Implementing planning capabilities for fine-tuning a goal-oriented conversational agent for low social-orientation users

Tommaso Mencattini
nr: 2743537
Amsterdam, Netherlands
t.mencattini@student.vu.nl

Luigi Tisci
nr: 2740559
Amsterdam, Netherlands
l.tisci@student.vu.nl

Hung Tran
nr: 2723572
Amsterdam, Netherlands
h.tran@student.vu.nl

Jip van der Kroef
nr: 2706098
Amsterdam, Netherlands
j.f.vander.kroef@student.vu.nl

Nam Vu Hoang
nr:2723402
Amsterdam, Netherlands
n.vuhoang@student.vu.nl

Pablo Garcia-Legaz Levi
nr: 2727290
Amsterdam, Netherlands
p.garcia.@student.vu.nl

ABSTRACT
This paper presents a task-oriented conversational agent for the cuisine domain. The agent was designed for a low social-orientation group, who values efficiency and is less concerned with purely conversational features. The agent expands on previous efforts by incorporating planning elements such as calendar management and grocery list creation. The results of a user study showed that the agent was well-received and perceived to have high quality. However, further work is required to enhance the perceived quality of the agent, including improvements to the visual design of the grocery list and the effectiveness of the calendar. The Natural Language Understanding (NLU) module of the agent has also been improved, but additional user studies may be required to test for optimal performance.

2. INTRODUCTION

Thanks to the recent advancement in Natural Language Understanding, conversational agents have become ubiquitous and their usage in users' daily routines has drastically improved their efficiency [1]. In this paper, a task-oriented conversational agent for the cuisine domain is introduced. Hussain classifies task-oriented conversational agents as one of the two main classes of chatbots, along with purely conversational agents [3]. In particular, task-oriented chatbots are those agents that are designed to accomplish specific goals, such as searching for or buying an item. Our conversational agent builds on a previous version designed for searching recipes over a database [4] and expands its capabilities by incorporating planning elements such as calendar management and grocery list creation. Before delving into the technical details, the papers contextualize the developed conversational agent concerning the current literature (Introduction); thereafter, it presents a general overview of the agent's capabilities, its design rationale, and its technical presentation, the agent testing process and its outcome, a user study developed to measure the perceived quality of the agent and the resulting analysis.

Despite the increased popularity of conversational agents and chatbots, there is still no consensus in the literature concerning their differences [1]. This paper deploys these terms based on the distinction made by Nuseibeh [7], whereby a conversational agent is a software program that can interpret and respond to statements made by the user in natural language, whereas a chatbot is a particular type of CA designed to simulate a conversation with the user. In particular, we introduce a task-oriented conversational agent, which is explicitly designed for a low social-orientation group. As will be explained in the Design Rationale paragraph, this focus group has been proven to be extremely interested in efficient conversational agents, while being opaque to pure conversational features [5]. In particular, the main functional implementation of our task-oriented conversational agent is to support planning in the domain of cuisine. On top of the pre-defined querying capability, the user is allowed to easily plan their diet routine and their grocery list, based on the required ingredients for the weekly chosen recipes.

To outline a general overview of our conversational agents, it is useful to deploy the four criteria outlined by Nimavat and Champaniera to classify conversational agents: the knowledge domain, the type of service provided, the chatbot goal, and the response-generation method [6]. Our agent has as its knowledge domain a well-defined set of recipes, written in a prolog file, along with contextual information regarding the ongoing conversation. With regards to the service and the agent's goal, the CA assists in the planning and searching of recipes (and their ingredients) for the user's weekly diet; therefore, the goal of the chatbot is to automatize these manual tasks to allow the user to save time. Lastly, the method deployed to generate a response is a rule-based approach focused on pattern matching. LaTeX.

3. AGENT OVERVIEW

3.1 Basic conversational agent

The basic conversational agent's main objective is to simply retrieve information from a recipe dataset. In particular, it

should help the user find a recipe that satisfies all the user's selected filters (or preferences). To do so, the SUPPLE dialogue management framework [4] is used. The conversation follows a semi-strict sequence of conversation patterns that are defined in the initial agenda (an ordered list of patterns that has the role of the backbone of the conversation), however, a sub-pattern can (almost always) be started by the user, and the agent can add new patterns to the agenda during the conversation. The conversation starts with the user pressing the 'start' button, after which pattern c10 begins with the agent greeting the user and, if its name was defined, presenting itself. To that, the user is expected to greet the agent back. The conversation then moves to the recipe selection with pattern a50recipeSelect, during which the agent specifies its goal and proceeds to ask the user for filters. One of the variations of the pattern a21featureRequest (depending on the situation) is started each time the user demands a new filter. Differently, pattern a21removeKeyFromMemory works similarly but for situations where the user wants to remove filters. Pattern a21noMoreFilters allows the user to see all the recipes remaining after applying the filters requested (only if less than 100 recipes remain) when the user does not want any more filters. When a user has found the desired recipe and tells it to the agent, pattern a50recipeConfirm begins and the recipe, ingredients, and steps are shown. First, the agent asks the user if the selected recipe is the wanted one, and if it is, pattern c40 is started, if it is not, the user gets sent back to a50recipeSelect. During pattern c40, which handles the session closer, the agent says goodbye to the user and waits for the user to do the same. Some patterns that have not been cited are c30, which starts when the user asks the agent what its capabilities are; b12 which starts when the agent could not understand what the user said; b13 which starts when the agent did understand the intent of the user but it was expecting a different intent; b42 which starts when the user shows some kind of appreciation. In conclusion, the sole purpose and functionality of the agent are to filter recipes based on the user's preferences and to display the chosen recipe. The agent utilizes the SUPPLE dialogue management framework to guide the conversation and achieve this objective.

3.2 Extended agent's overview

The initial CA has been improved and new functionalities have been added. In particular, the main focus was on the implementation of a calendar, thanks to which the user can plan meals and then extract a list of all the ingredients and respective quantities needed to cook all the planned meals of the week. To do so new patterns were created. After the pattern a50recipeConfirm, where the user has selected a recipe, the new pattern e1 is inserted into the agenda. The agent is going to ask the user if they want to add the chosen recipe to the calendar and, if they do, the agent proceeds by asking on which day the user would like to cook the aforementioned recipe, after which the pattern e12 starts. Pattern e12 asks the user if they want to plan the recipe for breakfast, lunch, or dinner. After the information collected from the two previous answers is added to the knowledge base of the conversational agent, pattern e3 begins. The calendar is displayed and the agent asks the user if they want to add new recipes to the calendar, if they do the pattern a50recipeSelect is inserted in the memory if they don't they get asked if they want to visualize the grocery list and if they do not want to the pattern c40 starts. The grocery list is shown as the pattern f1 starts, during which the agent tells the user to take their time looking at the grocery list and when they are fone to say goodbye, after which c40 begins. The second set of extensions to the basic agent regards the recipe selection process and repair capabilities. New patterns have been implemented to add functionalities. a40random is the pattern that starts when the user expresses the intent of wanting a random recipe, and to that, the agent shows an arbitrarily chosen recipe and moves the conversation to pattern a50recipeConifrm. To aid the agent's ability to repair any kind of problem, the pattern a40restart is implemented for the agent to restart the conversation from the c10 pattern. Moreover, the simple agent intent 'lastTopicCheck' was added to the pattern a50recipeConfirm to confirm the user's recipe choice and, in case they wanted to change the recipe they would get the chance to do so by answering the question and starting the recipe selection process again. The last meaningful addition is the a40stop pattern, which allows the user to stop the conversation at any point and move directly to the session closer, pattern c40.

