Plotting model output with ggplot2

Modern Techniques in Modelling

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

Introduction

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

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)
```

Long and wide 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**
- 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

Wide data

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

Long data

Long and wide tidy data

- Our numerical solution to the SIR model is a wide data frame, values of $S(t), I(t), R(t)$ at given values of t
- We *pivot* the columns in `SIR` so that the data frame is *longer*
- This pivoting to a longer data frame helps us put the data in *key-value* pairs
- The key is the unique identifier
  - state - S, I, or R, and
  - time
- The value is the proportion of the population in this state at this time

Long and wide tidy data

– To make this pivot, we specify in `pivot_longer()`
  - which `cols` are to be converted from being k columns of length n to one column of length $n \times k$
  - the *names* column, `state`, contains the names of the columns being pivoted
  - the name of the column containing the *value* (proportion) of each `state` at given `time`

```
SIR_long <- pivot_longer(
  data                 = SIR,
  cols                 = c(S, I, R),
  names_to             = 'state',
  values_to            = 'proportion')
```

Long and wide tidy data

LONDON SCHOOL of HYGIENE & TROPICAL MEDICINE

Wide data

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

Long data

Visualisation with the grammar of graphics

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

Visualisation with ggplot2

- R package `ggplot2` uses a grammar of graphics
  - 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

Visualisation with ggplot2

– 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)
```

Visualisation with ggplot2

– 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))
```

Visualisation with ggplot2

- 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()
```

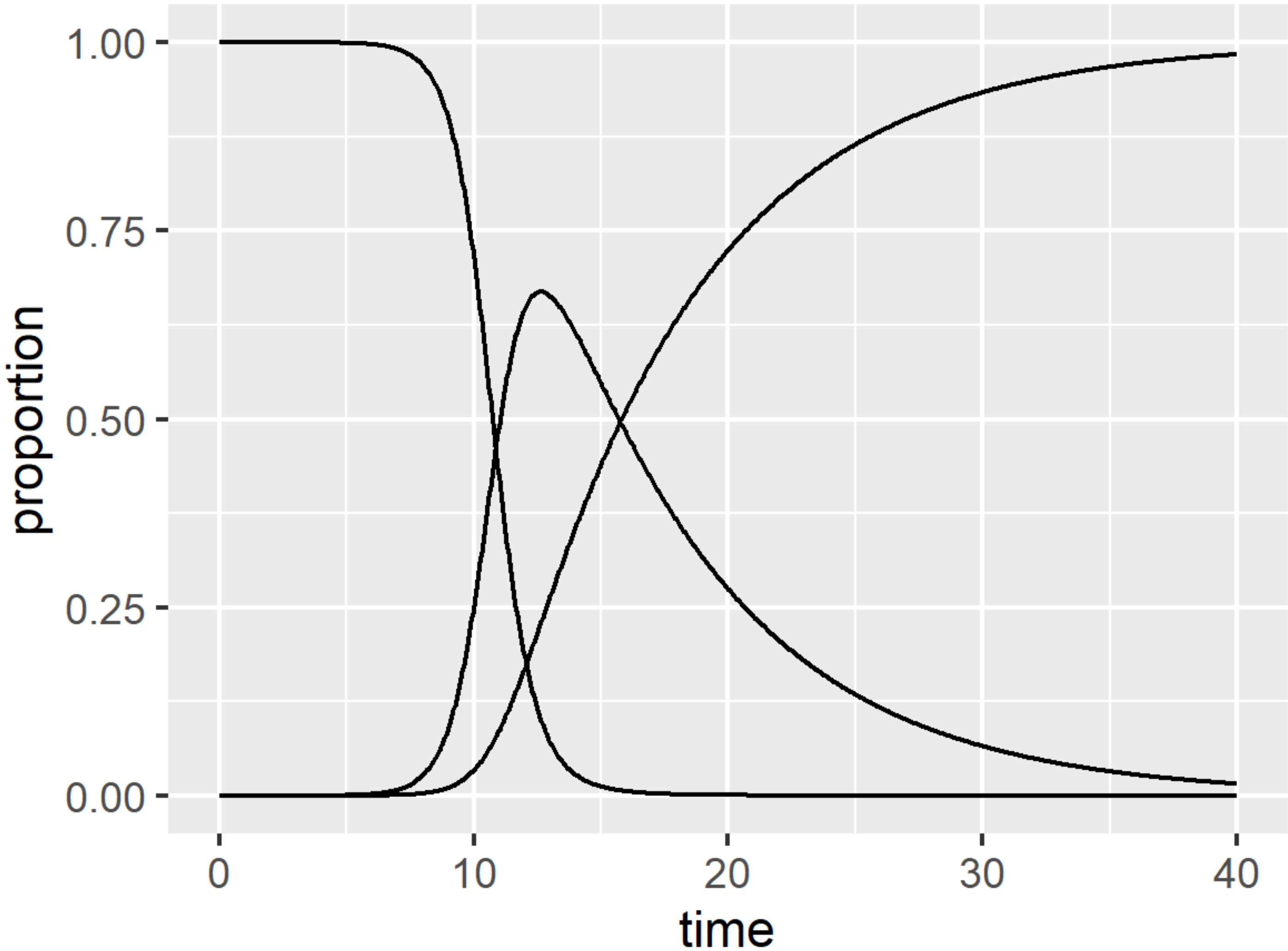
- We can set aesthetics `aes(...)` inside a geometry to modify the color, fill, alpha transparency, etc. according to a variable in the data frame

