Homework 2 Report
Name: Viprav Lipare
Student ID: 801288922
Homework Number: 2
Github Repository: https://github.com/vipravlipare/ECGR-5105-Intro-to-Machine-Learning

Problem 1A

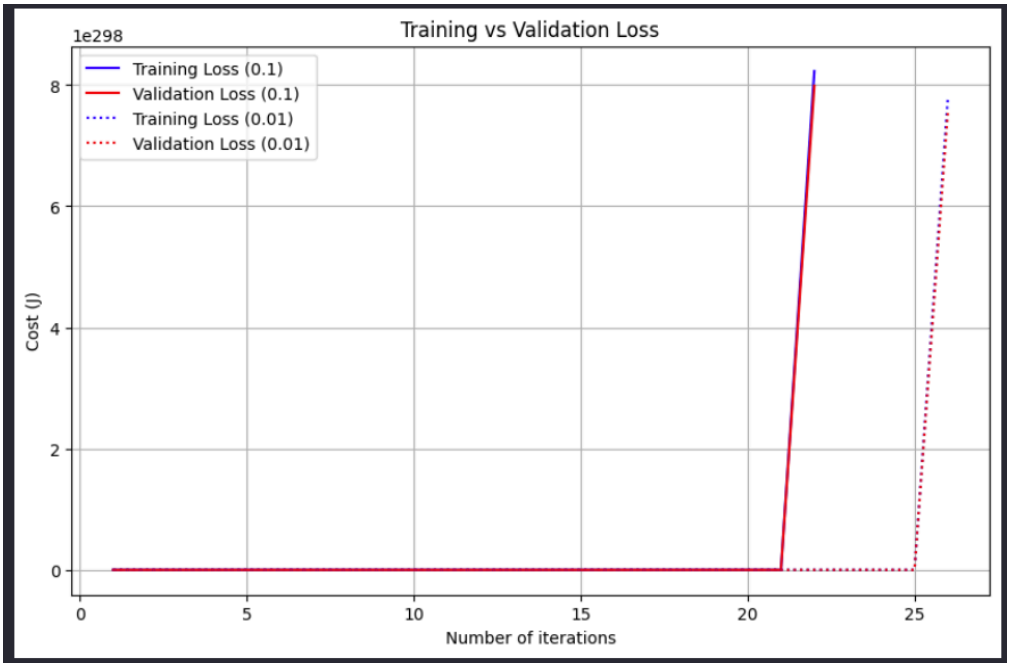
Source Code (1A):

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
file_path = "Housing.csv"
sample = df = pd.read_csv(file_path)

inputs = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking'];
X = sample[inputs].values
Y = sample.values[:, 0]; #Price

m = len(Y)
X_0 = np.ones((m, 1))
X = np.hstack((X_0, X))
theta = np.zeros(X.shape[1])
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

# defining function for computing the cost for linear regression
def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    errors = predictions - y
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J

# defining function for gradient descent algorithm
def gradient_descent(Xt, Yt, Xv, Yv, theta, alpha, iterations):
  train_cost_history = np.zeros(iterations)
  valid_cost_history = np.zeros(iterations)
  m = len(Yt)
  for i in range(iterations):
```

```
    predictions = Xt.dot(theta)
    errors = np.subtract(predictions, Yt)
    sum_delta = (alpha / m) * Xt.transpose().dot(errors)
    theta = theta - sum_delta
    train_cost_history[i] = compute_cost(Xt, Yt, theta)
    valid_cost_history[i] = compute_cost(Xv, Yv, theta)
  return theta, train_cost_history, valid_cost_history

theta = np.zeros(X.shape[1])
iterations = 1000

alpha = 0.1
theta_1, train_cost_history_1, valid_cost_history_1  =
gradient_descent(X_train, Y_train, X_test, Y_test, theta, alpha,
iterations) # 0.1 Learning Rate

alpha = 0.01
theta_2, train_cost_history_2, valid_cost_history_2  =
gradient_descent(X_train, Y_train, X_test, Y_test, theta, alpha,
iterations) # 0.01 Learning Rate

plt.plot(range(1, iterations+1), train_cost_history_1, color='blue',
label='Training Loss (0.1)')
plt.plot(range(1, iterations+1), valid_cost_history_1, color='red',
label='Validation Loss (0.1)')
plt.plot(range(1, iterations+1), train_cost_history_2, color='blue',
linestyle='dotted', label='Training Loss (0.01)')
plt.plot(range(1, iterations+1), valid_cost_history_2, color='red',
linestyle='dotted', label='Validation Loss (0.01)')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Training vs Validation Loss')
plt.legend()
plt.show()

features = ['X_0', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']
```

```
print("\nBest Model Parameters (Theta) for Problem 1A: Learning Rate =
0.1\n")
theta_list = theta_1.flatten()
for label, value in zip(features, theta_list):
    print(f"{label:>12}: {value:,.2f}")
```

Figures / Results (1A):




```
Best Model Parameters (Theta) for Problem 1A: Learning Rate = 0.1

       X_0: nan
      area: nan
  bedrooms: nan
 bathrooms: nan
   stories: nan
   parking: nan
```

Problem 1B

Source Code (1B):

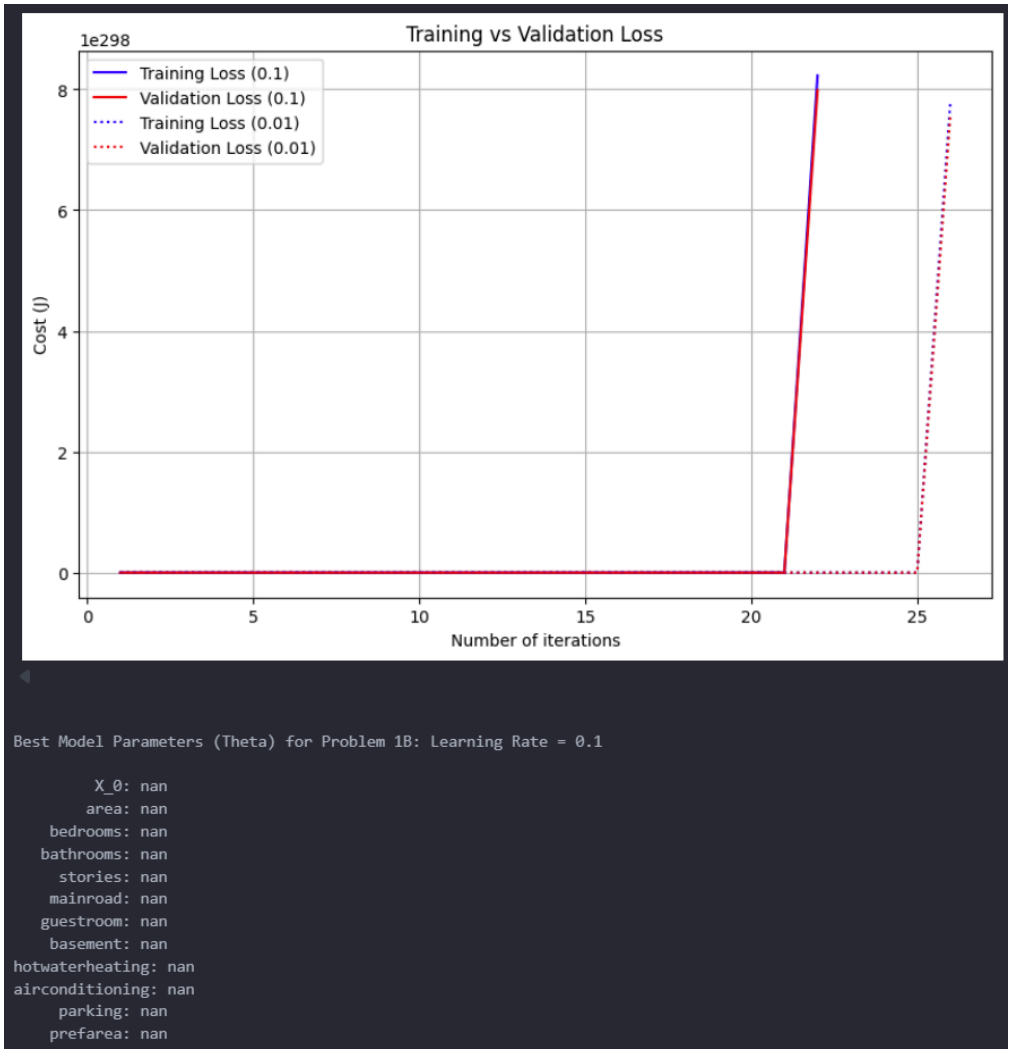
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
file_path = "Housing.csv"
sample = df = pd.read_csv(file_path)
```

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

X = sample[inputs].values
Y = sample.values[:, 0]; #Price

binary_cols = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',
'airconditioning', 'prefarea']
for col in binary_cols:
    sample[col] = sample[col].map({'yes': 1, 'no': 0})
m = len(Y)
X_0 = np.ones((m, 1))
X_cols = [sample['area'].values.reshape(m,1),
          sample['bedrooms'].values.reshape(m,1),
          sample['bathrooms'].values.reshape(m,1),
          sample['stories'].values.reshape(m,1),
          sample['mainroad'].values.reshape(m,1),
          sample['guestroom'].values.reshape(m,1),
          sample['basement'].values.reshape(m,1),
          sample['hotwaterheating'].values.reshape(m,1),
          sample['airconditioning'].values.reshape(m,1),
          sample['parking'].values.reshape(m,1),
          sample['prefarea'].values.reshape(m,1)]

X = np.hstack([X_0] + X_cols)
theta = np.zeros(X.shape[1])
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

# defining function for computing the cost for linear regression
def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    errors = predictions - y
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J

# defining function for gradient descent algorithm
```

```
def gradient_descent(Xt, Yt, Xv, Yv, theta, alpha, iterations):
  train_cost_history = np.zeros(iterations)
  valid_cost_history = np.zeros(iterations)
  m = len(Yt)
  for i in range(iterations):
    predictions = Xt.dot(theta)
    errors = np.subtract(predictions, Yt)
    sum_delta = (alpha / m) * Xt.transpose().dot(errors)
    theta = theta - sum_delta
    train_cost_history[i] = compute_cost(Xt, Yt, theta)
    valid_cost_history[i] = compute_cost(Xv, Yv, theta)
  return theta, train_cost_history, valid_cost_history

theta = np.zeros(X.shape[1])
iterations = 1000

alpha = 0.1
theta_1, train_cost_history_1, valid_cost_history_1  =
gradient_descent(X_train, Y_train, X_test, Y_test, theta, alpha,
iterations) # 0.1 Learning Rate

alpha = 0.01
theta_2, train_cost_history_2, valid_cost_history_2  =
gradient_descent(X_train, Y_train, X_test, Y_test, theta, alpha,
iterations) # 0.01 Learning Rate

plt.plot(range(1, iterations+1), train_cost_history_1, color='blue',
label='Training Loss (0.1)')
plt.plot(range(1, iterations+1), valid_cost_history_1, color='red',
label='Validation Loss (0.1)')
plt.plot(range(1, iterations+1), train_cost_history_2, color='blue',
linestyle='dotted', label='Training Loss (0.01)')
plt.plot(range(1, iterations+1), valid_cost_history_2, color='red',
linestyle='dotted', label='Validation Loss (0.01)')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Training vs Validation Loss')
plt.legend()
```

```
plt.show()

features = ['X_0', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'parking',
'prefarea']

print("\nBest Model Parameters (Theta) for Problem 1B: Learning Rate =
0.1\n")
theta_list = theta_1.flatten()
for label, value in zip(features, theta_list):
    print(f"{label:>12}: {value:,.2f}")
```

Figures / Results (1B):




```
Best Model Parameters (Theta) for Problem 1B: Learning Rate = 0.1

        X_0: nan
       area: nan
   bedrooms: nan
  bathrooms: nan
    stories: nan
   mainroad: nan
  guestroom: nan
   basement: nan
hotwaterheating: nan
airconditioning: nan
    parking: nan
   prefarea: nan
```

Problem 2A

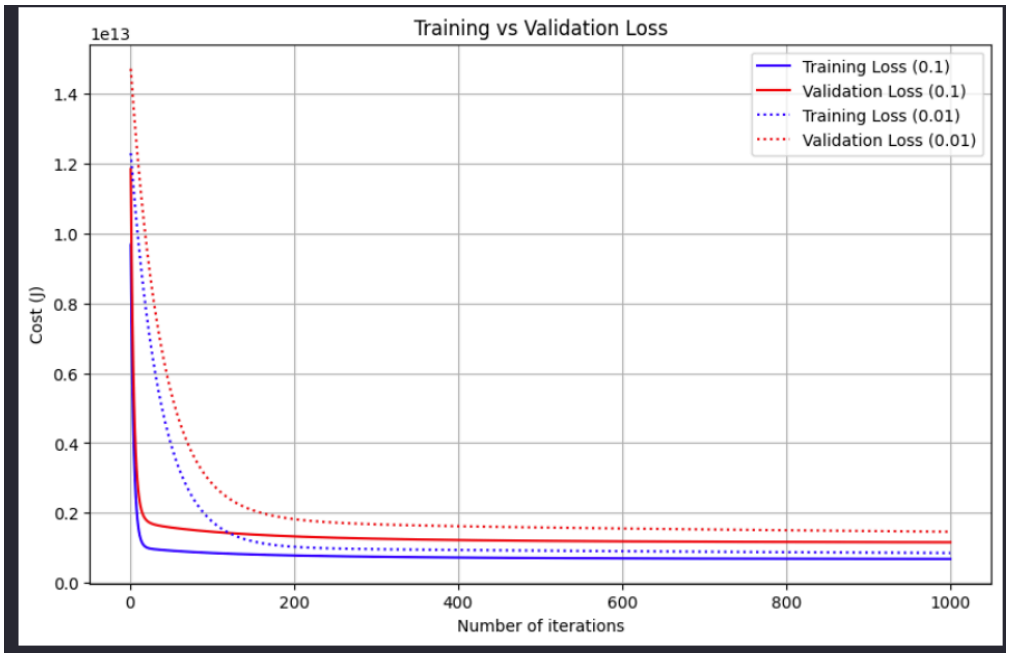
Source Code (2A):

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
file_path = "Housing.csv"
sample = df = pd.read_csv(file_path)
inputs = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking'];

X = sample[inputs].values
Y = sample.values[:, 0]; #Price
m = len(Y)
X_0 = np.ones((m, 1))
X = np.hstack((X_0, X))
theta = np.zeros(X.shape[1])

scaler = MinMaxScaler()
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)
X_train[:, 1:] = scaler.fit_transform(X_train[:, 1:])
X_test[:, 1:] = scaler.transform(X_test[:, 1:])

# defining function for computing the cost for linear regression
def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    errors = predictions - y
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J

# defining function for gradient descent algorithm
def gradient_descent(Xt, Yt, Xv, Yv, theta, alpha, iterations):
  train_cost_history = np.zeros(iterations)
  valid_cost_history = np.zeros(iterations)
  m = len(Yt)
  for i in range(iterations):
    predictions = Xt.dot(theta)
    errors = np.subtract(predictions, Yt)
```

```
    sum_delta = (alpha / m) * Xt.transpose().dot(errors)
    theta = theta - sum_delta
    train_cost_history[i] = compute_cost(Xt, Yt, theta)
    valid_cost_history[i] = compute_cost(Xv, Yv, theta)
  return theta, train_cost_history, valid_cost_history

theta = np.zeros(X.shape[1])
iterations = 1000

alpha = 0.1
theta_1, train_cost_history_1, valid_cost_history_1  =
gradient_descent(X_train, Y_train, X_test, Y_test, theta, alpha,
iterations) # 0.1 Learning Rate

alpha = 0.01
theta_2, train_cost_history_2, valid_cost_history_2  =
gradient_descent(X_train, Y_train, X_test, Y_test, theta, alpha,
iterations) # 0.01 Learning Rate

plt.plot(range(1, iterations+1), train_cost_history_1, color='blue',
label='Training Loss (0.1)')
plt.plot(range(1, iterations+1), valid_cost_history_1, color='red',
label='Validation Loss (0.1)')
plt.plot(range(1, iterations+1), train_cost_history_2, color='blue',
linestyle='dotted', label='Training Loss (0.01)')
plt.plot(range(1, iterations+1), valid_cost_history_2, color='red',
linestyle='dotted', label='Validation Loss (0.01)')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Training vs Validation Loss')
plt.legend()
plt.show()

features = ['X_0', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']

print("\nBest Model Parameters (Theta) for Problem 2B - Normalized:
Learning Rate = 0.1\n")
theta_list = theta_1.flatten()
```

```
for label, value in zip(features, theta_list):
    print(f"{label:>12}: {value:,.2f}")
```

Figures / Results (2A):




```
Best Model Parameters (Theta) for Problem 2A - Normalized: Learning Rate = 0.1

       X_0: 2,369,619.59
      area: 4,019,699.93
  bedrooms: 1,111,726.75
 bathrooms: 3,196,598.69
   stories: 1,493,003.10
   parking: 1,142,062.44
```

Problem 2B

Source Code (2B):

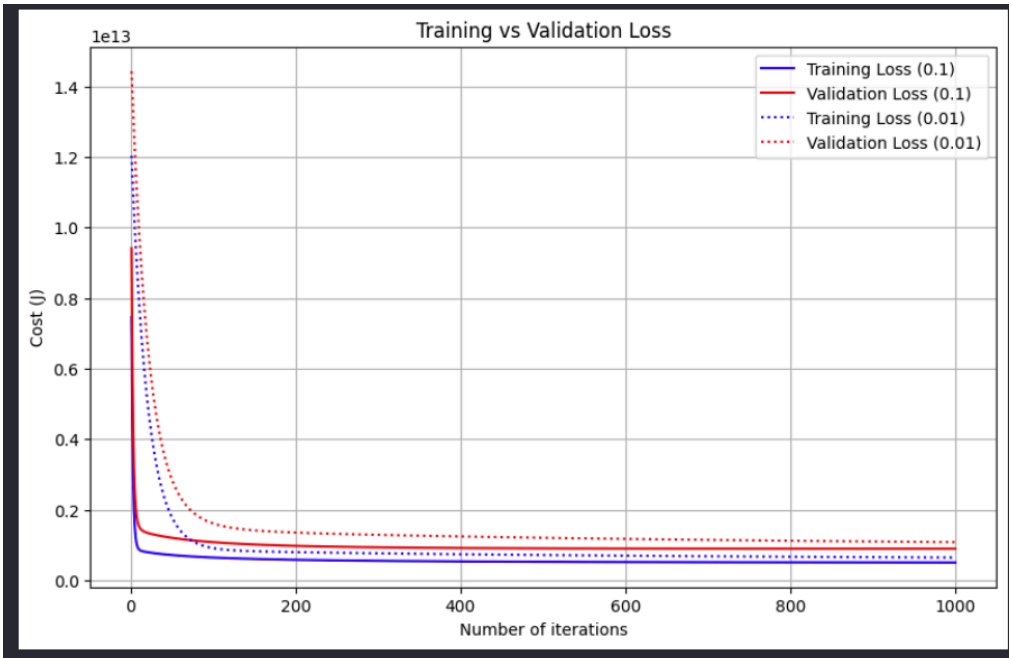
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
file_path = "Housing.csv"
sample = df = pd.read_csv(file_path)

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

```
X = sample[inputs].values
Y = sample.values[:, 0]; #Price

binary_cols = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',
'airconditioning', 'prefarea']
for col in binary_cols:
    sample[col] = sample[col].map({'yes': 1, 'no': 0})

m = len(Y)
X_0 = np.ones((m, 1))
X_cols = [sample['area'].values.reshape(m,1),
          sample['bedrooms'].values.reshape(m,1),
          sample['bathrooms'].values.reshape(m,1),
          sample['stories'].values.reshape(m,1),
          sample['mainroad'].values.reshape(m,1),
          sample['guestroom'].values.reshape(m,1),
          sample['basement'].values.reshape(m,1),
          sample['hotwaterheating'].values.reshape(m,1),
          sample['airconditioning'].values.reshape(m,1),
          sample['parking'].values.reshape(m,1),
          sample['prefarea'].values.reshape(m,1)]

X = np.hstack([X_0] + X_cols)
theta = np.zeros(X.shape[1])

scaler = MinMaxScaler()
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)
X_train[:, 1:] = scaler.fit_transform(X_train[:, 1:])
X_test[:, 1:] = scaler.transform(X_test[:, 1:])

# defining function for computing the cost for linear regression
def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    errors = predictions - y
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J
```

```
# defining function for gradient descent algorithm
def gradient_descent(Xt, Yt, Xv, Yv, theta, alpha, iterations):
  train_cost_history = np.zeros(iterations)
  valid_cost_history = np.zeros(iterations)
  m = len(Yt)
  for i in range(iterations):
    predictions = Xt.dot(theta)
    errors = np.subtract(predictions, Yt)
    sum_delta = (alpha / m) * Xt.transpose().dot(errors)
    theta = theta - sum_delta
    train_cost_history[i] = compute_cost(Xt, Yt, theta)
    valid_cost_history[i] = compute_cost(Xv, Yv, theta)
  return theta, train_cost_history, valid_cost_history

theta = np.zeros(X.shape[1])
iterations = 1000

alpha = 0.1
theta_1, train_cost_history_1, valid_cost_history_1  =
gradient_descent(X_train, Y_train, X_test, Y_test, theta, alpha,
iterations) # 0.1 Learning Rate

alpha = 0.01
theta_2, train_cost_history_2, valid_cost_history_2  =
gradient_descent(X_train, Y_train, X_test, Y_test, theta, alpha,
iterations) # 0.01 Learning Rate

plt.plot(range(1, iterations+1), train_cost_history_1, color='blue',
label='Training Loss (0.1)')
plt.plot(range(1, iterations+1), valid_cost_history_1, color='red',
label='Validation Loss (0.1)')
plt.plot(range(1, iterations+1), train_cost_history_2, color='blue',
linestyle='dotted', label='Training Loss (0.01)')
plt.plot(range(1, iterations+1), valid_cost_history_2, color='red',
linestyle='dotted', label='Validation Loss (0.01)')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Training vs Validation Loss')
```

```
plt.legend()
plt.show()

features = ['X_0', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']

print("\nBest Model Parameters (Theta) for Problem 2A - Normalized:
Learning Rate = 0.1\n")
theta_list = theta_1.flatten()
for label, value in zip(features, theta_list):
    print(f"{label:>12}: {value:,.2f}")
```

Figures / Results (2B):




```
Best Model Parameters (Theta) for Problem 2B - Normalized: Learning Rate = 0.1

        X_0: 1,848,477.44
       area: 2,997,408.53
   bedrooms: 802,331.69
  bathrooms: 2,959,618.99
    stories: 1,248,874.15
   mainroad: 466,323.29
  guestroom: 278,591.23
   basement: 416,674.93
hotwaterheating: 721,736.68
airconditioning: 830,060.67
    parking: 840,903.02
   prefarea: 643,925.90
```

Problem 3A

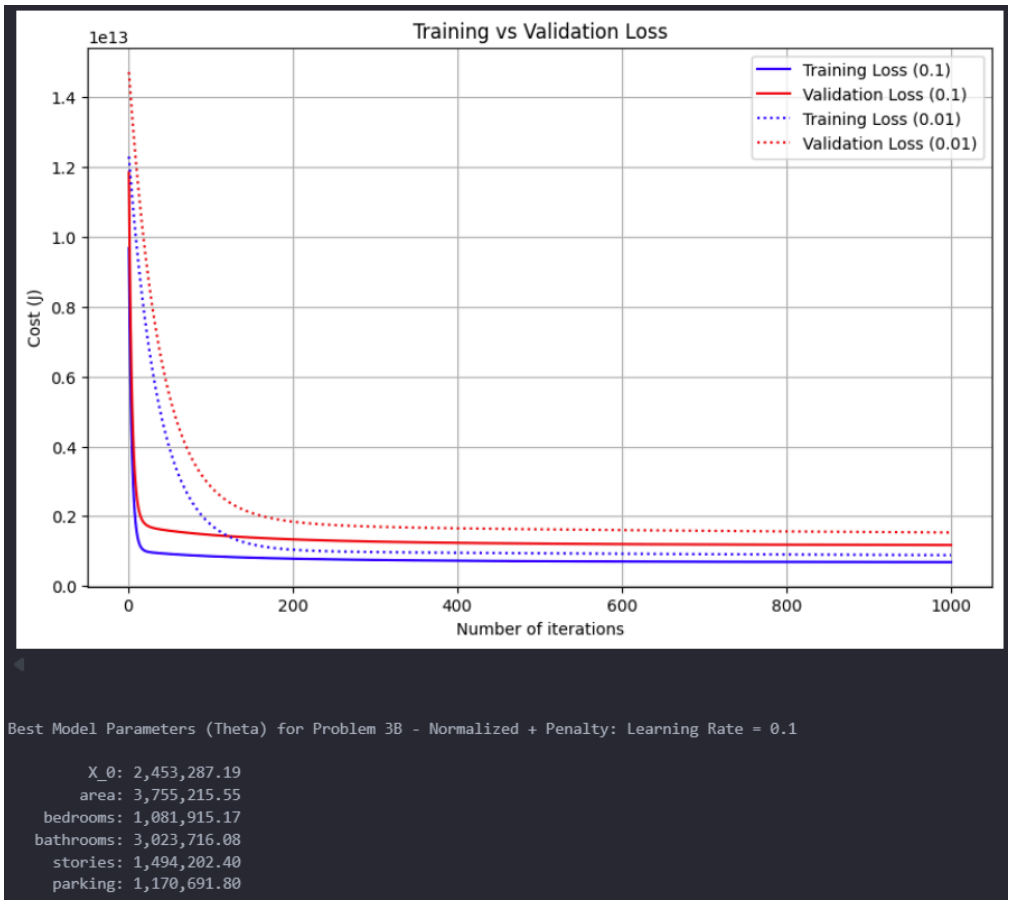
Source Code (3A):

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
file_path = "Housing.csv"
sample = df = pd.read_csv(file_path)

inputs = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking'];
X = sample[inputs].values
Y = sample.values[:, 0]; #Price
m = len(Y)
X_0 = np.ones((m, 1))
X = np.hstack((X_0, X))
theta = np.zeros(X.shape[1])

scaler = MinMaxScaler()
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)
X_train[:, 1:] = scaler.fit_transform(X_train[:, 1:])
X_test[:, 1:] = scaler.transform(X_test[:, 1:])

