# Data-Driven Methods For Engineers
# Linear Regression: Tast 1 & 2

*Coursework 2 – Group 61*

**The University of Manchester**
Aerospace Engineering
AERO40041 - Data-Driven Methods for Engineers - Saleh Rezaeiravesh
Semester 1, 2025

---

**Juan Ignacio Doval Roque, Sacha Muller**
09.12.2025 - final

# Contents

# 1 Linear Regression using SGD & MSE

## 1.1 Code: LinearRegression_SGD.py

The code produced for Task 1 can be found below:

```python
# Linear Regression using Stochastic Gradient Descent (SGD)
# AERO40041 Coursework 2 - Task 1
#
# This file was prepared by Juan Ignacio Doval Roque [10752534]
#
# NOTE: Plots have been hard coded for given data (Qdot vs dT), therefore, if other data is used,
# the plotting section may need to be adjusted accordingly. Also the path to the data file should be changed.

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Change as required
your_path = '' # Currently at root
data_file = "window_heat.csv"
y_label = '$\dot{Q}$ (W)'
x_label = 'ΔT (°C)'

def load_and_prep_data(filename):
    """
    Load dataset from CSV and prepare feature matrix X and target vector y.
    Adds a column of ones to X, for the bias term.
    """
    data = pd.read_csv(filename)

    X = data.iloc[:, :-1].values
    y = data.iloc[:, -1].values.reshape(-1, 1)

    ones = np.ones((X.shape[0], 1))
    X = np.concatenate((ones, X), axis=1)

    return X, y
```

Listing 1 - Information, Imports, Paths, and Data preparation

```python
def mse_loss(X, y, w):
    """
    Compute Mean Squared Error loss.

    MSE = (1/N) * sum((y_pred - y)^2)
    """
    N = X.shape[0]
    y_pred = X @ w
    error = y_pred - y
    loss = (1/N) * np.sum(error ** 2)
    return loss
```

Listing 2 - MSE Loss function

```python
def sgd_linear_regression(X, y, learning_rate=0.01, num_epochs=100, seed=42):
    """
    Train linear regression using Stochastic Gradient Descent.

    In SGD, we update weights after each individual sample.
    One EPOCH means we have processed every sample exactly once.

    Parameters:
    -----------
    X : numpy array of shape (N, D+1)
        Feature matrix with bias column
    y : numpy array of shape (N, 1)
        Target values
    learning_rate : float
        Step size for gradient updates
    num_epochs : int
        Number of complete passes through the dataset
    seed : int
        Random seed for reproducibility

    Returns:
    --------
    w : numpy array
        Learned weights
    loss_history : list
        Loss value recorded at the end of each epoch
    """
    np.random.seed(seed)

    N, D = X.shape
    w = np.random.randn(D, 1) * 0.01
    loss_history = []

    for epoch in range(num_epochs):
        indices = np.random.permutation(N)

        for i in indices:
            xi = X[i:i+1, :]
            yi = y[i:i+1, :]
            y_pred = xi @ w
            error = y_pred - yi
            gradient = 2 * xi.T @ error
            w = w - learning_rate * gradient

        current_loss = mse_loss(X, y, w)
        loss_history.append(current_loss)

        if (epoch + 1) % 10 == 0 or epoch == 0:
            print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {current_loss:.6f}")

    return w, loss_history

def predict(X, w):
    """Make predictions using learned weights."""
    return X @ w
```

Listing 3 - Stochastic Gradient Descent Linear Regression function and Prediction function

```python
if __name__ == "__main__":

    X, y = load_and_prep_data(your_path + data_file)

    dT_original = X[:, 1].copy()

    print("DATA CHECK:")
    print(f"dT range: {dT_original.min():.2f} to {dT_original.max():.2f} °C")
    print(f"{y_label} range: {y.min():.2f} to {y.max():.2f} W")
    print(f"Dataset shape: {X.shape[0]} samples, {X.shape[1]-1} features")

    X_normalized = X.copy()
    means = X[:, 1:].mean(axis=0)
    stds = X[:, 1:].std(axis=0)
    stds[stds == 0] = 1
    X_normalized[:, 1:] = (X[:, 1:] - means) / stds

    print("\nStart of SGD Linear Regression:")

    w_sgd, loss_history = sgd_linear_regression(
        X_normalized, y,
        learning_rate=0.01,
        num_epochs=100,
        seed=42
    )

    print(f"\nFinal Loss (MSE): {loss_history[-1]:.6f}")
    print(f"Learned weights (normalized): {w_sgd.flatten()}")

    y_pred = predict(X_normalized, w_sgd)

    final_mse = np.mean((y - y_pred) ** 2)
    final_rmse = np.sqrt(final_mse)
    print(f"Final RMSE: {final_rmse:.6f}")
```

