ST517 - Module 1 Lab

This lab explores some tools for manipulating data frames to get group summaries, both graphical and numeric. These tools facilitate exploratory analysis, as part of the process of comparing groups.

To learn these tools, we use the the *Diet Restriction and Longevity* study stored in `Sleuth3` as `case0501`. Run `?case0501` to read more about the study.

```
library(Sleuth3)
head(case0501) # First few rows of data
```

```
##   Lifetime Diet
## 1     35.5   NP
## 2     35.4   NP
## 3     34.9   NP
## 4     34.8   NP
## 5     33.8   NP
## 6     33.5   NP
```

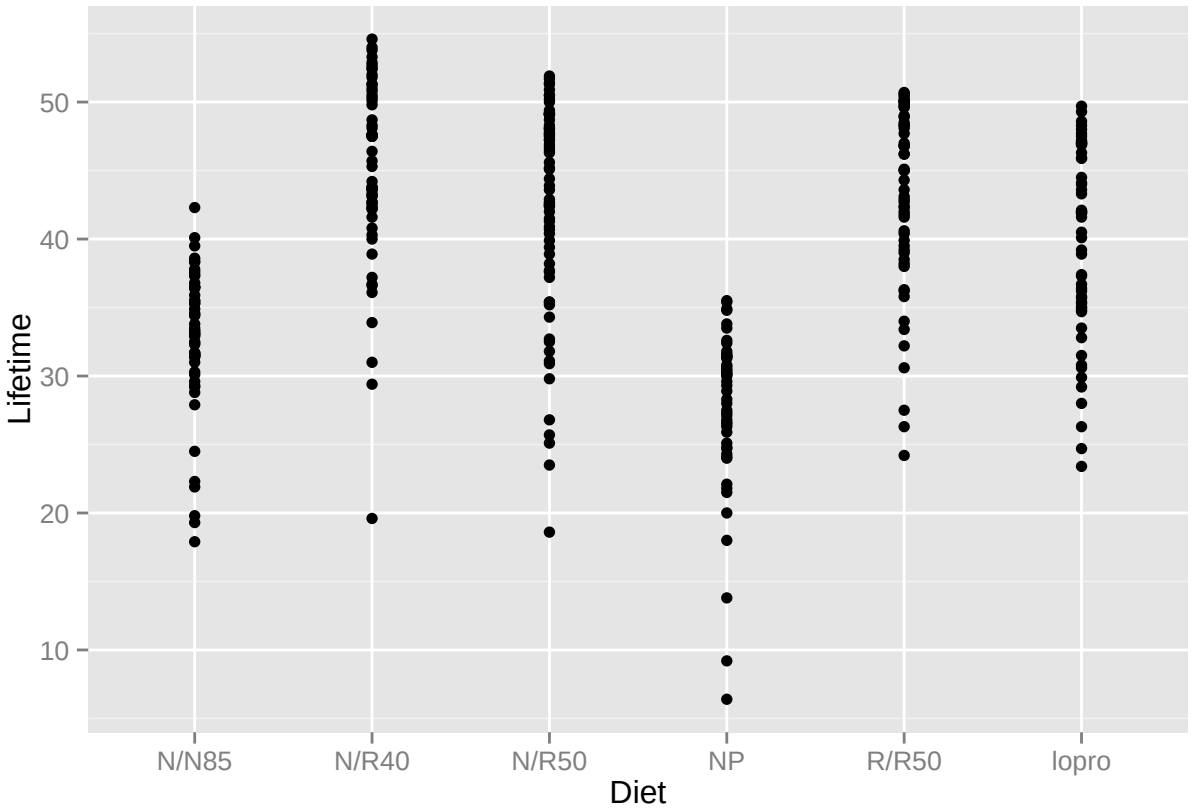
```
str(case0501) # Structure of case0501 data object
```

```
## 'data.frame':    349 obs. of  2 variables:
##  $ Lifetime: num  35.5 35.4 34.9 34.8 33.8 33.5 32.6 32.4 31.8 31.6 ...
##  $ Diet    : Factor w/ 6 levels "N/N85","N/R40",..: 4 4 4 4 4 4 4 4 4 4 ...
```

Recall that `str()` is great way to get a compact summary of an R data object. It returns the class of the object, the number of observations and variables, the class of those variables, and the first few values of each variable. As you can see in `str(case0501)`, mice are given six `Diet` treatments (`Factor w/ 6 levels`) to explore the effect of diet on `Lifetime` (class `numeric`).

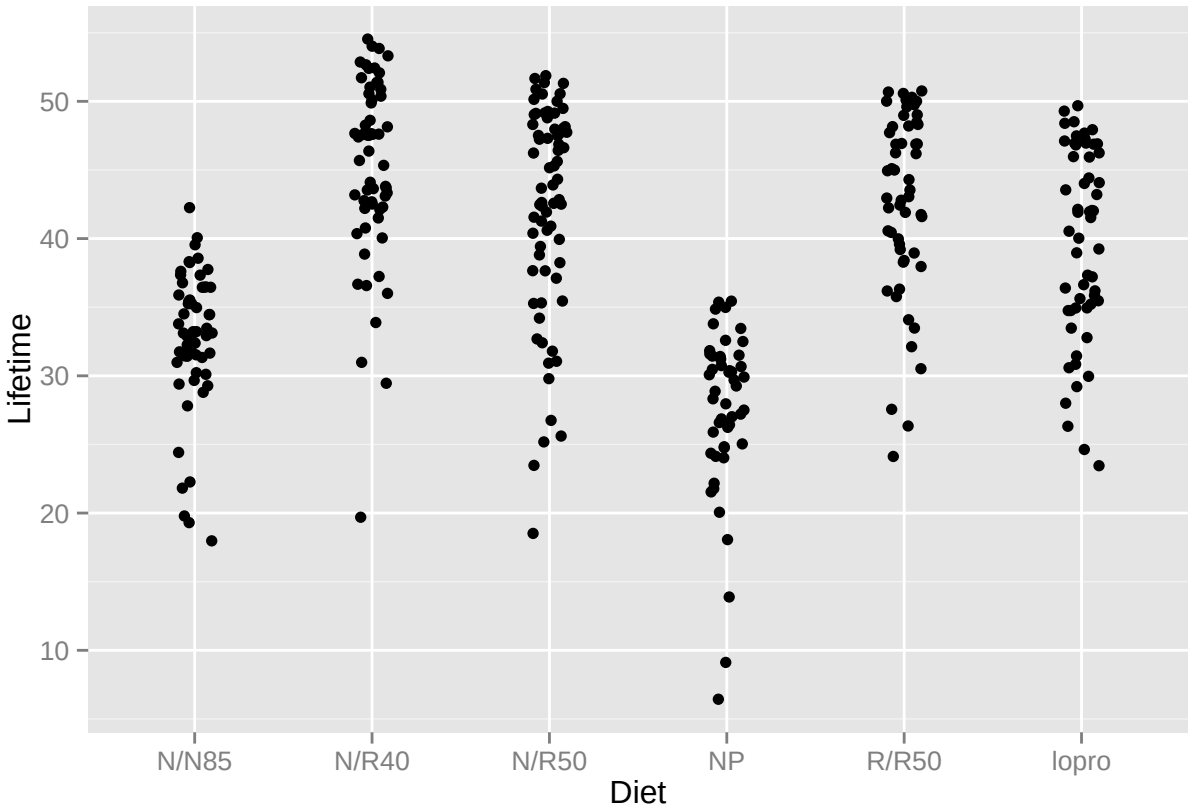
Graphical Group Summaries

```
library(ggplot2)
qplot(Diet, Lifetime, data = case0501)
```



