In [53]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
```

In [54]:
```
N = 5
iterations = 7
batch_size = 64



def gradient_descent(W, b, alpha, dW, db):
    '''
    Gradient Descent Algorithm:
    W : Weight term
    b : bias term
    alpha : learning rate
    dW : partial derivative of W value
    db : partial derivative of b value
            '''
    W = W - alpha * dW
    b = b - alpha * db
    return W,b

def error(W, b, X, y, m):
    # print(W.shape, X.shape)
    return 1/m * np.sum((np.dot(W.T,X.T)+b - y.T)**2)

def get_data():

    X = np.array([i for i in range(N)]).reshape(-1,1)
    arr1 = np.random.random(N)
    y = (2*np.array([i for i in range(N)]) + arr1 * 1).reshape(-1,1)

    W = np.array([0]).reshape(-1,1)
    b = np.array([0]).reshape(-1,1)

    return X, y, W, b
```

In [55]:
```
def batch_gradient(X, y, W, b, alpha=1e-02):
    cost = []
    for i in range(iterations):
        dW = 1* np.dot(np.dot(W.T,X.T)+b - y.T, X)
        db = 1* np.sum(np.dot(W.T,X.T)+b - y.T)
        W, b = gradient_descent(W, b, alpha, dW, db)
        cost.append(error(W, b, X, y, N))

    return cost
```

In [56]:
```
def stochastic(X, y, W, b, alpha=1e-02):
    cost = []
    for i in range(iterations):

        dW = 1* np.dot(np.dot(W.T,X.T)+b - y.T, X)
```

```
        db = 1* np.sum(np.dot(W.T,X.T)+b - y.T)

        W = W - alpha * dW
        b = b - alpha * db

        cost.append(error(W, b, X, y, 1))  # 1 for stochastic (single sample)

    return cost
```

In [57]:
```
def mini_batch(X, y, W, b, alpha=1e-02):
    cost = []

    for i in range(iterations):

        X = X[i*batch_size:(i+1)*batch_size]
        y = y[i*batch_size:(i+1)*batch_size]

        dW = 1* np.dot(np.dot(W.T,X.T)+b - y.T, X)
        db = 1* np.sum(np.dot(W.T,X.T)+b - y.T)

        W = W - alpha * dW
        b = b - alpha * db

        cost.append(error(W, b, y, batch_size))

    return cost
```

In [58]:
```
def mini_batch(X, y, W, b, alpha=1e-02):
    cost = []
    for i in range(iterations):

        X = X[i*batch_size:(i+1)*batch_size]
        y = y[i*batch_size:(i+1)*batch_size]

        dW = 1* np.dot(np.dot(W.T,X.T)+b - y.T, X)
        db = 1* np.sum(np.dot(W.T,X.T)+b - y.T)

        W = W - alpha * dW
        b = b - alpha * db


        cost.append(error(W, b, X, y, batch_size))

    return cost
```

In [59]:
```
def momentum(X, y, W, b, alpha=1e-02, beta=0.9):
    VdW = 0
    Vdb = 0
    cost = []

    for i in range(iterations):
        X = X[i*batch_size:(i+1)*batch_size]
        y = y[i*batch_size:(i+1)*batch_size]

        dW = 1* np.dot(np.dot(W.T,X.T)+b - y.T, X)
```

```
        db = 1* np.sum(np.dot(W.T,X.T)+b - y.T)

        VdW = beta*VdW + (1-beta) * dW
        Vdb = beta*Vdb + (1-beta) * db

        W = W - alpha * (1/batch_size) * VdW
        b = b - alpha * (1/batch_size) * Vdb

        cost.append(error(W, b, X, y, batch_size))

    return cost
```

In [60]:

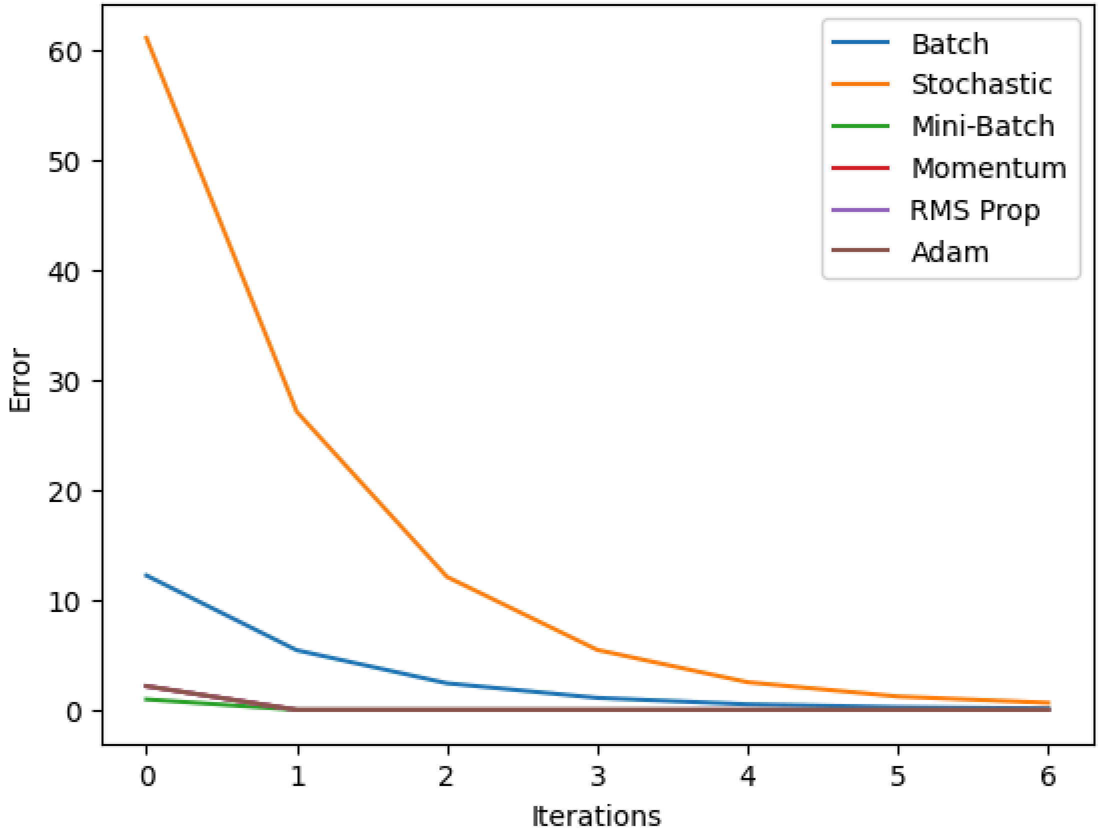
```
def RMSprop(X, y, W, b, alpha=1e-02, beta=0.9, epsilon= 1e-8):
    SdW = 0
    Sdb = 0
    cost = []

    for i in range(iterations):

        X = X[i*batch_size:(i+1)*batch_size]
        y = y[i*batch_size:(i+1)*batch_size]

        dW = 1* np.dot(np.dot(W.T,X.T)+b - y.T, X)
        db = 1* np.sum(np.dot(W.T,X.T)+b - y.T)

        SdW = beta*SdW + (1-beta) * dW**2
        Sdb = beta*Sdb + (1-beta) * db**2

        W = W - alpha * (1/batch_size) * dW/(np.sqrt(SdW) + epsilon)
        b = b - alpha * (1/batch_size) * db/(np.sqrt(Sdb) + epsilon)

        cost.append(error(W, b, X, y, batch_size))

    return cost
```

In [61]:

```
def Adam(X, y, W, b, alpha=1e-02, beta1=0.9, beta2=0.999, epsilon= 1e-8):
    SdW = 0
    Sdb = 0
    VdW = 0
    Vdb = 0
    cost = []

    for i in range(iterations):

        X = X[i*batch_size:(i+1)*batch_size]
        y = y[i*batch_size:(i+1)*batch_size]

        dW = 1* np.dot(np.dot(W.T,X.T)+b - y.T, X)
        db = 1* np.sum(np.dot(W.T,X.T)+b - y.T)

        VdW = beta2*VdW + dW
        Vdb = beta2*Vdb + db

        SdW = beta1*SdW + dW**2
        Sdb = beta1*Sdb + db**2
```

```
        VdW_corrected = VdW/(1-beta1**(i+1)) #instead of i use i+1 to resolve i=0 i
        Vdb_corrected = Vdb/(1-beta1**(i+1)) #instead of i use i+1 to resolve i=0 i

        SdW_corrected = SdW/(1-beta2**(i+1))
        Sdb_corrected = Sdb/(1-beta2**(i+1))

        W = W - alpha * (1/batch_size) * VdW_corrected/(np.sqrt(SdW_corrected) + ep
        b = b - alpha * (1/batch_size) * Vdb_corrected/(np.sqrt(Sdb_corrected) + ep

        cost.append(error(W, b, X, y, batch_size))

    return cost
```

In [66]:
```
X, y, W, b = get_data()

cost_batch = batch_gradient(X, y, W, b)
cost_stochastic = stochastic(X, y, W, b)
cost_mini = mini_batch(X, y, W, b)

cost_momentum = momentum(X, y, W, b)
cost_rms = RMSprop(X, y, W, b)
cost_adam = Adam(X, y, W, b)

plt.plot(range(iterations), cost_batch, label="Batch")
plt.plot(range(iterations), cost_stochastic, label="Stochastic")
plt.plot(range(iterations), cost_mini, label="Mini-Batch")

plt.plot(range(iterations), cost_momentum, label="Momentum")
plt.plot(range(iterations), cost_rms, label="RMS Prop")
plt.plot(range(iterations), cost_adam, label="Adam")

plt.legend()
plt.xlabel("Iterations")
plt.ylabel("Error")
plt.show()
```


Batch
Stochastic
Mini-Batch
Momentum
RMS Prop
Adam
Error
0
10
20
30
40
50
60
0
1
2
3
4
5
6
Iterations

In [ ]: