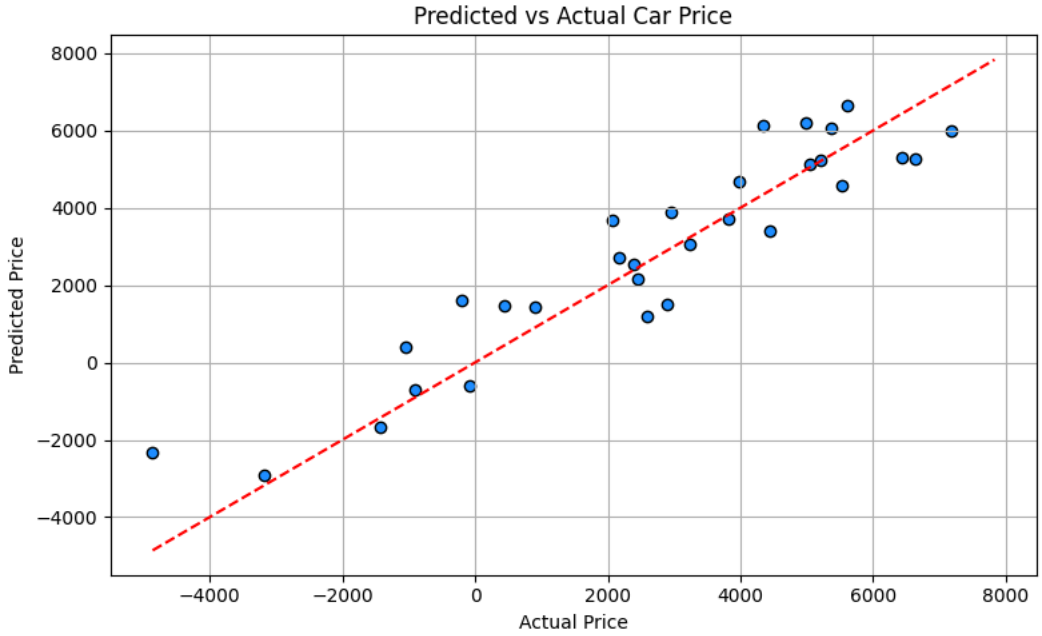
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Step 1: Create synthetic dataset
np.random.seed(42)
n_samples = 100

# Synthetic features
engine_size = np.random.randint(100, 300, size=n_samples)
curb_weight = np.random.randint(2000, 4000, size=n_samples)
horsepower = np.random.randint(70, 250, size=n_samples)
city_mpg = np.random.randint(15, 35, size=n_samples)
highway_mpg = np.random.randint(20, 45, size=n_samples)

# Synthetic price (target variable)
price = (engine_size * 20 + curb_weight * 0.5 + horsepower * 30 -
         city_mpg * 200 - highway_mpg * 100 + np.random.normal(0, 1000, n_samples))

# Create DataFrame
df = pd.DataFrame({
    'engine_size': engine_size,
    'curb_weight': curb_weight,
    'horsepower': horsepower,
    'city_mpg': city_mpg,
    'highway_mpg': highway_mpg,
    'price': price
})

# Step 2: Train-test split (70-30)
X = df[['engine_size', 'curb_weight', 'horsepower', 'city_mpg', 'highway_mpg']]
y = df['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 3: Train Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)

# Step 4: Predictions
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Step 5: Evaluation
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)

train_mae = mean_absolute_error(y_train, y_train_pred)
test_mae = mean_absolute_error(y_test, y_test_pred)

r2 = r2_score(y_test, y_test_pred)

# Step 6: Output results
print("🧮 Hypothesis Function:")
print(f"Price = {model.intercept_:.2f}", end="")
for i, col in enumerate(X.columns):
    print(f" + ({model.coef_[i]:.2f} * {col})", end="")
print("\n")

print(f"📉 Training MSE: {train_mse:.2f}")
print(f"📉 Testing MSE: {test_mse:.2f}")
print(f"📉 Training MAE: {train_mae:.2f}")
print(f"📉 Testing MAE: {test_mae:.2f}")
print(f"📈 R² Score on Test Set: {r2:.4f}")

# Step 7: Plot - Predicted vs Actual Price
plt.figure(figsize=(8, 5))
plt.scatter(y_test, y_test_pred, color='dodgerblue', edgecolor='black')
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', linestyle='--')
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Predicted vs Actual Car Price")
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```

