop_words = set(stopwords.words("english"))

# Load the dataset
df = pd.read_csv("/content/GWR with date.csv")

# Ensure no missing values in 'content' and 'rating'
df = df.dropna(subset=["content", "rating"])

# Map ratings to sentiment categories (3 and above is Positive, below 3 is
Negative)
df["sentiment"] = df["rating"].apply(lambda x: 1 if x >= 3 else 0)

# Function to clean text
def clean_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\d{10,}', '', text)  # Remove phone numbers (10+
digits)
    text = re.sub(r'\d+', '', text)  # Remove numbers
    text = re.sub(r'http\S+|www\S+', '', text)  # Remove URLs
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special characters
    text = ' '.join([word for word in text.split() if word not in
stop_words])  # Remove stopwords
    return text

# Apply text cleaning
df["cleaned_content"] = df["content"].apply(clean_text)

# Convert text to features using bag-of-words (unigrams & bigrams)
```

```
vectorizer = CountVectorizer(ngram_range=(1, 2))
X = vectorizer.fit_transform(df["cleaned_content"]).toarray()
y = df["sentiment"].values

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

# Logistic Regression Cost Function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def compute_cost(X, y, theta):
    m = len(y)
    predictions = sigmoid(np.dot(X, theta))
    cost = -(1/m) * np.sum(y * np.log(predictions) + (1 - y) * np.log(1 -
predictions))
    return cost

# Gradient Descent
def gradient_descent(X, y, theta, alpha, num_iters):
    m = len(y)
    cost_history = []

    for i in range(num_iters):
        predictions = sigmoid(np.dot(X, theta))
        gradient = (1/m) * np.dot(X.T, predictions - y)
        theta -= alpha * gradient
        cost_history.append(compute_cost(X, y, theta))

    return theta, cost_history

# Add intercept term to X (a column of ones)
X_train_with_intercept = np.c_[np.ones((X_train.shape[0], 1)), X_train]
X_test_with_intercept = np.c_[np.ones((X_test.shape[0], 1)), X_test]

# Initialize theta
initial_theta = np.zeros(X_train_with_intercept.shape[1])

# Different learning rates to test
alphas = [0.0001, 0.001, 0.01, 0.1, 1.0]
num_iters = 500

# Run gradient descent for different alpha values
plt.figure(figsize=(10, 6))
```

```
for alpha in alphas:
    theta = np.zeros(X_train_with_intercept.shape[1])  # Reset theta for
each alpha
    _, cost_history = gradient_descent(X_train_with_intercept, y_train,
theta, alpha, num_iters)
    plt.plot(range(num_iters), cost_history, label=f'α = {alpha}')

# Plot cost function history for different learning rates
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.title("Cost Function History for Different Learning Rates")
plt.legend()
plt.show()
```

Comparing learning rate 1.0 vs 0.1for overfitting/oscillations

```
import numpy as np
import matplotlib.pyplot as plt

# Simulated Cost Function (Logistic Regression or Linear Regression)
def compute_cost(theta, X, y):
    m = len(y)
    predictions = X.dot(theta)
    cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2)
    return cost

# Gradient Descent Function
def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)
    cost_history = []
    theta_history = [theta.copy()]

    for i in range(iterations):
        gradients = (1 / m) * X.T.dot(X.dot(theta) - y)
        theta -= alpha * gradients  # Parameter update
        cost = compute_cost(theta, X, y)  # Compute cost
        cost_history.append(cost)
        theta_history.append(theta.copy())

    return theta, cost_history, theta_history

# Generate Synthetic Data (Linear Regression Example)
np.random.seed(42)
m = 100  # Number of samples
X = 2 * np.random.rand(m, 1)
```

```
y = 4 + 3 * X + np.random.randn(m, 1)  # True relation: y = 4 + 3x + noise
X_b = np.c_[np.ones((m, 1)), X]  # Add bias term

# Initialize parameters
theta_init = np.random.randn(2, 1)

# Run gradient descent for alpha = 1.0
alpha = 1.0
iterations = 100
theta_final, cost_history, theta_history = gradient_descent(X_b, y,
theta_init, alpha, iterations)

# Plot cost function to check for oscillations
plt.figure(figsize=(8, 5))
plt.plot(range(iterations), cost_history, label=f"α = {alpha}",
color="purple")
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.title("Cost Function History (Checking for Overshooting/Oscillation)")
plt.legend()
plt.show()

# Plot parameter updates to check for instability
theta_0_vals = [t[0, 0] for t in theta_history]
theta_1_vals = [t[1, 0] for t in theta_history]

plt.figure(figsize=(8, 5))
plt.plot(range(iterations+1), theta_0_vals, label="Theta 0",
linestyle="dashed")
plt.plot(range(iterations+1), theta_1_vals, label="Theta 1",
linestyle="solid")
plt.xlabel("Iterations")
plt.ylabel("Parameter Value")
plt.title("Parameter Updates Over Iterations")
plt.legend()
plt.show()
```

Logistic model with gradient decent and cost function:

```
import numpy as np
import pandas as pd
import re
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix, classification_report

# Download stopwords if not already present
nltk.download("stopwords")
stop_words = set(stopwords.words("english"))

# Load the dataset
df = pd.read_csv("/content/GWR with date.csv")

# Ensure no missing values in 'content' and 'rating'
df = df.dropna(subset=["content", "rating"])

# Map ratings to sentiment categories (3 and above is Positive, below 3 is
Negative)
df["sentiment"] = df["rating"].apply(lambda x: 1 if x >= 3 else 0)

# Function to clean text
def clean_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\d{10,}', '', text)  # Remove phone numbers (10+
digits)
    text = re.sub(r'\d+', '', text)  # Remove numbers
    text = re.sub(r'http\S+|www\S+', '', text)  # Remove URLs
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special characters
    text = ' '.join([word for word in text.split() if word not in
stop_words])  # Remove stopwords
    return text

# Apply text cleaning
df["cleaned_content"] = df["content"].apply(clean_text)

# Convert text to features using bag-of-words (unigrams & bigrams)
vectorizer = CountVectorizer(ngram_range=(1, 2))
X = vectorizer.fit_transform(df["cleaned_content"]).toarray()
y = df["sentiment"].values

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42, stratify=y)

# Logistic Regression Cost Function
def sigmoid(z):
```

```
    return 1 / (1 + np.exp(-z))

def compute_cost(X, y, theta):
    m = len(y)
    predictions = sigmoid(np.dot(X, theta))
    cost = -(1/m) * np.sum(y * np.log(predictions) + (1 - y) * np.log(1 -
predictions))
    return cost

# Gradient Descent
def gradient_descent(X, y, theta, alpha, num_iters):
    m = len(y)
    cost_history = []

    for i in range(num_iters):
        predictions = sigmoid(np.dot(X, theta))
        gradient = (1/m) * np.dot(X.T, predictions - y)
        theta -= alpha * gradient
        cost_history.append(compute_cost(X, y, theta))

    return theta, cost_history

# Add intercept term to X (a column of ones)
X_train_with_intercept = np.c_[np.ones((X_train.shape[0], 1)), X_train]
X_test_with_intercept = np.c_[np.ones((X_test.shape[0], 1)), X_test]

# Initialize theta
initial_theta = np.zeros(X_train_with_intercept.shape[1])

# Set learning rate and number of iterations
alpha = 0.1
num_iters = 1000

# Run gradient descent
theta, cost_history = gradient_descent(X_train_with_intercept, y_train,
initial_theta, alpha, num_iters)

# Plot cost function history
plt.plot(range(num_iters), cost_history)
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.title("Cost Function History")
plt.show()

# Make predictions
```

```
y_pred_prob = sigmoid(np.dot(X_test_with_intercept, theta))
y_pred = [1 if prob >= 0.6 else 0 for prob in y_pred_prob]

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=["Negative", "Positive"], yticklabels=["Negative",
"Positive"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=["Negative",
"Positive"]))
```

LDA – Topic modelling

```
import pandas as pd
import numpy as np
import re
import nltk
import gensim
import gensim.corpora as corpora
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from gensim.models import Phrases, LdaModel, TfidfModel
from gensim.models.phrases import Phraser
from gensim.models import CoherenceModel
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Download required NLTK data
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')

# Load dataset
df = pd.read_csv("/content/GWR with date.csv")
df['content'] = df['content'].fillna('').astype(str)

# Define stopwords and add custom ones
stop_words = set(stopwords.words('english'))
```

```
custom_stopwords = {
    "gwr", "trains", "services", "get", "would", "even", "one", "told",
"say",
    "got", "still", "back", "week", "month",
    "really", "also", "much", "need",
"use","date_experience","getting","put","took","gave","half","extremely","
well","that's","could","given"
}
stop_words.update(custom_stopwords)

# Lemmatizer
lemmatizer = WordNetLemmatizer()

# Clean text
def clean_text(text):
    text = text.lower()
    text = re.sub(r"http\S+|www\S+", "", text)  # Remove links
    text = re.sub(r"[^a-zA-Z\s]", "", text)      # Remove special
characters
    return text

