

# Customizing Plots

scales, labels, facet\_wrap()

Emily Malcolm-White

```
library(tidyverse)
```

```
#Import the can_lang dataset
```

```
can_lang <- read.csv("https://raw.githubusercontent.com/ttimbers/canlang/master/inst/extdata/can_lang.csv")
```

## A starting graph: scatterplot of can\_lang

```
can_lang_plot <- ggplot(can_lang, aes(x=most_at_home, y=mother_tongue)) +  
  geom_point() +  
  xlab("Language spoken most at home \n (number of Canadian residents)") +  
  ylab("Mother tongue \n (number of Canadian residents)")
```

Notice anything weird about this plot?

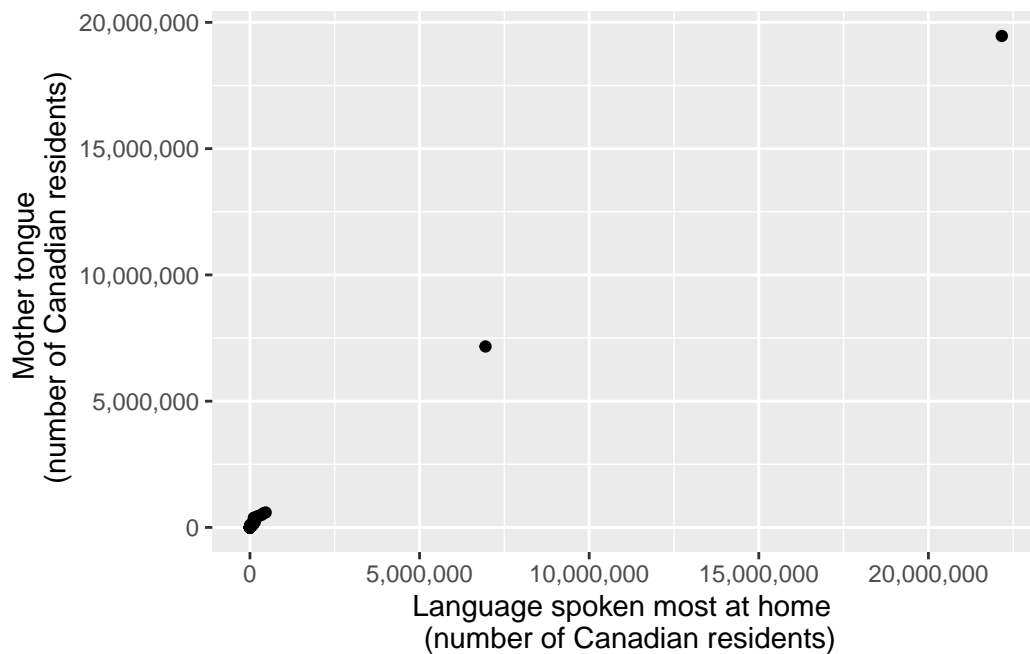
## Axis display format: scales package

```
# Install the package if needed  
library(scales)
```

We want to customize how the continuous x and y axes look, so we need to use the argument `labels=label_comma()` inside a `scale*_continuous()` layer:

```
can_lang_plot +  
  scale_x_continuous(labels = label_comma()) + #<1>  
  scale_y_continuous(labels = label_comma()) #<2>
```

- ① numbers on the x-axis are displayed with commas (and not in scientific notation)
- ② numbers on the y-axis are displayed with commas (and not in scientific notation)



**i** What other formats are available in the `scales` package?

When passing a formatting function inside `scale_*_continuous(labels = ...)` you have options!

Function	Use Case	Example Input	Example Output
<code>label_comma()</code>	Formats numbers with commas	1234567	"1,234,567"
<code>label_dollar()</code>	Formats numbers as dollar currency	99.99	"\$99.99"
<code>label_dollar(prefix = "€")</code>	Formats numbers as euro currency	99.99	"99.99€"
<code>label_percent()</code>	Converts decimals to percent	0.25	"25%"
<code>label_pvalue()</code>	Formats p-values	0.00005	"<0.0001"

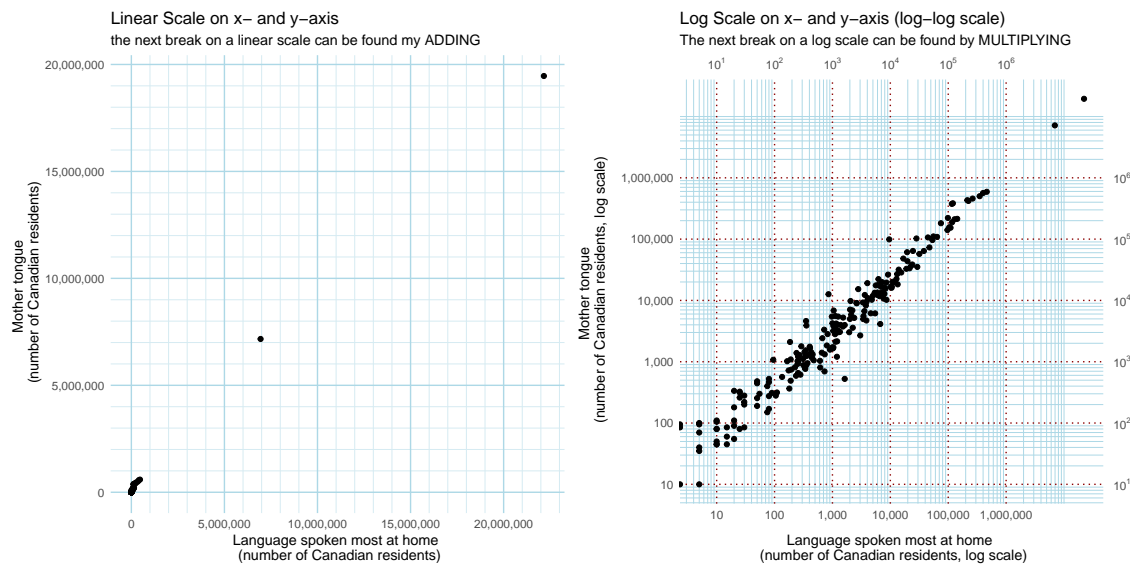
Anything else?

## Logarithmic Axes Transformations

### i Applying a Log Transformation

When you apply a log transformation to an axis (or both axes) in a plot, you convert values using a logarithmic scale instead of a linear scale. This means:

- Instead of evenly spaced values (1, 2, 3, 4, ...), a logarithmic scale spaces values exponentially (1, 10, 100, 1000, ...).
- The distance between ticks represents a multiplicative factor instead of an additive one.

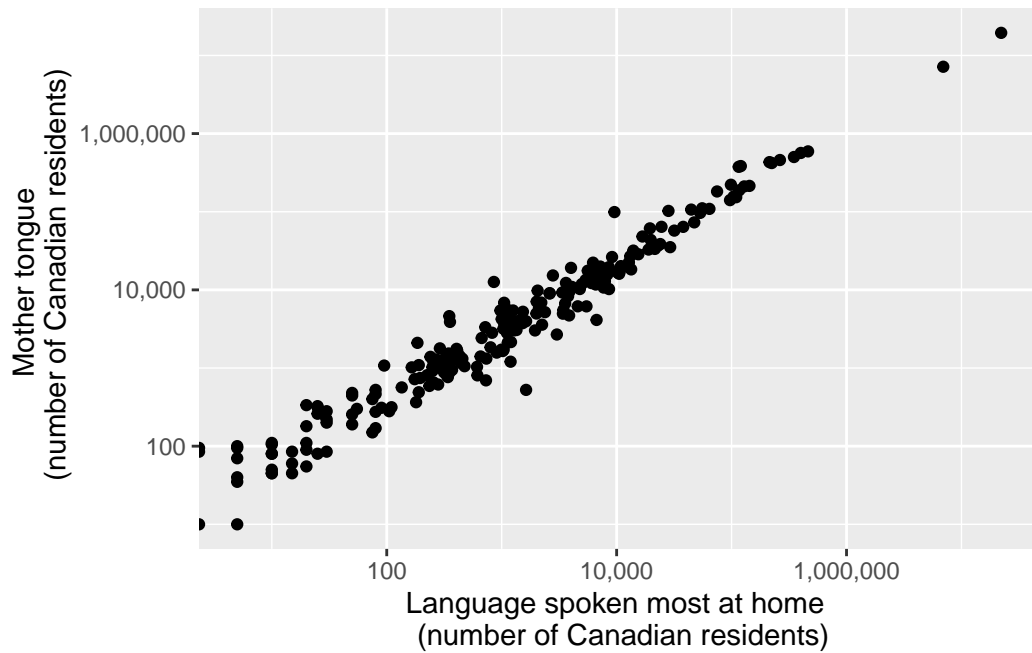


See how much more clearly we can see all the points!

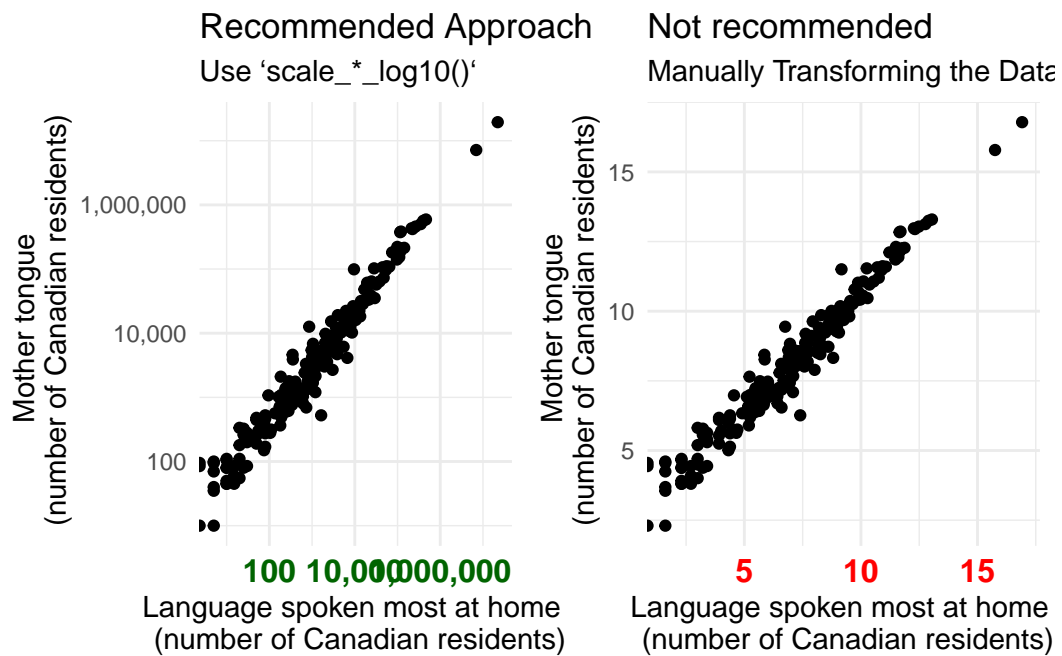
For you to do this yourself, you need to use `scale*_log10()` instead of `scale*_continuous()`:

```
can_lang_plot +  
  scale_x_log10(labels = label_comma()) + #<1>  
  scale_y_log10(labels = label_comma()) #<2>
```

- ① converts x-axis to a log-scale
- ② converts y-axis to a log-scale



💡 Use `scale_*_log10()` instead of `log(variable)`



## Using percents on a log scale

**mutate** to create new columns with percentage of Canadians who speak the language as their mother tongue:

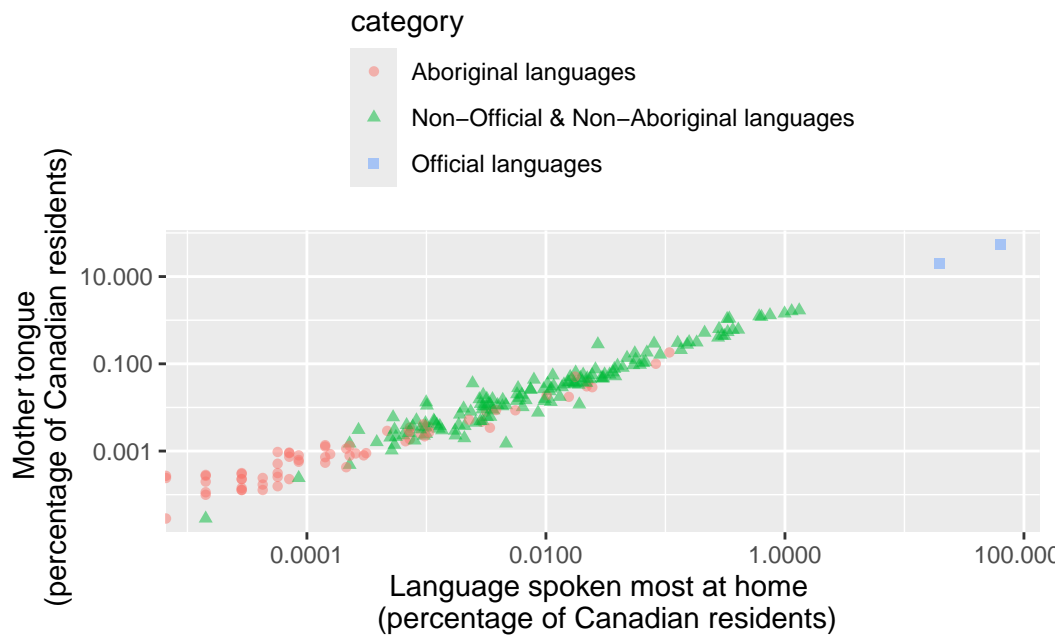
```
can_lang <- can_lang %>%  
  mutate(  
    mother_tongue_percent = (mother_tongue / 35151728) * 100,  
    most_at_home_percent = (most_at_home / 35151728) * 100  
  )
```

## Scatterplot with Percents and Colors

Create a scatterplot with `most_at_home_percent` and `mother_tongue_percent`. Vary the color and shape of the points depending on the category of language. You may need to adjust the position of the legend:

```
can_lang_percent_plot <- ggplot(can_lang, aes(x = most_at_home_percent, #<1>  
                                              y = mother_tongue_percent )) + #<2>  
  geom_point(aes(color = category, shape=category), alpha=0.5) + #<3>  
  xlab("Language spoken most at home \n (percentage of Canadian residents)") +  
  ylab("Mother tongue \n (percentage of Canadian residents)") +  
  theme(legend.position = "top", legend.direction = "vertical") + #<4>  
  scale_x_log10(labels = comma) +  
  scale_y_log10(labels = comma)  
  
can_lang_percent_plot
```

- ① Use `most_at_home_percent` as the x-axis
- ② Use `mother_tongue_percent` as the y-axis
- ③ vary the shape and the color based on the category of language. Note this is included in the aesthetics of the points. It also would have been okay to put these directly inside the global aesthetics (`ggplot(aes(...))`) so that these characteristics apply to any layers.
- ④ Adjusts the position of the legend

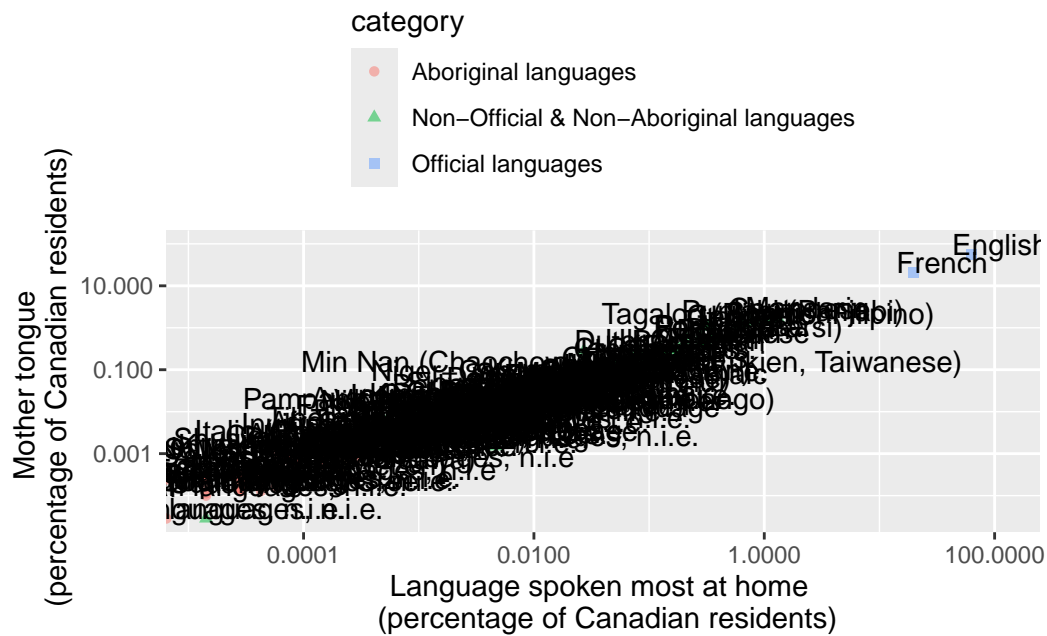


## Labels

Adding text to a plot is one of the most common forms of annotation. Most plots will not benefit from adding text to every single observation on the plot, but labeling outliers and other important points is very useful.

Add a label for each language in this dataset using `geom_text(aes(label = language))`:

```
can_lang_percent_plot +
  geom_text(aes(label=language),
            nudge_x = 0.25,
            nudge_y=0.25)
```



Yikes! This is way too much going on in one plot. A few options to try when this happens:

- Decrease the font size of the labels (using the `size=` argument inside `geom_text`).
- Use the `ggrepel` package to spread out the labels a bit more
- Pick out only a subset of the points to label

## Using ggrepel

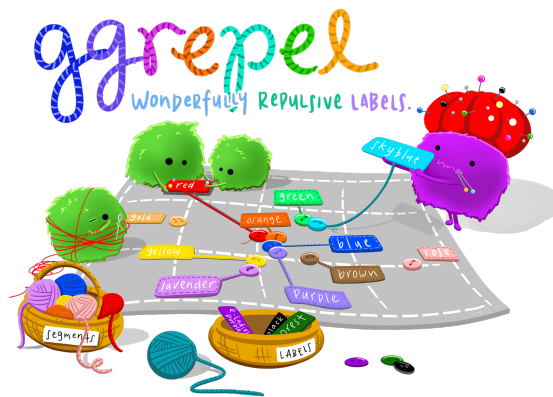
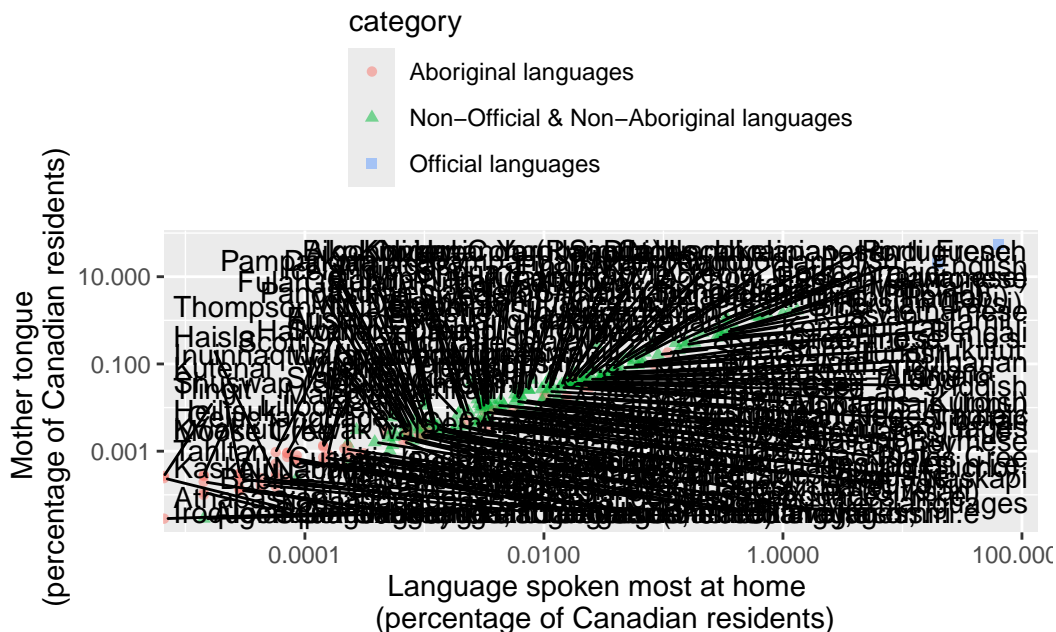


Figure 1: Artwork by @allisonhorst

```
library(ggrepel)
```

```
can_lang_percent_plot +  
  geom_text_repel(aes(label=language), max.overlaps = Inf)
```



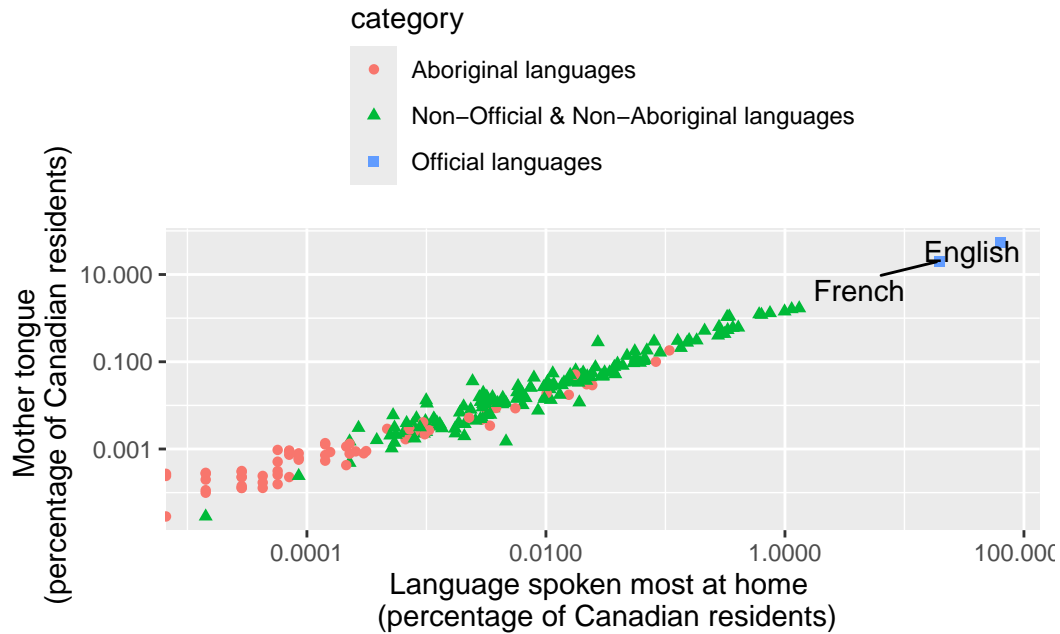
## Subset the labels

Create a new column for the labels. Use `case_when` (or `ifelse`) to only use the official language names and not to put a label for other language categories.

```
can_lang <- can_lang %>%  
  mutate(official_languages = case_when(category == "Official languages" ~ language, TRUE ~ NA))  
  
# We need to redo the baseplot with the new can_lang dataset with the new official_languages  
can_lang_percent_plot <- ggplot(can_lang, aes(x = most_at_home_percent, y = mother_tongue_percent)) +  
  geom_point(aes(color = category, shape=category)) +  
  xlab("Language spoken most at home \n (percentage of Canadian residents)") +  
  ylab("Mother tongue \n (percentage of Canadian residents)") +  
  theme(legend.position = "top", legend.direction = "vertical") +  
  scale_x_log10(labels = comma) +  
  scale_y_log10(labels = comma)
```



```
can_lang_percent_plot +
  geom_text_repel(aes(label=official_languages, min.segment.length=0, box.padding=1))
```



## Facet Wrap

`facet_wrap()` is a function in the `ggplot2` package that allows you to create a multi-panel plot showing a similar plot over different subsets of the data, usually different values of a categorical variable.

Create separate side-by-side plots for each different category of language.

```
can_lang_percent_plot +
  facet_wrap(~category)
```

