Visualizing DayCent Results in R

John Wainwright

11 June 2025

Getting Started

The file "Daycent example and outputs.zip" on Learn Ultra contains a complete set of DayCent input files and the related outputs. Download and unzip the file and ensure that the folder "DDcent_v16_MED_ENV" is in the same folder as this Rmd file. (Remember you can check using Session –> Set Working Directory –> To Source File Location, clicking the little arrow next to the path name at the top of the Console pane and looking at the list of files in the Files pane in RStudio.)

The output files all have the extension .out

File Name	Description
bio.out	daily above and below ground live carbon
soiln.out	daily soil ammonium and nitrate by layer
soiltavg.out	daily average soil temperature by layer
soiltmax.out	daily maximum soil temperature by layer
soiltmin.out	daily minimum soil temperature by layer
stemp_dx.out	daily soil temperature every few cm
vswc.out	daily volumetric soilwater content by layer
watrbal.out	daily water balance
wfps.out	daily water filled pore space by layer
co2.out	daily CO2 concentrations by layer
wflux.out	daily water flux through the bottom of soil layers
resp.out	daily maintenance respiration
year_summary.out	yearly gas fluxes yearly carbon flows
year_cflows.out	
livec.out	daily live C
deadc.out	daily dead C
soilc.out	daily soil C
sysc.out	daily system C
tgmonth.out	monthly trace gas fluxes

Let's start by looking at the year_summary.out file, as that contains yearly summaries and is thus the simplest file available. You can open it up in a text editor to see what it looks like, and you should see it is in space-delimited format, with no metadata and a header line. Thus, we can use the read.table command to get the values into R:

```
dayCentFolder <- "./DDcent_v16_MED_ENV/"
yearFile <- "year_summary.out"

yearSummary <- read.table (paste0 (dayCentFolder, yearFile),
                           header = TRUE, sep = "")
```

```
head (yearSummary)
```

```
##   time  N2Oflux   NOflux N2flux      CH4      NIT   ANNPPT
## 1 1900 0.015553 0.040656      0 0.281036 7.240733 47.00001
## 2 1901 0.009474 0.031216      0 0.274963 5.051083 26.50000
## 3 1902 0.017193 0.046739      0 0.289568 9.517772 40.00000
## 4 1903 0.007044 0.026411      0 0.273591 4.524124 22.70000
## 5 1904 0.015703 0.045697      0 0.284326 7.821037 34.00000
## 6 1905 0.007747 0.028813      0 0.272955 4.823972 29.10001
```

So we can see there are seven variables giving the time of the output, various gas fluxes (in g gas / m^2) and the annual precipitation (in cm). As centimetres aren't SI units, I'll convert these values to mm before doing anything else and then have a quick look at some summary statistics of the dataset:

```
yearSummary <- yearSummary %>%
  mutate (ANNPPT_mm = ANNPPT * 10.)

summary (yearSummary)
```

```
##       time         N2Oflux             NOflux             N2flux
##  Min.   :1900   Min.   :0.001437   Min.   :0.004428   Min.   :0
##  1st Qu.:1918   1st Qu.:0.002269   1st Qu.:0.006768   1st Qu.:0
##  Median :1935   Median :0.003342   Median :0.010076   Median :0
##  Mean   :1935   Mean   :0.004672   Mean   :0.013786   Mean   :0
##  3rd Qu.:1952   3rd Qu.:0.005972   3rd Qu.:0.016643   3rd Qu.:0
##  Max.   :1970   Max.   :0.017193   Max.   :0.046739   Max.   :0
##       CH4              NIT              ANNPPT        ANNPPT_mm
##  Min.   :0.2576   Min.   :0.5768   Min.   :22.70   Min.   :227.0
##  1st Qu.:0.2732   1st Qu.:0.8994   1st Qu.:29.95   1st Qu.:299.5
##  Median :0.2811   Median :1.7045   Median :34.00   Median :340.0
##  Mean   :0.2833   Mean   :2.2811   Mean   :34.41   Mean   :344.1
##  3rd Qu.:0.2941   3rd Qu.:2.6552   3rd Qu.:39.25   3rd Qu.:392.5
##  Max.   :0.3139   Max.   :9.5178   Max.   :47.20   Max.   :472.0
```

See the VisualizingMAHLERANResultsInR file for fuller details of what you can do with the read.table function.

Visualizing this Output

The tidyverse package has a very powerful plotting capability called ggplot (we can also use it without loading the rest of the tidyverse by opening the ggplot2 library).

To take full advantage of ggplot, we need first to do a little change to the layout of the data. At the moment yearSummary is in what's called wide format; in other words, it looks like a table with column headings relating to the variable names, and rows for each set of values (which in this case refer to a specific year). Although ggplot can use data in this format, it is far more powerful if we change our data to what's called long format by stacking the variables on top of each other. We use pivot_longer from tidyverse to do this:

```
yearSummaryLong <- yearSummary %>%
  select (-ANNPPT) %>%
  pivot_longer (-time, names_to = "variable")
head (yearSummaryLong, 8)
```

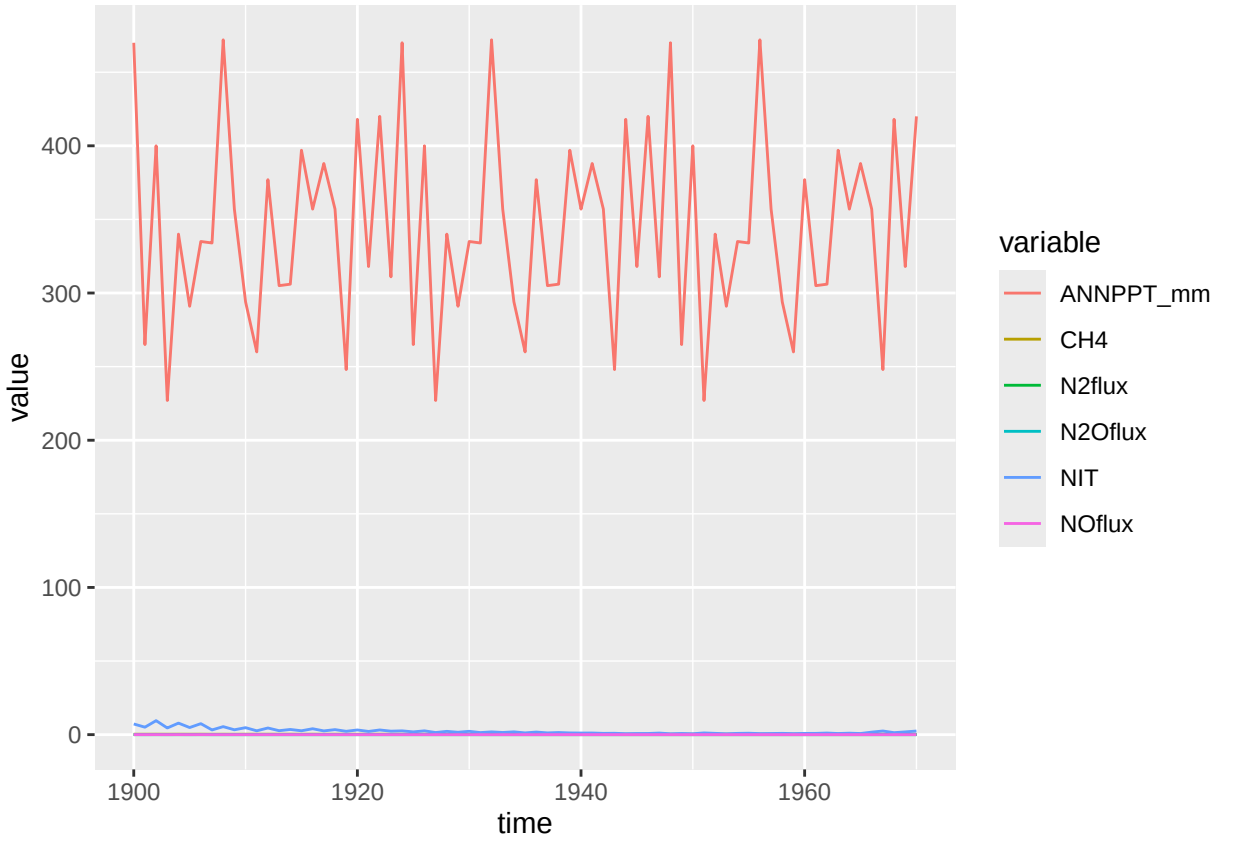
```
## # A tibble: 8 x 3
##    time variable      value
##   <dbl> <chr>         <dbl>
```

```
## 1  1900 N2Oflux     0.0156
## 2  1900 NOflux      0.0407
## 3  1900 N2flux      0
## 4  1900 CH4         0.281
## 5  1900 NIT         7.24
## 6  1900 ANNPPT_mm 470.
## 7  1901 N2Oflux     0.00947
## 8  1901 NOflux      0.0312
```

Note that in doing this, I've taken out the original ANNPPT value (the one in centimetres) as it will get in the way later on. (Note also that if you have data in long format, you can use pivot_wider to get the tabular format back.)