# Tokenizer
def tokenize(text):
    text = clean_text(text)
    tokens = text.split()
    return [word for word in tokens if word not in stop_words and
len(word) > 2]

# Apply tokenization
df['tokens'] = df['content'].apply(tokenize)

# Build bigram and trigram models
bigram = Phrases(df['tokens'], min_count=5, threshold=10)
trigram = Phrases(bigram[df['tokens']], threshold=10)
bigram_mod = Phraser(bigram)
trigram_mod = Phraser(trigram)

# Lemmatize + stopword filter
def process_tokens(tokens):
    tokens = bigram_mod[tokens]
    tokens = trigram_mod[tokens]
    return [
        lemmatizer.lemmatize(word)
        for word in tokens
        if word not in stop_words and len(word) > 2
```

```
    ]

# Apply token processing
df['processed_tokens'] = df['tokens'].apply(process_tokens)

# Filter out empty processed rows
df = df[df['processed_tokens'].map(len) > 0].reset_index(drop=True)

# Create dictionary and corpus
dictionary = corpora.Dictionary(df['processed_tokens'])
dictionary.filter_extremes(no_below=5, no_above=0.6)
corpus = [dictionary.doc2bow(tokens) for tokens in df['processed_tokens']]

# TF-IDF model
tfidf_model = TfidfModel(corpus)
corpus_tfidf = tfidf_model[corpus]

# Train LDA Model with 3 Topics (based on coherence tuning)
lda_model = LdaModel(
    corpus=corpus_tfidf,
    id2word=dictionary,
    num_topics=3,
    random_state=42,
    passes=20,
    iterations=200
)

# Display top words per topic
print("\n Top words per topic:\n")
for i, topic in lda_model.show_topics(num_topics=3, num_words=10,
formatted=False):
    top_words = [word for word, _ in topic]
    print(f"Topic #{i+1}: {' '.join(top_words)}")

# Plot word clouds
def plot_topic_wordcloud(lda_model, topic_id, num_words=20):
    words_probs = lda_model.show_topic(topic_id, topn=num_words)
    freqs = {word: prob for word, prob in words_probs}
    wc = WordCloud(width=600, height=400, background_color='white')
    wc.generate_from_frequencies(freqs)

    plt.figure(figsize=(6, 4))
    plt.imshow(wc, interpolation='bilinear')
    plt.axis("off")
    plt.title(f"Topic #{topic_id+1} Word Cloud")
```

```
    plt.show()

# Generate word clouds for each topic
for topic_id in range(3):
    plot_topic_wordcloud(lda_model, topic_id)

# Calculate Coherence Score
coherence_model = CoherenceModel(
    model=lda_model,
    texts=df['processed_tokens'],
    dictionary=dictionary,
    coherence='c_v'
)
coherence_score = coherence_model.get_coherence()
print(f"\n LDA Coherence Score (C_V): {coherence_score:.4f}")
print(df['processed_tokens'].sample(5))
```

Dirichlet Simplex visualization:

```
!pip install python-ternary

import numpy as np
import matplotlib.pyplot as plt
import ternary

# For demonstration, assume we have 3 topics.
# We generate a synthetic document-topic distribution using the Dirichlet
distribution.
# In your case, replace this with your actual document-topic distributions
from LDA.
np.random.seed(42)
num_docs = 50
doc_topic_distributions = np.random.dirichlet(alpha=[1, 1, 1],
size=num_docs)

# Create a ternary plot with scale=1 (each document's topic probabilities
sum to 1)
figure, tax = ternary.figure(scale=1.0)
tax.set_title("Dirichlet Simplex Visualization (3 Topics)", fontsize=15)

# Plot each document as a point in the ternary space.
points = doc_topic_distributions.tolist()
tax.scatter(points, marker='o', color='blue', label="Documents", s=30)

# Set custom corner labels using axis label functions.
```

```
tax.left_axis_label("Topic 1", fontsize=12)
tax.right_axis_label("Topic 2", fontsize=12)
tax.bottom_axis_label("Topic 3", fontsize=12)

# Additional formatting for clarity
tax.legend()
tax.gridlines(multiple=0.1, color="grey")
tax.clear_matplotlib_ticks()

plt.tight_layout()
plt.show()
```

BERTopic Model:

```
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from bertopic import BERTopic
from sklearn.feature_extraction.text import CountVectorizer