Visualisation with ggplot2

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

- Line geometry takes each (x_i, y_i) pair from the `aes()` specification and joins them with a line segment
- For each `state`, we want to plot a different *line*
- *group* aesthetic tells R that the data in `SIR_long` is grouped a particular way
- Line has `proportion` on y axis, `time` on x axis

Visualisation with ggplot2



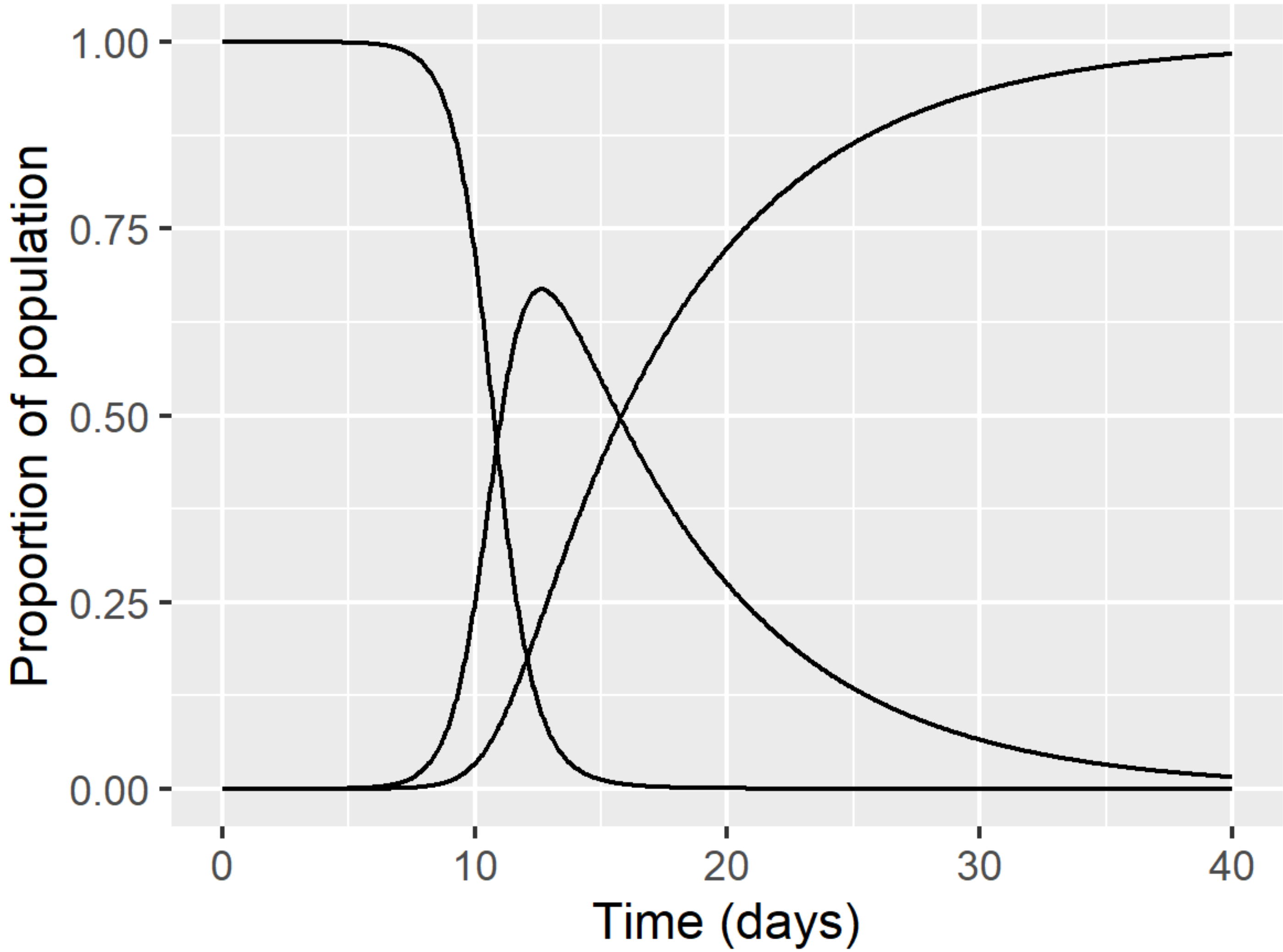
Visualisation with ggplot2

– Using our grammar of graphics' `+` operator let's add axis labels to the plot
  - `xlab()` and `ylab()` print their argument as axis labels

```
sir_ggplot <- sir_ggplot +
  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

Visualisation with ggplot2



Visualisation with ggplot2

- The plot on the previous slide didn’t give us much info on which line is which
- Consider a basic plot that we’ll recycle

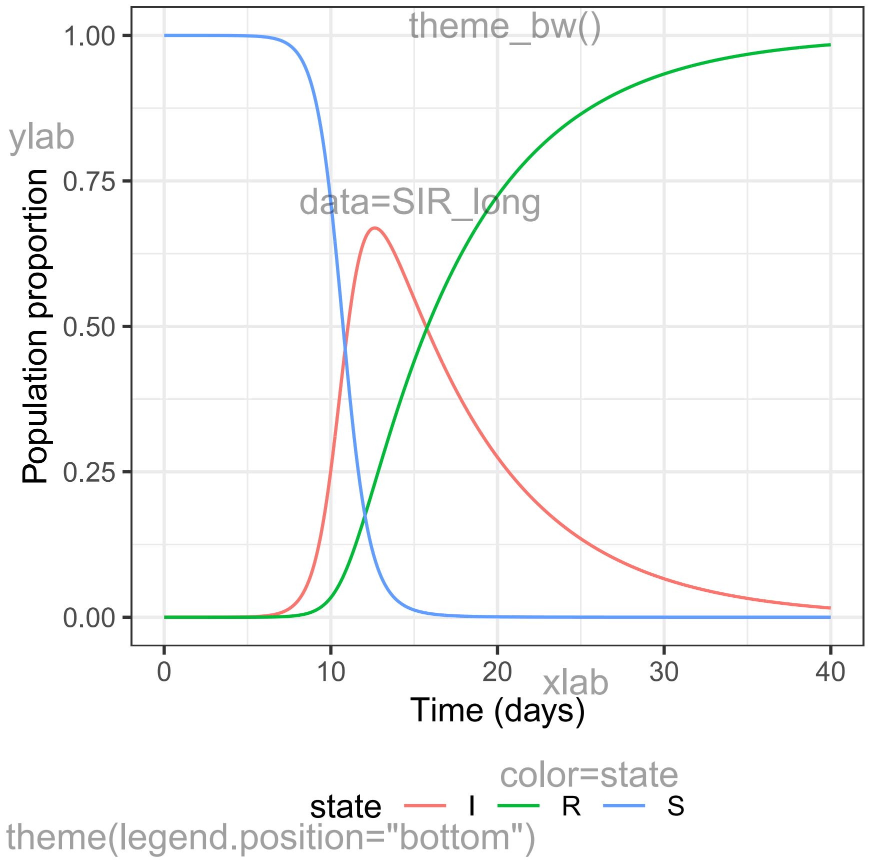
```
sir_ggplot_basic <-
  ggplot(data = SIR_long,              # where data lives
         aes(x = time,                 # set plot aesthetics...
             y = proportion)) +        # ...specifying x&y vars
  theme_bw() +                         # grey grid on white bg
  xlab('Time (days)') +                # 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
- we change the legend placement after we set the default theme, otherwise it will get overwritten

Visualisation with ggplot2

```
sir_ggplot_color <-
  sir_ggplot_basic +
  geom_line(
    aes(color = state))
```

- Mapping a variable, e.g. `state`, to part of our plot requires it is inside `aes(...)`
- Here we have *color*ed each line by state
- Static options go outside `aes(...)`

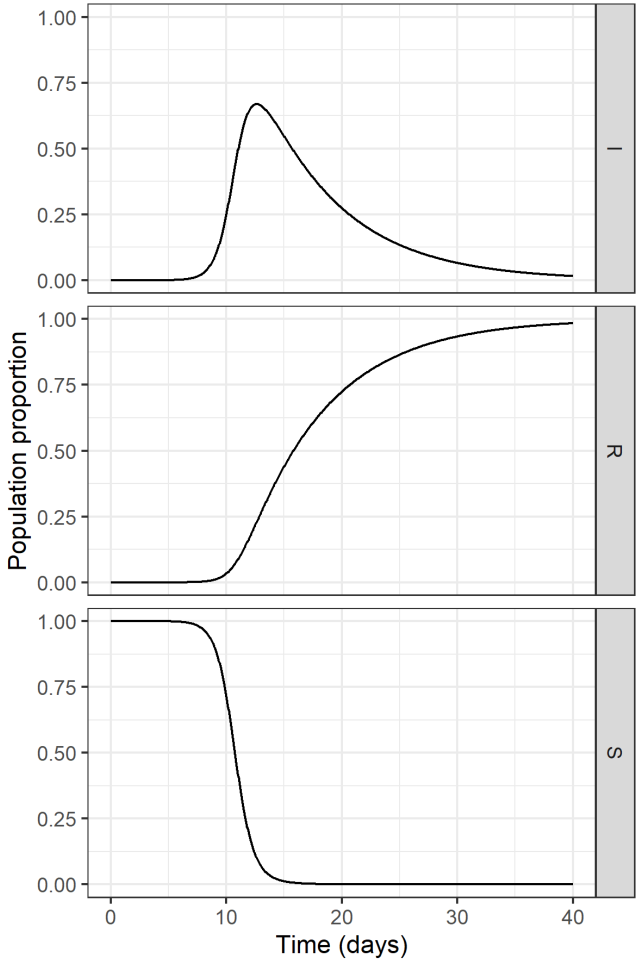


Visualisation with ggplot2

Draw small multiples with `facet_grid()`, repeating the geometry for each level of the grouping variable on the *rows* of the grid

```
sir_ggplot_facet <-
  sir_ggplot_basic +
  geom_line() +
  facet_grid(
    rows = vars(state)
  )
```

where `vars()` indicates that we are selecting a list of variables



Relevelling factors

LONDON SCHOOL of HYGIENE & TROPICAL MEDICINE

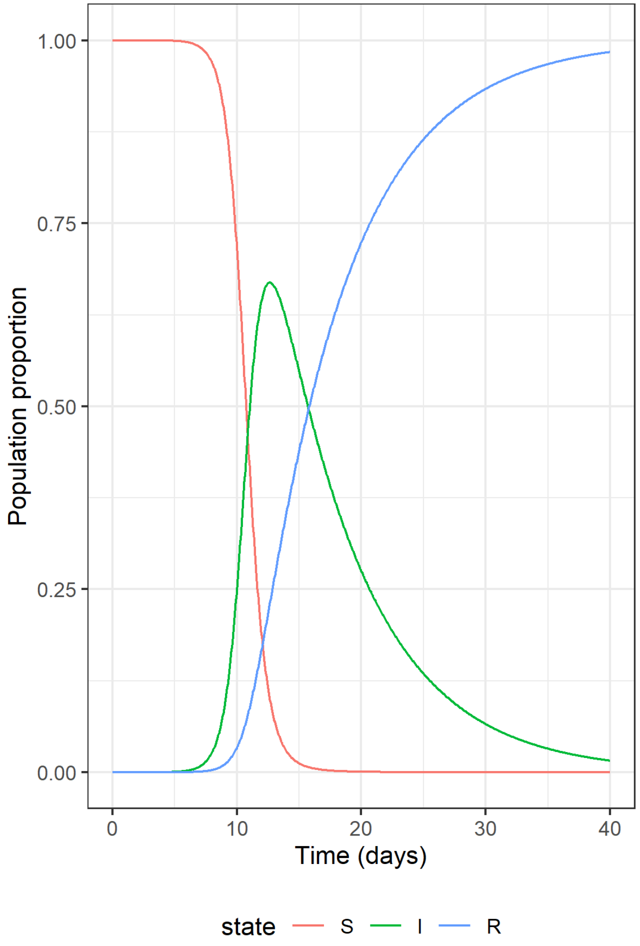
Relevelling factors

– Default behaviours are:
  - `pivot_longer()` 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

```r
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))
```



Plotting multiple simulations

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

Grouping in a factorial design

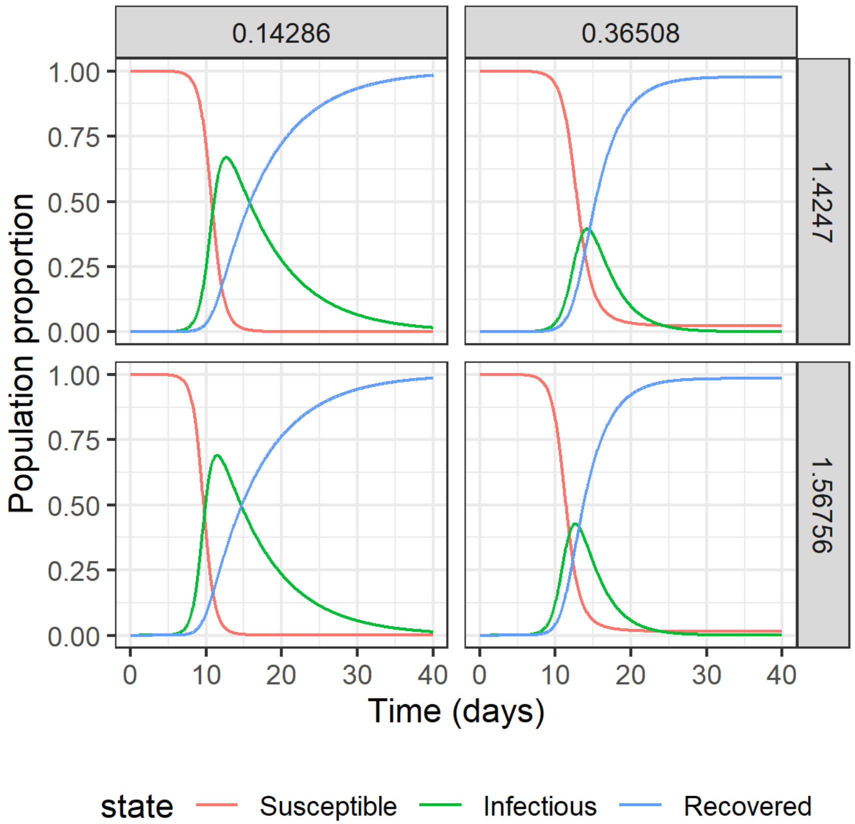
Consider a factorial design for SIR simulations with each combination of $\beta = 1.42470, 1.56756$ and $\gamma = 0.14286, 0.36508$

Grouping in a factorial design

– Ultimately want a line for each value of β, γ and state
– Build the line plots with `color = state` as before
– Use small multiples to show a plot for each combination of β and γ
– With `facet_grid()` we specify grouping variables for rows and/or columns of plot
  - Can specify the grouping structure explicitly with `facet_grid(rows = vars(beta), cols = vars(gamma))`
  - or with `row variables ~ column variables`, e.g. `facet_grid(beta ~ gamma)`

Grouping in a factorial design

```
SIR_plot_bg_basic <-
  ggplot(data =
        factorial_sim,
         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(rows = vars(beta),
             cols = vars(gamma))
```



Grouping in Monte Carlo simulation

Consider instead of a factorial design for an SIR we have 100 simulations of an SIR model from a Monte Carlo simulation. 12 of the 10100 rows are shown below:

Grouping in Monte Carlo simulation

Pivot the data, as before, and relevel the `state` variable

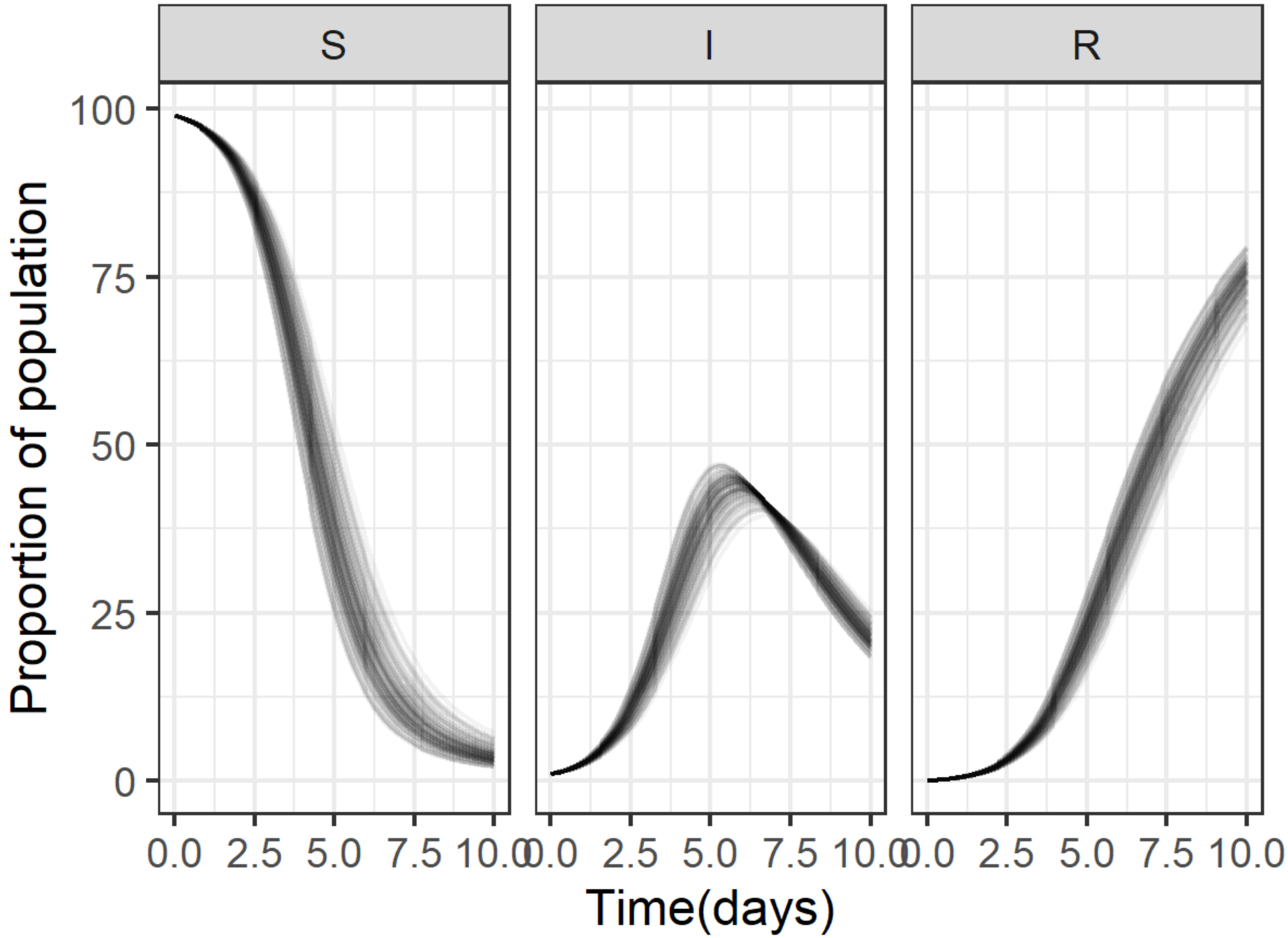
```
sol_sim_long <- pivot_longer(
  data              = sol_sim,
  cols              = c(S, I, R),
  names_to          = 'state',
  values_to         = 'proportion')

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

Grouping in Monte Carlo simulation

– 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_grid(cols = vars(state)) +
  theme_bw() +
  xlab('Time(days)') +
  ylab('Proportion of population')
```

