Intro to Wrangling R Data Frames

08/09/2021

Contents

Intro

This is a much more hands-on activity to get you introduced to `R`. I'd recommend opening up a new `R script` file and follow along!

About the Datasets

`mtcars` and `iris` are two data sets built into `R`, so you don't have to do anything to download them. They're very common in `R` tutorials, so that's why I'm using them.

mtcars

`mtcars` has some data on cars from 1974 or something. It's nice to use to teach `R` because it's relatively small.

First Steps

A good first step to analyzing any data set is to get a sense of what the data looks like. Run the following commands and see what's in it!

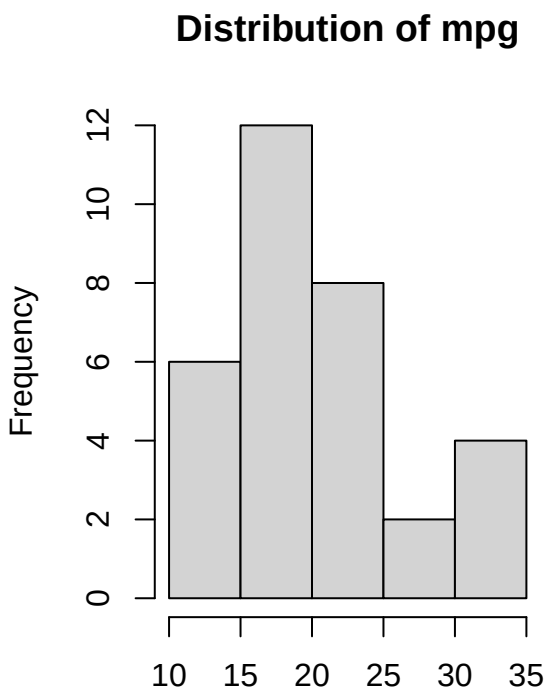
```
?mtcars  # metadata on the data set
dim(mtcars)  # the dimensions
names(mtcars)  # the column names
rownames(mtcars)  # the row names
summary(mtcars)  # summary statistics for each of the columns
head(mtcars)  # look at the first 6 rows of the data set
```

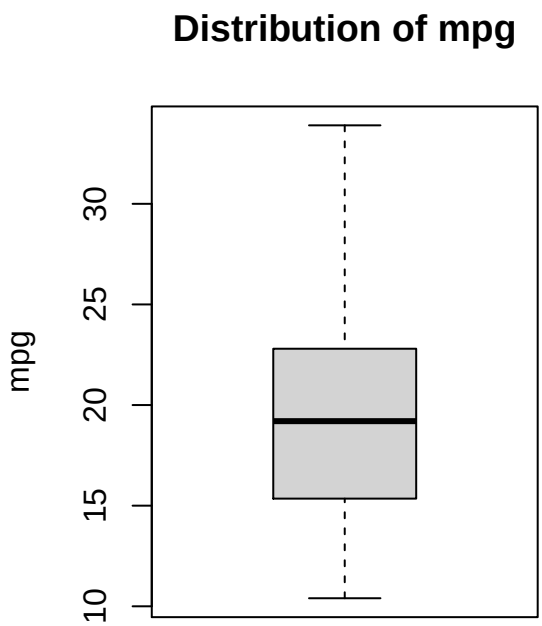
These numbers are all fine and interesting, but let's answer some questions about the data set.

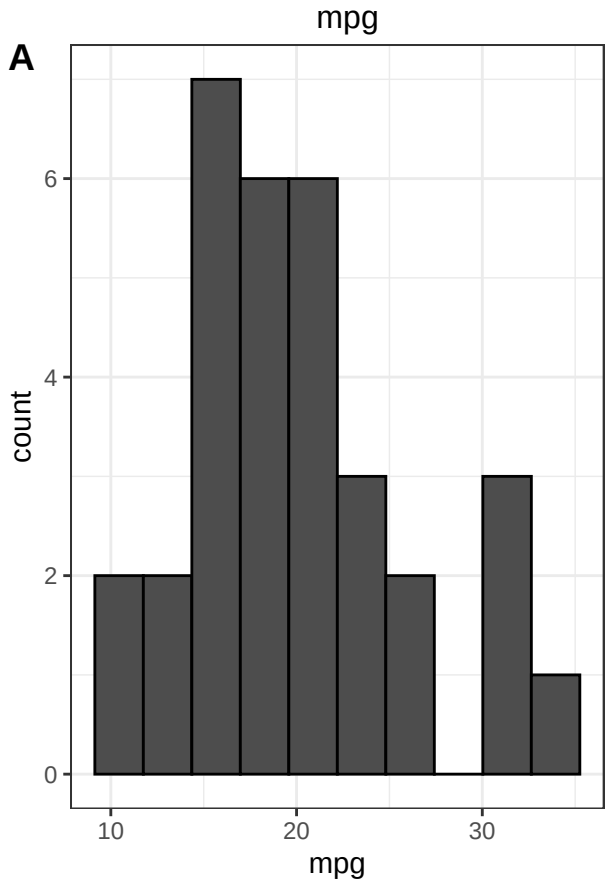
What Affects Miles Per Gallon?

