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

# Suppress warnings
warnings.filterwarnings('ignore')

print('Libraries imported')
lr = 0.3
batch_size = 300
iterations = 40

# Assume dataset loading functions are directly implemented here
def get_data():
# Implement your dataset loading logic here
# For demonstration, randomly generate data
num_train = 60000
num_test = 10000
X_train = np.random.randn(num_train, 784)
Y_train = np.random.randint(0, 2, size=(num_train, 1))
X_test = np.random.randn(num_test, 784)
Y_test = np.random.randint(0, 2, size=(num_test, 1))
return (X_train, Y_train), (X_test, Y_test)

def get_random_batch(X, Y, batch_size):
idx = np.random.randint(0, X.shape[0], size=batch_size)
return X[idx], Y[idx]

# Define Logistic Model class
class LogisticModel:
def __init__(self, num_features):
self.W = np.reshape(np.random.randn(num_features), (num_features, 1))
self.b = np.zeros((1, 1))
self.num_features = num_features
self.losses = []
self.accuracies = []
def summary(self):
print('================================')
print('Number of features:', self.num_features)
print('Shape of weights:', self.W.shape)
print('Shape of biases:', self.b.shape)
print('================================')
def _forward_pass(self, X, Y=None):
batch_size = X.shape[0]
Z = np.dot(X, self.W) + self.b
A = 1. / (1. + np.exp(-Z))
loss = float(1e5)
if Y is not None:
```

```
loss = -1 * np.sum(np.dot(np.transpose(Y), np.log(A)) +
np.dot(np.transpose(1-Y), np.log(1-A)))
loss /= batch_size
return A, loss
def _backward_pass(self, A, X, Y):
batch_size = X.shape[0]
dZ = A - Y
dW = np.dot(np.transpose(X), dZ) / batch_size
db = np.sum(dZ) / batch_size
return dW, db
def _update_params(self, dW, db, lr):
self.W -= lr * dW
self.b -= lr * db
def predict(self, X, Y=None):
A, loss = self._forward_pass(X, Y)
Y_hat = A > 0.5
return np.squeeze(Y_hat), loss
def evaluate(self, X, Y):
Y_hat, loss = self.predict(X, Y)
accuracy = np.sum(Y_hat == np.squeeze(Y)) / X.shape[0]
return accuracy, loss
def train(self, batch_size, get_batch, lr, iterations, X_train, Y_train,
X_test, Y_test):
print('Training..')
self.accuracies = []
self.losses = []
for i in range(iterations):
X, Y = get_batch(X_train, Y_train, batch_size)
A, _ = self._forward_pass(X, Y)
dW, db = self._backward_pass(A, X, Y)
self._update_params(dW, db, lr)
X_val, Y_val = get_batch(X_test, Y_test, batch_size)
val_acc, val_loss = self.evaluate(X_val, Y_val)
self.accuracies.append(val_acc)
self.losses.append(val_loss)
print('Iter: {}, Val Acc: {:.3f}, Val Loss: {:.3f}'.format(i, val_acc,
val_loss))
print('Training finished.')

# Define plotting functions
def plot_metrics(model):
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(model.losses, label='Loss')
plt.title('Training Loss')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(model.accuracies, label='Accuracy')
```

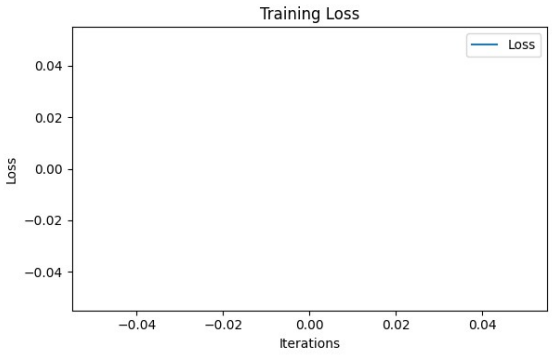
```
plt.title('Validation Accuracy')
plt.xlabel('Iterations')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()

def show_ten_examples(X, Y, preds):
plt.figure(figsize=(10, 4))
for i in range(10):
plt.subplot(2, 5, i + 1)
plt.imshow(X[i].reshape(28, 28), cmap='gray')
plt.title(f'True: {Y[i]}, Pred: {preds[i]}')
plt.axis('off')
plt.tight_layout()
plt.show()

# Main script
if __name__ == "__main__":
# Load dataset
(X_train, Y_train), (X_test, Y_test) = get_data()
print('Shape of X_train:', X_train.shape)
print('Shape of Y_train:', Y_train.shape)
print('Shape of X_test:', X_test.shape)
print('Shape of Y_test:', Y_test.shape)
# Initialize and train the model
model = LogisticModel(num_features=784)
model.summary()

# Evaluate untrained model
X, Y = get_random_batch(X_test, Y_test, batch_size)
acc, loss = model.evaluate(X, Y)
print('Untrained model accuracy: {:.3f}, loss: {:.3f}'.format(acc, loss))

# Train the model
model.train(batch_size, get_random_batch, lr, iterations, X_train, Y_train,
X_test, Y_test)

# Evaluate trained model
X, Y = get_random_batch(X_test, Y_test, batch_size)
acc, loss = model.evaluate(X, Y)
print('After training performance: Accuracy: {:.3f}, Loss:
{:.3f}'.format(acc, loss))

# Plot training metrics
plot_metrics(model)

# Show predictions after training
X, Y = get_random_batch(X_test, Y_test, batch_size)
preds, _ = model.predict(X)
```

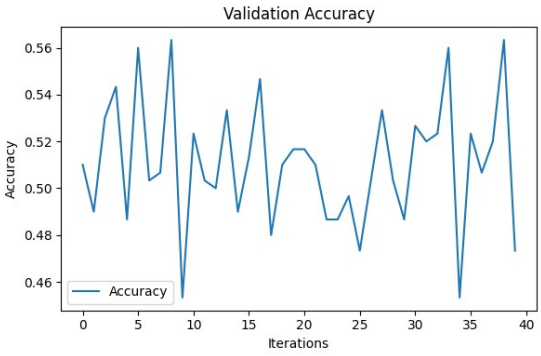
```
show_ten_examples(X, Y, preds)
```

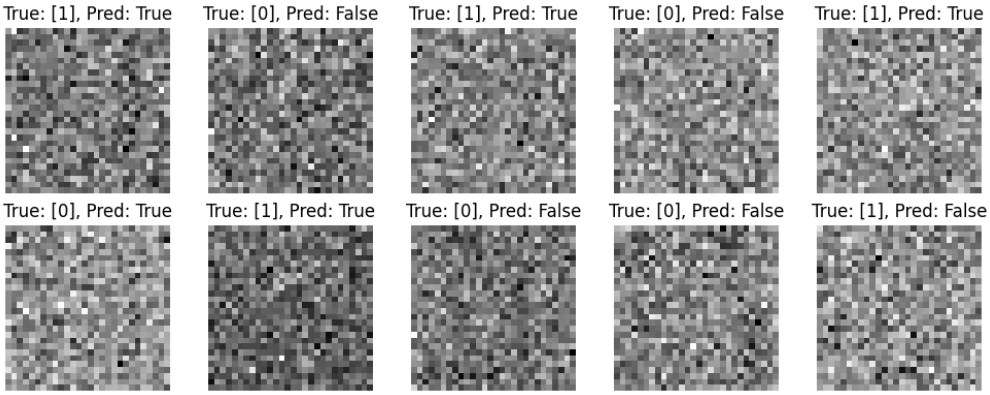
Correction done -
removed helpers with matplotlin library.

Output -

```
Libraries imported
Shape of X_train: (60000, 784)
Shape of Y_train: (60000, 1)
Shape of X_test: (10000, 784)
Shape of Y_test: (10000, 1)
================================
Number of features: 784
Shape of weights: (784, 1)
Shape of biases: (1, 1)
================================
Untrained model accuracy: 0.557, loss: nan
Training..
Iter: 0, Val Acc: 0.510, Val Loss: nan
Iter: 1, Val Acc: 0.490, Val Loss: nan
Iter: 2, Val Acc: 0.530, Val Loss: nan
Iter: 3, Val Acc: 0.543, Val Loss: nan
Iter: 4, Val Acc: 0.487, Val Loss: nan
Iter: 5, Val Acc: 0.560, Val Loss: nan
Iter: 6, Val Acc: 0.503, Val Loss: nan
Iter: 7, Val Acc: 0.507, Val Loss: nan
Iter: 8, Val Acc: 0.563, Val Loss: nan
Iter: 9, Val Acc: 0.453, Val Loss: nan
Iter: 10, Val Acc: 0.523, Val Loss: nan
Iter: 11, Val Acc: 0.503, Val Loss: nan
Iter: 12, Val Acc: 0.500, Val Loss: nan
Iter: 13, Val Acc: 0.533, Val Loss: nan
Iter: 14, Val Acc: 0.490, Val Loss: nan
Iter: 15, Val Acc: 0.513, Val Loss: nan
Iter: 16, Val Acc: 0.547, Val Loss: nan
Iter: 17, Val Acc: 0.480, Val Loss: nan
Iter: 18, Val Acc: 0.510, Val Loss: nan
Iter: 19, Val Acc: 0.517, Val Loss: nan
Iter: 20, Val Acc: 0.517, Val Loss: nan
Iter: 21, Val Acc: 0.510, Val Loss: nan
Iter: 22, Val Acc: 0.487, Val Loss: nan
Iter: 23, Val Acc: 0.487, Val Loss: nan
Iter: 24, Val Acc: 0.497, Val Loss: nan
Iter: 25, Val Acc: 0.473, Val Loss: nan
Iter: 26, Val Acc: 0.503, Val Loss: nan
Iter: 27, Val Acc: 0.533, Val Loss: nan
Iter: 28, Val Acc: 0.503, Val Loss: nan
Iter: 29, Val Acc: 0.487, Val Loss: nan
Iter: 30, Val Acc: 0.527, Val Loss: nan
Iter: 31, Val Acc: 0.520, Val Loss: nan
Iter: 32, Val Acc: 0.523, Val Loss: nan
Iter: 33, Val Acc: 0.560, Val Loss: nan
Iter: 34, Val Acc: 0.453, Val Loss: nan
Iter: 35, Val Acc: 0.523, Val Loss: nan
Iter: 36, Val Acc: 0.507, Val Loss: nan
Iter: 37, Val Acc: 0.520, Val Loss: nan
Iter: 38, Val Acc: 0.563, Val Loss: nan
Iter: 39, Val Acc: 0.473, Val Loss: nan
Training finished.
After training performance: Accuracy: 0.450, Loss: nan
```


Training Loss
Loss
0.04
0.02
0.00
−0.02
−0.04
Loss
−0.04
−0.02
0.00
0.02
0.04
Iterations


Validation Accuracy
0.56
0.54
0.52
0.50
0.48
0.46
Accuracy
Accuracy
0
5
10
15
20
25
30
35
40
Iterations


True: [1], Pred: True
True: [0], Pred: False
True: [1], Pred: True
True: [0], Pred: False
True: [1], Pred: True
True: [0], Pred: True
True: [1], Pred: True
True: [0], Pred: False
True: [0], Pred: False
True: [1], Pred: False