Homework #0

Due: February 2, 2026 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?

6. Using Python, construct matrix $\mathbf{X}$ and vector $\mathbf{y}$ corresponding to a dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2)\}$. Compute $\mathbf{w}$ using numerical values of your choice with $x_1 \neq x_2$. Your code should reflect the matrix formulation derived above.

Solution

Problem 1:
1. If a line passes through both of these points, they are both solutions to the form $y = w_0 + w_1 x$. In order to write a system of linear equations, we can plug in the two points into the same template for the equation of a line:

$$\begin{aligned} y_1 &= w_0 + w_1 x_1 \\ y_2 &= w_0 + w_1 x_2 \end{aligned}$$

We can solve for w_0 in the first equation and substitute it into the second equation:

$$\begin{aligned} w_0 &= y_1 - w_1 x_1 \\ y_2 &= (y_1 - w_1 x_1) + w_1 x_2 \\ y_2 - y_1 &= w_1 (x_2 - x_1) \\ w_1 &= \frac{y_2 - y_1}{x_2 - x_1} \end{aligned}$$

We can then substitute w_1 back into the equation for w_0:

$$\begin{aligned} w_0 &= y_1 - \frac{y_2 - y_1}{x_2 - x_1} x_1 \\ &= \frac{y_1 (x_2 - x_1) - (y_2 - y_1) x_1}{x_2 - x_1} \\ &= \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1} \end{aligned}$$

Looking at the above, this makes sense because the equation for w_1 ended up being the slope by looking at the rate of change, and the equation for w_0 is the intercept of the line passing through the two points after taking away the impact of the slope. The two solved forms for w_0 and w_1 are:

$$\begin{aligned} w_0 &= \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1} \\ w_1 &= \frac{y_2 - y_1}{x_2 - x_1} \end{aligned}$$

2. We can rewrite the two equations from part (1) in matrix form as follows:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

Each individual equation when multiplied out here corresponds to the equations we had:

$$y_1 = 1 \cdot w_0 + x_1 \cdot w_1 \qquad y_2 = 1 \cdot w_0 + x_2 \cdot w_1$$

Thus, we have $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$, $\mathbf{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix}$, and $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$ to yield the same results.

Solution

Problem 1 (Continued):

3. In an algebaic sense, a unique solution for $\mathbf{w}$ exists if for every input, there is exactly one unique output for that input. This makes the transformation from input to output one-to-one. In the matrices world, one way that guarantees this is if the matrix $\mathbf{X}$ is invertible. Converting from the original equation $\mathbf{y} = \mathbf{X}\mathbf{w}$, we can manipulate under the assumption of invertibility to get:

$$\mathbf{y} = \mathbf{X}\mathbf{w}$$

$$\mathbf{X}^{-1}\mathbf{y} = \mathbf{X}^{-1}\mathbf{X}\mathbf{w}$$

$$\mathbf{X}^{-1}\mathbf{y} = \mathbf{I}\mathbf{w}$$

$$\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$$

It is important to note that for invertibility to happen, the data space must only yield 0 from the equation $\mathbf{X}\mathbf{v} = 0$ when $\mathbf{v} = 0$. In other words, the null space of $\mathbf{X}$ must only contain the zero vector. In our case, this means that $x_1 \neq x_2$, otherwise the two rows of $\mathbf{X}$ would be linearly dependent and thus not invertible. Thus, when $x_1 \neq x_2$ (so that we have a basis of the data space), we have a unique solution for $\mathbf{w}$ given by:

$$\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$$

4. Continuing from part (3), we can compute $\mathbf{X}^{-1}$ using the determinant and simplifying:

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

Thus, we can compute $\mathbf{w}$:

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

This yields:

$$w_0 = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}, \quad w_1 = \frac{y_2 - y_1}{x_2 - x_1}$$

This matches exactly with our results from part 1. One advantage of phrasing the problem in terms of matrix notation is that it is easier to group both the relationship between w_0 and w_1 with the solution together. If we were to scale this to more dimensions, there's a set process on how to get there as opposed to having to derive the equations from scratch case by case.

