System Dynamics Modelling with R
An *Interactive Workshop*

Prof. Jim Duggan,
University of Galway

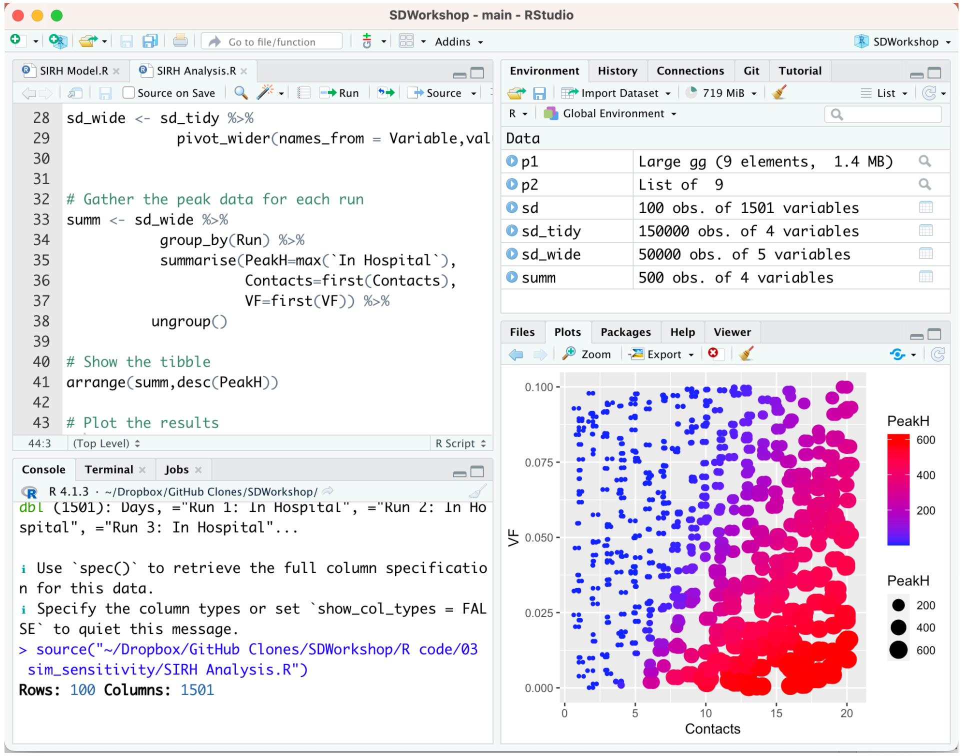
https://github.com/JimDuggan/SDWorshop

Overall Goals – Focus on breadth!

1. Understand data frames and plot simulation data using **ggplot2**
2. Analyse sensitivity runs with **dplyr**
3. Build a model using **deSolve**
4. Run sensitivity sweep with **purrr, deSolve** and **dplyr**
5. Create interactive simulation app using **Shiny**

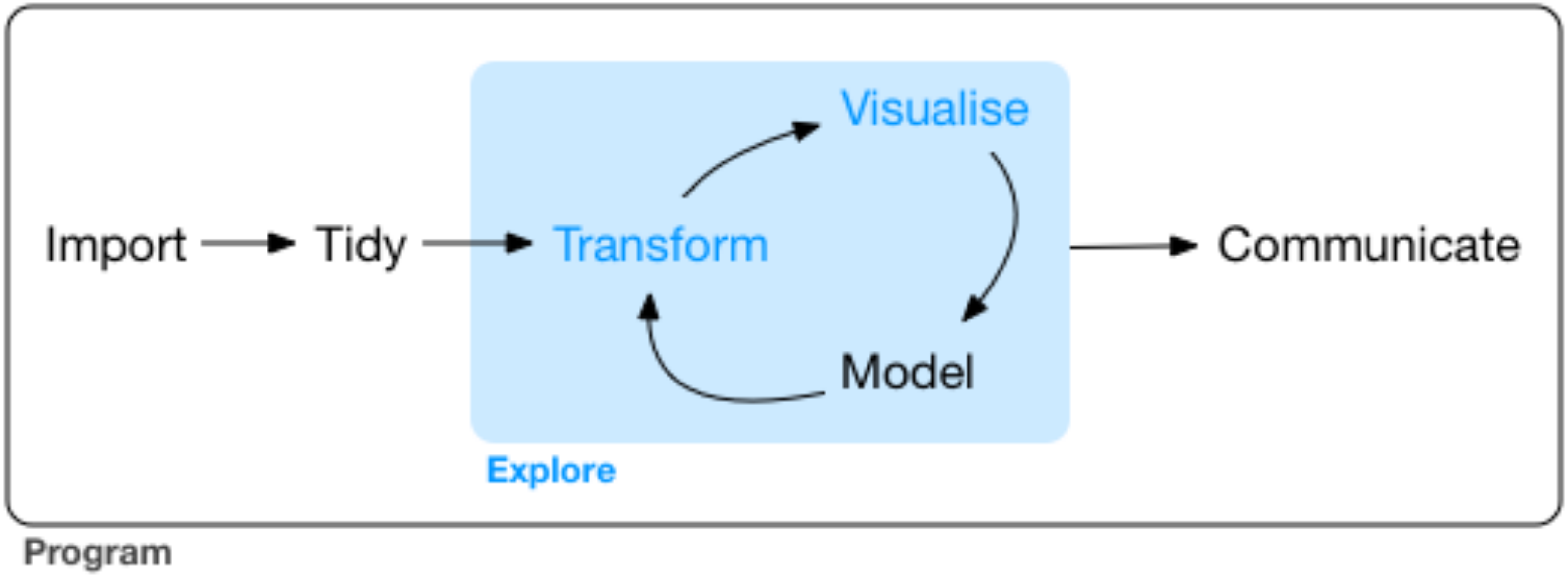
R

- R's *mission* is to enable the best and most thorough exploration of data possible (Chambers 2008).
- It is a dialect of the S language, developed at Bell Laboratories
- ACM noted that *S "will forever alter the way people analyze, visualize, and manipulate data"*

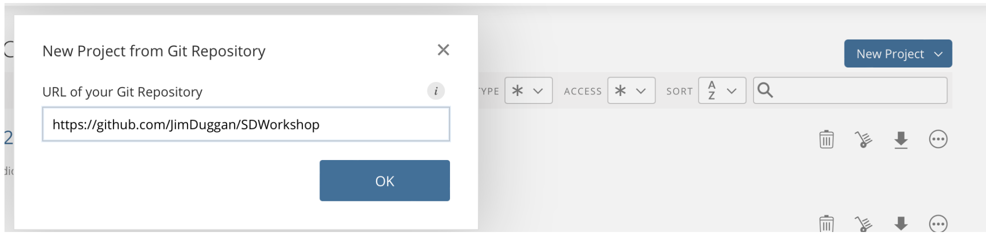


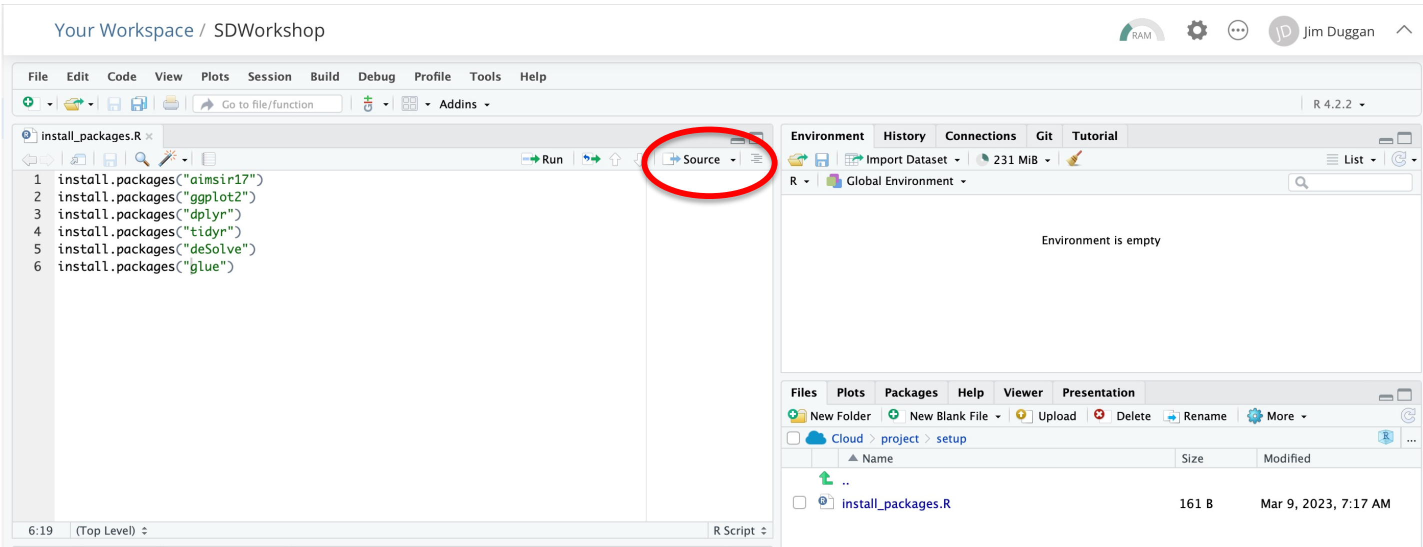
Data Science Process and Workflow

“Data exploration is the art of looking at your data, rapidly generating hypotheses, quickly testing them, then repeating again and again and again.” (Wickham and Grolemund 2017).



Access to workshop resources and setup





1. ggplot2 → 2. dplyr → 3. deSolve → 4. purrr → 5. Shiny

- Introducing the data frame (rectangular data)
- Plot graphs
- Import a simulation run from a CSV file (e.g. Stella/Vensim output)
- Plot simulation data

Data Frames/Tibbles – mpg

- The most common way of storing data in R
- A two-dimensional structure, with rows (observations) and columns (variables)

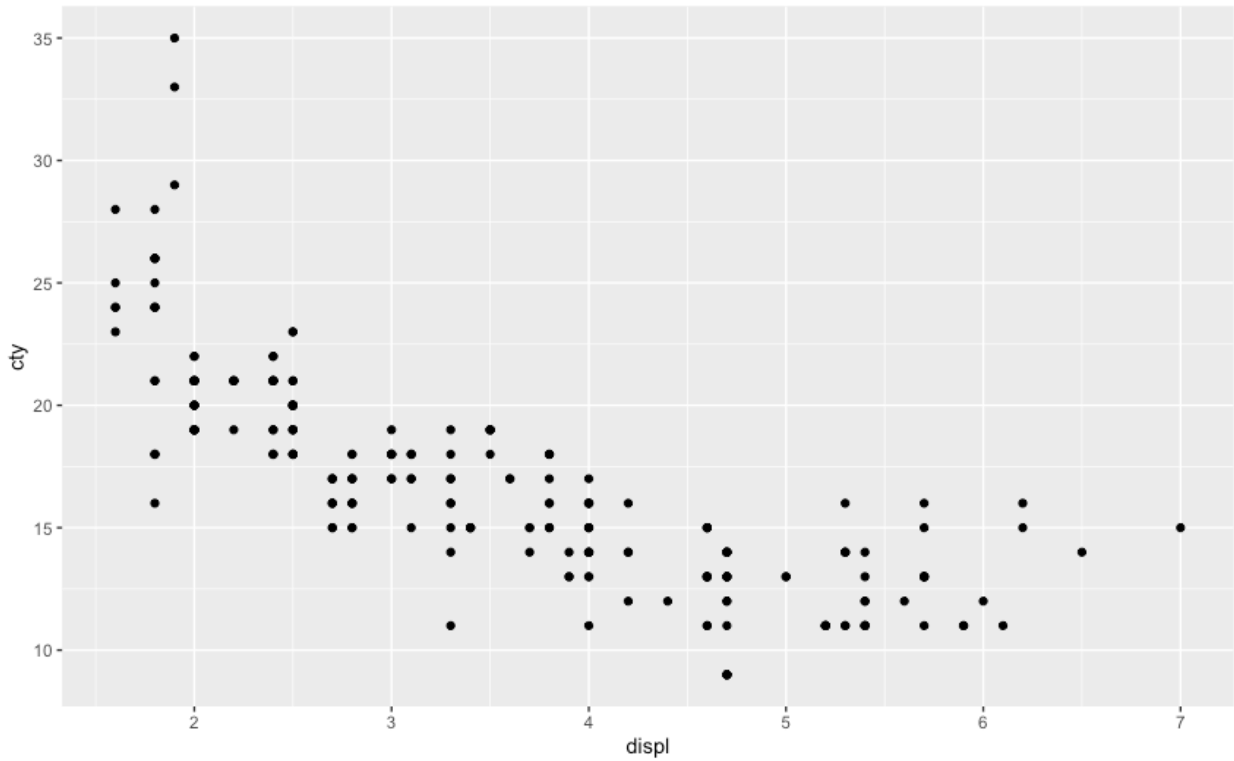
```
> mpg
# A tibble: 234 × 11
   manufacturer model      displ  year   cyl trans      drv     cty   hwy fl    class
   <chr>        <chr>      <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
 1 audi         a4           1.8  1999     4 auto(l5)   f        18    29 p     compact
 2 audi         a4           1.8  1999     4 manual(m5) f        21    29 p     compact
 3 audi         a4           2    2008     4 manual(m6) f        20    31 p     compact
 4 audi         a4           2    2008     4 auto(av)   f        21    30 p     compact
 5 audi         a4           2.8  1999     6 auto(l5)   f        16    26 p     compact
 6 audi         a4           2.8  1999     6 manual(m5) f        18    26 p     compact
 7 audi         a4           3.1  2008     6 auto(av)   f        18    27 p     compact
 8 audi         a4 quattro   1.8  1999     4 manual(m5) 4        18    26 p     compact
 9 audi         a4 quattro   1.8  1999     4 auto(l5)   4        16    25 p     compact
10 audi         a4 quattro   2    2008     4 manual(m6) 4        20    28 p     compact
# … with 224 more rows
# ℹ Use `print(n = ...)` to see more rows
```

Sample Code (engines.R)

```r
library(ggplot2)

summary(mpg)

ggplot(mpg,aes(x=displ,y=cty))+geom_point()
```

```
> summary(mpg)
 manufacturer          model               displ            year           cyl            trans          
 Length:234         Length:234         Min.   :1.600   Min.   :1999   Min.   :4.000   Length:234        
 Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999   1st Qu.:4.000   Class :character  
 Mode  :character   Mode  :character   Median :3.300   Median :2004   Median :6.000   Mode  :character  
                                       Mean   :3.472   Mean   :2004   Mean   :5.889                     
                                       3rd Qu.:4.600   3rd Qu.:2008   3rd Qu.:8.000                     
                                       Max.   :7.000   Max.   :2008   Max.   :8.000                     
     drv                 cty             hwy             fl               class          
 Length:234         Min.   : 9.00   Min.   :12.00   Length:234         Length:234        
 Class :character   1st Qu.:14.00   1st Qu.:18.00   Class :character   Class :character  
 Mode  :character   Median :17.00   Median :24.00   Mode  :character   Mode  :character  
                    Mean   :16.86   Mean   :23.44                                        
                    3rd Qu.:19.00   3rd Qu.:27.00                                        
                    Max.   :35.00   Max.   :44.00                                        
```

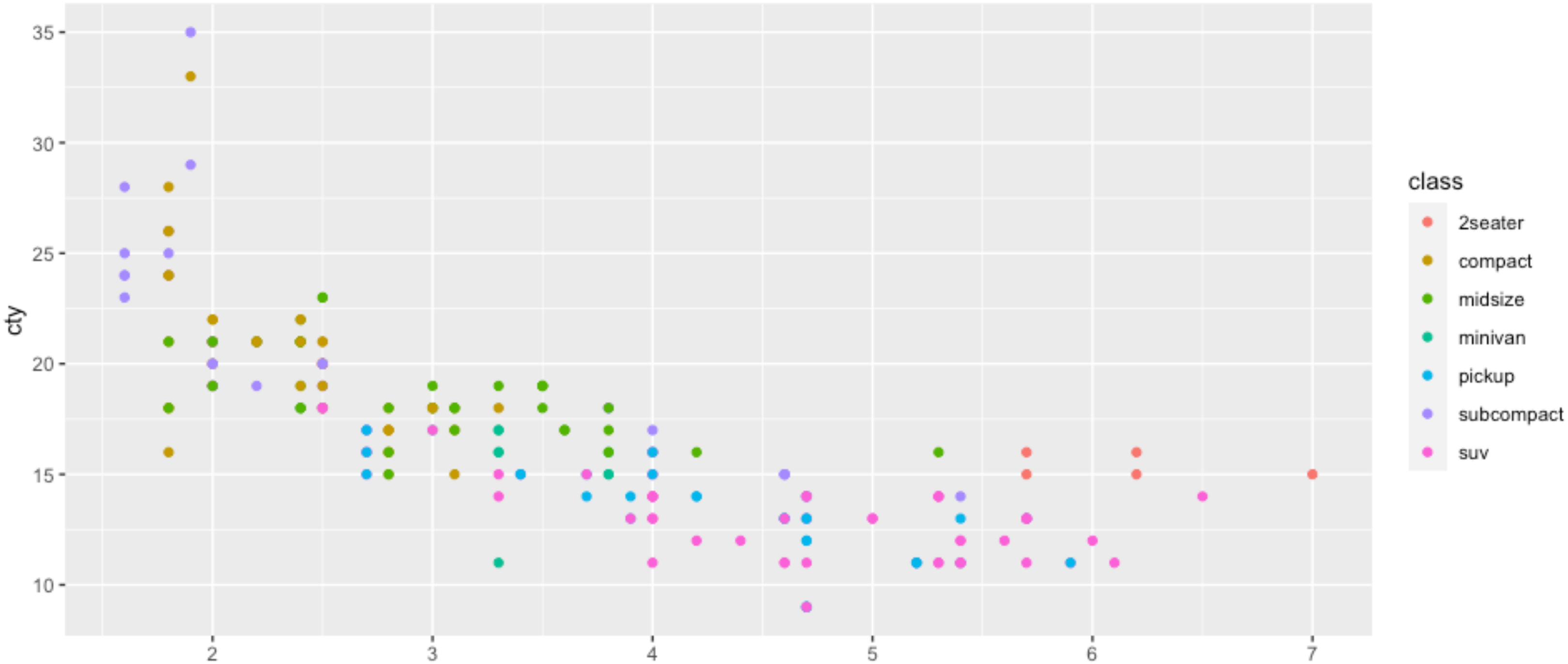
"The simple graph has brought more information to the data analyst's mind that any other device." – John Tukey




```
ggplot(mpg,aes(x=displ,y=cty))+geom_point()
```

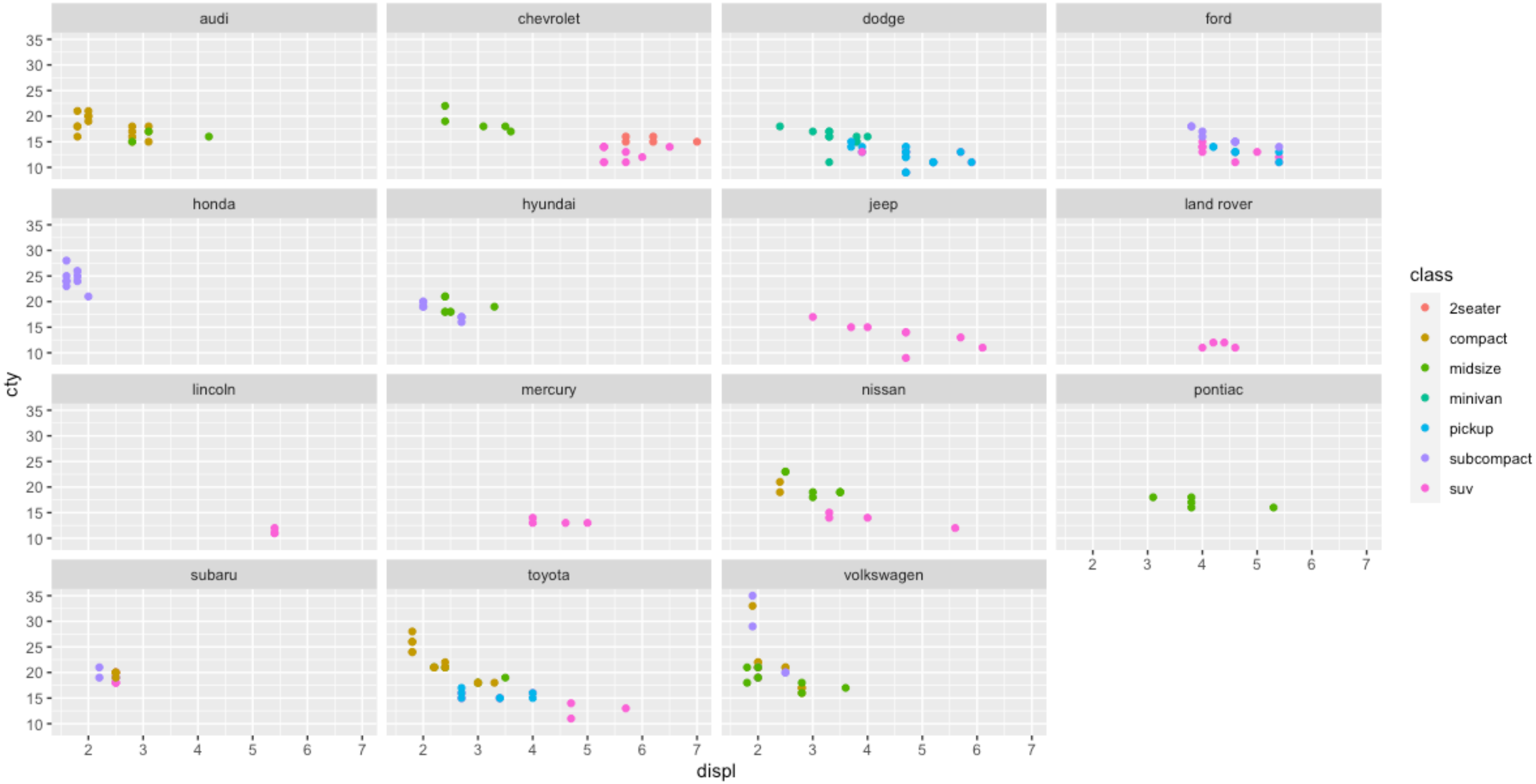
“The greatest value of a picture is when it forces us to notice what we never expected to see” – John Tukey

A third variable can be added to a 2-D plot by mapping it to an aesthetic: colour, size or shape

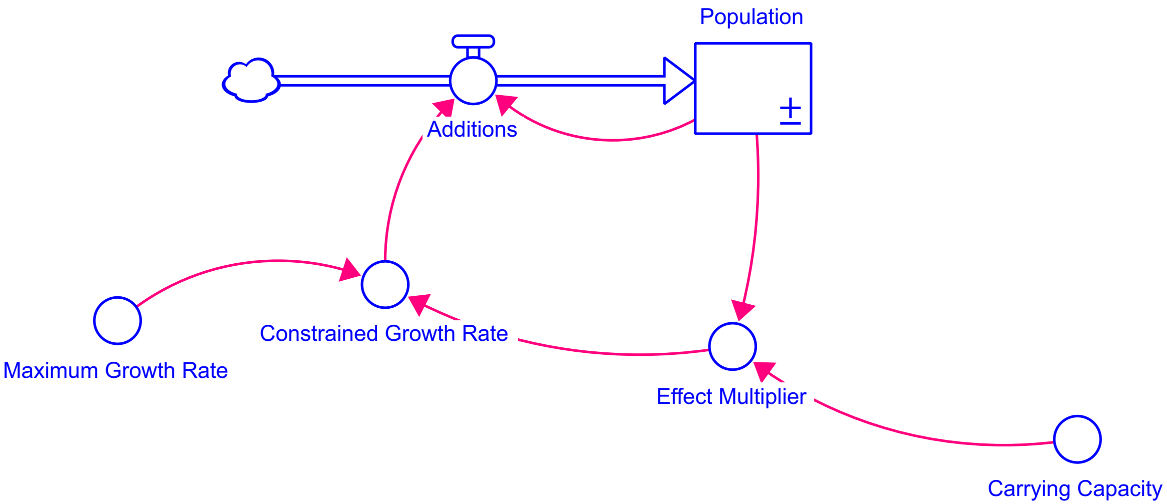


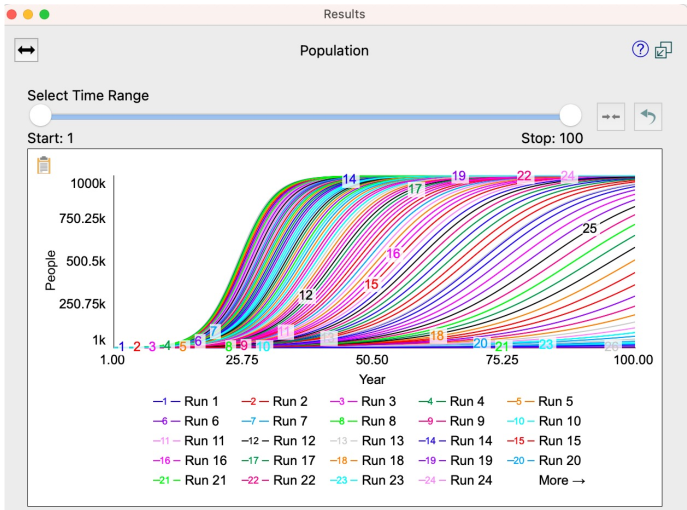

```
ggplot(mpg,aes(x=displ,y=cty,colour=class))+geom_point()
```

```r
ggplot(mpg,aes(x=displ,y=cty,colour=class))+geom_point()+facet_wrap(~manufacturer)
```



Simulation Output – CSV File

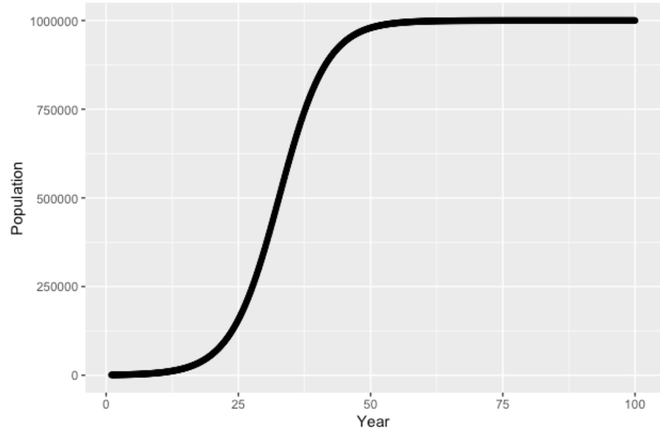




A	B	C	D	E	F	G
Year	Additions	Carrying Capacity	Constrained Growth Rate	Effect Multiplier	Maximum Growth Rate	Population
1	220.2795	1000000	0.2202795	0.999	0.2205	1000
1.125	226.3386436	1000000	0.220273429	0.998972465	0.2205	1027.534938
1.25	232.5641055	1000000	0.22026719	0.998944173	0.2205	1055.827268
1.375	238.9604315	1000000	0.22026078	0.998915102	0.2205	1084.897781
1.5	245.5322907	1000000	0.220254194	0.998885232	0.2205	1114.767835
1.625	252.2844784	1000000	0.220247426	0.998854541	0.2205	1145.459371

```r
library(ggplot2)
library(readr)
library(dplyr)

d <- read_csv("datasets/single runs/LTG.csv")

ggplot(d,aes(x=Year,y=Population)) +
  geom_point()+geom_line()

d <- d %>%
  mutate(ModeValue=round(c(NA,NA,diff(abs(diff(Population)))),3),
         Mode=as.factor(ifelse(ModeValue>0,"EXP",
                        ifelse(ModeValue<0,"LOG","LIN"))))

ggplot(d,aes(x=Year,y=Population,colour=Mode)) +
  geom_point()+theme_light()
```

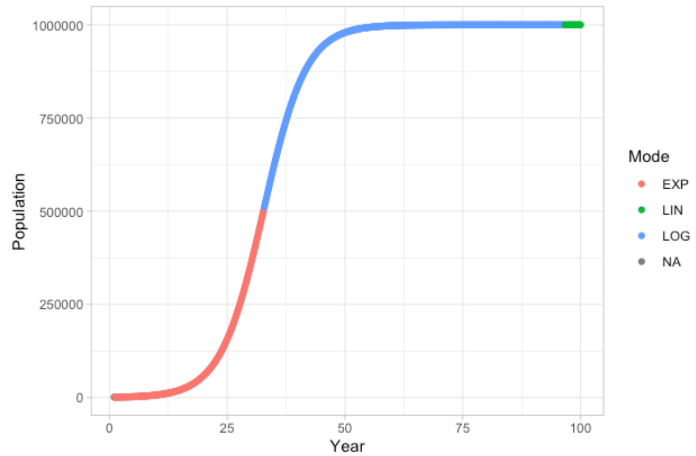
```
> d
# A tibble: 793 × 9
   Year Additions `Carrying Capacity` `Constrained Growth Rate` Effect …¹ Maxim…² Popul…³ ModeV…⁴ Mode
  <dbl>     <dbl>               <dbl>                     <dbl>     <dbl>   <dbl>   <dbl>   <dbl> <fct>
1 1          220.             1000000                     0.220     0.999   0.220   1000   NA      NA
2 1.12       226.             1000000                     0.220     0.999   0.220   1028.  NA      NA
3 1.25       233.             1000000                     0.220     0.999   0.220   1056.   0.757 EXP
4 1.38       239.             1000000                     0.220     0.999   0.220   1085.   0.778 EXP
5 1.5        246.             1000000                     0.220     0.999   0.220   1115.   0.8   EXP
6 1.62       252.             1000000                     0.220     0.999   0.220   1145.   0.821 EXP
7 1.75       259.             1000000                     0.220     0.999   0.220   1177.   0.844 EXP
```



A behavioral approach to feedback loop dominance analysis

David N. Ford[a]

linear atomic behavior pattern	$\partial(\lvert(\partial x/\partial t)\rvert)/\partial t = 0$
exponential atomic behavior pattern	$\partial(\lvert(\partial x/\partial t)\rvert)/\partial t > 0$
logarithmic atomic behavior pattern	$\partial(\lvert(\partial x/\partial t)\rvert)/\partial t < 0$



1. ggplot2 | **2. dplyr** | 3. deSolve | 4. purrr | 5. Shiny

- Visualisation is an important tool for insight generation, but it's rare that you get the data in exactly the right form you need (Wickham and Grolemund 2017)
  - Create new variables
  - Create summaries
  - Order data
- **dplyr** package is designed for data transformation

dplyr Basics: 5 key functions

Function	Purpose
filter()	Pick observations by their values
arrange()	Reorder the rows
select()	Pick variables by their names
mutate()	*Create new variables with functions of existing variables*
summarise()	*Collapse many values down to a single summary*

- ”A grammar of data manipulation”
- All verbs (functions) work similarly
  - The first argument is a data frame/tibble
  - The subsequent arguments decide what to do with the data frame/tibble
  - The result (data frame/tibble) supports chaining of steps – NOTE the “pipe operator” which we will cover later.

https://dplyr.tidyverse.org

Process Sensitivity Data (Stella)

A	B	C	D	E	F	G	H	I	J
Year	Run 1: Population	Run 2: Population	Run 3: Population	Run 4: Population	Run 5: Population	Run 6: Population	Run 7: Population	Run 8: Population	Run 9: Population
1	1000	1000	1000	1000	1000	1000	1000	1000	1000
2	1104.372133	1108.19136	1172.035949	1231.861465	1056.806787	1249.957467	1271.978986	1001.499482	1085.448046
3	1219.624672	1228.073885	1373.62837	1517.403111	1116.837003	1562.298647	1617.814294	1003.00121	1178.188937
4	1346.888976	1360.907683	1609.840283	1869.011748	1180.273137	1952.539704	2057.489958	1004.505187	1278.843588
5	1487.413409	1508.087943	1886.596445	2301.911204	1247.307964	2440.02602	2616.353021	1006.011418	1388.085504

```
> head(sd)
# A tibble: 6 × 203
   Year ="Run …¹ ="Run…² ="Run…³ ="Run…⁴ ="Run…⁵ ="Run…⁶ ="Run…⁷ ="Run…⁸ ="Run…⁹ ="Run…× ="Run…× ="Run…×
  <dbl>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
1     1    1000    1000    1000    1000    1000    1000    1000    1000    1000    1000    1000    1000
2     2    1104.   1108.   1172.   1232.   1057.   1250.   1272.   1001.   1085.   1017.   1152.   1179.
3     3    1220    1228    1374    1517    1117    1562    1618    1003    1178    1033    1326    1390
```

```r
library(readr)
library(dplyr)
library(ggplot2)
library(stringr)

# Load in the sensitivity data
sd <- read_csv("datasets/sensitivity runs/LTG_Sensitivity.csv")
```

Convert wide data to tidy data

```
> sd_tidy
# A tibble: 20,200 × 4
     Run  Year Variable   Value
   <int> <dbl> <chr>      <dbl>
 1     1     1 Population  1000
 2     2     1 Population  1000
 3     3     1 Population  1000
 4     4     1 Population  1000
 5     5     1 Population  1000
 6     6     1 Population  1000
 7     7     1 Population  1000
 8     8     1 Population  1000
 9     9     1 Population  1000
10    10     1 Population  1000
# … with 20,190 more rows
```

- The tidy data standard is designed to:
  - Facilitate initial exploration and analysis of data
  - Simplify the development of data analysis tools that work well together
- Rules
  - Each variable must have its own column
  - Each observation must have its own row
  - Each value must have its own cell

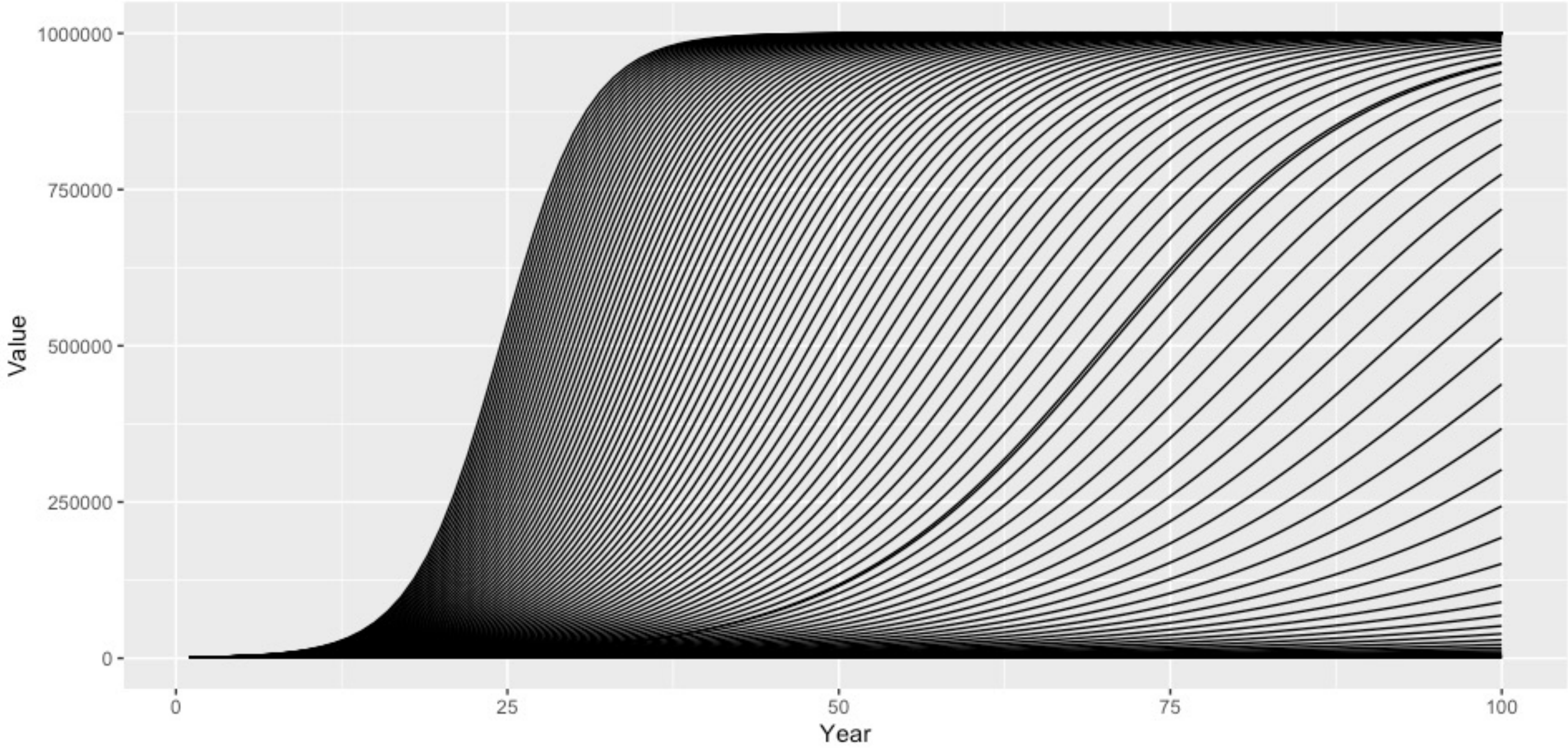
Code (tidyr and dplyr)

```
# Convert to tidy data format
sd_tidy <- sd %>%
             pivot_longer(cols=!Year,names_to="Variable",values_to="Value") %>%
             mutate(Variable=str_replace_all(Variable,'\"',""),
                    Variable=str_replace(Variable,"=","")) %>%
             separate(Variable,into=c("Run","Variable"),sep = ":") %>%
             mutate(Variable=trimws(Variable)) %>%
             separate(Run,c("TempRun","Run"),sep = " ",convert = TRUE) %>%
             select(Run,Year,Variable,Value)
```

