STA130F24_HW02_DueSep19

September 19, 2024

0.1 STA130 Homework 02

Please see the course wiki-textbook for the list of topics covered in this homework assignment, and a list of topics that might appear during ChatBot conversations which are "out of scope" for the purposes of this homework assignment (and hence can be safely ignored if encountered)

Introduction

0.1.1 Introduction

A reasonable characterization of STA130 Homework is that it simply defines a weekly reading comprehension assignment. Indeed, STA130 Homework essentially boils down to completing various understanding confirmation exercises oriented around coding and writing tasks. However, rather than reading a textbook, STA130 Homework is based on ChatBots so students can interactively follow up to clarify questions or confusion that they may still have regarding learning objective assignments.

> Communication is a fundamental skill underlying statistics and data science, so STA130 Homework based on ChatBots helps practice effective two-way communication as part of a "realistic" dialogue activity supporting underlying conceptual understanding building.

It will likely become increasingly tempting to rely on ChatBots to "do the work for you". But when you find yourself frustrated with a ChatBots inability to give you the results you're looking for, this is a "hint" that you've become overreliant on the ChatBots. Your objective should not be to have ChatBots "do the work for you", but to use ChatBots to help you build your understanding so you can efficiently leverage ChatBots (and other resources) to help you work more efficiently.

Instructions

0.1.2 Instructions

1. Code and write all your answers (for both the "Prelecture" and "Postlecture" HW) in a python notebook (in code and markdown cells)

   > It is *suggested but not mandatory* that you complete the "Prelecture" HW prior to the Monday LEC since (a) all HW is due at the same time; but, (b) completing some of the HW early will mean better readiness for LEC and less of a "procrastentation cruch" towards the end of the week…

2. Paste summaries of your ChatBot sessions (including link(s) to chat log histories if you're using ChatGPT) within your notebook

Create summaries of your ChatBot sessions by using concluding prompts such as "Please provide a summary of our exchanges here so I can submit them as a record of our interactions as part of a homework assignment" or, "Please provide me with the final working verson of the code that we created together"

3. Save your python jupyter notebook in your own account and "repo" on `github.com` and submit a link to that notebook though Quercus for assignment marking

Prompt Engineering?

0.1.3 Prompt Engineering?

The questions (as copy-pasted prompts) are designed to initialize appropriate ChatBot conversations which can be explored in the manner of an interactive and dynamic textbook; but, it is nonetheless **strongly recommendated** that your rephrase the questions in a way that you find natural to ensure a clear understanding of the question. Given sensible prompts the represent a question well, the two primary challenges observed to arise from ChatBots are

1. conversations going beyond the intended scope of the material addressed by the question; and,
2. unrecoverable confusion as a result of sequential layers logial inquiry that cannot be resolved.

In the case of the former (1), adding constraints specifying the limits of considerations of interest tends to be helpful; whereas, the latter (2) is often the result of initial prompting that leads to poor developments in navigating the material, which are likely just best resolve by a "hard reset" with a new initial approach to prompting. Indeed, this is exactly the behavior hardcoded into copilot…

0.1.4 Marking Rubric (which may award partial credit)

- [0.1 points]: All relevant ChatBot summaries [including link(s) to chat log histories if you're using ChatGPT] are reported within the notebook
- 0.3 points: Assignment completion confirmed by working "final" code and ChatBot summaries for "3"
-
-

1 "Pre-lecture" HW [*completion prior to next LEC is suggested but not mandatory*]

2 1. Begin (or restart) part "3(a)" of the TUT Demo and interact with a ChatBot to make sure you understand how each part the Monte Hall problem code above works

Further Guidance

> *ChatBots typically explain code fairly effectively, so a ChatBot will probably be very helpful if you share the full Monte Hall problem code; but, you can always introduce more specific and targetted follow-up prompts that help with focus, re-redirection, and response format regarding the ChatBot responses as needed.*

ChatBots won't always re-introduce and re-explain the Monte Hall problem itself, so if you need it to do so you may need to specifically request this as part of your prompt or follow up interactions.