5. The analytical expression for $\mathbf{w}$ derived in part 3 does not apply immediately to the case when $\mathcal{D}$ consists of more than two points. This is because when there are more than two points, the system of equations becomes overdetermined (more equations than unknowns). In such a case, there may not be a line that passes through all the points exactly. Instead, we would typically look for a line that minimizes the error (maybe using least squares) rather than one that fits all points perfectly. We basically run into an overfitting problem and more data does not necessarily mean better fitting data.

6. See the code displayed in the attached Jupyter Notebook file. The matrices used in there do reflect the matrix formulation derived above. The arbitrary numerical values chosen were $x_1 = 2$, $y_1 = 5$, $x_2 = 4$, and $y_2 = 6$. $\mathbf{w}$ ends up being $\begin{bmatrix} 4 \\ 0.5 \end{bmatrix}$ for these values.

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

Problem 2:
1. The plot of the height function is a downward-opening parabola, which indicates that there is a unique maximum point (see the end of this PDF). This is because the coefficient of the x^2 term is negative (-3), which means the parabola opens downwards, leading to a single peak (maximum) point at its vertex.

2. To find the maximum height, we can find the x value at the vertex of the parabola using the formula $x = -\frac{b}{2a}$, where $a = -3$ and $b = 72$:

$$x = -\frac{72}{2 \cdot -3} = 12$$

Plugging this back into the height function to find the maximum height:

$$y = -3(12)^2 + 72(12) + 70 = -432 + 864 + 70 = 502$$

Thus, Nari should give her plant 12 oz of water per week to maximize its height, which will be 502 mm.

Alternatively, we can find the vertex also by looking at where the dertivative is zero:

$$\frac{dy}{dx} = -6x + 72$$

Setting the derivative to zero:

$$-6x + 72 = 0$$

$$x = 12$$

Plugging this back into the height function gives the same maximum height of 502 mm.

3. The 3D plot of the height function $f(x_1, x_2) = \exp(-(x_1 - 2)^2 - (x_2 - 1)^2)$ shows a peak at the point $(x_1, x_2) = (2, 1)$. See the plot at the end of this PDF.This is because the exponential function reaches its maximum value of 1 when the exponent is zero, which occurs when both $(x_1 - 2)^2$ and $(x_2 - 1)^2$ are zero. Therefore, the values of x_1 and x_2 that maximize plant height are $x_1 = 2$ oz of water and $x_2 = 1$ hour of sunlight. This point yields a global maximum since the exponential function is always positive and approaches zero as we move away from this point in any direction.

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

Problem 3:
1a. The Gaussian distribution may not be appropriate for size and weight is because they may also yield negative values, which is not possible in real life. Something else could be that the distribution may be skewed or discreet, which the Gaussian distribution does not capture with this continuous and symmetric noise.
1b. The empirical estimate of the most likely combination of size and weight of a package is around (120, 4) because that's the given μ of the distribution. The theoretical value also centers around (120, 4) as well, so they are very close. The highest density values in the histogram are centering around that point (see the end of this PDF for the plot).

2. The two PDF plots show that as S increases, the crest of the graph shifts to the right, which the distribution of W also follows S as it increases. This indicates that S and W are positively correlated, which makes sense because larger packages tend to be heavier. (see the end of this file for the plots).

3. We can compute $\mathbb{E}(T)$ and $\text{Var}(T)$ using the properties of expectation and variance:

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

Given that $\mathbb{E}(W) = 4$, $\mathbb{E}(S) = 120$, and $\mathbb{E}(\epsilon) = 0$, we have:

$$\mathbb{E}(T) = 60 + 0.6(4) + 0.2(120) + 0 = 60 + 2.4 + 24 = 86.4$$

$$\text{Var}(T) = \text{Var}(60 + 0.6W + 0.2S + \epsilon) = \text{Var}(0.6W) + \text{Var}(0.2S) + \text{Var}(\epsilon)$$

Since W and S are independent, we can compute their variances using the covariance matrix information and the variance of ϵ given above:

$$\text{Var}(W) = 1.5, \quad \text{Var}(S) = 1.5, \quad \text{Var}(\epsilon) = 5$$

Thus:

$$\text{Var}(T) = (0.6)^2(1.5) + (0.2)^2(1.5) + 5 = 0.54 + 0.06 + 5 = 5.6$$

4. Conditional on $N = n$, the number of fragile packages that come in today follows a **Binomial distribution** because they are independent trials that come in with two possible outcomes (fragile or not fragile) and a fixed number of trials. The PMF is given by:

$$P(X = k|N = n) = \binom{n}{k} p^k (1-p)^{n-k}$$

where p is the probability that a package is fragile.

