Homework #0

Due: January 31, 2025 at 11:59 PM

Welcome to CS1810! The purpose of this assignment is to help assess your readiness for this course. It will be graded for completeness and effort. **Areas of this assignment that are difficult are an indication of areas in which *you* need to self-study. If you find you are struggling with many of these questions, it might be prudent to postpone taking this course until after you have mastered the necessary prerequisites. *During the term, the staff will be prioritizing support for new material taught in CS1810 over teaching prerequisites.* If you are unsure about your readiness, please contact the head TFs for advice.**

1. Please type your solutions after the corresponding problems using this LaTeX template, and start each problem on a new page.
2. Please submit the **writeup PDF to the Gradescope assignment 'HW0'**. Remember to assign pages for each question.
3. Please submit your **LaTeX file and code files (i.e., anything ending in `.py`, `.ipynb`, or `.tex`) to the Gradescope assignment 'HW0 - Supplemental'**.

Problem 1 (Modeling Linear Trends - Linear Algebra Review)

In this class, we will be exploring the question of "how do we model the trend in a dataset" under different guises. In this problem, we will explore the algebra of modeling a linear trend in data. We call the process of finding a model that capture the trend in the data, "fitting the model."

Learning Goals: In this problem, you will practice translating machine learning goals ("modeling trends in data") into mathematical formalism using linear algebra. You will explore how the right mathematical formalization can help us express our modeling ideas unambiguously and provide ways for us to analyze different pathways to meeting our machine learning goals.

Let's consider a dataset consisting of two points $\mathcal{D} = \{(x_1, y_1), (x_2, y_2)\}$, where x_n, y_n are scalars for $n = 1, 2$. Recall that the equation of a line in 2-dimensions can be written: $y = w_0 + w_1 x$.

1. Write a system of linear equations determining the coefficients w_0, w_1 of the line passing through the points in our dataset $\mathcal{D}$ and analytically solve for w_0, w_1 by solving this system of linear equations (i.e., using substitution). Please show your work.

2. Write the above system of linear equations in matrix notation, so that you have a matrix equation of the form $\mathbf{y} = \mathbf{X}\mathbf{w}$, where $\mathbf{y}, \mathbf{w} \in \mathbb{R}^2$ and $\mathbf{X} \in \mathbb{R}^{2\times 2}$. For full credit, it suffices to write out what $\mathbf{X}$, $\mathbf{y}$, and $\mathbf{w}$ should look like in terms of x_1, x_2, y_1, y_2, w_0, w_1, and any other necessary constants. Please show your reasoning and supporting intermediate steps.

3. Using properties of matrices, characterize exactly when an unique solution for $\mathbf{w} = (w_0\ w_1)^T$ exists. In other words, what must be true about your dataset in order for there to be a unique solution for $\mathbf{w}$? When the solution for $\mathbf{w}$ exists (and is unique), write out, as a matrix expression, its analytical form (i.e., write $\mathbf{w}$ in terms of $\mathbf{X}$ and $\mathbf{y}$).

   Hint: What special property must our $\mathbf{X}$ matrix possess? What must be true about our data points in $\mathcal{D}$ for this special property to hold?

4. Compute $\mathbf{w}$ by hand via your matrix expression in (3) and compare it with your solution in (1). Do your final answers match? What is one advantage for phrasing the problem of fitting the model in terms of matrix notation?

5. In real-life, we often work with datasets that consist of hundreds, if not millions, of points. In such cases, does our analytical expression for $\mathbf{w}$ that we derived in (3) apply immediately to the case when $\mathcal{D}$ consists of more than two points? Why or why not?

Solution

Your solution here.
Let's consider a dataset consisting of two points $\mathcal{D} = \{(x_1, y_1), (x_2, y_2)\}$, where x_n, y_n are scalars for $n = 1, 2$. Recall that the equation of a line in 2-dimensions can be written: $y = w_0 + w_1 x$.

1. The system of linear equations uses the base equation: $y = w_0 + w_1 x$. Using that, we can substitute our known values of our two data points for y and x and then we will have 2 (potentially unique) equations that can be solved for w_0, w_1.
   Equation 1: $y_1 = w_0 + w_1 x_1$
   Equation 2: $y_2 = w_0 + w_1 x_2$
   This is the system of linear equations which can then be solved using substitution.
   Step 1: Solve for w_0 in Equation 1
   $w_0 = y_1 - w_1 x_1$
   Step 2: Substitute the above equation for w_0 in Equation 2
   $y_2 = (y_1 - w_1 x_1) + w_1 x_2$
   Step 3: Solve for w_1
   $y_2 = y_1 + w_1(x_2 - x_1) \Rightarrow$
   $w_1 = \frac{y_2 - y_1}{x_2 - x_1}$
   Step 4: Now we have solved for w_1. Use that in the equation from Step 1 to solve for w_0
   $w_0 = y_1 - \frac{y_2 - y_1}{x_2 - x_1} x_1$

