AOV tutorial

S. Cook

September 26, 2018

Package import

You could do all of this in base-R, but trying to code in R without the tidyverse is like trying to eat dinner without your hands - it's possible, but nobody wants to see that. Load-in packages with the `require()` function.

```
require(tidyverse)
```

Data import

We need to get our data into R before we can actually do any work. The `read.csv()` function goes hunting for a `.csv` file with that name and assigns it to the variable `o2_data`. The rest of the stuff inside the `()` just tells the `read.csv()` function that we have headers, and that our file is comma-separated.

```
o2_data <- read.csv(file = "o2_first_exp.csv", header = T, sep = ",")

n2_data <- read.csv(file = "n2_first_exp.csv", header = T, sep = ",")
```

Now that we have `o2_data` imported, it will appear as a variable in the Enrivonment pane (upper right). If you click the blue drop-down, it will give you some information about the type of data stored in each column, and if you click directly on `o2_data` it will open a tab showing you the data in table-format.

Data wrangling

Some functions are easier if we combine the data first. The `rbind()` function is from base R, and throws the `n2_data` rows under the `o2_data` rows. We are only able to use this because our data frames had exactly the same headers. The piping function `%>%` is read as *and then I want to do this*. It is incredibly useful - it makes the code more readable, and if you need to perform a bunch of calculations, this is a good way of stringing together complex operations.

```
gas_combined_wide <- o2_data %>%
  rbind(n2_data)
```

It's a good practice to try and keep your datasets "tidy", meaning that *every row is an observation*, and *every column is a variable* of that observation. `TREATMENT` and `VALUE` give the names of the new columns you are creating, and 4:7 gathers those columns into those variables. The previous column names gets thrown into the first name (`TREATMENT`), and the numbers that fall under those columns get thrown into the second name (`CONC`).

```
gas_combined_long <- gas_combined_wide %>%
  gather("TREATMENT", "CONC", 4:7)
```

Plot it up

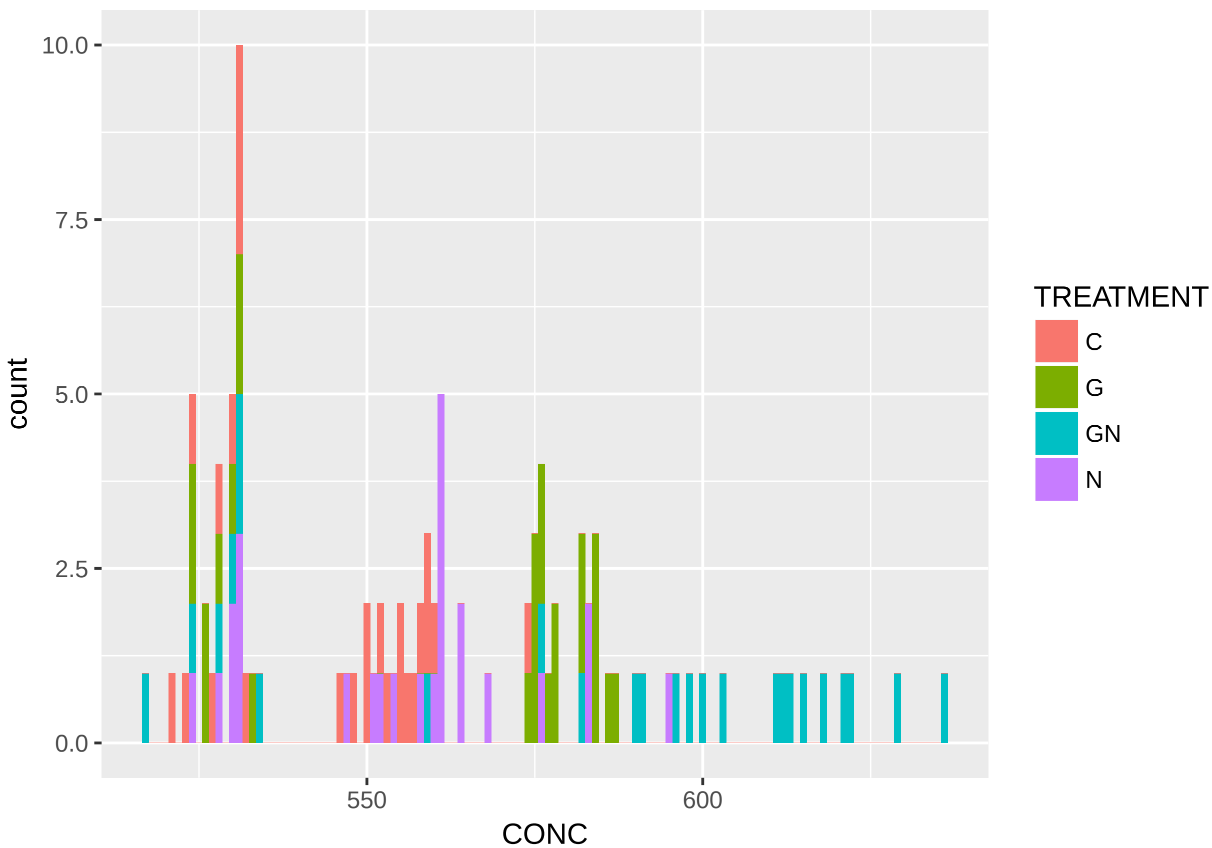
Let's pull out n2 values for analysis.

```
n2_long <- gas_combined_long %>%
  filter(ANALYTE == "n2")
```

It is usually a good idea to plot the raw data, and histograms are very useful to see how the data are looking (not to mention a good way to catch entry errors). The goal is to get an object called `n2_histo`, so everything after `<-` is what goes into creating `n2_histo`. `ggplot()` initializes the plot, and we can then start throwing layers on the plot (ggplot is a very powerful plotting add-on that lets you build layers on your graph using `geoms`).

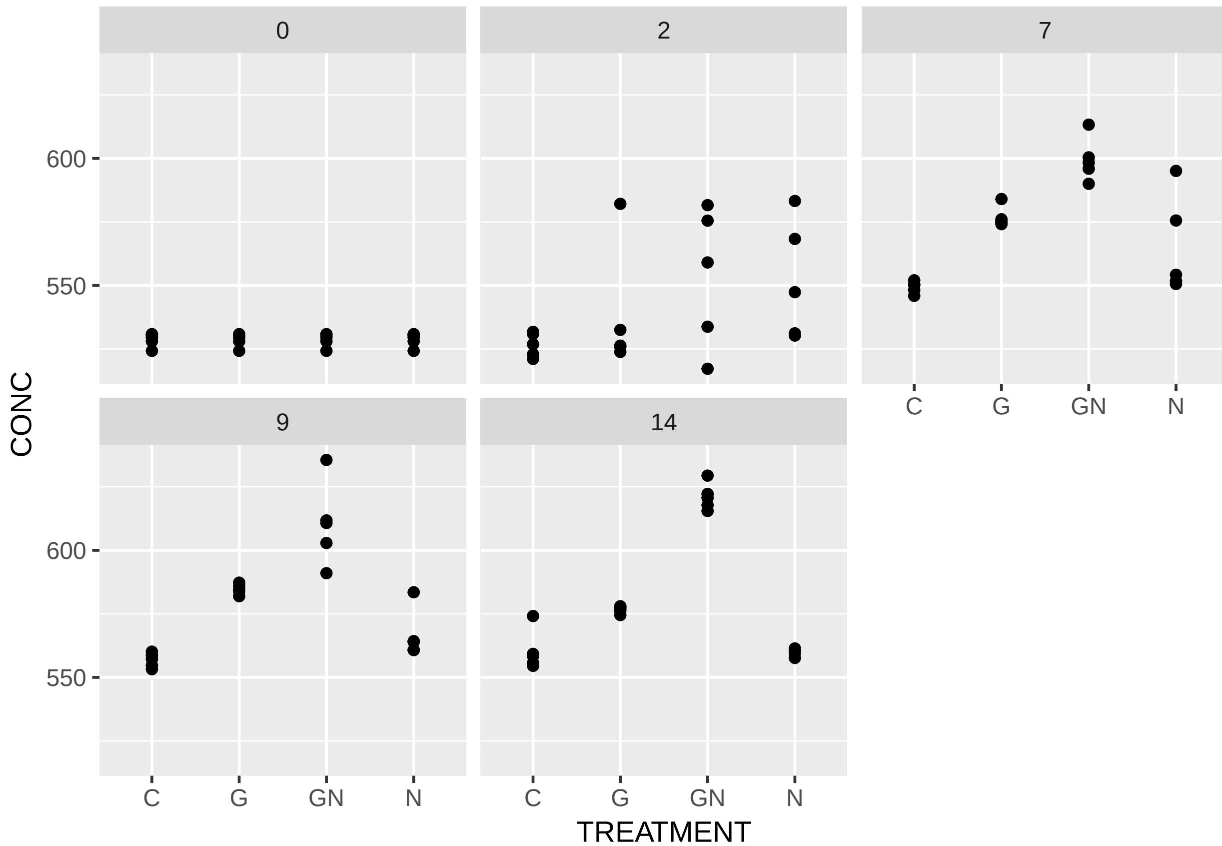
Every geom needs to know where to go to get the data (use the `data =` argument), as well as an aesthetics argument (`aes()`) that needs an x and y value to graph. In the histogram's case, it only needs an x value. The `binwidth` just specifies how large of a window on the x-axis we want to use to count the values.

```
n2_histo <- ggplot() +
  geom_histogram(data = n2_long,
                 aes(x = CONC, fill = TREATMENT),
                 binwidth = 1)

n2_histo
```



