

Developing an Intelligent Chatbot

Group 01: Cemal Ata Uzgoren (100447097), Joshua Newton (100401106) and Aman Seth (100442755)

May 30, 2024

Contents

1	Introduction	3
1.1	Background and Motivation	3
1.2	Aim and Objectives of this coursework	3
1.3	Difficulties and Risks	4
1.4	Work Plan	5
1.4.1	Cemal Ata Uzgoren	5
1.4.2	Aman Seth	5
1.4.3	Josh Newton	5
1.4.4	Ganttchart	5
2	Analysis of Related Work	6
2.1	Microsoft Bing Co-Pilot	6
2.1.1	Ability To Extract Live Train Ticket Prices	6
2.1.2	Ability to Predict Train Delays	7
2.1.3	Ability To Advise Contingency Plans	7
2.1.4	Conclusion	7
2.2	Phi-3-mini	7
2.2.1	Ability To Extract Live Train Ticket Prices	8
2.2.2	Ability to Predict Train Delays	8
2.2.3	Ability To Advise Contingency Plans	8
2.3	Evaluation and Comparison	8
3	Methods, Tools and Frameworks	8
3.1	Chatbot GUI	8
3.2	Web Scraping	9
3.3	Chatbot Conversation - NLP	9
3.4	Regression Model Prediction	9
3.5	Development Framework	10
3.5.1	Flowchart	10
3.5.2	Use Case and Sequence Diagrams	10
4	Implementation	10
4.1	Web Scraping	10
4.2	Graphical User Interface Development	11
4.3	NLP Implementation	11
4.4	Knowledge Base & Inferring Engine	12
4.5	Regression Model	12
4.5.1	Data Types	13

4.5.2	Encoding the tlp	13
4.5.3	Binary Valued Columns	13
4.5.4	Feature Extraction	13
4.5.5	Missing Values	13
4.5.6	Delay Model Sequence Diagram	14
5	Testing	14
5.1	Unit Testing	14
5.2	Integration Testing	14
5.3	Usability Testing	15
6	Evaluation and Discussion	17
6.1	NLP	17
6.1.1	Natural Language Understanding (NLU)	17
6.1.2	Response Accuracy	18
6.1.3	User Experience (UX)	18
6.1.4	Technical Robustness	18
6.2	Regression Models	18
6.2.1	Multi-layer Perceptron (MLP) Regressor	18
6.2.2	K Nearest Neighbors Regressor:	19
6.2.3	Random Forest Regressor:	19
6.2.4	XGBoost Regression:	19
6.2.5	Metrics and results:	19
6.2.6	Model Evaluation:	21
6.3	Areas for Improvement & Future Enhancements	21
6.3.1	Part A: Find cheapest train ticket	21
6.3.2	Part B: Find train delay	21
6.3.3	Part C: Fetch contingency plans	21
7	Conclusion	22
A	Appendix	24
A.1	Tables	24
A.2	Figures	28
A.3	Group Work	49
A.3.1	Trello	49
A.3.2	Github	52
A.3.3	Design phase	54
A.3.4	Coding Phase	56
A.3.5	Testing Phase	58

Abstract

The goal of this project is to create a chatbot that helps users by using natural language processing (NLP). A chatbot is an automated dialogue system that uses natural language processing (NLP) to analyse user inquiries and provide appropriate responses. In this project, we developed a working a user service chatbot that can converse with users to assist them in precisely three scenarios: locating the best train ticket in the United Kingdom; predicting delays during a train journey between London Liverpool Street and Norwich; and offering backup plans in case of blockages on the Great Eastern Tracks in Greater Anglia between Colchester and Norwich. Simple and complex user inquiries can be entered into this chatbot, categorising them into one of the pre-existing labels, and provide related responses. The chatbot is based on machine learning (ML) models that is built using Scikit-learn, spaCy pre-trained models for NLP, along with NLTK (Natural Language Tool Kit) and other in-built python libraries. The final chatbot system was able to accurately and confidently achieve all objects it set out to complete.

1 Introduction

This report will describe, outline and display the efforts made towards creating an autonomous conversational agent (chatbot) to interact with a human user. The primary purpose includes searching the internet for cheap train tickets between two destinations and present the findings to the user to for procurement. This objective should be achieved through a short series of dialogue interactions. The second motive this chatbot aims to realise is to improve customer service through conversation. There is no limitation to the depth of assistance the chatbot can provide; a regression model to predict the arrival time at a final station of a journey was designed and implemented. Lastly, the chatbot must have the ability to provide train operators with assistance when dealing with blockages through several suggested resolution actions. This is achieved by employing a knowledge base (KB) which includes information regarding possible contingencies and regulation documents with actions within accordance.

1.1 Background and Motivation

Chatbots are useful tools for improving efficiency and reducing the human labour during tedious tasks. Chatbots in their infancy used predefined rules to converse with humans which led to considerable limitations in communication. However, the succeeding chatbots utilised machine learning (ML) and natural language processing (NLP) to analyse and respond intelligently with context and nuance to the users input. Section 2 describes the current standard of chatbots which are free to acquire and use. At present no commercially available chatbot has the ability to confidently check live train times and prices or accurately predict delayed arrival times. Nor do they have the ability to produce contingency plans when faced with blockages. The absence of these abilities is the motivation behind the project with the objectives outlined in Section 1.2.

1.2 Aim and Objectives of this coursework

To successfully realise this project and develop a sufficient working chatbot, the following objectives must be met and accomplished:

1. Create a conversational agent
 - (a) Create code to clean and deconstruct a users input to a manageable format.
 - (b) Create a NLP mechanism to determine the users intention of their input.
 - (c) Create an interactive Graphical User Interface (GUI) for better experience.
2. Scrape train booking websites for live train times and prices

- (a) Accumulate a minimum of 5 online services or websites that can produce accurate live train times and prices.
 - (b) Extract the relevant information into a sufficient data structure.
3. Display and converse the data collected to the user in an understandable manner
- (a) Displays the relevant information i.e. website name, fare amount and cheapest fare clickable link.
4. Deploy a regression model to predict a delay train line's arrival and departure
- (a) Collect dataset of train journeys from Norwich to London Liverpool Street.
 - (b) Preprocess the data for optimal utilisation when using regression models.
 - (c) Develop a regression model which will efficiently output a numeric value based on the users input.
 - (d) Measure and refine the model until a satisfactory quality has been achieved.
 - (e) Deploy the model using the chatbot.
5. Create rule engine (RE) to provide contingency plans
- (a) Implement NLP technique to extract locations and blockage type from the staff user input.
 - (b) The train stations can only be between Colchester and Norwich on Great Eastern Track in Greater Anglia.
 - (c) Create Rule Engine (RE) to provide alternate plans / routes to the train station staff based on the blockage type and stations provided.

1.3 Difficulties and Risks

This project includes several inherent factors which need to be considered to avoid unethical practices and application restrictions. Firstly, a restriction against the use of paid APIs was established. During the attempted exploration and extraction of data, a large quantity of websites offered access to their data through their API after a paid transaction. The choice to circumvent this option was to avoid unwanted financial burden and mitigates any economic advantages against colleagues attempting the same project.

Moreover, web scraping is a topic of ethical consideration and requires a thorough evaluation before commencing an automated scraping script. Due to moral concerns, caution is needed to avoid overloading a website with too many requests in a short time span. Failure to acknowledge these concerns could overload the server hosting the website, which as a result, could suffer from an overall loss of speed. In an extreme case this can be considered a denial-of-service (DoS) attack. It is common practices to deploy anti-scraping tools on websites which require the user to pass a CAPTCHA before accessing the data. Failure to pass the CAPTCHA will temporarily and on occasion permanently revoke the users access to the website.

Furthermore, online websites can contain personal information that should be avoided by a scraping tool indefinitely. The nature of the data and websites should mitigate any personal information, however caution will be practiced to ensure no personal data is extracted. The deployment of artificial “user-agents” to access Chrome web browser via Selenium has shown great promise as an option to avoid an accidental security breach of the website and as a result the chatbot is blocked from said website. However, implementing web scraping scripts employing Selenium requires prior knowledge regarding accessing web elements and locators. Should the website be composed of React element, the challenge of creating

dynamic code which searches and extract specific elements becomes too large to become viable. Furthermore, a lot of websites constantly update their locators and web elements, particularly when new features are being deployed as software patches or builds. This periodic shift necessitates a time commitment to maintain a close eye on the automation scripts created to extract data. Because web scraping scripts may begin to malfunction if the locators are altered, resulting in unexpected results displayed on the chatbot's graphical user interface.

The NLP models used in this project which are outlined in Section 3.3 demonstrated considerable difficulty in determining small or all UK cities. Hence, the objective to extract locations from the sentences became a largely challenging aspect of this project. In addition to the cities, the indicated date and time were difficult to extract because the pre-trained spaCy models that are now in use are not strong enough to retrieve the information from a lengthy or intricate user input. Therefore, it was necessary to find alternative methods in order to complete the necessary chatbot tasks.

1.4 Work Plan

The following section will outline each member's contribution over the duration of the project.

1.4.1 Cemal Ata Uzgoren

- Took the position as team leader
- Led the design and development of the GUI for the chatbot
- Led the integrity issues handling across the all three NLP tasks (Task A, Task B, Task C)

1.4.2 Aman Seth

- Implemented multiprocessing among all scraper scripts resulting in reduced time consumption in fetching and displaying cheapest fare results on GUI.
- Led the NLP implementation at the backend of the chatbot graphical user interface for all three tasks (Task A, Task B, Task C)
- Contributed with embedding of the regression model into the conversational script

1.4.3 Josh Newton

- Created initial web scraping scripts which included the a method using either the Selenium library or sending a prefilled payload to the service
- Led the processing and exploration of the dataset provided for training a regression model to predict the arrival time at Norwich during a journey departing from London Liverpool Street
- Contributed to the refinement of the GUI, revising the appearance based on feedback given from user testing

1.4.4 Ganttchart

The following figure displays the planned and executed timeline of events required to complete the aims outlined in Section 1.2.

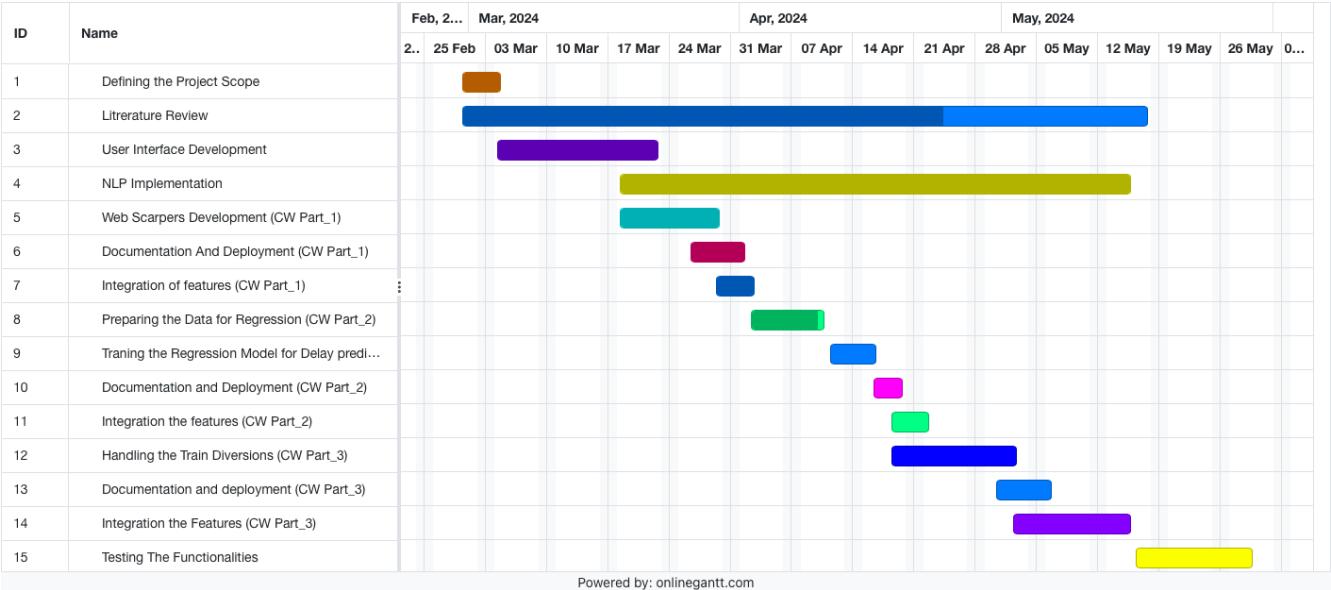


Figure 1: Gantt Chart displaying the execution timeline of tasks

2 Analysis of Related Work

This section outlines two examples of large language models (LLM) currently available for free and evaluate their performance with regards with the tasks relevant to this project. These include:

1. Accurately searching the live arrival and departure of train times
2. Offering the ability to provide the user with a URL link to purchase tickets extracted
3. Predicting arrival train times when a train is delayed at a specific station during a journey
4. Advising on the correct protocol when faced with train line blockages

2.1 Microsoft Bing Co-Pilot

The Microsoft Bing Co-pilot is currently developed using the GPT-4.0 LLM manufactured by OpenAI (Team 2024). Unlike the Chat-GPT LLM, Bing co-pilot has the ability to extract information from the internet and utilise it in a conversational response. Due to this unique ability, it has become a powerful search engine alternative. The following sections will outline Bing co-pilot's ability to complete the tasks outlined in Section 1.2.

2.1.1 Ability To Extract Live Train Ticket Prices

This model demonstrates great ability for conversation however the main objective is to achieve search & extract mechanism of live train times of a given journey. When the model is prompted with the request “find me the cheapest train ticket from London Liverpool Street to Norwich?” it returned a vague response with no exact value or information. Figure 6 demonstrates the response given which states a “starting price of £10” for the journey and also includes hyper links to online services to purchases train tickets.

Upon further inspection the quote of a “£10 stating price” was extracted from an online advertisements created by Trainline. A second attempt to request train tickets was explored with the following prompt “Please can you find me the cheapest train ticket from London Liverpool Street to Norwich for Monday 20th at 12:00?”. Figure 7 demonstrates a similar result to the previous prompt with a quoted starting

price of £10. The hyperlink provided points to the TrainLine online service, the service displays a month calendar with the cheapest available ticket per day. Figure 8 shows three days which match the quote of “£10” for the the cheapest ticket. Unfortunately, this response and suggested price is inaccurate as it is for days Wednesday 8th, Friday 7th and Saturday 8th, not Monday 20th of current month as requested in the prompt. Moreover, accessing the particular date which quotes tickets in access of £10 is vacant of such prices. This leads to the assumption a singular ticket could cost £10 but would be at less than desirable off-peak times.

2.1.2 Ability to Predict Train Delays

As per the secondary task described in Section 1.2, the LLM was prompted to predict the delayed arrival of a train journey between London Liverpool Street and Norwich when delayed at Ipswich. The following prompt “I am currently on a train as Ipswich during a journey from London Liverpool Street to Norwich. I left London at 12:00, the train was planned to depart from Ipswich at 13:07 but we are now departing at 13:15. What time would I arrive at Norwich?” was given. The response was an inaccurate attempt as the predicted arrival time at Norwich was 14:56 when the true planned arrival time was 13:46. Figure 9 shows the response in full. Using our regression model to predict the arrival time using this scenario, the predicated arrival time was “13:52” which was 6 minutes later than the planned time.

2.1.3 Ability To Advise Contingency Plans

As per the final task, the chatbot should be able to accurately advise a staff member with a contingency plan when faced with blockages. The scenario referenced when interacting with the chatbot is a “Full Line Blockage Between Ipswich and Stowmarket”. The prompt used was “I am a train conductor on a train between Ipswich and Stowmarket. Unfortunately there is a full line blockage contingency. What should we do?”. Figure 10 shows the response, review of the response shows general safety advisory steps such as ensuring passenger safety, contacting and communicating with the rail control centre to report the blockage. This advice can be acceptable but shows no specific form of action.

2.1.4 Conclusion

The model can extract inaccurate information which appears to be extracts from online advertisements and is dissimilar to the requirements found in the prompt. When asked to predict the delayed arrival at a station it gave an accurate description of the journey distance and travelling time but in despite of this, it gave an inaccurate arrival time. Overall the model shows potential for achieving the desired results. However due to the limitations of being a “smart” search engine, it relies on the collected internet sources to be accurate. With this key information the model cannot be confidently used to acquire train tickets or predict the delayed arrival times. When asked to advise on handling blockages the model gave general advice without any specific instructions. The same prompt was given to our model which returned detailed instructions as shown in Figure 15. A conclusion was drawn that within the given restrictions and goals, our chatbot was superior.

2.2 Phi-3-mini

The Microsoft developed MML Phi-3-mini boasts a moderate 3.8 billion parameters which were trained on 3.3 trillion tokens. It uses a transformer decoder architecture and limits itself with a default context length of 4K tokens. However, with the introduction of LongRope, this limit extends to 128K tokens, this extended model is named Phi-3-mini-128k (Abdin et al. 2024).

2.2.1 Ability To Extract Live Train Ticket Prices

When the model is prompted to fetch and present live train times, it will notify the user that the model is unable to get these values. Further testing shows that the model does not have access to the internet and thus live data. Figure 11 shows the response of the model. With this information no further testing was given.

2.2.2 Ability to Predict Train Delays

Similar to Section 2.1.2 the chatbot was asked “I am currently on a train at Ipswich during a journey from London Liverpool Street to Norwich. I left London at 12:00, the train was planned to depart from Ipswich at 13:07 but we are now departing at 13:15. What time would I arrive at Norwich?”. The response shown in Figure 12 describes the train journey from Ipswich to Norwich as a 2 hour journey. However the TrainLine service, which was used as a reference, describes the journey as being 39 minutes as shown in Figure 14. This model also presetned an inaccurate estimated time between these two stations.