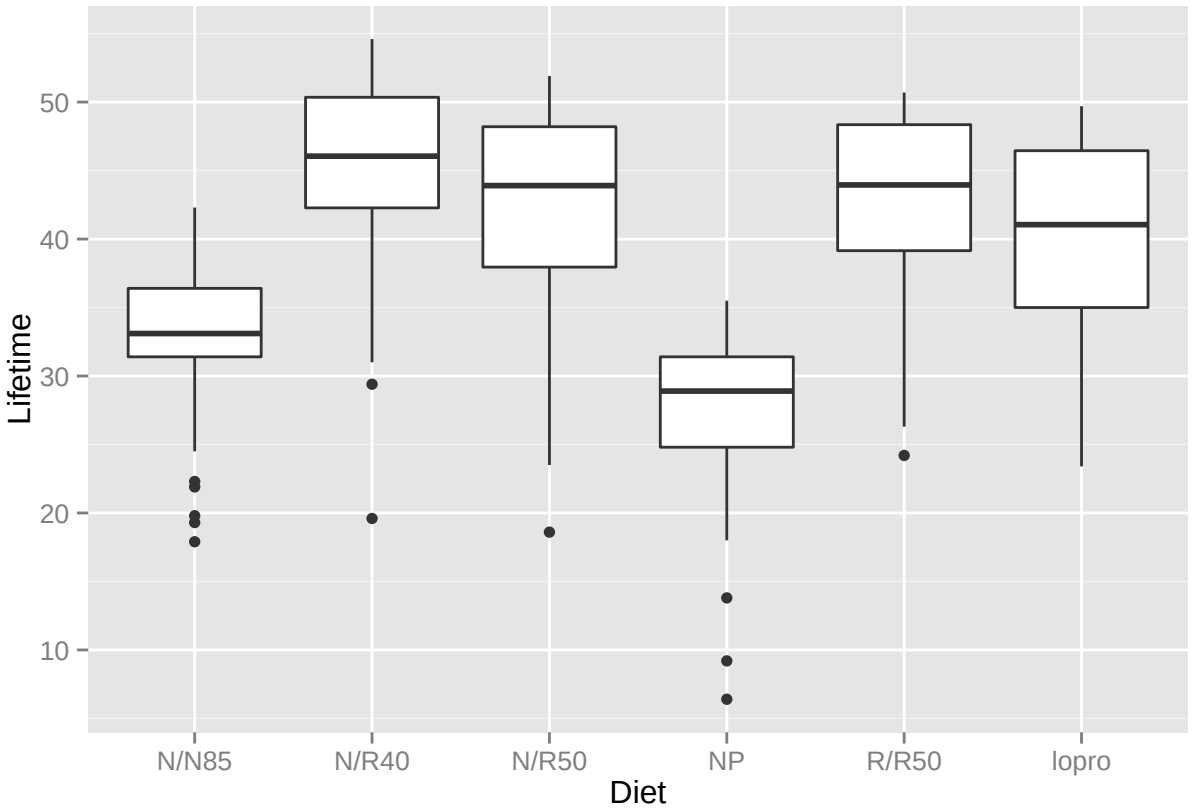
This plot helps us get an overall sense of the data we are studying. The diet treatments are on the x-axis, and the lifetimes are on the y-axis. With this plot we get a general sense of the location and spread of lifetimes within each diet, but not much more. Another drawback to this plot is the overlap problem. For example, if five mice have the same lifetime, on this plot it will look like one mouse! One solution to this problem is the `jitter` argument.

```
qplot(Diet, Lifetime, data = case0501, position = position_jitter(w = 0.1, h = 0.1))
```



This plot now gives us a better understanding of magnitude, but of course the locations are no longer exact. Another alternative is the boxplot.

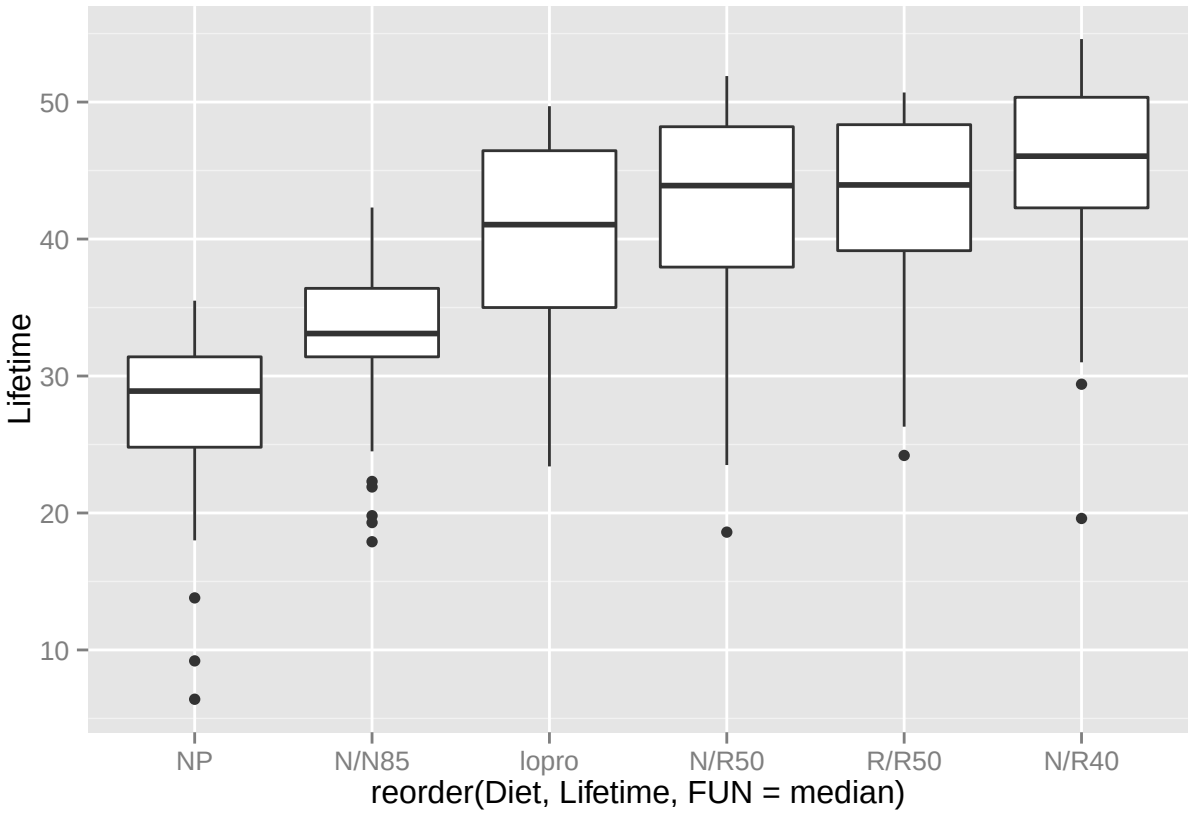
```
qplot(Diet, Lifetime, data = case0501, geom = "boxplot")
```



This plot gives more information in a similar format; side-by-side boxplots. These boxplots give the first and third quartiles, the mean, the lowest/highest values within 1.5*IQR of the first/third quartiles, and the values that lie beyond that range. This is a helpful summary, but be aware we necessarily lose some information by reducing our data to this smaller set of summary statistics.

In some cases it might be nice to see these plots ordered by increasing medians.

