```python
def mean_squared_error(y_true, y_pred):
  return np.mean((y_true - y_pred) ** 2)

def mean_absolute_error(y_true, y_pred):
  return np.mean(np.abs(y_true - y_pred))

import numpy as np
y_true = np.array([1, 0, 1, 0])
y_pred = np.array([0.9, 0.1, 0.8, 0.3])
print("MSE:", mean_squared_error(y_true, y_pred))
print("MAE:", mean_absolute_error(y_true, y_pred))
```

```
MSE: 0.037499999999999999
MAE: 0.175
```

```python
def binary_cross_entropy(y_true, y_pred):
# Clip predictions to prevent log(0)
  y_pred = np.clip(y_pred, 1e-15, 1 - 1e-15)
  return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 -
y_pred))

print("Binary Cross-Entropy:", binary_cross_entropy(y_true, y_pred))
```

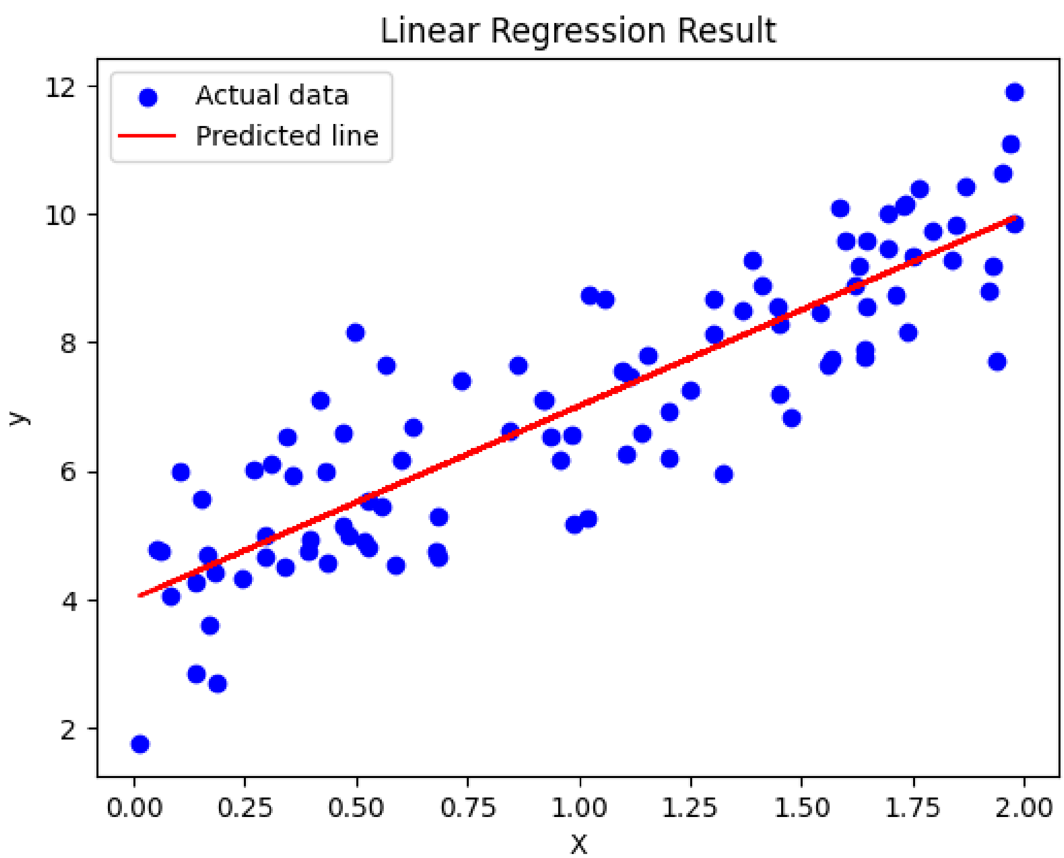
```
Binary Cross-Entropy: 17.26978799617044
```

```python
class LinearRegression:
    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.theta = None

    def fit(self, X, y):
        m = len(y)
        X_b = np.c_[np.ones((m, 1)), X]  # Add bias term
        self.theta = np.random.randn(2, 1)  # Random initialization

        for iteration in range(self.n_iterations):
            y_pred = X_b.dot(self.theta)
            gradients = 2/m * X_b.T.dot(y_pred - y)
            self.theta -= self.learning_rate * gradients
    def predict(self, X):
        X_b = np.c_[np.ones((len(X), 1)), X]  # Add bias term
        return X_b.dot(self.theta)


# Generate synthetic data for testing
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
# Train the model
model = LinearRegression(learning_rate=0.1, n_iterations=1000)
model.fit(X, y)
```

```
y_pred = model.predict(X)

y_pred

array([[6.95867313],
       [5.55784819],
       [5.03268131],
       [5.42344461],
       [5.88218218],
       [5.69996301],
       [8.69996135],
       [9.25016529],
       [5.50059219],
       [9.2855695 ],
       [4.51937017],
       [9.20333699],
       [6.59185506],
       [5.58589989],
       [6.77086405],
       [7.42032303],
       [4.05445206],
       [6.7545722 ],
       [7.90817007],
       [5.04426698],
       [5.42355594],
       [5.77347619],
       [8.62922954],
       [4.57187918],
       [7.18280602],
       [4.74174123],
       [8.85884954],
       [8.89030681],
       [5.18663491],
       [4.16995045],
       [4.82405247],
       [8.93688952],
       [9.53872983],
       [9.78630927],
       [7.7527572 ],
       [9.81695496],
       [8.33365688],
       [9.76516085],
       [7.07632048],
       [8.79123097],
       [7.4664544 ],
       [8.92197924],
       [7.34543062],
       [4.9018413 ],
       [9.08308922],
       [6.21478645],
```

