** This is a living document. I might make updates/changes over the course of the semester.

Components of Project

1. Either choose to work alone, or find a partner to work with. I think working in a group of two is usually best. Ed Discussion can be used to look for students with similar interests.
2. Select, read, and understand in depth **1 research paper** on a topic related to the course material. It must involve a theoretical result. Most students will choose algorithms papers, but it is okay if your paper is about something else, as long as it uses similar tools to those studied in class. For example, you might pick a learning theory paper that uses exponential tail bounds and ϵ-net methods (which we will learn). Additionally, when working on your project, I fully expect you to look at other papers related to the one you choose, but you should start with one.
3. Formulate a new **research question** about that paper. This can be a theoretical question **or** an empirical question, but empirical questions need to be more complicated than "Is Algorithm A faster than Algorithm B in practice?". See below for example questions.
4. Try to solve this question, with the understanding that you may not succeed, or you might succeed in learning something different from what you originally intended! You will not be graded on if you obtain a novel, research-level result, but on how you approach the problem and what you try. That said, every year I have run this class there have been projects with novel ideas that eventually turned into research papers.
5. Write-up what you learned in a **report of at least 6 pages**. You should clearly explain the problem you aimed to solve, any necessary context to appreciate the problem's relevance, and what you accomplished. If you found barriers to accomplishing what you originally desired, describe those! Empirical papers should report any empirical findings with effective plots and figures. If you complete an empirical project, you should also share your code with me in some way (GitHub, Collab notebook, zip file).

Components of Reading Group

1. 1.5 hours each week, ideally broken into 1-1.25 hours for paper presention, and the rest of the time for paper discussion. **All students completing a final project must participate in the reading group.** It is okay if you need to miss a week or two due ot scheduling conflicts (please let me know in advance), but otherwise you will be expected to attend each meeting.
2. One or two student **presenters** select a research paper on computational methods in data science and machine learning, and present the main results, proof ideas, and applications each week. This can be a paper the students are considering for their project. The paper should be chosen at least **one week in advance** of when it will be presented.
3. One or two **discussion leaders** also read the paper in depth and prepare a set of 3-4 discussion questions. These questions can cover topics like how the paper compares to prior work, what natural extensions of the problem setting could be considered for further research, what parts of the paper or proof seem like they could be improved, etc.
4. Everyone else reads the abstract and introduction of the paper prior to the reading group so that they can be prepare to learn the main results, and participate in discussion.

Deadlines

- By **Tuesday, 9/28**, complete the poll for reading group times and let me know if you are considering completing a project for the class.
- By **Tuesday, 10/5**, give me a final decision if you will complete a project, and if you will be working alone or who you have chosen as a partner. Let me know if you would like to meet to discuss possible project topics, or are having trouble choosing between a few options.
- By **Tuesday, 10/5**, signup here as either a presenter or discussion leader for one week of the reading group. You don't need to select your paper yet, unless you are one of the first weeks.
- By **Tuesday, 11/04**, select your paper and submit a 1-page proposal describing the paper's subject, and listing 3 possible ideas for research questions. The TAs and I will set up individual group meetings as needed to help you narrow down and refine your questions.
- By **Friday, 12/17**, submit final report (the week that our last class meets).

Tips for Choosing a Paper

- Look at my list of recommended papers, which will likely grow over the next few weeks. Feel free to add to this list yourself! If you don't have a partner, put your name down next to any papers you are interested in, in case someone else is also interested.
- If you find a paper on a topic you like, but don't want to work on that specific problem, the paper seems too complicated, etc., look at what else the authors have been publishing (either on their webpages or Google scholar.)
- I would lean towards choosing a short, simple paper (e.g., under 12 pages) over a long one. Since this is course project, you will only have a limited amount of time to work on it, so choosing a paper you can read and understand quickly makes sense.
- Look at papers from recent conferences. Lists of accepted papers can typically be found online. Try to choose papers from relatively well regarded conferences -- there will be a big difference in quality. Here are a some I recommend:
    - Symposium on Foundations of Computer Science (FOCS), Symposium on Theory of Computing (STOC), Symposium on Discrete Algorithms (SODA), Innovations in Theoretical Computer Science (ITCS).
        - These are a the "top 4" theoretical algortihms research conferences. Papers are typically high quality. Papers from ITCS are often on slightly more creative problems, and might be a bit shorter and less technical, so would be easier to start with.
    - European Symposium on Algorithms (ESA), International Colloquium on Automata, Languages and Programming (ICALP), International Conference on Randomization and Computation (RANDOM), International Symposium on Distributed Computing (DISC), Principles of Distributed Computing (PODC), etc.
        - Other great algorithms conferences. There are many more and I can't list them all, but feel free to run a venue by me if you are unsure of a paper's quality.
    - Symposium on Simplicity of Algorithms