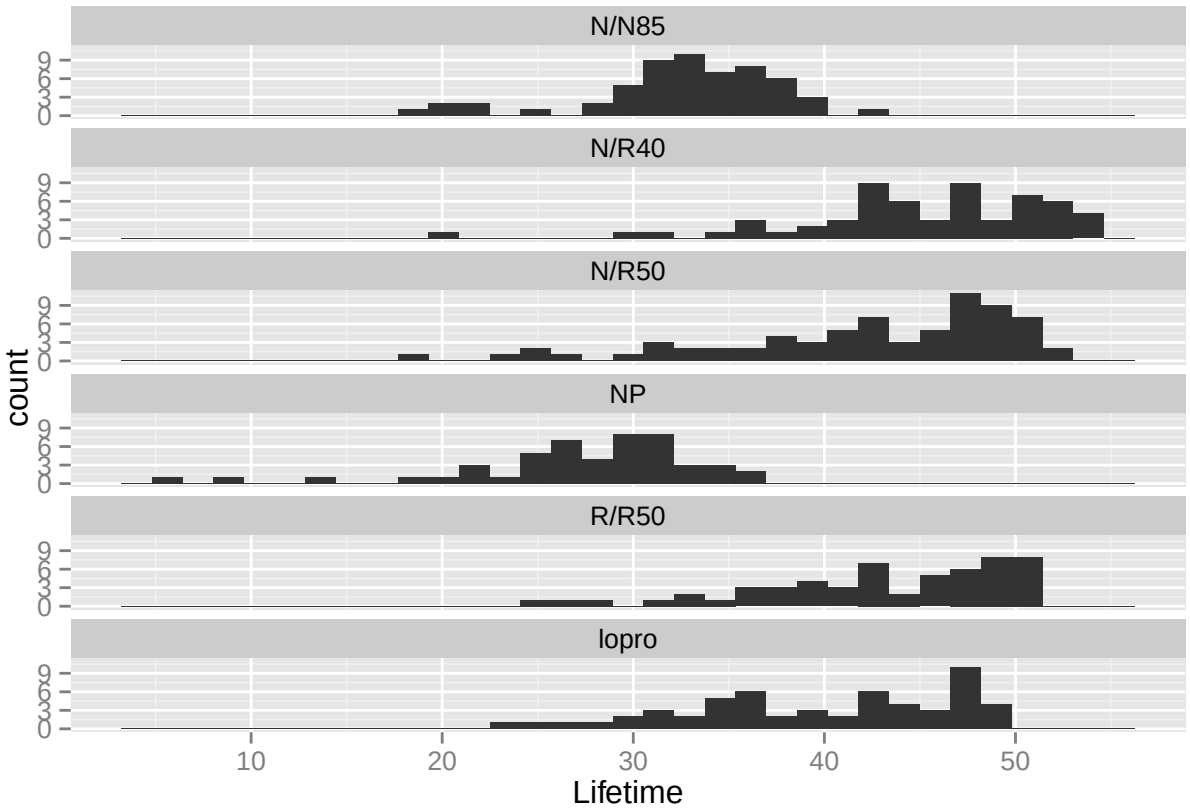
```
qplot(reorder(Diet, Lifetime, FUN = median), Lifetime, data = case0501, geom = "boxplot")
```



The `reorder()` function put the `Diet` boxplots in order of increasing median `Lifetime`.

Sometimes a histogram is preferred to visually convey summary information. The following code constructs one histogram for each diet.

```
qplot(Lifetime, data = case0501) + facet_wrap(~ Diet, ncol = 1)
```



Note that x-axis is fixed across all diets, which makes it easy to compare the spread and location of the lifetimes by diet type. In this code `qplot` automatically defines the bin width to range/30. Keep in mind that bin widths can drastically affect the appearance of a histogram.

These tools give an excellent visual understanding of the data, but we also want exact numeric characteristics.

Numerical Group Summaries

The next line calculates the mean lifetime of each diet.

```
with(case0501, tapply(Lifetime, Diet, mean)) # Group means
```

```
##    N/N85    N/R40    N/R50       NP    R/R50    lopro
## 32.69123 45.11667 42.29718 27.40204 42.88571 39.68571
```

The function `tapply()` applies a function to categorized data. The function takes 3 arguments here: (i) the first argument specifies the data we want to apply the function to, (ii) the second argument declares the category by which we want to divide the data before applying the function, and (iii) the third argument specifies the function we want to apply. However, `tapply()` does not know where the data is coming from, so `tapply()` is wrapped in `with()` for this reason.

```
with(case0501, tapply(Lifetime, Diet, sd)) # standard deviations by group
```

```
##    N/N85    N/R40    N/R50       NP    R/R50    lopro
## 5.125297 6.703406 7.768195 6.133701 6.683152 6.991695
```

```
n_by_group <- with(case0501, tapply(Lifetime, Diet, length)) # Group sizes
n_by_group
```

```
## N/N85 N/R40 N/R50    NP R/R50 lopro
##    57    60    71    49    56    56
```

These lines operate in the same way, but calculate different group statistics. Notice that the group sizes are saved as `n_by_group`, which can be very useful for later operations.