```
# ANDREW JOYNER
# 801293231
# HOMEWORK 2
```

PROBLEM 1.A

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
url = "https://raw.githubusercontent.com/HamedTabkhi/Intro-to-ML/main/Dataset/Housing.csv"
df = pd.read_csv(url)
display(df.head())
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no

```
print("\nColumn names:")
print(df.columns.tolist())
```

```
Column names:
['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'p
```

```
# prepare date for linear regression

features = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']
target = 'price'

X = df[features].values
y = df[target].values

print("first five feature rows")
display(X[:5])
print("\nfirst fice target values:")
display(y[:5])
```

```
first five feature rows
array([[7420,    4,    2,    3,    2],
       [8960,    4,    4,    4,    3],
       [9960,    3,    2,    2,    2],
       [7500,    4,    2,    2,    3],
       [7420,    4,    1,    2,    2]])

first fice target values:
array([13300000, 12250000, 12250000, 12215000, 11410000])
```

```
# 80% train, 20% test no scaling for problem 1
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) #arbitrary random state 42 (got it off t

print("Raw (unscaled) features; first five rows")
print(X_train[:5])

# split for validation set
X_train_final, X_val, y_train_final, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42
)

print(f"\nfinal training set: {X_train_final.shape}")
print(f"validation set: {X_val.shape}")
print(f"test set: {X_test.shape}")
```

```
Raw (unscaled) features; first five rows
[[6000    3    2    4    1]
 [7200    3    2    1    3]
 [3816    2    1    1    2]
 [2610    3    1    2    0]
 [3750    3    1    2    0]]

final training set: (348, 5)
validation set: (88, 5)
test set: (109, 5)
```

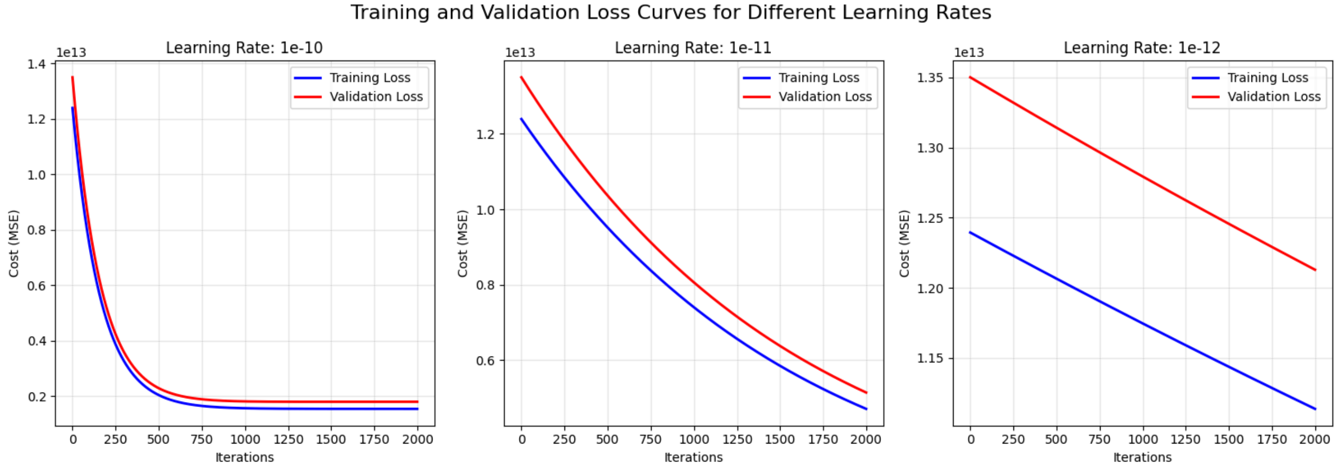
```
class LinearRegressionGD:

    def __init__(self, learning_rate=0.01, max_iterations=2000):
        self.learning_rate = learning_rate  # step size
        self.max_iterations = max_iterations
        self.costs_train = []  # training cost
        self.costs_val = []    # validation cost

    def add_bias(self, X):
        """adds bias terms to features"""
        return np.c_[np.ones(X.shape[0]), X]

    def compute_cost(self, X, y, theta):
        """how much cost..."""
        m = X.shape[0]
        predictions = X.dot(theta)  # predict using params
        cost = (1/(2*m)) * np.sum((predictions - y)**2)  # calculate error
        return cost

    def compute_gradients(self, X, y, theta):
        """calculate gradient for eacxh param"""
        m = X.shape[0]
        predictions = X.dot(theta)
        gradients = (1/m) * X.T.dot(predictions - y)
        return gradients

    def fit(self, X_train, y_train, X_val=None, y_val=None):
        """train model"""

        X_train_bias = self.add_bias(X_train) #add bias
        if X_val is not None:
            X_val_bias = self.add_bias(X_val)

        # init params to zero
        n_features = X_train_bias.shape[1]
        self.theta = np.zeros(n_features)
        print(f"Initialized {n_features} parameters to zero")

        # training loop
        for i in range(self.max_iterations):
            # 1.) calculate training cost
            train_cost = self.compute_cost(X_train_bias, y_train, self.theta)
            self.costs_train.append(train_cost)

            # calculate validation cost
            if X_val is not None:
                val_cost = self.compute_cost(X_val_bias, y_val, self.theta)
                self.costs_val.append(val_cost)

            # find gradients to impro e model
            gradients = self.compute_gradients(X_train_bias, y_train, self.theta)
```

```
            # iterate in direction of gradient
            self.theta = self.theta - self.learning_rate * gradients

            # print progress every 200th iteration so we can keep track of model
            if i % 200 == 0:
                val_info = f", Validation Cost: {val_cost:.2f}" if X_val is not None else ""
                print(f"Iteration {i:4d}: Training Cost: {train_cost:.2f}{val_info}")

        print("training done")

    def predict(self, X):
        """predict with new data"""
        X_bias = self.add_bias(X)
        return X_bias.dot(self.theta)

    def score(self, X, y):
        """calculate r squared score"""
        predictions = self.predict(X)
        ss_res = np.sum((y - predictions) ** 2)
        ss_tot = np.sum((y - np.mean(y)) ** 2)
        r2 = 1 - (ss_res / ss_tot)
        return r2
```

```
# testing different learning rates
learning_rates = [1e-10, 1e-11, 1e-12]  # using the normal 0.1, 0.05, 0.01 returned values too large so I had to make them smaller
models = {}
results = {}

print("Training with small learning rates for unscaled features")
print("(Balances numerical stability with learning effectiveness)")
print("=" * 60)

for lr in learning_rates:
    print(f"\nlearning rate: {lr}")
    print("-" * 40)

    # create and train model
    model = LinearRegressionGD(learning_rate=lr, max_iterations=2000)
    model.fit(X_train_final, y_train_final, X_val, y_val)


    models[lr] = model

    # evaluate
    test_predictions = model.predict(X_test)
    train_r2 = model.score(X_train_final, y_train_final)
    val_r2 = model.score(X_val, y_val)
    test_r2 = model.score(X_test, y_test)

    results[lr] = {
        'train_r2': train_r2,
        'val_r2': val_r2,
        'test_r2': test_r2,
        'final_train_cost': model.costs_train[-1],
        'final_val_cost': model.costs_val[-1]
    }

    print(f"\nresults:")
    print(f"   training R²: {train_r2:.4f}")
    print(f"   validation R²: {val_r2:.4f}")
    print(f"   test R²: {test_r2:.4f}")
    print(f"   final training cost: {model.costs_train[-1]:.2f}")
    print(f"   final validation cost: {model.costs_val[-1]:.2f}")
```

```
   validation R²: -0.1639
   test R²: 0.1956
   final training cost: 1537540029387.04
   final validation cost: 1794377594457.51

learning rate: 1e-11
----------------------------------------
Initialized 6 parameters to zero
Iteration    0: Training Cost: 12394222684850.72, Validation Cost: 13499946025980.11
Iteration  200: Training Cost: 11137038582943.59, Validation Cost: 12128309708093.23
Iteration  400: Training Cost: 10025433471255.81, Validation Cost: 10916473889284.93
Iteration  600: Training Cost: 9042549641755.90, Validation Cost: 9845876261521.09
Iteration  800: Training Cost: 8173481469742.88, Validation Cost: 8900107437243.61
Iteration 1000: Training Cost: 7405049367142.74, Validation Cost: 8064661440391.62
Iteration 1200: Training Cost: 6725599911486.75, Validation Cost: 7326715102263.58
Iteration 1400: Training Cost: 6124829119488.66, Validation Cost: 6674933014374.04
Iteration 1600: Training Cost: 5593626185130.00, Validation Cost: 6099295078175.08
Iteration 1800: Training Cost: 5123935312511.30, Validation Cost: 5590944034330.38
training done

results:
   training R²: -2.0648
   validation R²: -2.3354
   test R²: -1.5328
   final training cost: 4710585500404.75
   final validation cost: 5144159021314.86

learning rate: 1e-12
----------------------------------------
Initialized 6 parameters to zero
Iteration    0: Training Cost: 12394222684850.72, Validation Cost: 13499946025980.11
Iteration  200: Training Cost: 12261446686853.23, Validation Cost: 13355033359007.21
Iteration  400: Training Cost: 12130294517099.89, Validation Cost: 13211903759503.25
Iteration  600: Training Cost: 12000746316442.81, Validation Cost: 13070535354546.88
Iteration  800: Training Cost: 11872782468608.12, Validation Cost: 12930906539125.61
Iteration 1000: Training Cost: 11746383597225.73, Validation Cost: 12792995972856.86
Iteration 1200: Training Cost: 11621530562895.30, Validation Cost: 12656782576749.05
Iteration 1400: Training Cost: 11498204460288.17, Validation Cost: 12522245530002.43
Iteration 1600: Training Cost: 11376386615284.67, Validation Cost: 12389364266848.93
Iteration 1800: Training Cost: 11256058582146.51, Validation Cost: 12258118473430.77
training done

results:
   training R²: -6.2491
   validation R²: -6.8672
   test R²: -4.4142
   final training cost: 11137792792209.29
   final validation cost: 12129132251227.71
```

