

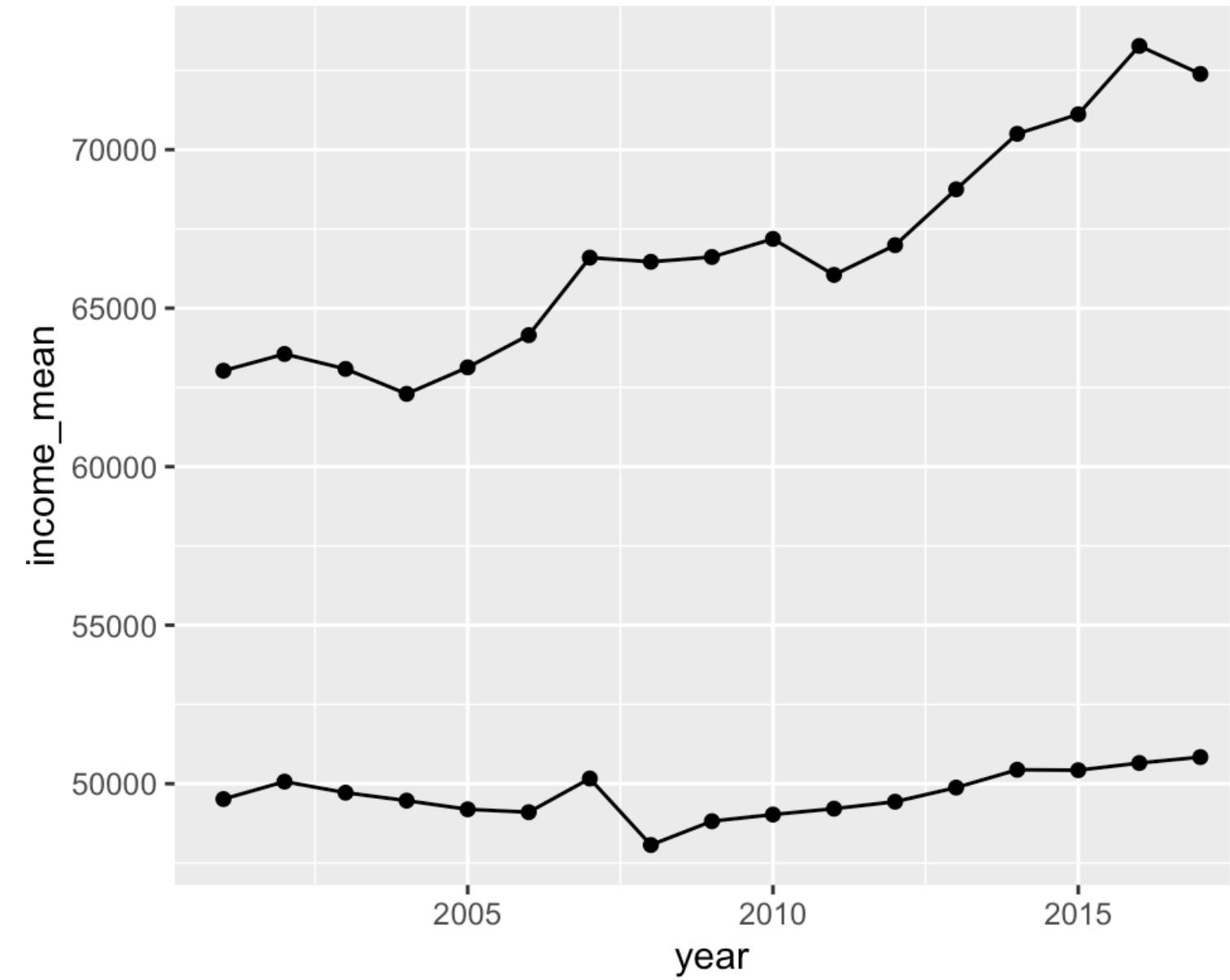


# Data Visualization Styling Plots

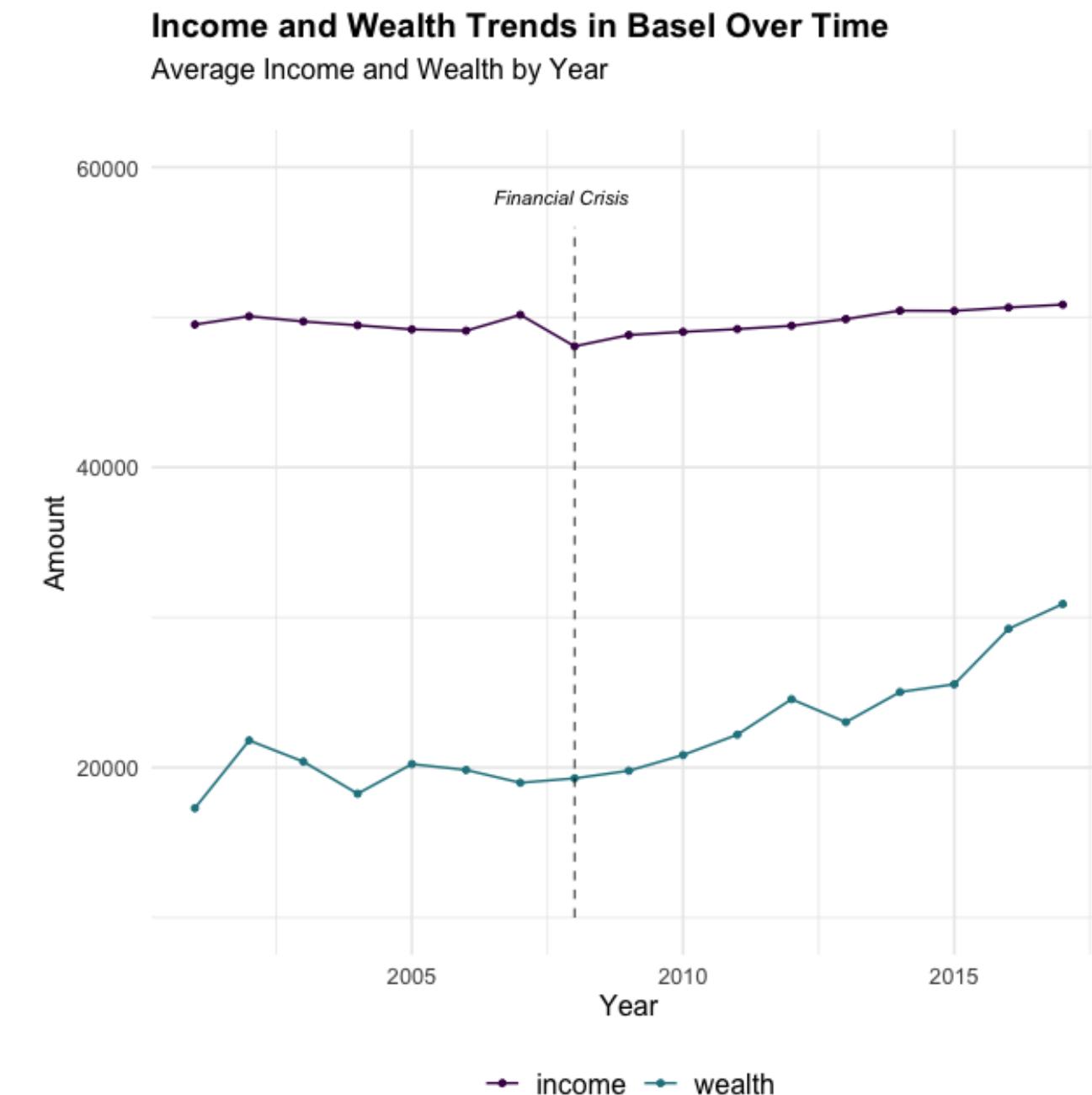
University  
of Cologne



# Why is styling important?

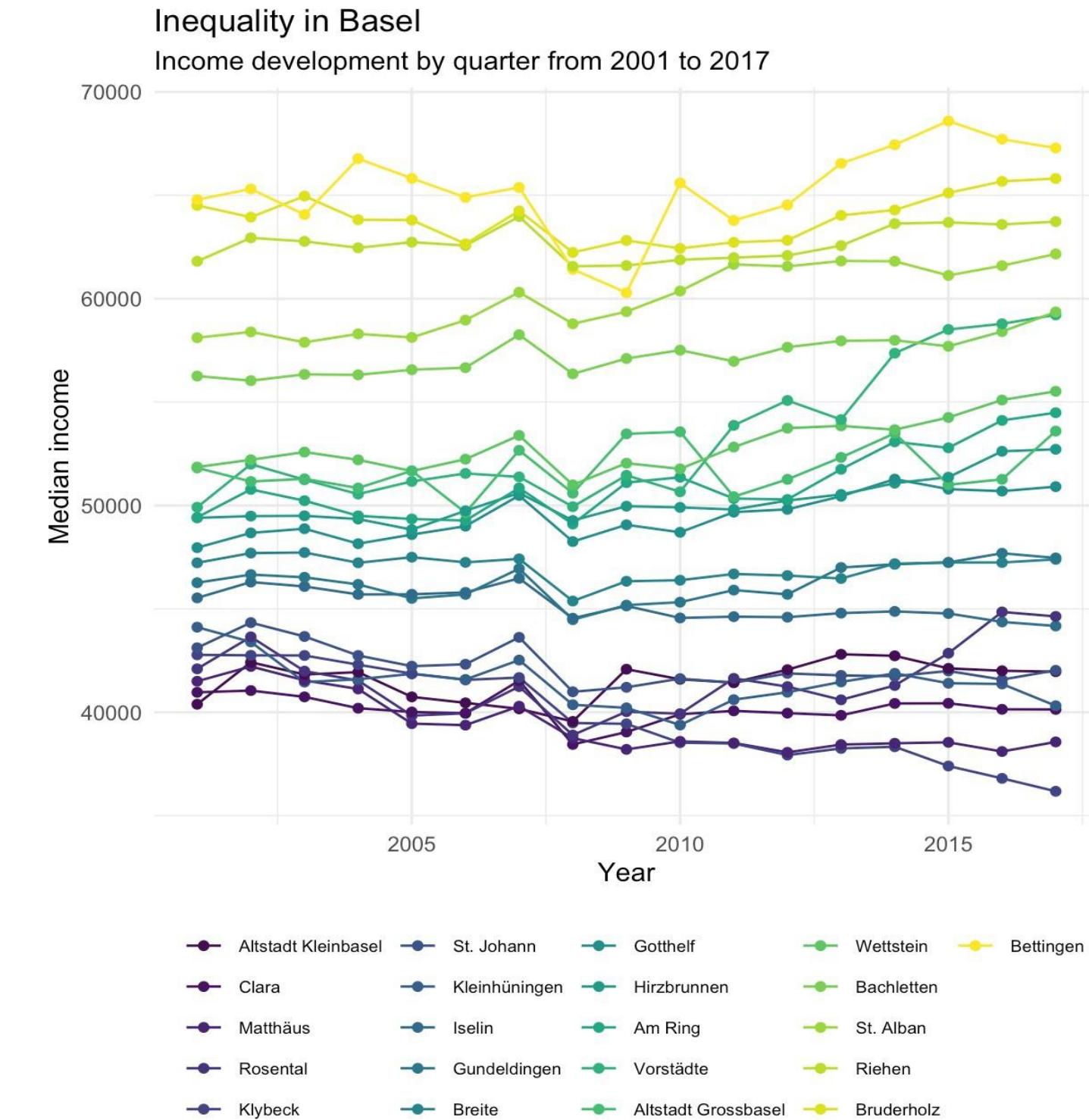


vs

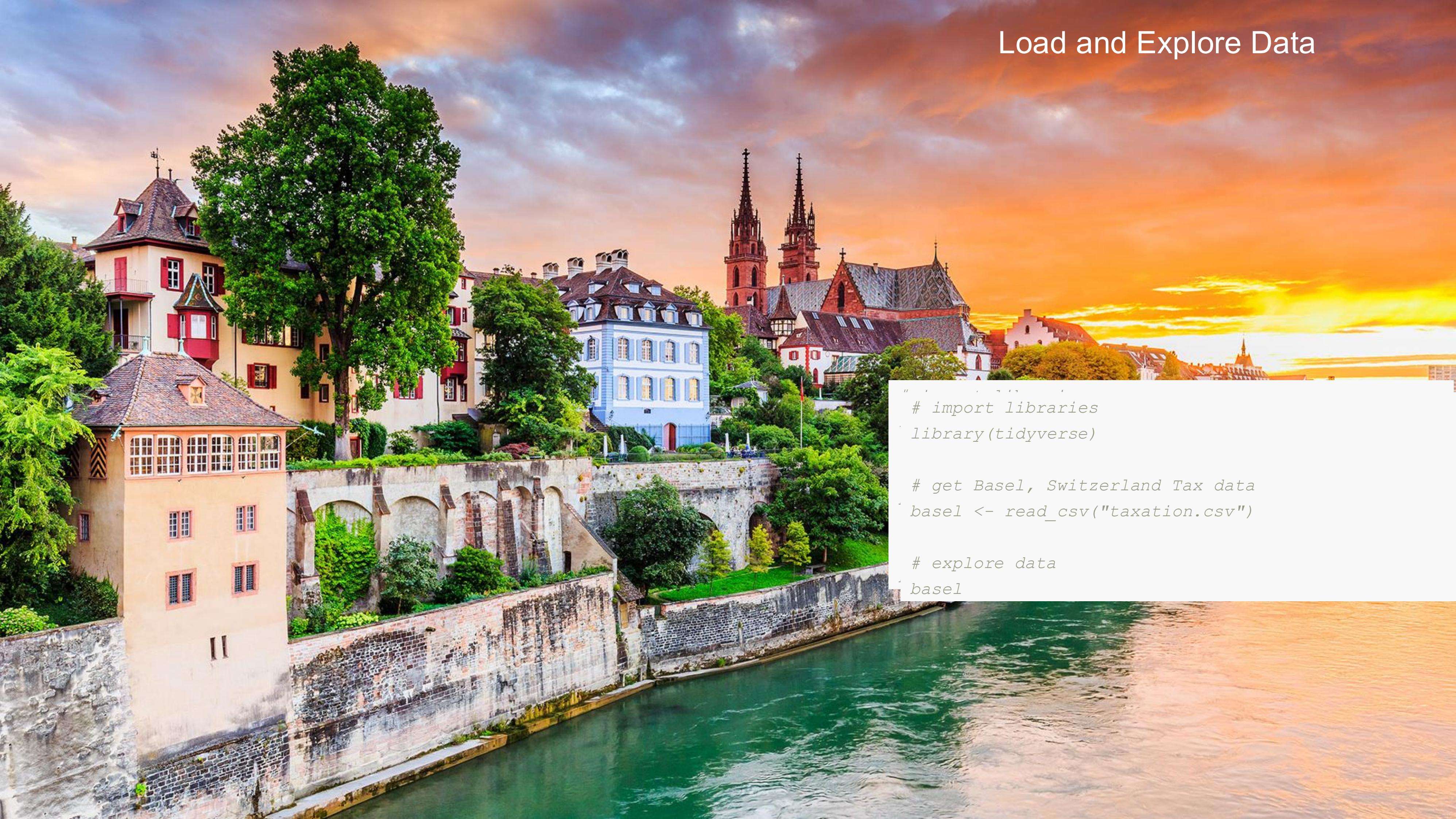


# Styling Plots

- (1) Use existing `theme_*`() presets.
- (2) Customize details using `theme()`
- (3) Adjust dimensions using `scale_*`().
- (4) Add an notion using `labs()`.



# Load and Explore Data



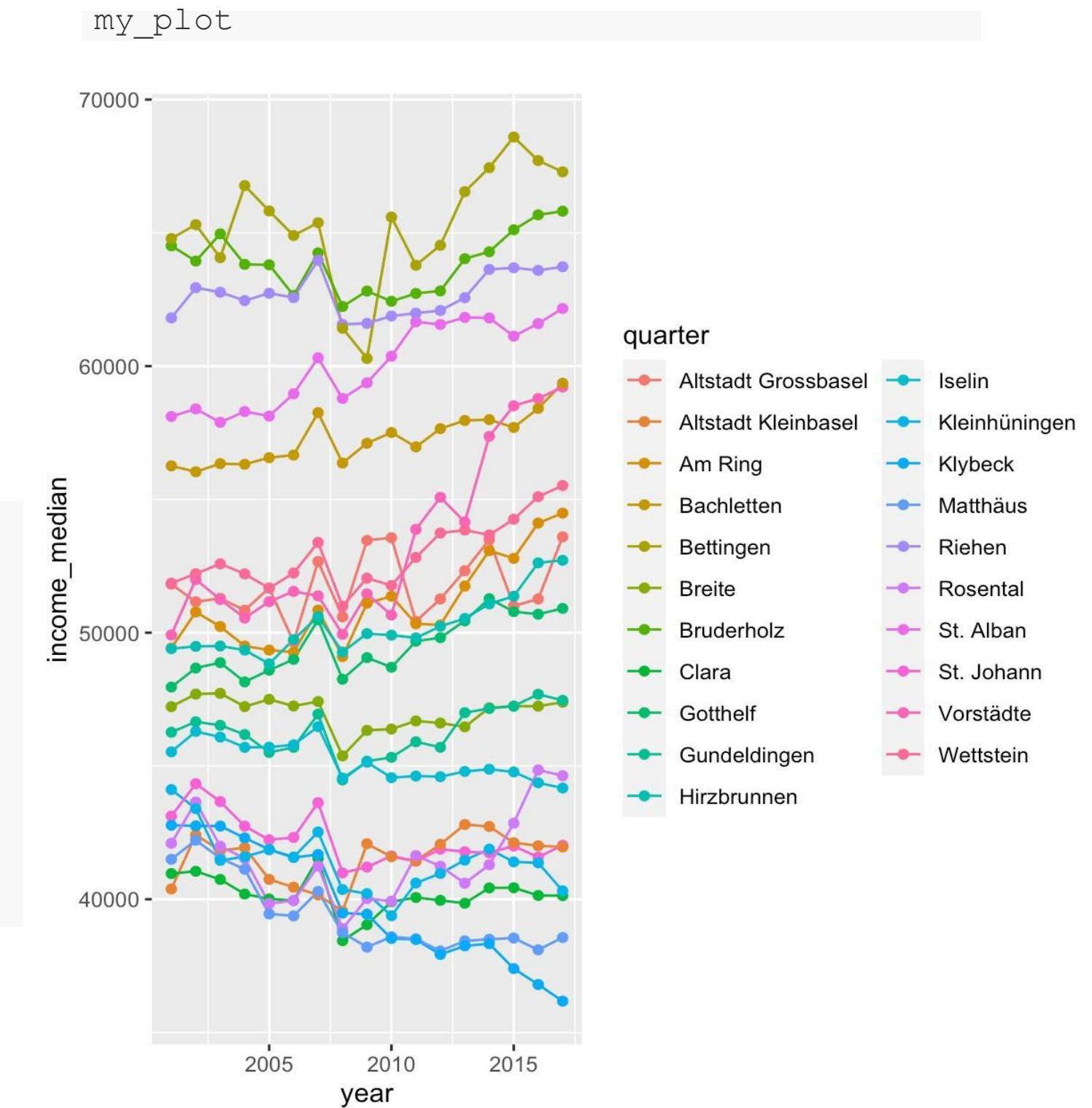
```
"# import libraries  
library(tidyverse)  
  
# get Basel, Switzerland Tax data  
basel <- read_csv("taxation.csv")  
  
# explore data  
basel
```

# The gg object

The output of `ggplot()` can be stored in an gg object.

The gg object can be expanded using `+` and the plot can be generated by a simple print.

```
# store plot as object  
  
myplot <- data %>%  
  ggplot(aes(x=year, y=income_median, col=quarter)) +  
  geom_line() +  
  geom_point()
```



# theme\_\*

Using `theme_*` the plot can be styled according to various presets.

A few themes (go through each theme):

`theme_grey()`

`theme_gray()`

`theme_bw()`

`theme_light()`

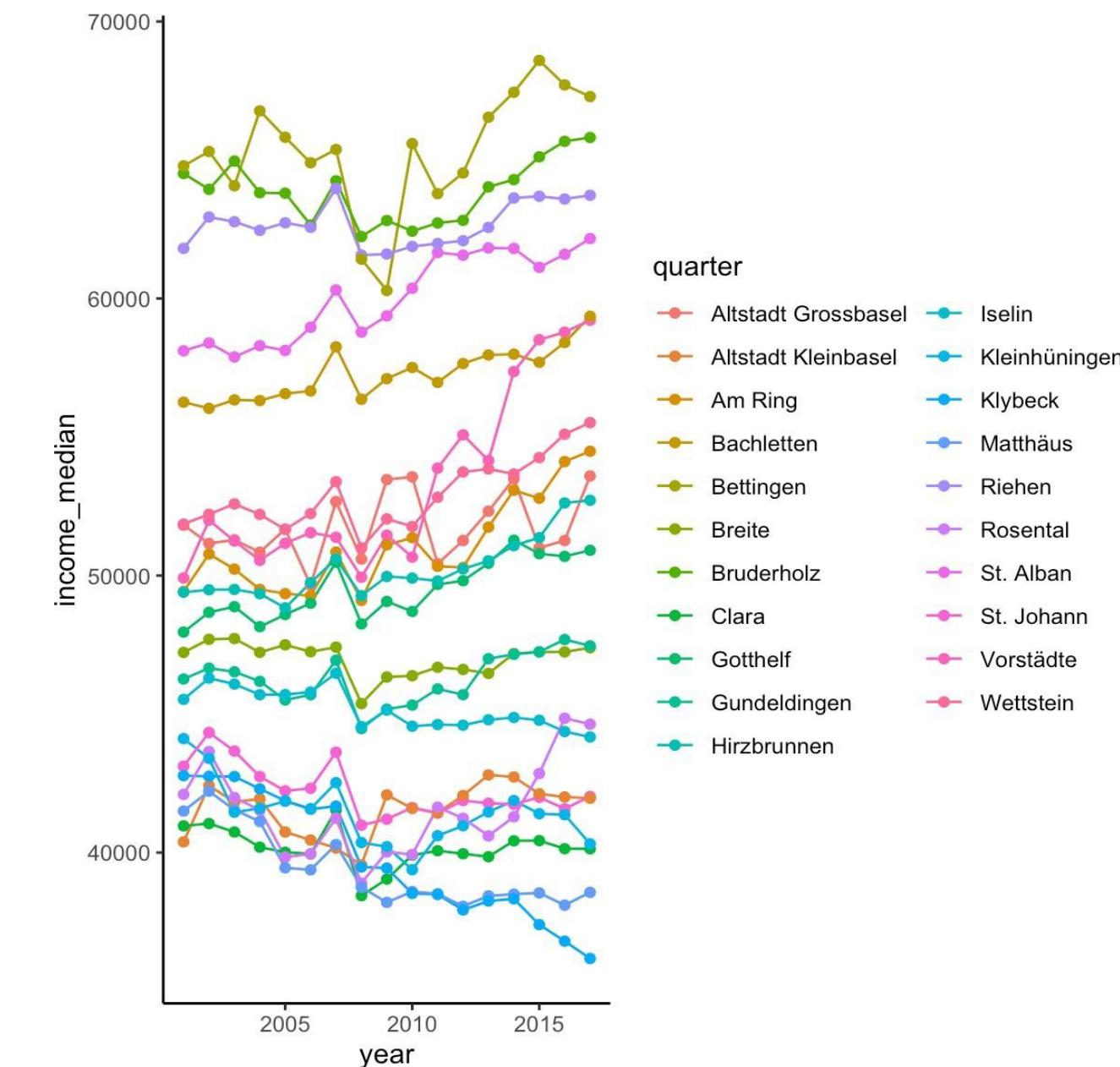
`theme_dark()`

`theme_minimal()`

`theme_classic()`

`theme_void()`

`my_plot + theme_classic()`



# theme\_\*

With **96 arguments** theme() permits specification of all aesthetic details.

Makes use of helper functions:

element\_rect() | for rectangles

element\_line() | for lines

element\_text() | for text

element\_blank() | for removals

```
# Using theme
my_plot +
  theme(argument =
    element_*(),
    argument =
    element_*(),
    ...)
```

theme {ggplot2}

R Documentation

## Modify components of a theme

### Description

Use theme() to modify individual components of a theme, allowing you to control the appearance of all non-data components of the plot. theme() only affects a single plot: see [theme\\_update\(\)](#) if you want modify the active theme, to affect all subsequent plots.

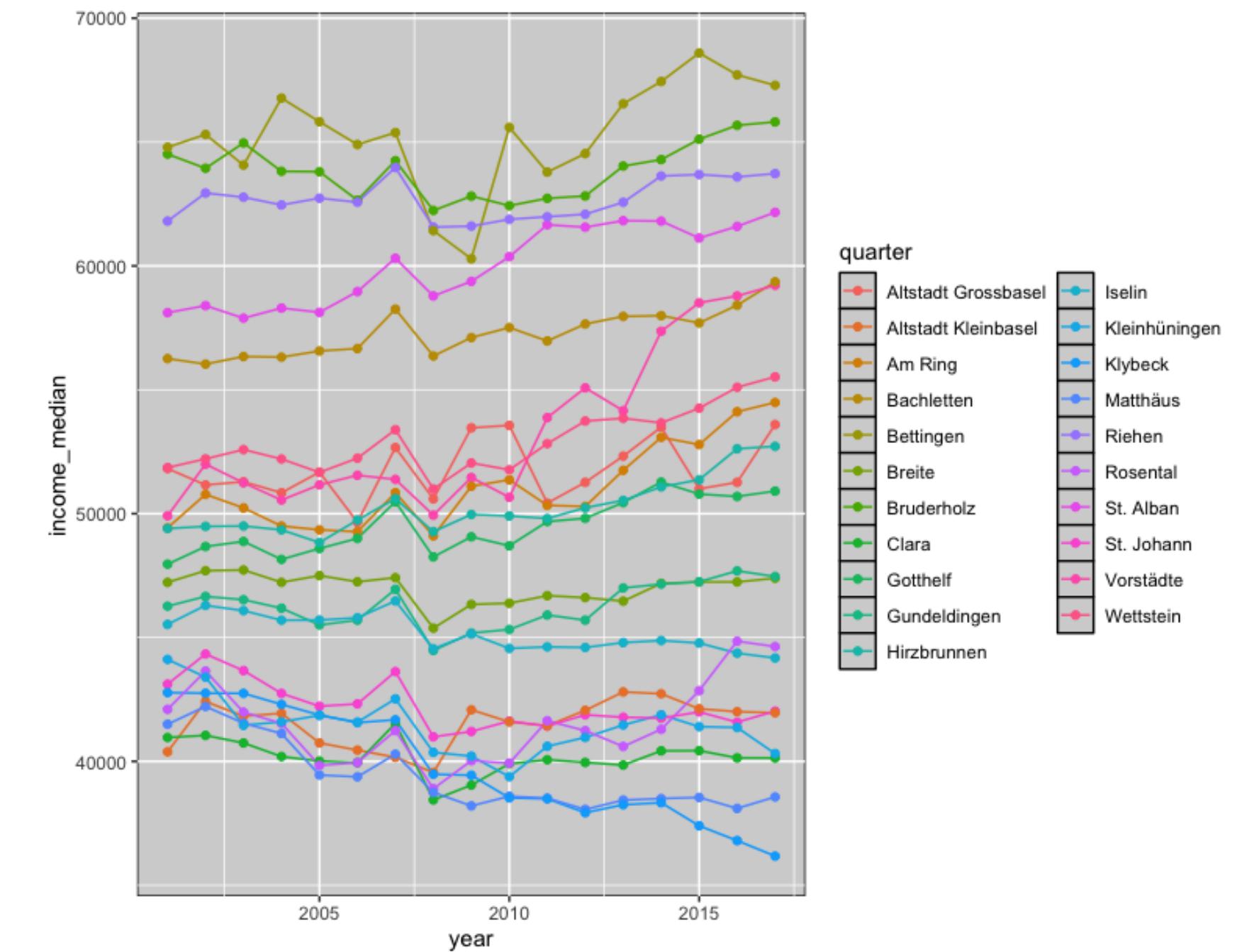
### Usage

```
theme(line, rect, text, title, aspect.ratio, axis.title, axis.title.x,
      axis.title.x.top, axis.title.x.bottom, axis.title.y, axis.title.y.left,
      axis.title.y.right, axis.text, axis.text.x, axis.text.x.top,
      axis.text.x.bottom, axis.text.y, axis.text.y.left, axis.text.y.right,
      axis.ticks, axis.ticks.x, axis.ticks.x.top, axis.ticks.x.bottom, axis.ticks.y,
      axis.ticks.y.left, axis.ticks.y.right, axis.ticks.length, axis.line,
      axis.line.x, axis.line.x.top, axis.line.x.bottom, axis.line.y,
      axis.line.y.left, axis.line.y.right, legend.background, legend.margin,
      legend.spacing, legend.spacing.x, legend.spacing.y, legend.key,
      legend.key.size, legend.key.height, legend.key.width, legend.text,
      legend.text.align, legend.title, legend.title.align, legend.position,
      legend.direction, legend.justification, legend.box, legend.box.just,
      legend.box.margin, legend.box.background, legend.box.spacing,
      panel.background, panel.border, panel.spacing, panel.spacing.x,
      panel.spacing.y, panel.grid, panel.grid.major, panel.grid.minor,
      panel.grid.major.x, panel.grid.major.y, panel.grid.minor.x,
      panel.grid.minor.y, panel.on top, plot.background, plot.title, plot.subtitle,
      plot.caption, plot.tag, plot.tag.position, plot.margin, strip.background,
      strip.background.x, strip.background.y, strip.placement, strip.text,
      strip.text.x, strip.text.y, strip.switch.pad.grid, strip.switch.pad.wrap, ...,
      complete = FALSE, validate = TRUE)
```

# theme\_\*

The `element_rect()` function is used to define the appearance of rectangular elements in a plot

```
# using element_rect ()  
myplot +  
  theme (panel.background = element_rect  
        (fill = "lightgray", color = "black"))
```

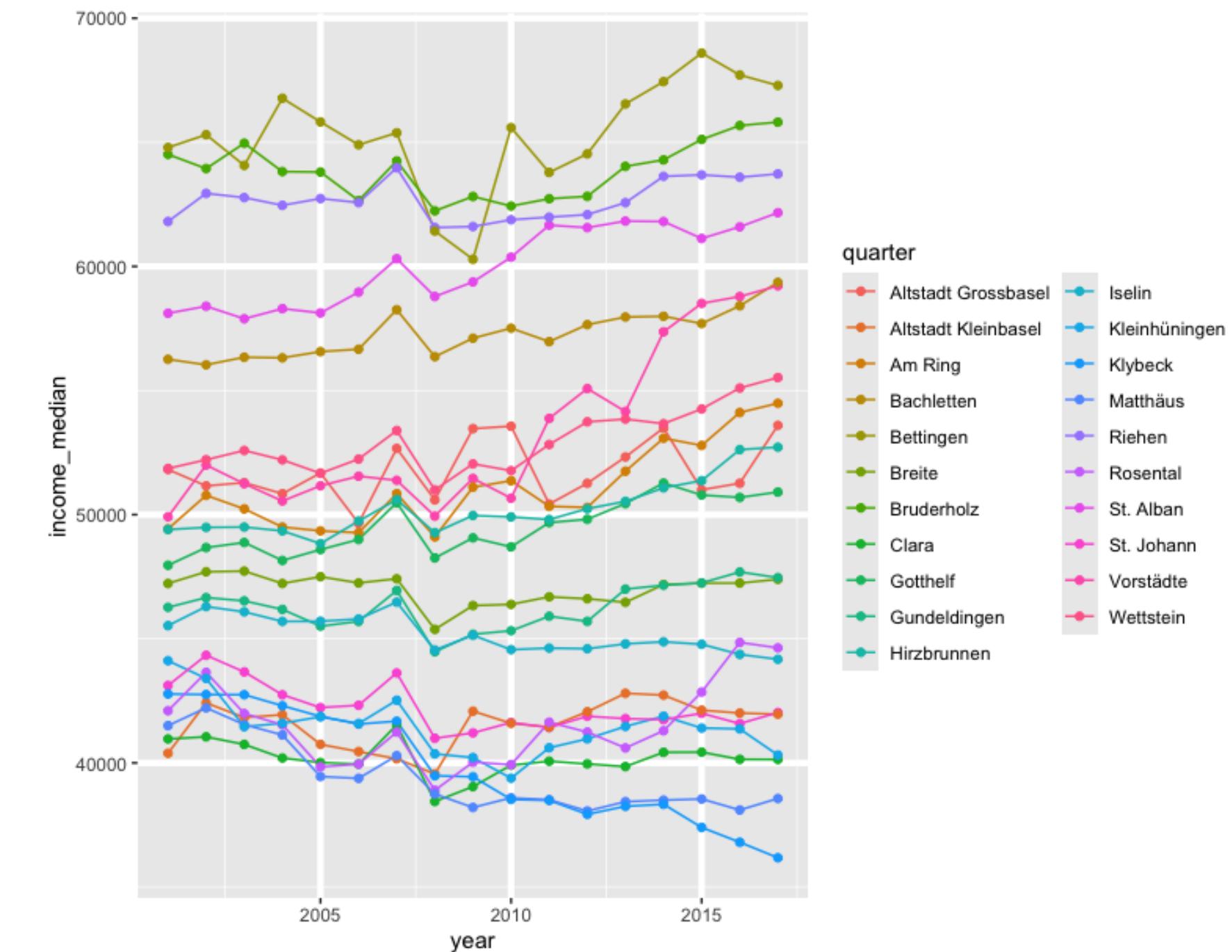


# theme\_\*

The `element_line()` function is used to define the appearance of **line-based elements** in a plot.

It allows you to customize elements such as grid lines, axis lines, and tick marks.

```
# linewidth  
myplot +  
  theme(panel.grid.major = element_line  
        (color = "white", linewidth= 1.5))
```

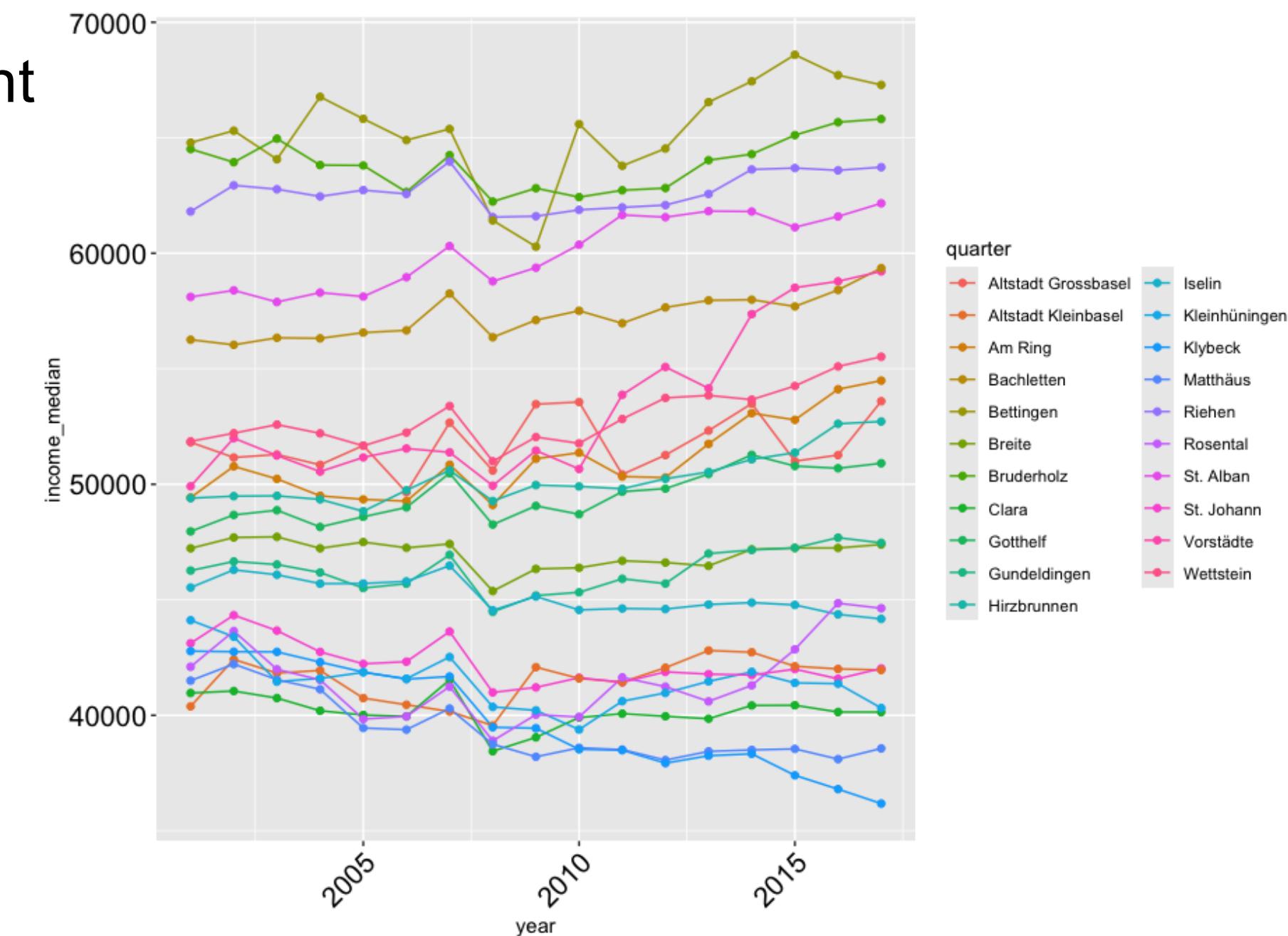


# theme\_\*

The `element_text()` function is used to customize the appearance of **text-based elements**

It provides control over the size, color, alignment, font style, and more.

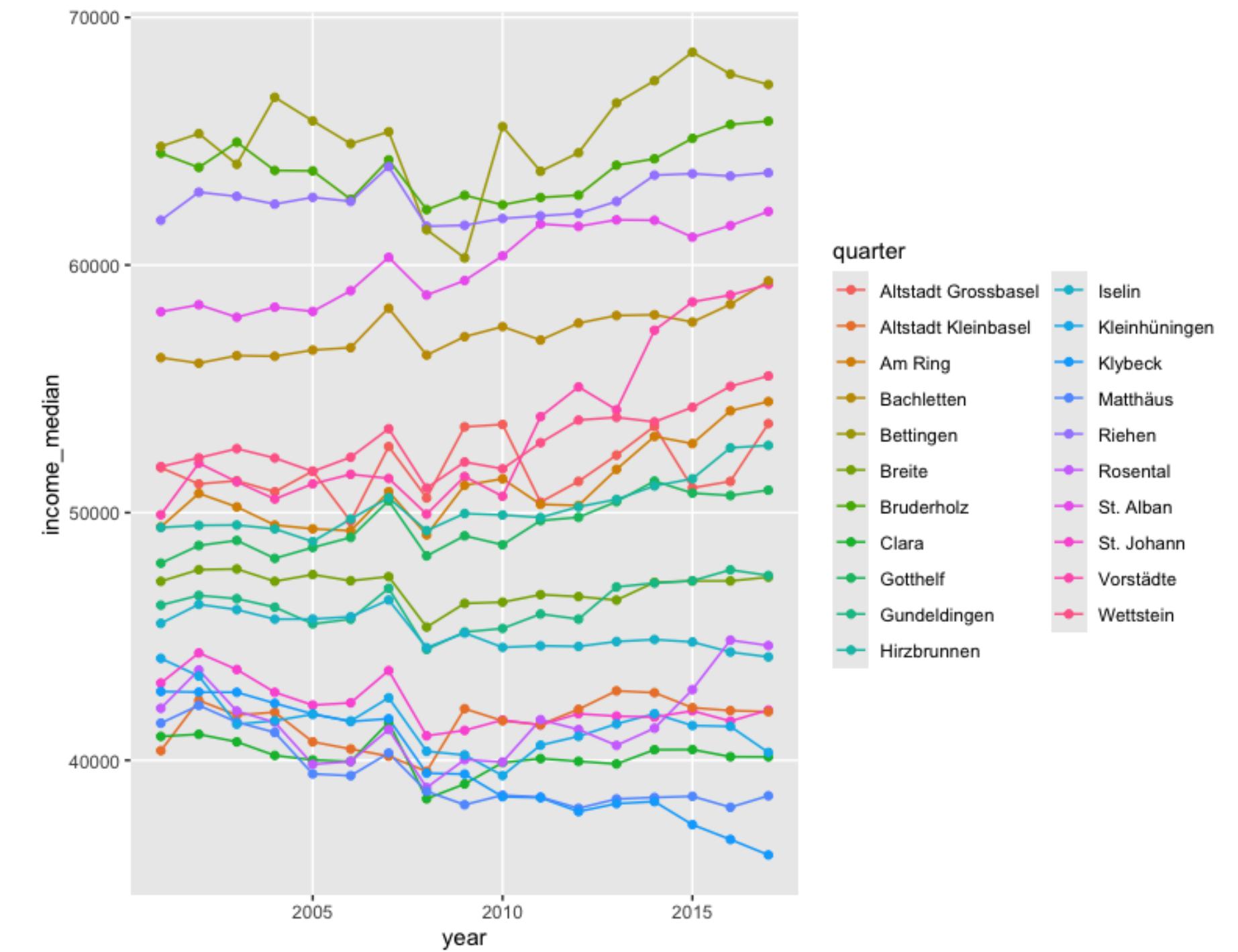
```
# element_text  
  
myplot +  
  theme (axis.text.x = element_text  
         (size = 15, angle = 45, hjust = 1,  
          color = "black"),  
        axis.text.y = element_text  
         (size = 15, color = "black"))
```



# theme\_\*

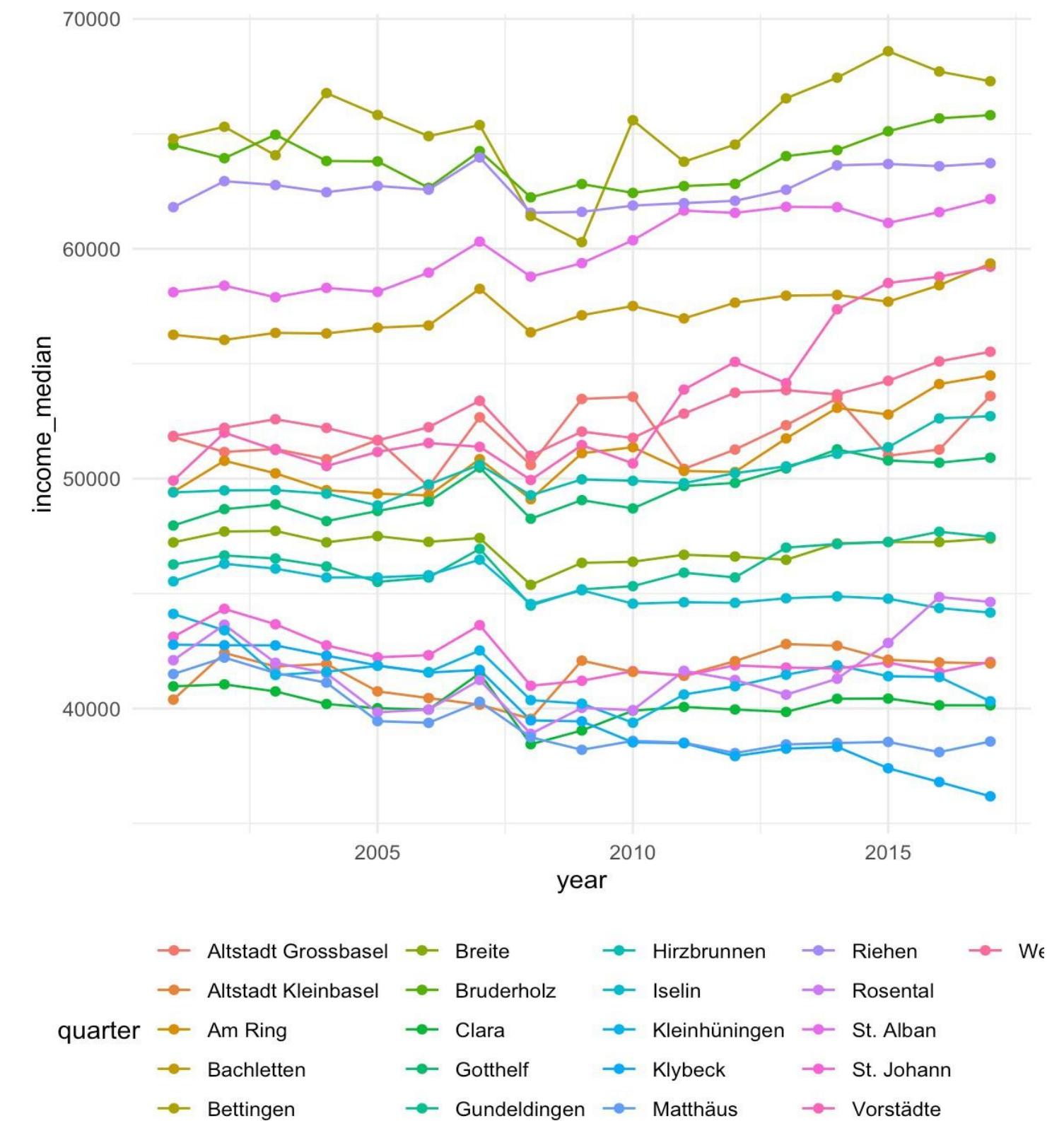
The `element_blank()` function is used to completely **remove or hide graphical elements** from a plot.

```
# element_blank  
myplot +  
  theme (panel.grid.minor =  
        element_blank() )
```



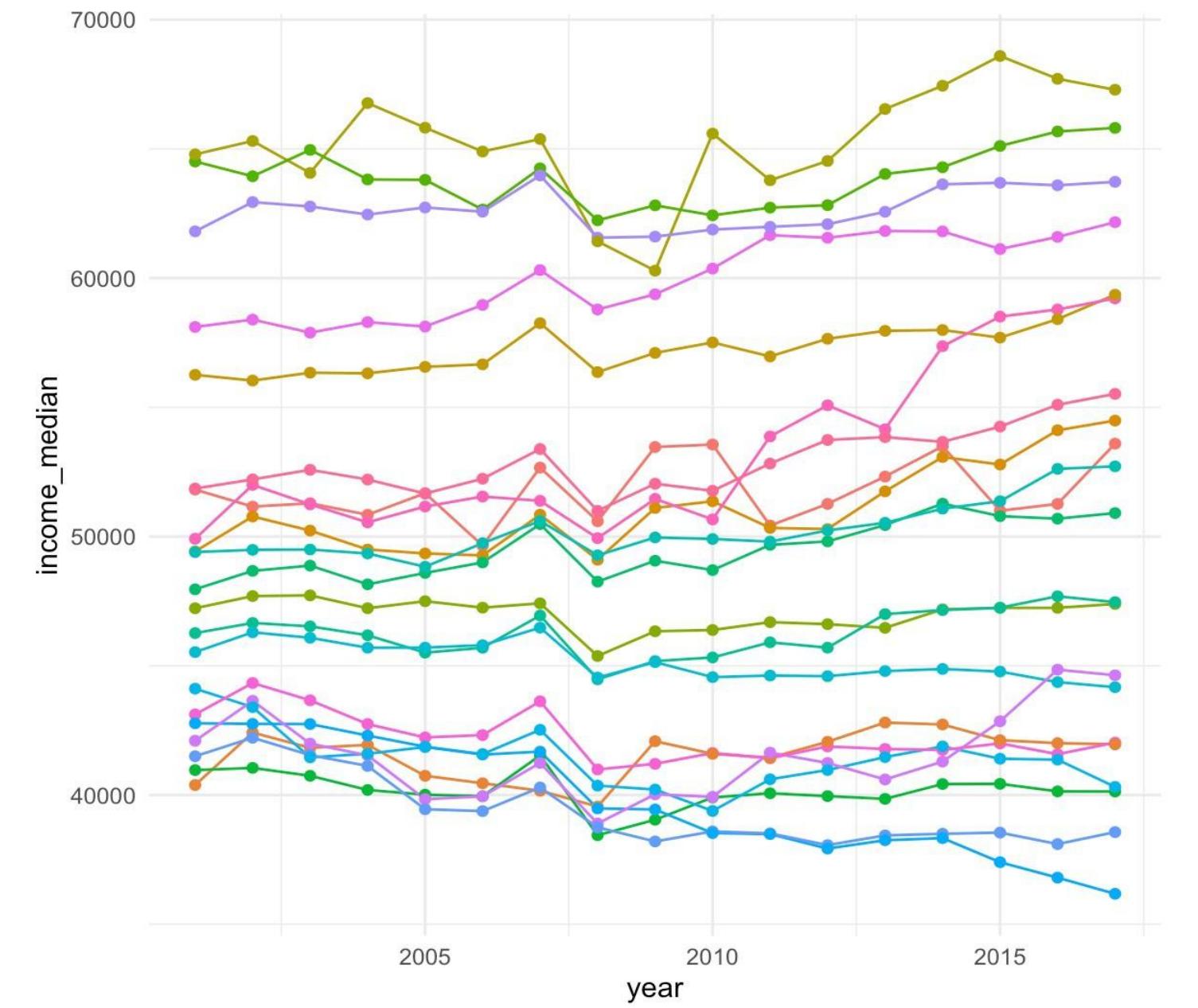
# theme\_\*

```
# Fixing the legend  
myplot +  
  theme_minimal() +  
  
# move legend to bottom  
theme(legend.position = "bottom")
```



# theme\_\*

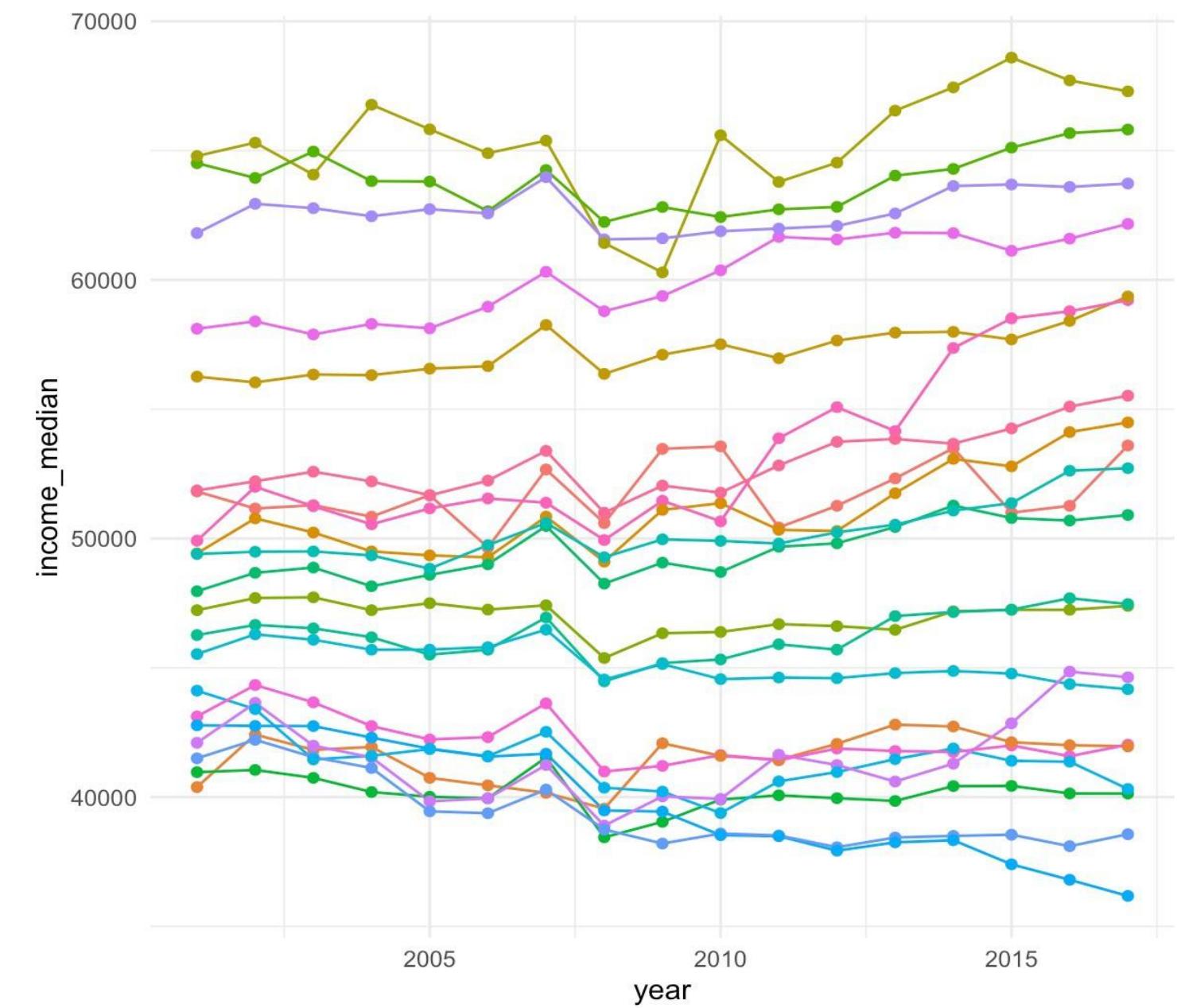
```
# Fixing the legend  
myplot +  
  theme_minimal() +  
  theme(legend.position = "bottom",  
  
# remove legend title  
  legend.title = element_blank())
```



Altstadt Grossbasel	Breite	Hirzbrunnen	Riehen	Wettste
Altstadt Kleinbasel	Bruderholz	Iselin	Rosental	
Am Ring	Clara	Kleinbüning	St. Alban	
Bachletten	Gotthelf	Klybeck	St. Johann	
Bettingen	Gundeldingen	Matthäus	Vorstädte	

# theme\_\*

```
# Fixing the legend
my_plot +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    legend.title =
      element_blank(),
    # reduce legend text size
    legend.text = element_text(size = 7))
```



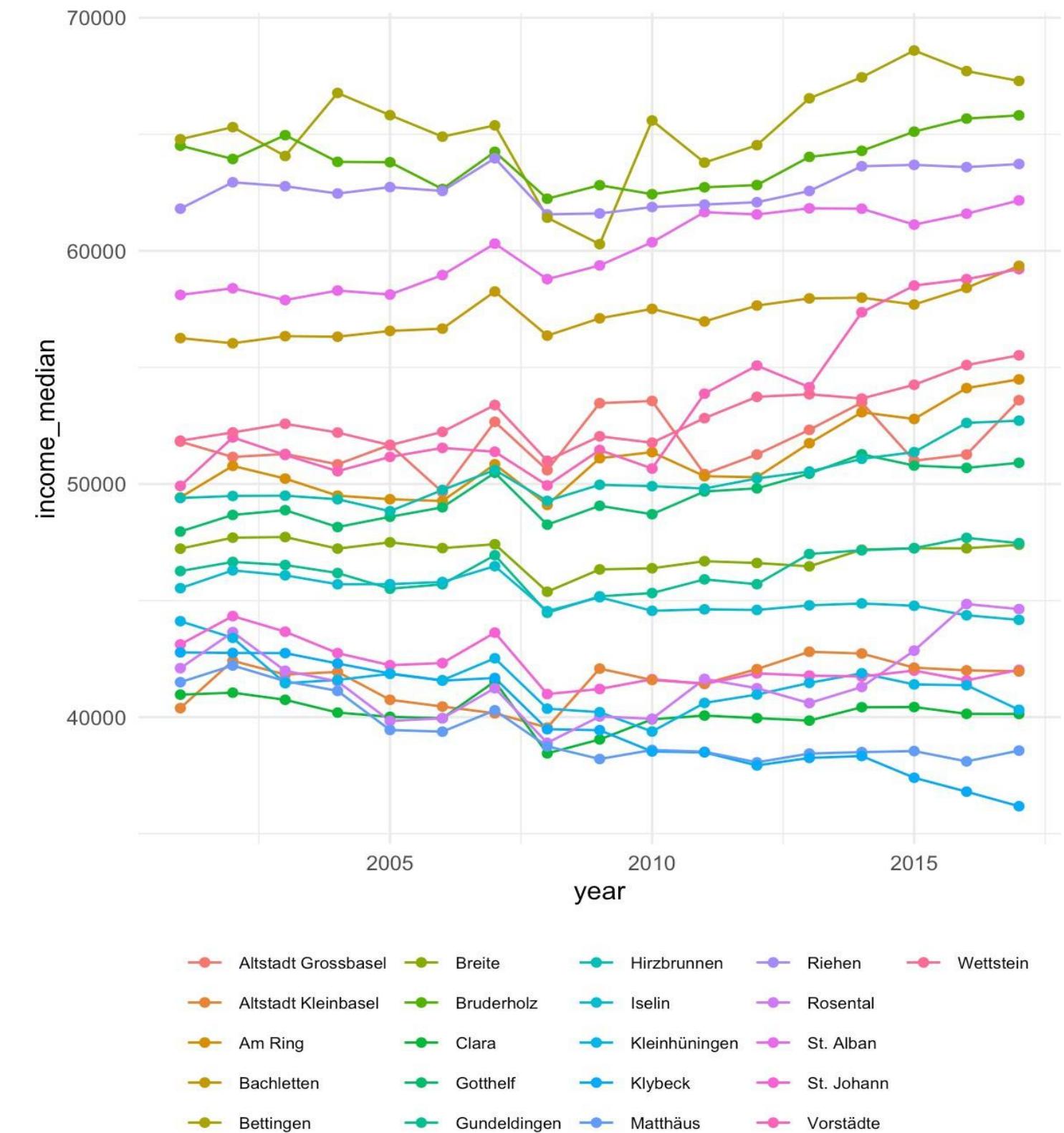
Altstadt Grossbasel	Breite	Hirzbrunnen	Riehen	Wettstein
Altstadt Kleinbasel	Bruderholz	Iselin	Rosental	
Am Ring	Clara	Kleinhüningen	St. Alban	
Bachletten	Gotthelf	Klybeck	St. Johann	
Bettingen	Gundeldingen	Matthäus	Vorstädte	

# theme\_\*

We can also save our adjustments as an object  
an combine it with a preexisting theme.

```
# save my theme
my_theme <-
  theme(legend.position =
    "bottom", legend.title =
    element_blank(),
    legend.text = element_text(size = 7))

# Add my theme
my_plot +
  theme_minimal() +
  my_theme
```



# scale\_\*

Various scale\_\*() functions permit specification of all **dimensions**, including axes, colors, sizes, etc.

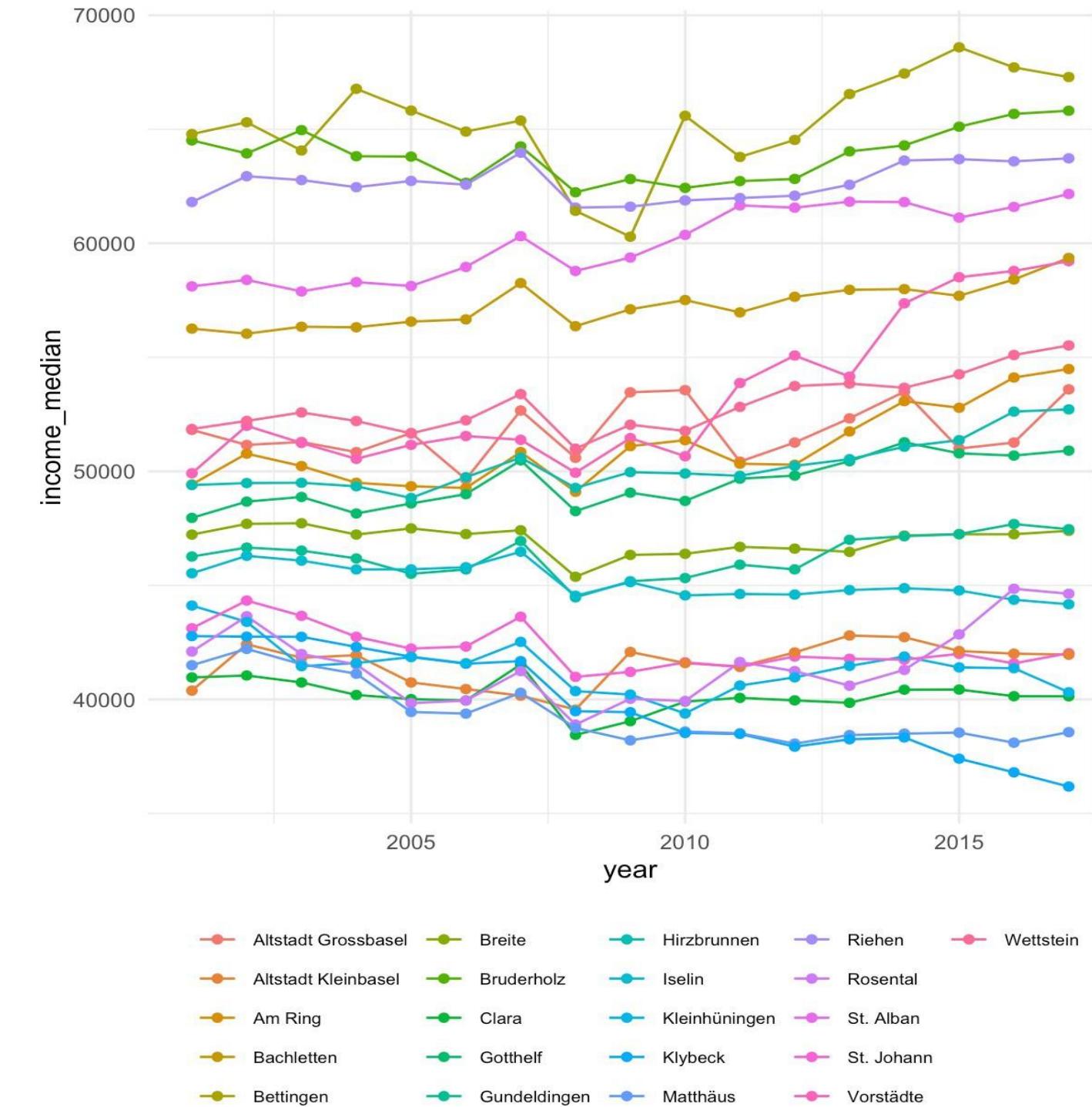
Groups of scale\_\*() functions:

scale\_xy\_\* | scales axes

scale\_size\_\* | scales sizes

scale\_alpha\_\* | scales opacity

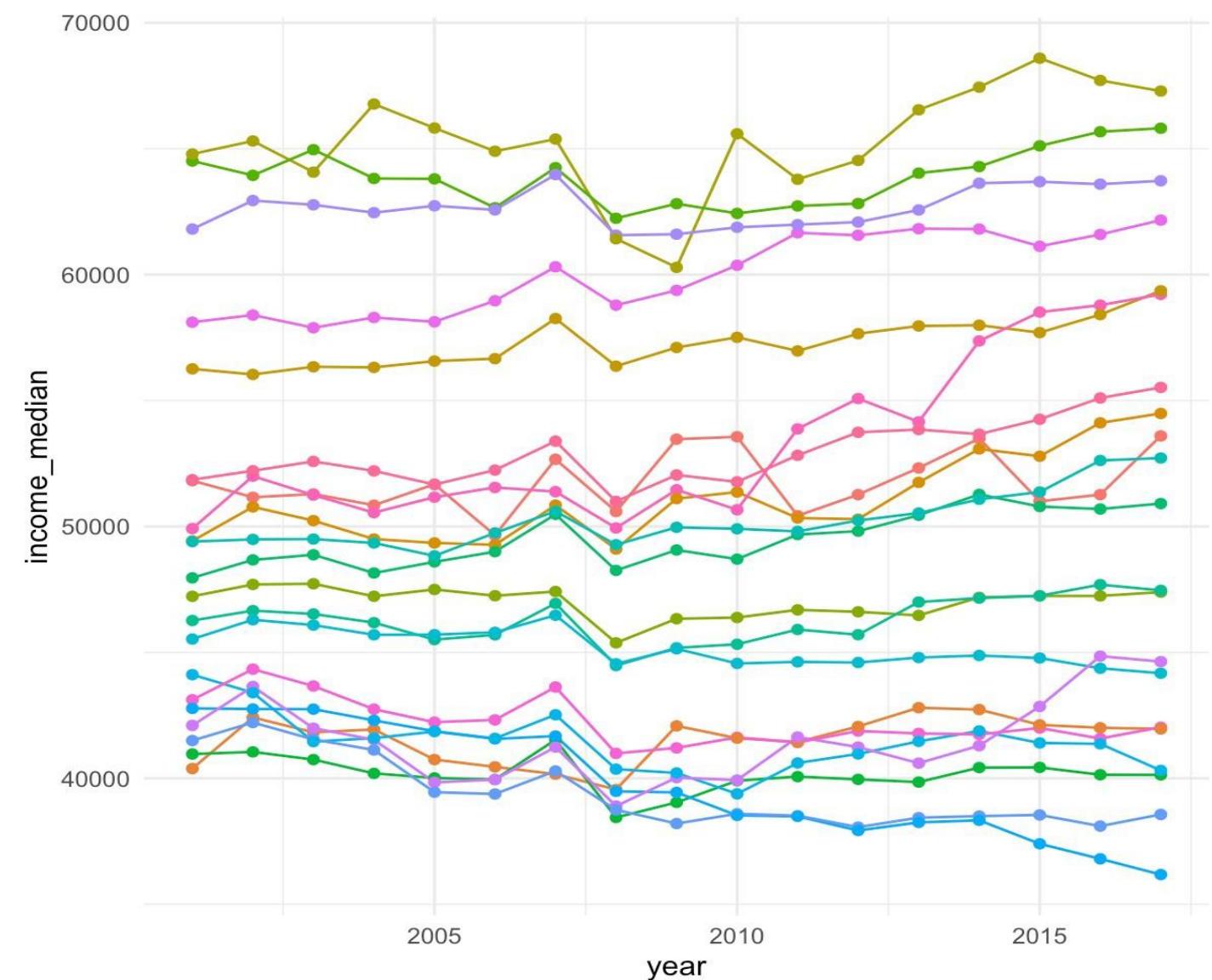
scale\_color\_\* | scales colors



# scale\_\*

scale\_xy\_\* can be used to set costum scales

```
# scale_xy  
myplot + scale_x_continuous(  
  limits = c(2001, 2017),  
  breaks = seq (2001, 2017, by = 3))
```

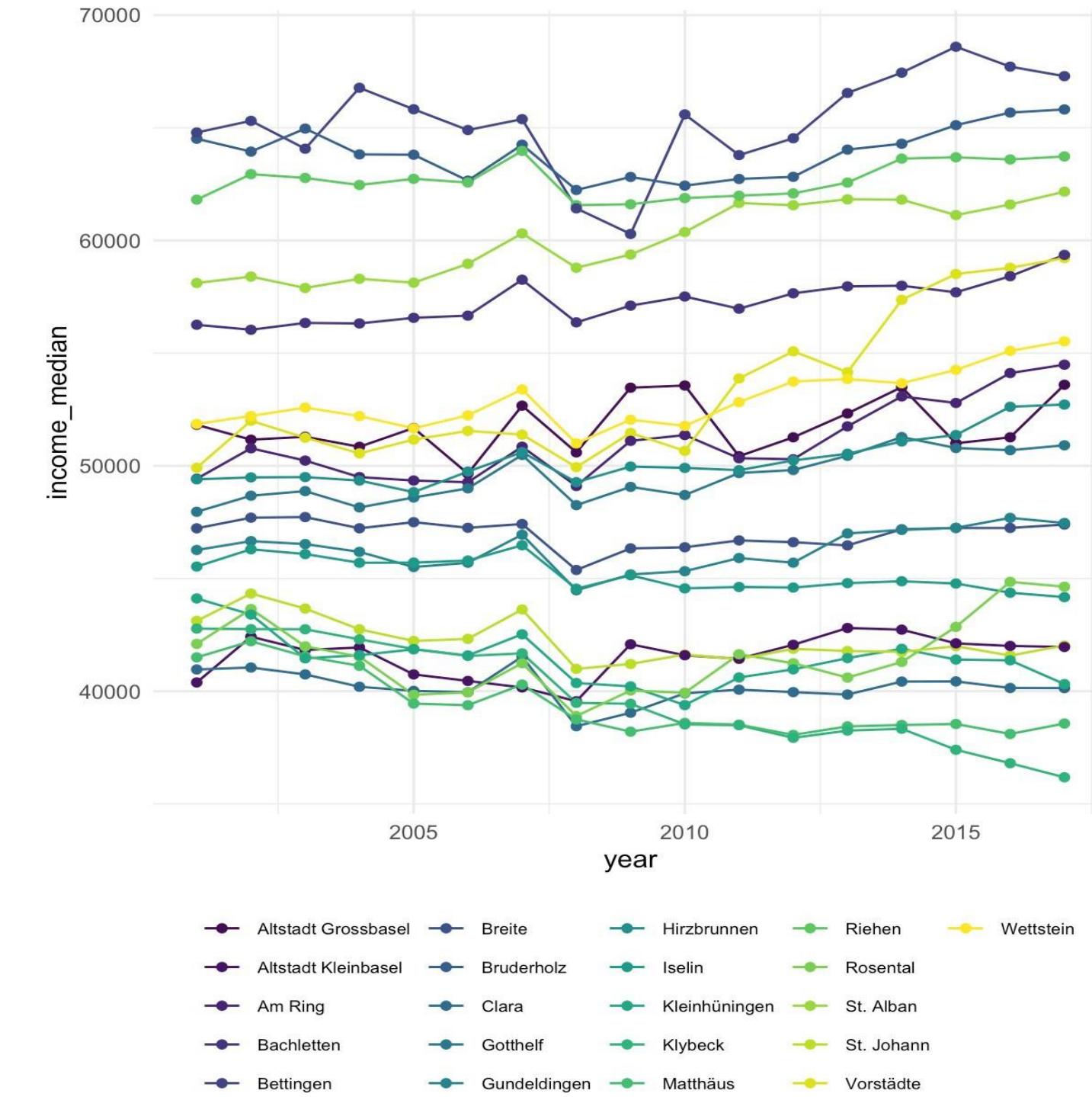


Altstadt Grossbasel	Breite	Hirzbrunnen	Riehen	Wettstein
Altstadt Kleinbasel	Bruderholz	Iselin	Rosental	
Am Ring	Clara	Kleinhüningen	St. Alban	
Bachletten	Gotthelf	Klybeck	St. Johann	
Bettingen	Gundeldingen	Matthäus	Vorstädte	

# scale\_\*

Various scale\_\*() functions permit specification of all **dimensions**, including axes, colors, sizes, etc.

```
# Fixing the legend  
my_plot +  
  theme_minimal() +  
  theme(  
    legend.position = "bottom",  
    legend.title = element_blank(),  
    legend.text = element_text(size=7) +  
  
# color using viridis  
  scale_color_viridis_d()
```



# Wrangling

Again, wrangling can help with plotting. The order of discrete variables can be controlled using **factors**.

```
# change data
basel_order <- data %>%
  # factor ordered by income in 2001
  arrange(year, income_median) %>%
  mutate(quarter = as_factor(quarter))

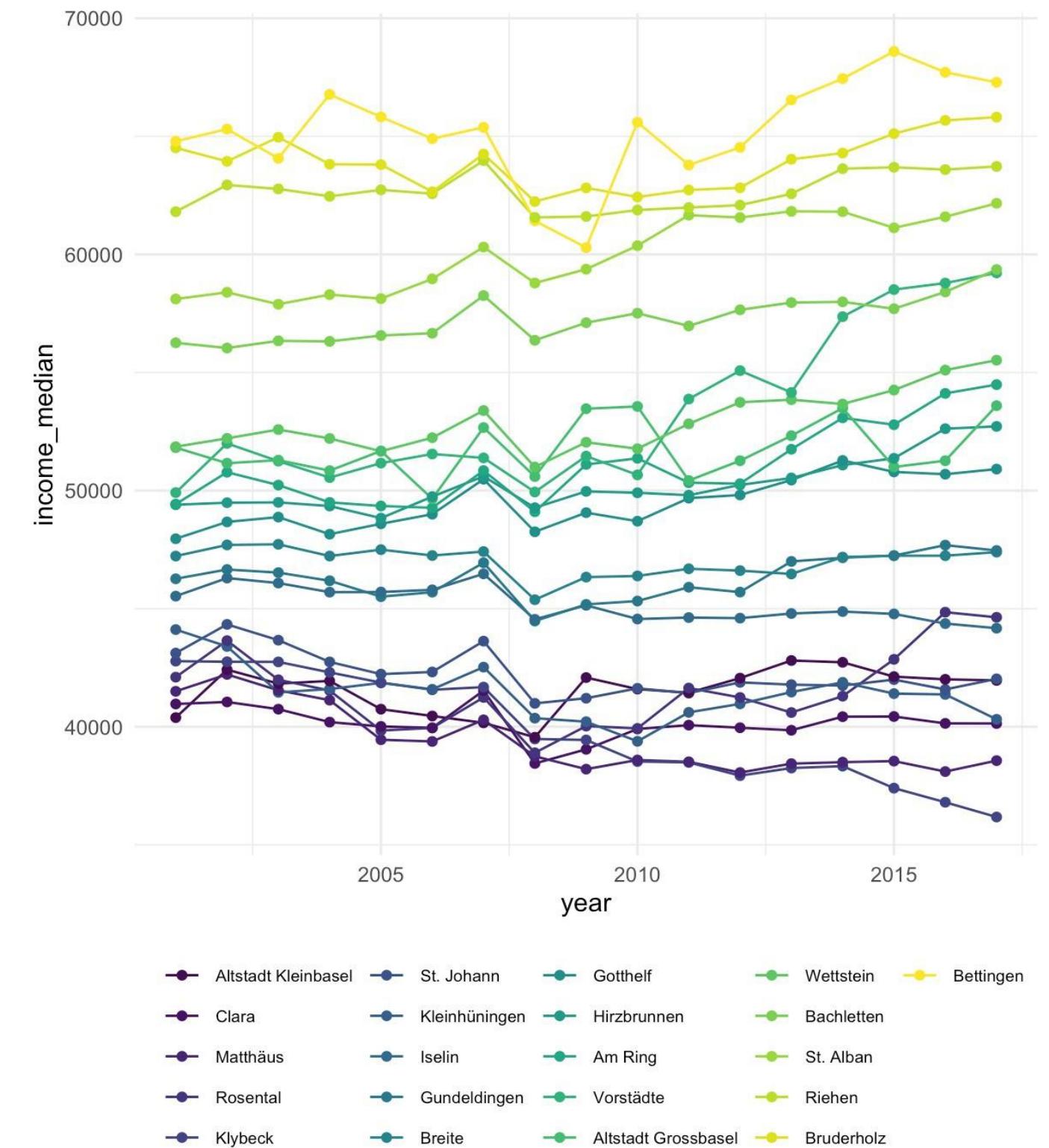
# show data
basel_order
```

	year	quarter	quarter_no	N	income_mean	income_median
	<dbl>	<fct>	<dbl>	<dbl>	<dbl>	<dbl>
1	2001	Altstadt Kleinbasel	12	1659	51648	40387
2	2001	Clara	13	2416	47435	40964
3	2001	Matthäus	17	9089	48892	41500
4	2001	Rosental	16	2499	46221	42100
5	2001	Klybeck	18	4053	45651	42777
6	2001	St. Johann	11	10493	48766	43118
7	2001	Kleinhüningen	19	1363	46859	44115
8	2001	Iselin	10	9853	49631	45530
9	2001	Gundeldingen	6	11224	51229	46265
10	2001	Breite	4	5433	52039	47227

# Wrangling

Colors are assigned according to the order of factor levels

```
basel_order %>%
  # original code
  ggplot(aes(x = year, y = income_median,
             col = quarter)) +
  geom_line() + geom_point() +
  theme_minimal() +
  theme(legend.position = "bottom",
        legend.title = element_blank(),
        legend.text = element_text(size=7)) +
  # ordered implicitly
  scale_color_viridis_d(option = "viridis")
```

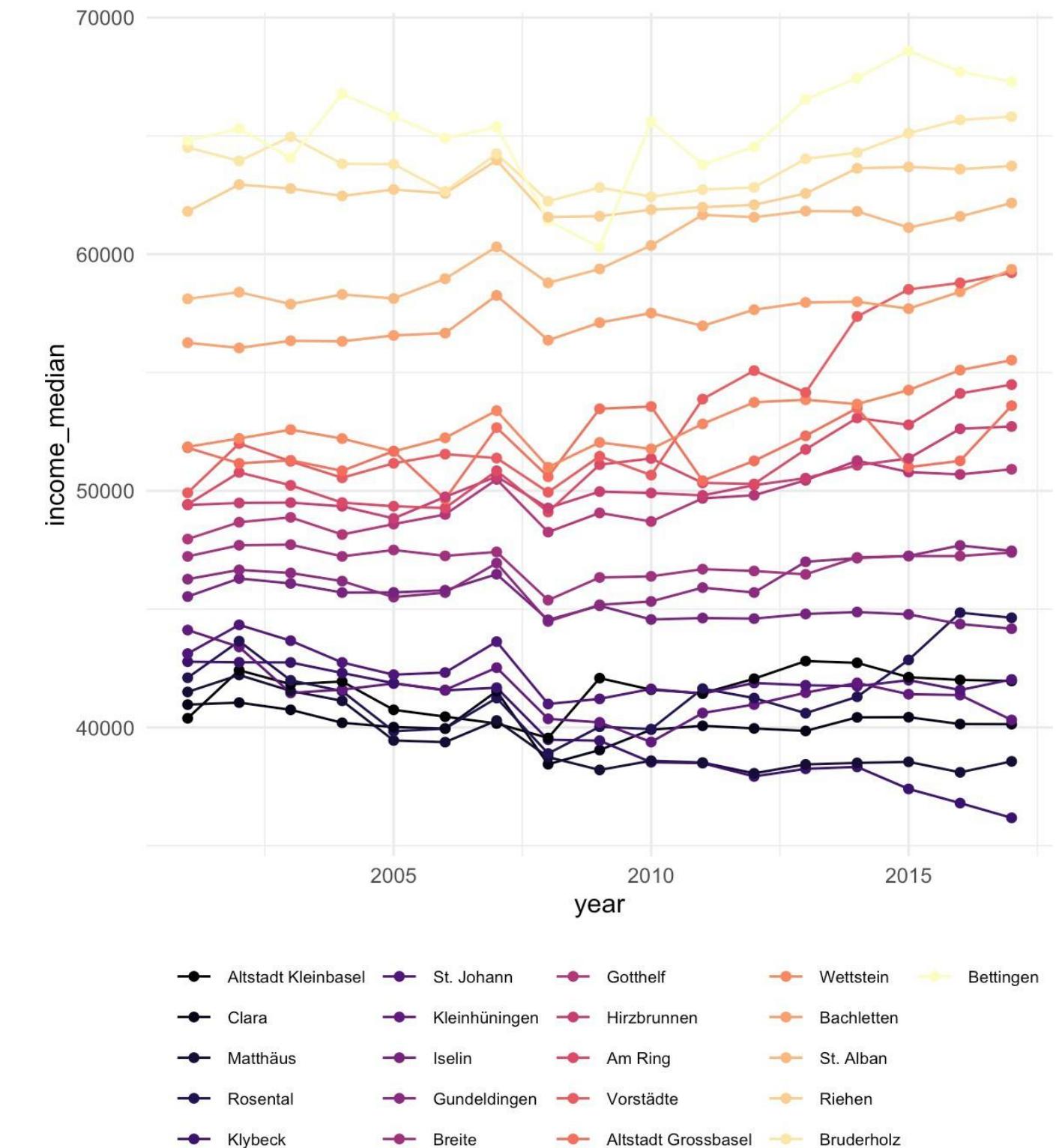




# Viridis

viridis includes several preset color sets that can **useful for different scenarios.**

```
basel_order %>%  
  
# original code  
ggplot(aes(x = year, y = income_median,  
col = quarter)) +  
  geom_line() + geom_point() +  
  theme_minimal() +  
  theme(legend.position = "bottom",  
        legend.title = element_blank(),  
        legend.text = element_text(size=7)) +  
  
# ordered implicitly  
  scale_color_viridis_d(option = "plasma")
```



# Labs ()

Various annotations can be added using `labs()`.

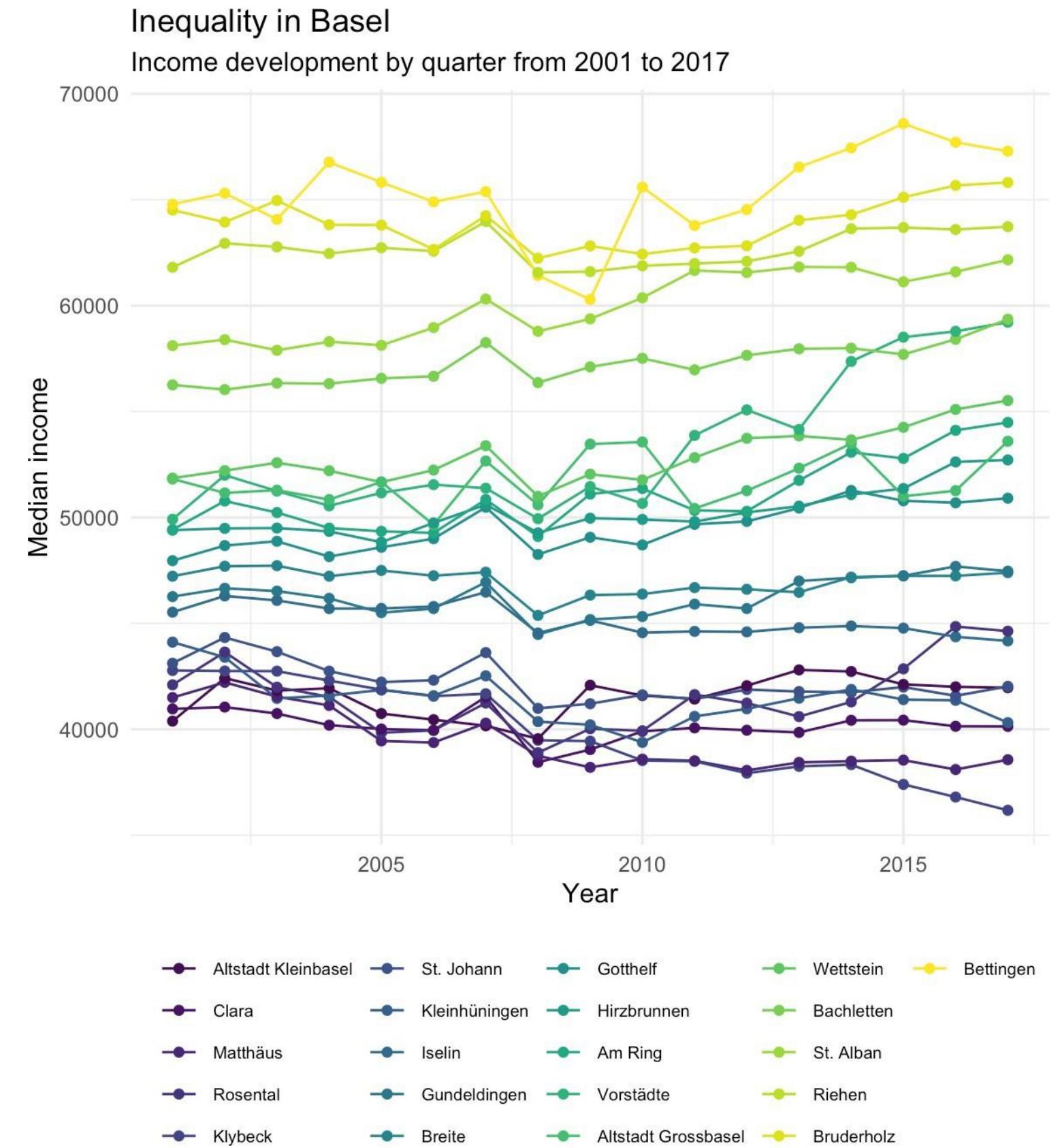
Key arguments:

`x, y | axes`

`title, subtitle | title and subtitle`

`caption | caption`

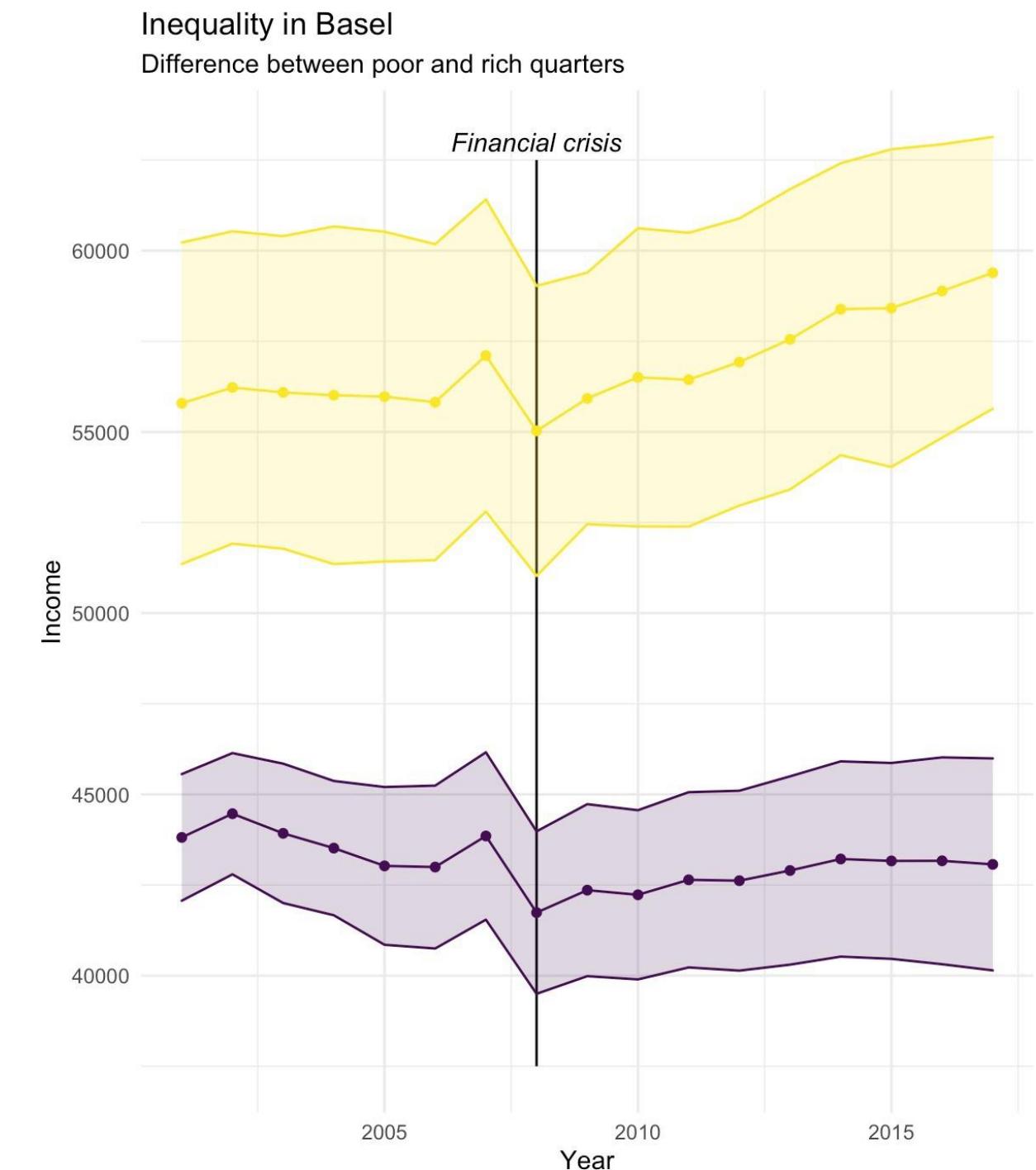
```
my_plot +  
  labs(x = "Year",  
       y = "Median income",  
       title = "Inequality in Basel",  
       subtitle = "Income development...",  
       caption = "Source: Open Data...")
```



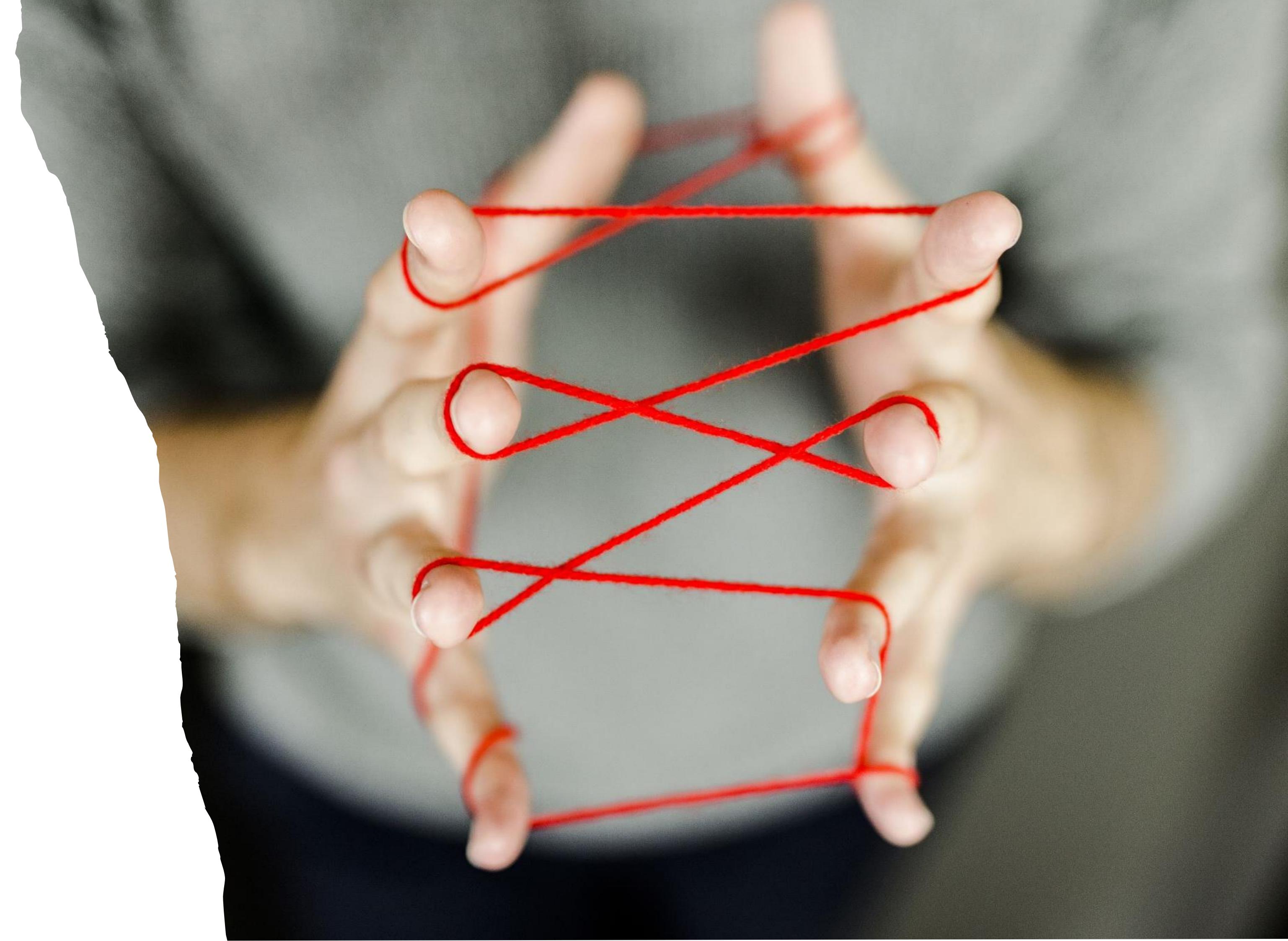
# annotate ()

annotate() can be used to **add any type of plot elements** (layers) using vector specification.

