Facetting Plots

Author: Cole Brookson **Date:** 20 July 2022

Displaying all the relevant information on a single plot panel is ideal as it minimizes the amount of space consumed, but it sometimes impossible to use only one single plot panel and maintain a readable plot that isn't crowded or trying to do so much. In this case, a handy approach is to *facet* the plot which essentially breaks up the plot into multiple panes so that they can be viewed together.

Often, this is the case when one or both axis are the same, and a separate variable or group of that variable is being shown on each plotting pane. For example here, let's refer to data on stream chemisty:

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(lterdatasampler)

df <- lterdatasampler::luq_streamchem
df
```

```
## # A tibble: 317 x 22
##    sample_id sample_date gage_ht  temp   p_h  cond    cl no3_n so4_s    na     k
##    <chr>     <date>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 QS        1987-01-05     2.82    20  7.22  48.2   7.3    97  0.52  4.75  0.18
##  2 QS        1987-01-13     2.66    20  7.34  49.8   7.5   114  0.73  4.81  0.19
##  3 QS        1987-01-20     2.61    20  7.12  50.3   7.5   115 NA     5.19  0.2
##  4 QS        1987-01-27     2.58    20  7.19  50.4   7.3   117 NA     5.08  0.18
##  5 QS        1987-02-03     2.8     20  7.36  49.6   7.3   103  0.85  4.86  0.17
##  6 QS        1987-02-10     2.63    20  7.19  53.3   7.2   110 NA     5.11  0.18
##  7 QS        1987-02-17     2.84    20 NA     43.7   7      94 NA     4.8   0.17
##  8 QS        1987-02-24     2.68    19  6.93  48.4   7.3    90 NA     5.08  0.18
##  9 QS        1987-03-03     2.76    20  7.02  48.4   7.6    86 NA     4.9   0.2
## 10 QS        1987-03-10     2.64    20  7.17  49.9   7.3    97 NA     4.89  0.2
## # ... with 307 more rows, and 11 more variables: mg <dbl>, ca <dbl>,
## #   nh4_n <dbl>, po4_p <dbl>, doc <dbl>, dic <dbl>, tdn <dbl>, tdp <dbl>,
## #   si_o2 <dbl>, don <dbl>, tss <dbl>
```

There are two variables we might want to compare - Sodium (Na) and Potassium (K). One option could be plotting them through time with two y-axes, but these plots are often confusing. An easier to interpret way would be to have two separate plot panels. This will require some data prep first.

Data Preparation

One (not the only) way to go about this, is to get both of our variables of interest into one column, with a separate grouping column that denotes what the values refer to. The easiest way to do this is by *re-shaping* the data (LINK TO RESHAPING HERE) Let's do that here:

```
df_long <- df %>%
  # remove columns we don't need
```

```
  dplyr::select(sample_date, na, k) %>%
  tidyr::pivot_longer(
    # specify which columns to join together
    cols = c(na, k),
    # specify what the new name of the grouping variable will be
    names_to = "element"
  )
df_long
```

```
## # A tibble: 634 x 3
##    sample_date element value
##    <date>      <chr>   <dbl>
##  1 1987-01-05  na       4.75
##  2 1987-01-05  k        0.18
##  3 1987-01-13  na       4.81
##  4 1987-01-13  k        0.19
##  5 1987-01-20  na       5.19
##  6 1987-01-20  k        0.2
##  7 1987-01-27  na       5.08
##  8 1987-01-27  k        0.18
##  9 1987-02-03  na       4.86
## 10 1987-02-03  k        0.17
## # ... with 624 more rows
```

Perfect! We will also want to pull out the month and year values as columns since it will be month we'll use to group by in our plots.

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

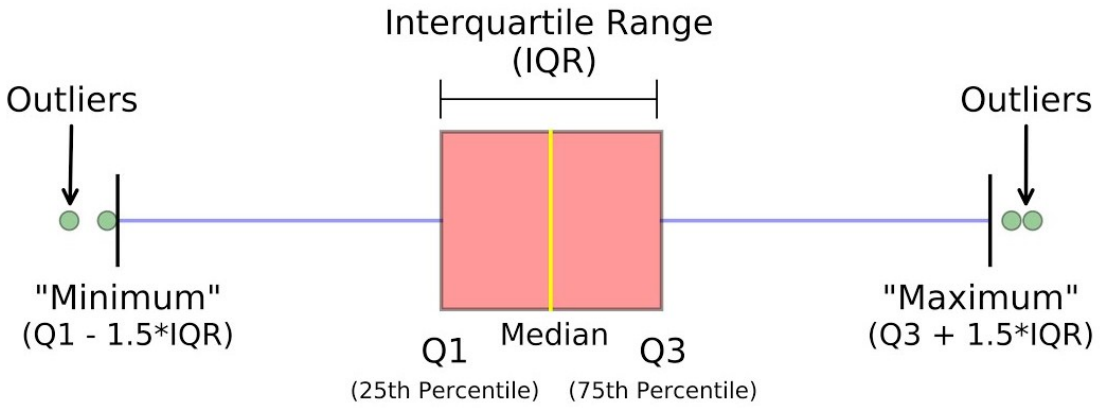
```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
df_long <- df_long %>%
  dplyr::mutate(
    year = lubridate::year(sample_date),
    month = lubridate::month(sample_date)
  )
df_long
```

```
## # A tibble: 634 x 5
##    sample_date element value  year month
##    <date>      <chr>   <dbl> <dbl> <dbl>
##  1 1987-01-05  na       4.75  1987     1
##  2 1987-01-05  k        0.18  1987     1
##  3 1987-01-13  na       4.81  1987     1
##  4 1987-01-13  k        0.19  1987     1
##  5 1987-01-20  na       5.19  1987     1
##  6 1987-01-20  k        0.2   1987     1
##  7 1987-01-27  na       5.08  1987     1
##  8 1987-01-27  k        0.18  1987     1
##  9 1987-02-03  na       4.86  1987     2
## 10 1987-02-03  k        0.17  1987     2
```

```
## # ... with 624 more rows
```

Now we can do about plotting this iteratively (LINK TO ITERATIVE PLOTTING). In this example, we'll use a box plot (LINK TO CHANGE OVER TIME) since it's one way to show a change through time. As a quick refresher, here's how we can interpret a boxplot:



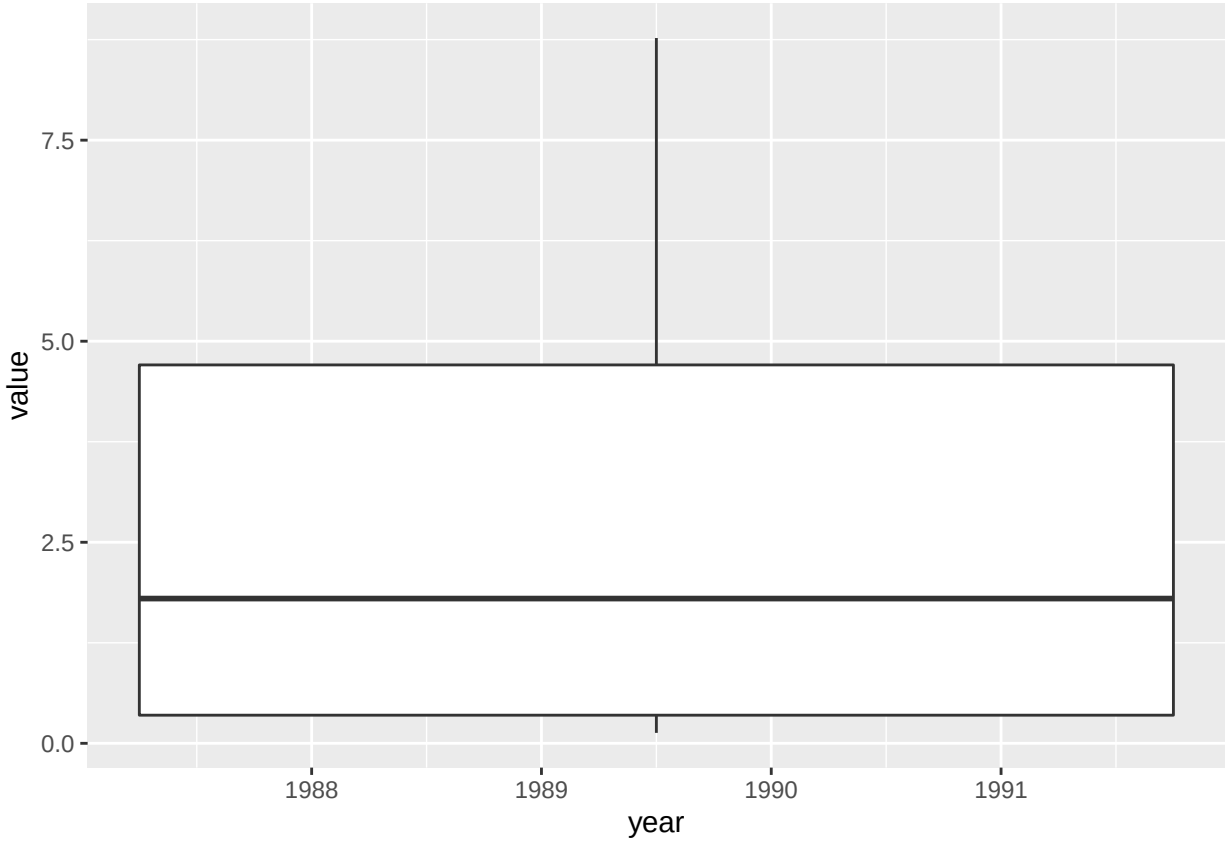
Plotting With No Facet

Let's begin by plotting only the bare necessities:

```
ggplot() +
  geom_boxplot(data = df_long, mapping = aes(x = year, y = value))
```

```
## Warning: Continuous x aesthetic -- did you forget aes(group=...)?
```

```
## Warning: Removed 11 rows containing non-finite values (stat_boxplot).
```



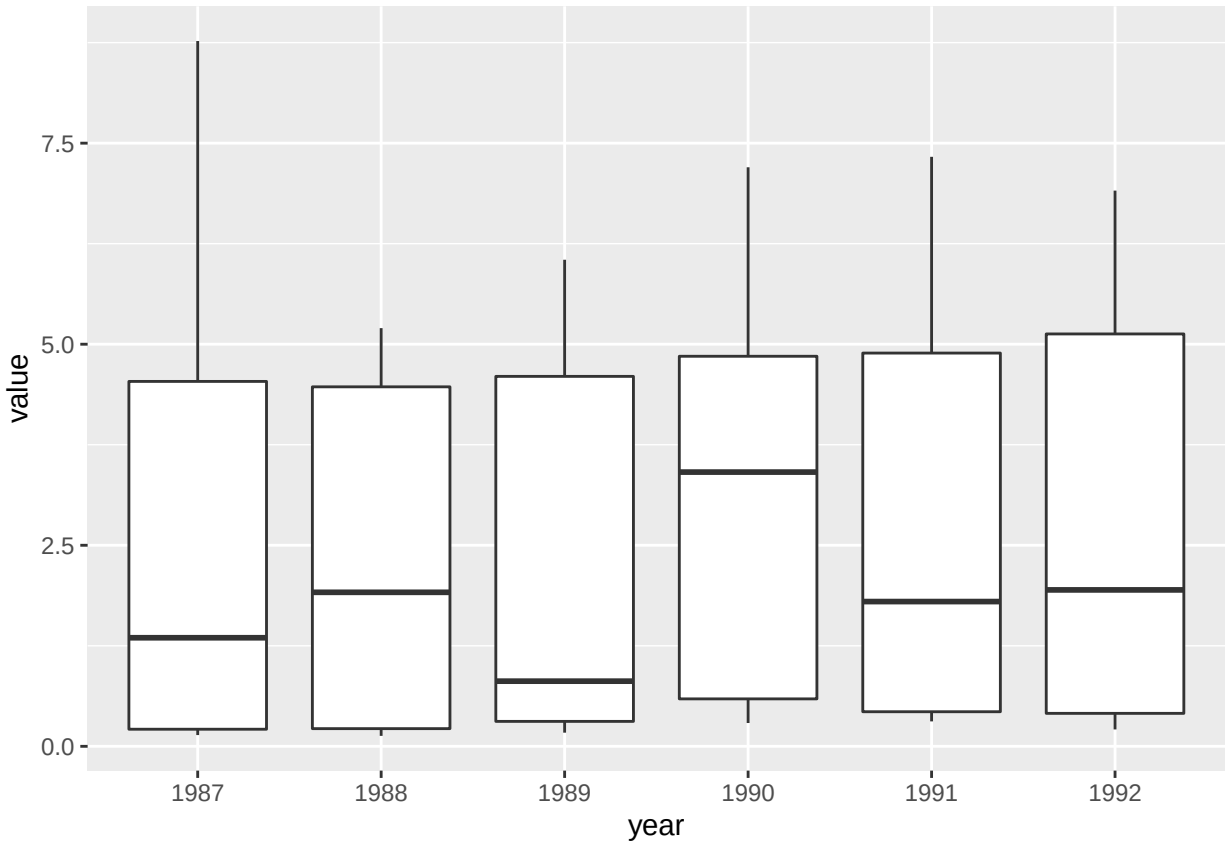
Well that didn't quite work out. We will need to make `year` a factor here:

```
df_long = df_long %>%
  dplyr::mutate(
    year = as.factor(year)
  )
```

Now we'll try again:

```
ggplot() +
  geom_boxplot(data = df_long, mapping = aes(x = year, y = value))
```

```
## Warning: Removed 11 rows containing non-finite values (stat_boxplot).
```



That looks better. **Now here's why we might want to facet:** right now our plot is displaying **both** sodium and potassium in the same `value` which doesn't makes sense since they're completely different scales of measurement. We could try to address this by separating them out by some aesthetic (e.g. colour), so let's try that:

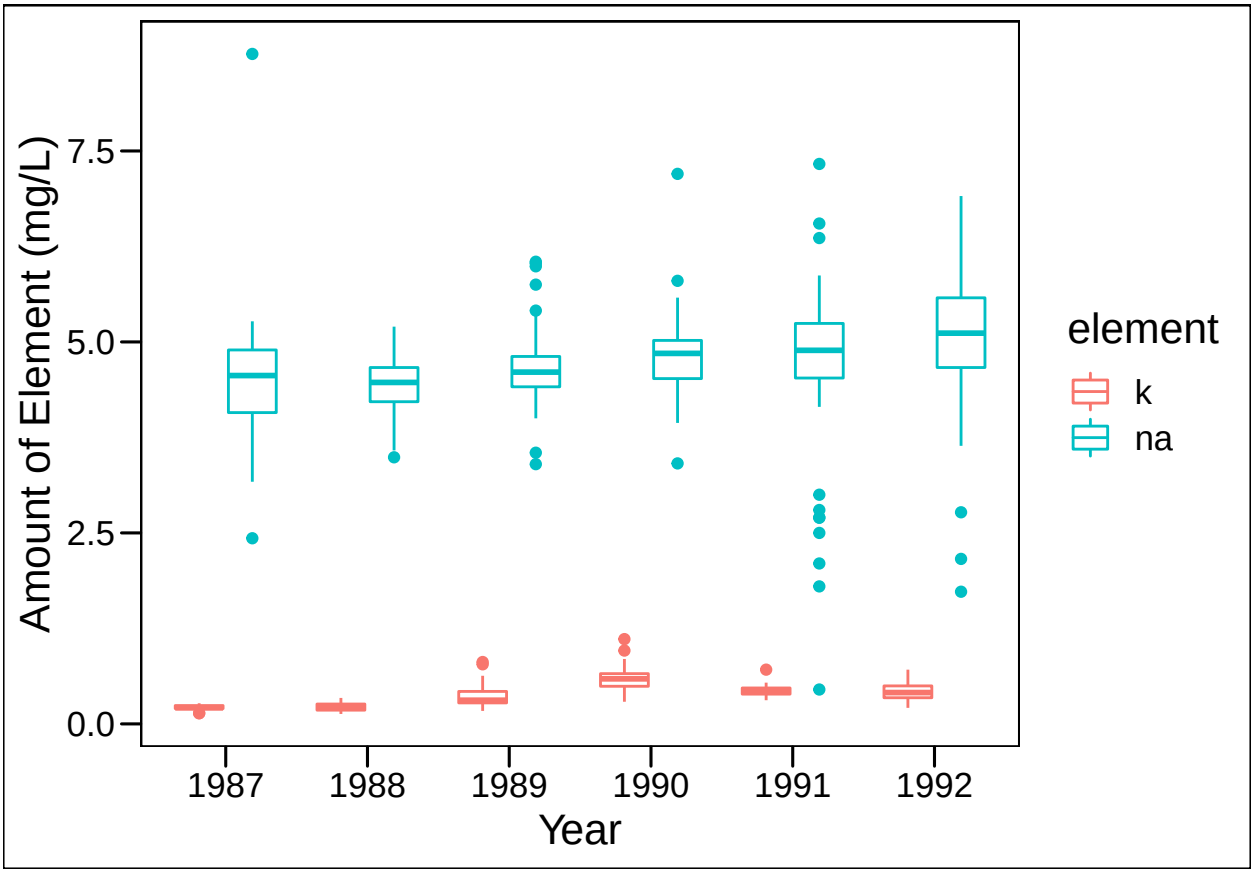
```
ggplot() +
  geom_boxplot(data = df_long, mapping = aes(x = year, y = value,
                                             colour = element))
```

```
## Warning: Removed 11 rows containing non-finite values (stat_boxplot).
```

Ok so now we have grouped box plots. I suppose in theory we could leave it like this, but I am not personally a fan. Let's add in the necessary components and then think about faceting:

```
ggplot() +
  geom_boxplot(data = df_long, mapping = aes(x = year, y = value,
                                             colour = element)) +
  labs(x = "Year", y = "Amount of Element (mg/L)") +
  ggthemes::theme_base()
```

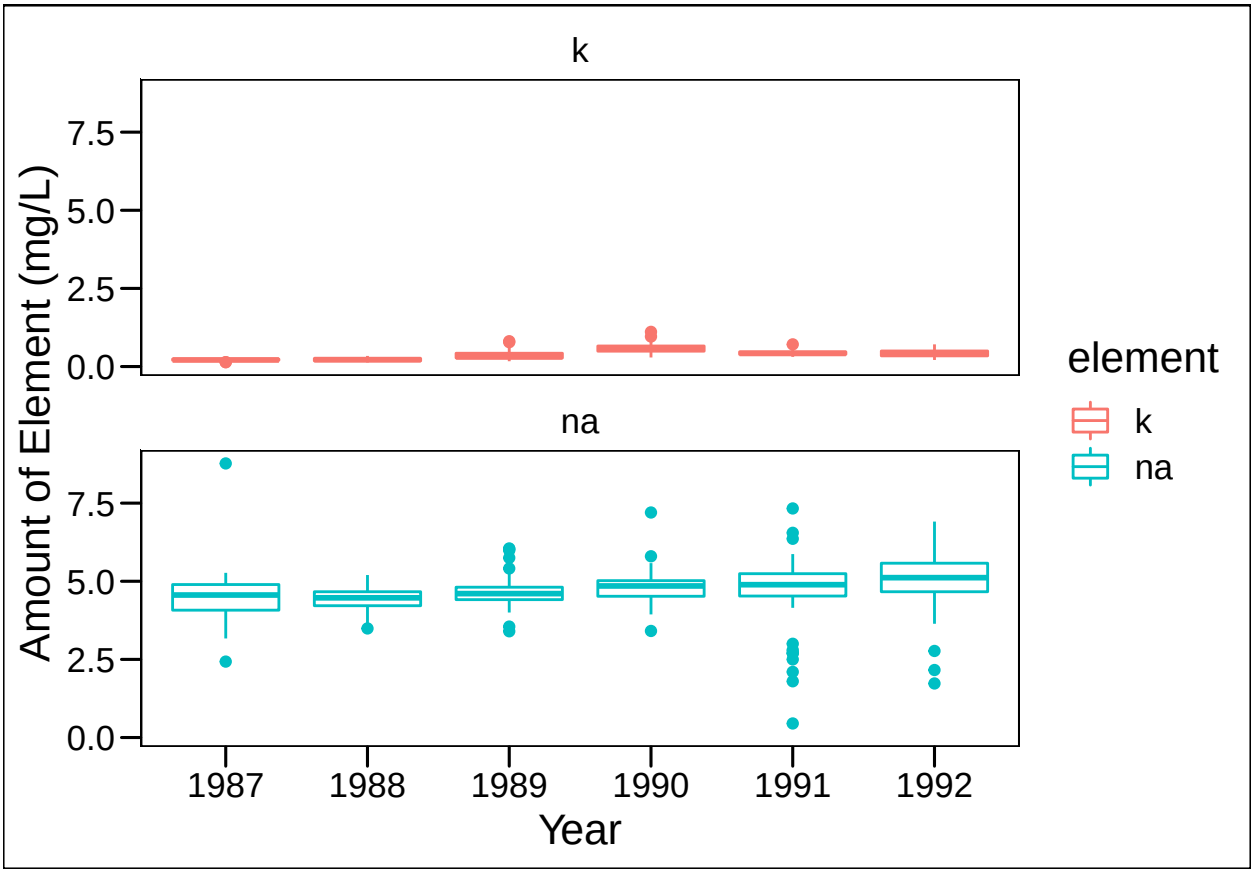
```
## Warning: Removed 11 rows containing non-finite values (stat_boxplot).
```



Nope, still heinous! Time to facet! We'll make this on two panels by faceting using `facet_wrap()` and operating on the `element` variable. I also want the plot to display as two plots stacked *on top* of one another, so I can refer to the number of columns as 1 to force the `facet_wrap()` to do this.

```
ggplot() +
  geom_boxplot(data = df_long, mapping = aes(x = year, y = value,
                                             colour = element)) +
  labs(x = "Year", y = "Amount of Element (mg/L)") +
  ggthemes::theme_base() +
  facet_wrap(~element,
             # I want this to be one column with two rows
             ncol = 1)
```

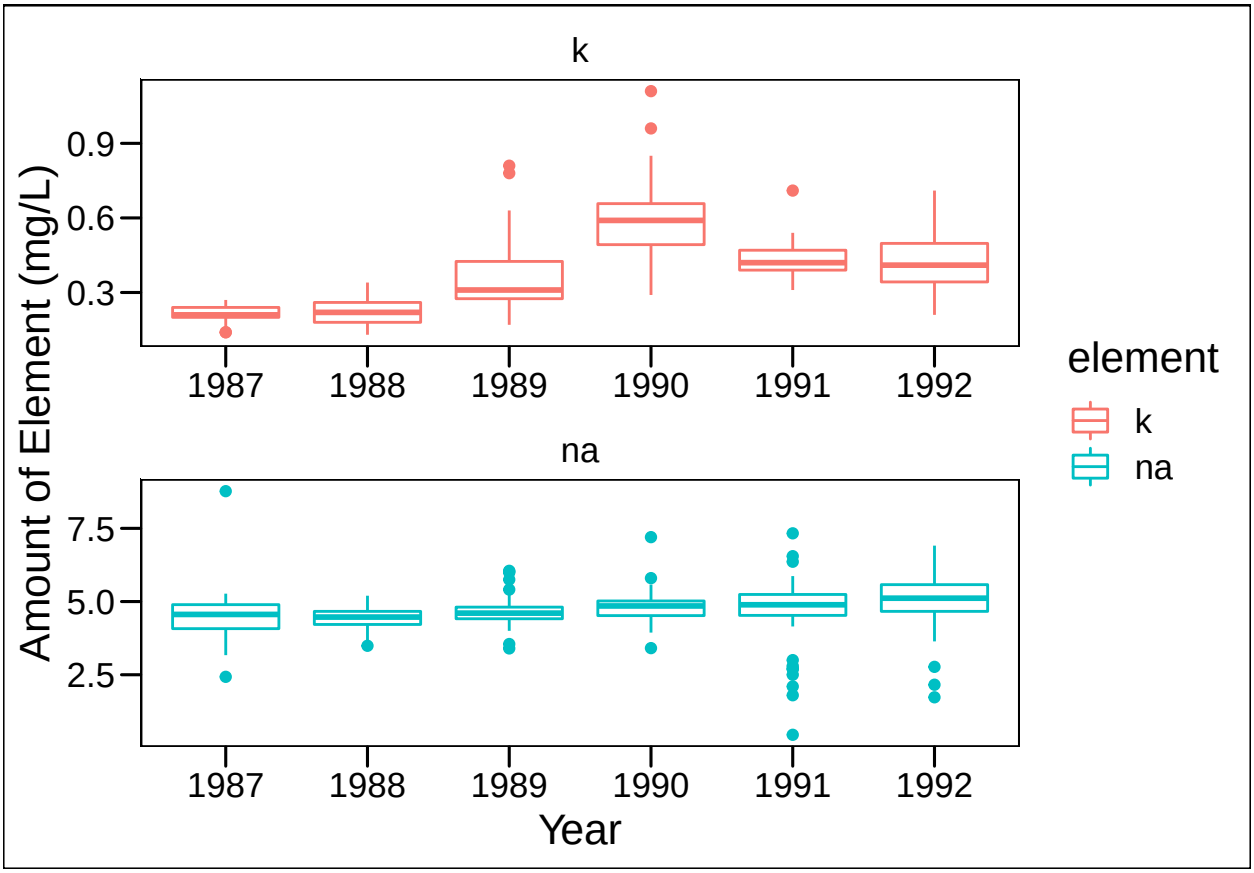
```
## Warning: Removed 11 rows containing non-finite values (stat_boxplot).
```



Ok, we're getting there! However, the scales for both are not helping. Let's fix it so the sodium can vary within it's max & min, and the potassium can vary within it's (different) max & min. We can do this with the `scales` argument in the facet_wrap.

```
ggplot() +
  geom_boxplot(data = df_long, mapping = aes(x = year, y = value,
                                             colour = element)) +
  labs(x = "Year", y = "Amount of Element (mg/L)") +
  ggthemes::theme_base() +
  facet_wrap(~element,
             # I want this to be one column with two rows
             ncol = 1,
             scales = "free")
```

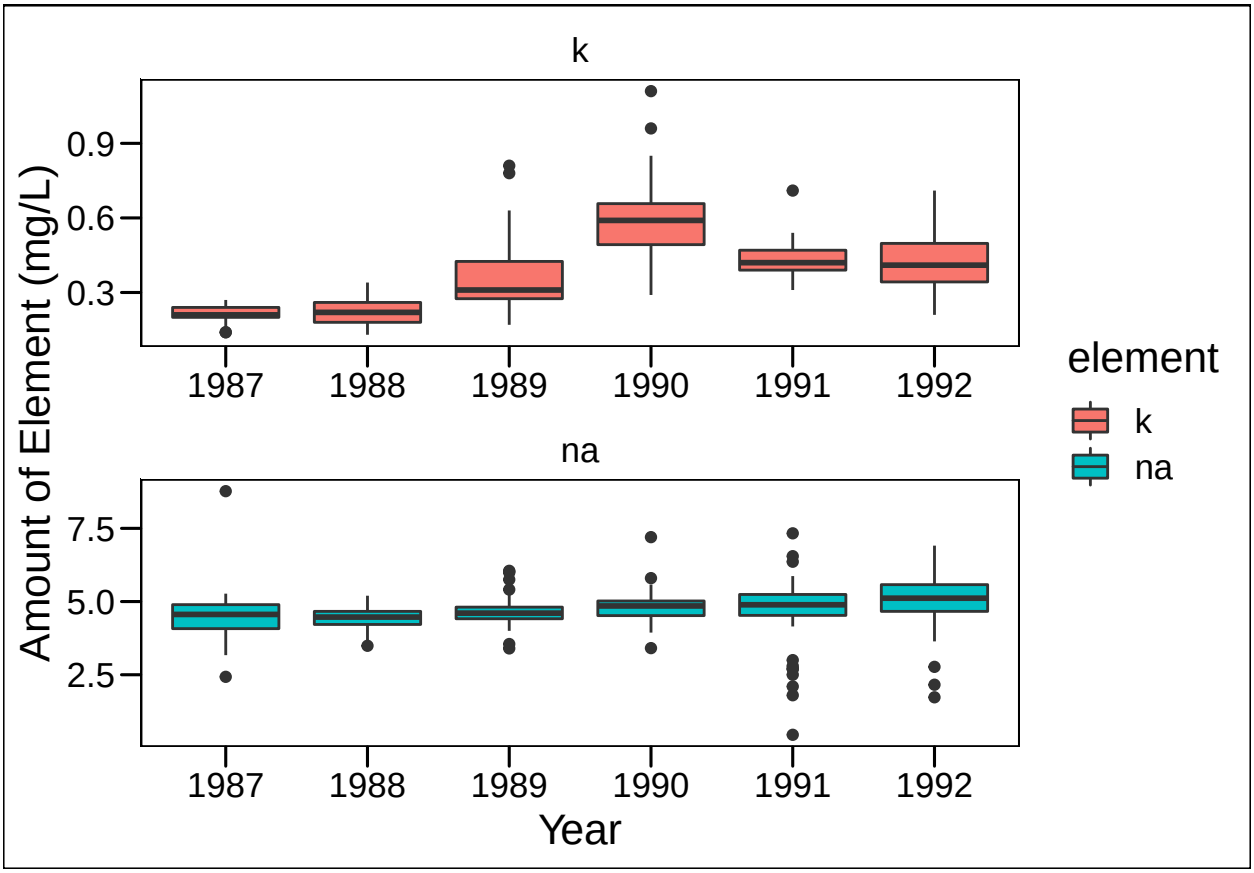
```
## Warning: Removed 11 rows containing non-finite values (stat_boxplot).
```



Okay, this is looking better! Now it feels like we can actually see the spread of the data. We can still make some aesthetic changes however. Let's change to a `fill` not a `colour` on our box plot:

```
ggplot() +
  geom_boxplot(data = df_long, mapping = aes(x = year, y = value,
                                             fill = element)) +
  labs(x = "Year", y = "Amount of Element (mg/L)") +
  ggthemes::theme_base() +
  facet_wrap(~element,
             # I want this to be one column with two rows
             ncol = 1,
             scales = "free")
```

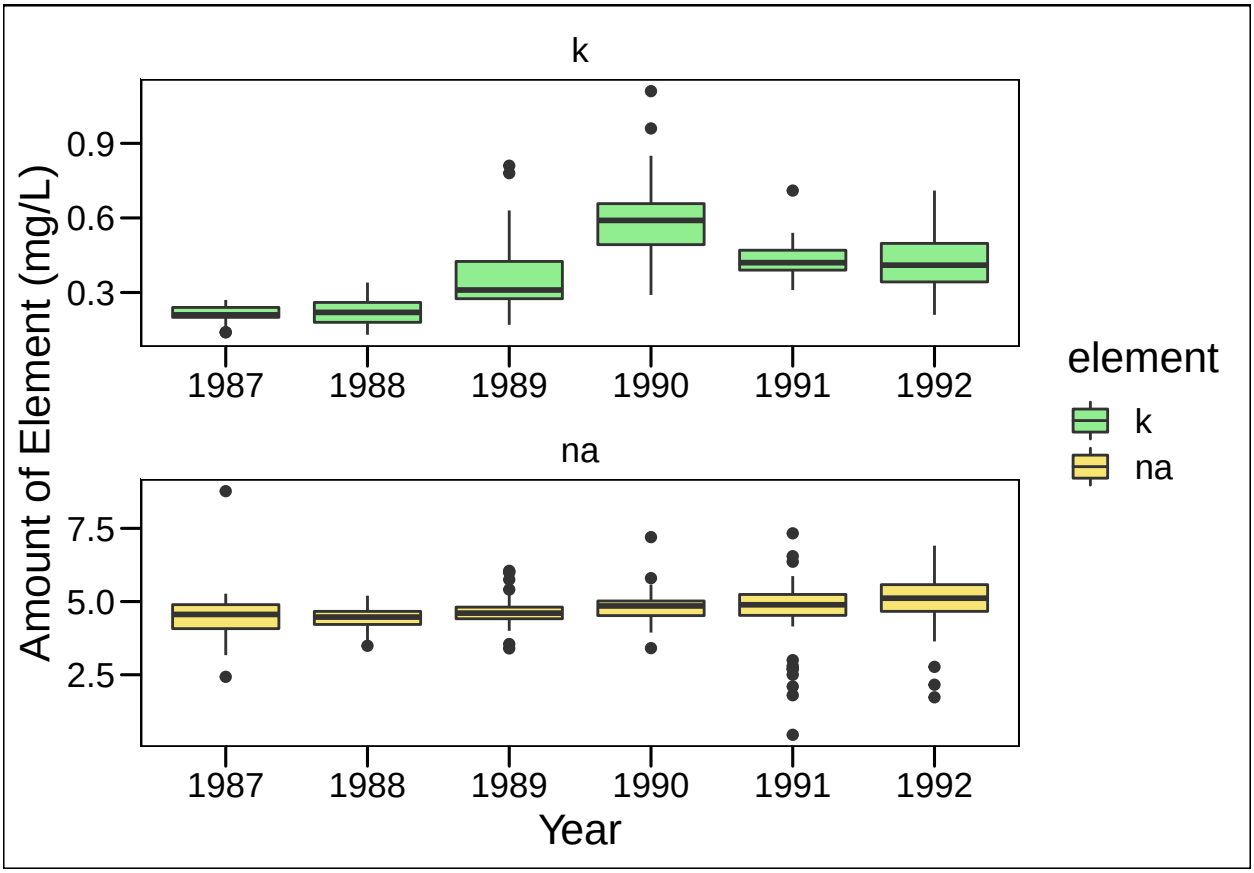
```
## Warning: Removed 11 rows containing non-finite values (stat_boxplot).
```



Good, and now we'll change the actual colours we're using with `scale_fill_manual()`:

```
ggplot() +
  geom_boxplot(data = df_long, mapping = aes(x = year, y = value,
                                             fill = element)) +
  labs(x = "Year", y = "Amount of Element (mg/L)") +
  ggthemes::theme_base() +
  facet_wrap(~element,
             # I want this to be one column with two rows
             ncol = 1,
             scales = "free") +
  scale_fill_manual(values = c("#90ee90", "#F8E473"))
```

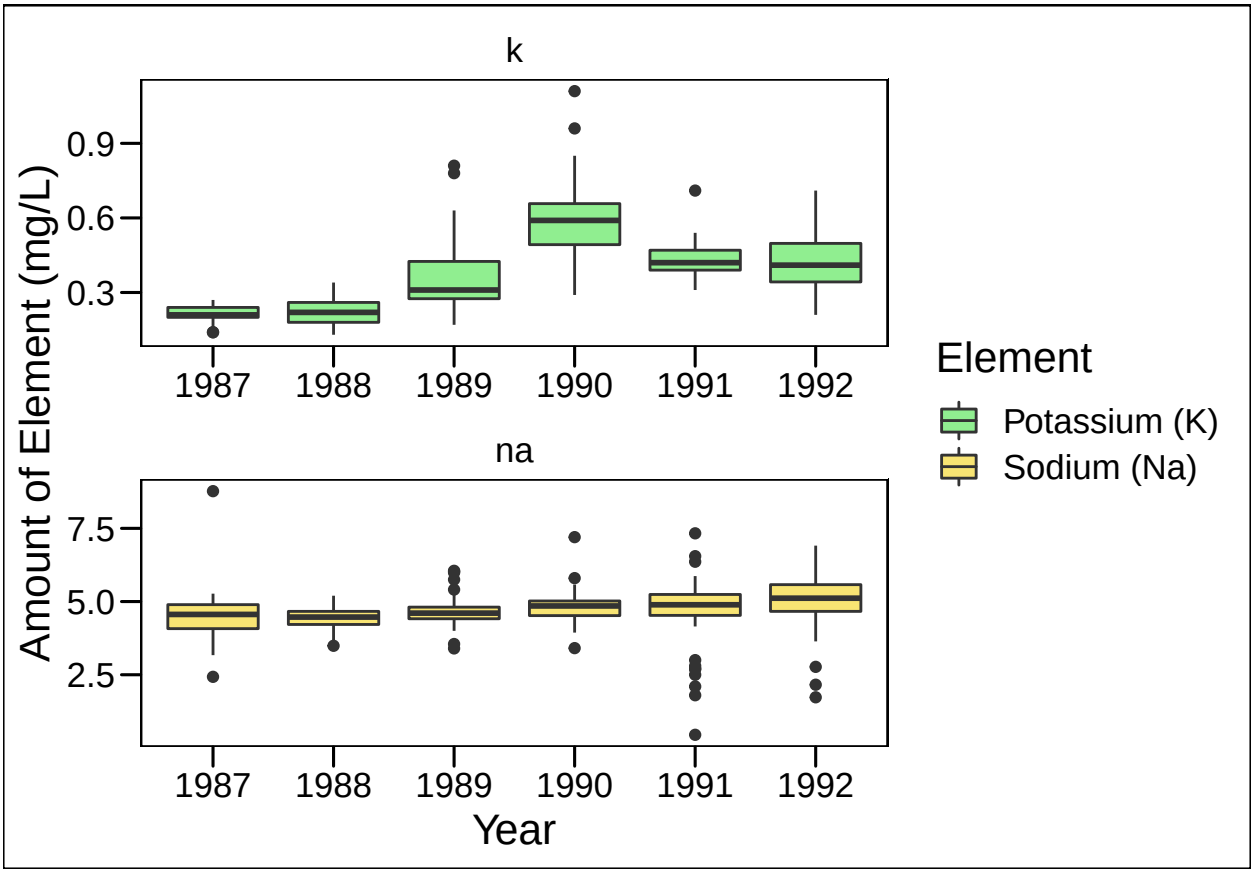
```
## Warning: Removed 11 rows containing non-finite values (stat_boxplot).
```



I would also like to change the legend to have the full names of the elements. Recall we can *also* use the `scale_fill_manual()` to perform this:

```
ggplot() +
  geom_boxplot(data = df_long, mapping = aes(x = year, y = value,
                                             fill = element)) +
  labs(x = "Year", y = "Amount of Element (mg/L)") +
  ggthemes::theme_base() +
  facet_wrap(~element,
             # I want this to be one column with two rows
             ncol = 1,
             scales = "free") +
    scale_fill_manual("Element",
                      values = c("#90ee90", "#F8E473"),
                      labels = c("Potassium (K)",
                                 "Sodium (Na)"))
```

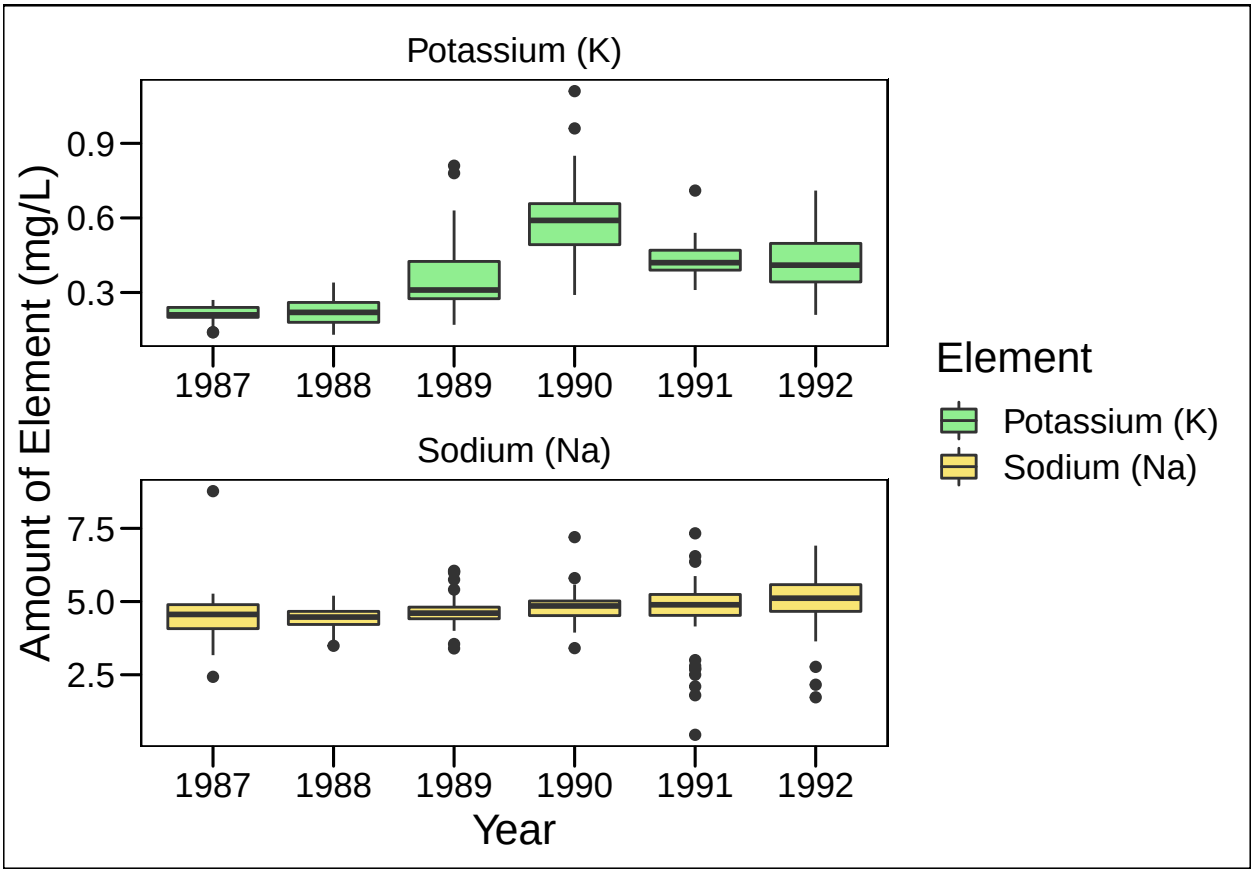
```
## Warning: Removed 11 rows containing non-finite values (stat_boxplot).
```



Looks good! But one last thing. Somewhere along the way, there have developed some ugly labels on the top of each of the plots. We could expand them so they have the same thing as the legend. This is done via the `labeller` argument inside the call to `facet_grid()`. We pass a vector of matching values to the `as_labeller()` function and that goes as the argument to `labeller`. Yes, it's all annoyingly complicated ...

```
ggplot() +
  geom_boxplot(data = df_long, mapping = aes(x = year, y = value,
                                             fill = element)) +
  labs(x = "Year", y = "Amount of Element (mg/L)") +
  ggthemes::theme_base() +
  facet_wrap(~element,
             # I want this to be one column with two rows
             ncol = 1,
             scales = "free",
             labeller = as_labeller(c(`k` = "Potassium (K)",
                                      `na` = "Sodium (Na)"))) +
    scale_fill_manual("Element",
                      values = c("#90ee90", "#F8E473"),
                      labels = c("Potassium (K)", "Sodium (Na)"))
```

```
## Warning: Removed 11 rows containing non-finite values (stat_boxplot).
```



Okay, well there we have it!! I would personally argue that you could likely leave off either the legend *or* the title as you really only need one of the two, but for the sake of practice, we can leave it as is.