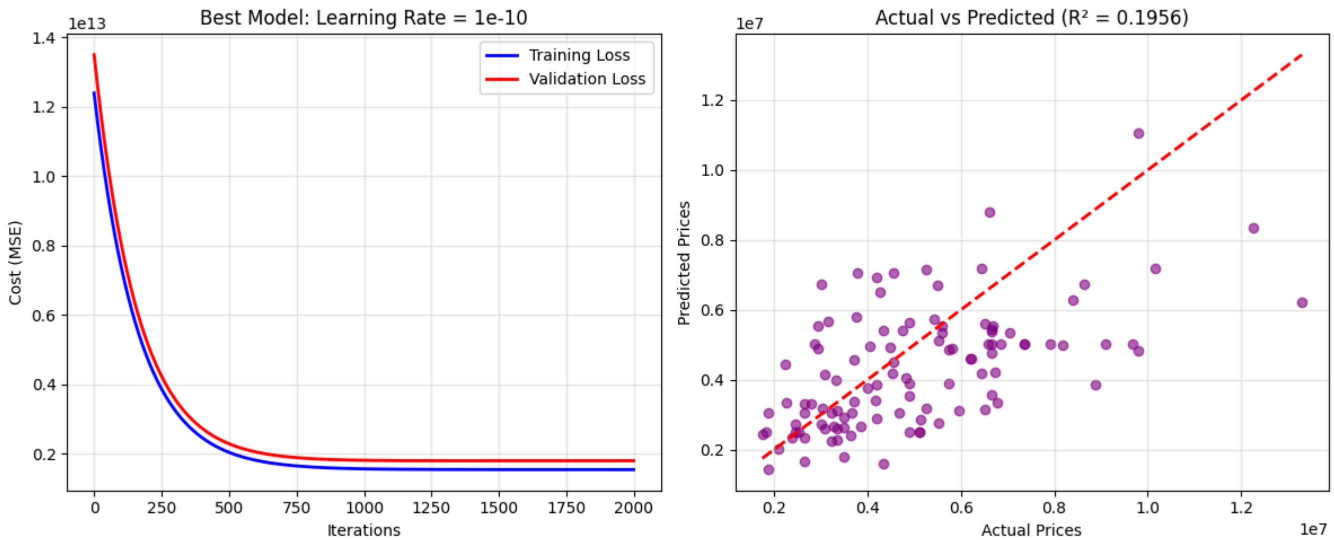
```python
# plot training and validation loss for each learning rate
plt.figure(figsize=(15, 5))

# plots for each learning rate
for i, lr in enumerate(learning_rates):
    plt.subplot(1, 3, i+1)
    model = models[lr]

    # plot
    plt.plot(model.costs_train, label='Training Loss', color='blue', linewidth=2)
    plt.plot(model.costs_val, label='Validation Loss', color='red', linewidth=2)

    plt.title(f'Learning Rate: {lr}')
    plt.xlabel('Iterations')
    plt.ylabel('Cost (MSE)')
    plt.legend()
    plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.suptitle('Training and Validation Loss Curves for Different Learning Rates',
             fontsize=16, y=1.05)
plt.show()

# determine best learning rate
best_lr = min(results.keys(), key=lambda x: results[x]['final_val_cost'])
print(f"\nbest learning rate: {best_lr}")
print(f"best model performance:")
for metric, value in results[best_lr].items():
    print(f"   {metric}: {value:.4f}")
```




```
best learning rate: 1e-10
best model performance:
   train_r2: -0.0008
   val_r2: -0.1639
   test_r2: 0.1956
   final_train_cost: 1537540029387.0376
   final_val_cost: 1794377594457.5090
```

```
#plot best learning curve
best_model = models[best_lr]

plt.figure(figsize=(12, 5))

# graph with training and validation loss
plt.subplot(1, 2, 1)
plt.plot(best_model.costs_train, label='Training Loss', color='blue', linewidth=2)
plt.plot(best_model.costs_val, label='Validation Loss', color='red', linewidth=2)
plt.title(f'Best Model: Learning Rate = {best_lr}')
plt.xlabel('Iterations')
plt.ylabel('Cost (MSE)')
plt.legend()
plt.grid(True, alpha=0.3)

# actual vs model predicted prices
test_predictions = best_model.predict(X_test)
plt.subplot(1, 2, 2)
plt.scatter(y_test, test_predictions, alpha=0.6, color='purple')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', linewidth=2)
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title(f'Actual vs Predicted (R² = {results[best_lr]["test_r2"]:.4f})')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# final model parameters
print(f"final model parameters (θ) for learning rate {best_lr}:")
print(f"   bias (θ₀): {best_model.theta[0]:.4f}")
for i, feature in enumerate(features):
    print(f"   {feature} (θ{i+1}): {best_model.theta[i+1]:.4f}")

# Calculate additional metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error

mse = mean_squared_error(y_test, test_predictions)
mae = mean_absolute_error(y_test, test_predictions)
rmse = np.sqrt(mse)
```

```
print(f"\n test set performance metrics:")
print(f"   mean squared error (MSE): {mse:.2f}")
print(f"   root mean squared rrror (RMSE): {rmse:.2f}")
print(f"   mean absolute error (MAE): {mae:.2f}")
print(f"   r-squared (R²): {results[best_lr]['test_r2']:.4f}")
```




```
final model parameters (θ) for learning rate 1e-10:
   bias (θ₀): 0.2138
   area (θ1): 838.3710
   bedrooms (θ2): 0.6946
   bathrooms (θ3): 0.3296
   stories (θ4): 0.4925
   parking (θ5): 0.1681

 test set performance metrics:
   mean squared error (MSE): 4065710415617.87
   root mean squared rrror (RMSE): 2016360.69
   mean absolute error (MAE): 1574561.59
   r-squared (R²): 0.1956
```

Problem 1.B

```
# prepare data for problem 1.b
from sklearn.preprocessing import LabelEncoder

# define all 11 features for 1.b
features_1b = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom',
               'basement', 'hotwaterheating', 'airconditioning', 'parking', 'prefarea']


# create dataframe copy for preprocessing
df_1b = df.copy()

# convert categorical variables to binary
le = LabelEncoder()
categorical_cols = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea']

for col in categorical_cols:
    if col in df_1b.columns:
        df_1b[col] = le.fit_transform(df_1b[col])


# rxtract features and target for 1.b
X_1b = df_1b[features_1b].values
y_1b = df_1b['price'].values
```

```
print("First five rows of processed features:")
display(pd.DataFrame(X_1b[:5]))
```

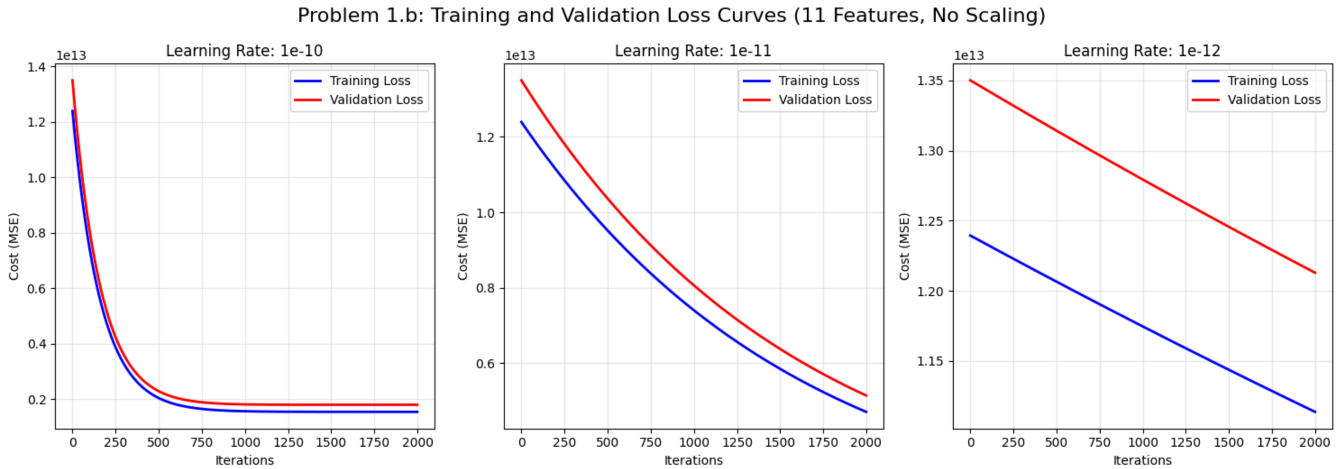
First five rows of processed features:

	0	1	2	3	4	5	6	7	8	9	10
0	7420	4	2	3	1	0	0	0	1	2	1
1	8960	4	4	4	1	0	0	0	1	3	0
2	9960	3	2	2	1	0	1	0	0	2	1
3	7500	4	2	2	1	0	1	0	1	3	1
4	7420	4	1	2	1	1	1	0	1	2	0

```
# split
X_train_1b, X_test_1b, y_train_1b, y_test_1b = train_test_split(X_1b, y_1b, test_size=0.2, random_state=42)

# validation
X_train_final_1b, X_val_1b, y_train_final_1b, y_val_1b = train_test_split(
    X_train_1b, y_train_1b, test_size=0.2, random_state=42
)

print(f"Problem 1.b - Training set: {X_train_final_1b.shape}")
print(f"Problem 1.b - Validation set: {X_val_1b.shape}")
print(f"Problem 1.b - Test set: {X_test_1b.shape}")

# ditto
learning_rates_1b = [1e-10, 1e-11, 1e-12]
models_1b = {}
results_1b = {}

print("\n" + "="*60)
print("PROBLEM 1.b: Training with ALL 11 features (NO SCALING)")
print("Using small learning rates to balance stability and learning")
print("="*60)

for lr in learning_rates_1b:
    print(f"\nLearning rate: {lr}")
    print("-" * 40)

    # create and train model
    model = LinearRegressionGD(learning_rate=lr, max_iterations=2000)
    model.fit(X_train_final_1b, y_train_final_1b, X_val_1b, y_val_1b)

    models_1b[lr] = model

    test_predictions = model.predict(X_test_1b)
    train_r2 = model.score(X_train_final_1b, y_train_final_1b)
    val_r2 = model.score(X_val_1b, y_val_1b)
    test_r2 = model.score(X_test_1b, y_test_1b)

    results_1b[lr] = {
        'train_r2': train_r2,
        'val_r2': val_r2,
        'test_r2': test_r2,
        'final_train_cost': model.costs_train[-1],
        'final_val_cost': model.costs_val[-1]
    }

    print(f"results:")
    print(f"   training R²: {train_r2:.4f}")
    print(f"   validation R²: {val_r2:.4f}")
    print(f"   test R²: {test_r2:.4f}")
    print(f"   final training cost: {model.costs_train[-1]:.2f}")
    print(f"   final validation cost: {model.costs_val[-1]:.2f}")


# find best learning rate for problem 1.b
best_lr_1b = min(results_1b.keys(), key=lambda x: results_1b[x]['final_val_cost'])
print(f"best learning rate for 1.b: {best_lr_1b}")
```

```
Iteration 1400: Training Cost: 1539437532588.01, Validation Cost: 1793214797890.89
Iteration 1600: Training Cost: 1538058939807.23, Validation Cost: 1793472183255.63
Iteration 1800: Training Cost: 1537656891270.74, Validation Cost: 1793989418928.68
training done
results:
   training R²: -0.0008
   validation R²: -0.1639
   test R²: 0.1956
   final training cost: 1537539893518.06
   final validation cost: 1794377434440.33

Learning rate: 1e-11
----------------------------------------
Initialized 12 parameters to zero
Iteration    0: Training Cost: 12394222684850.72, Validation Cost: 13499946025980.11
Iteration  200: Training Cost: 11137038536604.54, Validation Cost: 12128309654650.58
Iteration  400: Training Cost: 10025433388147.94, Validation Cost: 10916473793347.27
Iteration  600: Training Cost: 9042549529846.43, Validation Cost: 9845876132213.76
Iteration  800: Training Cost: 8173481335638.91, Validation Cost: 8900107282144.45
Iteration 1000: Training Cost: 7405049216299.41, Validation Cost: 8064661265766.63
Iteration 1200: Training Cost: 6725599748385.44, Validation Cost: 7326714913268.77
Iteration 1400: Training Cost: 6124828947789.36, Validation Cost: 6674932815228.11
Iteration 1600: Training Cost: 5593626007801.47, Validation Cost: 6099294872307.10
Iteration 1800: Training Cost: 5123935131942.03, Validation Cost: 5590943824505.76
training done
results:
   training R²: -2.0648
   validation R²: -2.3354
   test R²: -1.5328
   final training cost: 4710585318500.23
   final validation cost: 5144158809746.86

Learning rate: 1e-12
----------------------------------------
Initialized 12 parameters to zero
Iteration    0: Training Cost: 12394222684850.72, Validation Cost: 13499946025980.11
Iteration  200: Training Cost: 12261446681738.59, Validation Cost: 13355033353113.39
Iteration  400: Training Cost: 12130294506982.84, Validation Cost: 13211903747843.86
Iteration  600: Training Cost: 12000746301433.57, Validation Cost: 13070535337247.90
Iteration  800: Training Cost: 11872782448814.99, Validation Cost: 12930906516310.82
Iteration 1000: Training Cost: 11746383572755.05, Validation Cost: 12792995944647.83
Iteration 1200: Training Cost: 11621530533851.54, Validation Cost: 12656782543265.22
Iteration 1400: Training Cost: 11498204426773.92, Validation Cost: 12522245491361.10
Iteration 1600: Training Cost: 11376386577400.70, Validation Cost: 12389364223165.33
Iteration 1800: Training Cost: 11256058539991.78, Validation Cost: 12258118424818.07
training done
results:
   training R²: -6.2491
   validation R²: -6.8672
   test R²: -4.4142
   final training cost: 11137792745901.59
   final validation cost: 12129132197820.89
best learning rate for 1.b: 1e-10
```

```python
# ditto comments from 1.a
plt.figure(figsize=(15, 5))

for i, lr in enumerate(learning_rates_1b):
    plt.subplot(1, 3, i+1)
    model = models_1b[lr]

    plt.plot(model.costs_train, label='Training Loss', color='blue', linewidth=2)
    plt.plot(model.costs_val, label='Validation Loss', color='red', linewidth=2)

    plt.title(f'Learning Rate: {lr}')
    plt.xlabel('Iterations')
    plt.ylabel('Cost (MSE)')
    plt.legend()
    plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.suptitle('Problem 1.b: Training and Validation Loss Curves (11 Features, No Scaling)',
             fontsize=16, y=1.05)
plt.show()
```



PROBLEM 2.A

```
# 80% train, 20% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) #arbitrary random state 42 (got it off t

# standardize with sklearn.preprocessing StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("scaled features; first five rows")
print(X_train_scaled[:5])

# Split training data further to have validation set for plotting
X_train_final, X_val, y_train_final, y_val = train_test_split(
    X_train_scaled, y_train, test_size=0.2, random_state=42
)

print(f"\nfinal training set: {X_train_final.shape}")
print(f"validation set: {X_val.shape}")
print(f"test set: {X_test_scaled.shape}")
```

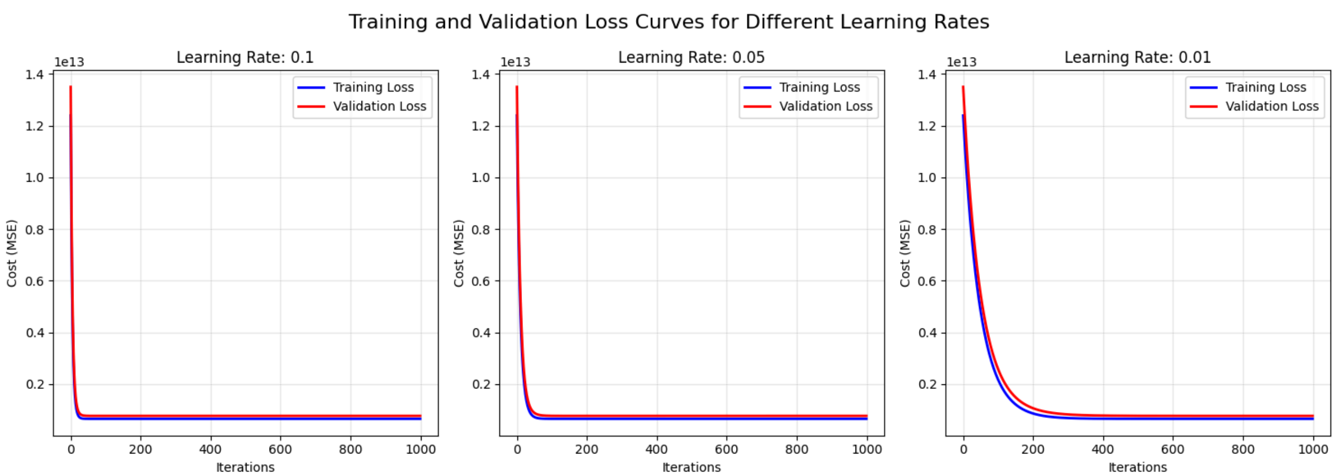
```
scaled features; first five rows
[[ 0.38416819  0.05527092  1.53917323  2.58764353  0.36795665]
 [ 0.9291807   0.05527092  1.53917323 -0.91249891  2.70998729]
 [-0.60775457 -1.28351359 -0.5579503  -0.91249891  1.53897197]
 [-1.15549214  0.05527092 -0.5579503   0.25421524 -0.80305867]
 [-0.63773026  0.05527092 -0.5579503   0.25421524 -0.80305867]]

final training set: (348, 5)
validation set: (88, 5)
test set: (109, 5)
```

```
# test new learning rates
learning_rates = [0.1, 0.05, 0.01]
models = {}
results = {}

print("we will train models with different learning rates")
print("=" * 60)

for lr in learning_rates:
    print(f"\nlearning rate: {lr}")
    print("-" * 40)
```

```
    # ditto
    model = LinearRegressionGD(learning_rate=lr, max_iterations=1000)
    model.fit(X_train_final, y_train_final, X_val, y_val)

    models[lr] = model

    test_predictions = model.predict(X_test_scaled)
    train_r2 = model.score(X_train_final, y_train_final)
    val_r2 = model.score(X_val, y_val)
    test_r2 = model.score(X_test_scaled, y_test)

    results[lr] = {
        'train_r2': train_r2,
        'val_r2': val_r2,
        'test_r2': test_r2,
        'final_train_cost': model.costs_train[-1],
        'final_val_cost': model.costs_val[-1]
    }

    print(f"\nresults:")
    print(f"   training R²: {train_r2:.4f}")
    print(f"   validation R²: {val_r2:.4f}")
    print(f"   test R²: {test_r2:.4f}")
    print(f"   final training cost: {model.costs_train[-1]:.2f}")
    print(f"   final validation cost: {model.costs_val[-1]:.2f}")
```

```
we will train models with different learning rates
============================================================

learning rate: 0.1
----------------------------------------
Initialized 6 parameters to zero
Iteration    0: Training Cost: 12394222684850.72, Validation Cost: 13499946025980.11
Iteration  200: Training Cost: 656361446834.11, Validation Cost: 769239026348.17
Iteration  400: Training Cost: 656361446833.66, Validation Cost: 769239197872.63
Iteration  600: Training Cost: 656361446833.66, Validation Cost: 769239197873.18
Iteration  800: Training Cost: 656361446833.66, Validation Cost: 769239197873.18
training done

results:
   training R²: 0.5728
   validation R²: 0.5010
   test R²: 0.5452
   final training cost: 656361446833.66
   final validation cost: 769239197873.18

learning rate: 0.05
----------------------------------------
Initialized 6 parameters to zero
Iteration    0: Training Cost: 12394222684850.72, Validation Cost: 13499946025980.11
Iteration  200: Training Cost: 656361635985.02, Validation Cost: 769119525229.97
Iteration  400: Training Cost: 656361446834.32, Validation Cost: 769238985394.82
Iteration  600: Training Cost: 656361446833.66, Validation Cost: 769239197492.98
Iteration  800: Training Cost: 656361446833.66, Validation Cost: 769239197872.37
training done

results:
   training R²: 0.5728
   validation R²: 0.5010
   test R²: 0.5452
   final training cost: 656361446833.66
   final validation cost: 769239197873.18

learning rate: 0.01
----------------------------------------
Initialized 6 parameters to zero
Iteration    0: Training Cost: 12394222684850.72, Validation Cost: 13499946025980.11
Iteration  200: Training Cost: 861683869182.48, Validation Cost: 1064288742201.52
Iteration  400: Training Cost: 660459486094.73, Validation Cost: 780634911828.82
Iteration  600: Training Cost: 656462977638.24, Validation Cost: 769178611822.48
Iteration  800: Training Cost: 656365318602.68, Validation Cost: 768915122933.33
training done

results:
   training R²: 0.5728
   validation R²: 0.5011
   test R²: 0.5452
   final training cost: 656361677947.66
   final validation cost: 769112176549.02
```

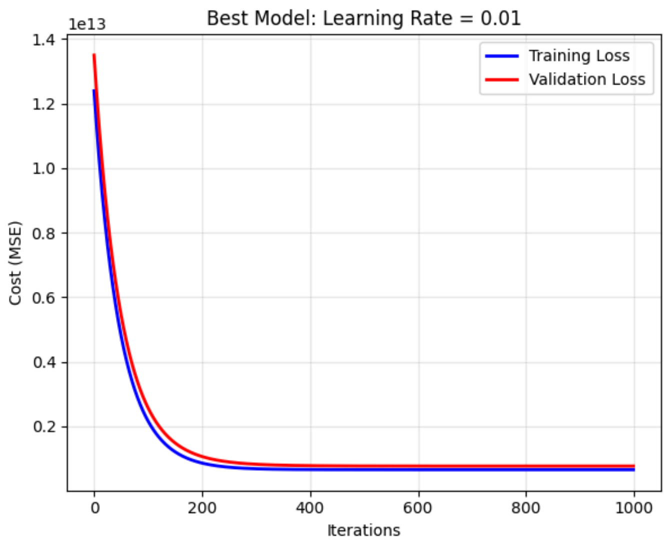
```
# plot training and validation loss for each learning rate
plt.figure(figsize=(15, 5))

# plots for each learning rate
for i, lr in enumerate(learning_rates):
    plt.subplot(1, 3, i+1)
    model = models[lr]

    # plot
    plt.plot(model.costs_train, label='Training Loss', color='blue', linewidth=2)
    plt.plot(model.costs_val, label='Validation Loss', color='red', linewidth=2)

    plt.title(f'Learning Rate: {lr}')
    plt.xlabel('Iterations')
    plt.ylabel('Cost (MSE)')
    plt.legend()
    plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.suptitle('Training and Validation Loss Curves for Different Learning Rates',
             fontsize=16, y=1.05)
plt.show()

# determine best learning rate
best_lr = min(results.keys(), key=lambda x: results[x]['final_val_cost'])
print(f"\nbest learning rate: {best_lr}")
print(f"best model performance:")
for metric, value in results[best_lr].items():
    print(f"   {metric}: {value:.4f}")
```

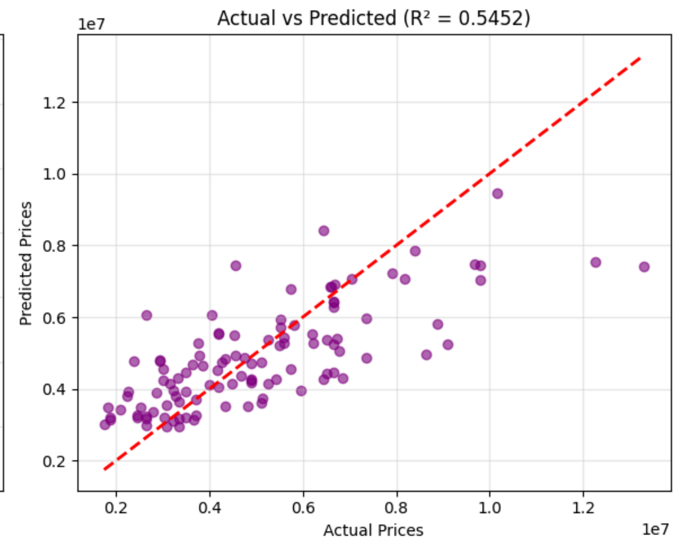



```
best learning rate: 0.01
best model performance:
   train_r2: 0.5728
   val_r2: 0.5011
   test_r2: 0.5452
   final_train_cost: 656361677947.6556
   final_val_cost: 769112176549.0182
```

```
# ditto
best_model = models[best_lr]

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(best_model.costs_train, label='Training Loss', color='blue', linewidth=2)
plt.plot(best_model.costs_val, label='Validation Loss', color='red', linewidth=2)
plt.title(f'Best Model: Learning Rate = {best_lr}')
```

```
plt.xlabel('Iterations')
plt.ylabel('Cost (MSE)')
plt.legend()
plt.grid(True, alpha=0.3)

test_predictions = best_model.predict(X_test_scaled)
plt.subplot(1, 2, 2)
plt.scatter(y_test, test_predictions, alpha=0.6, color='purple')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', linewidth=2)
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title(f'Actual vs Predicted (R² = {results[best_lr]["test_r2"]:.4f})')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print(f"final model parameters (θ) for learning rate {best_lr}:")
print(f"   bias (θ₀): {best_model.theta[0]:.4f}")
for i, feature in enumerate(features):
    print(f"   {feature} (θ{i+1}): {best_model.theta[i+1]:.4f}")

from sklearn.metrics import mean_squared_error, mean_absolute_error

mse = mean_squared_error(y_test, test_predictions)
mae = mean_absolute_error(y_test, test_predictions)
rmse = np.sqrt(mse)

print(f"\n test set performance metrics:")
print(f"   mean squared error (MSE): {mse:.2f}")
print(f"   root mean squared rrror (RMSE): {rmse:.2f}")
print(f"   mean absolute error (MAE): {mae:.2f}")
print(f"   r-squared (R²): {results[best_lr]['test_r2']:.4f}")
```






```
final model parameters (θ) for learning rate 0.01:
   bias (θ₀): 4672255.6424
   area (θ1): 709048.6217
   bedrooms (θ2): 88777.9103
   bathrooms (θ3): 531827.0686
   stories (θ4): 489700.2870
   parking (θ5): 234112.1171

 test set performance metrics:
   mean squared error (MSE): 2298890235352.86
   root mean squared rrror (RMSE): 1516209.17
   mean absolute error (MAE): 1120881.66
   r-squared (R²): 0.5452
```

Problem 2.B

```
from sklearn.preprocessing import LabelEncoder

features_2b = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom',
               'basement', 'hotwaterheating', 'airconditioning', 'parking', 'prefarea']

df_2b = df.copy()

le = LabelEncoder()
categorical_cols = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea']

for col in categorical_cols:
    if col in df_2b.columns:
        df_2b[col] = le.fit_transform(df_2b[col])

X_2b = df_2b[features_2b].values
y_2b = df_2b['price'].values

print("first five rows of processed features (pre-scaling):")
display(pd.DataFrame(X_2b[:5], columns=features_2b))
```

first five rows of processed features (pre-scaling):

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea
0	7420	4	2	3	1	0	0	0	1	2	1
1	8960	4	4	4	1	0	0	0	1	3	0
2	9960	3	2	2	1	0	1	0	0	2	1
3	7500	4	2	2	1	0	1	0	1	3	1
4	7420	4	1	2	1	1	1	0	1	2	0

```
# split data and apply scaling
X_train_2b, X_test_2b, y_train_2b, y_test_2b = train_test_split(X_2b, y_2b, test_size=0.2, random_state=42)

# standardize features with sklearn StandardScaler
scaler_2b = StandardScaler()
X_train_scaled_2b = scaler_2b.fit_transform(X_train_2b)
X_test_scaled_2b = scaler_2b.transform(X_test_2b)

print("Scaled features (first five rows):")
print(X_train_scaled_2b[:5])

# split for validation set
X_train_final_2b, X_val_2b, y_train_final_2b, y_val_2b = train_test_split(
    X_train_scaled_2b, y_train_2b, test_size=0.2, random_state=42
)

print(f"\nProblem 2.b - Training set: {X_train_final_2b.shape}")
print(f"Problem 2.b - Validation set: {X_val_2b.shape}")
print(f"Problem 2.b - Test set: {X_test_scaled_2b.shape}")
```

```
Scaled features (first five rows):
[[ 0.38416819  0.05527092  1.53917323  2.58764353  0.40715525 -0.46677307
  -0.74642003 -0.23052136  1.50124327  0.36795665 -0.55262032]
 [ 0.9291807   0.05527092  1.53917323 -0.91249891  0.40715525 -0.46677307
   1.33972825 -0.23052136  1.50124327  2.70998729 -0.55262032]
 [-0.60775457 -1.28351359 -0.5579503  -0.91249891  0.40715525 -0.46677307
   1.33972825 -0.23052136  1.50124327  1.53897197 -0.55262032]
 [-1.15549214  0.05527092 -0.5579503   0.25421524  0.40715525 -0.46677307
   1.33972825 -0.23052136 -0.66611456 -0.80305867  1.80956067]
 [-0.63773026  0.05527092 -0.5579503   0.25421524  0.40715525 -0.46677307
  -0.74642003 -0.23052136 -0.66611456 -0.80305867 -0.55262032]]

Problem 2.b - Training set: (348, 11)
Problem 2.b - Validation set: (88, 11)
Problem 2.b - Test set: (109, 11)
```

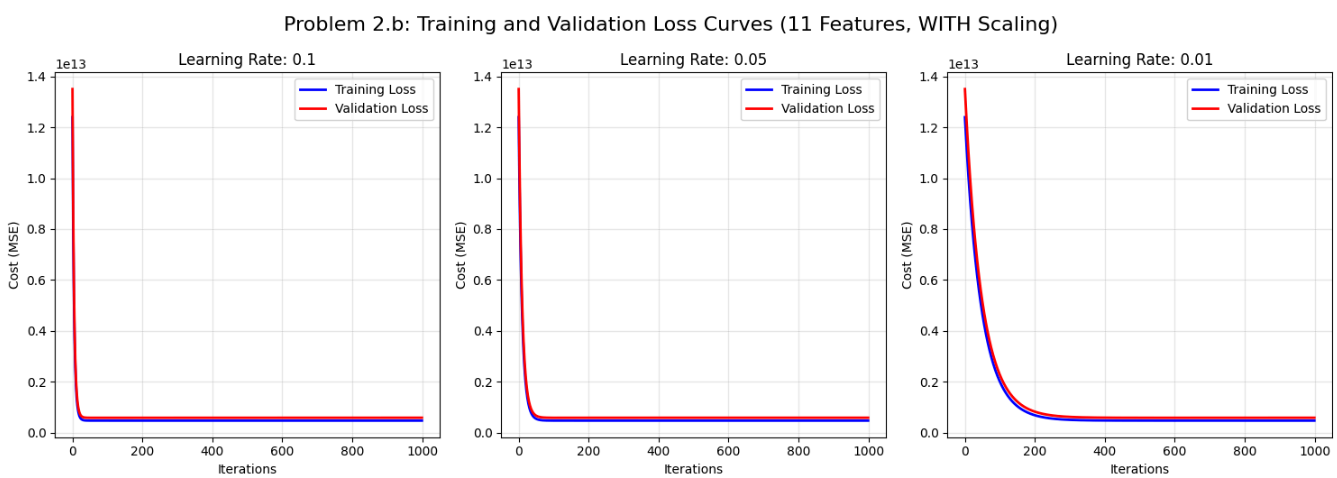
```
# train models with different learning rates for scaled 11 features
learning_rates_2b = [0.1, 0.05, 0.01]
models_2b = {}
results_2b = {}

print("\n" + "="*60)
print("PROBLEM 2.b: Training with ALL 11 features WITH SCALING")
print("Using standard learning rates for scaled features")
```

```
print("="*60)

for lr in learning_rates_2b:
    print(f"\nLearning rate: {lr}")
    print("-" * 40)

    # create and train model
    model = LinearRegressionGD(learning_rate=lr, max_iterations=1000)
    model.fit(X_train_final_2b, y_train_final_2b, X_val_2b, y_val_2b)

    models_2b[lr] = model

    # evaluate performance
    test_predictions = model.predict(X_test_scaled_2b)
    train_r2 = model.score(X_train_final_2b, y_train_final_2b)
    val_r2 = model.score(X_val_2b, y_val_2b)
    test_r2 = model.score(X_test_scaled_2b, y_test_2b)

    results_2b[lr] = {
        'train_r2': train_r2,
        'val_r2': val_r2,
        'test_r2': test_r2,
        'final_train_cost': model.costs_train[-1],
        'final_val_cost': model.costs_val[-1]
    }

    print(f"Results:")
    print(f"   Training R²: {train_r2:.4f}")
    print(f"   Validation R²: {val_r2:.4f}")
    print(f"   Test R²: {test_r2:.4f}")
    print(f"   Final training cost: {model.costs_train[-1]:.2f}")
    print(f"   Final validation cost: {model.costs_val[-1]:.2f}")

# find best learning rate for problem 2.b
best_lr_2b = min(results_2b.keys(), key=lambda x: results_2b[x]['final_val_cost'])
print(f"\nBest learning rate for 2.b: {best_lr_2b}")
print(f"Best model performance:")
for metric, value in results_2b[best_lr_2b].items():
    print(f"   {metric}: {value:.4f}")
```

```
Using standard learning rates for scaled features
============================================================

Learning rate: 0.1
----------------------------------------
Initialized 12 parameters to zero
Iteration    0: Training Cost: 12394222684850.72, Validation Cost: 13499946025980.11
Iteration  200: Training Cost: 475948556138.03, Validation Cost: 589744586334.99
Iteration  400: Training Cost: 475948554140.99, Validation Cost: 589757039326.29
Iteration  600: Training Cost: 475948554140.99, Validation Cost: 589757045769.42
Iteration  800: Training Cost: 475948554140.99, Validation Cost: 589757045772.72
training done
Results:
   Training R²: 0.6902
   Validation R²: 0.6175
   Test R²: 0.6441
   Final training cost: 475948554140.99
   Final validation cost: 589757045772.72

Learning rate: 0.05
----------------------------------------
Initialized 12 parameters to zero
Iteration    0: Training Cost: 12394222684850.72, Validation Cost: 13499946025980.11
Iteration  200: Training Cost: 475952959191.68, Validation Cost: 589220147053.05
Iteration  400: Training Cost: 475948556446.35, Validation Cost: 589743670535.96
Iteration  600: Training Cost: 475948554142.26, Validation Cost: 589756729254.50
Iteration  800: Training Cost: 475948554140.99, Validation Cost: 589757038332.28
training done
Results:
   Training R²: 0.6902
   Validation R²: 0.6175
   Test R²: 0.6441
   Final training cost: 475948554140.99
   Final validation cost: 589757045594.67

Learning rate: 0.01
----------------------------------------
Initialized 12 parameters to zero
```

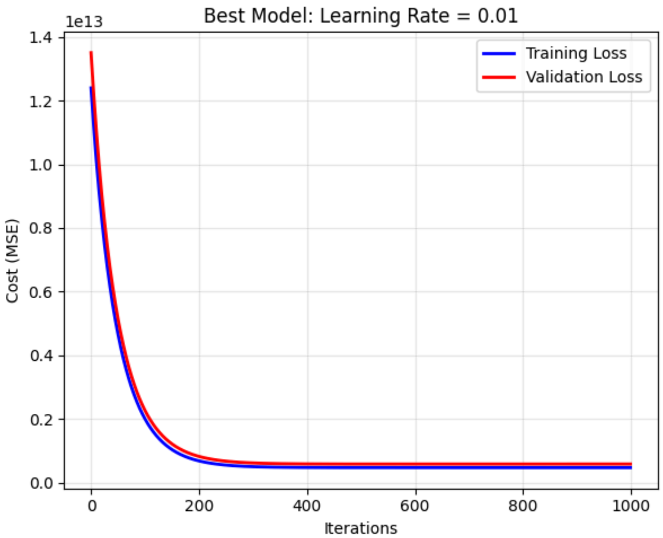
```
Iteration  600: Training Cost: 476137450549.21, Validation Cost: 587872200777.14
Iteration  800: Training Cost: 475972127890.15, Validation Cost: 588662139394.71
training done
Results:
   Training R²: 0.6902
   Validation R²: 0.6178
   Test R²: 0.6443
   Final training cost: 475953277335.34
   Final validation cost: 589204710521.63

Best learning rate for 2.b: 0.01
Best model performance:
   train_r2: 0.6902
   val_r2: 0.6178
   test_r2: 0.6443
   final_train_cost: 475953277335.3421
   final_val_cost: 589204710521.6343
```

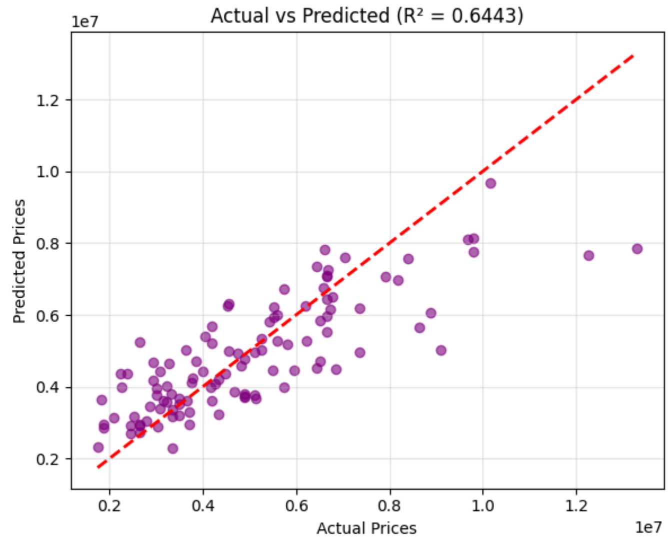
```python
# plot training and validation loss curves for problem 2.b
plt.figure(figsize=(15, 5))

for i, lr in enumerate(learning_rates_2b):
    plt.subplot(1, 3, i+1)
    model = models_2b[lr]

    plt.plot(model.costs_train, label='Training Loss', color='blue', linewidth=2)
    plt.plot(model.costs_val, label='Validation Loss', color='red', linewidth=2)

    plt.title(f'Learning Rate: {lr}')
    plt.xlabel('Iterations')
    plt.ylabel('Cost (MSE)')
    plt.legend()
    plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.suptitle('Problem 2.b: Training and Validation Loss Curves (11 Features, WITH Scaling)',
             fontsize=16, y=1.05)
plt.show()
```




```python
# detailed analysis of best model for problem 2.b
best_model_2b = models_2b[best_lr_2b]

plt.figure(figsize=(12, 5))

# plot learning curves
plt.subplot(1, 2, 1)
plt.plot(best_model_2b.costs_train, label='Training Loss', color='blue', linewidth=2)
plt.plot(best_model_2b.costs_val, label='Validation Loss', color='red', linewidth=2)
plt.title(f'Best Model: Learning Rate = {best_lr_2b}')
```

```
plt.xlabel('Iterations')
plt.ylabel('Cost (MSE)')
plt.legend()
plt.grid(True, alpha=0.3)

# actual vs predicted scatter plot
test_predictions_2b = best_model_2b.predict(X_test_scaled_2b)
plt.subplot(1, 2, 2)
plt.scatter(y_test_2b, test_predictions_2b, alpha=0.6, color='purple')
plt.plot([y_test_2b.min(), y_test_2b.max()], [y_test_2b.min(), y_test_2b.max()], 'r--', linewidth=2)
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title(f'Actual vs Predicted (R² = {results_2b[best_lr_2b]["test_r2"]:.4f})')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# display final model parameters
print(f"Final model parameters (θ) for learning rate {best_lr_2b}:")
print(f"   Bias (θ₀): {best_model_2b.theta[0]:.4f}")
for i, feature in enumerate(features_2b):
    print(f"   {feature} (θ{i+1}): {best_model_2b.theta[i+1]:.4f}")

# calculate additional performance metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error

mse_2b = mean_squared_error(y_test_2b, test_predictions_2b)
mae_2b = mean_absolute_error(y_test_2b, test_predictions_2b)
rmse_2b = np.sqrt(mse_2b)

print(f"\nTest set performance metrics:")
print(f"   Mean Squared Error (MSE): {mse_2b:.2f}")
print(f"   Root Mean Squared Error (RMSE): {rmse_2b:.2f}")
print(f"   Mean Absolute Error (MAE): {mae_2b:.2f}")
print(f"   R-squared (R²): {results_2b[best_lr_2b]['test_r2']:.4f}")
```






```
Final model parameters (θ) for learning rate 0.01:
   Bias (θ₀): 4699949.5319
   area (θ1): 541444.1046
   bedrooms (θ2): 68879.5131
   bathrooms (θ3): 498168.6310
   stories (θ4): 412231.6843
   mainroad (θ5): 153119.3167
   guestroom (θ6): 62943.6377
   basement (θ7): 233861.8795
   hotwaterheating (θ8): 177709.2287
   airconditioning (θ9): 341720.9448
   parking (θ10): 186497.9290
   prefarea (θ11): 292925.4434

Test set performance metrics:
   Mean Squared Error (MSE): 1798059704973.56
   Root Mean Squared Error (RMSE): 1340917.49
   Mean Absolute Error (MAE): 975571.53
   R-squared (R²): 0.6443
```

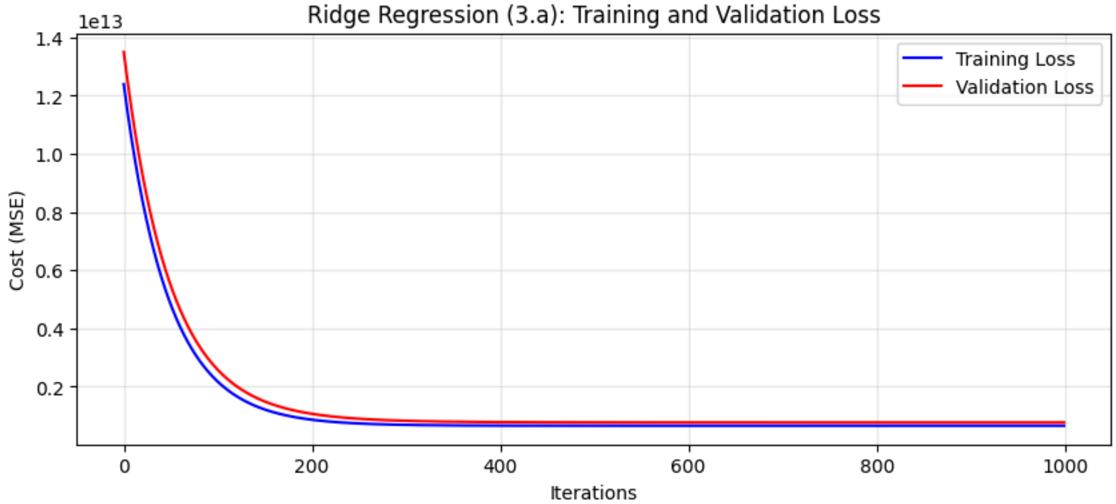
Problem 3.A

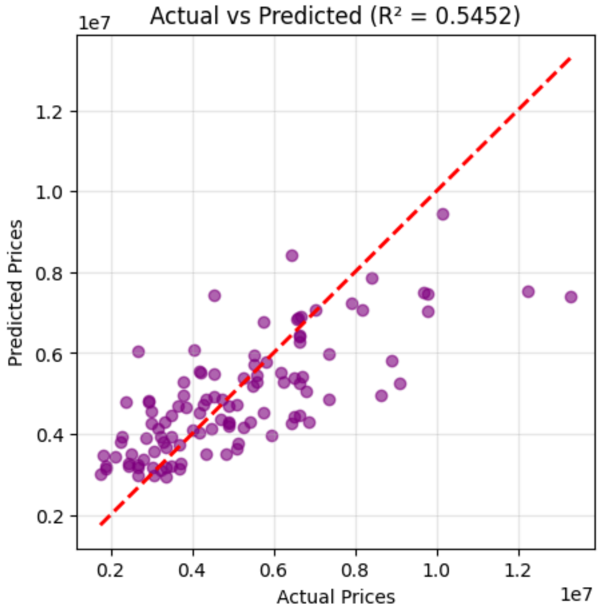
```
# new linear regression with linearization
class LinearRegressionGDRidge(LinearRegressionGD):
    def __init__(self, learning_rate=0.01, max_iterations=2000, lambda_=0.1):
        super().__init__(learning_rate, max_iterations)
        self.lambda_ = lambda_

    def compute_cost(self, X, y, theta, regularize=True):
        m = X.shape[0]
        predictions = X.dot(theta)
        cost = (1/(2*m)) * np.sum((predictions - y)**2)
        if regularize:
            cost += (self.lambda_/(2*m)) * np.sum(theta[1:]**2)
        return cost

    def compute_gradients(self, X, y, theta, regularize=True):
        m = X.shape[0]
        predictions = X.dot(theta)
        gradients = (1/m) * X.T.dot(predictions - y)
        if regularize:
            reg = np.concatenate([[0], self.lambda_ * theta[1:]/m])
            gradients += reg
        return gradients

    def fit(self, X_train, y_train, X_val=None, y_val=None):
        X_train_bias = self.add_bias(X_train)
        if X_val is not None:
            X_val_bias = self.add_bias(X_val)
```

```
        n_features = X_train_bias.shape[1]
        self.theta = np.zeros(n_features)
        self.costs_train = []
        self.costs_val = []
        for i in range(self.max_iterations):
            train_cost = self.compute_cost(X_train_bias, y_train, self.theta, regularize=True)
            self.costs_train.append(train_cost)
            if X_val is not None:
                val_cost = self.compute_cost(X_val_bias, y_val, self.theta, regularize=False)
                self.costs_val.append(val_cost)
            gradients = self.compute_gradients(X_train_bias, y_train, self.theta, regularize=True)
            self.theta = self.theta - self.learning_rate * gradients
            if i % 200 == 0:
                val_info = f", Validation Cost: {val_cost:.2f}" if X_val is not None else ""
                print(f"Iteration {i:4d}: Training Cost: {train_cost:.2f}{val_info}")
        print("training done (ridge)")
```

```
ridge_learning_rate = best_lr  # from 2.a
ridge_lambda = 0.1

ridge_model = LinearRegressionGDRidge(learning_rate=ridge_learning_rate, max_iterations=1000, lambda_=ridge_lambda)
ridge_model.fit(X_train_final, y_train_final, X_val, y_val)

# eval
ridge_test_predictions = ridge_model.predict(X_test_scaled)
ridge_train_r2 = ridge_model.score(X_train_final, y_train_final)
ridge_val_r2 = ridge_model.score(X_val, y_val)
ridge_test_r2 = ridge_model.score(X_test_scaled, y_test)

print(f"\nRidge Regression Results (3.a):")
print(f"   Training R²: {ridge_train_r2:.4f}")
print(f"   Validation R²: {ridge_val_r2:.4f}")
print(f"   Test R²: {ridge_test_r2:.4f}")
print(f"   Final training cost: {ridge_model.costs_train[-1]:.2f}")
print(f"   Final validation cost: {ridge_model.costs_val[-1]:.2f}")

# plot training and validation loss
plt.figure(figsize=(10, 4))
plt.plot(ridge_model.costs_train, label='Training Loss', color='blue')
plt.plot(ridge_model.costs_val, label='Validation Loss', color='red')
plt.title('Ridge Regression (3.a): Training and Validation Loss')
plt.xlabel('Iterations')
plt.ylabel('Cost (MSE)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# actual vs predicted
plt.figure(figsize=(5, 5))
plt.scatter(y_test, ridge_test_predictions, alpha=0.6, color='purple')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', linewidth=2)
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title(f'Actual vs Predicted (R² = {ridge_test_r2:.4f})')
plt.grid(True, alpha=0.3)
plt.show()
```

```
Iteration    0: Training Cost: 12394222684850.72, Validation Cost: 13499946025980.11
Iteration  200: Training Cost: 861827996615.85, Validation Cost: 1064304437418.85
Iteration  400: Training Cost: 660614514974.91, Validation Cost: 780624748001.54
Iteration  600: Training Cost: 656619253581.05, Validation Cost: 769157045325.10
Iteration  800: Training Cost: 656521722891.25, Validation Cost: 768889407582.36
training done (ridge)

Ridge Regression Results (3.a):
   Training R²: 0.5728
   Validation R²: 0.5011
   Test R²: 0.5452
   Final training cost: 656518094819.30
   Final validation cost: 769085074418.00
```





Problem 3.B

```
ridge_learning_rate_2b = best_lr_2b  # from 2.b
ridge_lambda = 0.1

ridge_model_2b = LinearRegressionGDRidge(learning_rate=ridge_learning_rate_2b, max_iterations=1000, lambda_=ridge_lambda)
ridge_model_2b.fit(X_train_final_2b, y_train_final_2b, X_val_2b, y_val_2b)

# evaluate
ridge_test_predictions_2b = ridge_model_2b.predict(X_test_scaled_2b)
ridge_train_r2_2b = ridge_model_2b.score(X_train_final_2b, y_train_final_2b)
ridge_val_r2_2b = ridge_model_2b.score(X_val_2b, y_val_2b)
ridge_test_r2_2b = ridge_model_2b.score(X_test_scaled_2b, y_test_2b)

print(f"\nRidge Regression Results (3.b):")
```

```
print(f"   Training R²: {ridge_train_r2_2b:.4f}")
print(f"   Validation R²: {ridge_val_r2_2b:.4f}")
print(f"   Test R²: {ridge_test_r2_2b:.4f}")
print(f"   Final training cost: {ridge_model_2b.costs_train[-1]:.2f}")
print(f"   Final validation cost: {ridge_model_2b.costs_val[-1]:.2f}")

# plot training and validation loss
plt.figure(figsize=(10, 4))
plt.plot(ridge_model_2b.costs_train, label='Training Loss', color='blue')
plt.plot(ridge_model_2b.costs_val, label='Validation Loss', color='red')
plt.title('Ridge Regression (3.b): Training and Validation Loss')
plt.xlabel('Iterations')
plt.ylabel('Cost (MSE)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# avctual vs predicted
plt.figure(figsize=(5, 5))
```