Question 1:
I appreciate this question; I will take some inspiration from the linear algebra course that I took last semester.

My favorite technique simplifying and extracting information from high-dimensional data. It's called Principal Component Analysis (PCA). In layman's terms, PCA is like taking a bunch of pictures of an object from different angles and then finding the best view that shows the most important features.

Imagine you have a large group of people, and you want to understand their characteristics, such as height and weight. However, you also have information on other features, like age, eye color, hair color, etc. Now, let's say you want to focus on just the most important features to analyze the group's overall trends. PCA helps you do this by finding patterns in the data, and then reducing the number of dimensions without losing too much important information.

In a real-world example, let's say you're working on a project to recommend movies to users. You have data on thousands of movies and their features, such as genres, actors, directors, and ratings. PCA can help you find the most important features that explain users' preferences, and then use these features to create a simplified model that can recommend movies based on these key characteristics. While Netflix may not reveal its secrets to its recommendation engine, this algorithm, in combination with other techniques, could be used to push movies and TV shows.

I haven’t had the chance to apply PCA to production-ready products; however, I have had the opportunity to explore had we can use PCA to help computers recognize handwriting (I’ll submit this project along with my answers). I used a very popular dataset; The MNIST (National Institute of Standards and Technology) datasets contain images of handwritten numerical digits (the M stands for 'mixed,' since the samples are due to both U.S. Census Bureau employees and high school students). This database is commonly used to help develop image processing systems and to gauge the performance of machine learning models. The full MNIST set contains roughly 6,000 labeled images of each digit '0' through '9'. Here’s a step-by-step process of what I did:

1. Prepare the data: First, we need to preprocess the data. This usually involves normalizing the pixel values, so they are in a similar range, like between 0 and 1, which helps the PCA algorithm work better.

2. Apply PCA: Next, we apply PCA to the MNIST dataset. PCA identifies the most important components (directions) in the data that capture the largest variance. These components are linear combinations of the original features (pixel values). By choosing a certain number of principal components (e.g., 50 or 100), we can significantly reduce the dimensionality of the data while still retaining a lot of information.

3. Transform the data: Now that we have the principal components, we can transform the original data into a lower-dimensional representation. Each image in the dataset will now have fewer features (e.g., 50 or 100 instead of 784), making it easier and faster to work with.

4. Train a machine learning model: With the transformed data, we can now train a simple machine learning model, like logistic regression, k-Nearest Neighbors, or a support vector machine (SVM), to classify the images. The model learns patterns in the lower-dimensional data to distinguish between the different digits (0-9).

5. Evaluate the model: After training, we can evaluate the model's performance using a test dataset that the model hasn't seen before. If the model has learned the patterns well, it should be able to classify the digits in the test dataset with reasonable accuracy.

6. Classify unlabeled images: Finally, we can use the trained model to classify new, unlabeled images of handwritten digits. We first apply PCA to the new images to transform them into lower-dimensional space, then use the trained model to predict the digit labels.

A use-case for models that can recognize handwriting is in assistive technologies. Handwriting recognition models can be incorporated into assistive devices for individuals with disabilities, such as dyslexia or impaired vision. These devices can convert handwritten text into digital text or even speech, making it more accessible for the user.

If I were to explain this concept to a friend who hasn't taken math or computer science courses, PCA is a way to look at a lot of complicated data, find the most important pieces of information, and simplify it so it's easier to understand and work with.

Question 2:
This situation/project shines a light on a very important topic that many undergraduate students don't consider when thinking about research, which is the cost! I understand that a corporate project may be motivated by money, but research always has a larger motivation. (Apologies for the mini-rant)

Imagining this project, to measure airborne COVID particle concentration in an airport terminal with a limited number of portable sensors, I will emphasize the importance of sensor placement, an understanding of the environment and way to control it, and finally analysis. Below are some thoughts/action items that I would consider:

1. Optimal sensor placement: The team should use optimization techniques, such as genetic algorithms or simulated annealing, to find the best positions for the sensors. These methods can help you maximize coverage and minimize spatial sampling bias by considering factors like sensor coverage, overlapping measurements, and constraints imposed by the airport terminal's layout. Furthermore, the team should leverage historical data on passenger flow, occupancy, and previous air quality measurements (if available). This information can help you make informed decisions about the optimal placement of sensors.

2. Understand the environment: The team should study the layout of the airport terminal, including the location of entrances, exits, ventilation systems, high-traffic areas, and seating zones. This understanding will help you identify critical points where the airborne COVID particle concentration may vary or be higher. Moreover, since the sensors are portable and can be easily repositioned, the team should consider using an adaptive monitoring strategy. By continuously analyzing the collected data, we can adjust sensor placements over time to respond to changes in the environment or passenger flow.

3. Analysis: To create a complete map of the COVID concentration, the team should combine data from different sensors using data fusion techniques. Spatial interpolation methods, such as kriging or inverse distance weighting (IDW), can help estimate COVID concentrations in areas between sensors. Further analysis using computational fluid dynamics (CFD) modeling/simulations can help with uncovering airflow patterns and particle distribution within the airport terminal. By integrating this information with sensor data, we can obtain a more accurate mapping of COVID particle concentration.

By incorporating these techniques, we can optimize the placement of the limited number of sensors to obtain the most accurate mapping of airborne COVID particle concentration in the airport terminal.

Question 3:
(I actually wanted to talk about a project where I built a wooden power rack and concrete weights during COVID-19 without any woodworking experience because all of the gyms around me closed due to the pandemic, but I thought that the project below may be more fitting to the role - I would love to talk about it more another time however 🙂)

I'll link the project here: https://github.com/cleeclee123/PChecker
Note: the project is not production ready

I'll touch on some background before I talk about the above project. Last summer, I spent my free time building this: (https://github.com/cleeclee123/Project-Helper-Function). In short, it's a search engine scraper that finds a code snippet that matches your query - I like to joke around saying that this is a dumber version of GitHub Copilot. The main roadblock that I had with this project is that search engines (Google, Bing, Yahoo) do not like people scraping search results. I would either get a reCaptcha to solve or get my IP address blacklisted for some time (I understand that scraping it against their terms of service but it project was purely for fun). Since I was scraping a bunch of data, the number of requests per second did not match what a human was capable of doing. It is easy to detect this by simply looking at the HTTP headers from the request and then shadow-banning that IP address and/or user agent from making more requests. To solve this problem, I needed IP addresses. It is easy to find free IP addresses on the internet but they might not be safe. Some reasons why:

1. Malicious intent: Some free proxies are set up by cybercriminals with malicious intent. They can use the proxy to intercept your sensitive information, such as login credentials, credit card numbers, or personal messages, and use this information for identity theft or other fraudulent activities.
2. Limited features: Free proxies often have limited features compared to paid alternatives. For example, they might not support secure protocols like HTTPS, restrict access to specific websites, or limit your ability to use streaming services or other online platforms.
3. Advertisements and malware: Free proxies may inject unwanted advertisements or pop-ups into your browsing sessions to generate revenue. Some free proxies may even inject malware or viruses into your computer, putting your data and system at risk.

I needed to be sure that the proxies that I was using from the internet were safe (I do not recommend this, again this project was purely for fun and exploration). This is where my project, PChecker comes in. I set up an endpoint that bounces back the request you sent using the proxy. With the response I get from the endpoint, I start to look at the HTTP headers to see if any critical information is exposed by looking for certain fields, then classifying them into 3 categories: Transparent, Anonymous, and Elite. In short, transparent proxies leak your actual public IP address and tell the server that you are using a proxy; anonymous proxies will not leak your actual public IP address but tells the server that you are using a proxy; elite proxies will not leak your actual public IP address and does not tell the server that you are using a proxy.

This project provides a summarized way for users to determine the anonymity and safety of HTTP proxies before using them.

A change that I would make for my next revision is determining a better way to check if a proxy supports SSL/HTTPS. Currently, I connect a socket connection to the proxy server and send an HTTP CONNECT request to open a secure tunnel (https://github.com/cleeclee123/PChecker/blob/main/PChecker/src/checker/PCheckerFaster.ts#LL209C10-L211C28). However, in cases where a proxy server does not support HTTPS, this CONNECT request will hang indefinitely or when it times out. This happens because the proxy server is not set up to handle CONNECT requests, so when the server gets one - it does not know what to do with it!

What I have concluded is that I need to approach this problem on the basis of why I started this project. The initial motivation of the PChecker was to determine the anonymity of an HTTP proxy in a blazingly fast manner - I have achieved that (https://github.com/cleeclee123/PChecker/blob/main/PChecker/src/checker/PCheckerFaster.ts#LL89C1-L183C4) I will set up two different endpoints, one checking just the anonymity of the proxy and the other endpoint checking the safely/HTTP support of the proxy.

Question 4:
I will have to admit that the majority of my experiences pulling signals out of noisy data streams have followed the “textbook approach” or have relied on software packages. I believe that as I progress as a student and as an academic; I will have more opportunities to derive more creative approaches. Below, I will talk about my explorations and findings with Discrete Cosine Transform (DCT). I will attach all supporting material.

Discrete Cosine Transform (DCT) is a mathematical technique used in signal processing and image compression. It is a method for transforming a finite sequence of discrete data points, such as digital images or audio samples, into a frequency domain representation. The primary purpose of DCT is to represent the data in such a way that it can be more efficiently compressed, stored, and transmitted. DCT works by decomposing the input data into a linear combination of basis functions, which are cosine functions with different frequencies. These cosine functions are orthogonal, meaning they are uncorrelated, and this property makes DCT particularly useful for data compression, as it allows for the separation of different frequency components in the data.

I have attached examples of Image compression and Audio compression.

In image compression, DCT is the core technique used in the widely adopted JPEG compression standard. It efficiently represents image data by capturing the most visually significant information in a compact form, enabling significant compression with minimal loss of perceived image quality.

In audio compression, DCT is used in audio codecs like MP3 and AAC, where it helps to identify and discard less audible frequency components, resulting in smaller file sizes without significantly affecting the perceived audio quality.