Listing 4 - Main Execution Block

```python
    fig, axes = plt.subplots(3, 1, figsize=(8, 12))

    # Plot 1: Loss vs Epoch
    axes[0].plot(range(1, len(loss_history) + 1), loss_history, 'b-', linewidth=2)
    axes[0].set_xlabel('Epoch', fontsize=12)
    axes[0].set_ylabel('MSE Loss', fontsize=12)
    axes[0].set_title('SGD Training: Loss vs Epoch', fontsize=14)
    axes[0].grid(True, alpha=0.3)

    # Plot 2: Qdot vs dT with fitted line
    axes[1].scatter(dT_original, y, alpha=0.6, edgecolors='black',
                    linewidth=0.5, label='Training data points', color='blue')
    dT_line = np.linspace(dT_original.min(), dT_original.max(), 100)
    dT_line_normalized = (dT_line - means[0]) / stds[0]
    X_line = np.column_stack([np.ones(100), dT_line_normalized])
    y_line = X_line @ w_sgd

    axes[1].plot(dT_line, y_line, 'r-', linewidth=2, label='Fitted line')
    axes[1].set_xlabel(x_label, fontsize=12)
    axes[1].set_ylabel(y_label, fontsize=12)
    axes[1].set_title('Heat Loss vs Temperature Difference', fontsize=14)
    axes[1].legend()
    axes[1].grid(True, alpha=0.3)
#Indented
```

Listing 5 - Plotting pt.1

```
    # Plot 3: Predictions vs Actual
    axes[2].scatter(y, y_pred, alpha=0.6, edgecolors='black',
                    linewidth=0.5, label='Training data points', color='blue')
    min_val = min(y.min(), y_pred.min())
    max_val = max(y.max(), y_pred.max())
    axes[2].plot([min_val, max_val], [min_val, max_val], 'r--',
                 linewidth=2, label='Prediction')
    axes[2].set_xlabel(f'Actual {y_label}', fontsize=12)
    axes[2].set_ylabel(f'Predicted {y_label}', fontsize=12)
    axes[2].set_title('Predictions vs Actual Values', fontsize=14)
    axes[2].legend()
    axes[2].grid(True, alpha=0.3)

    plt.tight_layout()
    plt.savefig(your_path + 'LinearRegression_SGD_output.png', dpi=150,
bbox_inches='tight')
    plt.show()
#Indented
```

Listing 6 - Plotting pt.2

## 1.2 Console Output

```
DATA CHECK:
dT range: 1.05 to 21.93 °C
$\dot{Q}$ (W) range: 482.65 to 11507.79 W
Dataset shape: 24 samples, 1 features

Start of SGD Linear Regression:
Epoch 1/100, Loss: 19754458.888643
Epoch 10/100, Loss: 595389.836357
Epoch 20/100, Loss: 594480.745186
Epoch 30/100, Loss: 594339.488119
Epoch 40/100, Loss: 594665.175322
Epoch 50/100, Loss: 596306.570341
Epoch 60/100, Loss: 595375.779382
Epoch 70/100, Loss: 595024.150064
Epoch 80/100, Loss: 594531.145023
Epoch 90/100, Loss: 594669.893471
Epoch 100/100, Loss: 594472.436477

Final Loss (MSE): 594472.436477
Learned weights (normalized): [6402.42813965 3115.93548705]
Final RMSE: 771.020387

Plot saved as 'LinearRegression_SGD_output.png'
```

Listing 7 - Console Output: Data Ranges, Dataset Shape, Epochs, Loss, Weights, Plot path

### 1.3 Plots

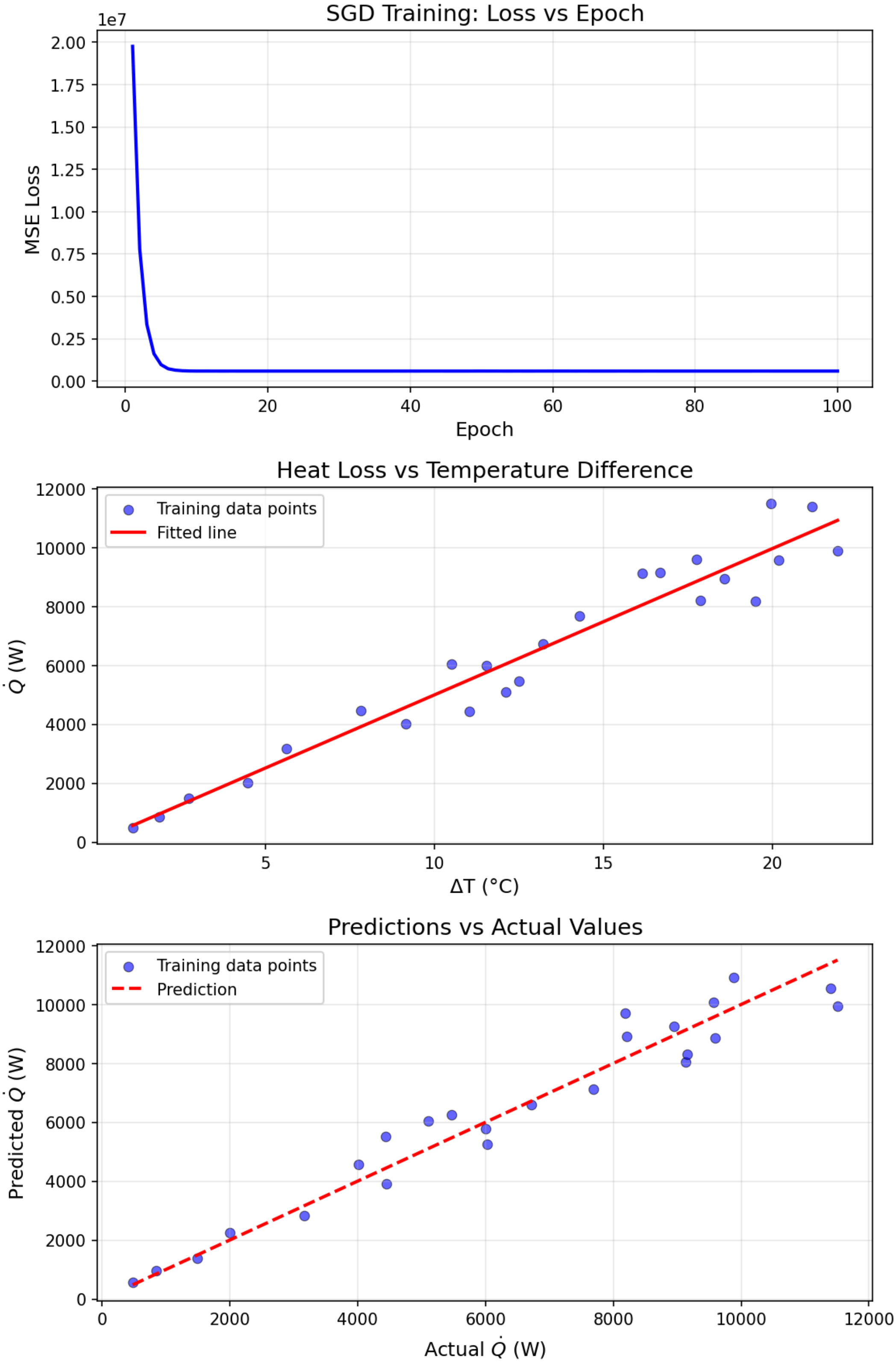


Figure 1 - SGD with MSE: (Top) Loss vs Epoch, (Middle) Heat Loss vs Temperature Difference, (Bottom) Prediction of Target (W) vs Actual Values from data file