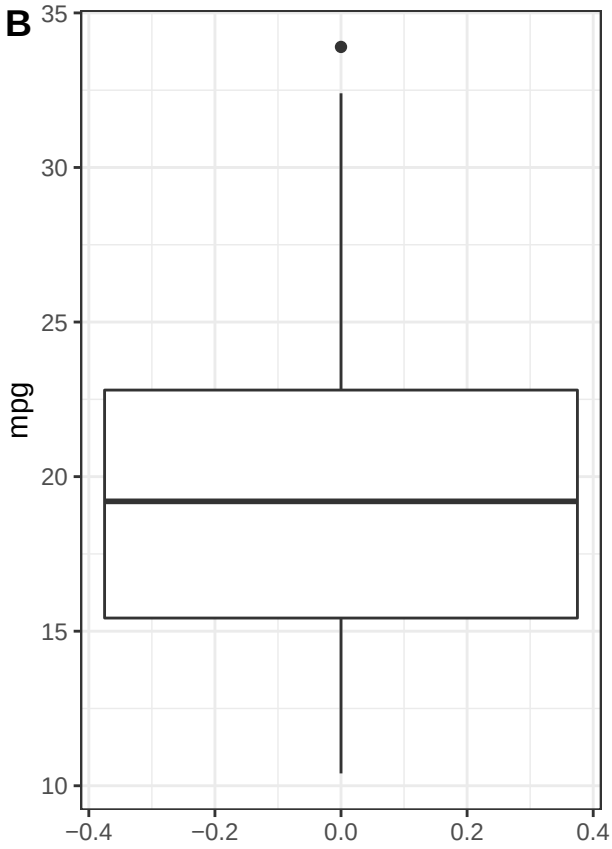
Miles Per Gallon (mpg) is an interesting statistics, so let's explore it first. Plotting is a very good first step to analyzing the data. Since we are just looking at one **continuous** variable, we should use a histogram to visualize the distribution.

Exercise. Generate the following histogram and boxplots. There's a solution using base `R` (first plot) and `ggplot2`. (I would recommend using base `R` to explore, and then making it pretty with `ggplot2` when you want to present your conclusions).



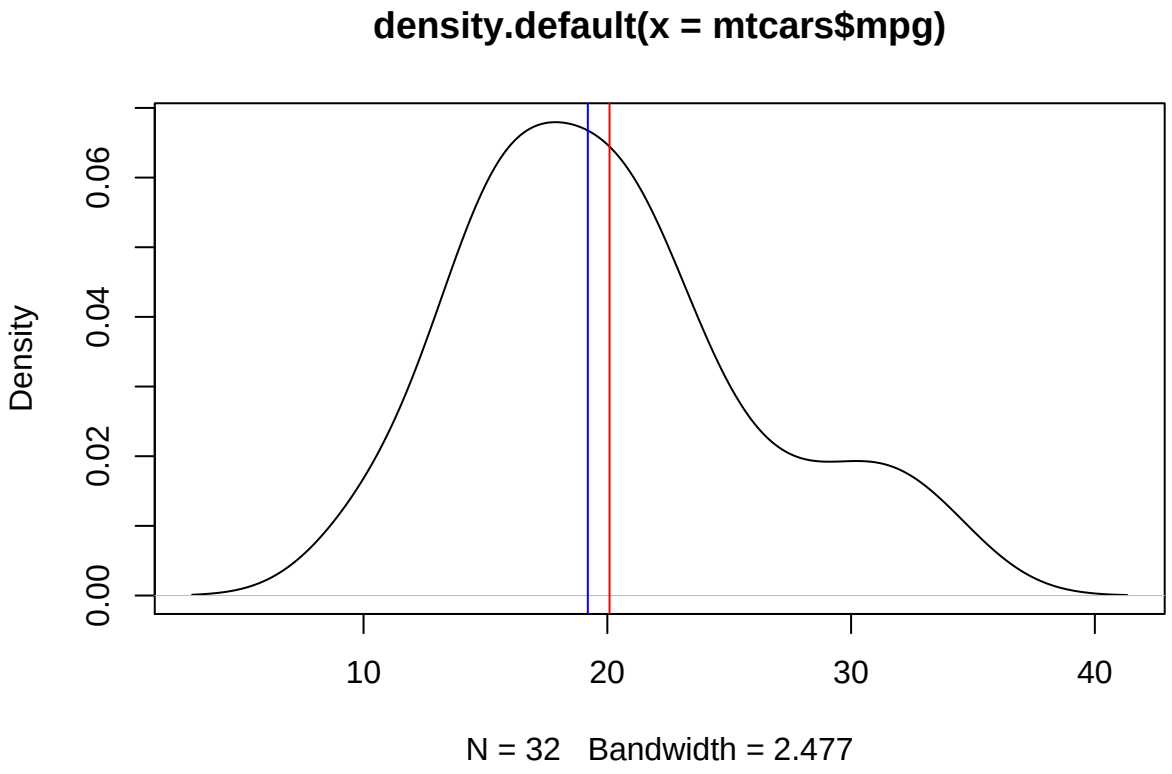






This data looks right skewed based on the long tail in the boxplot. We can fruther see that in the density plot:

```r
dens = density(mtcars$mpg)
plot(dens) +
  abline(v = mean(mtcars$mpg), col = "red") +  # adds red line at the mean
  abline(v = median(mtcars$mpg), col = "blue")  # adds a blue line at the median
```




```
## integer(0)
```

Since the mean is greater than the median, it suggests that the data is in-fact right skewed. Let's see what cars have these high miles per gallon by sorting on the `mpg` column.

Exercise. Show the features of the top 5 most efficient vehicles and the least 5 most efficient vehicles. The desired output is shown below (the format doesn't really matter):

```
##                 mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Toyota Corolla 33.9   4 71.1  65 4.22 1.835 19.90  1  1    4    1
## Fiat 128       32.4   4 78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4 75.7  52 4.93 1.615 18.52  1  1    4    2
## Lotus Europa   30.4   4 95.1 113 3.77 1.513 16.90  1  1    5    2
## Fiat X1-9      27.3   4 79.0  66 4.08 1.935 18.90  1  1    4    1
```

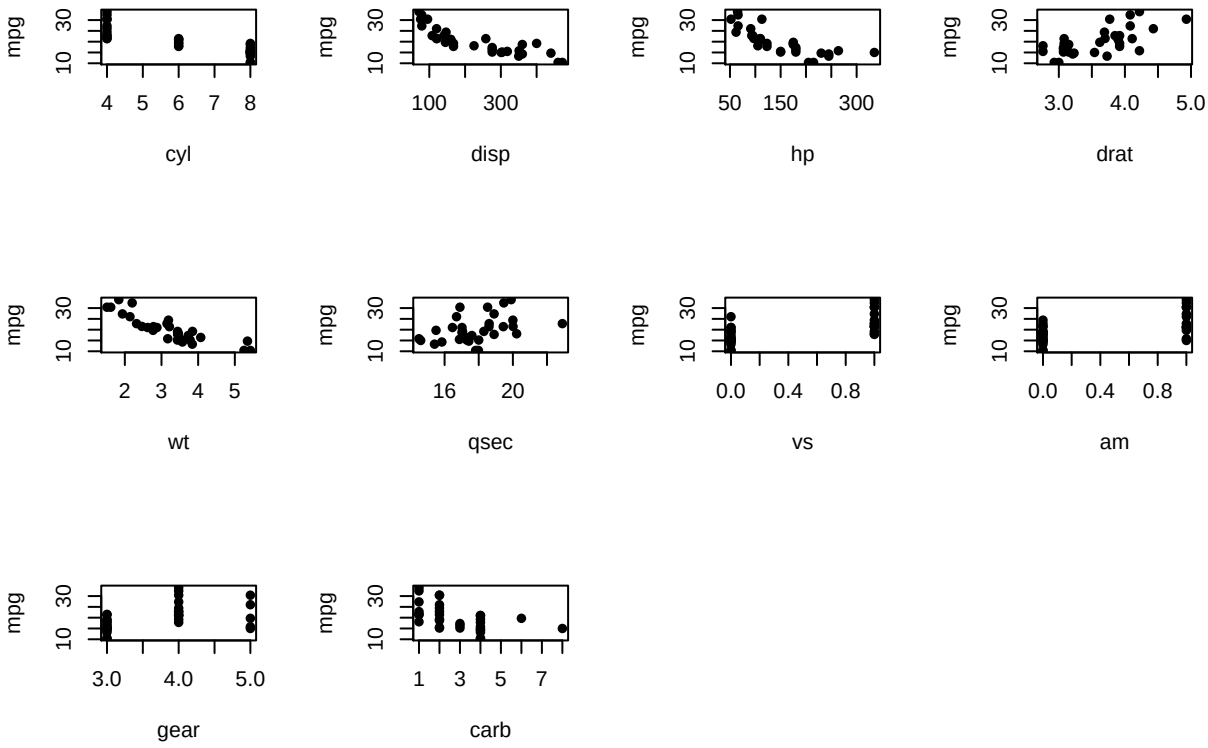
```
##                     mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Cadillac Fleetwood  10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
## Camaro Z28          13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
## Duster 360          14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
## Chrysler Imperial   14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
```

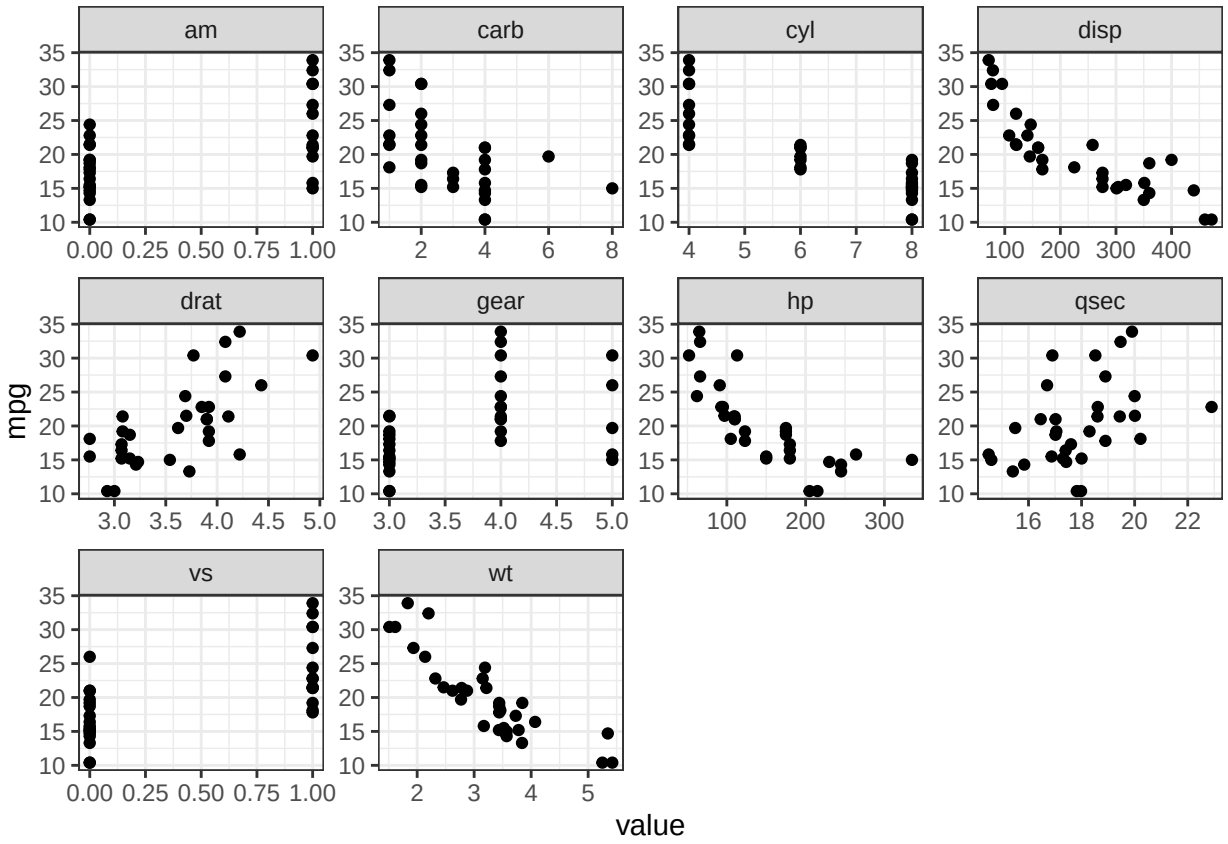
On a glance, it looks like most the attributes play a role in affecting the miles per gallon. How can we tell what is actually associated with mpg? One solution is to plot everything against miles per gallon.

Exercise. Create the following plots using a `for` loop (again, the aesthetics don't really matter when you're exploring the data – just make sure all the information is there. There's two versions just for your

edification):

```
par(mfrow = c(3, 4))  # this sets up a 3 x 4 environment for plotting

# for loop here. what do you need to loop over?
# put your plot function here
```