   That then produces our analytical solutions for w_0 and w_1. Which are:
   $w_1 = \frac{y_2 - y_1}{x_2 - x_1}$
   $w_0 = y_1 - \frac{y_2 - y_1}{x_2 - x_1} x_1$

2. Our system of linear equations, as established above, is Equation 1: $y_1 = w_0 + w_1 x_1$ and Equation 2: $y_2 = w_0 + w_1 x_2$. To turn this into matrix notation is easy
   $\mathbf{y} = (y_1, y_2)^T$
   $\mathbf{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix}$
   $\mathbf{w} = (w_0, w_1)^T$

   When you do $\mathbf{Xw} = \mathbf{y}$ you get $\begin{bmatrix} w_0 + x_1 w_1 \\ w_0 + x_2 w_1 \end{bmatrix} = (y_1, y_2)^T$ which is our system of linear equations.

3. For there to be a unique solution for $\mathbf{w}$, the system of equations must be independent from each other, meaning that there is no linear combination of one of the equations that equals the other. This happens if $x_1 \neq x_2$. Should they be equal then either there is no solution if $y_1 \neq y_2$ or there is infinite solutions if $y_1 = y_2$, because they would then represent the same line. When the solution exists then you can solve for $\mathbf{w}$ from $\mathbf{Xw} = \mathbf{y}$. So when try to find the $\mathbf{w}$ minimizes loss, you get $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. The solution then is only possible when $\mathbf{X}^T\mathbf{X}$ is invertible such that its product with its inverse is the identity matrix leaving just $\mathbf{w}$ on that side of the equation. Furthermore, a matrix is only invertible if its determinant is non-zero. We will see how this relates to our stipulation that for a unique solution $x_1 \neq x_2$ below when we find the determinant. So lets solve this for $\mathbf{w}$.
   $\mathbf{X}^T\mathbf{X} = \begin{bmatrix} 1 & 1 \\ x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} = \begin{bmatrix} 2 & x_1 + x_2 \\ x_1 + x_2 & x_1^2 + x_2^2 \end{bmatrix}$ Then we know that the inverse of that matrix is then $\frac{1}{2x_1^2 + 2x_2^2 - (x_1^2 + 2x_1 x_2 + x + x_2^2)} \begin{bmatrix} x_1^2 + x_2^2 & -x_1 - x_2 \\ -x_1 - x_2 & 2 \end{bmatrix}$
   The determinant then simplifies to $\frac{1}{x_1^2 - 2x_1 x_2 + x_2^2} = \frac{1}{(x_1 - x_2)^2}$
   Note importantly how for a matrix to be invertible, its determinant has to be non-zero, which only occurs here if $x_1 \neq x_2$, which is what we recognized above.

So $(\mathbf{X}^T\mathbf{X})^{-1} = \frac{1}{(x_1-x_2)^2}\begin{bmatrix} x_1^2+x_2^2 & -x_1-x_2 \\ -x_1-x_2 & 2 \end{bmatrix}$

Next step is to then find $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T = \frac{1}{(x_1-x_2)^2}\begin{bmatrix} x_1^2+x_2^2 & -x_1-x_2 \\ -x_1-x_2 & 2 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ x_1 & x_2 \end{bmatrix} = \frac{1}{(x_1-x_2)^2}\begin{bmatrix} x_2^2-x_1x_2 & x_1^2-x_1x_2 \\ x_1-x_2 & -x_1+x_2 \end{bmatrix}$

We would then multiply this by $\mathbf{y}$ to produce have final solution:

$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \frac{1}{(x_1-x_2)^2}\begin{bmatrix} x_2^2-x_1x_2 & x_1^2-x_1x_2 \\ x_1-x_2 & -x_1+x_2 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \frac{1}{(x_1-x_2)^2}\begin{bmatrix} y_1(x_2^2-x_1x_2)+y_2(x_1^2-x_1x_2) \\ y_1(x_1-x_2)+y_2(-x_1+x_2) \end{bmatrix}$

Thus the condition for a unique solution is that $x_1 \neq x_2$ allowing $\mathbf{X}^T\mathbf{X}$ to be invertible, and thus $\mathbf{w}$ to be calculated.

