# MACHINE LEARNING CSE 6363

Assignment – 1

Name : TEJAS RAGHUTTAHALLI MANJUNATH REDDY

ID : 1002157872

## LINEAR REGRESSION:

A supervised learning approach called linear regression is used to forecast continuous outcome variables based on one or more predictor variables. Finding the best-fit straight line to represent the relationship between the input features and the target variable is the aim of linear regression.
For this project, we created a class on linear regression from scratch.

## Fit method -

The fit approach uses gradient descent with mean squared error as the loss function in an attempt to optimize the model parameters. The input data, the target values, the batch size, the regularization factor (which is set to 0 by default), the maximum number of training epochs, and the likelihood of an early stop are the six parameters that the method requires. Optimally, a learning rate of 0.01 has been employed. The method divides the data into batches based on the batch size during training using gradient descent. L2 regularization is used in accordance with the given factor. The model's performance on the validation set is assessed after every iteration. Training is stopped if the loss rises for three consecutive steps. On the other hand, pickle is used to save the present model parameters if it drops.

```
            # Append to loss history
            training_loss_history.append(training_loss)
            val_loss_history.append(val_loss)
            print(f"Epoch {epoch + 1}, Training Loss: {training_loss}")
            # Check for early stopping
            if val_loss < best_loss:
                best_loss = val_loss
                best_weights = np.copy(self.weights)
                best_bias = np.copy(self.bias)
                count = 0
            else:
                count += 1
                if count >= patience:
                    print(f"Early stopping at epoch {epoch}")
                    break

        self.weights = best_weights
        self.bias = best_bias

        return training_loss_history, val_loss_history

```

## Predict method -

This technique includes code that predicts the target values by using the inputs. The estimated values are provided by

```
37      def predict(self, X):
38
39          if len(X.shape) == 1:
40              X = X.reshape(-1, 1)
41
42          z = np.dot(X, self.weights) + self.bias
43          pred = self.sigmoid(z)
44          return (pred >= 0.5).astype(int)
45
```

## Score Method -

Two parameters—input data and corresponding output values—are required for the score technique to function. The mean squared error between these estimated values and the actual target values is computed. Here is how the MSE values are determined:

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

## Save and Load -

The save function enables the model parameters to be saved to a specified file directory for easy future retrieval. The load method, on the other hand, is the opposite; it permits the reconstruction of the

model parameters from a given file location. This process guarantees our regression model's dependability.

```
# Load the model parameters from the saved file
with open('NRmodel3_params.pkl', 'rb') as file:
    model_params = pickle.load(file)

```

**Training** -
In order to train every model, four distinct scripts are developed, such as TrainRegression1, TrainRegression2, etc., each of which consists of a different combination of input and output features.

We load the iris dataset first. Sepal length is used as the input, while sepal width is the result. After that, we divided the data into training and testing sets, using a test size of 0.1. Initialization and fit of the non-linear regression model are done with regularization = 0. Using MSE as the loss function, this model uses batch gradient descent with a batch size of 32. A training loss history graph is plotted, and the model parameters are recorded. The model's bias and weights are published.

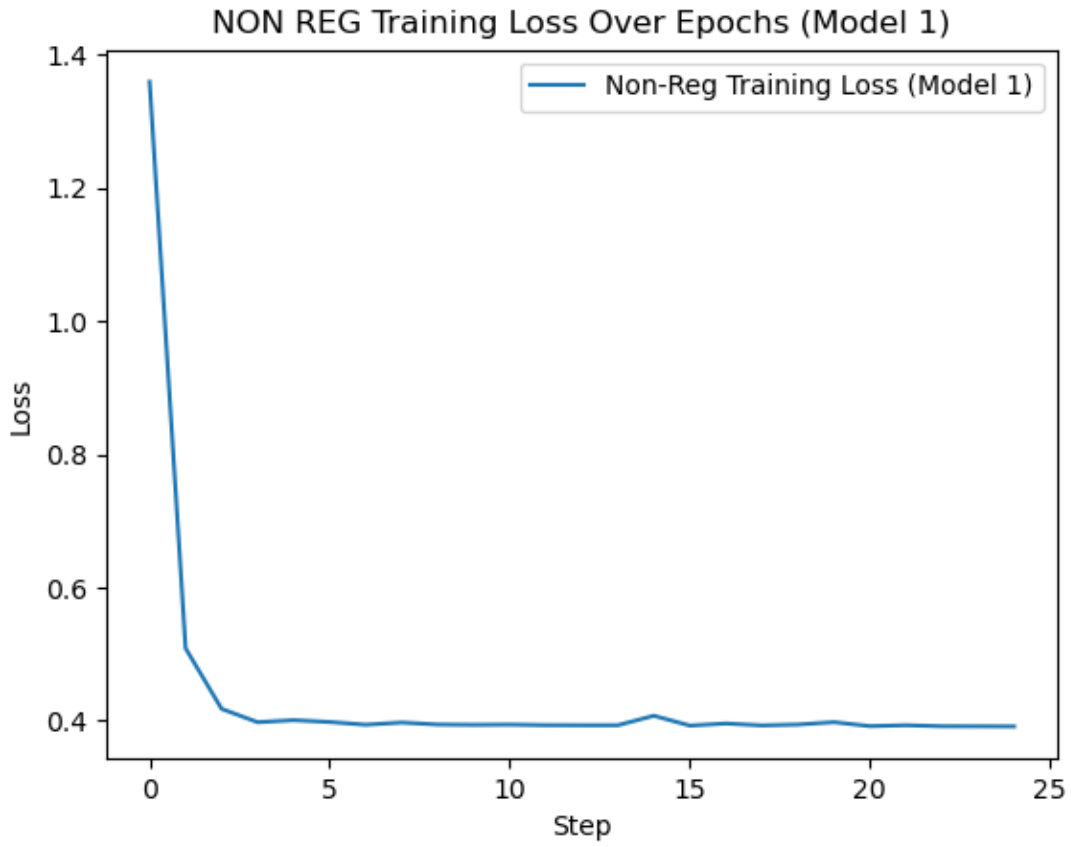
```
# Load Iris dataset for Model 1
iris_data_model1 = load_iris()
features_model1, labels_model1 = iris_data_model1.data, iris_data_model1.target

```

The model now incorporates L2 regularization. The  training loss history graph is plotted and the model parameters are recorded. The model's bias and weights are published. It is computed what the parameters of regularized and non-regularized models differ from one another.

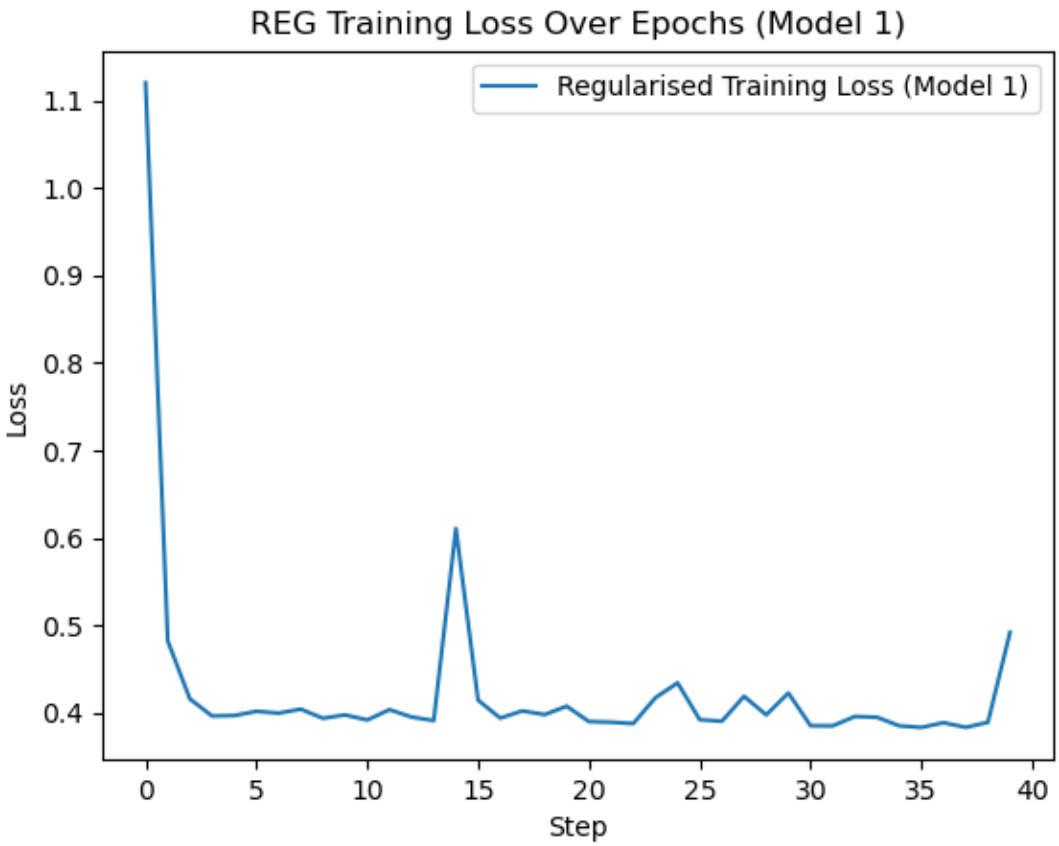
The loss, variance in weights, and bias between the models and the plots are provided by the script when it is executed.

Each training epoch sees a reduction in the loss. For this reason, we draw a graph of loss against step number.

This graph shows loss function without regularization.

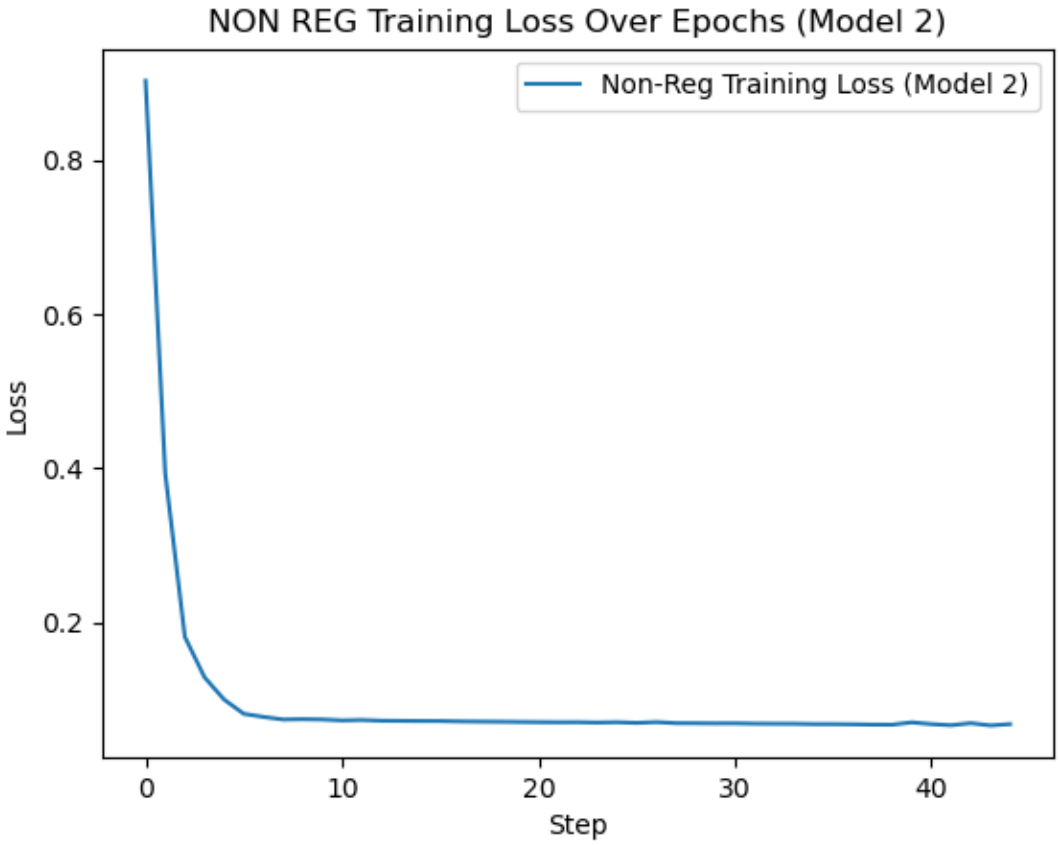


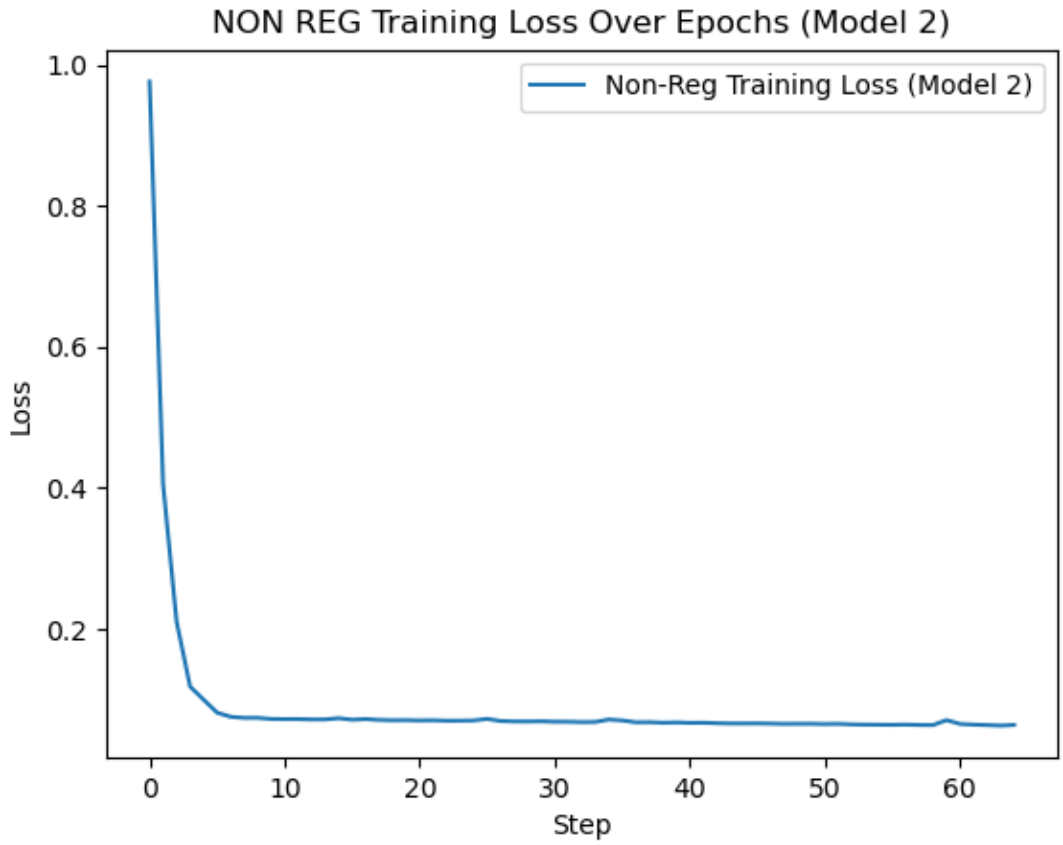
This graph shows loss function with regularization.