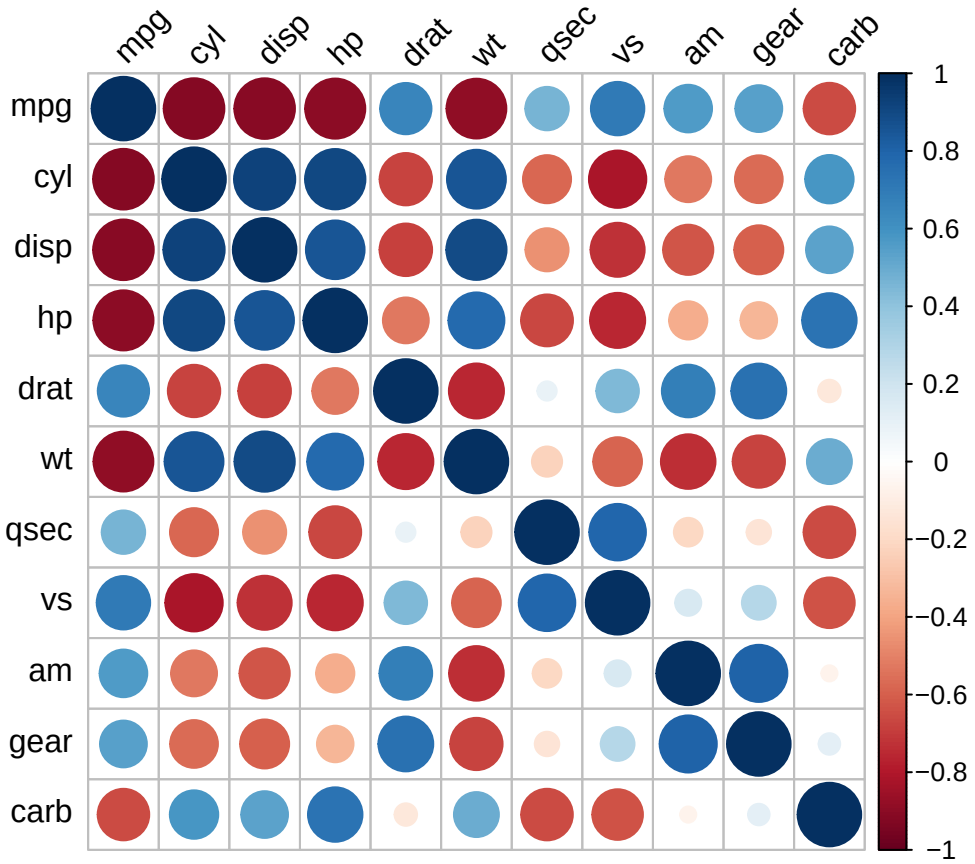
So on a visual inspection, it looks like all these variables have at least a little relation to miles per gallon (note that some of these are actually categorical variables, which makes the scatter plots look a little wonky). With a bigger data set, you can't just simply plot variables like this, however, since that's just way too many plots. What you *can* do is look at the correlation coefficients, which tells you how strongly a change in one variable causes a change in other. Remember that a positive correlation means that an increase in one variable indicates an increase in the other, and a larger magnitude of the correlation means that the two variables are more closely linked (you can't properly calculate correlation with categorical variables, i.e. with `vs` and `am` in this data set, so let's ignore them for now). Below is how to see the correlation matrix in `R` (i.e. each pairwise correlation).

```
mt_cors = cor(mtcars, method = "spearman")
mt_cors
```

```
##             mpg        cyl       disp         hp        drat         wt
## mpg   1.0000000 -0.9108013 -0.9088824 -0.8946646  0.65145546 -0.8864220
## cyl  -0.9108013  1.0000000  0.9276516  0.9017909 -0.67888119  0.8577282
## disp -0.9088824  0.9276516  1.0000000  0.8510426 -0.68359210  0.8977064
## hp   -0.8946646  0.9017909  0.8510426  1.0000000 -0.52012499  0.7746767
## drat  0.6514555 -0.6788812 -0.6835921 -0.5201250  1.00000000 -0.7503904
## wt   -0.8864220  0.8577282  0.8977064  0.7746767 -0.75039041  1.0000000
## qsec  0.4669358 -0.5723509 -0.4597818 -0.6666060  0.09186863 -0.2254012
## vs    0.7065968 -0.8137890 -0.7236643 -0.7515934  0.44745745 -0.5870162
## am    0.5620057 -0.5220712 -0.6240677 -0.3623276  0.68657079 -0.7377126
## gear  0.5427816 -0.5643105 -0.5944703 -0.3314016  0.74481617 -0.6761284
## carb -0.6574976  0.5800680  0.5397781  0.7333794 -0.12522294  0.4998120
##             qsec         vs          am       gear        carb
## mpg   0.46693575  0.7065968  0.56200569  0.5427816 -0.65749764
## cyl  -0.57235095 -0.8137890 -0.52207118 -0.5643105  0.58006798
```

```
## disp -0.45978176 -0.7236643 -0.62406767 -0.5944703  0.53977806
## hp   -0.66660602 -0.7515934 -0.36232756 -0.3314016  0.73337937
## drat  0.09186863  0.4474575  0.68657079  0.7448162 -0.12522294
## wt   -0.22540120 -0.5870162 -0.73771259 -0.6761284  0.49981205
## qsec  1.00000000  0.7915715 -0.20333211 -0.1481997 -0.65871814
## vs    0.79157148  1.0000000  0.16834512  0.2826617 -0.63369482
## am   -0.20333211  0.1683451  1.00000000  0.8076880 -0.06436525
## gear -0.14819967  0.2826617  0.80768800  1.0000000  0.11488698
## carb -0.65871814 -0.6336948 -0.06436525  0.1148870  1.00000000
```

```r
library(rstatix)
rstatix::cor_plot(mt_cors)
```



Let's look at the matrix and plot more closely. We're most interested in the first column of the correlation matrix, since that's where `mpg` is. By definition, miles per gallon has a perfect correlation with itself. In terms of the other attributes, it looks like `cyl`, `disp`, `hp`, and `wt` have very strong negative correlations with `mpg`. The positive correlations look more modest, though they're still somewhat strong.

If you want to test how significant the correlations are, you can use the `cor.test` function.

```r
for(i in 1:11){
  print(paste(names(mtcars)[i], cor.test(mtcars[, i], mtcars$mpg)$p.value))
}
```

```
## [1] "mpg 0"
## [1] "cyl 6.11268714258096e-10"
## [1] "disp 9.3803265373813e-10"
## [1] "hp 1.78783525412106e-07"
## [1] "drat 1.77623992875242e-05"
```

```
## [1] "wt 1.29395870135052e-10"
## [1] "qsec 0.0170819884965196"
## [1] "vs 3.41593725441997e-05"
## [1] "am 0.000285020743935065"
## [1] "gear 0.00540094822470765"
## [1] "carb 0.00108444622049168"
```

```
## better way to loop -- it's much better cleaner
sapply(mtcars, function(x){
  cor.test(x, mtcars$mpg)$p.value
})
```

```
##          mpg          cyl         disp           hp         drat           wt
## 0.000000e+00 6.112687e-10 9.380327e-10 1.787835e-07 1.776240e-05 1.293959e-10
##         qsec           vs           am         gear         carb
## 1.708199e-02 3.415937e-05 2.850207e-04 5.400948e-03 1.084446e-03
```

All of these are statistically significant at the p = 0.05 level, which is interesting. If you have a lot of p-values, you can quickly screen for the ones that are significant. Let's say we set an arbitrary cutoff at the p = 1e-5 level (1×10^{-5}). We can separate out these values by *boolean indexing*; that is, we can pass a vector of TRUE/FALSE values to select cells of interest (where TRUE means to include and FALSE means to exclude). We can make this array by simply comparing the `p_vals` to our desired value:

```
p_vals = sapply(mtcars, function(x){
  cor.test(x, mtcars$mpg, method = "spearman")$p.value
})

names(p_vals)[p_vals < 1e-5]  # this will compare each value in the p_vals to 1e-5 and replace each wit
```

```
## [1] "mpg"  "cyl"  "disp" "hp"   "wt"   "vs"
```

```
names(p_vals)[p_vals >= 1e-5]  # the ones removed
```

```
## [1] "drat" "qsec" "am"   "gear" "carb"
```

```
head(mtcars[, p_vals < 1e-5])
```

```
##                   mpg cyl disp  hp    wt vs
## Mazda RX4         21.0   6  160 110 2.620  0
## Mazda RX4 Wag     21.0   6  160 110 2.875  0
## Datsun 710        22.8   4  108  93 2.320  1
## Hornet 4 Drive    21.4   6  258 110 3.215  1
## Hornet Sportabout 18.7   8  360 175 3.440  0
## Valiant           18.1   6  225 105 3.460  1
```

Note that `drat`, `qsec`, `am`, `gear`, and `carb` are gone. You'll apply similar techniques to look at your genomic data.

A Note on Correlations

You might have noticed that we used *Spearman* correlation. *Spearman* works by comparing rank values (i.e. for each attribute, the highest value gets assigned 1, the next highest gets 2, etc.). This works better for non-linear correlations, unlike the default *Pearson* correlation, which is probably what you're more familiar with. Here's an example below:

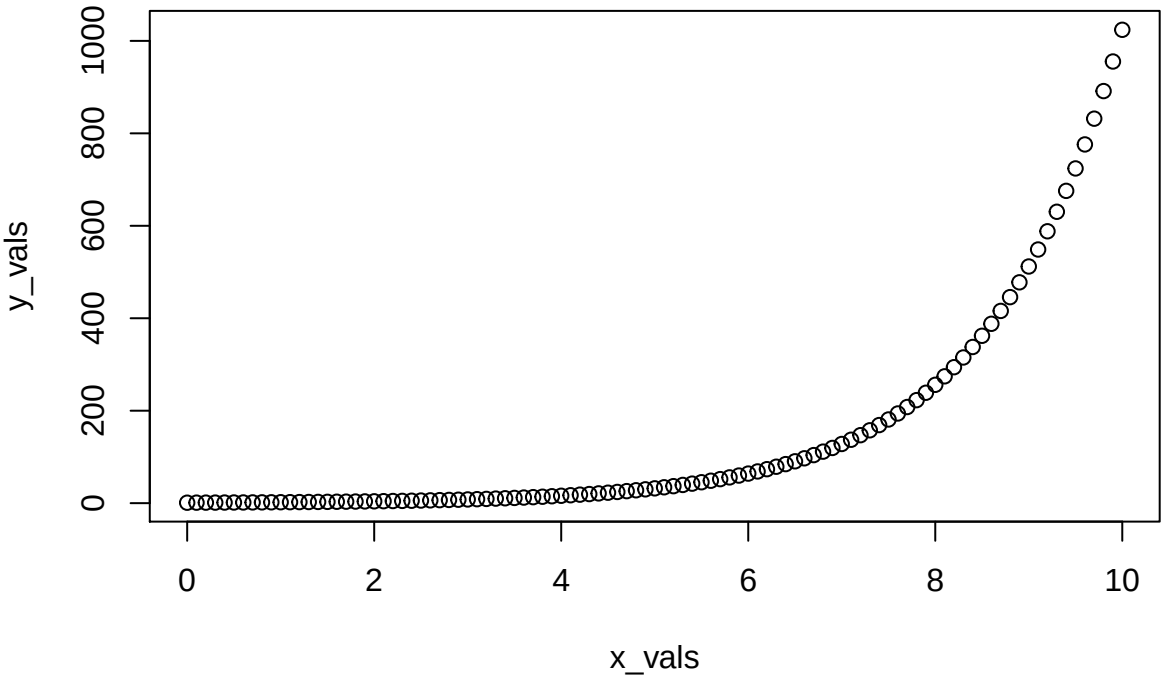
```
x_vals = 0:100 / 10  # 0 - 10 spaced by 0.1
y_vals = 2^x_vals

cor(x_vals, y_vals, method = "pearson")
```

```
## [1] 0.7835654
```

```
cor(x_vals, y_vals, method = "spearman")
```

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

```
plot(x_vals, y_vals)
```



Clearly, this isn't a linear relationship (it's exponential), so the Pearson correlation is less than 1. However, the Spearman correlation is 1 because the function is strictly increasing (i.e. every increase in `x` results in an increase in `y`; the sorted `y` values never decrease). In this case, the Spearman correlation is much better – there's obviously a relation between the data points (we made it as such), but the Pearson correlation doesn't capture it. In summary: use the Pearson correlation if you want to test if two variables are *linearly* correlated, and use Spearman correlation if you want to test if two variables are related in any manner.