# Download required NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')

# Load dataset
df = pd.read_csv("/content/GWR with date.csv")
df['content'] = df['content'].fillna('').astype(str)

# NLTK stopwords + custom domain stopwords
stop_words = set(stopwords.words('english'))
custom_stopwords = {
    "gwr", "train", "trains", "service", "ticket", "get", "one", "still",
    "brad", "whatsapp", "viawhatsapp", "date", "experience", "date
experience",
    "august", "september", "october", "june", "january", "quick", "bike
space"
}
stop_words.update(custom_stopwords)

lemmatizer = WordNetLemmatizer()

# Clean and preprocess text
```

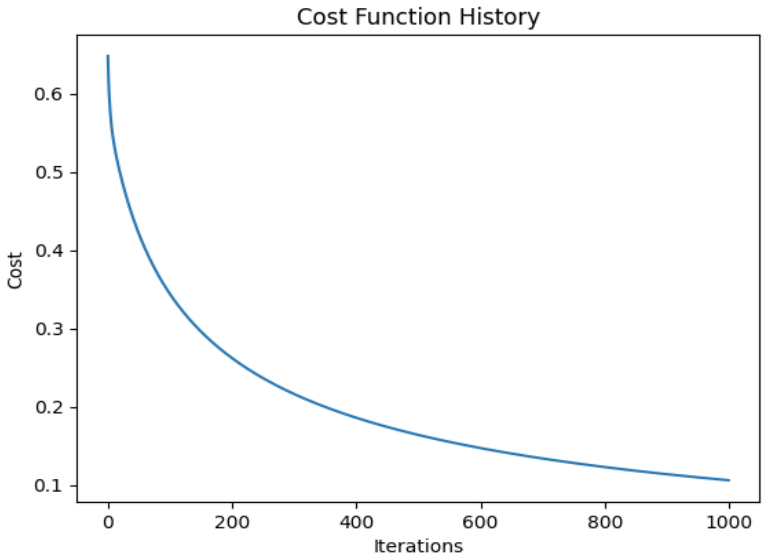
```
def clean_text(text):
    text = text.lower()
    text = re.sub(r"http\S+|www\S+", "", text)                # Remove URLs
    text = re.sub(r"[^a-zA-Z\s]", "", text)                   # Remove
numbers and punctuation
    text = re.sub(r'\s+', ' ', text).strip()
    return text

def preprocess(text):
    text = clean_text(text)
    tokens = text.split()
    cleaned = [
        lemmatizer.lemmatize(token)
        for token in tokens
        if token not in stop_words and len(token) > 2
    ]
    return " ".join(cleaned)

# Apply preprocessing
df["cleaned_text"] = df["content"].apply(preprocess)
df = df[df["cleaned_text"].str.strip() != ""]

# Set up BERTopic with CountVectorizer
vectorizer_model = CountVectorizer(ngram_range=(1, 3),
stop_words="english")
topic_model = BERTopic(vectorizer_model=vectorizer_model,
language="english", top_n_words=10)

# Fit the model
topics, probs = topic_model.fit_transform(df["cleaned_text"])

# Show top 3 topics
from itertools import islice
valid_topics = topic_model.get_topic_info().Topic.unique().tolist()
valid_topics = [t for t in valid_topics if t != -1]

print("\nTop 3 Topics:")
for topic_id in islice(valid_topics, 3):
    print(f"\nTopic #{topic_id}:")
    print(topic_model.get_topic(topic_id))
```

Finding Coherence score for BERTopic model:

```
from gensim.models import CoherenceModel
from sklearn.feature_extraction.text import CountVectorizer
```

```
# Extract topics as lists of words (excluding outlier topic -1)
topics = topic_model.get_topics()
topic_words = [ [word for word, _ in topic_model.get_topic(topic)]
                for topic in topic_model.get_topic_info().Topic.unique()
if topic != -1 ]

# Use preprocessed tokens used in BERTopic
texts = df["cleaned_text"].apply(lambda x: x.split()).tolist()

# Create a dictionary and corpus for coherence
from gensim.corpora.dictionary import Dictionary
dictionary = Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]

# Calculate coherence score
coherence_model = CoherenceModel(
    topics=topic_words,
    texts=texts,
    dictionary=dictionary,
    coherence='c_v'  # you can also try 'u_mass', 'c_uci', etc.
)
coherence_score = coherence_model.get_coherence()

print(f" BERTopic Coherence Score (c_v): {coherence_score:.4f}")
```

Cost function for LDA model:



Overshooting and oscillation of learning rate alpha = 0.1