Grouping in Monte Carlo simulation



Grouping in Monte Carlo simulation

- 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 (median, upper and lower bounds of a 95% interval)

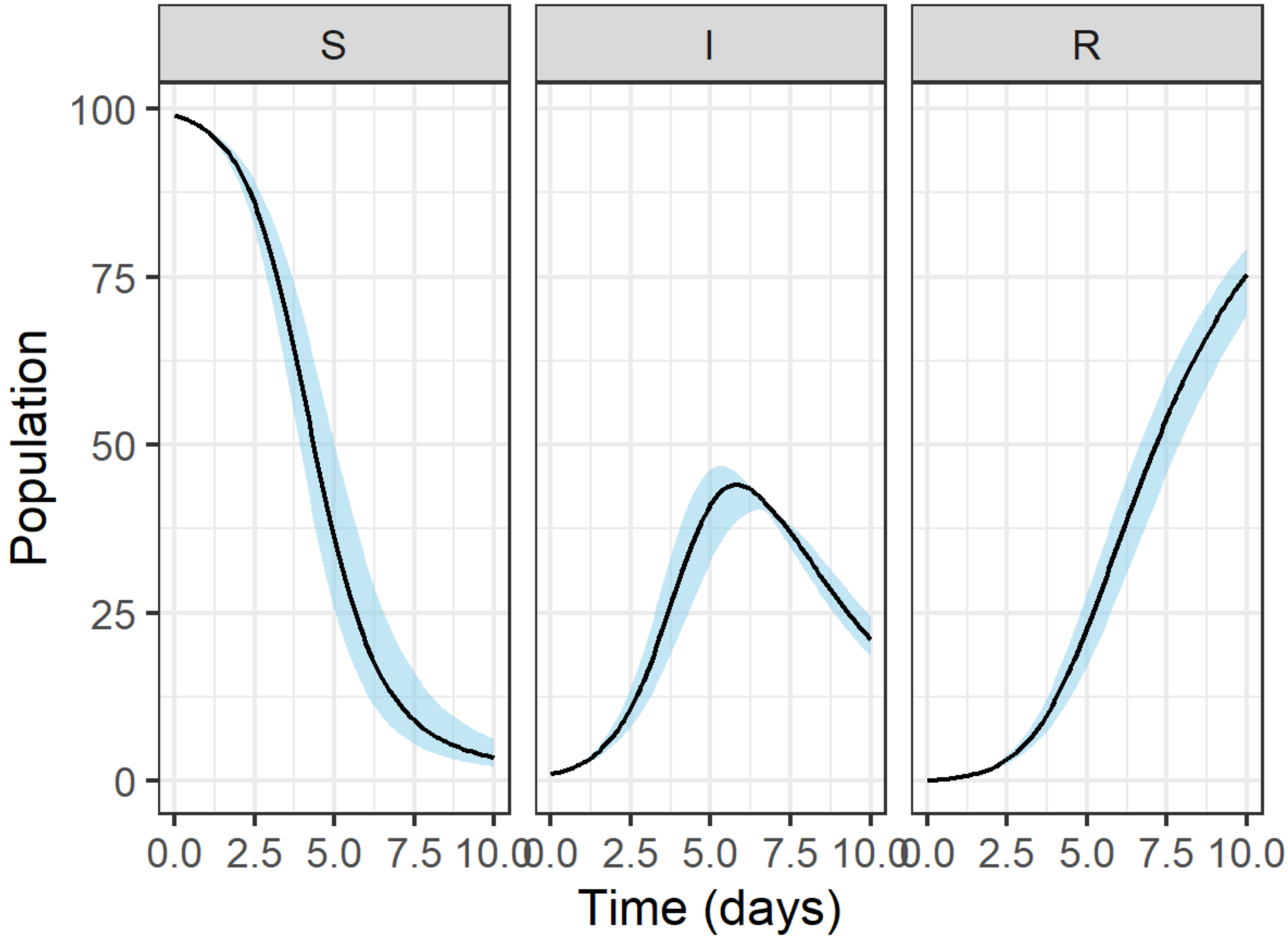
```
sol_sim_grouped <- group_by(sol_sim_long,
                            time, state)

sol_sim_summarised <-
  summarise(sol_sim_grouped,
            q0.025 = quantile(proportion, probs = 0.025),
            q0.500 = quantile(proportion, probs = 0.5),
            q0.975 = quantile(proportion, probs = 0.975))
```

Grouping in Monte Carlo simulation

- Can use multiple geometries with different aesthetics
- Plot the ribbon and then plot the median line

```
plot_sim_summarised_ribbon <-
  ggplot(data = sol_sim_summarised,
         aes(x = time)) +
  geom_ribbon(aes(ymin = q0.025,   # lower edge of ribbon
                  ymax = q0.975),  # upper edge of ribbon
              alpha = 0.5,         # make semi-transparent
              fill = 'skyblue',    # fill blue
              color = NA) +        # no border color
  geom_line(aes(y = q0.500)) +     # line for median
  theme_bw() +                     # nicer theme
  facet_grid(
    cols = vars(state)) +          # repeat for each state
  xlab('Time (days)') +            # human friendly axis label
  ylab('Population')               # human friendly axis label
```

Grouping in Monte Carlo simulation



Summary

LONDON SCHOOL of HYGIENE &TROPICAL MEDICINE

Summary

- ggplot2 uses `aes`thetics to map variables in data frame to elements of plot
- Plot is sequentially built up by adding elements
  - geometries (e.g. lines, ribbons)
  - annotations (e.g. axis labels)
  - theme options
- Data needs to be in key-value pairs for plotting
- Data in key-value pairs is easily summarised by key group

Additional Resources

- More help on ggplot2 and the tidyverse is available
- The `#r4ds` community have TidyTuesday
- Chang (2017) is very useful if a little out of date
- Wickham (2010) on philosophy behind ggplot2
- Wickham (2014) on what tidy data is

Chang, Winston. 2017. *R Graphics Cookbook: Practical Recipes for Visualizing Data*. 2nd ed. O'Reilly Media.

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.