Answer: Problem Setup: - Monty Hall problem: In the Monty Hall problem, there are three doors: behind one is a car (the prize), and behind the other two are goats. The player initially picks one door. Then, the host (who knows what's behind each door) opens one of the other two doors to reveal a goat. After that, the player has the option to switch to the other unopened door. The puzzle asks whether the player has a better chance of winning by switching or staying.

Code Breakdown: 1. Initial Setup:

```
[ ]: reps = 1000000   # Number of repetitions
     wins = 0   # Counter for wins
```

- reps: Number of times the simulation will be run(1,000,000 repetitions).
- wins: A counter to track how many times the player wins when switching doors.

2. Main Loop

```
[ ]: for _ in range(reps):
```

- The loop runs reps (1,000,000) times, representing each simulated game of the Monty Hall problem.

3. Assign the Winning Door

```
[ ]: winning_door = np.random.choice([1, 2, 3])
```

- The winning door is randomly chosen as either door 1, 2, or 3 using NumPy's random.choice() function.

4. Player's initial Choice

```
[ ]: player_choice = 1
```

- The player always chooses door 1 initially in this simulation.

5. Host Reveals a Goal

```
[ ]: remaining_doors = [door for door in [1, 2, 3] if door != player_choice and door␣
      ↪!= winning_door]
     goat_door = np.random.choice(remaining_doors)
```

- The host needs to reveal a door that is not the player's initial choice (player_choice) and not the winning door (winning_door).
- remaining_doors: This list contains the doors the host can reveal. These are the doors that the player didn't choose, and that don't have the car behind them.
- goat_door: The host randomly selects one of the remaining doors to reveal a goat.

6. Player Switches to the Remaining Door

[ ]:
```
switch_choice = [door for door in [1, 2, 3] if door != player_choice and door !
↪= goat_door][0]
```

- After the host reveals a goat door, the player switches to the other remaining door (the one that's neither the initial choice nor the door the host revealed).

7. Check if the Player Wins

[ ]:
```
if switch_choice == winning_door:
    wins += 1
```

- If the player's switch choice is the winning door, the player wins, and the wins counter is incremented by 1.

8. Calculate the WIn Probability

[ ]:
```
print(f"Win probability by switching: {wins / reps}")
```

- The win probability is calculated by dividing the number of wins by the total number of repetitions (reps). This prints the probability of winning when the player switches doors.

Monte Hall Conclusion: - In the Monte Hall problem, switching doors increases the chances of winning. The probability of winning by switching is 2/3 (about 66.67%), while staying gives only a 1/3 (33.33%) chance of winning. The code simulates this behavior, and after running the simulation, you should get a result close to 0.666 (2/3), confirming that switching is the better strategy.

2. Extend your ChatBot sessions to now address part "3(b)" of the TUT Demo and interact with your ChatBot to see if it can suggest a simpler, more streamlined way to code up this *for* loop simulation so the process is more clear and easier to understand; then, describe any preferences you have in terms of readibility or explainability between the original code and the code improvements suggested by the ChatBot Further Guidance

> *The links in the TUT Demo show that there can be variation in the quality of the code improvements suggested by ChatBots; however, it's most likely that a ChatBot is going to be able to greatly reduce the number of steps/lines of code and hence complexity of understanding the problem. ChatBots can provide a good amount of explanation and inline clarifying code comments and provide more simpler more intuitive code that can transform something that looks a bit scary at first to something that's easy to follow and make sense of. Of course, in doing so, a ChatBot may introduce functions that you've technically not learned or seen before; but, the amount of simplification and clarifying comments is probably going to more than compensate for this; and, you'll have seen a learned a little bit more about what's possible through this process, which is the ideal experience we're hoping you'll see here.*

