In [ ]:
```
Name - Shivraj Pandurang Mane.
Class - BE Artificial Intelligence and Data Science.
Roll No. - 37
Practical No. 05 - Optimization of genetic algorithm parameter in hybrid genetic algorithm-neural network
modelling: Application to spray drying of coconut milk.
```

In [6]:
```
# Import Required Libraries
```

In [18]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from deap import base, creator, tools, algorithms
import random
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
```

In [19]:
```
# Load and Preprocess Data
```

In [20]:
```
# Simulate dataset (replace with real-world data if available)
np.random.seed(42)
n_samples = 100
X = np.random.uniform(low=30, high=100, size=(n_samples, 2))  # e.g., temperature, feed rate
y = 0.8 * X[:, 0] - 0.5 * X[:, 1] + np.random.normal(0, 5, n_samples)  # Spray drying efficiency

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalize inputs
X_mean, X_std = X_train.mean(axis=0), X_train.std(axis=0)
X_train = (X_train - X_mean) / X_std
X_test = (X_test - X_mean) / X_std
```

In [21]:
```
# Define Neural Network
```

In [22]:
```
def build_nn(num_neurons, learning_rate):
    model = Sequential([
        Dense(num_neurons, activation='relu', input_shape=(X_train.shape[1],)),
        Dense(1)
    ])
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
                  loss='mse')
    return model
```

In [23]:
```
# Genetic Algorithm for Optimization
```

In [24]:
```
# Define GA to optimize number of neurons and learning rate
def evaluate_fitness(individual):
    num_neurons = int(individual[0])  # Number of neurons
    learning_rate = individual[1]     # Learning rate

    model = build_nn(num_neurons, learning_rate)
    model.fit(X_train, y_train, epochs=20, verbose=0, batch_size=10)

    # Predict and calculate mean squared error on test set
    y_pred = model.predict(X_test, verbose=0)
    mse = mean_squared_error(y_test, y_pred)
    return mse,

# GA Configuration
toolbox = base.Toolbox()
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))  # Minimize MSE
```

```
creator.create("Individual", list, fitness=creator.FitnessMin)
toolbox.register("attr_num_neurons", random.randint, 5, 50)  # Range for neurons
toolbox.register("attr_learning_rate", random.uniform, 0.001, 0.01)  # Range for learning rate
toolbox.register("individual", tools.initCycle, creator.Individual,
                 (toolbox.attr_num_neurons, toolbox.attr_learning_rate), n=1)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("mate", tools.cxBlend, alpha=0.5)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=0.1, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", evaluate_fitness)

# Run GA
random.seed(42)
population = toolbox.population(n=10)
ngen = 20
cxpb, mutpb = 0.5, 0.2

result, log = algorithms.eaSimple(population, toolbox, cxpb=cxpb, mutpb=mutpb,
                                  ngen=ngen, verbose=True)
```