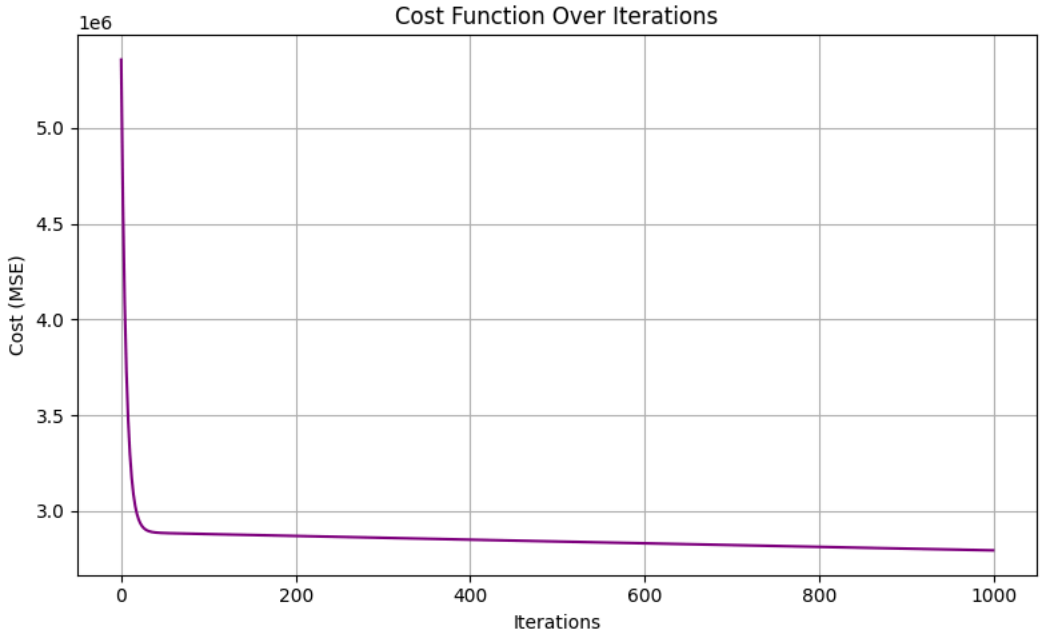
```
Hypothesis Function:
Price = 2031.73 + (16.25 * engine_size) + (0.45 * curb_weight) + (29.35 * horsepower) + (-206.84 * city_mpg) + (-119.40 * highway_mpg)

Training MSE: 877157.47
Testing MSE: 1154539.97
Training MAE: 748.03
Testing MAE: 878.98
R² Score on Test Set: 0.8668
```




```
import random
import matplotlib.pyplot as plt

# Step 1: Generate synthetic data
random.seed(42)

def generate_data(n_samples=100):
    data = []
    for _ in range(n_samples):
        engine_size = random.randint(100, 300)
        curb_weight = random.randint(2000, 4000)
        horsepower = random.randint(70, 250)
        city_mpg = random.randint(15, 35)
        highway_mpg = random.randint(20, 45)

        noise = random.gauss(0, 1000)
        price = (engine_size * 20 + curb_weight * 0.5 + horsepower * 30 -
                 city_mpg * 200 - highway_mpg * 100 + noise)

        features = [1, engine_size, curb_weight, horsepower, city_mpg, highway_mpg]  # 1 for intercept
        data.append((features, price))

    return data

# Generate data
dataset = generate_data()

# Step 2: Initialize weights
n_features = len(dataset[0][0])  # Including intercept
weights = [0.0 for _ in range(n_features)]

# Step 3: Define helper functions
def predict(features, weights):
    return sum([f * w for f, w in zip(features, weights)])

def compute_cost(dataset, weights):
    total_error = 0
    for features, actual_price in dataset:
```

```
            predicted = predict(features, weights)
            error = predicted - actual_price
            total_error += error ** 2
        return total_error / (2 * len(dataset))  # Mean Squared Error

    # Step 4: Gradient Descent
    alpha = 0.00000001  # Learning rate
    iterations = 1000
    cost_history = []

    for it in range(iterations):
        gradients = [0.0 for _ in range(n_features)]

        # Compute gradient for each weight
        for features, actual_price in dataset:
            prediction = predict(features, weights)
            error = prediction - actual_price
            for j in range(n_features):
                gradients[j] += error * features[j]

        # Update weights
        m = len(dataset)
        for j in range(n_features):
            weights[j] -= alpha * gradients[j] / m

        # Track cost
        cost = compute_cost(dataset, weights)
        cost_history.append(cost)

        if it % 100 == 0 or it == iterations - 1:
            print(f"Iteration {it:4d}: Cost = {cost:.2f}")

    # Step 5: Plot cost function
    plt.figure(figsize=(8, 5))
    plt.plot(range(iterations), cost_history, color='purple')
    plt.xlabel("Iterations")
    plt.ylabel("Cost (MSE)")
    plt.title("Cost Function Over Iterations")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    # Step 6: Output final weights
    print("\n📌 Final Learned Weights:")
    feature_names = ["intercept", "engine_size", "curb_weight", "horsepower", "city_mpg", "highway_mpg"]
    for name, weight in zip(feature_names, weights):
        print(f"{name:15s}: {weight:.4f}")
```

```
Iteration    0: Cost = 5354769.00
Iteration  100: Cost = 2880481.49
Iteration  200: Cost = 2870609.37
