ES.1802 Final Project Write-Up
3D Multiple Linear Regression With Gradient Descent

By: Ava Brule

1) Project Topic and Motivation

This project revolves around utilizing the gradient descent algorithm to fit a plane to a linearly-trending set of data points plotted in three-dimensional (3D) space. While selecting a project topic, I originally wanted to do a project on the variational quantum eigensolver algorithm, which is similar to gradient descent in the way that it is both iterative and seeks to minimize the value of a function. However, I realized that I did not yet have the mathematical/technical background in quantum circuits to complete a project on that topic in a reasonable time frame, making me shift to my current topic. I discovered gradient descent for linear regression while I was researching the variational quantum eigensolver algorithm, and I was interested in seeing if I could use it to fit a model in three dimensions, rather than the standard two-dimensional linear regression. This project topic also included a programming aspect, which was something that I was looking to have in my final project in general, as I enjoy learning more about programming.

2) Linear Regression & Multiple Linear Regression

2.1) Linear Regression Definition

Linear regression is the process of finding a model that can best describe a set of data points by minimizing the differences between the actual data point values and the predicted values the model suggests. The model that is generated is either a line or surface that exists in the same space that the data points do (i.e. if the points exist in 2D space, the model, which is a line, also exists in 2D space). **(1)**

There are different tools used for proceeding with linear regression; however, one of the most common tools (which is used in this project) is Mean Squared Error (MSE). The following formula is the MSE formula **(2)**:

$$\text{Mean Squared Error} = \frac{1}{n}\sum_{i=1}^{n}(y_{predicted_i} - y_{actual_i})^2$$

$$\text{Where } n = \text{number of data points}$$

This formula takes the predicted value generated by the model and subtracts it with that of the true data point value for each data point in the data set. This difference is then squared and averaged over the number of data points in the data set. Different methods of linear regression aim to minimize MSE by making changes in the properties of the model over a set of often many iterations. A value of 0 for MSE indicates that the model perfectly predicts each value in the data set; however, this occurrence is very rare, as most data sets are not perfectly linear by nature. **(3)**

2.2) Multiple Linear Regression

As hinted at in section 2.1, linear regression does not have to only exist in the 2D space. In fact, many applications of linear regression, such as machine learning, utilize datasets that can be represented in *n*-dimensional spaces, where *n* is an integer. In multiple linear regression there is

one dependent variable and multiple independent variables **(4)**. For example, if there are two independent variables and one dependent variable, the linear regression would be done in 3D. For each additional independent variable, the regression is done in one dimension higher of space. Therefore, the predicted y value of a linear regression model with *n* number of independent variables is as follows:

$$y_{predicted} = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n \ = f(\theta_0, \theta_1, ..., \theta_n)$$

Where each value of θ is a constant that characterizes the regression model
And n is the number of independent variables involved in the regression

3) Gradient Descent

3.1) Gradient Descent Definition

Gradient descent is a type of iterative optimization algorithm that seeks to minimize a function by adjusting parameters in the direction of the steepest descent of the function (i.e. the negative gradient) **(5)**.

3.2) Steps of Gradient Descent Algorithm

Step Number	Process
1	Define cost function and parameters. - The cost function is the function that the algorithm is aiming to minimize. Let the cost function be the following: $C(...\theta_n...)$ Where θ_n = the nth parameter of the cost function. A random set of initial parameter values should then be chosen. This essentially gives the gradient descent algorithm an initial condition/location to begin descending. **(5)**
2	Find how the cost function changes with respect to each parameter. - This is done by taking the partial derivative of the cost function with respect to each of its parameters. $\nabla C_{\theta n}$ **(5)**
3	Update the parameters by subtracting them with the value of the partial derivative of the cost function with respect to the parameter for the given parameter value multiplied by the learning rate. - The learning rate is a hyperparameter, which is a value that is configured before the algorithm runs **(6)**, that plays a role in controlling how large the change in parameter value is each time the algorithm runs. If the learning rate is too small, the gradient descent algorithm may take a very long time to

	converge on a minimum value of the cost function. Oppositely, if the learning rate is too large, the gradient descent algorithm may take steps that are too large, possibly skipping the value of the parameter(s) that yield the minimum cost function value, thus not allowing the algorithm to succeed in minimizing the cost function. The best learning rates vary between problems; however, an appropriate learning rate for the specific problem at hand can be determined via trial and error and observing the cost function minimum values that each learning rate yields **(7)**. $$\theta_{n,new} = \theta_{n,old} - \nabla C_{\theta n,old}\alpha$$ Where α = learning rate (controls how much the parameter changes with each iteration of the algorithm) Where $\theta_{n,new}$ and $\theta_{n,old}$ equal the newly updated θ_n parameter and the previous θ_n parameter, respectively **(5)**
4	Repeat Step 3 for the amount of times determined by the number of epochs set for the algorithm. - Epochs is another hyperparameter that denotes how many times the gradient descent algorithm should run before yielding its estimate of the minimum cost function value **(8)**. Once the algorithm iterations reach the number of epochs, the final value of the cost function is calculated using the final calculated parameter values. This final value of the cost function can be considered approximately the minimum value of the cost function (if the correct number of epochs and an appropriate learning value were initially chosen for the algorithm). **(5)**

4) Using Gradient Descent For 3D Multiple Linear Regression

4.1) Crafting the Relevant Equations for 3D Multiple Linear Regression

Equation Name	Equation Syntax
Regression Plane	$$y_{predicted_n} = \theta_0 + \theta_1 x_{1n} + \theta_2 x_{2n} = f(\theta_0, \theta_1, \theta_2)$$ Where each value of θ is a constant that characterizes the plane, and n indicates the applicable set of x values of interest for a given actual point, which will be estimated by the point defined by $(x_{1n}, x_{2n}, y_{predicted_n})$
MSE Function	$$\text{Mean Squared Error} = \frac{1}{n}\sum_{i=1}^{n}(\theta_0 + \theta_1 x_{1i} + \theta_2 x_{2i} - y_{actual_i})^2$$ Where n = number of data points

4.2) Converting all Relevant 3D Multiple Linear Regression Equations Into Matrix Form

Equation Name	Equation Syntax
Regression Plane	Let the following definitions be true: $X = \begin{bmatrix} \vert & \vert & \vert \\ 1 & x_{1,n} & x_{2,n} \\ \vert & \vert & \vert \end{bmatrix}$, $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$ Where n represents the nth data point Therefore, the following is true: $y_{predicted} = X\theta$ **(9)**
MSE Function	Let the following definition be true, along with the other definitions introduced in this section (Section 4): $Y = \begin{bmatrix} \vert \\ y_{actual_n} \\ \vert \end{bmatrix}$ Where n represents the nth data point Also consider the following identity and proof: Let $R = \begin{bmatrix} x \\ y \end{bmatrix}$, and $a = x^2 + y^2$ $a = R^T R$ Proof: $R^T = [x, y]$ and $R^T R = [x, y] \begin{bmatrix} x \\ y \end{bmatrix} = x^2 + y^2$ Therefore, the following must be true: $MSE = \frac{1}{n}(X\theta - Y)^T(X\theta - Y)$ **(9)**

4.3) Taking the Gradient of the MSE Function

The gradients of the MSE function with respect to the individual parameters will first be calculated in scalar form in order to give the intuition of what the gradient function for each parameter should look like. Then, the gradient of the MSE function with respect to all of the parameters will be calculated at once using the matrix form, in order to demonstrate how 3D MSE gradients are often represented, especially in computer programs that do linear regression.

a) Apply Gradient When the MSE Function is In Scalar Form

Let $MSE = C = \frac{1}{n}\sum_{i=1}^{n}((\theta_0 + \theta_1 x_{1,i} + \theta_2 x_{2,i}) - y_{actual_i})^2$

Therefore:

$$\frac{\partial C}{\partial \theta_u} = \frac{2}{n}\sum_{i=1}^{n}((\theta_0 + \theta_1 x_{1,i} + \theta_2 x_{2,i}) - y_{actual_i})x_{u,i}$$

Where $u =$ any integer between 0 and 2, inclusive
The above yields the following results:

$$\frac{\partial C}{\partial \theta_0} = \frac{2}{n}\sum_{i=1}^{n}((\theta_0 + \theta_1 x_{1,i} + \theta_2 x_{2,i}) - y_{actual_i})$$

$$\frac{\partial C}{\partial \theta_1} = \frac{2}{n}\sum_{i=1}^{n}((\theta_0 + \theta_1 x_{1,i} + \theta_2 x_{2,i}) - y_{actual_i})x_{1,i}$$

$$\frac{\partial C}{\partial \theta_2} = \frac{2}{n}\sum_{i=1}^{n}((\theta_0 + \theta_1 x_{1,i} + \theta_2 x_{2,i}) - y_{actual_i})x_{2,i}$$

(9)

b) Apply Gradient When the MSE Function is In Matrix Form

Consider the following identity and proof:

Let the following be true:

$$R = \begin{bmatrix} x \\ y \end{bmatrix},$$
$$R^T = \begin{bmatrix} x, y \end{bmatrix},$$
$$dR = \begin{bmatrix} dx \\ dy \end{bmatrix},$$
$$dR^T = \begin{bmatrix} dx, dy \end{bmatrix}$$

Therefore:
$dR^T R = R^T dR$

Proof:

$$dR^T R = R^T dR$$
$$\begin{bmatrix} dx, dy \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x, y \end{bmatrix}\begin{bmatrix} dx \\ dy \end{bmatrix}$$
$$\begin{bmatrix} dx, dy \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = xdx + ydy$$
$$\begin{bmatrix} x, y \end{bmatrix}\begin{bmatrix} dx \\ dy \end{bmatrix} = xdxd + ydy$$

Now see the following:

Let the following be true:
$H = (X\theta - Y)$
$C = MSE = \frac{1}{n}H^T H$

Implicitly differentiate C with respect to θ:

$$dC = \frac{1}{n} dH^T H + \frac{1}{n} H^T dH$$
$$dC = \frac{2}{n} H^T dH$$
$$dC = \frac{2}{n} (H^T)(X) d\theta$$
$$\frac{dC}{d\theta} = \frac{2}{n} (X\theta - Y)^T X$$

Note: The above expression represents a matrix of shape 1x3, but the gradient must be of shape 3x1 Therefore, the following must be true:

$$(\frac{dC}{d\theta})^T = \nabla C_\theta$$

Assuming the above is true,

$$(\frac{dC}{d\theta})^T = \frac{2}{n} X^T (X\theta - Y)$$
$$\nabla C_\theta = \frac{2}{n} X^T (X\theta - Y)$$

∇C_θ is a matrix that looks like the following:

$$\nabla C_\theta = \begin{bmatrix} \frac{\partial C}{\partial \theta_0} \\ \frac{\partial C}{\partial \theta_1} \\ \frac{\partial C}{\partial \theta_2} \end{bmatrix}$$

(9)

4.4) Update the Parameters of the Regression Plane

The individual parameters will first be updated in scalar form in order to give the intuition of how the update function works. Then, the parameters will be updated all at once in matrix form, in order to demonstrate how 3D MSE gradients are often represented, especially in computer programs that do linear regression.

a) Update the Parameters in Scalar Form

Let the following be true:

$$\theta_{n,new} = \theta_{n,old} - \nabla C_{\theta n,old} \alpha$$

Where α = learning rate (controls how much the parameter changes with each iteration of the algorithm)
And n is an integer between 0 and 2, inclusive

Therefore:

$$\theta_{0,new} = \theta_{0,old} - (\frac{2}{n}\sum_{i=1}^{n}((\theta_0 + \theta_1 x_{1,i} + \theta_2 x_{2,i}) - y_{actual_i}))\alpha$$

$$\theta_{1,new} = \theta_{1,old} - (\frac{2}{n}\sum_{i=1}^{n}((\theta_0 + \theta_1 x_{1,i} + \theta_2 x_{2,i}) - y_{actual_i})x_{1,i})\alpha$$

$$\theta_{2,new} = \theta_{2,old} - (\frac{2}{n}\sum_{i=1}^{n}((\theta_0 + \theta_1 x_{1,i} + \theta_2 x_{2,i}) - y_{actual_i})x_{2,i})\alpha$$

Where n in this section is NOT an integer between 0 and 2, inclusive, but is the number of data points involved in the regression

(9)

b) Update the Parameters in Matrix Form

Assume all definitions from section 4.3.b) are true and the following is true:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

The parameters can be updated as follows:

$$\theta_{new} = \theta_{old} - \nabla C_{\theta old}\alpha$$

Where θ_{new} and θ_{old} equal the newly updated θ matrix and the previous θ matrix, respectively

The expanded version of the above expression is:

$$\theta_{new} = \theta_{old} - (\tfrac{2}{n}X^T(X\theta_{old} - Y))\alpha$$

5) Example With Real Data

5.1) Dataset

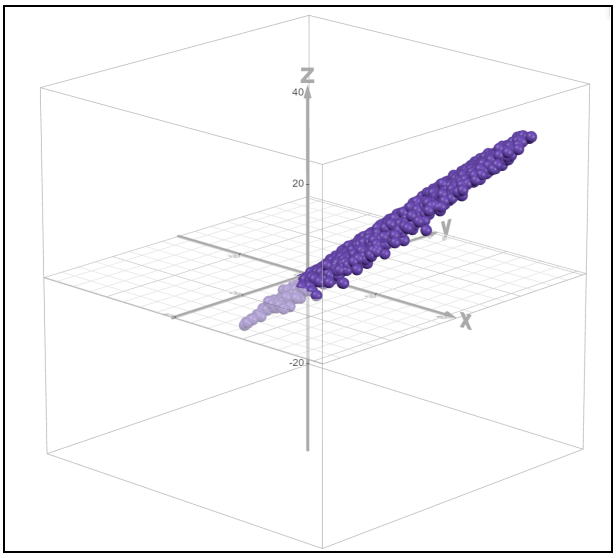
The dataset used for this project was the *Washington DC Historical Weather 2015/8~2024/07* found on Kaggle.com. This dataset includes different types of weather-related data collected in Washington DC for different days between August 2015 and July 2024 **(10)**.

I chose to look at the following types of data for each day:

- Minimum temperature
- Maximum temperature
- Overall temperature

I selected the minimum and maximum temperatures to be the two independent variables, and the overall temperature to be the dependent variable, since such a relationship proved to be approximately linear.

The 3D visualization of the data was created using Desmos.com, and can be seen below:



(11)

The x-axis represents the maximum temperature, the y-axis represents the minimum temperature, and the z axis represents the overall temperature.

5.2) Code

I programmed a 3D multiple linear regression using gradient descent for the selected dataset and variables on Google Colab **(12)**. I used a YouTube video by Kody Simpson **(13)** to help me craft the algorithm, and ChatGPT to give me pointers on how to use different Python functions (as well as debug some blocks of code), as I am not very familiar with the specific Python syntax **(9)**.

The full code can be found at this link:
https://colab.research.google.com/drive/1tbMq7-Yci7QyytdbuQ2yScQzRUwEae54#scrollTo=8-bcMl9-lPE8

