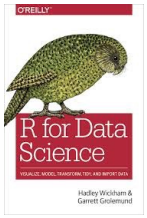
System Dynamics Modelling using R

Prof. Jim Duggan,
School of Computer Science
National University of Ireland Galway.

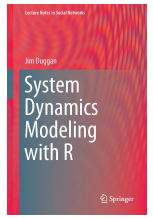
https://github.com/JimDuggan/SDMR

Lecture Overview

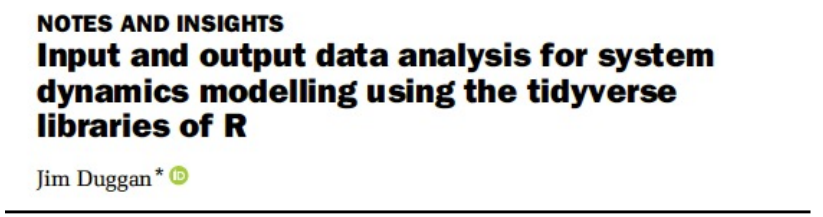
- Introduction to R and the tidyverse
  - Visualisation - **ggplot2**
  - Transformation – **dplyr** and **purrr**
- Using R with System Dynamics, 3 Examples
  - deSolve
  - readsdr
  - Exploring sensitivity runs



https://r4ds.had.co.nz



https://github.com/JimDuggan/SDMR

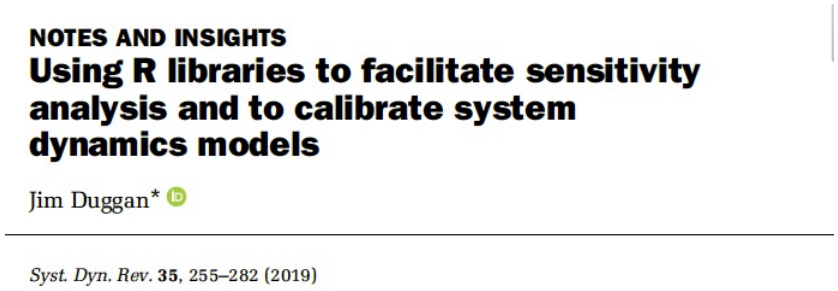


NOTES AND INSIGHTS

Input and output data analysis for system dynamics modelling using the tidyverse libraries of R

Jim Duggan*

Syst. Dyn. Rev. **34**, 438–461 (2018)



NOTES AND INSIGHTS

Using R libraries to facilitate sensitivity analysis and to calibrate system dynamics models

Jim Duggan*

Syst. Dyn. Rev. **35**, 255–282 (2019)

Jim Duggan

- Lecturer in:
  - Programming (R, MATLAB),
  - Modelling & Simulation
- Research interests:
  - System Dynamics
  - Data Science
  - Public Health Modelling
- Public Health Project Work
  - Infectious Disease Modelling (COVID-19)
  - PANDEM-2 Project
  - Contact Tracing (WHO)
  - Suicide Prevention (HSE, School of Psychology)

Article | Cited By (6) | Tweetations (14) | Metrics

Review

Mobile Health Technology Interventions for Suicide Prevention: Systematic Review

Ruth Melia[1,2], DClinPsych ; Kady Francis[1], MSc ; Emma Hickey[3], MSc ; John Bogue[1], DClinPsych ; Jim Duggan[4], PhD ; Mary O'Sullivan[1], MA, ; Karen Young[5], PhD

[1]School of Psychology, National University of Ireland Galway, Galway, Ireland
[2]Psychology Department, Health Service Executive Mid-West, Ennis, Ireland
[3]Psychology Department, Health Service Executive Mid-West, Limerick, Ireland
[4]Discipline of Information Technology, National University of Ireland Galway, Galway, Ireland
[5]Insight-Centre, Discipline of Information Technology, National University of Ireland Galway, Galway, Ireland

Review | Infodemiology and Infoveillance | Infoveillance, Infodemiology, Digital Disease Surveillance, Infodemic Management

Article | Cited By (6) | Tweetations (29) | Metrics

Review

The Application of Internet-Based Sources for Public Health Surveillance (Infoveillance): Systematic Review

Joana M Barros[1,2], MSc ; Jim Duggan[2], PhD ; Dietrich Rebholz-Schuhmann[3], PhD

[1]Insight Centre for Data Analytics, National University of Ireland Galway, Galway, Ireland
[2]School of Computer Science, National University of Ireland Galway, Galway, Ireland
[3]ZB MED - Information Centre for Life Sciences, University Cologne, Cologne, Germany

Epidemics 33 (2020) 100415

Contents lists available at ScienceDirect

Epidemics

journal homepage: www.elsevier.com/locate/epidemics

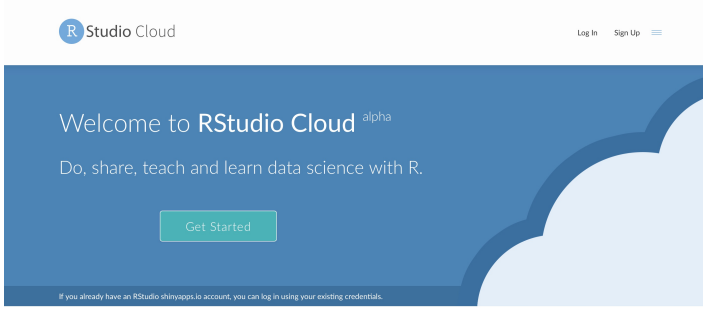
An evaluation of Hamiltonian Monte Carlo performance to calibrate age-structured compartmental SEIR models to incidence data

Jair Andrade *, Jim Duggan

Data Science Institute and School of Computer Science, National University of Ireland Galway, Ireland

(1) R and the tidyverse

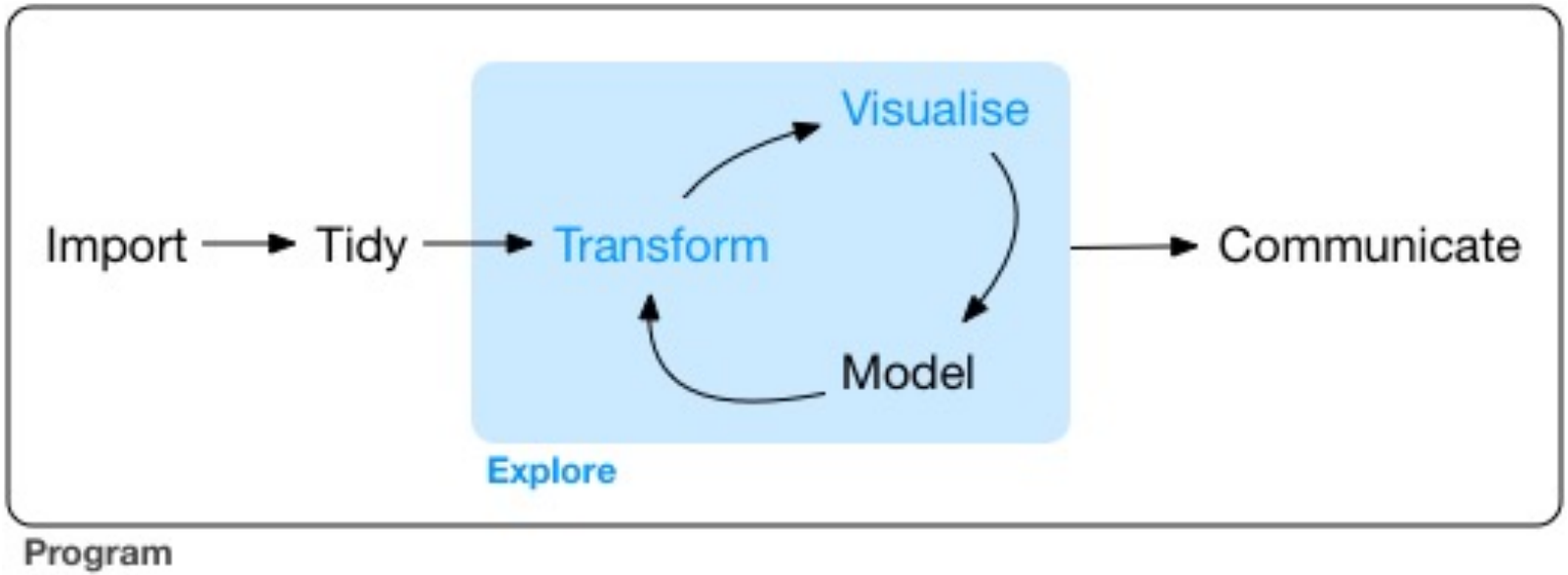
- R's *mission* is to enable the best and most thorough exploration of data possible (Chambers 2008).
- It is a dialect of the S language, developed at Bell Laboratories
- ACM noted that *S "will forever alter the way people analyze, visualize, and manipulate data"*




```
# We use this for processing the answer
# In programming, we "stand on the shoulders of giants"
library(stringi)

# This gets the input from the user.
# The result is stored in a variable
# Variables are important in programming!
name <- readline(prompt="Enter a name: ")

# We call a specially designed function to get the answer
# In R, we call functions all the time
# A function is a "mini-program"
ans <- stri_reverse(name)

# After all this work, we output the result
cat("The reverse of ", name, "is ===>", ans)
```

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).



Data Frames/Tibbles – aimsir17

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

```
> observations
# A tibble: 219,000 x 12
   station  year month   day  hour date                rain  temp  rhum
   <chr>   <dbl> <dbl> <int> <int> <dttm>              <dbl> <dbl> <dbl>
 1 ATHENRY  2017     1     1     0 2017-01-01 00:00:00   0     5.2    89
 2 ATHENRY  2017     1     1     1 2017-01-01 01:00:00   0     4.7    89
 3 ATHENRY  2017     1     1     2 2017-01-01 02:00:00   0     4.2    90
 4 ATHENRY  2017     1     1     3 2017-01-01 03:00:00   0.1   3.5    87
 5 ATHENRY  2017     1     1     4 2017-01-01 04:00:00   0.1   3.2    89
 6 ATHENRY  2017     1     1     5 2017-01-01 05:00:00   0     2.1    91
 7 ATHENRY  2017     1     1     6 2017-01-01 06:00:00   0     2      89
 8 ATHENRY  2017     1     1     7 2017-01-01 07:00:00   0     1.7    89
 9 ATHENRY  2017     1     1     8 2017-01-01 08:00:00   0     1      91
10 ATHENRY  2017     1     1     9 2017-01-01 09:00:00   0     1.1    91
# … with 218,990 more rows, and 3 more variables: msl <dbl>, wdsp <dbl>,
#   wddir <dbl>
```

