12/9/21

MET CS 767 Assignment 6T: Bayesian Network

*Nathan A. Horak*

Go [here](https://pomegranate.readthedocs.io/en/latest/BayesianNetwork.html) to understand the famous Monty problem and to locate the code for it. I have included an edited version with the assignment. Systematically modify this code or data as indicated below.

Please leave the gray text and the headings unchanged.

# Favoring “B”

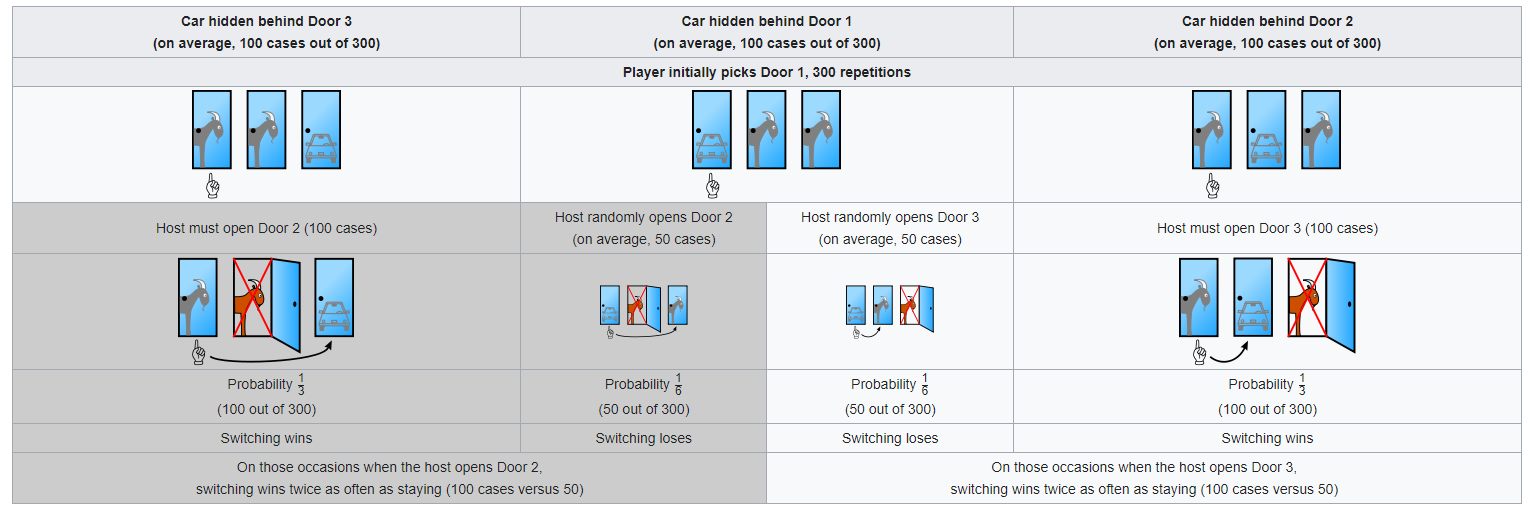
Suppose that Monty favors door B whenever he has the opportunity. Code this, selecting a particular degree of favor of your choice. Explain what you edited and why. Supply the output in this section of this document and explain why the output is what it is. Attach the code as a file named *part1*.

<https://colab.research.google.com/drive/1MBiGgN3GmwL2sEAw24QelNBWagnXs-ss#scrollTo=snbiRYs6xs-G>

When considering Section 1, it's first important to note when Monty can even choose to reveal door ‘B’. In the standard three-door setup, Monty can only reveal door ‘B’ if the guest did not originally select ‘B’ and the prize is not behind door ‘B’. Working on the code (see above), I added a favorability variable, which would represent the amount that Monty would favor door ‘B’ whenever he had an opportunity. I set an assert statement requiring the variable ‘favb1’ to be between ‘0’ (no favoring) and ‘0.50’ (fully favoring ‘B’)(see Appendix A). Then, I added probability adjustments within the ConditionalProbabilityTable (CPT). There are two instances where Monty decides between revealing two doors, one of which is door ‘B’. These occur when the door the guest selects contains the prize, and that door is not ‘B’. If the guest chooses ‘A’ and the prize is behind ‘A’, Monty can choose to reveal door ‘B’ or ‘C’. If the guest chooses ‘C’ and the prize behind ‘C’, Monty can choose to reveal door ‘A’ or ‘B’. In these two instances, I increased the probability that Monty selects door ‘B’ by ‘favb1’ and I reduced the probability that he selects the other door (‘A’ or ‘C’) by ‘favb1’. I chose to set ‘favb1’ to ‘0.50’- meaning that every time Monty chooses between door ‘B’ and another door, he always reveals door ‘B’. Here are the results given a ‘favb1’ value of ‘0.50’:



The first two distributions concern probabilities given that the guest began by picking door ‘A’. The first distribution shows the probability at start that each door holds the prize- these probabilities are all the same. The second distribution shows the probability that Monty will select each door. The probability that Monty will select door ‘A’ is ‘0’ as the guest picked that door at the start. Monty will reveal door ‘B’ with a probability of ‘0.66’ and door ‘C’ with a probability of ‘0.33’. Given the odds the prize is behind any door is the same across all doors and the guest initially chose ‘A’, Monty will reveal ‘B’ with 100% certainty (because of the favorability bonus) if the prize is behind ‘A’, reveal ‘C’ if the prize is behind ‘B’, and reveal ‘B’ if the prize is behind ‘C’. The third distribution shows the probabilities that the prize is behind ‘A’ and ‘C’ given that Monty revealed door ‘B’ to not contain the prize. Given a ‘favb1’ value of ‘0.50’, ‘A’ and ‘C’ are equally likely to hold the prize with a probability of 50%. By removing the chance completely that Monty would choose door ‘C’ when the guest initially picked door ‘A’ and the prize was behind ‘A’, all odds that Monty would reveal door ‘C’ in this situation( where the guest correctly guessed the prize location on first guess) shifted to revealing door ‘B’.



*Visual graphic of Monty Hall Problem with 300 cases and player initially picking first door, per Wikipedia[3].*

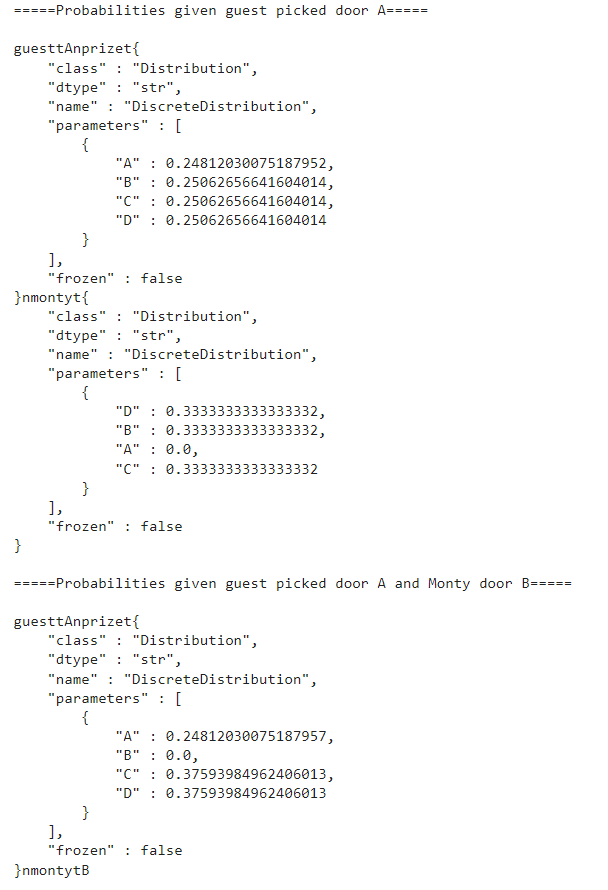
In the above graphic, the odds that switching wins are 2:1 given that the guest initially chooses the first door and Monty opens the second door (see the grey highlighted boxes). However, if Monty always picks ‘B’ for the middle instance (prize behind the first door) 50 instances of randomly opening the third door get transferred to the 50 instances of randomly picking the second door. Now it’s 100 vs 100, and a 50% chance to choose correctly. ‘0.50’ was the maximum possible value for ‘favb1’ but other values work in similar ways. Monty favoring to open a door raises the odds that he reveals that door in all simulations, but also lowers the bonus from switching doors given that he revealed from that door (see Appendix B).

# Four Doors

Suppose that the game has four doors rather than three. Give and explain new code fragments illustrative of what would be needed for this. (You don’t have to code an entire four-door version.)

<https://colab.research.google.com/drive/1MBiGgN3GmwL2sEAw24QelNBWagnXs-ss#scrollTo=0PpOKIjDlM5Z>

Adding an additional door presents a whole new dimension to the Monty Hall Problem. I first needed to decide how the addition of the fourth door would change the problem. Would Monty reveal only one door, leaving the guest to pick from his original choice or one of two unmarked doors, or would Monty open two doors- presenting the guest with a choice of sticking with their original door or swapping to the one remaining door? I thoroughly explored two methods of altering the code to fit Monty opening two doors (see Appendices C & D) but I fully worked through the method in which Monty reveals only one door (see above code). For the single door implementation of the Four Door Monty Hall problem, I began by altering the DiscreteDistribution (DD) objects ‘guest2’ and ‘prize2’ adding an additional key ‘D’ (representing the fourth door) as well as changing all percentages to 0.25 (25%), given the additional door. I altered and dramatically expanded the ‘monty2’ CPT accounting for the possibilities of ‘D’ being chosen for ‘prize2’, ‘guest2’, and ‘monty2’. I updated the conditional probability percentages to show the new problem framework. If the first two elements (‘guest2’, ‘prize2’) matched, the probability for the third element repeating would be ‘0.0’ [‘A’, ‘A’, ‘A’, 0.0] and it’d be 0.33 for each other value [‘C’, ‘C’, ‘A’, 0.33]. If the first two elements didn’t match, the probability would be 0.0 for a repeat prior element [‘A’, ‘B’, ‘A’, 0.0] and 0.50 for a new element [‘D’, ‘C’, ‘A’, 0.50]. Given these changes, see probability distributions below:



As mentioned in Section 1, the first distribution represents the odds that the prize is behind a given door at the start. Percentages around 0.25 (25%) for each make sense. The second distribution shows the odds that Monty will pick a given door to reveal given the guest’s initial pick. Given that the guest initially chose ‘A’, it makes sense that the chance of Monty to select that is 0.0 while the probability is spread equally among the other three doors. The last distribution shows the odds that the prize is behind a door given that the guest picked door ‘A’ and Monty revealed door ‘B’. Door ‘A’ has the same probability it had initially while door ‘B’ is 0.0, as it’s revealed. The remaining 75% is split among the two remaining doors- ‘C’ and ‘D’ at 37.5% each. Given the nature of the Monty Hall Problem explained in Section 1, these results make sense. Door ‘A’ still has the initial 25% chance of being the correct door, and the 75% chance from doors ‘B’, ‘C’, and ‘D’ is divided up between doors ‘C’ and ‘D’, given that ‘B’ was revealed to not hold the prize.

# Eldred the Dog

Suppose that an elderly dog named Eldred, who is good at sniffing presents, participates in the game. Eldred’s sniffing skills at a given time can be either *absent*, or *better than nothing*. Without supplying actual code but being as specific as you can, explain in a paragraph or how you would edit the given code to include Eldred.

<https://colab.research.google.com/drive/1MBiGgN3GmwL2sEAw24QelNBWagnXs-ss#scrollTo=DgwOGLwlQhlb>

Similar to Section 2, I experimented with multiple methods of approaching the problem (see Appendix E). I would begin by designating the door that the guest would initially select in the print statement. Then I’d flip a coin to determine if Eldred’s smell gave an improvement at selecting the prize. If the coin came up heads, ‘1’, Eldred’s skills were ‘better than nothing’- else, tails or ‘0’, they were ‘absent’. Given a variable of ‘door\_odds3’, representing base random chance that a guest would select the correct door plus the benefit of Eldred’s skills, ‘prize3’ would be set with a DD of odds for ‘favored\_door3’ at ‘door\_odds3’ while the other two doors were set to a smaller value dependant on number of doors in the problem and the value of ‘door\_odds3’. I chose to set ‘door\_odds3’ at 0.50. This means that if Eldred’s nose is active, given that the target door is door ‘A’, the prize odds for door ‘A’ will be 0.50 (50%) while the prize odds for doors ‘B’ and ‘C’ are 0.25 (25%) for each. Here are the distributions for favoring door ‘A’ with door odds of 0.50 (50%) and the guest initially selecting door ‘A’ and then also Monty revealing door ‘B’:

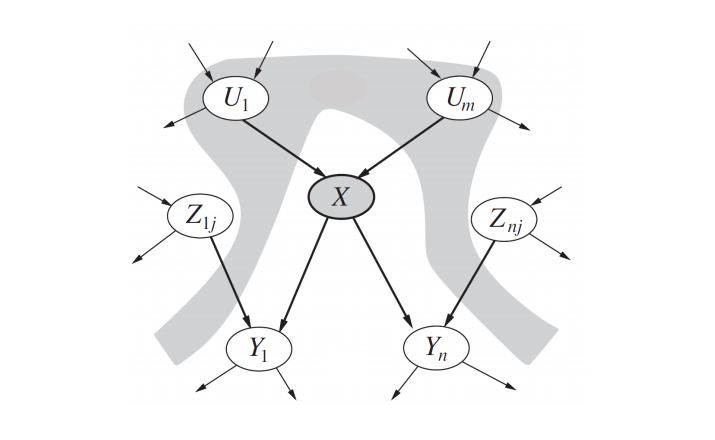


With Eldred’s nose active and ‘A’ as the favored door, the first distribution shows there is a 0.50 (50%) chance that the prize lies behind ‘A’ with a 0.25 (25%) chance that the prize is behind doors ‘B’ or ‘C’. The second distribution shows that given the initial pick of ‘A’, there is a 0.50 (50%) chance that Monty selects door ‘B’ or ‘C’. Lastly, the third distribution shows the probabilities given the guest picked door ‘A’ and Monty door ‘B’ that the prize is behind doors ‘A’ and ‘C’. Both odds are the same, and thus there is no pay-off to be gained by switching doors. In this way, Eldred’s enhanced nose actually hurts a player if it is this effective. Just as in Section 1, there is a trade-off relating to effectiveness of switching doors except this time, it's between Eldred’s skill at selecting the prize (his talent combined with natural luck) and effectiveness of switching doors. Interestingly enough, the guest can gain value if Eldred’s nose is not just absent, but actively bad (See Appendix F).

# Scaling

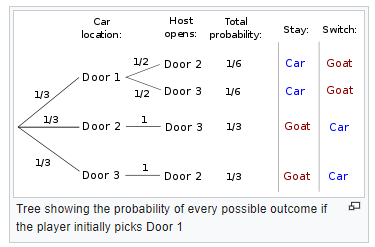
Imagine a real-world Bayesian network built to assess the economic impact of connected events, and implemented with pomegranate as in the given example. What would the main obstacles be to its practical development and use? Avoid generalities about Bayesian networks; concentrate on this type of application.