4. Let's simplify what we had above for $\mathbf{w}$:

$$\mathbf{w} = \frac{1}{(x_1-x_2)^2}\begin{bmatrix} y_1(x_2^2-x_1x_2)+y_2(x_1^2-x_1x_2) \\ y_1(x_1-x_2)+y_2(-x_1+x_2) \end{bmatrix}$$

$$w_0 = \frac{y_1x_2^2 - y_1x_1x_2 + y_2x_1^2 - y_2x_1x_2}{(x_1-x_2)^2} = \frac{y_1x_2^2+y_2x_1^2-(y_1+y_2)x_1x_2}{(x_1-x_2)^2}$$

We also know that $(x_1-x_2)^2 = (x_2-x_1)^2$, so lets do that then factor the numerator according to to (x_2-x_1). You get $\frac{(y_1x_2-y_2x_1)(x_2-x_1)}{(x_2-x_1)^2} \Rightarrow \frac{(y_1x_2-y_2x_1)}{(x_2-x_1)}$

Lets compare it to what we got in 1:

$$y_1 - \frac{y_2-y_1}{x_2-x_1}x_1 \Rightarrow \frac{y_1(x_2-x_1)+y_2x_1-y_1x_1}{x_2-x_1} \Rightarrow \frac{y_1x_2-y_1x_1-y_2x_1+y_1x_1}{x_2-x_1} \Rightarrow \frac{y_1x_2-y_2x_1}{x_2-x_1}$$

We can then see that we simplified both what we found in part 1, and part 4 for w_0 to be the equal.

Lets do the same for $w_1 = \frac{y_1(x_1-x_2)+y_2(-x_1+x_2)}{(x_1-x_2)^2} = \frac{y_1x_1-y_1x_2-y_2x_1+y_2x_2}{(x_1-x_2)^2}$. Lets make the same switch we made above here in the denominator. $\frac{y_1x_1-y_1x_2-y_2x_1+y_2x_2}{(x_2-x_1)^2}$ and factor out (x_2-x_1) in numerator you get $\frac{(y_2-y_1)(x_2-x_1)}{(x_2-x_1)^2} \Rightarrow \frac{y_2-y_1}{x_2-x_1}$.

Now we'll compare to what we got for w_1 in part 1 which was $\frac{y_2-y_1}{x_2-x_1}$, and we can see that these are equal.

Our answers clearly match! Although this is more strenuous calculation(at least it seems), when you are doing it by hand, for more complex problems where a computer is necessary for the calculations, the matrix form is a much more digestible method. This is one advantage, because matrix organization/phrasing generalizes well into higher dimensions where long form equations might look complicated. It also puts it into notation that the computer can take as an input and do matrix operations quickly.

5. Yes, our solution derived in 3 can work with much larger datasets. I did more calculations in 3 than necessary (they became necessary for 4) but the analytical expression for $\mathbf{w}$ where $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ is generalizable for more data points because in those cases X will be a much larger matrix incorporating the added data. This works because as we did in 2 we would organize the set of linear equations that incorporate the data points from $\mathcal{D}$ in matrix form, and simplifying for $\mathbf{w}$ is what allowed us to find the unique solution (assuming one exists). Thus in large datasets $\mathbf{X}$ may have many rows, making it computationally complex, but the analytical solution still functions.

Problem 2 (Optimizing Objectives - Calculus Review)

In this class, we will write real-life goals we want our model to achieve into a mathematical expression and then find the optimal settings of the model that achieves these goals. The formal framework we will employ is that of mathematical optimization. Although the mathematics of optimization can be quite complex and deep, we have all encountered basic optimization problems in our first calculus class!

Learning Goals: In this problem, we will explore how to formalize real-life goals as mathematical optimization problems. We will also investigate under what conditions these optimization problems have solutions.

In her most recent work-from-home shopping spree, Nari decided to buy several house plants. *Her goal is to make them to grow as tall as possible.* After perusing the internet, Nari learns that the height y in mm of her Weeping Fig plant can be directly modeled as a function of the oz of water x she gives it each week:

$$y = -3x^2 + 72x + 70.$$

1. First, plot the height function. What does the plot tell you about the existence and uniqueness of a maximum plant height? Next, support your claim solely based on the form of the function.

2. Use calculus to find how many ounces of water per week Nari should give to her plant in order to maximize its height. With this much water, how tall will her plant grow?

