Heba Mohamed hebamohamed14101998@gmail.com

SMART REHABILITATION

A GA-Based Solution for Optimal Exercise Plans.

CONTENTS

1 SOLUTION REPRESENTATION 6
1.1 Work preparation 6
2 FITNESS FUNCTION 8
3 GENETIC OPERATORS 9
3.1 Single Point one Crossover 9
3.2 Uniform Mutation 10
3.3 Selection by Roulette Wheel Selection 11
4 REPLACEMENT 12
5 TERMINATION CONDITION 13
6 RESULTS 13
6.1 User Information 13
6.2 Result 1 14
6.3 Result 2 15
6.4 Result 3 16
6.5 Result 4 17
6.6 Result 5 18
6.7 Result 6 19
6.8 Result 7 20
6.9 Result 8 21
6.10 Result 9 22
7 RESULTS ANALYSIS 23
8 RUNNING THE BEST SOLUTION 24

Heba Mohamed hebamohamed14101998@gmail.com

List of Figures

Figure 1: The smart rehabilitation workflow. 6
Figure 2: Create a general dataset. 7
Figure 7: General dataset as a DataFrame. 8
Figure 8: The probability density function of the continuous uniform distribution. 9
Figure 9: Weights initialization implementation. 9
Figure 10: calculate fitness. 9
Figure 11:Single point crossover. 10
Figure 12: Single point crossover implementation. 10
Figure 13:Uniform mutation implementation. 10
Figure 14: Roulette Wheel selection equation. 11
Figure 15: calculate the probability for each chromosome. 11
Figure 16: Roulette Wheel implementation. 12
Figure 17: Replacement implementation. 13
Figure 18: Termination conditions. 13
Figure 19: Getting information from a patient. 14
Figure 20: Result 1 graph. 15
Figure 21: Result 2 graph. 16
Figure 22: Result 3 graph. 17
Figure 23: Result 4 graph. 18
Figure 24: Result 5 graph. 19
Figure 25: Result 6 graph. 20
Figure 26: Result 7 graph. 21
Figure 27: Result 8 graph. 22
Figure 28: Result 9 graph. 23
Figure 29: User inputs. 24
Figure 30: Rehabilitation plan. 25
Figure 31:Rrehabilitation plan as a data frame. 25

Heba Mohamed hebamohamed14101998@gmail.com

1 SOLUTION REPRESENTATION

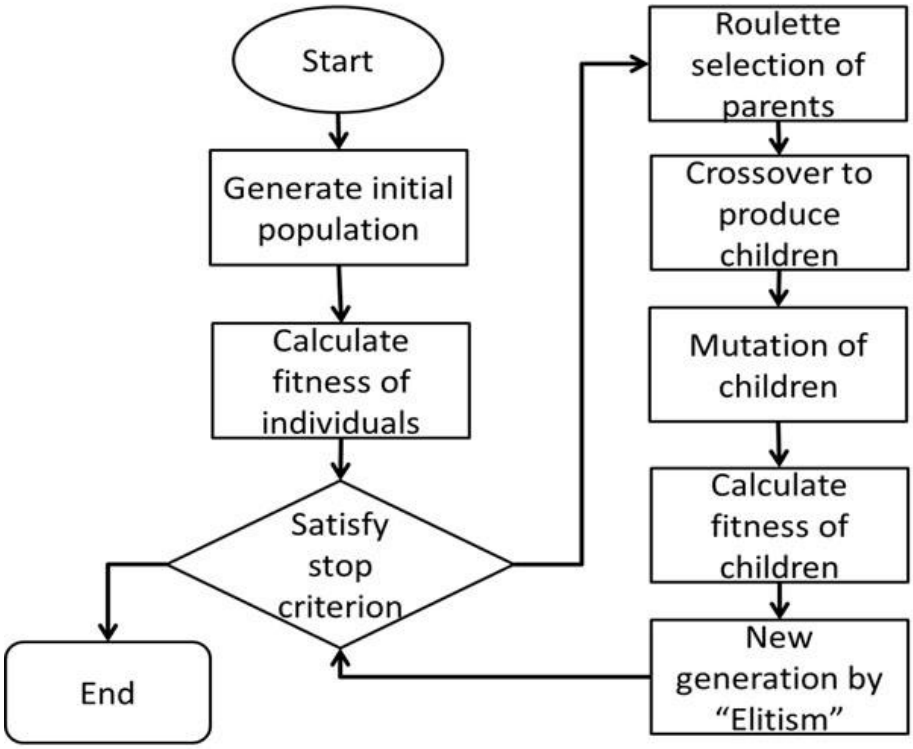

Figure 1: The smart rehabilitation workflow.

1.1 Work preparation

Before representing the solution, we need to create our dataset. Firstly, we create a dictionary for the body parts (elbow, upper arm, knee/lower leg, and wrist) then we create a DataFrame from these dictionaries which contains four columns (Body, exercises, Condition, and Age Category). Finally calculating the four body parts' DataFrames as follows:

```python
def create_general_dataset():
    '''
    This function used for creating the general dataset for each part in the body

    Returns
    -------
    df : General dataset which contains all the details about the exercises.

    '''

    # intialize dictionary of lists.
    Elbow_Data        = { 'Body':['Elbow', 'Elbow', 'Elbow', 'Elbow' , 'Elbow', 'Elbow'  , 'Elbow', 'Elbow' , 'Elbow', 'Elbow' , 'Elbow' ,'Elbow' ],

                          'Exercises':['Elbow extensor using free weights ', 'Crawling', 'Elbow flexor using free weights ', 'Bear walking' ,
                                       'Elbow extensor using theraband  ', 'Lifting in parallel bars '  , 'Elbow Flexor using theraband ',
                                       'Lifting in sitting using scale ' , 'Rotating forearm ', 'Forearm supination' ,
                                       'Learning forwards in a large ball ','Wheelbarrow walking on hands' ],
                          'Condition':['Stroke' , 'Stroke'  , 'Stroke'   , 'Stroke'  , 'Spinal cord injuries' , 'Spinal cord injuries',
                                       'Spinal cord injuries' , 'Spinal cord injuries' , 'Brain injury ' , 'Brain injury' , 'Brain injury' , 'Brain injury '  ] ,

                          'AgeCategory':['Adult', 'Child','Adult','Child','Adult','Child','Adult','Child','Adult','Child','Adult','Child']
                         }


    UpperArm_Data     = {'Body':['Upper Arm', 'Upper Arm', 'Upper Arm', 'Upper Arm' , 'Upper Arm', 'Upper Arm'  , 'Upper Arm', 'Upper Arm' ,
                                 'Upper Arm', 'Upper Arm' , 'Upper Arm' ,'Upper Arm' ],

                         'Exercises':['Shoulder external rotator using free weights ', 'Weight-bearing through one shoulder',
                                      'Shoulder extensor ', 'Crawling ' , 'Shoulder abductor using theraband ', 'Shoulder abductor Stritch in setting' ,
                                      'Shoulder adductor using theraband '  , 'Boxing in setting ', 'Push-ups in prone' , 'Reaching in four points kneeling',
                                      'Lowering and pushing-up in long setting ' , 'Crab-walking ' ],

                         'Condition':['Stroke' , 'Stroke'  , 'Stroke'   , 'Stroke'  , 'Spinal cord injuries' , 'Spinal cord injuries',  'Spinal cord injuries' ,
                                      'Spinal cord injuries' , 'Brain injury ' , 'Brain injury' , 'Brain injury' , 'Brain injury '  ] ,

                         'AgeCategory':['Adult', 'Child','Adult','Child','Adult','Child','Adult','Child','Adult','Child','Adult','Child']
                        }

    KneeLowerleg_Data = {'Body':['Knee/ Lower leg', 'Knee/ Lower leg', 'Knee/ Lower leg', 'Knee/ Lower leg' , 'Knee/ Lower leg', 'Knee/ Lower leg'  ,
                                 'Knee/ Lower leg', 'Knee/ Lower leg' , 'Knee/ Lower leg', 'Knee/ Lower leg' , 'Knee/ Lower leg' ,'Knee/ Lower leg' ],

                         'Exercises':['Knee flexor using a device ', 'Squatting against a wall', 'Knee extensor using a device ', 'Seated walking' ,
                                      'Bending the knee in standing', 'Walking on heels' , 'Standing and setting  '  , 'Walking on tiptoes  ',
                                      'Hamster stretch in setting ' , 'Stepping up onto a block ' ,'Leg stretching using sandbag ','Stepping sideways onto a block '],

                         'Condition':['Stroke' , 'Stroke'  , 'Stroke'   , 'Stroke'  , 'Spinal cord injuries' , 'Spinal cord injuries',
                                'Spinal cord injuries' , 'Spinal cord injuries' , 'Brain injury ' , 'Brain injury' , 'Brain injury' , 'Brain injury '  ] ,

                         'AgeCategory':['Adult', 'Child','Adult','Child','Adult','Child','Adult','Child','Adult','Child','Adult','Child']
                        }


    Wrist_Data        = {'Body':['Wrist', 'Wrist', 'Wrist', 'Wrist' , 'Wrist', 'Wrist'  , 'Wrist', 'Wrist' , 'Wrist', 'Wrist' , 'Wrist' ],
                         'Exercises':['Wrist extensor using free weights', 'Finger extensor strengthening using an elastic band',
                                      'Wrist flexor using free weights','Finger Flexor using grip device','Pincer grip ','Propping in-side setting',
                                      'Wrist extensor using theraband','Wrist extensor using electrical simulation','Reaching in standing ',
                                      'Popping bubble wrap','Walking hands on a ball'] ,

                         'Condition':['Stroke' , 'Stroke'  , 'Stroke'   , 'Stroke'  , 'Spinal cord injuries' , 'Spinal cord injuries',
                                      'Spinal cord injuries' , 'Brain injury ' , 'Brain injury' , 'Brain injury' , 'Brain injury '  ] ,

                         'AgeCategory':['Adult', 'Child','Adult','Child','Adult','Child','Adult','Adult','Child','Adult','Child']
                        }

    # Convert Dictionaries to DataFrame using Pandas
    Elbow_df        = pd.DataFrame(Elbow_Data)
    UpperArm_df     = pd.DataFrame(UpperArm_Data)
    KneeLowerleg_df = pd.DataFrame(KneeLowerleg_Data)
    Wrist_df        = pd.DataFrame(Wrist_Data)

    # Concate All DataFrames
    # Stack the DataFrames on top of each other
    df = pd.concat([Elbow_df, UpperArm_df , KneeLowerleg_df , Wrist_df ], axis=0)
    df = df.reset_index(drop=True)
    pd.set_option('display.width' , 10000)

    return df
```

Figure 2: Create a general dataset.

```
In [34]: general_df
Out[34]:
```

	Body	Exercises	Condition	AgeCategory
0	Elbow	Elbow extensor using free weights	Stroke	Adult
1	Elbow	Crawling	Stroke	Child
2	Elbow	Elbow flexor using free weights	Stroke	Adult
3	Elbow	Bear walking	Stroke	Child
4	Elbow	Elbow extensor using theraband	Spinal cord injuries	Adult
5	Elbow	Lifting in parallel bars	Spinal cord injuries	Child
6	Elbow	Elbow Flexor using theraband	Spinal cord injuries	Adult
7	Elbow	Lifting in sitting using scale	Spinal cord injuries	Child
8	Elbow	Rotating forearm	Brain injury	Adult
9	Elbow	Forearm supination	Brain injury	Child
10	Elbow	Learning forwards in a large ball	Brain injury	Adult
11	Elbow	Wheelbarrow walking on hands	Brain injury	Child
12	Upper Arm	Shoulder external rotator using free weights	Stroke	Adult
13	Upper Arm	Weight-bearing through one shoulder	Stroke	Child
14	Upper Arm	Shoulder extensor	Stroke	Adult
15	Upper Arm	Crawling	Stroke	Child
16	Upper Arm	Shoulder abductor using theraband	Spinal cord injuries	Adult
17	Upper Arm	Shoulder abductor Stritch in setting	Spinal cord injuries	Child
18	Upper Arm	Shoulder adductor using theraband	Spinal cord injuries	Adult
19	Upper Arm	Boxing in setting	Spinal cord injuries	Child
20	Upper Arm	Push-ups in prone	Brain injury	Adult
21	Upper Arm	Reaching in four points kneeling	Brain injury	Child
22	Upper Arm	Lowering and pushing-up in long setting	Brain injury	Adult
23	Upper Arm	Crab-walking	Brain injury	Child
24	Knee/ Lower leg	Knee flexor using a device	Stroke	Adult
25	Knee/ Lower leg	Squatting against a wall	Stroke	Child
26	Knee/ Lower leg	Knee extensor using a device	Stroke	Adult
27	Knee/ Lower leg	Seated walking	Stroke	Child
28	Knee/ Lower leg	Bending the knee in standing	Spinal cord injuries	Adult
29	Knee/ Lower leg	Walking on heels	Spinal cord injuries	Child
30	Knee/ Lower leg	Standing and setting	Spinal cord injuries	Adult
31	Knee/ Lower leg	Walking on tiptoes	Spinal cord injuries	Child
32	Knee/ Lower leg	Hamster stretch in setting	Brain injury	Adult

Figure 3: General dataset as a DataFrame.

2 FITNESS FUNCTION

The fitness function simply defined is a function that takes a candidate solution to the problem as input and produces as output how good the solution is with respect to the problem in consideration.

We are going to use the genetic algorithm and maximize y for the best values of the three weights $(w1, w2, and\ w3)$ after a number of generations.

$$y = w1x1 + w2x2 + w3x3$$

Where:

y : the value we want to maximize.

$x1$: age → 1 for adult, 2 for the child.

$x2$: total exercises → sum of different exercises entered by the user

$x3$: condition → 1 for stroke, 2 for spinal cord injuries, and 3 for brain injury.

At first, we initialize the three weights randomly using the uniform distribution with a=0 and b = 1.

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \le x \le b, \\ 0 & \text{for } x < a \text{ or } x > b \end{cases}$$

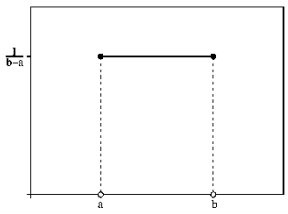

Figure 4: The probability density function of the continuous uniform distribution.

```
# weights initialization
new_population = numpy.random.uniform(low = 0, high = 1, size = (solution_per_pop, n_weights))
```

Figure 5: Weights initialization implementation.

```
def cal_pop_fitness(equation_inputs, population):
    '''
    This function calculating the fitness value of each solution in the current population
    by calculating the sum of products between each input and its corresponding weight.

    Parameters
    ----------
    equation_inputs : List of the three parameters.
    population      : The corresponding weights.

    Returns
    -------
    fitness         : The sum of products between each input and its corresponding weight.
    '''
    fitness = numpy.sum( population * equation_inputs, axis=1)

    return fitness
```

Figure 6: calculate fitness.

3 GENETIC OPERATORS

3.1 Single Point one Crossover

Single Point Crossover: A point on both parents' chromosomes is picked randomly and designated a 'crossover point'. Bits to the right of that point are swapped between the two parent

chromosomes. This results in two offspring, each carrying some genetic information from both parents.

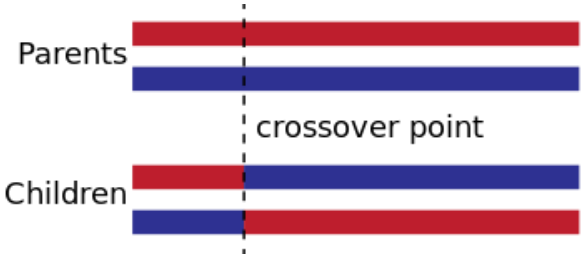

Figure 7:Single point crossover.

```
def single_point_crossover(parents, offspring_size, rate):

    offspring = numpy.empty(offspring_size)
    # The point at which crossover takes place between two parents. Usually it is at the center.
    crossover_point = numpy.uint8(offspring_size[1]/2)

    if random.random() < rate:
        for k in range(offspring_size[0]):
            # Index of the first parent to mate.
            parent1_idx = k%parents.shape[0]
            # Index of the second parent to mate.
            parent2_idx = (k+1)%parents.shape[0]
            # The new offspring will have its first half of its genes taken from the first parent.
            offspring[k, 0:crossover_point] = parents[parent1_idx, 0:crossover_point]
            # The new offspring will have its second half of its genes taken from the second parent.
            offspring[k, crossover_point:] = parents[parent2_idx, crossover_point:]

    return offspring
```

Figure 8: Single point crossover implementation.

3.2 Uniform Mutation

In uniform mutation we select a random gene from our chromosome, let's say x_i , and assign a uniform random value to it. Let x_i be within the range [a_i: bi] then we assign U (a_i, bi) to x_i U(-1.0, 1.0) denotes a uniform random number from within the range [-1.0: 1.0].

```
def uniform_mutation(offspring, rate):
    # Mutation changes a single gene in each offspring randomly.
    for i in range(offspring.shape[0]):
        # The random value to be added to the gene.
        val = numpy.random.uniform(-1.0, 1.0, 1)

        if random.random() < rate:
           offspring[i, 2] = offspring[i, 2] + val

    return offspring
```

Figure 9:Uniform mutation implementation.

3.3 Selection by Roulette Wheel Selection

Roulette selection is a stochastic selection method, where the probability for the selection of an individual is proportional to its fitness**.**

$$p_x = \frac{f_x}{\sum_{i=1}^{n} f_i}$$

Figure 10: Roulette Wheel selection equation.

1. In a population with n individuals, for each chromosome x with a corresponding fitness value f_i, we compute the corresponding probability $p_x = \{ p1 , p2 , p3 \}$
2. Generate a random number (r).
3. Choose the chromosome which has probability > r.

```
def get_probability_list(population_fitness):

    #fitness = population_fitness.values()
    total_fit = float(sum(fitness))
    relative_fitness = [f/total_fit for f in fitness]
    probabilities = [sum(relative_fitness[:i+1])
                     for i in range(len(relative_fitness))]
    return probabilities
```

Figure 11: calculate the probability for each chromosome.

```python
def roulette_wheel_selection(population, fitness, n_parents):
    '''
    This function Selecting the best individuals in the current generation as parents
    for producing the offspring of the next generation.

    Parameters
    ----------
    population : The corresponding weights of the population.
    fitness    : The sum of products between each input and its corresponding weight.
    n_parents : number of parents mating.

    Returns
    -------
    parents   : The sum of products between each input and its corresponding weight.
    '''

    probabilities = get_probability_list(fitness)

    chosen_parents = numpy.empty((n_parents, population.shape[1]))

    for p in range(n_parents):
        r = random.random()
        for (i, individual) in enumerate(population):
            if r <= probabilities[i]:
                chosen_parents[p, :] = population[i, :]
                #chosen_parents.append(list(individual))
                break

    return chosen_parents
```

Figure 12: Roulette Wheel implementation.

4 REPLACEMENT

A replacement/deletion strategy defines which member in the current population is forced to perish (or vacate a slot) in order to make room for the new offspring to compete (or occupy a slot) in the next iteration.

```
fitness_df = pd.DataFrame()
fits  = []

for i in range(iters):

    for g in range(n_generation):

        # Measuring the fitness of each chromosome in the population.
        fitness = cal_pop_fitness(equation_param, new_population)

        # Selecting the best parents in the population for mating.
        parents = roulette_wheel_selection(new_population, fitness,  n_parents_mating)

        # Generating next generation using crossover.
        offspring_crossover = single_point_crossover(parents,(pop_total_size[0]-parents.shape[0], n_weights), crossover_rate)

        # Adding some variations to the offsrping using mutation.
        offspring_mutation =  uniform_mutation(offspring_crossover, mutation_rate)

        # Creating the new population based on the parents and offspring.
        new_population[0:parents.shape[0], :] = parents
        new_population[parents.shape[0]:, :]  = offspring_mutation

        # The best result in the current iteration.
        #print("Best result : ", numpy.max(numpy.sum(new_population*equation_inputs, axis=1)))

        fits.append(numpy.max(numpy.sum(new_population * equation_param, axis=1)))

        fitness_df['fitness'] = pd.Series((numpy.sum(new_population * equation_param, axis=1)))
```

Figure 13: Replacement implementation.

5 TERMINATION CONDITION

We stopped running the algorithm after 20 generations with 100 times GA.

```
n_generation      = 100
iters = 20
```

Figure 14: Termination conditions.

6 RESULTS

6.1 User Information

In the beginning, we ask the patient to enter some information and store them in a DataFrame (custom_df) to prepare his rehabilitation plan.

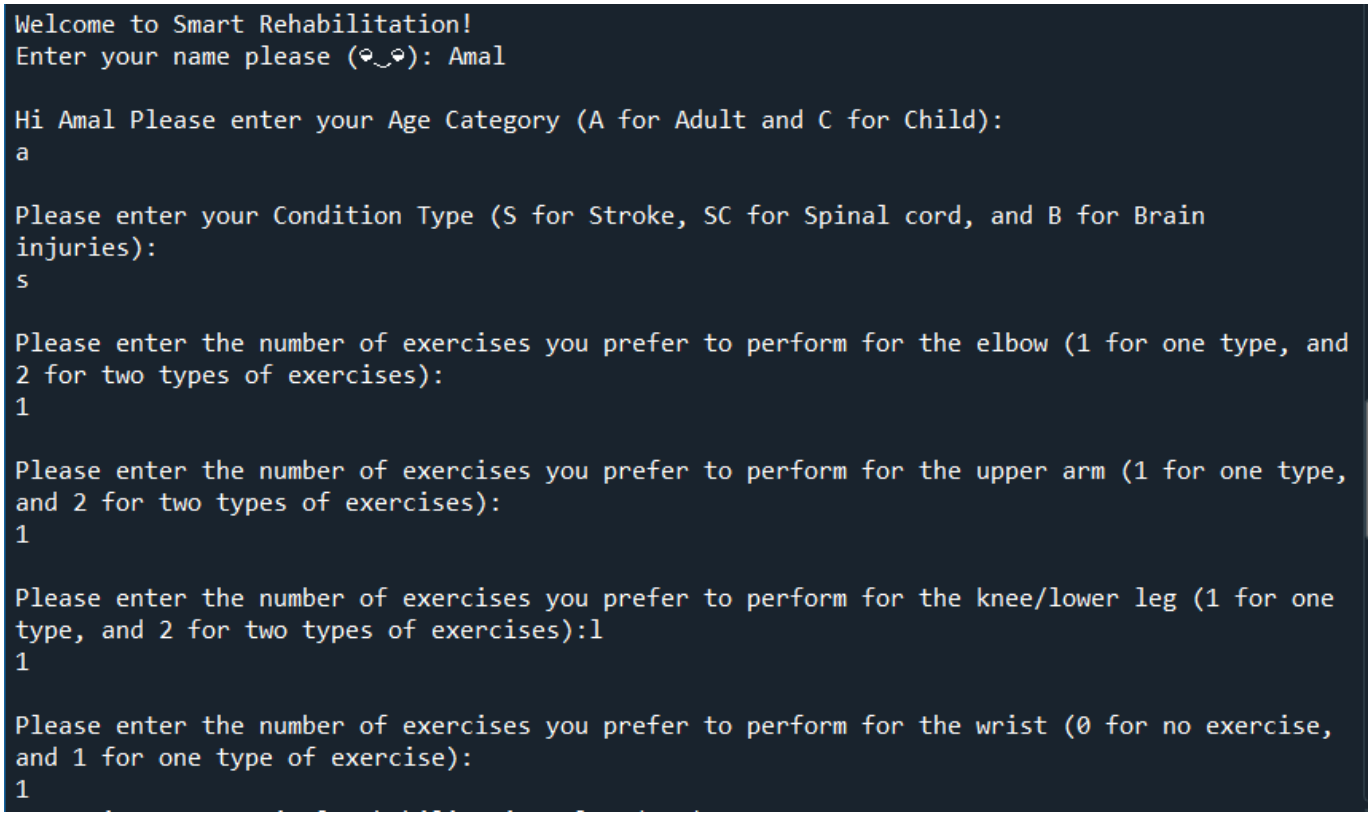

Figure 15: Getting information from a patient.

For the same information from the patient, we change the GA parameters and get results as follows:

6.2 Result 1

Parameter	**value**
Num of individuals	10
Num of weights	3
Crossover rate	0.6
Mutation rate	0.3
Fitness average	**Best solution fitness**
20.48212503	-

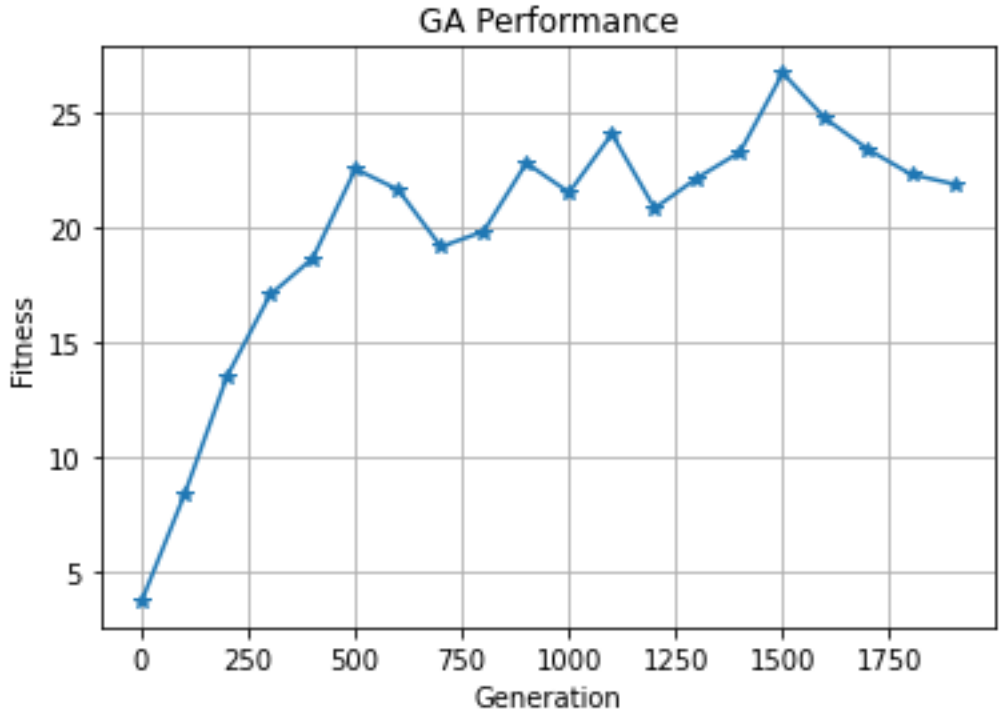

Figure 16: Result 1 graph.

6.3 Result 2

Parameter	**value**
Num of individuals	20
Num of weights	3
Crossover rate	0.6
Mutation rate	0.3
Fitness average	**Best solution fitness**
20.07154533	20.48212503

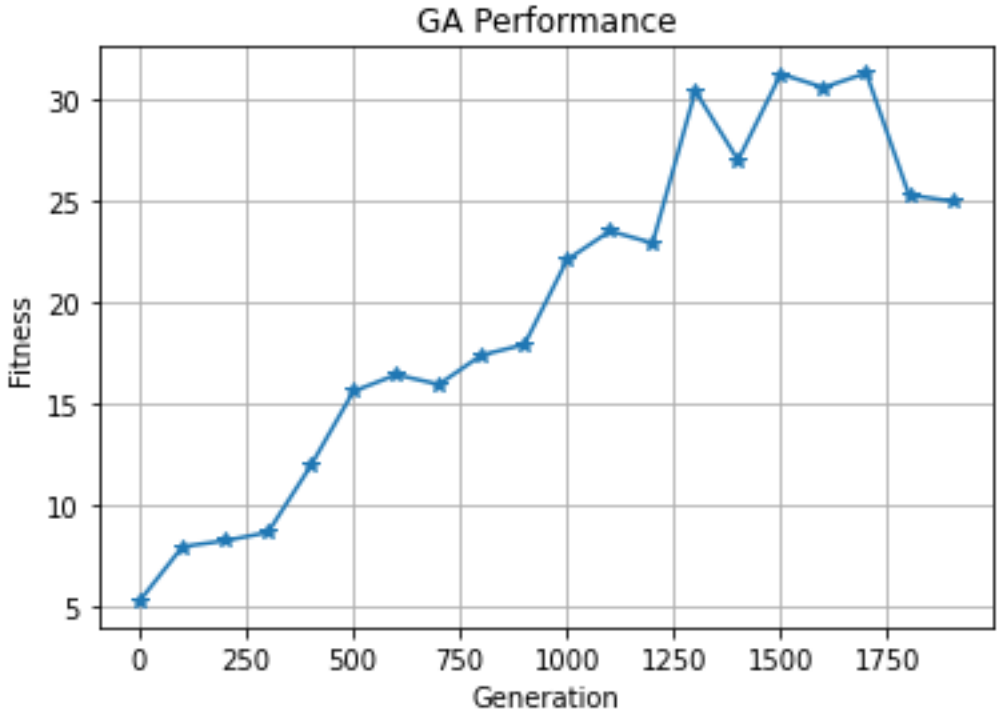

Figure 17: Result 2 graph.

6.4 Result 3

Parameter	**value**
Num of individuals	30
Num of weights	3
Crossover rate	0.6
Mutation rate	0.3
Fitness average	**Best solution fitness**
22.85638638	20.48212503

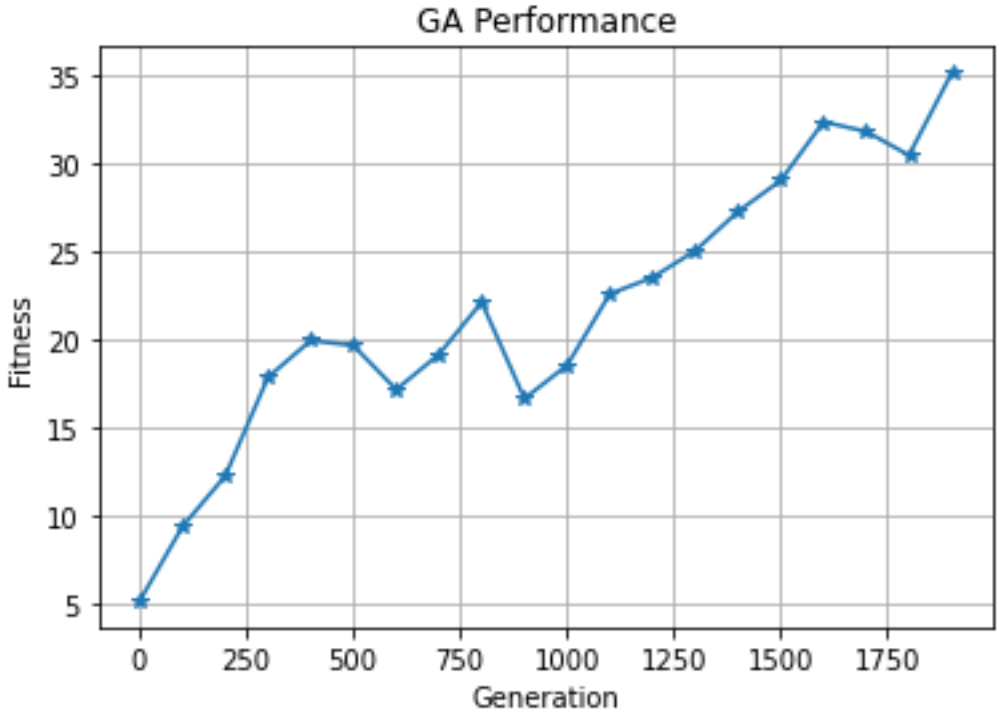

Figure 18: Result 3 graph.

6.5 Result 4

Parameter	**value**
Num of individuals	30
Num of weights	3
Crossover rate	0.7
Mutation rate	0.3
Fitness average	**Best solution fitness**
46.49883136	22.85638638

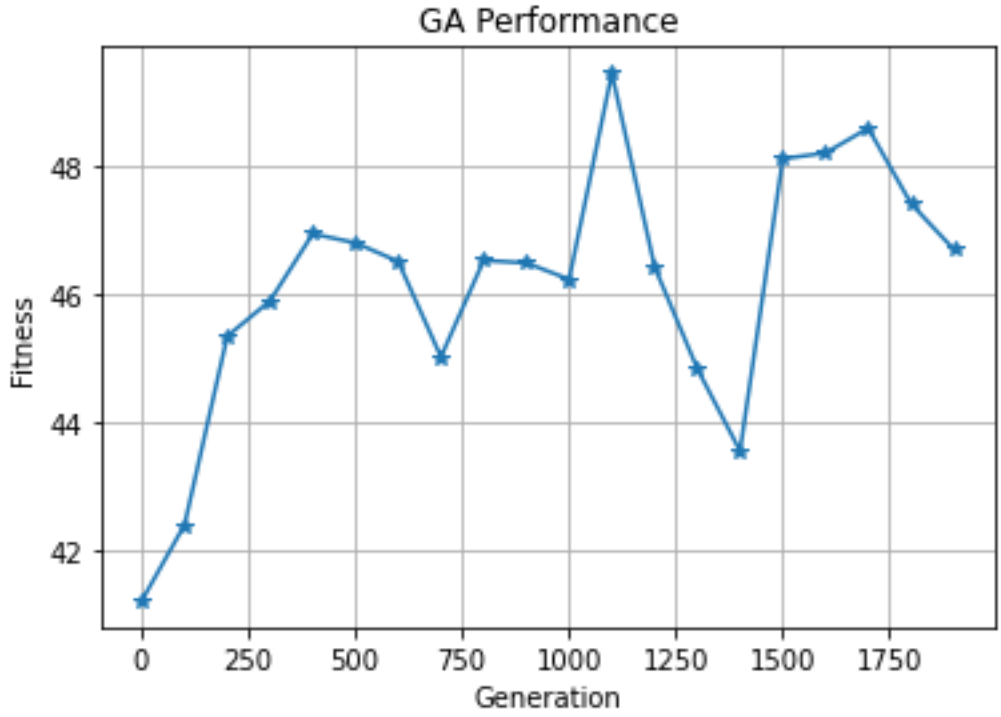

Figure 19: Result 4 graph.

6.6 Result 5

Parameter	**value**
Num of individuals	30
Num of weights	3
Crossover rate	0.8
Mutation rate	0.3
Fitness average	**Best solution fitness**
45.02166114	46.49883136

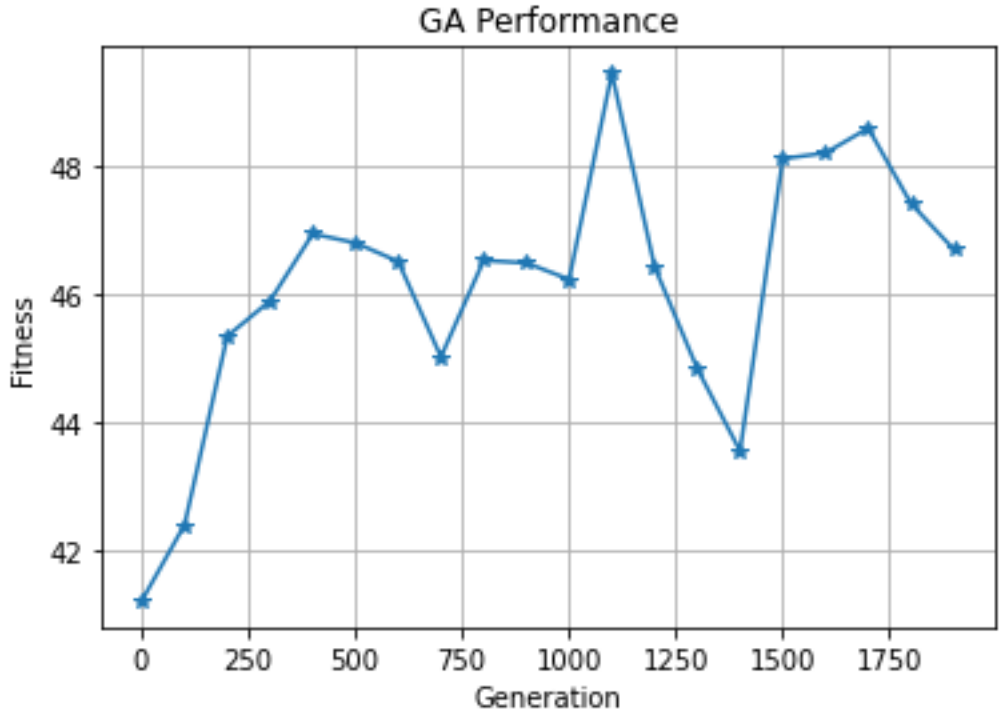

Figure 20: Result 5 graph.

6.7 Result 6

Parameter	value
Num of individuals	30
Num of weights	3
Crossover rate	0.9
Mutation rate	0.3
Fitness average	**Best solution fitness**
23.52061301	46.49883136

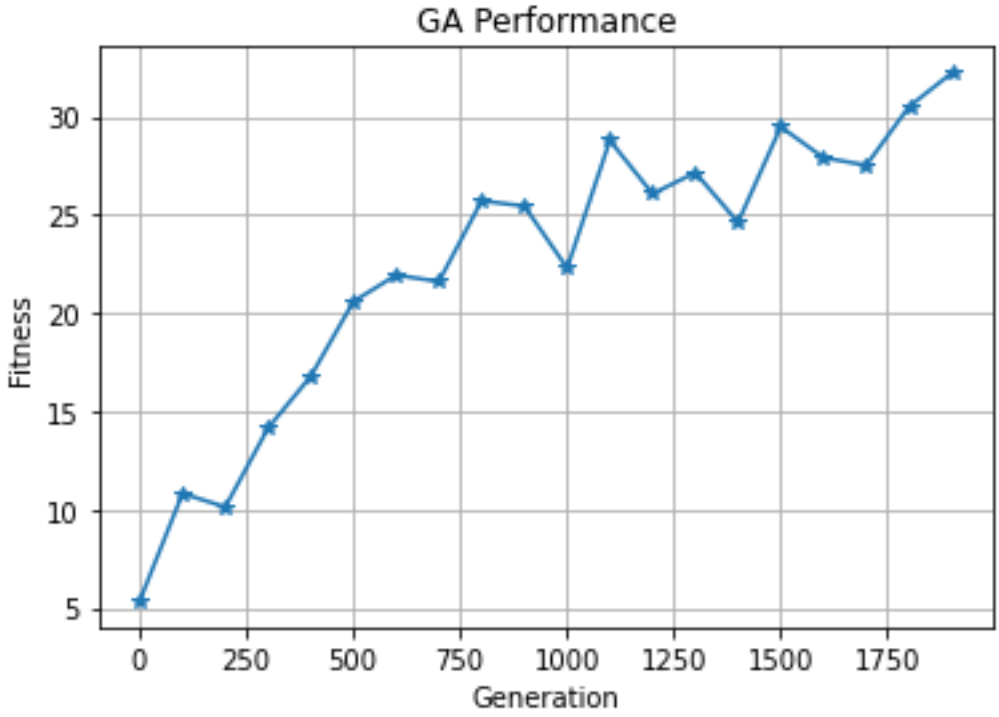

Figure 21: Result 6 graph.

6.8 Result 7

Parameter	**value**
Num of individuals	30
Num of weights	3
Crossover rate	0.7
Mutation rate	0.2
Fitness average	**Best solution fitness**
53.05064501	46.49883136

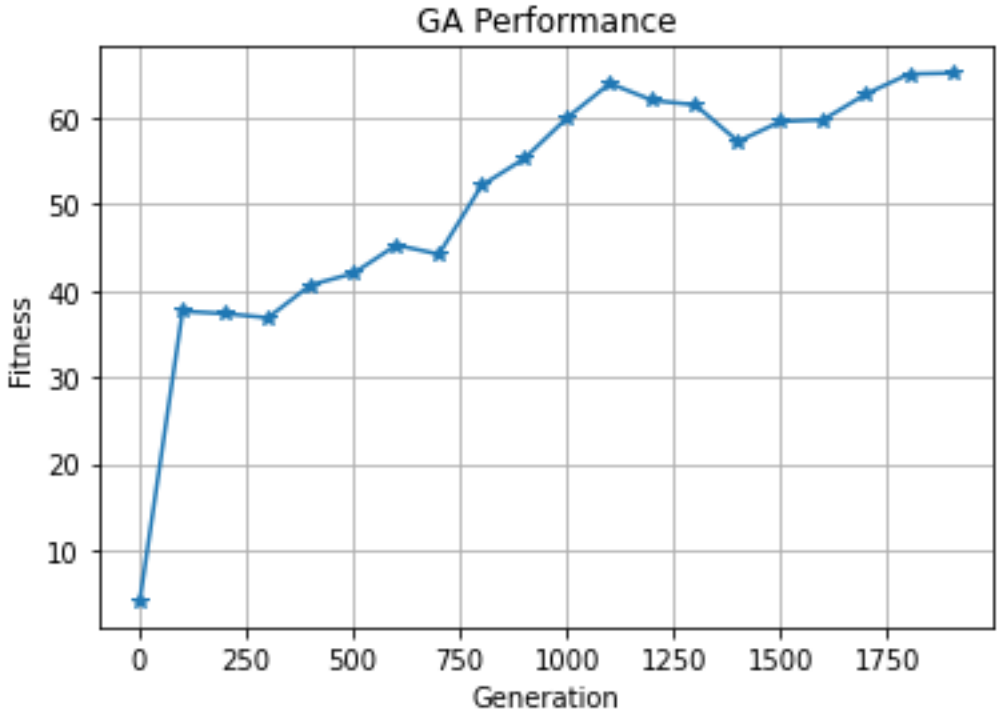

Figure 22: Result 7 graph.

6.9 Result 8

Parameter	**value**
Num of individuals	30
Num of weights	3
Crossover rate	0.7
Mutation rate	0.1
Fitness average	**Best solution fitness**
10.98219076	53.05064501

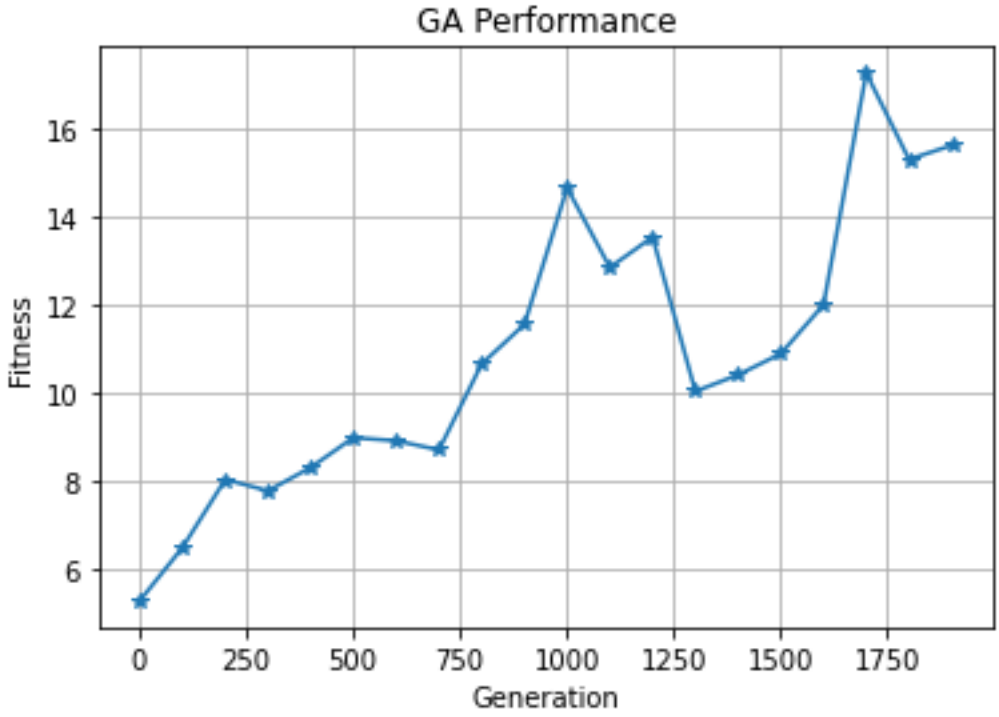

Figure 23: Result 8 graph.

6.10 Result 9

Parameter	**value**
Num of individuals	30
Num of weights	3
Crossover rate	0.7
Mutation rate	0.4
Fitness average	**Best solution fitness**
27.09441861	49.40730852

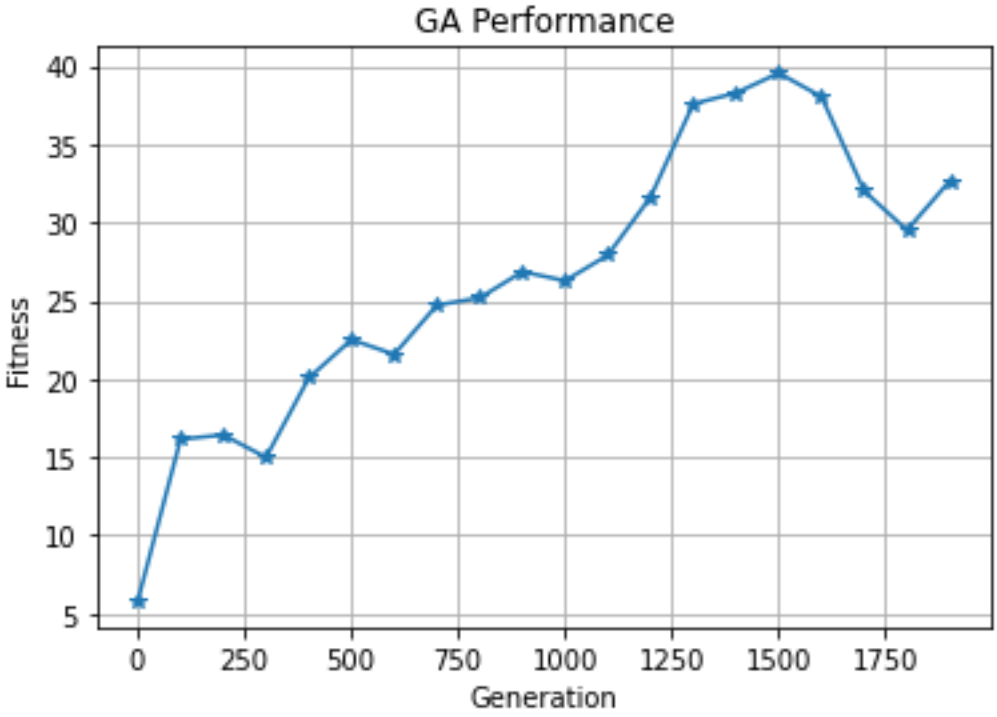

Figure 24: Result 9 graph.

7 Results Analysis

After a number of the experiment the best parameters are as follows:

Parameter	**value**
Num of individuals	30
Num of weights	3
Crossover rate	0.7
Mutation rate	0.2
Fitness average	
53.05064501	

- When we increase the number of individuals the fitness increased as shown in Result 3.
- When we increase the crossover rate the fitness increased as shown in Results 4, then the results begin to decrease again (as shown in Results 5,6) so the best values = 0.7.

- When we try to decrease the mutation rate the results improved as shown in Results 7, then the results begin to decrease again (as shown in Results 8,9) so the best values = 0.2.

Observation:

The number of individuals did not make a significant impact on outputs, but the crossover and mutation rates make a significant impact on the fitness results.

8 Running the best solution

```
Welcome to Smart Rehabilitation!
Enter your name please (◕‿◕): Amal

Hi Amal Please enter your Age Category (A for Adult and C for Child):
a

Please enter your Condition Type (S for Stroke, SC for Spinal cord, and B for Brain
injuries):
s

Please enter the number of exercises you prefer to perform for the elbow (1 for one type, and
2 for two types of exercises):
1

Please enter the number of exercises you prefer to perform for the upper arm (1 for one type,
and 2 for two types of exercises):
1

Please enter the number of exercises you prefer to perform for the knee/lower leg (1 for one
type, and 2 for two types of exercises):l
1

Please enter the number of exercises you prefer to perform for the wrist (0 for no exercise,
and 1 for one type of exercise):
1
```

Figure 25: User inputs.

```
Elbow :

1. Elbow flexor using free weights .

Upper Arm :

1. Shoulder external rotator using free weights .

Knee/ Lower leg :

1. Knee extensor using a device .

Wrist :

1. Wrist extensor using free weights.

Hope you fast recovery!
```

Figure 26: Rehabilitation plan.

```
In [7]: df
Out[7]:
```

	Body	Exercises	Condition	AgeCategory	fitness
0	Elbow	Elbow flexor using free weights	Stroke	Adult	39.876981
1	Knee/ Lower leg	Knee extensor using a device	Stroke	Adult	39.876981
2	Wrist	Wrist extensor using free weights	Stroke	Adult	39.876981
3	Wrist	Wrist flexor using free weights	Stroke	Adult	38.988098
4	Upper Arm	Shoulder external rotator using free weights	Stroke	Adult	38.824325
5	Elbow	Elbow extensor using free weights	Stroke	Adult	38.580819
6	Knee/ Lower leg	Knee flexor using a device	Stroke	Adult	38.053736
7	Upper Arm	Shoulder extensor	Stroke	Adult	37.778430

Figure 27:Rrehabilitation plan as a data frame.