Tidy Data - Overview

- 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

```
> observations
# A tibble: 219,000 x 12
   station  year month   day  hour date                rain  temp  rhum
   <chr>   <dbl> <dbl> <int> <int> <dttm>              <dbl> <dbl> <dbl>
 1 ATHENRY  2017     1     1     0 2017-01-01 00:00:00   0     5.2    89
 2 ATHENRY  2017     1     1     1 2017-01-01 01:00:00   0     4.7    89
 3 ATHENRY  2017     1     1     2 2017-01-01 02:00:00   0     4.2    90
 4 ATHENRY  2017     1     1     3 2017-01-01 03:00:00   0.1   3.5    87
 5 ATHENRY  2017     1     1     4 2017-01-01 04:00:00   0.1   3.2    89
 6 ATHENRY  2017     1     1     5 2017-01-01 05:00:00   0     2.1    91
 7 ATHENRY  2017     1     1     6 2017-01-01 06:00:00   0     2      89
 8 ATHENRY  2017     1     1     7 2017-01-01 07:00:00   0     1.7    89
 9 ATHENRY  2017     1     1     8 2017-01-01 08:00:00   0     1      91
10 ATHENRY  2017     1     1     9 2017-01-01 09:00:00   0     1.1    91
# … with 218,990 more rows, and 3 more variables: msl <dbl>, wdsp <dbl>,
#   wddir <dbl>
```

https://rpubs.com/bradleyboehmke/data_wrangling

Data Visualisation with **ggplot2**

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

```
> dt <- ggplot2::mpg
>
> dt
# 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   2.0  2008     4 manual(m6)     f    20    31     p compact
4          audi         a4   2.0  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   2.0  2008     4 manual(m6)     4    20    28     p compact
# ... with 224 more rows
```

Fuel Economy Data Set (ggplot2::mpg)

This dataset contains a subset of the fuel economy data that the EPA makes available on http://fueleconomy.gov. It contains only models which had a new release every year between 1999 and 2008 - this was used as a proxy for the popularity of the car.

manufacturer	manufacturer	**drv**	f = front-wheel drive, r = rear wheel drive, 4 = 4wd
model	model name	**cty**	city miles per gallon
displ	engine displacement, in litres	**hwy**	highway miles per gallon
year	year of manufacture	**fl**	fuel type
cyl	number of cylinders	**class**	"type" of car
trans	type of transmission		

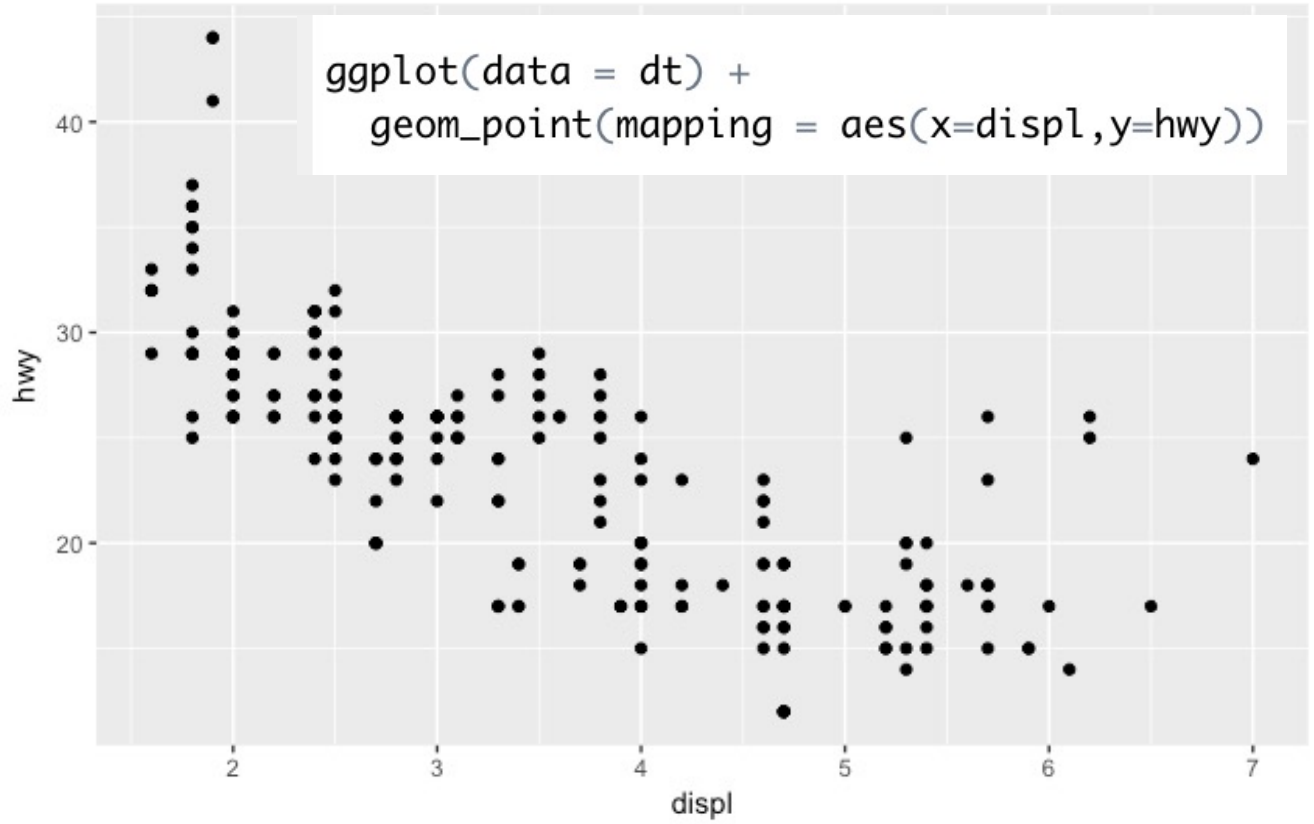
Selecting data

```
> dt
# 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   2.0  2008     4 manual(m6)     f    20    31     p compact
4          audi         a4   2.0  2008     4   auto(av)     f    21    30     p compact
5          audi         a4   2.8  1999     6   auto(l5)     f    16    26     p compact
```

- Among the variables are:
  - **displ**, a car's engine size in litres
  - **hwy**, a car's fuel efficiency on the highway in miles per gallon

Creating a ggplot

```
ggplot(data = dt) +
  geom_point(mapping = aes(x=displ,y=hwy))
```



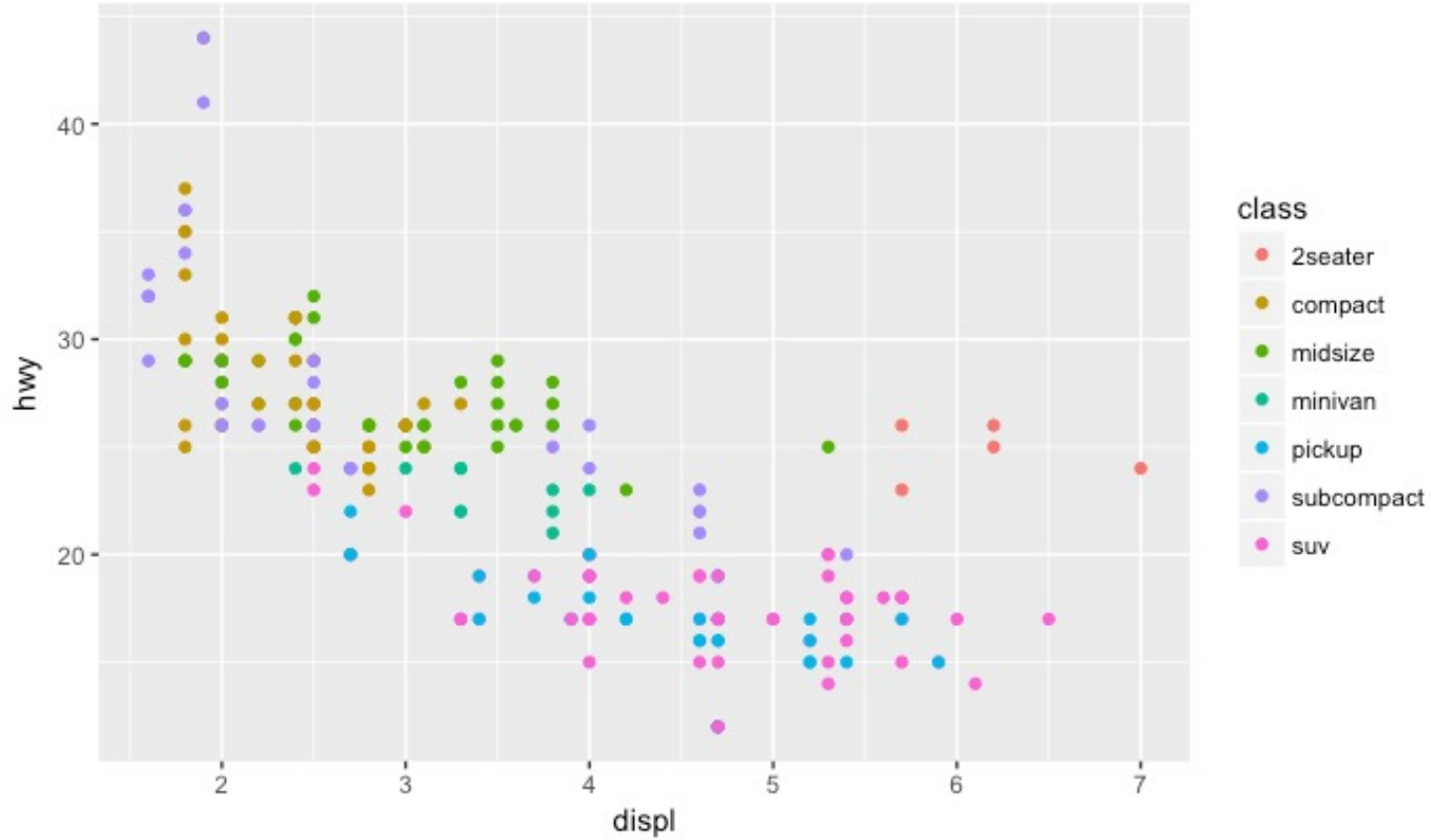
Aesthetic Mappings

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

```
> unique(dt$class)
[1] "compact"    "midsize"    "suv"        "2seater"    "minivan"
[6] "pickup"     "subcompact"
```

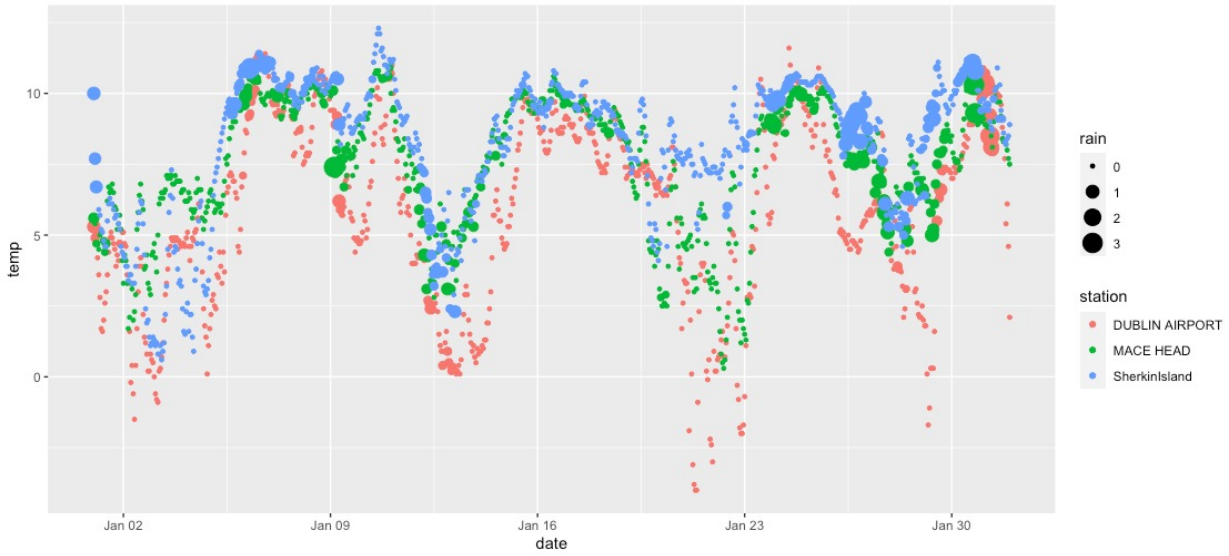
- A third variable can be added to a 2-D plot by mapping it to an aesthetic.
- An aesthetic is a visual property of the plot's objects.
- An aesthetic's *level* could be colour, size or shape.

```
ggplot(data = dt) +
  geom_point(mapping = aes(x=displ,y=hwy,colour=class))
```



Challenge

Generate the following graph from aimsir17 (January). Use the filter command to create the dataset.

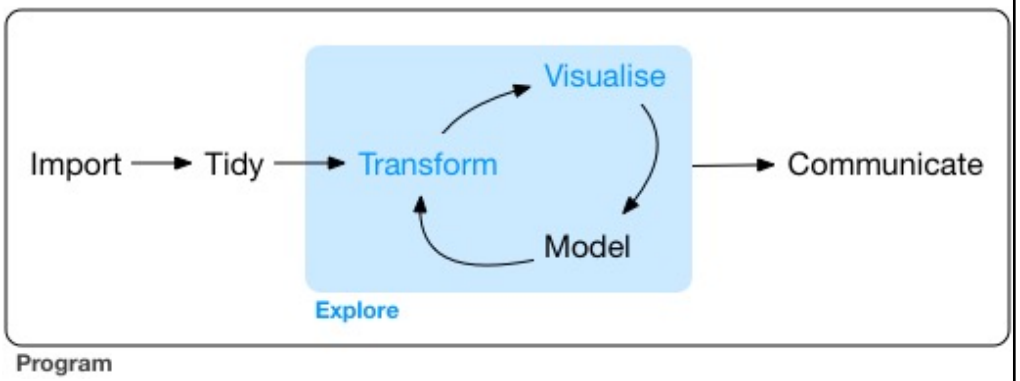



```r
library(aimsir17)
library(dplyr)
library(ggplot2)

target <- c("DUBLIN AIRPORT",
            "MACE HEAD",
            "SherkinIsland")

data <- observations %>%
          filter(month==1) %>%
          filter(station %in% target)
```

```
> data
# A tibble: 2,232 x 12
   station  year month   day  hour date                rain
   <chr>   <dbl> <dbl> <int> <int> <dttm>              <dbl>
 1 DUBLIN… 2017     1     1     0 2017-01-01 00:00:00   0.9
 2 DUBLIN… 2017     1     1     1 2017-01-01 01:00:00   0.2
 3 DUBLIN… 2017     1     1     2 2017-01-01 02:00:00   0.1
 4 DUBLIN… 2017     1     1     3 2017-01-01 03:00:00   0
 5 DUBLIN… 2017     1     1     4 2017-01-01 04:00:00   0
 6 DUBLIN… 2017     1     1     5 2017-01-01 05:00:00   0
 7 DUBLIN… 2017     1     1     6 2017-01-01 06:00:00   0
 8 DUBLIN… 2017     1     1     7 2017-01-01 07:00:00   0
 9 DUBLIN… 2017     1     1     8 2017-01-01 08:00:00   0
10 DUBLIN… 2017     1     1     9 2017-01-01 09:00:00   0
# … with 2,222 more rows, and 5 more variables: temp <dbl>,
#   rhum <dbl>, msl <dbl>, wdsp <dbl>, wddir <dbl>
```

Data Transformation

- 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” https://dplyr.tidyverse.org
- 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.

1. filter()

- First argument the name of the data frame
- Subsequent arguments are expressions that filter the data frame
- Subsequent arguments can be viewed as a succession of “and” statements
- Number of columns does not change
- Number of rows reduced (filtered)

```
> bel <- filter(observations,station=="BELMULLET")
> bel
# A tibble: 8,760 x 12
   station  year month   day  hour date                rain
   <chr>   <dbl> <dbl> <int> <int> <dttm>              <dbl>
 1 BELMUL… 2017     1     1     0 2017-01-01 00:00:00   0
 2 BELMUL… 2017     1     1     1 2017-01-01 01:00:00   0.5
 3 BELMUL… 2017     1     1     2 2017-01-01 02:00:00   0
 4 BELMUL… 2017     1     1     3 2017-01-01 03:00:00   0.4
 5 BELMUL… 2017     1     1     4 2017-01-01 04:00:00   0.6
 6 BELMUL… 2017     1     1     5 2017-01-01 05:00:00   0.1
 7 BELMUL… 2017     1     1     6 2017-01-01 06:00:00   0
 8 BELMUL… 2017     1     1     7 2017-01-01 07:00:00   0
 9 BELMUL… 2017     1     1     8 2017-01-01 08:00:00   0
10 BELMUL… 2017     1     1     9 2017-01-01 09:00:00   0
# … with 8,750 more rows, and 5 more variables: temp <dbl>,
#   rhum <dbl>, msl <dbl>, wdsp <dbl>, wddir <dbl>
```

Relational operators in R

Operators	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	not x
x | y	x OR y
x & y	x AND y

```
> bel <- filter(observations,station=="BELMULLET")
> bel
# A tibble: 8,760 x 12
   station  year month   day  hour date                rain
   <chr>   <dbl> <dbl> <int> <int> <dttm>              <dbl>
 1 BELMUL… 2017     1     1     0 2017-01-01 00:00:00   0
 2 BELMUL… 2017     1     1     1 2017-01-01 01:00:00   0.5
 3 BELMUL… 2017     1     1     2 2017-01-01 02:00:00   0
 4 BELMUL… 2017     1     1     3 2017-01-01 03:00:00   0.4
 5 BELMUL… 2017     1     1     4 2017-01-01 04:00:00   0.6
 6 BELMUL… 2017     1     1     5 2017-01-01 05:00:00   0.1
 7 BELMUL… 2017     1     1     6 2017-01-01 06:00:00   0
 8 BELMUL… 2017     1     1     7 2017-01-01 07:00:00   0
 9 BELMUL… 2017     1     1     8 2017-01-01 08:00:00   0
10 BELMUL… 2017     1     1     9 2017-01-01 09:00:00   0
# … with 8,750 more rows, and 5 more variables: temp <dbl>,
#   rhum <dbl>, msl <dbl>, wdsp <dbl>, wddir <dbl>
```

Show rows for “MACE HEAD” in January

```
> mhj <- filter(observations,station=="MACE HEAD",month==1)
>
> mhj
# A tibble: 744 x 12
```

	station	year	month	day	hour	date	rain	temp	rhum	msl	wdsp	wddir
	<chr>	<dbl>	<dbl>	<int>	<int>	<dttm>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	MACE HEAD	2017	1	1	0	2017-01-01 00:00:00	0.5	5.6	88	1023.	17	340
2	MACE HEAD	2017	1	1	1	2017-01-01 01:00:00	0	5.4	84	1023.	17	340
3	MACE HEAD	2017	1	1	2	2017-01-01 02:00:00	0.1	4.7	87	1023.	14	340
4	MACE HEAD	2017	1	1	3	2017-01-01 03:00:00	0	4.7	81	1023.	15	350
5	MACE HEAD	2017	1	1	4	2017-01-01 04:00:00	0	4.5	80	1024.	12	350
6	MACE HEAD	2017	1	1	5	2017-01-01 05:00:00	0	5	71	1024	13	20
7	MACE HEAD	2017	1	1	6	2017-01-01 06:00:00	0	5.1	66	1024.	13	30
8	MACE HEAD	2017	1	1	7	2017-01-01 07:00:00	0	4.8	76	1026.	19	10
9	MACE HEAD	2017	1	1	8	2017-01-01 08:00:00	0.1	4.8	78	1026.	16	360
10	MACE HEAD	2017	1	1	9	2017-01-01 09:00:00	0.1	4.4	82	1027.	15	10

```
# … with 734 more rows
```

2. arrange()

- Changes the order of rows.
- Used for sorting values
- Takes a tibble and a set of column names to order by

```
> arrange(observations,temp)
# A tibble: 219,000 x 12
   station     year month   day  hour date                rain  temp  rhum   msl  wdsp wddir
   <chr>      <dbl> <dbl> <int> <int> <dttm>              <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 CASEMENT    2017    12    11     4 2017-12-11 04:00:00     0  -6.2    91  989.     5   250
 2 GURTEEN     2017    12    11     3 2017-12-11 03:00:00     0  -6      94  989.     2   240
 3 GURTEEN     2017    12    11     4 2017-12-11 04:00:00     0  -6      95  990.     1   240
 4 GURTEEN     2017    12    11     1 2017-12-11 01:00:00     0  -5.9    92  988.     3   230
 5 GURTEEN     2017    12    11     5 2017-12-11 05:00:00     0  -5.8    95  990.     1   260
 6 GURTEEN     2017    12    11     0 2017-12-11 00:00:00     0  -5.7    94  988      2   280
 7 CASEMENT    2017    12    11     2 2017-12-11 02:00:00     0  -5.6    92  988.     4   230
 8 GURTEEN     2017    12    11     2 2017-12-11 02:00:00     0  -5.6    94  989.     3   230
 9 MOORE PARK  2017     1     3     9 2017-01-03 09:00:00     0  -5.6    91 1033.     1   330
10 CASEMENT    2017    12    11     3 2017-12-11 03:00:00     0  -5.4    92  988.     4   250
# … with 218,990 more rows
```

In descending order - desc()

```
> arrange(observations,desc(temp))
# A tibble: 219,000 x 12
   station       year month   day  hour date                rain  temp  rhum   msl  wdsp wddir
   <chr>        <dbl> <dbl> <int> <int> <dttm>              <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 PHOENIX PARK  2017     6    21    13 2017-06-21 13:00:00   0.1  28.3    51 1010     NA    NA
 2 PHOENIX PARK  2017     6    21    12 2017-06-21 12:00:00   0    27.5    54 1011.    NA    NA
 3 PHOENIX PARK  2017     6    21    14 2017-06-21 14:00:00   0    27.5    49 1010.    NA    NA
 4 PHOENIX PARK  2017     6    21    16 2017-06-21 16:00:00   0    26.8    61 1009.    NA    NA
 5 CASEMENT      2017     6    21    12 2017-06-21 12:00:00   0    26.6    54 1011.    11   150
 6 MOORE PARK    2017     6    19    16 2017-06-19 16:00:00   0    26.6    50 1018.     3   200
 7 DUNSANY       2017     6    21    12 2017-06-21 12:00:00   0    26.5    55 1010.     8   150
 8 PHOENIX PARK  2017     6    21    11 2017-06-21 11:00:00   0    26.5    56 1011.    NA    NA
 9 PHOENIX PARK  2017     6    17    16 2017-06-17 16:00:00   0    26.4    42 1024.    NA    NA
10 PHOENIX PARK  2017     6    21    15 2017-06-21 15:00:00   0    26.4    61 1009.    NA    NA
# … with 218,990 more rows
```

3. select()

- It is not uncommon to get datasets with hundreds, or even thousands, of variables
- A challenge is to narrow down on the variables of you're interested in
- select() allows you to rapidly zoom in on a useful subset using operations based on the variable names
- Number of rows does not change

```
> new_obs <- select(observations,station,year,month,day,hour,temp)
> new_obs
# A tibble: 219,000 x 6
   station  year month   day  hour  temp
   <chr>   <dbl> <dbl> <int> <int> <dbl>
 1 ATHENRY  2017     1     1     0   5.2
 2 ATHENRY  2017     1     1     1   4.7
 3 ATHENRY  2017     1     1     2   4.2
 4 ATHENRY  2017     1     1     3   3.5
 5 ATHENRY  2017     1     1     4   3.2
 6 ATHENRY  2017     1     1     5   2.1
 7 ATHENRY  2017     1     1     6   2
 8 ATHENRY  2017     1     1     7   1.7
 9 ATHENRY  2017     1     1     8   1
10 ATHENRY  2017     1     1     9   1.1
# … with 218,990 more rows
```

Useful options with select()

```
> select(observations,station:rain)
# A tibble: 219,000 x 7
   station  year month   day  hour date                rain
   <chr>   <dbl> <dbl> <int> <int> <dttm>              <dbl>
 1 ATHENRY  2017     1     1     0 2017-01-01 00:00:00   0
 2 ATHENRY  2017     1     1     1 2017-01-01 01:00:00   0
 3 ATHENRY  2017     1     1     2 2017-01-01 02:00:00   0
 4 ATHENRY  2017     1     1     3 2017-01-01 03:00:00   0.1
 5 ATHENRY  2017     1     1     4 2017-01-01 04:00:00   0.1
 6 ATHENRY  2017     1     1     5 2017-01-01 05:00:00   0
 7 ATHENRY  2017     1     1     6 2017-01-01 06:00:00   0
 8 ATHENRY  2017     1     1     7 2017-01-01 07:00:00   0
 9 ATHENRY  2017     1     1     8 2017-01-01 08:00:00   0
10 ATHENRY  2017     1     1     9 2017-01-01 09:00:00   0
# … with 218,990 more rows
```

```
> select(observations,-(station:rain))
# A tibble: 219,000 x 5
    temp  rhum   msl  wdsp wddir
   <dbl> <dbl> <dbl> <dbl> <dbl>
 1   5.2    89 1022.     8   320
 2   4.7    89 1022      9   320
 3   4.2    90 1022.     8   320
 4   3.5    87 1022.     9   330
 5   3.2    89 1023.     8   330
 6   2.1    91 1023.     8   330
 7   2      89 1024.     7   330
 8   1.7    89 1024.     7   340
 9   1      91 1025      7   330
10   1.1    91 1026.     8   330
# … with 218,990 more rows
```

Special functions with select()

Special functions

As well as using existing functions like `:` and `c`, there are a number of special functions that only work inside `select`

- `starts_with(x, ignore.case = TRUE)`: names starts with `x`
- `ends_with(x, ignore.case = TRUE)`: names ends in `x`
- `contains(x, ignore.case = TRUE)`: selects all variables whose name contains `x`
- `matches(x, ignore.case = TRUE)`: selects all variables whose name matches the regular expression `x`
- `num_range("x", 1:5, width = 2)`: selects all variables (numerically) from x01 to x05.
- `one_of("x", "y", "z")`: selects variables provided in a character vector.
- `everything()`: selects all variables.

4. mutate()

- It is often useful to add new columns that are functions of existing columns
- mutate() always adds new columns at the end of your data set.
- For example, convert the mph wind speed in observations to a new column, kph.
- Use a simplified observations tibble with day, month, station, wdsp as columns, and for “ROCHES POINT” on October 16th
- Assume 1 mi = 1.609344 km

Example of mutate

```
library(aimsir17)
library(dplyr)

CM2K <- 1.609344

obs1 <- observations %>% select(day, month, station, wdsp) %>%
          filter(station=="ROCHES POINT", day==16,month==10)

obs1 <- mutate(obs1, wdsp_kph=wdsp*CM2K)
```

```
> obs1
# A tibble: 24 x 5
     day month station       wdsp wdsp_kph
   <int> <dbl> <chr>        <dbl>    <dbl>
 1    16    10 ROCHES POINT    11     17.7
 2    16    10 ROCHES POINT    11     17.7
 3    16    10 ROCHES POINT    14     22.5
 4    16    10 ROCHES POINT    15     24.1
 5    16    10 ROCHES POINT    22     35.4
```

Useful Creation Functions

- There are many functions for creating new variables that can be used with mutate()
- The key property is that the function **must be vectorised:**
  - It must take a vector of values as input, and,
  - Return a vector with the same number of values as output

Grouping	Examples
Arithmetic Operators	+, -, *, /, ^
Modular Arithmetic	%/% - Integer division && - Remainder
Logical comparisons	<, <=, >, >=, !=
If-else Functions	ifelse(), case_when()

5. summarise()

- The last key verb is summarise()
- It collapses a data frame into a single row
- Not very useful unless paired with group_by()
- Very useful to combine with the pipe operator

```
test <- filter(observations,
               station=="MACE HEAD",
               month==10)
```

group_by()

- Most data operations are useful done on groups defined by variables in the the dataset.
- The group_by function takes an existing tbl and converts it into a grouped tbl where operations are performed "by group".

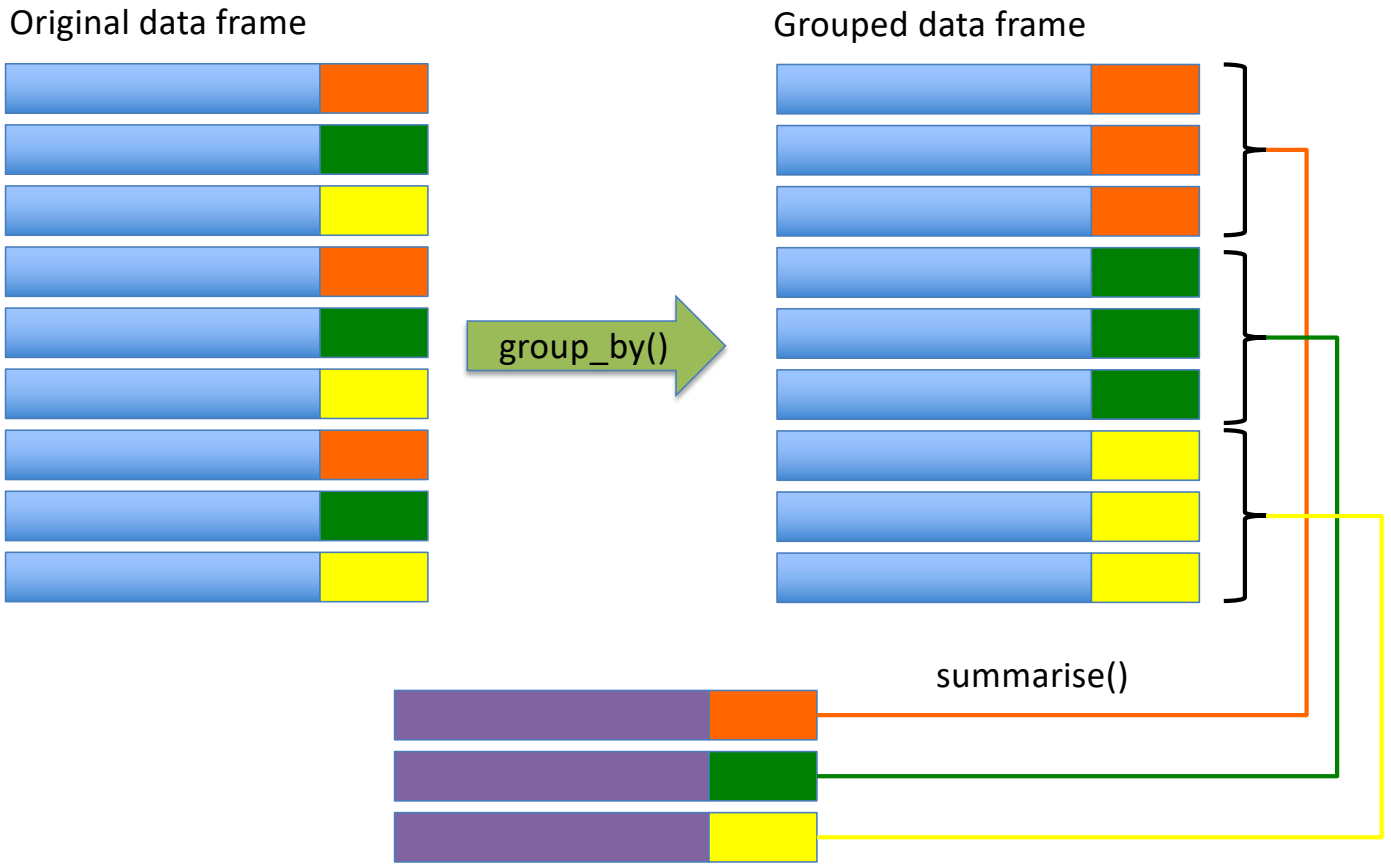
```
> test_g <- group_by(test,day)
> test_g
# A tibble: 744 x 12
# Groups:   day [31]
   station  year month   day  hour date                 rain  temp
   <chr>   <dbl> <dbl> <int> <int> <dttm>              <dbl> <dbl>
 1 MACE H… 2017    10     1     0 2017-10-01 00:00:00   1.2  11.7
 2 MACE H… 2017    10     1     1 2017-10-01 01:00:00   1.9  11.3
```

Key idea

- This grouping can then be exploited by summarise

```
> summarise(test_g,TotalRainfall=sum(rain,na.rm=T))
# A tibble: 31 x 2
     day TotalRainfall
   <int>         <dbl>
 1     1           4.7
 2     2           1.3
 3     3           0
 4     4           4.2
 5     5           0.1
 6     6           3.1
 7     7           0.4
 8     8           0.3
 9     9           1.7
10    10           1.6
# … with 21 more rows
```

Overall idea…

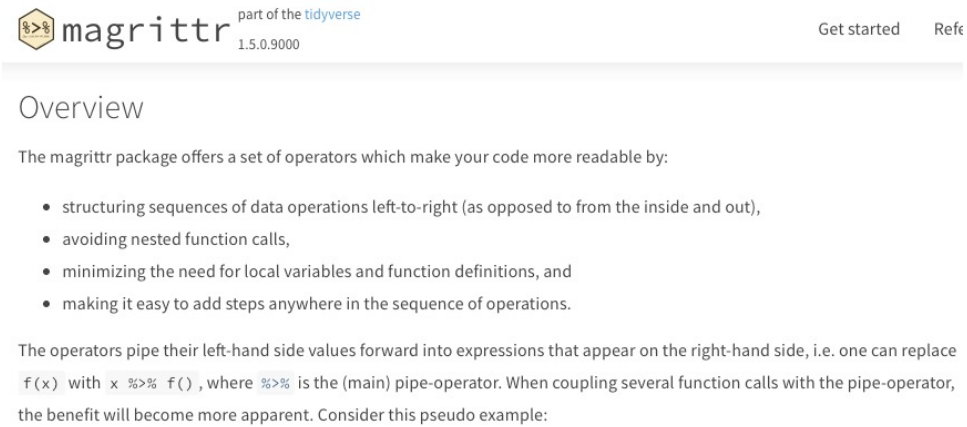


Useful Summary Functions

Grouping	Examples
Measures of location	mean(), median()
Measures of spread	sd(), IQR(),mad()
Measures of rank	min((), quantile(), max()
Measures of position	first(), nth(), last()
Counts	n(), n_distinct()
Counts and proportions of logical values	sum(x>0) when used with numeric functions, (T,F) converted to (1,0)

6. Combining operations with the Pipe

- The pipe %>% comes from the magrittr package (Stefan Milton Bache)
- Helps to write code that is easier to read and understand
- x %>% f(y) turns into f(x, y)
- x %>% f(y) %>% g(z) turns into g(f(x, y),z)



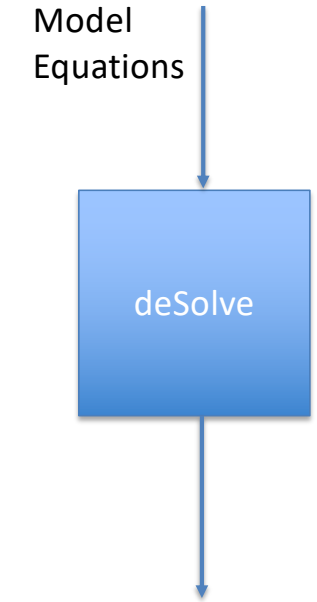
https://magrittr.tidyverse.org

```
> sqrt(1:5)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
> 1:5 %>% sqrt()
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

Examples

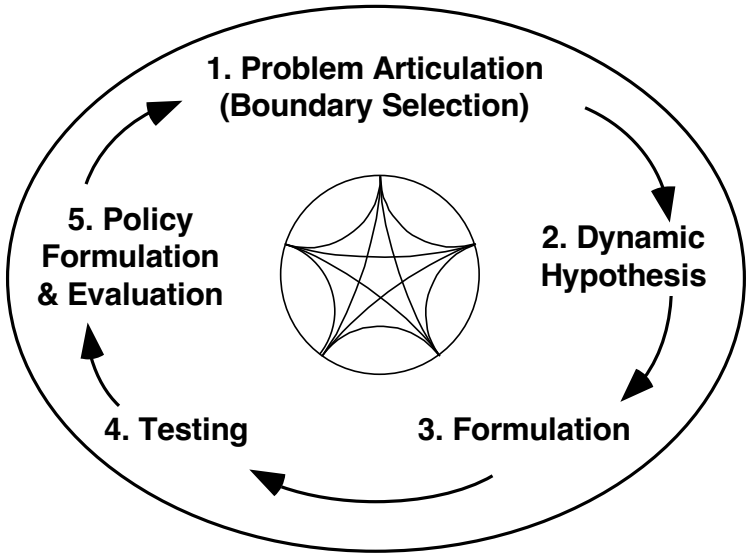
```
> observations %>% filter(day==1,station=="ATHENRY",hour==12,month==1)
# A tibble: 1 x 12
  station  year month   day  hour date                rain  temp  rhum   msl
  <chr>   <dbl> <dbl> <int> <int> <dttm>             <dbl> <dbl> <dbl> <dbl>
1 ATHENRY  2017     1     1    12 2017-01-01 12:00:00     0   5.1    75 1027.
# … with 2 more variables: wdsp <dbl>, wddir <dbl>
> observations %>% filter(station=="MACE HEAD") %>% arrange(desc(temp)) %>% head()
# A tibble: 6 x 12
  station  year month   day  hour date                rain  temp  rhum   msl
  <chr>   <dbl> <dbl> <int> <int> <dttm>             <dbl> <dbl> <dbl> <dbl>
1 MACE H…  2017     6    20    17 2017-06-20 17:00:00     0  22.7    69 1015.
2 MACE H…  2017     6    20    16 2017-06-20 16:00:00     0  22.6    67 1016.
3 MACE H…  2017     6    20    18 2017-06-20 18:00:00     0  22.3    71 1015.
4 MACE H…  2017     7    18    16 2017-07-18 16:00:00     0  22.3    61 1008.
5 MACE H…  2017     7    18    18 2017-07-18 18:00:00     0  22.2    65 1007.
6 MACE H…  2017     6    20    15 2017-06-20 15:00:00     0  22.1    68 1017.
# … with 2 more variables: wdsp <dbl>, wddir <dbl>
```

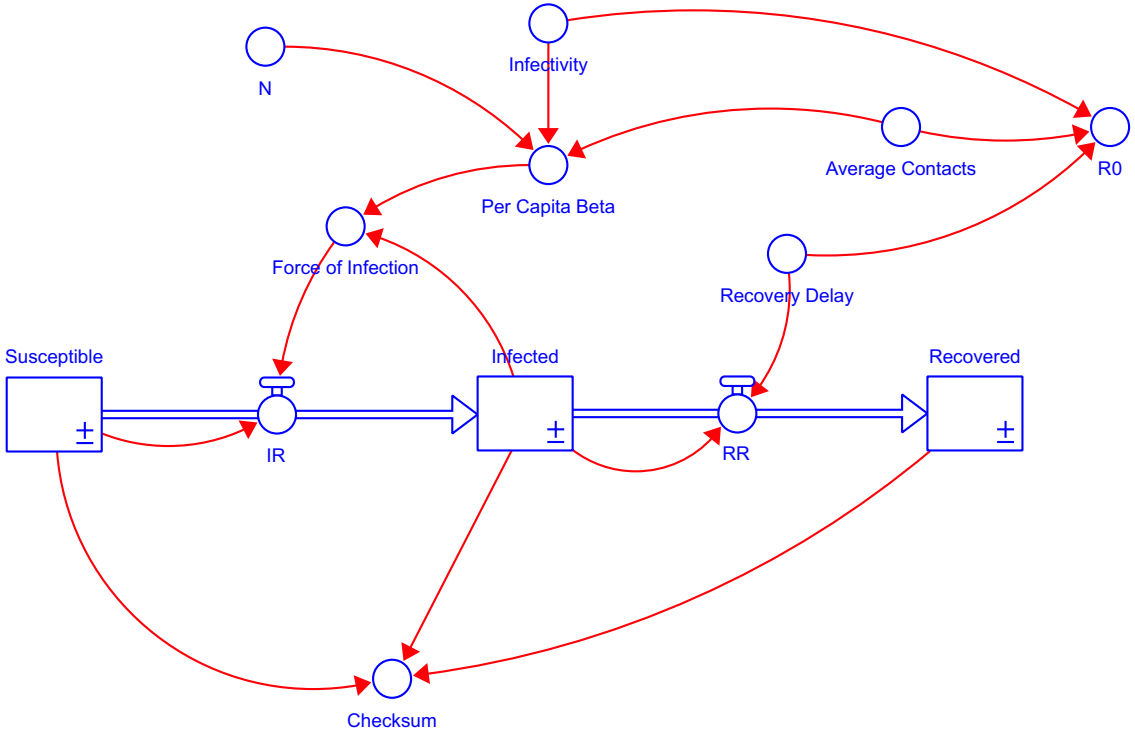
(2.1) Using deSolve – model as a function



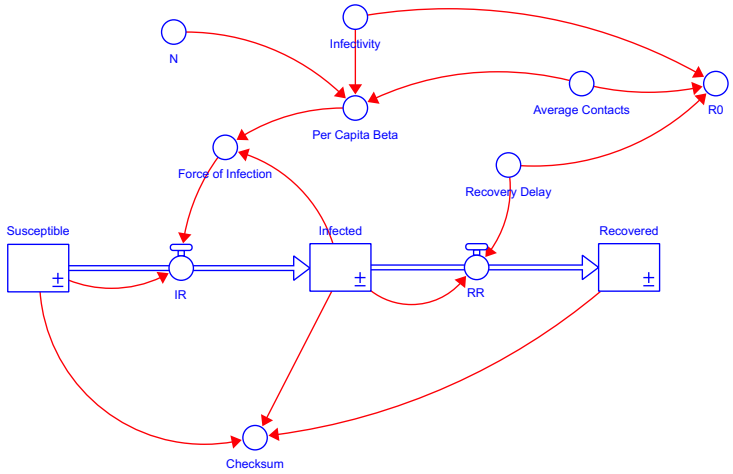
Argument	Description
`y`	The initial (state) values for the ODE system, a vector. If *y* has a name attribute, the names will be used to label the output matrix
`times`	Time sequence for which output is wanted; the first value of times must be the initial time
`func`	• An R function that computes the 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()`

Sample model





```r
sir_model <- function (time, stocks, auxs){
  with(as.list(c(stocks, auxs)), {
    Per_Capita_Beta <- Average_Contacts * Infectivity/N
    R0 <- Infectivity * Average_Contacts * Recovery_Delay
    Checksum <- Susceptible + Infected + Recovered
    Force_of_Infection <- Per_Capita_Beta * Infected

    IR <- Susceptible * Force_of_Infection
    RR <- Infected/Recovery_Delay

    d_Susceptible_dt <- -IR
    d_Infected_dt <- IR - RR
    d_Recovered_dt <- RR

    return(list(c(d_Susceptible_dt, d_Infected_dt, d_Recovered_dt),
                RR = RR, Per_Capita_Beta = Per_Capita_Beta, R0 = R0,
                Checksum = Checksum, Force_of_Infection = Force_of_Infection,
                IR = IR, Recovery_Delay = Recovery_Delay, N = N,
                Average_Contacts = Average_Contacts,
                Infectivity = Infectivity))
  })
}
```

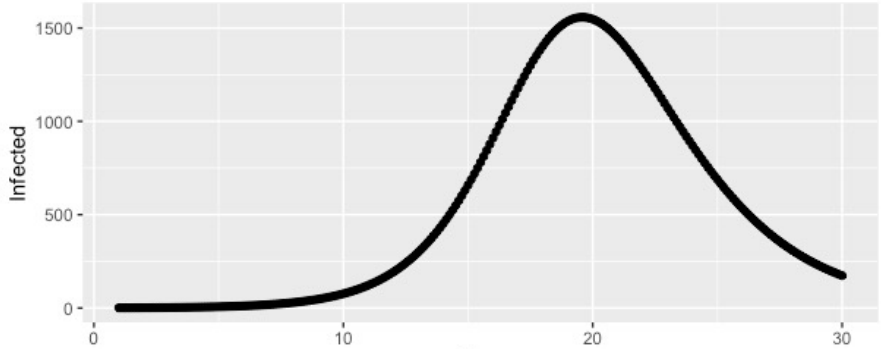


Calling the model, and initial conditions

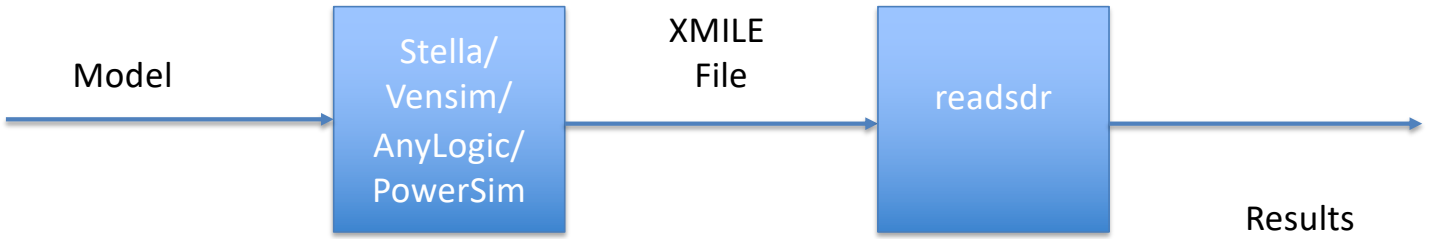
```
29  simtime <- seq(1,30,0.125)
30
31  stocks <- c(Susceptible=9999,
32              Infected=1,
33              Recovered=0)
34
35  params <- c(Recovery_Delay=2.0,
36              N = 10000,
37              Average_Contacts = 10.0,
38              Infectivity = 0.1)
39
40  results <- tibble(data.frame(ode(y      = stocks,
41                        times  = simtime,
42                        func   = sir_model,
43                        parms  = params,
44                        method = "euler")))
45
```

```
> results
# A tibble: 233 x 15
    time Susceptible Infected Recovered    RR Per_Capita_Beta    R0 Checksum
   <dbl>       <dbl>    <dbl>     <dbl> <dbl>           <dbl> <dbl>    <dbl>
 1  1          9999      1         0     0.5            0.0001     2   10000
 2  1.12       9998.     1.06      0.0625 0.531         0.0001     2   10000
 3  1.25       9998.     1.13      0.129  0.564         0.0001     2   10000.
 4  1.38       9998.     1.20      0.199  0.600         0.0001     2   10000.
 5  1.5        9998.     1.27      0.274  0.637         0.0001     2   10000.
 6  1.62       9998.     1.35      0.354  0.677         0.0001     2   10000.
 7  1.75       9998.     1.44      0.439  0.719         0.0001     2   10000
 8  1.88       9997.     1.53      0.529  0.764         0.0001     2   10000.
 9  2          9997.     1.62      0.624  0.812         0.0001     2   10000.
10  2.12       9997.     1.73      0.726  0.863         0.0001     2   10000.
# … with 223 more rows, and 7 more variables: Force_of_Infection <dbl>, IR <dbl>,
#   Recovery_Delay <dbl>, N <dbl>, Average_Contacts <dbl>, Infectivity <dbl>,
#   INIT_Susceptible <dbl>
```

```
ggplot(results,aes(x=time,y=Infected))+
  geom_point()+geom_line()
```



(2.2) Using readsdr package



Package 'readsdr'

January 8, 2021

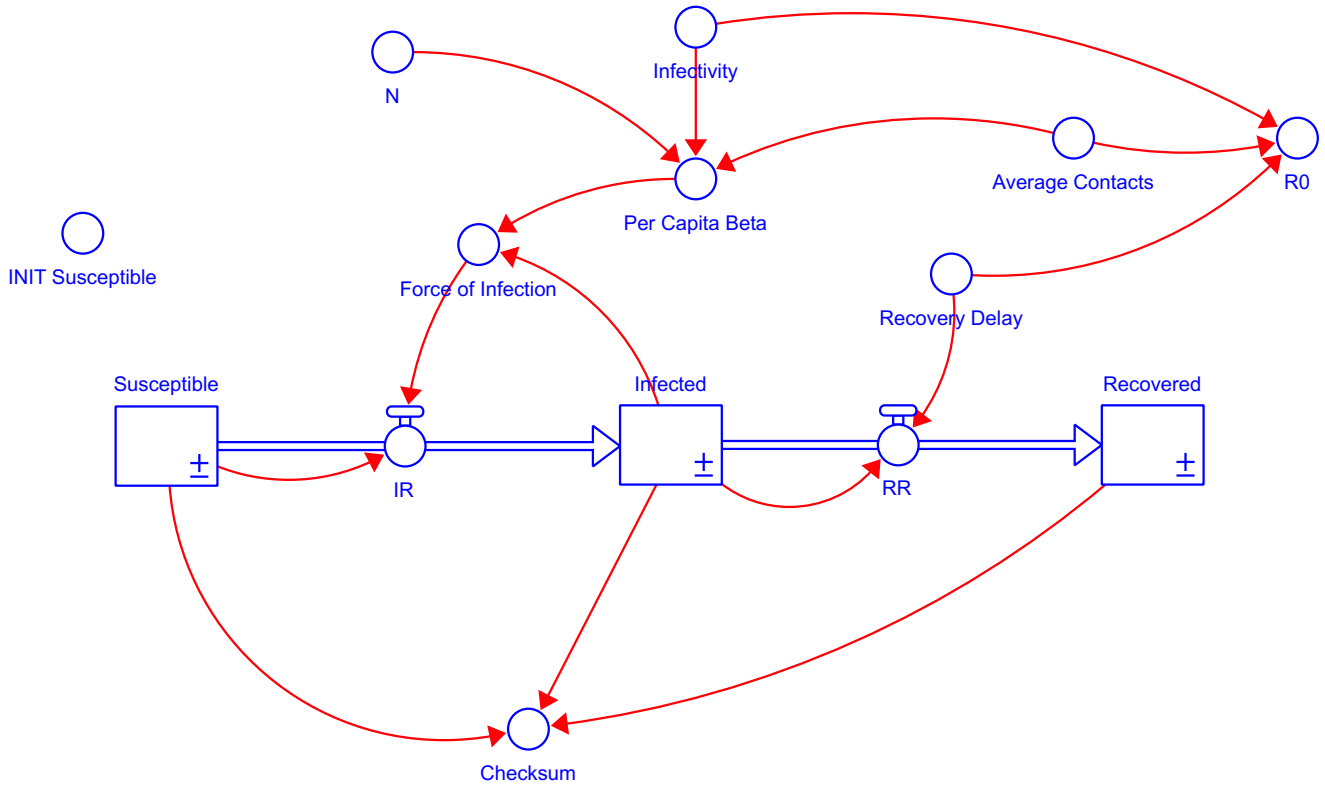
Type Package

Title Translate Models from System Dynamics Software into 'R'

Version 0.2.0

Description The goal of 'readsdr' is to bridge the design capabilities from specialised System Dynamics software with the powerful numerical tools offered by 'R' libraries. The package accomplishes this goal by parsing 'XMILE' files ('Vensim' and 'Stella') models into 'R' objects to construct networks (graph theory); 'ODE' functions for 'Stan'; and inputs to simulate via 'deSolve' as described in Duggan (2016) <doi:10.1007/978-3-319-34043-2>.

Model file


N
Infectivity
Average Contacts
R0
Per Capita Beta
INIT Susceptible
Force of Infection
Recovery Delay
Susceptible
Infected
Recovered
IR
RR
Checksum

Read in the Stella/XMILE file

```
3  library(readsdr)
4  library(deSolve)
5  library(dplyr)
6
7  filepath <- "workshops/03 Southampton 2021/Models/02 SIR deSolve/SIR.stmx"
8  mdl      <- read_xmile(filepath)
```

mdl	list [3]	List of length 3
description	list [4]	List of length 4
parameters	list [3]	List of length 3
levels	list [3]	List of length 3
variables	list [6]	List of length 6
constants	list [5]	List of length 5
deSolve_components	list [4]	List of length 4
stocks	double [3]	9999 1 0
consts	double [5]	2e+00 1e+04 1e+01 1e-01 1e+04
func	function	function(time, stocks, auxs) { ... }
sim_params	list [3]	List of length 3
graph_dfs	list [2]	List of length 2
nodes	list [9 x 3] (S3: data.frame)	A data.frame with 9 rows and 3 columns
edges	list [12 x 3] (S3: data.frame)	A data.frame with 12 rows and 3 columns

deSolve_components

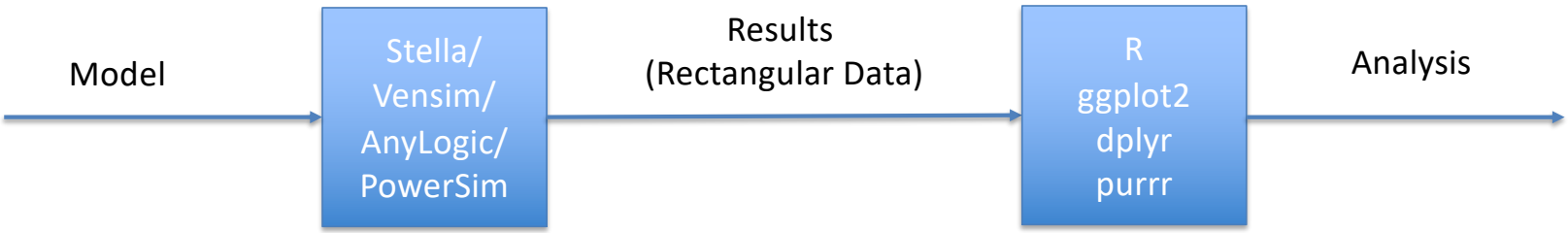
deSolve_components	list [4]	List of length 4
stocks	double [3]	9999 1 0
Susceptible	double [1]	9999
Infected	double [1]	1
Recovered	double [1]	0
consts	double [5]	2e+00 1e+04 1e+01 1e-01 1e+04
Recovery_Delay	double [1]	2
N	double [1]	10000
Average_Contacts	double [1]	10
Infectivity	double [1]	0.1
INIT_Susceptible	double [1]	9999
func	function	function(time, stocks, auxs) { ... }
formals	pairlist [3]	Pairlist of length 3
body	language	with(as.list(c(stocks, auxs)), { RR <- Infected/Recovery_Delay Per_Capit ...
environment	environment [13]	<environment: 0x7fa02463eae8>
sim_params	list [3]	List of length 3
start	double [1]	1
stop	double [1]	30
dt	double [1]	0.125

Running the model

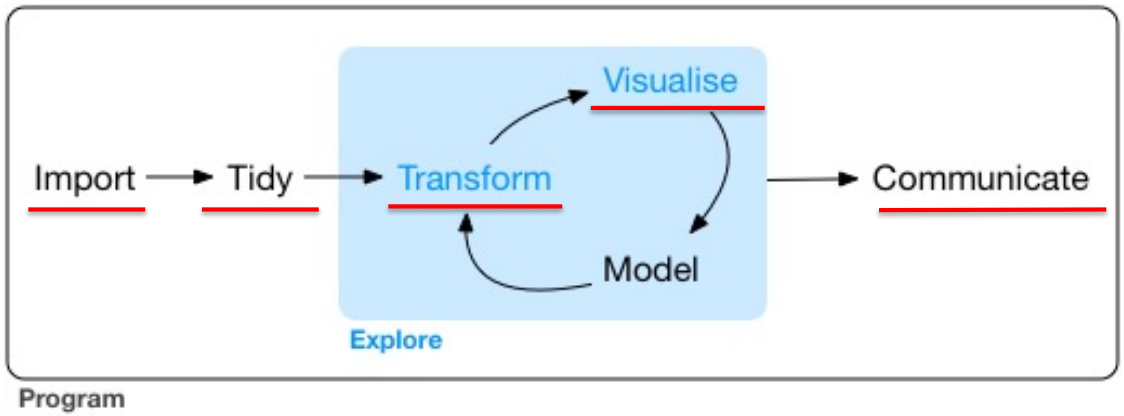
```
simtime <- seq(mdl$deSolve_components$sim_params$start,
               mdl$deSolve_components$sim_params$stop,
               mdl$deSolve_components$sim_params$dt)