A bonus activity with `mpg`: let's say we're interested in building a model to predict `mpg`. There's a bunch of ways to do so (re: machine learning techniques), but let's look at a very simple method: linear regression.[1]. The standard "workflow" behind machine learning is to partition your data into two datasets: a training set (typically about 4/5 of the data), and a validation set (the remaining 1/5). The goal is to train on the bigger set, then use the remaining data to see how well the model predicts. We'll (probably) talk more about machine learning later on in the year. This is a generalized way to do so in `R`:

```
# selects the samples to make the linear regression
sample_inds = sample(1:nrow(mtcars), 4*nrow(mtcars)/5)

# partitions into samples
train_sample = mtcars[sample_inds, ]
test_sample = mtcars[-sample_inds, ]

# makes a linear regression model using all the variables, which is the period
# you can examine this model with more methods
model = lm(mpg ~ ., train_sample)
```

[1] regression is actually a form of machine learning by the way

```r
# see how good the model is by making predictions on the test sample
predicted_mpg = predict.lm(model, test_sample)

# residuals are the difference between predicted values and the actual
residuals = test_sample$mpg - predicted_mpg

# calculate the root mean square error
rmse = sqrt(mean((residuals)^2))

rmse
```

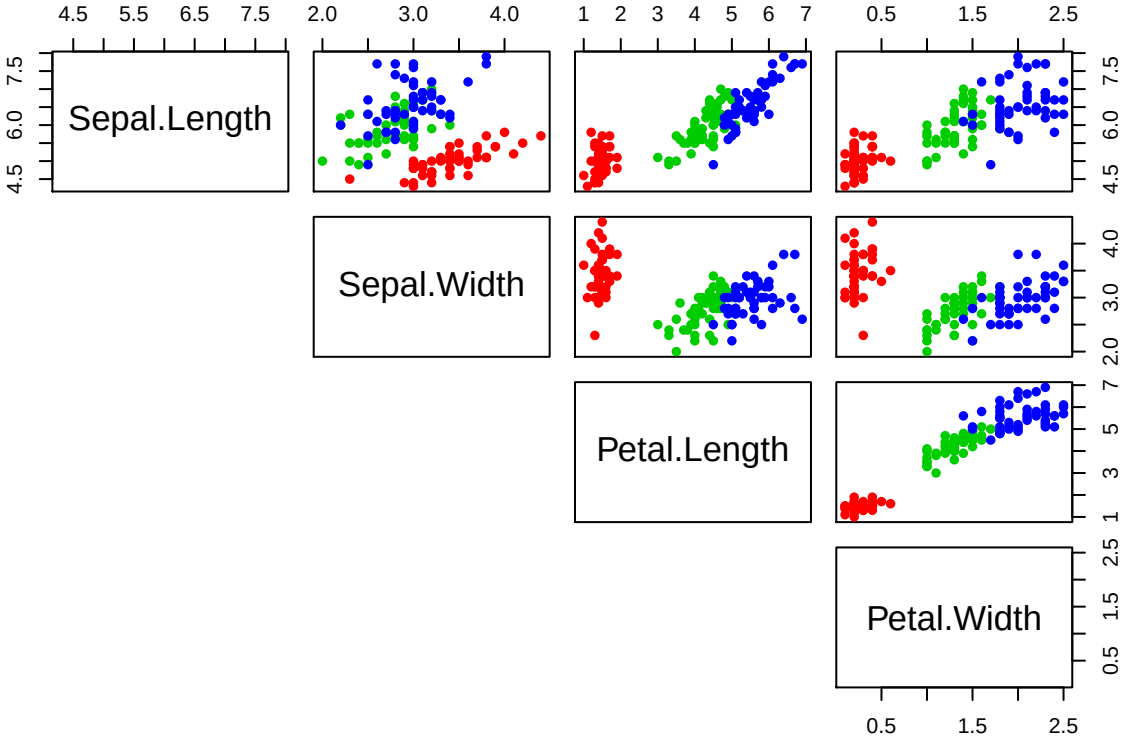
```
## [1] 4.033097
```

This error is kind of bad, since on average it's off by a pretty decent amount relative to the mean. It's not horrible, but ideally you'd like to do better. Feel free to play around with other methods to estimate mileage!