Now suppose that Nari want to optimize both the amount of water x_1 (in oz) *and* the amount of direct sunlight x_2 (in hours) to provide for her plants. After extensive research, she decided that the height y (in mm) of her plants can be modeled as a two variable function:

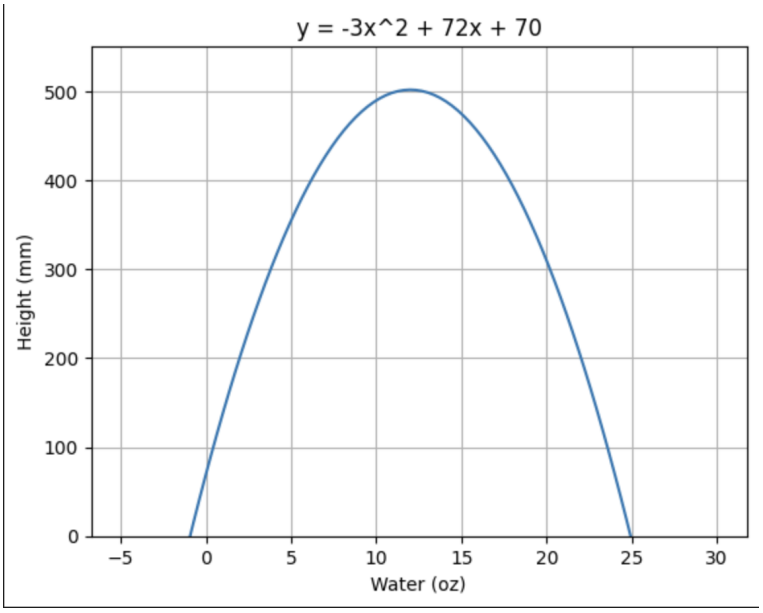
$$y = f(x_1, x_2) = \exp\left(-(x_1 - 2)^2 - (x_2 - 1)^2\right)$$

3. Using `matplotlib`, visualize in 3D the height function as a function of x_1 and x_2 using the `plot_surface` utility for $(x_1, x_2) \in (0, 6) \times (0, 6)$. Then, determine the values of x_1 and x_2 that maximize plant height. Do these yield a global maximum?

   Hint: You don't need to take any derivatives here; reasoning about the form of $f(x_1, x_2)$ suffices.

Solution

1. The plot tells us that there is a maximum plant height, with a specific amount of water being the cause of that maximum. We can see the proof of this on the basis of the graph: as we can see, it is an inverted parabola shape with a maximum height occurring around 12 ounces of water, and creating a plant height of about 500 mm. Furthermore, looking at the function itself $y = -3x^2 + 72x + 70$ we



can see similar information. This is because it is a second degree polynomial (i.e. parabola) with a negative coefficient in front of the x^2 term (i.e. inverted) suggesting there will be a maximum. If we wanted to calculate the maximum we could do further calculations, but the above information already proves there is a maximum for the function.

2. To find the maximum we will set the derivative equal to 0, because at the maximum the slope will be 0. We also do not need to worry about there being global versus local maximums because it is a second degree polynomial.

$$y' = -6x + 72 = 0$$

$$-72 = -6x$$

$$x = 12$$

The plant will grow to 502 mm if Nari provides 12 ozs of water per week.

$$y = -3(12)^2 + 72(12) + 70$$

$$y = 502$$

This means that the maximum height for the plant will occur when Nari gives 12 ozs of water per week.

3. Yes for the function $f(x_1, x_2)$ it will yield a global maximum. First, we can see there is a maximum on the plot. We know that this is a global maximum, not a local one, because the form of $y = f(x_1, x_2) = \exp(-(x_1 - 2)^2 - (x_2 - 1)^2)$, which is really a set of functions of the form $\frac{1}{\exp(x^2)}$, which have global maximums. This due to exponent rules that $a^{x-y} = \frac{a^x}{a^y}$ and in this case $a^{-x-y} = \frac{1}{a^x a^y}$. As such, in each case holding either x_1 or x_2 constant in locations simply creates a singular maximum. Thus when put together in a two variable function it will simply have one maximum (i.e. a global maximum).