4. DESIGN RATIONALE AND TECHNICAL IMPLEMENTATION

4.1 The theoretical motivations behind the development process

The main objective behind this paper is to adapt the foregoing generic and base conversational agent to the needs of a particular focus group: low social-orientation people. To fine-tune the capabilities of the agent with the needs of this focus group, the research team based its work on the findings of Liao et al. [5]. The aforementioned paper studies the different perceived qualities of conversational agents concerning two focus groups: low social-oriented and high social-oriented groups. This was possible through the combination of a field study, designed to gather logs and data about the users' interactions with the agent, and the combination of a survey and ad-hoc interviews, gather to both gain a qualitative overview of the agent and classify the users in the two groups based on mental models [5]. In the context of our work, the most interesting finding is that low social-oriented groups are not interested in carrying out a casual conversation with the agents, and subsequently in the general conversational capability of the CA. On the other hand, they are characterized by high utilitarian needs; indeed, it resulted that the main appreciated feature, and the suggested field for future improvement, has been the automatization capabilities of the agent and their efficiency. Indeed, these users were not interested in the possibility of carrying out casual conversations with the agent, as resulted by their opinions (gathered in the qualitative study) and by the logs of their conversations; but, they approached the agent as a mere tool for information retrieval. They asked for more features to support its goal and functionalities, rather than a better conversation design.

According to this finding, the research group decided to implement a brand-new feature in the normal agent: the possibility

of automating through the conversation the plan of the weekly diet and getting a grocery list based on it. In such a way, a short conversation, mixed with the efficient recipe retrieval capabilities of the CA, allows efficient planning, which can be carried out solely through natural language, and without the addition of any external tool. Furthermore, the chatbot has been implemented based on an intentional limitation: the CA will not take initiative in the planning of the weekly diet or the recipe's retrieval. Indeed, Chaves et al. argue that proactivity in goal-oriented conversational agents can have the effect of eliciting the negative feeling of being controlled [2]. Given that the CA's extension is based on planning, which can be perceived as a form of control, the research team offset this authoritarian effect by not allowing the CA to make any suggestion about the recipes that the user should select and when it should plan to cook them, according to economic, or health criteria. Furthermore, this kind of suggestion may have been perceived as useless interaction, and they may have caused a loss in the efficiency of the searching and planning conversations.

Lastly, the team decided to develop a persona for the CA; in particular, by adding CA's humanoid avatar and naming it "Vincenzo", in a way consistent with the overall design. While it may not be a central feature for low social-oriented users, it has been proved that the personification of conversational agents has three overall positive effects: enriching interpersonal relationships, providing unique services, and reducing interactional breakdowns [2].

4.2 Technical Implementation of the CA's Extention

To implement the calendar (as briefly mentioned in paragraph 3.2) five new patterns were added, e1, e12, e13, and e3 [appendix 1]. In these patterns, a total of seven agent intents and respective responses were created. The intents are questionCalendar, questions, insertDay, questionMeal, insertMeal, addCalendar, clearMemory. These intents appear in the patterns related to the calendar feature, and for each of them, a simple text response was added to the response.pl file. For the insertDay, insertMeal, and clearMemory intents, rules were added to `dialog_update.pl` [appendix 2], for the first two, the recipe that has been chosen gets added to the knowledge base in the predicates recipeDataDay(RecipeID, Params) and recipeDataMeal(RecipeID, Params) together with the answer to the respective questions (what day and what meal would the user plan this recipe for) and for the intent clearMemory, which is used to remove the filters applied to find the first recipe, the predicate memory and rendomRep (predicate used for the random recipe extension) are deleted and created new, the former with only an empty list as the parameter and the latter with a new random recipe as the parameter. The aforementioned predicates recipeDataDay/2, recipeDataMeal/2 have also been added to the dialog.pl file (as well as randomRep/1).

To complete the patterns no new user intents were created. However, since we used the addFilter intent (which represents the action the user wants to perform) to let the user answer to what day and what meal they would plan the recipe for, two new entities (which are used to extract information from the user input) were added to the Dialogflow agent. Namely, 'weekday' and 'mealType', the former simply contains the days of the week and the latter only contains breakfast, lunch, and dinner.

For each new pattern added, an HTML page was created, each specific to the point of the conversation where the respective pattern is triggered. Extensive effort was put into the creation of the calendar page [appendix 6], where data was to be extracted from the knowledge base from the agent and inserted with Prolog in the HTML script.

The implementation of the grocery list functionality was similar to the one explained above. Only pattern f1 [appendix 3] was added and four new agent intents, namely groceryList, groceryQuestion, showGrocery, and questionFinal [appendix 4], similarly to before a rule in the `dialog_update.pl` file was added for the intent groceryList [appendix 5]. This rule is needed to create the actual list of all ingredients and respective quantities, which get stored in the predicate groceryList predicate in the knowledge base. Furthermore, a simple HTML page that displays the grocery list was created [appendix 7].

Each pattern requires an HTML page to be created. The process is similar for all of them and the most complex is the calendar page which will now be explained; the other HTML pages work similarly but their complexity is significantly reduced. The page is created by using the Prolog predicate page/3, which takes as the first argument the respective pattern (in this case e3). The predicate starts by checking if the current top level is e3 using the currentTopLevel/1 predicate. If this condition is met, the code proceeds to construct the HTML page by concatenating several HTML elements using the `atomic_list_concat/2` predicate. The first element of the HTML page is a header text, 'This is your meal schedule for the week', which is stored in FirstRow. A table is created to display the meal schedule for each day of the week using the `atomic_list_concat/2` predicate and several variables. For each meal type and each day of the week, the code checks if there is a corresponding recipe in the knowledge base using predicates such as variableMondayBreakfast/1, and then finds its name using recipeName/2. If a recipe exists, its name is inserted into the HTML table; otherwise, an empty cell is displayed. Finally, the constructed HTML page is returned as the final argument Html.

See `\author` section of this template for instructions on how to format the authors. For more than three authors, you may have to place some address information in a footnote, or in a named section at the end of your paper. Names may optionally be placed in a single centered row instead of at the top of each column. Leave one 10-point line of white space below the last line of affiliations.