The format of ggplot will appear a little strange at first, but makes sense when you get used to it. You need to specify a data source, aesthetics (which are minimally the x and y values, but can also include colours and line types etc.), and a geometry (points, lines, bars, and many more). So let's try a simple call to plot all our data as lines:

```
ggplot (data = yearSummaryLong) +
  geom_line (aes (x = time, y = value, colour = variable))
```

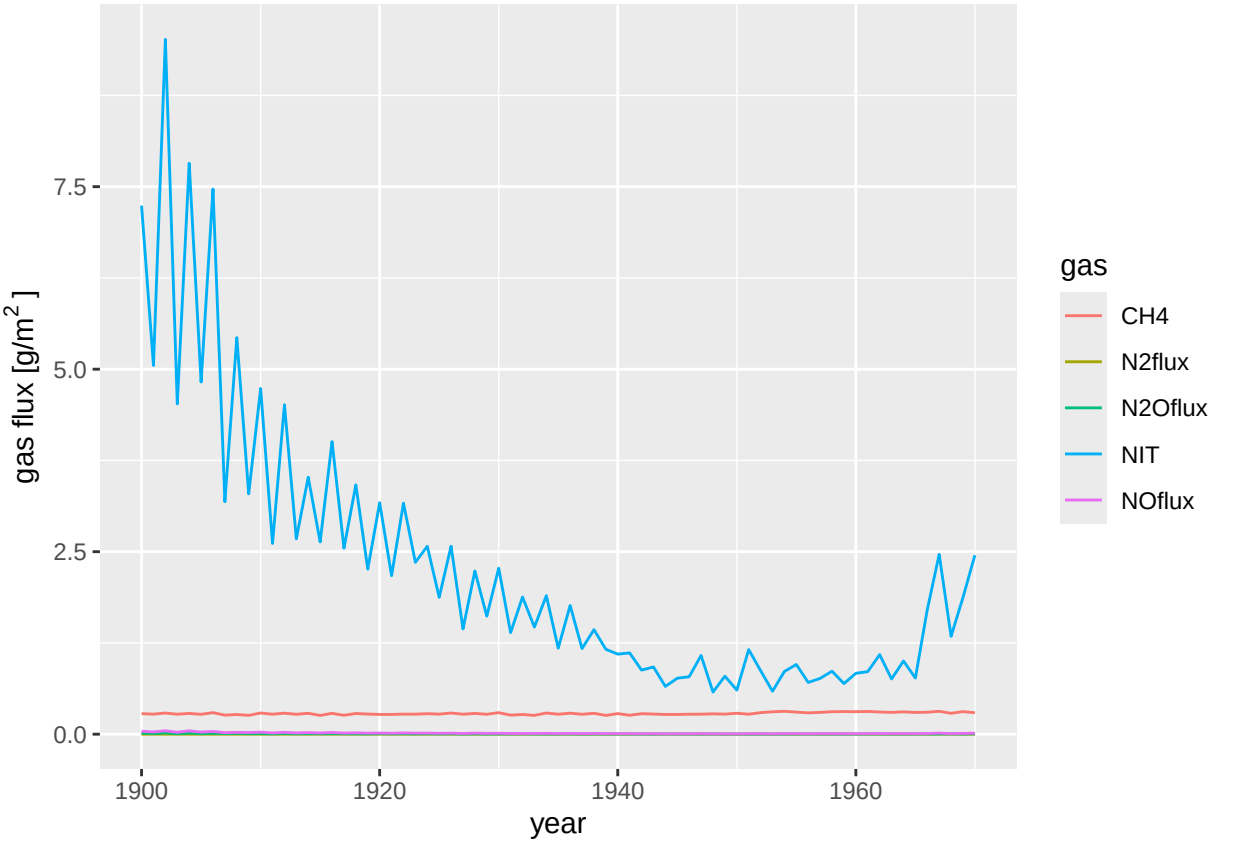


Not perfect, but it's produced a graph with all our data on with just two lines of code, including a basic legend.

What if we only want to plot the gas-flux data so we don't have the complication of the precipitation values. In this case, we use tidyverse to filter the dataset, then pass it to ggplot:

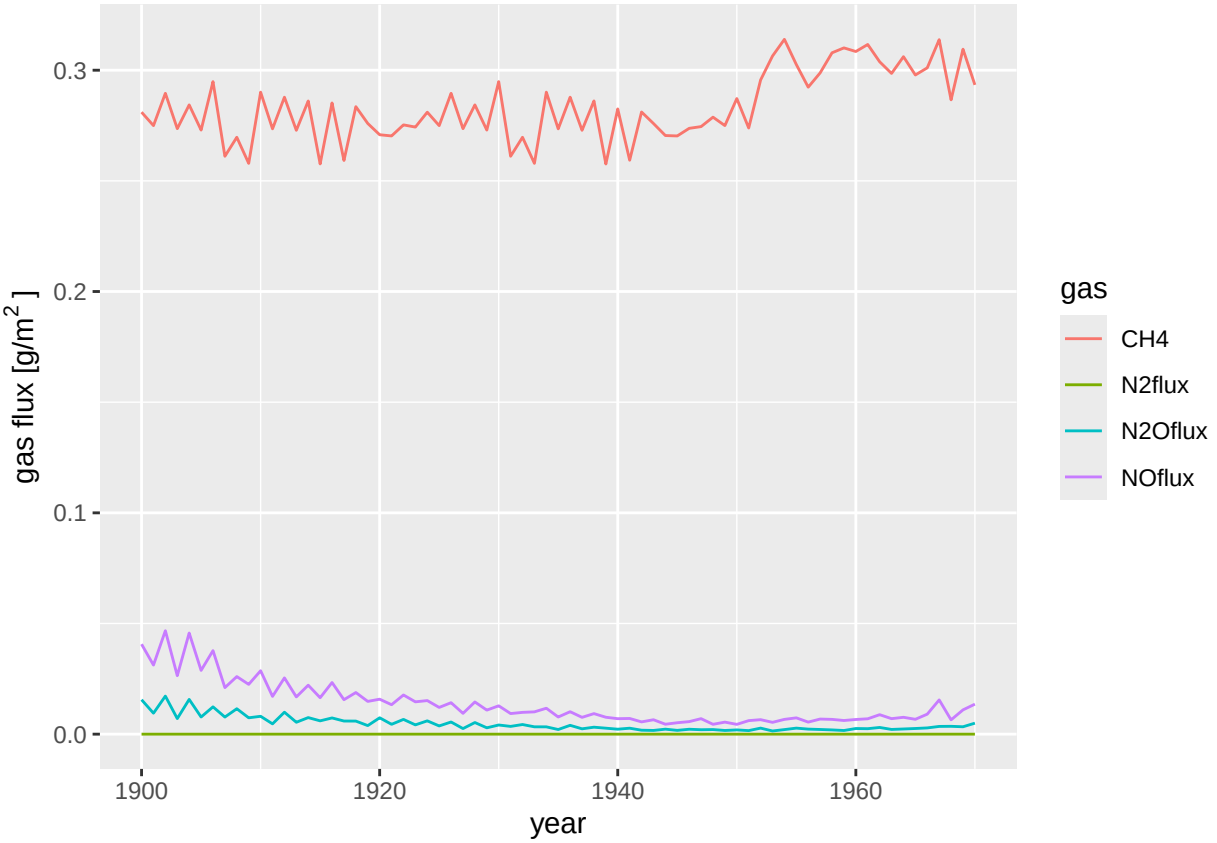
```
yearSummaryLong %>%
  filter (variable != "ANNPPT_mm") %>%
```

```
ggplot () +
  geom_line (aes (x = time, y = value, colour = variable)) +
  labs (x = "year",
        y = expression ("gas flux [g/m"^2~"]"),
        colour = "gas")
```



Notice that ggplot has done the hard work of rescaling the y axis for us. It's still difficult to see all the variability as the NIT values are generally higher than the others, but we could also filter out those values to see what happens:

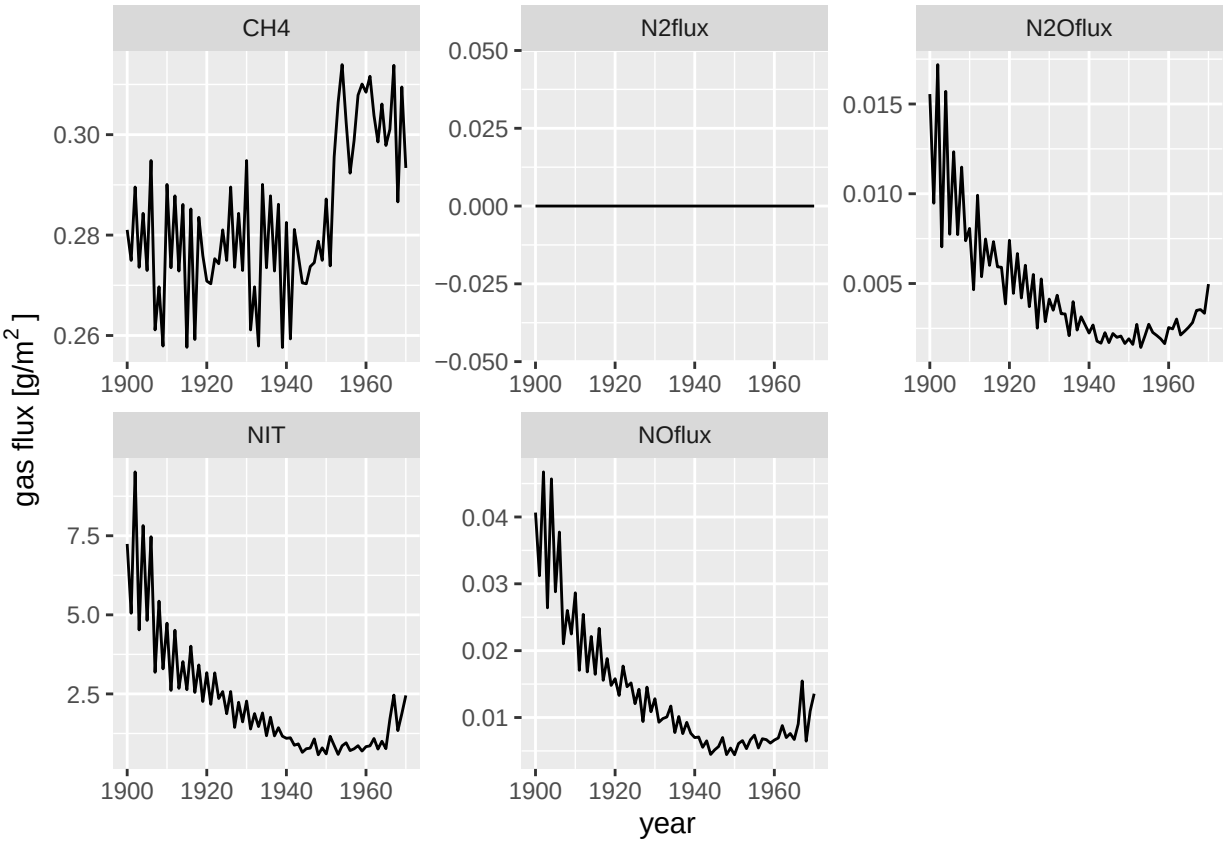
```
yearSummaryLong %>%
  filter (variable != "ANNPPT_mm" & variable != "NIT") %>%
  ggplot () +
    geom_line (aes (x = time, y = value, colour = variable)) +
    labs (x = "year",
          y = expression ("gas flux [g/m"^2~"]"),
          colour = "gas")
```



That's a bit clearer. Note that N2flux is always zero in these results, so we needn't tweak things further to see what's happening there.

Another approach is to use facet_wrap in ggplot to plot the data from the different variables into separate graphs:

```
yearSummaryLong %>%
  filter (variable != "ANNPPT_mm") %>%
  ggplot () +
    geom_line (aes (x = time, y = value)) +
    facet_wrap (~variable, scales = "free") +
    labs (x = "year",
          y = expression ("gas flux [g/m"^2~"]"))
```

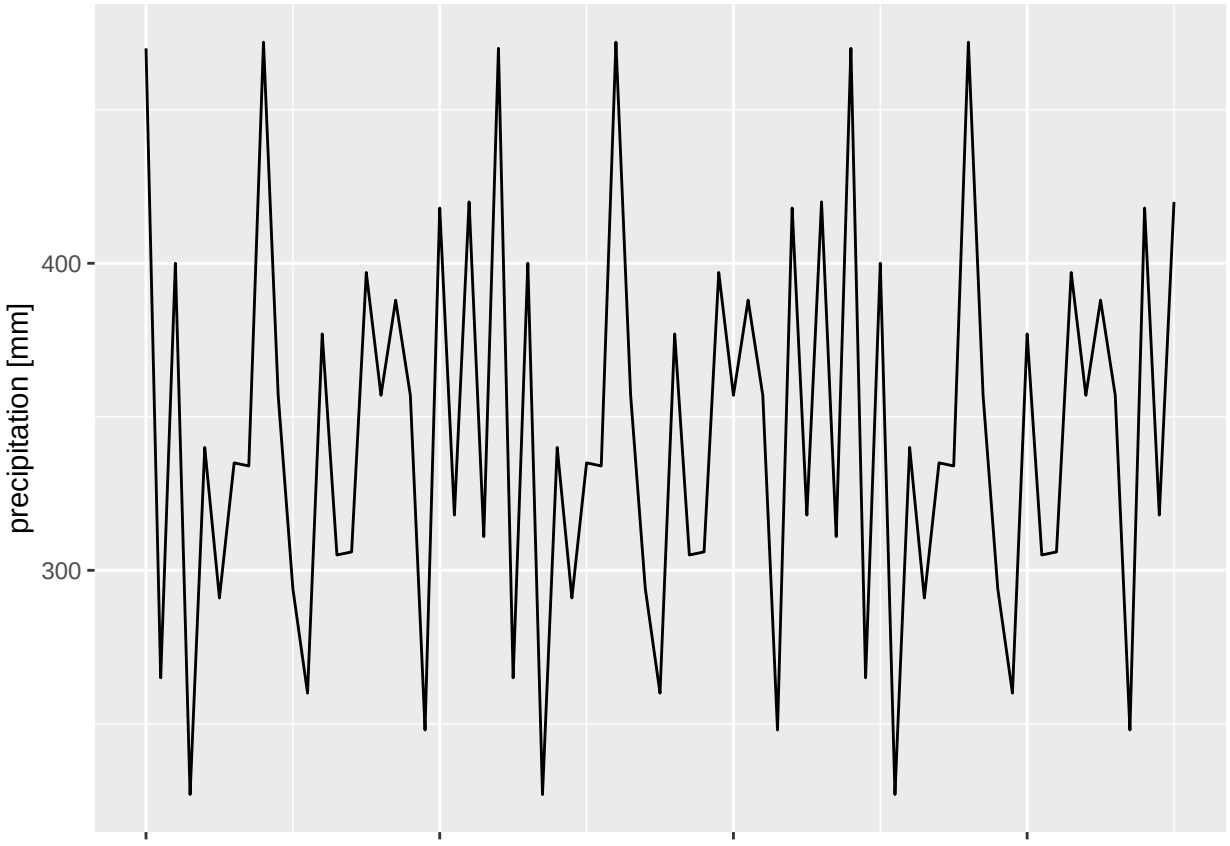


Check what happens when you remove the scales = "free" part of the facet_wrap function call. Can you think why it might be useful to have both forms of graph when exploring your data?

Now, we've conveniently forgotten about plotting the precipitation data on a second axis. The designers of ggplot are fundamentally opposed to second axes as a good way of visualizing data, so they've made adding one as difficult as possible! (Discuss whether you think this is a reasonable approach!!) Rather than fight against the design, let's try a different way to plot the values as separate panels.

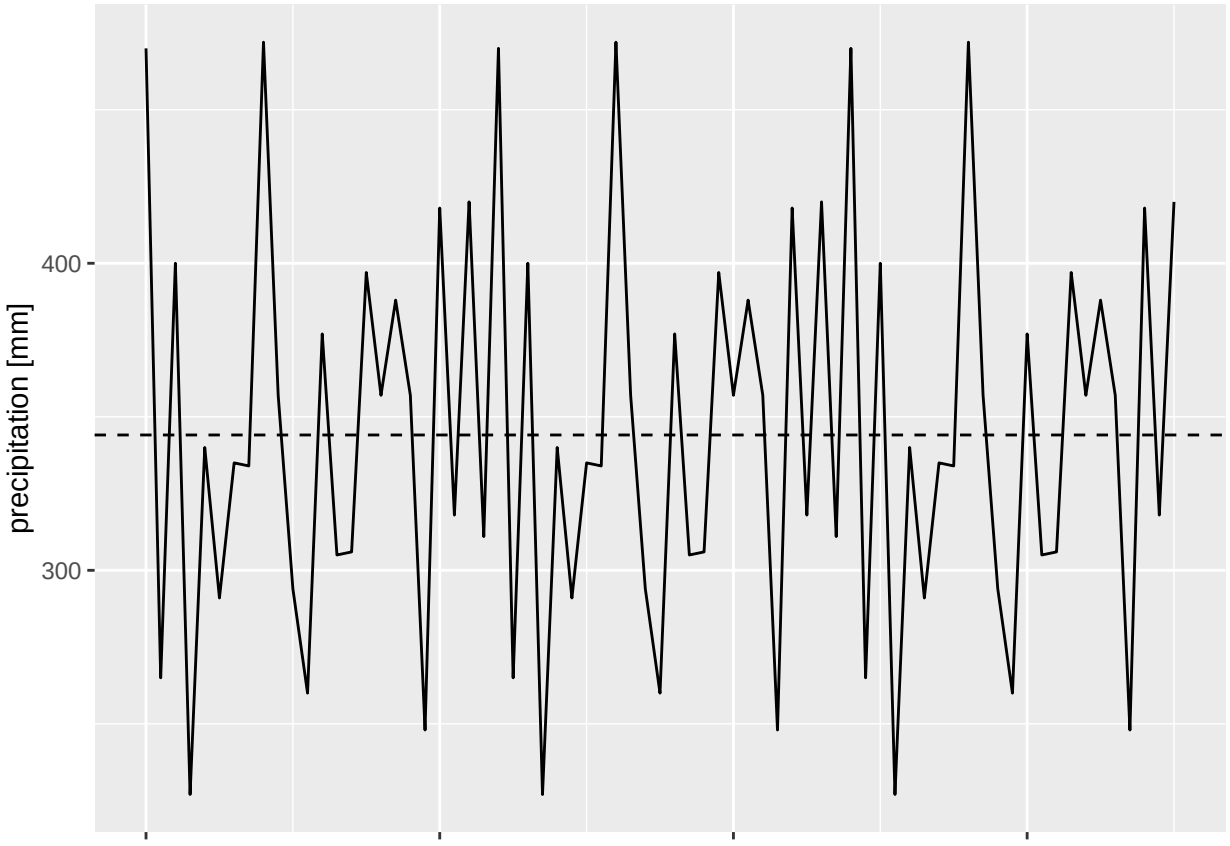
The design is to have separate panels for each of the variables stacked vertically above each other, starting with the precipitation data at the top and having a panel for each of the gas fluxes below. Here's precipitation:

```
pptnPanel <- ggplot (data = yearSummary) +
  geom_line (aes (x = time, y = ANNPPT_mm)) +
  labs (y = "precipitation [mm]") +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

pptnPanel
```



Notice two changes here: the x axis labelling has been switched off for reasons that will become clearer shortly, and I've stored the graph as a data structure, and then used its name to plot it out (comment out the line with the name in to see what happens). We can add other information to an existing plot if we store it this way. Say I want to add a horizontal dashed line to show the mean precipitation over the time period, I can simply take the existing stored ggplot object and add to it:

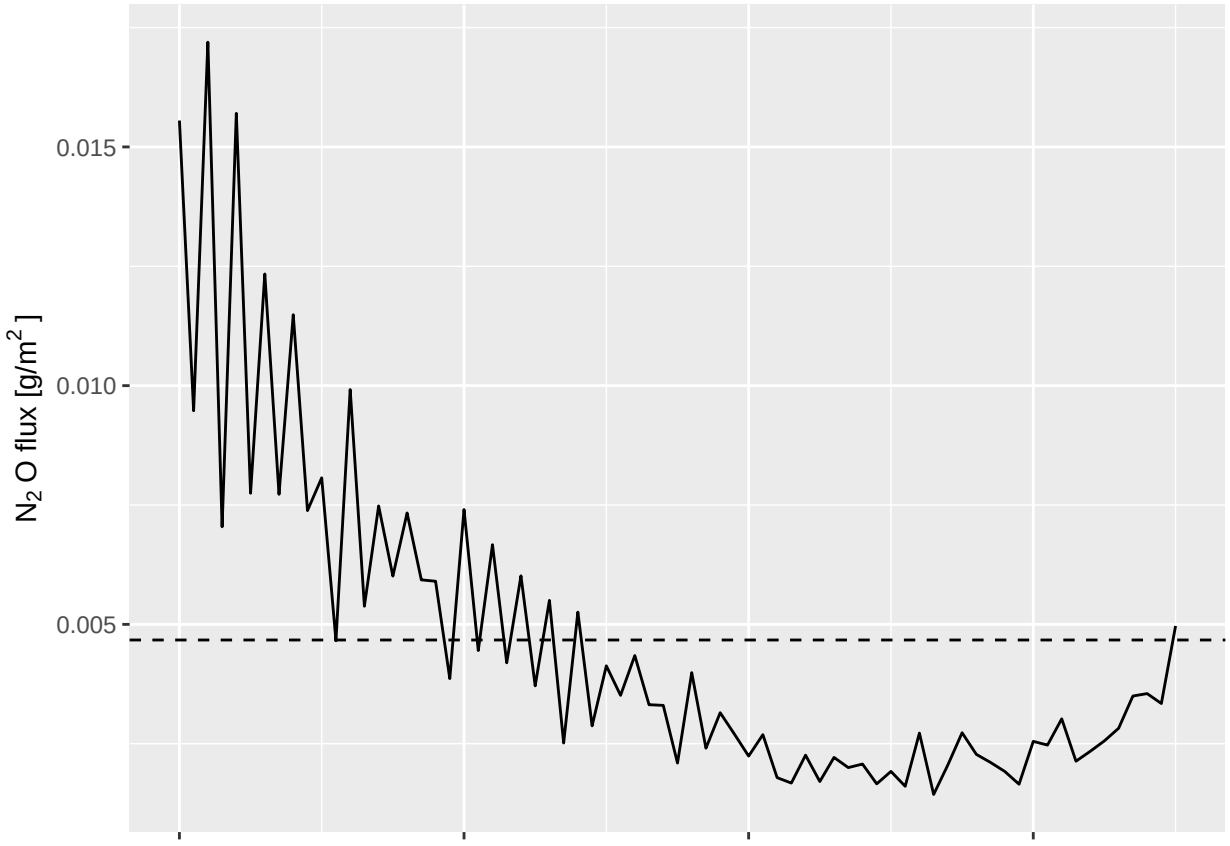
```
avePpt <- mean (yearSummary$ANNPPT_mm, na.rm = TRUE)
pptnPanel <- pptnPanel +
  geom_hline (yintercept = avePpt, linetype = "dashed")

pptnPanel
```



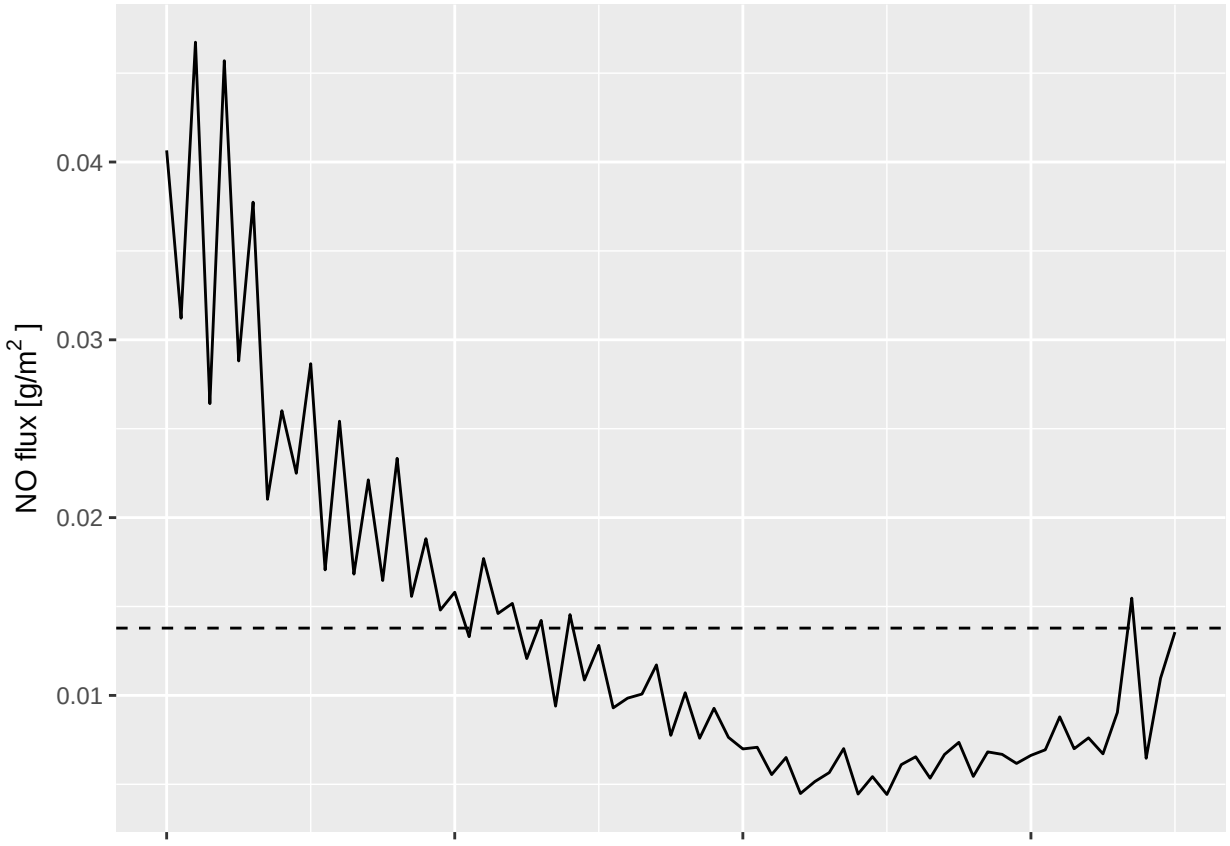
(Have a think about how to add standard errors to show confidence intervals on the mean.) Of course, we could have added the geom_hline as part of the original definition, but you now see the flexibility of this approach.

We can then follow through the same process to produce panels for the other variables:

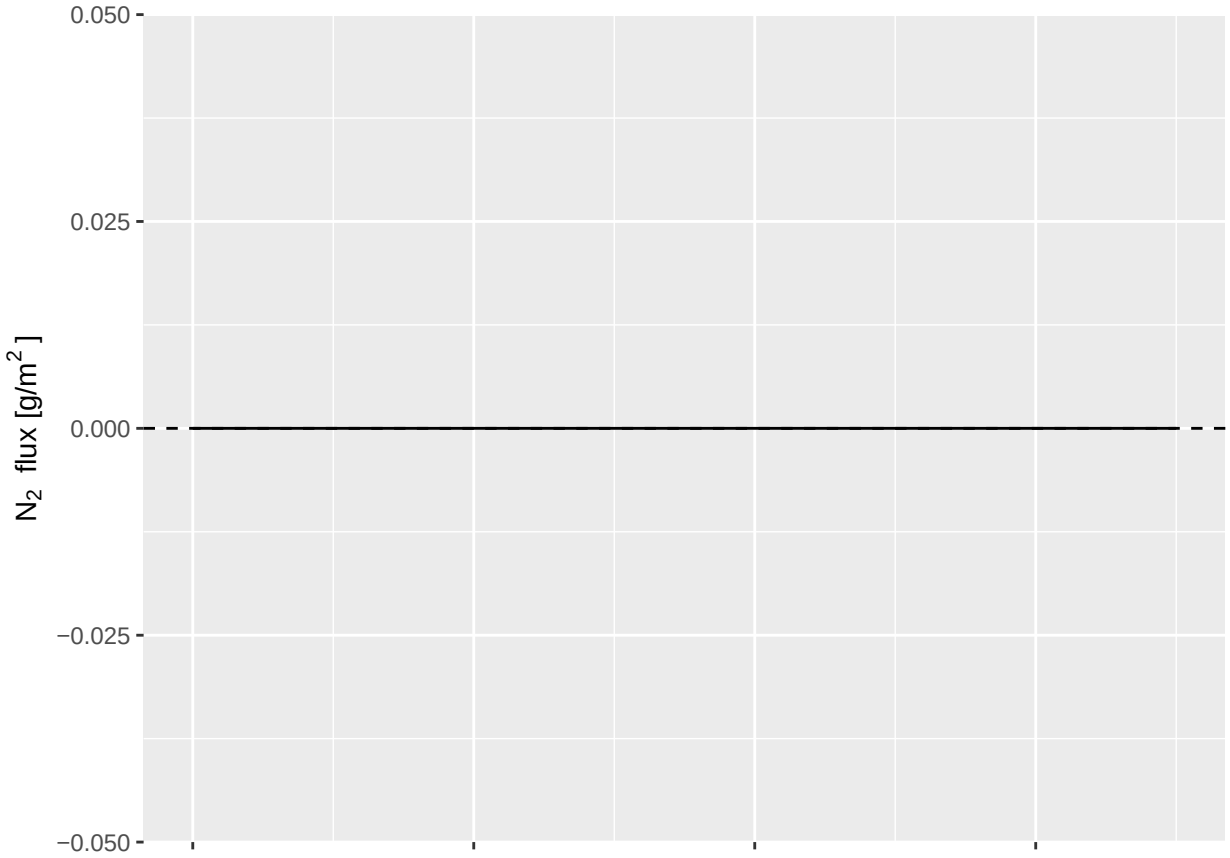
```
aveN2O <- mean (yearSummary$N2Oflux, na.rm = TRUE)

N2OPanel <- ggplot (data = yearSummary) +
  geom_line (aes (x = time, y = N2Oflux)) +
  geom_hline (yintercept = aveN2O, linetype = "dashed") +
  labs (y = expression ("N"[2]~"O flux [g/m"^2~"]")) +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

N2OPanel
```



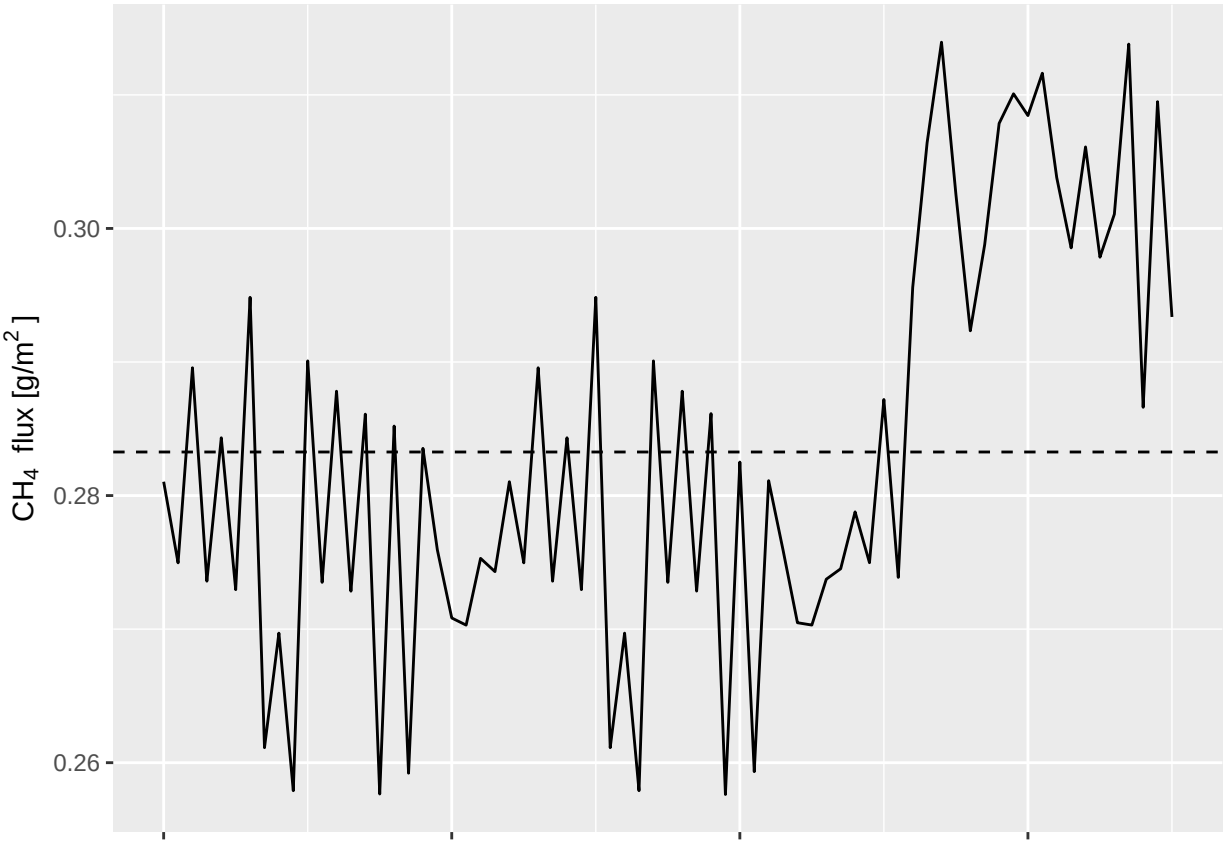

```
aveNO <- mean (yearSummary$NOflux, na.rm = TRUE)

NOPanel <- ggplot (data = yearSummary) +
  geom_line (aes (x = time, y = NOflux)) +
  geom_hline (yintercept = aveNO, linetype = "dashed") +
  labs (y = expression ("NO flux [g/m"^2~"]")) +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

NOPanel
```



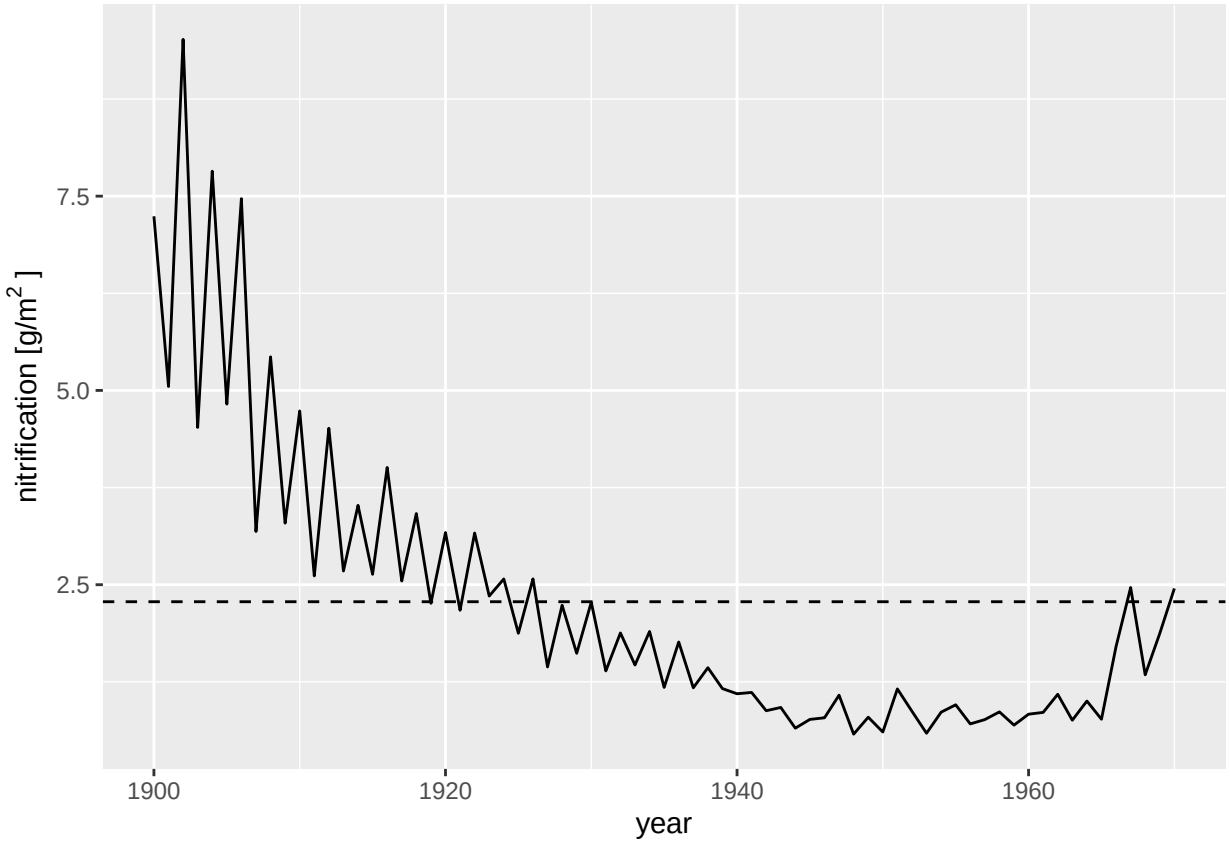

```
aveN2 <- mean (yearSummary$N2flux, na.rm = TRUE)

N2Panel <- ggplot (data = yearSummary) +
  geom_line (aes (x = time, y = N2flux)) +
  geom_hline (yintercept = aveN2, linetype = "dashed") +
  labs (y = expression ("N"[2]~" flux [g/m"^2~"]")) +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

N2Panel
```




```
aveCH4 <- mean (yearSummary$CH4, na.rm = TRUE)

CH4Panel <- ggplot (data = yearSummary) +
  geom_line (aes (x = time, y = CH4)) +
  geom_hline (yintercept = aveCH4, linetype = "dashed") +
  labs (y = expression ("CH"[4]~" flux [g/m"^2~"]")) +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

CH4Panel
```

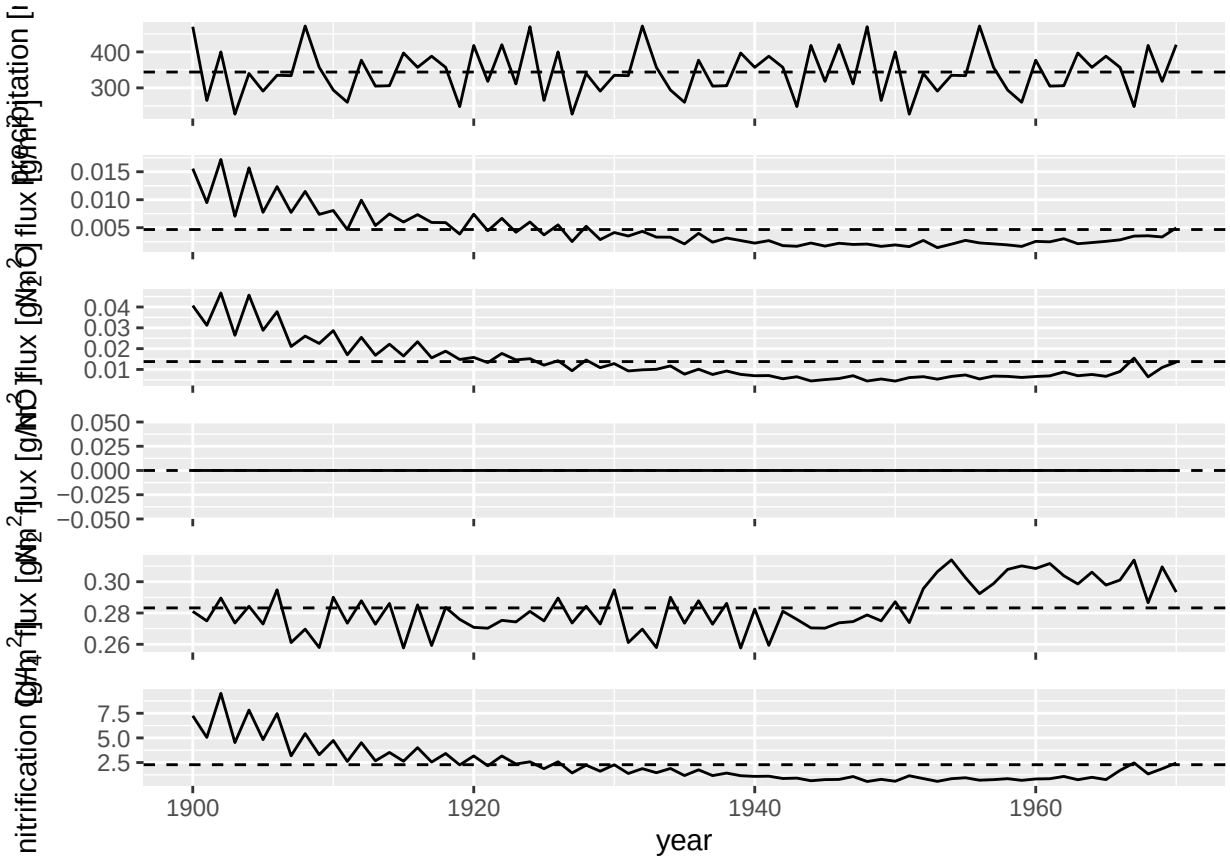



```
aveNIT <- mean (yearSummary$NIT, na.rm = TRUE)

NITPanel <- ggplot (data = yearSummary) +
  geom_line (aes (x = time, y = NIT)) +
  geom_hline (yintercept = aveNIT, linetype = "dashed") +
  labs (x = "year",
        y = expression ("nitrification [g/m"^2~"]"))

NITPanel
```



Notice for the last panel I have added the x-axis information. Now we could copy and paste all of these images into our favourite graphics editor to stack them up, but it's much easier to do it directly in R. There are two steps: first we create an object that has all of our panels as a set of graphics objects (Grobs), and then we use the grid package to plot it.

```
allPanels <- rbind (ggplotGrob (pptnPanel),
                    ggplotGrob (N2OPanel),
                    ggplotGrob (NOPanel),
                    ggplotGrob (N2Panel),
                    ggplotGrob (CH4Panel),
                    ggplotGrob (NITPanel),
                    size = "max")
grid.newpage ()
grid.draw (allPanels)
```



The y axis labels overwrite, but if you use the function windows() and then plot in a larger window, this issue goes away (so we don't have to tweak font sizes). Do this in the console if your window vanishes as soon as it's finished:

```
windows ()
grid.newpage ()
grid.draw (allPanels)
```

Even better, we don't have to copy and paste or save the final window, we can save the file directly from R:

```
ggsave (filename = "./yearSummary.png", allPanels,
        width = 210, height = 250, units = "mm")
```

You can save in different formats just by changing the filename extension ("eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (on windows only) are the different options).