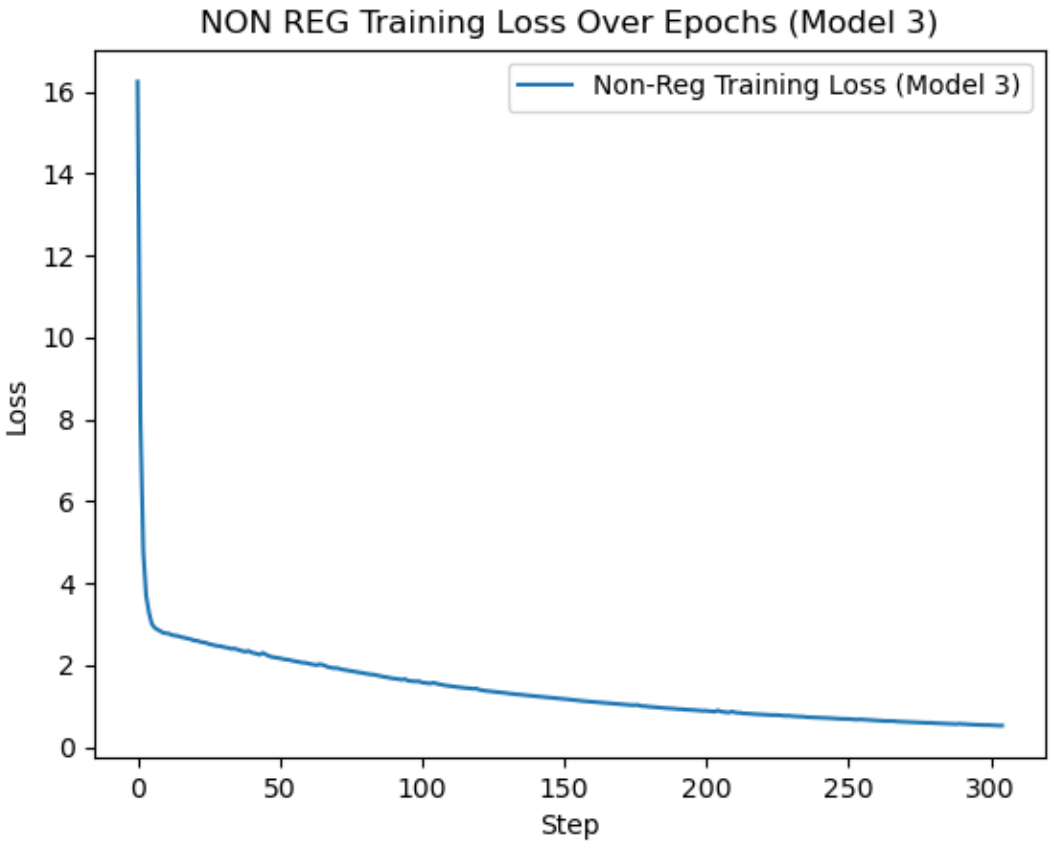

After this, pickle is used to save the model parameters. After that, this may be applied to any other testing data.
With varying inputs and outputs, the same process has been applied to the remaining three models as well. The other models' plots are shown below.

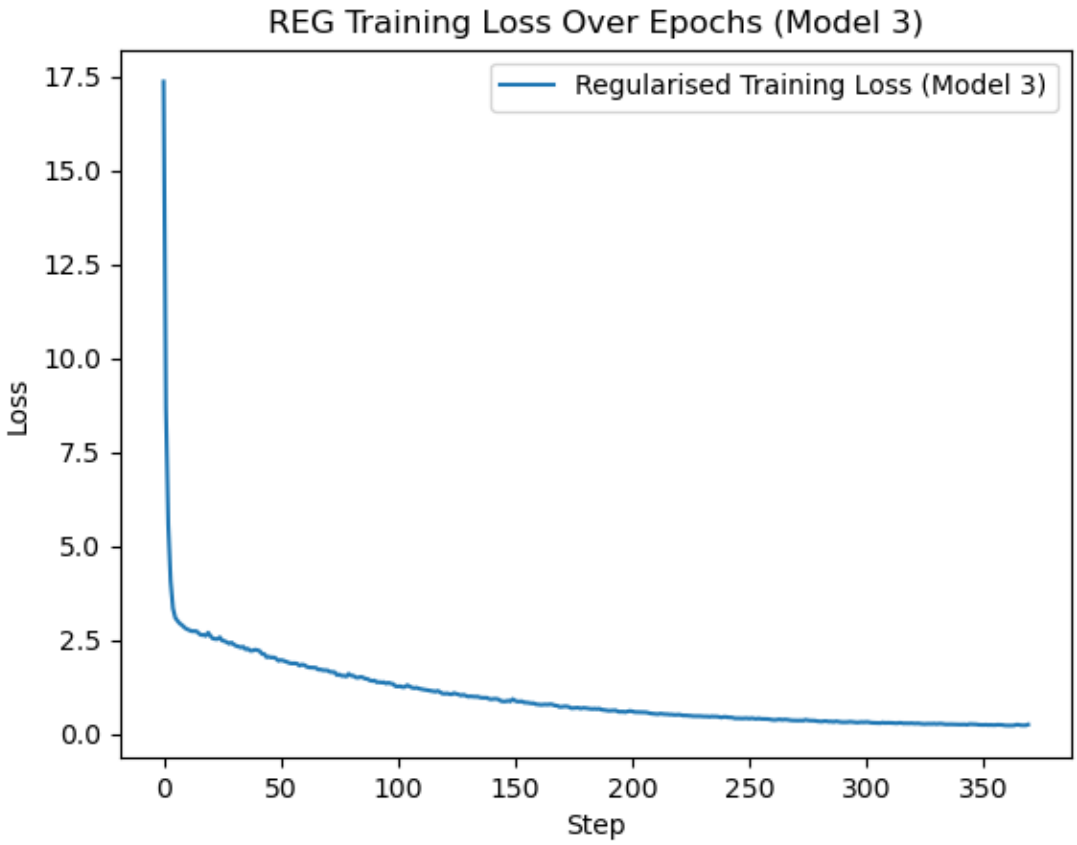
Model 2 plots –


NON REG Training Loss Over Epochs (Model 2)
Non-Reg Training Loss (Model 2)
Loss
0.8
0.6
0.4
0.2
0
10
20
30
40
Step

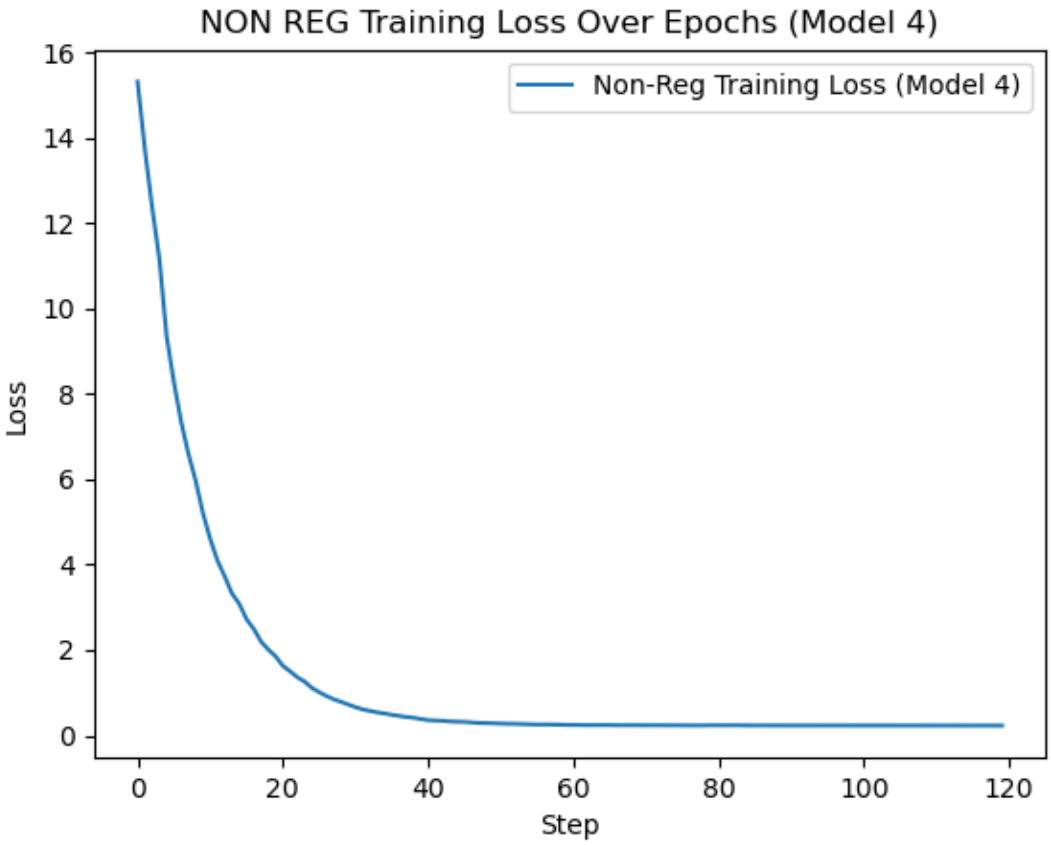


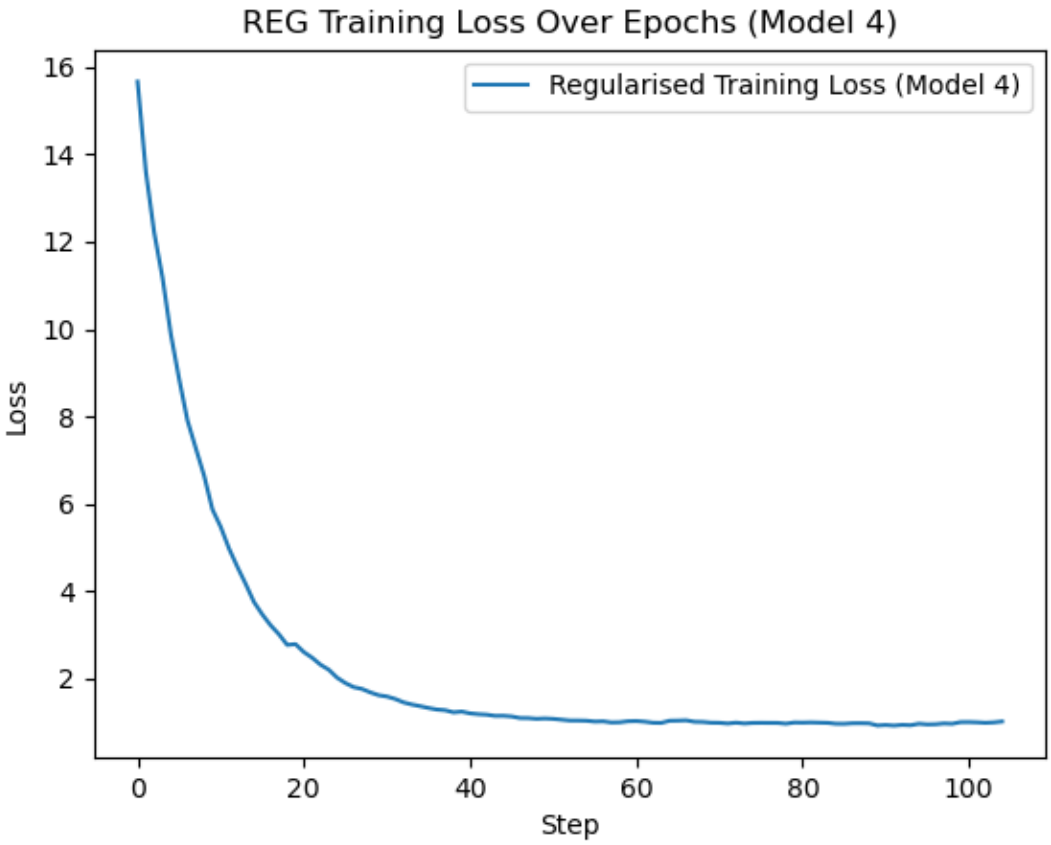

Model 3 plots –


NON REG Training Loss Over Epochs (Model 3)
Non-Reg Training Loss (Model 3)
Loss
Step
0
2
4
6
8
10
12
14
16
0
50
100
150
200
250
300




**Model 4 plots –**

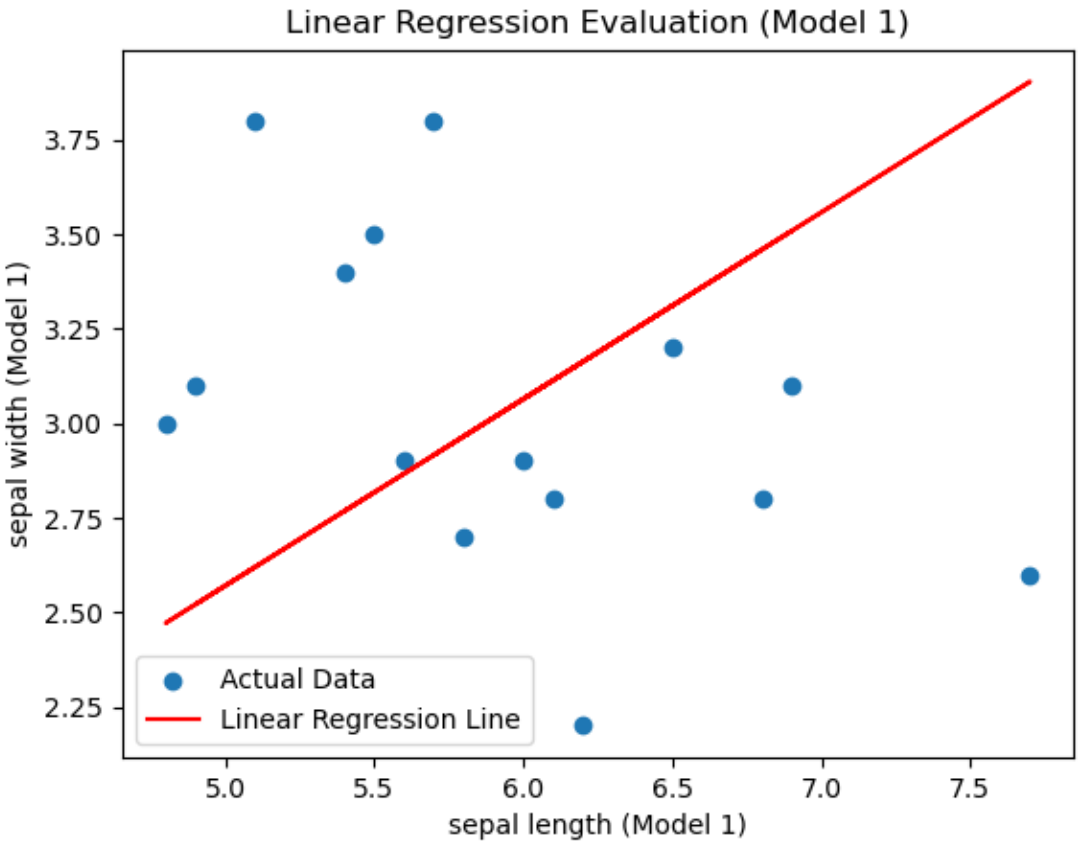

NON REG Training Loss Over Epochs (Model 4)
Non-Reg Training Loss (Model 4)
Loss
Step
0
2
4
6
8
10
12
14
16
0
20
40
60
80
100
120

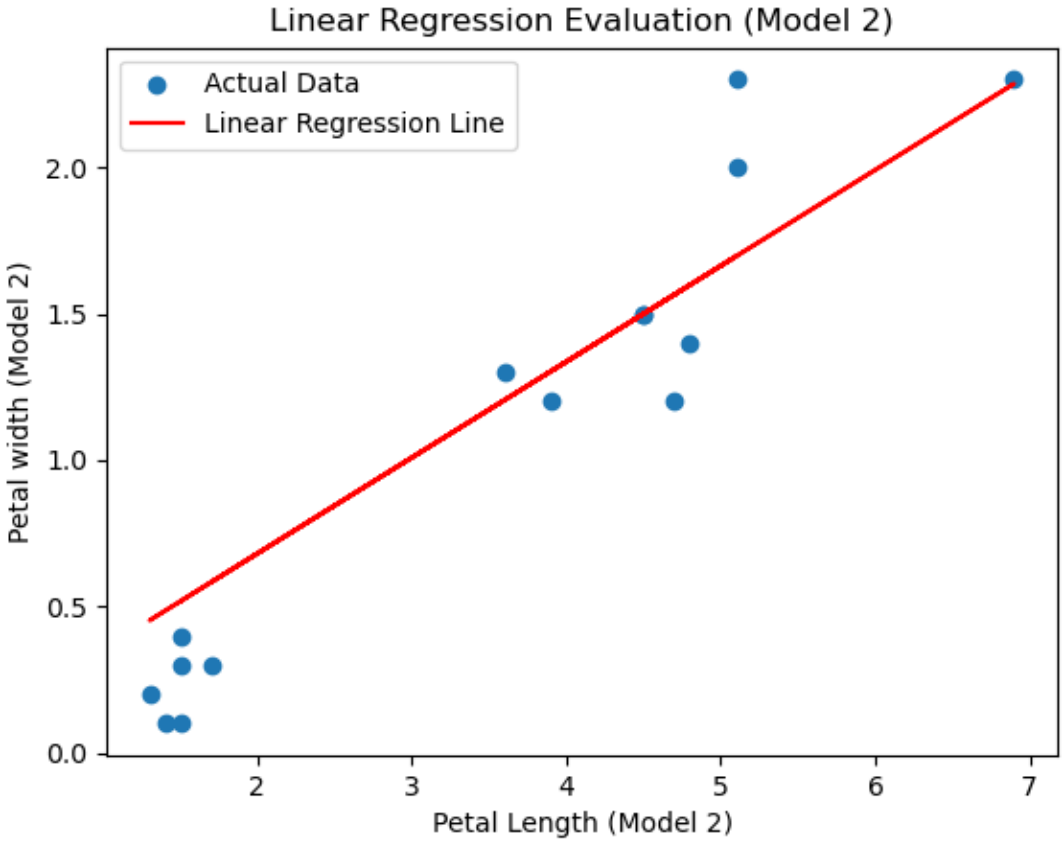


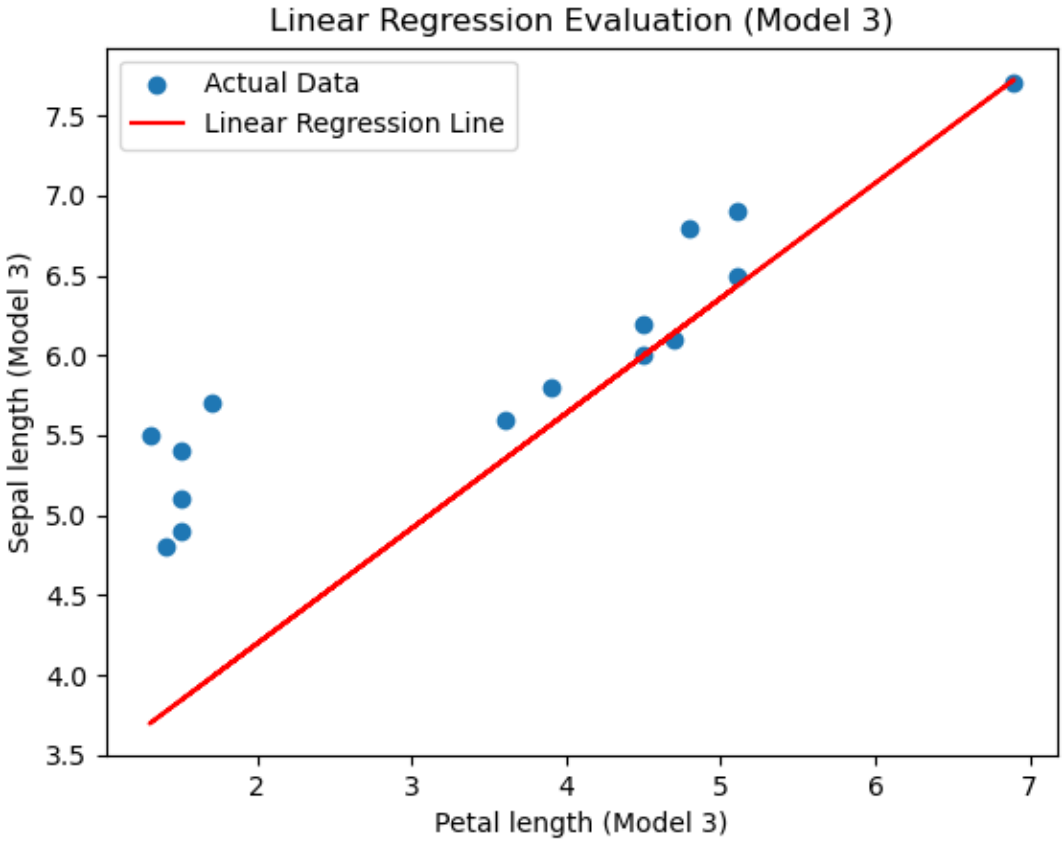

**Testing** - The evaluation script, such as Evaluation1, Evaluation2, and so on, loads the model parameters and calls the predictions method. The predictions and mean square error are printed by this script. Additionally, the regression best-fit line plot is implemented

