ASSOCC: Model Experimentation Tutorial

Created by Maarten Jensen (Umeå University) & Kurt Kreulen (TU Delft)

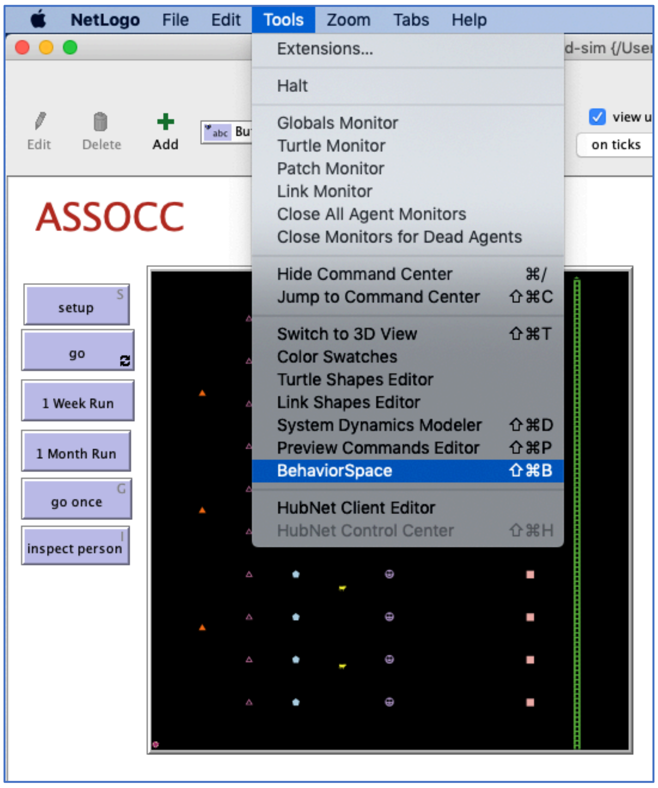
[latest update: 29-04-2020]

This document is meant to help those involved in building ASSOCC scenarios with setting up their experiments and analyzing + visualizing their results. This document is composed of three sections:

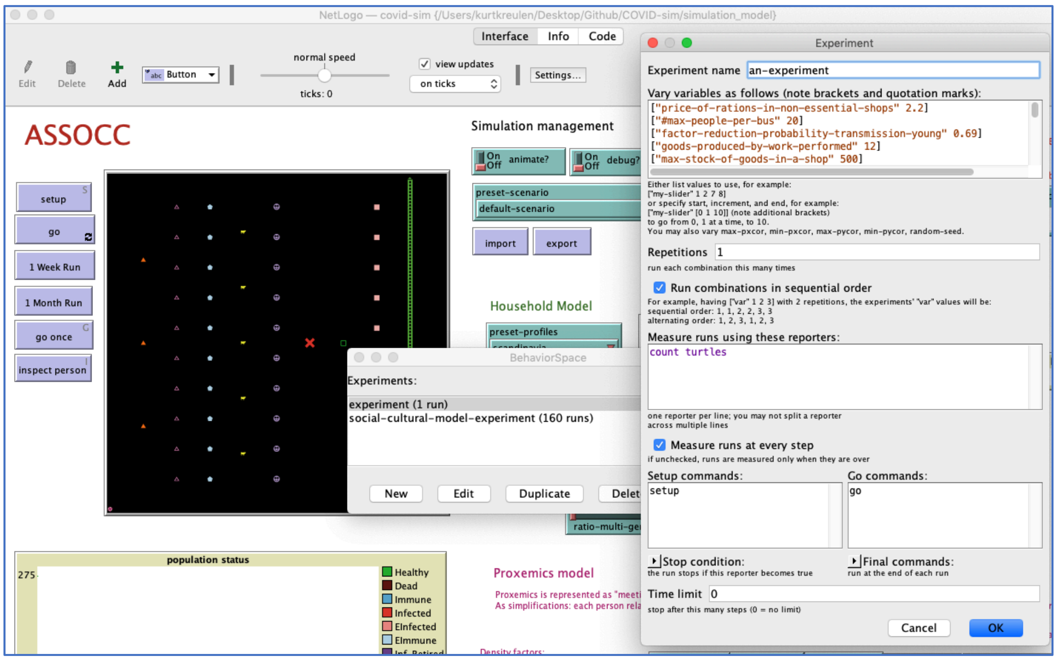
1. **Running Scenarios with Netlogo's Behaviorspace**
   - *This section should tell you how to set up and run your experiment in Behaviorspace.*
2. **Analyzing & Visualizing Model Output in R Studio**
   - *This section will tell you how to load the data into R studio and prepare it for analysis and visualization.*
3. **Building Plots with ggplot**
   - *This section provides a simple illustration of how ggplot can be used to build fancy looking plots.*
   - *At the end of section 3 there are two examples that show some plots created by chunks of ggplot code.*

Section 1: Running Scenarios with Netlogo's Behaviorspace

1. Download the latest version of the ASSOCC model from https://github.com/lvanhee/COVID-sim
2. Open the 'covid-sim.nls' file → click on 'Tools' → select BehaviorSpace



3. Click on 'New' to create a new experiment or click on one of the existing experiments to repeat a previously created experiment



Experiment name: an-experiment

Vary variables as follows (note brackets and quotation marks):

```
["price-of-rations-in-non-essential-shops" 2.2]
["#max-people-per-bus" 20]
["factor-reduction-probability-transmission-young" 0.69]
["goods-produced-by-work-performed" 12]
["max-stock-of-goods-in-a-shop" 500]
```

Either list values to use, for example:
["my-slider" 1 2 7 8]
or specify start, increment, and end, for example:
["my-slider" [0 1 10]] (note additional brackets)
to go from 0, 1 at a time, to 10.
You may also vary max-pxcor, min-pxcor, max-pycor, min-pycor, random-seed.

Repetitions: 1
run each combination this many times

✓ Run combinations in sequential order
For example, having ["var" 1 2 3] with 2 repetitions, the experiments' "var" values will be:
sequential order: 1, 1, 2, 2, 3, 3
alternating order: 1, 2, 3, 1, 2, 3

Measure runs using these reporters:

```
count turtles
```

one reporter per line; you may not split a reporter across multiple lines

✓ Measure runs at every step
if unchecked, runs are measured only when they are over

Setup commands: `setup`
Go commands: `go`

Stop condition:
the run stops if this reporter becomes true

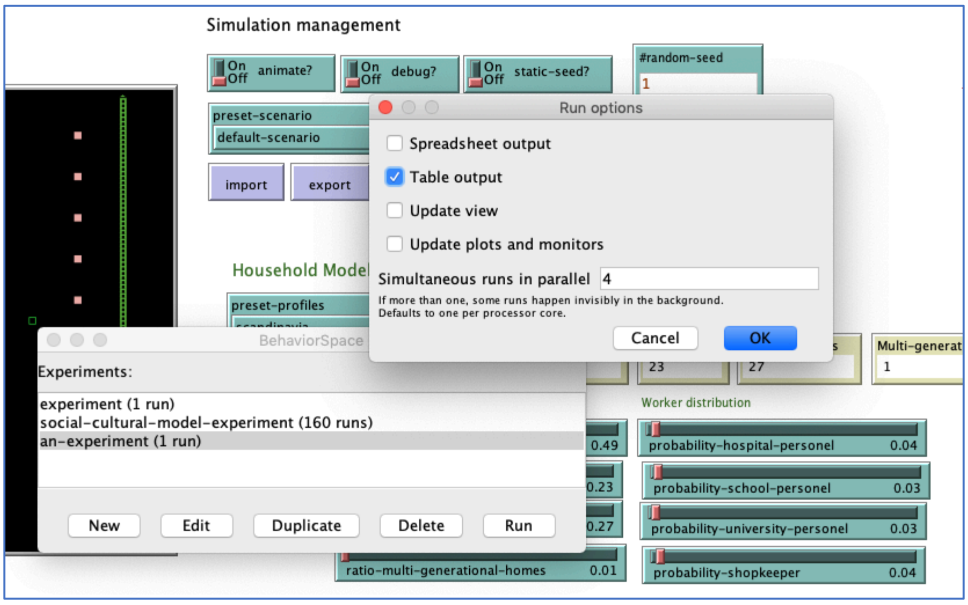
Final commands:
run at the end of each run

Time limit: 0
stop after this many steps (0 = no limit)

4. Specify the design of the experiment:
   a. Give the experiment a name
   b. Specify which variables are to stay **fixed** (*control* variables) and which variables must **vary** (*independent* variables) across simulation runs
   c. Specify the number of repetitions (i.e. replicate simulation runs); since ASSOCC contains stochastic elements it is important to run a certain number of repeated simulations to get a picture of inter-run variability due to the stochasticity present within the design of the model.
   d. Specify which variables you want to 'measure' (i.e. the *dependent* variables)
   e. You can leave setup commands and go commands as default.
   f. Specify the time limit (i.e. the number of ticks each simulation run should go for)
5. Press 'Run' to run the experiment, be sure to pick **'Table output'** before pressing OK (!)
6. Save the csv table in a convenient place (i.e. it is highly recommended to save it in the R file work-directory)
   a. Alternatively you can close Netlogo and run the experiment in 'headless' mode (which is to say: without the Netlogo interface open and consequently using up more working memory than necessary). Running simulations in headless mode is faster and enables parallelization of simulation runs if needed and if you have access to a network of distributed computers ;-) … In case you would like to run the simulation headless then simply execute your version of this script in the command line:

```
# NOTE the usage of "Netlog-6.1.1" instead of "Netlogo 6.1.1"

/Applications/Netlogo-6.1.1/netlogo-headless.sh \
  --model /Applications/Netlogo-6.1.1/covid-sim.nlogo \
  --experiment EXPERIMENT_NAME \
  --table ~/Desktop/"EXPERIMENT_NAME.csv"
```



#random-seed 1

preset-scenario: default-scenario

preset-profiles

Run options

- [ ] Spreadsheet output
- [✓] Table output
- [ ] Update view
- [ ] Update plots and monitors

Simultaneous runs in parallel 4

If more than one, some runs happen invisibly in the background.
Defaults to one per processor core.

BehaviorSpace

Experiments:

experiment (1 run)
social-cultural-model-experiment (160 runs)
an-experiment (1 run)

Multi-generat 1

probability-hospital-personel 0.04
probability-school-personel 0.03
probability-university-personel 0.03
probability-shopkeeper 0.04
ratio-multi-generational-homes 0.01

7. Once the simulation is done you will find a csv file within the directory you specified and you can begin the analysis in R Studio.

Section 2: Analyzing & Visualizing Model Output in R Studio

NOTE: if you have not yet installed R Studio downloaded, then do so via: https://rstudio.com/products/rstudio/download/

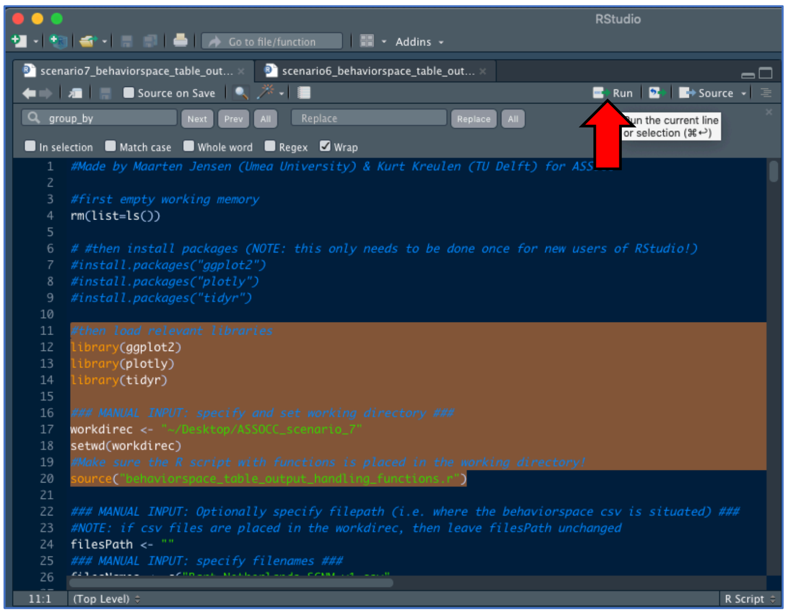
If this is the first time you are using this tutorial then chances are you will need to install some packages in R that enable you to manipulate data and make fancy plots. The packages used in our script are: "ggplot2", "tidyverse" and "plotly".

You can install them by calling the *install.packages("NAME_OF_PACKAGE")* command.

1. Go to the ASSOCC google drive and search for:
   a. *main_behaviorspace_table_output_handling.R*
   b. *functions_behaviorspace_table_output_handling.R*
      i. If the R scripts cannot be found because it has been renamed or updated then contact Maarten or Kurt.
2. The script you will be working with is called: *main_behaviorspace_table_output_handling.R*
   a. Make sure the *behaviorspace_table_output_handling_functions.R* script is located in the same folder as the main script!
3. Open the R script and note the following:
   a. In R, comments are preceded by a #
   b. The comments in the R script are important and useful, they tell you what to do and/or what the code you will be running is supposed to do … if errors occur please contact Maarten or Kurt.

c. If you search (ctrl+F) for 'MANUAL INPUT' you will find bits of code that need customization to fit your situation (i.e. such as changing file or variable names)

d. You run chunks of code by:
   i. Selecting them and pressing 'Run' in the top-right corner of the scripting window. There are shortcuts for this but if you're a complete beginner then this is the way to go ;-)



```
#Made by Maarten Jensen (Umea University) & Kurt Kreulen (TU Delft) for ASS

#first empty working memory
rm(list=ls())

# #then install packages (NOTE: this only needs to be done once for new users of RStudio!)
#install.packages("ggplot2")
#install.packages("plotly")
#install.packages("tidyr")

#then load relevant libraries
library(ggplot2)
library(plotly)
library(tidyr)

### MANUAL INPUT: specify and set working directory ###
workdirec <- "~/Desktop/ASSOCC_scenario_7"
setwd(workdirec)
#Make sure the R script with functions is placed in the working directory!
source("behaviorspace_table_output_handling_functions.r")

### MANUAL INPUT: Optionally specify filepath (i.e. where the behaviorspace csv is situated) ###
#NOTE: if csv files are placed in the workdirec, then leave filesPath unchanged
filesPath <- ""
### MANUAL INPUT: specify filenames ###
```

4. Running the entire script (after having thoroughly checked whether all file and variable names are specified correctly) should result in two dataframes:
   a. **clean_df** → this is a 'wide' format dataframe
      i. wide dataframes can be used to make plots of single arrays of data (i.e. 1x Y-variable)
   b. **df_long** → this is a 'long' format dataframe
      i. long dataframes can be used to make plots including multiple arrays of data (multiple Y-variables)
   c. you can use clean_df and df_long for feeding into *ggplot* or *plotly* in order to build plots. There exist ample tutorials for building graphics with ggplot and/or plotly so please go check those out. Otherwise you can always ask Maarten or Kurt for help when you're stuck with trying to build something pretty ;-) Good luck!

Section 3: Building Plots with ggplot

To put it super simply, plotting with ggplot works by first calling the ggplot function and then *adding layers* to your plot; making it more fancy layer by layer. Here is a simple example:

```
[L1] ggplot(dataframe, aes(x = varName, y = varName) +
[L2] geom_point(size=1, alpha=0.5, aes(group = varName, color = varName) +
[L2] geom_line(size=1, alpha=0.5, aes(group = varName, color = varName) +
[L3] geom_smooth(aes(color = varName, linetype = varName), method = "loess",
size = 1.5, span = 0.1, se = TRUE, fullrange = FALSE, level = 0.95) +
[L4]  scale_colour_manual(values=c("red1","gray10")) +
[L5]  xlim(0, 500) +
[L6]  ylim(0, 1000) +
[L7]  guides(colour = guide_legend(override.aes = list(size=5, alpha=1))) +
[L8]  xlab("Ticks [4 ticks = 1 day]") +
      ylab("y-label") +
      labs(title="title",
      subtitle="subtitle",
      caption="Agent-based Social Simulation of Corona Crisis (ASSOCC)") +
[L9]  theme_bw()
```

Layer 1 (L1) = call the ggplot function and define the main inputs (i.e. the data, the x-variable and the y-variable).

Layer 2 (L2) = if you want a scatterplot then use the *geom_point* function, if you want a line graph then use the *geom_line* function (if you want both then keep both activated). The 'size' defines the size of the graphics, the 'alpha' defines the transparency of the graphics, the *aes()* mapping defines the underlying data that determine the aesthetics of the graphics (generally you would want to define the number of run as the 'group variable' and a z-variable as 'color variable').

Layer 3 (L3) = the geom_smooth function adds a trendline to the plotted data. In the method you can specify whether to fit a linear regression to the data ("lm") or a local regression ("loess"). You can also fit a (x^{th} order) polynomial by adding "function = y ~ poly(x, 2)". For more detailed info on the possibilities: https://ropensci.github.io/plotly/ggplot2/geom_smooth.html

Layer 4 (L4) = specify the colors of your graph (see e.g. http://sape.inf.usi.ch/quick-reference/ggplot2/colour).

Layer 5 & 6 (L5 & L6) = specify the range of the x and y axes.

Layer 7 (L7) = specify the aesthetic design of the legend of the plot.