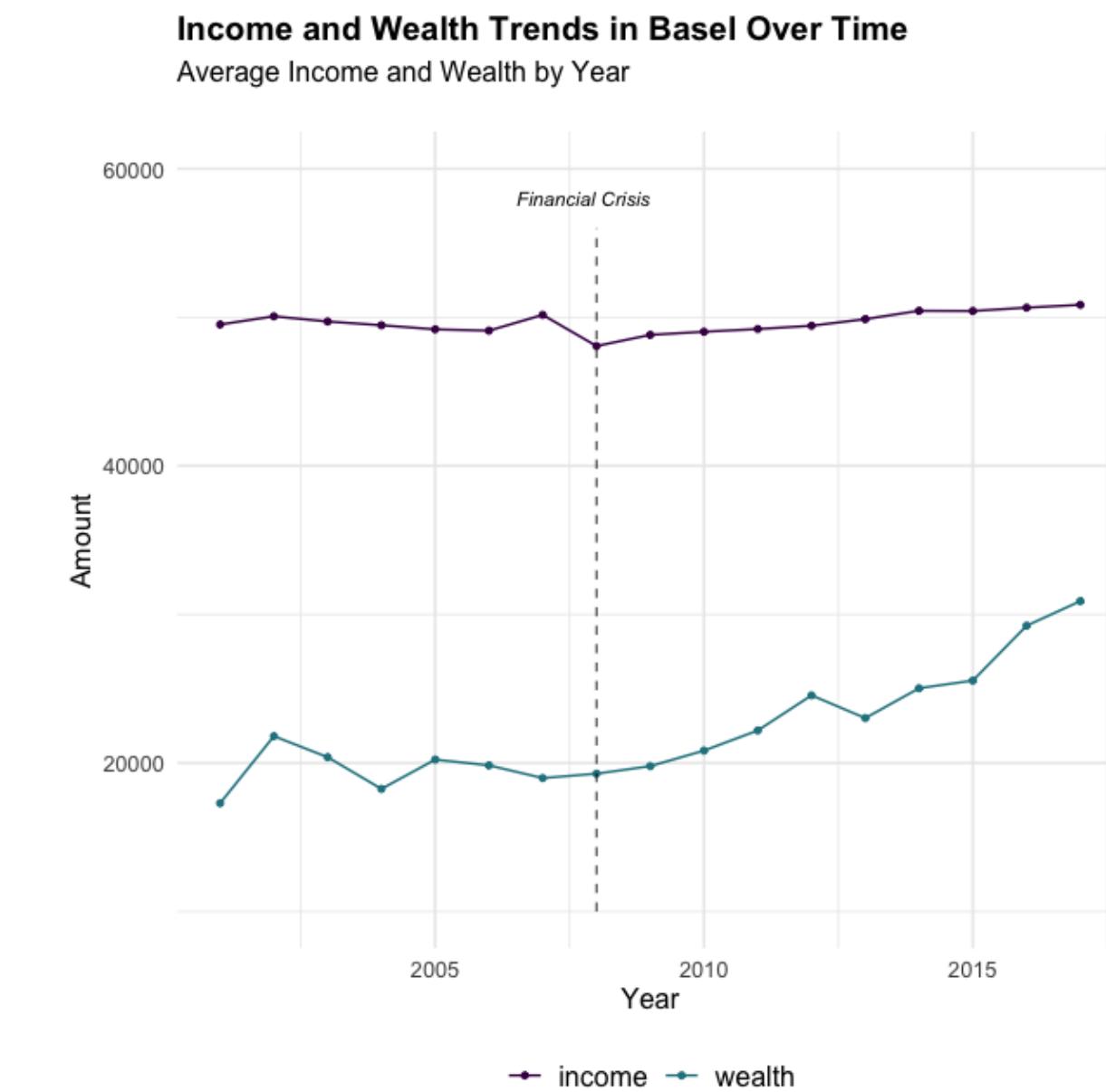
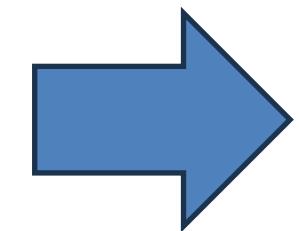
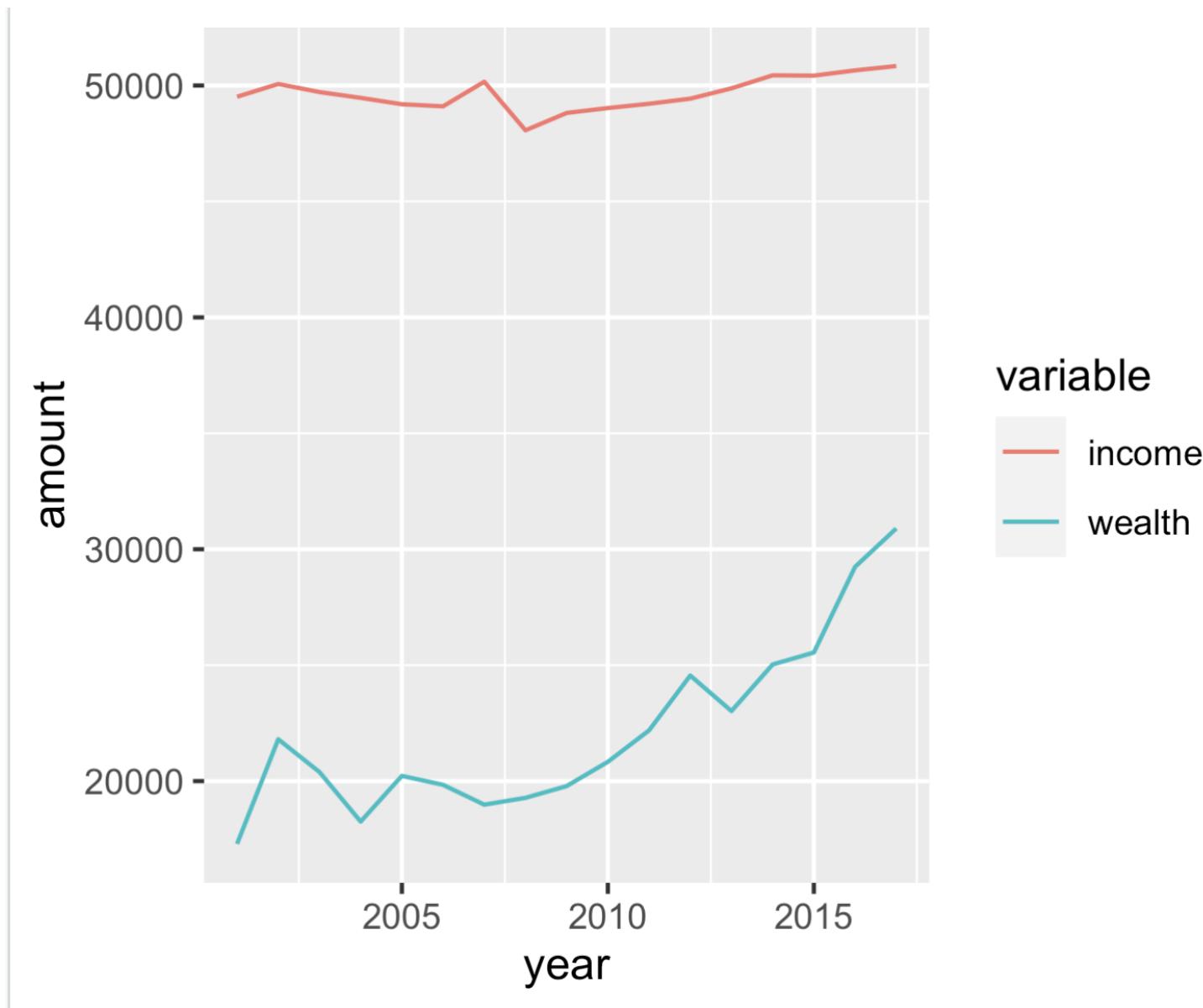
```
... +
# annotation
  labs(title = "Inequality in Basel", subtitle =
"Difference between...", x = "Year",
       y = "Income") +
  annotate('line',
           x = 2009, y = c(40000, 92000)) +
  annotate('text',
           x = 2005, y = 95000,
           label = "Financial crisis", fontface =
"italic")
```



# Practice

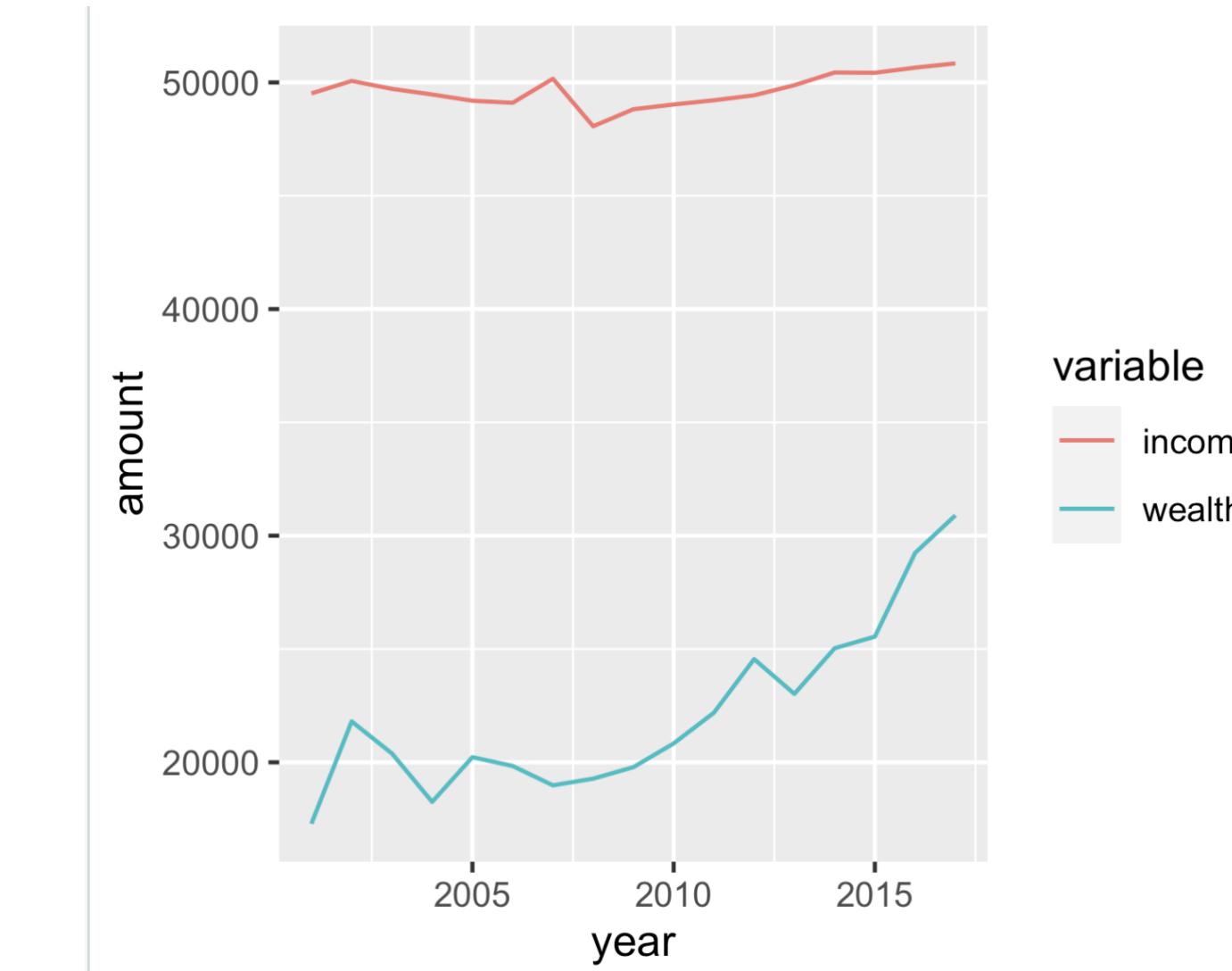


# Let's get from the left plot the right one



Practice:  
<https://ggplot2.tidyverse.org/reference/theme.html>

```
base %>%
  group_by(year) %>%
  summarize(income = mean(income_median),
            wealth = mean(wealth_median)) %>%
  pivot_longer(c(income, wealth),
               names_to = "variable",
               values_to = "amount") %>%
  ggplot(aes(x = year, y = amount, color = variable)) +
  geom_line()
```



```
# add points to the lines. Change line size to 0.5 and point size to 1
```

```
# assign color "#440154" to the data for income and "#26828E" to wealth
```

```
# Scale the y axes from 10000 to 60000
```

```
# add an annotation at x = 2009 to mark the financial crisis
```

```
# add a title, a subtitle and insert meaningful axis titles
```

```
# attach theme_minimal to the plot
```

```
# create margins on each side of the plot
```

```
# make the title of the plot bold and create a margin between the  
subtitle and the plot
```

```
# place the legend at the bottom of the plot, remove the legend title  
and set the font size to 12
```

```
# Add standard errors to the lines
```

# Exam Questions

- If I were to give you a line of code, could you select the correct answer explaining what the code does?