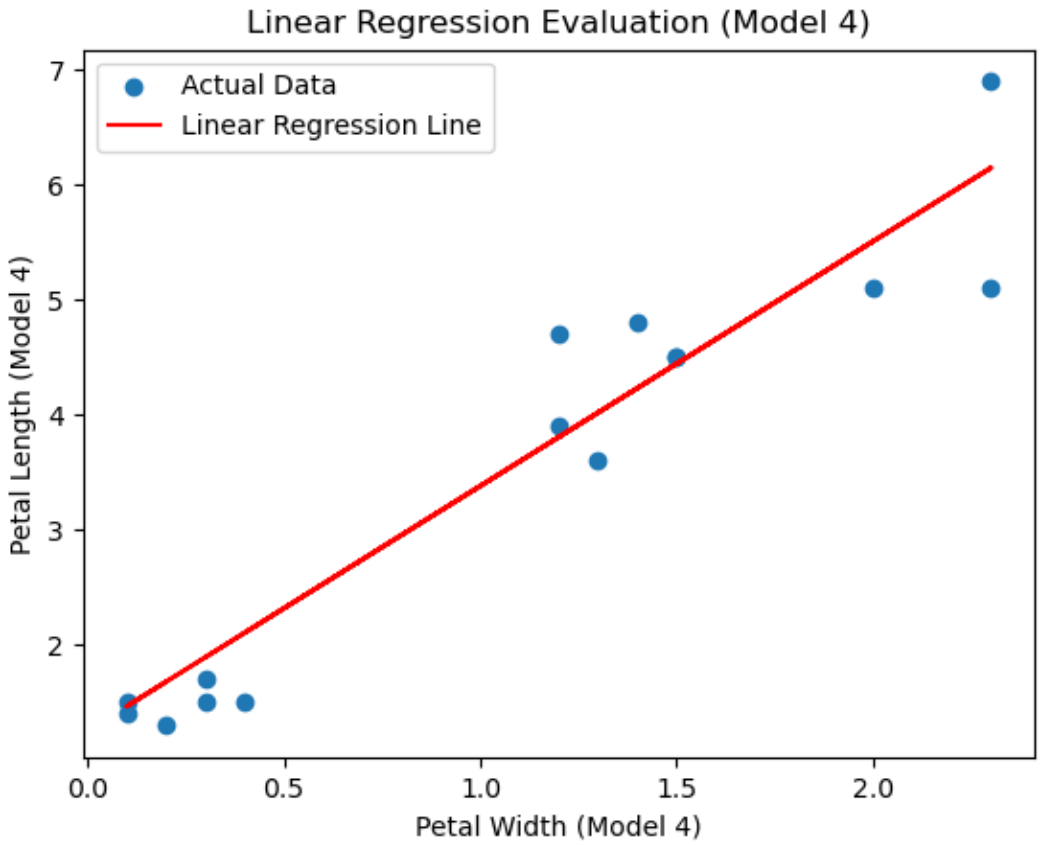
eval_regression1.py

```python
import pickle
from TrainRegression1 import X_test_model1, Y_test_model1  # Assuming TrainRegression4.py contains the test data
import matplotlib.pyplot as plt

# Reshaping test data to 2D arrays
X_test_model1_reshaped = X_test_model1.reshape(-1, 1)

# Load the model parameters from the saved file
with open('NRmodel1_params.pkl', 'rb') as file:
    model_params = pickle.load(file)

# Predict on the test set
y_pred_model1 = model_params.predict(X_test_model1_reshaped)

# Calculate and print the mean squared error using the model's score method
mse_model1 = model_params.score(y_pred_model1, Y_test_model1)
print(f"\nMean Squared Error (Model 4): {mse_model1}")

# plotting linear regression graph between sepal length and sepal width
plt.scatter(X_test_model1, Y_test_model1, label='Actual Data')
plt.plot(X_test_model1, y_pred_model1, color='red', label='Linear Regression Line')
plt.xlabel('sepal length (Model 1)')
plt.ylabel('sepal width (Model 1)')
plt.title('Linear Regression Evaluation (Model 1)')
plt.legend()
plt.show()
```


Linear Regression Evaluation (Model 1)
sepal width (Model 1)
sepal length (Model 1)
3.75
3.50
3.25
3.00
2.75
2.50
2.25
5.0
5.5
6.0
6.5
7.0
7.5
Actual Data
Linear Regression Line


Linear Regression Evaluation (Model 2)
Actual Data
Linear Regression Line
Petal width (Model 2)
Petal Length (Model 2)
0.0
0.5
1.0
1.5
2.0
2
3
4
5
6
7


Linear Regression Evaluation (Model 3)
Actual Data
Linear Regression Line
Sepal length (Model 3)
Petal length (Model 3)
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
2
3
4
5
6
7

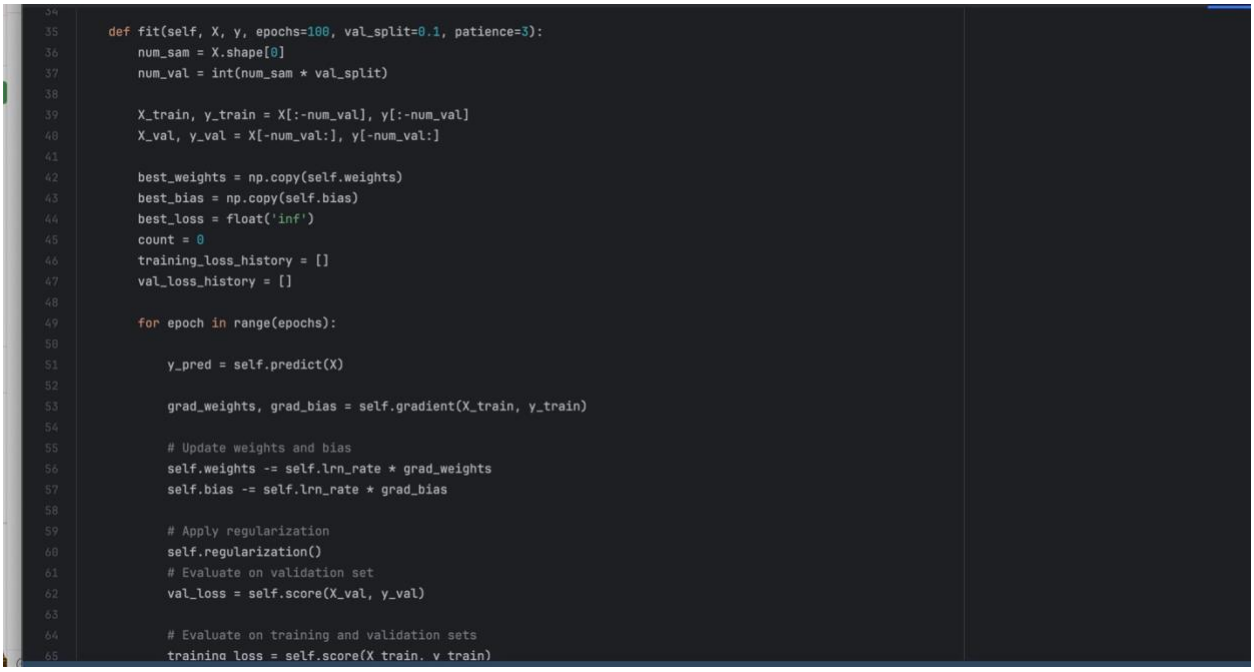



## REGRESSION WITH MULTIPLE OUTPUTS

Using the input variables of sepal length and width, our goal in this section is to construct a regression model that concurrently predicts petal length and width. The main change will be the prediction of two output values rather than just one. The data preparation procedures from the preceding linear regression section will be repeated. We'll use the same techniques for fit, predict, and score.