        - A relatively new theoretical algorithms conference emphasizing short + simple results. A great place to find project papers.
    - Neural Information Processing Systems (NeurIPS), International Conference on Machine Learning (ICML), International Conference on Learning Representations (ICLR), AAAI Conference on Artificial Intelligence (AAAI), International Conference on Artificial Intelligence and Statistics (AISTATS)
        - Big machine learning conferences. Lots of papers, and not all contain theory (or good theory) but there will be some that do.
    - Conference on Learning Theory (COLT), Conference on Algorithmic Learning Theory (ALT).
        - Theory focused machine learning conferences, typically with high quality papers.
    - Knowledge Discovery and Data Mining (KDD), Web Search and Data Mining (WSDM).
        - Big data mining conferences. Same caveat as big machine learning conferences.
- **Talk to me, Aarsvhi, Indu, or Teal.** Even if we don't know about the specific topic you are interested in, we can often help point you in the right direction, help you find reseachers whose work you might want to look at, etc. This can often save students a lot of time.

Example Research Questions

- Choosing a good research question is challenging. I'm here to help. A few tips:
    - Start small. Think about the smallest possible extension or generalization of a result you read in a paper and start with that. Big reseach results often come out of starting with small questions.
    - At the same time you do want to avoid *trivial* questions which e.g. could be solved by a very direct modification of the authors techniques. It's good to find extensions that, while small, would likely have been included in the paper if the authors themselves knew how to solve them.
    - Consider the other side of the coin: e.g. if a paper proves that a problem can be solved with $O(S)$ space, a natural question is to try to prove that *no algorithm* can do better, or no algorithm can do better than $O(T)$ for some $T < S$. I.e., prove a lower bound. This will often be easier than improving the authors algorithm directly, and can provide another angle to thinking about the problem.
    - Consider a different goal: if the goal of a paper is to minimize time complexity, can you ask about space complexity? Or vice versa?
    - The more important a problem is, the more it makes sense to ask smaller questions about it. For very important problems (e.g. distinct elements) improving even the constant factor infront of a runtime or space complexity result can really matter. Can you make a minor algorithmic change that improves the constant factor for some method? For important problems, it is also nice to find "parameter free" algorithms that are simple to implment: for example, this paper shows a step size schedule for gradient descent that works for any combination of strong convexity, smoothness, Lipschitzness, etc. In contrast, the methods we will analyze in class require "guessing" these parameters ahead of time.

Theoretical

- This paper shows that the Johnson-Lindenstrauss lemma provides optimal dimensionality reduction for preserving ℓ_2 distances between vectors. Pick any downstream appliction of JL (e.g., one of the problem set problems). Can you prove that it is not possible to solve the problem with a more compact dimensionality reduction than that provided by JL?
- This paper formalizes a notation of what a "good" 2D visualization of a high dimensional dataset it. They prove that under certain conditions, the popular t-SNE algorithm finds a "good" visualization. Can the same methodology be applied to other popular methods for visualizing high-dimensional data (like Isomap or spectral clustering)? Or is there some better definition for "good" visualization?
- This paper proves that some popular optimization algorithms like ADAM and RMSProp (ubiquitouos in training deep learning models) fail to converge for even very simple convex problems. They propose a small change to fix the issue. A natural question is if the set of "bad instances" is common, or extremely unlikely. A natural way to attack this would be to see if there is a natural set of random problems such that, if you choose a problem from this set, ADAM fails with high probability. If not, can you show that it succeeds on nearly all random instances? This might explain why it works well in practice.
- This paper gives optimal group testing schemes, but like those studied on our problem set, they require knowing the number of infected individuals k ahead of time, before tests are run. What if we don't know k ? Can you find an algorithm that is optimal or near optimal for any k?

Empirical

- This paper shows that a hashing method known as "simple tabulation hashing" performs nearly as well as fully-independent hashing on many tasks, even though the hashing scheme isn't even 4-indepenent. I suspect this paper originated from an experimental observation that simple tabulation hashing worked better for load balancing than expected. Can you design an experiment to find other commonly used hash functions with similar properties? E.g. methods like MD5, SHA256, MurmurHash, etc.? Can you design an experiment to figure out how "independent" these hash functions are? Can you experimentally check if possible low-independence limits their performce on a task like load balancing?
- Later in the class we will study simple randomized models for graphs (e.g. social networks) that make it easier to analyze certain algorithms, and ideally capture much of the structure of real word networks. Examples include the stochastic block model, and the preferential attachment model. Using real social network data, can you design an experiment which either confirms that a model is a good fit for real data, or rule this out? For example, a common claim is that the degree distribution of a random graph generated via the preferential attachement model matches oberserved degree distributions in real-world social networks. Do other properties match? E.g. the number of triangles in the network? Or number of 4 cliques? Can you think of a new random model which does capture these properties?
- This paper shows how to estimate the size of a network like the internet or a social network given the ability to crawl between nodes/edges in that network. For the analysis, at each step of the method, we must randomly walk for a number of steps proportional to the mixing time of the network. How sensitive is the method to this parameter? E.g. what if we misjudge the mixing time and randomly walk for less steps? You could test out the methods sensitivity both on large synthetic networks, and real-world data sets found online.