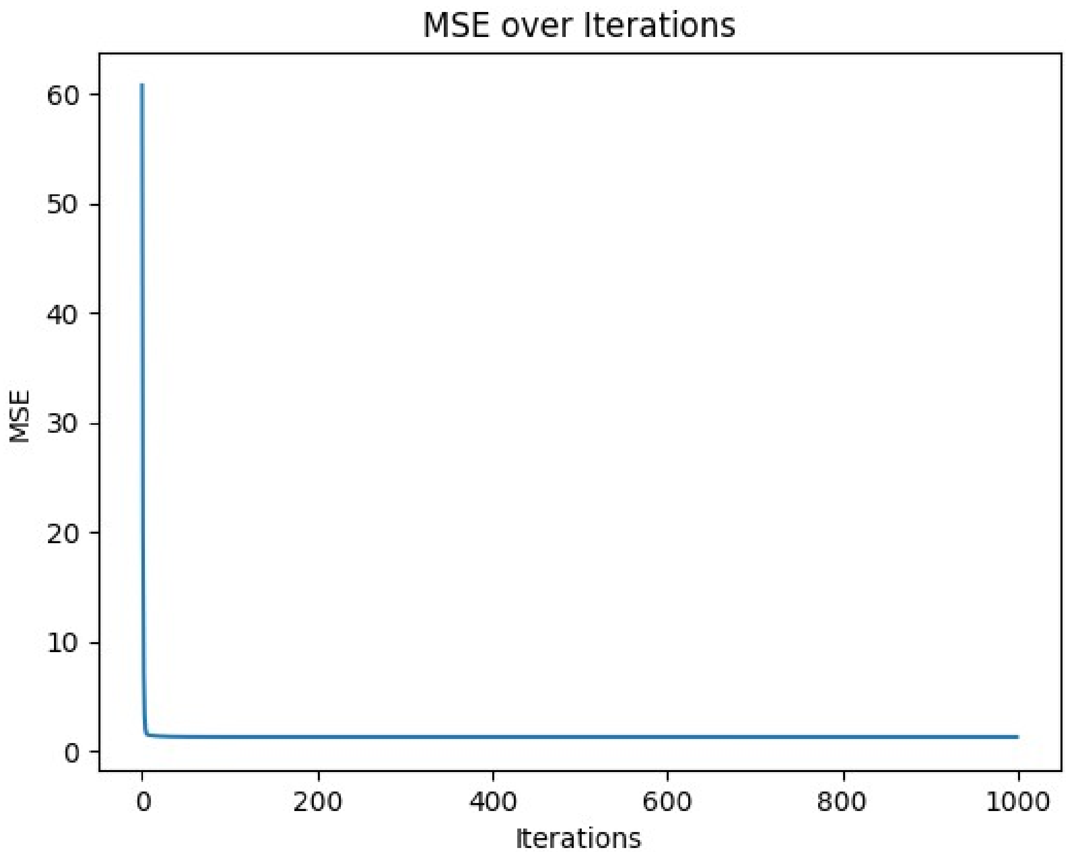
```
        [8.74984961],
        [5.31660092],
        [8.3472147 ],
        [5.2013031 ],
        [8.34927612],
        [5.2953823 ],
        [9.93055323],
        [8.92738854],
        [7.61314316],
        [4.4303852 ],
        [4.89248028],
        [6.81075419],
        [8.94280325],
        [4.50713554],
        [6.05756296],
        [9.50780586],
        [6.04599748],
        [8.10591381],
        [8.16423959],
        [4.32257977],
        [9.93445953],
        [9.18365111],
        [6.53620684],
        [9.20623224],
        [7.90512826],
        [5.80455493],
        [5.26072361],
        [4.46643294],
        [8.67911213],
        [4.93670917],
        [7.0556674 ],
        [9.13340568],
        [8.23674442],
        [9.37628474],
        [7.29494673],
        [5.67395697],
        [9.90811864],
        [6.88167367],
        [4.42322404],
        [9.85790761],
        [6.05574669],
        [7.32091781],
        [5.07675957],
        [6.97295386],
        [5.58420656],
        [7.97331631],
        [9.59969769],
        [4.25142529],
        [8.43555769],
```