Dot-plots (called using `geom_point()`) allow you to look at the actual data points organized by treatment and sampling time. There are numerous ways to display this data, but below sticks treatment on the x-axis and the concentration of n2 on the y-axis, and then uses `facet_wrap()` to divide all the plots by sampling time. You could easily swap `TREATMENT` and `TIME` to emphasize different aspects about these relationships.

```r
n2_plot <- ggplot() +
  geom_point(data = n2_long,
             aes(x = TREATMENT, y = CONC)) +
  facet_wrap(~TIME)

n2_plot
```



2-way analysis of variance

Also called the "one of these things is unlike the other" test. The data is already `tidy`, so we can immediately analyze the data using a 2-factor analysis of variance using the `aov()` function. It needs to know where to look for the data in a dataframe (`data =`). Read the `~` sign as "a function of". We want to model the n2 concentrations *as a function* of `TREATMENT` and `TIME`.

The `as.factor()` function makes sure that `TREATMENT` and `TIME` are both treated as factors.

```r
n2_anova_1 <- aov(data = n2_long,
                  CONC ~ as.factor(TREATMENT) + as.factor(TIME))

summary(n2_anova_1)
```

```
##                      Df Sum Sq Mean Sq F value   Pr(>F)    
## as.factor(TREATMENT)  3  19625    6542   27.62 7.94e-13 ***
## as.factor(TIME)       4  44282   11070   46.74  < 2e-16 ***
## Residuals            92  21788     237
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Above we used a + sign to simply test for the effects of TREATMENT and TIME, which was a good first step, but does not account for any interactive effects between TREATMENT and TIME. We tell the model we want to test for that by switching the + to a *.

```
n2_anova_2 <- aov(data = n2_long,
                  CONC ~ as.factor(TREATMENT)*as.factor(TIME))

summary(n2_anova_2)
```

```
##                                      Df Sum Sq Mean Sq F value   Pr(>F)
## as.factor(TREATMENT)                  3  19625    6542  45.105  < 2e-16
## as.factor(TIME)                       4  44282   11070  76.329  < 2e-16
## as.factor(TREATMENT):as.factor(TIME) 12  10185     849   5.852 3.57e-07
## Residuals                            80  11603     145
##
## as.factor(TREATMENT)                 ***
## as.factor(TIME)                      ***
## as.factor(TREATMENT):as.factor(TIME) ***
## Residuals
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Go google the methodological assumptions of an ANOVA. The best way to test that those assumptions are valid are the plots below. One of the assumptions is that the residuals of the model (may need to google residuals here too) are normally distributed (i.e. should look roughly like a bell curve if you squint your eyes). Use the $ notation to pull out columns in a dataframe.

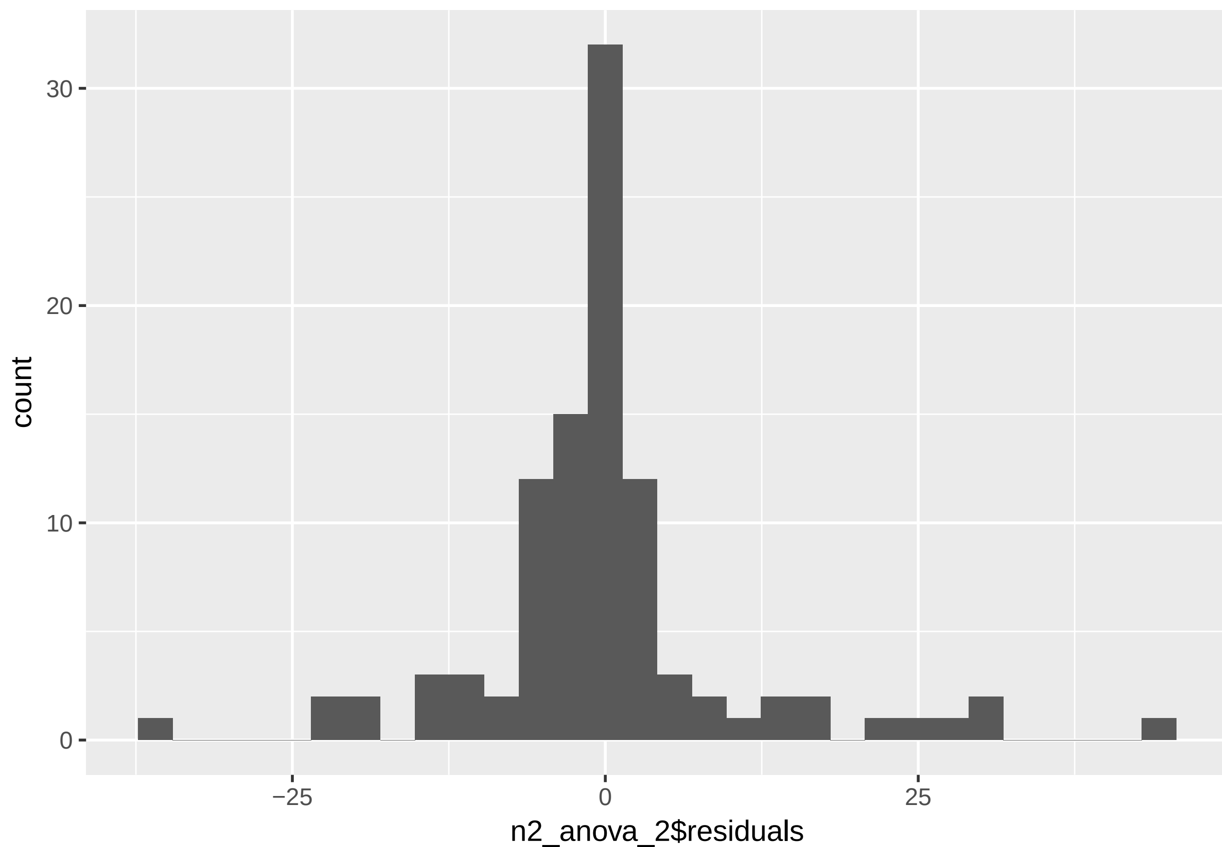
The resid_plot is a histrogram of the residuals, and they do not look all that bad.

Another important assumption is *homoscedasticity of error variances*, which is just an awful way of saying there shouldn't be any noticeable trends in the residuals if we were to plot them against the observed values (the values we plugged into aov() to create the model).

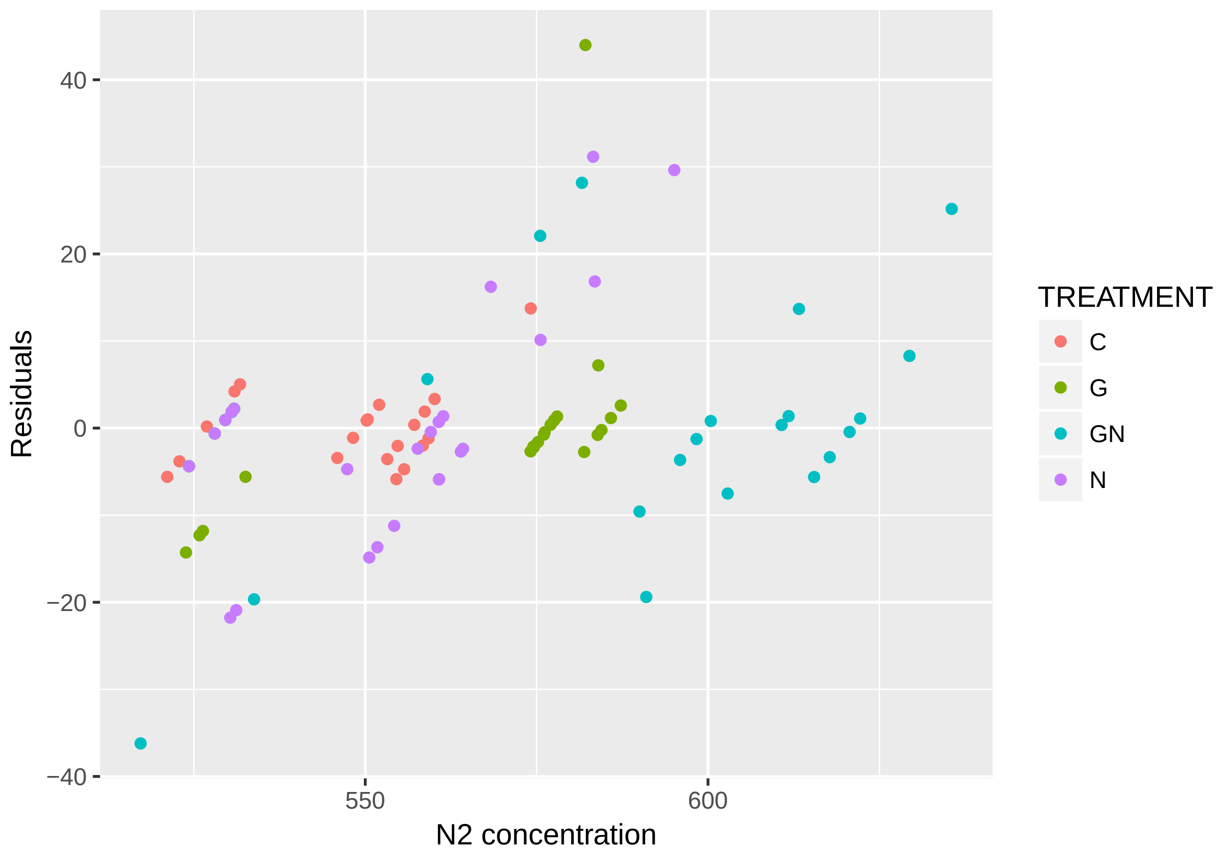
The scatter_plot is an example of this kind of diagnostic plot, and it doesn't look fantastic but nothing to worry about (in a perfect world would look like a starry sky - completely random).

```
combo <- n2_long %>%
  cbind(n2_anova_2$residuals)

resid_plot <- ggplot() +
  geom_histogram(data = combo,
                 aes(x = n2_anova_2$residuals))

resid_plot
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
scatter_plot <- ggplot() +
  geom_point(data = combo,
             aes(x = CONC, y = n2_anova_2$residuals, colour = TREATMENT)) +
  ylab("Residuals") +
  xlab("N2 concentration")

scatter_plot
```



Tukey post-hoc test

Also called the "lets figure out which one of these things is unlike the others" test. If you set up the `aov()` correctly, you should be able to plug it directly into `TukeyHSD()`, which will perform all contrasts between the two predictor variables. Notice that I have *commented out* the actual call to `n2_tukey` with a `#` sign. That is a *comment marker* that tells R I don't want to run that part thank you very much, and it is useful for blocking output or writing a quick note to yourself about what code does.

```
n2_tukey <- TukeyHSD(n2_anova_2)

#n2_tukey
```