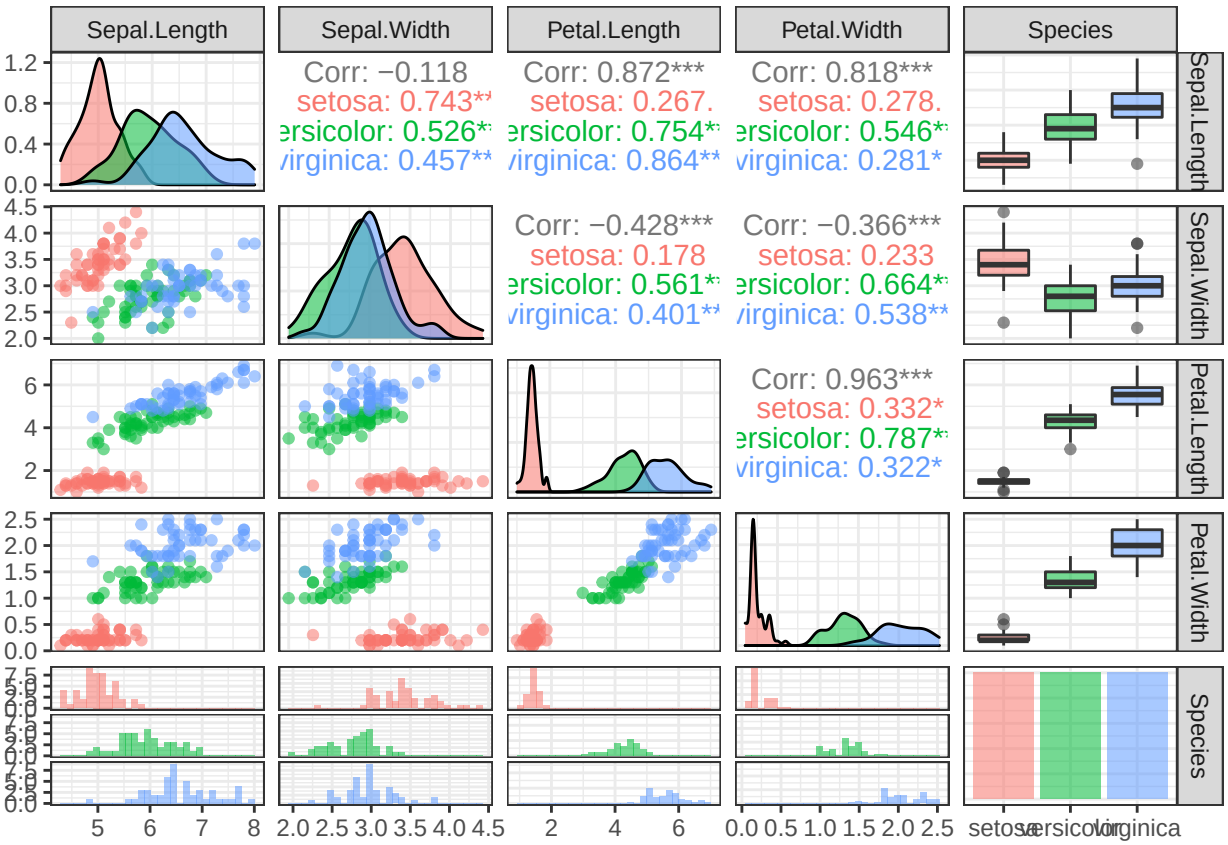
iris

This is your turn! Try to complete the following exercises. If you ever get confused by the coding, look at the `R` documentation for the functions If you're still confused or don't understand the instructions, please ask me for help!

1. What does the data set even describe? How many observations does it contain? What features does the data set contain?
2. Which features are continuous variables? Which are categorical? Can you tell me the `R` data type for each column?
3. Examine how each of the continuous variables are distributed, making a histogram for each of them (if you're feeling adventurous, slap a density plot on top of the histogram). Do you notice anything interesting in their distributions?
4. Create a pairwise plot of each continuous variable (hint: use the `pairs` plotting command). BONUS: can you think of a way to color the plots by species as below?



5. Look up the `GGally` package. Install and load it. Use the `ggpairs` function to make the following plot (note that the `alpha`, or the transparency, was set to 0.4. Also note that it's an extension of the `ggplot2` package, so you'll have to use the `aes` syntax to set the color and transparency).



6. Based on the plots you just made, which species looks the most unique out of the three? Which species look the most similar?
7. It looks like the `Sepal.Width` of *versicolor* and *virginica* look very similar. Follow the next few steps to determine if their `Sepal.Widths` are actually significant:

a. Create three new data frames, each containing only the data for a single species (they should be 50 rows x 5 columns).
b. Use the `t.test` function to compare the `Sepal.Width` of *versicolor* and *virginica*.

8. Let's say we want to split the data set into "narrow-sepaled" plants and "wide-sepaled" plants, and you want to add a new column to the iris data set containing this data and perform some more analysis. Follow these steps:

a. Find the mean sepal width in the dataframe, and store it in a variable.
b. Create a new vector using the `ifelse` function to fill out values (comparing `Sepal.Width` to the mean value).
c. Create a new column in the `iris` data set (using the dollar sign) and assign the new vector to that column.
d. Create a boxplot plotting the `Sepal.Width` based on the new column (narrow vs. wide).