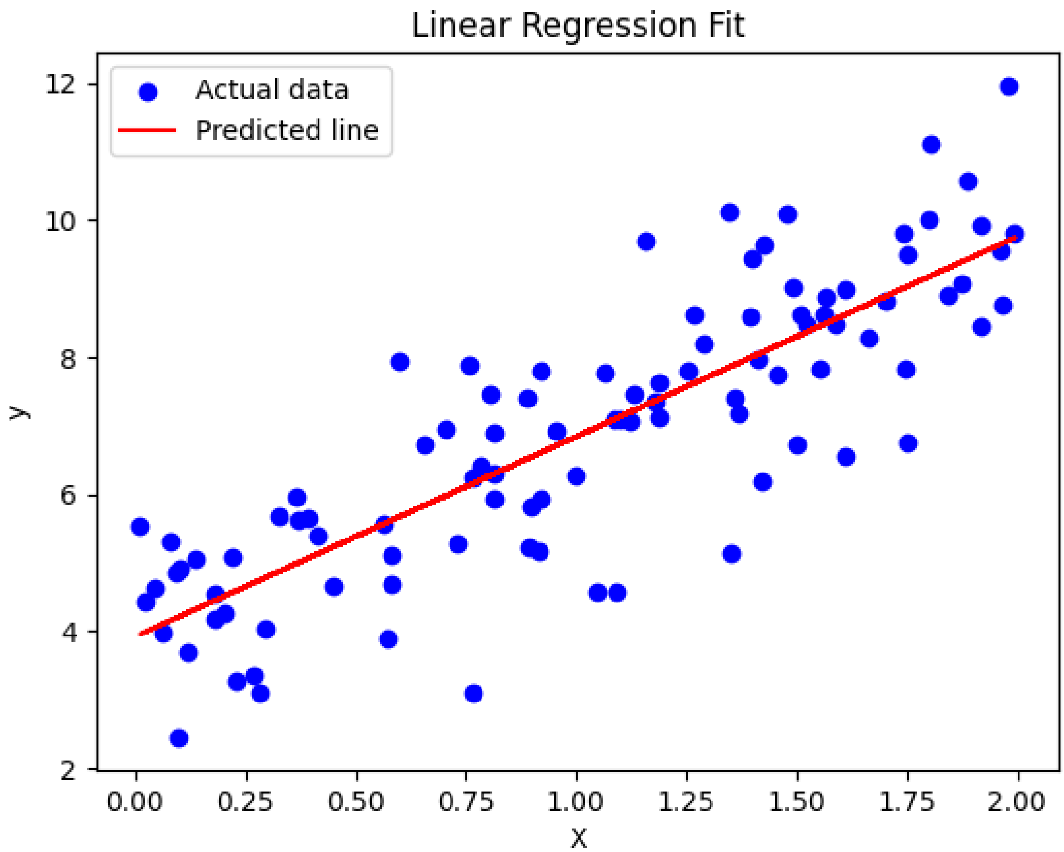
```
       [9.0774013 ],
       [7.60687441],
       [4.56262801],
       [4.19376151],
       [5.46290705]])

import matplotlib.pyplot as plt

# Plotting the results
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X, y_pred, color='red', label='Predicted line')
plt.title('Linear Regression Result')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

# Define the Linear Regression class
class LinearRegression:
    def __init__(self, learning_rate=0.01, n_iterations=1000):
```

```
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.theta = None

    def fit(self, X, y):
        m = len(y)
        X_b = np.c_[np.ones((m, 1)), X]  # Add bias term
        self.theta = np.random.randn(2, 1)  # Random initialization