2.2.3 Ability To Advise Contingency Plans

Similar to Section 2.1.3 the chatbot was asked how to handle a full Line Blockage Between Ipswich and Stowmarket. Figure 13 shows the results which appears correlated to the response from Bing Co-Pilot. As the response is not accurate, the conclusion was drawn deeming it insufficient, yielding our chatbot to be superior.

2.3 Evaluation and Comparison

The reviewed chatbots appear to be supe with general dialogue, specifically conversational responses. However, both models fall short of the tasks 2-5 outlined in 1.2. Solely based on these tasks, the aforementioned chatbots are not sufficient leaving our chatbot most acceptable.

3 Methods, Tools and Frameworks

The primary programming languages used for this project are Python and JavaScript. *PostgresDB* was initially considered for the KnowledgeBase, however the project has found sufficient success using local csv files as an alternative to databases.

Programming Languages: Python & JavaScript.

GUI development: HTML & CSS.

Packages: NLP, NLTK locationtagger(Bird & Klein 2009), Spacy’s “en_core_web_lg model”. For matching regex patterns: Selenium, requests, html-requests, BeautifulSoup, Minidom, lxml, urllib3 etc...

KnowledgeBase and Engine: CSV & JSON.

Below is the in detail description of the tools and frameworks used for each component of the project:-

3.1 Chatbot GUI

- Graphical User Interface: GUI development using JS, NodeJS, HTML, CSS.
- Voice Commands: Text-to-Speech & Speech-To-Text recognition using APIs such as webkitSpeechRecognition, SpeechRecognition etc.

3.2 Web Scraping

- Selenium: This library was used to scrape data from the websites who doesn't provide any API to extract data in form of response.
- Requests: This module was used to extract data via HTTP/HTTPS requests.
- BeautifulSoup: This library was used to scrape websites HTML elements to fetch data.

3.3 Chatbot Conversation - NLP

- Spacy: This library was used for natural language processing specifically for named-entity recognition (NER).
- Nltk: This toolkit was used along with spaCy for NLP.
- Fuzzywuzzy: This library was used to determine closes matches in the user input to extract relevant information.
- Experta: This library was used for building expert systems to provide suitable answers to the user.

3.4 Regression Model Prediction

Models Utilised A variety of commonly implemented regression models were employed to predict the arrival at Norwich from the departure from London Liverpool Street. These models incline:

- Multi Layer Perception (MLP) Regression: the sklearn.neural_network library was used to build a simple MLP model (Pedregosa et al. 2011).
- K-Nearest Neighbors (KNN) Regression: the sklearn.neighbors library was used to build and tune a K-Nearest Neighbors model (Pedregosa et al. 2011).
- XGBoost (XGB) Regression: the XGBoost model from Chen & Guestrin (2016) was implemented.
- Random Forest (RF) Regression: the sklearn.ensemble library was used to build and tune a Random Forest model (Pedregosa et al. 2011).
- Linear Regression: the sklearn.linear_model library was used to implement a Linear Regression model (Pedregosa et al. 2011).
- Huber Regressor: the sklearn.linear_model library was used to implement a Huber Regressor model (Pedregosa et al. 2011).
- Lasso Regressor: the sklearn.linear_model library was used to implement a Lasso Regressor model (Pedregosa et al. 2011).
- Stochastic Gradient Descent Regressor: the sklearn.linear_model library was used to implement a Stochastic Gradient Descent Regression model (Pedregosa et al. 2011).

Metrics and performance evaluation During the training process each model was fitted to 80% of the dataset while 20% was employed as validation data. Each model will predict the arrival time at Norwich based on the values in the training dataset.

- NumPy: This was used for data manipulation and numerical calculate (Harris et al. 2020).
- Pandas: This was used for data storage inside dataframes along with dataframe manipulation tasks (pandas development team 2020).

- sklearn: This library was used for data preprocessing and model metrics (Pedregosa et al. 2011).
- Seaborn and Matplotlib: was used to generate visual representation of the data (Waskom 2021), (Hunter 2007).

3.5 Development Framework

The following sections outline and describe the current efforts and developments made towards achieving the aims and objectives described in Section 1.2.

3.5.1 Flowchart

Figure 3 details a high-level flow chart of the chatbot when the user requests a train ticket.

3.5.2 Use Case and Sequence Diagrams

Figure 26 describes a use case diagram of the whole system. Also the general logic of the chatbot constructed by following the algorithms of the sequence diagrams which are stated in figure 22c, 22c, and 22c for each of the parts.

4 Implementation

4.1 Web Scraping

The model is developed to extract the required data from the following train booking websites:

1. LNER
2. Greater Anglia
3. My Train Ticket
4. National Rail
5. Southern Railways
6. Train Pal

The required information extracted are:

- Cheapest website name
- Fare amount
- Cheapest fare website link

Ticket Scraping Mechanism

Figure 23 shows the sequence diagrams which involves a structured sequence of interactions between the user (actor), graphical user interface (GUI), natural language processing (NLP) system, and web scrapers. The process begins when the user initiates a conversation via the GUI, which then requests the NLP system to extract information from the user's input. If the information provided is incomplete, the system prompts the user for additional details. Once complete information is obtained, the NLP system activates web scrapers to retrieve the cheapest available ticket details, which are then displayed to the user.

If the user requests a return ticket, the system processes this similarly by extracting the necessary return trip information and verifying its completeness. Incomplete information prompts further user queries. Upon obtaining complete return information, the NLP system runs the web scrapers to find the cheapest return ticket, which is subsequently presented to the user.

Throughout the interaction, the system may send warnings to keep the user informed of any issues or important notices. The conversation concludes once the user's inquiries are fully addressed, with the system ending the interaction through predefined methods. This model ensures efficient and accurate handling of ticket inquiries by leveraging robust communication between the user, GUI, NLP system, and web scrapers.

4.2 Graphical User Interface Development

The graphical user interface (GUI) is developed with text-to-speech and speech-to-text recognition feature embedded. The interface understands the voice commands given by user and displays on the UI. Based on the commands received, it attempts to fetch the result from the NLP and displays back on the screen. A throbber is also implemented while the response is being fetched by chatbot for a reliable experience. Figure 5 shows an example of the current GUI state.

4.3 NLP Implementation

The development of the natural language processing (NLP) behind the chatbot was made using a combination of SpaCy, NLTK (Natural Language Tool Kit), Scikit-learn and other in-built python libraries. The user input sentence was lemmatised first, meaning linking similar meaning words as one word *Lemmatization* (n.d.), then extracting the relevant details to fulfill the task and finally performing action as per the detected task and providing the appropriate response back to the user.

Lemmatisation: The stop words were removed during the sentence cleaning process but leaving *to* and *from*. The intention behind leaving these two words were to figure out the origin / departing and arriving stations from the users input.

Labels Creations & Detection: There were various different labels which were created as per the tasks which this chatbot should be able to perform. Some of the key labels are *train*, *location*, *date*, *time*, *return*, *delay*, *contingency*, *blockage*, *assist*, *help* etc. The sentences matched against these labels invoke their respective functionality functions to perform their tasks to achieve the end goal.

Similarity Check: The sanitised user input gets matched to the label which it belongs. The system calculate the similarity index via *similarity()* which is an in-built function within spaCy python module. If the similarity matches with the minimum similarity defined threshold, it invokes a certain task as per the matched label.

Stations Extraction: It was observed that the spaCy and NLTK in-built functions could only fetch the renowned cities of the UK such as London, Birmingham, Manchester etc but they could not fetch small cities / stations such as Exeter, Norwich, Seven Kings etc. Hence they were not fully reliable and another approach had to be implemented. The *stations.csv* containing all possible stations within the United Kingdom was used to match the stations against the user input. The already lemmatised user input which re-cleanse even more to shrink the user input as much as possible so that the match could have been easy. If there an exact match of the station in the user input to *stations.csv*, the system would return the exact matching stations. Even the input has a partial match in case there is spelling mistake or there is a genuine partial matching for instance, if user enters London as input but London contains

10+ train stations such as London Piccadilly or London Liverpool Street etc, the system would return all the partially matched station asking the user to choose the precise ones.

Date & Time Extraction: The time extraction was performed mainly using python regex matching. Since spaCy and NLTK libraries performed quite poorly in time extraction from a sanitised sentence, various regex matching were placed instead.

Return Ticket Mechanism: Return ticket functionality has been implemented in the first task of the chatbot that is to find the cheapest train fare within the UK. The chatbot asks the user once one-side train fare search is performed if the user wants to view cheapest return fares as well. The user may wish not to view return fares and simply quit the conversation. But NLP has been implemented if user wishes to view return fares too. Assuming the return would be within the same stations as one-way search, hence the chatbot would ask the user for only date and time of the return. Once the details are provided, the return fares are then searched and returned on the GUI.

Delay Calculation Mechanism: The delay calculation was implemented comprising of the details which are required from the user as input followed by invoking the saved ML model for predicted delay time calculation. The chatbot asks the user for the time of the last or current departed station, the name of the last or current departed station and lastly the time when the user departed from the origin station i.e by default London. Once all the details are received, the chatbot returns the predicted delay to reach to the destination i.e Norwich on it's user interface.

Contingency Plans: The NLP implemented behind the task of providing the contingency plans to the train station staff comprise of the details such has the name of the train stations on the Great Eastern Track in Greater Anglia facing the blockage and the type of the blockage i.e either *Partial* or *Full*. On receiving this information from the user, the alternate plans / routes are displayed on the chatbot GUI. Figure 25 depicts how the actor interacts with the chatbot to be advised on what contingency to follow in an unexpected event or emergency. The sequence starts with the actor invoking a conversation about contingencies. This function then calls the extracts the relevant information from the prompt, which gathers information about the contingency plan from the knowledge base. Following this, there is an alternate decision point, the decision is made based on whether the required information is filled enough to be satisfactory. If the “requiredInfo” is filled, then the a function is called to request the corresponding contingency plan. This function retrieves the complete contingency plan from the knowledge base. Finally the function will finally present the corresponding plan to the user. On the other hand, if the required info does not contain enough information, then a function is invoked to prompt the actor to insert the required information the missing fields through short warnings.

4.4 Knowledge Base & Inferring Engine

The knowledge base used for this project are in the form of JSON files containing the rules for different types of intentions expressed by the user in it's input. The JSON contained the labels as the keys. The labels refer to the intention or the type of task requested by the user. Once the intention matches the label key, the responses stored as values against the keys in the JSON were inferred and returned to the user.

4.5 Regression Model

The developments into the delayed arrival time predictive model was made using a combination of JuPyter Notebook and the later integration into the conversational Python script. The Notebook environment was chosen as an intuitive way to display and understand the data processing and model training. Once

the model(s) were trained and saved to a recallable file, code to recall each file embedded into Python scripts then become the main executable location.

4.5.1 Data Types

A detailed review of the data revealed that the majority of value types were string formats with 24-hour time values. Initially, there was a plan to encode each column containing string values using an array of methods including Binary Encoding, One Hot Encoding, Label Encoding, and Target Encoding. However, due to the complexity and quantity of unique values, encoding timestamp values proved inefficient. Instead, the Pandas library was utilised to convert the string value to datetime objects using the format “Hour : Minute”. Immutable or missing values are handled and replaced with a Not A Time (NaT) value.

The timecode objects were transformed to calculate the number of seconds since midnight. This calculation involves multiplying the hour component by 3600, the minute component by 60, and summing these products. The reason for using this method is that it provides a simpler normalisation process, simplifies calculations, and reduces dimensionality, which in turn can improve model performance.

4.5.2 Encoding the tlp

As described in Table 10 the column “tlp” held the timing point locations for all stations of the journey. Examples of the values included: “SHENFLD”, “NRCHTPJ”, “NRCH” and “TROWSEJ”. Due to the data only containing journeys between London Liverpool Street and Norwich, there were 34 unique values as described in Table 10. Due to to considerably small quantity of values, the Label Encoder method is utilised as the encoder. Table 9 showed the original “tlp” values and their corresponding encoded values.

4.5.3 Binary Valued Columns

In addition to the encoding practices described Section 4.5.2, it is important to note that while columns with binary values were encoded with either a “1”, “0”. These encoded representations did not feature in the final set of extracted features as they were deemed less effective for our predictive modelling objectives.

4.5.4 Feature Extraction

The dataset comprised an extensive range of timestamps that incorporated predicted, estimated, planned, and actual times. Table 10 outlines the features paired with a description. The essential features necessary to predict arrival time at Norwich (NRW) during a journey from London Liverpool Street (LST), included the station’s unique code, the departing time from LDN, departing time from each individual station and arrival time at NRW. As a result of adopting a simple feature set, there was an efficient systematic approach to handling missing values detailed in the following section.

4.5.5 Missing Values

The dataset contained an inherently large proportion of NaN values owing to a single time value being recoded at a singular point in time. To mitigate the issue of missing data, an systematic strategy was employed that involved the conditional imputation based on temporal availability. The process was structured hierarchically; firstly, the extraction of actual departure and arrival time instances were prioritised; however, when such records were unavailable, estimated values were introduced. Should the estimated values also be null, predicted values were employed as a final resort. As an additional method for dealing with missing information, passing times were occasionally used.

4.5.6 Delay Model Sequence Diagram

Figure 24 details the sequence diagram when the chatbot is utilising the regression model. The process is initiated when the actor starts the conversation. Subsequently, the system performs a completeness check on the required information. If all necessary data is available, the system extracts the relevant details and executes the prediction model. The resultant prediction is then presented to the user through the display function. However, if the system encounters insufficient information, it prompts the user for additional input. Additionally, the system may issue warnings to alert the user of any potential issues.

5 Testing

5.1 Unit Testing

As seen in Figure 27 a total of 23 unit tests were developed and implemented, these test can be split into three categorise: “conversational functions”, “scraping functions” and “contingency functions”. Table 12 shows the results of the conversational function tests. From these results we can see one failed case “test_lemmatise_4”, the failure was a result of the lemmatise function not removing the “@” from the input string. Table 13 show the results of each web scraping function implemented. From these results we can see 3 scraping scripts have failed, these failures can be a result of various errors. The suspected issues are:

- Blocked Selenium client from anti bot scripts
- Website HTML alterations which could break the Selenium object detection functions

5.2 Integration Testing

Test ID	Integration point	Test Description	Expected result
P1	Part 1 (Finding the cheapest fare)	Test the functionality of finding the cheapest fare. Start a conversation to provide the required information for ticket booking.	Chatbot returns the cheapest train ticket fare.
P2	Part 2 (Delay Estimation)	Test delay estimation intelligence of the chatbot. Ask for the delayed train's information and get the predicted arriving time.	Chatbot estimates the delay correctly regarding to provided details.
P3	Part 3 (Contingency recommendations)	Test the contingency function of the chatbot. Give a to get a contingency plan for that circumstance.	Chatbot provides appropriate probability recommendations according to predefined rules and regulations

Table 1

Test ID	Integration point	Test Description	Expected result
TC1	Part 1 and Part 2	Initialise a conversation about fare search. After completing the train ticket search, ask for delay estimation.	Chatbot finds the cheapest fare and estimated the delay correctly without being influenced by each other.
TC2	Part 2 and Part 3	Start conversation for both part 2 and 3 respectively, and examine the results to make sure that the part 2 and 3 not effect each other.	Chatbot should perform the delay estimation and contingency plan recommendation correctly.
TC3	Part 1 and Part 3	Start conversation for both part 1 and 3 respectively, and examine the results to make sure that the part 1 and 3 not effect each other.	Chatbot should perform the cheapest train fare search and delay estimation correctly.
TC4	Contradictions	Test the contradiction feature that restricts the user switch to another subject while an ongoing conversation takes place about something else.	Chatbot displays a informative message which says to switch to another conversation either reset the current conversation or finalise it.
GUI	GUI Integration	Test the integration of the graphical user interface with all parts of the chatbot. Perform tasks related to ticket finding, delay estimation and contingency recommendation using the GUI.	GUI handles the conversation about all parts flawlessly and orients the user for next steps.

5.3 Usability Testing

Two UEA students earning master's degrees, one in business and the other in computer science, conducted the usability testing demonstrated in the table below:-

ID	Description	Preconditions	Test Steps	Expected Results	Actual Results
TC01	Find cheapest ticket for specified route, date and time	NodeJS server is up and running	1. Enter departure and arrival locations 2. Enter travel date and time 3. Submit search query	Display cheapest ticket for the specified route, date and time	One-way ticket details displayed: Ticket price, Website name and link
TC02	Find cheapest ticket for partially matched train stations	NodeJS server is up and running	1. Enter partially matched departure and arrival stations	Display a list of partially matched available stations with input	Partially matched stations displayed asking user to give precise name choosing from the output
TC03	Find cheapest ticket if time is not provided	NodeJS server is up and running	1. Provide departing and arriving stations 2. Enter date but not time 3. Submit search query	Chatbot should ask the user to provide time	Time missing warning displayed on GUI
TC04	Find cheapest ticket when stations are mis-spelled	NodeJS server is up and running	1. Enter departure and arrival locations with misspelled words 2. Submit search query	Display a list of partially matched available stations with input	Partially matched stations displayed asking user to give precise name choosing from the output
TC05	Find cheapest return ticket	NodeJS server is up and running	1. Enter departure and arrival locations 2. Enter travel date and time 3. Perform one-way ticket search 4. Ask for return 5. Submit search	Display return ticket cheapest fare details: Ticket price, website name and link	Return ticket details are displayed
TC06	Find train delay by providing current or last departed station and time along with time from origin station	NodeJS server is up and running	1. Enter current or previously departed station 2. Enter departure time of previously departed station 3. Enter departure time from origin station	Train delay to destination Norwich should be displayed	Delayed time is displayed on GUI

ID	Description	Preconditions	Test Steps	Expected Results	Actual Results
TC07	Find train delay without providing departed time of origin station	NodeJS server is up and running	1. Enter current or previously departed station 2. Enter departure time of previously departed station 3. Click submit button	Chatbot should ask the user to enter the departed time from origin station	Departed time from origin missing warning is displayed on GUI
TC08	Find train delay of an unknown station	NodeJS server is up and running	1. Enter current or previously departed station which is not between london and norwich 2. Click submit button	Chatbot should try partial matching and display warning message of station not found	Station not found message is displayed
TC09	Find contingency plans for partial or full link blockage for valid stations on great easter track	NodeJS server is up and running	1. Enter blockage type as full or partial 2. Enter train stations on great eastern track 3. Click submit button	Contingency plans should be displayed as per the blockage and stations given	Contingency plans displayed accordingly
TC10	Find contingency plans without providing the blockage type	NodeJS server is up and running	1. Enter train stations on great eastern track 3. Click submit button	Chatbot should ask the user to enter the blockage type providing a hint to choose either partial or full	Missing blockage type warning is displayed on GUI

Table 2: Usability Test Cases

6 Evaluation and Discussion

6.1 NLP

The following sections outline the chatbot's performance in natural language understanding, response accuracy, user experience, and technical robustness.