If you remove the `#` and look at the output, you'll notice that we get a very large number of comparisons. If you wanted to clean this up a bit, you could `filter()` those p-values by a level of `alpha` that you set before you started the analysis. Walk through the code below and run each part bit by bit. Remember, each `%>%` operator just reads as *and then I want to do this*, so a good way to see what each part is doing is to run each bit step-by-step.

```
n2_contrasts <- as.data.frame(n2_tukey$`as.factor(TREATMENT):as.factor(TIME)`) %>%
  rownames_to_column() %>%
  filter(`p adj` < 0.05)

n2_contrasts
```

```
##      rowname     diff        lwr       upr        p adj
## 1    G:7-C:0 48.13932 20.2222179 76.056413 2.518108e-06
```

```
## 2    GN:7-C:0  70.93410  43.0170018  98.851197 0.000000e+00
## 3     N:7-C:0  36.79337   8.8762708  64.710466 1.029799e-03
## 4     C:9-C:0  28.12642   0.2093228  56.043518 4.621887e-02
## 5     G:9-C:0  56.03063  28.1135306  83.947725 2.712126e-08
## 6    GN:9-C:0  81.72711  53.8100076 109.644202 0.000000e+00
## 7     N:9-C:0  37.98920  10.0721021  65.906297 5.701915e-04
## 8    C:14-C:0  31.74589   3.8287939  59.662989 1.062512e-02
## 9    G:14-C:0  47.99761  20.0805169  75.914712 2.726447e-06
## 10  GN:14-C:0  92.43964  64.5225422 120.356737 0.000000e+00
## 11   N:14-C:0  31.35604   3.4389379  59.273133 1.256367e-02
## 12    G:7-G:0  48.13932  20.2222179  76.056413 2.518108e-06
## 13   GN:7-G:0  70.93410  43.0170018  98.851197 0.000000e+00
## 14    N:7-G:0  36.79337   8.8762708  64.710466 1.029799e-03
## 15    C:9-G:0  28.12642   0.2093228  56.043518 4.621887e-02
## 16    G:9-G:0  56.03063  28.1135306  83.947725 2.712126e-08
## 17   GN:9-G:0  81.72711  53.8100076 109.644202 0.000000e+00
## 18    N:9-G:0  37.98920  10.0721021  65.906297 5.701915e-04
## 19   C:14-G:0  31.74589   3.8287939  59.662989 1.062512e-02
## 20   G:14-G:0  47.99761  20.0805169  75.914712 2.726447e-06
## 21  GN:14-G:0  92.43964  64.5225422 120.356737 0.000000e+00
## 22   N:14-G:0  31.35604   3.4389379  59.273133 1.256367e-02
## 23   G:7-GN:0  48.13932  20.2222179  76.056413 2.518108e-06
## 24  GN:7-GN:0  70.93410  43.0170018  98.851197 0.000000e+00
## 25   N:7-GN:0  36.79337   8.8762708  64.710466 1.029799e-03
## 26   C:9-GN:0  28.12642   0.2093228  56.043518 4.621887e-02
## 27   G:9-GN:0  56.03063  28.1135306  83.947725 2.712126e-08
## 28  GN:9-GN:0  81.72711  53.8100076 109.644202 0.000000e+00
## 29   N:9-GN:0  37.98920  10.0721021  65.906297 5.701915e-04
## 30  C:14-GN:0  31.74589   3.8287939  59.662989 1.062512e-02
## 31  G:14-GN:0  47.99761  20.0805169  75.914712 2.726447e-06
## 32 GN:14-GN:0  92.43964  64.5225422 120.356737 0.000000e+00
## 33  N:14-GN:0  31.35604   3.4389379  59.273133 1.256367e-02
## 34    G:7-N:0  48.13932  20.2222179  76.056413 2.518108e-06
## 35   GN:7-N:0  70.93410  43.0170018  98.851197 0.000000e+00
## 36    N:7-N:0  36.79337   8.8762708  64.710466 1.029799e-03
## 37    C:9-N:0  28.12642   0.2093228  56.043518 4.621887e-02
## 38    G:9-N:0  56.03063  28.1135306  83.947725 2.712126e-08
## 39   GN:9-N:0  81.72711  53.8100076 109.644202 0.000000e+00
## 40    N:9-N:0  37.98920  10.0721021  65.906297 5.701915e-04
## 41   C:14-N:0  31.74589   3.8287939  59.662989 1.062512e-02
## 42   G:14-N:0  47.99761  20.0805169  75.914712 2.726447e-06
## 43  GN:14-N:0  92.43964  64.5225422 120.356737 0.000000e+00
## 44   N:14-N:0  31.35604   3.4389379  59.273133 1.256367e-02
## 45    G:7-C:2  50.08032  22.1632265  77.997421 8.410129e-07
## 46   GN:7-C:2  72.87511  44.9580104 100.792205 0.000000e+00
## 47    N:7-C:2  38.73438  10.8172794  66.651474 3.920313e-04
## 48    C:9-C:2  30.06743   2.1503314  57.984526 2.153384e-02
## 49    G:9-C:2  57.97164  30.0545392  85.888734 8.691412e-09
## 50   GN:9-C:2  83.66811  55.7510162 111.585211 0.000000e+00
## 51    N:9-C:2  39.93021  12.0131107  67.847306 2.128985e-04
## 52   C:14-C:2  33.68690   5.7698025  61.603997 4.480860e-03
## 53   G:14-C:2  49.93862  22.0215255  77.855720 9.115371e-07
## 54  GN:14-C:2  94.38065  66.4635508 122.297746 0.000000e+00
## 55   N:14-C:2  33.29704   5.3799465  61.214141 5.349137e-03
```

```
## 56    G:7-G:2  38.66398  10.7468814  66.581076 4.062344e-04
## 57   GN:7-G:2  61.45876  33.5416653  89.375860 1.103227e-09
## 58    G:9-G:2  46.55529  18.6381941  74.472389 6.093839e-06
## 59   GN:9-G:2  72.25177  44.3346711 100.168866 0.000000e+00
## 60    N:9-G:2  28.51386   0.5967656  56.430961 3.987907e-02
## 61   G:14-G:2  38.52228  10.6051804  66.439375 4.363489e-04
## 62  GN:14-G:2  82.96430  55.0472057 110.881401 0.000000e+00
## 63  GN:7-GN:2  46.14779  18.2306906  74.064886 7.635670e-06
## 64   G:9-GN:2  31.24432   3.3272193  59.161414 1.317674e-02
## 65  GN:9-GN:2  56.94079  29.0236964  84.857891 1.592046e-08
## 66 GN:14-GN:2  67.65333  39.7362310  95.570426 1.826184e-11
## 67   GN:7-N:2  47.50468  19.5875799  75.421775 3.592623e-06
## 68    G:9-N:2  32.60121   4.6841087  60.518304 7.304844e-03
## 69   GN:9-N:2  58.29768  30.3805857  86.214781 7.174083e-09
## 70  GN:14-N:2  69.01022  41.0931203  96.927315 2.622347e-12
## 71   GN:7-C:7  50.23801  22.3209169  78.155112 7.688641e-07
## 72    G:9-C:7  35.33454   7.4174456  63.251641 2.079994e-03
## 73   GN:9-C:7  61.03102  33.1139227  88.948118 1.423598e-09
## 74  GN:14-C:7  71.74355  43.8264573  99.660652 0.000000e+00
## 75   GN:9-G:7  33.58779   5.6706923  61.504887 4.688027e-03
## 76  GN:14-G:7  44.30032  16.3832269  72.217422 2.101351e-05
## 77   N:7-GN:7 -34.14073 -62.0578284  -6.223634 3.637929e-03
## 78   C:9-GN:7 -42.80768 -70.7247765 -14.890582 4.697473e-05
## 79   N:9-GN:7 -32.94490 -60.8619972  -5.027802 6.267437e-03
## 80  C:14-GN:7 -39.18821 -67.1053054 -11.271111 3.113621e-04
## 81  N:14-GN:7 -39.57806 -67.4951614 -11.660967 2.551245e-04
## 82   GN:9-N:7  44.93374  17.0166393  72.850834 1.488046e-05
## 83  GN:14-N:7  55.64627  27.7291739  83.563369 3.394631e-08
## 84   GN:9-C:9  53.60068  25.6835874  81.517782 1.114906e-07
## 85  GN:14-C:9  64.31322  36.3961220  92.230317 1.951878e-10
## 86  GN:14-G:9  36.40901   8.4919142  64.326109 1.241836e-03
## 87   N:9-GN:9 -43.73791 -71.6550029 -15.820808 2.849627e-05
## 88  C:14-GN:9 -49.98121 -77.8983112 -22.064116 8.897462e-07
## 89  G:14-GN:9 -33.72949 -61.6465881  -5.812393 4.394516e-03
## 90  N:14-GN:9 -50.37107 -78.2881672 -22.453972 7.127765e-07
## 91  GN:14-N:9  54.45044  26.5333426  82.367538 6.810963e-08
## 92 GN:14-C:14  60.69375  32.7766509  88.610846 1.739802e-09
## 93 GN:14-G:14  44.44203  16.5249278  72.359123 1.945580e-05
## 94 N:14-GN:14 -61.08360 -89.0007018 -33.166507 1.379727e-09
```