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

out <- data.frame(output_deSolve)

results <- tibble(out)
```

```
> results
# A tibble: 233 x 15
    time Susceptible Infected Recovered    RR Per_Capita_Beta    R0 Checksum
   <dbl>       <dbl>    <dbl>     <dbl> <dbl>           <dbl> <dbl>    <dbl>
 1  1          9999      1       0      0.5            0.0001     2   10000
 2  1.12       9998.     1.06    0.0625 0.531          0.0001     2   10000
 3  1.25       9998.     1.13    0.129  0.564          0.0001     2   10000.
 4  1.38       9998.     1.20    0.199  0.600          0.0001     2   10000.
 5  1.5        9998.     1.27    0.274  0.637          0.0001     2   10000.
 6  1.62       9998.     1.35    0.354  0.677          0.0001     2   10000.
 7  1.75       9998.     1.44    0.439  0.719          0.0001     2   10000
 8  1.88       9997.     1.53    0.529  0.764          0.0001     2   10000.
 9  2          9997.     1.62    0.624  0.812          0.0001     2   10000.
10  2.12       9997.     1.73    0.726  0.863          0.0001     2   10000.
# … with 223 more rows, and 7 more variables: Force_of_Infection <dbl>, IR <dbl>,
#   Recovery_Delay <dbl>, N <dbl>, Average_Contacts <dbl>, Infectivity <dbl>,
#   INIT_Susceptible <dbl>
```

(2.3) Processing Sensitivity Results



- Extended SIR model (with vaccination)
- Use purrr libraries



Introduction to purrr

http://www.rebeccabarter.com/blog/2019-08-19_purrr/

- purrr is all about iteration.
- purrr introduces map functions (the tidyverse's answer to base R's apply functions, but more in line with functional programming practices) as well as some new functions for manipulating lists

- While the workhorse of dplyr is the data frame, the workhorse of purrr is the list.
  - A **vector** is a way of storing many individual elements (a single number or a single character or string) of the same type together in a single object,
  - A **data frame** is a way of storing many vectors of the same length but possibly of different types together in a single object
  - A **list** is a way of storing many objects of any type (e.g. data frames, plots, vectors) together in a single object

(1) Map functions – beyond apply

- 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

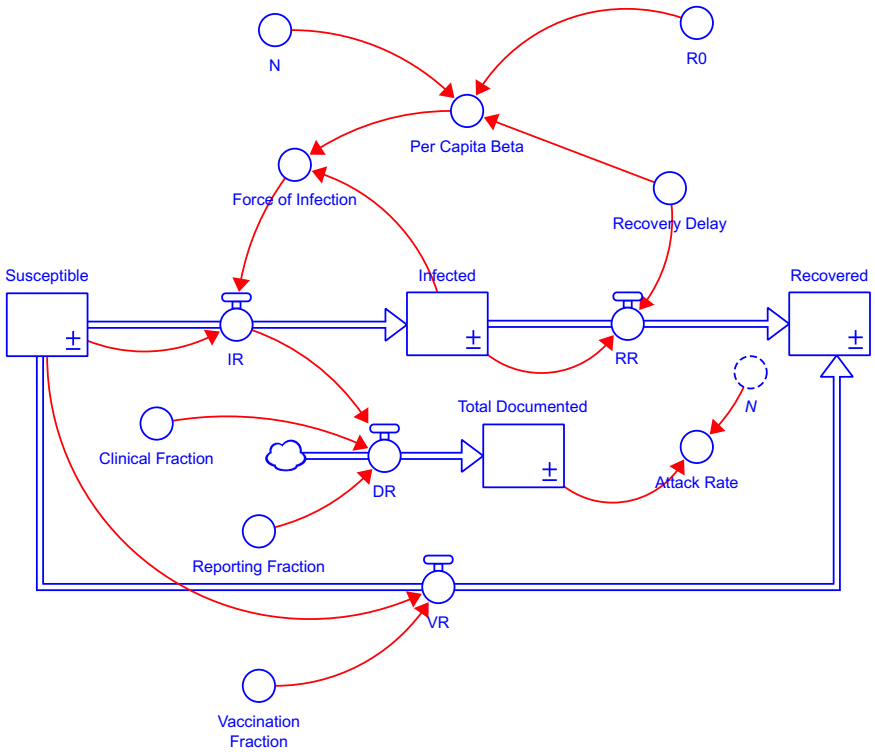
- `map(.x, .f)` is the main mapping function and returns a list
- `map_df(.x, .f)` returns a data frame
- `map_dbl(.x, .f)` returns a numeric (double) vector
- `map_chr(.x, .f)` returns a character vector
- `map_lgl(.x, .f)` returns a logical vector

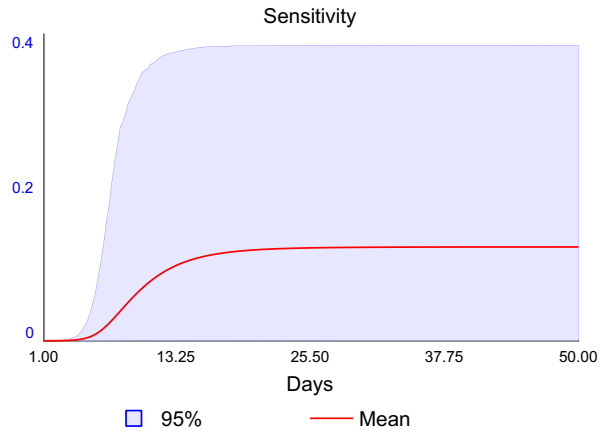
The tilde-dot shorthand for functions in map

- To make the code more concise you can use the tilde-dot shorthand for anonymous functions (the functions that you create as arguments of other functions).
- ~ indicates that you have started an anonymous function, and the argument of the anonymous function can be referred to using **.x** (or simply .).
- Unlike normal function arguments that can be anything that you like, ***the tilde-dot function argument is always .x.***

```
> map_dbl(c(1,4,7),~.x+10)
[1] 11 14 17
>
> map_dbl(c(1,4,7),~{.x+10})
[1] 11 14 17
>
> map_dbl(c(1,4,7),~.+10)
[1] 11 14 17
```

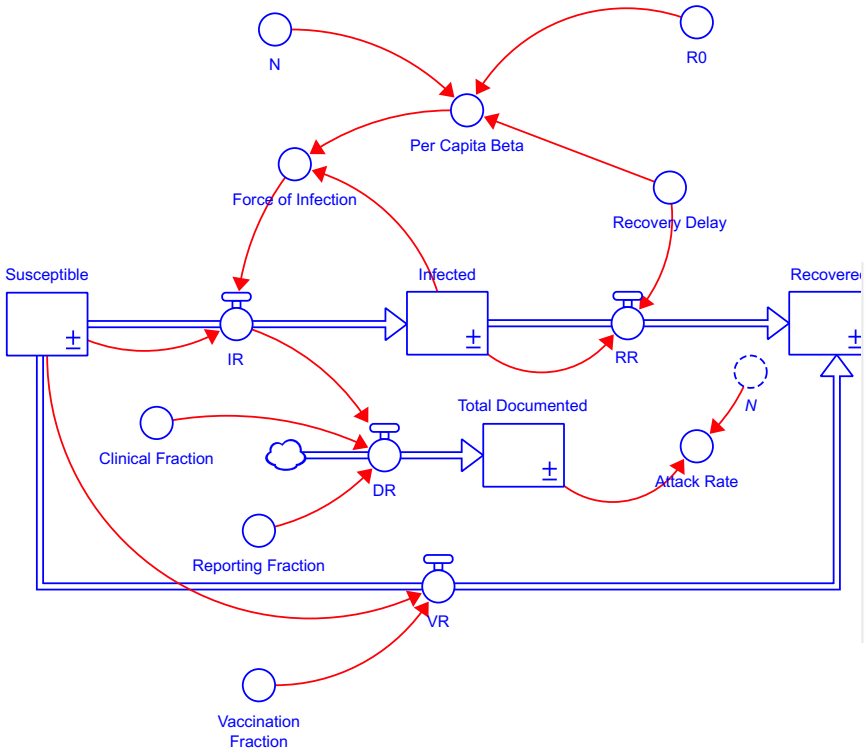
Sample Model (Stella) and Output

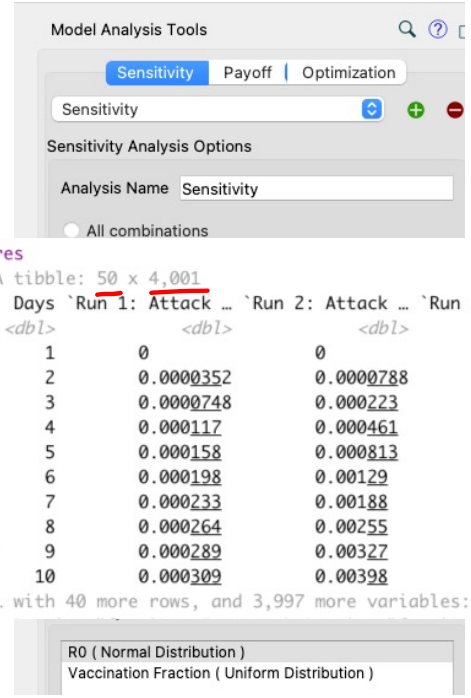




Days	Run 1: Attack Rate	Run 2: Attack Rate	Run 3: Attack Rate	Run 4: Attack Rate	Run 5: Attack Rate
1	0	0	0	0	0
2	3.52017E-05	7.88358E-05	5.62767E-05	8.41314E-05	0.000100987
3	7.48378E-05	0.000223469	0.000151448	0.000261265	0.000337615
4	0.000116714	0.00046095	0.000311716	0.000614574	0.000848201
5	0.000158448	0.000813046	0.000580409	0.001282961	0.001865558
6	0.000197874	0.001288277	0.001028714	0.002482835	0.003740133
7	0.000233343	0.001876634	0.001772639	0.004525767	0.006932958
8	0.000263855	0.002549509	0.002999163	0.00781852	0.011946385
9	0.00028905	0.003265129	0.005004942	0.012824492	0.019168741
10	0.000309092	0.003977398	0.008249615	0.019964553	0.028656342
11	0.000324501	0.004644898	0.013419388	0.029454549	0.039960365
12	0.000335984	0.005237266	0.021478781	0.041127013	0.052151605
13	0.000344302	0.005737757	0.033651523	0.054343335	0.06409591
14	0.000350173	0.006142347	0.051220981	0.068098712	0.074829072

Explore relationship between R0, VF for DR