6.1.1 Natural Language Understanding (NLU)

- **Intent Recognition:** The chatbot correctly identifies user intentions, such as searching for train tickets, specifying travel date and time, querying about prices or requesting delay information or requesting for contingency plans in case of partial or full line blockage.

- **Entity Recognition:** The chatbot accurately identifies key entities like locations (departure and arrival), dates, times, and ticket preferences (e.g., on-way or return), blockage type etc.
- **Context Handling:** The chatbot is effective in maintaining context over a multi-turn conversation and understanding follow-up queries.

6.1.2 Response Accuracy

- **Correctness:** The chatbot accurately provided the information of the cheapest train ticket, delayed time display or alternate routes as per contingency plans.
- **Relevance:** The chatbot's responses are appropriate as per user queries.
- **Completeness:** The chatbot responses are comprehensive, providing all necessary information to the user.

6.1.3 User Experience (UX)

- **Ease of Use:** As per the usability testing carried out by different UEA students, the feedback was positive about chatbot being easy to use.
- **Response Time:** The response time of the answer was fairly short. The multi-processing implemented which runs the scraper scripts asynchronously rather than in series, saved a lot of response time.
- **User Satisfaction:** The overall response from both of the UEA students who carried usability testing was positive and satisfactory.

6.1.4 Technical Robustness

- **Reliability:** The developed chatbot have consistent performance with correct and appropriate responses.
- **Error Handling:** The chatbot is robust enough to handle conversations related to different tasks as well as if there are grammatical errors in the user's input or even there is partial information given.
- **Integration:** The integrity constraints were applied in the chatbot to prevent delivering invalid information to the users.

6.2 Regression Models

The sections below outline and describe the research efforts made into regression models and the performance metrics of each.

6.2.1 Multi-layer Perceptron (MLP) Regressor

A Multi-layer Perceptron (MLP) model was implemented using both the Scikit and Keras libraries, however the results were considerably inaccurate when compared to other models. Table 8 shows its inaccuracy. Due to the limited time and computational resources a hyper parameter tuning mechanism such as Gridsearch or RandomSearch was not implemented. Neither was an extensive training run executed. With these circumstances and results the MLP models were made redundant and not used in the final predictions.

6.2.2 K Nearest Neighbors Regressor:

The parameter which configured the number of neighbours of the K-Nearest Neighbors (KNN) regression model was manually tuned using a *for* loop which iteratively increased the value of k and scored the model accordingly using metrics outlined in Section 3.4. Figure 16a shows the iteration of k , the results of these k values can be seen in Figures: 16b, 16d, 16c. From these results we can see minimal prediction accuracy improvement when $k > 30$, on the contrary we can see great improvements when k is increased to 20. With these results in mind, a k value of 50 was used to maximise potential accuracy as the trade off between prediction time per value of k was minimal.

6.2.3 Random Forest Regressor:

The Random Forest (RF) regression model was built with near-default parameters. The only specific change made to the configuration of the model was the number of estimators. Similar to the manual tuning of the KNN model outlined in Section 6.2.2, an iterative *for* loop was employed to score the model per number of estimators. Figure 17a displays the number of estimators searched, an initial value of 5 was chosen with an iterative increase of 1 until a value of 10 was selected. Once the value of estimators reached 10, an incrementation of 5 was chosen instead to increase efficiency. Table 4 displays scores per value of k . Figures 17c, 17b, 17d show the models performance per iteration. From these results we can see the performance peaked at an estimator value of 8. -

n_estimators	RMSE	MAE	R2
5	1692.184531	322.930468	0.993185
6	1692.829572	322.482591	0.993180
7	1689.592844	322.124198	0.993206
8	1683.224923	321.827668	0.993257
9	1682.711353	321.882941	0.993261
10	1686.329066	322.128959	0.993232
...
100	1685.108628	322.103318	0.993242

Table 4: Performance results of the Random Forest Regression model

6.2.4 XGBoost Regression:

Similar to K Nearest Neighbors and Random Forest models, the XGBoost model was implemented with default parameters with iteratively increasing number of estimators. Table 6 and Figure 18 show the performance of the model per number of estimators. From these results we can see a correlation between an increased value of estimators and the models improved performance.

6.2.5 Metrics and results:

Table 8 and Figures 19a-19d outline each models performance using the metrics described in Section. For a secondary evaluation, previous journey arrival and departure times were extracted from an online resource “Recent Train Times” to be used as new unseen data. Figure 20 and 21 are two examples of the ground truth values extracted from the online source. Figure 2 and 22 show the highest attaining classifiers predictions of the arrival time at Norwich using the data extracted.

n_estimators	RMSE	MAE	R2
1	14423.881550	11689.875945	0.504871
2	10194.480707	8214.249700	0.752666
3	7270.192730	5782.620130	0.874210
4	5271.073107	4084.681260	0.933877
5	3932.805430	2900.675047	0.963191
...
50	1770.881199	392.856774	0.992537
55	1766.922512	391.254378	0.992570
65	1752.241861	388.335768	0.992693
75	1740.280743	385.322977	0.992792
85	1733.922944	383.150441	0.992845
95	1728.680439	381.605032	0.992888

Table 6: Performance metrics for XGBoost regressor per value of n_estimators

Model Name	R2 Score	Mean Absolute Error	Root Squared Error	Mean Squared Error
MLP Regressor	0.45369	7876.414635	15151.049907	229554313.277678
KN Regressor	0.992975	337.436982	1718.046845	2951684.960503
XGB Regressor	0.992888	381.605032	1728.680439	2988336.06081
RandomForest	0.993242	322.103318	1685.108628	2839591.088781
Linear Regression	0.394573	8650.629986	15949.750001	254394525.082963
Huber Regressor	0.2222805	5853.991688	18071.2504	326570091.003966
Lasso	0.394573	8650.630322	15949.749997	254394524.969961

Table 8: Each model paired with the metric results

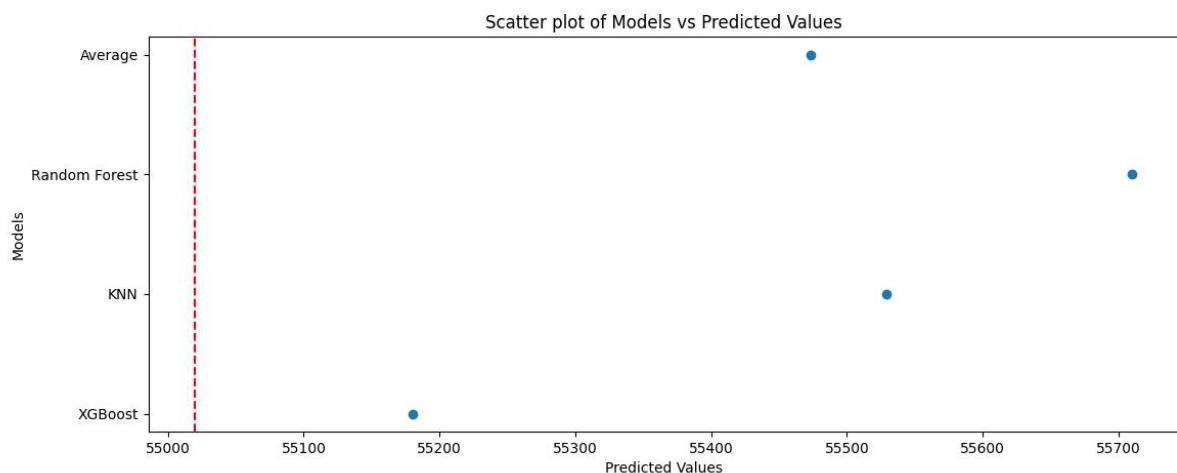


Figure 2: Regression models prediction of arrival time at Norwich from a departure from London Liverpool Station if delayed at Ipswich. The red dotted vertical line is ground truth.

6.2.6 Model Evaluation:

From the results described in Section 6.2.5 Random Forest, K Nearest Neighbors and XGBoost regressor models appear superior. However, as shown in Figures 2 and 22 each model will have a degree of inaccuracy when making predictions, as a result no single model would be a better choice for deployment. An average value was instead used to capture the accuracy of each model.

6.3 Areas for Improvement & Future Enhancements

The following sections describe the areas which could be subject to improvement should the project continue.

6.3.1 Part A: Find cheapest train ticket

- In order to make the search for the cheapest train tickets more dynamic and accurate, the user should be allowed to enter other parameters, such as the number of adults and children and whether or not they have a rail card. If so, inquire about the kind of railcard, etc.
- The user should be able to request a return ticket all at once rather than having to wait for the one-way search to finish before responding to the chatbot's request to inquire whether they would also like a return ticket. The user may be able to save more money in this method.
- Even though multi-processing is used to run all of the scrapers asynchronously, the cheapest train fare search still takes a long time since the system needs time to render the response on the GUI. Redesigning the system could improve its performance.

6.3.2 Part B: Find train delay

- In an attempt to improve the model's accuracy, more features could be introduced such as: days of the week, current weather at time of recording or public holiday occasion. One can assume these parameters should see considerable difference.
- The data provided was of decent size and quality, however an increased volume could be advantageous.
- At present the model will only predict the arrival time at Norwich station from London Liverpool Street. To improve upon this model, predictions of each mid-way station as well as predictions of other journeys should be possible.

6.3.3 Part C: Fetch contingency plans

- Give the user the ability to request backup plans between any station on the Great Eastern track, not only the one between Colchester and Norwich.
- The system should be able to receive additional inputs based on various situations, such as whether the obstruction is the result of a natural disaster like heavy rain or flooding, and then return the contingency plans in accordance with those inputs in order to make the search more dynamic. Train services must be halted, etc., if the temperature rises beyond thirty degrees.
- It is also possible to give the backup plans due consideration. In the event of a blockage between two stations, the system ought to provide specific solutions, especially if it is close to midnight. A certain kind of response should be given if it's peak hour and so on.

7 Conclusion

This study introduces an innovative form of an interactive chatbot that displays the cheapest train fares across the UK, displays train arrival times and assists in forming contingency plans. Based on the crucial demand for chatbots that can rely on a GUI to continually query accurate train timings and fares and repeatedly deploy numerous web scraping scripts to discover the cheapest ticket, this project established a model with clear advantages for the display of train arrival time. The user's destination data is collected via a GUI and fed to the developed predictive models of XGBoost, Random Forest and K-Nearest Neighbours to derive findings. The predictive models exhibited robustness and even a margin of error of less than a few minutes, testifying to the reliability of the chip-processing logic of the chatbot. Furthermore, the chatbot incorporated extensive contingency plans capable of providing rapid and thorough responses to a variety of situations, including route obstructions. To complement this, a KB (knowledge base) containing information about various scenarios and regulatory papers enabled the chatbot to assist in training operators and establishing its credibility. The project's successful outcome was probably based on the combination of ML (machine learning) and NLP (natural language processing) technologies that can help understand and respond to varying types of user requests in a correct contextual manner. Extensive testing, including unit testing, integration testing, and usability testing, was carried out to prove that the chatbot performed well and that users were satisfied.

This project also emphasises the ethical aspects of online scraping and data protection, following best practices to prevent overloading servers and obtaining personal information. By avoiding commercial APIs and using ethical web scraping techniques, the project struck a compromise between usefulness and moral responsibility.

Finally, this study displays substantial advances in chatbot capabilities by combining natural language processing and machine learning to improve user interactions in the rail travel area. Future improvements include broadening the chatbot's knowledge base, enhancing its NLP capabilities for more sophisticated comprehension, and incorporating more transit options to create a more comprehensive travel assistant. The success of this initiative demonstrates the potential for intelligent chatbots to improve customer service and operational efficiency in the transportation industry.

References

- Abdin, M., Jacobs, S. A., Awan, A. A., Aneja, J., Awadallah, A., Awadalla, H., Bach, N., Bahree, A., Bakhtiari, A., Behl, H. et al. (2024), ‘Phi-3 technical report: A highly capable language model locally on your phone’, *arXiv preprint arXiv:2404.14219* .
- Bird, Steven, E. L. & Klein, E. (2009), *Natural Language Processing with Python*.
- Chen, T. & Guestrin, C. (2016), Xgboost: A scalable tree boosting system, in ‘Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining’, pp. 785–794.
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. & Oliphant, T. E. (2020), ‘Array programming with NumPy’, *Nature* **585**(7825), 357–362.
URL: <https://doi.org/10.1038/s41586-020-2649-2>
- Hunter, J. D. (2007), ‘Matplotlib: A 2d graphics environment’, *Computing in Science & Engineering* **9**(3), 90–95.
- Lemmatization (n.d.), <https://www.techtarget.com/searchenterpriseai/definition/lemmatization>. Accessed: 2024-05-19.
- pandas development team, T. (2020), ‘pandas-dev/pandas: Pandas’.
URL: <https://doi.org/10.5281/zenodo.3509134>
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. (2011), ‘Scikit-learn: Machine learning in Python’, *Journal of Machine Learning Research* **12**, 2825–2830.
- Team, B. S. (2024), ‘Bing search powered by gpt-4’, *Bing Blog* .
URL: <https://www.bing.com/blog/bing-search-powered-by-gpt-4>
- Waskom, M. L. (2021), ‘seaborn: statistical data visualization’, *Journal of Open Source Software* **6**(60), 3021.
URL: <https://doi.org/10.21105/joss.03021>

A Appendix

A.1 Tables

Original Value	Encoded Value	Original Value	Encoded Value
BOWJ	0	ILFELEJ	18
BROXBRN	1	ILFORD	19
BRTWOOD	2	INGTSTL	20
BTHNLGR	3	INGTSTN	21
CHDWLHT	4	IPSWEPJ	22
CHESHNT	5	IPSWESJ	23
CHLMSFD	6	IPSWHJN	24
CLCHSTR	7	IPSWICH	25
DISS	8	KELVEDN	26
FRSTGT	9	LIVST	27
FRSTGTJ	10	MANNGTR	28
GIDEAPK	11	MANRPK	29
GIDEPKJ	12	MRKSTEY	30
GODMAYS	13	MRYLAND	31
HAGHLYJ	14	NEEDHAM	32
HAKNYNM	15	NRCH	33
HFLPEVL	16	NRCHTPJ	34
HRLDWOD	17	ROMFORD	35
SEVNSIS	36	STWMDGL	39
SHENFLD	37	STWMRKT	40
STFD	38	SVNKNGS	41
TROWFLR	42	WARE	45
TROWSEJ	43	WITHAME	46
TRWSSBJ	44		

Table 9: Mapping of original values to encoded values for the 'tpl' column (Part 1)

Column Name	Data Type	No. Unique Values	Description	Notes
rid	int64	660	Train RTTI Train Identifier	Unique code for train travel
tpl	object	34	TIPLOC (Timing point locations)	Unique station code
pta	object	285	Planned Time of Arrival	24hr Time value
ptd	object	282	Planned Time of Departure	24hr Time value
wta	object	384	Working (staff) Time of Arrival	24hr Time value- with seconds
wtp	object	959	Working Time of Passing	24hr Time value
wtd	object	373	Working Time of Departure	24hr Time value- with seconds
arr_et	object	11	Estimated Arrival Time	24hr Time value
arr_wet	object	9	Working Estimated Time	24hr Time value
arr_atRemoved	object	2	true if actual replaced by estimated	True / False
pass_et	object	851	Estimated Passing Time	24hr Time value
pass_wet	float64	1	Working Estimated Time	24hr Time value
pass_atRemoved	object	2	true if actual replaced by estimated	True / False
dep_et	object	14	Estimated Departure	24hr Time value
dep_wet	float64	1	Working Estimated Time	24hr Time value
dep_atRemoved	object	2	true if actual replaced by estimated	True / False
arr_at	object	1048	True time of arrival	24hr Time value
pass_at	object	1121	True time of train passing through	24hr Time value
dep_at	object	1008	True time of train departure	24hr Time value
cr_code	float64	12	Cancellation Reason Code	Float value
lr_code	float64	26	Late Running Reason	Float Value

Table 10: Table of dataset column headers and the number of unique values

Index	arr_at_seconds_since_midnight	my_prediction_since_midnight	arr_at	my_prediction
9129	67440.0	67903.125000	18:44:00	18:51:43
9130	68040.0	68576.476562	18:54:00	19:02:56
9132	68760.0	68962.398438	19:06:00	19:09:22
9137	70140.0	70106.796875	19:29:00	19:28:26
9141	71340.0	71103.328125	19:49:00	19:45:03
...
27006	1620.0	1125.753540	00:27:00	00:18:45
27008	2220.0	2186.079834	00:37:00	00:36:26
27011	3000.0	2819.950684	00:50:00	00:46:59
27013	3720.0	3619.740723	01:02:00	01:00:19
27017	4860.0	4857.188477	01:21:00	01:20:57