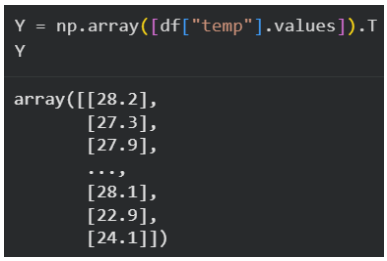
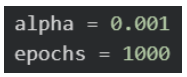
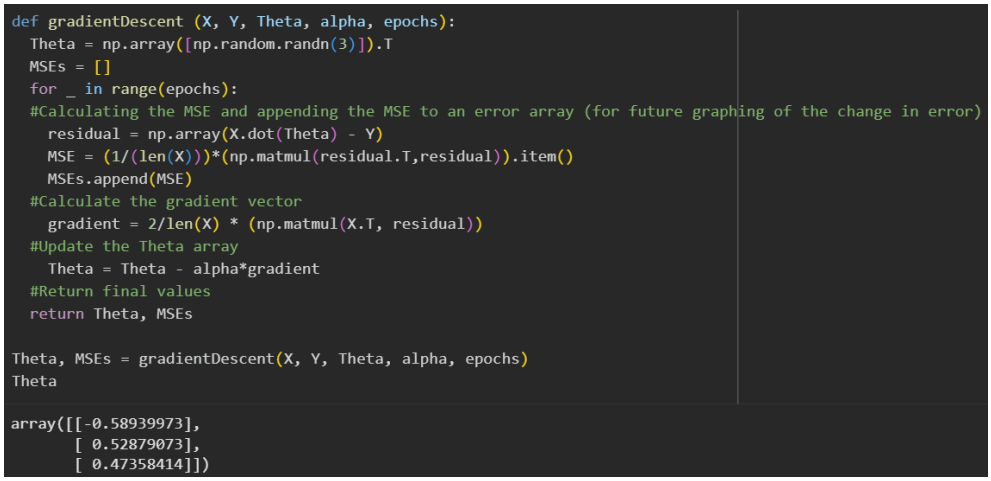
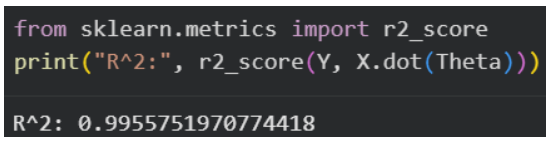
Screenshots of the code and its output for a given run are shown below:

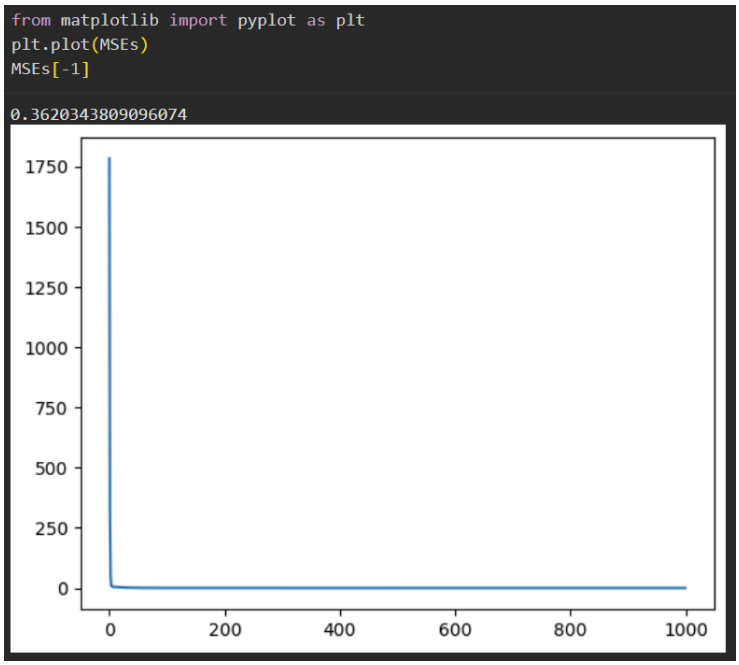
Content	Code
Imports and basic initializations	

```
!pip install kagglehub
import numpy as np
import pandas as pd
import kagglehub

# Download latest version
path = kagglehub.dataset_download("taweilo/washington-dc-historical-weather-20158202407")

print("Path to dataset files:", path)
df = pd.read_csv("/kaggle/input/washington-dc-historical-weather-20158202407/dc_weather.csv")
df = df.iloc[0:3321, 2:5]
df.head()
```

```
Requirement already satisfied: kagglehub in /usr/local/lib/python3.12/dist-packages (0.3.13)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from kagglehub) (25.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.12/dist-packages (from kagglehub) (6.0.3)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from kagglehub) (2.32.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from kagglehub) (4.67.1)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->kagglehub) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->kagglehub) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->kagglehub) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->kagglehub) (2025.11.12)
Using Colab cache for faster access to the 'washington-dc-historical-weather-20158202407' dataset.
Path to dataset files: /kaggle/input/washington-dc-historical-weather-20158202407
```

	tempmax	tempmin	temp
0	33.1	22.8	28.2
1	32.0	22.8	27.3
2	33.2	21.8	27.9
3	35.3	24.9	29.3
4	33.6	24.0	28.6

Initializing the Y matrix	
Initializing the Theta matrix	
Initializing the alpha and epochs variables	 **Note:** I determined these values through trial and error with my code, particularly by analyzing the MSE over epochs graph shown at the end of this section.
Creating and declaring the gradient descent algorithm (which first populates Theta with random values in order to give the algorithm a starting point), as well as printing the final values of Theta	
Finding the R^2 score of the given run	 **Note:** R^2 score is a measure of how much the independent variables in a statistical model account for the variation of the actual dataset's dependent variables. A value of 0 means the model does not fit the data set at all, while a value of 1 means the model fits the data perfectly **(14)**. The R^2 value yielded for this run, along with all of the other runs I conducted using this program, all indicate a strong fit of the model to the true data.

Finding the final MSE value and generating an MSE over epochs graph	

```
from matplotlib import pyplot as plt
plt.plot(MSEs)
MSEs[-1]
```

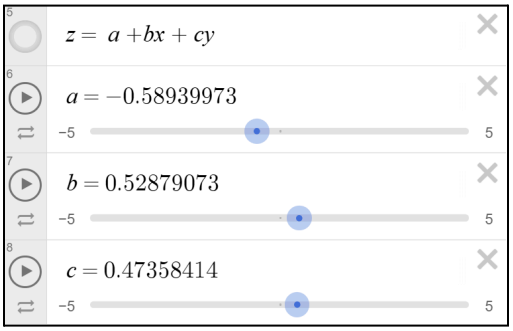
```
0.3620343809096074
```


5.3) Final Results

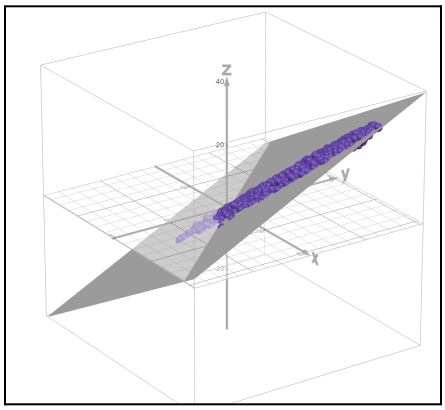
Using the final values of the Theta matrix yielded by the program,

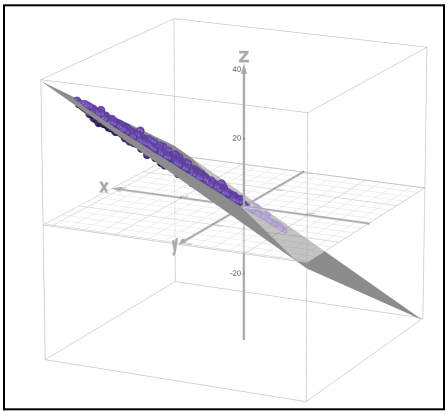
```
array([[-0.58939973],
       [ 0.52879073],
       [ 0.47358414]])
```

I created the following plane equation on Desmos **(11)**,



and visualized the plane alongside the data points:





6) Sources

6.1) Write-Up Sources

- (1) https://www.ibm.com/think/topics/linear-regression
- (2) https://www.geeksforgeeks.org/maths/mean-squared-error/
- (3) https://en.wikipedia.org/wiki/Mean_squared_error
- (4) https://www.geeksforgeeks.org/machine-learning/ml-multiple-linear-regression-using-python/
- (5) https://www.geeksforgeeks.org/data-science/what-is-gradient-descent/
- (6) https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)
- (7) https://medium.com/@karurpabe/gradient-descent-how-to-find-the-learning-rate-142f6b843244
- (8) https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-epoch-in-machine-learning
- (9) https://chatgpt.com/ *Used to help me understand the derivation behind equations, as well as understand Python syntax and debug code*
- (10) https://www.kaggle.com/datasets/taweilo/washington-dc-historical-weather-20158202407/data
- (11) https://www.desmos.com/
- (12) https://colab.google/
- (13) https://www.youtube.com/watch?v=O96hzKRx3O4
- (14) https://www.investopedia.com/terms/r/r-squared.asp

6.2) Slide-Specific Sources

- (1) https://statsandr.com/blog/multiple-linear-regression-made-simple
  *The above source is for the following image on the slideshow: