```
begin
    using CSV
    using DataFrames
    using Random
    using MLDataUtils
    using BenchmarkTools

    using GP_NLS
    using PlutoUI
end
```

	variable	mean	min	median	max
1	Symbol("Frequency(Hz)")	2886.38	200	1600.0	20000
2	Symbol("Angle of attack(degrees)")	6.7823	0.0	5.4	22.2
3	Symbol("Chord length(m)")	0.136548	0.0254	0.1016	0.3048
4	Symbol("Free-stream velocity(m/s)")	50.8607	31.7	39.6	71.3
5	Symbol("Suction side displacement thic	0.0111399	0.000400682	0.00495741	0.058411
6	Symbol("Scaled sound pressure level(dB	124.836	103.38	125.721	140.987

```
begin
    df_data = CSV.File("../datasets/airfoil.csv") |> DataFrame

    df_data = df_data[Random.shuffle(1:end), :]

    train, test = splitobs(df_data, at = 0.7)

    train_X = convert(Matrix{Float64}, train[:, 1:end-1])
    train_y = convert(Vector{Float64}, train[:, end])

    test_X  = convert(Matrix{Float64}, test[:, 1:end-1])
    test_y  = convert(Vector{Float64}, test[:, end])

    describe(df_data, :mean, :min, :median, :max)
end
```

```
begin

    # Creating the variable nodes for the data set
    varSet = Union{Var, WeightedVar}[
        WeightedVar(var_name, i)
        for (i, var_name) in enumerate(names(df_data)[1:end-1])]

    # Creating ERC nodes
    ERCSet = ERC[
        ERC(-100.0, 100.0),
    ]

    # Creating const nodes
    ConstSet = Const[
        Const(1.5707),
        Const(3.1415),

```

```
        Const(-1.5707),
        Const(-3.1415),
    ]

    # Terminals will be picked from the union
    terminalSet = Array{Union{Const, Var, WeightedVar, ERC}}(
        vcat(ERCSet, varSet, ConstSet))

    # Using default functions set
    functionSet = defaultFunctionSet
end;
```

```
Terminal nodes:
     - GP_NLS.ERC(-100.0, 100.0)
     - GP_NLS.WeightedVar("Frequency(Hz)", 1, "1.0*Frequency(Hz)", 1.0)
     - GP_NLS.WeightedVar("Angle of attack(degrees)", 2, "1.0*Angle of attack(de
grees)", 1.0)
     - GP_NLS.WeightedVar("Chord length(m)", 3, "1.0*Chord length(m)", 1.0)
     - GP_NLS.WeightedVar("Free-stream velocity(m/s)", 4, "1.0*Free-stream veloc
ity(m/s)", 1.0)
     - GP_NLS.WeightedVar("Suction side displacement thickness(m)", 5, "1.0*Suct
ion side displacement thickness(m)", 1.0)
     - GP_NLS.Const(1.5707, "1.571")
     - GP_NLS.Const(3.1415, "3.142")
     - GP_NLS.Const(-1.5707, "-1.571")
     - GP_NLS.Const(-3.1415, "-3.142")
```

```
with_terminal() do
    println("Terminal nodes:")
    for t in terminalSet
        println("\t - $(t)")
    end
end
```

```
Function nodes:
     - GP_NLS.Func(+, 2, "+")
     - GP_NLS.Func(-, 2, "-")
     - GP_NLS.Func(GP_NLS.myprod, 2, "myprod")
     - GP_NLS.Func(GP_NLS.mydiv, 2, "mydiv")
     - GP_NLS.Func(GP_NLS.mysquare, 1, "mysquare")
     - GP_NLS.Func(GP_NLS.mysqrt, 1, "mysqrt")
     - GP_NLS.Func(GP_NLS.myexp, 1, "myexp")
     - GP_NLS.Func(GP_NLS.mylog, 1, "mylog")
```

```
with_terminal() do
    println("Function nodes:")
    for f in functionSet
        println("\t - $(f)")
    end
end
```

```
Number of nodes => 15
String infix representation => +(myprod(+(-(0.825*Free-stream velocity(m/s), 0.0
12*Frequency(Hz)), -2303.291*Suction side displacement thickness(m)), 0.098), 12
6.782)
Execution time => 20.86264
Test RMSE => 5.875739019985444
```

```
Train RMSE => 5.365616628115571
Depth => 5
```

```julia
with_terminal() do
    exec_time = @elapsed(bestsol = GP(

        # Mandatory arguments
        train_X,     # Train independent variables matrix
        train_y,     # Train dependent variable vector
        functionSet, # Function set
        terminalSet, # Terminal set

        # From this point every argument should be named and are optional
        minDepth               = 1,
        maxDepth               = 5 - 2 - 1,
        maxSize                = 25 - 4,
        popSize                = 100,
        gens                   = 100,
        mutationRate           = 0.05,
        elitism                = true,
        verbose                = false,
        init_method            = "PTC2",
        lm_optimization        = true,
        keep_linear_transf_box = true
    ))

    results = Dict(
        "Execution time"             => exec_time,
        "Train RMSE"                 => fitness(bestsol, train_X, train_y),
        "Test RMSE"                  => fitness(bestsol, test_X, test_y),
        "Number of nodes"            => true_numberofnodes(bestsol),
        "Depth"                      => depth(bestsol),
        "String infix representation" => getstring(bestsol),
    )

    for (k, v) in results
        println("$(k) => $(v)")
    end
end
```