We now have a publication-quality reproducible output of our results.

In terms of data management, it might be better to save our results (e.g. the image) into a separate Results folder within the project, so you have versions of the script, input data, and output results all in separate places.

At Last! A(nother) Benefit of Coding

So far we have used code that has read in data and plotted them out. We can read in data from a new file and use the same code if the data frame name is the same. But what if we want to make the process much more flexible and produce output for any input that has the right format? This is where being able to define our own functions comes into play.

We define a function as follows. This very simple example takes two input values and adds them together:

```
simpleFunction <- function (a, b){
  answer <- a + b
  return (answer)
}
```

Nothing seems to have happened – but if you look over in the Environment pane in RStudio (top right), you should see your simpleFunction now appears as being available to you. We can now call this as a function in the same way as any other:

```
simpleFunction (1, 2)
```

```
## [1] 3
```

It's not a very good function as it's easy to break:

```
# this call WILL produce an error
simpleFunction ("fish", "bananas")
```

```
## Error in a + b: non-numeric argument to binary operator
```

but it shows the principle: we pass input values in brackets to the function definition, and use return to return a value to the user. That returned value can be used in the same way as other variables:

```
addAB <- simpleFunction (1, 2)

addAB
```

```
## [1] 3
```

So let's make a more complicated function where we collect all the panel plots to automate the process, starting from reading the data to saving the file:

```
DayCentAnnSummPlot <- function (filePath = "./",
                                fileName = "year_summary.out",
                                outPath = "./",
                                outFile = "yearSummary.png"){

  yearSummary <- read.table (paste0 (filePath, fileName),
                             header = TRUE, sep = "")
  yearSummary <- yearSummary %>%
    mutate (ANNPPT_mm = ANNPPT * 10.)

  avePpt <- mean (yearSummary$ANNPPT_mm, na.rm = TRUE)

  pptnPanel <- ggplot (data = yearSummary) +
    geom_line (aes (x = time, y = ANNPPT_mm)) +
    geom_hline (yintercept = avePpt, linetype = "dashed") +
    labs (y = "precipitation [mm]") +
    theme (axis.title.x = element_blank (),
           axis.text.x = element_blank ())

  aveN2O <- mean (yearSummary$N2Oflux, na.rm = TRUE)


  N2OPanel <- ggplot (data = yearSummary) +
    geom_line (aes (x = time, y = N2Oflux)) +
    geom_hline (yintercept = aveN2O, linetype = "dashed") +
    labs (y = expression ("N"[2]~"O flux [g/m"^2~"]")) +
```

```
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

aveNO <- mean (yearSummary$NOflux, na.rm = TRUE)

NOPanel <- ggplot (data = yearSummary) +
  geom_line (aes (x = time, y = NOflux)) +
  geom_hline (yintercept = aveNO, linetype = "dashed") +
  labs (y = expression ("NO flux [g/m"^2~"]")) +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

aveN2 <- mean (yearSummary$N2flux, na.rm = TRUE)

N2Panel <- ggplot (data = yearSummary) +
  geom_line (aes (x = time, y = N2flux)) +
  geom_hline (yintercept = aveN2, linetype = "dashed") +
  labs (y = expression ("N"[2]~" flux [g/m"^2~"]")) +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

aveCH4 <- mean (yearSummary$CH4, na.rm = TRUE)

CH4Panel <- ggplot (data = yearSummary) +
  geom_line (aes (x = time, y = CH4)) +
  geom_hline (yintercept = aveCH4, linetype = "dashed") +
  labs (y = expression ("CH"[4]~" flux [g/m"^2~"]")) +
  theme (axis.title.x = element_blank (),
         axis.text.x = element_blank ())

aveNIT <- mean (yearSummary$NIT, na.rm = TRUE)

NITPanel <- ggplot (data = yearSummary) +
  geom_line (aes (x = time, y = NIT)) +
  geom_hline (yintercept = aveNIT, linetype = "dashed") +
  labs (x = "year",
        y = expression ("nitrification [g/m"^2~"]"))

allPanels <- rbind (ggplotGrob (pptnPanel),
                    ggplotGrob (N2OPanel),
                    ggplotGrob (NOPanel),
                    ggplotGrob (N2Panel),
                    ggplotGrob (CH4Panel),
                    ggplotGrob (NITPanel),
                    size = "max")
windows ()
grid.newpage ()
grid.draw (allPanels)

ggsave (filename = paste0 (outPath, outFile), allPanels,
        width = 210, height = 250, units = "mm")

summaryOutput <- data.frame ("Ave precipitation" = avePpt,
```

```
                                "Ave N2O flux" = aveN2O,
                                "Ave NO flux" = aveNO,
                                "Ave N2 flux" = aveN2,
                                "Ave CH4" = aveCH4,
                                "Ave nitrification" = aveNIT)

  return (summaryOutput)

}
```

Then call the function with the relevant parameters (note that I've cunningly used the default names specified in the function for all but the folder name, so that's the only thing specified here). I've also provided a simple data frame outputting the average statistics.

```
DayCentAnnSummPlot (filePath = dayCentFolder)
```

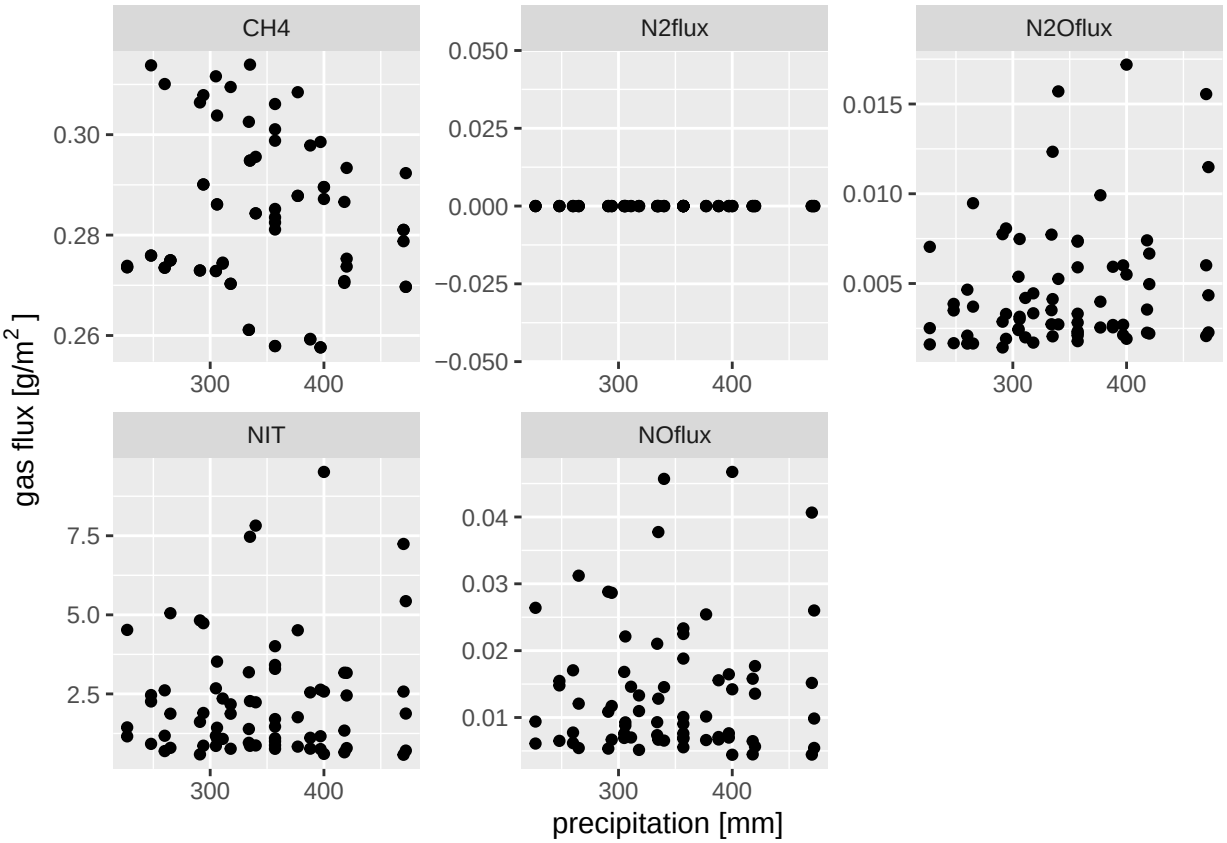
```
##   Ave.precipitation Ave.N2O.flux Ave.NO.flux Ave.N2.flux   Ave.CH4
## 1          344.0845   0.00467231  0.01378585           0 0.2832625
##   Ave.nitrification
## 1          2.281064
```

Now copy the code for this function and paste it into a new R script file called DayCentVis.R (with a .R extension, NOT an Rmd file). Then all you need to do to be able to use the function in any other code is to add:

```
# DON'T run this -- it won't work as you need to specify the path name properly
#  and you need to have generated the DayCentVis.R file
source ("Path/To/Script/File/On/Your/Computer/DayCentVis.R")
```

Before we leave this dataset, let's have a quick think about what else we might want to do with it. For example, we might want to see if there are patterns in the link between precipitation and gas flux rather than just plotting results as time series. Again we can plot variable by variable, or use facet_wrap in ggplot to simplify things:

```
yearSummary %>%
  select (c (-ANNPPT, -time)) %>%
  pivot_longer (-ANNPPT_mm, names_to = "variable") %>%
  ggplot () +
    geom_point (aes (x = ANNPPT_mm, y = value)) +
    facet_wrap (~variable, scales = "free") +
    labs (x = "precipitation [mm]",
          y = expression ("gas flux [g/m"^2~"]"))
```



Note the approach again not to change the dataset, but we take the dataframe, pass it through a couple of formatting statements from tidyverse and then plot.

What About the Other Output Files?

The file bio.out contains the daily above and below ground live carbon. Let's read it in and take a look:

```
bioFile <- "bio.out"

bioData <- read.table (paste0 (dayCentFolder, bioFile),
                       header = TRUE, sep = "")

head (bioData)
```

```
##   time dayofyr aglivc bglivcj bglivcm aglivn bglivnj bglivnm rleavc frootcj
## 1 1900       1  5.078  4.8254  0.4143 0.2482  0.1072  0.0092      0       0
## 2 1900       2  5.078  4.8074  0.4139 0.2482  0.1068  0.0092      0       0
## 3 1900       3  5.078  4.7896  0.4134 0.2482  0.1064  0.0092      0       0
## 4 1900       4  5.078  4.7718  0.4130 0.2482  0.1060  0.0092      0       0
## 5 1900       5  5.078  4.7540  0.4125 0.2482  0.1056  0.0092      0       0
## 6 1900       6  5.078  4.7363  0.4120 0.2482  0.1053  0.0092      0       0
##   frootcm fbrchc rlwodc crootc h2ogef.1. h2ogef.2.
## 1       0      0      0      0  0.355837         0
## 2       0      0      0      0  0.349164         0
## 3       0      0      0      0  0.346768         0
## 4       0      0      0      0  0.337498         0
## 5       0      0      0      0  0.313377         0
## 6       0      0      0      0  0.283331         0
```

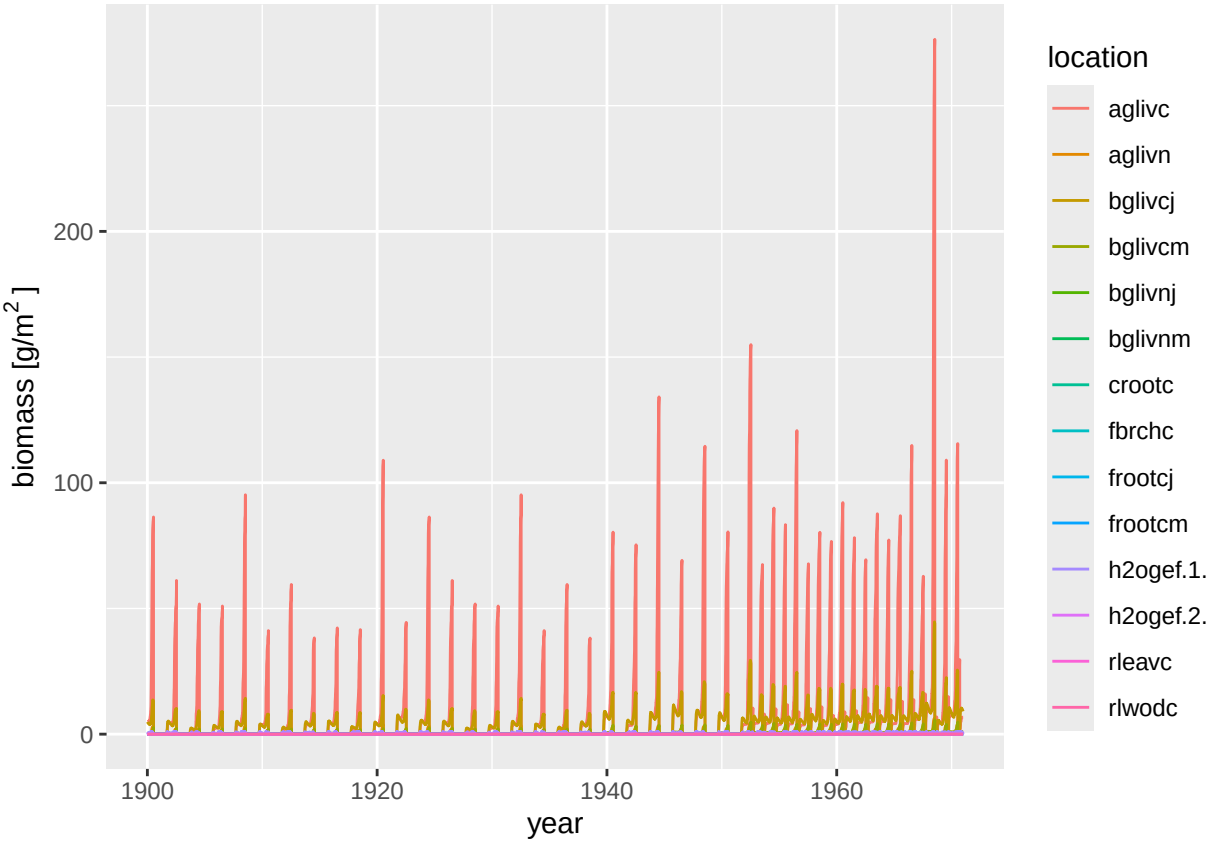
And then look at the summary values:

```
summary (bioData)
```

```
##       time         dayofyr          aglivc           bglivcj
##  Min.   :1900   Min.   :  1.0   Min.   :  0.000   Min.   : 0.000
##  1st Qu.:1917   1st Qu.: 92.0   1st Qu.:  0.000   1st Qu.: 0.000
##  Median :1935   Median :183.0   Median :  2.015   Median : 2.321
##  Mean   :1935   Mean   :183.1   Mean   :  9.859   Mean   : 3.769
##  3rd Qu.:1953   3rd Qu.:274.0   3rd Qu.:  6.617   3rd Qu.: 5.902
##  Max.   :1970   Max.   :366.0   Max.   :276.387   Max.   :44.657
##     bglivcm           aglivn          bglivnj           bglivnm
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.000000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.000000
##  Median :0.1882   Median :0.0947   Median :0.05050   Median :0.004100
##  Mean   :0.4307   Mean   :0.3764   Mean   :0.08077   Mean   :0.009181
##  3rd Qu.:0.6099   3rd Qu.:0.2926   3rd Qu.:0.12510   3rd Qu.:0.012700
##  Max.   :5.6520   Max.   :6.4123   Max.   :0.92140   Max.   :0.118700
##      rleavc     frootcj     frootcm      fbrchc      rlwodc      crootc
##  Min.   :0   Min.   :0   Min.   :0   Min.   :0   Min.   :0   Min.   :0
##  1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0
##  Median :0   Median :0   Median :0   Median :0   Median :0   Median :0
##  Mean   :0   Mean   :0   Mean   :0   Mean   :0   Mean   :0   Mean   :0
##  3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0
##  Max.   :0   Max.   :0   Max.   :0   Max.   :0   Max.   :0   Max.   :0
##    h2ogef.1.        h2ogef.2.
##  Min.   :0.0000   Min.   :0
##  1st Qu.:0.0000   1st Qu.:0
##  Median :0.1525   Median :0
##  Mean   :0.1919   Mean   :0
##  3rd Qu.:0.1999   3rd Qu.:0
##  Max.   :0.9963   Max.   :0
```

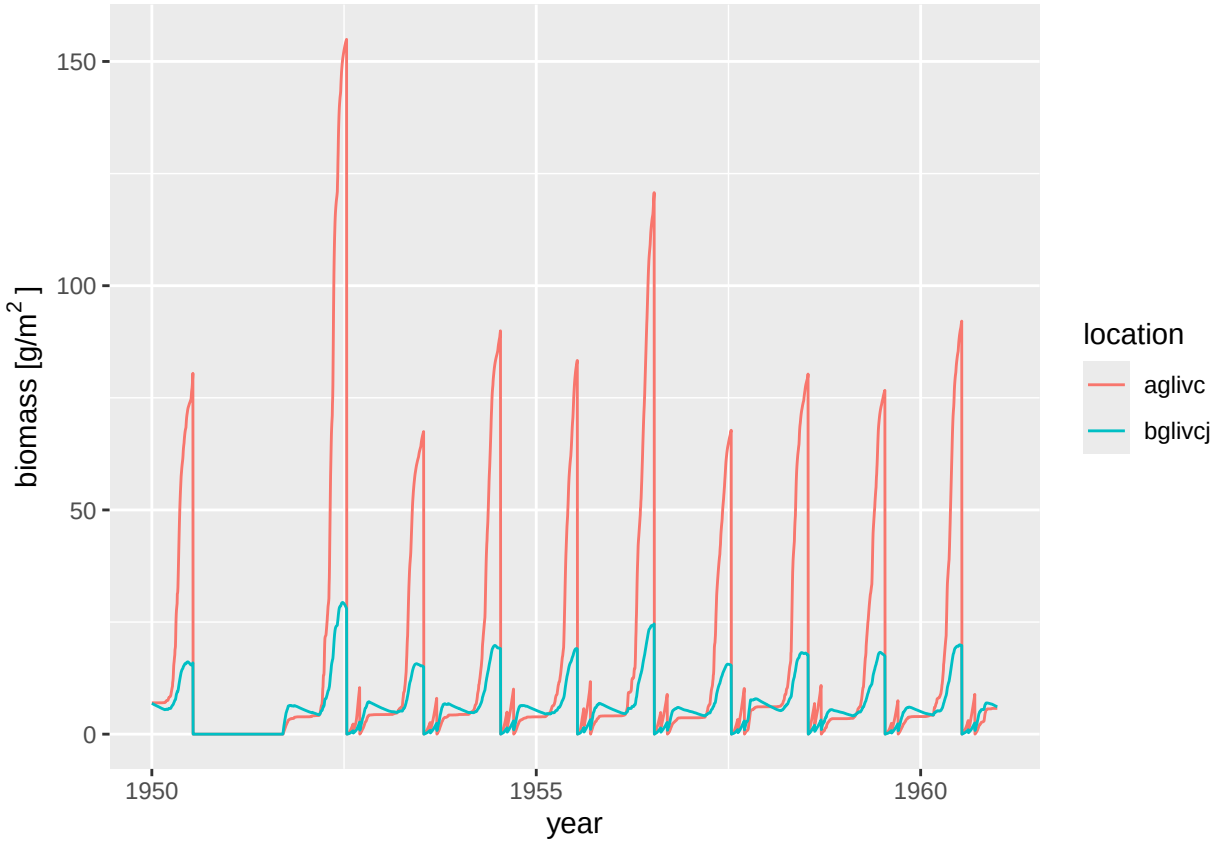
Let's just plot out the aboveground and belowground carbon through time and use a simple fix to give us a continuous time variable:

```
bioData %>%
  mutate(date = as.Date(dayofyr - 1, origin = paste0(time, "-01-01"))) %>%
  select (-dayofyr, -time) %>%
  pivot_longer(-date, names_to = "variable")  %>%
  ggplot () +
    geom_line (aes (x = date, y = value, colour = variable)) +
    labs (x = "year",
          y = expression ("biomass [g/m"^2~"]"),
          colour = "location")
```



We can also specify a subset of the data to look at, for example to see what happens between 1950 and 1960:

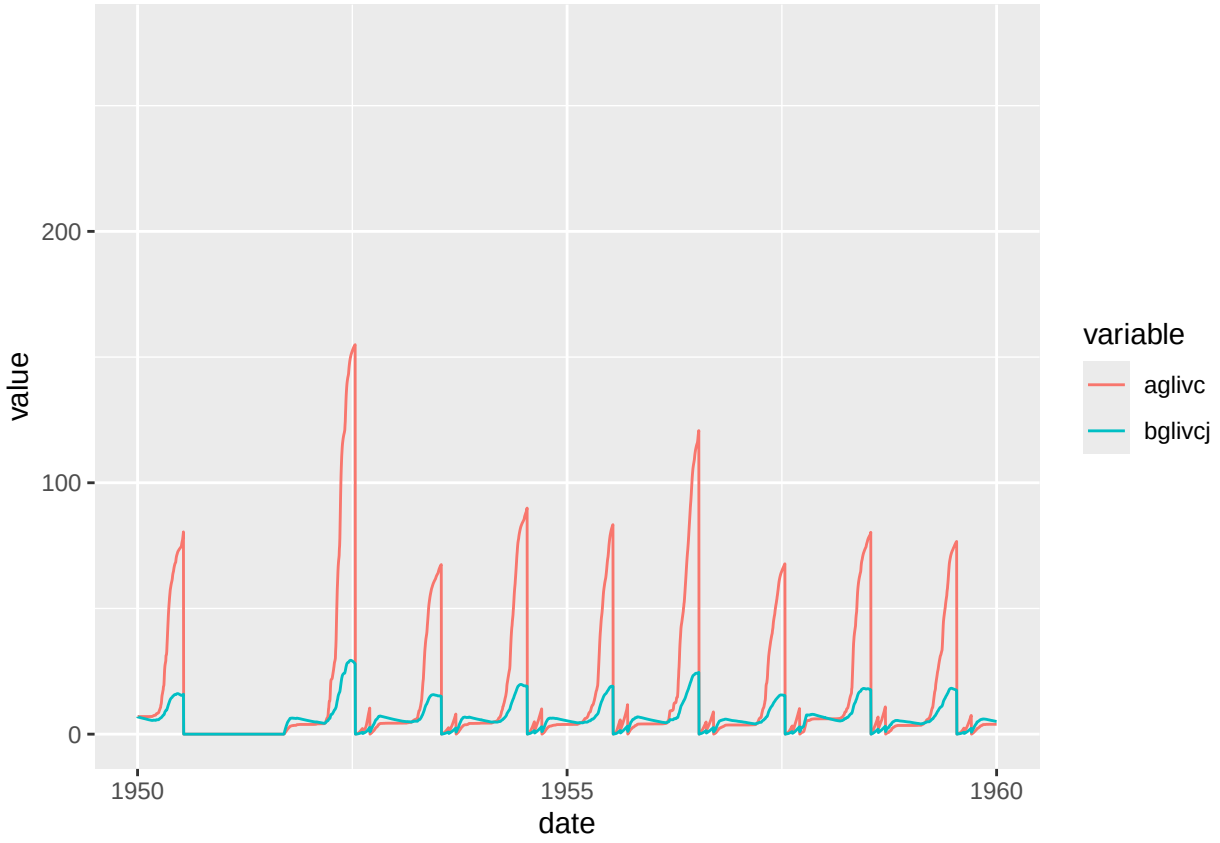
```
bioData %>%
  select (c (time, dayofyr, aglivc, bglivcj)) %>%
  filter (time >= 1950 & time <= 1960) %>%
  mutate(date = as.Date(dayofyr - 1, origin = paste0(time, "-01-01"))) %>%
  select (-dayofyr, -time) %>%
  pivot_longer (-date, names_to = "variable") %>%
  ggplot () +
    geom_line (aes (x = date, y = value, colour = variable)) +
    labs (x = "year",
          y = expression ("biomass [g/m"^2~"]"),
          colour = "location")
```



We also subset by specifying the x axis limits:

```
bioData %>%
  select (c (time, dayofyr, aglivc, bglivcj)) %>%
  mutate(date = as.Date(dayofyr - 1, origin = paste0(time, "-01-01"))) %>%
  select (-dayofyr, -time) %>%
  pivot_longer (-date, names_to = "variable") %>%
  ggplot () +
    geom_line (aes (x = date, y = value, colour = variable)) +
    xlim(as.Date(c("1950-01-01", "1960-01-01")))
```

```
## Warning: Removed 44558 rows containing missing values or values outside the scale range
## (`geom_line()`).
```




```
    labs (x = "date",
          y = expression ("biomass [g/m"^2~"]"),
          colour = "location")
```

```
## $x
## [1] "date"
##
## $y
## expression("biomass [g/m"^2 ~ "]")
##
## $colour
## [1] "location"
##
## attr(,"class")
## [1] "labels"
```

What causes these graphs to be different?

Consider how you might turn this code into a function in a separate script.

Finally, let's look at some daily date from the vswc.out (daily volumetric soilwater content by layer) file:

```
vswcFile <- "vswc.out"

vswcData <- read.table (paste0 (dayCentFolder, vswcFile),
                        header = FALSE, sep = "")

head (vswcData)
```

```
##     V1 V2     V3     V4     V5     V6     V7     V8     V9    V10    V11    V12
## 1 1900  1 0.1275 0.1165 0.1258 0.1270 0.1292 0.1296 0.1290 0.1289 0.1290 0.1292
## 2 1900  2 0.1268 0.1160 0.1249 0.1258 0.1281 0.1293 0.1291 0.1289 0.1290 0.1292
## 3 1900  3 0.1267 0.1159 0.1245 0.1250 0.1272 0.1288 0.1291 0.1290 0.1291 0.1292
## 4 1900  4 0.1245 0.1157 0.1233 0.1243 0.1265 0.1283 0.1289 0.1290 0.1291 0.1292
## 5 1900  5 0.1158 0.1180 0.1197 0.1224 0.1255 0.1278 0.1288 0.1290 0.1291 0.1292
## 6 1900  6 0.0958 0.1183 0.1191 0.1212 0.1243 0.1272 0.1285 0.1289 0.1291 0.1292
##      V13    V14    V15
## 1 0.1293 0.1293 0.1291
## 2 0.1293 0.1293 0.1291
## 3 0.1293 0.1293 0.1291
## 4 0.1292 0.1292 0.1291
## 5 0.1292 0.1292 0.1291
## 6 0.1292 0.1292 0.1291
```

Note that this file does NOT have a header, because the number of soil layers depends on the input parameters. We can add variable names back into our data frame (and do it automatically so we don't need to change the code if the number of layers changes in the input files):

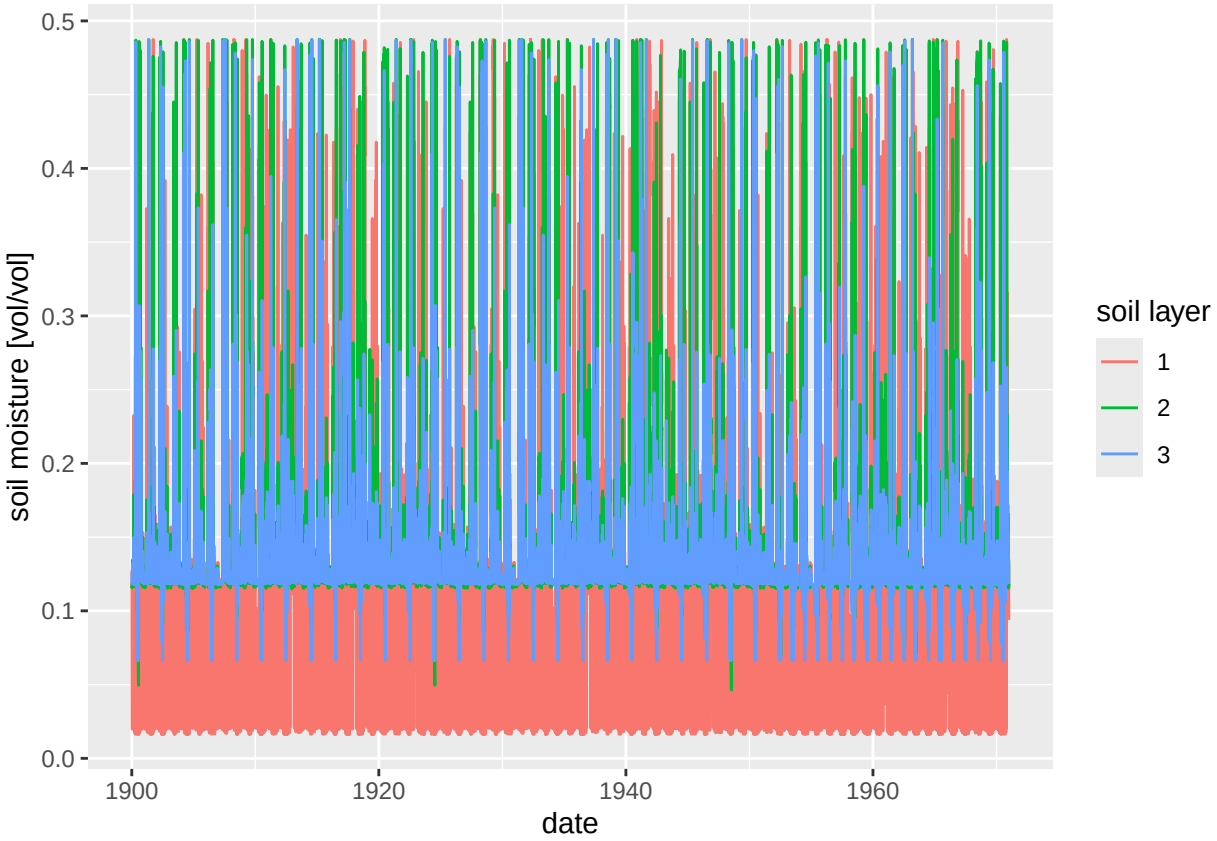
```
varNames <- c ("time", "day")
for (layerNo in seq (3, ncol (vswcData))){
  varNames <- c (varNames, paste0 ("layer", layerNo - 2))
}
names (vswcData) <- varNames
```

The day gives the day of the year (Julian day) and we can use part of the lubridate package to give the actual date in each case:

```
vswcData <- vswcData %>%
  mutate (date = as.Date (day - 1, origin = paste0 (floor (time), "-01-01")))
```

And use that date value for plotting:

```
ggplot (data = vswcData) +
  geom_line (aes (x = date, y = layer1, colour = "1")) +
  geom_line (aes (x = date, y = layer2, colour = "2")) +
  geom_line (aes (x = date, y = layer3, colour = "3")) +
  labs (x = "date",
        y = "soil moisture [vol/vol]",
        colour = "soil layer"  )
```



Again, we could think about plotting as separate panels and turning the resulting code into a reuseable function.