Data visualization with ggplot2

Visualizing data in R with the ggplot2 package

Authors: **Mateusz Kuzak**, **Diana Marek**, **Hedi Peterson**, **Dmytro Fishman**

Disclaimer We will be using the functions in the ggplot2 package. There are basic plotting capabilities in basic R, but ggplot2 adds more powerful plotting capabilities.

> **Learning Objectives**
>
> - Visualize some of the global temperature data from Figshare temperature.csv
> - Understand how to plot these data using R ggplot2 package. For more details on using ggplot2 see official documentation.
> - Building step by step complex plots with the ggplot2 package, see handy cheatsheet

Load required packages

```
# plotting package
library(ggplot2)
# piping / chaining
library(magrittr)
# modern dataframe manipulations
library(dplyr)
```

```
#>
#> Attaching package: 'dplyr'
#>
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#>
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
```

```
temperature_raw <- read.csv('data/temperature.csv')
```

Data cleaning and preparing for plotting

Let's look at the summary

```
summary(temperature_raw)
```

There are few things we need to clean in the dataset.

There are missing values for City and Country in some records. Let's remove those.

```
temperature_complete <- temperature_raw %>%
                    filter(!is.na(City)) %>%
                    filter(!is.na(Country))
```

We saw in summary, there were NA's in AverageTemperatureFahr and AverageTemperatureUncertaintyFahr Let's remove rows with missing values in those fields. In fact, let's combine this with removing empty City and Country, so we have one command and don't make lots of intermediate variable names. This is where piping becomes really handy!

```
temperature_complete <- temperature_raw %>%
                    filter(!is.na(City)) %>%
                    filter(!is.na(Country)) %>%
                    filter(!is.na(AverageTemperatureFahr)) %>%
                    filter(!is.na(AverageTemperatureUncertaintyFahr))
```

`day` field has no other values besides `1`, let's remove it from our data.

```
# get rid of `day` column
temperature_complete <- select(temperature_complete, -(day))
head(temperature_complete)
```

We already saw how changing Fahrenheit to Celsius helps to understand the data better, let's convert AverageTemperatureFahr and AverageTemperatureUncertaintyFahr into Celsius.

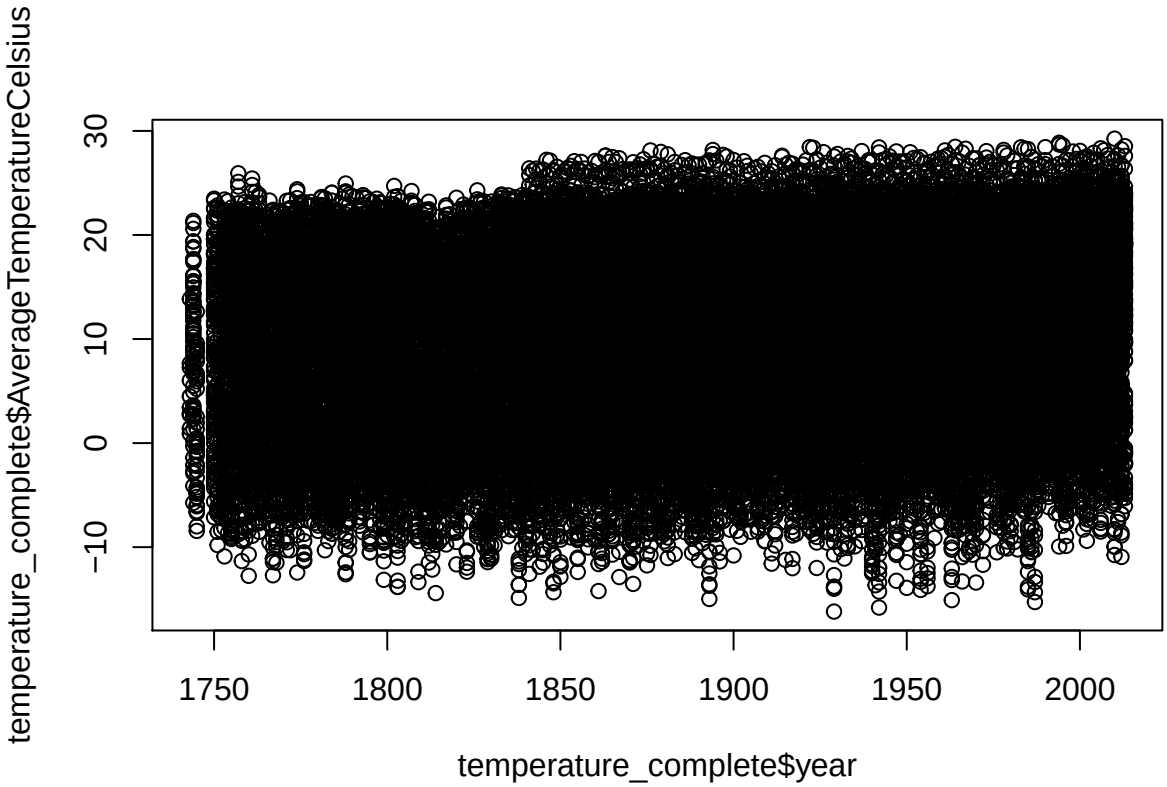
```
# Use mutate to convert Fahrenheit to Celsius and store values in
# separate columns
temperature_complete <- temperature_complete %>%
  mutate(AverageTemperatureCelsius = (AverageTemperatureFahr-32)*(5/9)) %>%
  mutate(AverageTemperatureUncertaintyCelsius = (AverageTemperatureUncertaintyFahr-32)*(5/9))
```

Now we can actually get rid of unnecessary columns with Fahrenheit information by assigning `NULL` value to these columns.

```
# Remove AverageTemperatureFahr and AverageTemperatureUncertaintyFahr
# by assigning them with NULL
temperature_complete$AverageTemperatureFahr <- NULL
temperature_complete$AverageTemperatureUncertaintyFahr <- NULL
head(temperature_complete)
```

Make simple scatter plot of `AverageTemperatureCelsius` as a function of `year`, using basic R plotting capabilities.

```
plot(x = temperature_complete$year, y = temperature_complete$AverageTemperatureCelsius)
```



Plotting with ggplot2

We will make the same plot using the `ggplot2` package.

`ggplot2` is a plotting package that makes it simple to create complex plots from data in a dataframe. It uses default settings, which help creating publication quality plots with a minimal amount of settings and tweaking.

ggplot graphics are built step by step by adding new elements.

To build a ggplot we need to:

- bind the plot to a specific data frame using the `data` argument

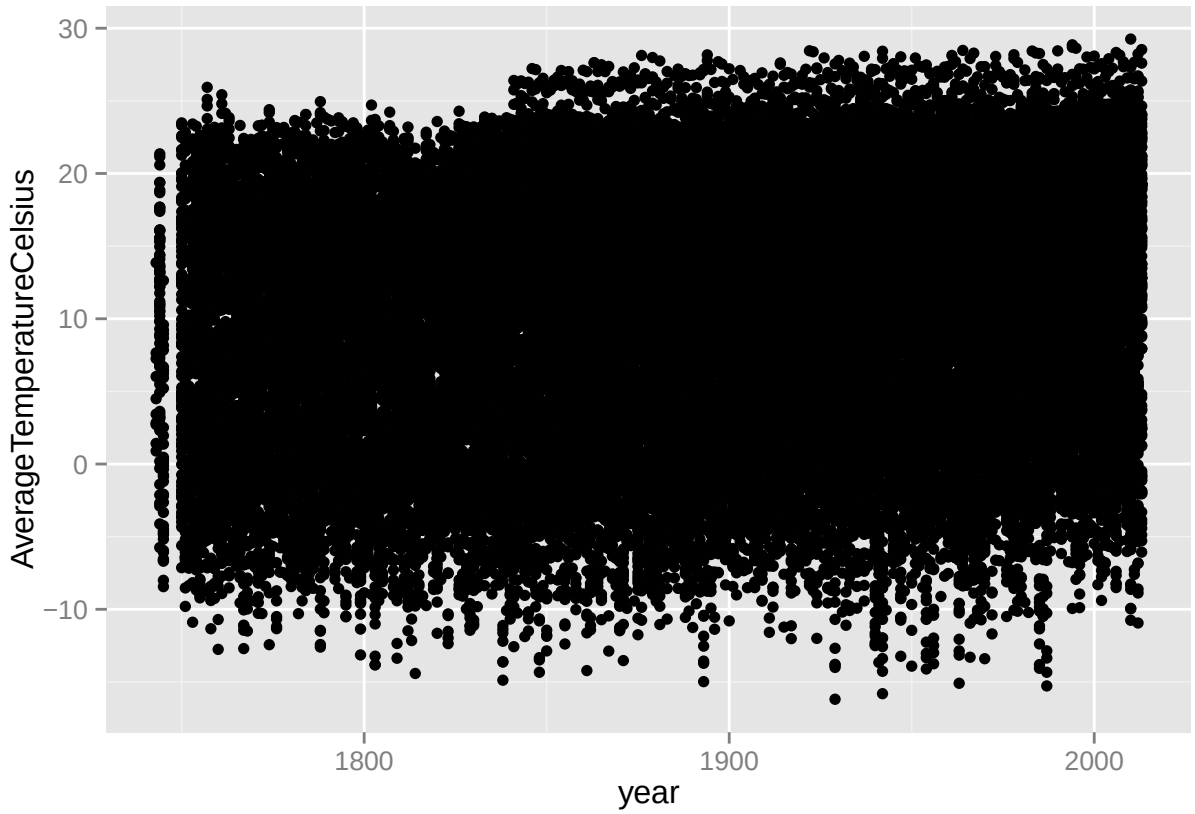
```
ggplot(data = temperature_complete)
```

- define aesthetics (`aes`), that maps variables in the data to axes on the plot or to plotting size, shape color, etc.,

```
ggplot(data = temperature_complete, aes(x = year, y = AverageTemperatureCelsius))
```

- add `geoms` – graphical representation of the data in the plot (points, lines, bars). To add a geom to the plot use `+` operator:

```
ggplot(data = temperature_complete, aes(x = year, y = AverageTemperatureCelsius)) +
  geom_point()
```



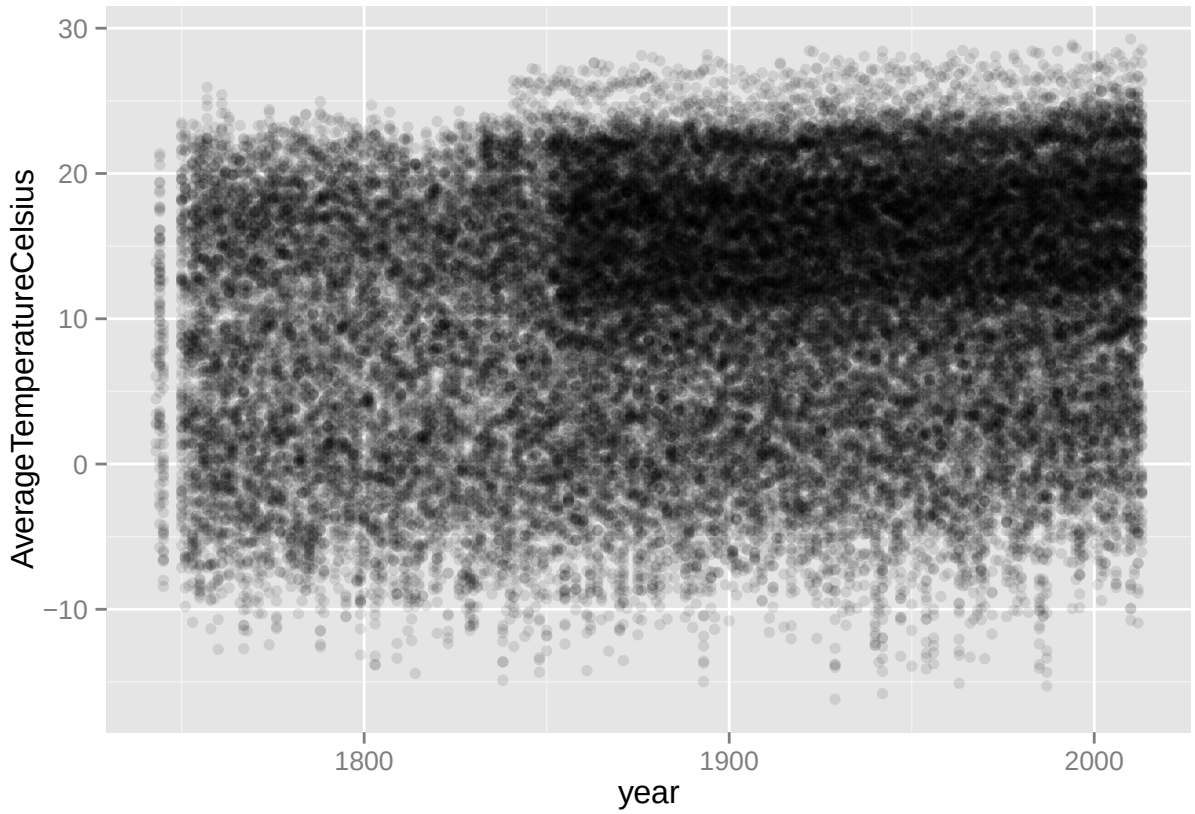
Notes:

- Anything you put in the `ggplot()` function can be seen by any geom layers that you add. i.e. these are universal plot settings
- This includes the x and y axis you set up in `aes()`

Modifying plots

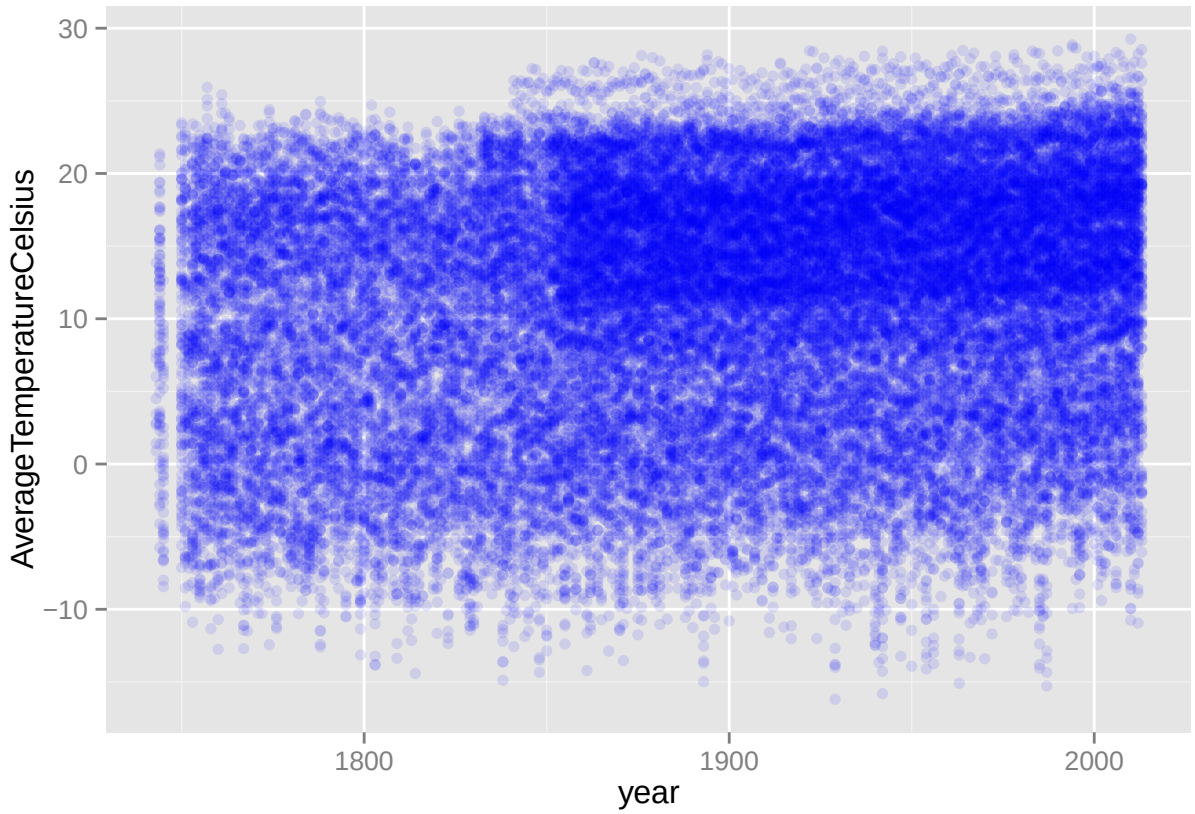
- adding transparency (alpha)

```
ggplot(data = temperature_complete, aes(x = year, y = AverageTemperatureCelsius)) +
  geom_point(alpha = 0.1)
```



- adding colors

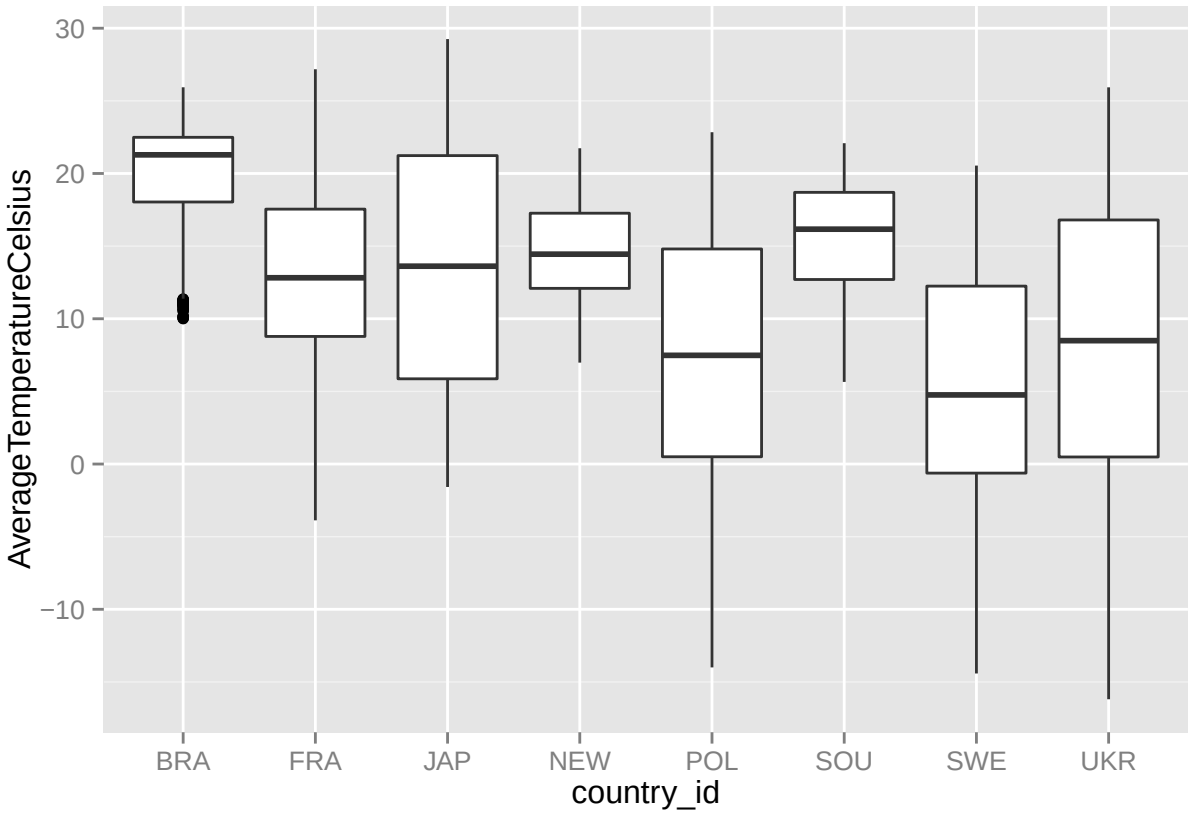
```
ggplot(data = temperature_complete, aes(x = year, y = AverageTemperatureCelsius)) +
  geom_point(alpha = 0.1, color = "blue")
```



Boxplot

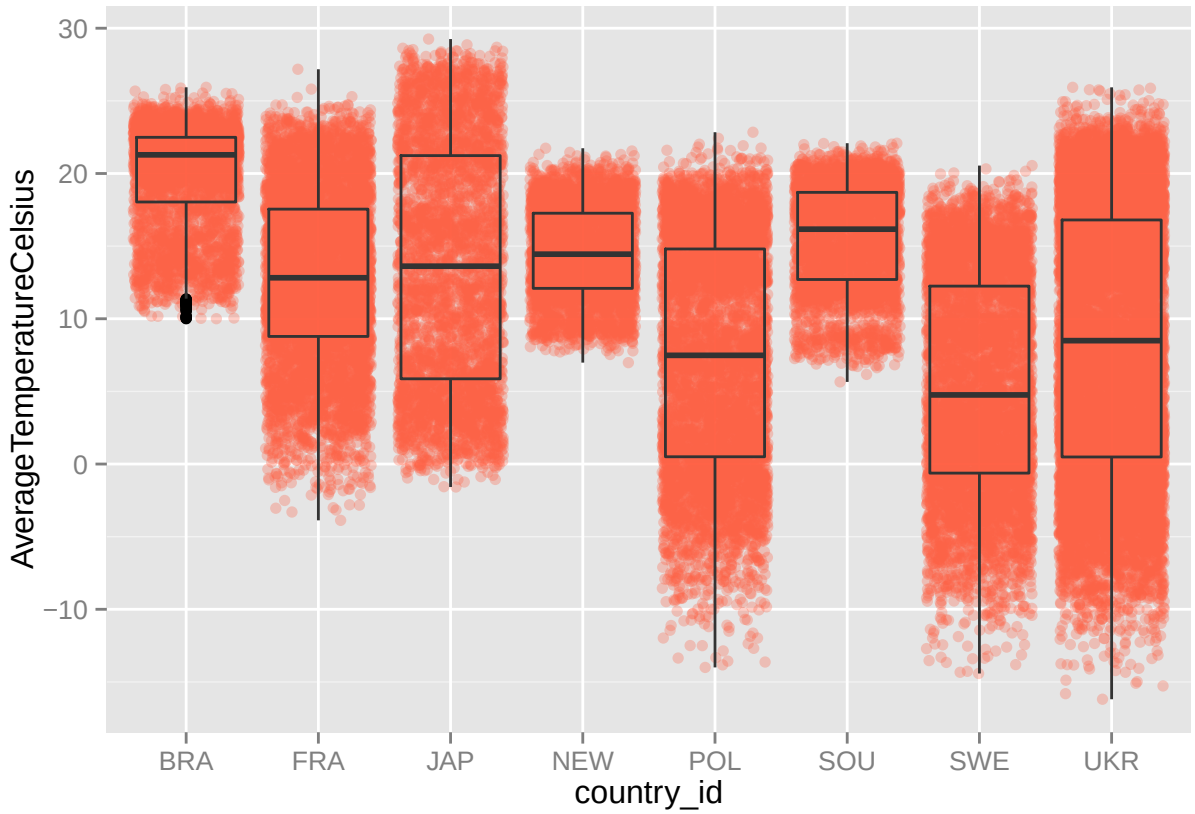
Visualising the distribution of weight within each species.

```
ggplot(data = temperature_complete, aes(x = country_id,  y = AverageTemperatureCelsius)) +
      geom_boxplot()
```



By adding points to boxplot, we can see particular measurements and the abundance of measurements.

```
ggplot(data = temperature_complete, aes(x = country_id,  y = AverageTemperatureCelsius)) +
                  geom_jitter(alpha = 0.3, color = "tomato") +
                  geom_boxplot(alpha = 0)
```



Notice how the boxplot layer is on top of the jitter layer? Play around with the order of geoms and adjust transparency to see how to build up your plot in layers.

```
## For example
ggplot(data = temperature_complete, aes(x = country_id,  y = AverageTemperatureCelsius)) +
                    geom_boxplot(alpha = 0.9, size = 0.9)  +
                    geom_jitter(alpha = 0.1, color = "tomato")
```

Challenges

Boxplots are useful summaries, but hide the *shape* of the distribution. For example, if there is a bimodal distribution, this would not be observed with a boxplot. An alternative is a violin plot (sometimes known as a beanplot), where the shape (of the density of points) is drawn.

- Replace the box plot with a violin plot; see `geom_violin()`

```
## Answer
ggplot(data = temperature_complete, aes(x = country_id,  y = AverageTemperatureCelsius)) +
                    geom_jitter(alpha = 0.3, color = "tomato") +
                    geom_violin(alpha = 0)
```

- Create boxplot for `AverageTemperatureUncertaintyCelsius`, which is a 95% confidence interval from average temperature, it is always positive.

```
## Answer
ggplot(data = temperature_complete, aes(x = country_id,  y = AverageTemperatureUncertaintyCelsius)) +
                  geom_jitter(alpha = 0.3, color = "tomato") +
                  geom_boxplot(alpha = 0)
```

In many types of data, it is important to consider the *scale* of the observations. For example, it may be worth changing the scale of the axis to better distribute the observations in the space of the plot. Changing the scale of the axes is done similarly to adding/modifying other components (i.e., by incrementally adding commands).

- Represent AverageTemperatureUncertaintyCelsius on the log10 scale; see `scale_y_log10()`

```
## Answer
ggplot(data = temperature_complete, aes(x = country_id,  y = AverageTemperatureUncertaintyCelsius)) +
                  geom_jitter(alpha = 0.3, color = "tomato") +
                  geom_boxplot(alpha = 0) +
                  scale_y_log10()
```

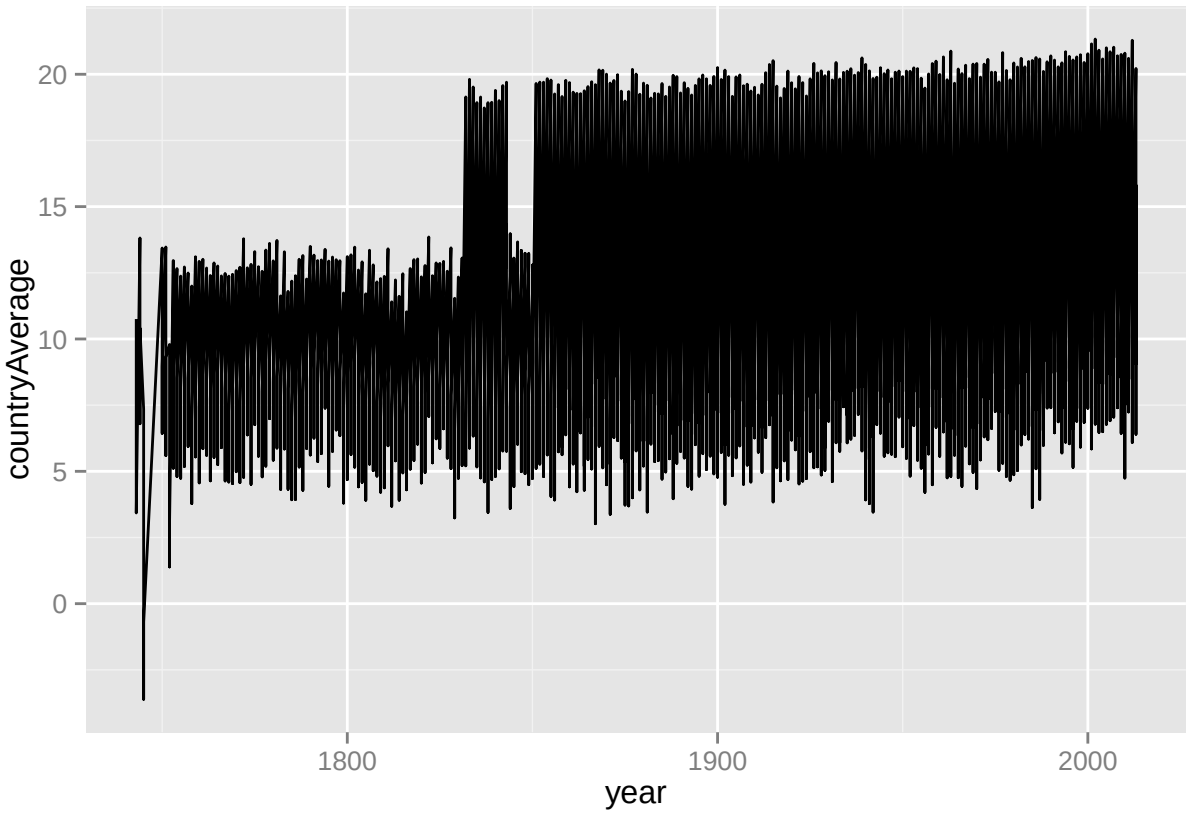
Plotting time series data

Let's calculate average temperature per year per each country. To do that we need to group data first by year and country and summarise it by computing mean from every group.

```
yearly_country_temp <- temperature_complete %>%
                        group_by(year, Country) %>%
                        summarise(countryAverage = mean(AverageTemperatureCelsius))
```

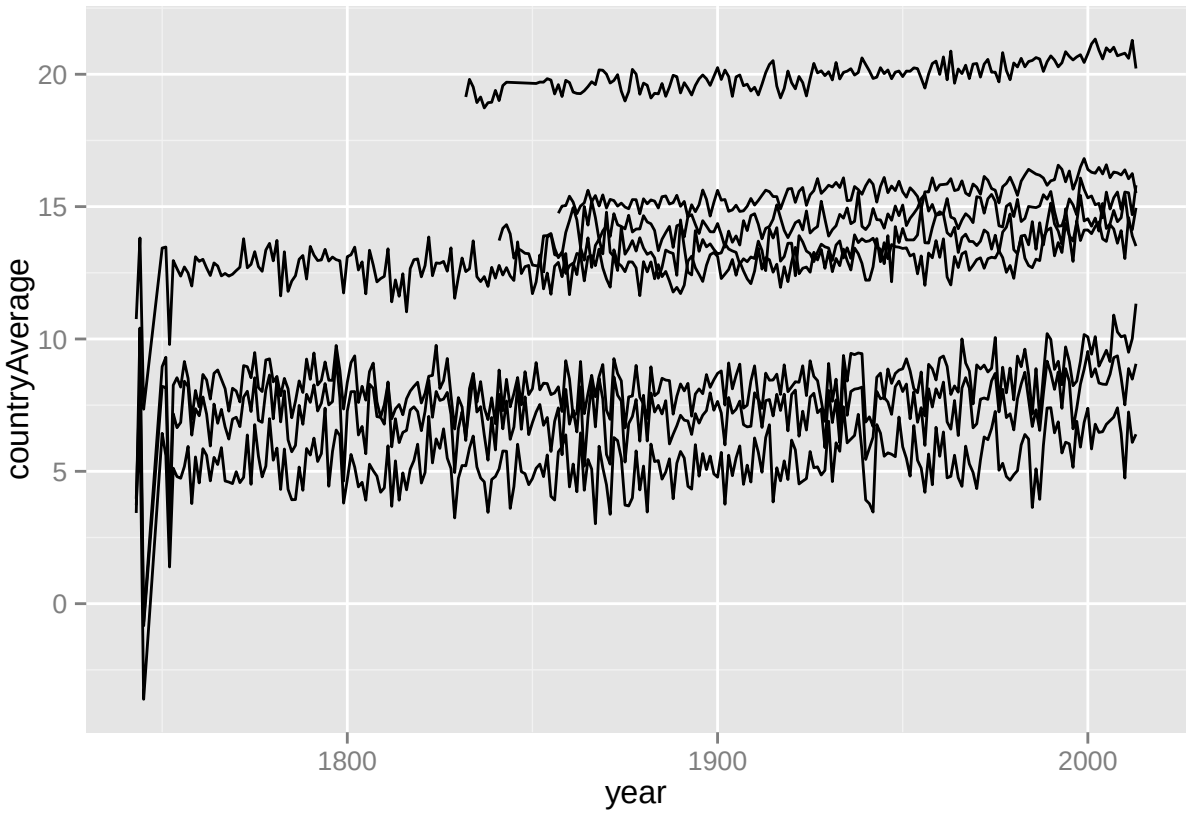
Timelapse data can be visualised as a line plot with years on x axis and temperature on y axis.

```
ggplot(data = yearly_country_temp, aes(x = year, y = countryAverage)) +
                  geom_line()
```



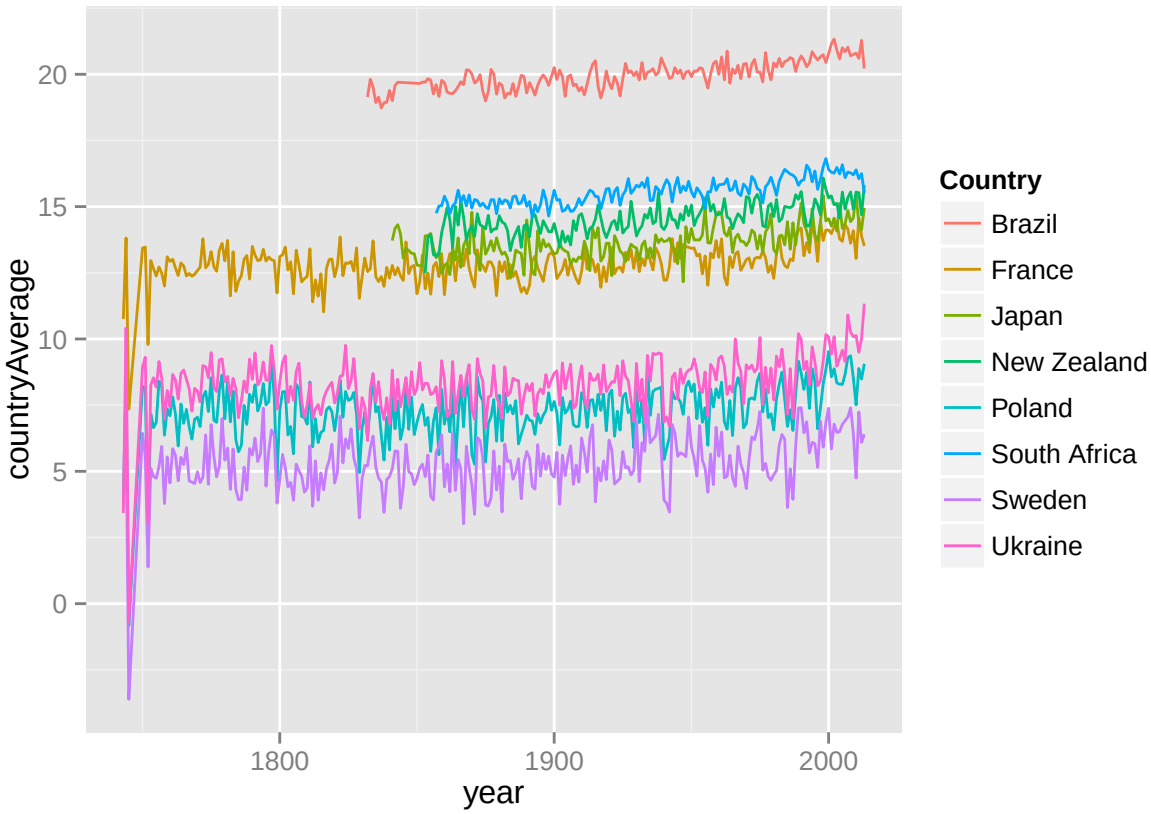
Unfortunately this does not work, because we plot data for all the countries together. We need to tell ggplot to split graphed data by `Country`

```
ggplot(data = yearly_country_temp, aes(x = year, y = countryAverage, group = Country)) +
  geom_line()
```



We will be able to distinguish countries even better in the plot if we add colors.

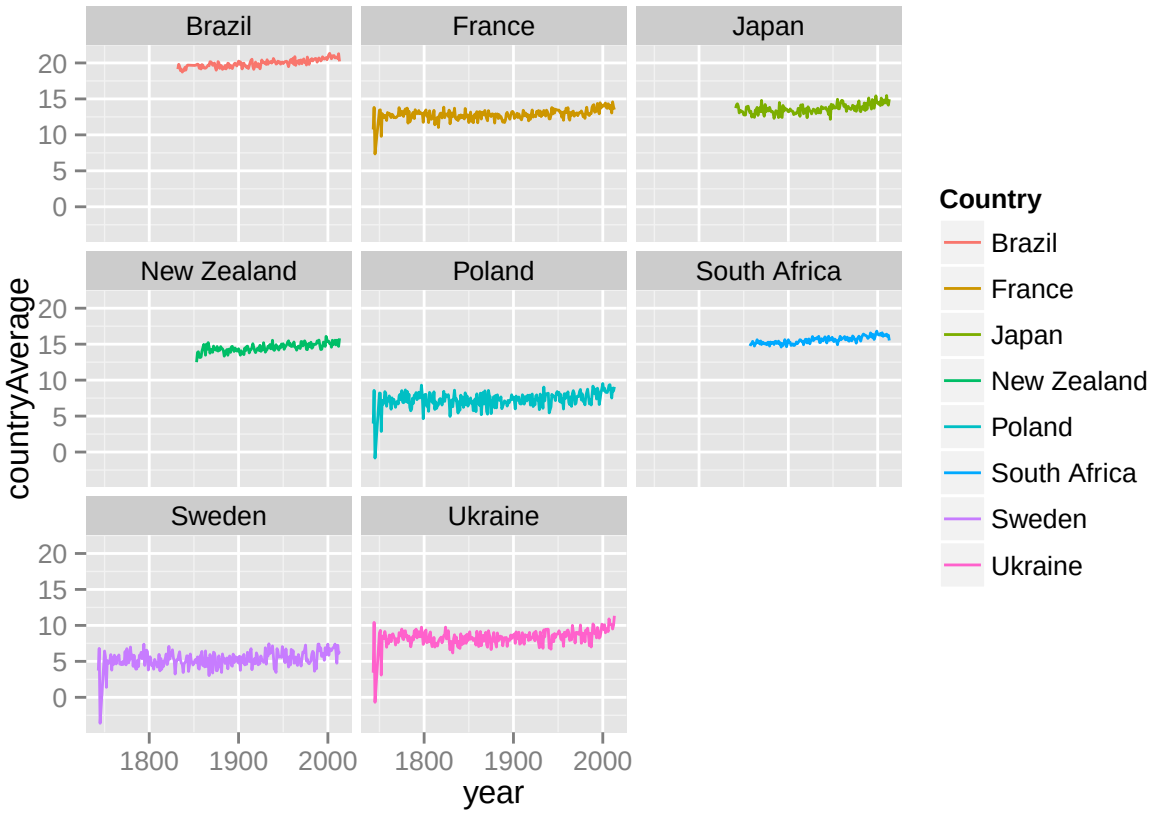
```
ggplot(data = yearly_country_temp, aes(x = year, y = countryAverage, group = Country, color = Country))
  geom_line()
```



Faceting

ggplot has a special technique called *faceting* that allows to split one plot into multiple plots based on some factor. We will use it to plot one time series for each species separately.

```
ggplot(data = yearly_country_temp, aes(x = year, y = countryAverage, color = Country)) +
  geom_line() +
  facet_wrap(~Country)
```



Now we would like to split line in each plot by city of each country. To do that we need to make measurements in dataframe grouped by country and city.

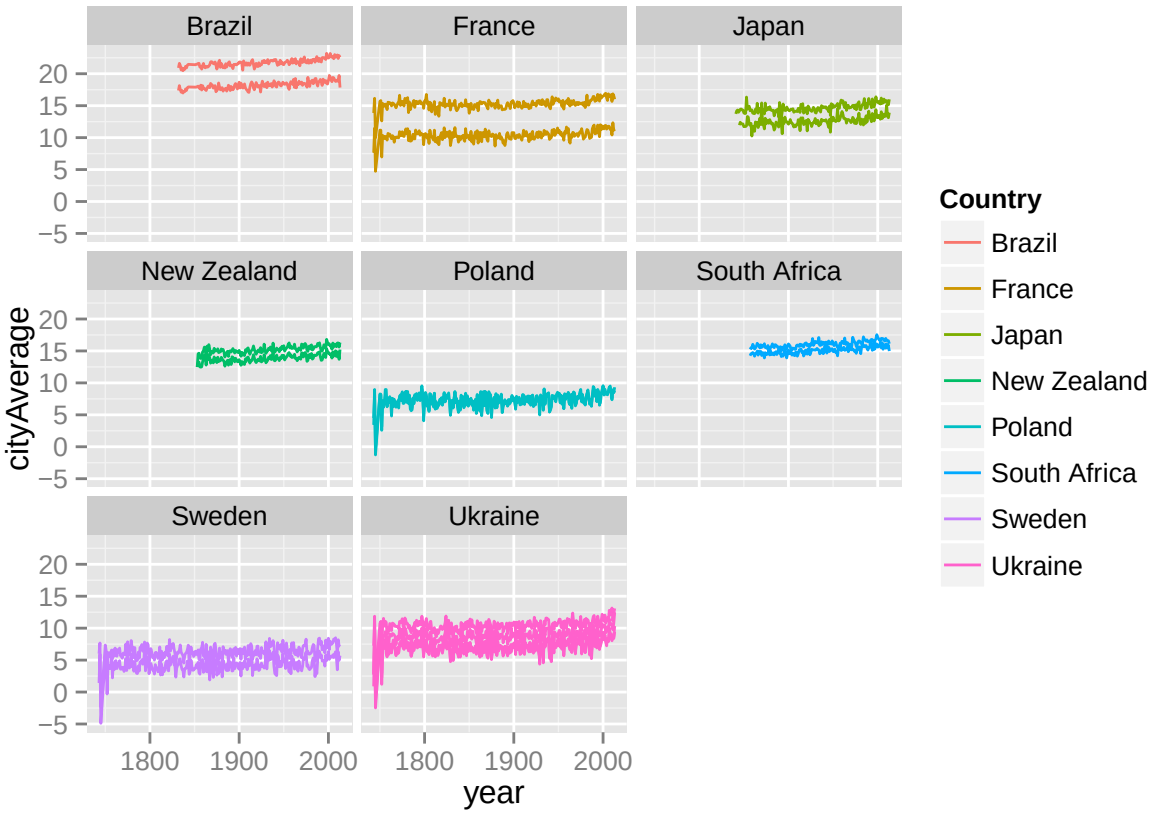
Challenges:

- group by year, Country, City

```
yearly_country_city_temp <- temperature_complete %>%
                          group_by(year, Country, City) %>%
                          summarise(cityAverage = mean(AverageTemperatureCelsius))
```

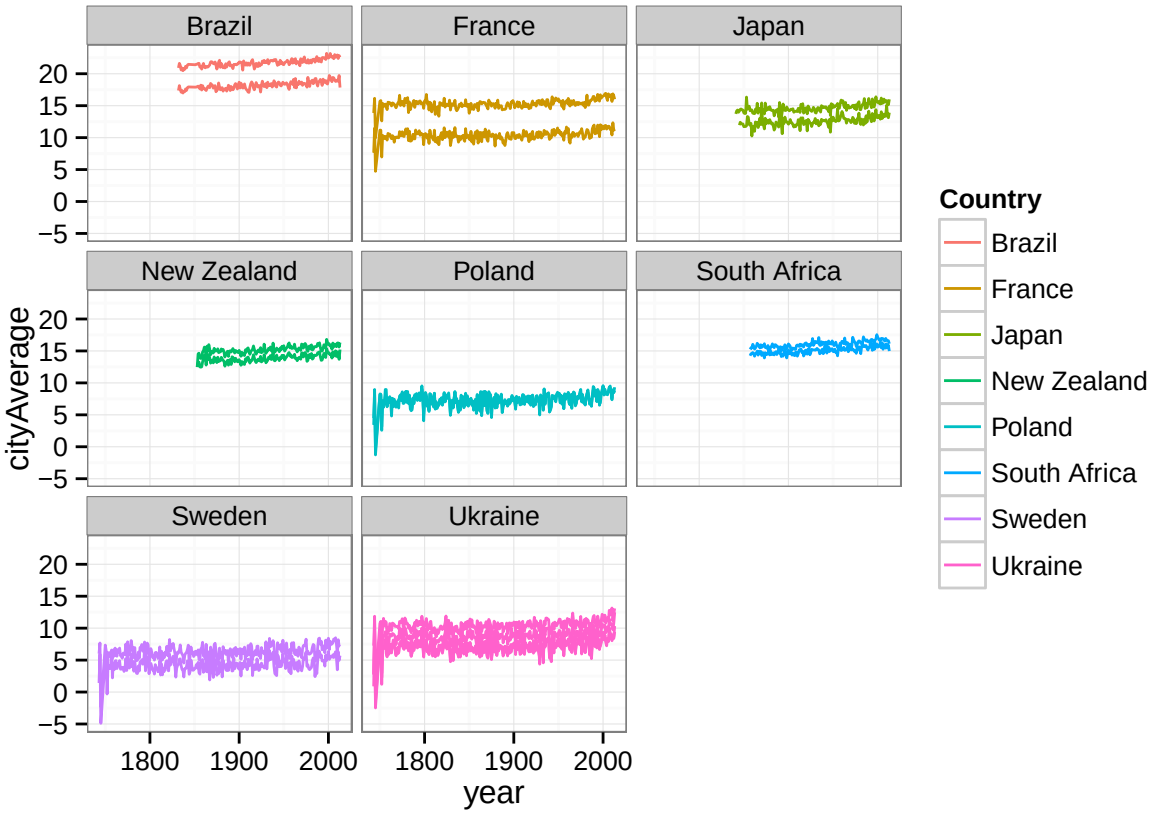
- make the faceted plot splitting further by City (within single plot)

```
ggplot(data = yearly_country_city_temp, aes(x = year, y = cityAverage, color = Country, group = City)) +
  geom_line() +
  facet_wrap(~ Country)
```



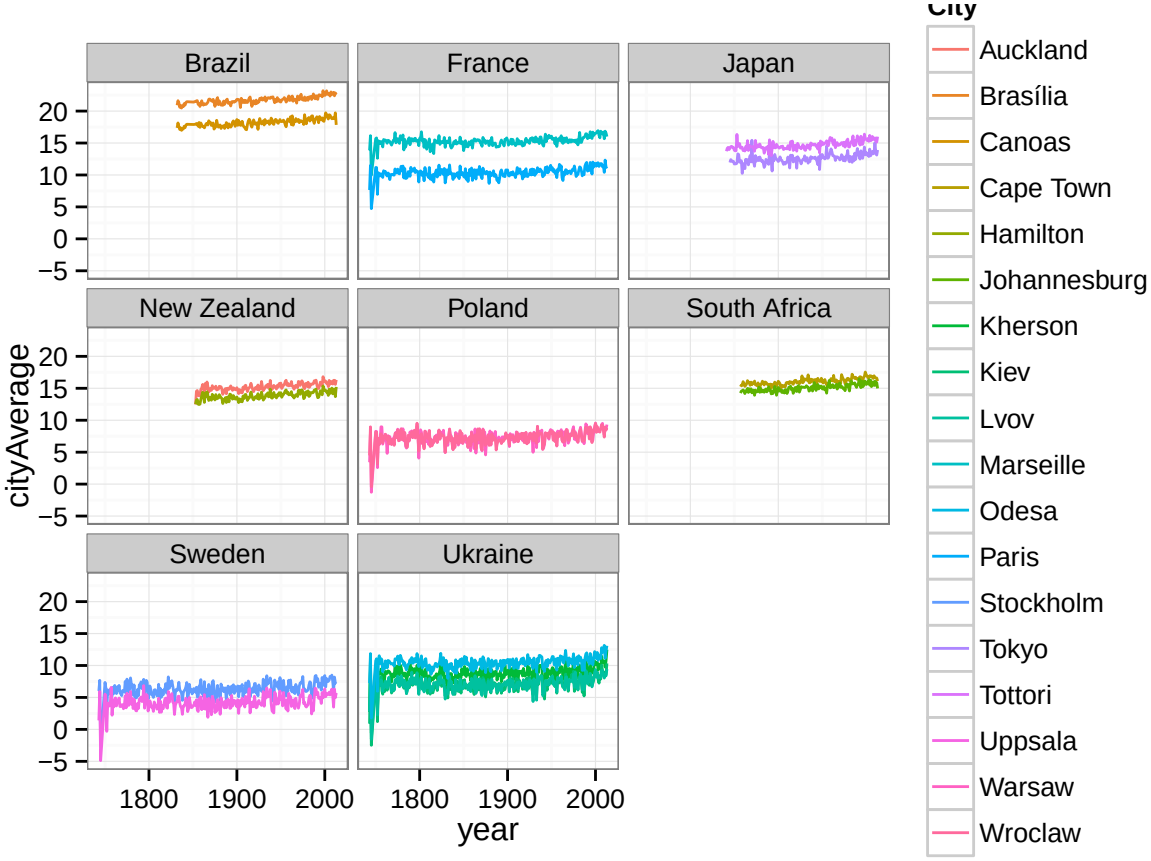
Usually plots with white background looks more readable in the publications. We can set the background to white using the function `theme_bw()`.

```
ggplot(data = yearly_country_city_temp, aes(x = year, y = cityAverage, color = Country, group = City)) +
  geom_line() +
  facet_wrap(~ Country) +
  theme_bw()
```



We can improve the plot by coloring by city instead of country (countries are already in separate plots, so we don't need to distinguish them better)

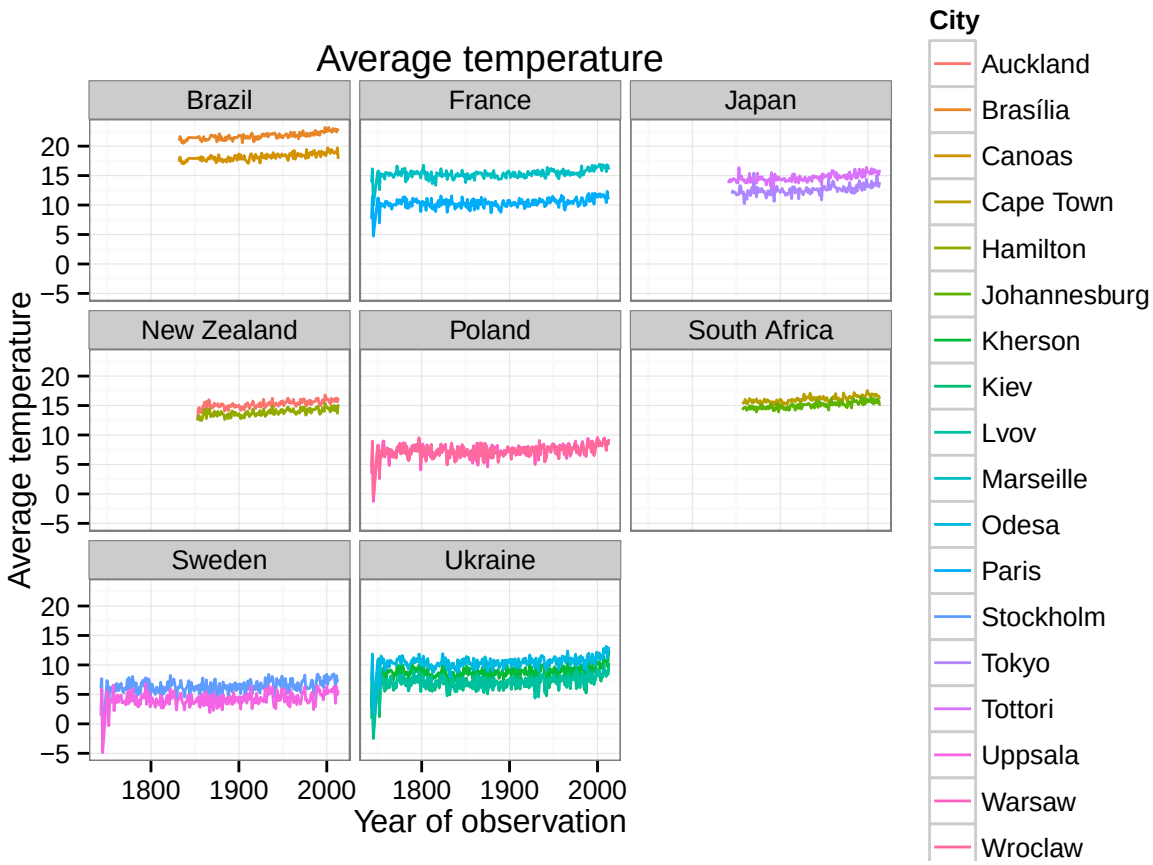
```
ggplot(data = yearly_country_city_temp, aes(x = year, y = cityAverage, color = City, group = City)) +
  geom_line() +
  facet_wrap(~ Country) +
  theme_bw()
```



So far our result looks quite good, but it is yet far from being publishable. What are other ways one can improve it? Take a look at the ggplot2 cheat sheet (https://www.rstudio.com/wp-content/uploads/2015/08/ggplot2-cheatsheet.pdf), and write down at least three more ideas (can leave them as comments to Etherpad).

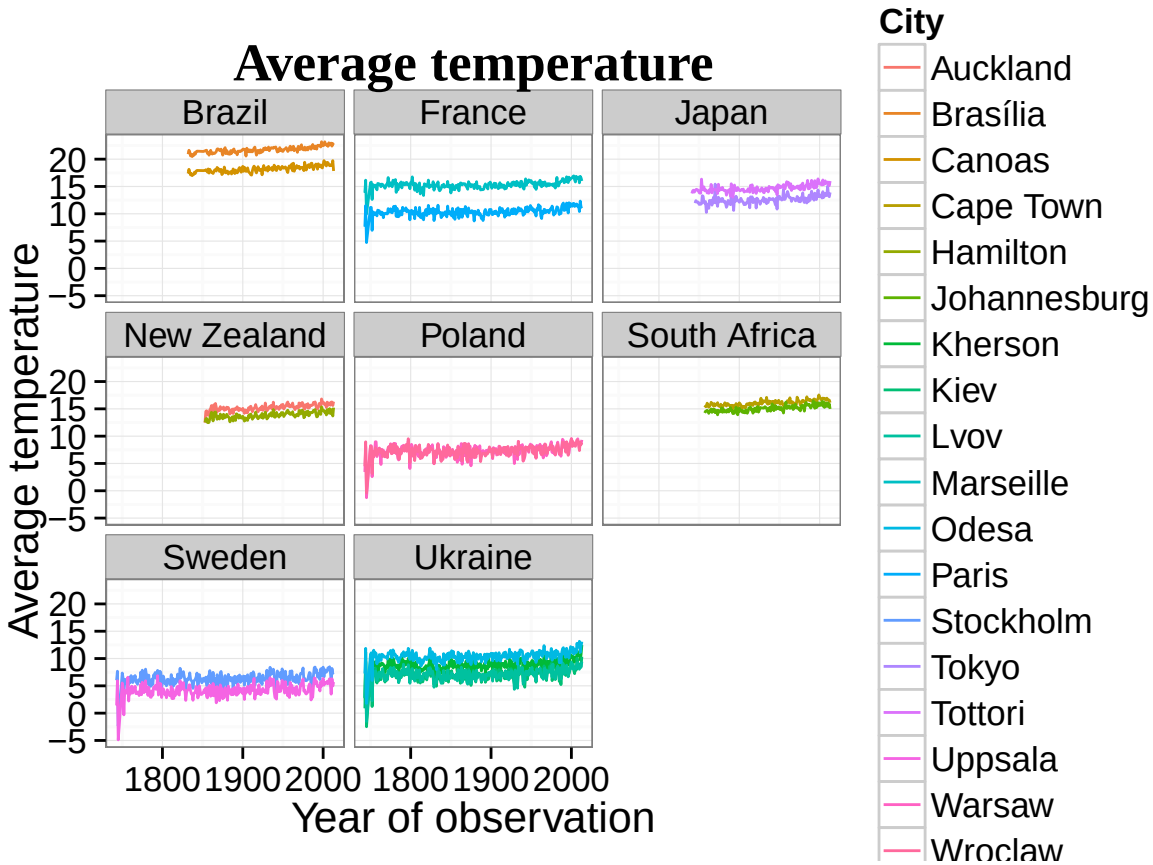
Now, let's change names of axes to something more informative than 'year' and 'cityAverage' and add title to this figure:

```
ggplot(data = yearly_country_city_temp, aes(x = year, y = cityAverage, color = City, group = City)) +
  geom_line() +
  facet_wrap(~ Country) +
  labs(title = 'Average temperature',
       x = 'Year of observation',
       y = 'Average temperature') +
  theme_bw()
```



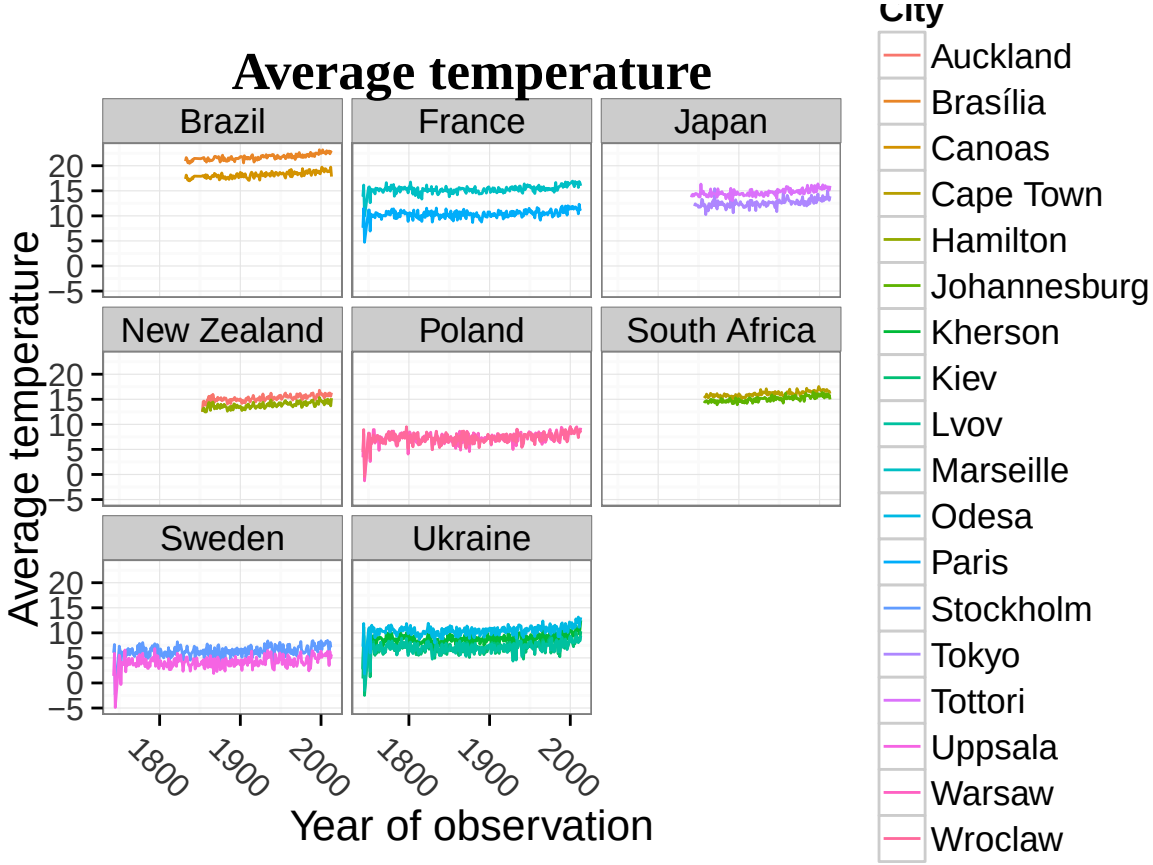
Now, thanks to our efforts, axes have much more informative names, yet quite small so it could be hard to read them. Let's change their size (and font just in sake of fun):

```
ggplot(data = yearly_country_city_temp, aes(x = year, y = cityAverage, color = City, group = City)) +
  geom_line() +
  facet_wrap(~ Country) +
  labs(title = 'Average temperature',
       x = 'Year of observation',
       y = 'Average temperature') +
  theme_bw(base_size = 16) +
  theme(plot.title=element_text(family="Times", face="bold", size=20))
```



After our manipulations we notice that the values on the x-axis are still not properly readable. Let's change the orientation of the labels and adjust them vertically and horizontally so they don't overlap. You can use a 90 degree angle, or experiment to find the appropriate angle for diagonally oriented labels.

```
my_perfect_plot <- ggplot(data = yearly_country_city_temp, aes(x = year, y = cityAverage, color = City,
                      geom_line() +
                      facet_wrap(~ Country) +
                      labs(title = 'Average temperature',
                        x = 'Year of observation',
                        y = 'Average temperature') +
                      theme_bw(base_size = 16) +
                      theme(plot.title=element_text(family="Times", face="bold", size=20),
                      axis.text.x = element_text(colour="grey20", size=12, angle=315, hjust=.5, vjust=.5
                      axis.text.y = element_text(colour="grey20", size=12))
my_perfect_plot
```



Now, labels became bigger and readable, but there are still few things that one could be improved. Please, take another five minutes and try to add another one or two things, to make it look even more beautiful. Use ggplot2 cheat sheet, which we linked earlier for inspiration.

Here are some ideas:

- See if you can change thickness of the lines.
- Can you find a way to change the name of the legend? What about its labels?
- Use different color palette to improve the look (http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/)

When you created your perfect plot. You can save it to the file in your favourite format. You can easily change the size by specifying the width and height of your plot.

```
ggsave(filename = "figures/observed_species_in_time.png", plot = my_perfect_plot, width=15, height=10)
```

Enjoy plotting with ggplot2!