Homework #2

Student Name: Sam Crane

Student ID: 801101091

GitHub: https://github.com/samofuture/Intro-to-ML

In [ ]:
```
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 StandardScaler
```

In [ ]:
```
def find_loss(theta, input, expected, penalty: float = 0) -> float:
    m = len(input)
    predicted = input.dot(theta)
    difference = np.square(predicted - expected)

    reg = (penalty / (2 * m)) * np.sum(np.square(theta))

    J = 1 / (2 * m) * np.sum(difference) + reg
    return J
```

In [ ]:
```
def validate(test_x, test_y, thetas) -> float:
    loss = 0
    for input, expected in zip(test_x, test_y):
        loss += find_loss(thetas, input, expected)
    return loss
```

In [ ]:
```
def plot(j, validation_loss, title: str):
    fig, ax = plt.subplots()

    ax.plot(j, label='Training Loss')
    ax2 = ax.twinx()
    ax2.plot(validation_loss, label='Validation Loss', color='orange')
    ax.legend()
    ax.set_ylabel('Error')
    ax.set_xlabel('Iteration')
    ax.set_ylim(0, ax.get_ylim()[1])
    ax2.set_ylim(0, ax2.get_ylim()[1])

    # Combine legends for both lines
    lines1, labels1 = ax.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    ax.legend(lines1 + lines2, labels1 + labels2, loc='upper right')
    ax.set_title(title)
    plt.show()
```

In [ ]:
```
def linear_regression(factors_list, iterations, x, y, alpha, m, test_x, test
    theta = np.zeros(len(factors_list)+1)

    j : list = []
    validation_loss : list[float] = []
    total_thetas : list = []

    for i in range(iterations):
        predictions = x.dot(theta)
        errors = np.subtract(predictions, y)
        sum_delta = (alpha / m) * x.transpose().dot(errors)
        theta = theta * (1 - alpha*(penalty/m)) - sum_delta
        loss = find_loss(theta, x, y, penalty)
        j.append(loss)
        v = validate(test_x, test_y, theta)
        validation_loss.append(v)

    total_thetas.append(theta)

    return j, validation_loss
```

In [ ]:
```
def prep_data(prep_type: str) -> pd.DataFrame:
    df = pd.read_csv("Housing.csv")

    df['mainroad'] = df['mainroad'].apply(lambda x: 1 if x == 'yes' else 0)
    df['guestroom'] = df['guestroom'].apply(lambda x: 1 if x == 'yes' else 0
    df['basement'] = df['basement'].apply(lambda x: 1 if x == 'yes' else 0)
    df['hotwaterheating'] = df['hotwaterheating'].apply(lambda x: 1 if x ==
    df['airconditioning'] = df['airconditioning'].apply(lambda x: 1 if x ==
    df['prefarea'] = df['prefarea'].apply(lambda x: 1 if x == 'yes' else 0)
    df['furnishingstatus'] = df['furnishingstatus'].apply(lambda x: 2 if x =

    temp_y = df.pop('price')
    columns_to_separate = ['mainroad', 'guestroom', 'basement', 'hotwaterhea
    bool_df = {col: df.pop(col) for col in columns_to_separate}

    # Normalization
    if prep_type == 'N':
        norm = np.linalg.norm(df)
        df = df/norm

    # Standardization
    if prep_type == 'S':
        scaler = StandardScaler()
        scaled_data = scaler.fit_transform(df)
        df = pd.DataFrame(scaled_data, columns=df.columns)

    df['price'] = temp_y
    for col in columns_to_separate:
        df[col] = bool_df[col]


    return df
```

Problem 1

In [ ]:
```
df = prep_data('0')

train_df, test_df = train_test_split(df, test_size=0.2, random_state=15)

y = np.array(train_df.pop('price'))

test_y = np.array(test_df.pop('price'))

m = len(y)
test_m = len(test_y)
```

Part A

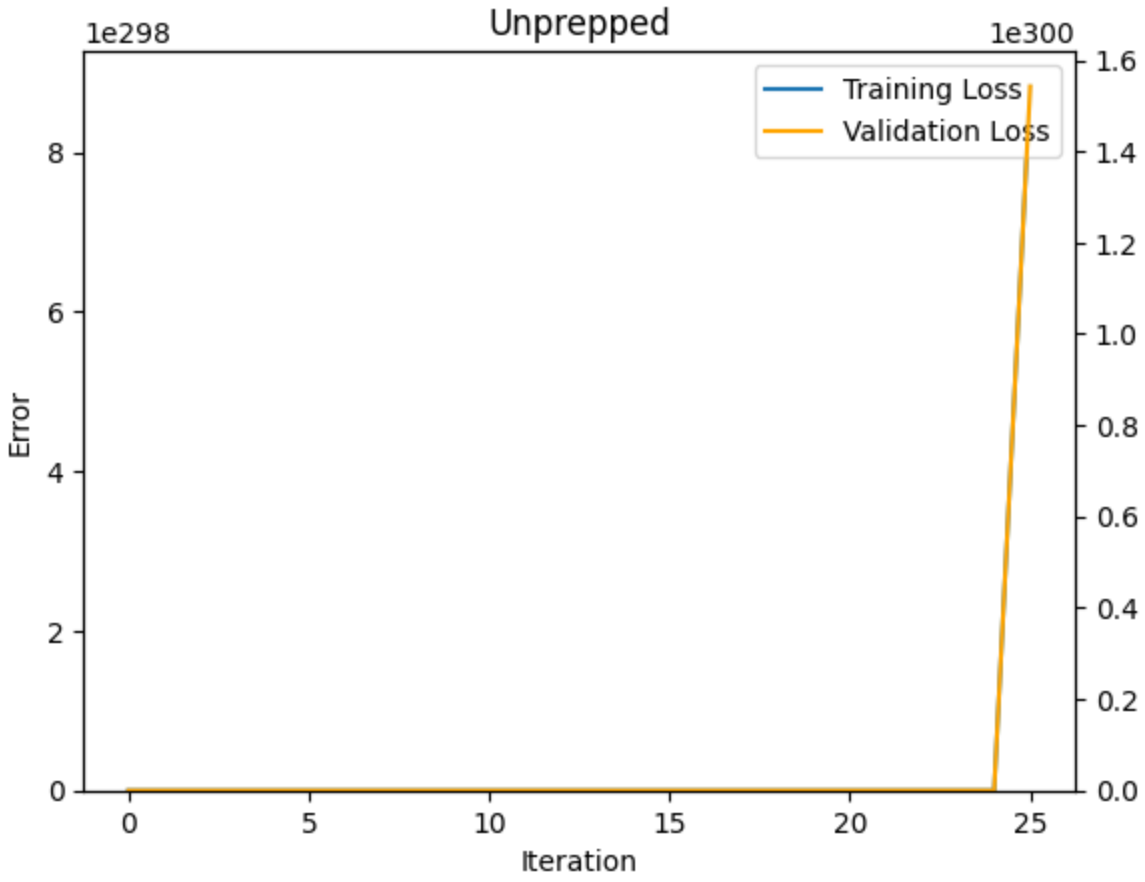
In [ ]:
```
factors_list : list[str] = ['area', 'bedrooms', 'bathrooms', 'stories', 'par
iterations = 50
alpha = 0.01

x = np.ones((m, 1))
test_x = np.ones((test_m, 1))

for factor in factors_list:
    temp = np.array(train_df[factor])
    temp = temp.reshape(m, 1)
    x = np.hstack((x, temp))
    temp = np.array(test_df[factor])
    temp = temp.reshape(test_m, 1)
    test_x = np.hstack((test_x, temp))

j, validation_loss = linear_regression(factors_list, iterations, x, y, alpha
plot(j, validation_loss, 'Unprepped')
```

```
/var/folders/g9/zflnfx0s3fd8bcn9khy4c6k40000gn/T/ipykernel_21607/4065570765.
py:4: RuntimeWarning: overflow encountered in square
  difference = np.square(predicted - expected)
/var/folders/g9/zflnfx0s3fd8bcn9khy4c6k40000gn/T/ipykernel_21607/4065570765.
py:6: RuntimeWarning: overflow encountered in square
  reg = (penalty / (2 * m)) * np.sum(np.square(theta))
/var/folders/g9/zflnfx0s3fd8bcn9khy4c6k40000gn/T/ipykernel_21607/4065570765.
py:6: RuntimeWarning: invalid value encountered in scalar multiply
  reg = (penalty / (2 * m)) * np.sum(np.square(theta))
```



Part B

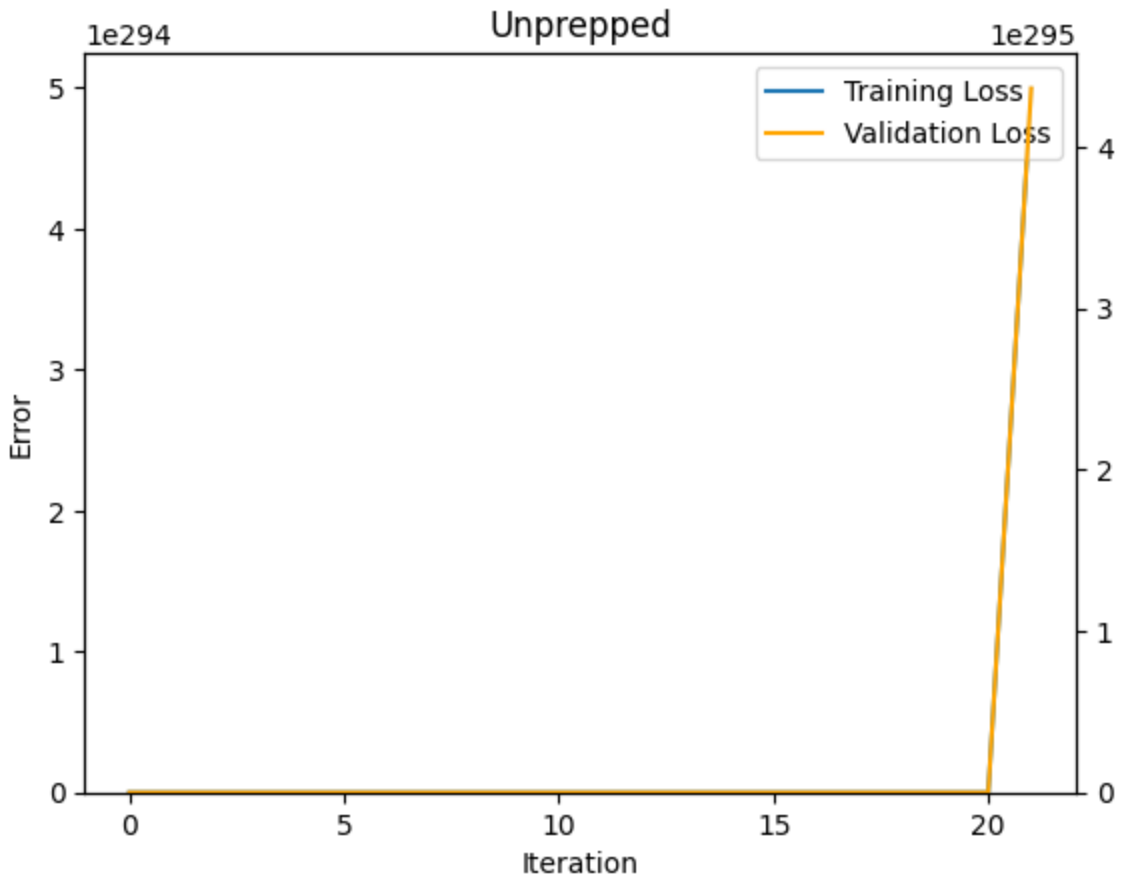
In [ ]:
```
factors_list : list[str] = ['area', 'bedrooms', 'bathrooms', 'stories', 'mai
                            'guestroom', 'basement', 'hotwaterheating', 'air
                            'parking', 'prefarea']
iterations = 50
alpha = 0.08

x = np.ones((m, 1))
test_x = np.ones((test_m, 1))

for factor in factors_list:
    temp = np.array(train_df[factor])
    temp = temp.reshape(m, 1)
    x = np.hstack((x, temp))
    temp = np.array(test_df[factor])
    temp = temp.reshape(test_m, 1)
    test_x = np.hstack((test_x, temp))

j, validation_loss = linear_regression(factors_list, iterations, x, y, alpha
plot(j, validation_loss, 'Unprepped')
```

```
/var/folders/g9/zflnfx0s3fd8bcn9khy4c6k40000gn/T/ipykernel_21607/4065570765.
py:4: RuntimeWarning: overflow encountered in square
  difference = np.square(predicted - expected)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packa
ges/numpy/core/fromnumeric.py:88: RuntimeWarning: overflow encountered in re
duce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
/var/folders/g9/zflnfx0s3fd8bcn9khy4c6k40000gn/T/ipykernel_21607/4065570765.
py:6: RuntimeWarning: overflow encountered in square
  reg = (penalty / (2 * m)) * np.sum(np.square(theta))
/var/folders/g9/zflnfx0s3fd8bcn9khy4c6k40000gn/T/ipykernel_21607/4065570765.
py:6: RuntimeWarning: invalid value encountered in scalar multiply
  reg = (penalty / (2 * m)) * np.sum(np.square(theta))
```



Both methods could not finish training here because the error was too great with all of the thetas starting at 0.

Problem 2

Normalized

In [ ]:
```
df = prep_data('N')

train_df, test_df = train_test_split(df, test_size=0.2, random_state=15)
```

```
y = np.array(train_df.pop('price'))

test_y = np.array(test_df.pop('price'))

m = len(y)
test_m = len(test_y)
```

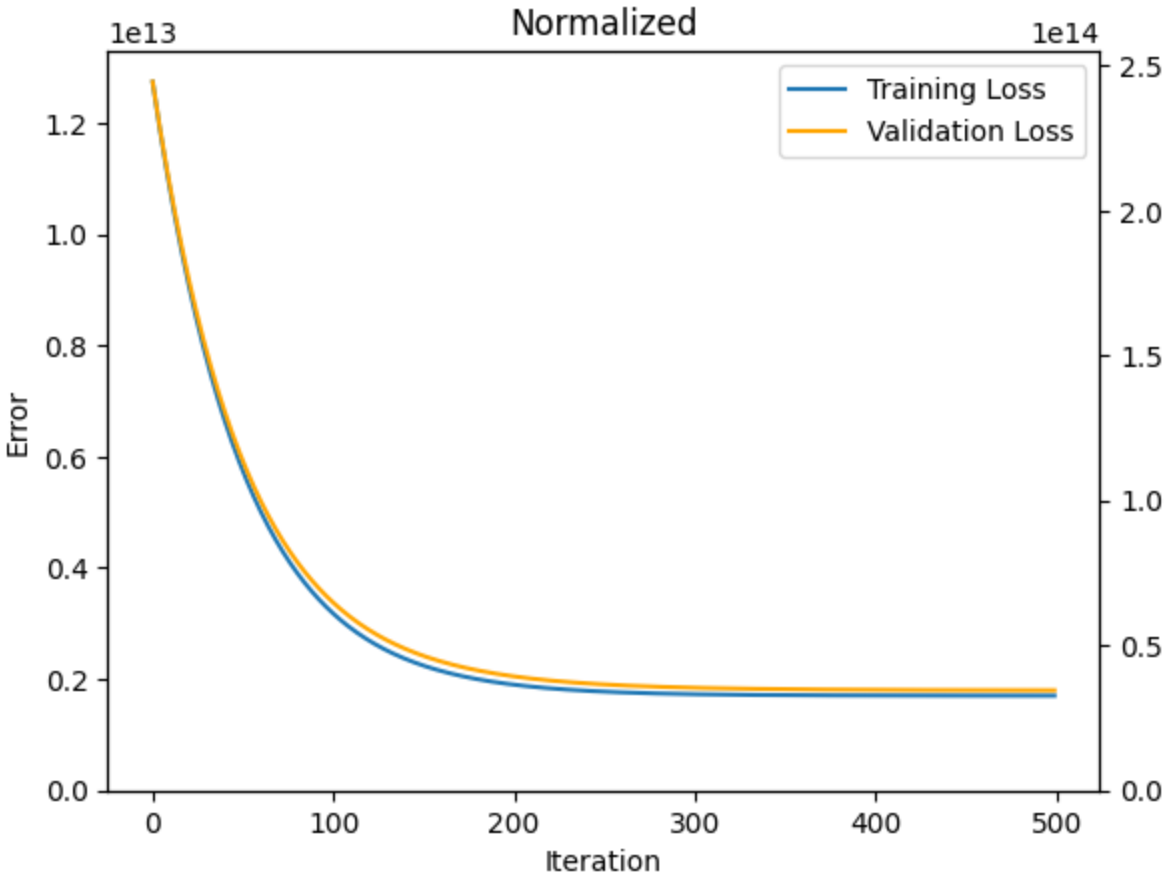
Part A

In [ ]:
```
factors_list : list[str] = ['area', 'bedrooms', 'bathrooms', 'stories', 'par
iterations = 500
alpha = 0.01

x = np.ones((m, 1))
test_x = np.ones((test_m, 1))

for factor in factors_list:
    temp = np.array(train_df[factor])
    temp = temp.reshape(m, 1)
    x = np.hstack((x, temp))
    temp = np.array(test_df[factor])
    temp = temp.reshape(test_m, 1)
    test_x = np.hstack((test_x, temp))

j, validation_loss = linear_regression(factors_list, iterations, x, y, alpha
plot(j, validation_loss, 'Normalized')
```



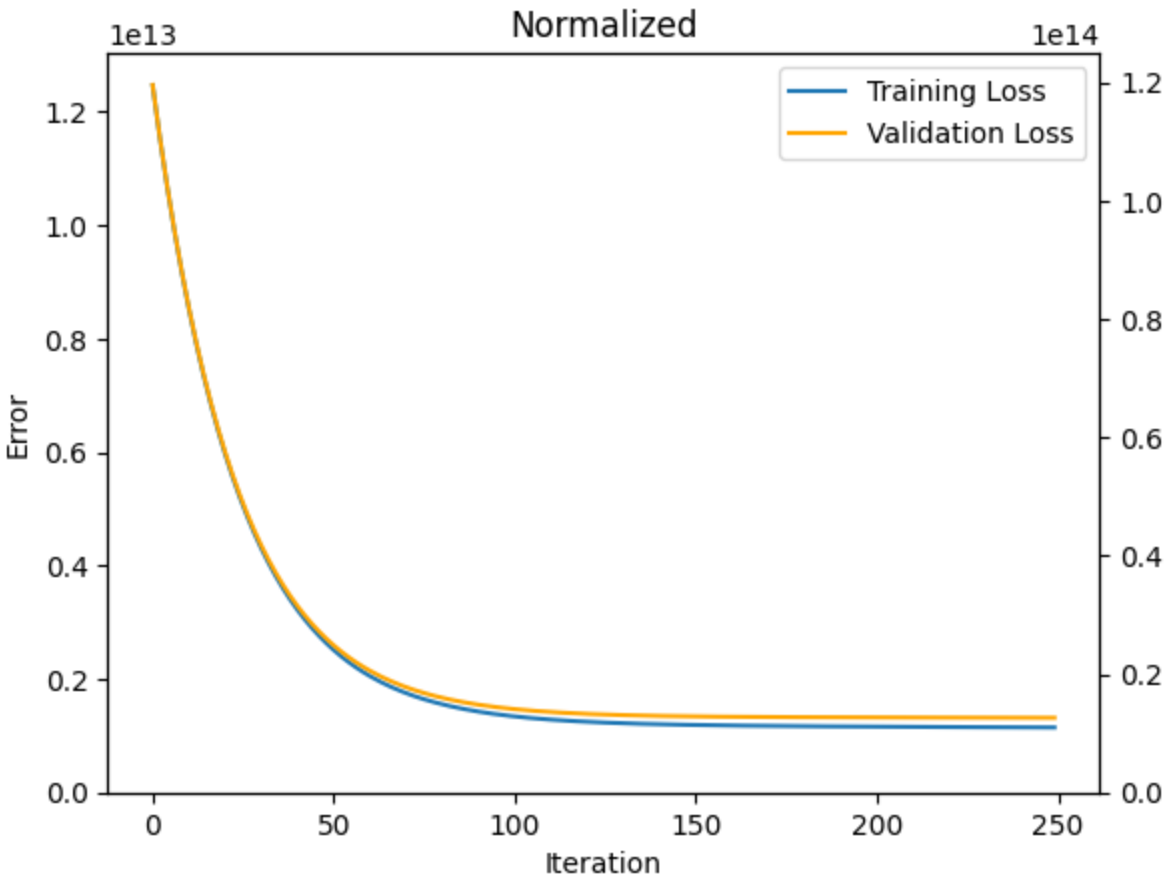
Part B

In [ ]:
```
factors_list : list[str] = ['area', 'bedrooms', 'bathrooms', 'stories', 'mai
                            'guestroom', 'basement', 'hotwaterheating', 'air
                            'parking', 'prefarea']
iterations = 250
alpha = 0.01

x = np.ones((m, 1))
test_x = np.ones((test_m, 1))

for factor in factors_list:
    temp = np.array(train_df[factor])
    temp = temp.reshape(m, 1)
    x = np.hstack((x, temp))
    temp = np.array(test_df[factor])
    temp = temp.reshape(test_m, 1)
    test_x = np.hstack((test_x, temp))

j, validation_loss = linear_regression(factors_list, iterations, x, y, alpha
plot(j, validation_loss, 'Normalized')
```



Standardized

In [ ]:
```
df = prep_data('S')
train_df, test_df = train_test_split(df, test_size=0.2, random_state=15)

y = np.array(train_df.pop('price'))

test_y = np.array(test_df.pop('price'))

m = len(y)
test_m = len(test_y)
```

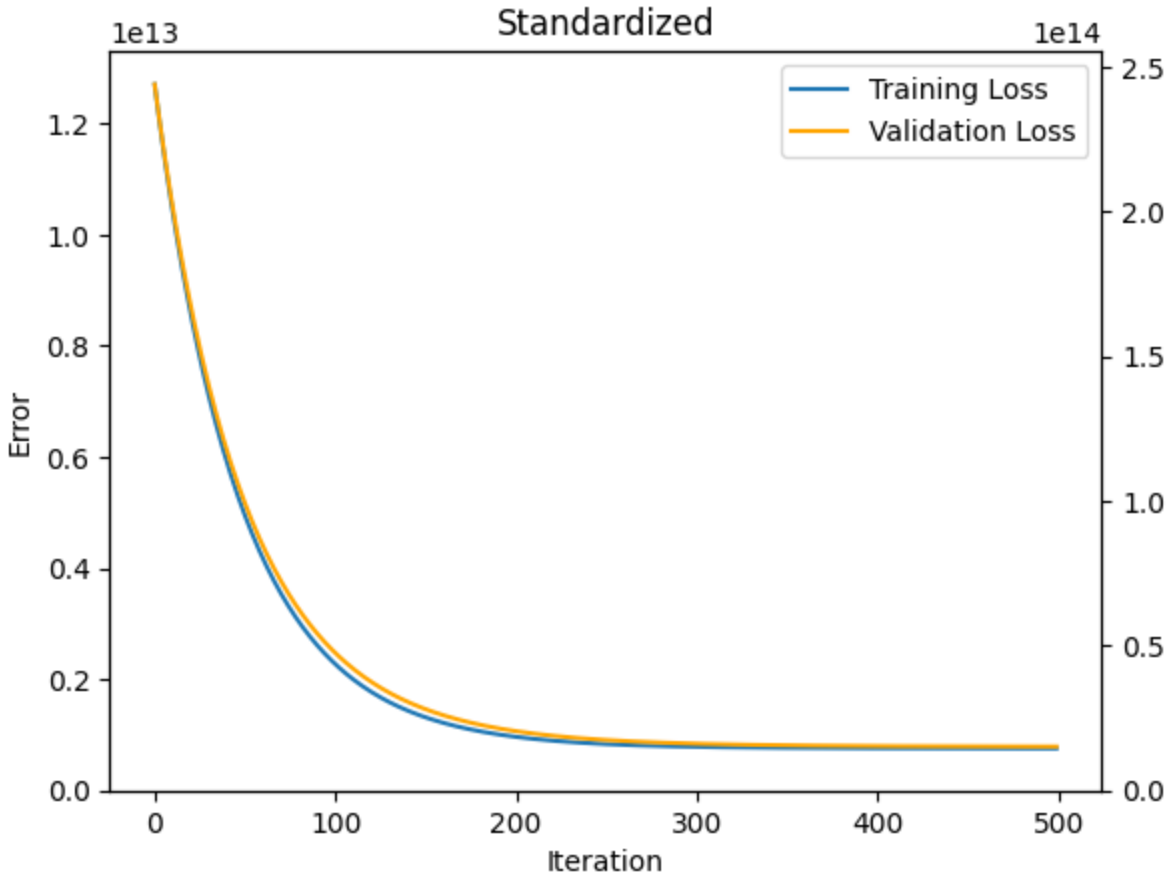
Part A

In [ ]:
```
factors_list : list[str] = ['area', 'bedrooms', 'bathrooms', 'stories', 'par
iterations = 500
alpha = 0.01

x = np.ones((m, 1))
test_x = np.ones((test_m, 1))

for factor in factors_list:
    temp = np.array(train_df[factor])
    temp = temp.reshape(m, 1)
    x = np.hstack((x, temp))
    temp = np.array(test_df[factor])
    temp = temp.reshape(test_m, 1)
    test_x = np.hstack((test_x, temp))

j, validation_loss = linear_regression(factors_list, iterations, x, y, alpha
plot(j, validation_loss, 'Standardized')
```



Part B

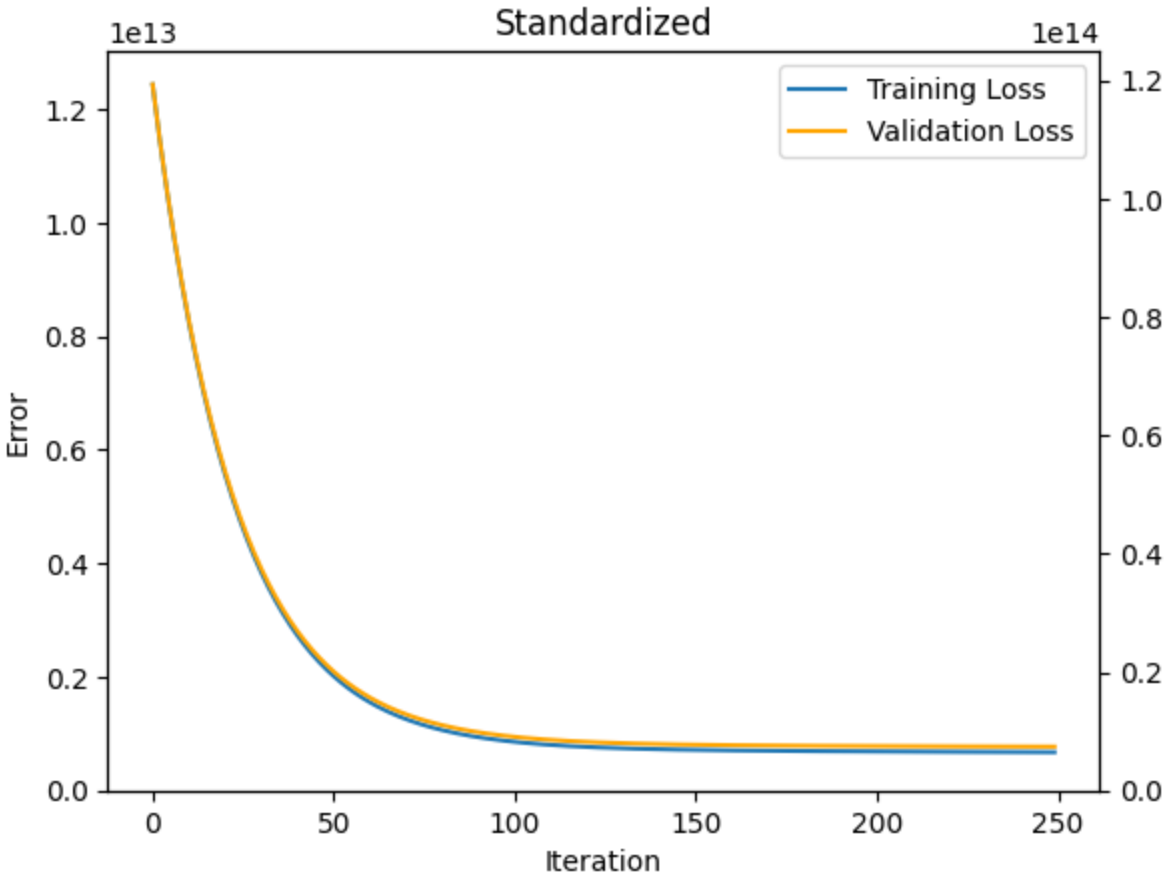
In [ ]:
```
factors_list : list[str] = ['area', 'bedrooms', 'bathrooms', 'stories', 'mai
                            'guestroom', 'basement', 'hotwaterheating', 'air
                            'parking', 'prefarea']
iterations = 250
alpha = 0.01

x = np.ones((m, 1))
test_x = np.ones((test_m, 1))

for factor in factors_list:
    temp = np.array(train_df[factor])
    temp = temp.reshape(m, 1)
    x = np.hstack((x, temp))
    temp = np.array(test_df[factor])
    temp = temp.reshape(test_m, 1)
    test_x = np.hstack((test_x, temp))

j, validation_loss = linear_regression(factors_list, iterations, x, y, alpha

plot(j, validation_loss, 'Standardized')
```



Comparing part A and B, part A had about 2x more error starting out than part B when validating the test sets. However, the loss functions on both look very similar.