Probabilistic models can define relationships between variables and be used to calculate probabilities, according to Jason Brownlee at machinglearningmastery.com[1]. Fully conditional models may require an enormous amount of data and probabilities may be intractable to calculate in practice. Per Brownlee, an alternative is to develop a model that preserves known conditional dependence between random variables and conditional independence in all other cases[1]. This can be done with a Bayesian network (see more in Appendix G).



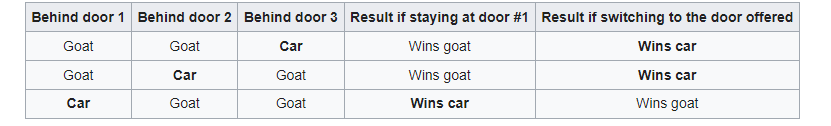
*Bayesian Network showing edges between variables, from towardsdatascience.com[2].*

Although the pomegranate package has significant capability, I foresee a few issues with using it to implement a Bayesian Network for the purpose of assessing the economic impact of connected events. Structurally, it seems like it would be very difficult to create a CPT with accurate conditional probabilities. Economic changes are spurred by many different events (some of which are dependent on one another and some not). While the conditional probabilities for a situation like Monty Hall are easy to calculate and firmly set in stone (0.0, 0.33, 0.50, or 1.0), who is to say that p(recession=’Y’ | interest\_rate=’Low’, pandemic=’Y’, president\_party=’D’, total\_party\_control=’Y’, AND unemployment\_rate=’Low’) = 0.05 (5%)? Why not 0.10 (10%)? And over what time period is this percentage chance reflecting the odds of? The Monty Hall probabilities are constructed concretely given the context of the problem, logic, and basic statistics with mathematics (see below graphic), but each conditional probability value would have to be justified.



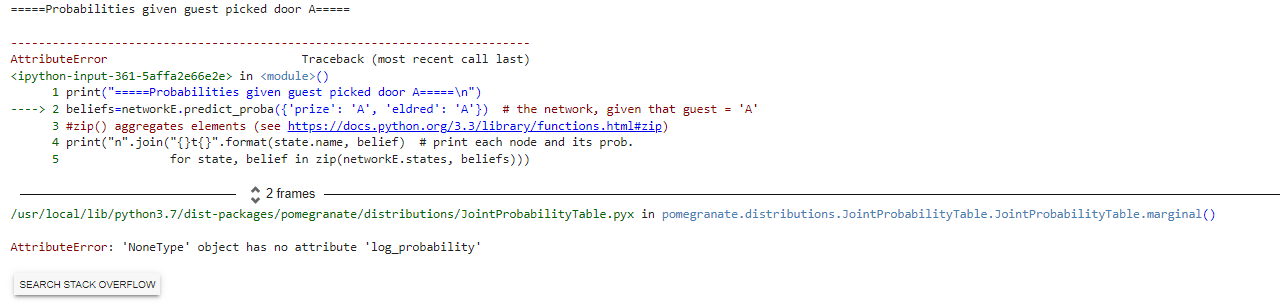
*Probability layout of Monty Hall Problem, per Wikipedia[3].*

The aforementioned conditional probability shows some additional issues using the pomegranate package. If one chose to represent ‘interest\_rate’ as a continuous variable (0.005, 0.010, 0.015, etc) this would not work as the pomegranate package would require these nodes to be in categorical data form. So, the user would be required to designate interest rates in categories (say: ‘Low’, ‘Medium’, ‘High’). But this presents another issue: is three categories the best way to represent interest rates? Who decides the cut-off points between categories? There is a lot of subjectivity here which simply isn’t present in the Monty Hall problem (see below).



*Outcomes of the Monty Hall Problem are easily shown in table form, from Wikipedia[3].*

Another issue relates to the complexity of assessing economic impact from connected events. As mentioned above, there are many different factors that come into play with economic events. It would probably be very hard to find a consensus among economists regarding variables and their impacts on financial recession or depression. Even if you did reach somewhat of a consensus, there would be so many variables that constructing a CPT would be an extremely huge endeavor. Adding another variable to the CPT for my attempts on Section 2 was a lot of work and given that each additional variable increases the size of the CPT, a CPT with enough variables to model a real-world economic model would be truly massive. Furthermore, as mentioned above, you would have to justify the conditional probability for each row of the CPT which presents compounding issues.



*Output error given additional node added in Appendix E method attempting to incorporate Eldred.*

Lastly, there would likely be issues establishing conditional or independent relationships between variables. With so many complex variables, of which people have differing views on, it is not hard to see disputes popping up. Given the layout of the Monty Hall Problem, these issues were not present. But it’s not hard to see a disagreement on conditionality/independence between, say, ‘pandemic’ and ‘president\_party’ (perhaps even depending on the value of ‘president\_party’) or ‘unemployment\_rate’ and ‘president\_party’. Additionally, I did have some issues implementing additional nodes to the pomegranate Monty Hall Problem (see Appendices D & E and resulting error message above) and assuming that a foundation was agreed upon and CPTs coded, coding the complex web of nodes would still have to be addressed.

# References

Show that you used a wide variety of resources by listing them below and clearly indicating in the body above where you used. Make sure to use proper referencing in your paper. We suggest using the APA format, but other formats are fine as long as they clearly distinguish your work from the work of others in your response. In general, observe the stated plagiarism rules.

[1] Brownlee, Jason. (2019, October 11). *A Gentle Introduction to Bayesian Belief Networks*. machinelearningmastery.com. Retrieved December 13, 2021. <https://machinelearningmastery.com/introduction-to-bayesian-belief-networks/>

[2] Soni, Devin. (2018, June 8). *Introduction to Bayesian Networks*. towardsdatascience.com. Retrieved December 13, 2021. <https://towardsdatascience.com/introduction-to-bayesian-networks-81031eeed94e>

[3] *Monty Hall Problem*. wikipedia.org. Retrieved December 13, 2021. <https://en.wikipedia.org/wiki/Monty_Hall_problem>

**Appendix**

[A]

As mentioned in Section 1, Monty can only reveal door ‘B’ if the guest did not originally select the door and the prize is not behind door ‘B’. In the exception of either of those circumstances, Monty is not able to reveal door ‘B’ to the guest. For the ‘favb1’ variable, ‘0.50’ represents fully favoring ‘B’ as the only time that Monty has a choice of which door to reveal, the odds to pick each are 0.50 in the standard three-door setup (0.50 + 0.50 = 1.00, or 100% chance). I decided to disallow ‘favb1’ from being a negative number due to the second print statement (showing that guest picked door ‘A’ and Monty door ‘B’) which I wanted to use to show the output. A value of ‘-0.50’ would indicate that Monty never selects B, but this is logically incompatible with the probability statement I wanted to use.

[B]

From Section 1, there is sort of a trade-off between the probabilities of the second distribution and third distribution when adjusting favoring a door to reveal. See below the three probabilities with no favoring:



With no favoring, there is an equal chance (50% each) that Monty reveals the contents of doors ‘B’ and ‘C’ given that the guest initially chose door ‘A’ and a 66% chance (compared to 33%) that switching doors selects the correct door given that the guest initially chose ‘A’ and Monty revealed ‘B’.



For a ‘favb1’ value of 0.25, Monty will reveal door ‘B’ given door ‘A’ was initially revealed 58.3% of the time while the pay-off for switching doors given initial door ‘A’ and Monty revealed door ‘B’ falls from 66% to 57.1%. From these probabilities and the odds from Section 1, we can see that removing the random chance that Monty selects doors tanks the pay-off of the switching doors strategy. This is something to keep in mind when considering variations of the Monty Hall game.

[C]

<https://colab.research.google.com/drive/1MBiGgN3GmwL2sEAw24QelNBWagnXs-ss#scrollTo=Z_2rBULj6eXM>

As mentioned in Section 2, I looked at two ways to code the Monty Hall Problem with four doors and two door reveals. I first decided that I would attempt to alter the meaning of the States associated with the Bayesian Network. The first two States would remain the guest’s choice and the prize location, but the third would be changed from door Monty opens to the non-initially chosen door left after Monty revealed two doors. The logic I used was that if Monty revealed two doors in a four-door problem, there would always be another door left (not chosen initially by the guest). I could have a CPT represent the probability of each door not chosen remaining at the end. Given that the true prize door could be the door left at the end not initially chosen, I removed the edge connecting nodes ‘prize\_nodeC’ and ‘no\_monty\_nodeC’. I updated all probabilities to fit- if the first element and third element matched, the probability would be 0.0 (as the node not chosen by Monty that wasn’t initially chosen can’t be the door initially chosen) [‘B’, ‘B’, ‘B’, 0.0] and all others to 0.33 (33%) [‘C’, ‘A’, ‘D’, 0.33]. The model does not work appropriately if the edge between ‘prize\_nodeC’ and ‘no\_monty\_nodeC’ is removed and the results do not make sense if the edge is left in. Examining the code, this approach seems faulty given that for every two element combination of the first two elements [‘A’, ‘C’] there are three probabilities of 0.33 (33%) [‘A’, ‘C’, ‘D’, 0.33] and one of 0.0 [‘A’, ‘C’, ‘A’, 0.0]. Although my reasoning was sound, it did not translate well over to using the BayesianNetwork object. I tried a different approach in Appendix D.

[D]

<https://colab.research.google.com/drive/1MBiGgN3GmwL2sEAw24QelNBWagnXs-ss#scrollTo=p3wSUs0n8zEk>

For Appendix D, noting my issues from my prior attempt in Appendix C, I sought to remedy the issue. I opted to create an extra node- one for the second door opened by Monty. There were now two nodes for Monty- one for the first door, and another for the second. I began setting up the implementation much like my rendition in Section 2. The first node would be chosen with a probability chance dependent on the first two elements (‘guestD’ and ‘prizeD’). Then I would incorporate a second CPT for the next Monty selection, conditionally basing it off of the first Monty selection. For the second CPT, the probability would be 0.0 for the two elements matching (as Monty can’t remove the same door twice) and 0.33 (33%) otherwise. I then added the code for the additional state and would create an additional edge between the first Monty selection and the second. I added an additional print statement to reflect Monty selecting two doors. This is the output:





Interestingly enough, the correct output for Section 2 was visible in two locations- the first distribution under “Probabilities given guest picked door A and first Monty door B” and the lone distribution under “Probabilities given guest picked door A, first Monty door B, and second Monty door C”. The first of these is correct and what is expected with one door opening, however, the second is not. It should show door ‘A’ with a probability of 0.25 (25%), doors ‘B’ and ‘C’ with a probability of 0.0 given that Monty revealed them both, and door ‘D’ with a probability of 0.75 (75%). The 0.375 (37.5%) allotted to door ‘C’ should be bundled with door ‘D’s odds. Furthermore, it seems as though the transitive property does not apply to node edges. I established an edge between the first and second Monty choices, but that did apparently not transitively create an edge between the second Monty reveal and the chosen guest and prize nodes. It’s worth mentioning that if I try to add edges here to fix the issue, the code produces an error “Attribute Error: ‘NoneType’ object has no attribute ‘log\_probability’”. This can be seen in the third distribution of the first print statement and the second distribution of the second print statement. I think my logic here is much stronger than in Appendix C. The addition of another node incorporated in the manner I explained above should allow for the four-door variant of the Monty Hall Problem to be evaluated.