Iteration  300: Cost = 2860847.74
Iteration  400: Cost = 2851195.30
Iteration  500: Cost = 2841650.79
Iteration  600: Cost = 2832212.97
Iteration  700: Cost = 2822880.61
Iteration  800: Cost = 2813652.48
Iteration  900: Cost = 2804527.40
Iteration  999: Cost = 2795593.91
```




```
📌 Final Learned Weights:
intercept      : 0.0005
engine_size    : 0.7615
curb_weight    : 0.7424
horsepower     : 0.7106
city_mpg       : -0.0329
highway_mpg    : -0.0138
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Step 1: Generate synthetic dataset
np.random.seed(42)
n_samples = 150

# Features
engine_size = np.random.randint(100, 300, size=n_samples)
curb_weight = np.random.randint(2000, 4000, size=n_samples)
horsepower = np.random.randint(70, 250, size=n_samples)
price = engine_size * 20 + curb_weight * 0.5 + horsepower * 30 + np.random.normal(0, 1000, n_samples)

# Targets (city and highway mpg)
city_mpg = 40 - (engine_size * 0.02 + horsepower * 0.03 + np.random.normal(0, 1, n_samples))
highway_mpg = 50 - (engine_size * 0.015 + horsepower * 0.025 + np.random.normal(0, 1, n_samples))

# Create DataFrame
df = pd.DataFrame({
    'engine_size': engine_size,
    'curb_weight': curb_weight,
    'horsepower': horsepower,
    'price': price,
    'city_mpg': city_mpg,
    'highway_mpg': highway_mpg
})
```

```
# Step 2: Define features and targets
X = df[['engine_size', 'curb_weight', 'horsepower', 'price']]
Y = df[['city_mpg', 'highway_mpg']]  # Multiple outputs

# Train-test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=42)

# Step 3: Train multivariate linear regression model
model = LinearRegression()
model.fit(X_train, Y_train)

# Predict on test set
Y_pred = model.predict(X_test)

# Step 4: Report MSE for each output
mse_city = mean_squared_error(Y_test['city_mpg'], Y_pred[:, 0])
mse_highway = mean_squared_error(Y_test['highway_mpg'], Y_pred[:, 1])

print("📉 Mean Squared Error (MSE):")
print(f" - City MPG: {mse_city:.2f}")
print(f" - Highway MPG: {mse_highway:.2f}")

# Step 5: Visualizations

# City MPG plot
plt.figure(figsize=(6, 4))
plt.scatter(Y_test['city_mpg'], Y_pred[:, 0], color='green', edgecolors='black')
plt.plot([Y_test['city_mpg'].min(), Y_test['city_mpg'].max()],
         [Y_test['city_mpg'].min(), Y_test['city_mpg'].max()], 'r--')
plt.xlabel("Actual City MPG")
plt.ylabel("Predicted City MPG")
plt.title("Actual vs Predicted: City MPG")
plt.grid(True)
plt.tight_layout()
plt.show()

# Highway MPG plot
plt.figure(figsize=(6, 4))
plt.scatter(Y_test['highway_mpg'], Y_pred[:, 1], color='blue', edgecolors='black')
plt.plot([Y_test['highway_mpg'].min(), Y_test['highway_mpg'].max()],
         [Y_test['highway_mpg'].min(), Y_test['highway_mpg'].max()], 'r--')
plt.xlabel("Actual Highway MPG")
plt.ylabel("Predicted Highway MPG")
plt.title("Actual vs Predicted: Highway MPG")
plt.grid(True)
plt.tight_layout()
plt.show()

# Step 6: Output theta (coefficients) for each target
print("\n📌 Model Coefficients (Theta values):")
theta_df = pd.DataFrame(model.coef_.T, index=X.columns, columns=['city_mpg', 'highway_mpg'])
print(theta_df)
```

```
Mean Squared Error (MSE):
- City MPG: 0.90
- Highway MPG: 1.02
```