ML ASSIGNMENT

Name: Nafis Islam
Roll No: 2206187
Section: IT-06

full code available at: github.com/nafis71041/linear_regression

1. (a) Use linear regression to fit a straight line to the given database. Set your learning rate to 0.5.

In [20]:
```
slope_m, intercept_c, loss = linear_regression(inputs_x_normalized, targets_y_normalized, learning_rate=0.5)
```

1. (b) What are the cost function value and learning parameters values after convergence?

In [21]:
```
print(f'm = {slope_m:.6f} c = {intercept_c:.6f} loss = {loss[-1]:.6f}')
```

```
m = 0.655107 c = -0.000000 loss = 0.285445
```

1. (c) Also, mention the convergence criteria you used.

Ans. Convergence Criteria: Absolute change in loss in less than 5e-9

`abs(loss[-2] - loss[-1]) < tolerance`

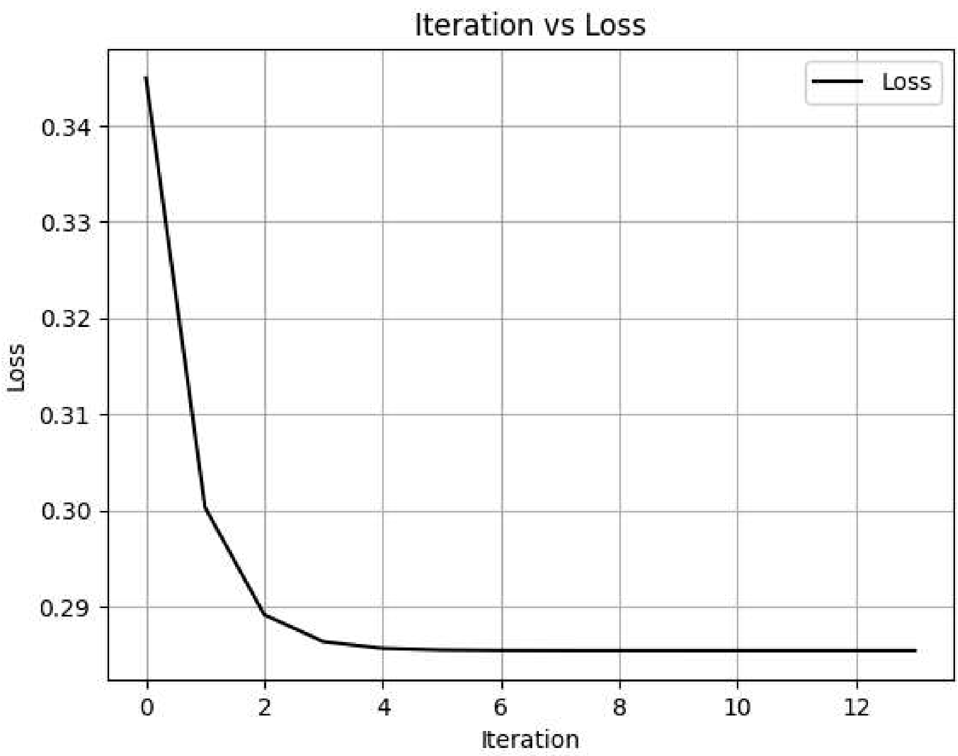
2. What is the advantage of averaging the cost?

`cost = np.mean(np.square(predictions - targets_y)) / 2`

Ans. Averaging the cost ensures that the cost function is not influenced by the number of data points, making it easier to compare models trained on datasets of different sizes.

3. Plot cost function v/s iteration graph for the model in question 1 for first 50 iterations.

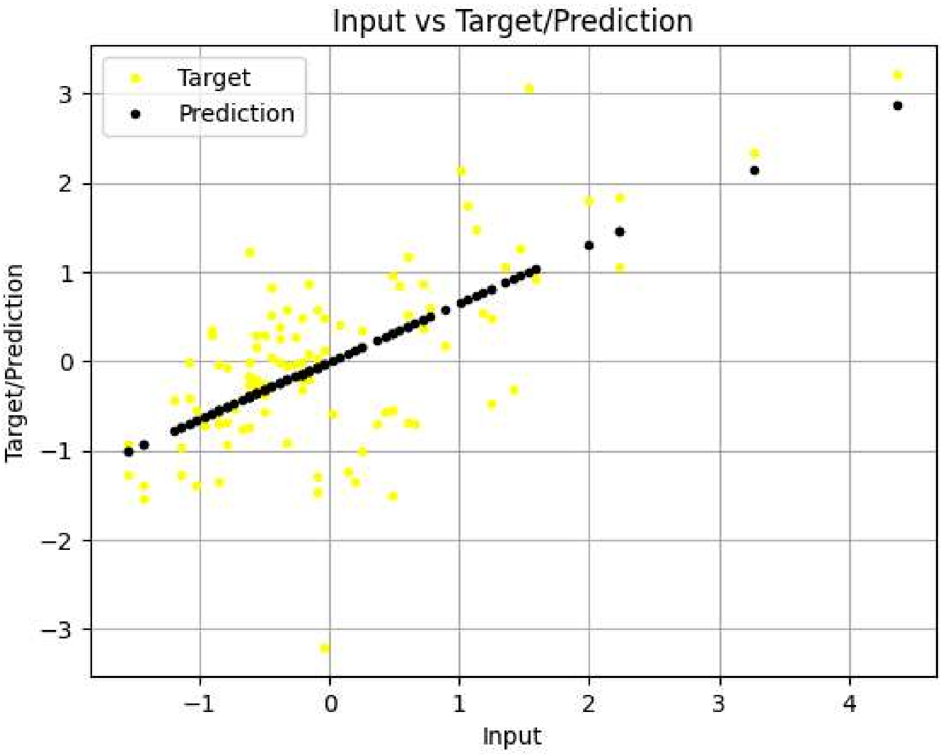
In [22]:
```
plt.plot(loss[:50], '-', color='black', label='Loss')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.title('Iteration vs Loss')
plt.grid(True)
plt.legend()
plt.show()
```



4. Plot the given dataset on a graph and also print the straight line you obtained in question 1 to show how it fits the data.