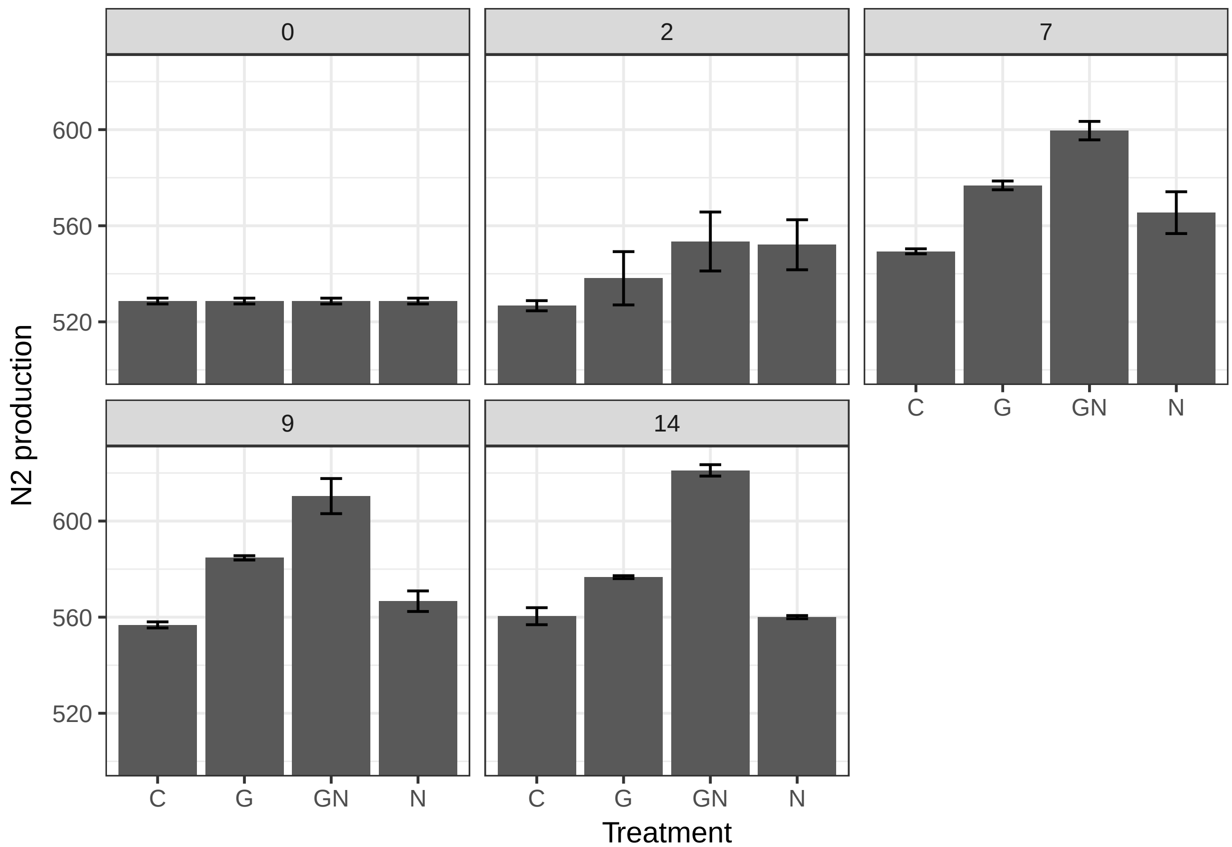
Summarizing data

There are many ways you could visually display the information in `n2_long` (get creative and play around). Below is a way to calculate the group means and standard errors using the `tidyverse`.

```
n2_summary <- n2_long %>%
  group_by(TIME, TREATMENT) %>%
  summarise(N2_MEAN = mean(CONC),
            N2_SE = sd(CONC)/sqrt(n()))
```

`ggplot2` offers the ability to completely customize your plots to get them exactly how you want them to look. The code below is just a small example of the different things you can tweak. If you want to know more about some of those commands, look them up by clicking on the R console window (it might be hidden below), and type `?facet_wrap()`. If you throw a `?` in front of any R function or command, it will pull up a help menu entry on that particular item.

```
n2_plot_treatment <- ggplot() +
  geom_col(data = n2_summary,
           aes(x = TREATMENT, y = N2_MEAN),
           position = "dodge") +
  facet_wrap(~TIME) +
  coord_cartesian(ylim = c(500, 625)) +
  geom_errorbar(data = n2_summary,
                aes(x = TREATMENT,
                    ymin = N2_MEAN - N2_SE,
                    ymax = N2_MEAN + N2_SE),
                width = 0.25) +
  theme_bw() +
  ylab("N2 production") +
  xlab("Treatment")

n2_plot_treatment
```



There are multiple ways to save plots as whatever file type you wish. `.pdf`s ideal for publication, and high-resolution `.png`s are great for powerpoint presentations. You can even save at retina-display resolution using `dpi = "retina"` (for all you weird Mac people out there).

```
#ggsave(n2_plot_treatment, file = "n2_plot_treatment.pdf", device = pdf, height = 6, width = 6)
```