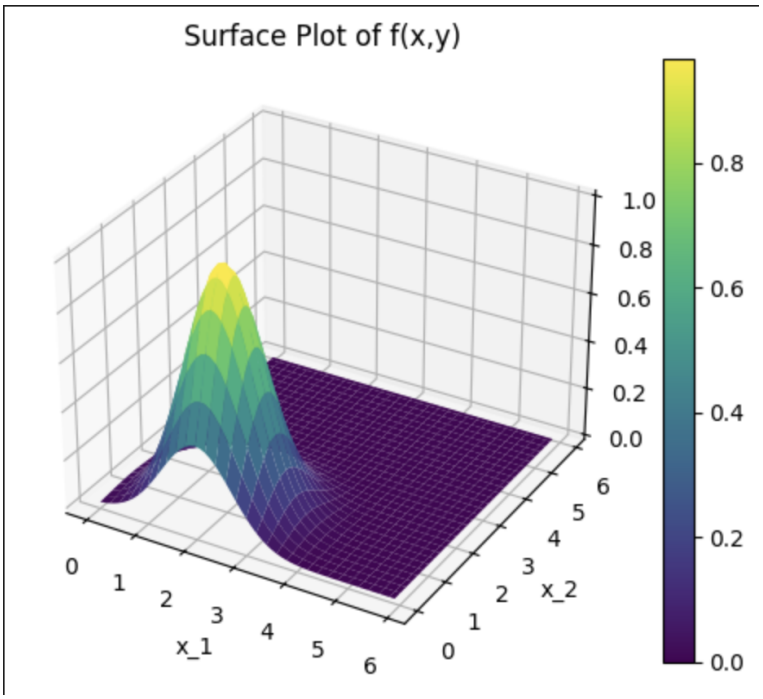


Figure 1: Surface Plot of f(x,y)

Note that ChatGPT was used here to help create the code for plotting surface plot

Problem 3 (Reasoning about Randomness - Probability and Statistics Review)

In this class, one of our main focuses is to model the unexpected variations in real-life phenomena using the formalism of random variables. In this problem, we will use random variables to model how much time it takes an USPS package processing system to process packages that arrive in a day.

Learning Goals: In this problem, you will analyze random variables and their distributions both analytically and computationally. You will also practice drawing connections between said analytical and computational conclusions.

Consider the following model for each package that arrives at the US Postal Service (USPS):

- Every package has a random size S (measured in in^3) and weight W (measured in pounds), with joint distribution

$$(S, W)^T \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \text{ with } \boldsymbol{\mu} = \begin{bmatrix} 120 \\ 4 \end{bmatrix} \text{ and } \boldsymbol{\Sigma} = \begin{bmatrix} 1.5 & 1 \\ 1 & 1.5 \end{bmatrix}.$$

- The size and weight of each package is independent of those of all the other packages.
- Processing time T (in seconds) for each package is given by $T = 60 + 0.6W + 0.2S + \epsilon$, where ϵ is an independent random noise variable with Gaussian distribution $\epsilon \sim \mathcal{N}(0, 5)$.

1. Perform the following tasks:
   (a) Give one reason for why the Gaussian distribution may not be appropriate for modeling the size and weight of packages.
   (b) Empirically estimate the most likely combination of size and weight of a package by sampling 500 times from the joint distribution of S and W and generating a bivariate histogram of your S and W samples. A visual inspection is sufficient – you do not need to be incredibly precise. How close are these empirical values to the theoretical expected size and expected weight of a package, according to the given Bivariate Gaussian distribution?
   Hint: For this part, you may find the `multivariate_normal` module from `scipy.stats` especially helpful. You may also find the `seaborn.histplot` function quite helpful.
2. For 1001 evenly-spaced values of W between 0 and 10, plot W versus the joint Bivariate Gaussian PDF $p(W, S)$ with S fixed at $S = 118$. Repeat this procedure for S fixed at $S = 122$. Comparing these two PDF plots, what can you say about the correlation of random variables S and W?
3. Because T is a linear combination of random variables, it itself is a random variable. Using properties of expectations and variance, please compute $\mathbb{E}(T)$ and $\text{Var}(T)$ analytically.
4. Define N to be the number of packages that arrive today, and suppose that packages that weigh less than 4 pounds are considered fragile. Conditional on $N = n$, what is the name and PMF of the distribution of the number of fragile packages that arrive today?
5. Now suppose that $N = \sum_{h=1}^{24} P_h$, where the P_h are independent and identically distributed as $\text{Pois}(\lambda = 3)$. Then define $T^* = \sum_{i=1}^{N} T_i$ as the *total* amount of time it takes to process *all* these packages, where T_i follows the distribution of T that we previously defined for each package.
   (a) Write a function to simulate draws from the distribution of T^*.
   (b) Using your function, empirically estimate the mean and standard deviation of T^* by generating 1000 samples from the distribution of T^*.

Solution

1. (a) The Gaussian distribution may not be appropriate for modeling this situation, because it is possible for the size or weight to be negative with this distribution. A negative weight or size does not make intuitive sense. Although, the size is very unlikely to be negative given that the mean is 120, and the variance is only 1.5, it is still possible. Furthermore, the weight R.V. also has a probability of producing negative values.

   (b) The most likely combination of size and weight of a package is quite close to the expected size and weights, 120 in^3 and 4 lbs respectively. Although in the bivariate histogram it is hard to tell which the specific values would be, you can see the darker colors (where the PDF is larger) is generally around their means. I also plotted the individual histograms, and we can see how close they are from their means. Importantly, this is not the exact same sampling as I resampled, however it is illustrative of the general distribution.