Tidy data makes it easier to leverage tools such as ggplot2 and dplyr

Plotting multiple runs with ggplot2

```
ggplot(filter(sd_tidy,Variable=="Population"),aes(x=Year,y=Value,group=Run))+geom_line()
```



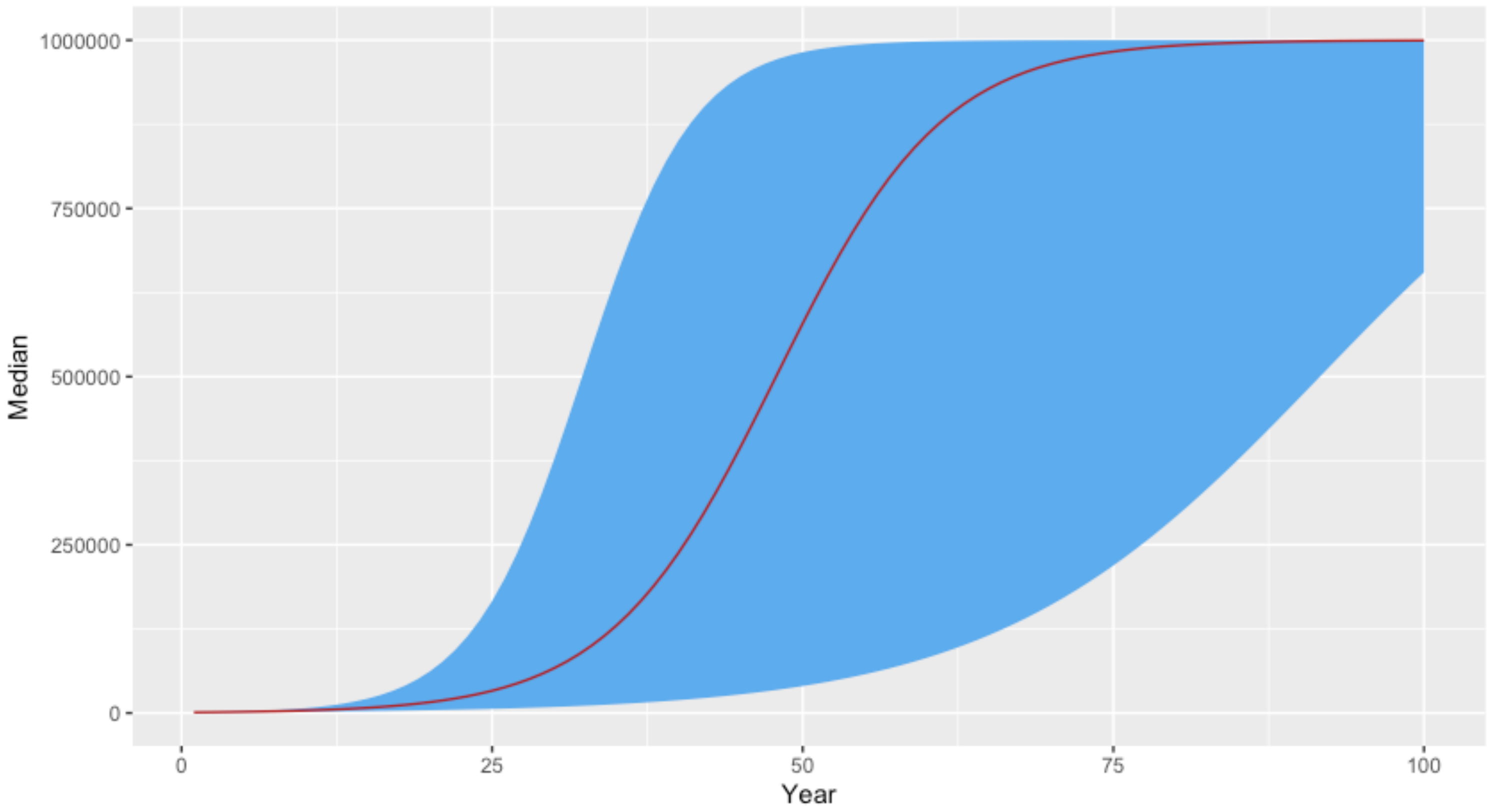
Finding summary data - quantiles

```
sum_runs <- sd_tidy %>%
              filter(Variable=="Population") %>%
              group_by(Variable,Year) %>%
              summarise(Q75=quantile(Value,0.75),
                        Q25=quantile(Value,0.25),
                        Median=median(Value),
                        Mean=mean(Value))
```

Finding summary data - quantiles

```
> sum_runs
# A tibble: 100 × 6
# Groups:   Variable [1]
   Variable    Year   Q75   Q25 Median  Mean
   <chr>      <dbl> <dbl> <dbl>  <dbl> <dbl>
 1 Population     1 1000  1000   1000  1000 
 2 Population     2 1246. 1079.  1158. 1163.
 3 Population     3 1553. 1164.  1342. 1362.
 4 Population     4 1936. 1256.  1554. 1607.
 5 Population     5 2412. 1356.  1800. 1909.
 6 Population     6 3005. 1463.  2085. 2282.
 7 Population     7 3743. 1578.  2414. 2745.
 8 Population     8 4662. 1703.  2796. 3322.
 9 Population     9 5805. 1837.  3238. 4043.
10 Population    10 7227. 1982.  3749. 4945.
# … with 90 more rows
# ℹ Use `print(n = ...)` to see more rows
```

```r
ggplot(sum_runs,aes(Year, Median)) +
  geom_ribbon(aes(ymin = Q25,ymax = Q75),fill = "steelblue2") +
  geom_line(color = "firebrick")
```



1. ggplot2 | 2. dplyr | **3. deSolve** | 4. purrr | 5. Shiny

R's deSolve package solves initial value problems written as ordinary differential equations (ODE), differential algebraic equations (DAE), and partial differential equations (PDE)

Journal of Statistical Software

February 2010, Volume 33, Issue 9. *http://www.jstatsoft.org/*

Solving Differential Equations in R: Package deSolve

Karline Soetaert
Netherlands Institute of Ecology

Thomas Petzoldt
Technische Universität Dresden

R. Woodrow Setzer
US Environmental Protection Agency

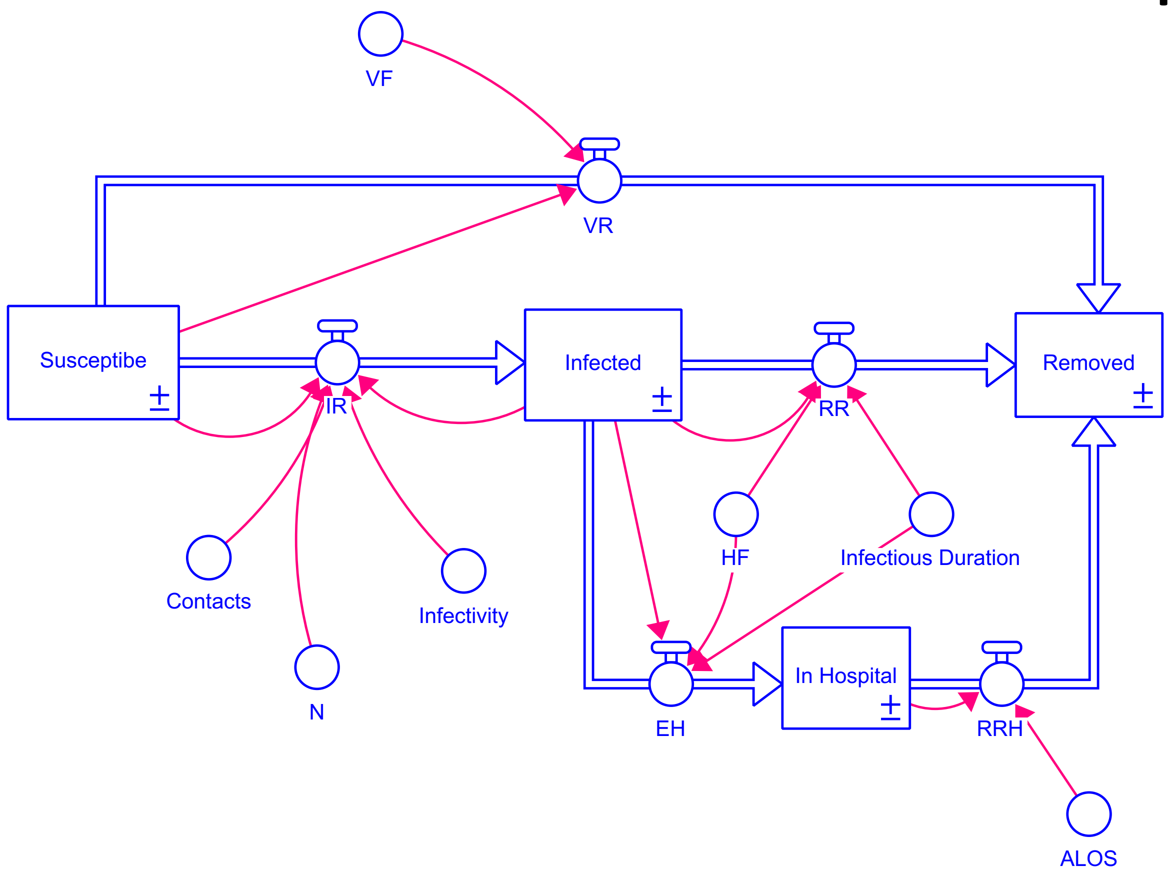
The model is implemented as an R function, and called via ode()