5a. See the code displayed in the attached Jupyter Notebook file for the function to simulate draws from the distribution of T^*.
5b. Using the function, the empirical estimate of the mean of T^* is approximately 6234 seconds, and the standard deviation is approximately 706 seconds (see the supporting Jupyter Notebook for the code and results, and the end of this PDF for the plot).

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

Problem 4:
1. The original code read the data horizontally, which caused the variables to not be read correctly (single row instead of columns). The fixed code should read:

```
data = pd.read_csv("data/points.csv").values
x = data[:, 0]
y = data[:, 1]
```

There does seem to be a clear linear trend in the data (see the end of this PDF for the plot).

2. Steve's code for `X` used `np.hstack`, which stacks arrays in sequence horizontally (column-wise). However, since `x` is a 1D array, you need to reshape it correctly, otherwise the hstack will not work as intended and does some weird combination on one column that is not intented (something like (2n,)) instead of (n, 2). The fixed code should read:

```
intercept = np.ones((x.shape[0], 1))
x_col = x.reshape(-1, 1)
X = np.hstack([intercept, x_col])
```

After fixing the code, `y.shape` is (100,) and `X.shape` is (100, 2). There is no second coordinate in the output for `y.shape` because `y` is a 1D array representing the target values, while `X` is a 2D array representing the feature matrix with two columns (intercept and x-coordinates).

3. The shape of $\hat{\boldsymbol{w}}$ should be (2,) because there are two parameters to estimate: the intercept and the slope of the line. They can be treated as w_0 and w_1 in the vector $\boldsymbol{w}$.

4. The issues in Steve's code are several places of the `fit` and `predict` functions. Firstly in the `fit` function, the matrix multiplication is not done correctly. It should use the `@` operator for matrix multiplication. It should also assign the result to `self.w`. Secondly, in the `predict` function, the parameter and the return statement both missed the self reference. The fixed code should read:

```
class LinearRegression:
    def __init__(self):
        self.w = None

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

    def predict(self, X):
        return X @ self.w
```

The plot of the fitted line is at the end of this PDF.

Solution

Problem 4 (Continued):
5. The vectorized MSE can be computed using the following mathematical expression:

$$\text{MSE} = \frac{1}{n}(\boldsymbol{y} - \hat{\boldsymbol{y}})^\top (\boldsymbol{y} - \hat{\boldsymbol{y}})$$

The vectorized code for computing MSE should read:

```
def mean_squared_error(y, y_pred):
    n = y.shape[0]
    mse = np.sum((y - y_pred) ** 2) / n
    return mse
```

Alternatively even without vectors, it can also be computed as:

```
def mean_squared_error(y, y_pred):
    mse = np.mean((y - y_pred)**2)
    return mse
```

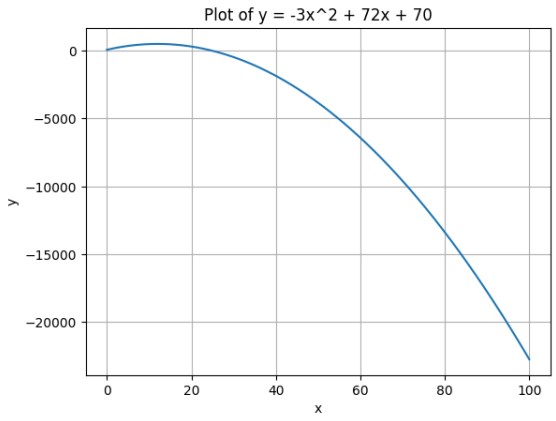

Figure 1: Plot for Problem 2, part 1

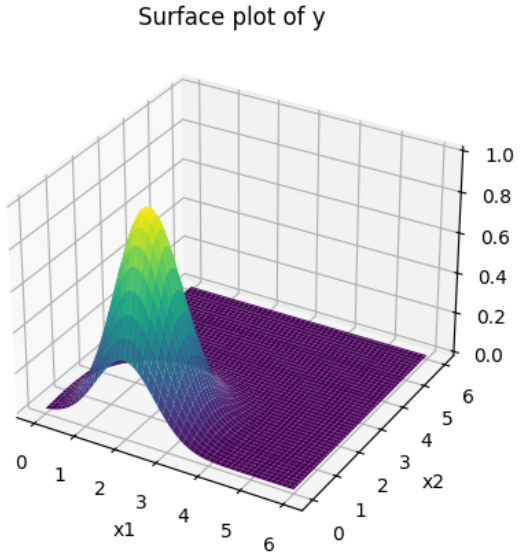

Figure 2: Histogram for Problem 2, part 3

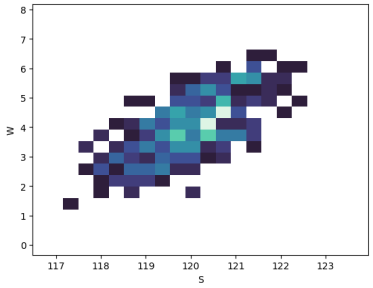

Figure 3: Histogram for Problem 3, part 1b

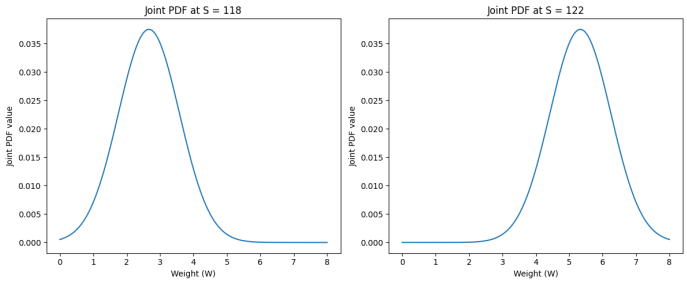

Figure 4: PDF plot for Problem 3, part 2

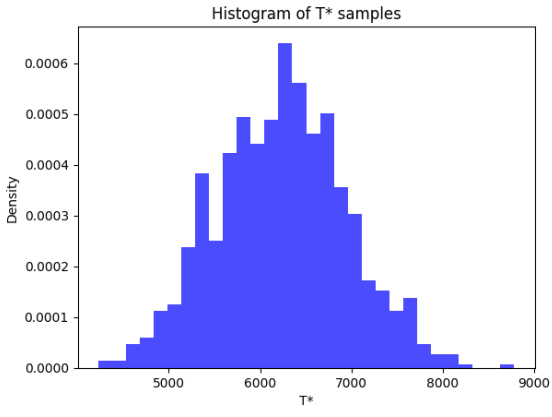

Figure 5: Histogram of T* for Problem 3, part 5b

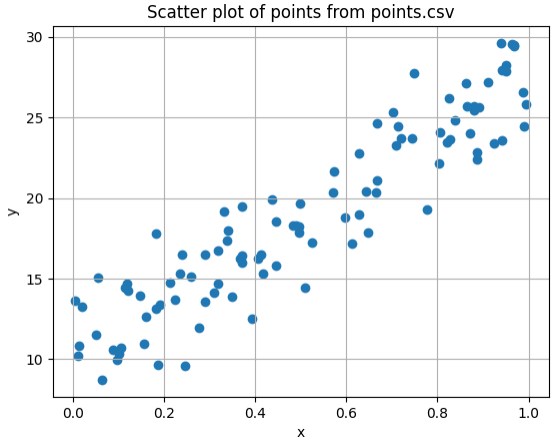

Figure 6: Plot of data points for Problem 4, part 1

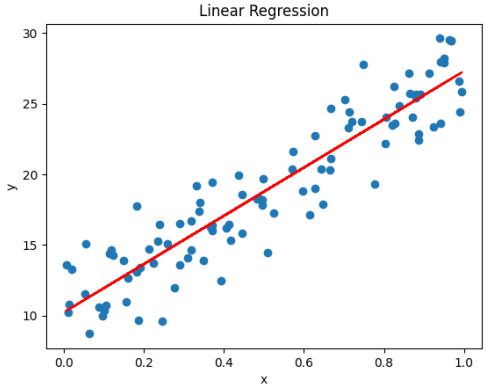

Figure 7: Plot of fitted line for Problem 4, part 4