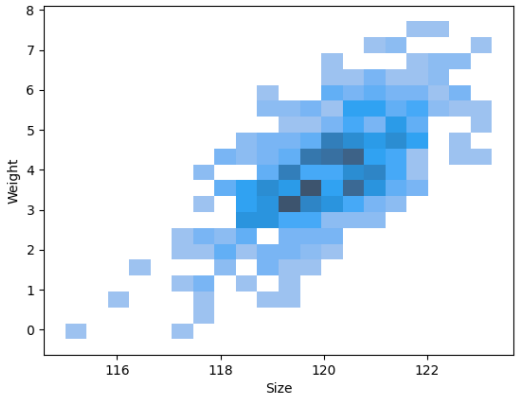
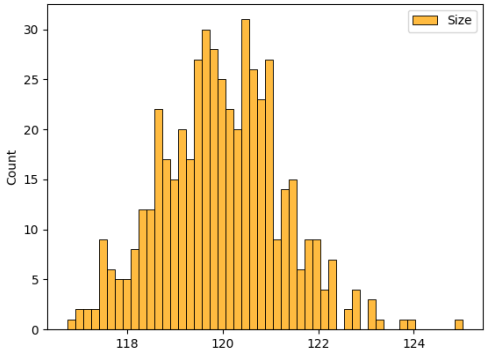
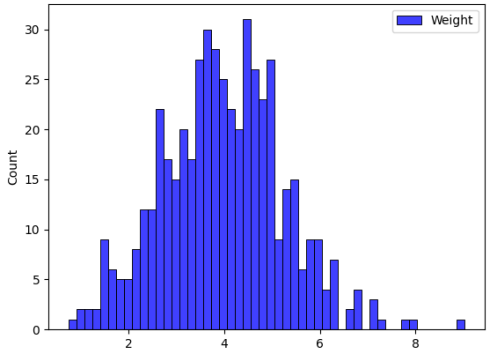


Figure 2: Bivariate Histogram



(a) Size Histogram



(b) Weight Histogram

2. Based on the distributions that we see below, we can tell that the correlation of R.Vs S and W is positive. That means when S increases, so does W.

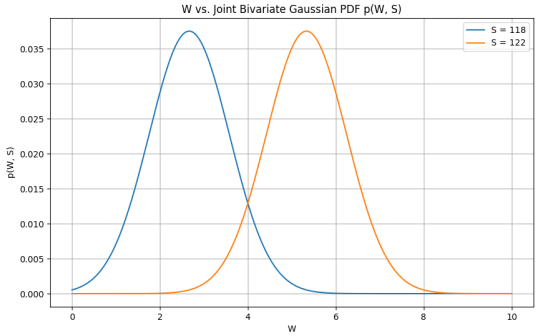


Figure 4: Joint PDF $p(W, S)$ with fixed S

Note that ChatGPT was used here to help create the code for plotting the Joint PDFs

3. Since T is a linear combination of R.Vs then we can calculate $\mathbb{E}(T)$ and $\text{Var}(T)$ easily. $\mathbb{E}(T)$ is found using the linear combination as well as $\mathbb{E}(S)$ and $\mathbb{E}(W)$.

$$T = 60 + 0.6W + 0.2S + \epsilon$$

$$\mathbb{E}(T) = 60 + 0.6\mathbb{E}(W) + 0.2\mathbb{E}(S) + \mathbb{E}(\epsilon)$$

We know that $\mathbb{E}(W) = 4$, $\mathbb{E}(S) = 120$, and $\mathbb{E}(\epsilon) = 0$ So we can plug in and calculate.

$$\mathbb{E}(T) = 60 + 0.6(4) + 0.2(120) + 0 = 86.4$$

To calculate $\text{Var}(T)$ you rely on that $\text{Var}(T) = \text{Var}(0.6W + 0.2S + \epsilon)$ (variance of a constant is 0).

$$\text{Var}(0.6W + 0.2S + \epsilon) = 0.6^2\text{Var}(W) + 0.2^2\text{Var}(S) + \text{Var}(\epsilon) + 2(0.6)(0.2)\text{Cov}(W, S)$$

Note that there are no covariance terms between S, W and ϵ because since ϵ is simply a noise term there is no variance between the two.

$$\text{Var}(T) = 0.36(1.5) + 0.04(1.5) + 25 + 0.24(1) = 25.84$$

$$\text{Var}(T) = 25.84$$

4. Lets say that X is the R.V. that counts the number of packages that are fragile. This is clearly a binomial distribution, because there are two outcomes for every package n_i in $N = n$, either it is fragile, or it is not fragile. Thus $X|N = n \sim \text{Binomial}(n, p)$. However we know $p = P(W < 4)$ for any given package. Since $\mathbb{E}(W) = 4$ then we know that $p = P(W < 4) = 0.5$, thus

$$X|N = n \sim \text{Binomial}(n, 0.5)$$

or

$$P(X = k|N = n) = \binom{n}{k}(0.5)^k(0.5)^{n-k}$$

5. (a)

```
def simulate_T_star(num_simulations):
    lambda_N = 3

    T_star_samples = []

    for _ in range(num_simulations):
        N = np.random.poisson(lambda_N)

        # Calculate W, S, eps for each sample
        W = np.random.normal(4, np.sqrt(1.5), size=N)
        S = np.random.normal(120, np.sqrt(1.5), size=N)
        eps = np.random.normal(0, 5, size=N)

        # Calculate T for each sample
        T = 60 + 0.6 * W + 0.2 * S + eps

        # Sum to get T*
        T_star = np.sum(T) if N > 0 else 0
        T_star_samples.append(T_star)

    return np.array(T_star_samples)
```

(b) The empirically estimated mean and standard deviations of T^* from 1000 samples are: Mean = 266.92 and Standard Deviation = 156.52

Problem 4 (Implementing a Linear Regression - Coding Review)

In this class, we will bridge theory and practice through implementing the methods that we cover from scratch. In this problem, we follow up on Problem 1 through exploring a more practical version of linear regression (fitting a linear model). Namely, we use ordinary least squares (OLS) to estimate a *line of best fit* rather than a perfect fit to our data. Note that the focus of this problem is on coding rather than math—we will cover the relevant theory in much more depth during the course.

Learning Goals: In this problem, you will gain experience with the procedure of modeling real-world data. You will also get useful practice with debugging and writing clean, efficient code in Python.

Steve is a fictional CS 1810 TF giving a live demo of how to fit a linear regression. However, he quickly realizes that coding live in front of an audience isn't for the faint of heart. As a star student, you will help him with his code. Just like Problem 1, the demo uses a 2-D dataset, so that the goal is to model the relationship between the x and y coordinates. The data are stored in the `data` variable, with the first column corresponding to the x-coordinate and the second corresponding to the y-coordinate.

1. Using the provided data, Steve has defined variables `y` and `x` corresponding to the respective coordinates. What is wrong with his current code? Fix the code and then plot the data. Does there appear to be a linear trend?

2. Steve then defines a new variable `X`, which is meant to resemble $\boldsymbol{X}$ from Problem 1. Specifically, `X` is supposed to have one column of all ones (recall that this allows us to fit an intercept) and one column which is just `x`, the x-coordinates. However, he realizes that his code yields the wrong shape for `X`. What's going on here? Fix the code and then report what `y.shape` and `X.shape` are. Why is there no second coordinate in the output for `y.shape`?

   Hint: check the documentation for `np.hstack`.

3. Steve takes a much-needed break from coding to give the following high level overview of linear regression: given a target (response) $\boldsymbol{y}$ and features (predictors) $\boldsymbol{X}$, the goal of linear regression is to find weights $\boldsymbol{w}$ such that $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}$ closely approximates the true data $\boldsymbol{y}$. In OLS, we estimate $\boldsymbol{w}$ to be

   $$\hat{\boldsymbol{w}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

   Steve skips over the derivation of the result but assures you that you will learn it later in the course. What should the shape of $\hat{\boldsymbol{w}}$ be in Steve's demo?

4. Having walked through the idea of linear regression, Steve then attempts to implement a

   `LinearRegression` class. He correctly identifies that we need 3 components: a constructor, a `fit` function for computing $\hat{\boldsymbol{w}}$ from the data, and a `predict` function for computing the estimate $\boldsymbol{X}\hat{\boldsymbol{w}}$. However, he realizes that there is something wrong (meaning logic or syntax) with at least one of these components. Please point out the issues, fix them, and include the plot of the fitted line.

5. As his final act for the day, Steve introduces the Mean Squared Error (MSE) loss function:

   $$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

   This captures how well the outputs of our model, $\hat{\boldsymbol{y}}$, fit the actual data $\boldsymbol{y}$. Steve manages to correctly implement an MSE computation! However, you realize that he can vectorize his code to make it faster, meaning that he can directly compute the MSE from NumPy arrays without using any for loops. Implement the vectorized MSE and write down the corresponding mathematical expression, which should directly be in terms of the vectors $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ rather than their components.

Solution

1. Steve's code was wrong because it essentially pulled the first and second (x, y) pair in the dataset, as opposed to pulling all the x values and all the y values. The change was

```
x = data[:,0]
y = data[:,1]
```

   The resulting plot, does show a pretty clear linear trend.

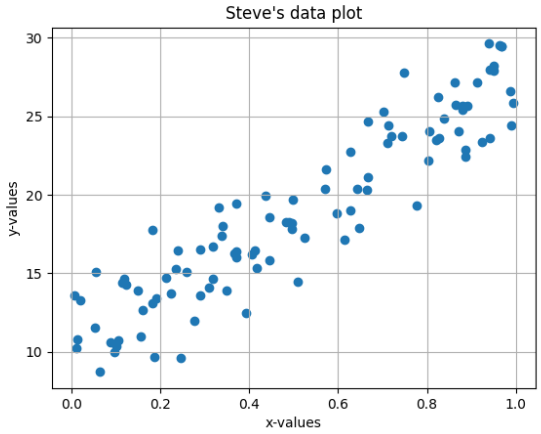


Figure 5: Steve's Data plot

2. Steve's attempt at adding a column of all ones, does not create the correct shape for `X`, because it just puts a set of ones before the x values, (i.e. $[1, 1...1, x_0, ...x_n]$). To fix you simple needed to adjust the shape of input array of ones to be vertical, and also adjust the shape of x.

```
intercept = np.ones((x.shape[0],1))

x_reshaped = x.reshape(-1, 1)
X = np.hstack([intercept, x_reshaped])
```

   `X.shape` is $(100, 2)$, and `y.shape` is $(100,)$. This simply means that y is a vector as opposed to a matrix. This is fine, as the matrix multiplication will be able to handle that.

3. The shape of $\hat{\boldsymbol{w}}$ should be $(2,)$ You can see this 2 ways: Since we want a linear combination it will be of form $y = ax + b$ thus there will be 2 coefficients in w, (a, b) making it of shape $(2,)$. Second reasoning is that matrix multiplication of matrices of shape $(2, 100) * (100, 2)$ results in matrix shape $(2, 2)$. That is $\boldsymbol{X}^\top \boldsymbol{X}$, and the inverse doesn't change its shape. Then $\boldsymbol{X}^\top \boldsymbol{y}$ is $(2, 100)$ and $(2,)$ so results in $(2,)$. Finally put those together you have shaped matrices of $(2, 2)$ and $(2,)$ which results in $(2,)$, a 2-dimensional vector (i.e. the shape of $\hat{\boldsymbol{w}}$).

4. The issues that I fixed in `fit` were:
   Before it simply returned a value, but below in the implementation of the class, it simply expected `lr.fit` to store a value automatically, so I changed it to store as `self.w`, so that it could be called upon in predict. Second, the original implementation used $*$ for matrix multiplication, and $**-1$ for

inverse. However, this would simply just do positional-respective operations to the matrices. This is not what we want, so instead we use the @ symbol. Finally, Steve forgot to make a $\boldsymbol{X}$ transpose. The result was:

```
def fit(self, X, y):
    self.w = np.linalg.inv(X.T @ X) @ X.T @ y.T
```

In the `predict` function, I had to ensure that the weights stored above in `fit` were called upon correctly, so added `self.w` as the refrerence to the weights. I also added a edge case where if `fit` was not called, it would recognize there were no weights set, and thus output 'Model has not been fitted yet.' The result was:

```
def predict(self, X):
    if self.w is None:
        raise ValueError("Model has not been fitted yet.")
return X @ self.w
```

The resulting fitted line looks accurate too the data.

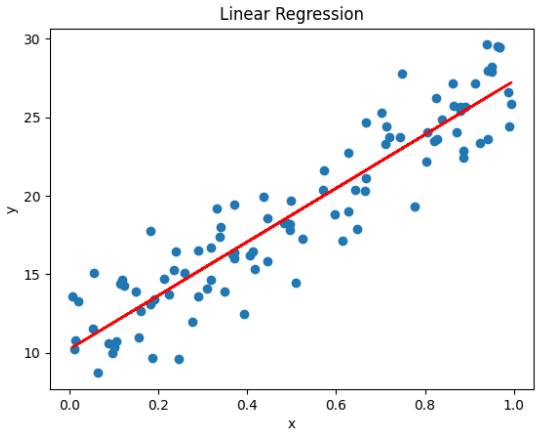


Figure 6: Data plot with Fitted line

5. The implementation of the vectorized MSE is easy, as it simply relies calculating the squared error for each vector component (can be done as whole vector), then summing the resulting array, and dividing by number of data points.

```
mse = np.sum((y-y_pred)**2)/len(y)
```

The resulting MSE is 4.439.