This is a more streamlined and clear version of the Monte hall code

[ ]:
```
import numpy as np

# Set up initial parameters
reps = 1000000  # Number of repetitions
```

```
wins = 0  # Counter for wins

# Run the simulation
for _ in range(reps):
    # Randomly assign the winning door (1, 2, or 3) and the player's initial␣
 ↪choice (always door 1)
    winning_door = np.random.choice([1, 2, 3])
    player_choice = 1  # Player always chooses door 1

    # Determine the door the player would switch to (the only remaining␣
 ↪unopened door)
    switch_choice = 6 - player_choice - winning_door  # 1+2+3 = 6, subtract␣
 ↪player's choice and winning door

    # Check if switching wins
    if switch_choice == winning_door:
        wins += 1

# Calculate and print the win probability from switching
print(f"Win probability by switching: {wins / reps}")
```

The original code is more readable and explainable. It breaks down the problem into clear with logical steps which makes easier to understand.

The code suggested by the Chatbot is more efficient and clever but less intuitive and harder to explain.

3. Submit your preferred version of the Monty Hall problem that is verified to be running and working with a final printed output of the code; then, add code comments explaining the purpose of each line of the code Further Guidance

> *Your ChatBot will likely do much of this for you, but verify for yourself that you understand each comment and reword comments wherever you think it would be better to explain it differently.*
>
> *Remember to ask for summaries of your current session and paste these into your homework notebook (including link(s) to chat log histories if you're using ChatGPT)*

[ ]:
```
import numpy as np

# Set up initial parameters
reps = 1000000  # Number of repetitions
wins = 0  # Counter for wins

# Run the simulation
for _ in range(reps):
    # Randomly assign the winning door (1, 2, or 3)
    winning_door = np.random.choice([1, 2, 3])
    # Assume the player always chooses door 1
    player_choice = 1

    # The host reveals a goat door (not the player's choice and not the winning␣
↪door)
    remaining_doors = [door for door in [1, 2, 3] if door != player_choice and␣
↪door != winning_door]
    goat_door = np.random.choice(remaining_doors)

    # The player switches to the other remaining door
    switch_choice = [door for door in [1, 2, 3] if door != player_choice and␣
↪door != goat_door][0]

    # Check if the switch choice is the winning door
    if switch_choice == winning_door:
        wins += 1

# Calculate and print the win probability from switching
print(f"Win probability by switching: {wins / reps}")
```

code explanation(code breeak down) is in the Question 1.

1. Setup:

- The code runs the simulation 1,000,000 times (reps).
- It keeps track of how many times the player wins after switching (wins).

2. Main Loop:

- In each iteration, the winning door (with the car) is randomly assigned as door 1, 2, or 3.
- The player always initially selects door 1.
- The host then reveals a goat behind one of the other two doors.
- The player switches to the remaining door.

3. Winning Check:

- If the player's switch leads to the winning door (the one with the car), the wins counter is incremented.

4. Result:

- After all repetitions, the probability of winning by switching is calculated by dividing the number of wins by the total number of repetitions (wins / reps).
- The result is printed, and it should be close to 2/3 (approximately 0.666), reflecting that switching doors is a better strategy.

Link to the Chatbot: https://chatgpt.com/share/66eaeb88-ac50-8006-a0a2-2f5f46260f10

4. Watch the embedded video tutorial on Markov chains in the next Jupyter cell below to understand their application and relevance for ChatBots; then, after watching the video, start a new ChatBot session by prompting that you have code that creates a

“Markovian ChatBot”; show it the first version of the “Markovian ChatBot code” below; and interact with the ChatBot session to make sure you understand how the original first version of the “Markovian ChatBot code” works Further Guidance

> *If the ChatBot prompts you as to how you will “train” your own “Markovian ChatBot” you can reply that you’ll just use a series of stories with a lot of different characters*
>
> *Ask for summaries of this second ChatBot session and paste these into your homework notebook (including link(s) to chat log histories if you’re using ChatGPT)*

[ ]:
```python
# Markov Chains and Text Generation
from IPython.display import YouTubeVideo
YouTubeVideo('56mGTszb_iM', width = 550)
```

1. Training the Model on Text Data The bot begins by analyzing a large text corpus (which could be anything from books to chat logs). The text is broken down into sequences of words or tokens. The Markov chain algorithm then identifies which words are likely to follow which other words, based on how often specific word pairs or groups appear together in the training data.

2. Building the Transition Matrix Once the model processes the text, it builds a “transition matrix” (or table) that stores the probabilities of a word following another word. For example:

[ ]:
```
hello -> world (60% probability)
hello -> there (40% probability)
```

If the bot is asked to generate a response and the word “hello” comes up, it will randomly choose between “world” and “there” based on these probabilities.

3. Generating Responses When generating responses, the bot looks at the last word (or several words) in the input and uses the transition matrix to determine which word(s) should follow. It does this until it generates a sufficiently long response or reaches an end state (like a punctuation mark).

Basic Code Structure Here’s a high-level example of how a basic Markov Chain-based chatbot might be implemented in Python:

[ ]:
```python
import random

class MarkovChatBot:
    def __init__(self):
        self.model = {}

    def train(self, text):
        words = text.split()
        for i in range(len(words) - 1):
            if words[i] not in self.model:
                self.model[words[i]] = []
            self.model[words[i]].append(words[i+1])

    def generate_response(self, start_word, num_words=20):
        if start_word not in self.model:
            return "I don't know what to say!"

        response = [start_word]
        current_word = start_word

        for i in range(num_words - 1):
            next_words = self.model.get(current_word)
            if not next_words:
                break
            current_word = random.choice(next_words)
            response.append(current_word)

        return ' '.join(response)

# Example usage
bot = MarkovChatBot()
text_corpus = "hello world hello there world is big world is small"
bot.train(text_corpus)

print(bot.generate_response('hello'))
```

Steps Breakdown: 1. Training Phase: The train function processes the input text (split into words) and builds a dictionary (or a "transition matrix"). For each word, it stores the list of words that can follow it.

2. Generation Phase: The generate_response function takes a starting word and generates a sequence by randomly picking the next word based on the transition matrix until it reaches the desired length.

Link to Chatbot: https://chatgpt.com/share/66eaedb7-dff0-8006-bdb3-cf5ddbb18cf0

Continue now...?

2.0.1 Pre-lecture VS Post-lecture HW

Feel free to work on the "Post-lecture" HW below if you're making good progress and want to continue: for **HW 02** continuing could be reasonable because questions "5-7" below directly follow up and extend "Pre-lecture" HW question "4"

The benefits of continue would are that (a) it might be fun to try to tackle the challenge of working through some problems without additional preparation or guidance; and (b) this is a very valable skill to be comfortable with; and (c) it will let you build experience interacting with ChatBots (and beginning to understand their strengths and limitations in this regard)... it's good to have sense of when using a ChatBot is the best way to figure something out, or if another approach (such as course provided resources or a plain old websearch for the right resourse) would be more effective

2.0.2 "Post-lecture" HW [*submission along with "Pre-lecture" HW is due prior to next TUT*]

5. Recreate (or resume) the previous ChatBot session from question "4" above, and now prompt the ChatBot session that you have a couple extensions of the code to show it, and then show it each of the extentions of the "Markovian ChatBot code" below in turn

1. Without just supplying your ChatBot session with the answers, see if the ChatBot can figure out what the extensions in the code do; namely, making character specific Markov chains, and using bigrams (rather than just the previous word alone) dependency... prompt your ChatBot session with some hints if it's not seeming to "get it"
2. Interact with your ChatBot session to have it explain details of the code wherever you need help understanding what the code is doing and how it works
3. Start yet another new ChatBot session and first show the ChatBot the original "Markovian ChatBot code" below, and then tell ChatBot that you have an extension but this time just directly provide it the more complicated final extension without ever providing the intermediate extension code to the ChatBot session and see if it's still able to understand everything extension does; namely, making character specific Markov chains, and using bigrams (rather than just the previous word alone) dependency... prompt the ChatBot with some hints if it's not seeming to understand what you're getting at...

Further Guidance

> **ALERT: Time Warning**. Regarding the comments below (which will likely be relevant and useful for you), you might find the potential learning experience that this provides to be a quite the rabbit total rabbit hole and time sink. You might end up finding out that you spent way more time than I should on learning the code!! So be mindful of your time management as there is much to do for many classes!
>
> *As you may or may not have already experienced in the previous problem, a ChatBot applied to this problem is likely to start explaining a bit more knowledge about Python than you need to know (as a student just trying to learn stats+DS); however, you'll probably feel like this "out of scope" context information is helpful to know (or at least be aware of) and easy to understand and learn if you use some addtional prompts to dig deeper into them. A ChatBot will be quite good at explaining and helping understand smaller chunks of code; however, if given too much information at once it can gloss over some information.*
>
> *That said, some topics here are potentially quite and advanced and too tricky! You might be able to ask the ChatBot to simplify its explanations and that might help a bit. But on the other hand, some topics, such as, "how does `nested_dict = lambda: defaultdict(nested_dict)` work?" might just simply be too advanced to really admit a simpler explanation via a ChatBot. You'll have to let these sorts of things go, if you come across explanations that just aren't improving or helping at at. In the case of `defaultdict(nested_dict)` specifically, the details here are well beyond the scope of STA130 and can be very safely ignored for now. The code will have reviewed and "walked thorugh" in LEC, but the perspectives espoused there will be the extent of the formal commentary and information regarding the coding topics we encounter in the*

Markov ChatBots code here.

Unlike with the Monte Hall problem, we will not inquire with the ChatBot to see if it can suggest any streamlining, readability, or usability improvements to the alternative versions of the "Markovian ChatBot code" we're examining

- *because doing so seems to result in the attempted creation of dubiously functional modular code with a focus on reusability (which is likely a result of ChatBot design being primarily a "computer science" topic), so ChatBot reponses here tend to orient around programming and system design principles (despite "Markovian" very much being a "statistics" topic)*

Programming and system design principles are beyond the scope of STA130; but, they are critical for modern data science careers... if you are interested in pursuing a data science career, it is imperitive that you complete courses like CSC263, CSC373, and perhaps an additional "systems design" course

---

Don't forget to ask for summaries of all your different ChatBot sessions and organize and paste these into your homework notebook (including link(s) to chat log histories if you're using ChatBot)

6. Report on your experience interacting with ChatBots to understand the Monte Hall problem and "Markovian ChatBot" code

1. Discuss how quickly the ChatBot was able to be helpful for each of the above questions, and if so, how?

I used a chatbot when I have code to understand and it was really helpful. I can ask it whenever I want and it responds quickly, often within a few seconds. This is very different from traditional Q&A forums where I might have to wait hours or days for a response

2. Discuss whether or not interacting with ChatBot to try to figure things out was frustrating or unhelpful, and if so, how?

Interaction with ChatBot to try to figure things out in terms of coding was helpful since it provided specific information and related examples. Also, step-by-step explanation with well-structured sentences allowed me to understand the code easily.

3. Based on your experiences to date (e.g., including using ChatBots to troubleshoot coding errors in the previous homework), provide an overall assessment evaluating the usefulness of ChatBots as tools to help you understand code

ChatBots are highly useful for troubleshooting coding errors, offering quick solutions with detailed explanation(step-by-step guidance). While they are not perfect for every scenario such as not fully understanding the context, they are still variable tools for learning and improving coding skills.