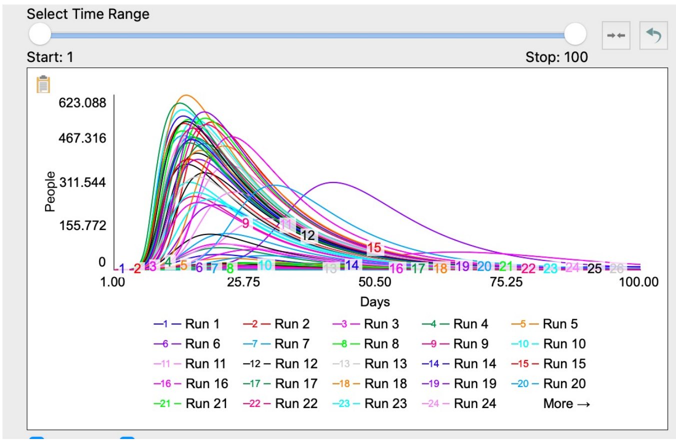
ode function in deSolve

Argument	Description
`y`	The initial (state) values for the ODE system, a vector. If *y* has a name attribute, the names will be used to label the output matrix
`times`	Time sequence for which output is wanted; the first value of times must be the initial time
`func`	• An R function that computes the values of the derivatives in the ODE system at time *t*. It must be defined as: `func <- function(t, y, parms,...)`, where **t** is the current time point in the integration and **y** is the current estimate of the variables in the ODE system. • If the initial value **y** has a names attribute, the names will be available inside func. • `parms` is a vector or list of parameters; ... (optional) are any other arguments passed to the function. • The return value of `func` should be a list, whose first element is a vector containing the derivatives of `y` with respect to time, and whose next elements are global values that are required at each point in times. The derivatives must be specified in the same order as the state variables `y`.
`parms`	Parameters passed to `func`.
`method`	Normally a string to indicate the integration method, for example, `"euler"`, `"rk4"`, `"ode23"`, `"ode45"`.
`returns`	A matrix of class `deSolve` with up to as many rows as elements in times and as many columns as elements in *y* plus the number of "global" values returned in the second element of the return from `func`, plus an additional column (the first) for the time value. This can be easily converted to a data frame object using the function `data.frame()`

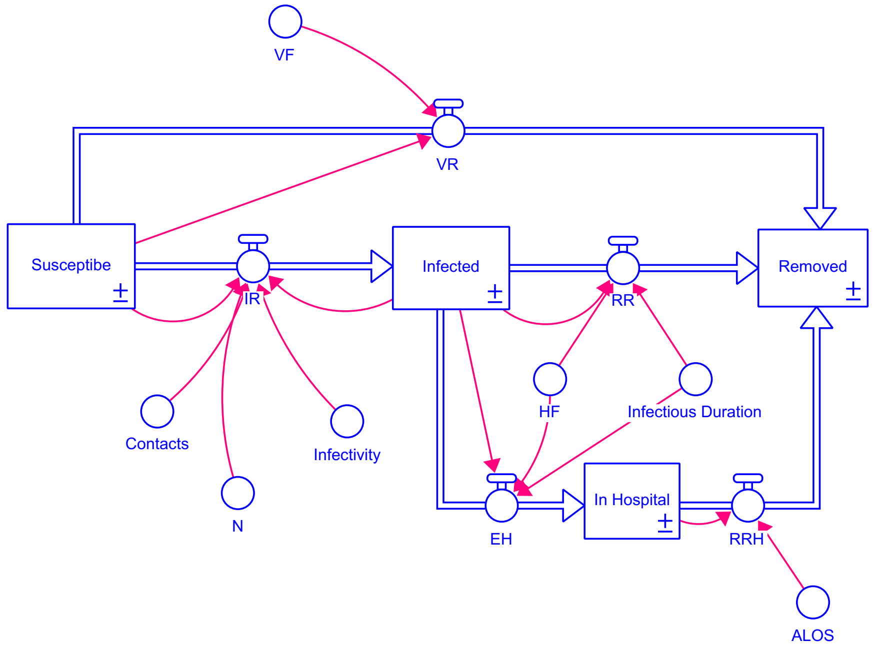
```
sim <-data.frame(ode(y=stocks,
                     times  = simtime,
                     func   = sirh,
                     parms  = auxs,
                     method = "euler"))
```

Model Example



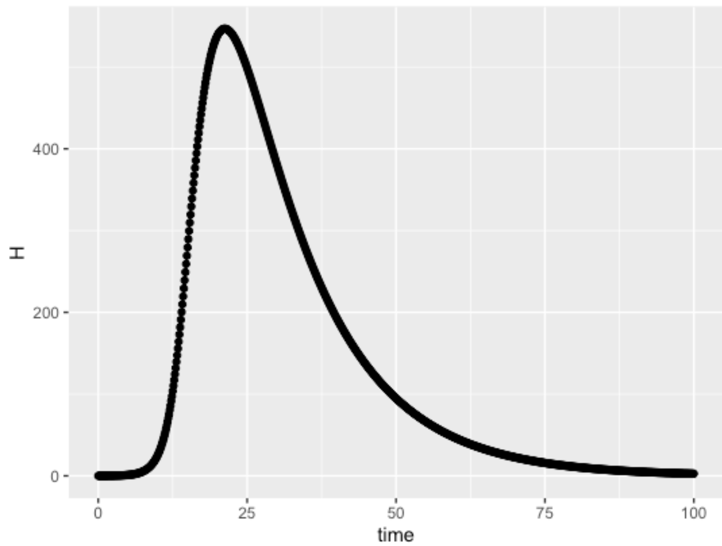


```r
sirh <- function(time, stocks, auxs){
  with(as.list(c(stocks, auxs)),{
    IR  <- contacts * S * I/N * i
    RR  <- I * (1-HF) /D
    EH  <- I * HF / D
    RRH <- H / ALOS
    VR  <- S * VF

    dS_dt <- -IR - VR
    dI_dt <- IR - RR - EH
    dR_dt <- RR + RRH + VR
    dH_dt <- EH - RRH

    CheckSum <- S+I+R+H

    # browser()

    return (list(c(dS_dt,dI_dt,dR_dt,dH_dt),
                 IR=IR,RR=RR,EH=EH,RRH=RRH,Contacts=contacts,
                 Infectivity=i,ALOS=ALOS,HospFrac=HF,VaccFrac=VF,
                 CheckSum=CheckSum,N=N))
  })
}
```



```r
simtime <- seq(start, finish, step)
# initialise vector of stocks
stocks  <- c(S=inits[1],
             I=inits[2],
             R=inits[3],
             H=inits[4])

# initialise vector of auxiliaries
auxs    <- c(contacts=contacts,  # Contacts
             i=infectivity,      # Infectivity
             D=duration,         # Duration of infectiousness
             N=N,                # Total Population
             ALOS=ALOS,          # Average length of stay
             HF=HF,              # Hospitalisation Fraction
             VF=VF)              # VaccinationFraction


sim <-data.frame(ode(y=stocks,
                     times  = simtime,
                     func   = sirh,
                     parms  = auxs,
                     method = "euler"))
```

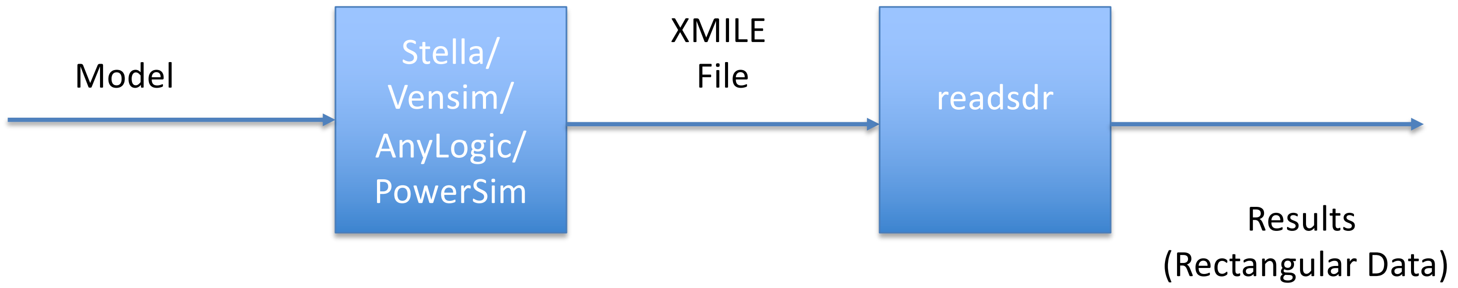
```r
p1 <- ggplot(sim,aes(x=time,y=H)) +
        geom_point()+
        geom_line()
```



Simulation output

```
> sim
# A tibble: 801 × 16
    time      S     I     R       H    IR    RR     EH      RRH Contacts Infectivity  ALOS HospFrac VaccFrac Check…¹     N
   <dbl>  <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl>  <dbl>    <dbl>    <dbl>       <dbl> <dbl>    <dbl>    <dbl>   <dbl> <dbl>
 1 0     99990   10   0     0        10.0  3.3  0.0333 0              10         0.1    14     0.01        0  100000   1e5
 2 0.125 99989.  10.8 0.413 0.00417  10.8  3.57 0.0361 0.000298       10         0.1    14     0.01        0  100000   1e5
 3 0.25  99987.  11.7 0.859 0.00864  11.7  3.87 0.0391 0.000617       10         0.1    14     0.01        0  100000   1e5
 4 0.375 99986.  12.7 1.34  0.0135   12.7  4.20 0.0424 0.000961       10         0.1    14     0.01        0  100000   1e5
 5 0.5   99984.  13.8 1.87  0.0186   13.8  4.55 0.0459 0.00133        10         0.1    14     0.01        0  100000   1e5
 6 0.625 99983.  14.9 2.44  0.0242   14.9  4.92 0.0497 0.00173        10         0.1    14     0.01        0  100000   1e5
 7 0.75  99981.  16.2 3.05  0.0302   16.2  5.33 0.0539 0.00216        10         0.1    14     0.01        0  100000   1e5
 8 0.875 99979.  17.5 3.72  0.0367   17.5  5.78 0.0584 0.00262        10         0.1    14     0.01        0  100000   1e5
 9 1     99977.  19.0 4.44  0.0436   19.0  6.26 0.0632 0.00312        10         0.1    14     0.01        0  100000   1e5
10 1.12  99974.  20.5 5.22  0.0512   20.5  6.78 0.0685 0.00365        10         0.1    14     0.01        0  100000   1e5
# … with 791 more rows, and abbreviated variable name ¹CheckSum
# ℹ Use `print(n = ...)` to see more rows
```

readsdr package https://github.com/jandraor/readsdr



Usage

```
library(readsdr)
filepath <- system.file("models/", "SIR.stmx", package = "readsdr")
mdl      <- read_xmile(filepath)
summary(mdl)
#>                    Length Class  Mode
#> description        4      -none- list
#> deSolve_components 4      -none- list
```

```
library(deSolve)

simtime <- seq(deSolve_components$sim_params$start,
               deSolve_components$sim_params$stop,
               deSolve_components$sim_params$dt)

output_deSolve <- ode(y      = deSolve_components$stocks,
                      times  = simtime,
                      func   = deSolve_components$func,
                      parms  = deSolve_components$consts,
                      method = "euler")

result_df <- data.frame(output_deSolve)
```

1. ggplot2 | 2. dplyr | 3. deSolve | **4. purrr** | 5. Shiny

- A **map function** is one that applies the same action/function to every element of an object (e.g. each entry of a list or a vector, or each of the columns of a data frame)
- The naming convention of the map functions are such that the type of the **output** is specified by the term that follows the underscore in the function name
- Consistent with the way of the tidyverse, the **first argument** of each mapping function is always the **data object** that you want to map over, and the **second argument** is always the ***function*** that you want to iteratively apply to each element of the input object

- `map(.x, .f)` is the main mapping function and returns a list
- `map_df(.x, .f)` returns a data frame
- `map_dbl(.x, .f)` returns a numeric (double) vector
- `map_chr(.x, .f)` returns a character vector
- `map_lgl(.x, .f)` returns a logical vector

map functions support iteration…

Sensitivity Sweep

```r
# Sensitivity sweep, modify 2 params
NSIMS <- 10
s_contacts <- sample(3:20,NSIMS,replace = T)
s_vacc     <- runif(NSIMS,min=0,max = 0.10)
```

```
> s_contacts
 [1] 15 19 14 17 10  5  8  4 13  9
>
>
> s_vacc
 [1] 0.03728884 0.05089462 0.06664778 0.06672938 0.09171309 0.05087442 0.07575453 0.03544004
 [9] 0.03325082 0.01643648
```

R Code to support sensitivity sweep

```r
run_sirh <- function(start=0,
                     finish=100,
                     step=0.125,
                     contacts=10,
                     infectivity=0.1,
                     duration=3,
                     N=100000,
                     ALOS=14,
                     HF=0.01,
                     inits=c(N-10,10,0,0),
                     VF=0.0){

  simtime <- seq(start, finish, step)
  # initialise vector of stocks
  stocks  <- c(S=inits[1],
               I=inits[2],
               R=inits[3],
               H=inits[4])
```

```r
  # initialise vector of auxiliaries
  auxs    <- c(contacts=contacts, # Contacts
               i=infectivity,     # Infectivity
               D=duration,        # Duration of infectiousness
               N=N,               # Total Population
               ALOS=ALOS,         # Average length of stay
               HF=HF,             # Hospitalisation Fraction
               VF=VF)             # VaccinationFraction


  sim <-data.frame(ode(y=stocks,
                       times  = simtime,
                       func   = sirh,
                       parms  = auxs,
                       method = "euler"))

  as_tibble(sim)
}
```

map2 Iteration code…

```r
count <- 1
# map2 is an iterator over two vectors
sens <- map2(s_contacts,s_vacc,~{
  message(glue("Sim {count} contacts {.x} vacc Fr {.y}"))
  out_sim <- run_sirh(contacts = .x,VF = .y) %>%
               mutate(Run=count) %>%
               select(Run,everything())
  count <<- count+1
  out_sim
})

full_sims <- bind_rows(sens)
```

Full results

```
> full_sims
# A tibble: 200,250 × 17
     Run  time       S     I     R        H    IR    RR     EH      RRH Contacts Infectiv…¹  ALOS HospF…² VaccF…³
   <dbl> <dbl>   <dbl> <dbl> <dbl>    <dbl> <dbl> <dbl>  <dbl>    <dbl>    <dbl>      <dbl> <dbl>   <dbl>   <dbl>
 1     1 0     99990   10      0   0         18.0  3.3  0.0333 0              18        0.1    14    0.01 0.00753
 2     1 0.125 99894.  11.8   94.5 0.00417   21.3  3.90 0.0394 0.000298       18        0.1    14    0.01 0.00753
 3     1 0.25  99797.  14.0  189.  0.00906   25.1  4.62 0.0467 0.000647       18        0.1    14    0.01 0.00753
 4     1 0.375 99700.  16.6  283.  0.0148    29.7  5.46 0.0552 0.00106        18        0.1    14    0.01 0.00753
 5     1 0.5   99602.  19.6  378.  0.0216    35.1  6.46 0.0653 0.00154        18        0.1    14    0.01 0.00753
 6     1 0.625 99504.  23.2  473.  0.0295    41.5  7.64 0.0772 0.00211        18        0.1    14    0.01 0.00753
 7     1 0.75  99405.  27.4  567.  0.0389    49.0  9.03 0.0913 0.00278        18        0.1    14    0.01 0.00753
 8     1 0.875 99306.  32.4  662.  0.0500    57.8 10.7  0.108  0.00357        18        0.1    14    0.01 0.00753
 9     1 1     99205.  38.2  757.  0.0630    68.3 12.6  0.127  0.00450        18        0.1    14    0.01 0.00753
10     1 1.12  99103.  45.2  851.  0.0784    80.6 14.9  0.151  0.00560        18        0.1    14    0.01 0.00753
# … with 200,240 more rows, 2 more variables: CheckSum <dbl>, N <dbl>, and abbreviated variable names
#   ¹Infectivity, ²HospFrac, ³VaccFrac
# ℹ Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

Summary, Peak Cases

```r
summ <- full_sims %>%
  group_by(Run) %>%
  summarise(PeakH=max(H),
            Contacts=first(Contacts),
            VF=first(VaccFrac)) %>%
  ungroup()
```

```
> summ
# A tibble: 250 × 4
     Run   PeakH Contacts      VF
   <dbl>   <dbl>    <dbl>   <dbl>
 1     1 605.          18 0.00753
 2     2 439.          19 0.0463
 3     3 227.          19 0.0961
 4     4   7.92         8 0.0512
 5     5   0.141        3 0.0943
 6     6   0.827        7 0.0953
 7     7 154.          17 0.0980
 8     8 217.          15 0.0675
 9     9 420.           9 0.00715
10    10  73.4          9 0.0373
# … with 240 more rows
# ℹ Use `print(n = ...)` to see more rows
```

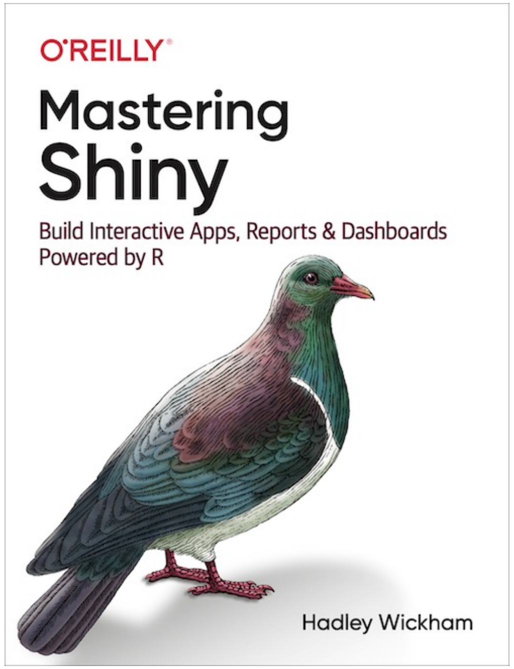
```r
p3 <- ggplot(summ,aes(x=Contacts,y=VF,size=PeakH,colour=PeakH))+
  geom_point()+
  scale_color_gradient(low="blue", high="red")+geom_jitter()
```

1. ggplot2 → 2. dplyr → 3. deSolve → **4. purrr** → **5. Shiny**

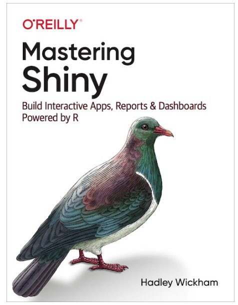
Spreadsheets are closely related to reactive programming: you declare the relationship between cells using formulas, and when one cell changes, all of its dependencies automatically update.

Hadley Wickham – Mastering Shiny

https://mastering-shiny.org

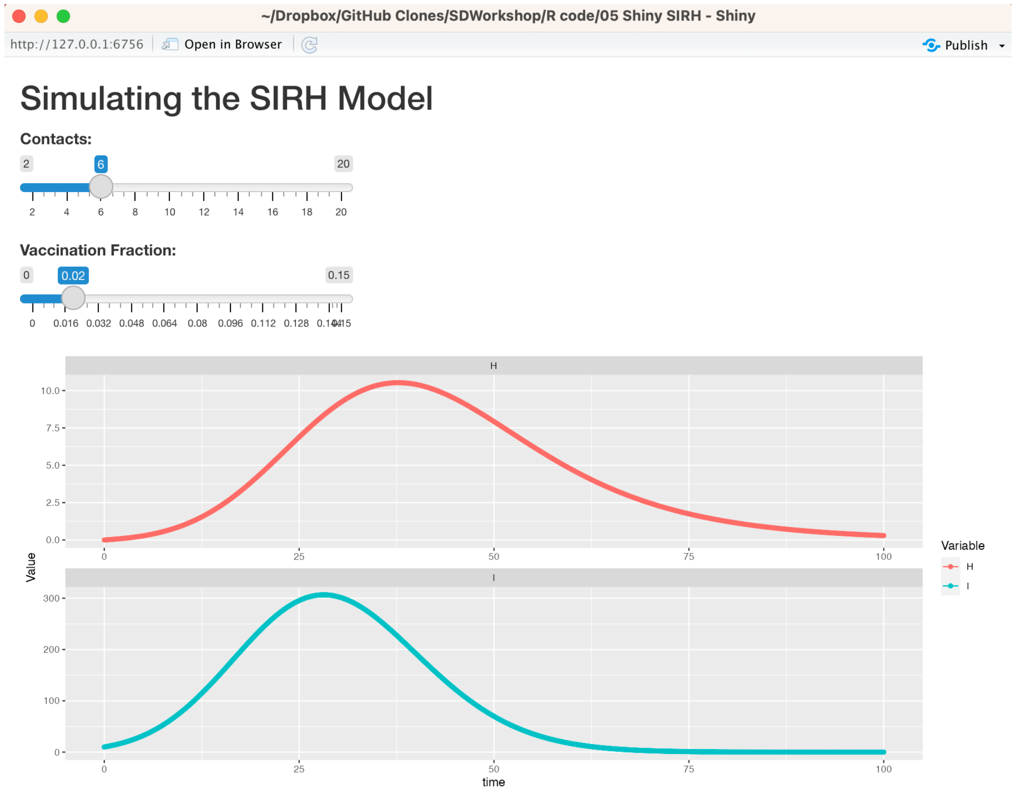


Shiny (Wickham)



- Shiny is a framework for creating web applications using R code.
- It is designed primarily with data scientists in mind, and to that end, you can create pretty complicated Shiny apps with no knowledge of HTML, CSS, or JavaScript.
- On the other hand, Shiny doesn't limit you to creating trivial or prefabricated apps: its user interface components can be easily customized or extended, and its server uses reactive programming to let you create any type of back end logic you want.
- Shiny is designed to feel almost magically easy when you're getting started, and yet the deeper you get into how it works, the more you realize it's built out of general building blocks that have strong software engineering principles behind them

The App

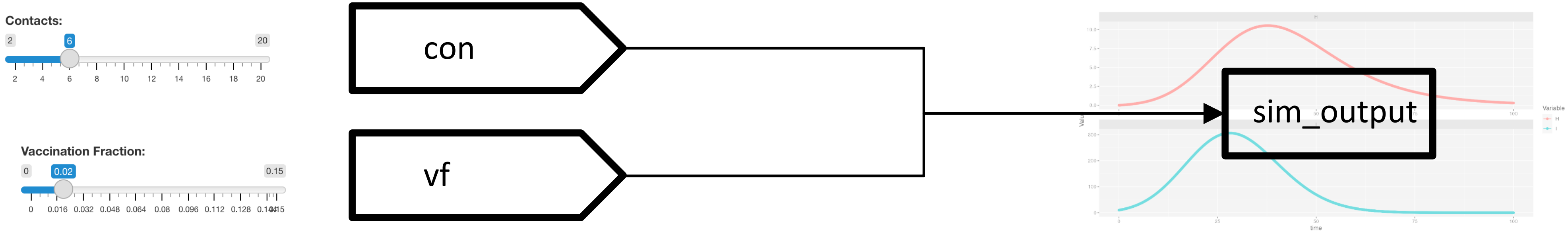



```
shinyApp(ui, server)
```

The ui object

```
ui <- fluidPage(
  titlePanel("Simulating the SIRH Model"),
  sliderInput("con", "Contacts:",
              min = 2, max = 20, value = 6
  ),
  sliderInput("vf", "Vaccination Fraction:",
              min = 0, max = 0.15, value = .02
  ),
  plotOutput("sim_output")
)
```

The reactive graph – spreadsheet cell analogy

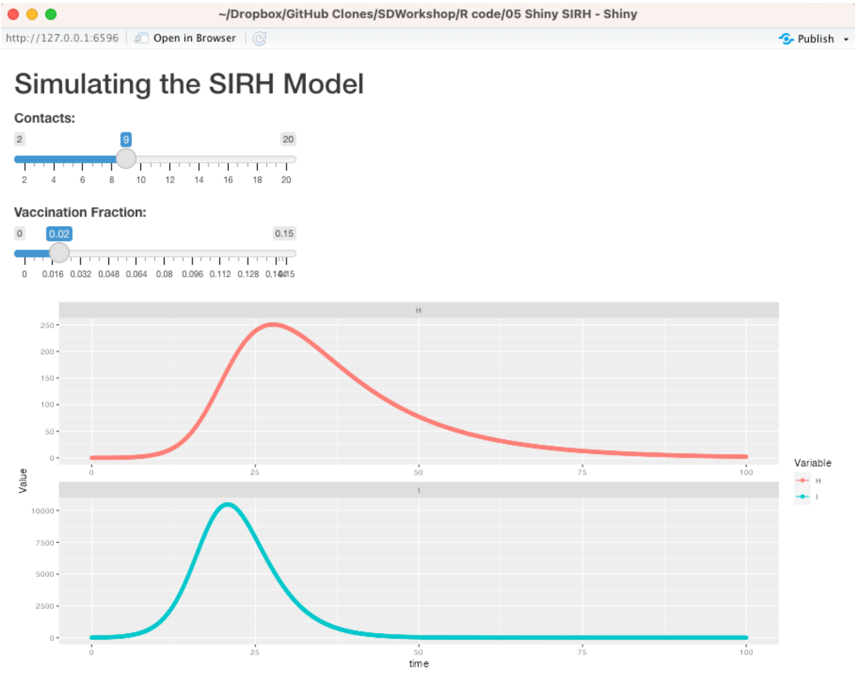


- Any change to an input control
- Leads to a reaction in the server code
- Which then updates the output

The Server

```r
server <- function(input, output, session){
  message("\nStarting the server...")
  output$sim_output <- renderPlot({
    sim <- run_sirh(contacts = input$con,
                    VF = input$vf) %>%
             select(time,I,H) %>%
             pivot_longer(cols = -time,
                          names_to = "Variable",
                          values_to = "Value")

    ggplot(sim,aes(x=time,y=Value,colour=Variable))+
      geom_point()+geom_line()+
      facet_wrap(~Variable,nrow = 2,scales="free")
  })
}
```

Outputs


~/Dropbox/GitHub Clones/SDWorkshop/R code/05 Shiny SIRH - Shiny
http://127.0.0.1:6596
Open in Browser
Publish
Simulating the SIRH Model
Contacts:
Vaccination Fraction:
Value
time
Variable


~/Dropbox/GitHub Clones/SDWorkshop/R code/05 Shiny SIRH - Shiny
http://127.0.0.1:6596
Open in Browser
Publish
Simulating the SIRH Model
Contacts:
Vaccination Fraction:
Value
time
Variable

Conclusion

1. ggplot2 → 2. dplyr → 3. deSolve → 4. purrr → 5. Shiny

- R can integrate in a flexible way within your System Dynamics workflow
- Visualization, summaries, modelling, sensitivity and web apps
- Can also support calibration workflows - Andrade and Duggan 2021