# Exercise 4 – pseudo-code:

Function **genetic programming** (lambda, n, m, data, time budget):

Initialize parameters (terminal rate, float range, max depth, tournament size, crossover rate, mutation rate)

***Initialize Population***

Start timer

While timer doesn't exceed time budget:

Parents ***= select parents*** from Population (produces 'lambda * crossover rate' many parents)

Children = ***Crossover*** Parents (for every 2 parents)

Children = ***Mutation*** Children ('mutation rate' chance that each child mutates)

Population = Children + Best performing from previous population (until 'lambda' is reached)

Get best solution from Population

Return best solution

Function **initialize population**:

for each individual in population size:

expression = ***generate tree***

population append(expression)

return population

Function **generate tree**:

If current depth >= max depth-1 or random number < terminal rate

Return random float between float range (rounded)

Else:

Randomly choose an expression function

If function takes one input:

Return chosen function + ***generate tree*** (current depth + 1)

If it takes four:

Return chosen function + ***generate tree*** (current depth + 1) + 3 more times

If it takes two:

Return chosen function + ***generate tree*** (current depth + 1) + 1 more time

Function ***select parents***:

Sample tournament size many solutions from the population

Best solution = min(sample) ***evaluate fitness***

Return best solution

Function **crossover:**

If both parents are terminal nodes:

Parents are just swapped

Else:

Select a **random subtree**

**Replace subtree** of one parent with the others

do the same with the other parents

return Parents

Function **mutation:**

If solution is just a terminal node:

**Generate tree** (an entirely new one)

Else:

Select a **random subtree**

**Mutate subtree** of solution

return solution

Function **random subtree**:

Choose a random element

If it's a list:

**Random subtree** (chosen element)

If it's an expression

Return that expression with its inputs

If it's a float

Return the float

Function **replace subtree:**

If found the original subtree

Return new subtree

Else:

Return **replace subtree** for every item in the current tree

Function **mutate subtree:**

If found the original subtree

Return **generate tree** (starting from the current depth)

Else:

Return **mutate subtree** for every item in the current tree (current depth + 1)

# Exercise 5:

My tuneable parameters are:

- Crossover rate – how many solutions undergo crossover
- Mutation rate – how many offspring also get mutated
- Tournament size – the number of solutions selected per tournament
- Max depth – how deep the trees can get when initially made
- Terminal rate – the likelihood of a terminal node appearing early (growth method of tree generation)
- Value range – the possible values that terminal nodes can initially have

Lambda and time budget were kept the same (100 and 10 respectably) for every experiment and Shampoo Sales Dataset (kaggle.com) was used as a basic dataset for testing

| **Crossover Rate:** | **Mutation Rate:** | **Tournament Size:** | **Max Depth:** | **Terminal Rate:** | **Value Range:** | **Average Fitness over 100 runs (rounded):** |
|---|---|---|---|---|---|---|
| 0.8 | 0.2 | 2 | 3 | 0.3 | 10 | 10914 |
| 0.8 | **0.7** | 2 | 3 | 0.3 | 10 | 14455 |
| 0.8 | **0.5** | 2 | 3 | 0.3 | 10 | 11772 |
| 0.8 | **0.3** | 2 | 3 | 0.3 | 10 | 9909 |
| 0.8 | **0.4** | 2 | 3 | 0.3 | 10 | 10187 |
| 0.8 | **0.25** | 2 | 3 | 0.3 | 10 | 9786 |
| **0.3** | 0.2 | 2 | 3 | 0.3 | 10 | 17544 |
| **0.5** | 0.2 | 2 | 3 | 0.3 | 10 | 16119 |
| **0.7** | 0.2 | 2 | 3 | 0.3 | 10 | 11764 |
| **0.9** | 0.2 | 2 | 3 | 0.3 | 10 | 13854 |
| **0.75** | 0.2 | 2 | 3 | 0.3 | 10 | 10505 |
| 0.8 | 0.2 | **3** | 3 | 0.3 | 10 | 12355 |
| 0.8 | 0.2 | **5** | 3 | 0.3 | 10 | 12633 |
| 0.8 | 0.2 | **7** | 3 | 0.3 | 10 | 17825 |
| 0.8 | 0.2 | 2 | **10** | 0.3 | 10 | 28193 |
| 0.8 | 0.2 | 2 | **5** | 0.3 | 10 | 18537 |
| 0.8 | 0.2 | 2 | **4** | 0.3 | 10 | 13186 |
| 0.8 | 0.2 | 2 | **2** | 0.3 | 10 | 15271 |
| 0.8 | 0.2 | 2 | 3 | **0.5** | 10 | 14713 |
| 0.8 | 0.2 | 2 | 3 | **0.35** | 10 | 10301 |
| 0.8 | 0.2 | 2 | 3 | **0.1** | 10 | 10744 |
| 0.8 | 0.2 | 2 | 3 | **0.25** | 10 | 13165 |
| 0.8 | 0.2 | 2 | 3 | 0.3 | **100** | 18604 |
| 0.8 | 0.2 | 2 | 3 | 0.3 | **50** | 15928 |
| 0.8 | 0.2 | 2 | 3 | 0.3 | **20** | 9424 |
| 0.8 | 0.2 | 2 | 3 | 0.3 | **25** | 10791 |
| 0.8 | 0.2 | 2 | 3 | 0.3 | **15** | 8461 |
| **0.75** | **0.25** | **2** | **3** | **0.35** | **15** | 7303 |

It should be noted that just because the parameter settings work here, it doesn't necessarily mean it will work well with every dataset, as a more complex dataset (the Shampoo Sales Dataset is somewhat basic) may require more complex trees, determined by a greater Max Depth etc…

**Statistics for box plot:**

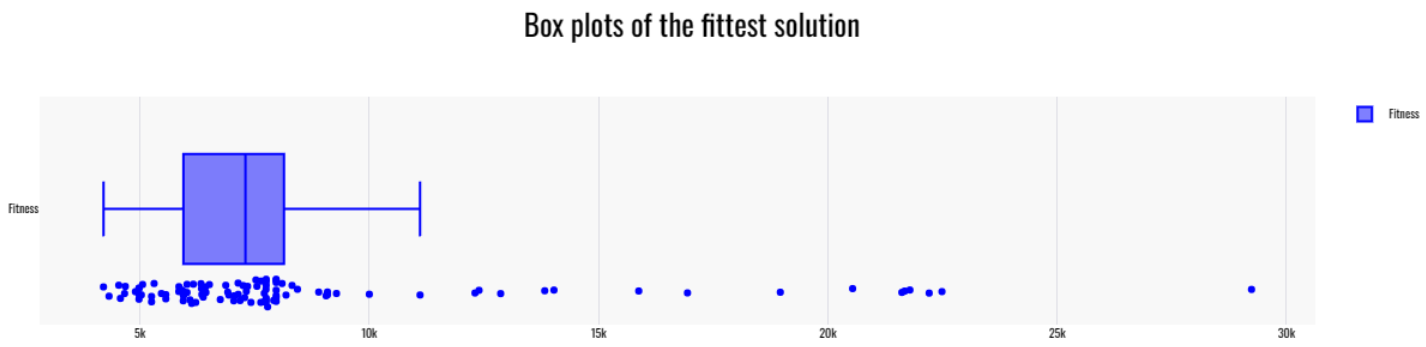
Minimum Fitness: 4207

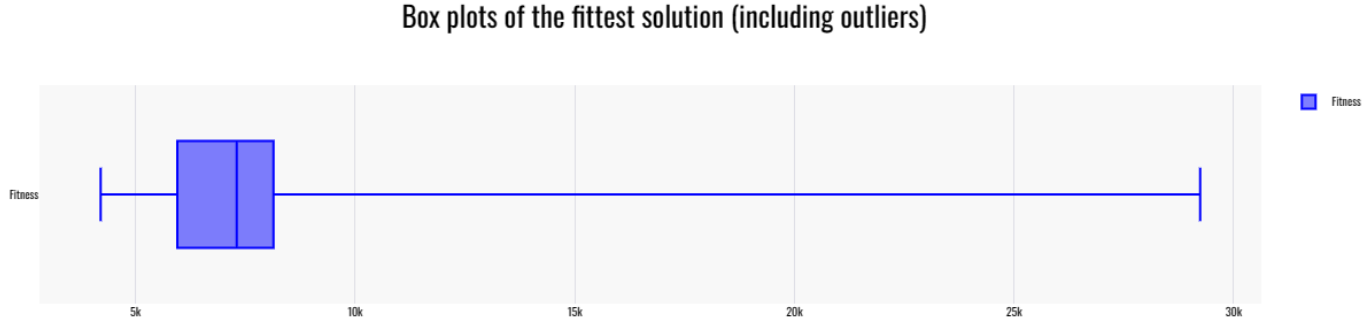
First Quartile (Q1): 5953

Median Fitness: 7303

Third Quartile (Q3): 8123

Maximum Fitness: 29239




## Screenshot from Docker:

```
lmb-dev@lmbdev-VirtualBox:~/Documents/ec2024-lab2-lxb150$ sudo docker run ec2024-lab2-lxb150
[sudo] password for lmb-dev:
Test version: 18:19, Feb 26th 2024
Question 1: Evaluation of expression.

The solution is probably correct.

Question 2: Evaluation of the fitness function.

The solution is probably correct.

Question 3: Does the GP produce better solutions? (approx 10 min)

lxb150  0.000000        0.000000        0.824672
The solution is probably correct.
```