7. Reflect on your experience interacting with ChatBot and describe how your perception of AI-driven assistance tools in the context of learning coding, statistics, and data science has been evolving (or not) since joining the course It is true that I have used ChatBots in working on assignments so far, but I did not fully trust them because of unclear source notation in the sources. As a result of joining sta130 and making use of chatbots frequently throughout classes, I ended up discovering many advantages of using chatbots. A chatbot can be used to load coding to help you understand complex codes by completing the analysis within seconds. Additionally, detecting errors in the code can be used to help you debug the code as well. In my opinion, chatbots can be very helpful while studying coding because they provide well-organized answers and summaries to my questions, which makes it easier for me to learn the subject.

Further Guidance

> *Question "7" and the next question "8" are somewhat related to the first bullet point in the suggested interactions of the "Afterword" to the Homework from last week... consider reviewing that if you'd like a little extra orienting around what these questions are trying to have you explore*

8. ChatBots consume text data available on the web or platforms, and thus represents a new way to "search consensensus" that condenses and summarizes mainstream human thought

1. Start a new ChatBot session and discuss the relevance of learning and adaptability, communication, coding, and statistics and data analysis as skills in the modern world, especially with respect to career opportunities (particularly in the context of the data science industry)

Learning and adaptability, communication, coding, and statistics and data analysis are crucial in the modern world, especially within the data science industry. Together, they will create a powerful skill set that enhances employability, supports innovation, and facillitates career advancement in a rapidly evolving job market.

2. See if ChatBot thinks you could be a statistician or data scientist without coding or doing data analysis, and then transition your ChatBot conversation into a career exploration discussion, using the ChatBot to identify the skills that might be the most valuable for a career that you're interested

In statistics or data science, avoiding coding or data analysis is challenging, as both are central to the field. However, certain roles, such as consulting, research, or business strategy, may emphasize high-level thinking, decision-making, or statistical interpretation without requiring much hands-on technical work.

Skills for being statistician or data scientist - Communication: Translating data insights into business strategies. - Statistical Knowledge: Deep understanding of theory without hands-on coding. - Domain Expertise: Industry-specific knowledge (finance, healthcare, etc.). - Project Management: Managing data projects and leading teams.

3. Ask for a summary of this ChatBot session and paste it into your homework notebook (including link(s) to chat log histories if you're using ChatBot)

4. Paraphrase the assessments and conclusions of your conversation in the form of a reflection on your current thoughts regarding your potential future career(s) and how you can go about building the skills you need to pursue it

5. Give your thoughts regarding the helpfulness or limitations of your conversation with a ChatBot, and describe the next steps you would take to pursue this conversation further if you felt the information the ChatBot provides was somewhat high level and general, and perhaps

lacked the depth and detailed knowledge of a dedicated subject matter expert who had really take the time to understand the ins and outs of the industry and career path in question.

Further Guidance

While question 8 is not a part of the rubric, it is nonetheless a very good exercise that will likely be valuable for you if you engage it them sincerely

9. Have you reviewed the course wiki-textbook and interacted with a ChatBot (or, if that wasn't sufficient, real people in the course piazza discussion board or TA office hours) to help you understand all the material in the tutorial and lecture that you didn't quite follow when you first saw it? Further Guidance

Just answering "Yes" or "No" or "Somewhat" or "Mostly" or whatever here is fine as this question isn't a part of the rubric; but, the midterm and final exams may ask questions that are based on the tutorial and lecture materials; and, your own skills will be limited by your familiarity with these materials (which will determine your ability to actually do actual things effectively with these skills… like the course project…)

[ ]:
```
# Markovian Chatbot

# from collections import defaultdict
word_used = dict() # defaultdict(int)
next_word = dict() # defaultdict(lambda: defaultdict(int))
for i,word in enumerate(words[:-1]):

    if word in word_used:
        word_used[word] += 1
    else:
        word_used[word] = 1
        next_word[word] = {}

    if words[i+1] in next_word[word]:
        next_word[word][words[i+1]] += 1
    else:
        next_word[word][words[i+1]] = 1
```

[ ]:
```
# Markovian Chatbot Extension #1

word_used2 = defaultdict(int)
next_word2 = defaultdict(lambda: defaultdict(int))
for i,word in enumerate(words[:-2]):
    word_used2[word+' '+words[i+1]] += 1
    next_word2[word+' '+words[i+1]][words[i+2]] += 1
```

[ ]:
```
# Markovian Chatbot Extension #2

from collections import Counter, defaultdict
# `avatar` is a dataset, and `character` is one of it's columns
```

```
characters = Counter("\n"+ avatar.character.str.upper().str.replace(' ','.')+":
 ↪")
# this code changes the type of the `character` column to `str`; then,
# makes the text uppercase, and replaces spaces with '.'

nested_dict = lambda: defaultdict(nested_dict)
word_used2C = nested_dict()
next_word2C = nested_dict()

for i,word in enumerate(words[:-2]):
    if word in characters:
        character = word

    if character not in word_used2C:
        word_used2C[character] = dict()
    if word+' '+words[i+1] not in word_used2C[character]:
        word_used2C[character][word+' '+words[i+1]] = 0
    word_used2C[character][word+' '+words[i+1]] += 1

    if character not in next_word2C:
        next_word2C[character] = dict()
    if word+' '+words[i+1] not in next_word2C[character]:
        next_word2C[character][word+' '+words[i+1]] = dict()
    if words[i+2] not in next_word2C[character][word+' '+words[i+1]]:
        next_word2C[character][word+' '+words[i+1]][words[i+2]] = 0
    next_word2C[character][word+' '+words[i+1]][words[i+2]] += 1
```

2.1 Recommended Additional Useful Activities [Optional]

The "Ethical Profesionalism Considerations" and "Current Course Project Capability Level" sections below **are not a part of the required homework assignment**; rather, they are regular weekly guides covering (a) relevant considerations regarding professional and ethical conduct, and (b) the analysis steps for the STA130 course project that are feasible at the current stage of the course

Ethical Professionalism Considerations

2.1.1 Ethical Professionalism Considerations

1. If you've not heard of the "reproducibility crisis" in science, have a ChatBot explain it to you
2. If you've not heard of the "open source software" (versus proprietary software), have a ChatBot explain it to you
3. "Reproducibility" can also be considered at the level of a given data analysis project: can others replicate the results of code or analysis that you've done?
    1. Discuss with a ChatBot how jupyter notebooks and github can be used facilitate transparency and reproducibility in data analysis
4. Discuss with a ChatBot what the distinction is between replicability of scientific

experiments, versus the replicability of a specific data analysis project, and what your responsibility as an analyst should be with respect to both
5. Do you think proprietary (non "open source software") software, such as Microsoft Word, Outlook, and Copilot tends to result in high quality products?
    1. Do you think software product monopolies (such as the UofT dependence on Microsoft products) makes the world a better place?

Current Course Project Capability Level

2.1.2 Current Course Project Capability Level

Remember to abide by the data use agreement at all times.

Information about the course project is available on the course github repo here, including a draft course project specfication (subject to change). - The Week 01 HW introduced STA130F24_CourseProject.ipynb, and the available variables. - Please do not download the data accessible at the bottom of the CSCS webpage (or the course github repo) multiple times.

> At this point in the course you should be able to create a `for` loop to iterate through and provide **simple summaries** of some of the interesting columns in the course project data

1. Create two versions of the code, one for numeric and the other for categorical data, which provide a printout format that displays relavent summaries and the missing data counts for a given set of (either numerical or categorical) columns being examined

2. Combine the two separate `for` loops into a single `for` loop using an `if`/`else` **conditional logic structure** that determines the correct printout format based on the data type of the column under consideration

    1. *Being able to transform existing code so it's "resuable" for different purposes is one version of the programming design principle of "polymorphism" (which means "many forms" or "many uses") [as in the first task above]*
    2. *A better version of the programming design principle of "polymorphism" is when the same code can handle different use cases [as in the second tast above]*
    3. *Being able run your code with different subsets of columns as interest in different variables changes is a final form of the programming design principle of "polymorphism" that's demonstrated through this exercise*