Plotting model output with ggplot2

July 2019

Introduction

Tidyverse

- The *tidyverse* suite of R packages is designed to make working with data as easy as possible
- The relevant packages from tidyverse for us are
  - ggplot2: for plotting data
  - dplyr: for manipulating data frames
  - tidyr: for making data tidy

```
library(tidyverse)
```

Principles of tidy data

- Every data set has its own quirks
- Tidy **data frames** consist of a number of observations (rows) of variables (columns), they can be either **wide** or **long** (Wickham 2014)
- Data needs to be the right shape for the functions being used
- ggplot2 usually requires long data

Long and wide tidy data

- An example of a wide data frame which we might encounter is the output of an SIR model

time	S	I	R
0.00	0.9999990	1.0e-06	0e+00
0.05	0.9999989	1.1e-06	0e+00
0.10	0.9999988	1.1e-06	0e+00
0.15	0.9999988	1.2e-06	0e+00
0.20	0.9999987	1.3e-06	0e+00
0.25	0.9999986	1.4e-06	0e+00
0.30	0.9999985	1.5e-06	1e-07
0.35	0.9999984	1.6e-06	1e-07
0.40	0.9999983	1.7e-06	1e-07
0.45	0.9999981	1.8e-06	1e-07

- Here we have values of $S(t), I(t), R(t)$ at given values of t

Long and wide tidy data

- Our numerical solution to the SIR model is a wide data frame
- We reshape our data to be long

```
SIR_long <- gather(data  = SIR,
                   key   = state,
                   value = proportion,
                   S, I, R)
```

- This says *gather* the columns in `SIR` as *key-value* pairs where
  - the new *key* column is called `state` and contains the names of the gathered columns
  - the *value* of each `state` at a given `time` is in the column called `proportion`
  - the `S`, `I`, `R` columns (not `time`) are the gathered variables

Long and wide tidy data

Wide data

time	S	I	R
0.00	0.9999990	1.0e-06	0
0.05	0.9999989	1.1e-06	0
0.10	0.9999988	1.1e-06	0

key: this `state` at this `time`
value: `proportion`

Long data

time	state	proportion
0.00	S	0.9999990
0.05	S	0.9999989
0.10	S	0.9999988
0.00	I	0.0000010
0.05	I	0.0000011
0.10	I	0.0000011
0.00	R	0.0000000
0.05	R	0.0000000
0.10	R	0.0000000

Visualisation with the tidyverse

Grammar of graphics

- R package `ggplot2` uses a grammar of graphics (Wickham 2010)
    - adding extra commands in a “do this, then do this” manner
    - assign variables in data frame to aesthetic options in the plot
    - choose a plotting style for how to display these variables
    - adjustments to axis scales
    - adjustments to colors, themes, etc.
    - additional annotation
- Focus is on visual relationships between variables rather than drawing points and lines
- Options are properties of the elements of the plot rather than of plot itself

Building a plot

- How do we tell the `ggplot()` function to make a plot?
  - Load the ggplot2 package, which contains the `ggplot()` function
  - Specify a data frame to use, containing the variables we want to plot

```
library(ggplot2)
ggplot(data = my.data.frame)
```

Building a plot

- How do we tell the `ggplot()` function to make a plot?
  - Then we set some **aesthetic options** to tell R which variables from `my.data.frame` to map to the x and y axes of the plot

```
ggplot(data = my.data.frame,
       aes(x = my.x.variable,
           y = my.y.variable))
```

Building a plot

- How do we tell the `ggplot()` function to make a plot?
  - Geometries are the shapes we use to draw plots, e.g. lines, points, polygons, bars, boxplots
  - We will use the *line geom*etry to build a time series plot

```
ggplot(data = my.data.frame,
       aes(x = my.x.variable,
           y = my.y.variable)) +
  geom_line()
```

- We can set aesthetics `aes(...)` inside a geometry to modify the color, fill, alpha transparency, etc. according to a variable in the data frame

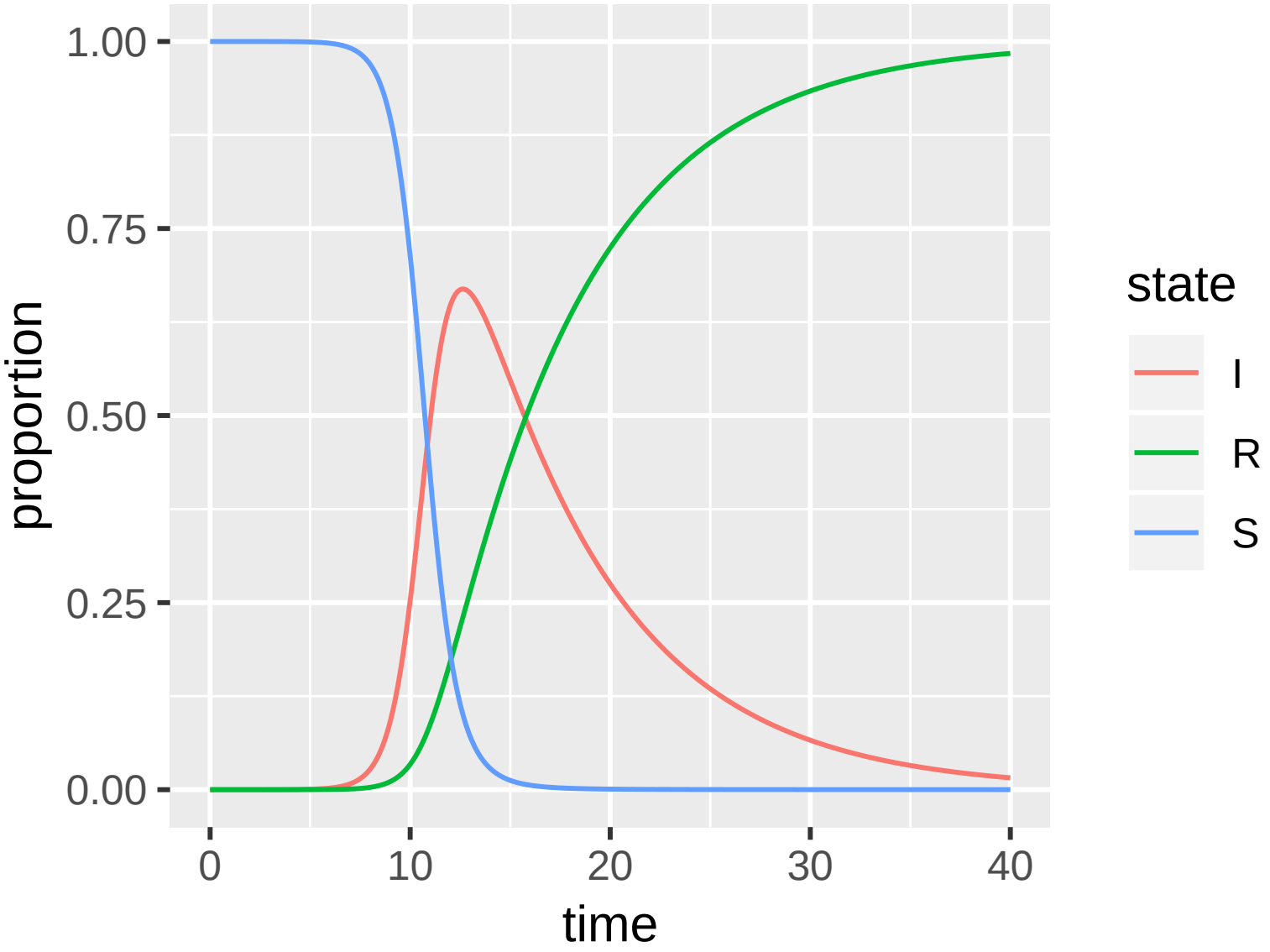
Using ggplot2

time	state	proportion
0.00	S	0.9999990
0.05	S	0.9999989
0.10	S	0.9999988
0.00	I	0.0000010
0.05	I	0.0000011
0.10	I	0.0000011
0.00	R	0.0000000
0.05	R	0.0000000
0.10	R	0.0000000

```
sir_ggplot <-
  ggplot(data = SIR_long,
         aes(x = time,
             y = proportion)) +
  geom_line(aes(color = state))
```

- For each `state`, we want to plot a different *line* with its own *colour*
- Line has `proportion` on y axis, `time` on x axis

Using `ggplot2`



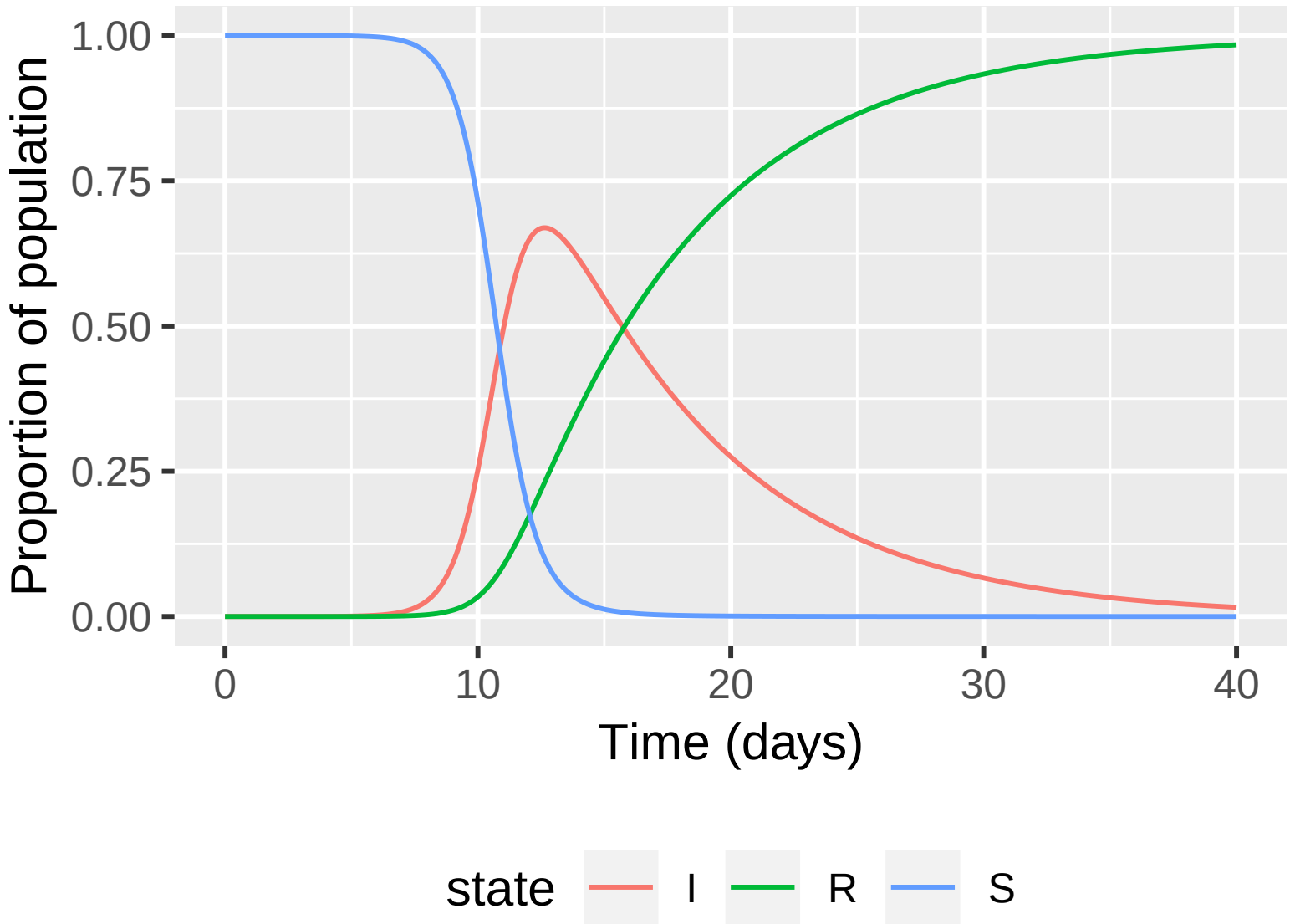
Using ggplot2

- This plot isn't especially pretty, so let's add some extra things to it (using + for our grammar of graphics)
  - `xlab()` and `ylab()` print their argument as axis labels
  - `theme()` sets options about the way the axis/background/legend annotations are drawn

```
sir_ggplot <- sir_ggplot +
  theme(legend.position = "bottom") +
  xlab("Time (days)") +
  ylab("Proportion of population")
```

- We are sequentially adding functions that modify the plot rather than passing arguments to a `plot()` to replace default options

Using ggplot2



Using ggplot2

- Consider a basic plot that we'll recycle

```
sir_ggplot_basic <-
  ggplot(data = SIR_long,                  # where data lives
         aes(x = time,                     # set plot aesthetics...
             y = proportion)) +            # ...specifying x&y vars
  theme_bw() +                             # grey grid on white bg
  xlab("Time (days)") +                    # replace time as x label
  ylab("Population proportion") +          # replace proportion as y
  theme(legend.position = "bottom")        # change legend placement
```

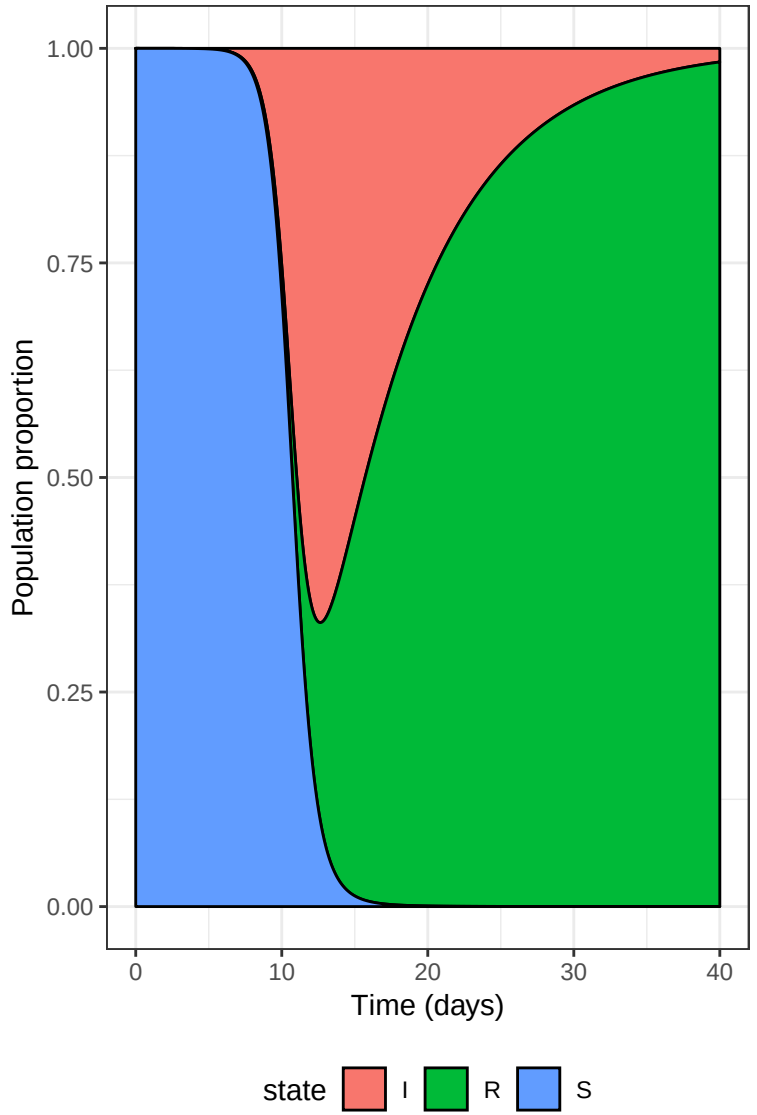
- NB no geometry specified
- `theme_bw()` is a collection of options for `theme()` that specify a white background with a light grey grid and black text

Using `ggplot2`

- `geom_area()` gives a *fill*ed polygon with a border *color*

```
sir_ggplot_area <-
  sir_ggplot_basic +
  geom_area(
    aes(fill = state),
    color = "black")
```

- Mapping a variable, e.g. `state`, to part of our plot requires it is inside `aes(...)`
- Static options go outside `aes(...)`

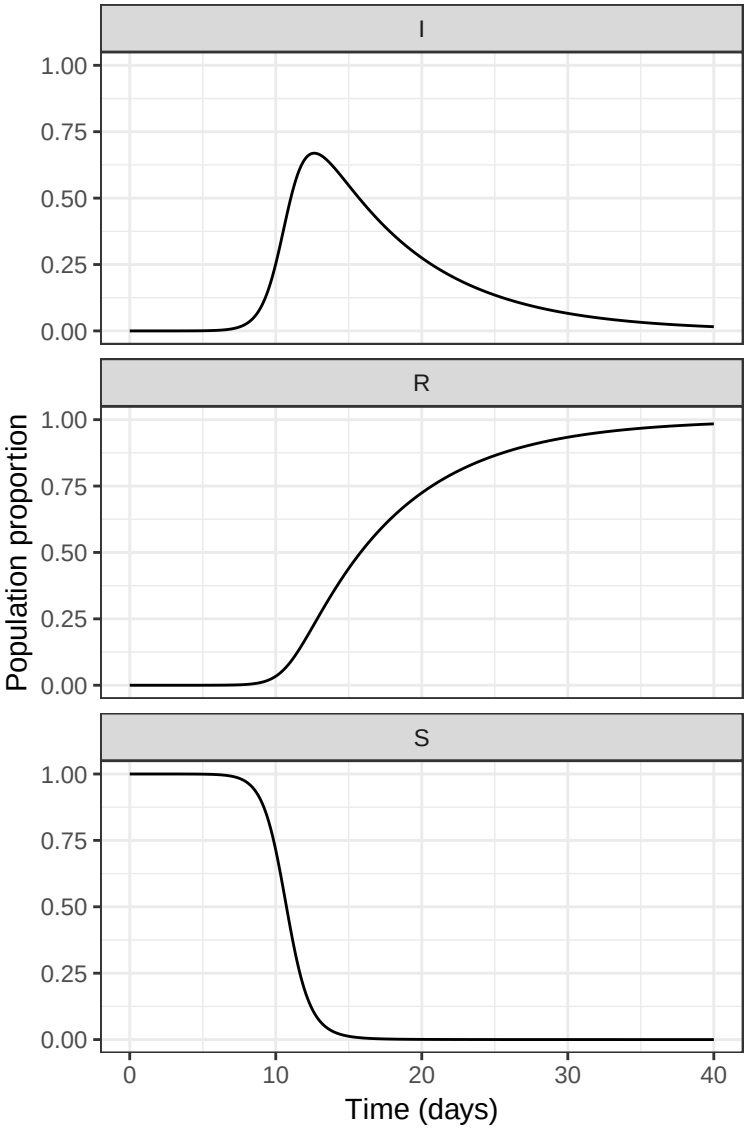


Using ggplot2

Draw small multiples with `facet_wrap()`, repeating the geometry for the levels of the grouping variable

```
sir_ggplot_facet <-
  sir_ggplot_basic +
  geom_line() +
  facet_wrap( ~ state,
              ncol = 1)
```

We indicate grouping by the `state` variable with `~ state`



Relevelling factors

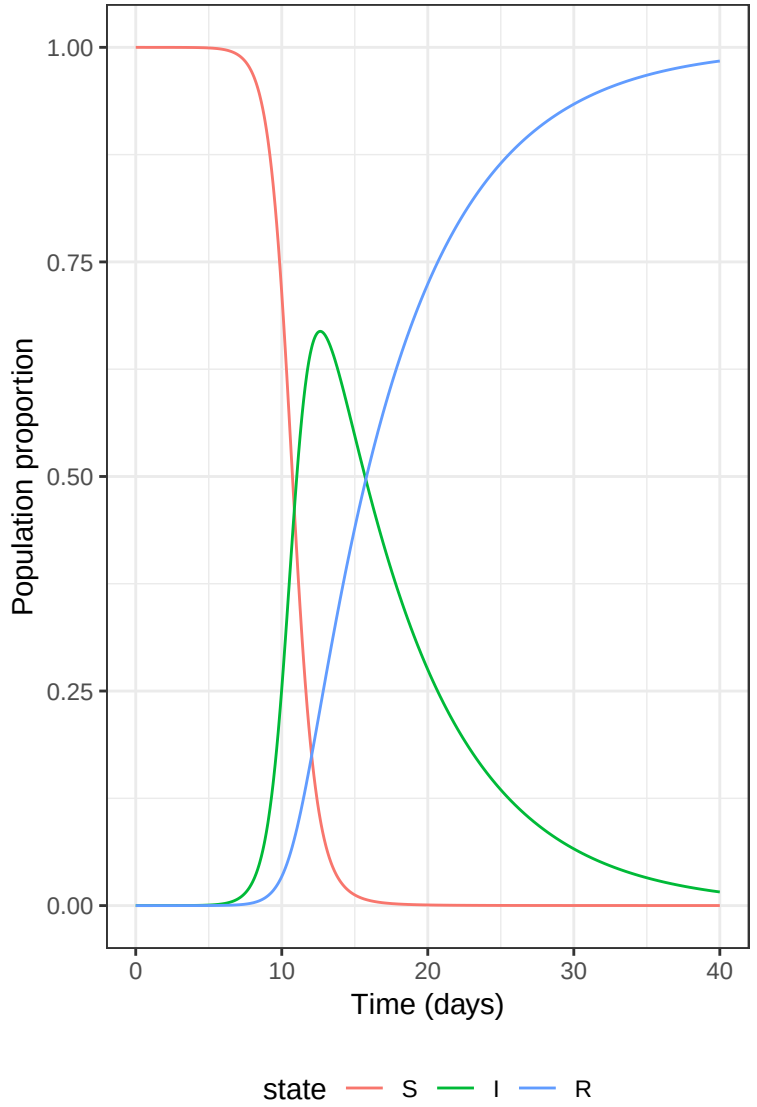
Relevelling factors

- Default behaviours are:
  - `gather()` respects column order when reshaping
  - `key` column is character variable
  - character variables coerced to alphabetic factors
- We can set order of `state` variable by specifying levels

```
factor(state, levels = c("S","I","R"))
```

Relevelling factors

```
SIR_long$state <-
  factor(SIR_long$state,
         levels = c("S",
                    "I",
                    "R"))

sir_ggplot_lines <-
  ggplot(data = SIR_long,
         aes(x = time,
             y = proportion)) +
  theme_bw() +
  xlab("Time (days)") +
  ylab("Population proportion")
  theme(
    legend.position = "bottom")
  geom_line(aes(color = state))
```



Increasing the complexity

Using `ggplot2`

File `all_sims_labelled_SIR.csv` contains 4 SIR simulations with each combination of $\beta = 1.42470, 1.56756$ and $\gamma = 0.14286, 0.36508$, representing different parameter values from a factorial design, e.g.

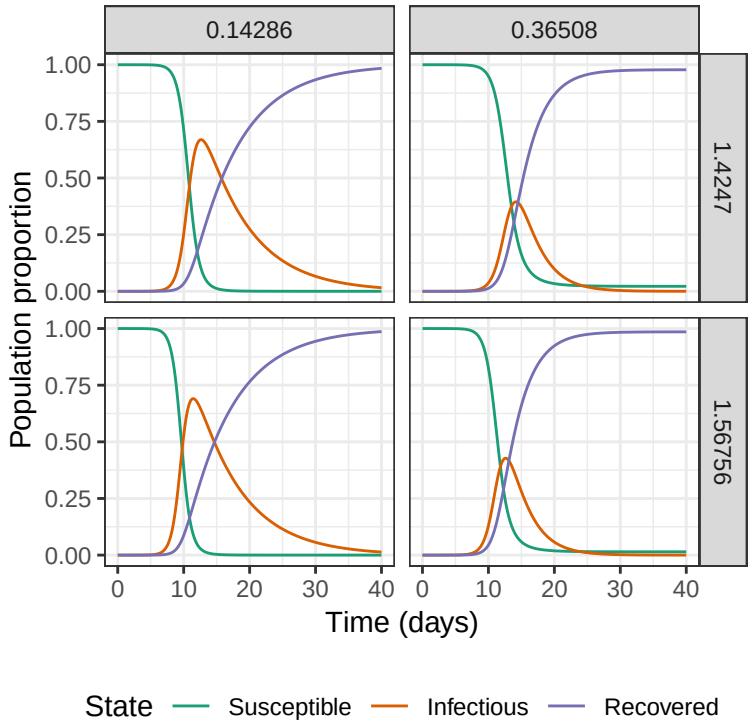
beta	gamma	time	proportion	state
1.4247	0.14286	0.2	0.9999987	Susceptible
1.4247	0.14286	0.2	0.0000013	Infectious
1.4247	0.14286	0.2	0.0000000	Recovered
1.4247	0.36508	0.2	0.9999987	Susceptible
1.4247	0.36508	0.2	0.0000012	Infectious
1.4247	0.36508	0.2	0.0000001	Recovered

Using `ggplot2`

- We want to specify a line plot for each β, γ and state
- Use `color = state` as before
- Want a 2×2 grid of β, γ
  - `facet_grid()` allows us to specify the grouping structure as `row_variables ~ column_variables`
  - For a different `beta` on each row and `gamma` in each column we specify `facet_grid(beta ~ gamma)`

Using ggplot2

```
SIR_plot_bg_basic <-
  ggplot(data =
      all_sims_labelled_SIR,
       aes(x = time,
           y = proportion)) +
  xlab("Time (days)") +
  ylab("Population proportion") +
  theme_bw() +
  theme(legend.position = "bottom")

SIR_plot_bg_grid <-
  SIR_plot_bg_basic +
  geom_line(aes(color = state)) +
  facet_grid(beta ~ gamma) +
  scale_color_brewer(
    palette = "Dark2",
    name = "State")
```

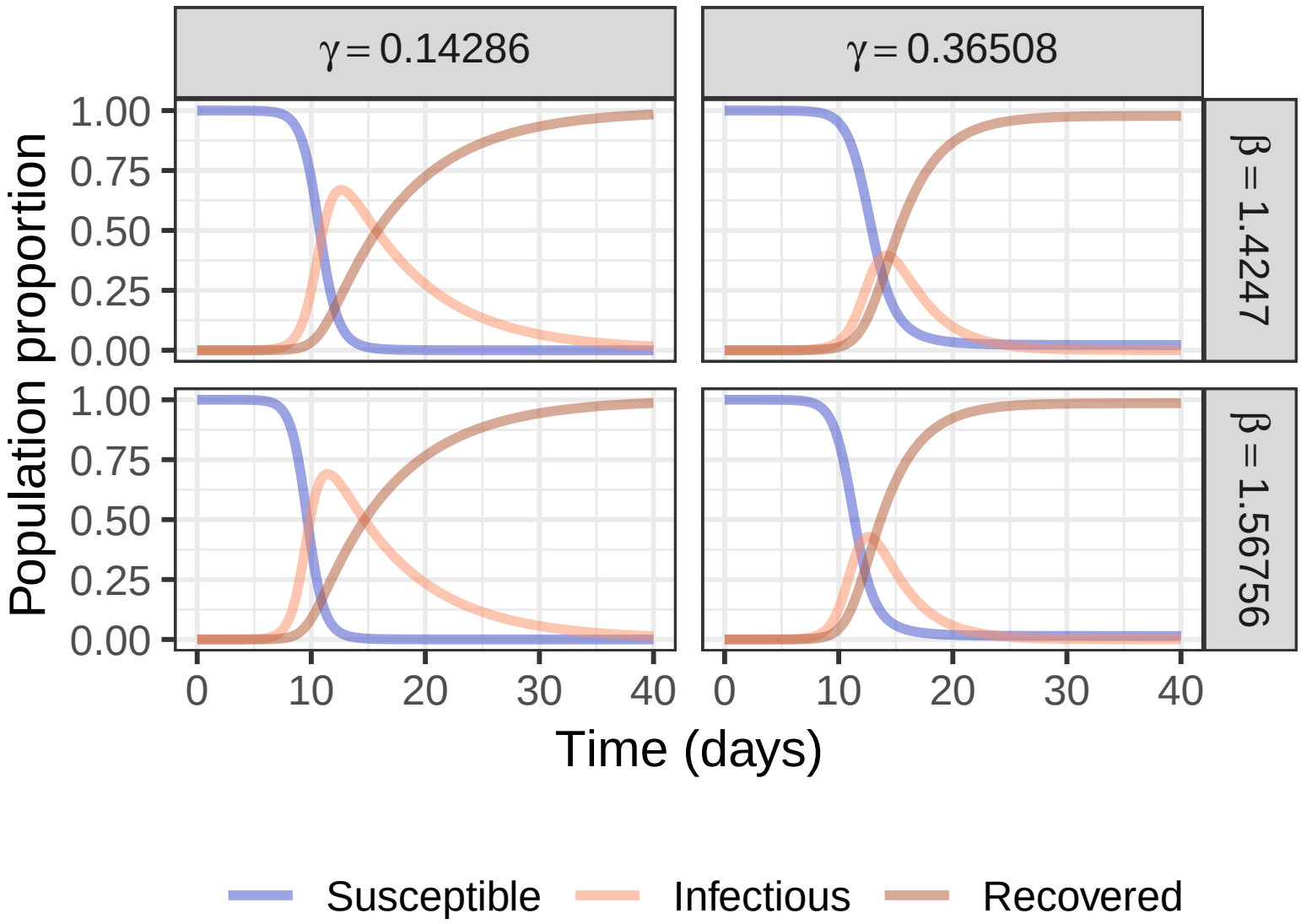


Using ggplot2

- Grammar of graphics means we can keep adding things to our plot to make it more complex
- We can keep adding more elements and options to show more details of the data
- e.g. to change the theme options to have a white background, different colours

```
my_palette <- c("#3A49C9", "#FC8E62", "#B05633")
names(my_palette) <- c("Susceptible", "Infectious", "Recovered")

SIR_plot_bg2 <- SIR_plot_bg_basic +
  geom_line(aes(color = state),
            alpha = 0.5, # semi-transparent
            size  = 1) + # thicker lines
  # let's use a custom color palette
  scale_color_manual(values = my_palette,
  # name = NULL omits the name in the legend for color
                     name   = NULL) +

  # custom labeller to include greek characters
  facet_grid(beta ~ gamma,
             labeller =
               label_bquote(cols = gamma == .(gamma),
                            rows = beta  == .(beta)))
```

Using ggplot2



Grouping and summarising

Consider instead of a factorial design for an SIR we have 100 simulations of an SIR model from a Monte Carlo simulation. 102 of the 10100 rows are shown below:

sim	time	S	I	R
1	0.0	99.000	1.000	0.000
1	0.1	98.867	1.102	0.031
1	0.2	98.721	1.213	0.066
2	0.0	99.000	1.000	0.000
2	0.1	98.875	1.093	0.031
2	0.2	98.740	1.195	0.065
3	0.0	99.000	1.000	0.000
3	0.1	98.856	1.113	0.032
3	0.2	98.696	1.237	0.067
4	0.0	99.000	1.000	0.000
4	0.1	98.862	1.106	0.031
4	0.2	98.710	1.224	0.066

Grouping and summarising

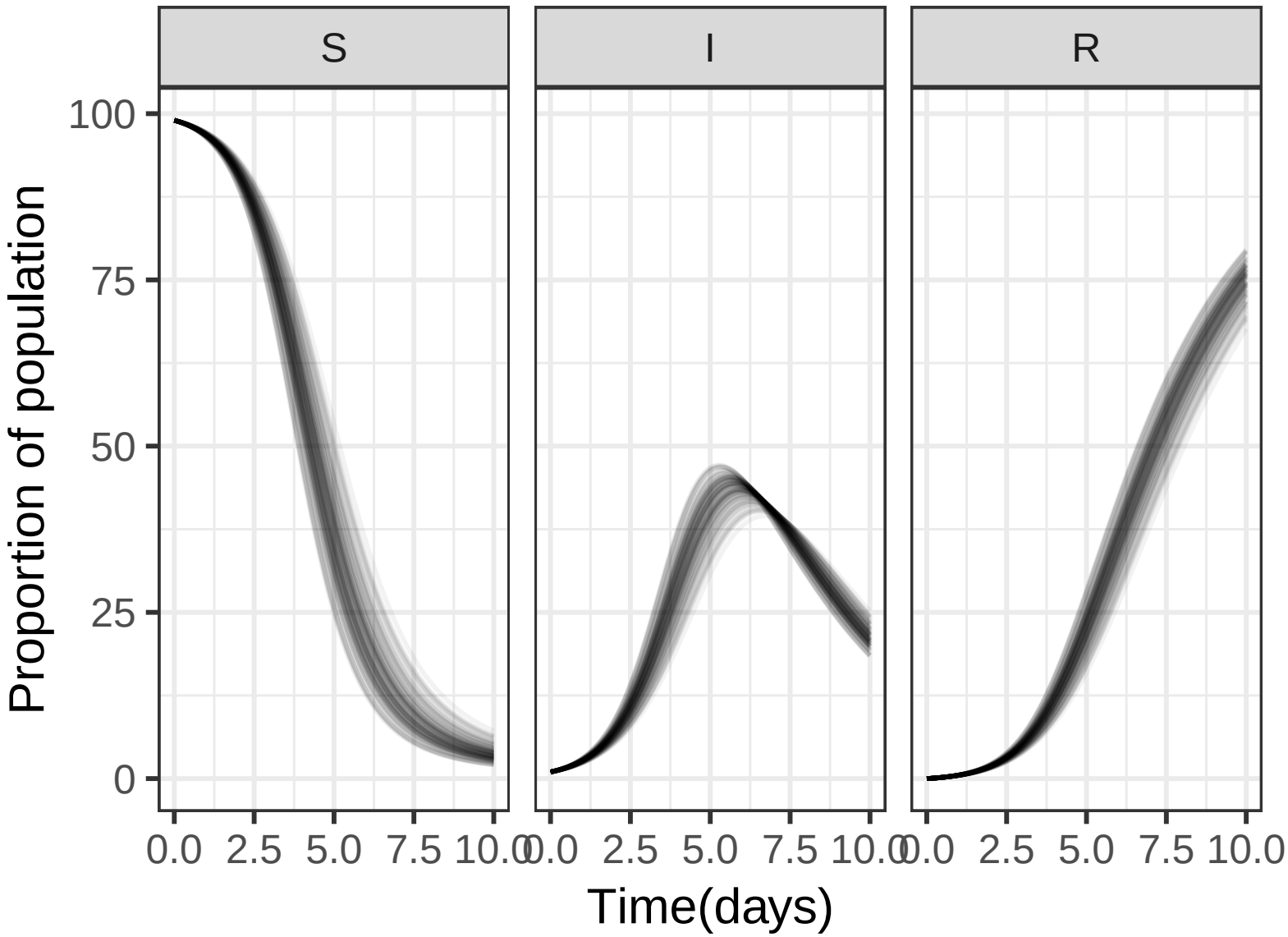
```
sol_sim_long <-
  gather(data  = sol_sim,
         key   = state,
         value = proportion,
         S, I, R)

sol_sim_long$state <-
  factor(sol_sim_long$state,
         levels = c("S", "I", "R"))
```

Grouping and summarising

- We can *group* by simulation index, `sim`, to show each as a line
- Use *alpha* transparency so we don't have a giant blob of black

```
plot_sim <-
  ggplot(data = sol_sim_long,
         aes(x = time,
             y = proportion)) +
  geom_line(aes(group = sim), alpha = 0.05) +
  facet_wrap(~state) + theme_bw() +
  xlab("Time(days)") +
  ylab("Proportion of population")
```

Grouping and summarising



Grouping and summarising

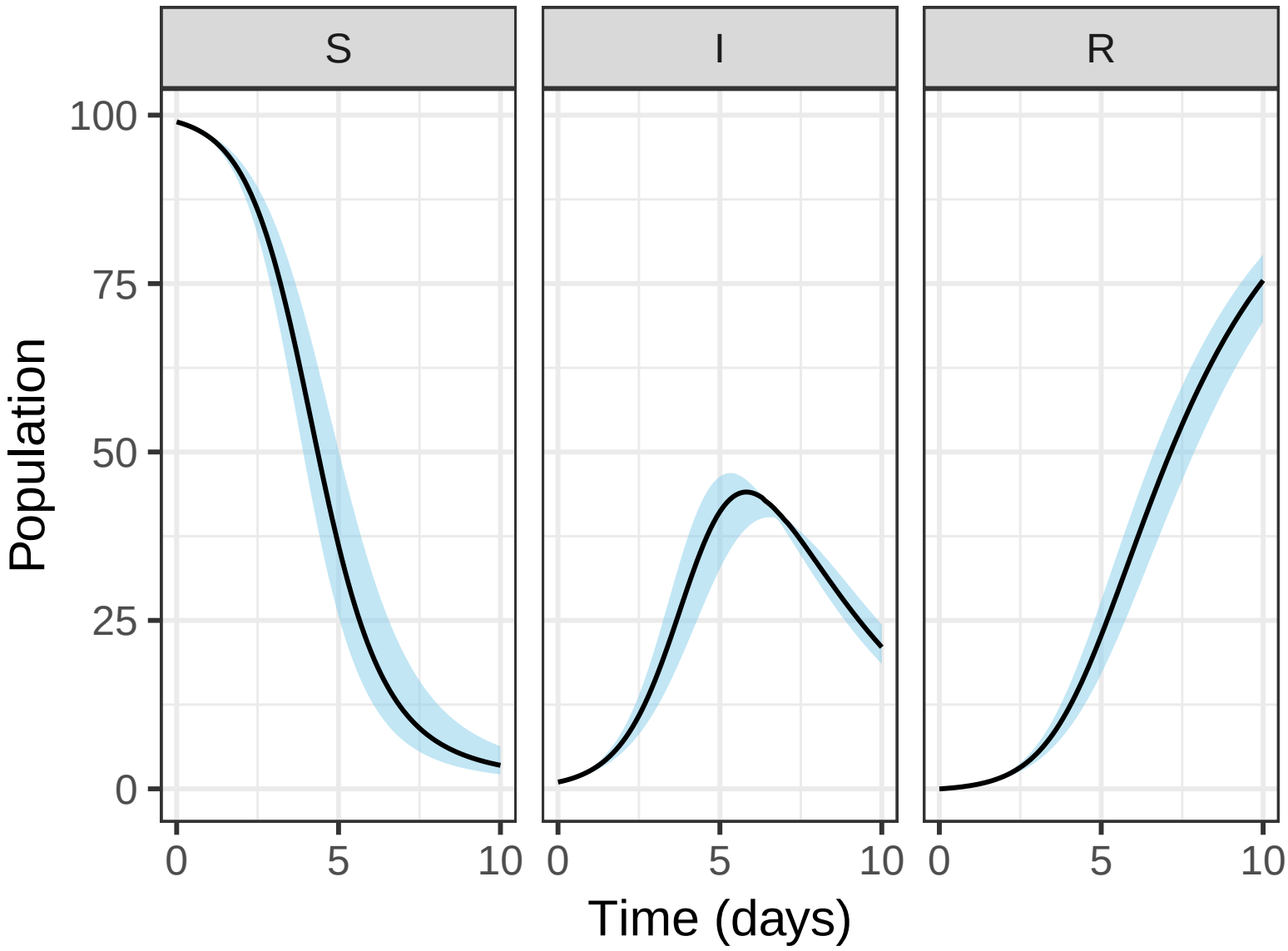
- To simplify this plot, we could calculate a 95% interval at each `time` for `S, I, R` and show these
- Use dplyr's
  - `group_by()` to define a grouping structure, and
  - `summarise()` to calculate summary statistics for each group (here we want the median and upper and lower bounds of a 95% interval)

```
sol_sim_grouped <- group_by(sol_sim_long,
                            time, state)

sol_sim_summarised <-
  summarise(sol_sim_grouped,
            L = quantile(proportion, probs = 0.025),
            M = quantile(proportion, probs = 0.5),
            U = quantile(proportion, probs = 0.975))
```

Grouping and summarising

```
plot_sim_summarised_ribbon <-
  ggplot(data = sol_sim_summarised,
         aes(x = time)) +
  geom_ribbon(aes(ymin = L,  # lower edge of ribbon
                  ymax = U), # upper edge of ribbon
              alpha = 0.5,   # make semi-transparent
              fill = "skyblue", # fill blue
              color = NA) +     # no border color
  geom_line(aes(y = M)) +       # line for median
  theme_bw() +                  # nicer theme
  facet_wrap( ~ state) +        # repeat for each state
  xlab("Time (days)") +         # human friendly axis label
  ylab("Population") +          # human friendly axis label
  scale_x_continuous(breaks = c(0, 5, 10)) # neater axis
```

Grouping and summarising



References

References

- More help on ggplot2 and the tidyverse is available
- The `#r4ds` community have TidyTuesday
- Chang (2012) is very useful if a little out of date

Chang, W. 2012. *R Graphics Cookbook: Practical Recipes for Visualizing Data*. O'Reilly Media. https://books.google.co.uk/books?id=fxL4tu5bzAAC.

Wickham, Hadley. 2010. "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics* 19 (1):3–28. https://doi.org/10.1198/jcgs.2009.07098.

———. 2014. "Tidy Data." *Journal of Statistical Software* 59 (1):1–23. https://doi.org/10.18637/jss.v059.i10.