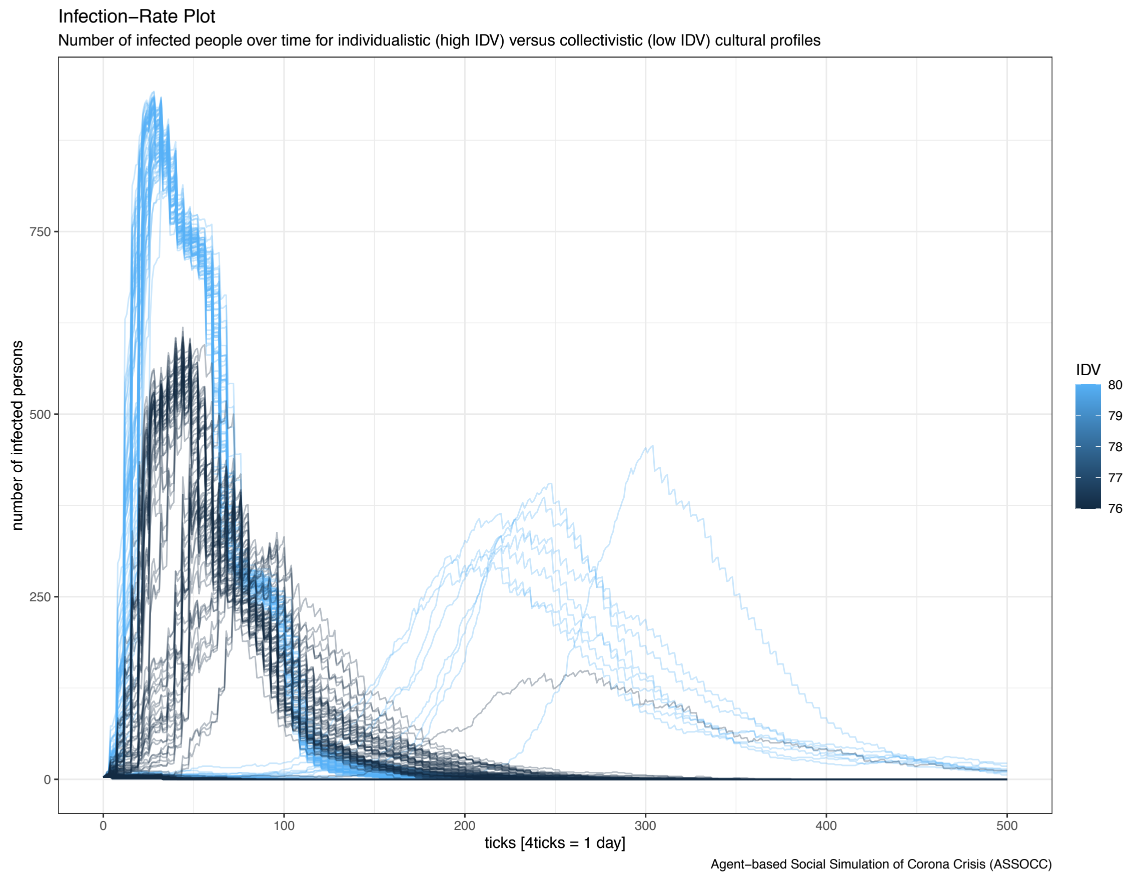
Layer 8 (L8) = specify the titles & labels of the plot.

Layer 9 (L9) = alter the general theme of the plot (I personally like theme_bw and theme_linedraw).

EXAMPLE 1: Line Plot

Running the following piece of code should result in the plot below:

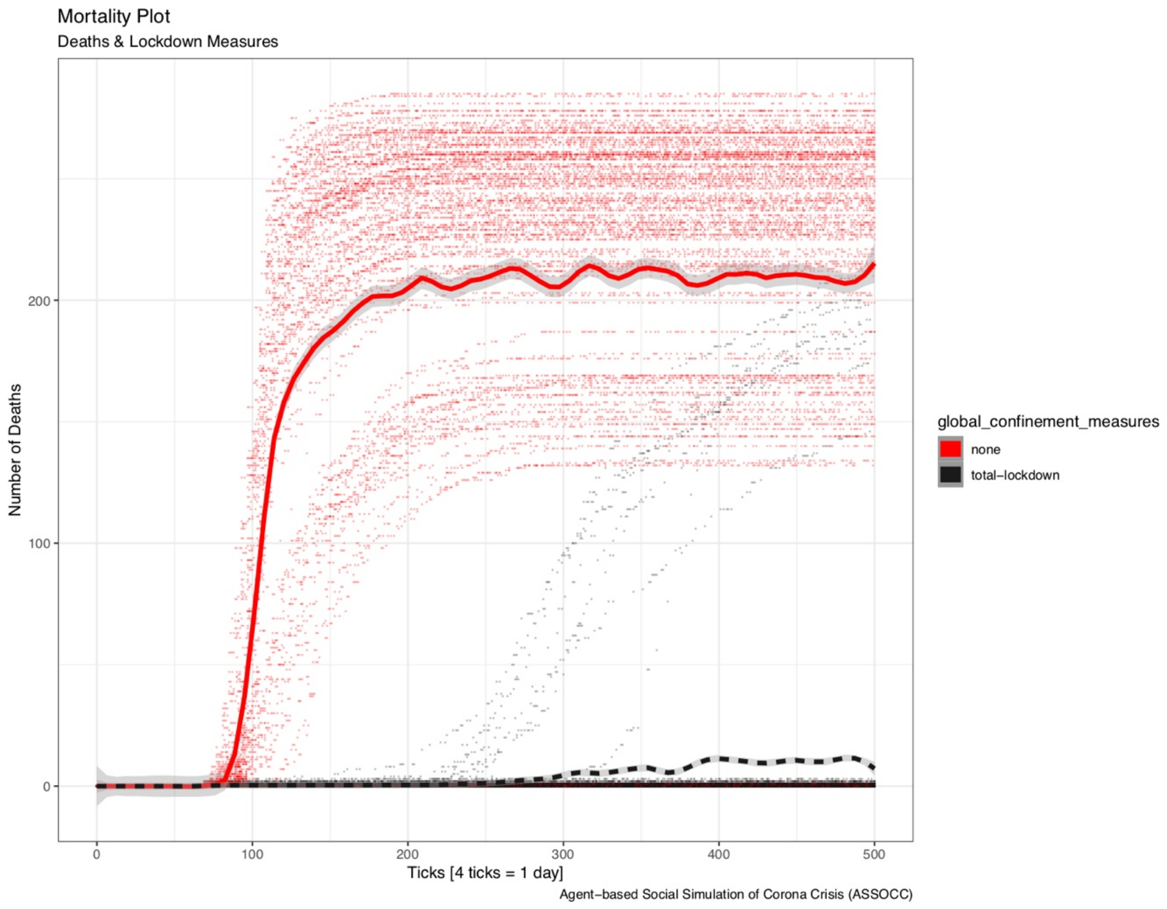
```
ggplot(clean_df, aes(x = tick, y = infected_people, group = run_number)) +
      geom_line(size=0.5,alpha=0.3,aes(color=IDV)) +
      xlab("ticks [4ticks = 1 day]") +
      ylab("number of infected persons") +
      labs(title="Infection-Rate Plot", subtitle="Number of infected people over
      time for individualistic (high IDV) versus collectivistic (low IDV) cultural
      profiles", caption="Agent-based Social Simulation of Corona Crisis (ASSOCC)")+
      theme_bw()
```



EXAMPLE 2: Scatter Plot with Trendline Fitted to the Data

Running the following piece of code should result in the plot below. Note the use of '`dplyr::sample_frac(clean_df,0.33)`' in the specification of which dataframe to input to the ggplot function. The *dplyr* package contains a function (`sample_frac`) that enables you to randomly sample a fraction from the data that you want to plot. This function can be useful if you have to plot a HUGE amount of data. The `sample_frac` function can help you speed up the creation of plots in ggplot significantly. In this example, a random sample is drawn from the data that comprises 33% of the original size.

```
ggplot(dplyr::sample_frac(clean_df, 0.33), aes(x = tick, y = dead_people)) +
       geom_point(size=0.1,alpha=0.2,aes(group = run_number, color =
       global_confinement_measures)) +
       geom_smooth(aes(color = global_confinement_measures, linetype =
       global_confinement_measures), method = "loess", size = 1.5, span = 0.1, se =
       TRUE, fullrange = FALSE, level = 0.95) +
       scale_colour_manual(values = c("red1","gray10")) +
       guides(colour = guide_legend(override.aes = list(size = 5, alpha = 1))) +
       xlab("Ticks [4 ticks = 1 day]") +
       ylab("Number of Deaths") +
       labs(title = "Mortality Plot",
        subtitle = "Deaths & Network Generation Method",
        caption = "Agent-based Social Simulation of Corona Crisis (ASSOCC)") +
       theme_bw()
```


Mortality Plot
Deaths & Lockdown Measures
Number of Deaths
Ticks [4 ticks = 1 day]
global_confinement_measures
🟥 none
⬛ total−lockdown
Agent−based Social Simulation of Corona Crisis (ASSOCC)