Data Visualization

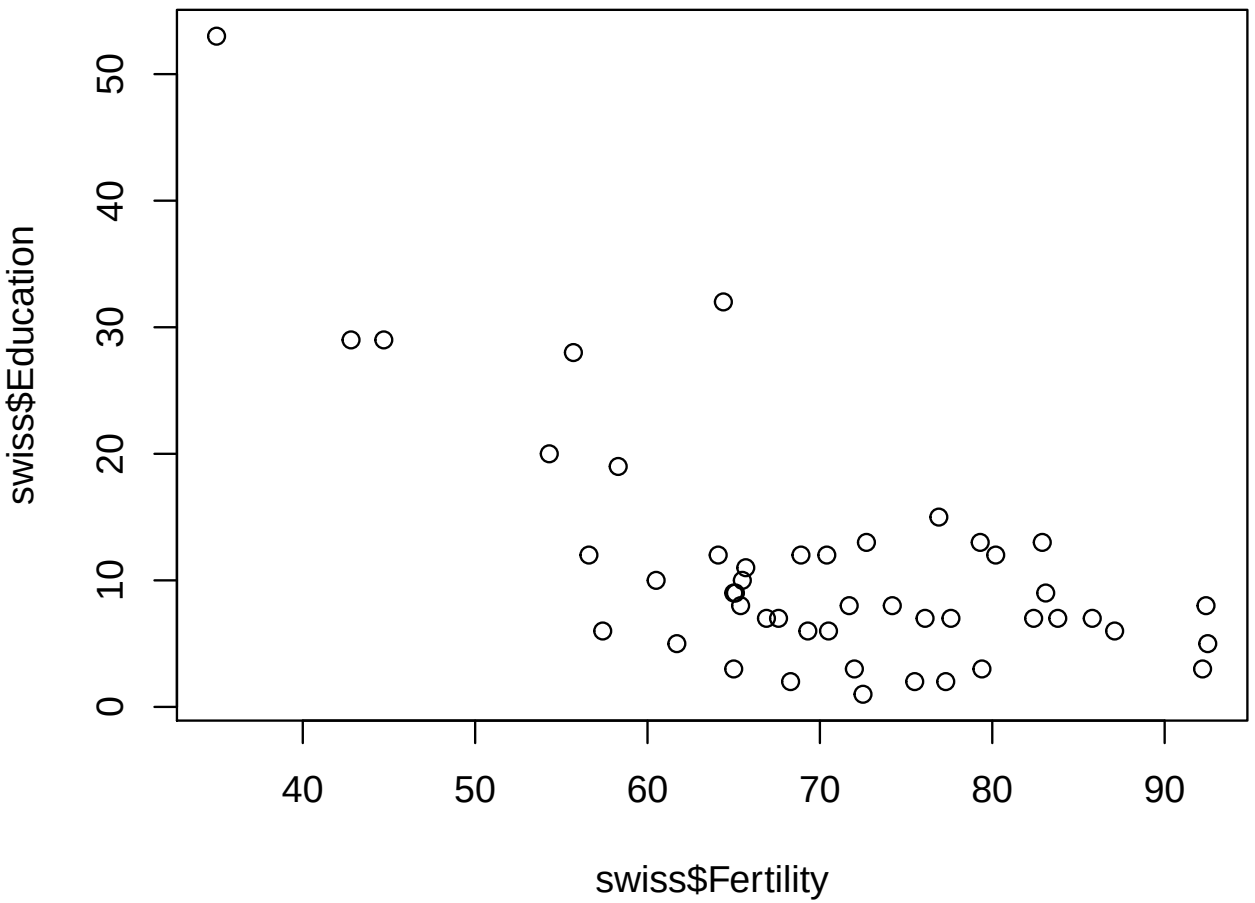
Guy J. Abel

R graphics Package

- R has fantastic graphic capabilities.
  - There are many approaches and packages designed for this specific task.
  - The `graphics` package (pre-loaded with R) has many plotting functions.
  - Simple look, may not be consistent across different types of plots

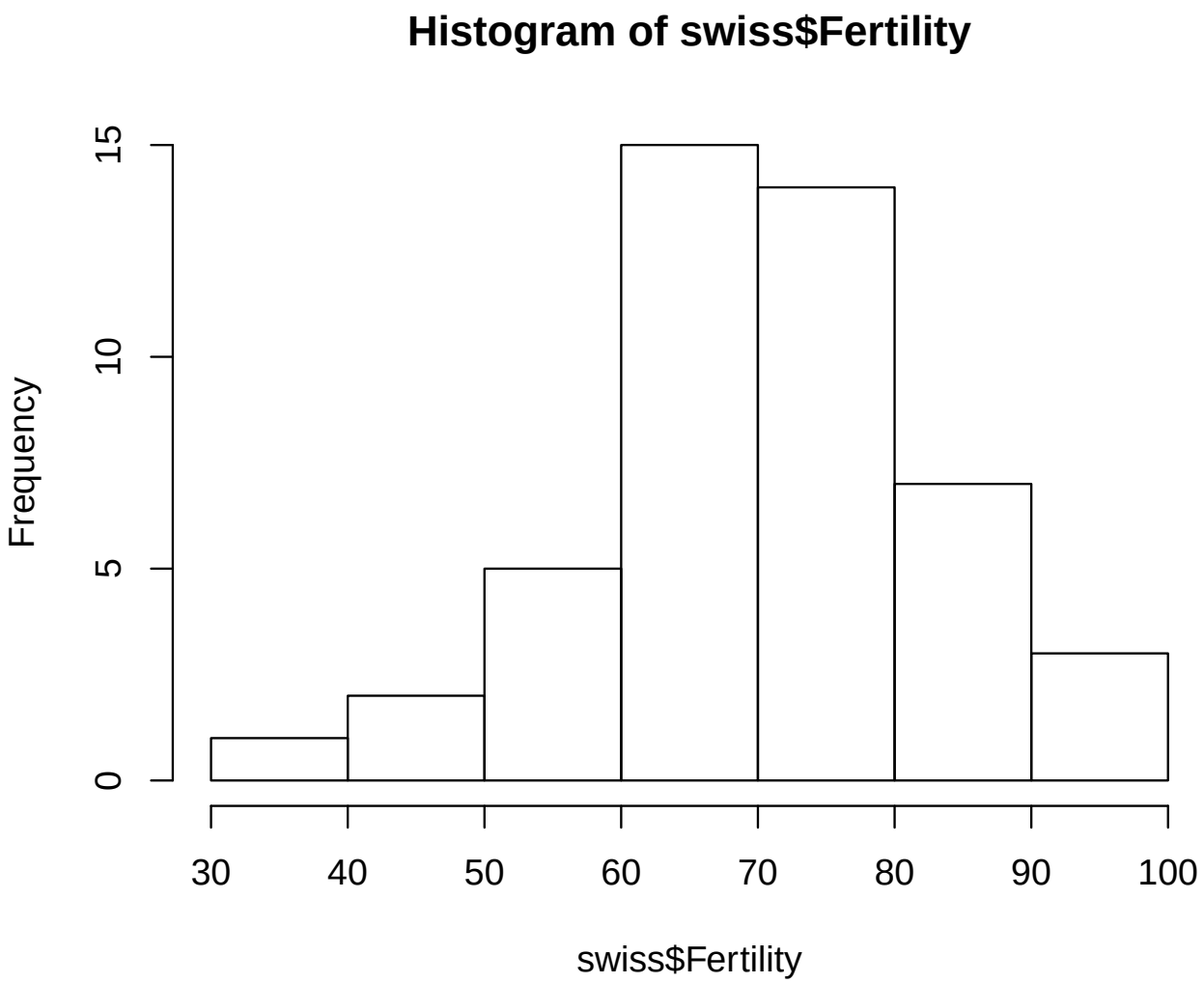
R graphics Package

```
> plot(x = swiss$Fertility, y = swiss$Education)
```



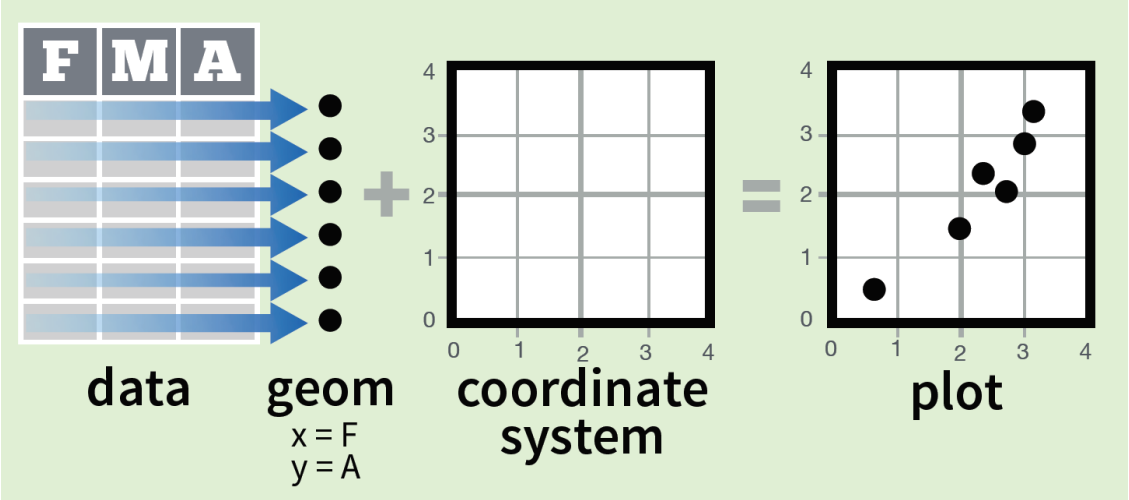
R graphics Package

```
> hist(swiss$Fertility)
```



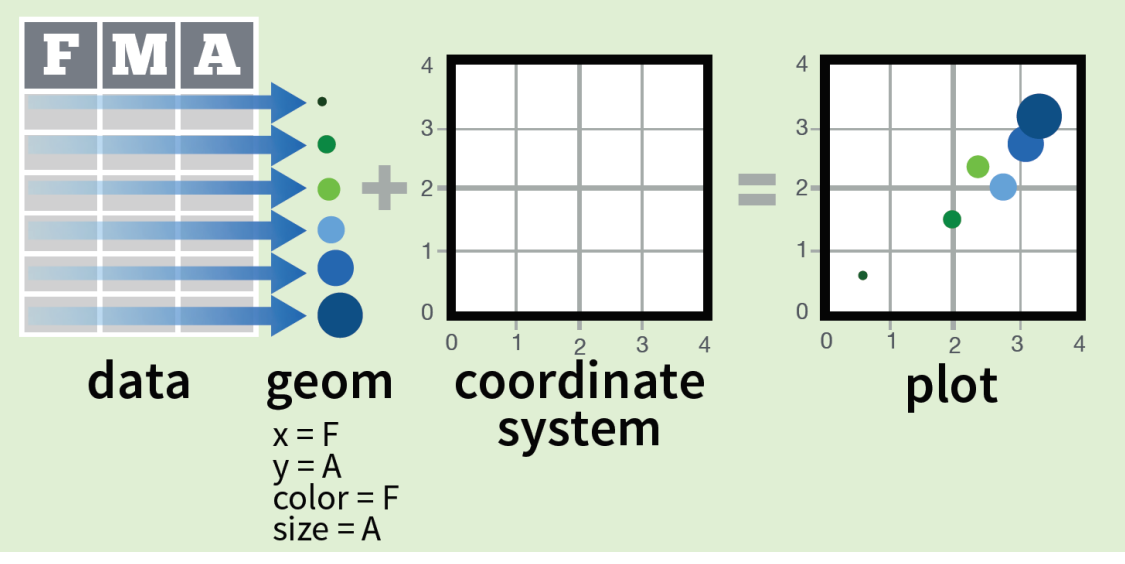
Grammer of Graphics

- The `ggplot2` package by Hadley Wickham implements the grammar of graphics, a coherent system for describing and building graphs.
  - One of the most beautiful and most versatile.
  - Flexible and consistent approach across different plots
  - Do more faster as a single learning one system for all plots types that you can apply in many data situations.
- Build every graph from the same components:
  - data set
  - coordinate system
  - geoms (visual marks that represent data points)
- Included in the tidyverse package (more on tidyverse later)



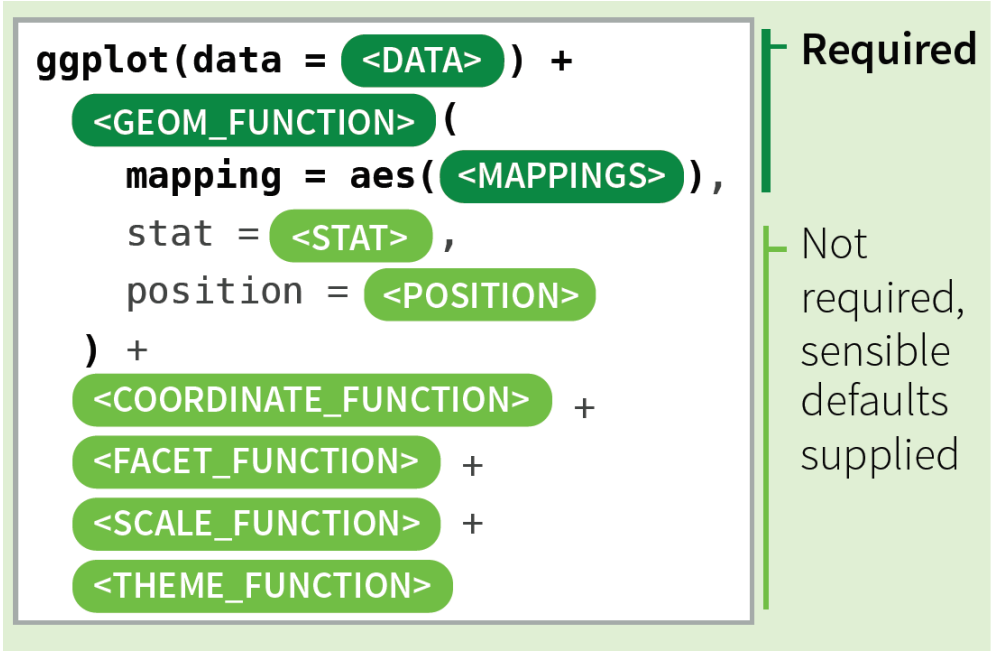
Grammer of Graphics

- Map variables in the data to visual properties of the geom (aesthetics) like size, color, and x and y locations.
  - Aesthetics meaning something you can see.



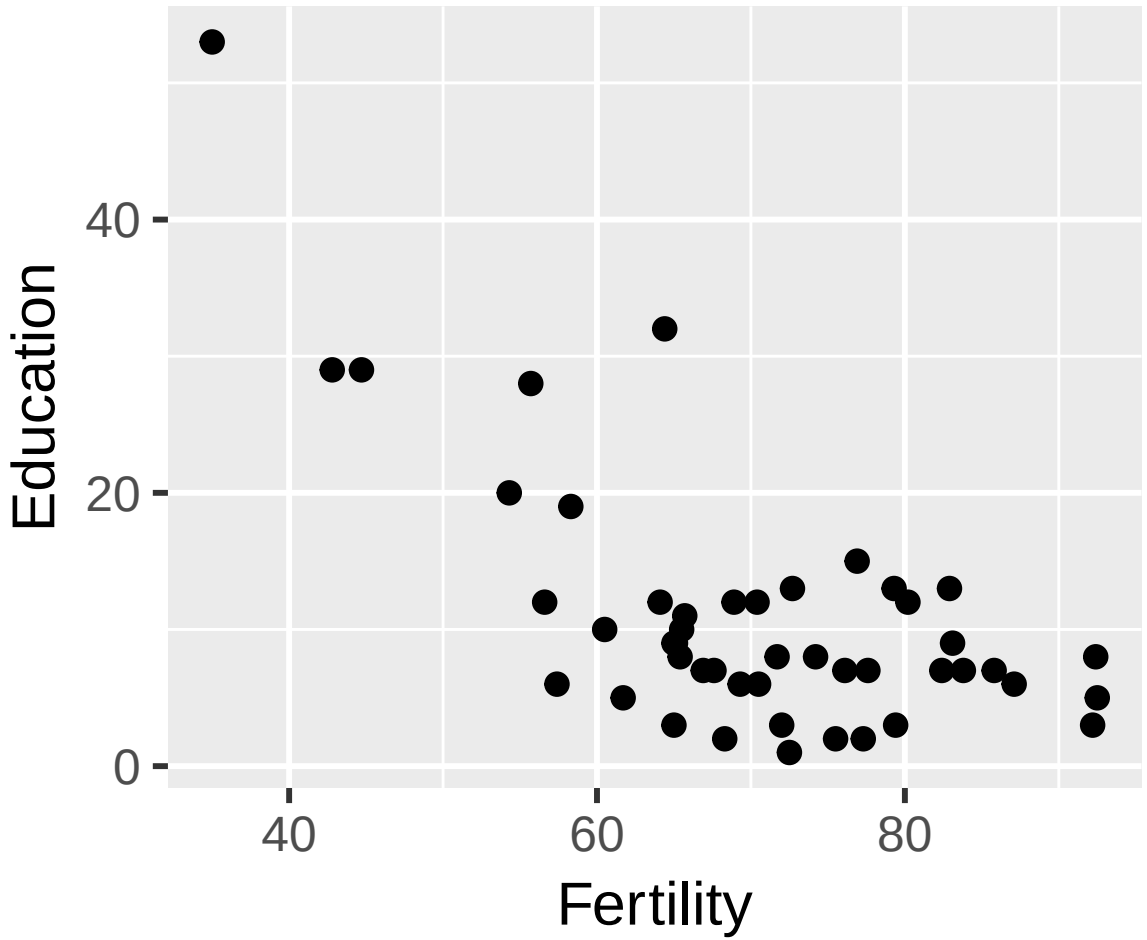
Grammer of Graphics

- The R code follows a common template



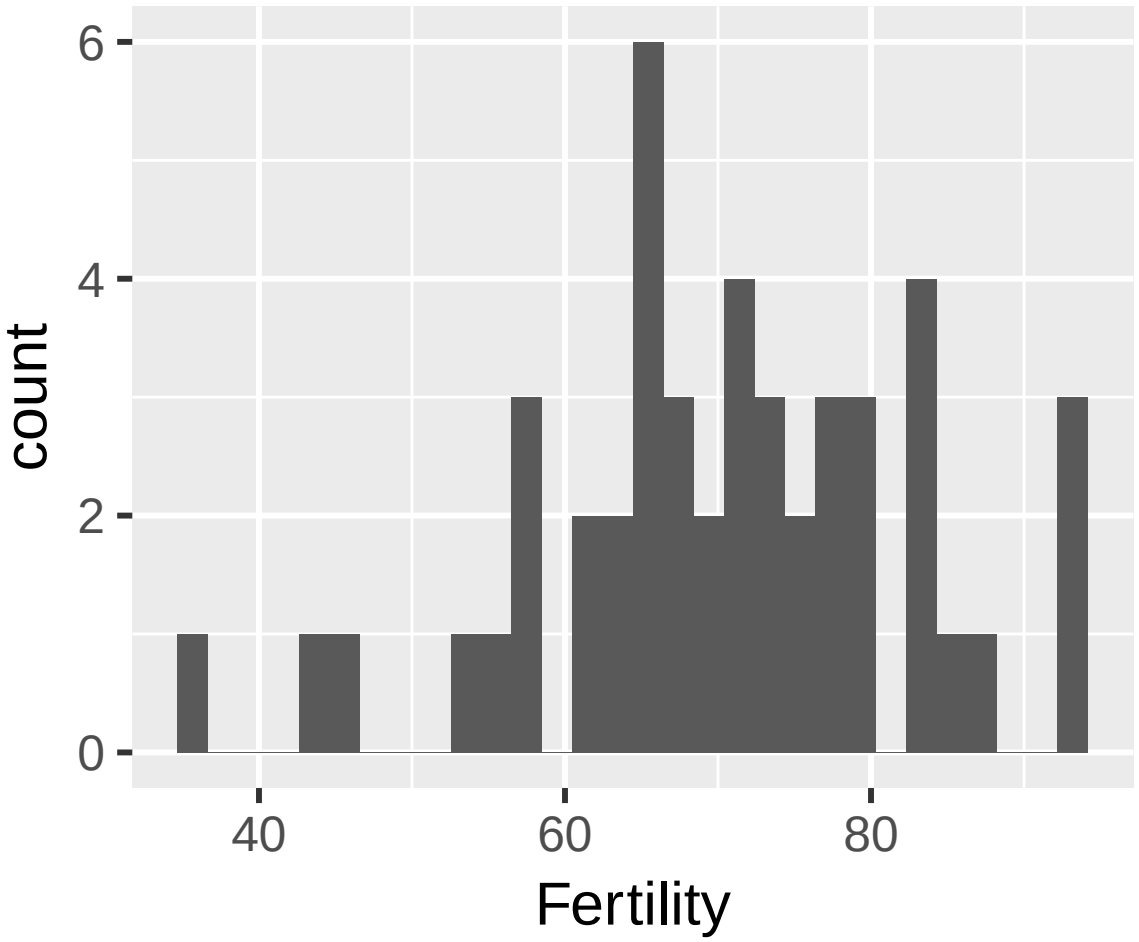
Grammer of Graphics

```
> library(tidyverse)
> ggplot(data = swiss, mapping = aes(x = Fertility, y = Education)) +
+   geom_point()
```



Grammer of Graphics

```
> ggplot(data = swiss, mapping = aes(x = Fertility)) +
+   geom_histogram()
```



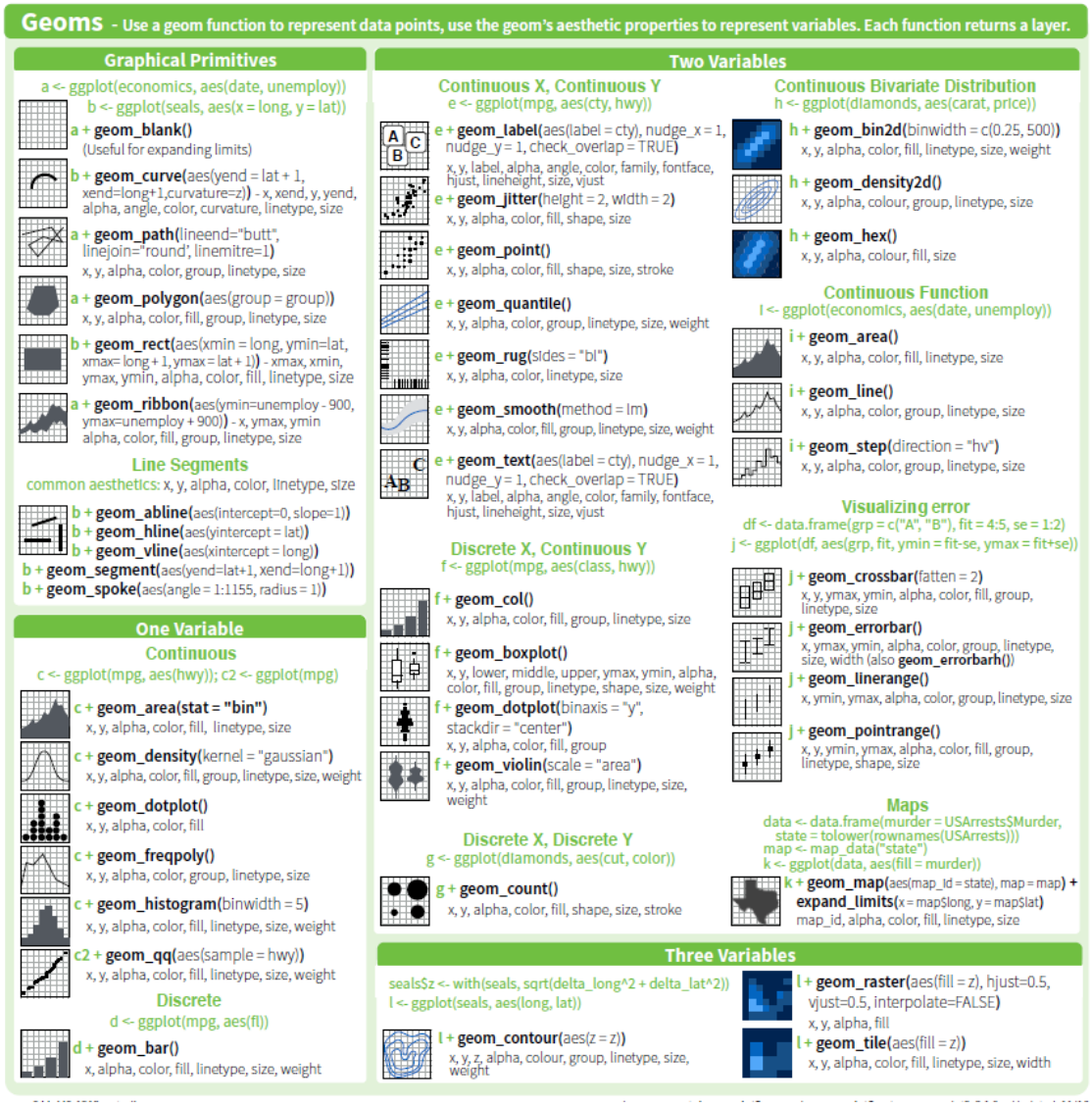
Grammer of Graphics

- The function `ggplot()` creates a coordinate system to add layers onto.
- The first argument of `ggplot()` is the data set used to graph
  - For example, `ggplot(data = swiss)` creates an empty graph that will use the `swiss` data set.
  - Can then build on the empty graphs by adding one or more layers to `ggplot()`.
- The geom functions adds a layer of points to your plot.
  - The `ggplot2` library comes with many geom functions
  - Each add a different type of layer to a plot.
  - Each geom function in ggplot2 takes a `mapping` argument.
- The `mapping` argument explains where your points should go
  - Set in the `mapping` argument with a call to the `aes()` function.
  - For example the `x` and `y` arguments of `aes()` explain which variables to map to the x- and y-axes of your plot.
  - The `ggplot()` function looks for those variables in your data set.
  - The `mapping` arguments are different for each geom function

Geom

- The ggplot2 package provides over 30 geom functions
  - Further packages provide even more.
- Each geom function is suitable for visualizing a certain type of data or relationship.
  - Listed on the first page of the ggplot cheat sheet.
  - Next to geom is a visual representation of the geom.
  - Beneath geom is a list of aesthetics that apply to the geom.
  - Required aesthetics are in bold.

