```
def random_search(initial_board, goal_state): # Solve the 8-
puzzle using random search
    current_board = initial_board
    num_moves = 0

    while not is_goal_state(current_board, goal_state):
        directions = ['up', 'down', 'left', 'right']
        random_direction = random.choice(directions)

        if move_blank_tile(current_board, random_direction):
          num_moves += 1

        if is_goal_state(current_board, goal_state):
          return current_board, num_moves

initial_board = [[2, 8, 3], [1, 6, 4], [7, 0, 5]]
goal_state = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]

solved_board, num_moves = random_search(initial_board,
goal_state)
print(solved_board,"\n",num_moves)
```


Q2. WAPP to find the minimum of quadratic equation $2x^2-5x+3$ in the range [-10,10] using random search.


```
import random

def quadratic_function(x): # Finding the value of the function
    return 2 * x**2 - 5 * x + 3

def find_minimum_random_search(max_iterations=1000): #Finding
the minimum
    best_x = None
    best_value = float('inf')  # Initialize with positive
infinity

    for _ in range(max_iterations):
        x = random.uniform(-10, 10)
        value = quadratic_function(x)
        if value < best_value:
            best_x = x
            best_value = value

    return best_x, best_value

minimum_x, minimum_value = find_minimum_random_search()
print("Minimum found at x =", minimum_x)
```

```
def get_lat_long(city):
  loc = geolocator.geocode(city+", India")
  return [loc.latitude, loc.longitude]

def calculateDistance(lat1,long1,lat2,long2):
  return geopy.distance.geodesic((lat1,long1),(lat2,long2)).km

graph = {"nodes": set(), "edges": {}}

def add_locations(location1,location2):
    if location1 not in graph["nodes"]:
        graph["nodes"].add(location1)
        graph["edges"][location1] = []

    if location2 not in graph["nodes"]:
        graph["nodes"].add(location2)
        graph["edges"][location2] = []

    print(graph)

    position1 = get_lat_long(location1)
    position2 = get_lat_long(location2)

    print(position1, position2)

    distance =
round(calculateDistance(position1[0],position1[1],position2[0]
,position2[1]),2)

    print(distance)

    graph["edges"][location1].append((location2,distance))
    graph["edges"][location2].append((location1,distance))

def uniform_cost_search(start, goal):
  priority_queue = [(0,start,[])]

  while priority_queue:
    (cost, current_location,path) = priority_queue.pop(0)

    if current_location in path:
      continue

    path = path + [current_location]

    if current_location ==goal:
      return path
```

```
    plt.figure(figsize=(2, 2))
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
    plt.show()
```



```
# Normalize the data
x_train, x_test = x_train / 255.0, x_test / 255.0

# Expand dimensions to fit into the CNN model
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)

x_train
```

```
# Define CNN model
def build_cnn(input_shape):
    model = models.Sequential([
        layers.Conv2D(32, kernel_size=(3, 3),
activation='relu', input_shape=input_shape),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3),
activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(10, activation='softmax')  # 10 clusters
for MNIST
    ])
    model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```



```
cnn_model = build_cnn(input_shape=x_train[0].shape)
```

```
from keras.utils import plot_model

plot_model(cnn_model, show_shapes=True, show_layer_names=True)
```

```
cnn_model.fit(x_train, y_train, epochs=5, batch_size=128,
validation_data=(x_test, y_test))
```

```
# Extract features using CNN model
```