Assignment 2: Multi-Variable Linear Regression

```
import numpy as np
import pandas as pd
from keras.datasets import boston_housing
import matplotlib.pyplot as plt
import seaborn as sns
```

```
(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()

print("Shape of training data: ", train_data.shape)
print("Shape of testing data: ", test_data.shape)
print("Shape of training targets: ", train_targets.shape)
print("Shape of testing targets: ", test_targets.shape)

feature_variances = np.var(train_data, axis=0)
max_var_idx, min_var_idx = np.argmax(feature_variances), np.argmin(feature_variances)

print("Index of feature with highest variance: ", max_var_idx, " with value:", feature_variances[max_var_idx])
print("Index of feature with lowest variance: ", min_var_idx, " with value:", feature_variances[min_var_idx])
```

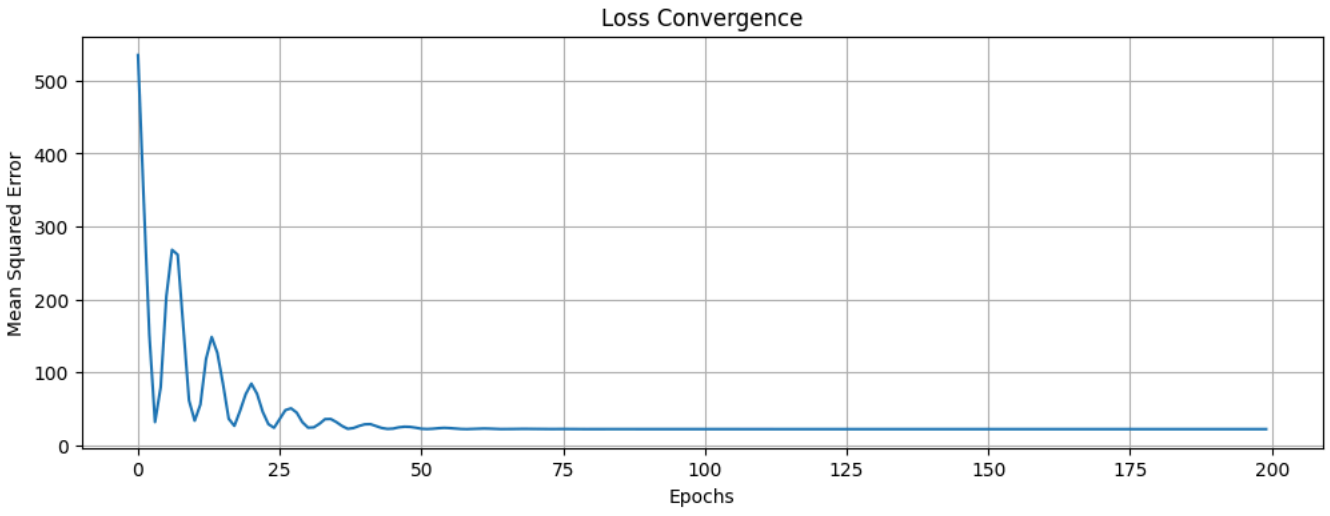
```
Shape of training data:  (404, 13)
Shape of testing data:  (102, 13)
Shape of training targets:  (404,)
Shape of testing targets:  (102,)
Index of feature with highest variance:  9  with value: 27611.97237403192
Index of feature with lowest variance:  4  with value: 0.013723618009263812
```

```
mean = train_data.mean(axis=0)
std = train_data.std(axis=0)
train_data_norm = (train_data - mean) / std
test_data_norm = (test_data - mean) / std
```

```
# Add bias term
X_train = np.hstack([np.ones((train_data_norm.shape[0], 1)), train_data_norm])
X_test = np.hstack([np.ones((test_data_norm.shape[0], 1)), test_data_norm])
Y_train = train_targets.reshape(-1, 1)
Y_test = test_targets.reshape(-1, 1)

# Initialize parameters
np.random.seed(42)
W = np.random.randn(X_train.shape[1], 1)

# Hyperparameters
learning_rate = 0.1
momentum = 0.9
epochs = 200
velocity = np.zeros_like(W)

loss_history = []

# Gradient Descent
for epoch in range(epochs):
    Y_pred = X_train @ W # Prediction
    error = Y_pred - Y_train # Loss
    loss = (1 / len(Y_train)) * np.sum(error**2) # MSE
    loss_history.append(loss)
    gradient = (2 / len(Y_train)) * X_train.T @ error # Gradient computation
    velocity = momentum * velocity - learning_rate * gradient # Velocity update
    W += velocity

    if epoch % 25 == 0:
        print(f"Epoch {epoch}: Loss = {loss:.4f}")
```

```
Epoch 0: Loss = 535.0718
Epoch 25: Loss = 36.0553
Epoch 50: Loss = 22.4927
Epoch 75: Loss = 22.1959
Epoch 100: Loss = 22.0093
Epoch 125: Loss = 22.0050
Epoch 150: Loss = 22.0049
Epoch 175: Loss = 22.0048
```

```
plt.figure(figsize=(12, 4))
plt.plot(loss_history)
plt.title("Loss Convergence")
plt.xlabel("Epochs")
plt.ylabel("Mean Squared Error")
plt.grid(True)
plt.show()
```




```
def plot_regression_line(X_raw, X_norm, feature_idx, feature_name):
    X_feature_raw = X_raw[:, feature_idx]
    plt.figure(figsize=(12, 5))
    plt.scatter(X_feature_raw, Y_train, color="blue", alpha=0.5, label="True Values")
    x_range_raw = np.linspace(X_feature_raw.min(), X_feature_raw.max(), 100)
    x_range_norm = (x_range_raw - mean[feature_idx]) / std[feature_idx]

    bias = W[0][0]
```

```
    weight = W[feature_idx + 1][0]
    y_pred = bias + weight * x_range_norm

    plt.plot(x_range_raw, y_pred, color="red", label="Regression Line")
    plt.title(f"Regression on Feature {feature_idx} ({feature_name})")
    plt.xlabel(f"{feature_name} (Feature {feature_idx})")
    plt.ylabel("House Price ($1000s)")
    plt.legend()
    plt.grid(True)
    plt.show()
    print('\n')

feature_names = [
    "CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE",
    "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT"
]

plot_regression_line(train_data, train_data_norm, max_var_idx, feature_names[max_var_idx])
plot_regression_line(train_data, train_data_norm, min_var_idx, feature_names[min_var_idx])
```