# 2 Linear Regression using GD & MAE

## 2.1 Code: LinearRegression_MAE.py

The code produced for Task 2 can be found below:

```python
# Linear Regression using Mean Absolute Error (MAE) Loss
# AERO40041 Coursework 2 - Task 2
#
# This file was prepared by Juan Ignacio Doval Roque [10752534]
#
# NOTE: Plots have been hard coded for given data (Qdot vs dT), therefore, if other data is used,
# the plotting section may need to be adjusted accordingly. Also the path to the data file should be changed.

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

your_path = '' # Currently at root
data_file = "window_heat.csv"
y_label = '$\dot{Q}$ (W)'
x_label = 'ΔT (°C)'

def load_and_prep_data(filename):
    """
    Load dataset from CSV and prepare feature matrix X and target vector y.
    Adds a column of ones to X for the bias term.
    """
    data = pd.read_csv(filename)

    X = data.iloc[:, :-1].values
    y = data.iloc[:, -1].values.reshape(-1, 1)

    ones = np.ones((X.shape[0], 1))
    X = np.concatenate((ones, X), axis=1)

    return X, y
```

Listing 8 - Information, Imports, Paths, and Data preparation

```python
def mae_loss(X, y, w):
    """
    Compute Mean Absolute Error loss.

    MAE = (1/N) * sum(|y_pred - y|)

    MAE is more robust to outliers than MSE because it doesn't square errors.
    """
    N = X.shape[0]
    y_pred = X @ w
    error = y_pred - y
    loss = (1/N) * np.sum(np.abs(error))
    return loss
```

Listing 9 - MAE Loss function

```python
def mae_gradient(X, y, w):
    """
    Compute the gradient of MAE loss with respect to weights.

    The gradient of |x| is sign(x), where:
        sign(x) = +1 if x > 0
        sign(x) = -1 if x < 0
        sign(x) =  0 if x = 0

    Therefore:
        d(MAE)/dw_j = (1/N) * sum(sign(y_pred - y) * x_j)

    In matrix form:
        gradient = (1/N) * X^T @ sign(y_pred - y)
    """
    N = X.shape[0]
    y_pred = X @ w
    error = y_pred - y

    gradient = (1/N) * X.T @ np.sign(error)

    return gradient
```

Listing 10 - MSE gradient computation

```python
def gd_mae(X, y, learning_rate=0.01, num_iterations=1000):
    """
    Train linear regression using Gradient Descent with MAE loss.

    This is BATCH gradient descent: we use all samples for each update.

    Parameters:
    -----------
    X : numpy array of shape (N, D+1)
        Feature matrix with bias column
    y : numpy array of shape (N, 1)
        Target values
    learning_rate : float
        Step size for gradient updates
    num_iterations : int
        Number of gradient descent steps

    Returns:
    --------
    w : numpy array
        Learned weights
    loss_history : list
        MAE loss value at each iteration
    """
    N, D = X.shape

    w = np.zeros((D, 1))
    w[0] = np.mean(y)

    loss_history = []
    best_loss = float('inf')
    patience = 500
    no_improve = 0

    for iteration in range(num_iterations):
        gradient = mae_gradient(X, y, w)

        w = w - learning_rate * gradient

        current_loss = mae_loss(X, y, w)
        loss_history.append(current_loss)

        if current_loss < best_loss - 1e-6:
            best_loss = current_loss
            no_improve = 0
        else:
            no_improve += 1

        if no_improve >= patience:
            print(f"Early stopping at iteration {iteration + 1}")
            break

        if (iteration + 1) % 500 == 0 or iteration == 0:
            print(f"Iteration {iteration + 1}/{num_iterations}, MAE Loss:
{current_loss:.6f}")

    return w, loss_history

def predict(X, w):
    """Make predictions using learned weights."""
    return X @ w
```

Listing 11 - Gradient Descent Linear Regression function and Prediction function

```python
if __name__ == "__main__":
    X, y = load_and_prep_data(your_path + data_file)

    dT_original = X[:, 1].copy()

    print("DATA CHECK:")
    print(f"dT range: {dT_original.min():.2f} to {dT_original.max():.2f} °C")
    print(f"{y_label} range: {y.min():.2f} to {y.max():.2f} W")
    print(f"Dataset shape: {X.shape[0]} samples, {X.shape[1]-1} features")

    X_normalized = X.copy()
    means = X[:, 1:].mean(axis=0)
    stds = X[:, 1:].std(axis=0)
    stds[stds == 0] = 1
    X_normalized[:, 1:] = (X[:, 1:] - means) / stds

    print("\nTraining w/ Gradient Descent (MAE Loss):")

    w_mae, loss_history_mae = gd_mae(
        X_normalized, y,
        learning_rate=50.0,
        num_iterations=5000
    )

    print(f"\nFinal MAE Loss: {loss_history_mae[-1]:.6f}")
    print(f"Learned weights (normalized): {w_mae.flatten()}")

    y_pred_mae = predict(X_normalized, w_mae)

    final_mae = np.mean(np.abs(y - y_pred_mae))
    final_mse = np.mean((y - y_pred_mae) ** 2)
    final_rmse = np.sqrt(final_mse)
    print(f"\nFinal MAE: {final_mae:.6f}")
    print(f"Final MSE: {final_mse:.6f}")
    print(f"Final RMSE: {final_rmse:.6f}")
```

Listing 12 - Main Execution Block

```python
    fig, axes = plt.subplots(3, 1, figsize=(8, 12))

    # Plot 1: Loss vs Iteration
    axes[0].plot(range(1, len(loss_history_mae) + 1), loss_history_mae, 'b-', linewidth=2)
    axes[0].set_xlabel('Iteration', fontsize=12)
    axes[0].set_ylabel('MAE Loss', fontsize=12)
    axes[0].set_title('Gradient Descent with MAE: Loss vs Iteration', fontsize=14)
    axes[0].grid(True, alpha=0.3)

    # Plot 2: Qdot vs dT with fitted line
    axes[1].scatter(dT_original, y, alpha=0.6, edgecolors='black',
                    linewidth=0.5, label='Data points', color='blue')
    dT_line = np.linspace(dT_original.min(), dT_original.max(), 100)
    dT_line_normalized = (dT_line - means[0]) / stds[0]
    X_line = np.column_stack([np.ones(100), dT_line_normalized])
    y_line = X_line @ w_mae

    axes[1].plot(dT_line, y_line, 'r-', linewidth=2, label='Fitted line (MAE)')
    axes[1].set_xlabel(x_label, fontsize=12)
    axes[1].set_ylabel(y_label, fontsize=12)
    axes[1].set_title('Heat Loss vs Temperature Difference', fontsize=14)
    axes[1].legend()
    axes[1].grid(True, alpha=0.3)
#Indented
```

Listing 13 - Plotting pt.1

```python
    # Plot 3: Predictions vs Actual
    axes[2].scatter(y, y_pred_mae, alpha=0.6, edgecolors='black',
                    linewidth=0.5, label='Training data points', color='blue')
    min_val = min(y.min(), y_pred_mae.min())
    max_val = max(y.max(), y_pred_mae.max())
    axes[2].plot([min_val, max_val], [min_val, max_val], 'r--',
                 linewidth=2, label='Prediction')
    axes[2].set_xlabel(f'Actual {y_label}', fontsize=12)
    axes[2].set_ylabel(f'Predicted {y_label}', fontsize=12)
    axes[2].set_title('MAE Model: Predictions vs Actual Values', fontsize=14)
    axes[2].legend()
    axes[2].grid(True, alpha=0.3)

    plt.tight_layout()
    plt.savefig(your_path + 'LinearRegression_MAE_output.png', dpi=150,
bbox_inches='tight')
    plt.show()

    print(f"\nPlot saved as '{your_path}LinearRegression_MAE_output.png'")
#Indented
```

Listing 14 - Plotting pt.2

## 2.2 Console Output

```
DATA CHECK:
dT range: 1.05 to 21.93 °C
$\dot{Q}$ (W) range: 482.65 to 11507.79 W
Dataset shape: 24 samples, 1 features

Training w/ Gradient Descent (MAE Loss):
Iteration 1/5000, MAE Loss: 2734.424588
Iteration 500/5000, MAE Loss: 645.291521
Early stopping at iteration 941

Final MAE Loss: 645.346541
Learned weights (normalized): [6523.29413845 3217.08664574]

Final MAE: 645.346541
Final MSE: 617986.882721
Final RMSE: 786.121417

Plot saved as 'LinearRegression_MAE_output.png'
```

Listing 15 - Console Output: Data Ranges, Dataset Shape, Epochs, Loss, Weights, Plot path