**Fit Method** - Using gradient descent and mean squared error as the

loss function, the fit method's objective is to optimize the model parameters. The approach uses gradient descent to separate the data into batches based on the batch size during training. Based on the given factor, L2 regularization is applied. The model's performance is assessed on the validation set after every iteration. Training is discontinued after three consecutive steps where the loss increases.



```
def fit(self, X, y, epochs=100, val_split=0.1, patience=3):
    num_sam = X.shape[0]
    num_val = int(num_sam * val_split)

    X_train, y_train = X[:-num_val], y[:-num_val]
    X_val, y_val = X[-num_val:], y[-num_val:]

    best_weights = np.copy(self.weights)
    best_bias = np.copy(self.bias)
    best_loss = float('inf')
    count = 0
    training_loss_history = []
    val_loss_history = []

    for epoch in range(epochs):

        y_pred = self.predict(X)

        grad_weights, grad_bias = self.gradient(X_train, y_train)

        # Update weights and bias
        self.weights -= self.lrn_rate * grad_weights
        self.bias -= self.lrn_rate * grad_bias

        # Apply regularization
        self.regularization()
        # Evaluate on validation set
        val_loss = self.score(X_val, y_val)

        # Evaluate on training and validation sets
        training_loss = self.score(X_train, y_train)
```

**Predict method** - Code for predicting the target values based on the inputs is included in this method. The anticipated values are provided by

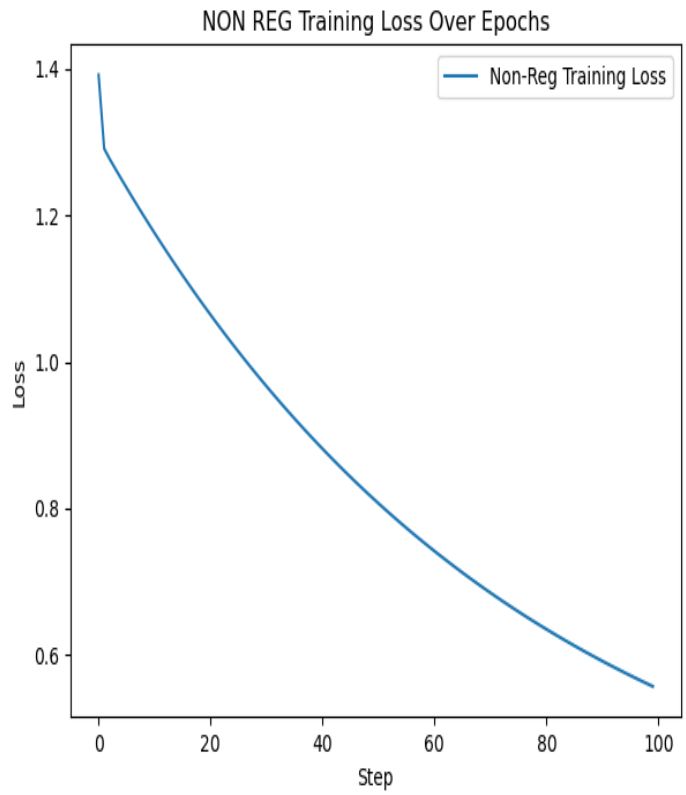
```
15      def predict(self, X):
16          return np.dot(X, self.weights) + self.bias
17
```

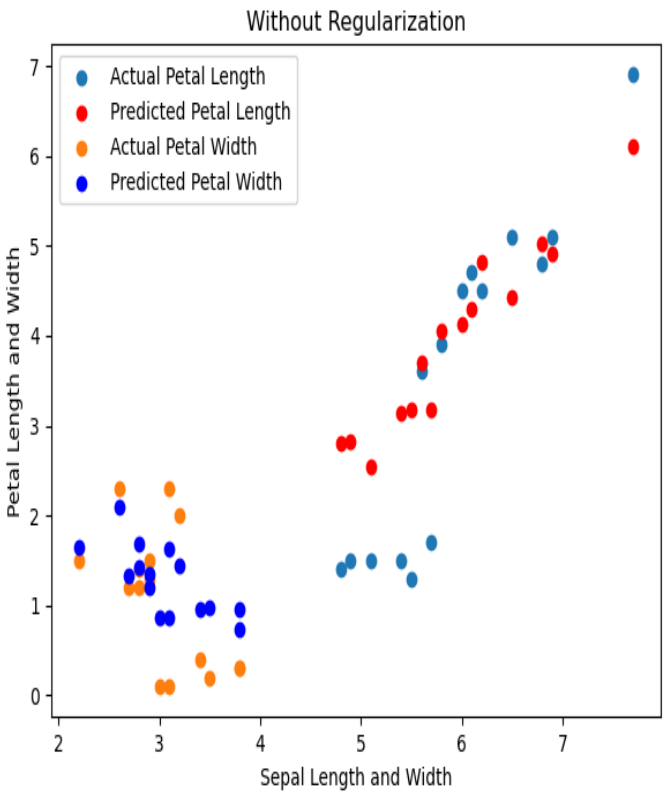
**Score method** - Two parameters—input data and corresponding output values—are required for the score method to function. The

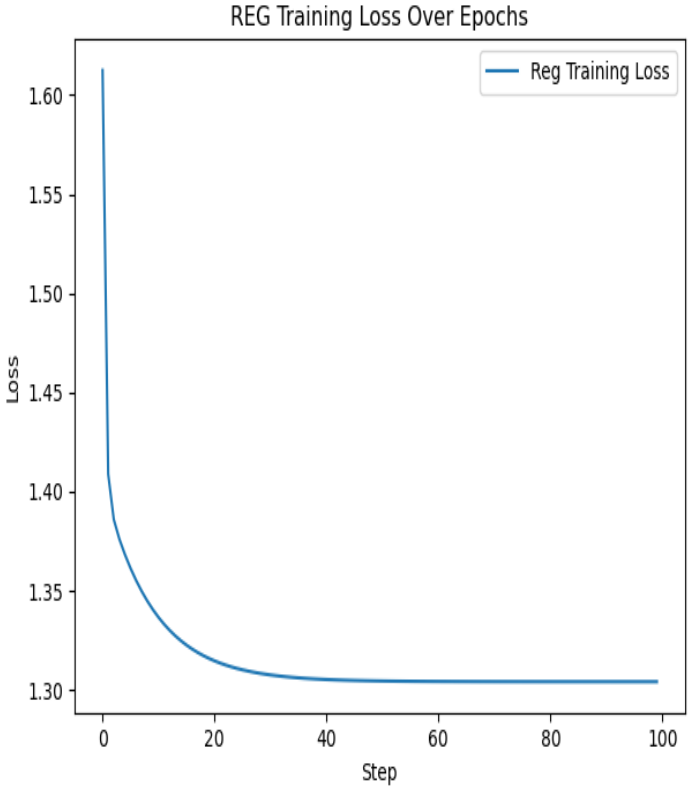
mean squared error between these estimated values and the actual target values is computed. The MSE values are determined:

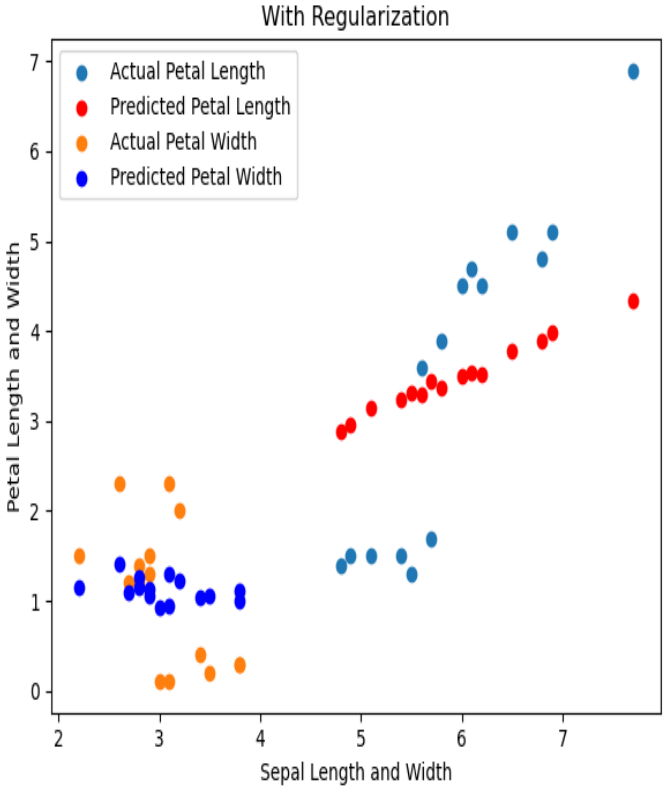
```
    def score(self, X, y):
        y_pred = self.predict(X)
        n, m = y.shape
        mse = np.sum((y - y_pred) ** 2) / (n * m)
        return mse
```

Loss history plots are plotted


NON REG Training Loss Over Epochs
Non-Reg Training Loss
Loss
Step


Without Regularization
Actual Petal Length
Predicted Petal Length
Actual Petal Width
Predicted Petal Width
Petal Length and Width
Sepal Length and Width


REG Training Loss Over Epochs
Reg Training Loss
Loss
Step


With Regularization
Actual Petal Length
Predicted Petal Length
Actual Petal Width
Predicted Petal Width
Petal Length and Width
Sepal Length and Width

# LOGISTIC REGRESSION

**Fit Method** - With this approach, batch gradient descent is used to optimize the model parameters. It first determines the weights and bias of the incoming data, reshapes it if necessary, then iteratively updates it using gradients and predictions. The process teaches the model to forecast binary results based on input data.

```
18      def gradient(self, X, y):
19          num_sam = X.shape[0]
20
21          # gradients for weights
22          grad_weights = (-2 / num_sam) * np.dot(X.T, (y - self.predict(X)))
23
24          # gradients for bias
25          grad_bias = (-2 / num_sam) * np.sum(y - self.predict(X), axis=0)
26
27          return grad_weights, grad_bias
```

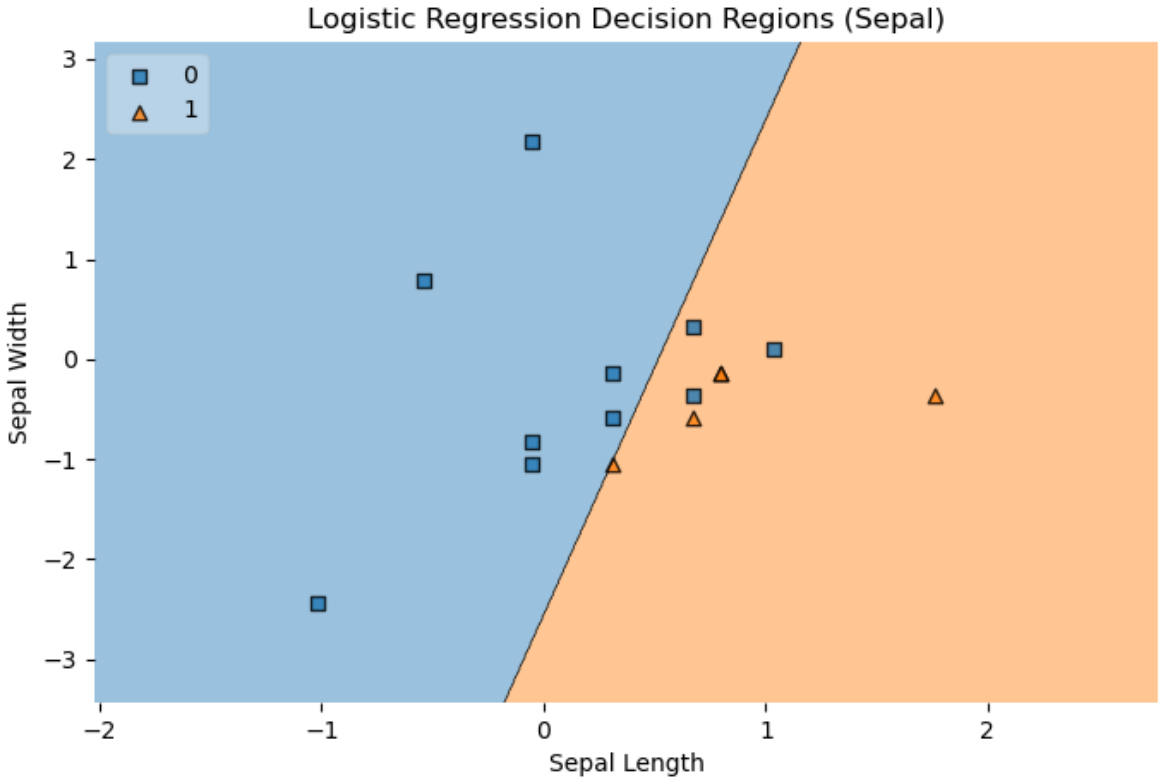
**Predict Method** - Based on the feature input, this method's code predicts the target values.
The sigmoid must be used to process the dot products of the input and weights in order to scale the values to fall between 0 and 1.

```
37      def predict(self, X):
38
39          if len(X.shape) == 1:
40              X = X.reshape(-1, 1)
41
42          z = np.dot(X, self.weights) + self.bias
43          pred = self.sigmoid(z)
44          return (pred >= 0.5).astype(int)
45
```

**Evaluation** - For the evaluation part, three scripts are made (eval_classifier1, eval_classifier2 and eval_classifier3 in which each script has iris data loaded, the input features are standardised, the dataset is split into testing and training with test size= 0.1 and then the fit method is called on test data to train the model. It performs binary classification on the Iris dataset, specifically classifying class 2 against other classes. The predictions are made on the test set and accuracy is estimated for each model and the classifiers are visualised for understanding their decision boundaries.

**Model 1** - Petal length is the input used in this model. Our accuracy comes out to be 86.67%. Below is a graph of the choice region along with the actual and anticipated values.


**Model 2** - The inputs for this model are the sepal width and length. An accuracy of 80% is obtained. Below is a graph of the choice region along with the actual and anticipated values.




**Model 3** - All of the features are used as input in this model. Our accuracy comes out to be 66.66%. The following table displays the model's actual and anticipated values.

```
Actual values [1 2 2 1 2 0 0 0 2 1 0 2 1 1 0]

Predicted values [1 1 1 1 1 0 0 0 1 1 0 1 1 1 0]

Accuracy (All Features): 66.66666666666666%
```