5. AGENT TESTING AND RESULTING IMPLEMENTATION

5.1 Testing Description and Findings

The evaluation of the conversational agent capabilities has been divided into two phases: a series of agent testing running in parallel with its development, and a post-production user study. While this user study has been run only when the first complete beta of the software was completed, the agent testing helped the daily work of the developers. Indeed, half of the

team held tests on a daily basis; the latter testing was divided into 2 processes: (i) the CA's natural language understanding was tested by the team-member themselves, (ii) whereas external testers had daily conversations to check the conversation's robustness. On one hand, it was necessary to deploy external testers for checking the conversation's robustness, because the developers were biased by their knowledge of the structure and the content of the dialogs. On the other hand, the conversation's robustness has been tested in two ways: either by asking voluntary speakers to interact with the CA or by using ChatGPT. In particular, we asked to ChatGPT to carry out a conversation without CA. We manually typed the answer of our CA in ChatGPT and we used text-to-voice software to reproduce the answers of ChatGPT so it could interact with the CA. Obviously, we could not check the NLU in these interactions due to the usage of text-to-voice software. Nevertheless, as mentioned before, the NLU was tested aside by the team members themselves. With regards to the NLU testing, the main aim was checking the intent recognition confidence on the Docker Consoler, where it is displayed in a numerical format; furthermore, the team also monitored the occurrence of b12, which is the default fallback pattern.

As a result of the first type of testing (NLU), the team discovered that the intent recognition related to the "addFilter " intent was the most problematic. To improve it, several training phrases have been manually added to it on DialogFlow. At the end of the whole project, the number of training phrases was 440. With regards to the conversation's structure analysis, it resulted that the main shortcomings of our design were the lack of flexibility in navigating through the different phases of the dialogue, either for restarting a recipe filtering process, flexibly deleting a filter, or starting a second filtering process directly after completing the first. To overcome these limitations, the development team implemented additional extensions: random, restart, last topic check, and quit. The following paragraph will handle their usage and their technical implementation.

5.2 Technical Implementation After Testing

The remaining extensions not cited before, namely, last topic check, quit, restart, and random, were less complex when compared to the calendar and grocery list functionalities. The implementation of the last topic check was very straightforward, being only an agent intent (lastTopicCheck) inserted in the a50recipeConfirm pattern [appendix 8]. Slightly different is the implementation of the quit functionality, which appears in the pattern a40stop [appendix 9] which can be initiated by the user at almost any point of the conversation and brings them directly to the session closer pattern, also a new intent was added to the Dialogflow agent (stop). The restart functionality was implemented in the same way. A new pattern a40restart [appendix 10] was added to the pattern.pl file while a new intent (restart) was created in the Dialogflow agent. However, for the new agent intent 'terminate' a rule was added to the `dialog_update.pl` file [appendix 11], which erases the agenda and the memory (also finds a new random recipe). The last extension is the random recipe selection. To implement it the new predicate randomRep/1 was added to the dialog.pl file and was defined in the `recipe_selection.pl` file [appendix 12], moreover, the new user intent 'randomChoice' that appears in the a40random pattern was added to the Dialogflow agent. The pattern proceeds with the agent intent update, which is needed to store in the memory random=true.

Each pattern has a respective page created with HTML on Prolog. Most of the pages required very little effort and the process of their creation was almost identical to one another. The most complex of them was the calendar page which has been explained in paragraph 4.2.

6. USER STUDY

6.1 Aim of the user study

This pilot user aims to inform and validate the design of VINCENZO. The desired outcome of the user study involves: 1) the understanding of the drawbacks and shortcomings of VINCENZO's task-oriented capabilities and its overall user experience. In particular, there are two central capabilities: the ability to retrieve a desired recipe for the user from its knowledge base, and the possibility of using the bot to efficiently plan the user's food-related ability (assist the user in planning a weekly diet and in getting a list of the ingredients that have to be bought in the grocery store). Regarding the user experience, three features are explored: the ability to understand the user (NLU), the conversation naturality (patterns), and the visual interface.

6.2 Approach to the Pilot Study

To measure these features, data are collected in three ways:

- automatically extracted from the console
- gathered through a Likert-scale questionnaire
- collected through a short interview (collapsed at the end of the questionnaire)

In such a way, three types of data will be collected through the study: quantitative (e.g., the confidence of intent recognition mined from the console), qualitative (transcripts of the interview), and ordinal (Likert-scale questionnaire)

6.3 participants

Participants are members of the course Project Multi-Agent Systems of the BSc in Artificial Intelligence of the Vrije Universiteit. They are expected to be between 18 and 30 years old and equally distributed between natives and internationals. Participants are motivated to participate in the study due to its part of mandatory working groups. This should easily allow for forming a participant group of 3 people.

6.4 Procedure

The evaluation takes place at the Vrije Universiteit. Once the participants' group is confirmed, they will be distributed over a timeline and one by one will attend the experiment. Once the participant reaches the room, at the suggested time slot, they will be led to the desks and they will need to carry out three complete conversations with VINCENZO. The user will be asked to complete one task per conversation, to experiment with the searching ability of the agent and its two planning

abilities: (i) getting a recipe to cook, (ii) planning its weekly calendar, (iii) and getting the grocery list for the week. Once the conversation is started there will be three possible ends: a correct successful end (the user finds the recipe / uses the calendar feature), a correct unsuccessful end (the user does not find the recipe), and an error end (where a bug freeze the agent). Once the conversation end, the researchers have to take notes of the type of end and save both the docker console log and the eclipse console log. Once the three conversations have been carried out by the user, the questionnaire and the interview should be performed. Collecting all the data, they have to be achieved under the ID of the user. In the end, there should be a database with 7 IDs and the relative data.

6.5 Attributes for task-oriented capabilities

In this paragraph, we will define some attributes required to describe the capability of VINCENZO. The metrics for measuring this feature are divided into two: on one hand, each of these attributes is measured in the questionnaire; on the other, the specific metric is outlined above if available. The attributes that can describe the quality of the task-oriented capabilities are four:
1) Effectiveness, namely how often the user manages to complete its task;

- How often did it complete the search task?
- How often did it complete the planning task?

2) Efficiency, namely how much time the user saved by interacting with the bot (rather than manually carrying out the searching and planning for the recipe);

- How quickly can the user find something to cook compared to manual research (searching efficiency)?
- How fast can the user plan its week and get its grocery store list?

3) Utility, namely how useful was the chatbot for the user;

- Can the chatbot planning function retrieve good recipes? (for evaluating the database rather than the searching capability over the database)
- Would you use the chatbot planning list to plan your week?

4) Completeness, does the chatbot miss some functionality that the user would require for searching and planning tasks;

- Would the chatbot need more ways to query its database? (searching completeness)
- Would the chatbot need more ways to support a user in its food planning? (planning completeness)

6.6 Attributes for user experience

In this paragraph, we will define some attributes required to describe the user experience with VINCENZO. The metrics for measuring this feature are divided into two: on one hand, each of these attributes is measured in the questionnaire; on the other, the specific metric is outlined above if available. The attributes that can describe the user experience of VINCENZO are;
1) Robustness, namely how good is the understanding of user's utterances;

- how often did the agent misunderstand the user?

SPECIFIC METRICS: (i) the number of times b12 is called for in the conversation (mined in the eclipse console); (ii) average confidence in detecting intents (minable in the terminal).

2) Conversation naturality;

- How good is the agent at understanding the context
- User satisfaction with the conversation

SPECIFIC METRIC: number of conversation mismatches b13

3) Analysis of the visual design;

- Utility, namely whether was it useful to have visual support (e.g. for visualizing the recipe)
- Being appropriate, namely whether the user finds the visual appropriate for a visual bot
- Usability, namely whether the user finds it useful to retrieve the required command on the visual
- Likeability of the visual

6.7 Interview

The interview counted 4 open-ended questions: (i) what other searching capabilities would you like to have in the application? (ii) what other planning capabilities would you like to have in the application? (iii) How would you change the aesthetic of the UI? (iv) how would you re-design the interfaces to gain a UI more consistent with the idea of an efficient CA? In the GitHub repository of the work can be found a printed version of the final questionnaire, also accessible through this link: GoogleformQuestionnaire

7. RESULTING DATA ANALYSIS

The analysis of the results is divided into qualitative (questionnaire and interview) and quantitative (mined data). Regarding the interview, the users' suggestions were grouped into 3 categories;

- Overall visual suggestion; the users were satisfied overall with the visual part and they did not outline meaningful improvement (just subjective tips, such as adding more colors)
- Planning capabilities suggestions; also here the users asserted to be completely satisfied. The only requirement was to change the visualization of the grocery list
- Usability, namely whether the user finds it useful to retrieve the required command on the visual
- Searching capabilities suggestions; the users asked for two additional filters to better query the database: filtering by price and filtering by recipes. Nevertheless, the CA was perceived as generally complete also under this criterion

With regards to the questionnaire, we took the mode of each subset of questions, ordered by the tested attribute. Therefore, for each attribute we have a graphical summary, that can be checked in the appendix. Overall, the performance has always been satisfying for the user, with a mean value above 3. It also resulted that the most preferred planning capability in terms

of utility has been accessing the grocery list over the calendar. This is in contrast with the visual feature's considerations, which resulted in that only the grocery list demands additional work.

With regards to the quantitative data analysis, the log documents and the copy of the Docker console have been examined to mine information about the conversation's naturality and robustness. A copy of the Docker console was used to compute the mean and the mode of the intent recognition. While the mode was 100/100, the mean is 78/100. On GitHub, the data-analysis pipeline, for both this task and the questionnaire's modes of extraction, is compressed as a Jupyter notebook. With regards to the conversation robustness, the conversation naturality and robustness are measured by counting respectively the occurrence of b12 and b13 in the conversations' logs. This has been possible by selecting the last session predicate and counting their appearances. In seven conversations, pattern b12 appeared a total of four times (on average 0.57 times per conversation, and the mode is zero) and pattern b13 appeared thirteen times (on average 1.86 times per conversation). While both the robustness and naturality results are satisfactory, nevertheless they require further work to better fine-tune the NLU's capabilities of the CA, by implementing further training phrases on DialogFlow. Indeed, this problem can only be offset by improving the understanding capability of the CA, which is completely determined by its training phrases.

8. CONCLUSION

In conclusion, this paper introduced a task-oriented conversational agent for the cuisine domain, which expands on previous efforts by incorporating planning elements such as calendar management and grocery list creation. The agent was developed for a low social-orientation group that values efficiency and is less concerned with purely conversational features. The results of the user study indicate that the agent was well-received and perceived to have high quality. Nevertheless, further work is required to enhance the perceived quality of the CA; concerning the extension, the visual design of the grocery list must be improved, while the calendar must be made more effective. A solution for the latter problem may be allowing the user to modify manually the calendar also through keyboard interactions. Furthermore, the NLU module of the CA, which was perceived not as optimal, was already improved by working on the CA's training phrases. Nevertheless, additional user study may be required to test whether optimal performance is already reached through these changes.

REFERENCES

[1] Merav Allouch, Amos Azaria, and Rina Azoulay. Conversational agents: Goals, technologies, vision and challenges. *Sensors*, 21(24):8448, 2021.

[2] Ana Paula Chaves and Marco Aurelio Gerosa. How should my chatbot interact? a survey on social characteristics in human–chatbot interaction design. *International Journal of Human–Computer Interaction*, 37(8):729–758, 2021.

[3] Shafquat Hussain, Omid Ameri Sianaki, and Nedal Ababneh. A survey on conversational agents/chatbots classification and design techniques. In *Web, Artificial Intelligence and Network Applications: Proceedings of the Workshops of the 33rd International Conference on Advanced Information Networking and Applications (WAINA-2019) 33*, pages 946–956. Springer, 2019.

[4] Florian Kunneman and Koen V Hindriks. A sequence-based dialog management framework for co-regulated dialog. In *HHAI2022: Augmenting Human Intellect*, pages 143–156. IOS Press, 2022.

[5] Q Vera Liao, Matthew Davis, Werner Geyer, Michael Muller, and N Sadat Shami. What can you do? studying social-agent orientation and agent proactive interactions with an agent for employees. In *Proceedings of the 2016 acm conference on designing interactive systems*, pages 264–275, 2016.

[6] Ketakee Nimavat and Tushar Champaneria. Chatbots: An overview types, architecture, tools and future possibilities. *Int. J. Sci. Res. Dev*, 5(7):1019–1024, 2017.

[7] Rajai Nuseibeh. What is a chatbot. *Chatbots Magazine*, 2018.

APPENDIX

```
pattern([e1, [agent, questionCalendar], [user, disconfirmation],[agent, insert(c40)]]).

pattern([e1, [agent, questionCalendar], [user, confirmation],[agent, questionDay], [agent, insert(e12)]]).

pattern([e12,[user, addFilter], [agent, insertDay(Params)], [agent, questionMeal], [agent, insert(e13)]]):- getParamsPatternInitiatingIntent(user, addFilter, Params).

pattern([e13,[user, addFilter], [agent, insertMeal(Params)], [agent, insert(e3)]]):- getParamsPatternInitiatingIntent(user, addFilter, Params).

pattern([e3,[agent, addCalendar], [user, confirmation],[agent, clearMemory], [agent, insert(a50recipeSelect)]]).

pattern([e3,[agent, addCalendar], [user, disconfirmation],[agent, groceryQuestion],[user, confirmation], [agent, groceryList], [agent, insert(f1)]]).

pattern([e3,[agent, addCalendar], [user, disconfirmation],[agent, groceryQuestion],[user, disconfirmation], [agent, insert(c40)]]).
```

Figure 1

```
if Intent = insertDay(Params),(currentRecipe(RecipeID), randomRep(RecipeID))

then insert(recipeDataDay(RecipeID, Params)) .


if Intent = insertMeal(Params),(currentRecipe(RecipeID), randomRep(RecipeID))

then insert(recipeDataMeal(RecipeID, Params)) .

if Intent = clearMemory, memory(Mem), randomRecipe(NewRecipeID), randomRep(RecipeID)

then delete(memory(Mem)) + insert(memory([]))

+ delete(randomRep(RecipeID)) + insert(randomRep(NewRecipeID)).
```

Figure 2

```
pattern([f1,[agent, showGrocery],[agent, questionFinal],[user, farewell], [agent, insert(c40)]]).
```

Figure 3

```
text(groceryList, "I am hereby making your grocery list").

text(groceryQuestion, "Would you like to see your grocery list?").

text(showGrocery, "This is your grocery list.").

text(questionFinal, "It's been a pleasure to help you out, when you are done with the grocery list, you can just say Bye Bye.").
```

Figure 4

```
if Intent = insertDay(Params),(currentRecipe(RecipeID), randomRep(RecipeID))

then insert(recipeDataDay(RecipeID, Params)) .


if Intent = insertMeal(Params),(currentRecipe(RecipeID), randomRep(RecipeID))

then insert(recipeDataMeal(RecipeID, Params)) .

if Intent = clearMemory, memory(Mem), randomRecipe(NewRecipeID), randomRep(RecipeID)

then delete(memory(Mem)) + insert(memory([]))

+ delete(randomRep(RecipeID)) + insert(randomRep(NewRecipeID)).
```

Figure 5

```
page(e3, _, Html) :-

  % Condition for when to show this page

  currentTopLevel(e3),

  % Constructing HTML page

  atomic_list_concat([<br><div class="text-center style="font-size: 3rem;"><center><h1> This is your meal schedule for the week </h1></center></br></div>], FirstRow),

  %atomic_list_concat(['<h3>  on ~a </h3>'], TemplateDay),

  %atomic_list_concat(['<h3>  at ~a.</h3></div></div></div></div>'], TemplateMeal),

  % Get the bot's name if it has one; other call it 'your assistant'

  %(recipeDataDay(RecipeID, Day), member(weekdays = A, Day)),

  %(recipeDataMeal(RecipeID, Meal), member(mealType = B, Meal)),

  %recipeName(RecipeID, Name),

  %applyTemplate(TemplateRecipe, Name, FirstRow),

  %applyTemplate(TemplateDay, A, SecondRow),

  %applyTemplate(TemplateMeal, B, ThirdRow),

  FourthRow = '<div class="text-center mt-5"></div>',

  %table

  atomic_list_concat([<div style="display: flex; justify-content: center; "><table border="1" bgcolor="black" width="1150" height="330"><tr bgcolor="sienna"><th width="100" height="30"></th><th style="text-align: center" width="150">Monday</th><th style="text-align: center" width="150">Tuesday</th><th style="text-align: center" width="150">Wednesday</th><th style="text-align: center" width="150">Thursday</th><th style="text-align: center" width="150">Friday</th><th style="text-align: center" width="150">Saturday</th><th style="text-align: center" width="150">Sunday</th></tr> ], TableRow1),

  atomic_list_concat([<tr bgcolor="wheat" height="100" align="center"><td>Breakfast</td>], Breakfast),

  ((atomic_list_concat([<td>~a</td>], Breakfast1), variableMondayBreakfast(RecipeID1), recipeName(RecipeID1, Name1), applyTemplate(Breakfast1, Name1, Breakfast11)), atomic_list_concat([<td> </td>], Breakfast11)),

  ((atomic_list_concat([<td>~a</td>], Breakfast2), variableTuesdayBreakfast(RecipeID2),recipeName(RecipeID2, Name2), applyTemplate(Breakfast2, Name2, Breakfast22)), atomic_list_concat([<td> </td>], Breakfast22)),

  ((atomic_list_concat([<td>~a</td>], Breakfast3),variableWednesdayBreakfast(RecipeID3),recipeName(RecipeID3, Name3),
```

Figure 6

```
applyTemplate(Breakfast3, Name3, Breakfast33)), atomic_list_concat([<td> </td>],
Breakfast33)),

((atomic_list_concat([<td>-a</td>],
Breakfast4),variableThursdayBreakfast(RecipeID4), recipeName(RecipeID4, Name4),
applyTemplate(Breakfast4, Name4, Breakfast44)), atomic_list_concat([<td> </td>],
Breakfast44)),

((atomic_list_concat([<td>-a</td>],
Breakfast5),variableFridayBreakfast(RecipeID5), recipeName(RecipeID5, Name5),
applyTemplate(Breakfast5, Name5, Breakfast55)), atomic_list_concat([<td> </td>],
Breakfast55)),

((atomic_list_concat([<td>-a</td>],
Breakfast6),variableSaturdayBreakfast(RecipeID6), recipeName(RecipeID6,
Name6),applyTemplate(Breakfast6, Name6, Breakfast66)), atomic_list_concat([<td>
</td>], Breakfast66)),

((atomic_list_concat([<td>-a</td></tr>],
Breakfast7),variableSundayBreakfast(RecipeID7), recipeName(RecipeID7, Name7),
applyTemplate(Breakfast7, Name7, Breakfast77)); atomic_list_concat([<td>
</td></tr>], Breakfast77)),

atomic_list_concat([<tr bgcolor="wheat" height="100"
align="center"><td>Lunch</td>], Lunch),

((atomic_list_concat([<td>-a</td>], Lunch1),variableMondayLunch(RecipeID8),
recipeName(RecipeID8, Name8),applyTemplate(Lunch1,Name8, Lunch11)),
atomic_list_concat([<td> </td>], Lunch11)),

((atomic_list_concat([<td>-a</td>],
Lunch2),variableTuesdayLunch(RecipeID9),recipeName(RecipeID9, Name9),
applyTemplate(Lunch2,Name9, Lunch22)), atomic_list_concat([<td> </td>],
Lunch22)),

((atomic_list_concat([<td>-a</td>],
Lunch3),variableWednesdayLunch(RecipeID10),recipeName(RecipeID10, Name10),
applyTemplate(Lunch3,Name10, Lunch33)), atomic_list_concat([<td> </td>],
Lunch33)),

((atomic_list_concat([<td>-a</td>], Lunch4),
variableThursdayLunch(RecipeID11),recipeName(RecipeID11, Name11),
applyTemplate(Lunch4,Name11, Lunch44)), atomic_list_concat([<td> </td>],
Lunch44)),

((atomic_list_concat([<td>-a</td>], Lunch5),
variableFridayLunch(RecipeID12),recipeName(RecipeID12, Name12),
```

Figure 7

```
((atomic_list_concat([<td>-a</td>],
Breakfast5),variableFridayBreakfast(RecipeID5), recipeName(RecipeID5, Name5),
applyTemplate(Breakfast5, Name5, Breakfast55)), atomic_list_concat([<td> </td>],
Breakfast55)),

((atomic_list_concat([<td>-a</td>],
Breakfast6),variableSaturdayBreakfast(RecipeID6), recipeName(RecipeID6,
Name6),applyTemplate(Breakfast6, Name6, Breakfast66)), atomic_list_concat([<td>
</td>], Breakfast66)),

((atomic_list_concat([<td>-a</td></tr>],
Breakfast7),variableSundayBreakfast(RecipeID7), recipeName(RecipeID7, Name7),
applyTemplate(Breakfast7, Name7, Breakfast77)); atomic_list_concat([<td>
</td></tr>], Breakfast77)),

atomic_list_concat([<tr bgcolor="wheat" height="100"
align="center"><td>Lunch</td>], Lunch),

((atomic_list_concat([<td>-a</td>], Lunch1),variableMondayLunch(RecipeID8),
recipeName(RecipeID8, Name8),applyTemplate(Lunch1,Name8, Lunch11)),
atomic_list_concat([<td> </td>], Lunch11)),

((atomic_list_concat([<td>-a</td>],
Lunch2),variableTuesdayLunch(RecipeID9),recipeName(RecipeID9, Name9),
applyTemplate(Lunch2,Name9, Lunch22)), atomic_list_concat([<td> </td>],
Lunch22)),

((atomic_list_concat([<td>-a</td>],
Lunch3),variableWednesdayLunch(RecipeID10),recipeName(RecipeID10, Name10),
applyTemplate(Lunch3,Name10, Lunch33)), atomic_list_concat([<td> </td>],
Lunch33)),

((atomic_list_concat([<td>-a</td>], Lunch4),
variableThursdayLunch(RecipeID11),recipeName(RecipeID11, Name11),
applyTemplate(Lunch4,Name11, Lunch44)), atomic_list_concat([<td> </td>],
Lunch44)),

((atomic_list_concat([<td>-a</td>], Lunch5),
variableFridayLunch(RecipeID12),recipeName(RecipeID12, Name12),
applyTemplate(Lunch5,Name12, Lunch55)), atomic_list_concat([<td> </td>],
Lunch55)),

((atomic_list_concat([<td>-a</td>], Lunch6),
variableSaturdayLunch(RecipeID13),recipeName(RecipeID13, Name13),
applyTemplate(Lunch6,Name13, Lunch66)), atomic_list_concat([<td> </td>],
Lunch66)),
```

Figure 8

```
((atomic_list_concat(['<td>~a</td></tr>'],
Lunch7),variableSundayLunch(RecipeID14),recipeName(RecipeID14,          Name14),
applyTemplate(Lunch7,Name14,  Lunch77));  atomic_list_concat(['<td>  </td></tr>'],
Lunch77)),

 atomic_list_concat(['<tr          bgcolor="wheat"          height="100"
align="center"><td>Dinner</td>'], Dinner),

 ((atomic_list_concat(['<td>~a</td>'],                                  Dinner1),
variableMondayDinner(RecipeID15),recipeName(RecipeID15,                Name15),
applyTemplate(Dinner1,  Name15,  Dinner11));  atomic_list_concat(['<td>  </td>'],
Dinner11)),

 ((atomic_list_concat(['<td>~a</td>'],                                  Dinner2),
variableTuesdayDinner(RecipeID16),recipeName(RecipeID16,               Name16),
applyTemplate(Dinner2,  Name16,  Dinner22));  atomic_list_concat(['<td>  </td>'],
Dinner22)),

 ((atomic_list_concat(['<td>~a</td>'],                                  Dinner3),
variableWednesdayDinner(RecipeID17),recipeName(RecipeID17,             Name17),
applyTemplate(Dinner3,  Name17,  Dinner33));  atomic_list_concat(['<td>  </td>'],
Dinner33)),

 ((atomic_list_concat(['<td>~a</td>'],                                  Dinner4),
variableThursdayDinner(RecipeID18),recipeName(RecipeID18,              Name18),
applyTemplate(Dinner4,  Name18,  Dinner44));  atomic_list_concat(['<td>  </td>'],
Dinner44)),

 ((atomic_list_concat(['<td>~a</td>'],                                  Dinner5),
variableFridayDinner(RecipeID19),recipeName(RecipeID19,                Name19),
applyTemplate(Dinner5,  Name19,  Dinner55));  atomic_list_concat(['<td>  </td>'],
Dinner55)),

 ((atomic_list_concat(['<td>~a</td>'],                                  Dinner6),
variableSaturdayDinner(RecipeID20),recipeName(RecipeID20,              Name20),
applyTemplate(Dinner6,  Name20,  Dinner66));  atomic_list_concat(['<td>  </td>'],
Dinner66)),

 ((atomic_list_concat(['<td>~a</td></tr></table>'],
Dinner7),variableSundayDinner(RecipeID21),recipeName(RecipeID21,        Name21),
applyTemplate(Dinner7,  Name21,  Dinner77));  atomic_list_concat(['<td>
</td></tr></table></div>'], Dinner77)),

 atomic_list_concat([FirstRow, FourthRow, TableRow1, Breakfast, Breakfast11,
Breakfast22, Breakfast33, Breakfast44, Breakfast55, Breakfast66, Breakfast77, Lunch,
Lunch11, Lunch22, Lunch33, Lunch44, Lunch55, Lunch66, Lunch77, Dinner,
Dinner11, Dinner22, Dinner33, Dinner44, Dinner55, Dinner66, Dinner77], Body),
```

Figure 9


```
((atomic_list_concat(['<td>~a</td></tr>'],
Lunch7),variableSundayLunch(RecipeID14),recipeName(RecipeID14,          Name14),
applyTemplate(Lunch7,Name14,  Lunch77));  atomic_list_concat(['<td>  </td></tr>'],
Lunch77)),

 atomic_list_concat(['<tr          bgcolor="wheat"          height="100"
align="center"><td>Dinner</td>'], Dinner),

 ((atomic_list_concat(['<td>~a</td>'],                                  Dinner1),
variableMondayDinner(RecipeID15),recipeName(RecipeID15,                Name15),
applyTemplate(Dinner1,  Name15,  Dinner11));  atomic_list_concat(['<td>  </td>'],
Dinner11)),

 ((atomic_list_concat(['<td>~a</td>'],                                  Dinner2),
variableTuesdayDinner(RecipeID16),recipeName(RecipeID16,               Name16),
applyTemplate(Dinner2,  Name16,  Dinner22));  atomic_list_concat(['<td>  </td>'],
Dinner22)),

 ((atomic_list_concat(['<td>~a</td>'],                                  Dinner3),
variableWednesdayDinner(RecipeID17),recipeName(RecipeID17,             Name17),
applyTemplate(Dinner3,  Name17,  Dinner33));  atomic_list_concat(['<td>  </td>'],
Dinner33)),

 ((atomic_list_concat(['<td>~a</td>'],                                  Dinner4),
variableThursdayDinner(RecipeID18),recipeName(RecipeID18,              Name18),
applyTemplate(Dinner4,  Name18,  Dinner44));  atomic_list_concat(['<td>  </td>'],
Dinner44)),

 ((atomic_list_concat(['<td>~a</td>'],                                  Dinner5),
variableFridayDinner(RecipeID19),recipeName(RecipeID19,                Name19),
applyTemplate(Dinner5,  Name19,  Dinner55));  atomic_list_concat(['<td>  </td>'],
Dinner55)),

 ((atomic_list_concat(['<td>~a</td>'],                                  Dinner6),
variableSaturdayDinner(RecipeID20),recipeName(RecipeID20,              Name20),
applyTemplate(Dinner6,  Name20,  Dinner66));  atomic_list_concat(['<td>  </td>'],
Dinner66)),

 ((atomic_list_concat(['<td>~a</td></tr></table>'],
Dinner7),variableSundayDinner(RecipeID21),recipeName(RecipeID21,        Name21),
applyTemplate(Dinner7,  Name21,  Dinner77));  atomic_list_concat(['<td>
</td></tr></table></div>'], Dinner77)),

 atomic_list_concat([FirstRow, FourthRow, TableRow1, Breakfast, Breakfast11,
Breakfast22, Breakfast33, Breakfast44, Breakfast55, Breakfast66, Breakfast77, Lunch,
Lunch11, Lunch22, Lunch33, Lunch44, Lunch55, Lunch66, Lunch77, Dinner,
Dinner11, Dinner22, Dinner33, Dinner44, Dinner55, Dinner66, Dinner77], Body),
```


Figure 10

```
variableFridayDinner(RecipeID):- recipeDataDay(RecipeID,Day), member(weekdays =
A, Day),recipeDataMeal(RecipeID, Meal), member(mealType = B, Meal), A = 'Friday',
B = 'dinner'.

variableSaturdayDinner(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Saturday', B = 'dinner'.

variableSundayDinner(RecipeID):- recipeDataDay(RecipeID,Day), member(weekdays
= A, Day),recipeDataMeal(RecipeID, Meal), member(mealType = B, Meal), A =
'Sunday', B = 'dinner'.

variableMondayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Monday', B = 'breakfast'.

variableTuesdayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Tuesday', B = 'breakfast'.

variableWednesdayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Wednesday', B = 'breakfast'.

variableThursdayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Thursday', B = 'breakfast'.

variableFridayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Friday', B = 'breakfast'.

variableSaturdayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Saturday', B = 'breakfast'.

variableSundayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Sunday', B = 'breakfast'.
```

Figure 11

```
variableFridayDinner(RecipeID):- recipeDataDay(RecipeID,Day), member(weekdays =
A, Day),recipeDataMeal(RecipeID, Meal), member(mealType = B, Meal), A = 'Friday',
B = 'dinner'.

variableSaturdayDinner(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Saturday', B = 'dinner'.

variableSundayDinner(RecipeID):- recipeDataDay(RecipeID,Day), member(weekdays
= A, Day),recipeDataMeal(RecipeID, Meal), member(mealType = B, Meal), A =
'Sunday', B = 'dinner'.

variableMondayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Monday', B = 'breakfast'.

variableTuesdayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Tuesday', B = 'breakfast'.

variableWednesdayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Wednesday', B = 'breakfast'.

variableThursdayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Thursday', B = 'breakfast'.

variableFridayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Friday', B = 'breakfast'.

variableSaturdayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Saturday', B = 'breakfast'.

variableSundayBreakfast(RecipeID):- recipeDataDay(RecipeID,Day),
member(weekdays = A, Day),recipeDataMeal(RecipeID, Meal), member(mealType =
B, Meal), A = 'Sunday', B = 'breakfast'.
```

Figure 12

```
pattern([a50recipeConfirm, [agent, recipeCheck], [user, confirmation], [agent,
lastTopicCheck], [user, disconfirmation], [agent, insert(e1)]]).

pattern([a50recipeConfirm, [agent, recipeCheck], [agent, lastTopicCheck], [user,
confirmation], [agent, terminate], [agent, insert(a50recipeSelect)]]).
```

Figure 13

```
pattern([a40stop, [user, stop], [agent, insert(c40)]]).
```

Figure 14

```
pattern([a40restart, [user, restart], [agent, terminate], [agent, insert(c10)]]).
```

Figure 15

```
if Intent = terminate, randomRecipe(NewRecipeID), randomRep(RecipeID),
        agenda(Agenda), memory(Params)
then updateSession(agent, terminate, [], '')
        + delete(agenda(Agenda)) + insert(agenda([c10]))
        + delete(randomRep(RecipeID)) + insert(randomRep(NewRecipeID))
        + delete(memory(Params)) + insert(memory([])) .
```

Figure 16

```
randomRecipe(RecipeID) :- recipeIDs(RecipesIDs), random_member(RecipeID,
RecipesIDs).
```

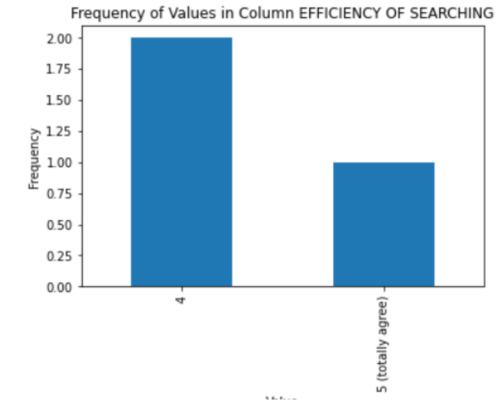
Figure 17

```
if Intent = randomInsert, randomRecipe(NewRecipeID), randomRep(RecipeID)
        then delete(randomRep(RecipeID)) + insert(randomRep(NewRecipeID)).
```

Figure 18

```
pattern([a40random, [user, randomChoice], [agent, update(['random'='true'])], [agent,
insert(a50recipeConfirm)]]).
```

Figure 19



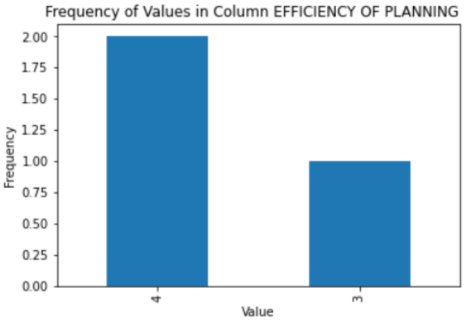
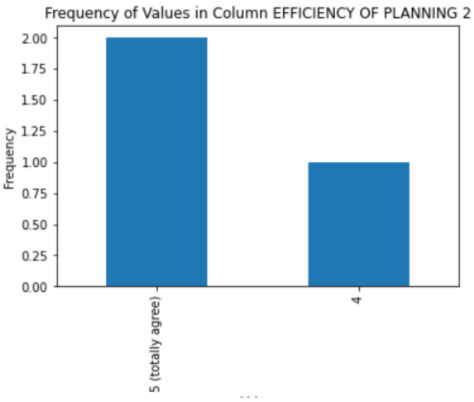


Figure 20



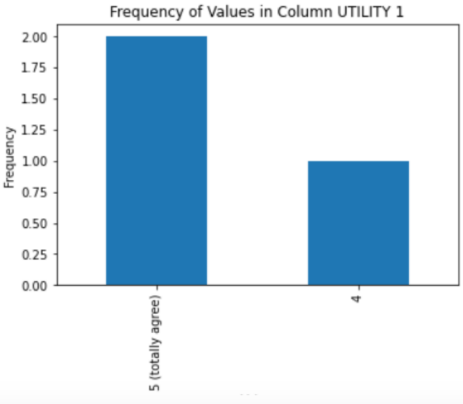
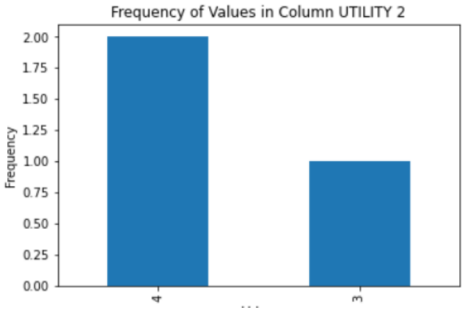


Figure 21



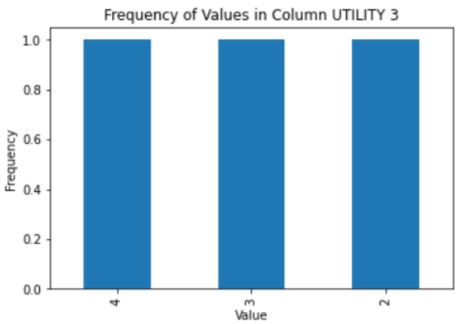


Figure 22

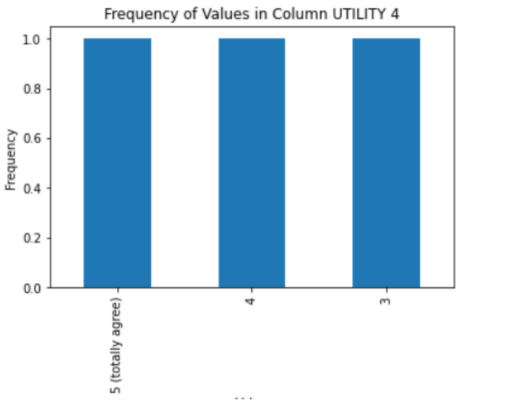


Figure 23

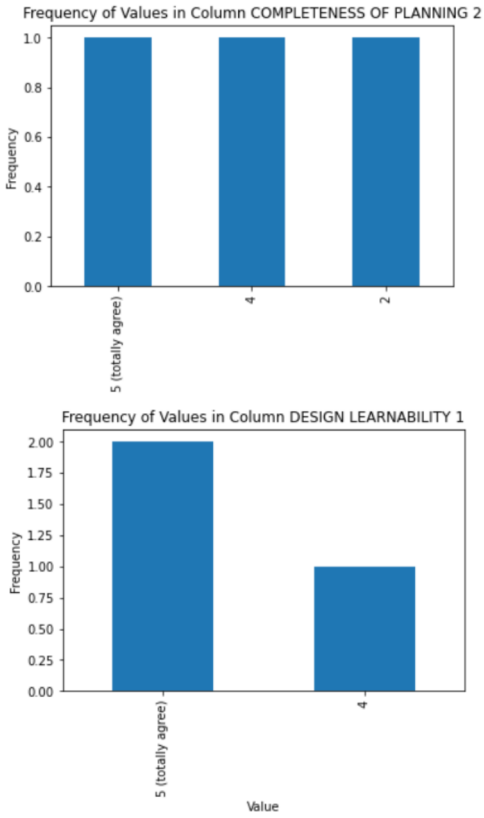
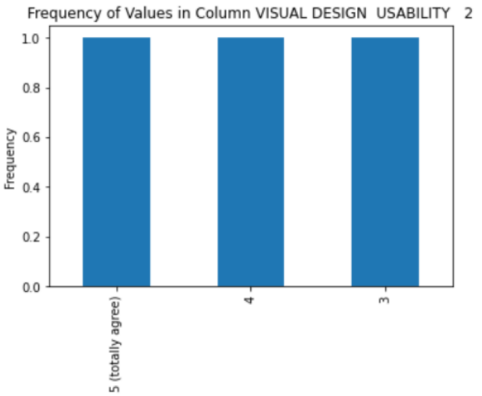


Figure 24



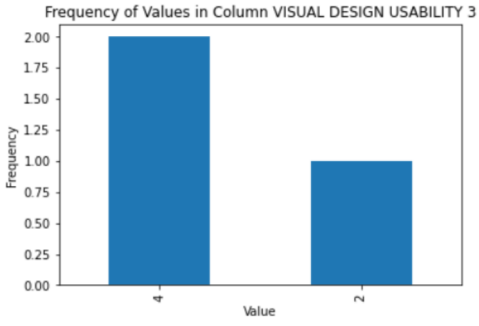
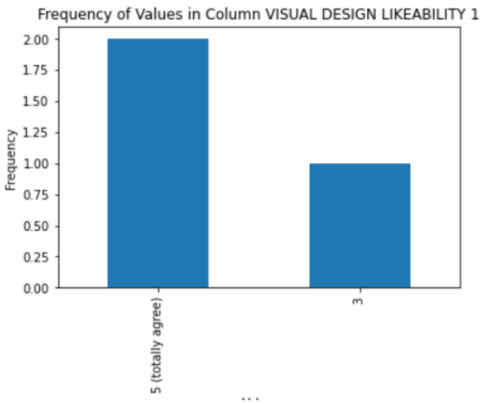


Figure 25



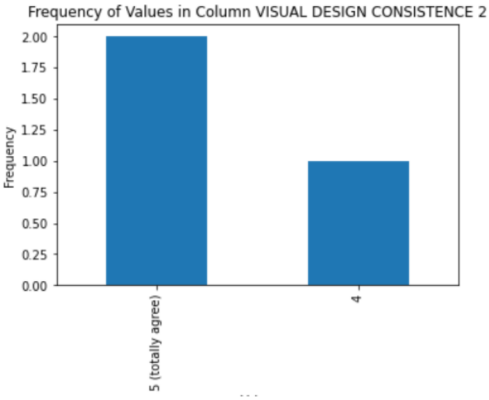
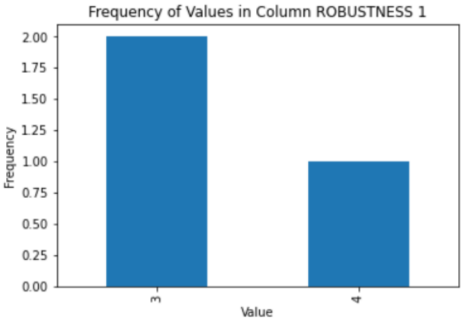


Figure 26



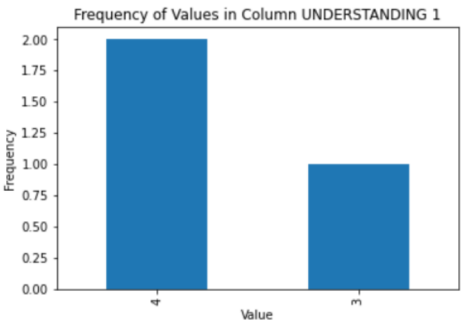


Figure 27