Geoms



Geoms - Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

Graphical Primitives

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + **geom_blank()**
(Useful for expanding limits)

b + **geom_curve**(aes(yend = lat + 1, xend=long+1,curvature=z)) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + **geom_path**(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, group, linetype, size

a + **geom_polygon**(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + **geom_rect**(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + **geom_ribbon**(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin
alpha, color, fill, group, linetype, size

Line Segments

common aesthetics: x, y, alpha, color, linetype, size

b + **geom_abline**(aes(intercept=0, slope=1))
b + **geom_hline**(aes(yintercept = lat))
b + **geom_vline**(aes(xintercept = long))
b + **geom_segment**(aes(yend=lat+1, xend=long+1))
b + **geom_spoke**(aes(angle = 1:1155, radius = 1))

One Variable

Continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + **geom_area(stat = "bin")**
x, y, alpha, color, fill, linetype, size

c + **geom_density**(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + **geom_dotplot()**
x, y, alpha, color, fill

c + **geom_freqpoly()**
x, y, alpha, color, group, linetype, size

c + **geom_histogram**(binwidth = 5)
x, y, alpha, color, fill, linetype, size, weight

c2 + **geom_qq**(aes(sample = hwy))
x, y, alpha, color, fill, linetype, size, weight

Discrete

d <- ggplot(mpg, aes(fl))

d + **geom_bar()**
x, alpha, color, fill, linetype, size, weight

Two Variables

Continuous X, Continuous Y

e <- ggplot(mpg, aes(cty, hwy))

e + **geom_label**(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE)
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + **geom_jitter**(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + **geom_point()**
x, y, alpha, color, fill, shape, size, stroke

e + **geom_quantile()**
x, y, alpha, color, group, linetype, size, weight

e + **geom_rug**(sides = "bl")
x, y, alpha, color, linetype, size

e + **geom_smooth**(method = lm)
x, y, alpha, color, fill, group, linetype, size, weight

e + **geom_text**(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE)
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

f <- ggplot(mpg, aes(class, hwy))

f + **geom_col()**
x, y, alpha, color, fill, group, linetype, size

f + **geom_boxplot()**
x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + **geom_dotplot**(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill, group

f + **geom_violin**(scale = "area")
x, y, alpha, color, fill, group, linetype, size, weight

Discrete X, Discrete Y

g <- ggplot(diamonds, aes(cut, color))

g + **geom_count()**
x, y, alpha, color, fill, shape, size, stroke

Continuous Bivariate Distribution

h <- ggplot(diamonds, aes(carat, price))

h + **geom_bin2d**(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + **geom_density2d()**
x, y, alpha, colour, group, linetype, size

h + **geom_hex()**
x, y, alpha, colour, fill, size

Continuous Function

i <- ggplot(economics, aes(date, unemploy))

i + **geom_area()**
x, y, alpha, color, fill, linetype, size

i + **geom_line()**
x, y, alpha, color, group, linetype, size

i + **geom_step**(direction = "hv")
x, y, alpha, color, group, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

j + **geom_crossbar**(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + **geom_errorbar()**
x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + **geom_linerange()**
x, ymin, ymax, alpha, color, group, linetype, size

j + **geom_pointrange()**
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + **geom_map**(aes(map_id = state), map = map) + **expand_limits**(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
l <- ggplot(seals, aes(long, lat))

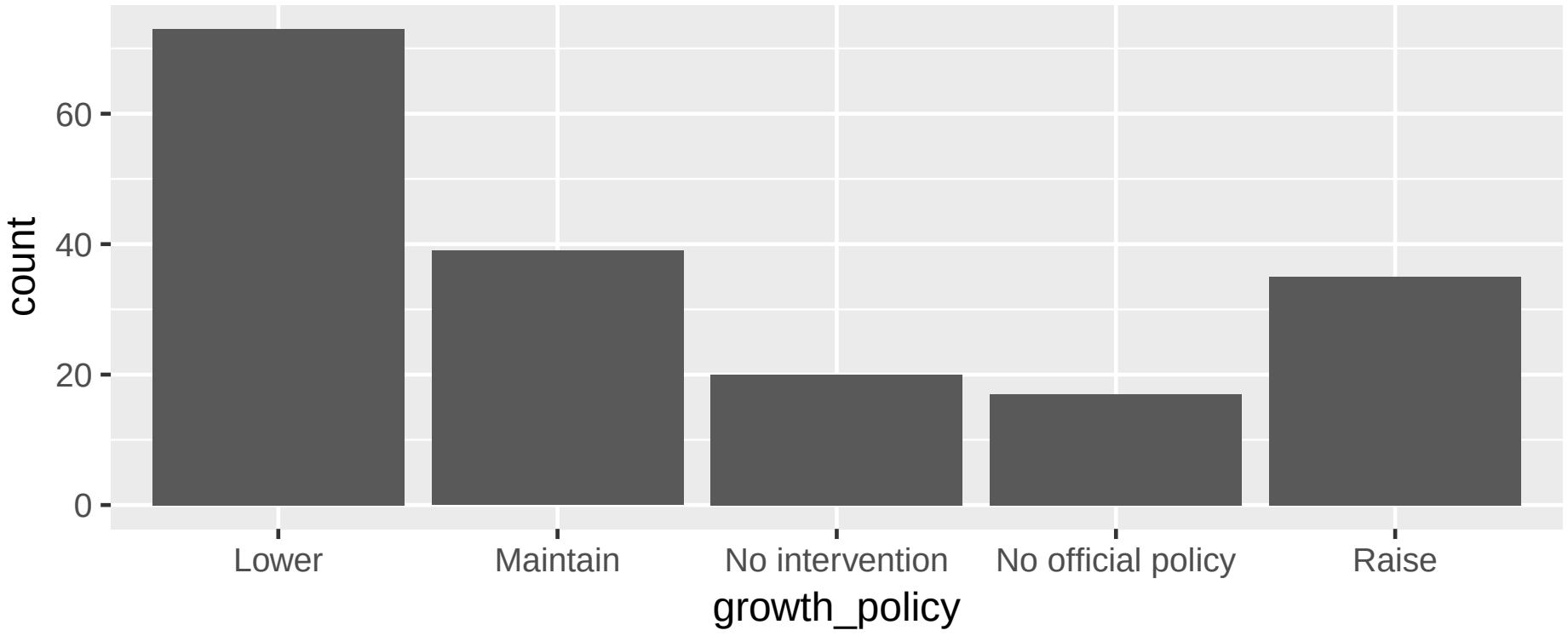
l + **geom_contour**(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + **geom_raster**(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + **geom_tile**(aes(fill = z))
x, y, alpha, color, fill, linetype, size, width

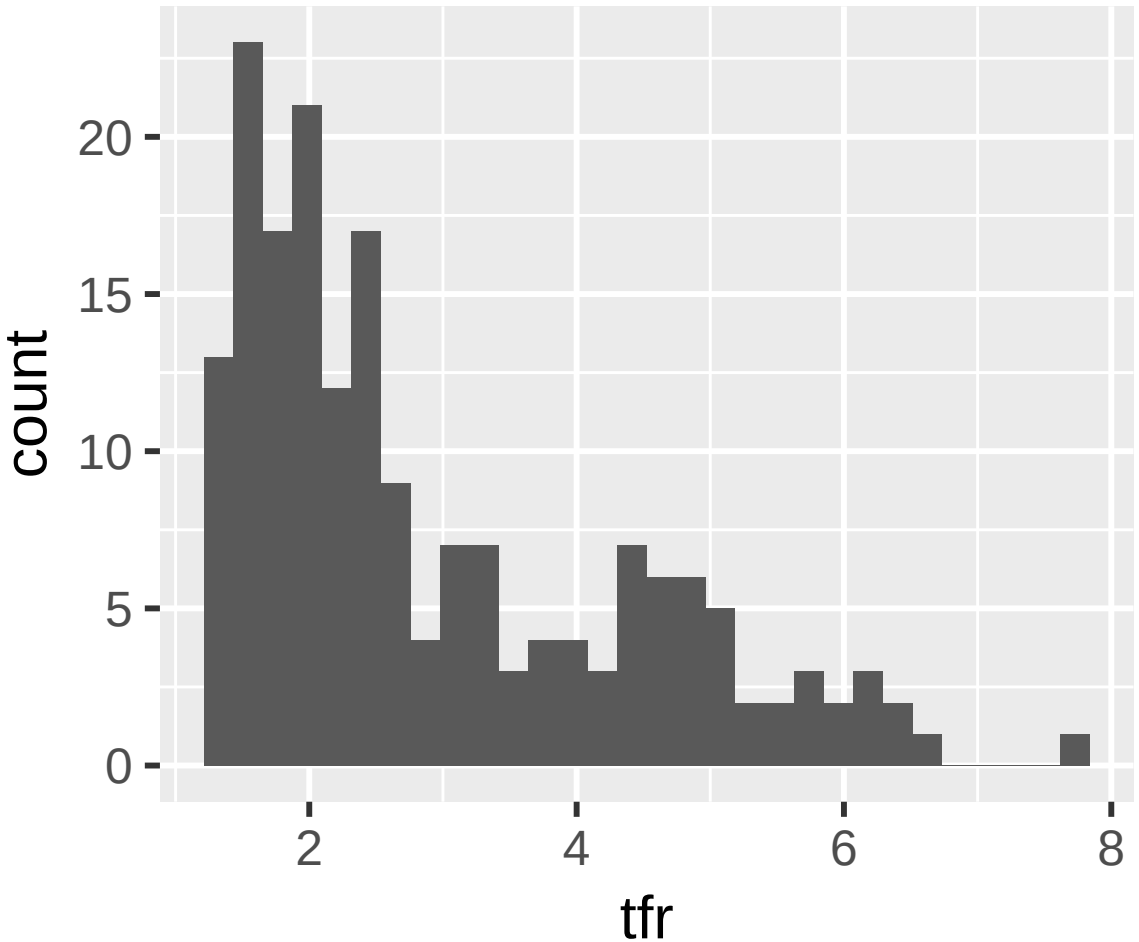
One Variable: Discrete - geom_bar()

```
> df0 <- read_csv("./data/wpp_all.csv")
> df1 <- filter(df0, period == "2010-2015")
> ggplot(data = df1, mapping = aes(x = growth_policy)) +
+   geom_bar()
```



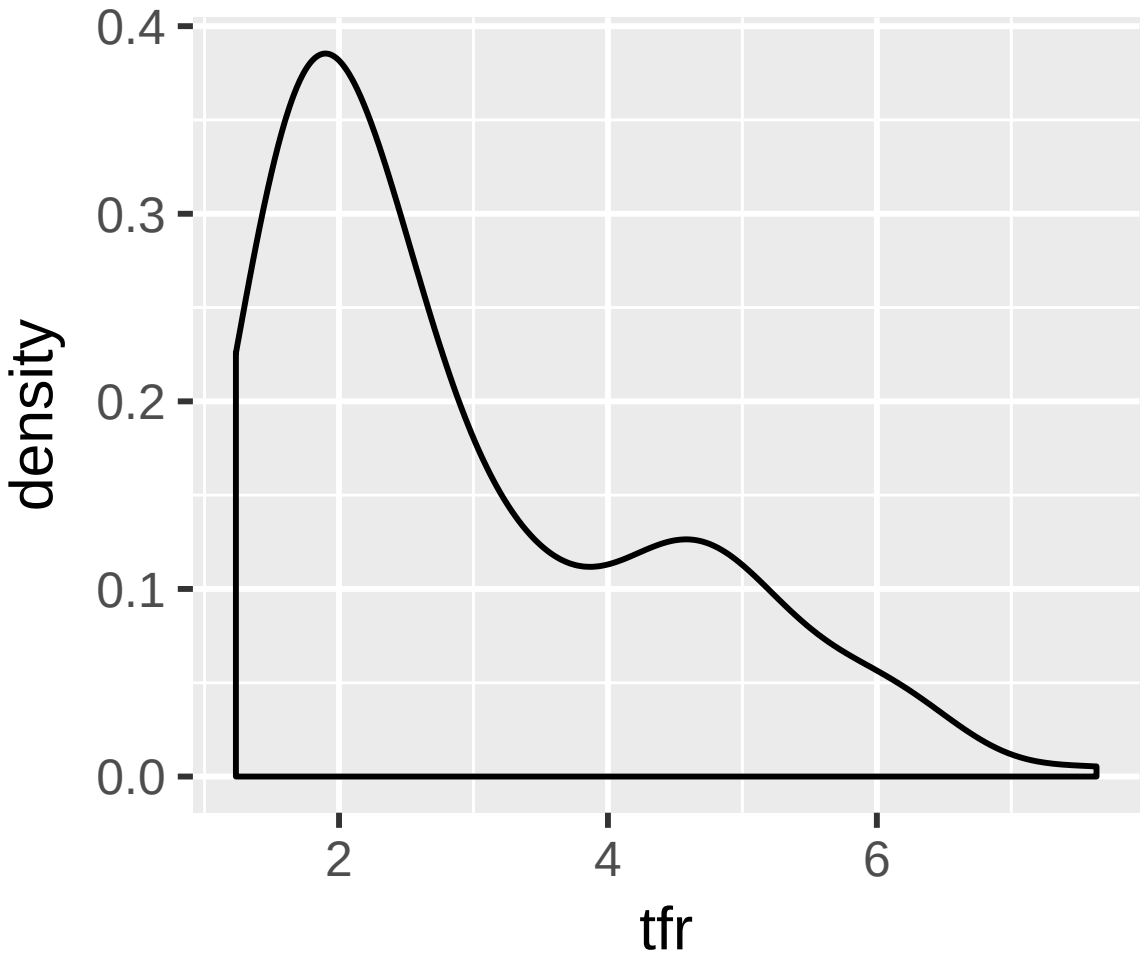
One Variable: Continuous - geom_histogram()

```
> # all countries 2010-15
> ggplot(data = df1, mapping = aes(x = tfr)) +
+   geom_histogram()
```



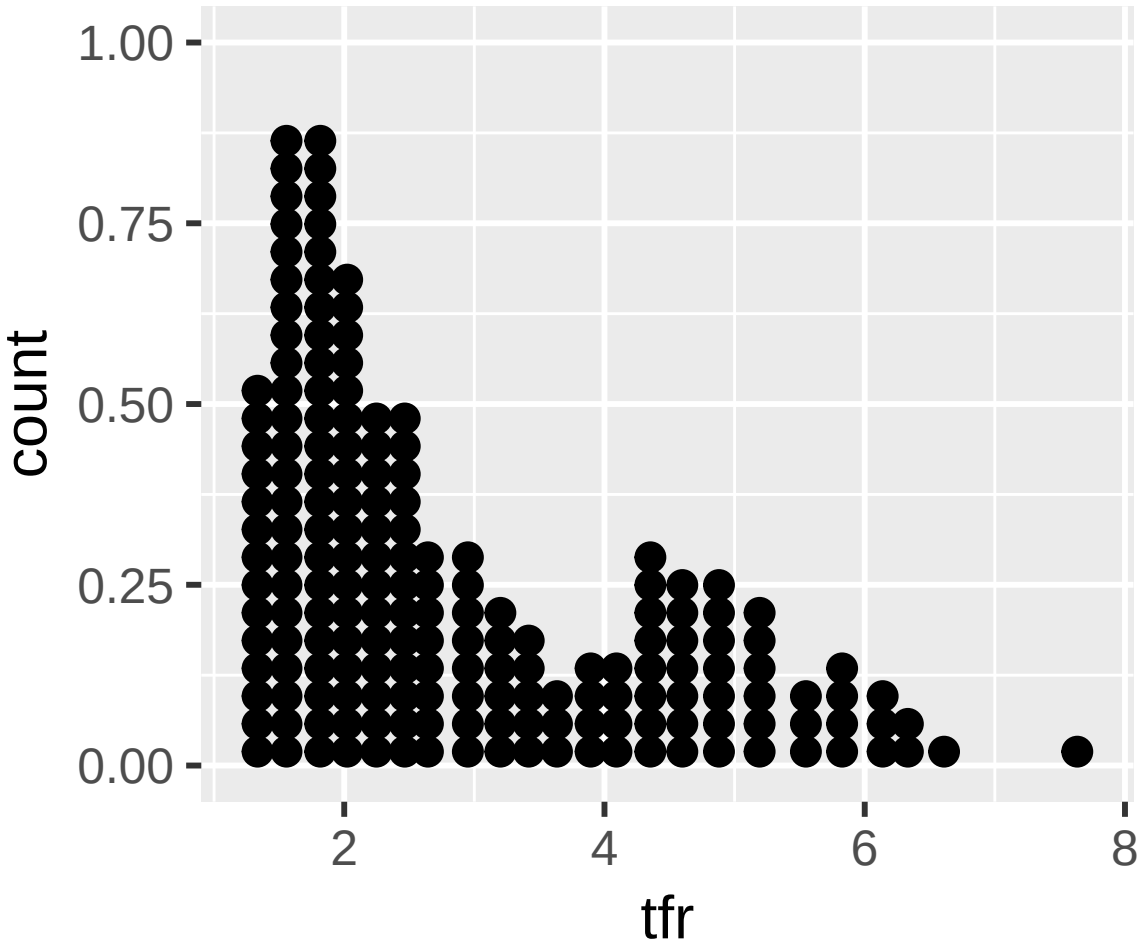
One Variable: Continuous - geom_density()

```
> # all countries 2010-15
> ggplot(data = df1, mapping = aes(x = tfr)) +
+   geom_density()
```



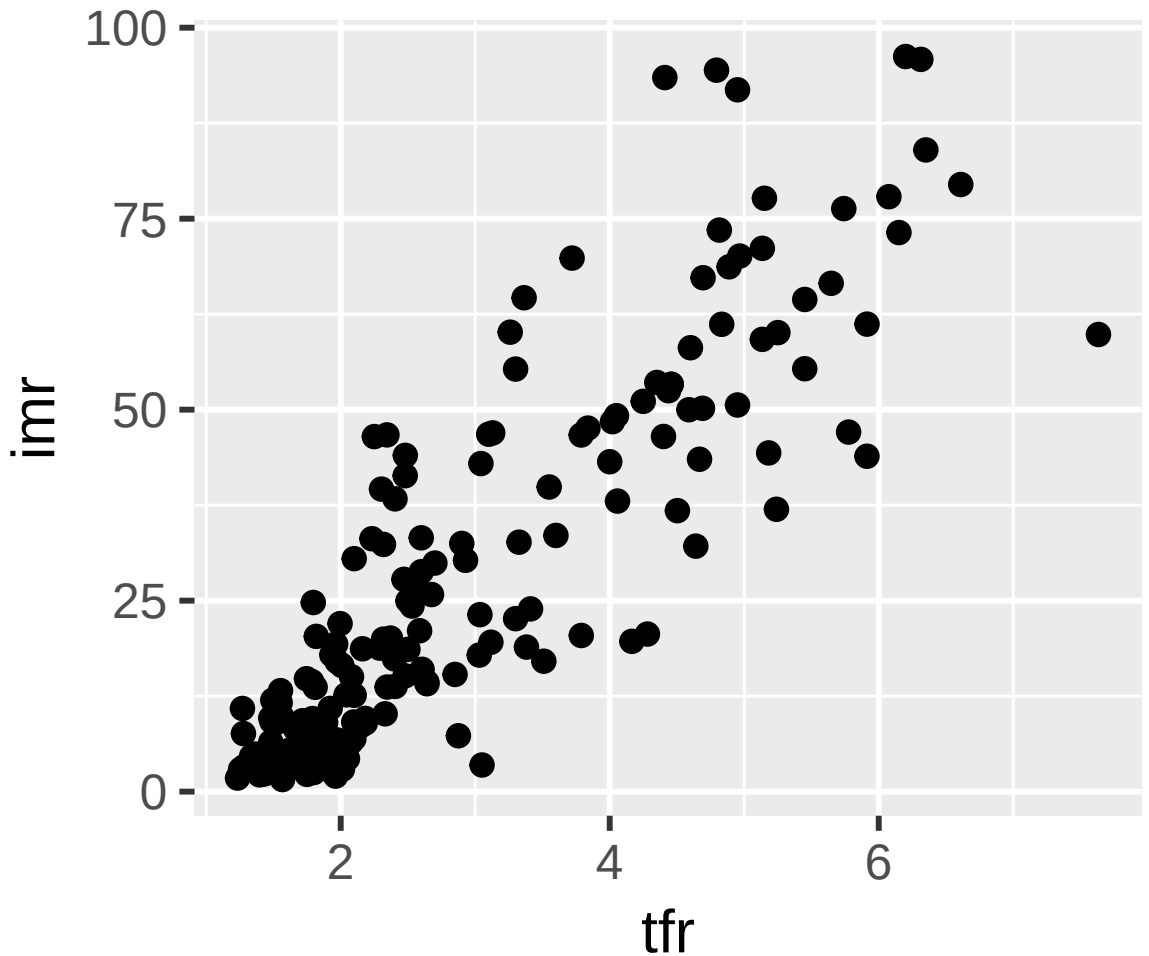
One Variable: Continuous - geom_dotplot()

```
> # all countries 2010-15
> ggplot(data = df1, mapping = aes(x = tfr)) +
+   geom_dotplot()
```



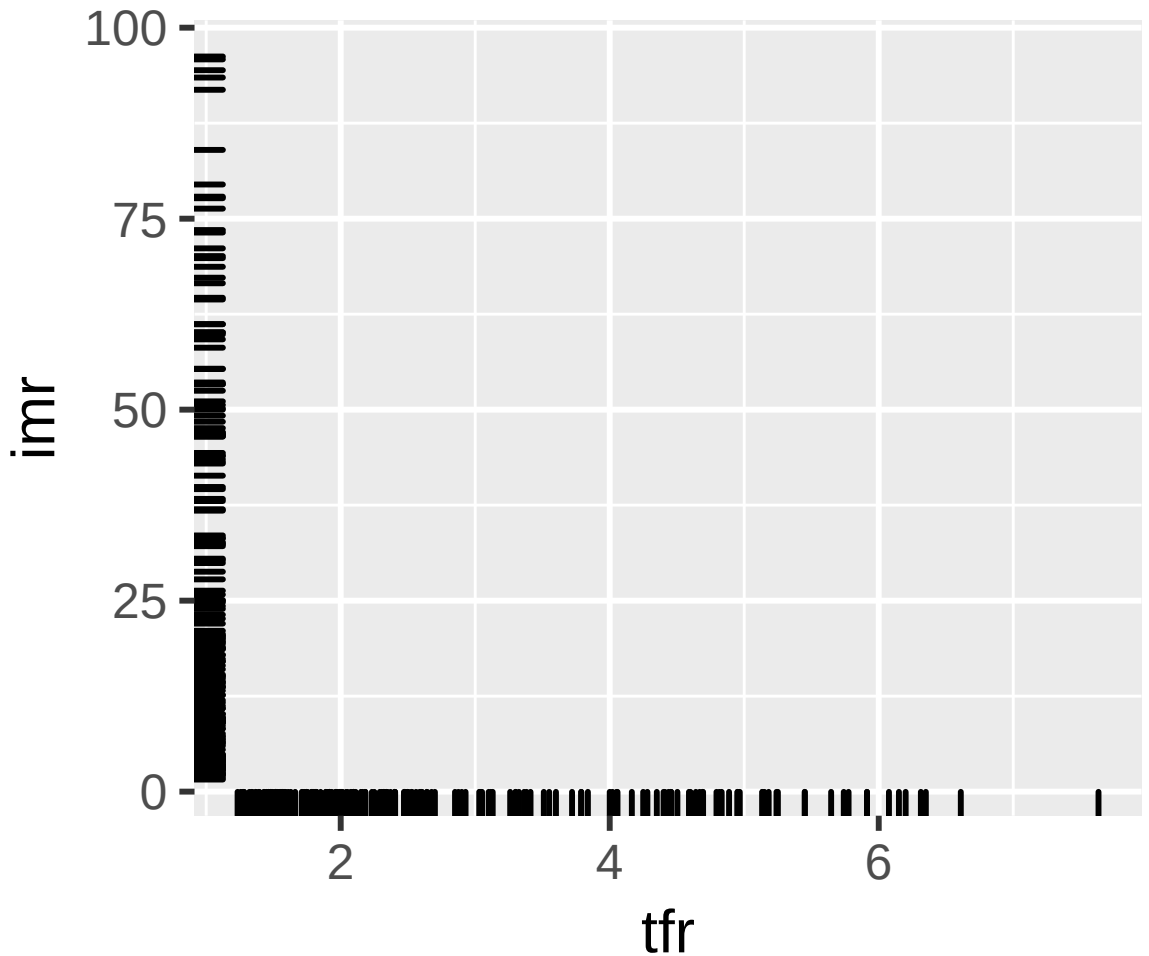
Two Variables: Both Continuous - geom_point()

```
> # all countries 2010-15
> ggplot(data = df1, mapping = aes(x = tfr, y = imr)) +
+   geom_point()
```



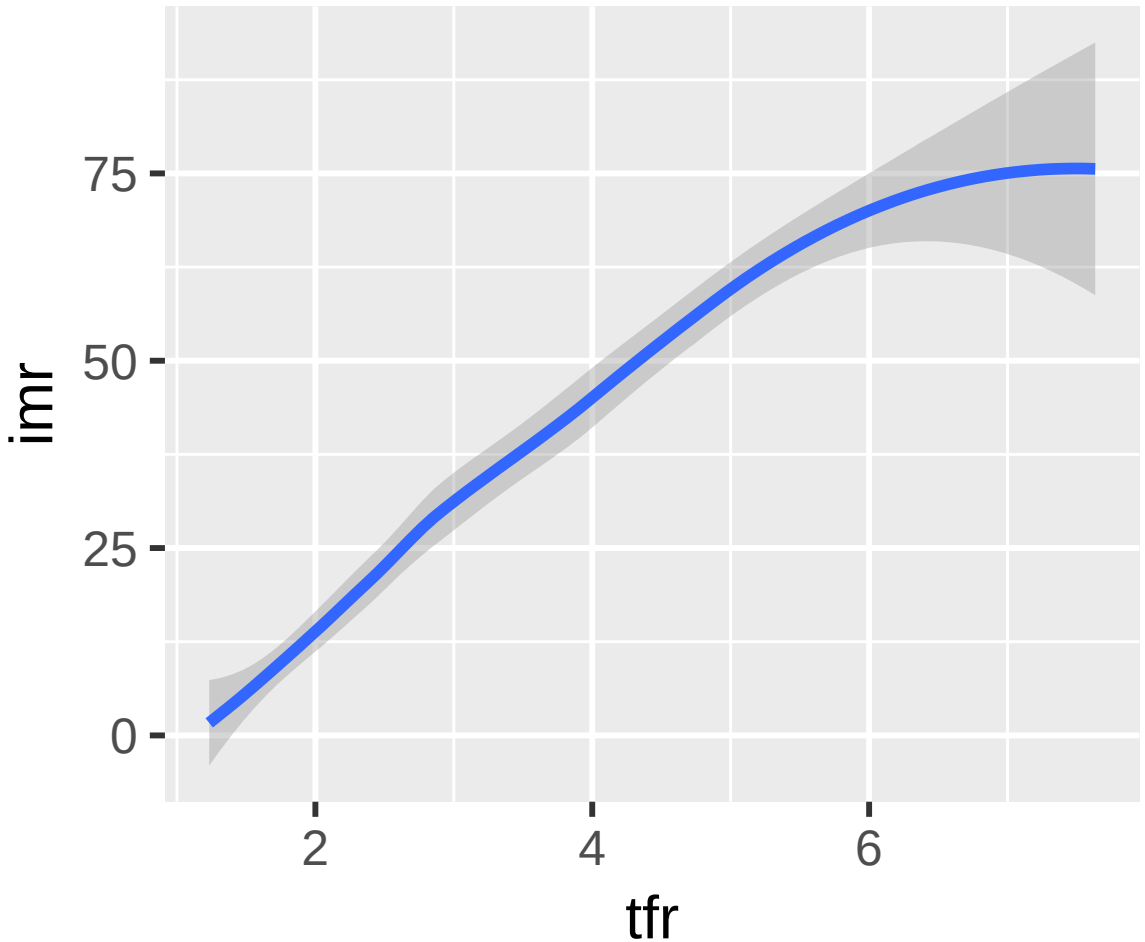
Two Variables: Both Continuous - geom_rug()

```
> # all countries 2010-15
> ggplot(data = df1, mapping = aes(x = tfr, y = imr)) +
+   geom_rug()
```



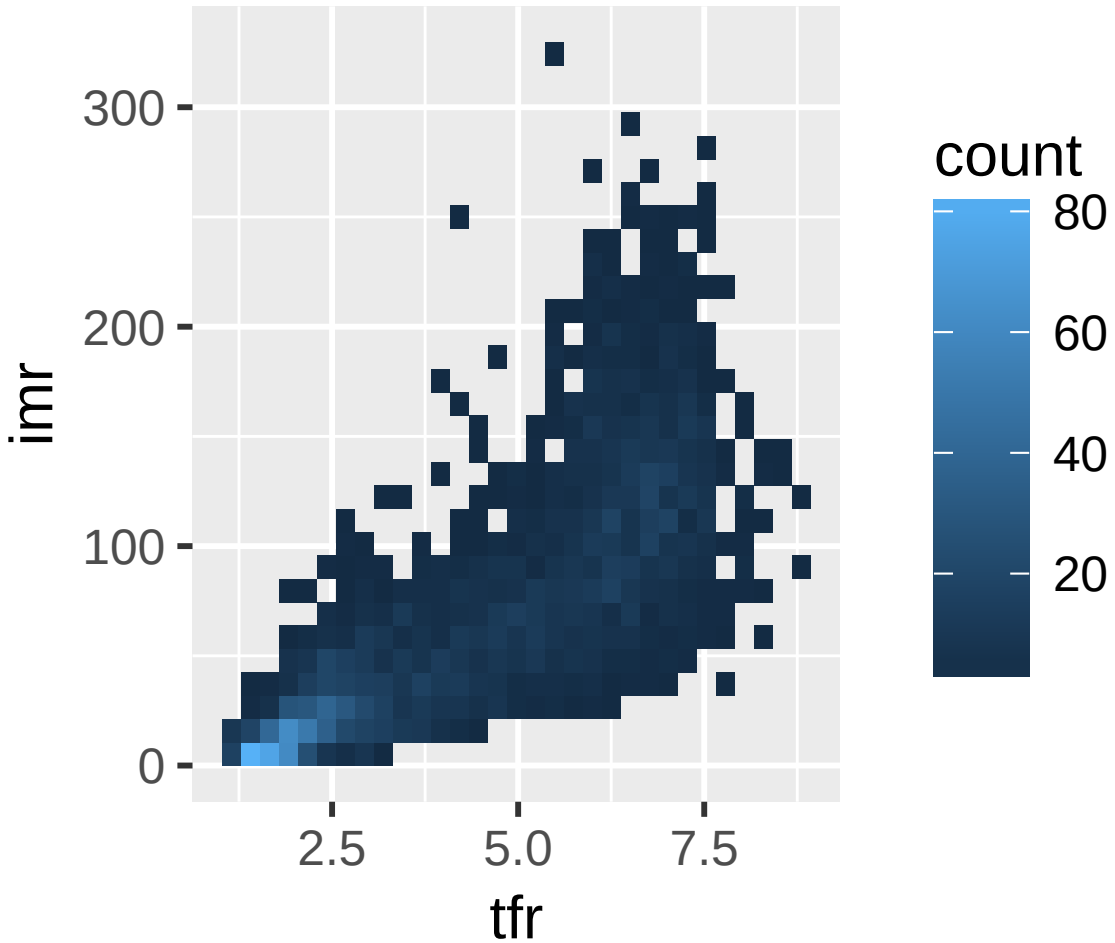
Two Variables: Both Continuous - geom_smooth()

```
> # all countries 2010-15
> ggplot(data = df1, mapping = aes(x = tfr, y = imr)) +
+   geom_smooth()
```



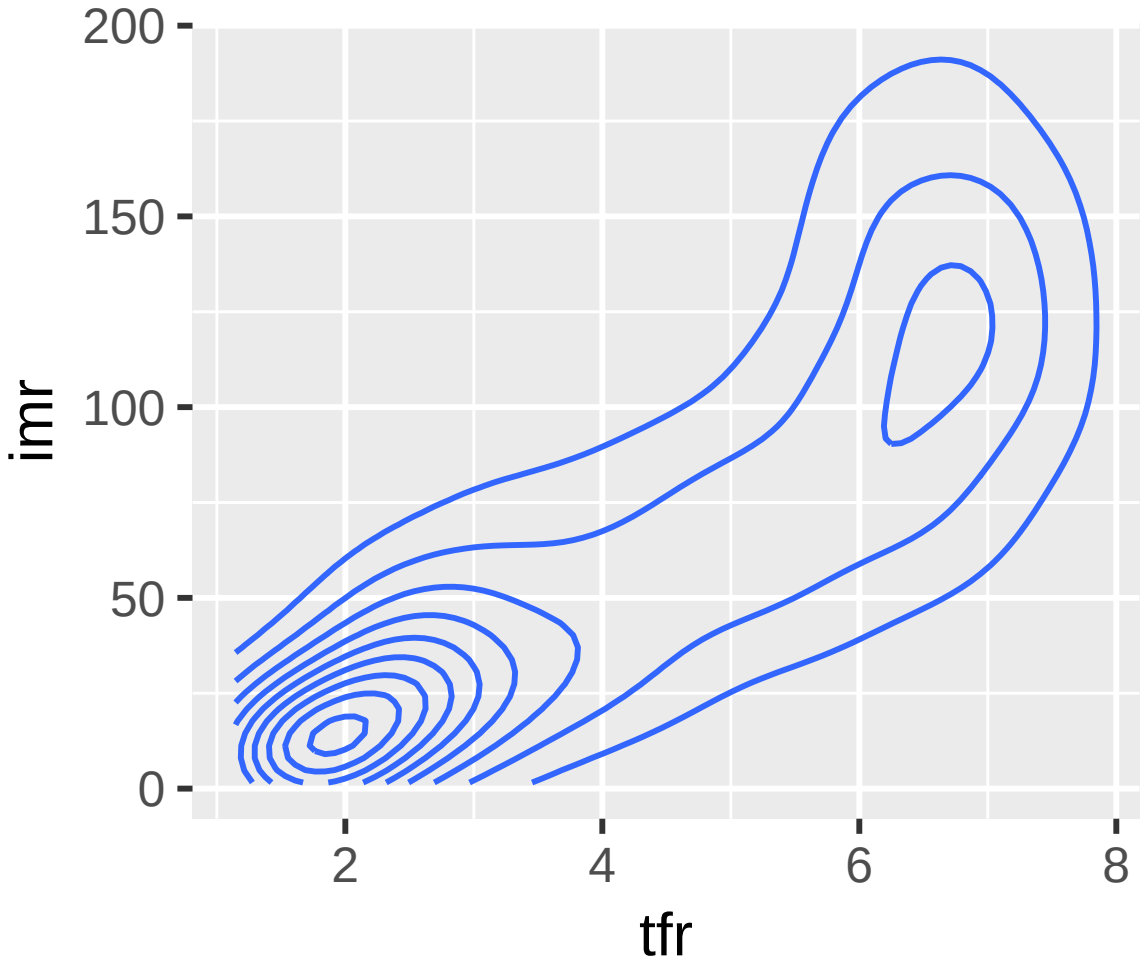
Two Variables: Both Continuous - geom_bin2d()

```
> # all countries all periods
> ggplot(data = df0, mapping = aes(x = tfr, y = imr)) +
+   geom_bin2d()
```



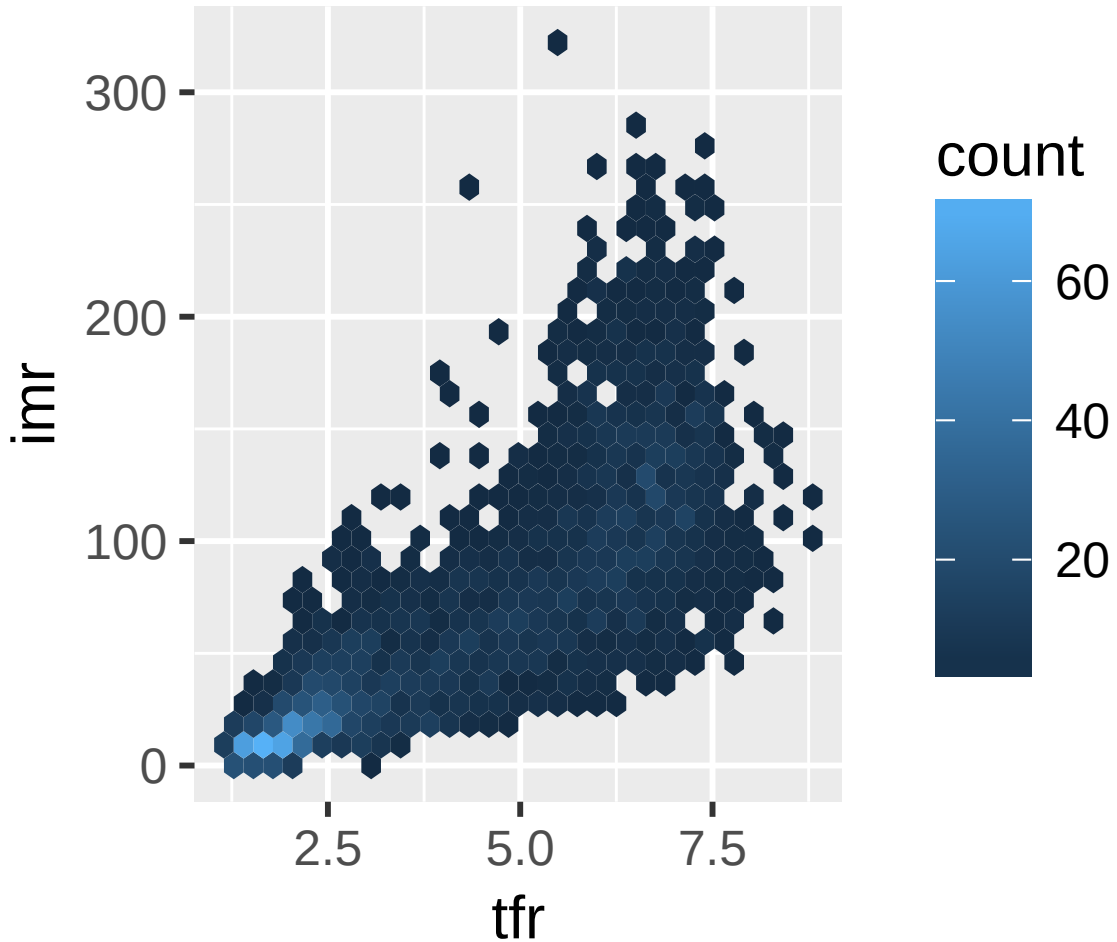
Two Variables: Both Continuous - geom_density2d()

```
> # all countries all periods
> ggplot(data = df0, mapping = aes(x = tfr, y = imr)) +
+   geom_density2d()
```



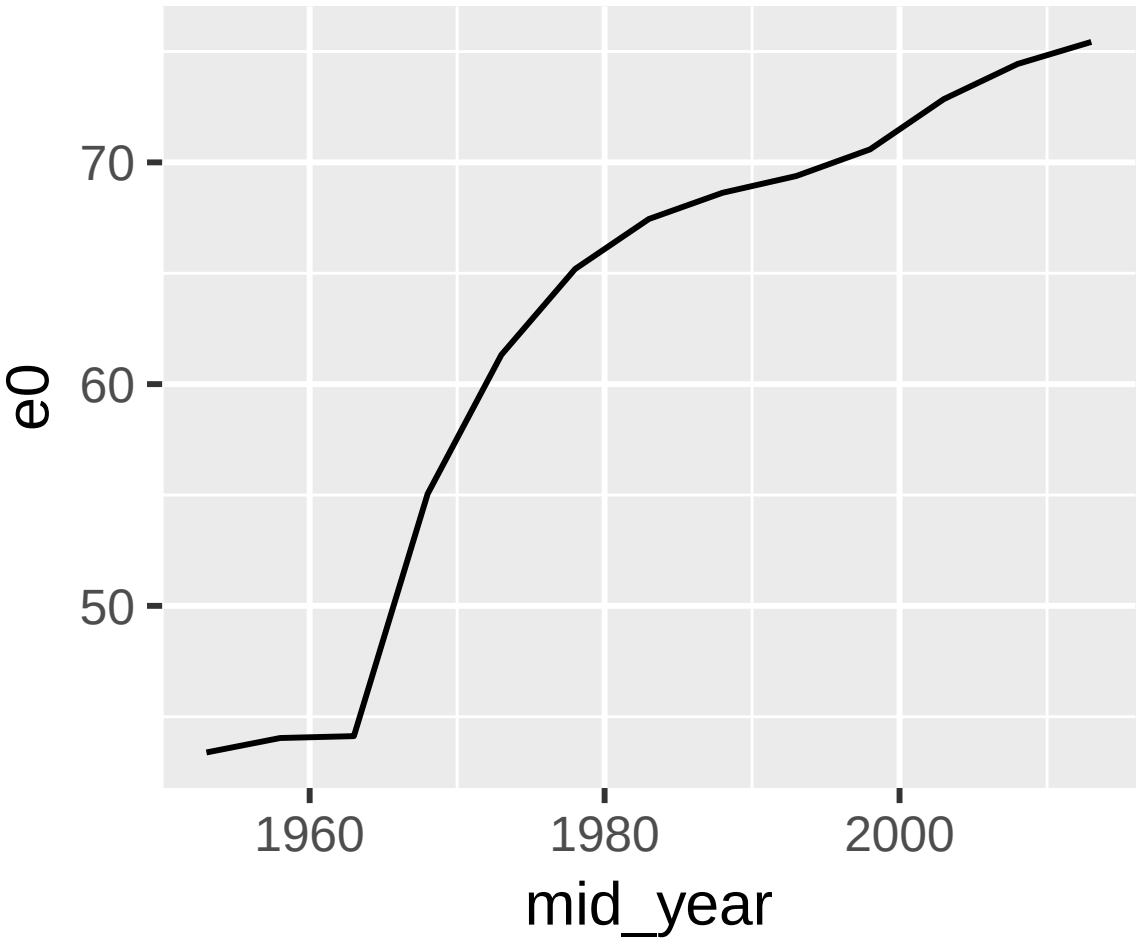
Two Variables: Both Continuous - geom_hex()

```
> # install.packages("hexbin")
> ggplot(data = df0, mapping = aes(x = tfr, y = imr)) +
+   geom_hex()
```



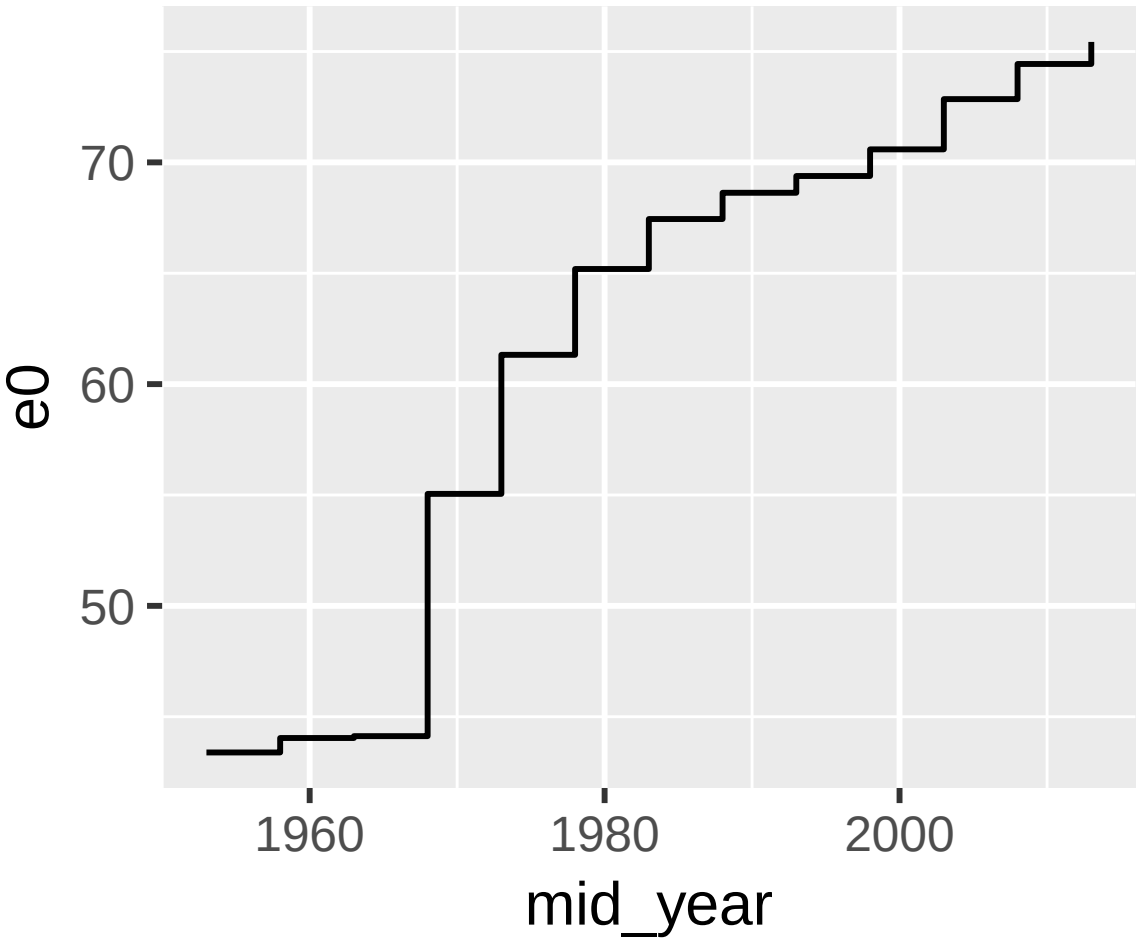
Two Variables: Continuous and Time - geom_line()

```
> df2 <- filter(df0, name == "China", !is.na(e0))
> ggplot(data = df2, mapping = aes(x = mid_year, y = e0)) +
+   geom_line()
```



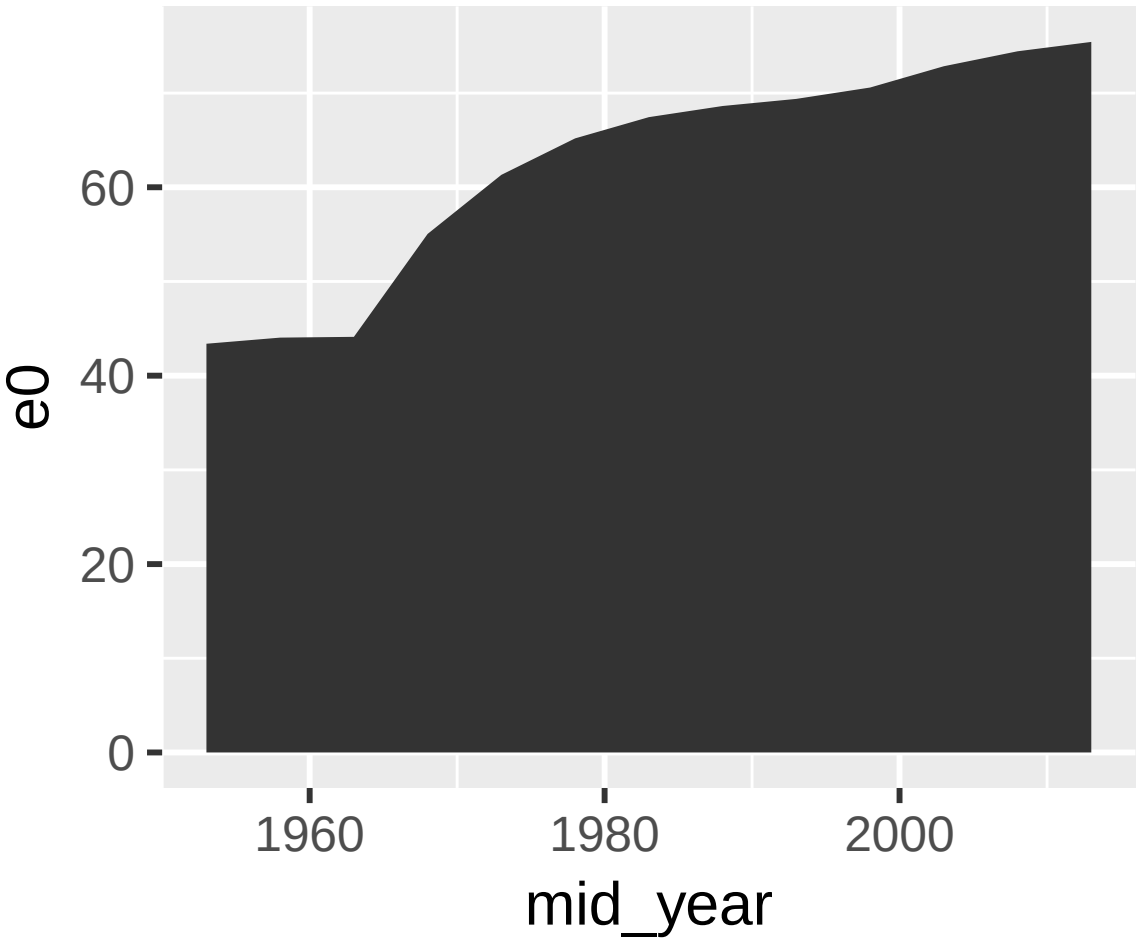
Two Variables: Continuous and Time - geom_step()

```
> # china 1960-65 to 2010-15
> ggplot(data = df2, mapping = aes(x = mid_year, y = e0)) +
+   geom_step()
```



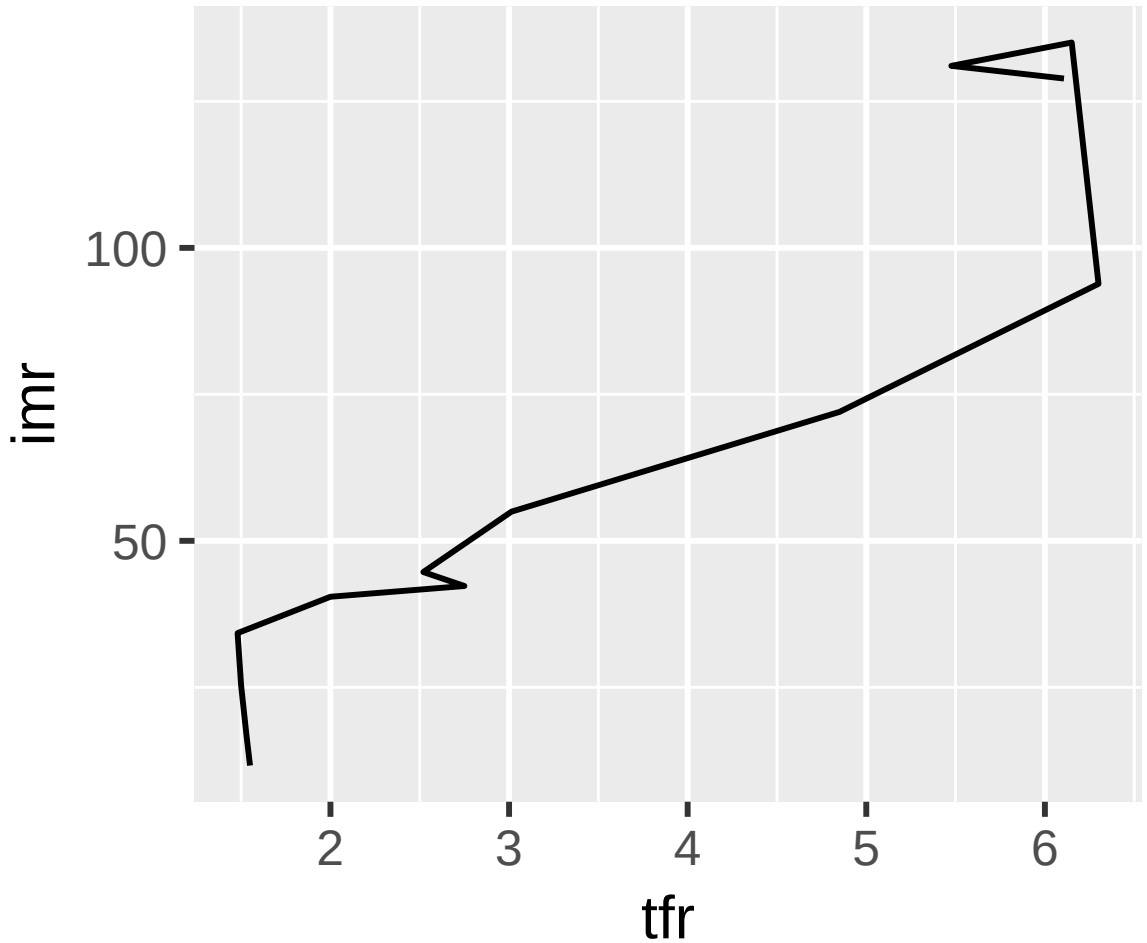
Two Variables: Continuous and Time - geom_bar()

```
> # china 1960-65 to 2010-15
> ggplot(data = df2, mapping = aes(x = mid_year, y = e0)) +
+   geom_area()
```



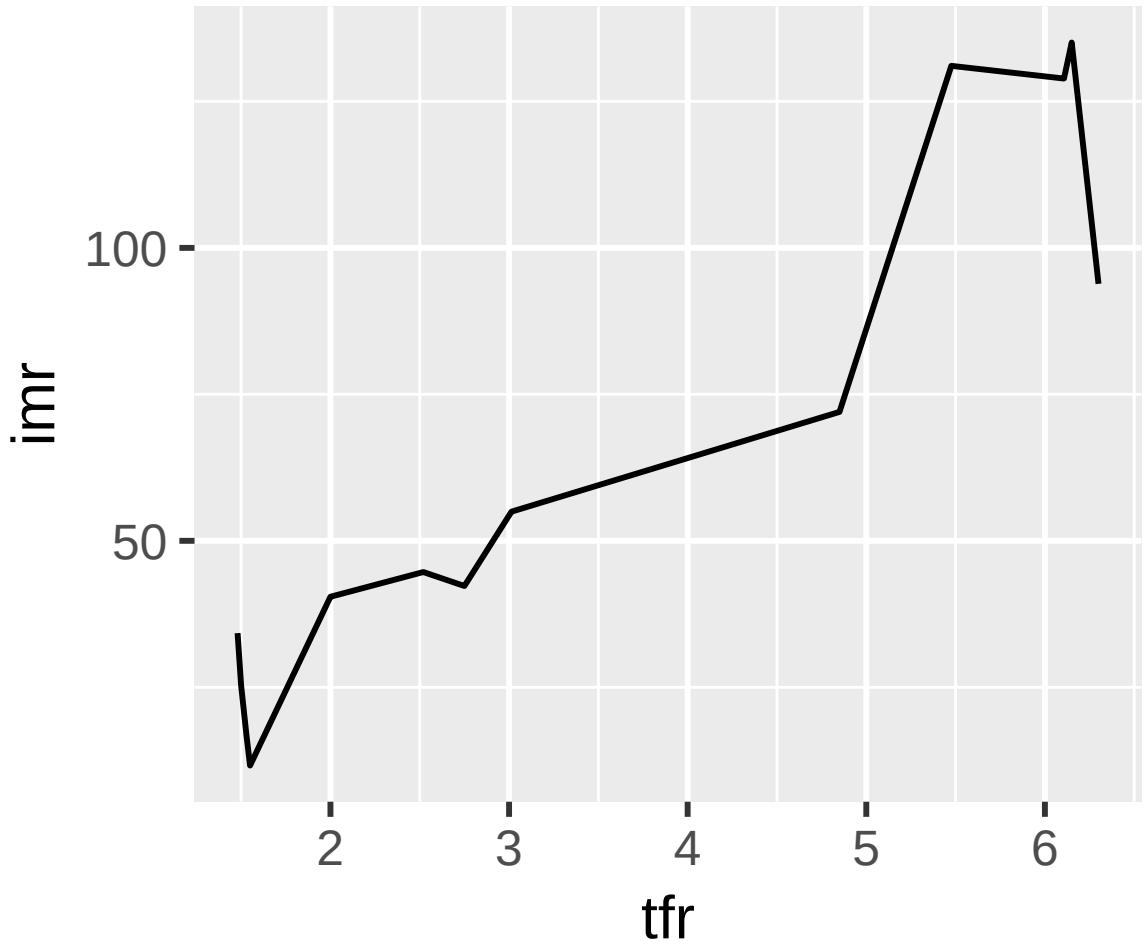
Two Variables: Continuous and Time - geom_path()

```
> # geom_path() follows order in data
> ggplot(data = df2, mapping = aes(x = tfr, y = imr)) +
+   geom_path()
```



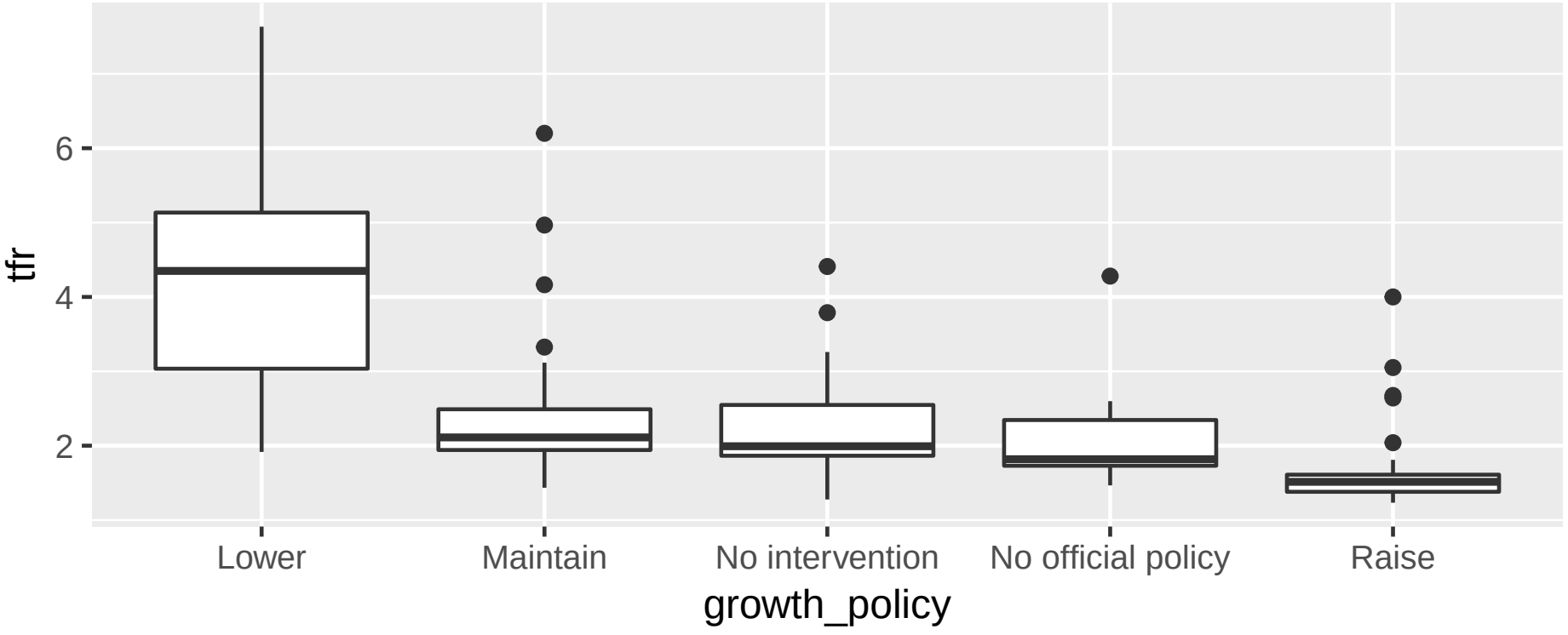
Two Variables: Continuous and Time - geom_path()

```
> # geom_line() follows order on x-axis (fine if time, if not need geom_path())
> ggplot(data = df2, mapping = aes(x = tfr, y = imr)) +
+   geom_line()
```



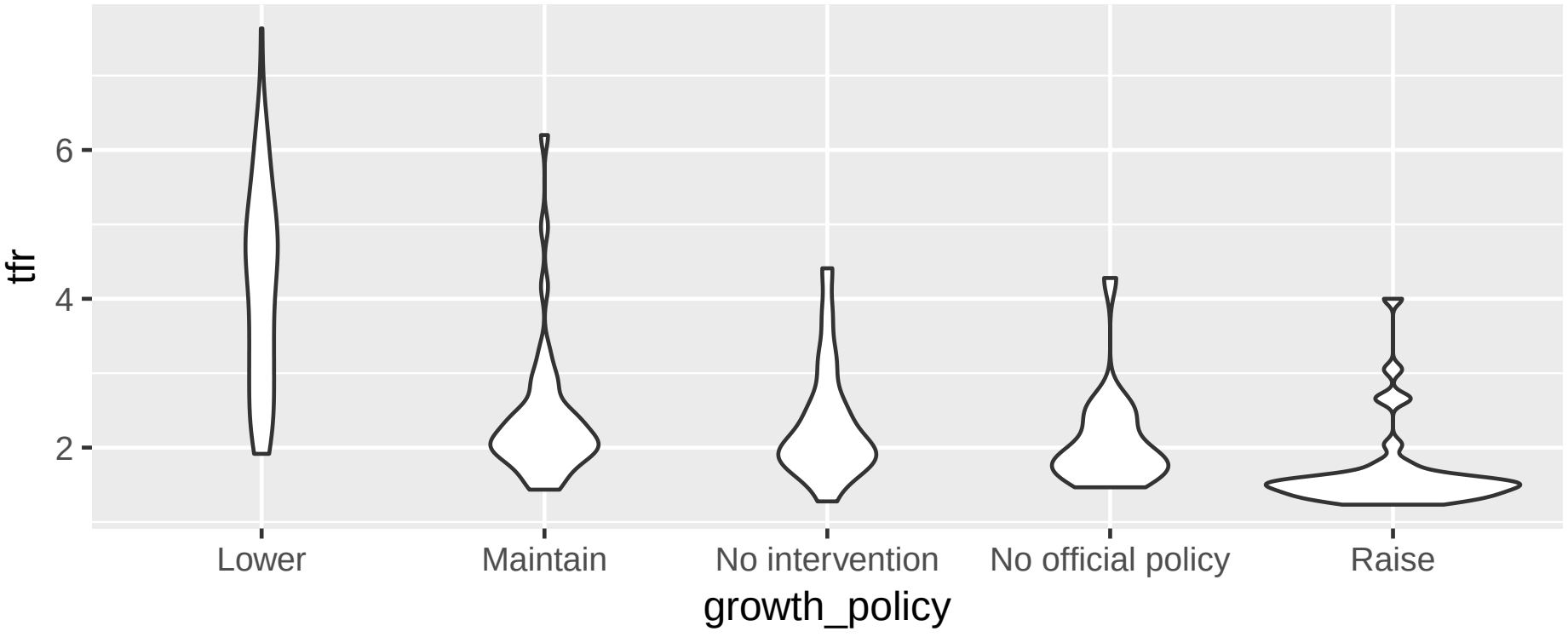
Two Variables: Continuous and Discrete - geom_boxplot()

```
> # all countries 2010-15
> ggplot(data = df1, mapping = aes(x = growth_policy, y = tfr)) +
+   geom_boxplot()
```



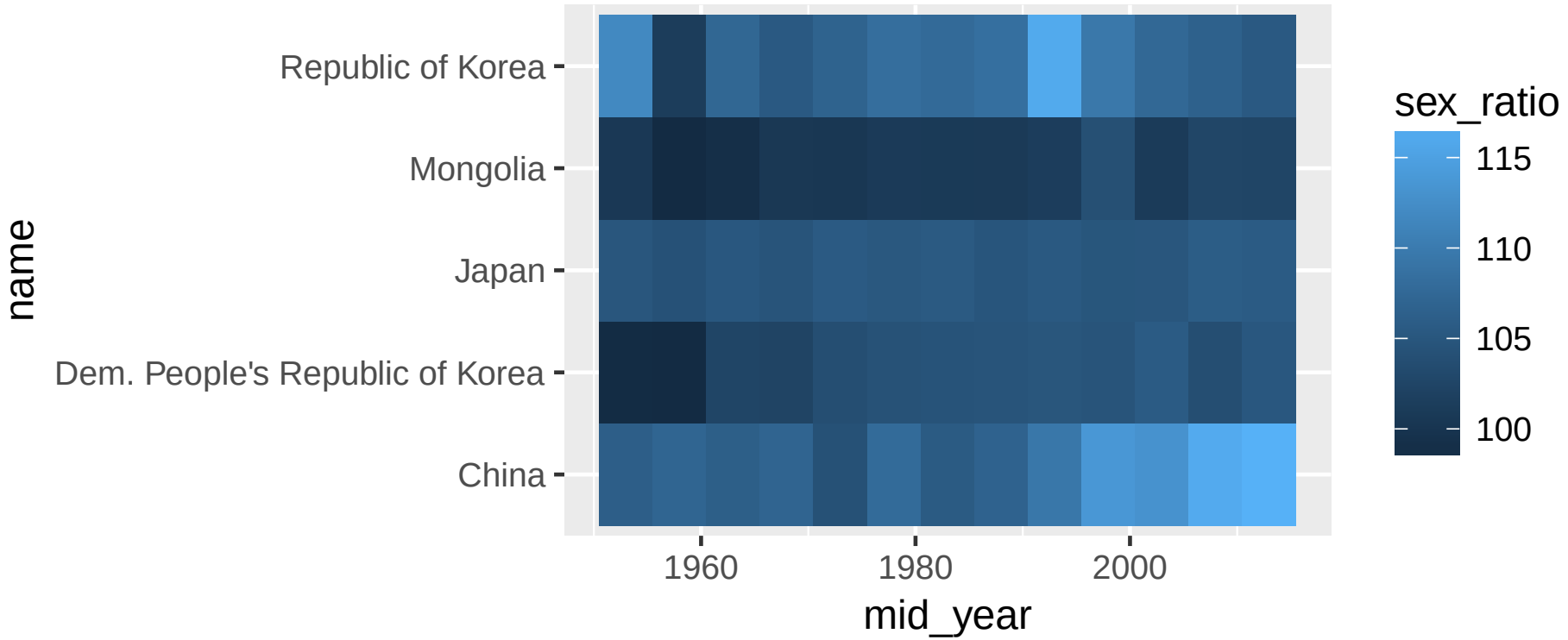
Two Variables: Continuous and Discrete - geom_violin()

```
> # all countries 2010-15
> ggplot(data = df1, mapping = aes(x = growth_policy, y = tfr)) +
+   geom_violin()
```



Three Variables: Continuous/Discrete Mix - geom_tile()

```
> df3 <- filter(df0, reg_name == "Eastern Asia", !is.na(mid_year))
> ggplot(data = df3, mapping = aes(x = mid_year, y = name, fill = sex_ratio)) +
+   geom_tile()
```

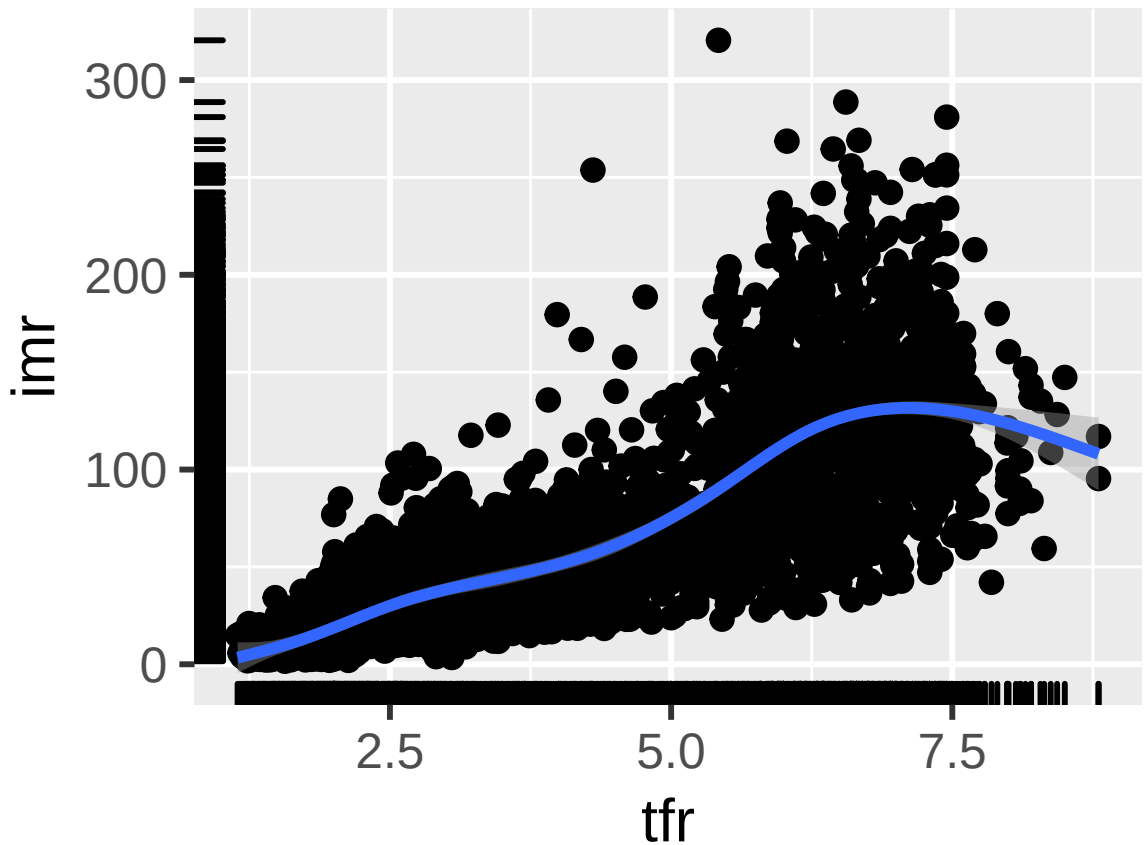


Layers

- Geoms can be added in layers
  - Adds more detail the plots
- The `data` and `mapping` aesthetics in the `ggplot` function are shared by all the geom functions
- To overwrite for a individual geom you can place `mappings` or `data` arguments in the geom function
  - This will treat mappings or data local to the layer only.

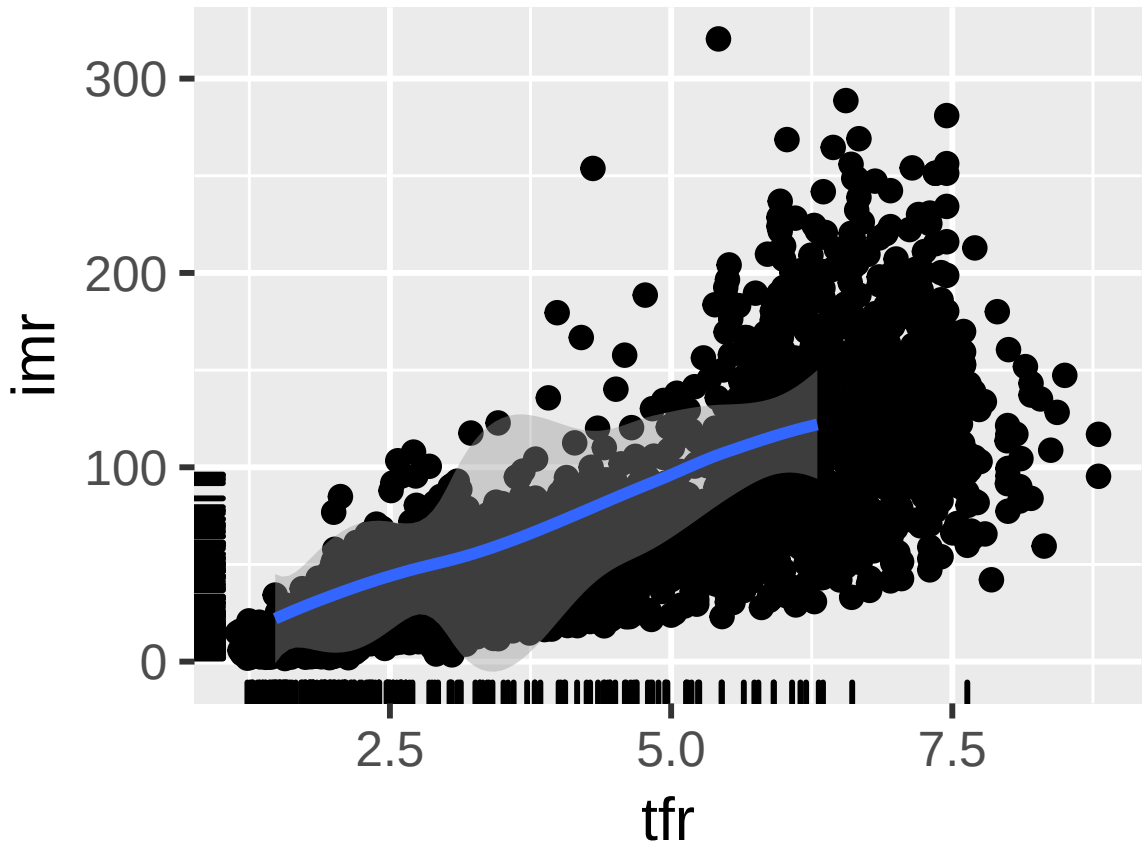
Layers

```
> # layers on all countries all periods
> ggplot(data = df0, mapping = aes(x = tfr, y = imr)) +
+   geom_point() +
+   geom_rug() +
+   geom_smooth()
```



Layers

```
> # points for on all countries all periods (df0), rug for 2010-15 data (df1)
> # smooth line for china (df2) only
> ggplot(data = df0, mapping = aes(x = tfr, y = imr)) +
+   geom_point() +
+   geom_rug(data = df1) +
+   geom_smooth(data = df2)
```

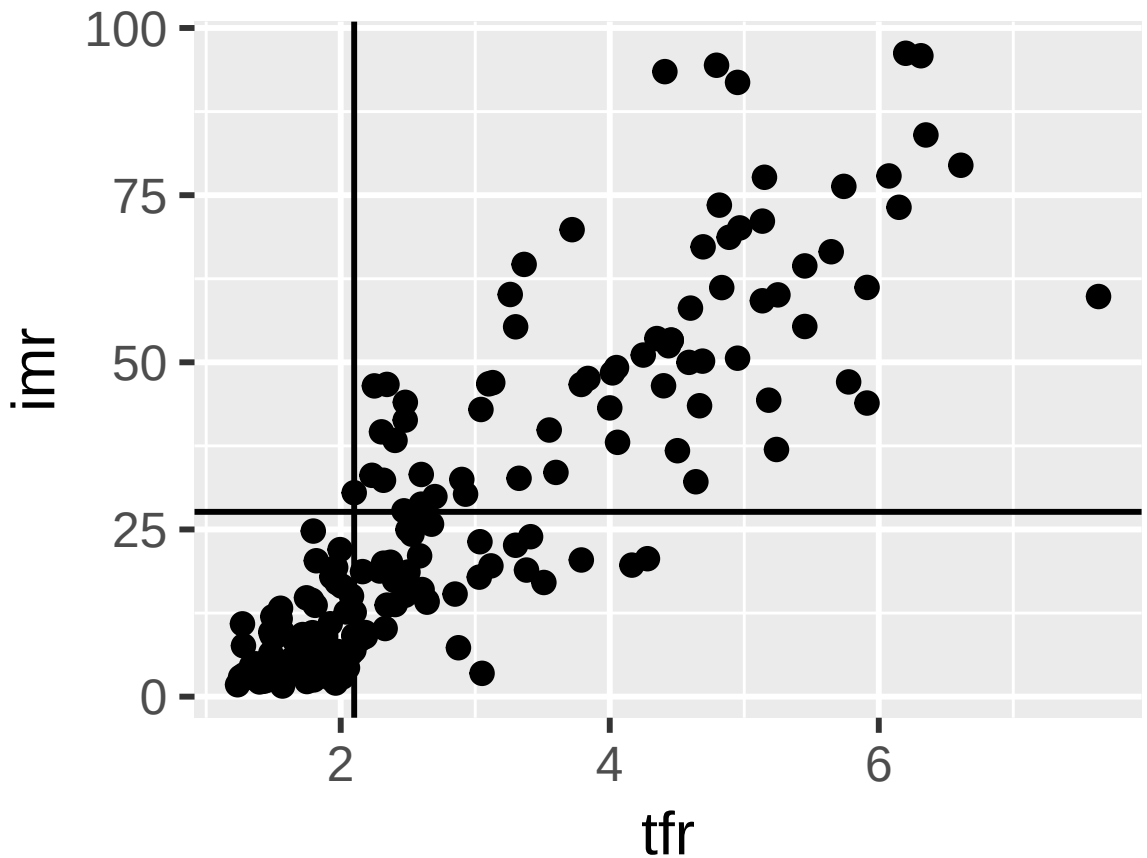


Layers

- Line geoms can help indicate important thresholds
  - The `geom_hline()` function for horizontal line with mapping `yintercept`
  - The `geom_vline()` function for vertical line with mapping `xintercept`
  - The `geom_abline()` function for a line with mappings `intercept` and `slope`

Layers

```
> # all countries 2010-15
> ggplot(data = df1, mapping = aes(x = tfr, y = imr)) +
+   geom_point() +
+   geom_vline(mapping = aes(xintercept = 2.1)) +
+   geom_hline(mapping = aes(yintercept = mean(imr)))
```



Exercise 1 (ex21.R)

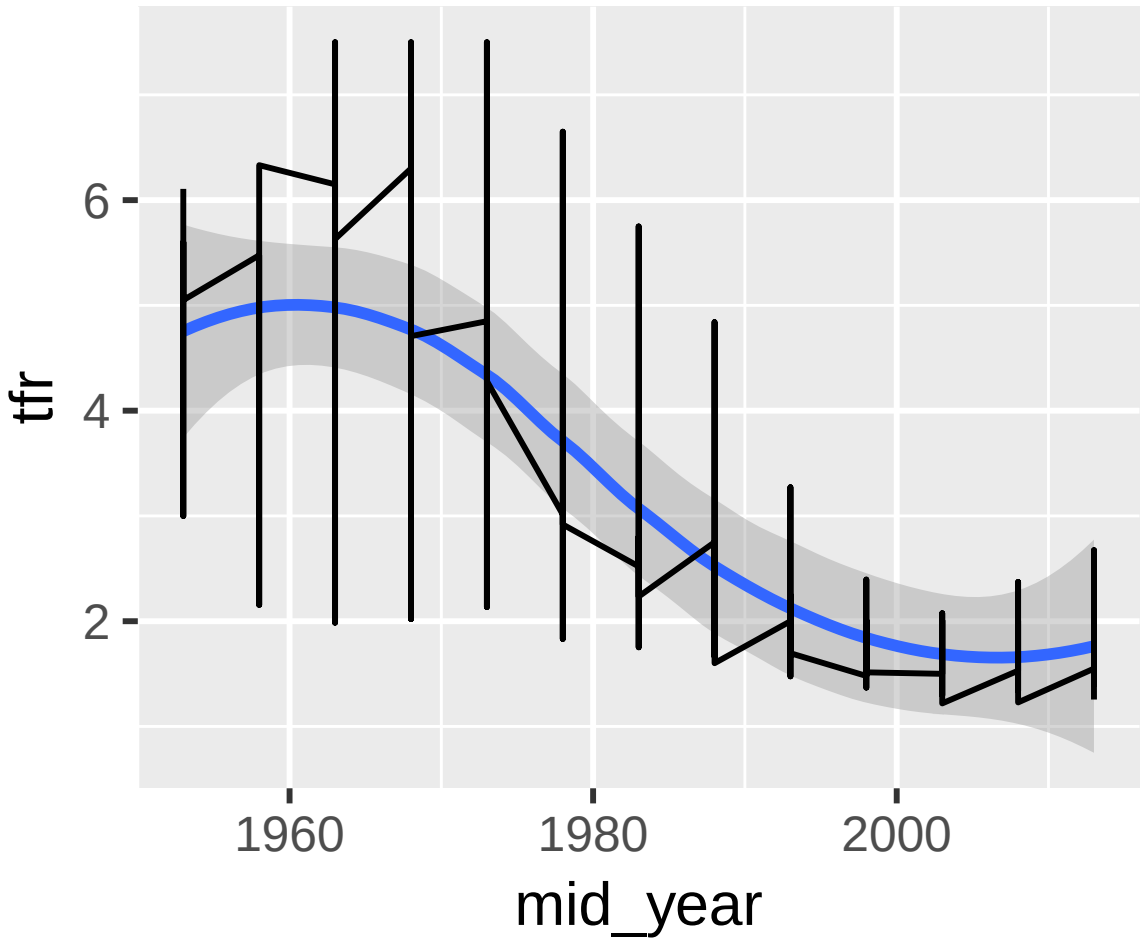
```
# 0.  a) Clear your workspace and set the working directory to the course folder on
rm(list = #####)
setwd(dir = "C:/Users/Guy/Dropbox/SHU2017-3XS371026/")
#     b) Load the tidyverse package (which loads the ggplot2 package amongst others)
library(#####)
#     c) Run the following code to import the UN data
# d0 <- read_csv("./exercise/data/wpp_all.csv")
# d1 <- filter(d0, period == "2010-2015")
# d2 <- filter(d0, alpha3 == "GBR", !is.na(period))
# d3 <- filter(d0, reg_name == "South-Eastern Asia", !is.na(period))
# d4 <- filter(d0, area_name == "Asia", !is.na(period))
# # all data
# d0
# # 2015 data
# d1
# # UK data
# d2
# # South Eastern Asia data
# d3
# # Asia data
# d4
##
##
##
```

Groups

- Many `geoms_` functions use a single object to describe all of the data.
- You can set the `group` aesthetic in the `mappings` to a **discrete** variable to draw multiple objects.
  - This will tell ggplot2 to draw separate object for each unique value of the grouping variable.
- Will not add a legend or distinguishing features to the plot.
  - These are added when we have some more aesthetics for the `mappings`, e.g. `colour` or `linetype` (will get to these soon).

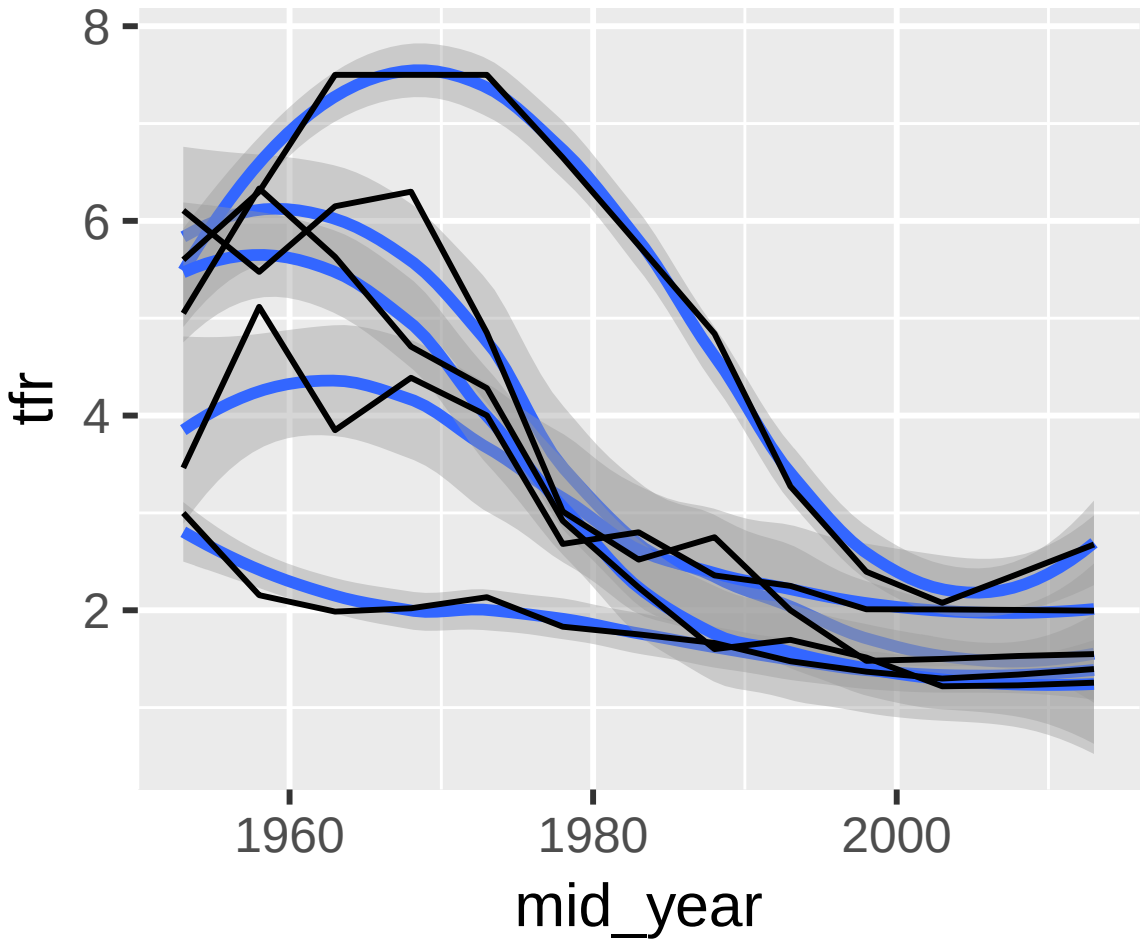
Groups

```
> # eastern asia: one smooth line and one data line through all points
> ggplot(data = df3, mapping = aes(x = mid_year, y = tfr)) +
+   geom_smooth() +
+   geom_line()
```



Groups

```
> # eastern asia: smooth line and data line for each country (identified by name)
> ggplot(data = df3, mapping = aes(x = mid_year, y = tfr, group = name)) +
+   geom_smooth() +
+   geom_line()
```



Optional Aesthetics

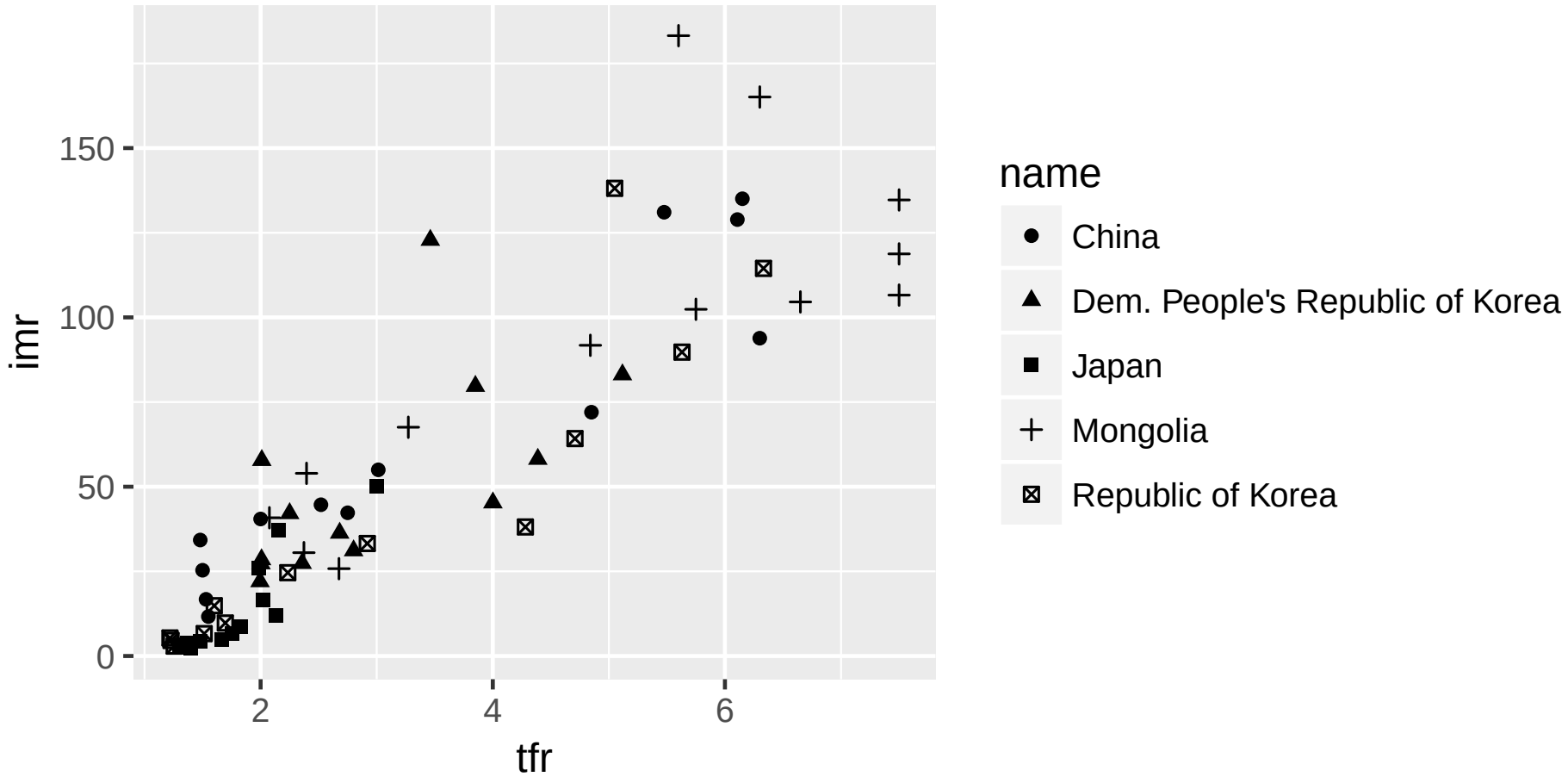
- Beyond `group`, there are other aesthetics (`aes`) that can be used depending on the geom, for example:
  - `shape`: shape of point
  - `linetype`: type of line used
  - `colour`: border colour
  - `size`: size of object
  - `fill`: internal colour
  - `alpha`: transparency (between 0 and 1)
  - and many more ...
- Not all aesthetics work with all geom.
  - For example `geom_point()` has no `linetype` option.
  - Check the cheatsheet to see what is available.

Optional Aesthetics

- You can set the aesthetic to a
  - Single value to change the appearance for all plotted objects
  - Variables (columns) in the data
- When set to variable names in the data
  - If discrete will operate by each group (category) of the variable.
  - If continuous will operate along the scale of the variable.
- Not all aesthetics can be set to continuous variables
  - For example, `linetype = mid_year`
- Can set different aesthetics to different variables, allowing you to show more many dimensions of your data.

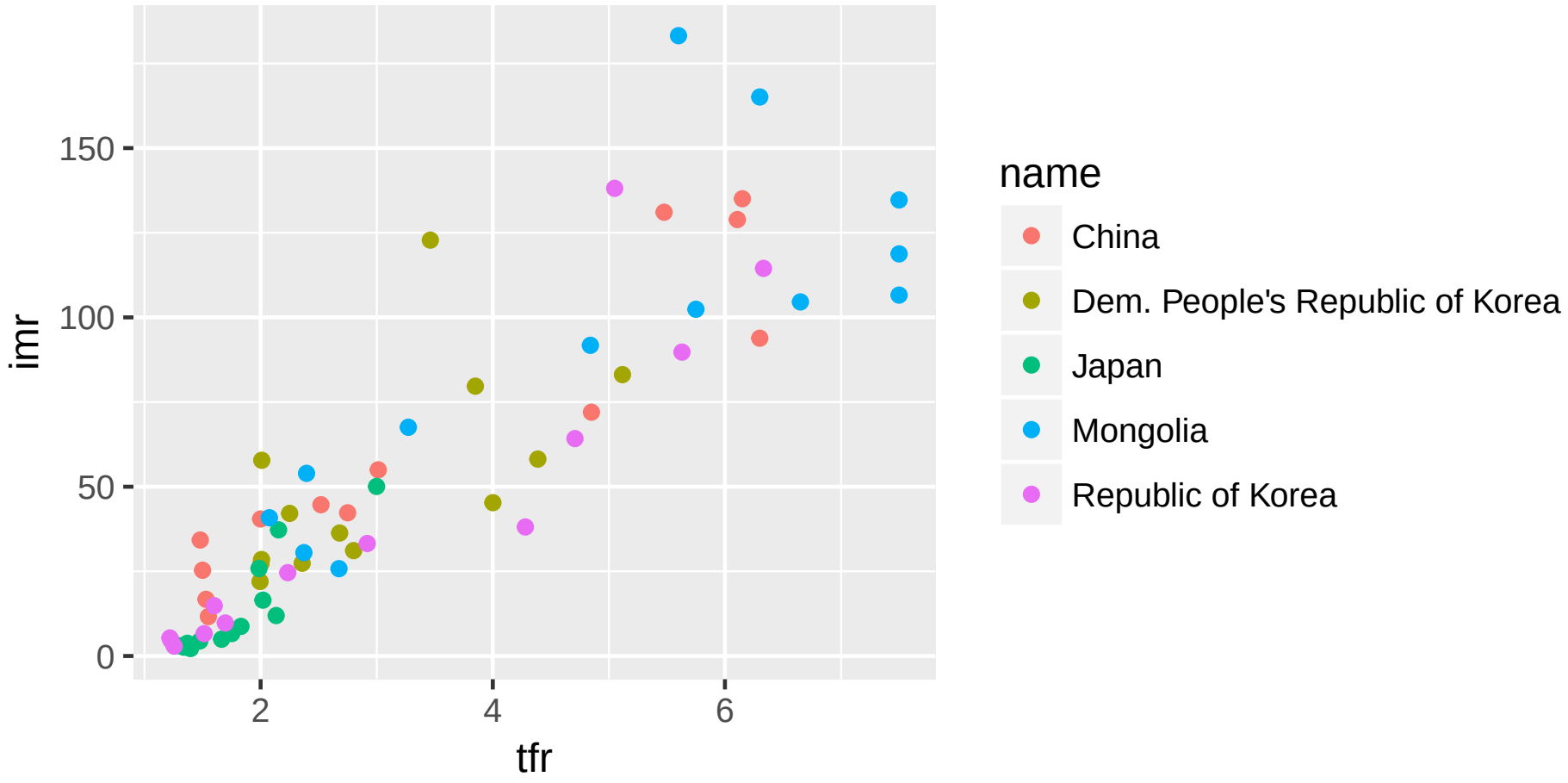
Optional Aesthetics

```
> # eastern asia: set shape from discrete variable
> ggplot(data = df3, mapping = aes(x = tfr, y = imr, shape = name)) +
+   geom_point()
```



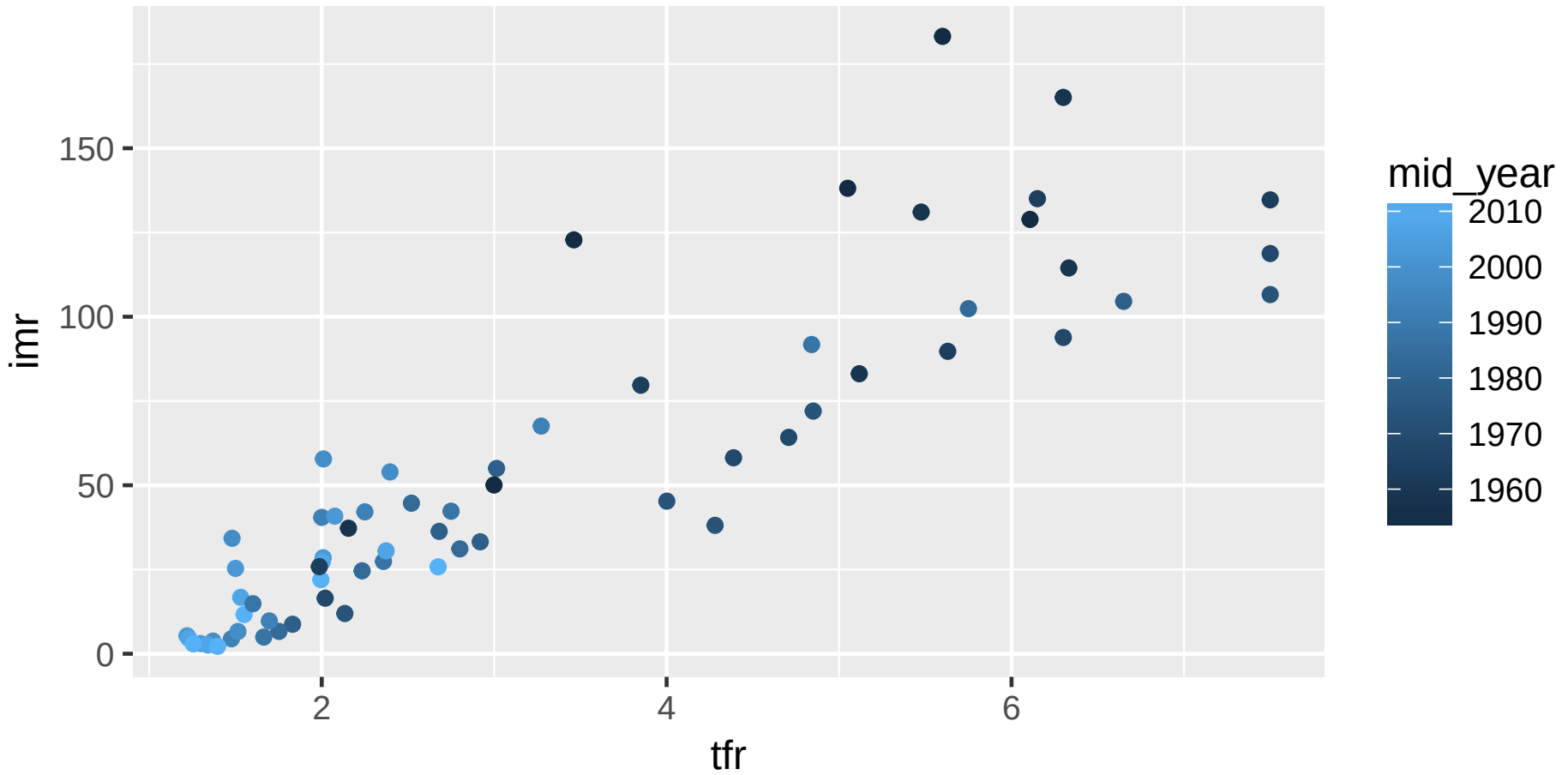
Optional Aesthetics

```
> # eastern asia: set colour from continuous variable
> ggplot(data = df3, mapping = aes(x = tfr, y = imr, colour = name)) +
+   geom_point()
```



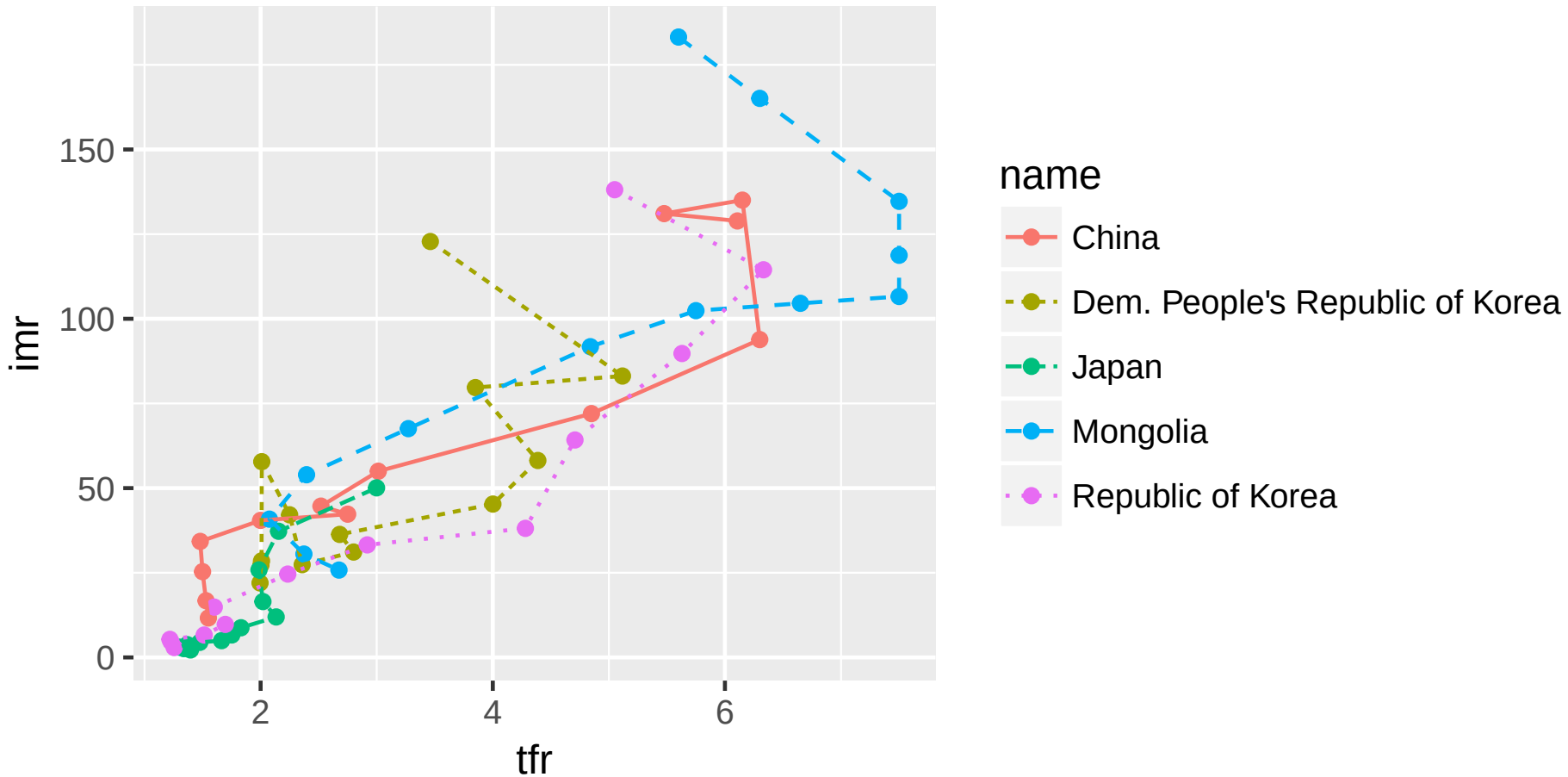
Optional Aesthetics

```
> # eastern asia: set colour from continuous variable
> ggplot(data = df3, mapping = aes(x = tfr, y = imr, colour = mid_year)) +
+   geom_point()
```



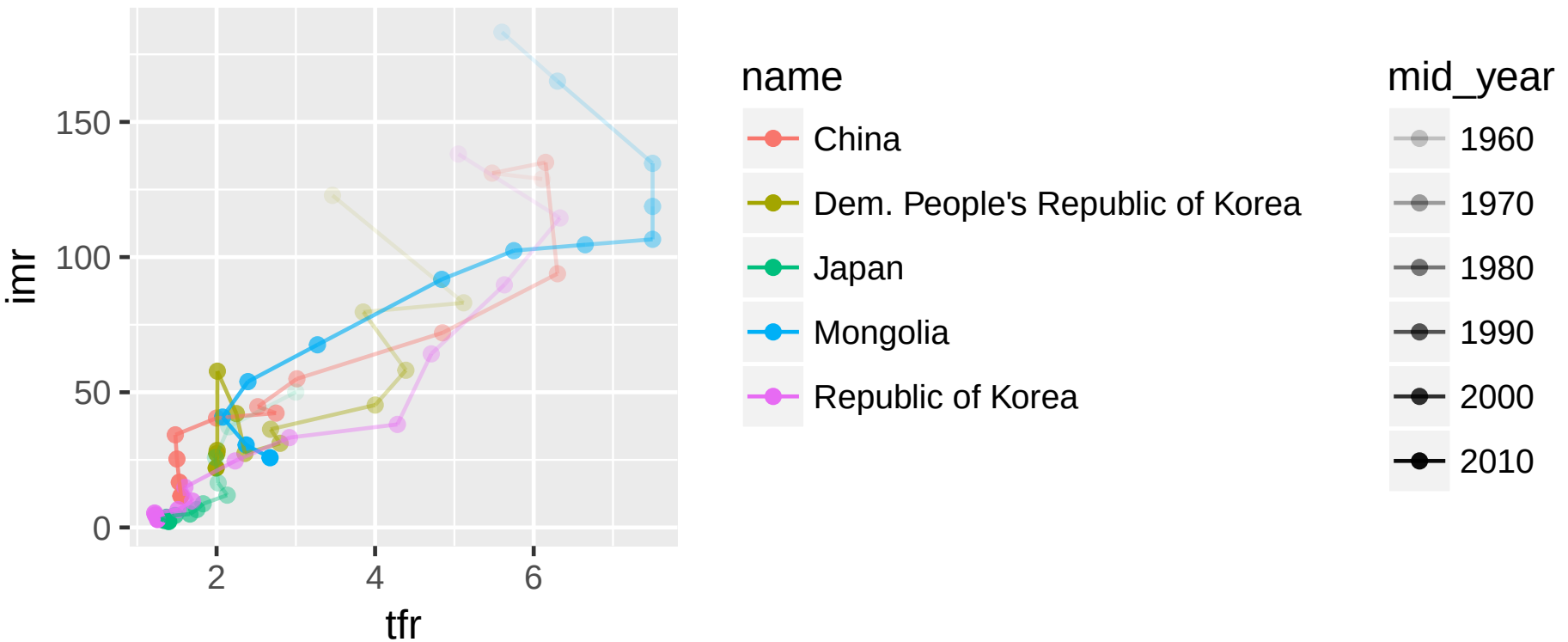
Optional Aesthetics

```
> # eastern asia: set colour and linetype from discrete variable
> ggplot(data = df3,
+        mapping = aes(x = tfr, y = imr, colour = name, linetype = name)) +
+   geom_point() +
+   geom_path()
```



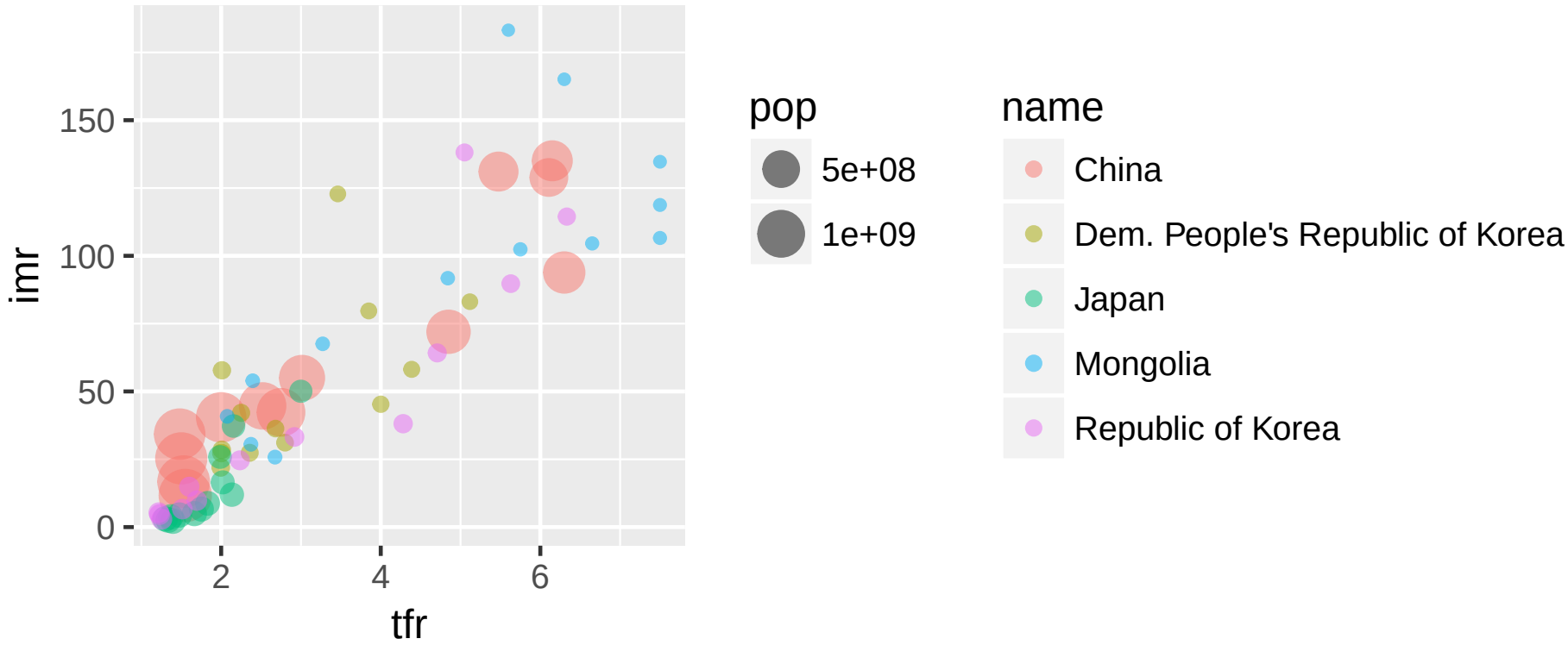
Optional Aesthetics

```
> # eastern asia: set colour from discrete and alpah from continuous variable
> ggplot(data = df3,
+        mapping = aes(x = tfr, y = imr, colour = name, alpha = mid_year)) +
+   geom_point() +
+   geom_path() +
+   # to fit legend on my slides
+   theme(legend.box = "horizontal")
```



Optional Aesthetics

```
> # eastern asia: set colour and size from data, set alpha to 0.5 for all.
> ggplot(data = df3,
+        mapping = aes(x = tfr, y = imr, colour = name, size = pop)) +
+   geom_point(alpha = 0.5) +
+   # to fit legend on my slides
+   theme(legend.box = "horizontal")
```

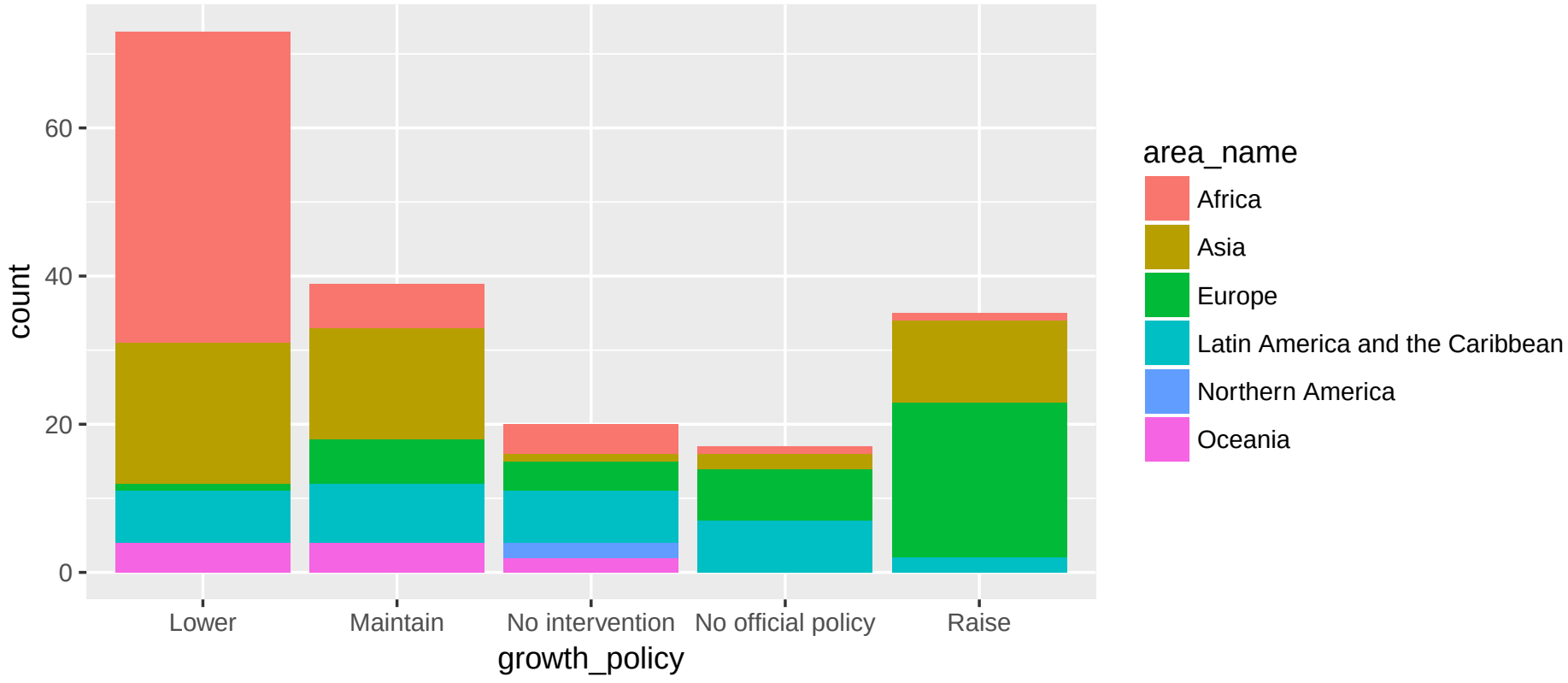


Position

- Each geoms has three main arguments (`mapping`, `position`, `stat`)
- The `position` adjustments determine how to arrange `geoms` that would otherwise occupy the same space.
- For most `geoms` you are unlikely to want to change from the default.
- However, for `geom_bar()` it is useful to know
  - `stack`: stacks elements on top of one another
  - `dodge`: arrange elements side by side
  - `fill`: stack elements on top of one another, normalize height
- For other `geoms` there are other position adjustment possibilities
  - `jitter`: adds random noise to `x` and `y` position of each element to avoid over plotting
  - `nudge`: nudge labels (in `geom_label()`) away from points

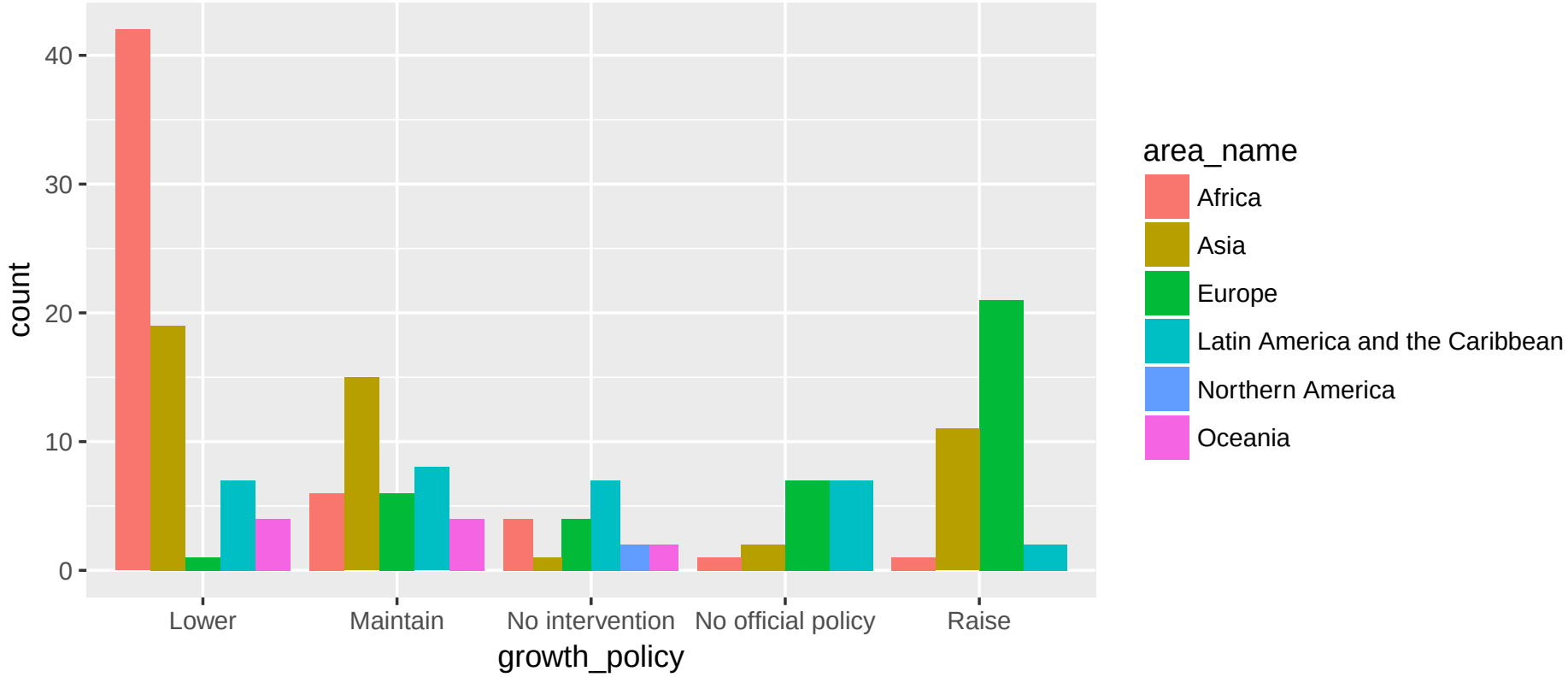
Position

```
> # all countries 2010-15: stack position (default for geom_bar())
> ggplot(data = df1, mapping = aes(x = growth_policy, fill = area_name)) +
+   geom_bar()
```



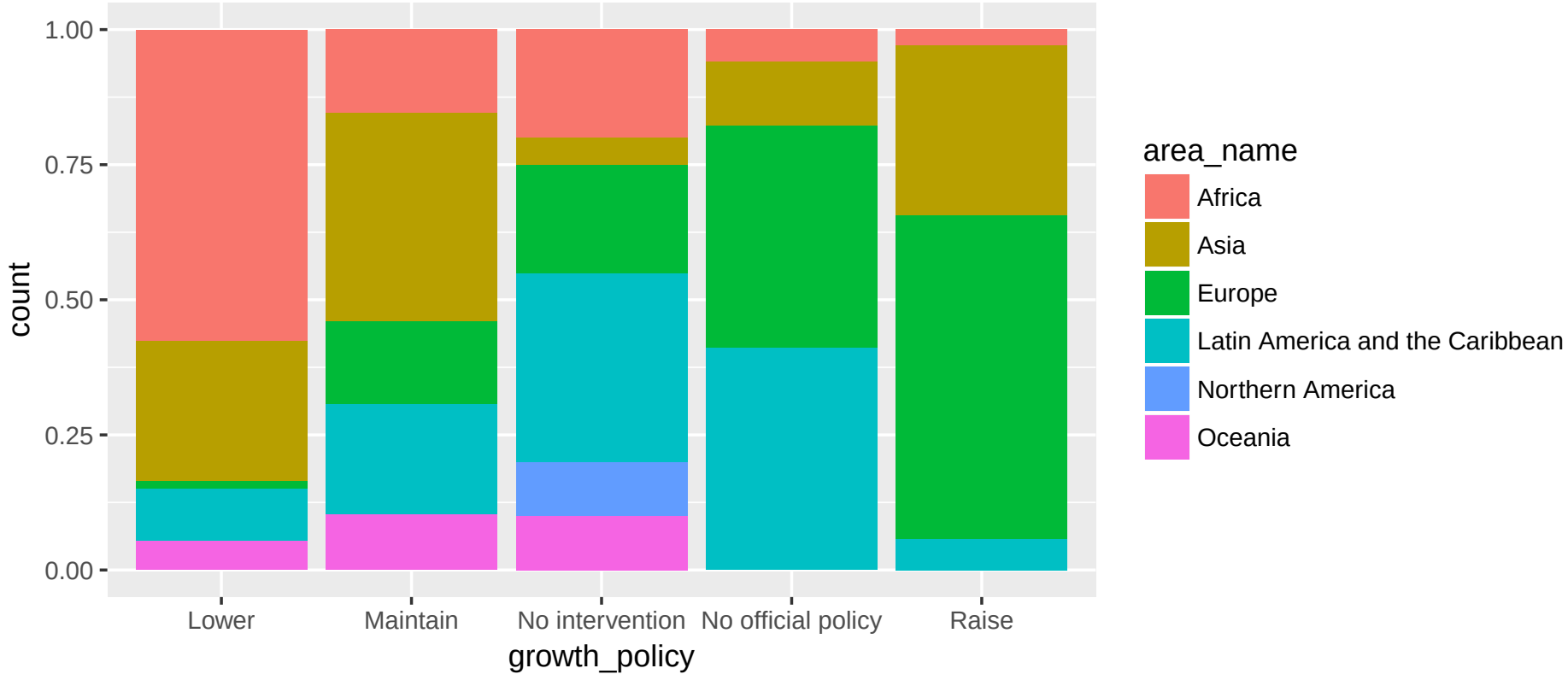
Position

```
> # all countries 2010-15: dodge position
> ggplot(data = df1, mapping = aes(x = growth_policy, fill = area_name)) +
+   geom_bar(position = "dodge")
```



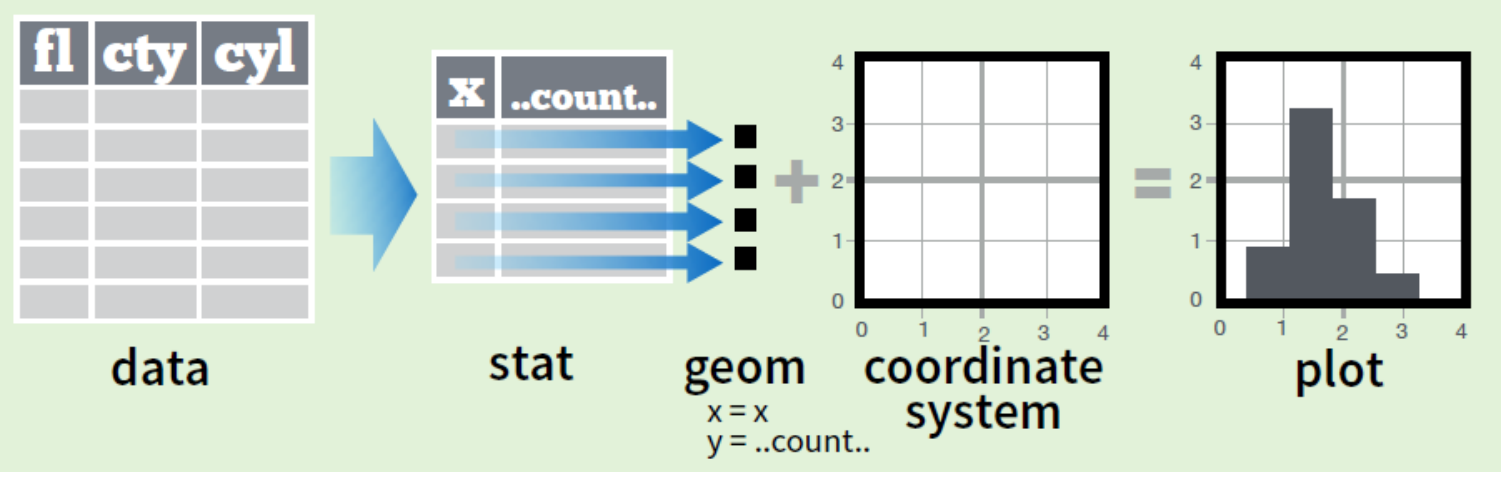
Position

```
> # all countries 2010-15: fill position
> ggplot(data = df1, mapping = aes(x = growth_policy, fill = area_name)) +
+   geom_bar(position = "fill")
```



Stats

- Each geoms has three main arguments (`mapping`, `position`, `stat`)
- The `stat` argument builds new variables to plot (e.g., count, prop).
  - Short for statistical transformation.
- Each geom is associated with a default stat that it uses to calculate values to plot.
  - Applied the transformation and stores the results behind the scenes.
  - Uses and intuitive set of defaults.
  - Rarely need to adjust a geom `stat`.

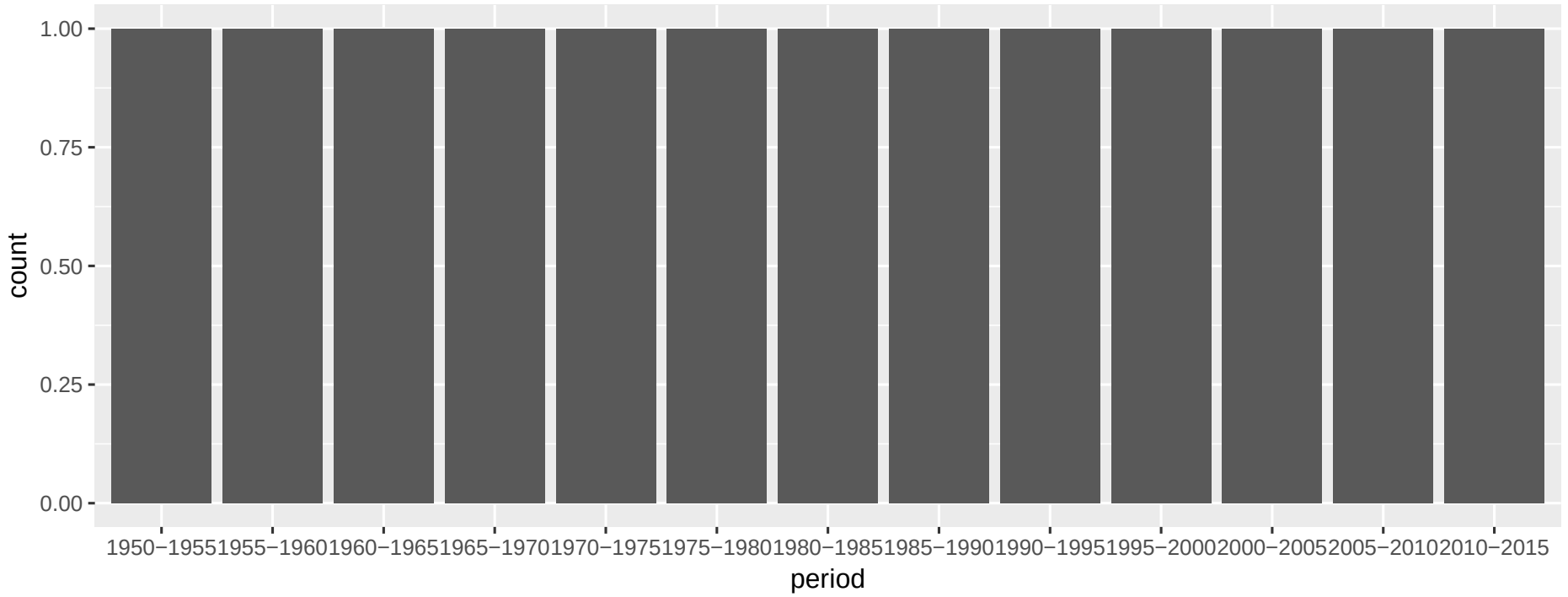


Stats

- Some stats we have already seen in action:
  - `identity`: use raw values, e.g. `geom_point()`
  - `count`: computes two new variables, count and proportion, e.g. `geom_bar()`
  - `bin`: arrange data into bins and counts, e.g. `geom_histogram()`
  - `density`: computes kernel density estimates, e.g. `geom_density()`
  - `smooth`: fit a model to your data and then plot the model line, e.g. `geom_smooth()`
  - `boxplot`: computes box plots statistics (min, max, IQR, median), e.g. `geom_boxplot()`
- For most `geoms` you are unlikely to want to change from the default.
  - For `geom_bar()` it is useful to know

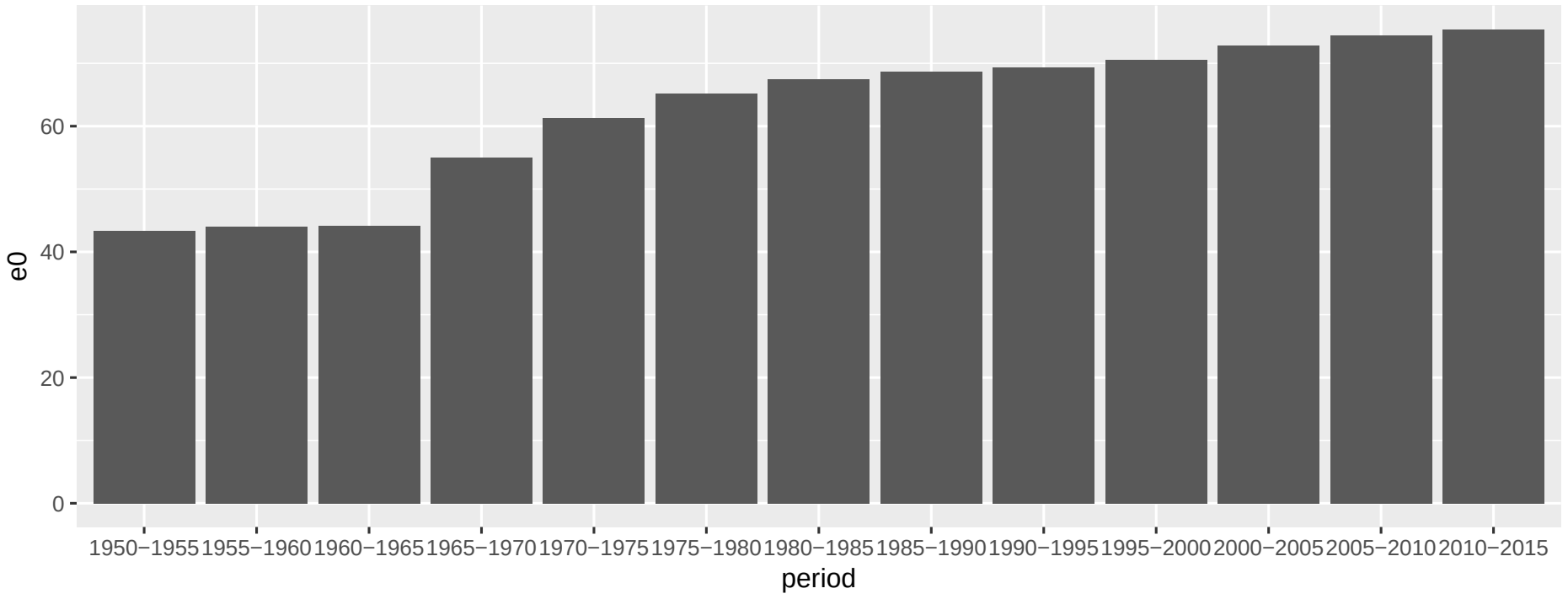
Stats

```
> # china: count stat (default for geom_bar())
> ggplot(data = df2, mapping = aes(x = period)) +
+   geom_bar()
```



Stats

```
> # china: identity stat
> ggplot(data = df2, mapping = aes(x = period, y = e0)) +
+   geom_bar(stat = "identity")
```



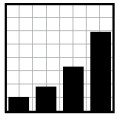
Exercise 2 (ex22.R)

```
# 0.  a) Check your working directory is in the course folder

#     b) Load the data and packages by sourcing the solution file for ex21.R
#####("./exercise-solutions/ex21.R")
##
##
##
# 1. Create create a line chart of life expectancy during the past 55 years in South
#    countries with d3
#    (Hint: use the group aesthetic to get lines for each country)


# 2. Create a boxplots of life expectancy during 2010-2015 for all countries with d1
#    a) boxplots for each region group
#    b) plotted with their area names along the horizontal axis (x)


# 3. Create a scatter plot of infant mortalitiy rate against total fertility rate in
#    from d1 with
#    a) point colours matching area names
#    b) point sizes matching the population size


# 4. Create a time plot of life expectancy from Southeast Asian countries in d3 with
```

Coordinates

- Plots are based on a coordinate system.
  - The `ggplot()`, like most software, uses the Cartesian coordinate system by default
- Can change the coordinates system of easily using a `coord_` function with `ggplot()`

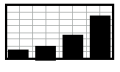
```
r <- d + geom_bar()
```



r + coord_cartesian(xlim = c(0, 5)**)**

xlim, ylim

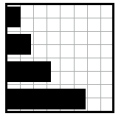
The default cartesian coordinate system



r + coord_fixed(ratio = 1/2**)**

ratio, xlim, ylim

Cartesian coordinates with fixed aspect ratio between x and y units



r + coord_flip()

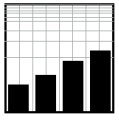
xlim, ylim

Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1 **)**

theta, start, direction

Polar coordinates



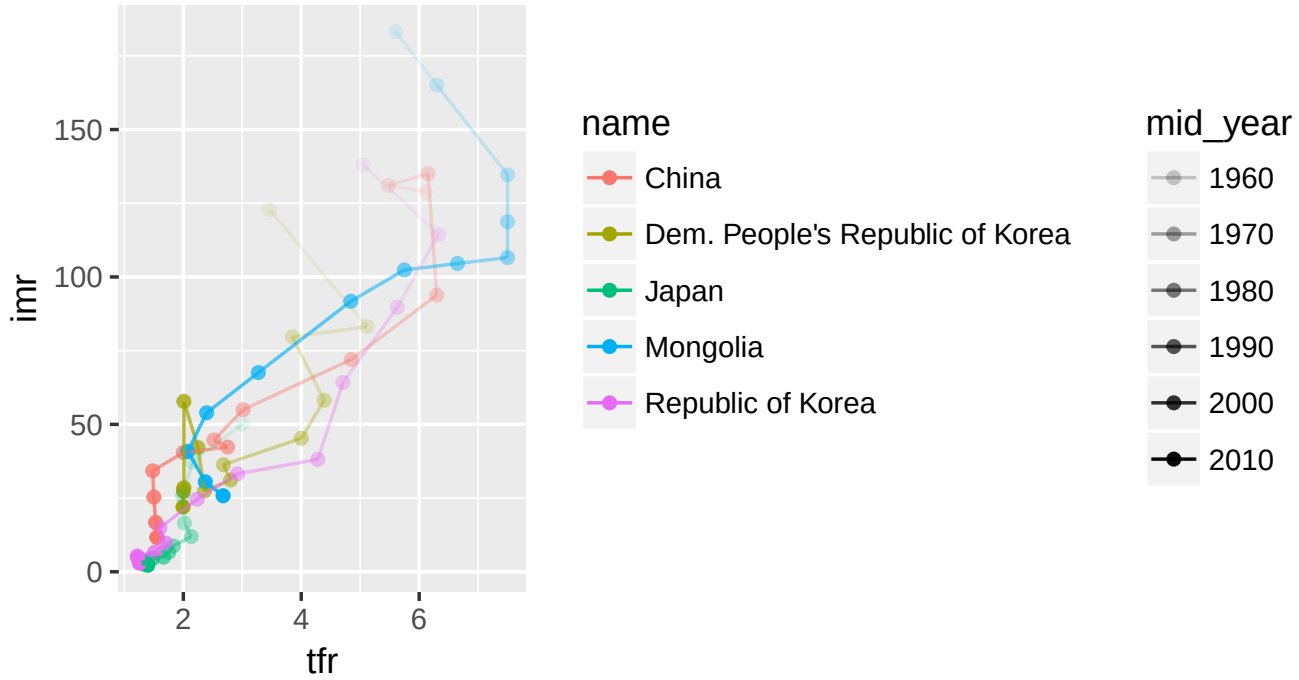
r + coord_trans(ytrans = "sqrt"**)**

xtrans, ytrans, limx, limy

Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

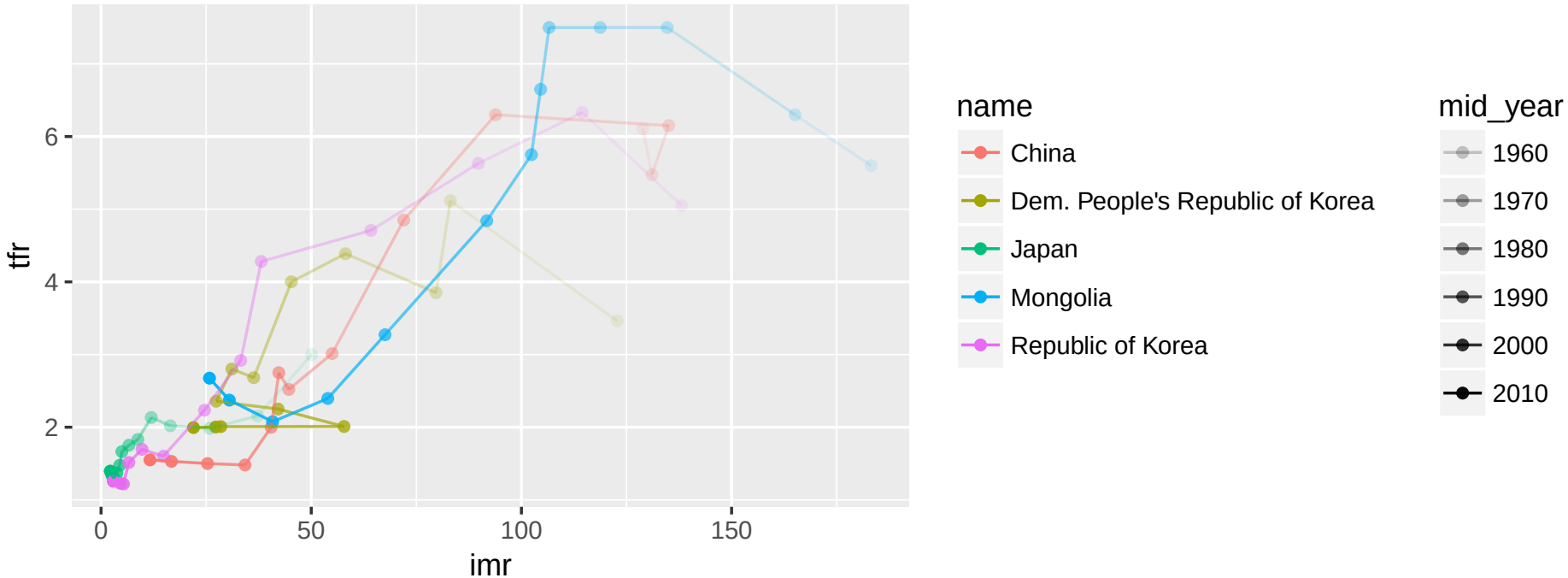
Coordinates

```
> # eastern asia: fix coordinates: 1 coord unit of e0 (y) same as 0.05 tfr (x)
> ggplot(data = df3,
+        mapping = aes(x = tfr, y = imr, colour = name, alpha = mid_year)) +
+   geom_point() +
+   geom_path() +
+   coord_fixed(ratio = 0.05) +
+   # to fit legend on my slides
+   theme(legend.box = "horizontal")
```



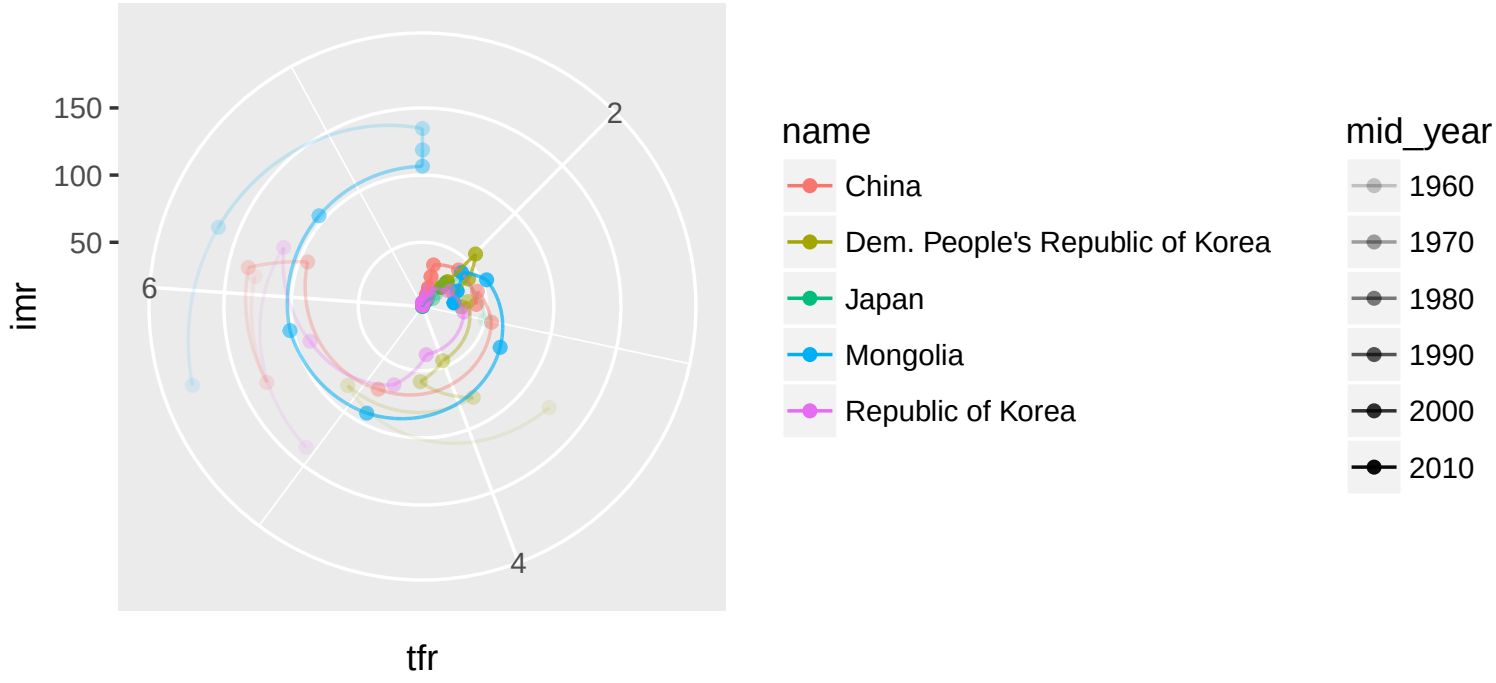
Coordinates

```
> # eastern asia flip coordinates: x is y, y is x
> ggplot(data = df3,
+        mapping = aes(x = tfr, y = imr, colour = name, alpha = mid_year)) +
+   geom_point() +
+   geom_path() +
+   coord_flip() +
+   # to fit legend on my slides
+   theme(legend.box = "horizontal")
```



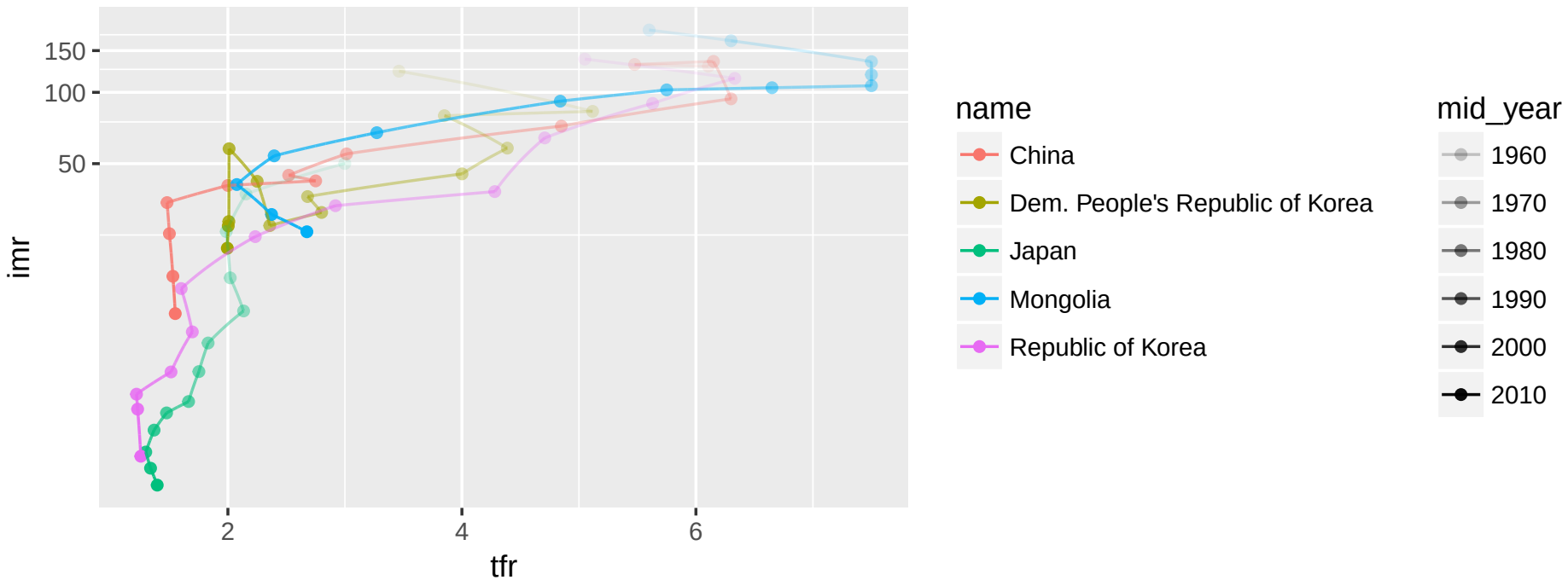
Coordinates

```
> # polar coordinates: x is now position in circle and y is distance from centre
> ggplot(data = df3,
+        mapping = aes(x = tfr, y = imr, colour = name, alpha = mid_year)) +
+   geom_point() +
+   geom_path() +
+   coord_polar() +
+   # to fit legend on my slides
+   theme(legend.box = "horizontal")
```



Coordinates

```
> # transform coordinates: put y on a log scale
> ggplot(data = df3,
+        mapping = aes(x = tfr, y = imr, colour = name, alpha = mid_year)) +
+   geom_point() +
+   geom_path() +
+   coord_trans(y = "log") +
+   # to fit legend on my slides
+   theme(legend.box = "horizontal")
```



Facets

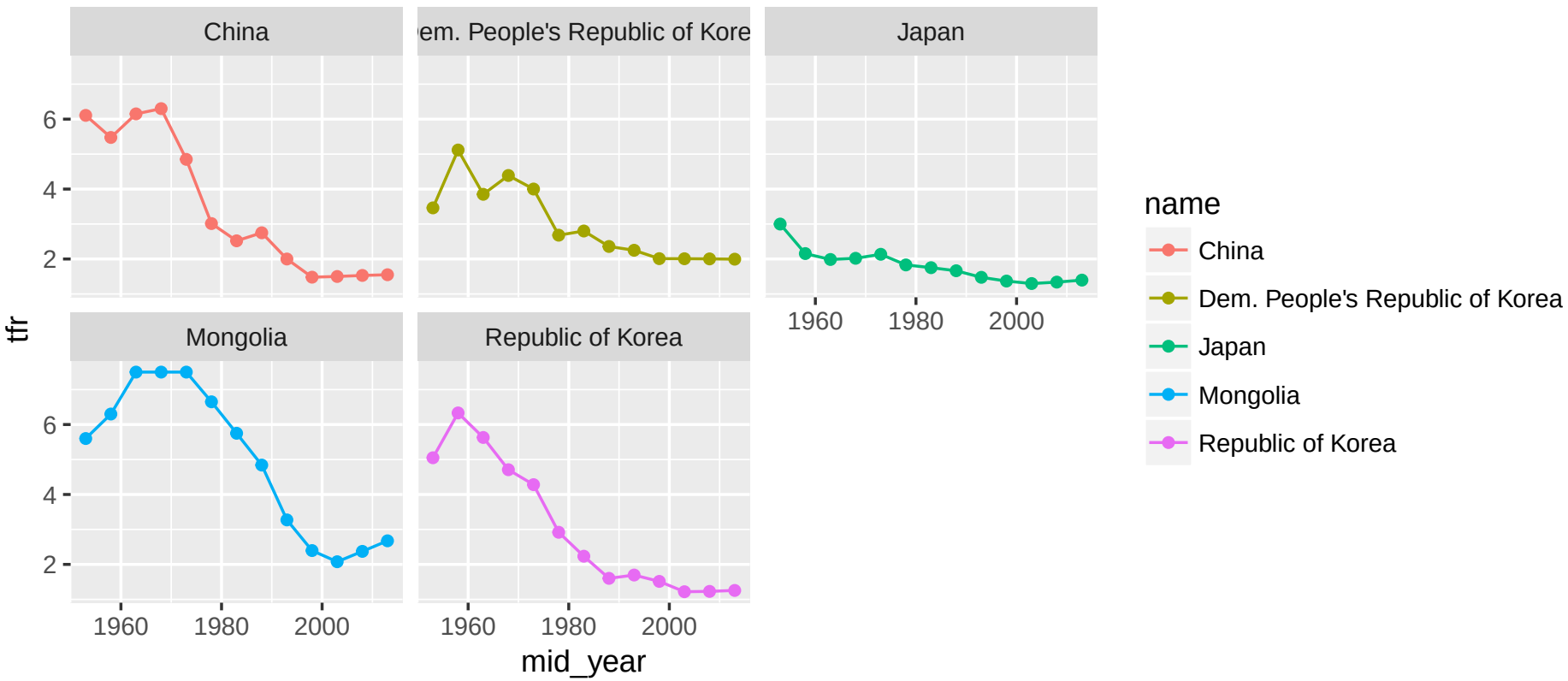
- Facets divide a plot into subplots based on the values of one or more discrete variables
  - Subplots can be a very effective way to compare across discrete (categorical) values
- Two functions:
  - `facet_warp()` wraps a sequence of panels defined by a discrete variable(s) onto a page in roughly rectangular form.
  - `facet_grid()` forms a matrix of panels defined by row and column faceting variables. It is most useful when you have two discrete variables, and all combinations of the variables exist in the data.

Facets

- Both functions have similar arguments
  - `facet`: either a formula or character vector, such as use either
    - character vector(s) `"a"` or `c("a", "b")` (`facet_wrap()` only)
    - one sided formula, `~ a` or `~ a + b`
    - two sided formula `a ~ b` `a ~ .` or `. ~ b` (`facet_grid()` only)
  - `nrow, ncol`: Number of rows and columns.
  - `scales`: fixes or frees scales in facets
    - `"fixed"`: both scales fixed (default)
    - `"free"`: both scales free
    - `"free_x"` or `"free_y"`: scales free in one dimension.
  - `labeller`: adjust facet strip labels,
    - for example `labeller = label_wrap_gen(10)` to make sure each facet strip label is only a maximum 10 characters wide.

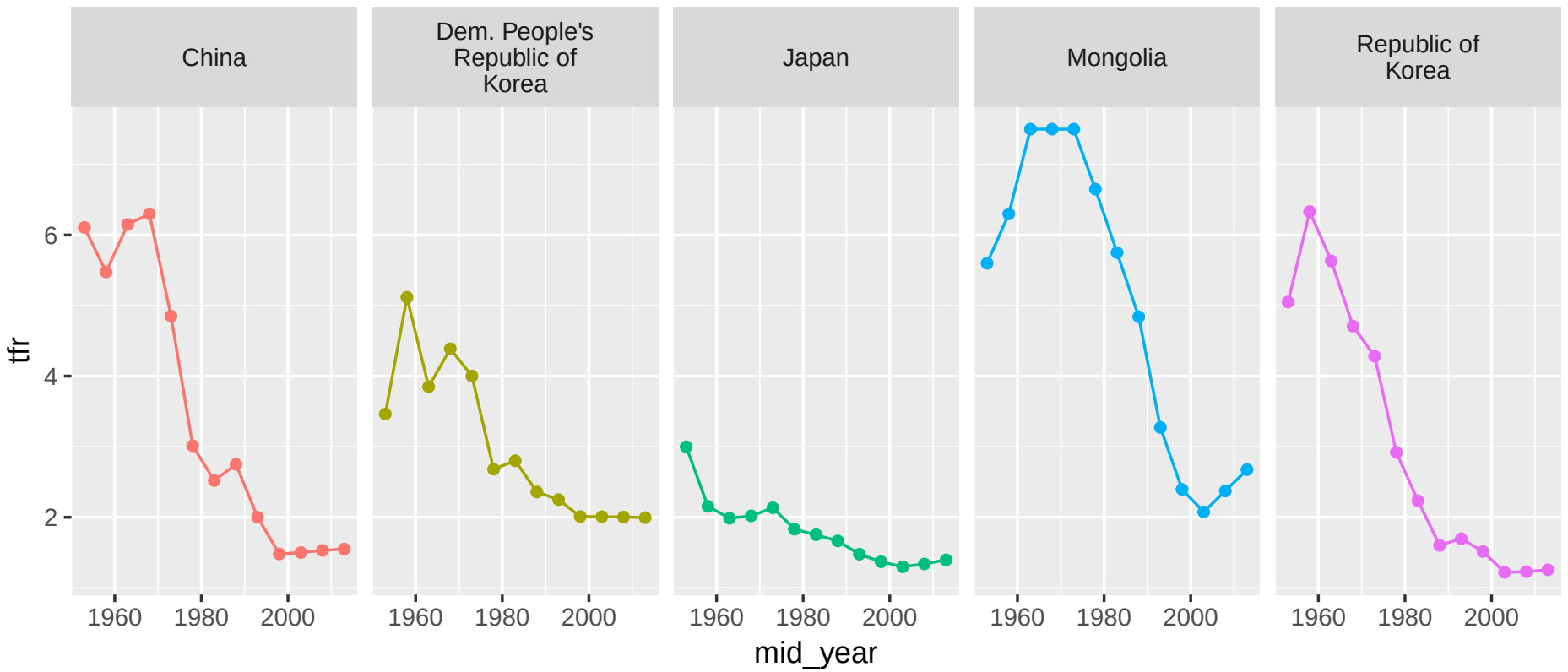
Facets

```
> # eastern asia: facet_wrap() function for each country (name)
> ggplot(data = df3, mapping = aes(x = mid_year, y = tfr, colour = name)) +
+   geom_point() +
+   geom_line() +
+   facet_wrap(facets = "name") +
+   # to fit legend on my slides
+   theme(legend.box = "horizontal")
```



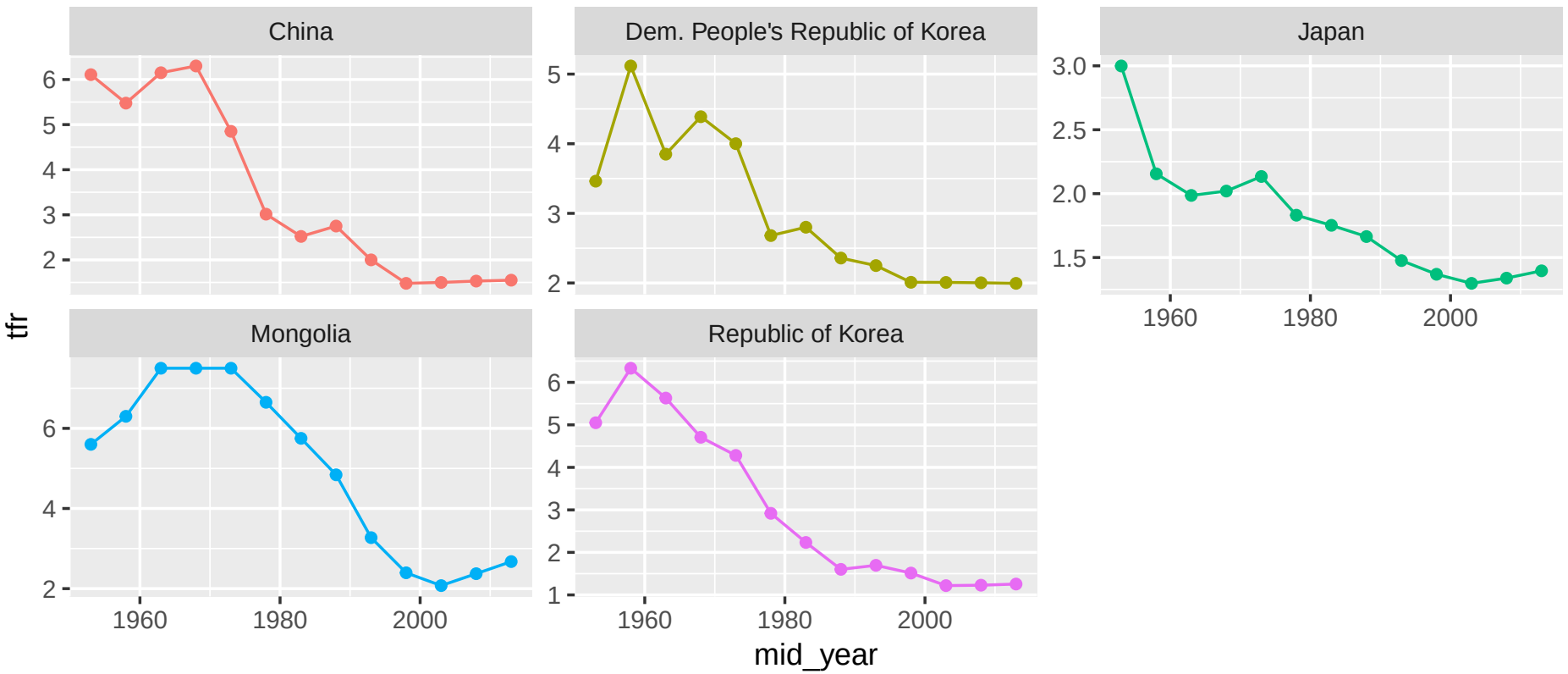
Facets

```
> # facet_wrap() function using one row and restrict label width
> ggplot(data = df3, mapping = aes(x = mid_year, y = tfr, colour = name)) +
+   geom_point() +
+   geom_line() +
+   facet_wrap(facets = "name", nrow = 1, labeller = label_wrap_gen(15)) +
+   # drop country name guide
+   guides(colour = FALSE)
```



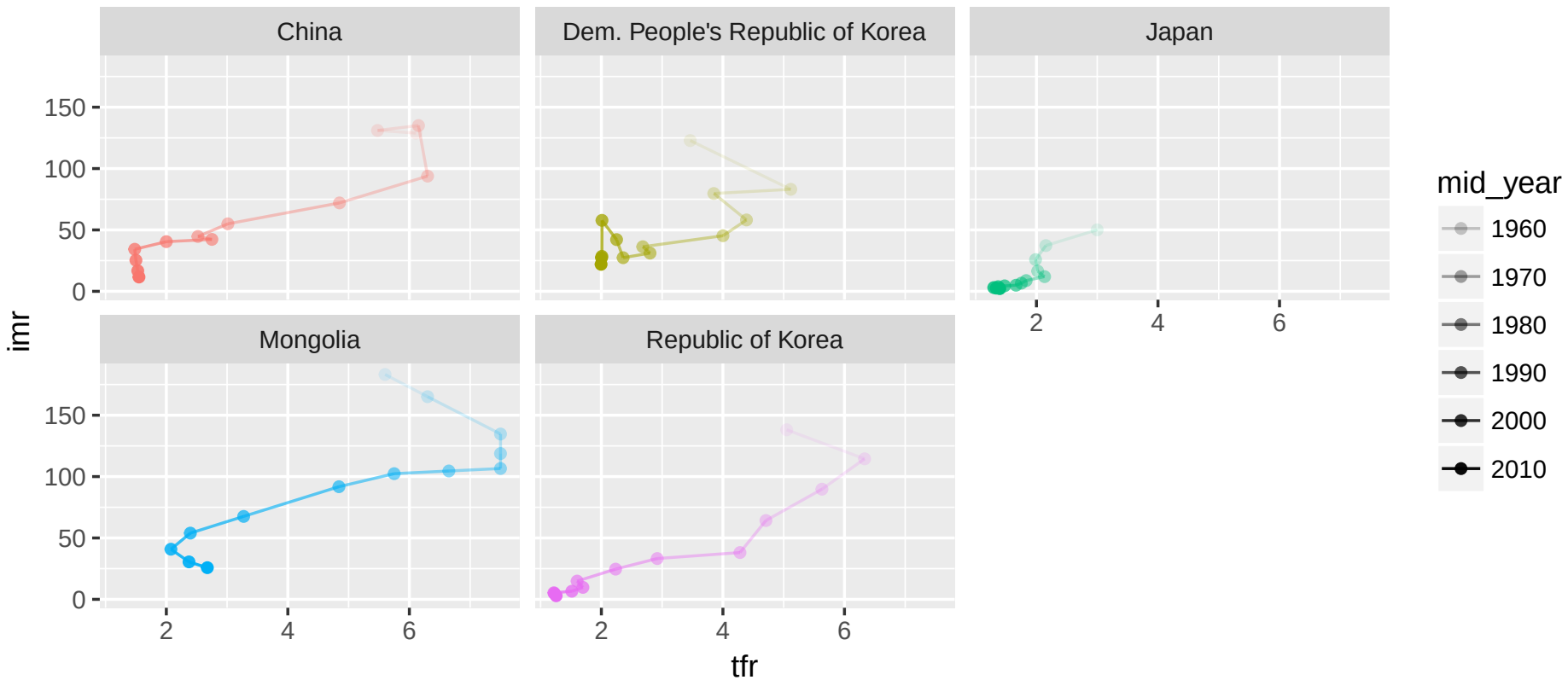
Facets

```
> # facet_wrap() function with different y axis limits
> ggplot(data = df3, mapping = aes(x = mid_year, y = tfr, colour = name)) +
+   geom_point() +
+   geom_line() +
+   facet_wrap(facets = "name", scales = "free_y") +
+   guides(colour = FALSE)
```



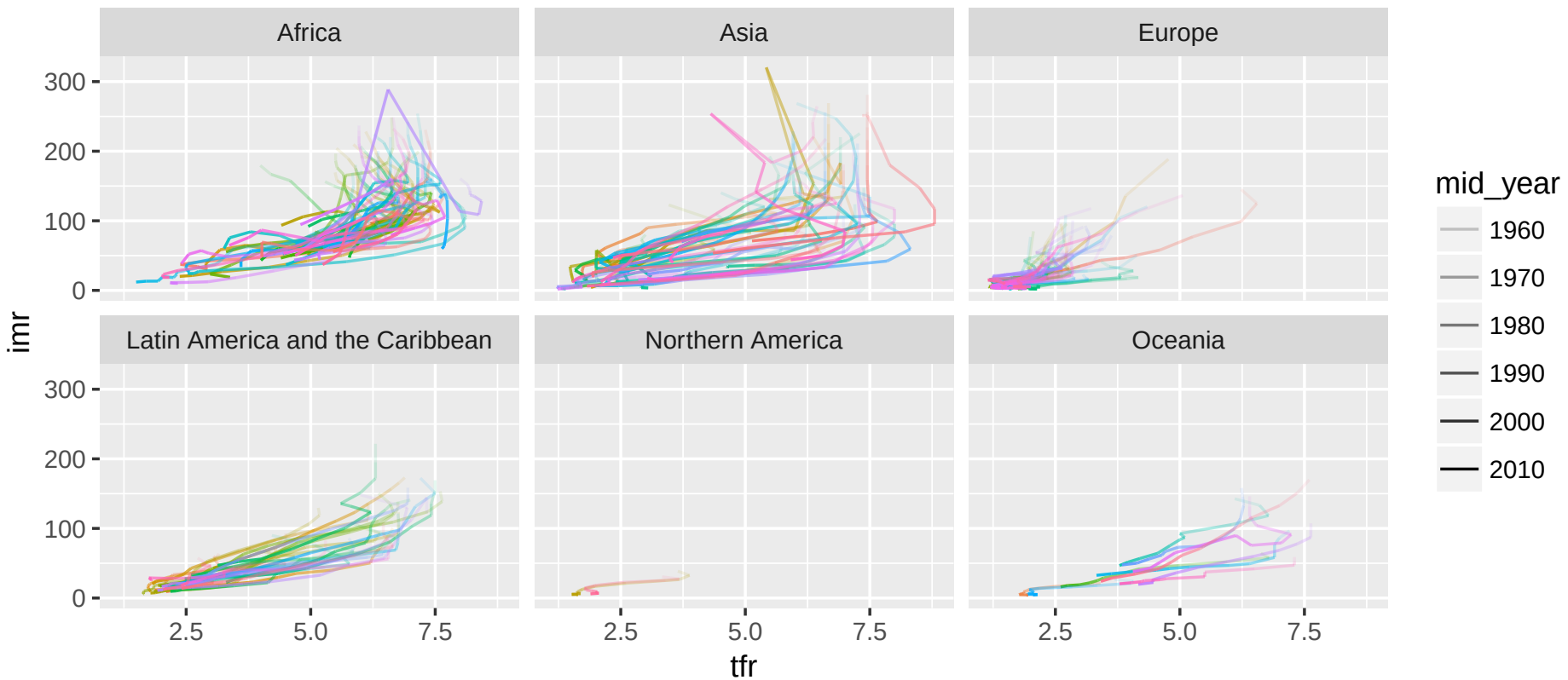
Facets

```
> # facet_wrap() function scatter plot
> ggplot(data = df3,
+        mapping = aes(x = tfr, y = imr, colour = name, alpha = mid_year)) +
+   geom_point() +
+   geom_path() +
+   facet_wrap(facets = "name") +
+   guides(colour = FALSE)
```



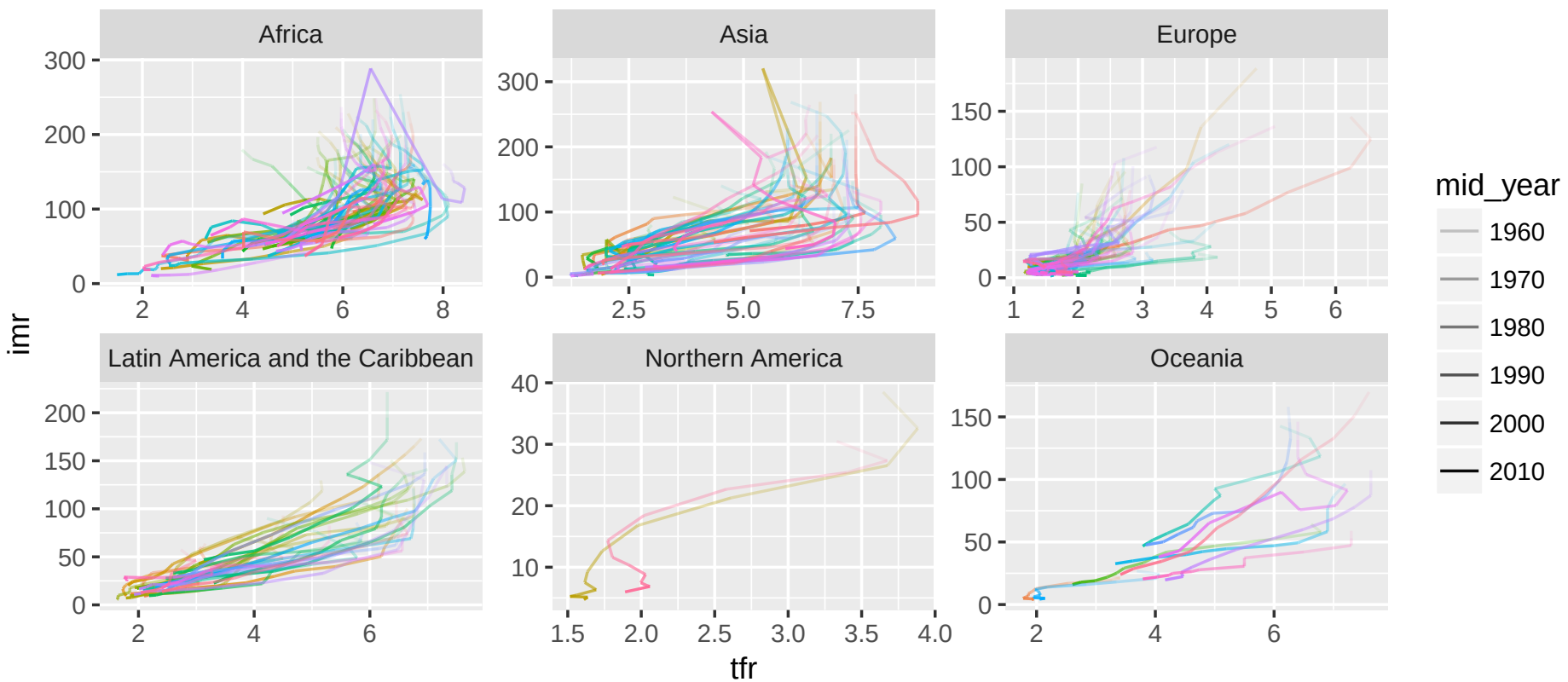
Facets

```
> # all data: facet_wrap() function scatter plot
> ggplot(data = df0,
+        mapping = aes(x = tfr, y = imr, colour = name, alpha = mid_year)) +
+   geom_path() +
+   facet_wrap(facets = "area_name") +
+   guides(colour = FALSE)
```



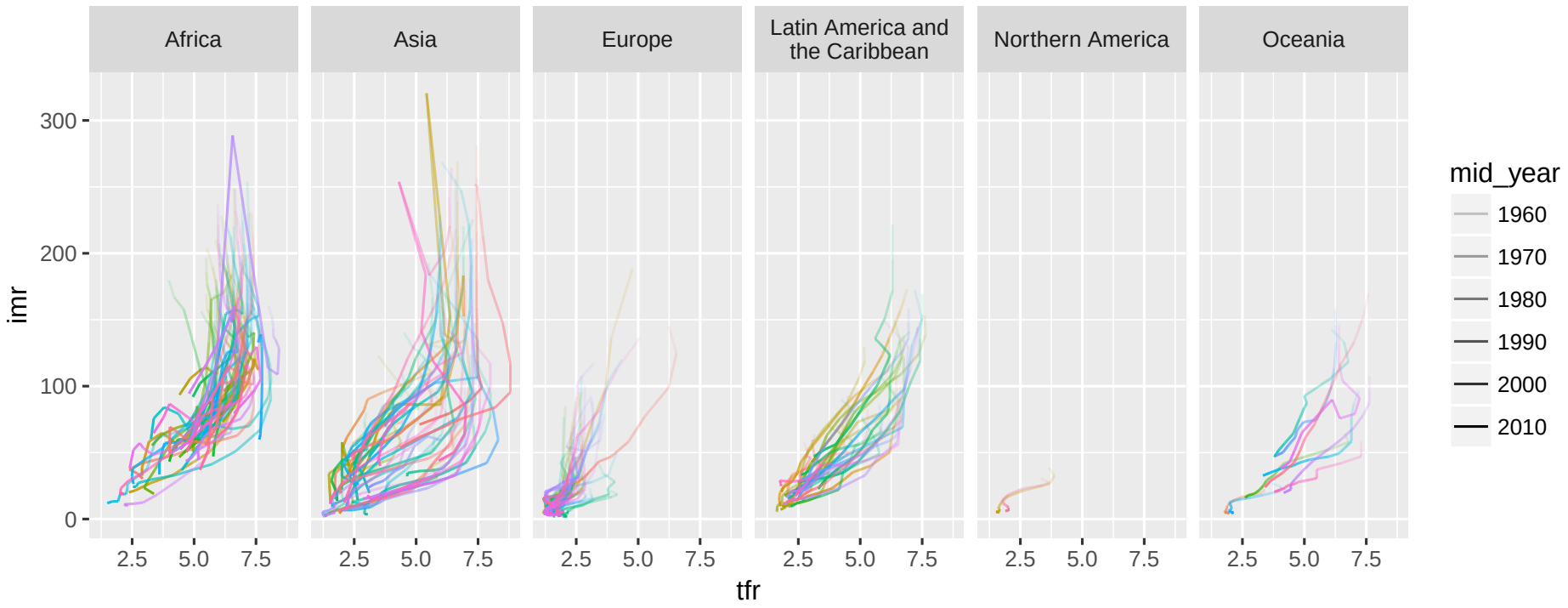
Facets

```
> # all data: facet_wrap() function scatter plot with x and y scales free
> ggplot(data = df0,
+        mapping = aes(x = tfr, y = imr, colour = name, alpha = mid_year)) +
+   geom_path() +
+   facet_wrap(facets = "area_name", scales = "free") +
+   guides(colour = FALSE)
```



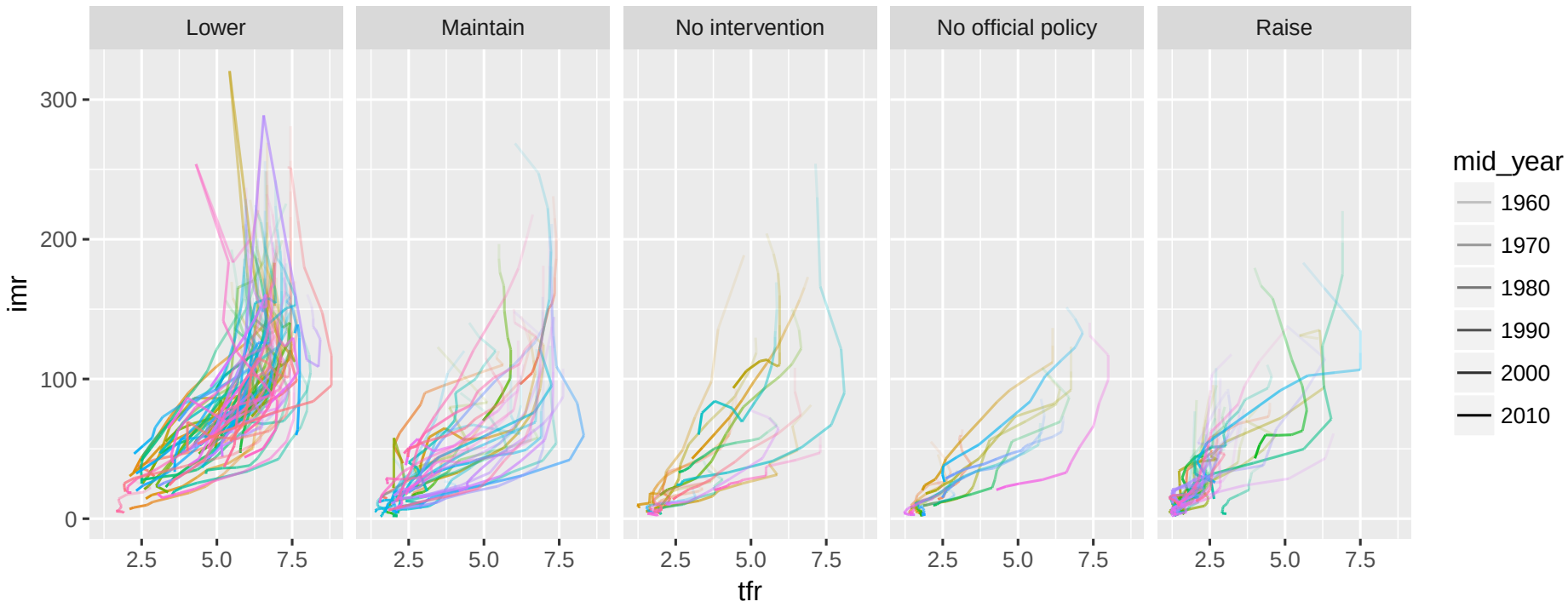
Facets

```
> # facet_grid() function scatter plot using area
> ggplot(data = df0,
+        mapping = aes(x = tfr, y = imr, colour = name, alpha = mid_year)) +
+   geom_path() +
+   facet_grid(facets =  ~ area_name, labeller = label_wrap_gen(18)) +
+   guides(colour = FALSE)
```



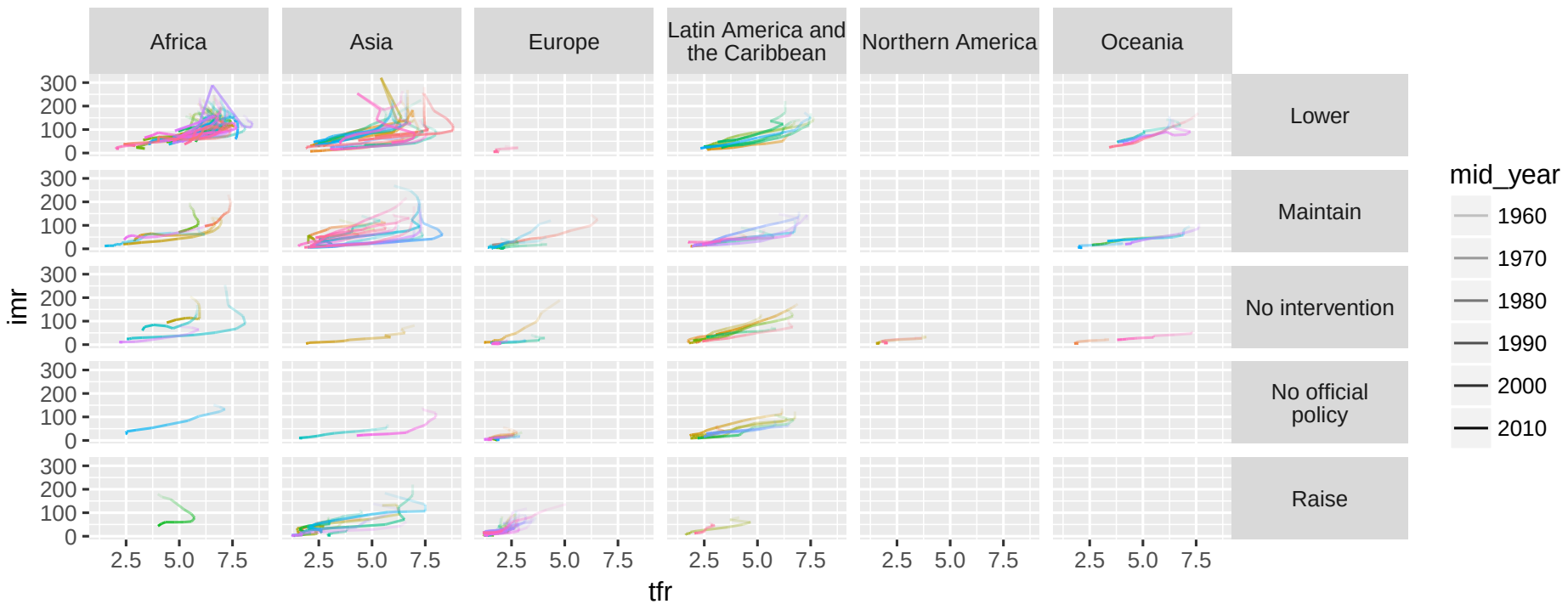
Facets

```
> # facet_grid() function scatter plot using population growth policy
> ggplot(data = df0,
+        mapping = aes(x = tfr, y = imr, colour = name, alpha = mid_year)) +
+   geom_path() +
+   facet_grid(facets = ~ growth_policy) +
+   guides(colour = FALSE)
```



Facets

```
> # facet_grid() function scatter plot using area and population growth policy
> ggplot(data = df0,
+        mapping = aes(x = tfr, y = imr, colour = name, alpha = mid_year)) +
+   geom_path() +
+   facet_grid(facets = growth_policy ~ area_name, labeller = label_wrap_gen(18)) +
+   guides(colour = FALSE) +
+   # rotate growth policy labels to fit legend on my slides
+   theme(strip.text.y = element_text(angle = 0))
```



Exercise 3 (ex23.R)

```
# 0.  a) Check your working directory is in the course folder

#     b) Load the data and packages by sourcing the solution file for ex21.R

##
##
##
# 1. Create a scatter plot of total fertility rate against infant mortalitiy rate in
#    a) point colours matching area names
#    b) point sizes matching the population size in millions
#    c) infant mortality axis transformed to the log10 scale



# 2. Create a bar plot of the immigration policies in different countries in 2015 fr
#    a) fill colour based on continent
#    b) bars are horizontal not vertical



# 3. Create a scatter plot of total fertility rate against life expectancy in all co
#    a) point sizes matching the population size in millions
#    b) scatter plots for each continent in their own facet
```

Scales

- The `scale` functions change aesthetics from their default.
- Each aesthetic has its own functions. Take a general form:
  - `scale_*_continuous()`: alter aesthetics based on continuous variables
  - `scale_*_discrete()`: alter aesthetics based on discrete variables
  - `scale_*_manual(values = c())`: map discrete values to manually chosen visual ones

Color and fill scales (Discrete)

n <- d + geom_bar(aes(fill = fl))

n + **scale_fill_brewer**(palette = "Blues")
For palette choices: RColorBrewer::display.brewer.all()

n + **scale_fill_grey**(start = 0.2, end = 0.8, na.value = "red")

Color and fill scales (Continuous)

o <- c + geom_dotplot(aes(fill = ..x..))

o + **scale_fill_distiller**(palette = "Blues")

o + **scale_fill_gradient**(low="red", high="yellow")

o + **scale_fill_gradient2**(low="red", high="blue", mid = "white", midpoint = 25)

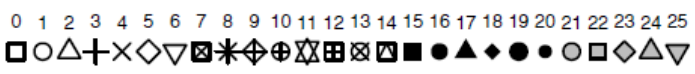
o + **scale_fill_gradientn**(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

Shape and size scales

p <- e + geom_point(aes(shape = fl, size = cyl))

p + **scale_shape**() + **scale_size**()

p + **scale_shape_manual**(values = c(3:7))



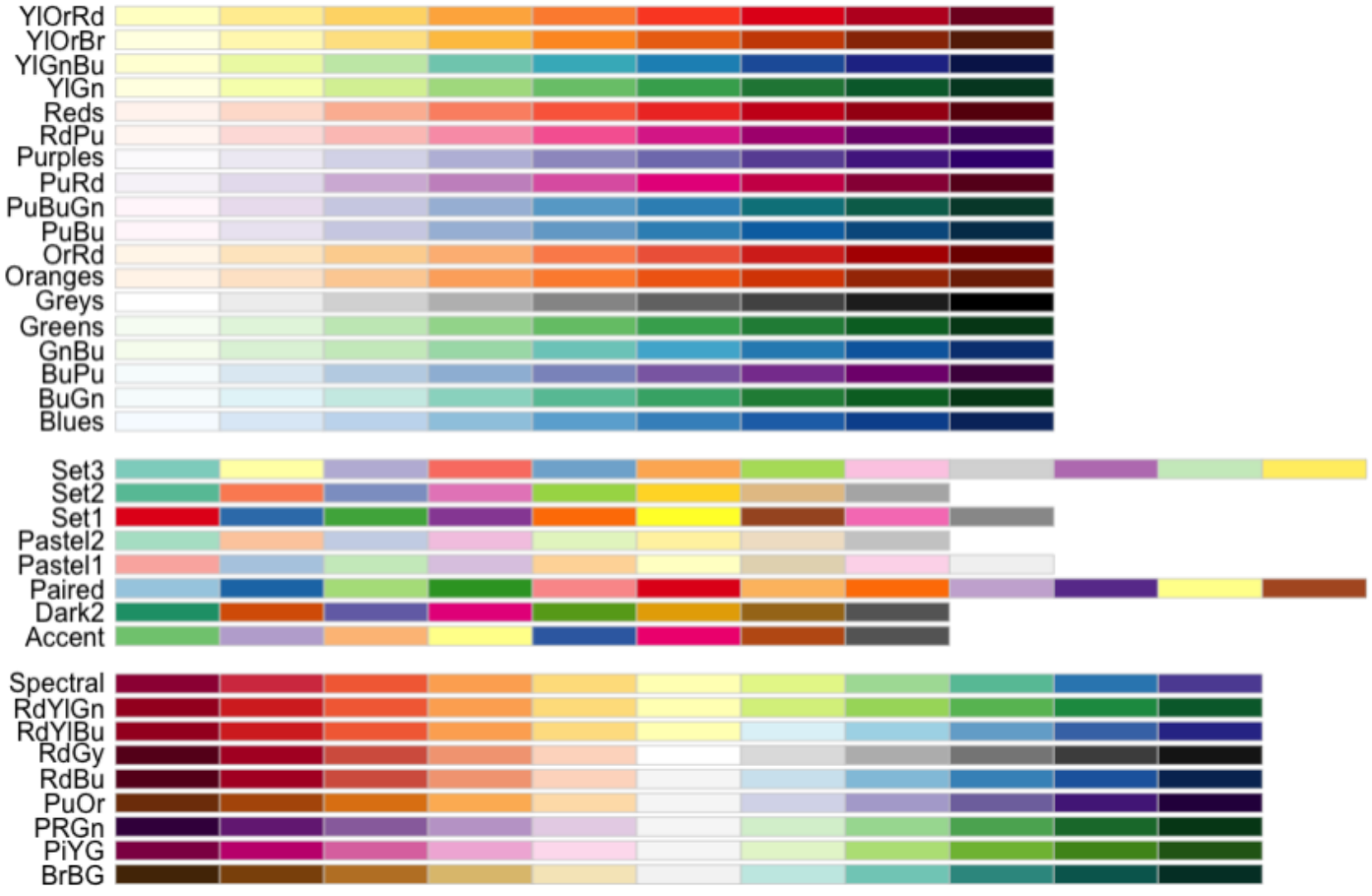
p + **scale_radius**(range = c(1,6))

p + **scale_size_area**(max_size = 6)

Maps to radius of circle, or area

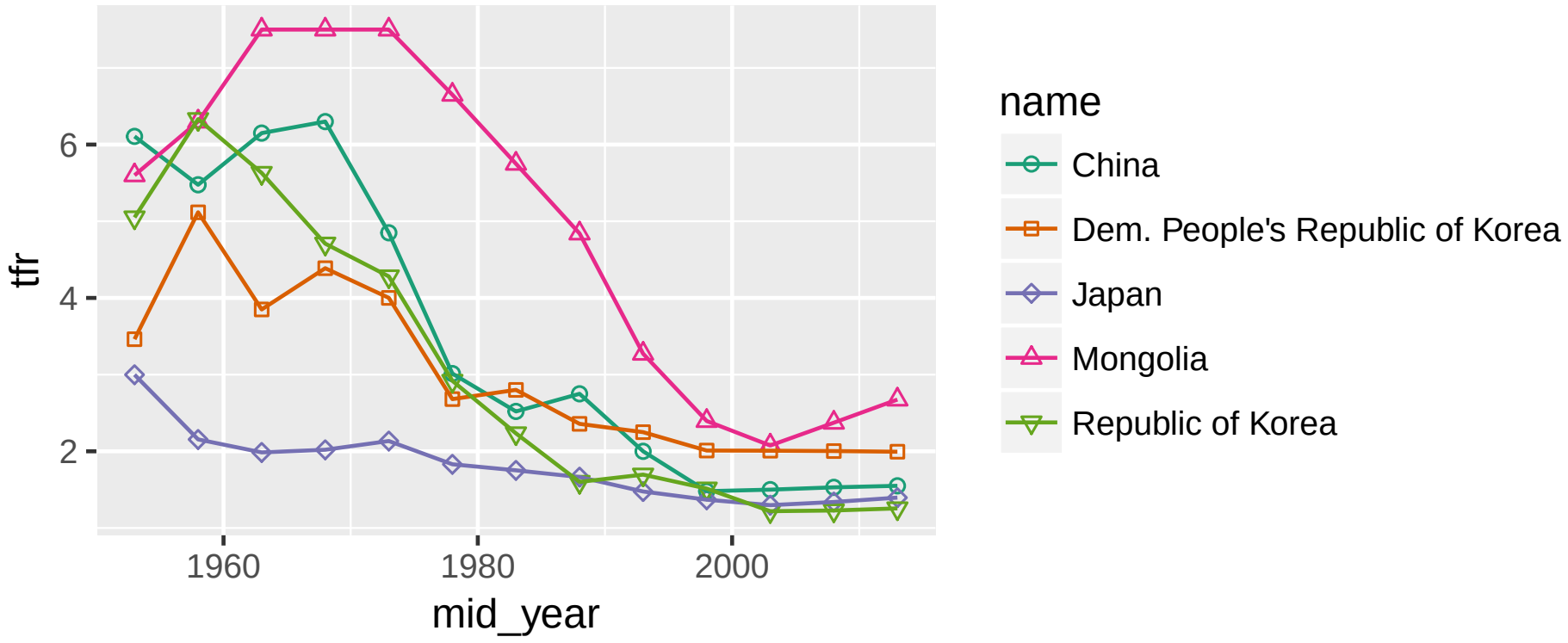
Scales

- Colour schemes from ColorBrewer that are loaded with `ggplot2`
  - http://colorbrewer2.org



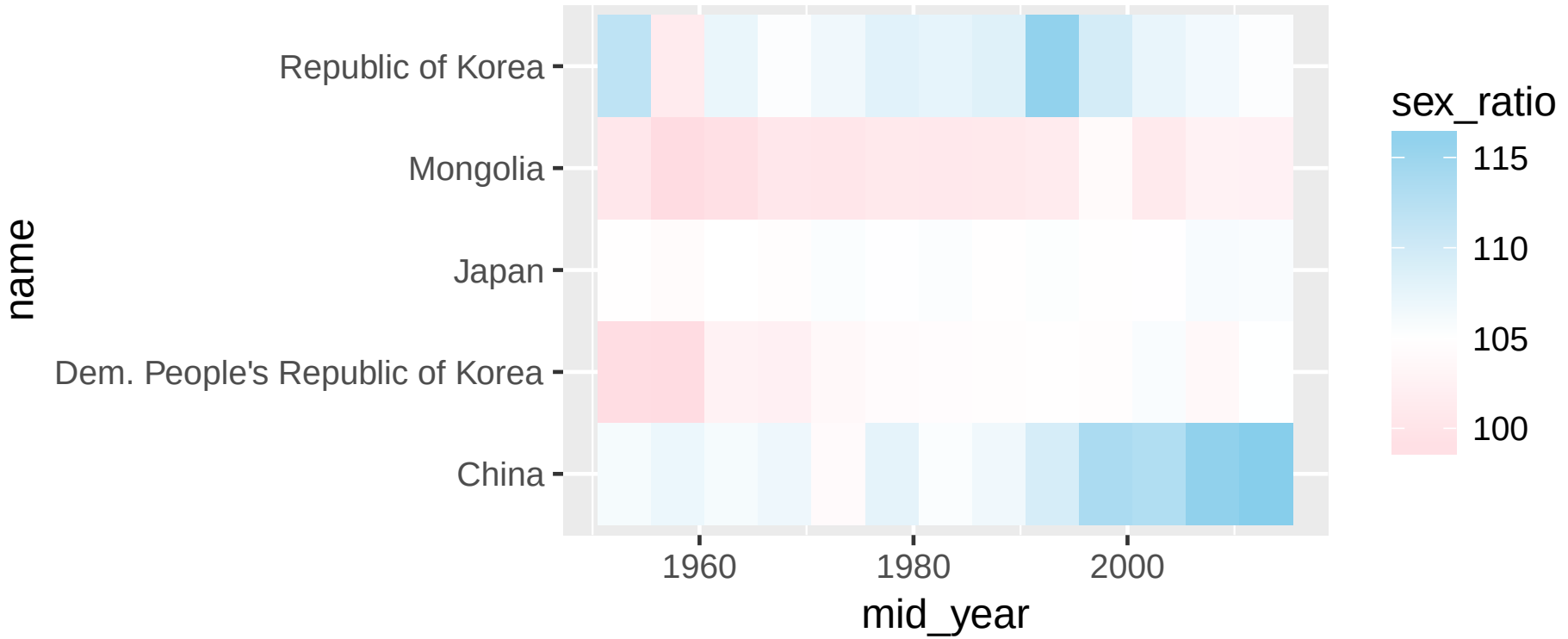
Scales

```
> # eastern asia
> ggplot(data = df3, mapping = aes(x = mid_year, y = tfr, colour = name, shape = na
+   geom_point() +
+   geom_line() +
+   scale_color_brewer(palette = "Dark2") +
+   scale_shape_manual(values = 21:25)
```



Scales

```
> ggplot(data = df3, mapping = aes(x = mid_year, y = name, fill = sex_ratio)) +
+   geom_tile() +
+   scale_fill_gradient2(low="pink", high="skyblue", mid = "white", midpoint = 105)
```

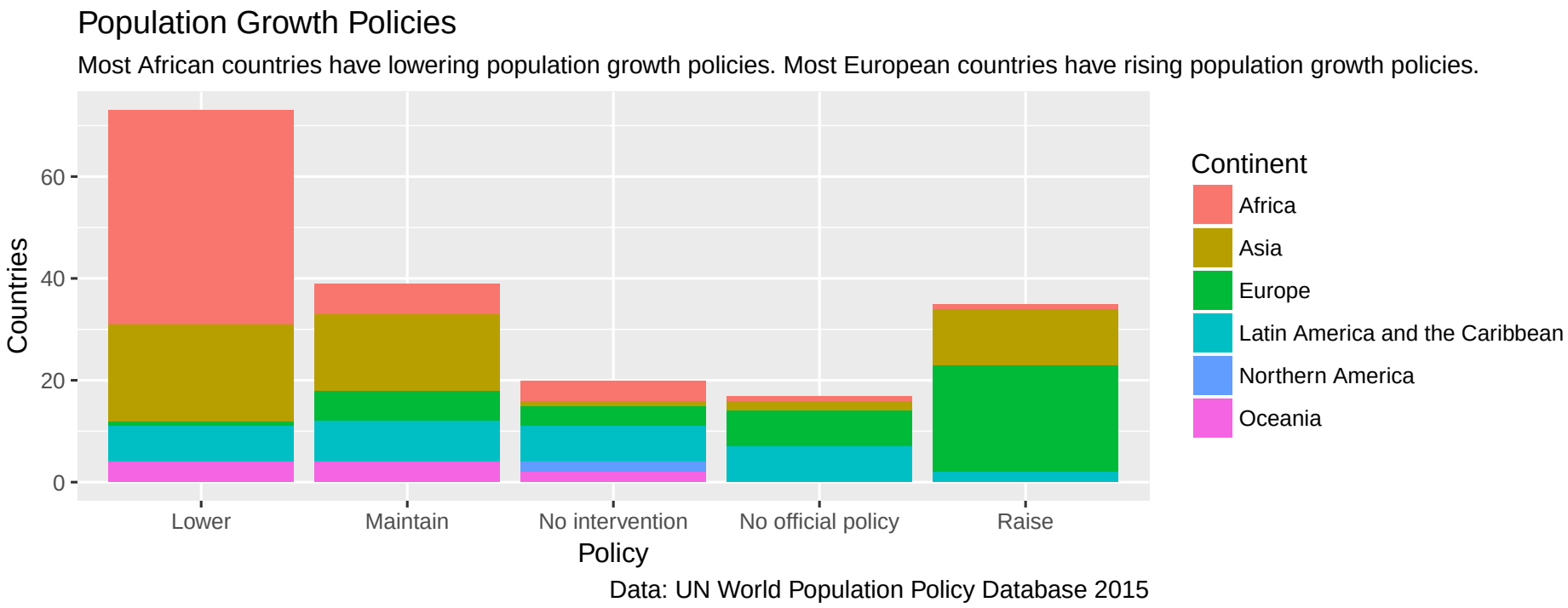


Labels

- Good labels are critical for making your plots accessible to a wider audience.
- The `labs()` function can add
  - `title`: main title
  - `subtitle`: highlight main message
  - `caption`: data source information
  - `x`, `y`, `fill`, `colour`, `alpha`, ... : full variable names for aesthetics

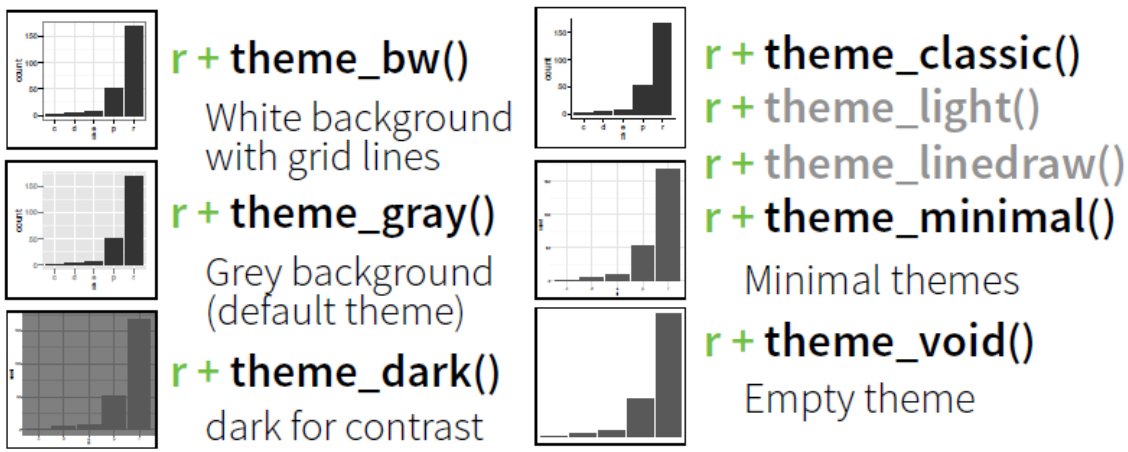
Labels

```
> # all countries 2010-15 data
> ggplot(data = df1, mapping = aes(x = growth_policy, fill = area_name)) +
+   geom_bar() +
+   labs(title = "Population Growth Policies",
+        subtitle = "Most African countries have lowering population growth policie
+        caption = "Data: UN World Population Policy Database 2015",
+        x = "Policy", y = "Countries", fill = "Continent")
```



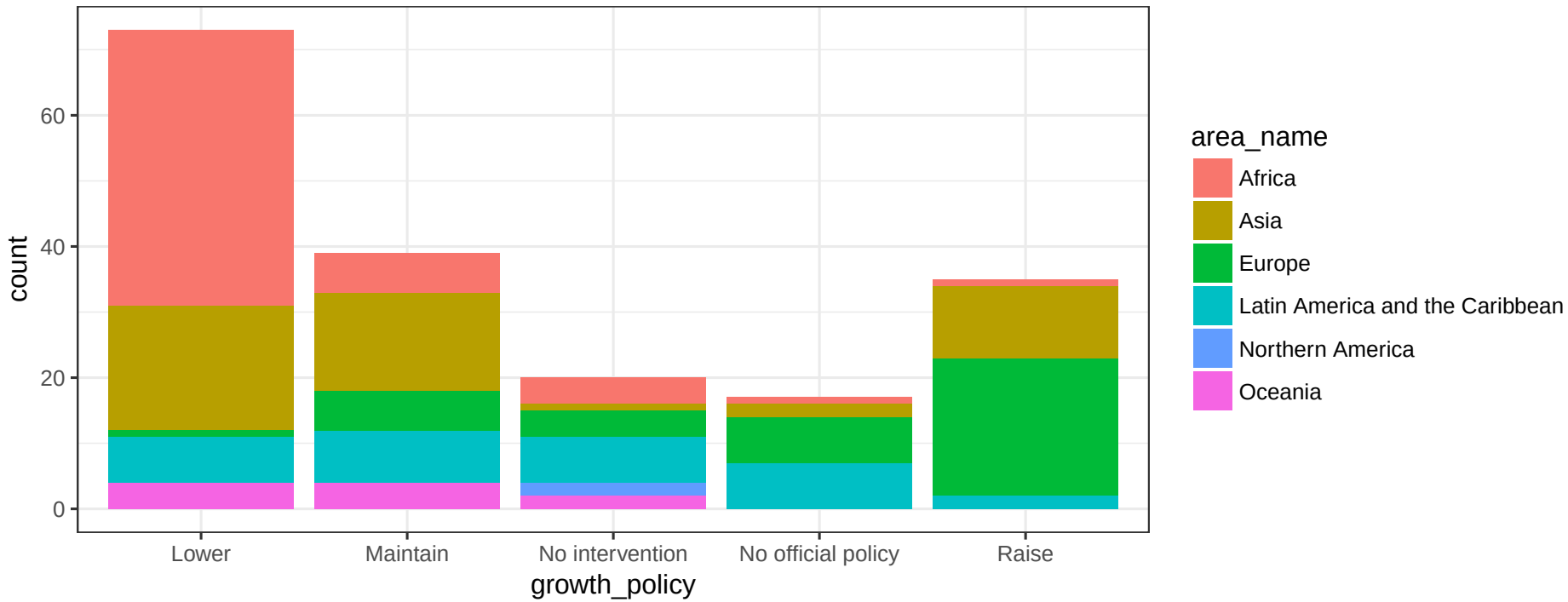
Themes

- The `theme` functions customize the "look" of plots
  - Changes how the plot looks without changing the information that the plot displays.
- There are eight theme functions in `ggplot2`
- The `ggthemes` package has many themes to match publication styles e.g. Economist or FiveThirtyEight



Themes

```
> # change theme
> ggplot(data = df1, mapping = aes(x = growth_policy, fill = area_name)) +
+   geom_bar() +
+   theme_bw()
```



Saving

- Can assign a plot to an R object
  - Can then keep adding to the plot with the +

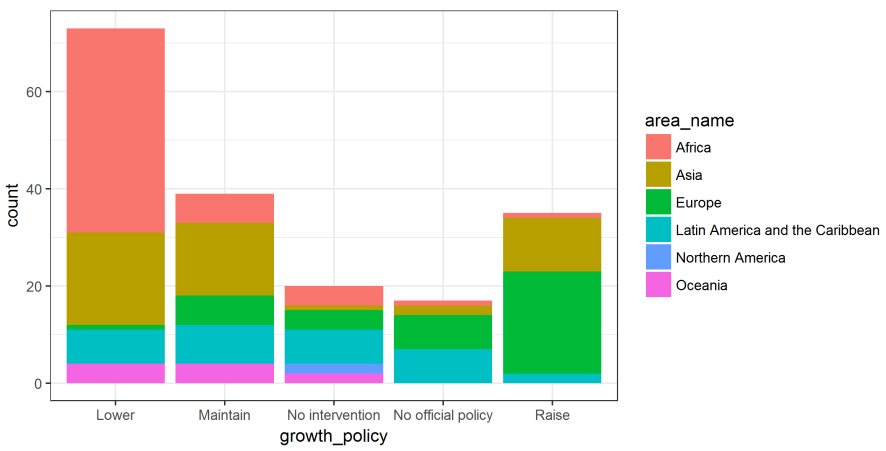
```
> # save a plot to an object
> g <- ggplot(data = df1, mapping = aes(x = growth_policy, fill = area_name)) +
+   geom_bar()
> # plot g with the black and white theme
> g + theme_bw()
```

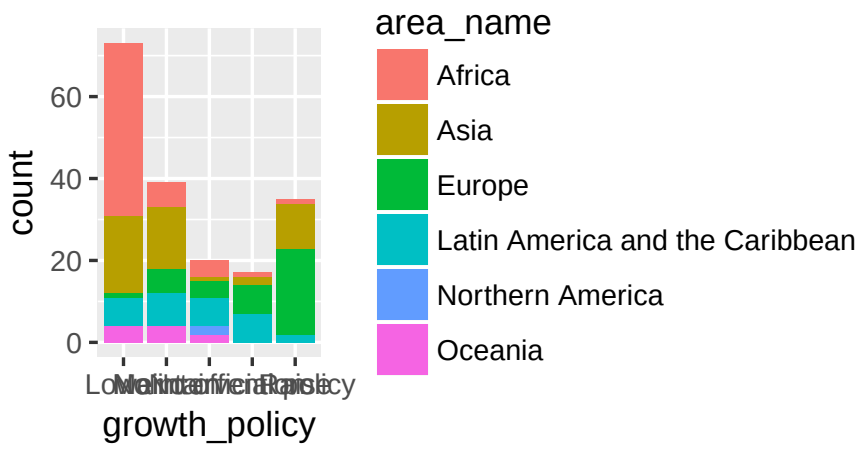
- The ggsave() function to save plot from
  - `filename`: matches file type to file extension, e.g. `filename = myplot.pdf` will create a PDF.
  - `width`, `height`, `units`: plot size, uses RStudio window size by default
  - `scale`: scales the size of the plots objects

```
> # saves the last plot from RStudio as a PNG
> ggsave(filename = "myplot.png", width = 10, height = 5, unit = "cm", scale = 2)
> # saves the plot from object g as a PDF
> ggsave(filename = "myplot.pdf", plot = g, width = 10, height = 5, unit = "cm")
```

Scale

- Scale alters the point size:
  - Left: PNG with scale = 2
  - Right: PDF with scale = 1 (default)





RStudio Cheatsheet

Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

required

```
ggplot (data = <DATA>) +
<GEOM_FUNCTION> (mapping = aes(<MAPPINGS>),
 stat = <STAT> , position = <POSITION> ) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

Not required, sensible defaults supplied

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings | data | geom

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend=lat+1, xend=long+1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)) x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(fl))
```

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"), x, y, alpha, color, linetype, size

e + geom_smooth(method = lm), x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group

f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

```
g <- ggplot(diamonds, aes(cut, color))
```

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
```

j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

k + geom_map(aes(map_id = state), map = map)
+ expand_limits(x = map$long, y = map$lat), map_id, alpha, color, fill, linetype, size

THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

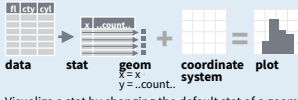
l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width

RStudio Cheatsheet

Stats An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

geom to use | stat function | geommappings

i + stat_density2d(aes(fill = ..level..), geom = "polygon")

variable created by stat

c + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y,** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins=30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5) **x, y** | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area") **x, y** | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se=T, level=0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) **sample, x, y** | ..sample.., ..theoretical..

e + stat_sum() x, y, size | ..n.., ..prop..

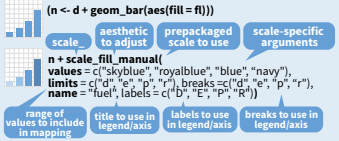
e + stat_summary(fun.data = "mean_cl_boot"**)**

h + stat_summary_bin(fun.y = "mean", geom = "bar")

e + stat_unique()

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.



(n <- d + geom_bar(aes(fill = fl)))

scale_ | aesthetic to adjust | prepackaged scale to use | scale-specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()**)** - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks"**)** - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

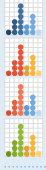
COLOR AND FILL SCALES (DISCRETE)

n <- d + geom_bar(aes(fill = fl)**)**

n + scale_fill_brewer(palette = "Blues"**)**
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red"**)**

COLOR AND FILL SCALES (CONTINUOUS)



o <- c + geom_dotplot(aes(fill = ..x..))

o + scale_fill_distiller(palette = "Blues"**)**

o + scale_fill_gradient(low="red", high="yellow"**)**

o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25**)**

o + scale_fill_gradientn(colours=topo.colors(6)**)**
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7)**)**

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

p + scale_radius(range = c(1,6)**)**
p + scale_size_area(max_size = 6**)**

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5)**)**
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2**)**
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1 **)**
theta, start, direction
Polar coordinates

r + coord_trans(xtrans = "sqrt"**)**
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation=c(41, -74, 0)**)**projection, orienztation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height

e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

e + geom_label(position = "nudge")
Nudge labels away from points

s + geom_bar(position = "stack")
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments
s + geom_bar(position = position_dodge(width = 1))

Themes

r + theme_bw()
White background with grid lines

r + theme_gray()
Grey background (default theme)

r + theme_dark()
dark for contrast

r + theme_classic()
r + theme_light()
r + theme_linedraw()
r + theme_minimal()
Minimal themes
r + theme_void()
Empty theme

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(. ~ fl)
facet into columns based on fl

t + facet_grid(year ~ .)
facet into rows based on year

t + facet_grid(year ~ fl)
facet into both rows and columns

t + facet_wrap(~ fl)
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(drv ~ fl, scales = "free")
x and y axis limits adjust to individual facets
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(. ~ fl, labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(fl ~ ., labeller = label_bquote(alpha ^ .(fl)**))**

α^{c} α^{d} α^{e} α^{p} α^{r}

t + facet_grid(. ~ fl, labeller = label_parsed)

c	d	e	p	r

Labels

t + labs(**x** = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
<AES> = "New <AES> legend title")

Use scale functions to update legend labels

t + annotate(geom = "text", x = 8, y = 9, label = "A"**)**

geom to place | manual values for geom's aesthetics

Legends

n + theme(legend.position = "bottom"**)**
Place legend at "bottom", "top", "left", or "right"

n + guides(fill = "none"**)**
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E")**)**
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred)
t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20)**)**

With clipping (removes unseen data points)
t + xlim(0, 100**) + ylim(**10, 20**)**
t + scale_x_continuous(limits = c(0, 100)**) + scale_y_continuous(**limits = c(0, 100)**)**

RStudio

Exercise 4 (ex24.R)

```
# 0.  a) Check your working directory is in the course folder

#     b) Load the data and packages by sourcing the solution file for ex21.R

##
##
##
# 1. Uncomment the code below (from ex23.R) and adapt to
#    a) change the colour of the points to come from the "Set1" palette
#    b) give sensible names to the x and y axis, colour and size guides
# ggplot(data = d1, mapping = aes(x = tfr, y = imr, colour = area_name, size = pop/1
#   geom_point() +
#   coord_trans(y = "log10")


# 2. save the last plot as a PNG file with name "myplot1" height 6cm and width 10cm

# 3. Create a tile chart of net migration (in millions) over time periods for countr
#    Save the plot as an object g
#    (Hint: net migration is in thousands, to get to millions divide by 1000)


# 4. Modify g to scale the fill gradient from red to black via white (white is at ne
```

Assignment 2 (assign2.R)

```
##
## Assignment 2
##

# Take a look at the assign2.pdf files in the assignment folder
# You are going to write code to produce these plots from the ukcensus2011.csv data
#
# 0) load the tidyverse package, set your working directory to the folder of ukcensu



##
##
##
# 1) uncomment the code to read in the ukcensus2011.csv data into R and save as d0
# d0 <- read_csv("ukcensus2011.csv")
# d0
# 2) create a histogram of
#    a) the percentage of car ownership (car variable) in each district
#    b) with fill colours based on the region of the district


# 3) create a scatter plot of
#    a) the mean age in each district against the percentage of the population that
#    b) the colour each point based on the region of the district
```