```
begin
    using CSV
    using DataFrames
    using StatsPlots
    using NLsolve
    using Flux
    using RollingFunctions
end
```

```
celsius2kelvin (generic function with 1 method)
```

```
celsius2kelvin(c) = c + 273.15
```

```
prepare_dataframe (generic function with 1 method)
```

```
function prepare_dataframe(data_frame_path)

    _df = DataFrame(CSV.File(data_frame_path))

    start_times_stat = unique(_df, :run)

    _df.start_time =  start_times_stat[trunc.(Int, _df.run).+1, :time]
    _df.duration = _df.time - _df.start_time
    _df.T1 = _df.T1 .|> celsius2kelvin
    _df.T1prev = prepend!(_df.T1[1:end-1], _df.T1[1], )
    _df.Column1 = _df.Column1 .+ 1
    return _df
end
```

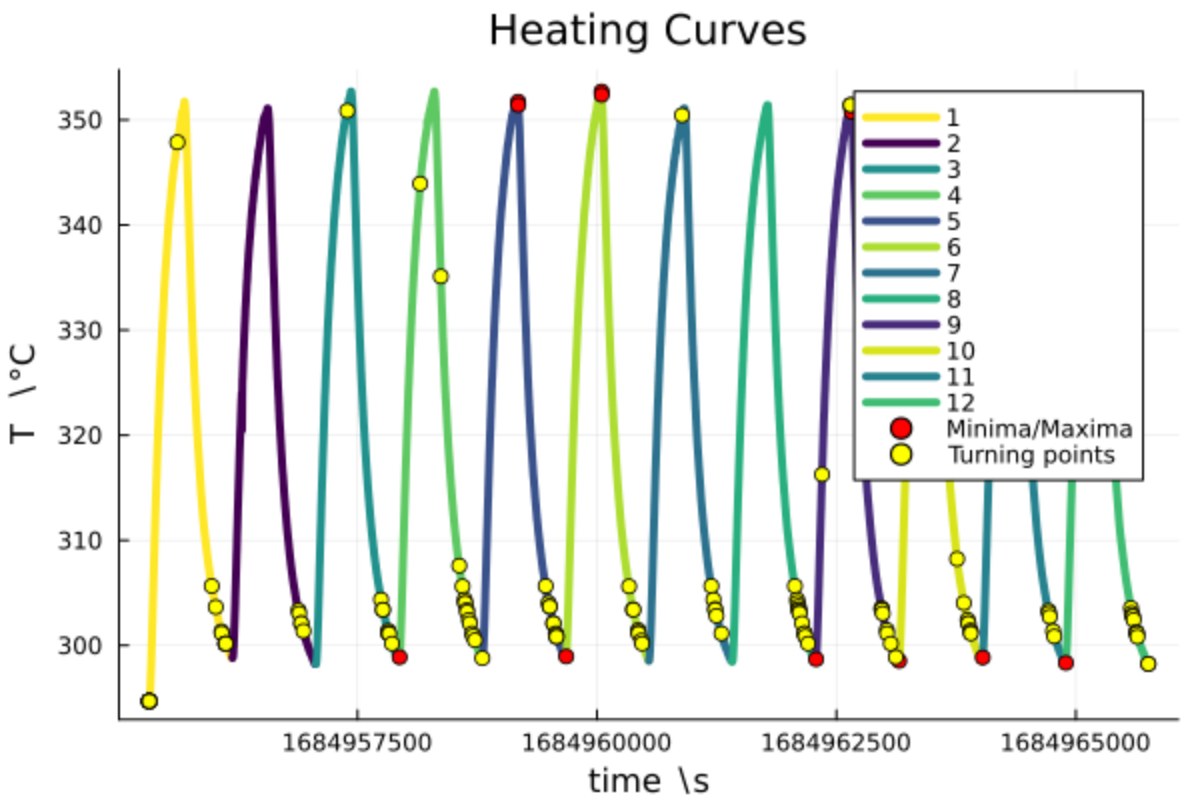
```
8640×9 Matrix{Float64}:
    1.0  294.693  22.349  100.0  1.68496e9   0.0  1.68496e9    0.0      294.693
    2.0  294.693  22.188  100.0  1.68496e9   0.0  1.68496e9    1.35462  294.693
    3.0  294.693  22.316  100.0  1.68496e9   0.0  1.68496e9    2.5741   294.693
    4.0  294.693  22.316  100.0  1.68496e9   0.0  1.68496e9    3.794    294.693
    5.0  294.693  22.316  100.0  1.68496e9   0.0  1.68496e9    5.01308  294.693
    6.0  294.693  22.413  100.0  1.68496e9   0.0  1.68496e9    6.23228  294.693
    7.0  295.015  22.413  100.0  1.68496e9   0.0  1.68496e9    7.45149  294.693
    ⋮                                       ⋮
 8635.0  298.238  26.377    0.0  1.68497e9  11.0  1.68496e9  862.372    298.238
 8636.0  298.238  26.345    0.0  1.68497e9  11.0  1.68496e9  863.571    298.238
 8637.0  298.238  26.377    0.0  1.68497e9  11.0  1.68496e9  864.771    298.238
 8638.0  298.238  26.377    0.0  1.68497e9  11.0  1.68496e9  865.971    298.238
 8639.0  298.238  26.377    0.0  1.68497e9  11.0  1.68496e9  867.17     298.238
 8640.0  298.238  26.377    0.0  1.68497e9  11.0  1.68496e9  868.37     298.238
```

```
begin
    heating_df_path = joinpath(@__DIR__, "measurements_heating_and_cooling.csv")
    heating_df = prepare_dataframe(heating_df_path)
    heating_curves  =  heating_df |> Matrix

end
```

Anzeige des Verlaufes der Heizkurven.

Die Messpunkte werden wurden alle 1.1 s bis 1.3 s genommen. Zeitpunkte sind als Timestamps abgespeichert, die Temperatur wirde gemessen in °C und Kelvin umgewandelt. Anzeige von Punkten mit erster Ableitung und zweiter Ableitung null des rollenden Mittelwertes

```
md"""
Anzeige des Verlaufes der Heizkurven.

Die Messpunkte werden wurden alle 1.1 s bis 1.3 s genommen. Zeitpunkte sind als Timestamps abgespeichert, die Temperatur wirde gemessen in °C und Kelvin umgewandelt.
Anzeige von Punkten mit erster Ableitung und zweiter Ableitung null des rollenden Mittelwertes
"""
```




```
begin
    p1 = @df heating_df plot(:time, [:T1 ], group = Int.(heating_df.run) .+ 1,
    legend = :topright,
        xlabel = "time \s", ylabel = "T  \°C",
        title = "Heating Curves", lw = 4, fmt = :png,
        palette=cgrad(:viridis, rev=true, categorical = true))


    dT₁ = heating_df.T1 |> (y -> runmean(y, 21)) |> diff |> (y -> prepend!([0.0, ],
    y))
    T1ₑₓₜᵣₑₘₑ = heating_df[:, :T1][dT₁ .== 0]
    tₑₓₜᵣₑₘₑ = heating_df[:, :time][dT₁ .== 0]
    rowsₑₓₜᵣₑₘₑ = heating_df[:, :Column1][dT₁ .== 0]
    scatter!(p1, tₑₓₜᵣₑₘₑ, T1ₑₓₜᵣₑₘₑ, label="Minima/Maxima", mc=:red);

    d2T₁ = dT₁ |> (y -> runmean(y, 21)) |> diff |> (y -> prepend!([0.0, ], y))
    T1ₜᵤᵣₙ  = heating_df[:, :T1][d2T₁ .== 0.0]
    tₜᵤᵣₙ = heating_df[:, :time][d2T₁ .== 0.0]
    rowsₜᵤᵣₙ = heating_df[:, :Column1][d2T₁ .== 0]
    scatter!(p1, tₜᵤᵣₙ, T1ₜᵤᵣₙ, label="Turning points", mc=:yellow);

end
```

Compute PINN

Erstellen des physikalischen Loss, Modell: FOPDT

Der physikalische Loss-Termin wird mittels FOPDT model erstellt, um die Faktoren τ ,K, θ zu bestimmen. Da nur τ in die Abkühlung eingeht, wird der Term erst für die Abkühlung berechnet und dann, der Faktor τ in die Heizungsphase eingesetzt.

Das ODE wird mit nlsove gelöst. Eingesetzt werden die Messwerte den ersten, 75ten und 130ten Messpunkt.

```
md"""

## Compute PINN
### Erstellen des physikalischen Loss, Modell: FOPDT

Der physikalische Loss-Termin wird mittels [FOPDT model](https://apmonitor.com/pdc/index.php/Main/FirstOrderOptimization) erstellt, um die Faktoren τ ,K, θ zu bestimmen.
Da nur τ in die Abkühlung eingeht, wird der Term erst für die Abkühlung berechnet und dann, der Faktor τ in die Heizungsphase eingesetzt.

Das ODE wird mit nlsove gelöst.
Eingesetzt werden die Messwerte den ersten, 75ten und 130ten Messpunkt.

"""
```

```
FOPDT_off (generic function with 1 method)
```

```
function FOPDT_off(z)
    τ = z[1]

    return [dT + celsius2kelvin(T) / τ for (dT, T) in zip([-0.2425,], [76.006,])]
end
```

```
FOPDT_on (generic function with 1 method)
```

```
begin
    function FOPDT_on(z)
        τ = 1439.8
        K, θ = z

        return [dT + (1 / τ) * (celsius2kelvin(T) + K * 100 *(θ -i)) for (dT, T, i) in zip(
            [0.279, 0.3065],
            [46.035,60.279],
            [75, 130])]

    end
end
```

```
Results of Nonlinear Solver Algorithm
 * Algorithm: Trust-region with dogleg and autoscaling
 * Starting Point: [1439.8]
 * Zero: [1439.8]
 * Inf-norm of residuals: 0.000003
 * Iterations: 0
 * Convergence: true
   * |x - x'| < 1.0e-07: false
   * |f(x)| < 1.0e-04: true
 * Function Calls (f): 1
 * Jacobian Calls (df/dx): 1
```

```
begin
    initial_x_off = [1439.8, ]
    sol_off = nlsolve(FOPDT_off, initial_x_off, xtol=1e-7, ftol=1e-4)
    sol_off
end
```

```
Results of Nonlinear Solver Algorithm
 * Algorithm: Trust-region with dogleg and autoscaling
 * Starting Point: [0.0097889, -666.1]
 * Zero: [0.009788818181809787, -661.4415445515547]
 * Inf-norm of residuals: 0.000000
 * Iterations: 1
 * Convergence: true
   * |x - x'| < 1.0e-04: false
   * |f(x)| < 1.0e-04: true
 * Function Calls (f): 2
 * Jacobian Calls (df/dx): 2
```

```
begin
    initial_x_on = [0.0097889, -666.1]
    sol_on = nlsolve(FOPDT_on, initial_x_on, xtol=1e-4, ftol=1e-4)
    sol_on
end
```