```
C:\Users\saira\AppData\Local\Programs\Python\Python310\lib\site-packages\deap\creator.py:185: RuntimeWarning:
A class named 'FitnessMin' has already been created and it will be overwritten. Consider deleting previous cr
eation of that class or rename it.
  warnings.warn("A class named '{0}' has already been created and it "
C:\Users\saira\AppData\Local\Programs\Python\Python310\lib\site-packages\deap\creator.py:185: RuntimeWarning:
A class named 'Individual' has already been created and it will be overwritten. Consider deleting previous cr
eation of that class or rename it.
  warnings.warn("A class named '{0}' has already been created and it "
gen     nevals
0       10
1       7
2       4
3       9
4       8
5       7
6       6
7       7
8       10
9       7
10      7
11      8
12      4
13      6
14      4
15      6
16      6
17      4
18      9
19      6
20      3
```

In [25]:
```
# Best Solution
```

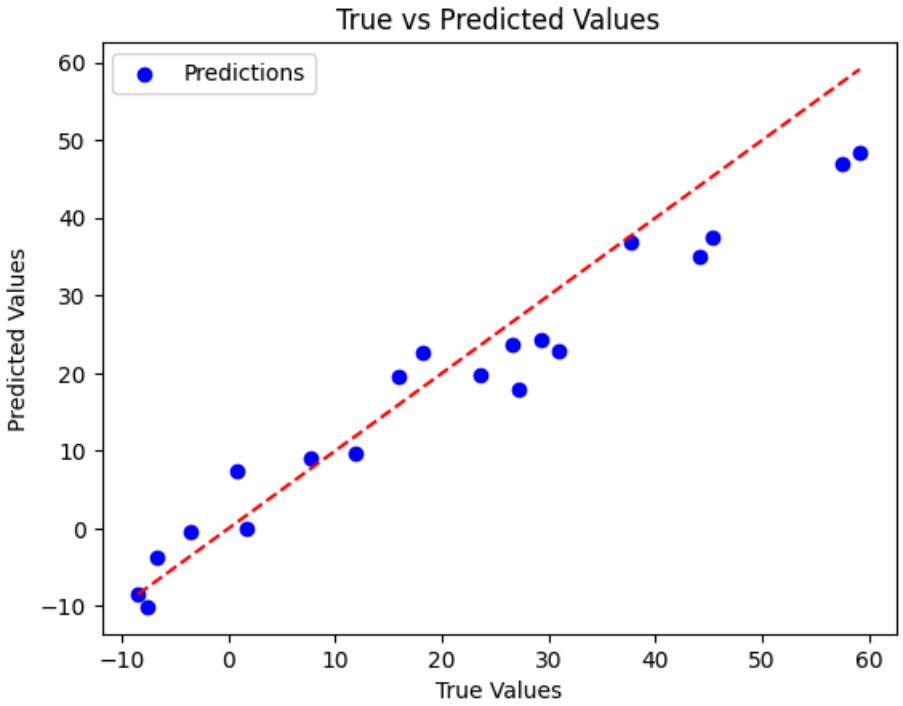
In [26]:
```
best_individual = tools.selBest(population, k=1)[0]
print(f"Best Individual: Neurons={best_individual[0]}, Learning Rate={best_individual[1]}")

# Train and evaluate the final model
final_model = build_nn(int(best_individual[0]), best_individual[1])
final_model.fit(X_train, y_train, epochs=50, verbose=1, batch_size=10)

# Test the model
y_pred = final_model.predict(X_test)
final_mse = mean_squared_error(y_test, y_pred)
print(f"Final MSE on Test Data: {final_mse}")
```

```
Best Individual: Neurons=47.006821615194, Learning Rate=0.32985446714314715
Epoch 1/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 489.9717
Epoch 2/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 50.7741
Epoch 3/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 33.9357
Epoch 4/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 31.5023
Epoch 5/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 41.3352
Epoch 6/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 40.3212
Epoch 7/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 48.9730
Epoch 8/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 35.9454
Epoch 9/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 27.6753
Epoch 10/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 30.9002
Epoch 11/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 32.0088
Epoch 12/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 31.3612
Epoch 13/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.2624
Epoch 14/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 32.8632
Epoch 15/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 23.9839
Epoch 16/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 22.8324
Epoch 17/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 22.8549
Epoch 18/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 20.8978
Epoch 19/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 35.2586
Epoch 20/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 36.9907
Epoch 21/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 34.2124
Epoch 22/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 36.0794
Epoch 23/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 25.2957
Epoch 24/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 23.1455
Epoch 25/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 36.7362
Epoch 26/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 66.9416
Epoch 27/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 54.6479
Epoch 28/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 31.6343
Epoch 29/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 41.8209
Epoch 30/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 42.4975
Epoch 31/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 43.6928
Epoch 32/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 47.8714
Epoch 33/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 36.5030
Epoch 34/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 32.2641
Epoch 35/50
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 20.8831
Epoch 36/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 36.4942
Epoch 37/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 32.7672
Epoch 38/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 30.7013
Epoch 39/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 41.3716
Epoch 40/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 69.1590
Epoch 41/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 31.7092
Epoch 42/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 26.9307
Epoch 43/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 29.0356
Epoch 44/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 34.7665
Epoch 45/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 33.8882
Epoch 46/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 25.0320
Epoch 47/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 25.4510
Epoch 48/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 33.5734
Epoch 49/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 27.6305
Epoch 50/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 36.5471
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step
Final MSE on Test Data: 34.340513656320596
```

In [27]:
```
# Visualize Results
```

In [28]:
```
# Plot true vs predicted values
plt.scatter(y_test, y_pred, c='blue', label='Predictions')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.legend()
plt.title("True vs Predicted Values")
plt.show()
```


True vs Predicted Values
Predictions
Predicted Values
True Values
60
50
40
30
20
10
0
−10
−10
0
10
20
30
40
50
60

In [ ]: