Evolutionary Algorithm Project Report

Muhammad Khubaib Akram

Attarde Mohit Chandrashekhar

Master in AIML

Module Leader: Conor Ryan

Problem Statement:

Predict whether income exceeds $50K/yr based on census data.

Objectives:

The main objective is to build a classifier using evolutionary algorithms, Grammatical Evolution (GE) to predict the income whether it exceeds $50K/yr or not.

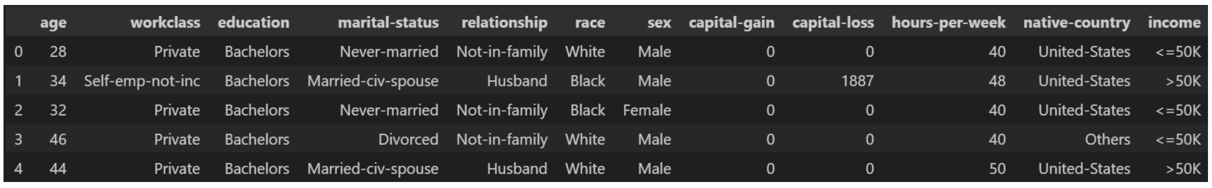
Dataset Overview:

The Adult dataset, also known as the "Census Income" dataset, includes various features like age, workclass, education, marital status, relationship, race, sex, capital gain, capital loss, hours per week, and native country. The target variable is income, categorised into two classes: ">50K" and "<=50K".

We had **12 columns in our training data** set:

1. Age
2. Workclass
3. Education
4. Marital-status
5. Relationship
6. Race
7. Sex
8. Capital-gain
9. Capital-loss
10. Hours-per-week
11. Native-country
12. Income

Below is a screenshot of the dataset:



	age	workclass	education	marital-status	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	28	Private	Bachelors	Never-married	Not-in-family	White	Male	0	0	40	United-States	<=50K
1	34	Self-emp-not-inc	Bachelors	Married-civ-spouse	Husband	Black	Male	0	1887	48	United-States	>50K
2	32	Private	Bachelors	Never-married	Not-in-family	Black	Female	0	0	40	United-States	<=50K
3	46	Private	Bachelors	Divorced	Not-in-family	White	Male	0	0	40	Others	<=50K
4	44	Private	Bachelors	Married-civ-spouse	Husband	White	Male	0	0	50	United-States	>50K

Listing of features and their types:

Sr #	Column	Type
1	age	Continuous
2	workclass (Private, Self-emp-not-inc, Local-gov, State-gov)	Categorical
3	education(Bachelors, Some-college, HS-grad, Masters, Doctorate)	Categorical

4	marital-status(Married-civ-spouse, Divorced, Never-married)	Categorical
5	relationship(Wife, Husband, Not-in-family, Other-relative)	Categorical
6	race(White, Asian-Pac-Islander, Black)	Categorical
7	sex(Female, Male)	Categorical
8	capital-gain	Continuous
9	capital-loss	Continuous
10	hours-per-week	Continuous
11	native-country(United-States, Others)	Categorical

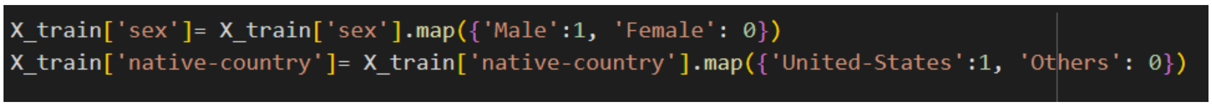
Pre-processing the Data

Mapping Sex and Native-country to 0's and 1's

The features sex and native-country had only two values we mapped them to 0 and 1s.

Sex (Male=1 and female=0)

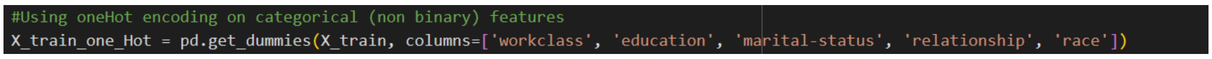
Native-country(United-States=1 and Others=0)



```
X_train['sex']= X_train['sex'].map({'Male':1, 'Female': 0})
X_train['native-country']= X_train['native-country'].map({'United-States':1, 'Others': 0})
```

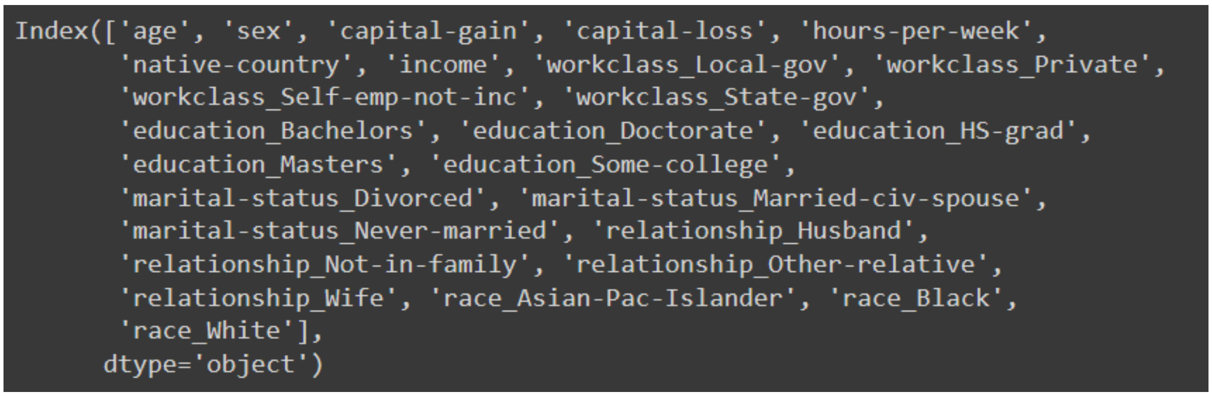
Hot-Encoding the Categorical Data

One-hot encoding is a technique used to convert categorical data, especially categorical labels, into a numerical format that can be fed into machine learning models. This is often necessary when working with algorithms that require numerical inputs.



```
#Using oneHot encoding on categorical (non binary) features
X_train_one_Hot = pd.get_dummies(X_train, columns=['workclass', 'education', 'marital-status', 'relationship', 'race'])
```

After one-hot encoding we have 25 columns in our data. Below are the column names



```
Index(['age', 'sex', 'capital-gain', 'capital-loss', 'hours-per-week',
       'native-country', 'income', 'workclass_Local-gov', 'workclass_Private',
       'workclass_Self-emp-not-inc', 'workclass_State-gov',
       'education_Bachelors', 'education_Doctorate', 'education_HS-grad',
       'education_Masters', 'education_Some-college',
       'marital-status_Divorced', 'marital-status_Married-civ-spouse',
       'marital-status_Never-married', 'relationship_Husband',
       'relationship_Not-in-family', 'relationship_Other-relative',
       'relationship_Wife', 'race_Asian-Pac-Islander', 'race_Black',
       'race_White'],
      dtype='object')
```

We dropped income from our training set.

Grammatical Evolution (GE)

We opted for Grammatical Evolution (GE) as our tool for predicting income. To effectively utilize GE, it is imperative to create a grammar file. In the realm of Grammatical Evolution, a grammar file plays a pivotal role in steering the evolutionary process by outlining the syntax and structure of potential solutions.

Key attributes of a grammar file in GE include:

1. **Backus-Naur Form (BNF):** Grammar files in GE are commonly composed in BNF or a similar notation. BNF provides a formal and succinct representation of the language's grammar.
2. **Non-terminal Symbols:** These symbols act as placeholders, awaiting substitution with actual elements, whether terminal or non-terminal. They are typically enclosed in angle brackets, such as <expression>.
3. **Terminal Symbols:** These symbols represent the tangible elements of the language, such as numbers, operators, or variable names, that manifest in the final program or expression.
4. **Production Rules:** These rules dictate how non-terminal symbols can be replaced with a combination of terminal and non-terminal symbols. For instance, <expression> ::= <number> | (<expression> <operator> <expression>).

Grammar:

Grammar <log_op> ::= <conditional_branches> | and_(<log_op>,<log_op>) | or_(<log_op>,<log_op>) | not_(<log_op>) | <boolean_feature>

<conditional_branches> ::= less_than_or_equal(<num_op>,<num_op>) | greater_than_or_equal(<num_op>, <num_op>)

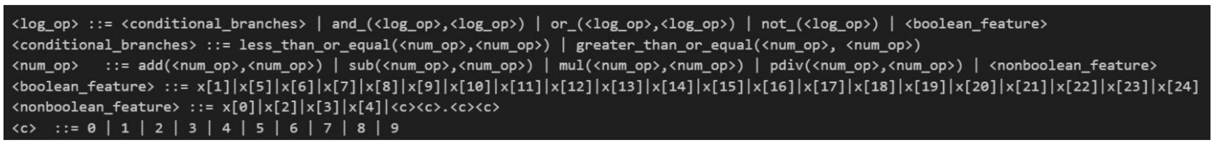
<num_op>   ::= add(<num_op>,<num_op>) | sub(<num_op>,<num_op>) | mul(<num_op>,<num_op>) | pdiv(<num_op>,<num_op>) | <nonboolean_feature>

<boolean_feature> ::= x[1]|x[5]|x[6]|x[7]|x[8]|x[9]|x[10]|x[11]|x[12]|x[13]|x[14]|x[15]|x[16]|x[17]|x[18]|x[19]|x[20]|x[21]|x[22]|x[23]|x[24]

<nonboolean_feature> ::= x[0]|x[2]|x[3]|x[4]|<c><c>.<c><c>

<c> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Below is the screenshot of our grammar file



```
<log_op> ::= <conditional_branches> | and_(<log_op>,<log_op>) | or_(<log_op>,<log_op>) | not_(<log_op>) | <boolean_feature>
<conditional_branches> ::= less_than_or_equal(<num_op>,<num_op>) | greater_than_or_equal(<num_op>, <num_op>)
<num_op>   ::= add(<num_op>,<num_op>) | sub(<num_op>,<num_op>) | mul(<num_op>,<num_op>) | pdiv(<num_op>,<num_op>) | <nonboolean_feature>
<boolean_feature> ::= x[1]|x[5]|x[6]|x[7]|x[8]|x[9]|x[10]|x[11]|x[12]|x[13]|x[14]|x[15]|x[16]|x[17]|x[18]|x[19]|x[20]|x[21]|x[22]|x[23]|x[24]
<nonboolean_feature> ::= x[0]|x[2]|x[3]|x[4]|<c><c>.<c><c>
<c>  ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

We are using 3 Logical operations

1. And
2. Or
3. not

Two conditional Branches

1. Less than or equal to
2. Greater than or equal

We have 4 Non boolean features which are below:

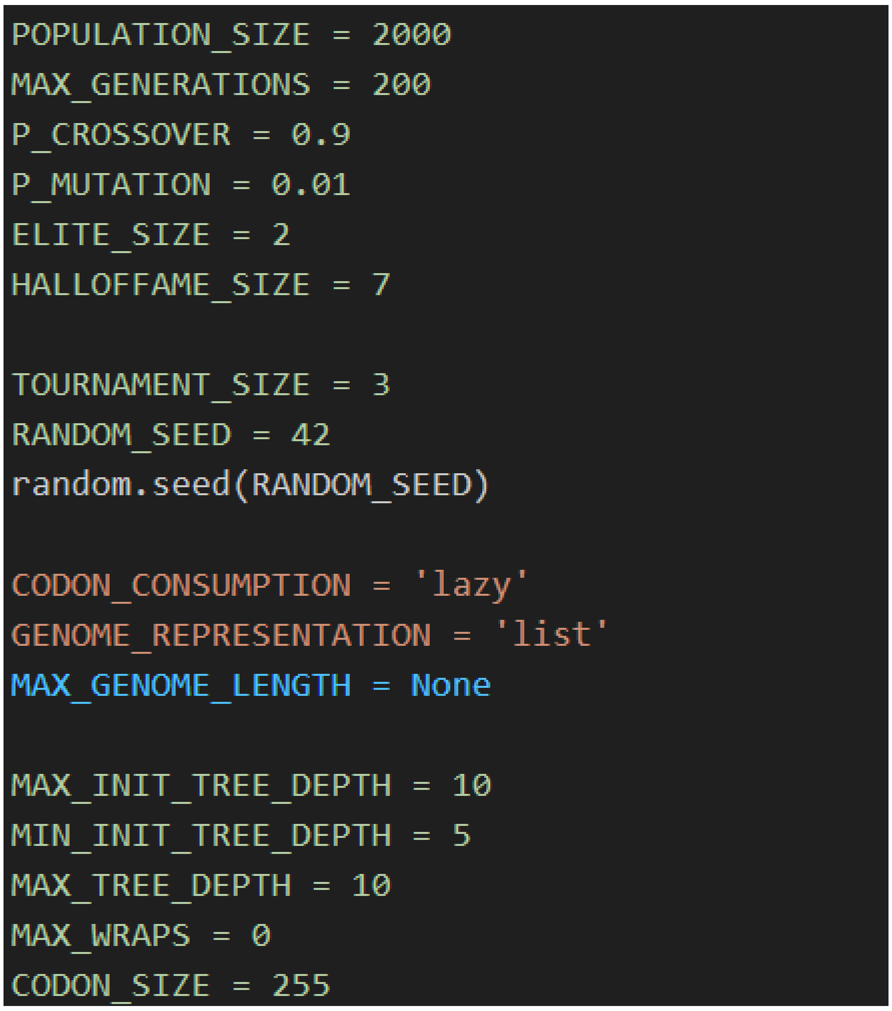
1. Age
2. Capital-gain
3. Capital-loss
4. Hours-per-week

All others(21) are boolean features as we have hot-encoded the categorical columns.

Experiments

Parameter Tuning: Experiment with different parameters like population size, mutation rate, crossover rate, and the depth of tree.
Multiple Runs: Perform multiple runs of the algorithm to account for the stochastic nature of evolutionary algorithms.

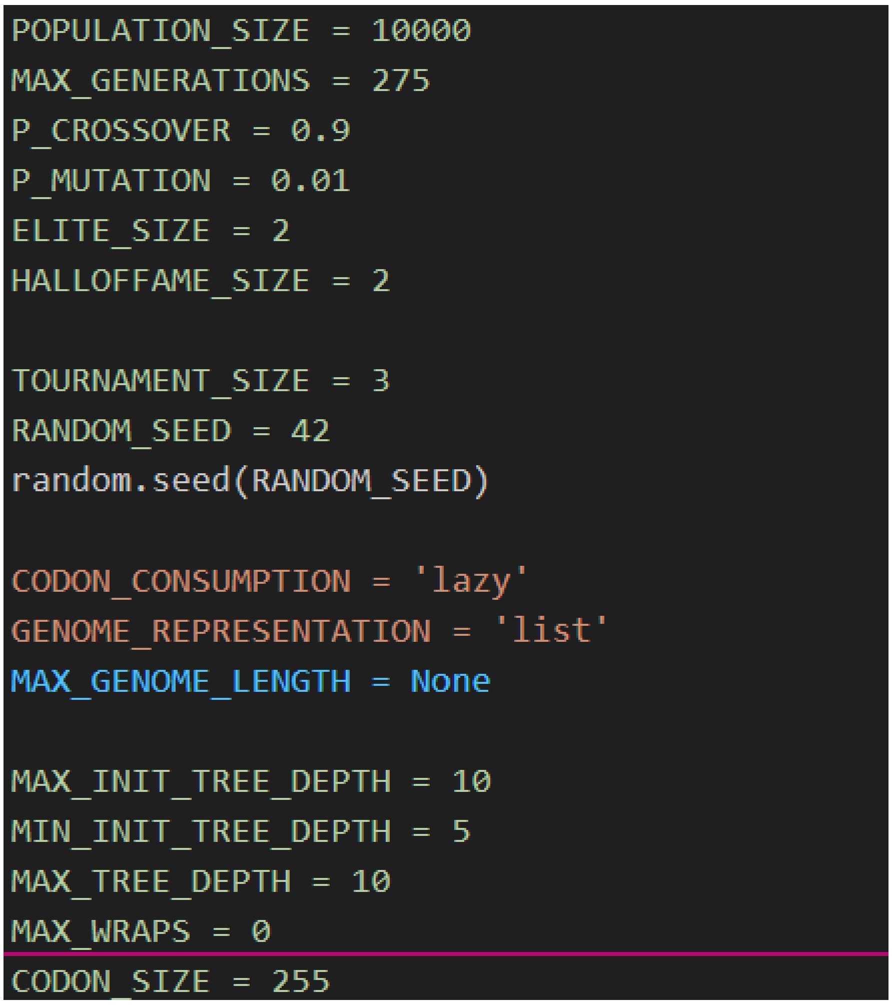
Setup 1



```
POPULATION_SIZE = 2000
MAX_GENERATIONS = 200
P_CROSSOVER = 0.9
P_MUTATION = 0.01
ELITE_SIZE = 2
HALLOFFAME_SIZE = 7

TOURNAMENT_SIZE = 3
RANDOM_SEED = 42
random.seed(RANDOM_SEED)

CODON_CONSUMPTION = 'lazy'
GENOME_REPRESENTATION = 'list'
MAX_GENOME_LENGTH = None

MAX_INIT_TREE_DEPTH = 10
MIN_INIT_TREE_DEPTH = 5
MAX_TREE_DEPTH = 10
MAX_WRAPS = 0
CODON_SIZE = 255
```

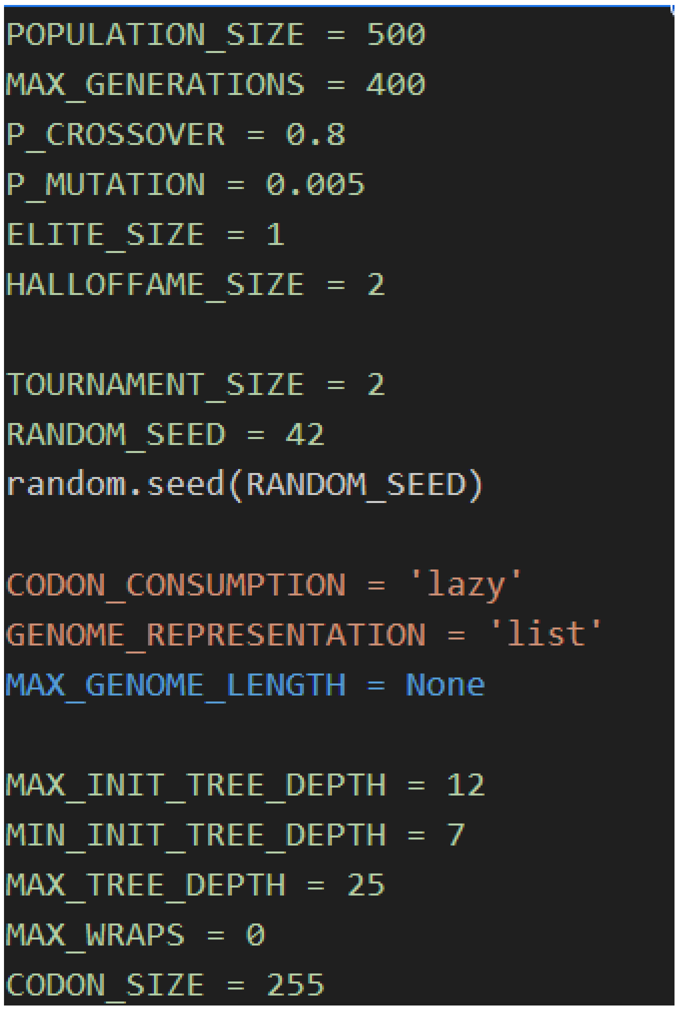
Training Fitness: 0.22750000000000004
Depth: 10
Length of the genome: 759
Used portion of the genome: 0.11
Number of invalids = 0
Kaggle Score: 0.7539252

Setup 2


```
POPULATION_SIZE = 10000
MAX_GENERATIONS = 275
P_CROSSOVER = 0.9
P_MUTATION = 0.01
ELITE_SIZE = 2
HALLOFFAME_SIZE = 2

TOURNAMENT_SIZE = 3
RANDOM_SEED = 42
random.seed(RANDOM_SEED)

CODON_CONSUMPTION = 'lazy'
GENOME_REPRESENTATION = 'list'
MAX_GENOME_LENGTH = None

MAX_INIT_TREE_DEPTH = 10
MIN_INIT_TREE_DEPTH = 5
MAX_TREE_DEPTH = 10
MAX_WRAPS = 0
CODON_SIZE = 255
```

Training Fitness: 0.2253846153846154
Depth: 10
Length of the genome: 614
Used portion of the genome: 0.12
Gen = 275 ,
Number of invalids = 0
kaggle score= 0.76677133

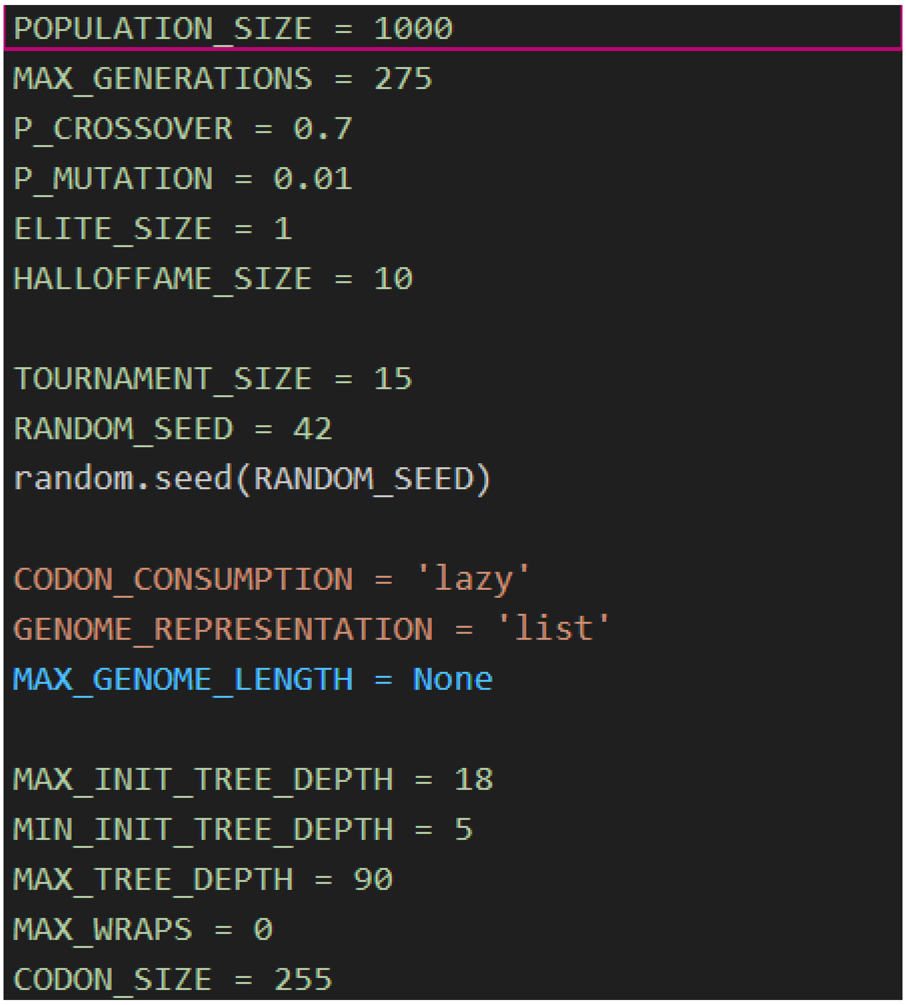
Setup 3



```
POPULATION_SIZE = 500
MAX_GENERATIONS = 400
P_CROSSOVER = 0.8
P_MUTATION = 0.005
ELITE_SIZE = 1
HALLOFFAME_SIZE = 2

TOURNAMENT_SIZE = 2
RANDOM_SEED = 42
random.seed(RANDOM_SEED)

CODON_CONSUMPTION = 'lazy'
GENOME_REPRESENTATION = 'list'
MAX_GENOME_LENGTH = None

MAX_INIT_TREE_DEPTH = 12
MIN_INIT_TREE_DEPTH = 7
MAX_TREE_DEPTH = 25
MAX_WRAPS = 0
CODON_SIZE = 255
```

Training Fitness: 0.22269230769230774
Depth: 10
Length of the genome: 462
Used portion of the genome: 0.45
Gen = 400,
Number of invalids = 0
Kaggle score= 0.75335426

Setup 4



```
POPULATION_SIZE = 1000
MAX_GENERATIONS = 275
P_CROSSOVER = 0.7
P_MUTATION = 0.01
ELITE_SIZE = 1
HALLOFFAME_SIZE = 10

TOURNAMENT_SIZE = 15
RANDOM_SEED = 42
random.seed(RANDOM_SEED)

CODON_CONSUMPTION = 'lazy'
GENOME_REPRESENTATION = 'list'
MAX_GENOME_LENGTH = None

MAX_INIT_TREE_DEPTH = 18
MIN_INIT_TREE_DEPTH = 5
MAX_TREE_DEPTH = 90
MAX_WRAPS = 0
CODON_SIZE = 255
```

Training Fitness= 0.19942307692307693
Depth= 32
Length of the genome: 21336
Used portion of the genome: 0.02
kaggle score = 0.77733371

Concluding Remarks:

In summary, reducing the Max_Tree_Depth size resulted in faster program execution, but the training fitness suffered. Conversely, increasing the depth size yielded better results with the same parameter values. A smaller mutation rate of 0.01 produced satisfactory results, but exceeding a mutation rate of 0.1 led to suboptimal outcomes. A larger tournament size proved beneficial in achieving superior results. Decreasing the halloffame size below 10 resulted in poor fitness outcomes. The highest Kaggle score was attained by increasing both the Max_Tree_Depth size and the tournament size.