Table 11: Arrival predictions made my draft RNN model

Test_ID	Input	Output	Pass/Fail
test_convert_date_type	“15th October”	“15/10/2024”	Pass
test_convert_date_value	“15th October”	d.type = string	Pass
test_extract_location_multiple_matches	“Find me the cheapest train ticket from London Liverpool Street station to Norwich”	{“depart_station”:“LONDON LIVERPOOL STREET”, “arrival_station”: “NORWICH”}	Pass
test_extract_location_single	“Find me the cheapest train ticket to Exeter Central”	{“arrival_station”:“EXETER CETNRAL”}	Pass
test_extraction_delay_time_invalid_time_format	“I want to book a ticket for the 25:00 pm train”	False	Pass
test_extraction_delay_time_no_time_found	“I want to book a ticket”	None	Pass
test_extraction_delay_time_single_time 1	“I want to book a ticket for the 10:30am train”	“10:30”	Pass
test_extraction_delay_time_single_time 2	“Please find me a train ticket for 3:00pm”	“15:00”	Pass
test_extraction_delay_time_single_time 3	“Please find me a train ticket for 14:47pm”	“14:45”	Pass
test_lemmatise_1	“I am going to the park.”	“going to park	Pass
test_lemmatise_2	“I have 5 apples and 3 oranges.”	N/A	Pass
test_lemmatise_3	“I am going from New York to Los Angeles.”	“going from new york to los angeles”	Pass
test_lemmatise_4	“Hello, @world! How are you?”	“hello world”	Fail
test_predictions	{“tpl”:25,“depart_from_LDN”:[“17:49”,“depart_from_current_station”:[“19:02”]]}	$(0.95 * y) \leq \hat{y} \leq (1.05 * y)$	Pass
test_predictions_2	{“tpl”:25,“depart_from_LDN”:[“17:49”,“depart_from_current_station”:[“19:02”]]}	$\hat{y}.instance = pd.DataFrame$	Pass

Table 12: The unit test results for conversational functions

Test_ID	Input	Output	Pass/Fail
test_LNER_scraper	“journey_data.csv”	“return.json != empty”	Pass
test_greateranglia_scraper	“journey_data.csv”	“return.json != empty”	Pass
test_myTrainTicket_scraper	“journey_data.csv”	“return.json != empty”	Fail
test_nationalrail_scraper	“journey_data.csv”	“return.json != empty”	Pass
test_southernRailways_scraper	“journey_data.csv”	“return.json != empty”	Pass
test_trainpal_scraper	“journey_data.csv”	“return.json != empty”	Fail
test_traintickets_scraper	“journey_data.csv”	“return.json != empty”	Fail

Table 13: The unit test results for web scraping functions

Test_ID	Input	Output	Pass/Fail
test_contingencyPlans	“can you tell me the contingency plan as there is full blockage between train station diss and norwich?”	“outcome!=False”	Pass

Table 14: The unit test results for contingencies

A.2 Figures

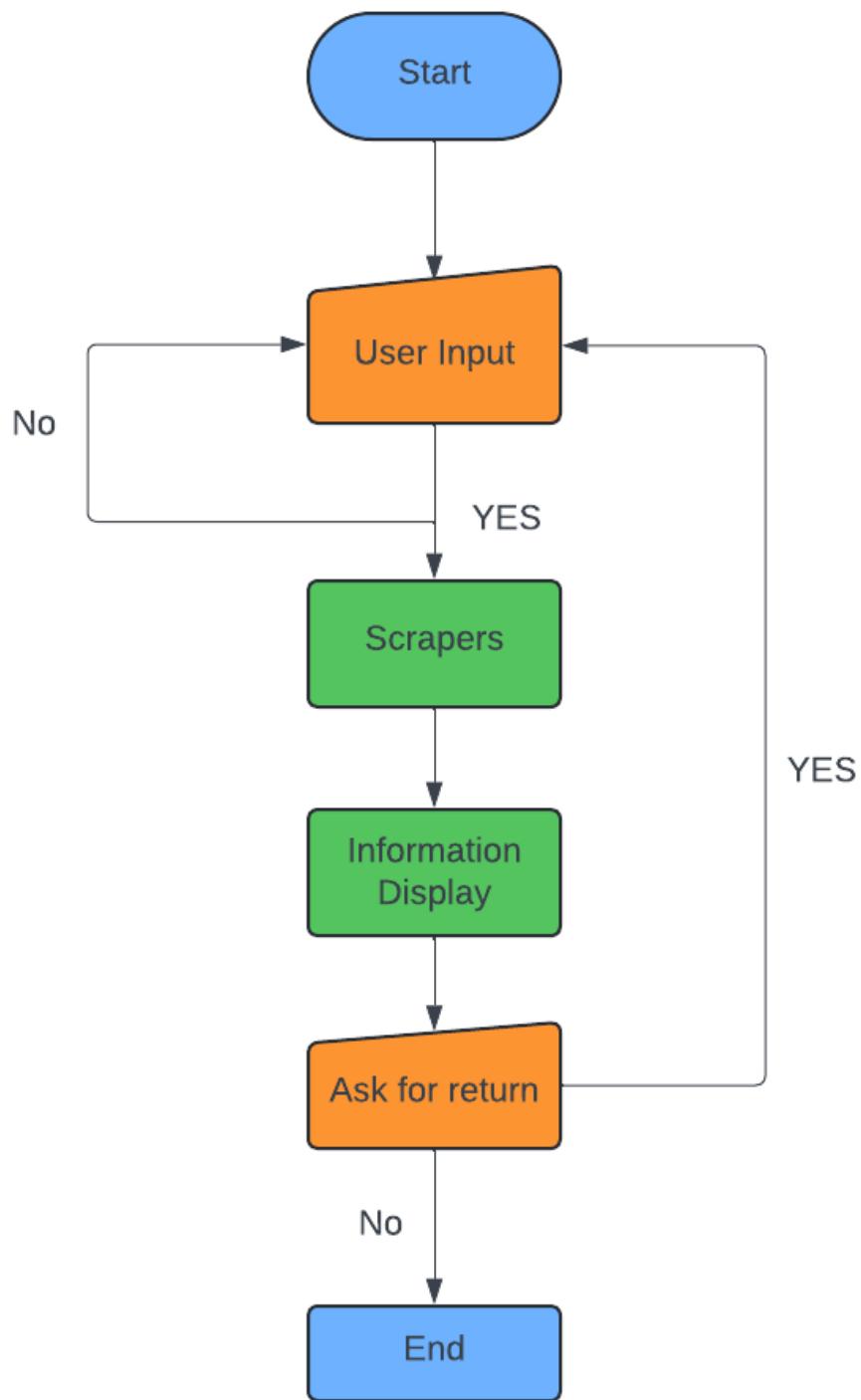


Figure 3: A high-level flowchart of the chatbot system when asking for a train ticket

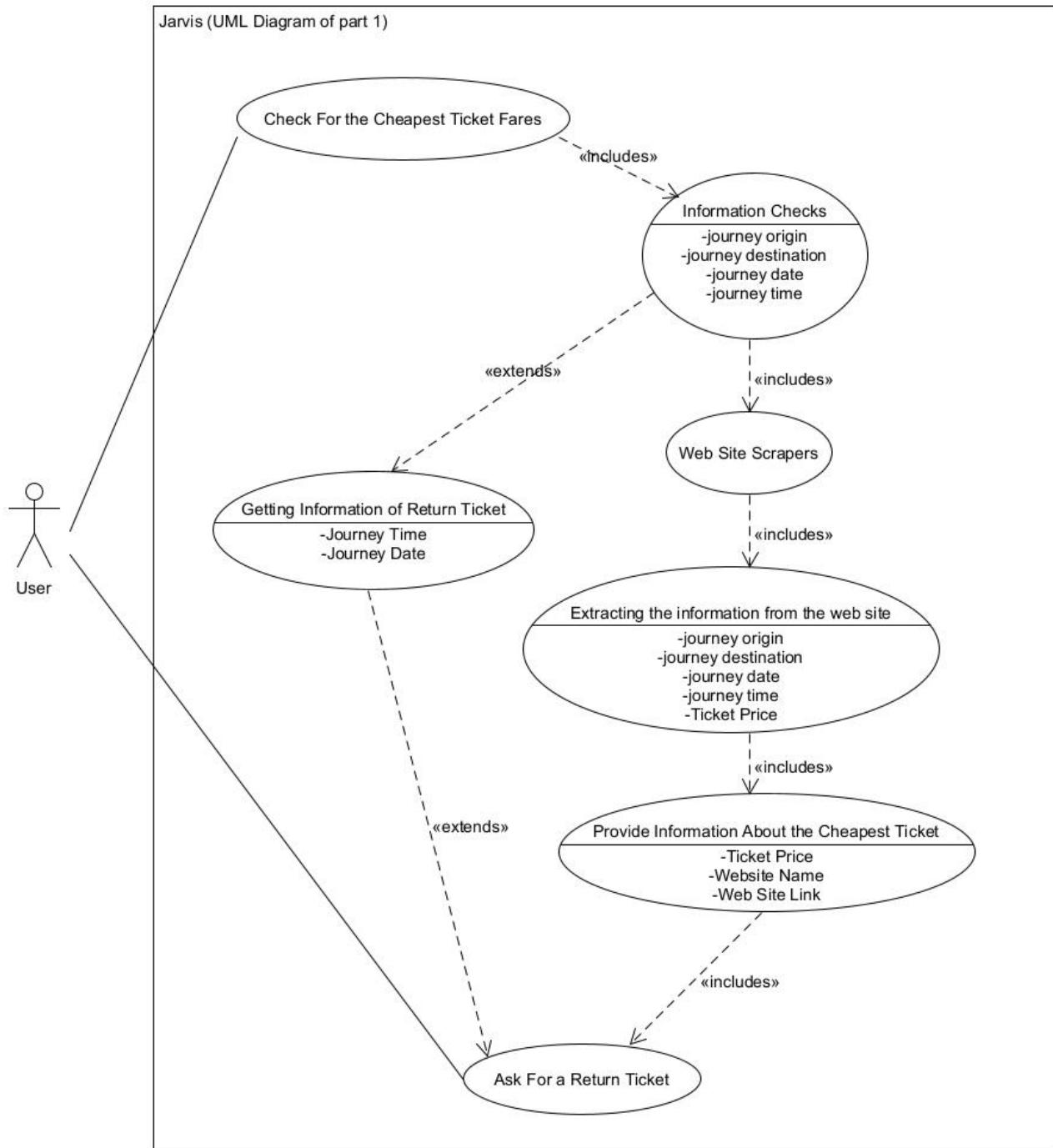


Figure 4: Use Case Diagram for completing Task A

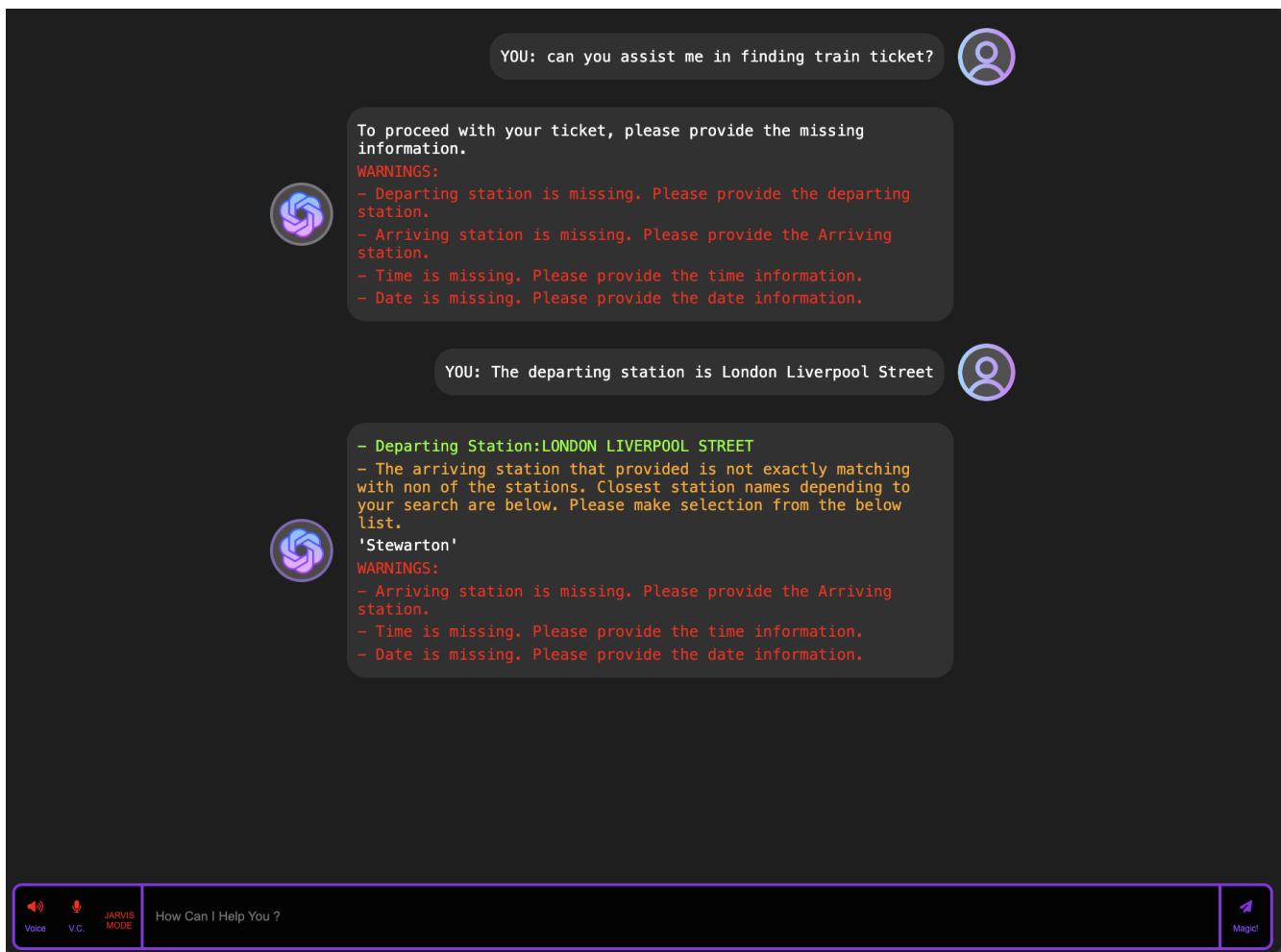


Figure 5: A screenshot from the initial GUI of the chatbot

 You

Please can you find me the cheapest train ticket from London Liverpool Street to Norwich?

 Copilot

Certainly! The **cheapest train ticket** from **London Liverpool Street to Norwich** starts at **£10** for a one-way Standard Class ticket when you book in advance [1](#) [2](#). The average journey time by train between these two stations is approximately **1 hour and 50 minutes**, covering a distance of around **97 miles (156 km)**. There are normally **40 trains per day** traveling from London Liverpool Street to Norwich, and direct trains are available [1](#). If you're looking to save even more, consider booking an **Off-Peak** ticket, which is usually cheaper than traveling during peak times. Additionally, if you have a **Railcard**, you can save up to **1/3** on eligible journeys for a whole year. So go ahead and plan your trip – whether it's for business or leisure, Greater Anglia has you covered! 🚂🌟

Related to: cheapest train ticket London Liverpool Street to Norwich

Ads ⓘ

greateranglia.co.uk | Cheap Train Tickets | Quick & Easy Train Travel
Hop on the train for the easiest journey. Save time and money - book tickets in advance. Save money and dodge the booking fees when you buy train tickets direct from us.
 <https://www.greateranglia.co.uk/tickets>

Learn more [1](#)  [thetrainline.com](https://www.thetrainline.com) [2](#)  [greateranglia.co.uk](https://www.greateranglia.co.uk) [3](#)  [mytrainpal.com](https://www.mytrainpal.com)

Like Share Reply ⏪ ⏴ ⏵ ⏲ ⏳ 1 of 30

Figure 6: An example response when asked to find live train times from London Liverpool Street to Norwich

 You

Please can you find me the cheapest train ticket from London Liverpool Street to Norwich for Monday 20th at 12:00?

 Copilot

Certainly! The **cheapest train ticket from London Liverpool Street to Norwich for Monday, 20th at 12:00** starts at £10¹. You can book in advance to secure this fare. If you're looking for even more savings, consider traveling during **Off-Peak** times or using a **Railcard** to save up to 1/3 on eligible journeys for a whole year^{1 2}. Enjoy your journey! 🚅✨

Related to: cheapest train ticket from London Liverpool Street to Norwich for Monday 20th at 12:00 Ads 

greateranglia.co.uk | Cheap Train Tickets | Quick & Easy Train Travel
Hop on the train for the easiest journey. Save time and money - book tickets in advance. Save money and dodge the booking fees when you buy train tickets direct from us.
 <https://www.greateranglia.co.uk/tickets>

Learn more 1  thetrainline.com 2  mytrainpal.com 3  greateranglia.co.uk

      ● 1 of 30

Figure 7: An example response when asked to find live train times from London Liverpool Street to Norwich at a specified date and time

Low prices found by our customers							
Mon	Tue	Wed	Thu	Fri	Sat	Sun	
			15	Today £57.70 >	17 May £22 >	18	19 £34 >
13	14						
20	21	22	23	24	25	26	
£22 >	£19 >	£19 >	£19 >	£19 >	£30.50 >	£34 >	
27	28	29	30	31	1 Jun	2	
£34.50 >	£19.50 >	£22 >	£18 >	£19 >	£22 >	£15.80 >	
3	4	5	6	7	8	9	
£18 >	£18 >	£10 > Cheapest	£15.80 >	£10 > Cheapest	£10 > Cheapest	£13.20 >	

Each fare is the lowest adult economy ticket fare found by our customers on Trainline. Fares booked further in advance than the dates displayed may be cheaper. Prices are updated daily, subject to change and indicative of price last update. Limited availability. Discount cards and coach journeys excluded, prices exclusive of booking fee.

Figure 8: TrainLine ticket booking results when following the hyper link provided with response shown in Figure 7

 You

I am currently on a train at Ipswich during a journey from London Liverpool Street to Norwich. I left London at 12:00, the train was planned to depart from Ipswich at 13:07 but we are now departing at 13:15. What time would I arrive at Norwich?

 Copilot

Your journey from **London Liverpool Street** to **Norwich** covers a distance of approximately **97 miles (156 km)** by train. Typically, there are **40 trains per day** traveling this route, and tickets start from **£10** when booked in advance. The average travel time is **1 hour and 50 minutes**.

As for your journey from **Ipswich** to **Norwich**, it's about **40 miles (65 km)**, and there are approximately **35 trains per day** between these two stations. The first train from Ipswich to Norwich departs at **06:39**, and the last one leaves at **00:45**. The journey usually takes around **41 minutes** on average, with the quickest services completing it in as little as **37 minutes**.

Given that your train from Ipswich is now departing at **13:15**, you can expect to arrive in Norwich around **14:56**. Enjoy your journey! 🚅🌟



• 2 of 30

Figure 9: The response from Bing Co-Pilot when asked to predict delayed arrival at Norwich

 You

I am a train conductor on a train between Ipswich and Stowmarket. Unfortunately there is a full line blockage contingency. What should we do?



Copilot

I apologize for the inconvenience you're facing due to the full line blockage contingency. Based on the information I found, there was a fallen tree between Ipswich and Stowmarket that caused major disruption to rail services in Suffolk. Overhead electric wires were damaged, affecting lines including London Liverpool Street to Ipswich and Norwich, Ipswich to Cambridge, and Ipswich to Peterborough services [1](#). Here are some steps you can take:

1. **Safety First:** Ensure the safety of passengers and crew. Communicate with passengers about the situation and any potential delays.
2. **Stay Informed:** Keep passengers updated on the situation. Let them know about alternative travel options, such as rail replacement services or using existing train tickets on unaffected services.
3. **Coordinate with Control Center:** Contact your control center or dispatch to report the blockage. They will provide guidance on how to proceed.
4. **Assist Passengers:** Be empathetic and assist passengers who may need help. Provide clear information about delays and alternative arrangements.
5. **Follow Company Procedures:** Adhere to your company's protocols for handling disruptions. They may have specific instructions for situations like this.

Remember, safety and communication are key during disruptions. Passengers appreciate transparency, so keep them informed as best as you can. I hope this helps, and I wish you a smooth resolution to the situation!  [2](#) [3](#)

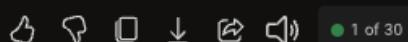
Learn more

[1](#)  bbc.co.uk

[2](#)  greateranglia.co.uk

[3](#)  greateranglia.co.uk

[4](#)  bbc.co.uk



• 1 of 30

Figure 10: The response from Bing Co-Pilot when asked for advice for a full line blockage between Ipswich and Stowmarket

```
>>> Can you please show me live train times from London Liverpool Street to Norwich?  
I'm unable to provide real-time data, but I can guide you on how to find the live train schedule for a journey  
from London Liverpool Street to Norwich. To get up-to-date train schedules, you may use the National Rail  
Enquiries website or apps like Trainline. Here is a step-by-step process:
```

1. Visit the official National Rail Enquiries (nationalrail.co.uk) website.
2. Enter "Liverpool Street to Norwich" in the train search bar.
3. Select your preferred travel date and time range if necessary.
4. The results will display available trains, including departure times from London Liverpool Street station and estimated arrival times at Norwich.

Please note that for live updates during your journey, you may want to use a smartphone app or website like Trainline which can send notifications about train statuses in real-time.

Figure 11: The response from Phi-3-mini when asked to show live train times

```
>>> I am currently on a train as Ipswich during a journey from London Liverpool Street to Norwich. I left London at 12:00, the train was planned to depart from Ipswich at 13:07 but we are now departing at 13:15. What time would I arrive at Norwich?  
To calculate your arrival time in Norwich, you'll need to know the total travel time and then adjust for any delays. Assuming there were no further delays after the one mentioned (from 13:07 to 13:15), let's estimate with an average train speed based on typical Ipswich-Norwich service times which is approximately 2 hours under normal conditions. However, since you left at a later time than initially planned (one minute delay), this may slightly affect your arrival time depending on the length of any additional delays and if there's extra travel to make up for lost time.
```

Here are the steps:

1. You departed Ipswich at 13:15.
2. Assuming a total travel time of approximately 2 hours, you would expect to arrive in Norwich around 15:15 (without considering any additional delay).

However, if there's an extended delay after the departure delay or potential compensatory speed adjustments by the train company, your arrival could be slightly earlier or later. It is best to check with the latest travel updates from Ipswich for a precise estimated time of arrival.

Figure 12: The response from Phi-3-mini when asked to predict a delayed arrival at Norwich

o >>> [200~I am a train conductor on a train between Ipswich and Stowmarket. Unfortunately there is a full line blockage contingency. What should we do?

As the train conductor facing an unexpected full-line blockage, you must act swiftly to ensure passenger safety, compliance with regulations, and communication:

1. Assess the situation immediately: Determine if passengers can safely disembark or require assistance. If it's safe for them to stay on board, proceed cautiously but calmly.
2. Inform passengers: Use public announcement systems to inform passengers of the blockage and your next steps clearly. Be transparent about the situation while maintaining a calm demeanor to prevent panic.
3. Coordinate with dispatch: Contact train control and other relevant authorities for information on alternative routes, scheduling adjustments, or safety measures they recommend.
4. Consider rerouting options: Collaborate with dispatch to explore possible detours, if available, keeping in mind time constraints, passenger comfort, and logistical feasibility (e.g., connecting services).
Written by a professional for the passengers' understanding.
6. Compliance and reporting: Document all actions taken, decisions made, and communications with authorities. This documentation will be critical for any subsequent investigations or reports that need to be filed.
7. Review policies: Post-event review should take place to learn from this incident, adapting your protocols as needed to improve future responses.

Figure 13: The response from Phi-3-mini when asked for advice for a full line blockage between Ipswich and Stowmarket

Live tracker

[Close](#)

Thu 23 May, 12:00 → 13:46

⚠ Disruption Alert

^

The escalators between the Bishopgate entrance and the concourse are out of order at London Liverpool Street station.

 Greater Anglia LE0114

 6 bike spaces • [Reservation required](#)

12:00 London Liverpool Street

Plat. 10 estimated

1h 46m [Hide stops](#)

12:47 Colchester

12:55 Manningtree

13:07 Ipswich

13:29 Diss

13:46 Norwich

Plat. 2 estimated

Figure 14: An example of a train journey from London Liverpool Street to Norwich departing 23/05/24 at 12:00 taken from TrainLine

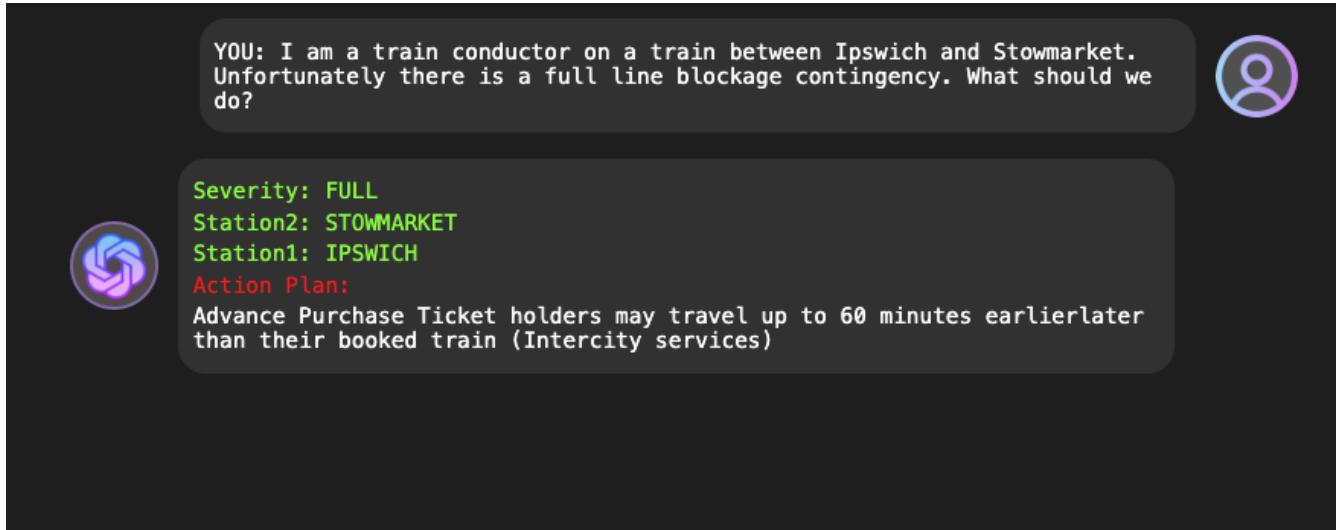


Figure 15: The response from our bespoke chatbot when asked to for advice for a full line blockage between Ipswich and Stowmarket

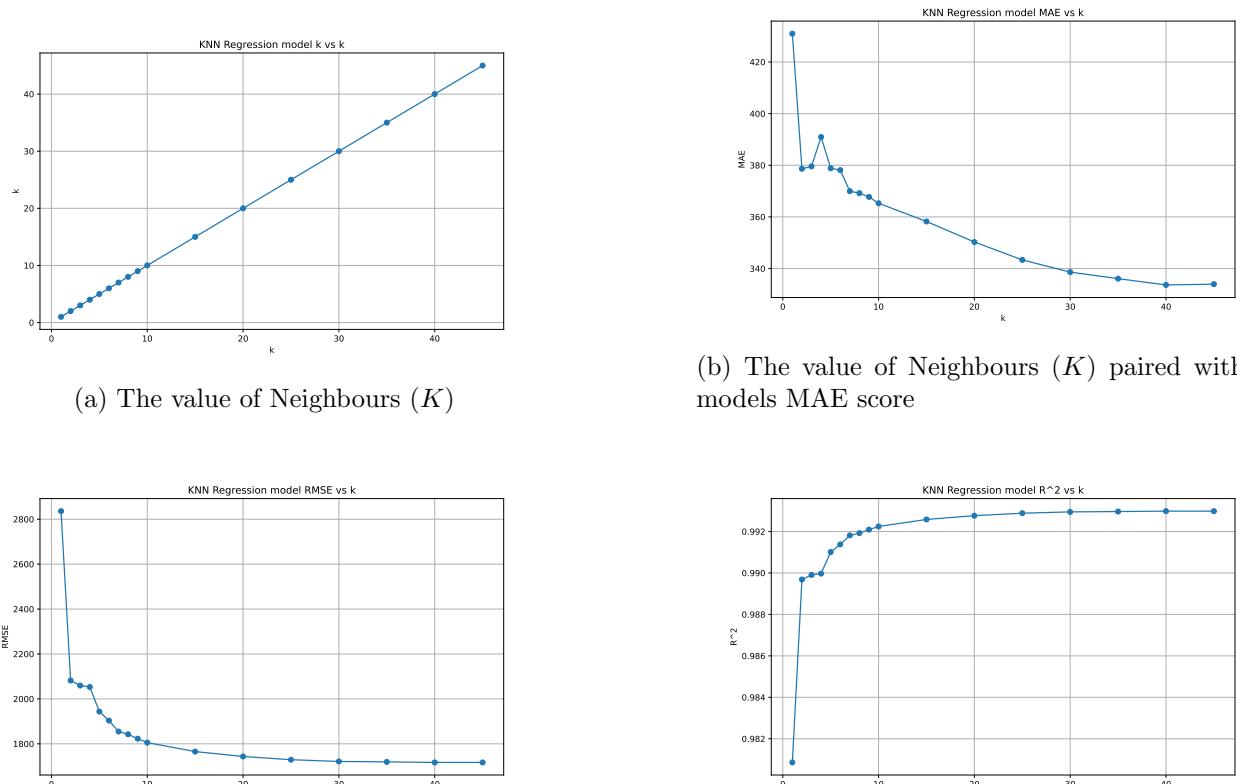
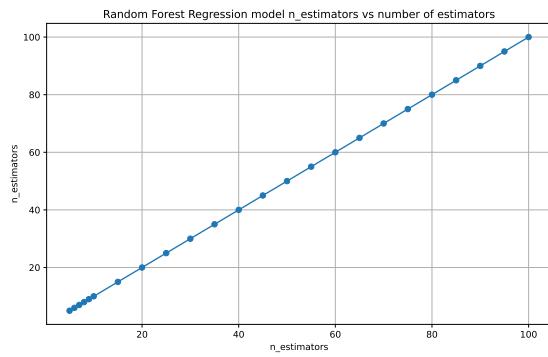
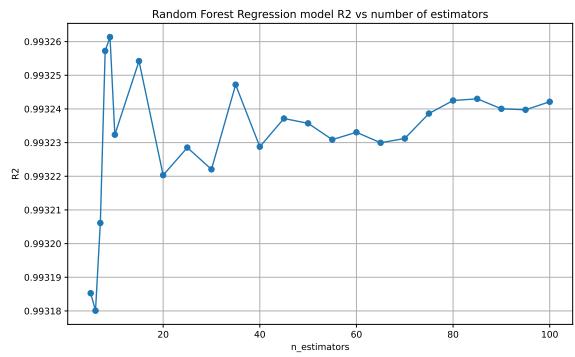


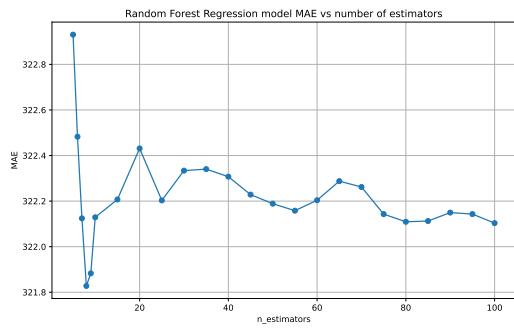
Figure 16: KNN Regression model plots



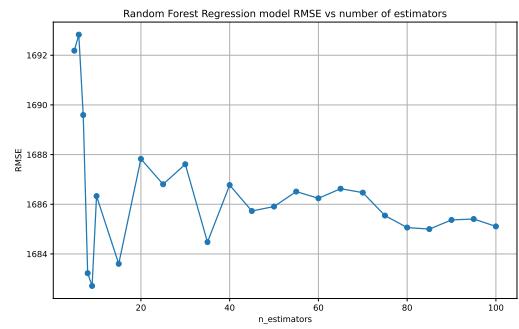
(a) The number of estimators



(b) R^2 vs number of estimators

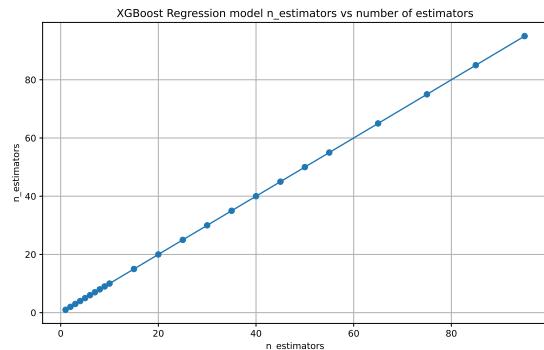


(c) MAE vs number of estimators

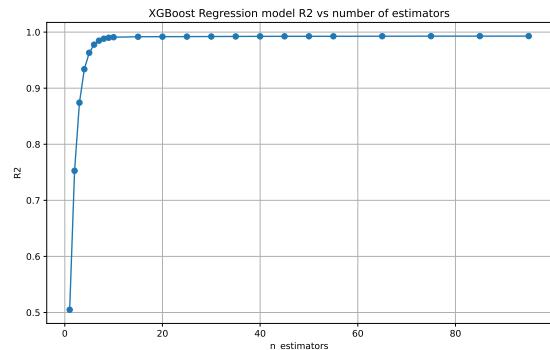


(d) RMSE vs number of estimators

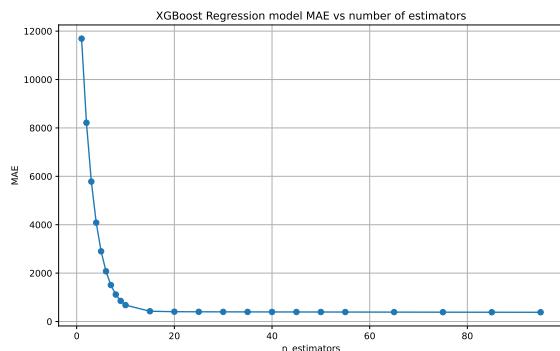
Figure 17: Random Forest regression model results plots



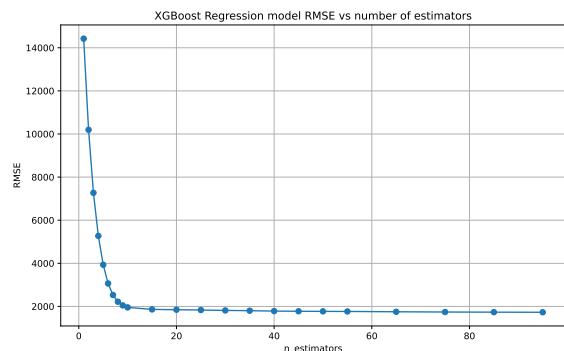
(a) The number of estimators



(b) R^2 score per number of estimators

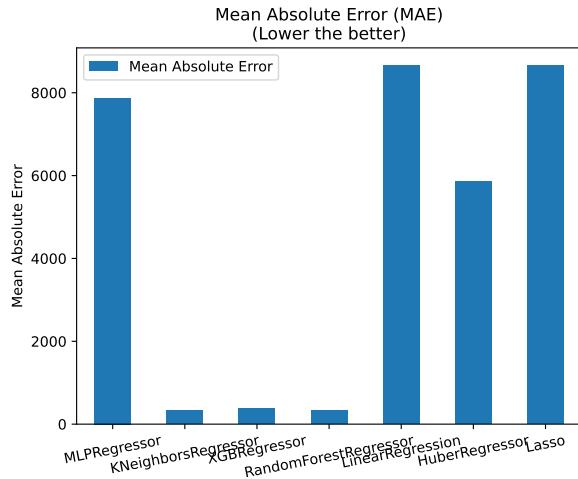


(c) MAE score per number of estimators

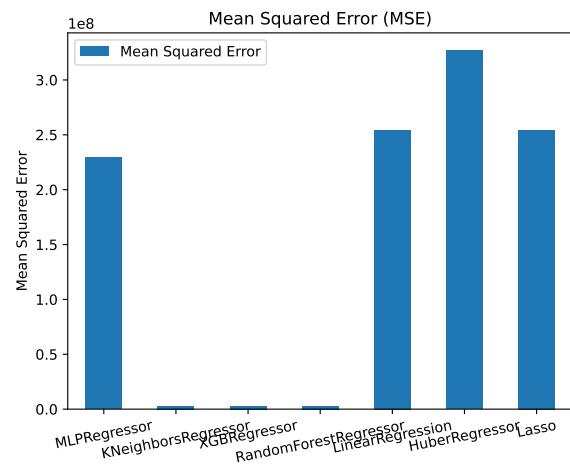


(d) RMSE vs number of estimators

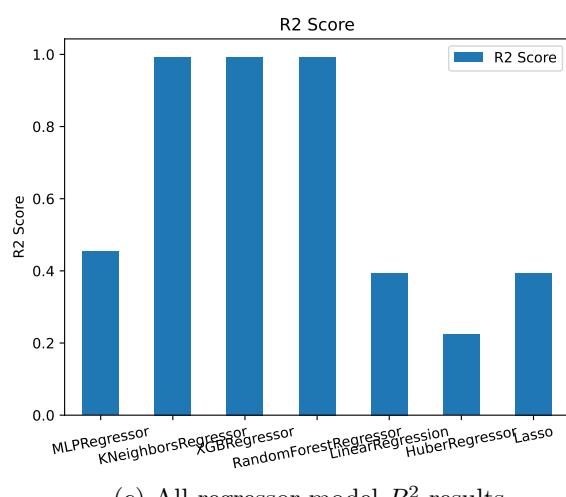
Figure 18: XGBoost regression model results plots



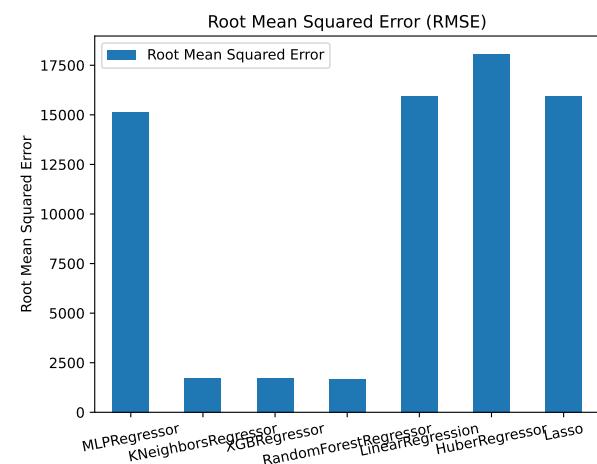
(a) All regressor model mean absolute error results (lower the better)



(b) All regressor model squared absolute error results (lower the better)



(c) All regressor model R^2 results



(d) All regressor model root mean squared error results (lower the better)

Thu 25 Apr 2024: 19:30 London Liverpool Street to Norwich

Service Summary

IDs: 1P60 521P60MY25 G17751
Type: Express Passenger
Operator: LE: Greater Anglia
Power Type: EMU: Electric Multiple Unit
Sched Speed: 100mph
Class(es): First and Standard

Service Calling Points

#	Station	Sch. Arr	Actual Arr	Sch. Dep	Actual Dep
1	London Liverpool Street			19:30	19:29 RT
2	Colchester	20:19	20:17 RT	20:20	20:20 RT
3	Manningtree	20:28	20:27 RT	20:29	20:37 8L
4	Ipswich	20:40	20:45 5L	20:41	20:46 5L
5	Stowmarket	20:51	20:56 5L	20:52	20:57 5L
6	Diss	21:03	21:07 4L	21:04	21:08 4L
7	Norwich	21:21	21:24 3L		

Data reflects train services up to: **15:34 on Thu 9 May 2024**.

Figure 20: An example of a delayed train journey

Thu 02 May 2024: 17:50 London Liverpool Street to Norwich

Service Summary

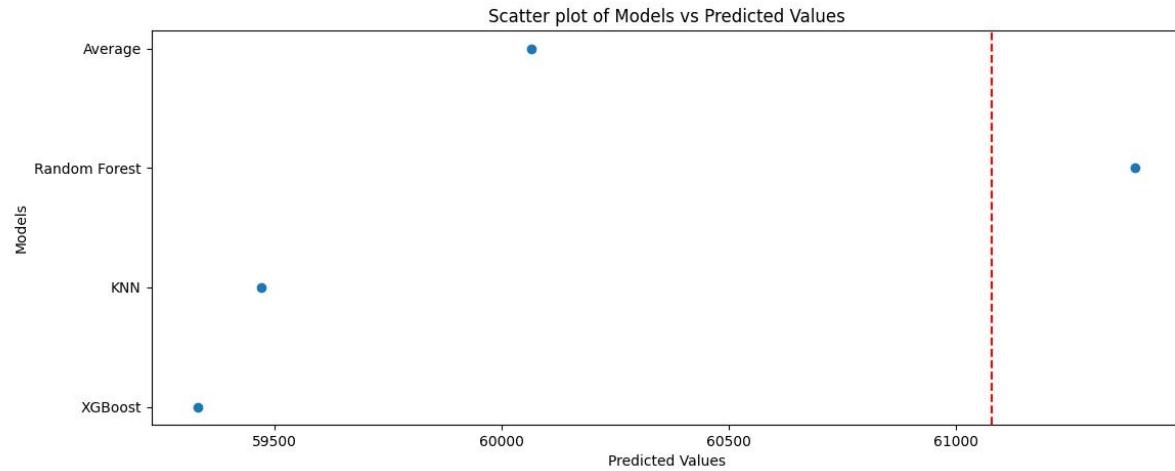
IDs: 1P52 521P52MV02 G17731
Type: Express Passenger
Operator: LE: Greater Anglia
Power Type: EMU: Electric Multiple Unit
Sched Speed: 100mph
Class(es): First and Standard

Service Calling Points

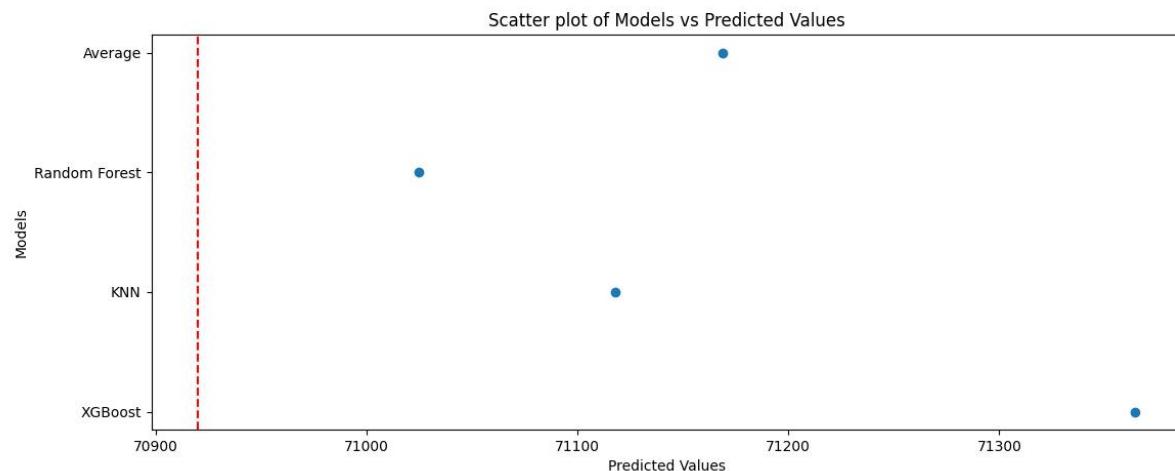
#	Station	Sch. Arr	Actual Arr	Sch. Dep	Actual Dep
1	London Liverpool Street			17:50	17:49 RT
2	Colchester	18:42	18:40 RT	18:43	18:42 RT
3	Manningtree	18:50	18:49 RT	18:51	18:50 RT
4	Ipswich	19:01	19:00 RT	19:03	19:02 RT
5	Stowmarket	19:13	19:13 RT	19:14	19:14 RT
6	Diss	19:26	19:25 RT	19:27	19:26 RT
7	Norwich	19:46	19:43 RT		

Data reflects train services up to: **15:34** on **Thu 9 May 2024**.

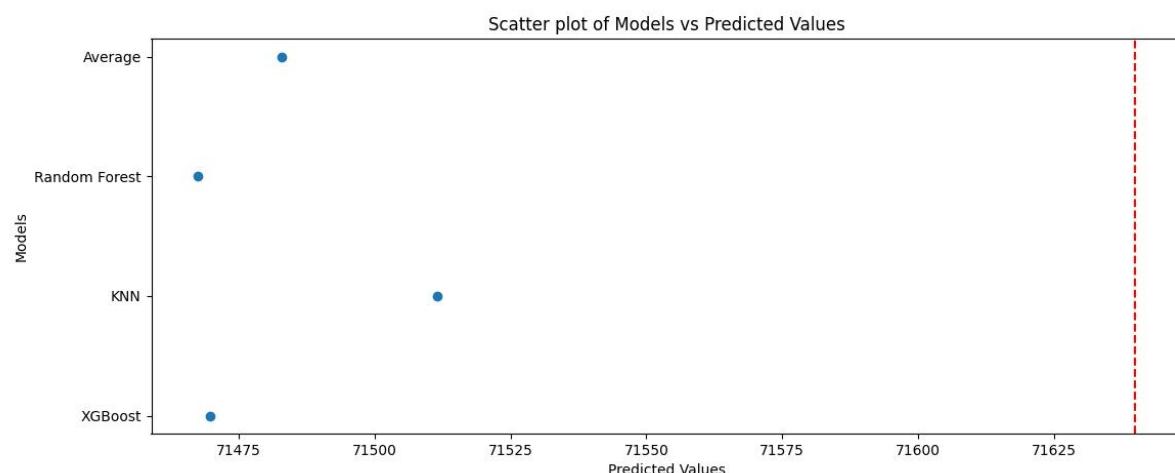
Figure 21: An example of an on-time train journey



(a)



(b)



(c)

Figure 22: Regression model predicted arrival time at Norwich from London Liverpool Street if delayed at Ipswich. The red dotted vertical line is ground truth.

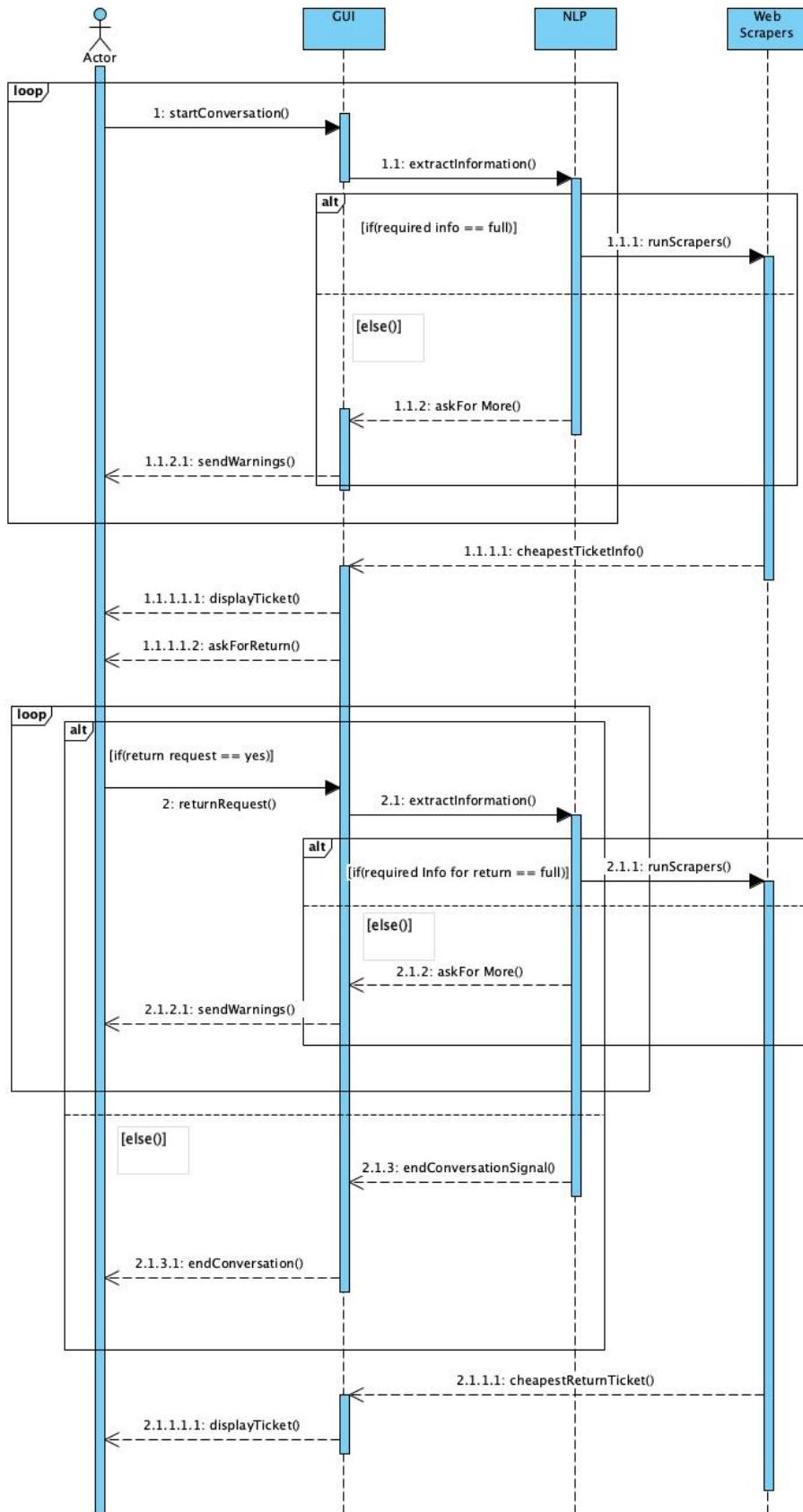


Figure 23: A sequence diagram for the chatbot's ticket scraping mechanism

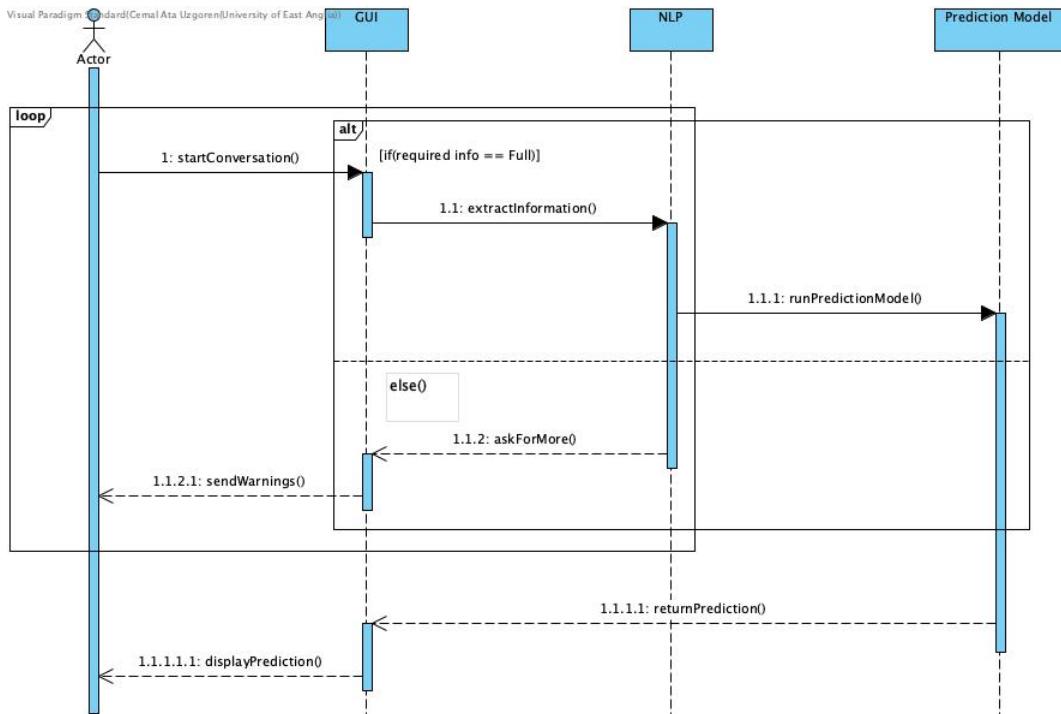


Figure 24: A sequence diagram for the chatbot's delayed journey mechanism

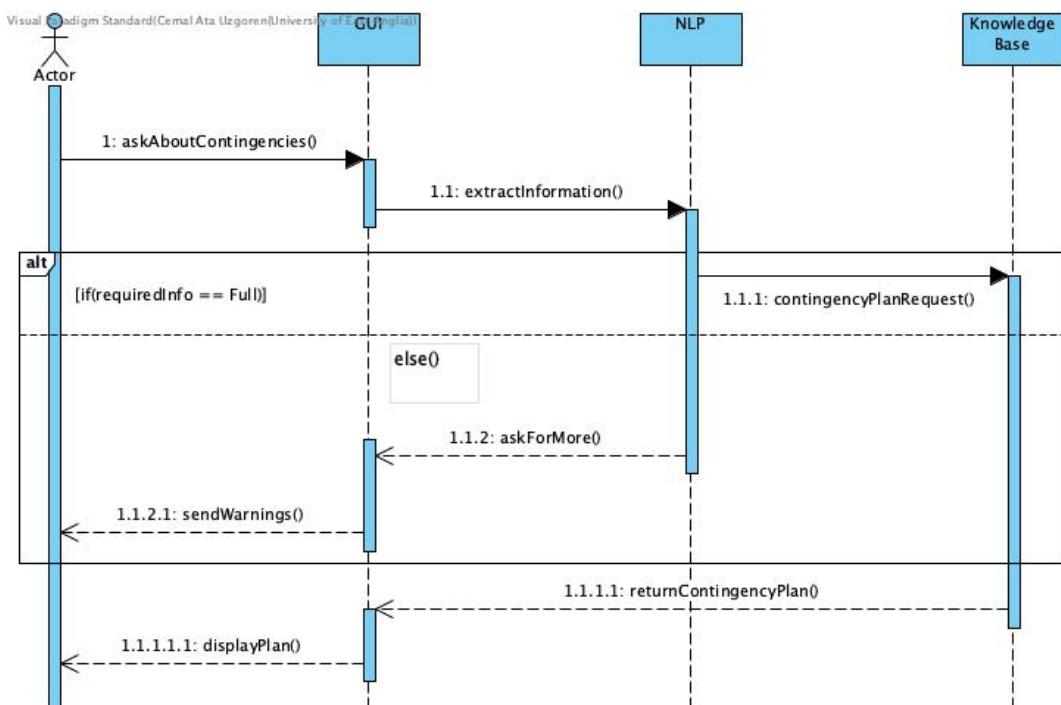


Figure 25: A sequence diagram for the chatbot's contingency mechanism

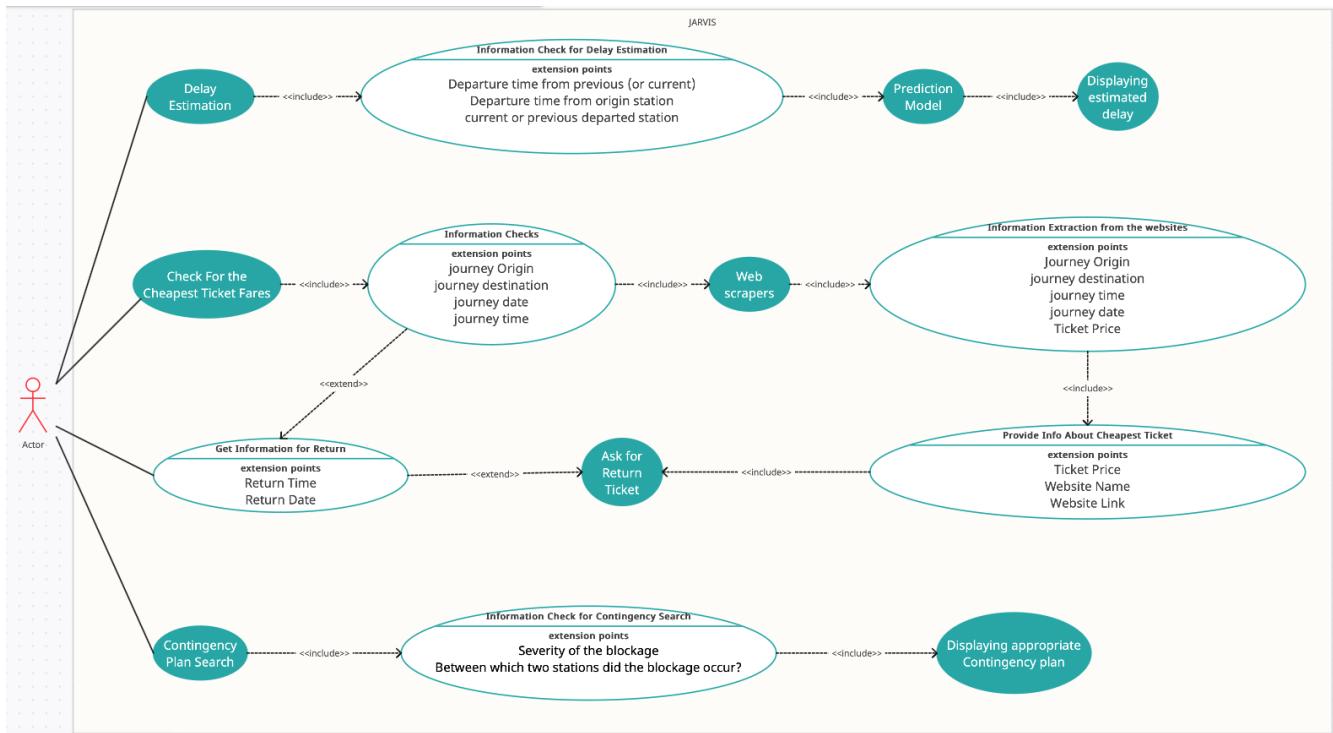


Figure 26: A use case diagram for the entire chatbot system

Unittest Results

Start Time: 2024-05-28 11:22:39

Duration: 32.17 s

Summary: Total: 23, Pass: 19, Fail: 4

	Status
__main__.TestChatbotConversation	
test_convert_date_type	Pass
test_convert_date_value	Pass
test_extract_location_multiple_matches	Pass View
test_extract_location_single	Pass View
test_extraction_delay_time_invalid_time_format	Pass View
test_extraction_delay_time_no_time_found	Pass View
test_extraction_delay_time_single_time1	Pass View
test_extraction_delay_time_single_time2	Pass View
test_extraction_delay_time_single_time3	Pass View
test_lemmatise_1	Pass
test_lemmatise_2	Pass
test_lemmatise_3	Pass
test_lemmatise_4	Fail View
test_predictions	Pass
test_predictions_2	Pass

Total: 15, Pass: 14, Fail: 1 -- Duration: 19.29 s

	Status
__main__.TestScrapers	
test_LNER_scraper	Pass
test_greateranglia_scraper	Pass
test_myTrainTicket_scraper	Fail View
test_nationalrail_scraper	Pass
test_southernRailways_scraper	Pass
test_trainpal_scraper	Fail View
test_traintickets_scraper	Fail View

Total: 7, Pass: 4, Fail: 3 -- Duration: 10.37 s

	Status
__main__.TestContingencies	
test_contingencyPlans	Pass View

Total: 1, Pass: 1 -- Duration: 2.50 s

Figure 27: Results from the unit tests executed during the development of the chatbot

A.3 Group Work

Team Member	Contribution (%)
Cemal Ata Uzgoren	33.33%
Aman Seth	33.33%
Joshua Newton	33.33%

Table 15: Team Members and their Contributions

A.3.1 Trello

Link of the Trello page: <https://trello.com/b/CipZLfxi/cmp-7028b-assignment-02>

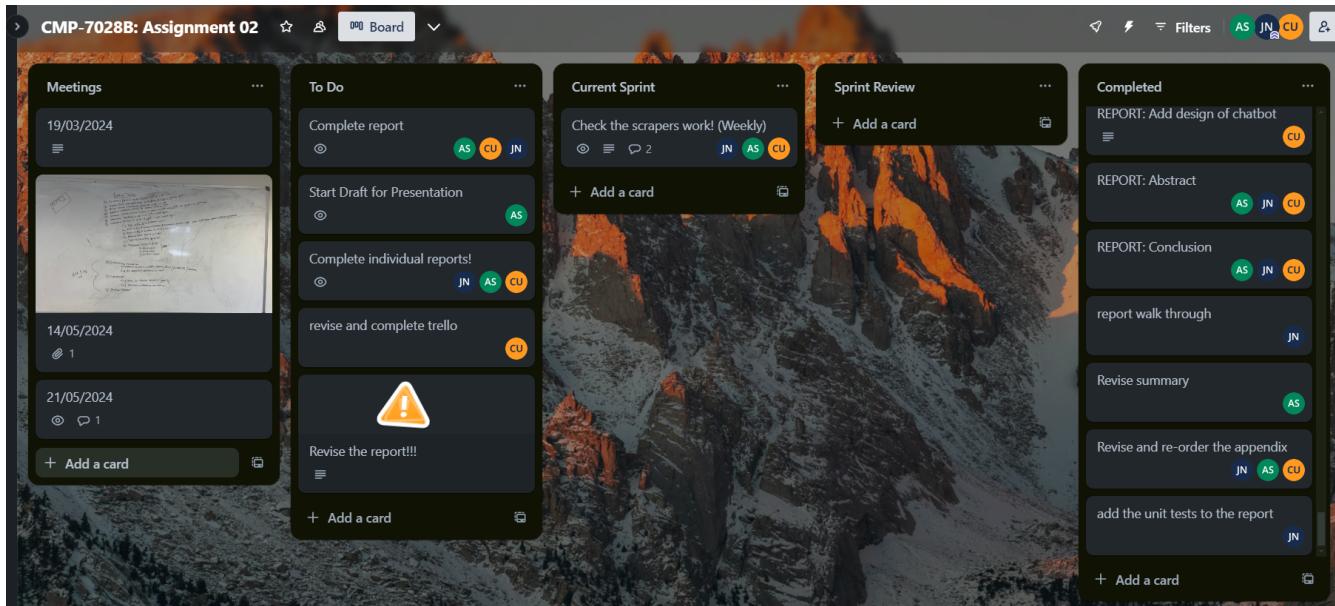


Figure 28: Trello Board

Description

Part C: Contingencies Plans

- Implementation to be done as per brief: **Please Note: you need only consider blockages occurring between Colchester and Norwich in Greater Anglia's Great Eastern Tracks, specified in document D3, Sections 1 and 2.**
- Rules Enginer:** JSON to be implemented with blockage type as parent key, followed by locations and the alternate plan suggestions in the values per station pair.

Sample

```

1  {
2      "Partial": {
3          "Stations": {
4              "Colchester,Manningtree": [
5                  "Advance Purchase Ticket holders may travel up
6                  "Off-Peak: None train service operates as revis
7              ],
8              "Manningtree,Ipswich": [
9                  "Customers London-Norwich to travel via Cambrid
10             ],
11              "Ipswich,Stowmarket": [
12                  "Local Bus 87/88 operates all stations Ipswich
13             ],
14              "Stowmarket,Diss": [
15                  "Local Bus 459 operates Stowmarket to Diss."

```

Edit

Labels
 Checklist
 Dates
 Attachment
 Cover
 Custom Fields

Power-Ups
+ Add Power-Ups

Automation
(i)

+ Add button

Actions

Move
 Copy
 Make template

Archive
 Share

Figure 29: Trello Card - Part C

AS

Aman Seth 19 May at 09:20

Location Extraction

- Finalised strategy is to fetch locations by using `fuzzywuzzy` by matching every word of the input to the stations.csv and finding the exact or closest match.
- If exact match is found, return `true` with stations `dict`. If multiple matches found, return `false` with multiple matches list. If not found, return empty list with status `false`.

Date Extraction

- Finalised strategy is to try fetching location via spacy's large model `en_core_web_lg`. If empty `list` is returned, the use `regex` patter matching using `re` python module.
- `date_pattern_1 = r'\b(January|February|March|April|May|June|July|August|September|October|November|December)\s+(\d{1,2})\s+(\d{4})\b'`
- `date_pattern_2 = r'\b(\d{1,2}(?:st|nd|rd|th)?\s+(?:January|February|March|April|May|June|July|August|September|October|November|December)|(?:January|February|March|April|May|June|July|August|September|October|November|December)\s+\d{1,2})\s+(\d{4})\b'`
- `date_pattern_3 = r'\b\d{1,2}(?:st|nd|rd|th)?\s+(?:january|february|march|april|may|june|july|august|september|october|november|december)\b'`

Time Extraction

- Finalised strategy is to fetch time from user input using python `regex` matching via `re` module.
- First pattern: `time_pattern = r'\b(\d{1,2})(?::(\d{2}))?\s*(?:am|pm)\b'`
- Second pattern: `time_pattern = r'\b(\d{1,2})(?::(\d{2}))?(?:\s*(?:am|pm))?\b'`
- The above regex match pattern 10th may 2024 or 10 may, 23rd june etc

Figure 30: Trello Card - Name Entity Recognition

A.3.2 Github

Link of the Github page : <https://github.com/Staying-Inside/chatbot.git>

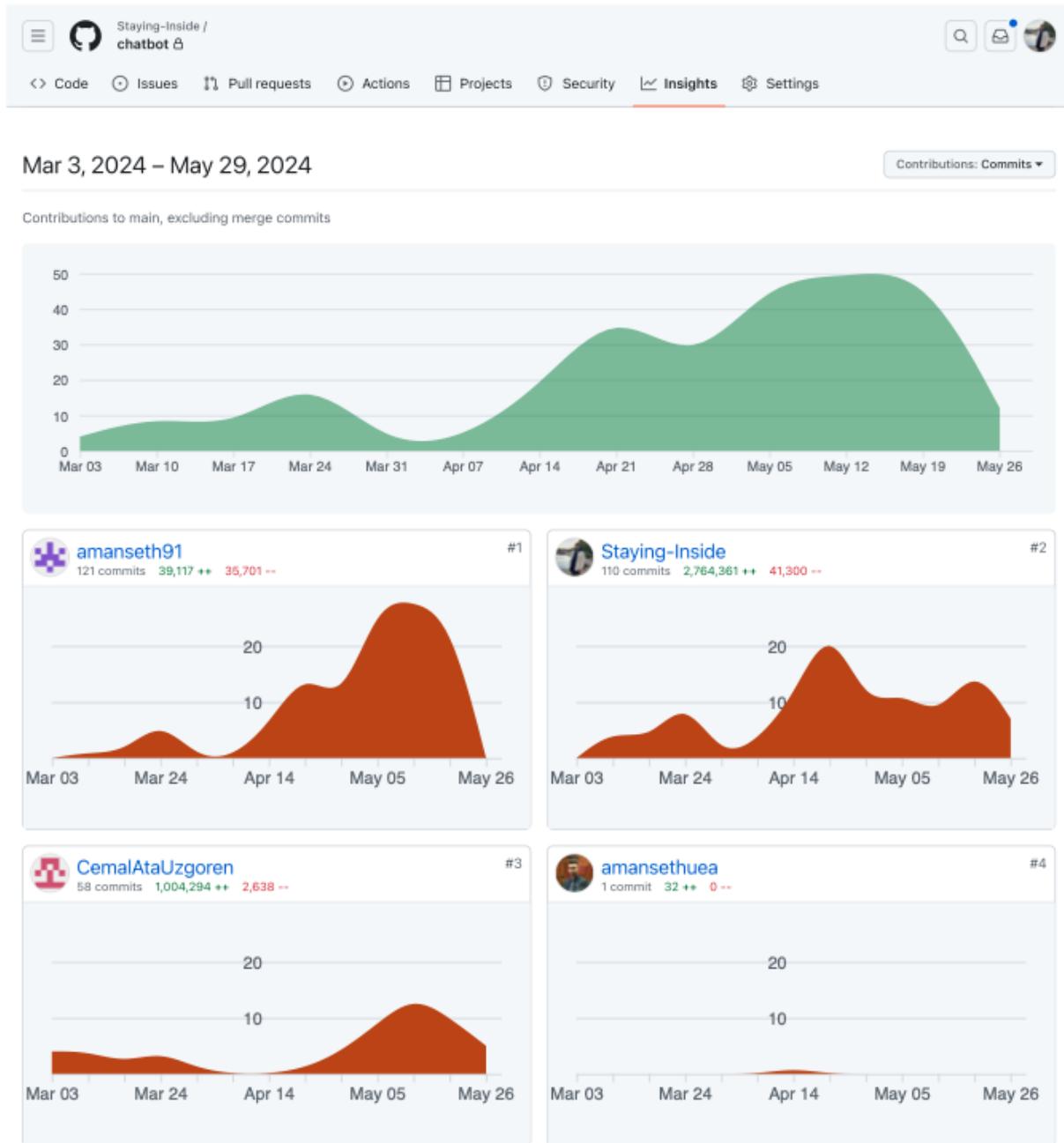


Figure 31: Insights of the project repository

Commits

main ▾

All users ▾ All time ▾

Commits on May 21, 2024

Merge pull request #62 from amansethuea/UI_update
amansethuea committed last week

Verified 375b237 ⌂ ⌄

Gui fixes and updates
CemalAtaUzgoren committed last week

8c64c30 ⌂ ⌄

Merge pull request #61 from amansethuea/creating-unit-tests
Staying-Inside committed last week

Verified cd2e0f3 ⌂ ⌄

added a few unit tests, also changed regression output to ignore the seconds.
Staying-Inside committed last week

a33b7d7 ⌂ ⌄

Merge pull request #60 from amansethuea/GUI-redesign-fix
Staying-Inside committed last week

Verified ca27c39 ⌂ ⌄

fin.
Staying-Inside committed last week

ba124bf ⌂ ⌄

Commits on May 20, 2024

Completed TODOs in Introduction.txt
amanseth91 committed last week

3779daa ⌂ ⌄

NLP implementation complete in report
amanseth91 committed last week

ca9c915 ⌂ ⌄

The screenshot shows a list of commits in a GitHub repository. At the top, there are dropdown menus for 'main' and 'All users' with 'All time'. Below that, a section for 'Commits on May 21, 2024' is shown. It includes a merge commit for pull request #62, which was merged by amansethuea and is marked as 'Verified'. Another commit by CemalAtaUzgoren adds GUI fixes. A third commit by Staying-Inside adds unit tests and ignores seconds in regression output. A fourth commit by Staying-Inside is a merge for pull request #60, also marked as 'Verified'. A final commit by Staying-Inside marks the completion of TODOs in the Introduction.txt file. Below this, a section for 'Commits on May 20, 2024' shows two more commits from amanseth91, one for completing the NLP implementation and another for TODOs.

Figure 32: Commit screen of the project repository

A.3.3 Design phase

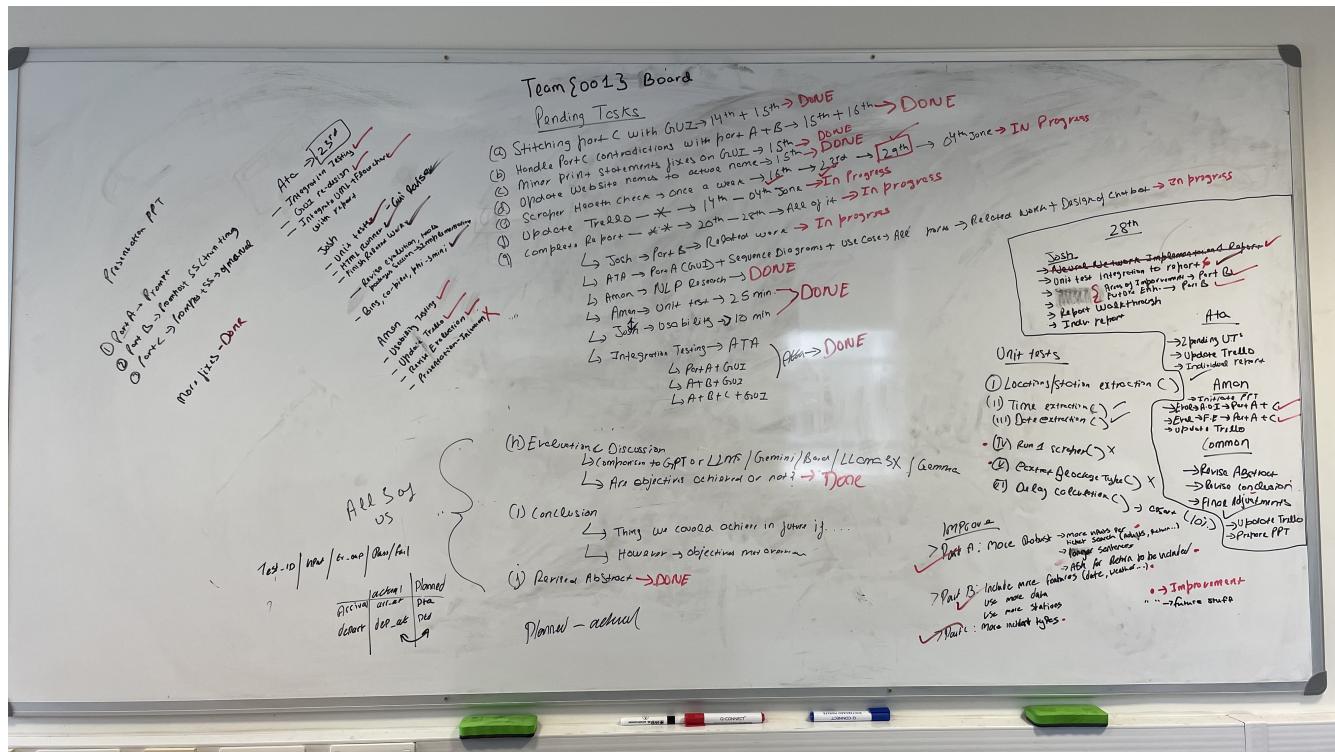


Figure 33: Work-load distribution for reporting

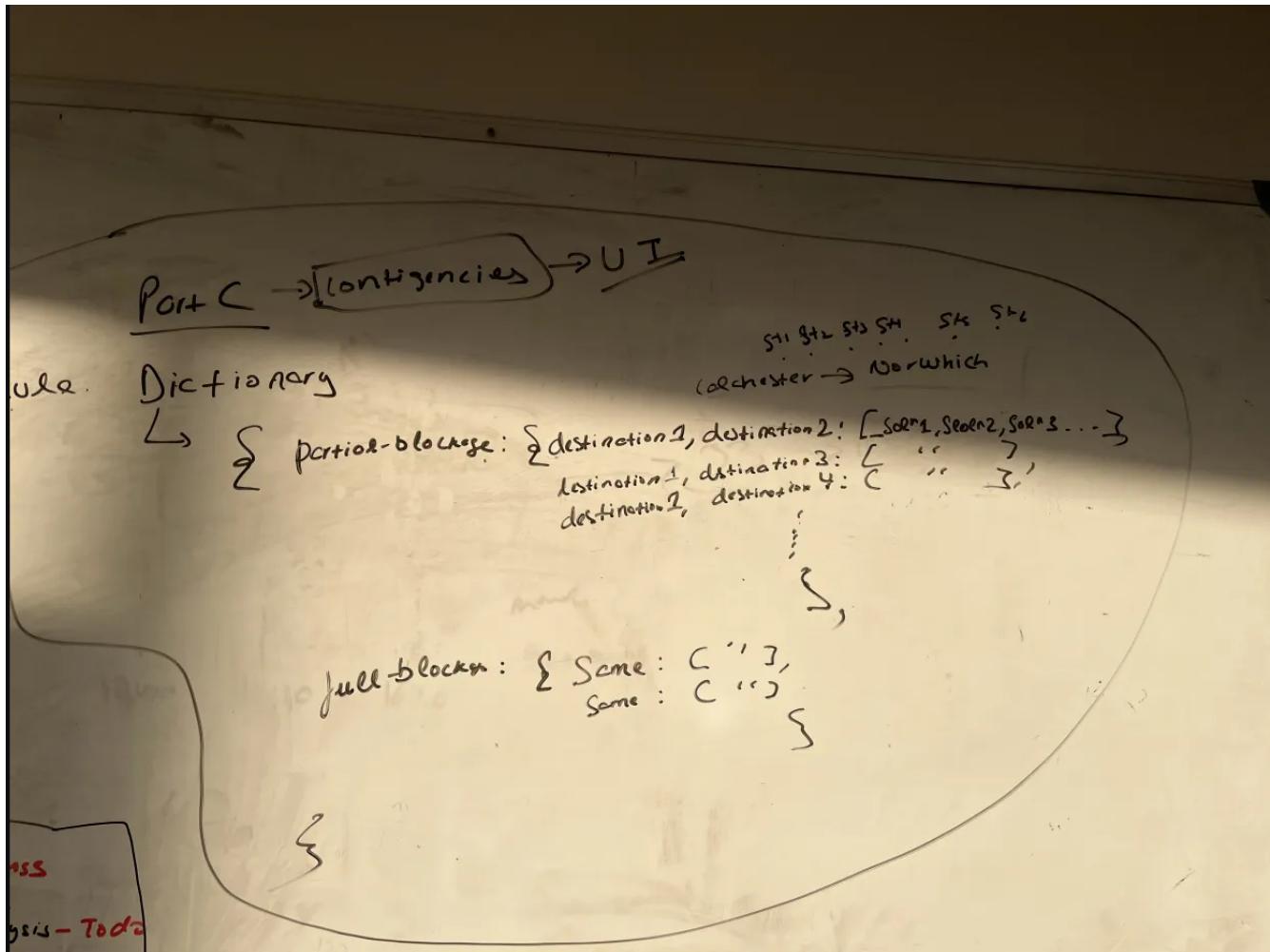


Figure 34: Part-3 plan

A.3.4 Coding Phase

Spawning Python via JS (GUI)

```

setTimeout(function(){
  const reunion=split_user_input.join(' ')
  const python = spawn('python', ['chatbot_conversation.py', JSON.stringify(user_input)]); // sending the request (user input) to the python file to generate responses based on that.
  python.stdout.on('data', (data) => {// fetching outputs of print statements to the node terminal.
    console.log(`***Python script's print outputs:***\n ${data}`);
  });
  python.on('error', (err) => {//error detection in the python code.
    console.error('Python error:', err);
  });

  python.on('close', (code) => {// fetching the results from python code
    const result = fs.readFileSync('output.txt', 'utf-8');
    console.log(result)
    res.send({message: result.toString('utf-8')});      CemalAtaUzgoren, 3 weeks ago • Auto typo fix
  });
},1000)

```

Figure 35: Code: Spawning Python NLP code via JS

NLP - Date Extraction

```

def extract_date_via_nlp(self, text):
    lemma_text = self.lemmatise_and_clean(text)
    try:
        doc = self.nlp(lemma_text)
        dates = []
        for ent in doc.ents:
            if ent.label_ == 'DATE':
                _ = self.convert_date(ent.text)
                dates.append(_)
    print(f'DATES EXTRACTED BY NLP: {dates}')
    if not dates:
        dates = self.date_extraction_via_regex(text)
        input_date = datetime.strptime(dates[0], "%d/%m/%Y")
        today = datetime.today().replace(hour=0, minute=0, second=0, microsecond=0)
        if input_date >= today:
            return dates
        else:
            print("Date can't be of previous day.")
            self.fo.write("CMP_7028BPastDateCMP_7028BDate can't be of previous day")
            return False
    except ValueError:
        dates = self.date_extraction_via_regex(text)
        input_date = datetime.strptime(dates[0], "%d/%m/%Y")
        today = datetime.today().replace(hour=0, minute=0, second=0, microsecond=0)
        if input_date >= today:
            return dates
        else:
            print("Date can't be of previous day.")
            self.fo.write("CMP_7028BPastDateCMP_7028BDate can't be of previous day")
            return False

```

Figure 36: Code: Date Extraction Mechanism

Prediction Model

```
def model_loading_and_prediction(self):
    df = self.time_conversion_and_encoding()
    print("Predicting Delay..")
    final_delay_time = regression_model_functions.make_delay_prediction(df, str)
    print(final_delay_time)
    self fo.write(f"CMP_7028BDelayPrediction:CMP_7028B{final_delay_time}")
    return final_delay_time
```

Figure 37: Code: Model Prediction for Part B

Blockage Type Extraction

```
def contingency_type_extraction(self, user_input):
    type_dictionary = {}
    text = self.lemmatise_and_clean_contingency(user_input)
    text = text.upper()

    matches = []
    for type in self.blockage_type:
        if type.upper() in text.upper():
            matches.append(type.upper())

    sent = text.split()
    new_match = []
    for t in matches:
        t = t.split()
        new_match.extend(t)
    for s in new_match:
        if s not in sent:
            matches.remove(s)

def sub_function():
    multiple_matches = []
    # temporarily remove to, from and other words related to departure or arrival
    departure_words = ["from", "depart", "departure", "departing", "city", "visiting"]
    arrival_words = ["to", "arrive", "arrival", "arriving", "book", "ticket", "train", "Colchester", "Manningtree", "Ipswich",
    "Stowmarket", "Diss", "Norwich"]
    # Convert text to lowercase for case-insensitive matching
    text_lower = text.lower()
    # Split the text into words
    words = text_lower.split()
    # Filter out words related to departure and arrival
    filtered_words = [word for word in words if word not in departure_words and word not in arrival_words]
    # Join the remaining words back into a string
    cleaned_text = ' '.join(filtered_words)
    #from fuzzywuzzy import fuzz
```

Figure 38: Code: Blockage Type Extraction in Part C

A.3.5 Testing Phase

Testing Scenario: Part A and B integrity constraint The below tests demonstrate that the user cannot directly switch conversation to part B (Delay prediction) if there is an on-going conversation regarding part A (Cheapest train fare search) in place. The user may or may not reset the search explicitly in order to proceed.

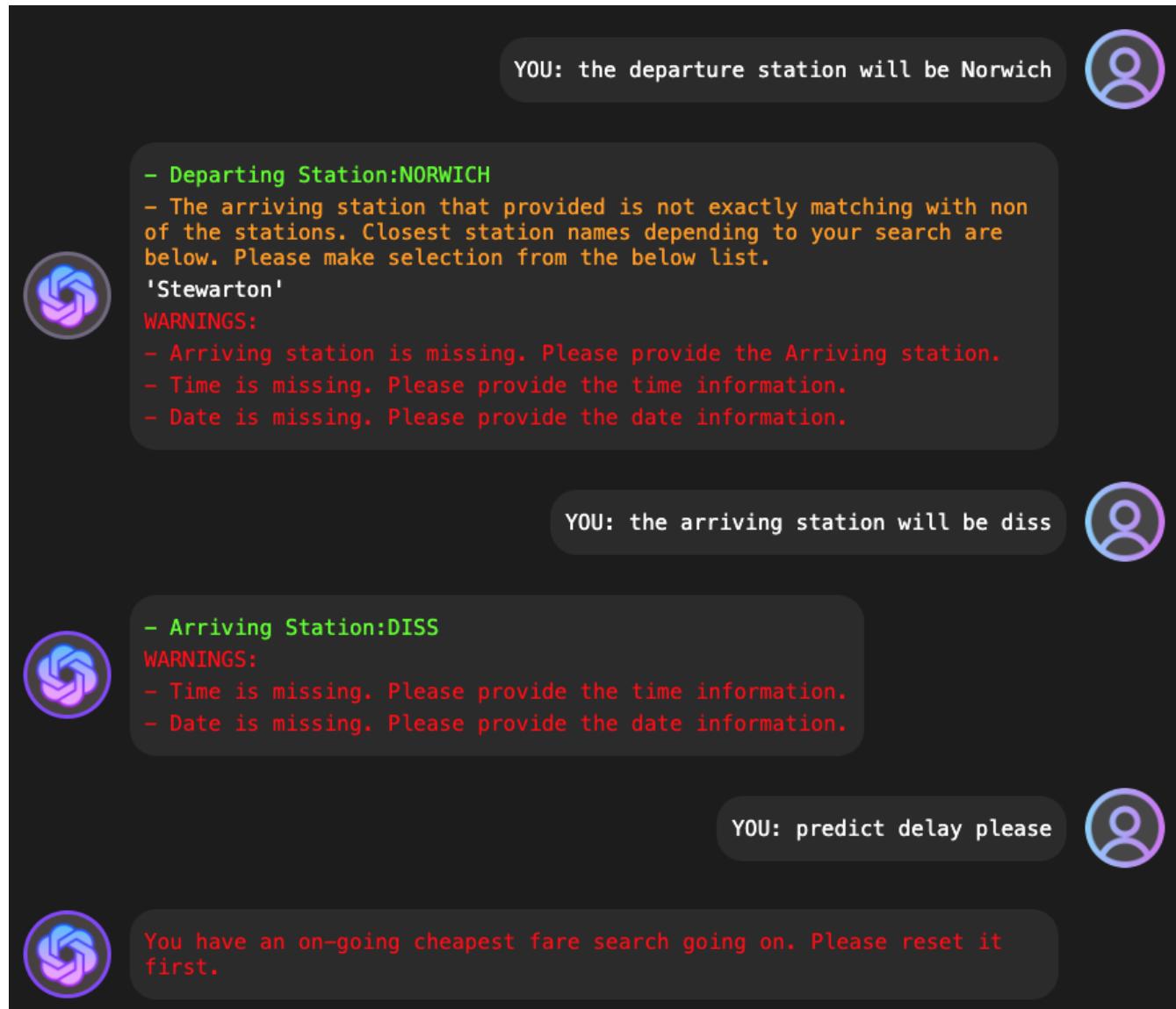


Figure 39: Test: Part A and B Integrity Constraint Check

Testing Scenario: Part B and A integrity constraint The below tests demonstrate that the user cannot directly switch conversation to part A (Cheapest train fare search) if there is an on-going conversation regarding part B (Delay prediction) in place. The user may or may not reset the search explicitly in order to proceed.

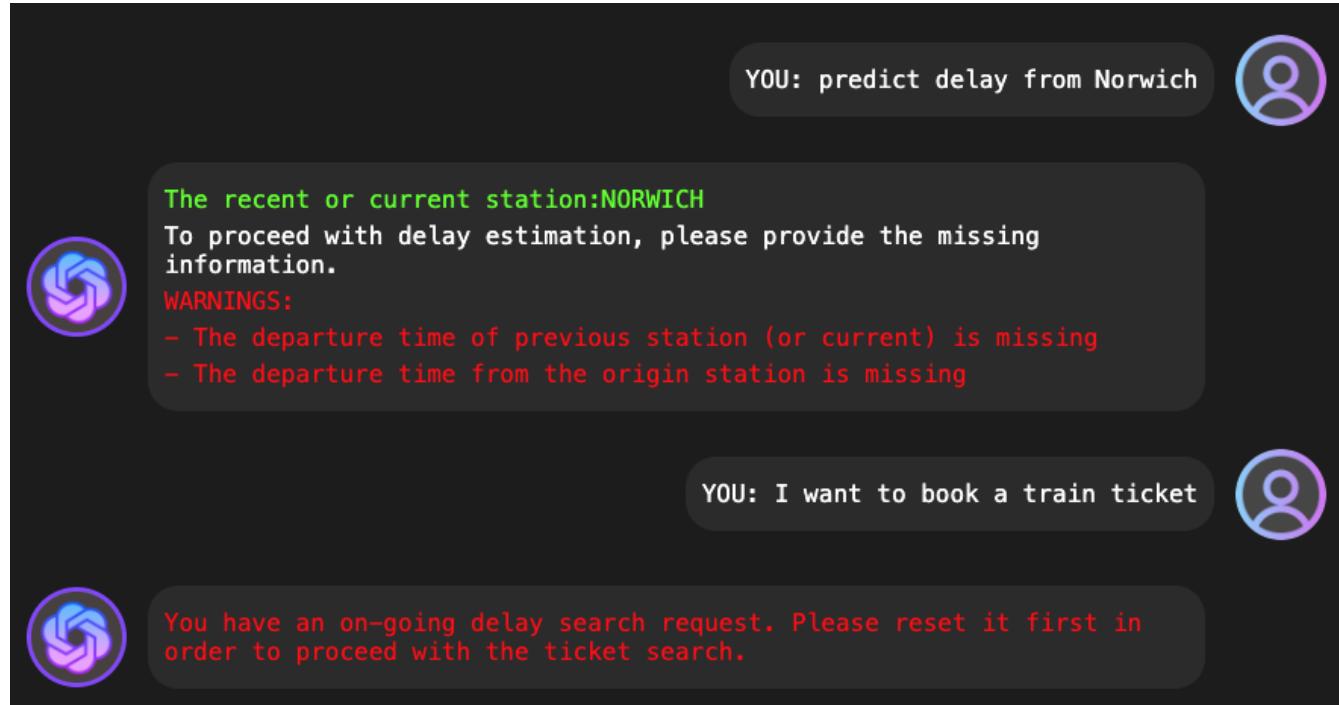


Figure 40: Test: Part B and A Integrity Constraint Check