```python
import numpy as np

# Objective function (to minimize)
def objective_function(x):
    return np.sum(x**2)

# Grey Wolf Optimizer (GWO) implementation
class GreyWolfOptimizer:
    def __init__(self, objective_function, n_wolves, max_iter, dim, lower_bound, upper_bound):
        self.obj_func = objective_function
        self.n_wolves = n_wolves
        self.max_iter = max_iter
        self.dim = dim
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound
        self.alpha_position = np.zeros(dim)
        self.beta_position = np.zeros(dim)
        self.delta_position = np.zeros(dim)
        self.alpha_score = float("inf")
        self.beta_score = float("inf")
        self.delta_score = float("inf")
        self.wolves_position = np.random.uniform(lower_bound, upper_bound, (n_wolves, dim))

    def update_position(self, A, C, wolf_position, best_position):
        # Update the position of each wolf based on alpha, beta, or delta wolves
        D = np.abs(C * best_position - wolf_position)
        return best_position - A * D

    def optimize(self):
        for t in range(self.max_iter):
            for i in range(self.n_wolves):
                fitness = self.obj_func(self.wolves_position[i])

                # Update alpha, beta, and delta wolves
                if fitness < self.alpha_score:
                    self.alpha_score = fitness
                    self.alpha_position = self.wolves_position[i].copy()

                elif fitness < self.beta_score and fitness > self.alpha_score:
                    self.beta_score = fitness
                    self.beta_position = self.wolves_position[i].copy()

                elif fitness < self.delta_score and fitness > self.beta_score:
                    self.delta_score = fitness
                    self.delta_position = self.wolves_position[i].copy()

            # Calculate A and C coefficients
            a = 2 - t * (2 / self.max_iter)  # Linearly decreases from 2 to 0
            r1, r2, r3 = np.random.rand(3, self.n_wolves, self.dim)  # Random values for each wolf and dimension
            A1 = 2 * a * r1 - a
            A2 = 2 * a * r2 - a
            A3 = 2 * a * r3 - a
            C1 = 2 * r1
            C2 = 2 * r2
            C3 = 2 * r3

            # Update positions of all wolves
            for i in range(self.n_wolves):
                # Update the position of each wolf using alpha, beta, delta
                self.wolves_position[i] = self.update_position(A1[i], C1[i], self.wolves_position[i], self.alpha_position)
                self.wolves_position[i] = self.update_position(A2[i], C2[i], self.wolves_position[i], self.beta_position)
                self.wolves_position[i] = self.update_position(A3[i], C3[i], self.wolves_position[i], self.delta_position)

                # Ensure the wolves stay within the bounds
                self.wolves_position[i] = np.clip(self.wolves_position[i], self.lower_bound, self.upper_bound)

        return self.alpha_position, self.alpha_score

# Parameters for the optimizer
n_wolves = 50
max_iter = 100
dim = 2
lower_bound = -10
upper_bound = 10

# Instantiate the GWO
gwo = GreyWolfOptimizer(objective_function, n_wolves, max_iter, dim, lower_bound, upper_bound)

# Run the optimization
best_position, best_score = gwo.optimize()
```

```
# Output the results
print("Best solution found: ", best_position)
print("Objective function value: ", best_score)
```

```
Best solution found:  [-5.91008635e-18 -2.84204651e-18]
Objective function value:  4.300634905642505e-35
```

```
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.6.0)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
```

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Define the fitness function
def fitness_function(features, X, y):
    # Train a Random Forest model
    model = RandomForestClassifier()
    model.fit(X[:, features], y)
    predictions = model.predict(X[:, features])
    return 1 - accuracy_score(y, predictions)  # Minimize the error

# Grey Wolf Optimizer
class GreyWolfOptimizer:
    def __init__(self, num_wolves, num_features, max_iter):
        self.num_wolves = num_wolves
        self.num_features = num_features
        self.max_iter = max_iter
        self.wolves = np.random.randint(2, size=(num_wolves, num_features))
        self.alpha = None
        self.beta = None
        self.gamma = None
        self.alpha_fitness = float("inf")
        self.beta_fitness = float("inf")
        self.gamma_fitness = float("inf")

    def optimize(self, X, y):
        for iteration in range(self.max_iter):
            for i in range(self.num_wolves):
                fitness = fitness_function(self.wolves[i], X, y)

                # Update alpha, beta, and gamma wolves
                if fitness < self.alpha_fitness:
                    self.gamma_fitness = self.beta_fitness
                    self.gamma = self.beta.copy() if self.beta is not None else None
                    self.beta_fitness = self.alpha_fitness
                    self.beta = self.alpha.copy() if self.alpha is not None else None
                    self.alpha_fitness = fitness
                    self.alpha = self.wolves[i].copy()
                elif fitness < self.beta_fitness:
                    self.gamma_fitness = self.beta_fitness
                    self.gamma = self.beta.copy() if self.beta is not None else None
                    self.beta_fitness = fitness
                    self.beta = self.wolves[i].copy()
                elif fitness < self.gamma_fitness:
                    self.gamma_fitness = fitness
                    self.gamma = self.wolves[i].copy()

            # Update positions of wolves
            a = 2 - iteration * (2 / self.max_iter)
            for j in range(self.num_wolves):
                r1 = np.random.rand(self.num_features)
                r2 = np.random.rand(self.num_features)
                A1 = 2 * a * r1 - a
                C1 = 2 * r2
                D_alpha = np.abs(C1 * self.alpha - self.wolves[j])
                X1 = self.alpha - A1 * D_alpha

                r1 = np.random.rand(self.num_features)
                r2 = np.random.rand(self.num_features)
                A2 = 2 * a * r1 - a
                C2 = 2 * r2
                D_beta = np.abs(C2 * self.beta - self.wolves[j])
                X2 = self.beta - A2 * D_beta

                r1 = np.random.rand(self.num_features)
                r2 = np.random.rand(self.num_features)
                A3 = 2 * a * r1 - a
                C3 = 2 * r2
                D_gamma = np.abs(C3 * self.gamma - self.wolves[j])
                X3 = self.gamma - A3 * D_gamma

                self.wolves[j] = (X1 + X2 + X3) / 3
                self.wolves[j] = np.clip(self.wolves[j], 0, 1)  # Ensure binary values

        return self.alpha

# Example usage
if __name__ == "__main__":
    # Load the Iris dataset
    iris = load_iris()
    X = iris.data  # Features
    y = iris.target  # Target

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

    # Initialize and run GWO
    gwo = GreyWolfOptimizer(num_wolves=10, num_features=X.shape[1], max_iter=50)
    best_features = gwo.optimize(X_train, y_train)

    # Display selected features
    selected_features = np.where(best_features == 1)[0]
    print("Selected features:", selected_features)

    # Evaluate the model with selected features
    model = RandomForestClassifier()
    model
```

```
Selected features: [0 1]
```