```
> res
# A tibble: 50 x 4,001
   Days `Run 1: Attack … `Run 2: Attack … `Run 3: Attack …
  <dbl>            <dbl>            <dbl>            <dbl>
1     1        0                0                0
2     2        0.0000352        0.0000788        0.0000563
3     3        0.0000748        0.000223         0.000151
4     4        0.000117         0.000461         0.000312
5     5        0.000158         0.000813         0.000580
6     6        0.000198         0.00129          0.00103
7     7        0.000233         0.00188          0.00177
8     8        0.000264         0.00255          0.00300
9     9        0.000289         0.00327          0.00500
10    10       0.000309         0.00398          0.00825
# … with 40 more rows, and 3,997 more variables: `Run 4: Attack
```

Days	Run 1: Attack Rate	Run 2: Attack Rate	Run 3: Attack Rate	Ru
1	0	0	0	
2	3.52017E-05	7.88358E-05	5.62767E-05	
3	7.48378E-05	0.000223469	0.000151448	
4	0.000116714	0.00046095	0.000311716	
5	0.000158448	0.000813046	0.000580409	
6	0.000197874	0.001288277	0.001028714	
7	0.000233343	0.001876634	0.001772639	
8	0.000263855	0.002549509	0.002999163	
9	0.00028905	0.003265129	0.005004942	
	09092	0.003977398	0.008249615	
	24501	0.004644898	0.013419388	
	35984	0.005237266	0.021478781	
	44302	0.005737757	0.033651523	
	50173	0.006142347	0.051220981	

Data Input (Excel into a tibble)

Days	Run 1: Attack Rate	Run 2: Attack Rate	Run 3: Attack Rate	Ru
1	0	0	0	
2	3.52017E-05	7.88358E-05	5.62767E-05	
3	7.48378E-05	0.000223469	0.000151448	
4	0.000116714	0.00046095	0.000311716	
5	0.000158448	0.000813046	0.000580409	
6	0.000197874	0.001288277	0.001028714	
7	0.000233343	0.001876634	0.001772639	
8	0.000263855	0.002549509	0.002999163	
9	0.00028905	0.003265129	0.005004942	
10	0.000309092	0.003977398	0.008249615	
11	0.000324501	0.004644898	0.013419388	
12	0.000335984	0.005237266	0.021478781	
13	0.000344302	0.005737757	0.033651523	
14	0.000350173	0.006142347	0.051220981	

```
> res
# A tibble: 50 x 4,001
    Days `Run 1: Attack … `Run 2: Attack … `Run 3: Attack …
   <dbl>            <dbl>            <dbl>            <dbl>
 1     1        0                0                0
 2     2        0.0000352        0.0000788        0.0000563
 3     3        0.0000748        0.000223         0.000151
 4     4        0.000117         0.000461         0.000312
 5     5        0.000158         0.000813         0.000580
 6     6        0.000198         0.00129          0.00103
 7     7        0.000233         0.00188          0.00177
 8     8        0.000264         0.00255          0.00300
 9     9        0.000289         0.00327          0.00500
10    10        0.000309         0.00398          0.00825
# … with 40 more rows, and 3,997 more variables: `Run 4: Attack
```

Extract the 3 variables

```
12  v1_tib <- res %>% select(Days,contains("R0")) %>%
13    pivot_longer(names_to="Temp",values_to="R0",-Days) %>%
14    mutate(RunNumber=as.integer(str_extract(Temp,"\\d+"))) %>%
15    select(-Temp) %>%
16    select(RunNumber,Days,everything())
```

```
> v1_tib
# A tibble: 50,000 x 3
   RunNumber  Days    R0
       <int> <dbl> <dbl>
 1         1     1  1.56
 2         2     1  2.73
 3         3     1  2.10
 4         4     1  2.78
 5         5     1  3.12
 6         6     1  3.08
 7         7     1  2.57
 8         8     1  3.07
 9         9     1  2.84
10        10     1  2.05
# … with 49,990 more rows
```

```
> v2_tib
# A tibble: 50,000 x 3
   RunNumber  Days      VF
       <int> <dbl>   <dbl>
 1         1     1 0.0950
 2         2     1 0.102
 3         3     1 0.00398
 4         4     1 0.0451
 5         5     1 0.0637
 6         6     1 0.0391
 7         7     1 0.140
 8         8     1 0.0827
 9         9     1 0.110
10        10     1 0.134
# … with 49,990 more rows
```

```
> v3_tib
# A tibble: 50,000 x 3
   RunNumber  Days    DR
       <int> <dbl> <dbl>
 1         1     1 0.328
 2         2     1 0.573
 3         3     1 0.442
 4         4     1 0.583
 5         5     1 0.656
 6         6     1 0.646
 7         7     1 0.539
 8         8     1 0.645
 9         9     1 0.596
10        10     1 0.430
# … with 49,990 more rows
```

Join into a new table and nest

```
> sim <- v1_tib
> sim <- full_join(sim,v2_tib)
Joining, by = c("RunNumber", "Days")
> sim <- full_join(sim,v3_tib)
Joining, by = c("RunNumber", "Days")
> sim
# A tibble: 50,000 x 5
   RunNumber  Days    R0      VF    DR
       <int> <dbl> <dbl>   <dbl> <dbl>
 1         1     1  1.56 0.0950  0.328
 2         2     1  2.73 0.102   0.573
 3         3     1  2.10 0.00398 0.442
 4         4     1  2.78 0.0451  0.583
 5         5     1  3.12 0.0637  0.656
 6         6     1  3.08 0.0391  0.646
 7         7     1  2.57 0.140   0.539
 8         8     1  3.07 0.0827  0.645
 9         9     1  2.84 0.110   0.596
10        10     1  2.05 0.134   0.430
# … with 49,990 more rows
```

```
> sim_nested <- sim %>%
+                 group_by(RunNumber) %>%
+                 nest()
>
>
> sim_nested
# A tibble: 1,000 x 2
# Groups:   RunNumber [1,000]
   RunNumber data
       <int> <list>
 1         1 <tibble [50 × 4]>
 2         2 <tibble [50 × 4]>
 3         3 <tibble [50 × 4]>
 4         4 <tibble [50 × 4]>
 5         5 <tibble [50 × 4]>
 6         6 <tibble [50 × 4]>
 7         7 <tibble [50 × 4]>
 8         8 <tibble [50 × 4]>
 9         9 <tibble [50 × 4]>
10        10 <tibble [50 × 4]>
# … with 990 more rows
```

Get the results for each simulation

```
summ <- sim_nested %>%
            pull(data) %>%
                 map_df(~{
                   tibble(Max_DR=max(.$DR),
                          R0=first(.$R0),
                          VF=first(.$VF))
})
```

```
> summ
# A tibble: 1,000 x 3
    Max_DR    R0       VF
     <dbl> <dbl>    <dbl>
 1   0.422  1.56 0.0950
 2   7.21   2.73 0.102
 3 346.     2.10 0.00398
 4 137.     2.78 0.0451
 5 122.     3.12 0.0637
 6 305.     3.08 0.0391
 7   2.09   2.57 0.140
 8  42.9    3.07 0.0827
 9   7.55   2.84 0.110
10   0.803  2.05 0.134
# … with 990 more rows
```

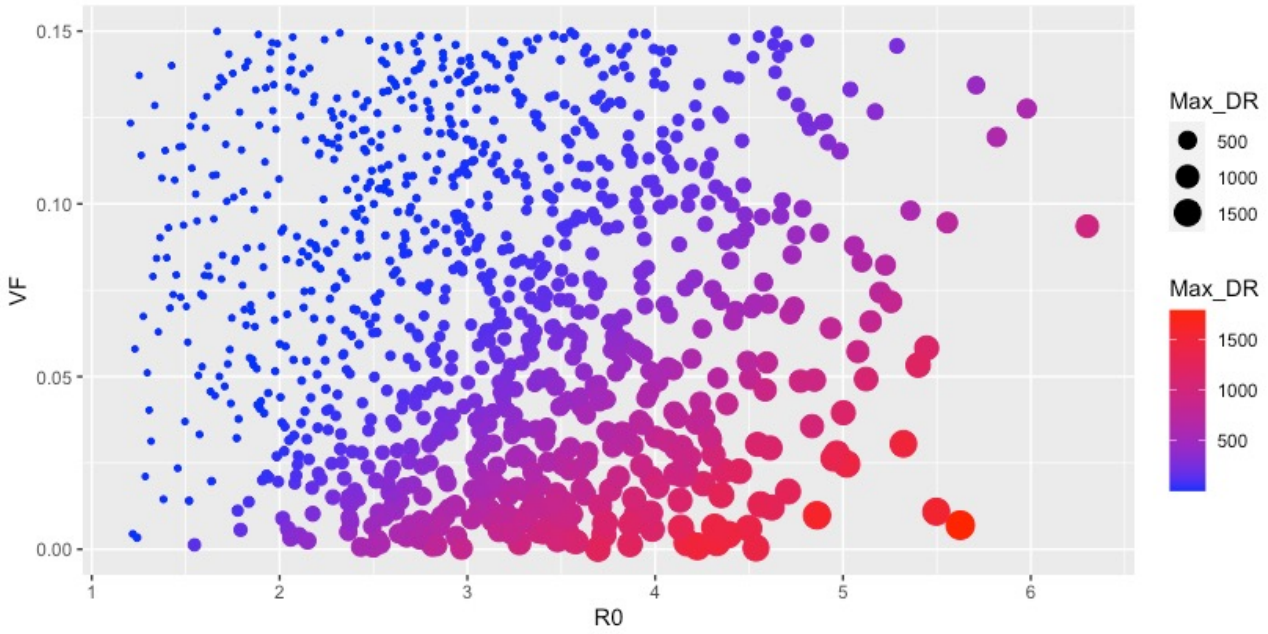
Merge Back into the original table.

```
> sim_nested
# A tibble: 1,000 x 2
# Groups:   RunNumber [1,000]
   RunNumber data
       <int> <list>
 1         1 <tibble [50 × 4]>
 2         2 <tibble [50 × 4]>
 3         3 <tibble [50 × 4]>
 4         4 <tibble [50 × 4]>
 5         5 <tibble [50 × 4]>
 6         6 <tibble [50 × 4]>
 7         7 <tibble [50 × 4]>
 8         8 <tibble [50 × 4]>
 9         9 <tibble [50 × 4]>
10        10 <tibble [50 × 4]>
# … with 990 more rows
```

```
> sim_nested <- bind_cols(sim_nested, summ)
>
> sim_nested
# A tibble: 1,000 x 5
# Groups:   RunNumber [1,000]
   RunNumber data                Max_DR    R0      VF
       <int> <list>                <dbl> <dbl>   <dbl>
 1         1 <tibble [50 × 4]>   0.422   1.56 0.0950
 2         2 <tibble [50 × 4]>   7.21    2.73 0.102
 3         3 <tibble [50 × 4]> 346.      2.10 0.00398
 4         4 <tibble [50 × 4]> 137.      2.78 0.0451
 5         5 <tibble [50 × 4]> 122.      3.12 0.0637
 6         6 <tibble [50 × 4]> 305.      3.08 0.0391
 7         7 <tibble [50 × 4]>   2.09    2.57 0.140
 8         8 <tibble [50 × 4]>  42.9     3.07 0.0827
 9         9 <tibble [50 × 4]>   7.55    2.84 0.110
10        10 <tibble [50 × 4]>   0.803   2.05 0.134
# … with 990 more rows
```

Plot the Exploratory Analysis

```
51  ggplot(sim_nested,aes(x=R0,y=VF,size=Max_DR,colour=Max_DR))+
52    geom_point()+
53    scale_color_gradient(low="blue", high="red")
```



Summary

- Introduction to R and the tidyverse
  - Visualisation - **ggplot2**
  - Transformation – **dplyr** and **purrr**
- Using R with System Dynamics, 3 Examples
  - **deSolve**
  - **readsdr**
  - Exploring sensitivity runs