In [23]:
```
predictions = slope_m * inputs_x_normalized + intercept_c

plt.plot(inputs_x_normalized, targets_y_normalized, '.', color='yellow', label='Target')
plt.plot(inputs_x_normalized, predictions, '.', color='black', label='Prediction')
plt.xlabel('Input')
plt.ylabel('Target/Prediction')
plt.title('Input vs Target/Prediction')
plt.grid(True)
plt.legend()
plt.show()
```



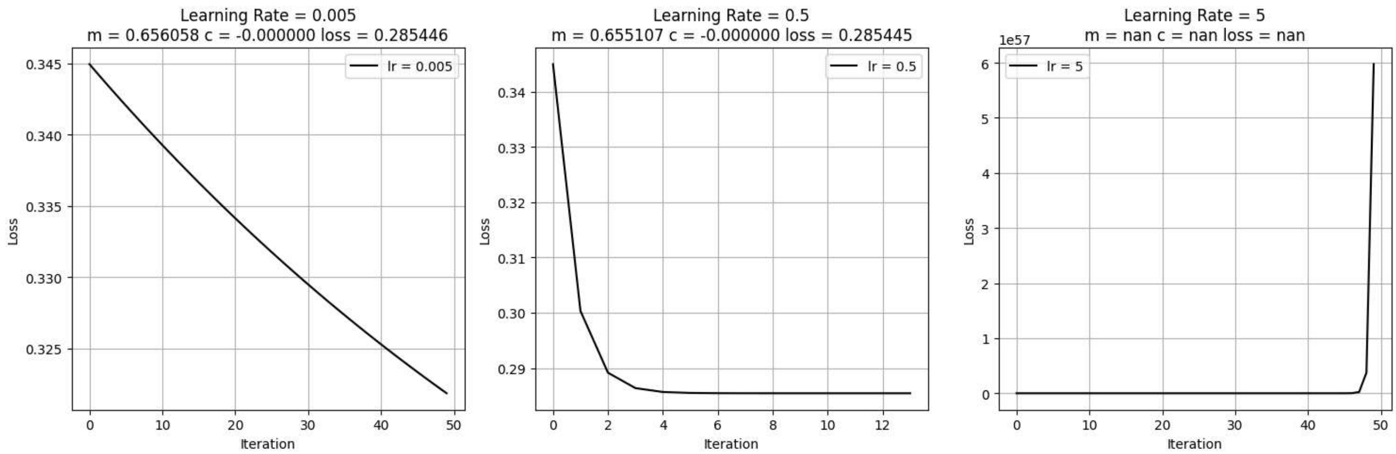
5. Test your regression model with the learning rates [0.005, 0.5, 5]. For each learning rate, plot a graph showing how the cost function changes for the first 50 iterations and write your observation.

In [24]:
```
learning_rates = [0.005, 0.5, 5]
plt.figure(figsize=(15, 5))

for i, lr in enumerate(learning_rates):
    slope_m, intercept_c, loss = linear_regression(inputs_x_normalized, targets_y_normalized, learning_rate=lr)
    plt.subplot(1, 3, i + 1)

    plt.plot(loss[:50], '-', color='black', label=f"lr = {lr}")
    plt.xlabel('Iteration')
    plt.ylabel('Loss')
    plt.title(f'Learning Rate = {lr}\nm = {slope_m:.6f} c = {intercept_c:.6f} loss = {loss[-1]:.6f}')
    plt.grid(True)
    plt.legend()

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



Ans. For lr=5 the training was neumerically unstable (giving NaN error)

For lr=0.5 and 0.005 the model converged but faster for lr=0.5

6. Choose a suitable learning rate, then implement stochastic and min-batch gradient descent, plot the cost function against iteration, and observe how your cost function changes compared to batch gradient descent.

In [26]:

```python
gradient_descents = {
    'Batch Gradient Descent': linear_regression,
    'Mini-Batch Gradient Descent': MBGD,
    'Stochastic Gradient Descent': SGD
}

plt.figure(figsize=(15, 5))

for i, (gradient_descent, func) in enumerate(gradient_descents.items()):
    slope_m, intercept_c, loss = func(inputs_x_normalized, targets_y_normalized, learning_rate=0.005)
    plt.subplot(1, 3, i + 1)

    plt.plot(loss, '-', color='black', label=f"Loss")
    plt.xlabel('Iteration')
    plt.ylabel('Loss')
    plt.title(gradient_descent + f'\nm = {slope_m:.6f} c = {intercept_c:.6f} loss = {loss[-1]:.6f}')
    plt.grid(True)
    plt.legend()

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