Compute PINN

```
md"""

### Compute PINN
"""
```

```
18×9 Matrix{Float64}:
    5.0  294.693  22.316  100.0  1.68496e9  0.0  1.68496e9    5.01308  294.693
  244.0  347.867  45.584  100.0  1.68496e9  0.0  1.68496e9  296.366    347.738
  543.0  305.65   32.822    0.0  1.68496e9  0.0  1.68496e9  655.65     305.65
  578.0  303.652  31.211    0.0  1.68496e9  0.0  1.68496e9  697.736    303.684
  628.0  301.332  29.277    0.0  1.68496e9  0.0  1.68496e9  757.818    301.396
  629.0  301.171  29.277    0.0  1.68496e9  0.0  1.68496e9  759.018    301.332
  661.0  300.171  28.085    0.0  1.68496e9  0.0  1.68496e9  797.543    300.171
    ⋮                                      ⋮
 1316.0  302.105  29.922    0.0  1.68496e9  1.0  1.68496e9  718.979    302.105
 1317.0  302.105  29.922    0.0  1.68496e9  1.0  1.68496e9  720.179    302.105
 1336.0  301.364  29.213    0.0  1.68496e9  1.0  1.68496e9  743.097    301.428
 1713.0  350.864  47.26   100.0  1.68496e9  2.0  1.68496e9  331.539    351.057
 2007.0  304.361  31.211    0.0  1.68496e9  2.0  1.68496e9  684.243    304.361
 2168.0  298.882  25.732  100.0  1.68496e9  3.0  1.68496e9    8.67026  298.625
```

```
begin
    rows_sample = vcat(rows_extreme[1:10], rows_turn[1:20]) |> sort |> unique
    rows_sample =rows_sample[1+4:end-4]

    is_at_sample_time = heating_df.Column1 .∈ Ref(rows_sample)
    sample_df = heating_df[is_at_sample_time, :]
    sample_df
    samples = sample_df |> Matrix
end
```

Das physikalische Model wird mittels FOPDT ertellt. Die bereits ermittelten Faktoren K, τ, θ werden hier eingesetzt. Die linke Seite des ODE wird durch einen delta zum vorherigen Messpunkt bestimmt. Da beide Seiten des ODE gleich sein sollten, wird diese Bedingung zum Minimierung (als Loss-Term) genutzt.

```
md"""
Das physikalische Model wird mittels FOPDT ertellt. Die bereits ermittelten
Faktoren K, τ, θ werden hier eingesetzt. Die linke Seite des ODE wird durch einen
delta zum vorherigen Messpunkt bestimmt. Da beide Seiten des ODE gleich sein
sollten, wird diese Bedingung zum Minimierung (als Loss-Term) genutzt.
"""
```

```
loss_total (generic function with 1 method)
```

```
begin
    NN_pinn = Chain(
        Dense(2 => 16, relu),
        Dense(16 => 1)
    ) |> f64

    function loss_physical(x)

        K = 0.0097889
        τ = 1439.8
        θ = -661.1

        # Q1 := index 4 in vector, T1prev := index 9
        ŷ = [x[4], x[9]] |> NN_pinn |> first
        prev_index = convert(Int, x[1]) - 1
        x_prev = heating_curves[prev_index, :]

        ŷ_prev = [x_prev[4], x_prev[9]] |> NN_pinn |> first
        Δy_l = ŷ - ŷ_prev
        U = x[4]
        i_θ = x[8]  # duration

        y_r = (.-ŷ .+ K .* U .*(i_θ .- θ)) ./ τ

        return sum((Δy_l .- y_r) .^2)
    end

    loss_physical() = eachrow(samples) .|> loss_physical |> sum

    # loss_ratio = 1 / size(sample_m)[1]
    loss_ratio = 0.5

    loss_total(ŷ, y) = (1 - loss_ratio) * Flux.mae(ŷ, y)+ loss_ratio *
    loss_physical()
end
```

Als Eingabe in das Netzwerk wird die aktuelle Q1 und die vorherige Temperatur gewählt.

Wie bestimmt man nur aus t oder Q mit in einem PINN die Temperatur, die ja eine Aggregation ist? Mit RNN?

Batchgröße ist 1 (keine Mini-Batches). (Bin froh, dass es lief.)

Wenn man die grads aus Flux.withgradient nutzt, mann man dass denn im physikalischen Loss nutzen, wenn dieses für das aktuelle Batch berechnet, aber nicht für andere Punkte?

```
md"""
Als Eingabe in das Netzwerk wird die aktuelle Q1 und die vorherige Temperatur gewählt.

Wie bestimmt man nur aus t oder Q mit in einem PINN die Temperatur, die ja eine Aggregation ist? Mit RNN?

Batchgröße ist 1 (keine Mini-Batches). (Bin froh, dass es lief.)

Wenn man die grads aus Flux.withgradient nutzt, mann man dass denn im physikalischen Loss nutzen, wenn dieses für das aktuelle Batch berechnet, aber nicht für andere Punkte?
"""
```

```
begin
    train_mask = 1: Int.(length(heating_df.T1)//2)

    train_df = heating_df[train_mask, :]

    x_train_df = train_df[:,[:Q1, :T1prev]]
    x_train_m = x_train_df|> Matrix
    x_train_m  =  [x for x in eachrow(x_train_m)]

    y_train = train_df[:, :T1]

    train_set = [(x, y) for (x, y) in zip(x_train_m, y_train)]

    η = 1e-5
    opt_pinn = Flux.setup(Adam(η), NN_pinn)
    epochs = 10

    losses_pinn = []

    for i in 1:epochs
        local_loss = []
        for (x, y) in train_set
            loss, grads = Flux.withgradient(NN_pinn) do m
                ŷ = m(x)
                loss_total(ŷ, y)
            end
            Flux.update!(opt_pinn, NN_pinn, grads[1])
            push!(local_loss, loss)
        end
        push!(losses_pinn, Flux.mean(local_loss))
    end
end
```

hier ein weiterer Lauf auf einem ähnlichen Netztwerk. Allerdings nur mit mae als Lossfunktion.

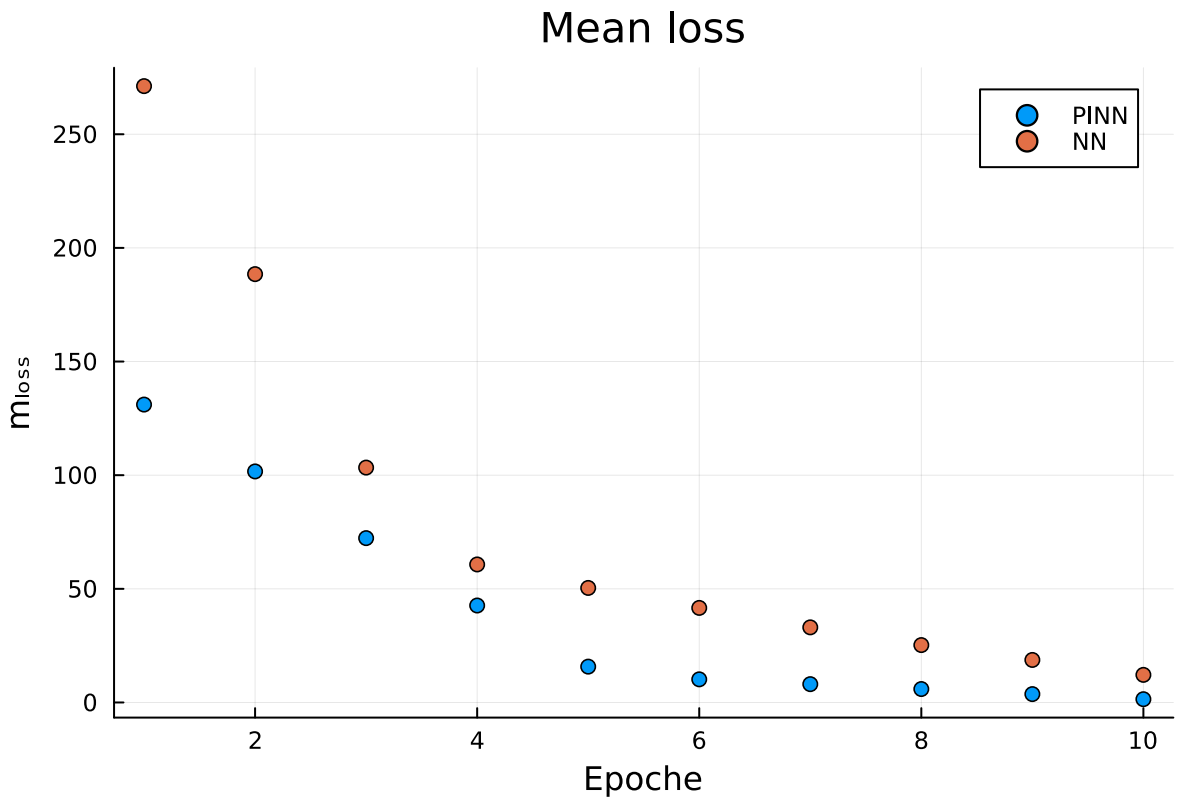
```
md"""
hier ein weiterer Lauf auf einem ähnlichen Netztwerk. Allerdings nur mit mae als
Lossfunktion.
"""
```

```
begin
    NN_dense = Chain(
        Dense(2 => 16, relu),
        Dense(16 => 1)
    ) |> f64

    opt_nn = Flux.setup(Adam(η), NN_dense)
    losses_nn = []

    for i in 1:epochs
        local_loss = []
        for (x, y) in train_set
            loss, grads = Flux.withgradient(NN_dense) do m
                ŷ = m(x)
                Flux.mae(ŷ, y)
            end
            Flux.update!(opt_nn, NN_dense, grads[1])
            push!(local_loss, loss)
        end
        push!(losses_nn, Flux.mean(local_loss))
    end
end
```

Auftragung des über alle Datenpunkte gemittelten Losses.

Oft PINN nicht besser als NN, daher mit Absicht Gewichtung 0.5. 1 Trainungspunkt in der mae, ist dabei gleich gewichtet der 12 Punkten der physikalischen Lossfunktion.

```
md"""
Auftragung des über alle Datenpunkte gemittelten Losses.

Oft PINN nicht besser als NN, daher mit Absicht Gewichtung 0.5.
1 Trainungspunkt in der mae, ist dabei gleich gewichtet der 12 Punkten der
physikalischen Lossfunktion.
"""
```




```
scatter([losses_pinn losses_nn], title = "Mean loss", lw = 4,
        xlabel = "Epoche", ylabel = "mloss", label=["PINN" "NN"],)
```

```
[297.055, 297.055, 297.055, 297.055, 297.055, 297.055, 297.055, 297.376, 297.376, 297.698
```

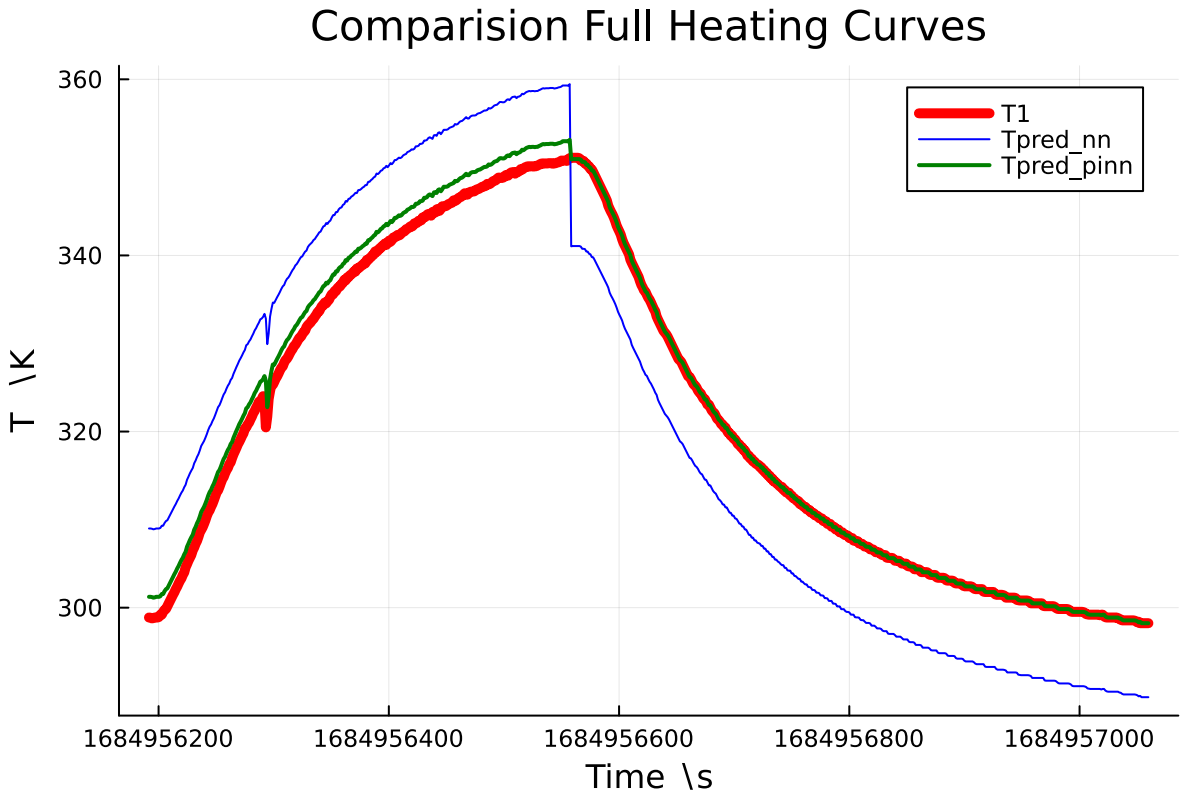
```
begin
    x_full_df = heating_df[:,[:Q1, :T1prev]]
    x_full_m = x_full_df |> Matrix # |> eachrow
    x_full_m  =  [x for x in eachrow(x_full_m)]

    heating_df.Tpred_nn = [NN_dense(x) for x in x_full_m] .|> first
    heating_df.Tpred_pinn = [NN_pinn(x) for x in x_full_m] .|> first
end
```

Darstellung der Vorhersagen einmal des PINN undes einfachen Multilayerperzeptron.

Ab und an (nicht immer), erscheint beim Umschalten zwischen Heizen und Kühlen (und vice versa) ein sehr hoher Offset. Ensteht das durch

- die Q1 Eingabe
- Fehler im phyikalischen Loss (Nein, dass Phenomen sieh man auch im MLP).
- Lernrate
- keine Batches
- niedrige Anzahl von Trainings?

```
md"""
Darstellung der Vorhersagen einmal des PINN undes einfachen Multilayerperzeptron.

Ab und an (nicht immer), erscheint beim Umschalten zwischen Heizen und Kühlen (und vice versa) ein sehr hoher Offset.
Ensteht das durch
- die Q1 Eingabe
- Fehler im phyikalischen Loss (Nein, dass Phenomen sieh man auch im MLP).
- Lernrate
- keine Batches
- niedrige Anzahl von Trainings?
"""
```




```
begin
    first_run_df = heating_df[(heating_df.run .== 1)  , : ]
    # first_run_df = first_run_df[300:310, :]

    @df first_run_df plot(:time, [:T1 :Tpred_nn :Tpred_pinn], colour = [:red :blue
    :green],
    legend = :topright,
            xlabel = "Time \s", ylabel = "T  \K",
            title = "Comparision Full Heating Curves", lw = lw = [5 1 2])
end
```