[E]

<https://colab.research.google.com/drive/1MBiGgN3GmwL2sEAw24QelNBWagnXs-ss#scrollTo=rjA2S7VPGubr>

Just as in Section 2, I experimented with multiple ways of updating the code. I attempted to construct a way of incorporating Eldred’s nose in a manner similar to Appendix D using two CPTs. I created a DD for ‘prizeE’ prize door and ‘eldredE’ for whether Eldred’s nose was active or inactive. I established a ‘dog\_bonusE’ of 0.17- the increase in effectiveness of identifying the prize door from purely guessing. I also incorporated a ‘dog\_penaltyE’ variable for the penalty to assign to each non-chosen probability value when Eldred’s nose wasn’t active. I created a CPT for ‘guestE’, showing the conditional probabilities the guest has at choosing doors depending on the prize location and whether Eldred’s nose is online. When Eldred’s nose was on ‘better than nothing’ or ‘B’, the odds that the guest’s pick would be the true prize would increase. Next, I created a second CPT showing the probabilities for Monty’s selection given ‘guestE’ guest’s choice and ‘prizeE’ prize location. I updated the code to add Eldred’s node to the network and then compiled. With this setup, the odds for selecting an initial door will be conditionally dependent by whether Eldred’s nose is working and where the prize is located. Then, this probability would be used in the second CPT to finish the network. Just as in Appendix D, I think my logic here is sound but there is something still to be ironed out in my implementation.

[F]

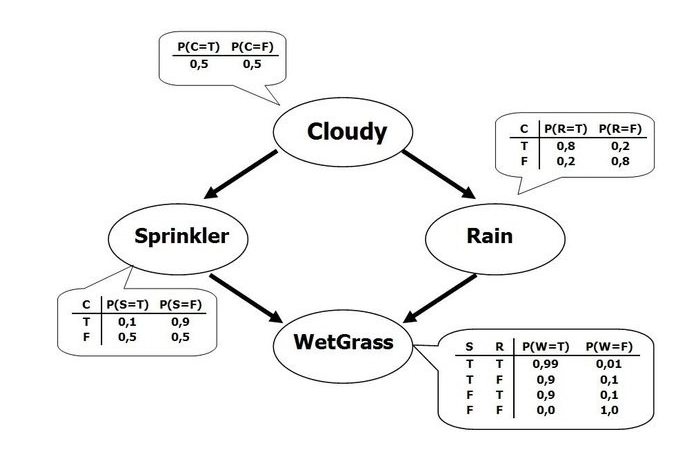
As mentioned in Section 3, Eldred’s nose can help the guest find the present more often by being actively bad (worse than random) at the first door selection. Given that Eldred identifies the correct door among three doors 0.25 (25%) of the time, here are the associated probability distributions:



As seen above, given that Eldred’s nose is actively bad and that the odds of him identifying the correct door from among three is 0.25 (25%), the remaining 0.75 (75%) is split initially among doors ‘B’ and ‘C’. After Monty reveals the contents behind door ‘B’ as empty, there is now a 0.75 (75%) chance that the prize is behind door ‘C’. In this case, Eldred’s horrendous nose now further improves the effectiveness of the switching door strategy.

[G]

Bayesian Networks are a type of probabilistic graphical model that uses Bayesian inference for probability computations, according to Devin Soni at towardsdatascience.com[2]. Per machinelearningmastery.com, a Bayesian Network is a useful tool to visualize the probabilistic model for a domain, review relationships between random variables, and reason about causal probabilities for scenarios given available evidence[1].



*Visual representation of simple Bayesian Network, from towardsdatascience.com[2].*

A Bayesian Network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable[2]. If an edge (A, B) exists in the graph connecting random variables A and B, it means that P(B|A) is a factor in the joint probability distribution, so we must know P(B|A) for all values of B and A in order to conduct inference. The Markov property allows us to simplify the joint distribution for a Bayesian Network to the product of P(node|parents(node)) for all nodes[2].

# Evaluation