Comparing between problem 1 and problem 2, it's not really a fair comparison because problem 1 couldn't finish training, but problem 2 is more accurate.

Comparing between Standardization and Normalization, standardization appeared to achieve a higher accuracy.

Problem 3

Normalized

In [ ]:
```
df = prep_data('N')

train_df, test_df = train_test_split(df, test_size=0.2, random_state=15)

y = np.array(train_df.pop('price'))

test_y = np.array(test_df.pop('price'))
```

```
m = len(y)
test_m = len(test_y)
```

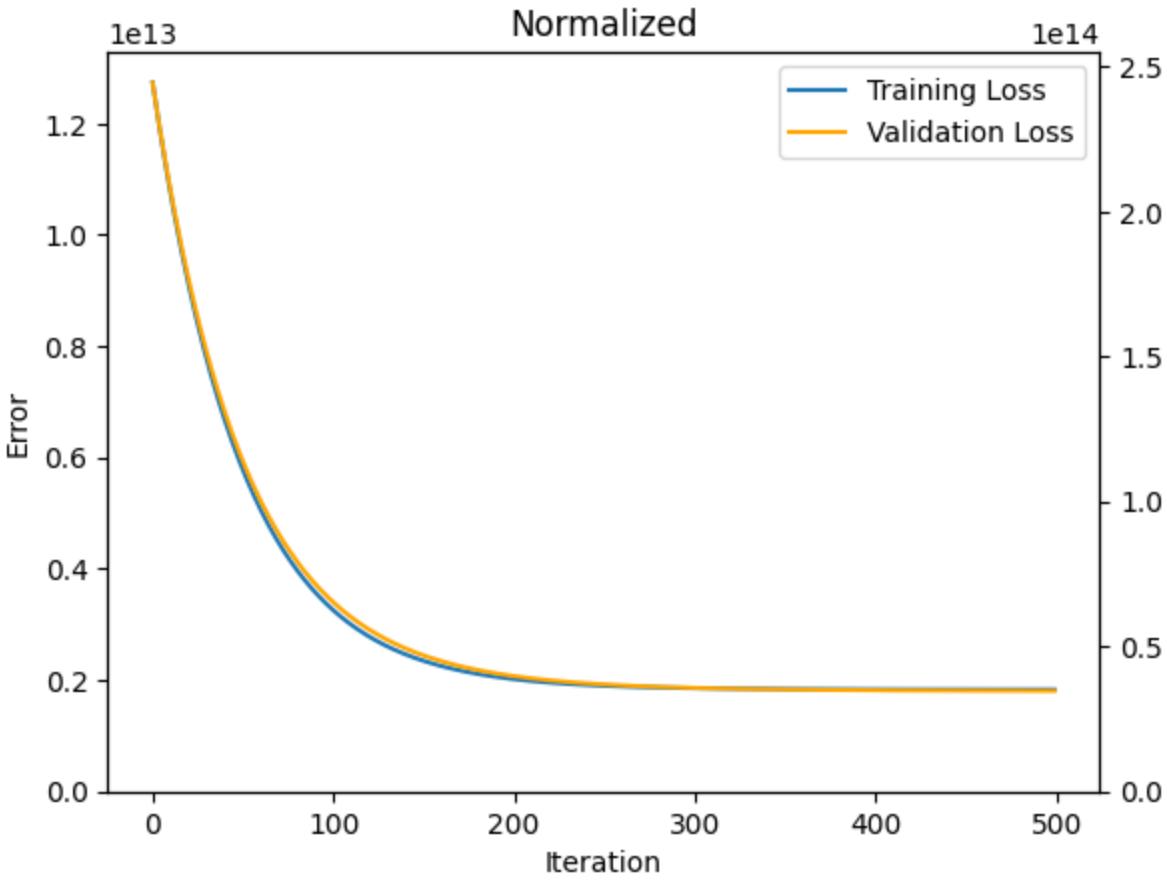
Part A

In [ ]:
```
factors_list : list[str] = ['area', 'bedrooms', 'bathrooms', 'stories', 'par
iterations = 500
alpha = 0.01
penalty = 5

x = np.ones((m, 1))
test_x = np.ones((test_m, 1))

for factor in factors_list:
    temp = np.array(train_df[factor])
    temp = temp.reshape(m, 1)
    x = np.hstack((x, temp))
    temp = np.array(test_df[factor])
    temp = temp.reshape(test_m, 1)
    test_x = np.hstack((test_x, temp))

j, validation_loss = linear_regression(factors_list, iterations, x, y, alpha
plot(j, validation_loss, 'Normalized')
```



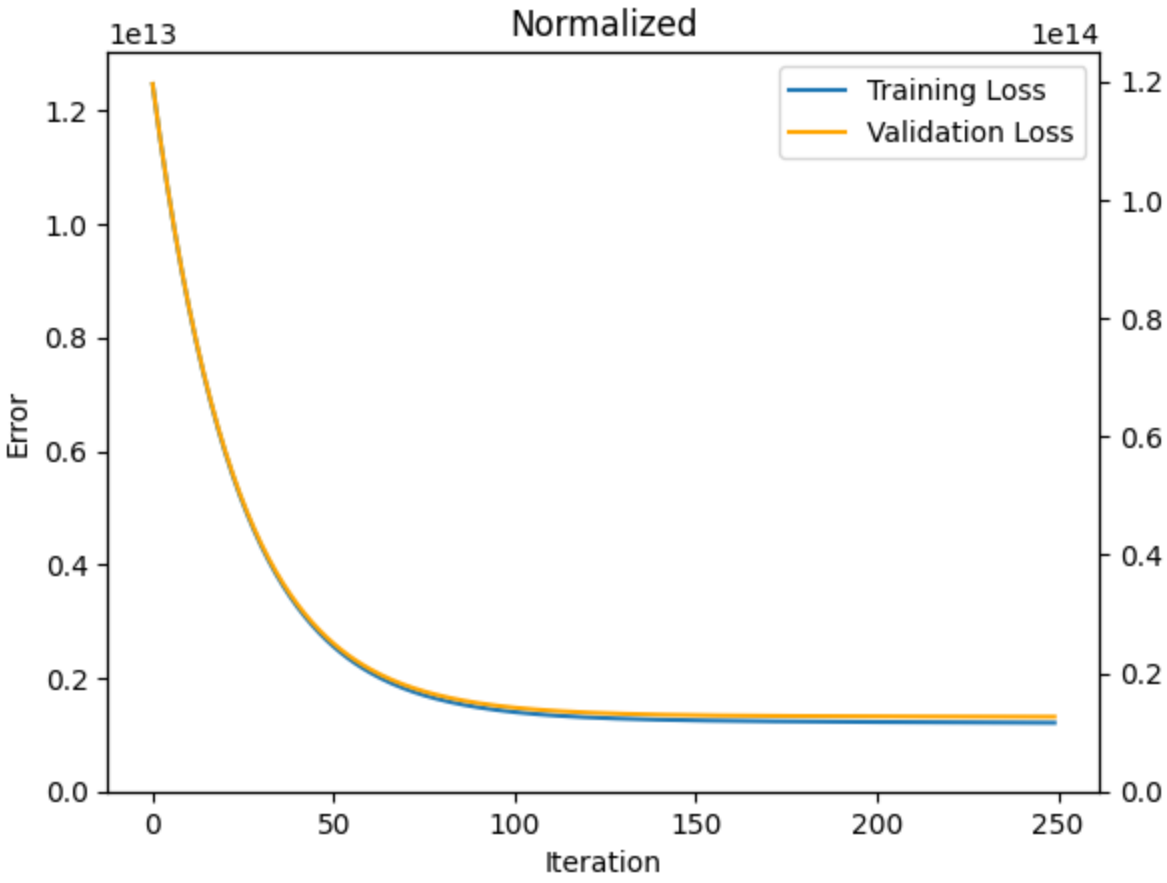
Part B

In [ ]:
```
factors_list : list[str] = ['area', 'bedrooms', 'bathrooms', 'stories', 'mai
                            'guestroom', 'basement', 'hotwaterheating', 'air
                            'parking', 'prefarea']
iterations = 250
alpha = 0.01
penalty = 5

x = np.ones((m, 1))
test_x = np.ones((test_m, 1))

for factor in factors_list:
    temp = np.array(train_df[factor])
    temp = temp.reshape(m, 1)
    x = np.hstack((x, temp))
    temp = np.array(test_df[factor])
    temp = temp.reshape(test_m, 1)
    test_x = np.hstack((test_x, temp))

j, validation_loss = linear_regression(factors_list, iterations, x, y, alpha
plot(j, validation_loss, 'Normalized')
```



Standardized

In [ ]:
```
df = prep_data('S')
train_df, test_df = train_test_split(df, test_size=0.2, random_state=15)
```

```
y = np.array(train_df.pop('price'))

test_y = np.array(test_df.pop('price'))

m = len(y)
test_m = len(test_y)
```

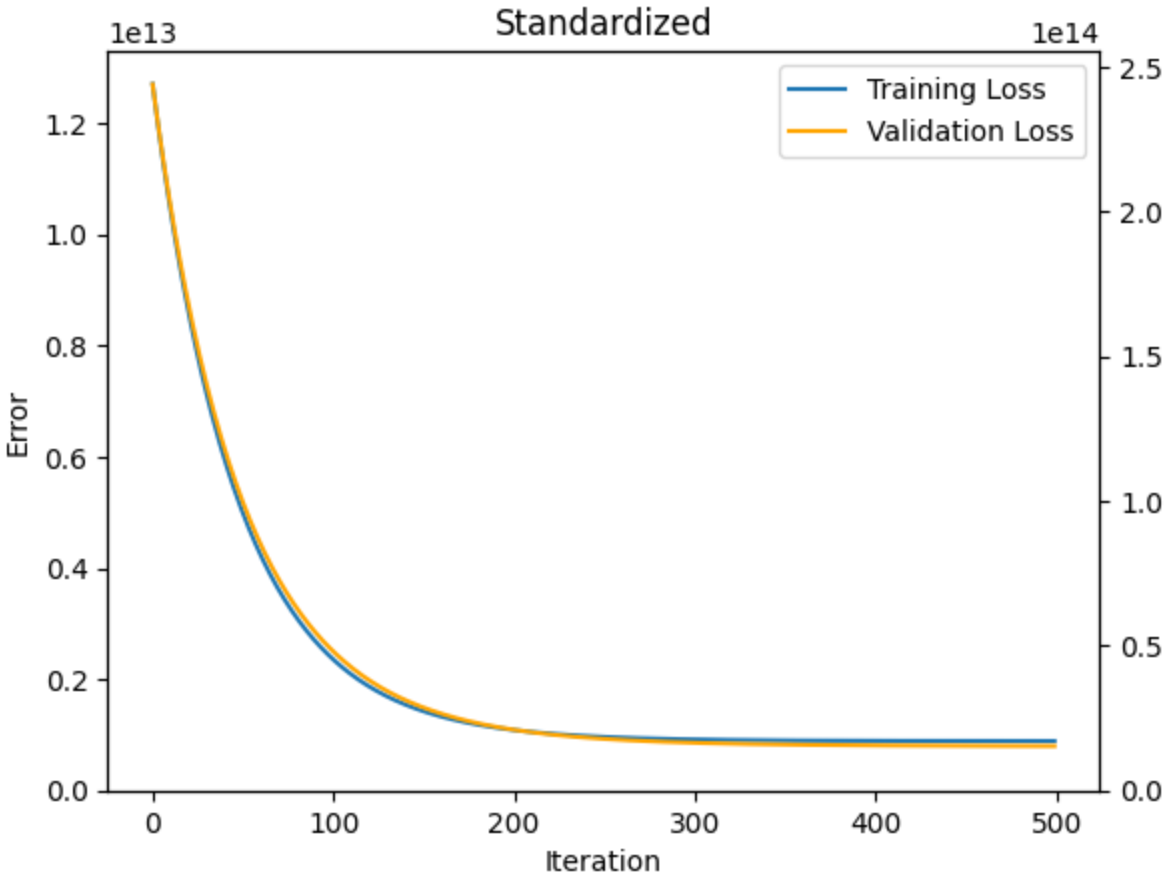
Part A

In [ ]:
```
factors_list : list[str] = ['area', 'bedrooms', 'bathrooms', 'stories', 'par
iterations = 500
alpha = 0.01
penalty = 5

x = np.ones((m, 1))
test_x = np.ones((test_m, 1))

for factor in factors_list:
    temp = np.array(train_df[factor])
    temp = temp.reshape(m, 1)
    x = np.hstack((x, temp))
    temp = np.array(test_df[factor])
    temp = temp.reshape(test_m, 1)
    test_x = np.hstack((test_x, temp))

j, validation_loss = linear_regression(factors_list, iterations, x, y, alpha
plot(j, validation_loss, 'Standardized')
```



Part B

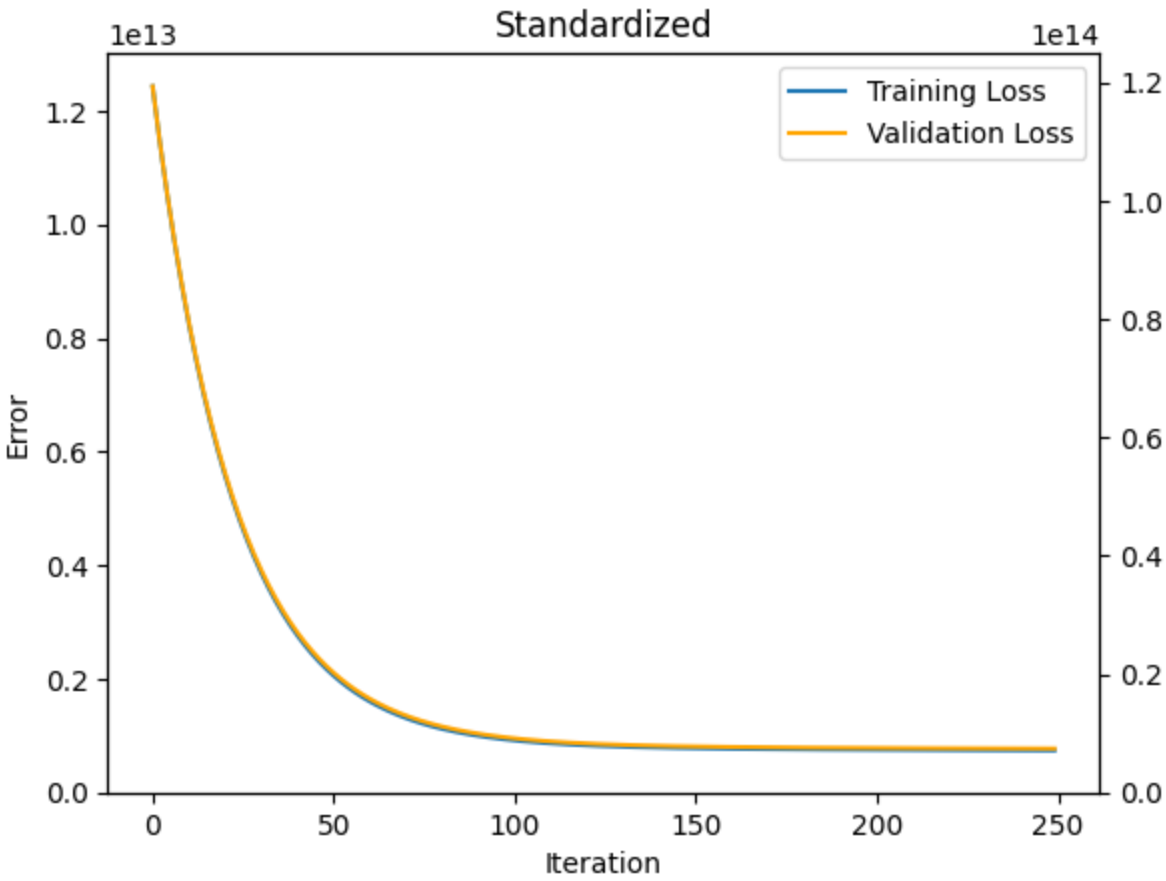
In [ ]:
```
factors_list : list[str] = ['area', 'bedrooms', 'bathrooms', 'stories', 'mai
                            'guestroom', 'basement', 'hotwaterheating', 'air
                            'parking', 'prefarea']
iterations = 250
alpha = 0.01
penalty = 5

x = np.ones((m, 1))
test_x = np.ones((test_m, 1))

for factor in factors_list:
    temp = np.array(train_df[factor])
    temp = temp.reshape(m, 1)
    x = np.hstack((x, temp))
    temp = np.array(test_df[factor])
    temp = temp.reshape(test_m, 1)
    test_x = np.hstack((test_x, temp))

j, validation_loss = linear_regression(factors_list, iterations, x, y, alpha

plot(j, validation_loss, 'Standardized')
```



The standardization methods seem to work better than normalizing here as well as in problem 2.

It looks like the error also dropped a bit from problem 2 with the addition of regularization.