        self.mse_history = []  # Track MSE history

        for iteration in range(self.n_iterations):
            y_pred = X_b.dot(self.theta)
            mse = np.mean((y - y_pred) ** 2)  # Calculate Mean Squared
Error
            self.mse_history.append(mse)  # Track MSE

            gradients = 2/m * X_b.T.dot(y_pred - y)
            self.theta -= self.learning_rate * gradients

    def predict(self, X):
        X_b = np.c_[np.ones((len(X), 1)), X]  # Add bias term
        return X_b.dot(self.theta)

# Generate synthetic data for testing
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Train the model
model = LinearRegression(learning_rate=0.1, n_iterations=1000)
model.fit(X, y)

# Predict values
y_pred = model.predict(X)

# Plot MSE history
plt.plot(model.mse_history)
plt.title('MSE over Iterations')
plt.xlabel('Iterations')
plt.ylabel('MSE')
plt.show()

# Plot original data and predictions
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X, y_pred, color='red', label='Predicted line')
plt.title('Linear Regression Fit')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

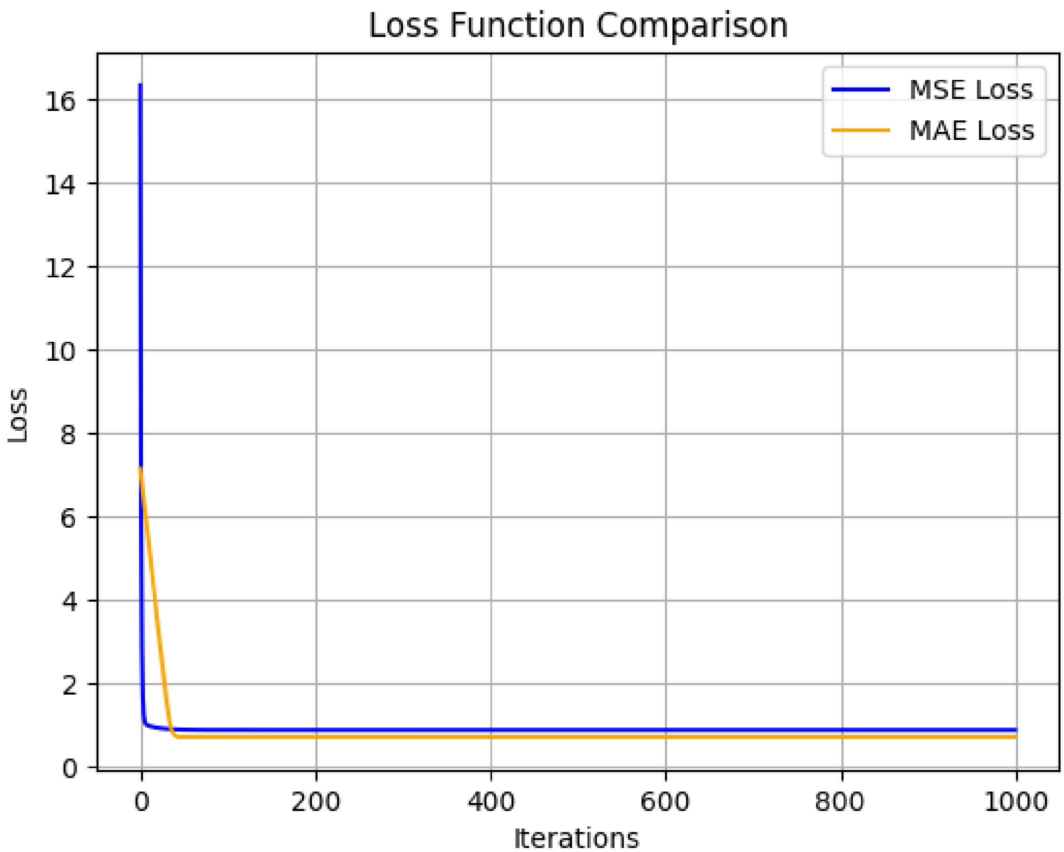

MSE over Iterations
MSE
0
10
20
30
40
50
60
Iterations
0
200
400
600
800
1000



task 1

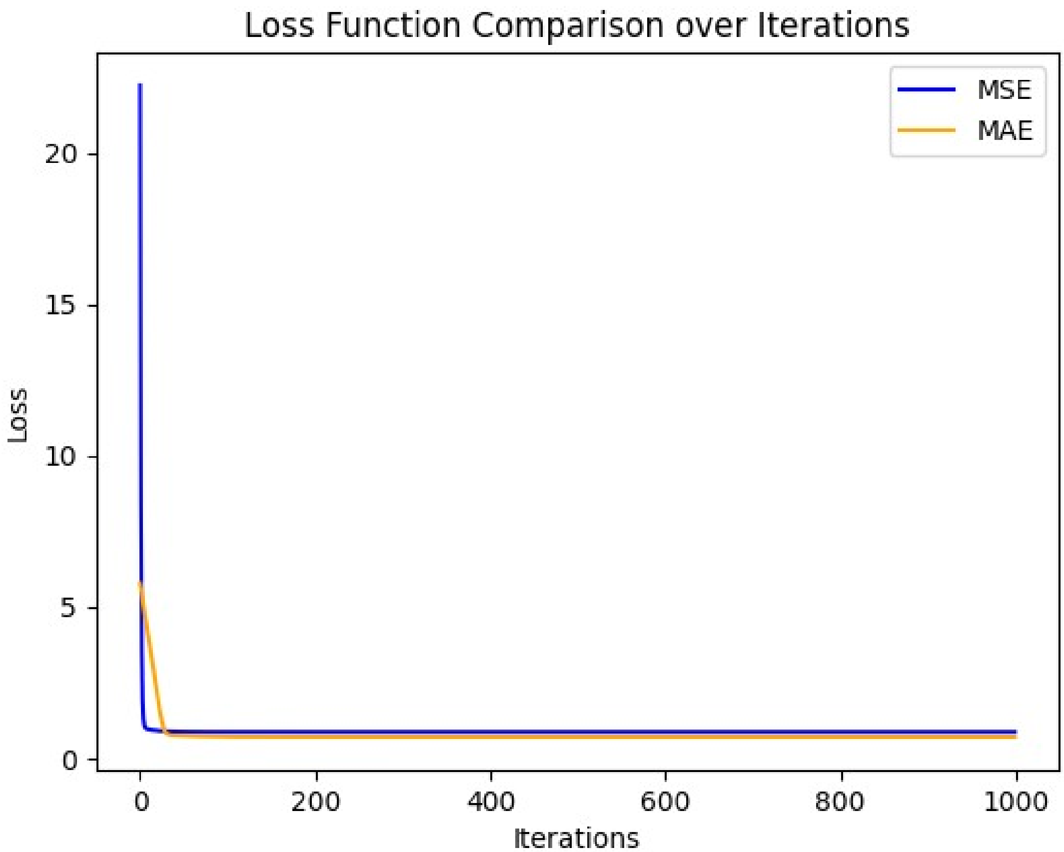
```
class LinearRegression:
    def __init__(self, learning_rate=0.01, n_iterations=1000,
loss_function='mse'):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.loss_function = loss_function
        self.theta = None
        self.loss_history = []

    def compute_loss(self, y, y_pred):
        if self.loss_function == 'mse':
            return np.mean((y - y_pred) ** 2)  # MSE
        elif self.loss_function == 'mae':
            return np.mean(np.abs(y - y_pred))  # MAE

    def fit(self, X, y):
        m = len(y)
        X_b = np.c_[np.ones((m, 1)), X]  # Add bias term
        self.theta = np.random.randn(2, 1)  # Random initialization
```

```
        for iteration in range(self.n_iterations):
            y_pred = X_b.dot(self.theta)
            loss = self.compute_loss(y, y_pred)
            self.loss_history.append(loss)

            if self.loss_function == 'mse':
                gradients = 2/m * X_b.T.dot(y_pred - y)
            elif self.loss_function == 'mae':
                gradients = X_b.T.dot(np.sign(y_pred - y)) / m

            # Ensure correct shape for gradients and theta update
            self.theta -= self.learning_rate *
gradients.reshape(self.theta.shape)

    def predict(self, X):
        X_b = np.c_[np.ones((len(X), 1)), X]  # Add bias term
        return X_b.dot(self.theta)


# Generate synthetic data for testing
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)



# Train models with both loss functions
model_mse = LinearRegression(learning_rate=0.1, n_iterations=1000,
loss_function='mse')
model_mse.fit(X, y)

model_mae = LinearRegression(learning_rate=0.1, n_iterations=1000,
loss_function='mae')
model_mae.fit(X, y)

#compare performance



# Plotting loss history
plt.plot(model_mse.loss_history, color='blue', label='MSE Loss')
plt.plot(model_mae.loss_history, color='orange', label='MAE Loss')
plt.title('Loss Function Comparison')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.grid()
plt.show()
```

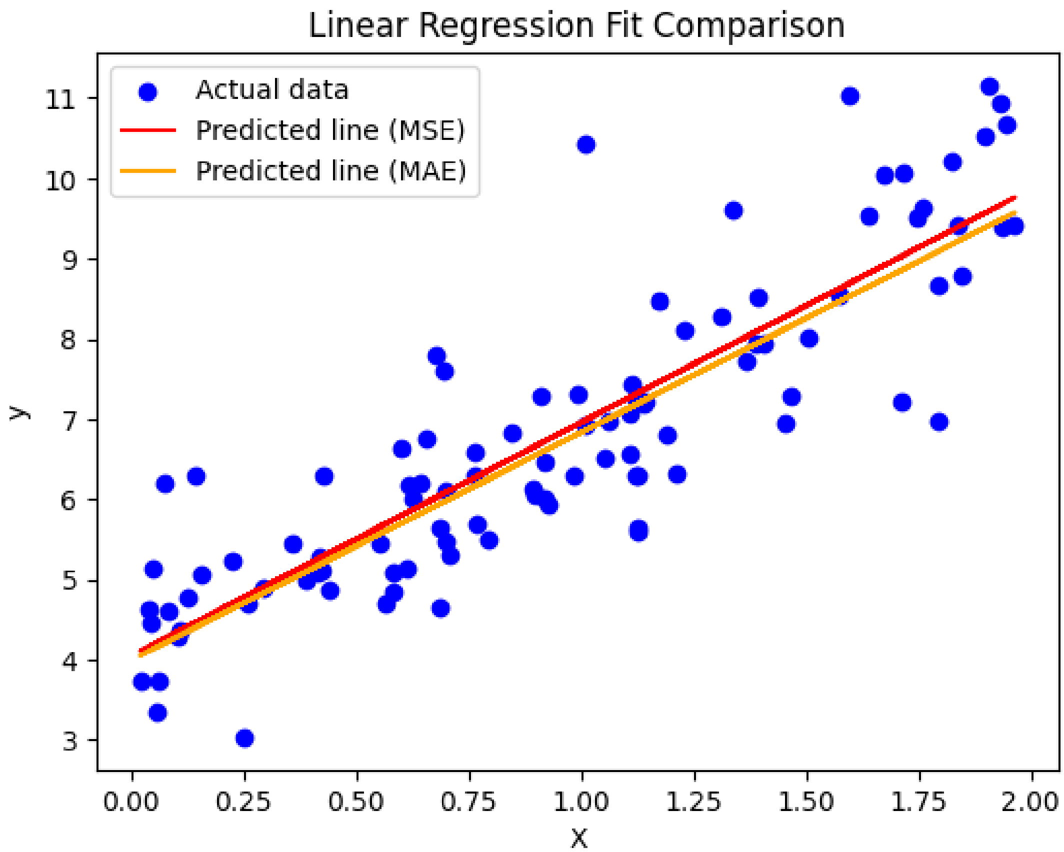



```
import numpy as np

class LinearRegression:
    def __init__(self, learning_rate=0.01, n_iterations=1000,
loss_function='mse'):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.loss_function = loss_function
        self.theta = None
        self.mse_history = []
        self.mae_history = []

    def fit(self, X, y):
        self.theta = np.random.randn(2, 1)  # Random initialization

        for iteration in range(self.n_iterations):
            y_pred = self.predict(X)

            if self.loss_function == 'mse':
                loss = (1/m) * np.sum((y_pred - y) ** 2)
                self.mse_history.append(loss)
                gradients = (2/m) * X_b.T.dot(y_pred - y)
            elif self.loss_function == 'mae':
```

```python
                loss = (1/m) * np.sum(np.abs(y_pred - y))
                self.mae_history.append(loss)
                gradients = (1/m) * X_b.T.dot(np.sign(y_pred - y))

            self.theta -= self.learning_rate * gradients

    def predict(self, X):
        X_b = np.c_[np.ones((len(X), 1)), X]  # Add bias term
        return X_b.dot(self.theta)

# Train with MSE
model_mse = LinearRegression(learning_rate=0.1, n_iterations=1000,
loss_function='mse')
model_mse.fit(X, y)
y_pred_mse = model_mse.predict(X)

# Train with MAE
model_mae = LinearRegression(learning_rate=0.1, n_iterations=1000,
loss_function='mae')
model_mae.fit(X, y)
y_pred_mae = model_mae.predict(X)

# Plot MSE and MAE history
plt.plot(model_mse.mse_history, label='MSE', color='blue')
plt.plot(model_mae.mae_history, label='MAE', color='orange')
plt.title('Loss Function Comparison over Iterations')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
# Plot original data and predictions for both models
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X, y_pred_mse, color='red', label='Predicted line (MSE)')
plt.plot(X, y_pred_mae, color='orange', label='Predicted line (MAE)')
plt.title('Linear Regression Fit Comparison')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```




```
import numpy as np
x=np.random.randn(2, 1)
```

```
array([[2.23027983],
       [0.9237355 ]])
```