# defining function for computing the cost for linear regression
def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    errors = predictions - y
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J

# defining function for gradient descent algorithm
def gradient_descent(Xt, Yt, Xv, Yv, theta, alpha, iterations, lam=0.1):
  train_cost_history = np.zeros(iterations)
  valid_cost_history = np.zeros(iterations)
  m = len(Yt)
  for i in range(iterations):
    predictions = Xt.dot(theta)
    errors = np.subtract(predictions, Yt)
    sum_delta = (alpha / m) * Xt.transpose().dot(errors)
```

```
    sum_delta[1:] += (lam/m) * theta[1:]
    theta = theta - sum_delta
    train_cost_history[i] = compute_cost(Xt, Yt, theta)
    valid_cost_history[i] = compute_cost(Xv, Yv, theta)
  return theta, train_cost_history, valid_cost_history

theta = np.zeros(X.shape[1])
iterations = 1000

alpha = 0.1
theta_1, train_cost_history_1, valid_cost_history_1  =
gradient_descent(X_train, Y_train, X_test, Y_test, theta, alpha,
iterations) # 0.1 Learning Rate

alpha = 0.01
theta_2, train_cost_history_2, valid_cost_history_2  =
gradient_descent(X_train, Y_train, X_test, Y_test, theta, alpha,
iterations) # 0.01 Learning Rate

plt.plot(range(1, iterations+1), train_cost_history_1, color='blue',
label='Training Loss (0.1)')
plt.plot(range(1, iterations+1), valid_cost_history_1, color='red',
label='Validation Loss (0.1)')
plt.plot(range(1, iterations+1), train_cost_history_2, color='blue',
linestyle='dotted', label='Training Loss (0.01)')
plt.plot(range(1, iterations+1), valid_cost_history_2, color='red',
linestyle='dotted', label='Validation Loss (0.01)')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Training vs Validation Loss')
plt.legend()
plt.show()

features = ['X_0', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']

print("\nBest Model Parameters (Theta) for Problem 3B - Normalized +
Penalty: Learning Rate = 0.1\n")
theta_list = theta_1.flatten()
```

```
for label, value in zip(features, theta_list):
    print(f"{label:>12}: {value:,.2f}")
```

Figures / Results (3A):




```
Best Model Parameters (Theta) for Problem 3B - Normalized + Penalty: Learning Rate = 0.1

       X_0: 2,453,287.19
      area: 3,755,215.55
  bedrooms: 1,081,915.17
 bathrooms: 3,023,716.08
   stories: 1,494,202.40
   parking: 1,170,691.80
```

Problem 3B

Source Code (3B):

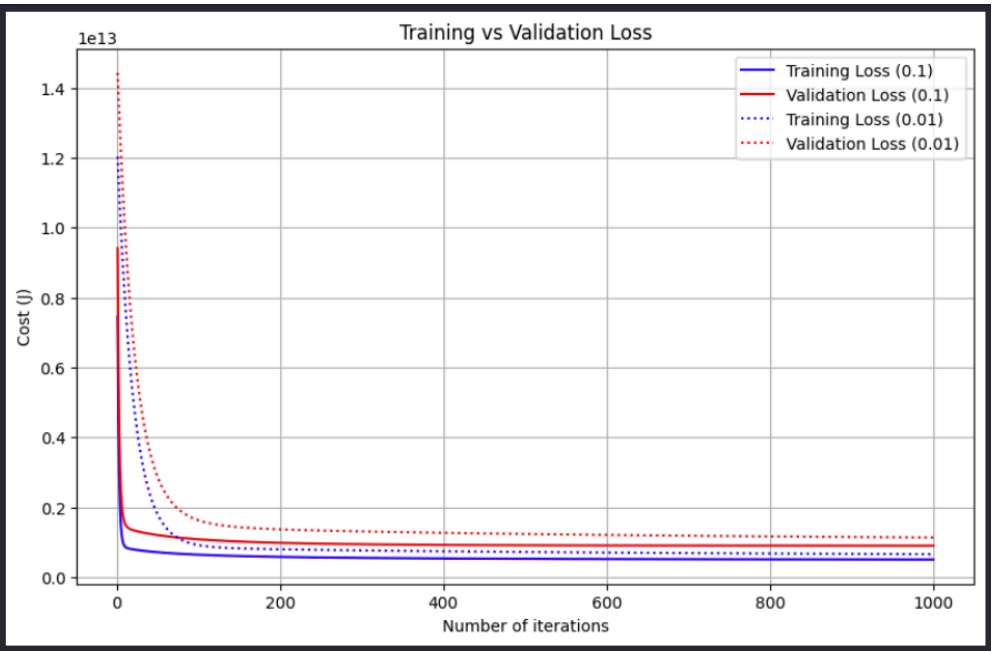
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
file_path = "Housing.csv"
sample = df = pd.read_csv(file_path)

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

```
X = sample[inputs].values
Y = sample.values[:, 0]; #Price

binary_cols = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',
'airconditioning', 'prefarea']
for col in binary_cols:
    sample[col] = sample[col].map({'yes': 1, 'no': 0})

m = len(Y)
X_0 = np.ones((m, 1))
X_cols = [sample['area'].values.reshape(m,1),
          sample['bedrooms'].values.reshape(m,1),
          sample['bathrooms'].values.reshape(m,1),
          sample['stories'].values.reshape(m,1),
          sample['mainroad'].values.reshape(m,1),
          sample['guestroom'].values.reshape(m,1),
          sample['basement'].values.reshape(m,1),
          sample['hotwaterheating'].values.reshape(m,1),
          sample['airconditioning'].values.reshape(m,1),
          sample['parking'].values.reshape(m,1),
          sample['prefarea'].values.reshape(m,1)]

X = np.hstack([X_0] + X_cols)
theta = np.zeros(X.shape[1])

scaler = MinMaxScaler()
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)
X_train[:, 1:] = scaler.fit_transform(X_train[:, 1:])
X_test[:, 1:] = scaler.transform(X_test[:, 1:])

# defining function for computing the cost for linear regression
def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    errors = predictions - y
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J
```

```
# defining function for gradient descent algorithm
def gradient_descent(Xt, Yt, Xv, Yv, theta, alpha, iterations, lam=0.1):
  train_cost_history = np.zeros(iterations)
  valid_cost_history = np.zeros(iterations)
  m = len(Yt)
  for i in range(iterations):
    predictions = Xt.dot(theta)
    errors = np.subtract(predictions, Yt)
    sum_delta = (alpha / m) * Xt.transpose().dot(errors)
    sum_delta[1:] += (lam/m) * theta[1:]
    theta = theta - sum_delta
    train_cost_history[i] = compute_cost(Xt, Yt, theta)
    valid_cost_history[i] = compute_cost(Xv, Yv, theta)
  return theta, train_cost_history, valid_cost_history

theta = np.zeros(X.shape[1])
iterations = 1000

alpha = 0.1
theta_1, train_cost_history_1, valid_cost_history_1  =
gradient_descent(X_train, Y_train, X_test, Y_test, theta, alpha,
iterations) # 0.1 Learning Rate

alpha = 0.01
theta_2, train_cost_history_2, valid_cost_history_2  =
gradient_descent(X_train, Y_train, X_test, Y_test, theta, alpha,
iterations) # 0.01 Learning Rate

plt.plot(range(1, iterations+1), train_cost_history_1, color='blue',
label='Training Loss (0.1)')
plt.plot(range(1, iterations+1), valid_cost_history_1, color='red',
label='Validation Loss (0.1)')
plt.plot(range(1, iterations+1), train_cost_history_2, color='blue',
linestyle='dotted', label='Training Loss (0.01)')
plt.plot(range(1, iterations+1), valid_cost_history_2, color='red',
linestyle='dotted', label='Validation Loss (0.01)')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
```

```
plt.title('Training vs Validation Loss')
plt.legend()
plt.show()

features = ['X_0', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'parking',
'prefarea']

print("\nBest Model Parameters (Theta) for Problem 3B - Normalized +
Penalty: Learning Rate = 0.1\n")
theta_list = theta_1.flatten()
for label, value in zip(features, theta_list):
    print(f"{label:>12}: {value:,.2f}")
```

Figure / Results (3B):




```
Best Model Parameters (Theta) for Problem 3B - Normalized + Penatly: Learning Rate = 0.1

         X_0: 1,900,563.24
        area: 2,796,088.95
    bedrooms: 786,167.01
   bathrooms: 2,788,723.16
     stories: 1,242,479.05
    mainroad: 479,021.23
   guestroom: 291,412.56
    basement: 413,403.10
hotwaterheating: 697,124.62
airconditioning: 843,144.28
     parking: 861,042.73
    prefarea: 644,538.66
```