```
import numpy as np
import matplotlib.pyplot as plt


def error_return(x,y):
    m_best , b_best = np.polyfit(x,y,1)
    print(m_best,b_best)
    y_hat = m_best * x +b_best
    error = y -y_hat
    absolute_error = np.abs(error)
    squared_error = np.square(error)
    # absolute_error
    mean_absolute_error = np.mean(absolute_error)
    mean_squared_error = np.mean(squared_error)/2
    return mean_absolute_error,mean_squared_error
```

```
# Example training data
x = np.array([1, 2, 3, 4, 5])
y = np.array([3000, 3500, 5000, 5500, 6500])
print(error_return(x,y))
# x = np.append(x, 6)
# y = np.append(y, 60000)
# print(error_return(x,y))
```

```
# Define cost function J(m, b)
def compute_cost(m, b, x, y):
    predictions = m * x + b
    errors = predictions - y
    J = np.mean(errors ** 2) / 2
    return J

# Create values for m and b
m_values = np.arange(800, 1000, 10)
b_values = np.arange(1900, 2500, 10)

M, B = np.meshgrid(m_values, b_values)
J = np.zeros(M.shape)
```

```
# Compute J for every m, b pair
```

```
for i in range(M.shape[0]):
    for j in range(M.shape[1]):
        J[i, j] = compute_cost(M[i, j], B[i, j], x, y)
```

```
# 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")

ax.plot_surface(M, B, J)

ax.set_xlabel("m")
ax.set_ylabel("b")
ax.set_zlabel("J(m, b)")
ax.set_title("Cost Function J(m, b)")

plt.show()
```

```
import numpy as np
import plotly.graph_objects as go


fig = go.Figure(
    data=[
        go.Surface(
            x=M,
            y=B,
            z=J
        )
    ]
)

fig.update_layout(
    title="Interactive Cost Function J(m, b)",
    scene=dict(
        xaxis_title="m",
        yaxis_title="b",
        zaxis_title="J(m, b)"
    )
)

fig.show()
```

Without convergence

```
m = 0
b = 0
learning_rate = 0.1
for i in range(1000):
    y_hat = m * x + b

    loss = np.mean((y_hat - y) ** 2)

    dm = np.mean((y_hat - y) * x)
    db = np.mean(y_hat - y)
    m = m - learning_rate * dm
    b = b - learning_rate * db
print("m =", m)
print("b =", b)
```

```
import numpy as np
import matplotlib.pyplot as plt

m = 0
b = 0
learning_rate = 0.1

tol = 1e-6
prev_loss = float("inf")
converged_epoch = None

m_history = []
b_history = []
loss_history = []

for i in range(1000):
    y_hat = m * x + b

    loss = np.mean((y_hat - y) ** 2)

    dm = np.mean((y_hat - y) * x)
    db = np.mean(y_hat - y)

    # store current values before update
    m_history.append(m)
    b_history.append(b)
    loss_history.append(loss)
```

```
    m = m - learning_rate * dm
    b = b - learning_rate * db

    if abs(prev_loss - loss) < tol:
        converged_epoch = i
        print(f"Converged at epoch {i}")
        break

    prev_loss = loss

print("m =", m)
print("b =", b)
```

Try with learning_rate `0.2 ,0.1 , 0.01,.15`

```
m = 0
b = 0

x = np.array([1, 2, 3, 4, 5])
y = np.array([3000, 3500, 5000, 5500, 6500])

learning_rate = 0.01
n = len(x)

for epoch in range(1000):
    indices = np.random.permutation(n)

    for j in indices:
        x_j = x[j]
        y_j = y[j]

        y_hat = m * x_j + b
        error = y_hat - y_j

        dm = error * x_j
        db = error

        m -= learning_rate * dm
        b -= learning_rate * db

print("m =", m)
print("b =", b)
```

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

```
import numpy as np
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

x = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
y = np.array([3000, 3500, 5000, 5500, 6500])

model = make_pipeline(
    StandardScaler(),
    SGDRegressor(max_iter=1000,
                 learning_rate="constant",
                 eta0=0.01,
                 random_state=42))

model.fit(x, y)

print(model.predict([[6]]))
```

[8] ✓ 8.9s

```
[7397.35657664]
```


```
import numpy as np
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

x = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
y = np.array([3000, 3500, 5000, 5500, 6500])

model = make_pipeline(
    StandardScaler(),
    SGDRegressor(max_iter=1000,
                 learning_rate="constant",
                 eta0=0.01,
                 random_state=42))

model.fit(x, y)

print(model.predict([[6]]))
```

[8] ✓ 8.9s

```
[7397.35657664]
```