## 2.3 Plots

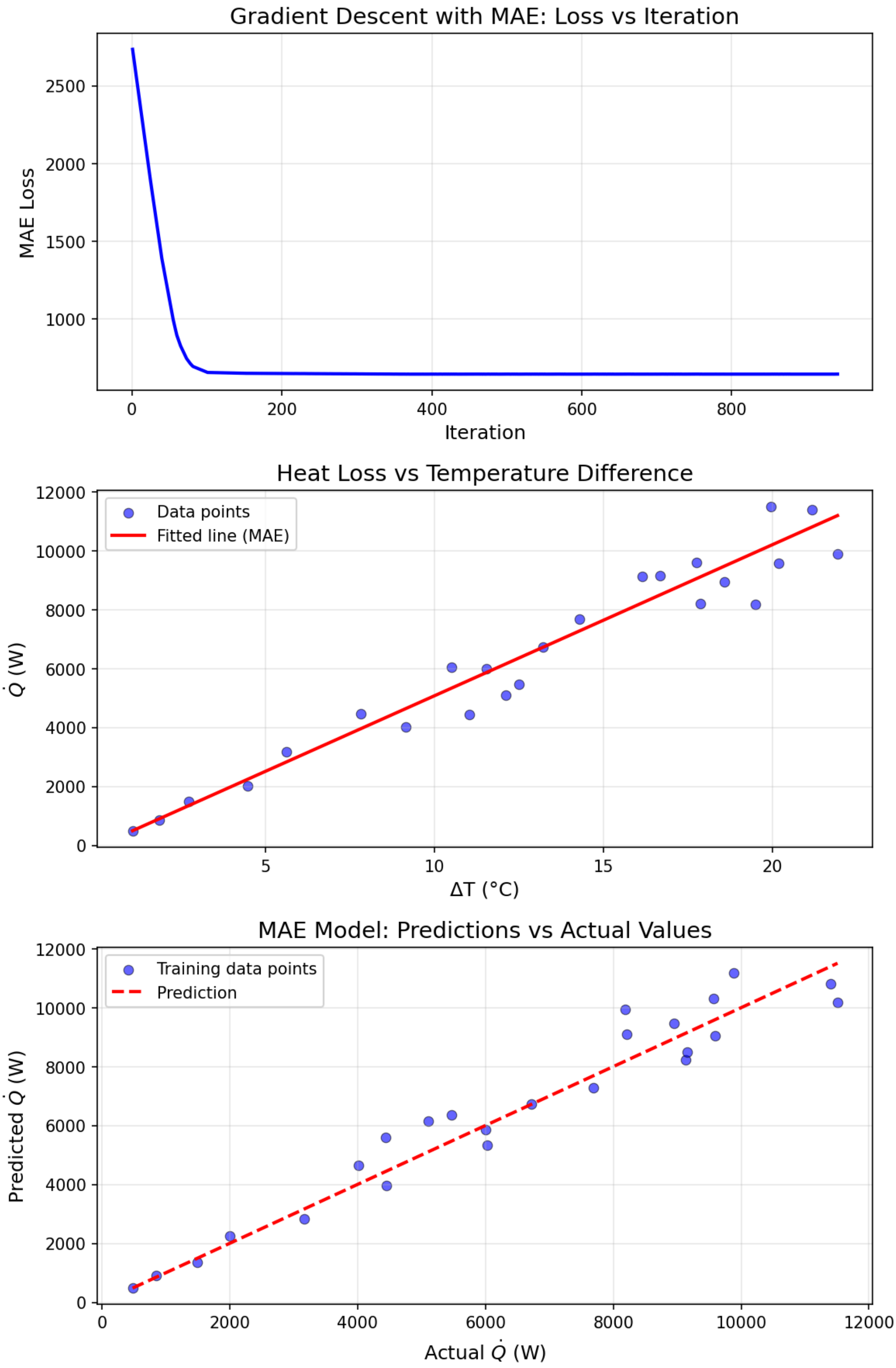


Figure 2 - GD with MAE: (Top) Loss vs Epoch, (Middle) Heat Loss vs Temperature Difference, (Bottom) Prediction of Target (W) vs Actual Values from data file

# 3 Contributions

Juan created and ran the code for Tasks 1 & 2; Linear Regression. This pdf was created by Juan.

Sacha created and ran the code for Task 3; Neural Network for classification. Sacha provides his pdf separately.