Introduction to Statistics in R

Catherine Barber

2022-10-31

Goal and Learning Outcomes

Goal: The goal of this lesson is for you to navigate in RStudio and use R functions to complete basic data analysis tasks and visualizations.

Learning Outcomes: During this lesson, you will demonstrate your ability to…

- Import data from a csv file.
- Assign data to objects.
- Explore analyze data with basic statistics.
- Create histograms, boxplots, and scatterplots.

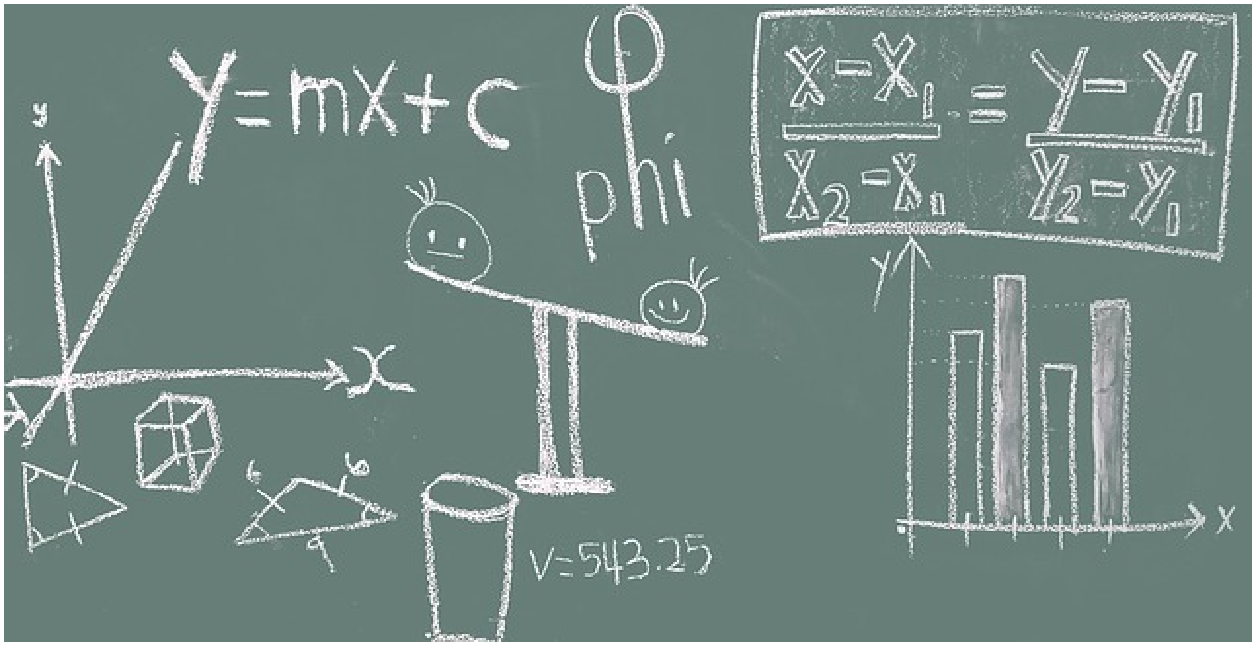


Figure 1: Math Anxiety; image by Chuk Yong from Pixabay - free license for commercial use

Scenario: An Experiment to Reduce Math Anxiety in a Statistics Class

You are working in a research lab of a faculty member who is interested in psychosocial interventions to reduce anxiety in academic settings. Your team has just finished running an experiment. Twenty students in an introductory statistics class were randomly sampled and agreed to participate.

At the time of enrollment in the study, the participants completed a 100-point statistics pretest assessing their baseline knowledge of statistics. In addition, participants completed a 20-point measure of trait anxiety and a 100-point measure of college-level math knowledge. They also indicated their major (coded as a number): psychology (1), sociology (2), or political science (3). The

You and your team then randomly assigned participants to one of two groups, also coded as a number:

- The treatment group (1) received 30 minutes of math-related relaxation training weekly.
- The control group (2) received 30 minutes of group math practice and coaching weekly.

The intervention phase continued for six weeks, during which time all participants attended their statistics class as usual.

After six weeks, the participants completed an equivalent-forms, 100-point statistics posttest and a 30-point measure of state anxiety.

Finally, a team member did a quick calculation to determine each participant's difference score, computed as the difference between the posttest and the pretest.

The data were manually entered into a spreadsheet and saved as a CSV file. You are now ready to start working with the dataset!

Import the Dataset

When working in RStudio, you can import a file directly into the Global Environment using a point-and-click method or by calling the function `read.csv()` on a file path. This lesson covers both options.

Importing Using the GUI

In the Environment tab of the Global Environment pane, click Import Dataset and select the file format that best fits your dataset. In this lesson, you will select From Text (base) because the data are stored in a CSV file.

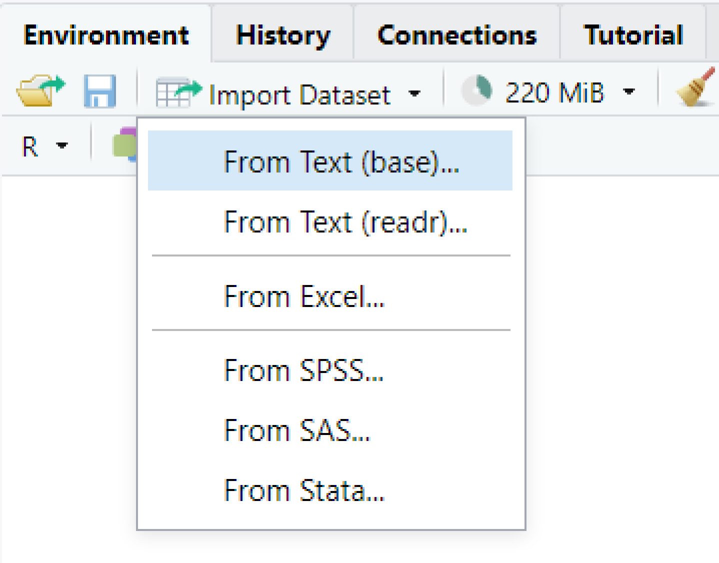
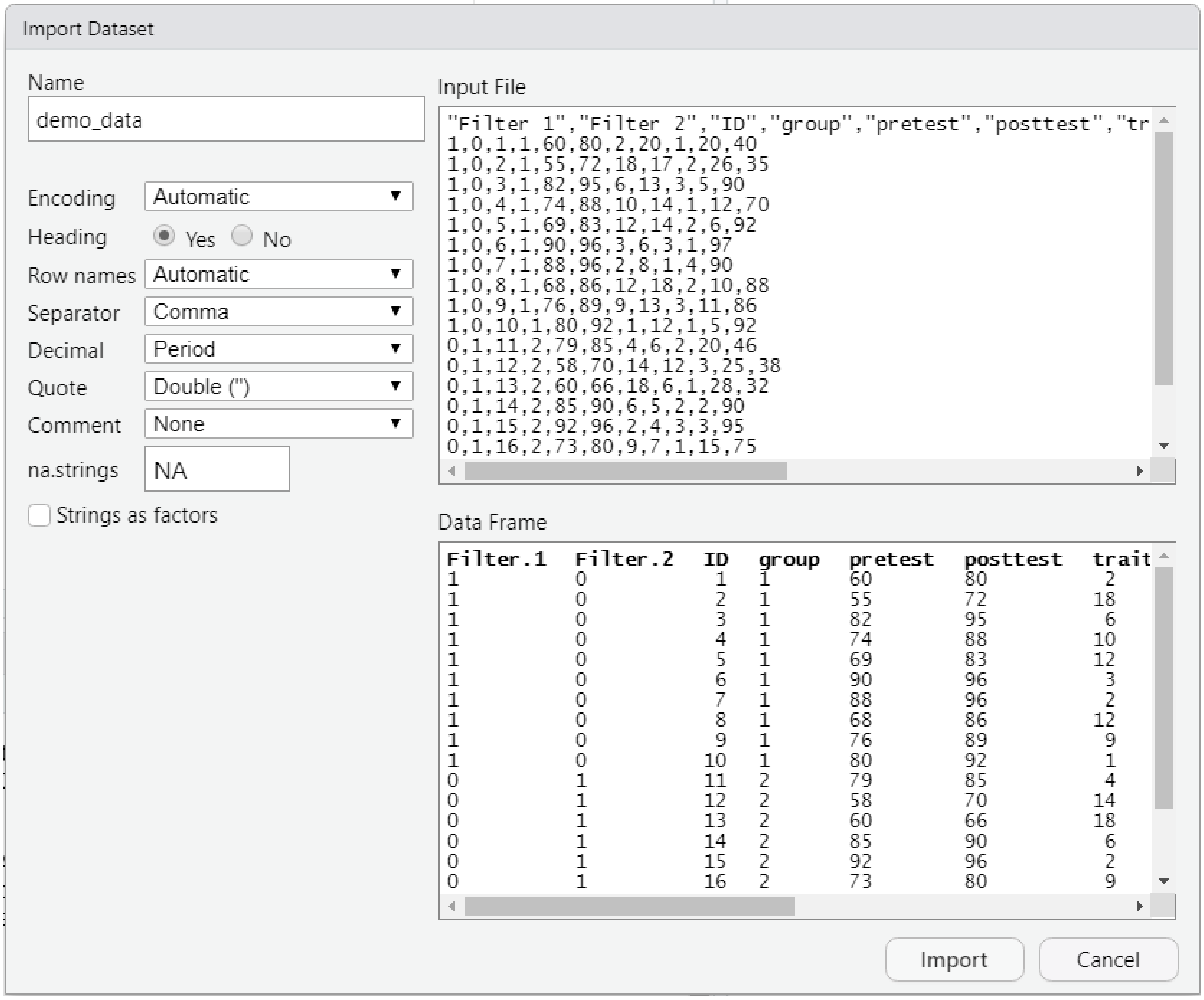


Figure 2: GUI import

Name the dataset `demo_data` and select Yes for Heading; this will read in the first row of data (variable names) as column headings. In the current example, do not check the Strings as factors box; however, be aware that if you want R to treat strings as factors (e.g., if Treatment and Control groups were coded as strings rather than as numbers), you could check this box to automatically treat those string variables as factors upon import.



Name: demo_data

Encoding: Automatic ▼
Heading: ● Yes ○ No
Row names: Automatic ▼
Separator: Comma ▼
Decimal: Period ▼
Quote: Double (") ▼
Comment: None ▼
na.strings: NA
□ Strings as factors

```
"Filter 1","Filter 2","ID","group","pretest","posttest","tr
1,0,1,1,60,80,2,20,1,20,40
1,0,2,1,55,72,18,17,2,26,35
1,0,3,1,82,95,6,13,3,5,90
1,0,4,1,74,88,10,14,1,12,70
1,0,5,1,69,83,12,14,2,6,92
1,0,6,1,90,96,3,6,3,1,97
1,0,7,1,88,96,2,8,1,4,90
1,0,8,1,68,86,12,18,2,10,88
1,0,9,1,76,89,9,13,3,11,86
1,0,10,1,80,92,1,12,1,5,92
0,1,11,2,79,85,4,6,2,20,46
0,1,12,2,58,70,14,12,3,25,38
0,1,13,2,60,66,18,6,1,28,32
0,1,14,2,85,90,6,5,2,2,90
0,1,15,2,92,96,2,4,3,3,95
0,1,16,2,73,80,9,7,1,15,75
```

```
Filter.1  Filter.2  ID  group  pretest  posttest  trait
1         0          1  1      60       80            2
1         0          2  1      55       72           18
1         0          3  1      82       95            6
1         0          4  1      74       88           10
1         0          5  1      69       83           12
1         0          6  1      90       96            3
1         0          7  1      88       96            2
1         0          8  1      68       86           12
1         0          9  1      76       89            9
1         0         10  1      80       92            1
0         1         11  2      79       85            4
0         1         12  2      58       70           14
0         1         13  2      60       66           18
0         1         14  2      85       90            6
0         1         15  2      92       96            2
0         1         16  2      73       80            9
```

Figure 3: Import interface

Finally, click Import, and your data are now available in a dataframe object in the Global Environment!

Importing Using `read.csv()` and the File Name/Path

Begin by saving the CSV file in your current working directory. If you aren't sure what directory you are in, remember that you can call `getwd()` to check and call `setwd()` with your preferred directory path as the argument to change the working directory.

Next, use read.csv to import your data as a dataframe with the name `demo_data`. If the file is already in the working directory, you will simply indicate the file name as the argument; otherwise, the file path will be specified. Note that the file path will be specific to your directory structure.

```
demo_data <- read.csv("demo_data.csv", stringsAsFactors = FALSE)
```

You should see the dataframe `demo_data` in your Global Environment. Next, take a look at this object’s structure.

Examine the Structure of the Dataframe

```
str(demo_data)
```

```
## ’data.frame':    20 obs. of  9 variables:
##  $ ID           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ group        : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ pretest      : int  60 55 82 74 69 90 88 68 76 80 ...
##  $ posttest     : int  80 72 95 88 83 96 96 86 89 92 ...
##  $ trait_anxiety: int  2 18 6 10 12 3 2 12 9 1 ...
##  $ difference   : int  20 17 13 14 14 6 8 18 13 12 ...
##  $ major        : int  1 2 3 1 2 3 1 2 3 1 ...
##  $ state_anxiety: int  20 26 5 12 6 1 4 10 11 5 ...
##  $ math_score   : int  40 35 90 70 92 97 90 88 86 92 ...
```

This output reveals that the dataframe has 20 observations of 9 variables.

Prepare the Dataframe

All variables were imported as integer data, as the team had previously coded the two categorical variables (group and major) with numeric codes. However, you want to treat these two variables as factors, so you need to convert them to factors. In addition, you want to remove the variable `demo_data$ID`, as it contains redundant information.

Create Factors

You can use the `as.factor()` function to convert numeric data to a factor. Recall that variables (columns in the dataframe) do not exist as separate objects in the Global Environment; to work with them, you must refer to these variables in the context of the dataframe. For example, the group variable (treatment vs. control) is `demo_data$group`, and the major variable (psychology, sociology, political science) is `demo_data$major`.

```
demo_data$group <- as.factor(demo_data$group)
demo_data$major <- as.factor(demo_data$major)
```

Now check the structure of `demo_data` again to see what has changed.

```
str(demo_data)
```

```
## ’data.frame':    20 obs. of  9 variables:
##  $ ID           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ group        : Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ pretest      : int  60 55 82 74 69 90 88 68 76 80 ...
##  $ posttest     : int  80 72 95 88 83 96 96 86 89 92 ...
##  $ trait_anxiety: int  2 18 6 10 12 3 2 12 9 1 ...
##  $ difference   : int  20 17 13 14 14 6 8 18 13 12 ...
##  $ major        : Factor w/ 3 levels "1","2","3": 1 2 3 1 2 3 1 2 3 1 ...
##  $ state_anxiety: int  20 26 5 12 6 1 4 10 11 5 ...
##  $ math_score   : int  40 35 90 70 92 97 90 88 86 92 ...
```

Both group and major are now listed as factors, and the number of levels (2 and 3, respectively) is indicated.

Remove Unnecessary Variables

You may recall from a previous lesson that indexing can be used to look at and save parts of a dataframe. The current dataframe `demo_data` includes an unnecessary variable–ID. To remove that variable and save the result in a new object called `d`, use an index:

```
d <- demo_data[,c(2:9)]
```

Recall that the absence of a number or criterion before the comma tells R to return all rows. The vector following the comma tells R which columns to return (in this case, all columns except column 1: ID). The result is a slightly narrower dataframe named `d`, containing 20 rows and 8 variables.

Obtain Summary Statistics

The function `summary` can be used to obtain summary statistics on all integer, numeric, and factor variables in the dataset.

```
summary(d)
```

```
##  group     pretest         posttest     trait_anxiety     difference    major
##  1:10   Min.   :55.00   Min.   :66.00   Min.   : 1.00   Min.   : 4.00   1:7
##  2:10   1st Qu.:63.50   1st Qu.:79.25   1st Qu.: 3.75   1st Qu.: 6.75   2:7
##         Median :75.00   Median :86.50   Median : 7.50   Median :12.00   3:6
##         Mean   :73.65   Mean   :84.80   Mean   : 8.00   Mean   :11.15
##         3rd Qu.:81.25   3rd Qu.:92.25   3rd Qu.:11.25   3rd Qu.:13.25
##         Max.   :92.00   Max.   :96.00   Max.   :18.00   Max.   :20.00
##  state_anxiety    math_score
##  Min.   : 1.0   Min.   :32.00
##  1st Qu.: 5.0   1st Qu.:52.75
##  Median :11.0   Median :80.50
##  Mean   :12.4   Mean   :71.60
##  3rd Qu.:20.0   3rd Qu.:90.00
##  Max.   :28.0   Max.   :97.00
```

The output includes the number of observations in each level of the factors (e.g., 10 in the treatment group and 10 in the control group) as well as the minimum, first quartile, median, mean, third quartile, and maximum values for each continuous variable.

Obtain Descriptive Statistics for Specific Variables

At times, you may want just one or two descriptive statistics for specific variables rather than all of the output provided by `summary()`. This is particularly true when you want to use a statistic as the argument of another function or when you want to embed a statistic within a larger chunk of code. Alternatively, you may want to calculate a statistic that was not included in the summary statistics provided by `summary()`. In each case, you can call a specific statistical function on a specific variable.

```
mean(d$pretest)
```

```
## [1] 73.65
```

```
median(d$pretest)
```

```
## [1] 75
```

```
min(d$pretest)
```

```
## [1] 55
```

```
max(d$pretest)
```

```
## [1] 92
```

```
sd(d$pretest)
```

```
## [1] 11.23095
```

The output includes the mean, median, minimum, maximum, and standard deviation for the entire sample's pretest scores.

Obtain Descriptive Statistics for Subgroups

Recall from a previous lesson that the function `tapply()` can be used to repeat a function (such as the mean or standard deviation) across all levels of a factor. For example, you want to obtain the mean and standard deviation of pretest scores for the treatment and control groups separately.

```
tapply(d$pretest, d$group, mean, na.rm = TRUE)
```

```
##    1    2
## 74.2 73.1
```

```
tapply(d$pretest, d$group, sd, na.rm = TRUE)
```

```
##        1        2
## 11.41928 11.62803
```

Note that the output includes two small arrays–the means and standard deviations for the two groups.

In these commands, you added an argument that was not necessary for your data but that is extremely important when there are missing data: `na.rm = TRUE`. This argument tells R to remove any rows that have NA rather than a value for the variable being evaluated. If you fail to include this argument and your variable has missing data, R will return an error.

As another example of `tapply()`, let's say that you are interested in calculating summary statistics for the three majors within each of the experimental groups. You will need to use the function `list()` to specify the factors in the argument.

```
tapply(d$pretest, list(d$group, d$major), mean, na.rm = TRUE)
```

```
##      1     2        3
## 1 75.5 64.00 82.66667
## 2 70.0 77.25 70.66667
```

```
tapply(d$pretest, list(d$group, d$major), sd, na.rm = TRUE)
```

```
##           1       2         3
## 1 11.818065 7.81025  7.023769
## 2  8.888194 9.17878 18.583146
```

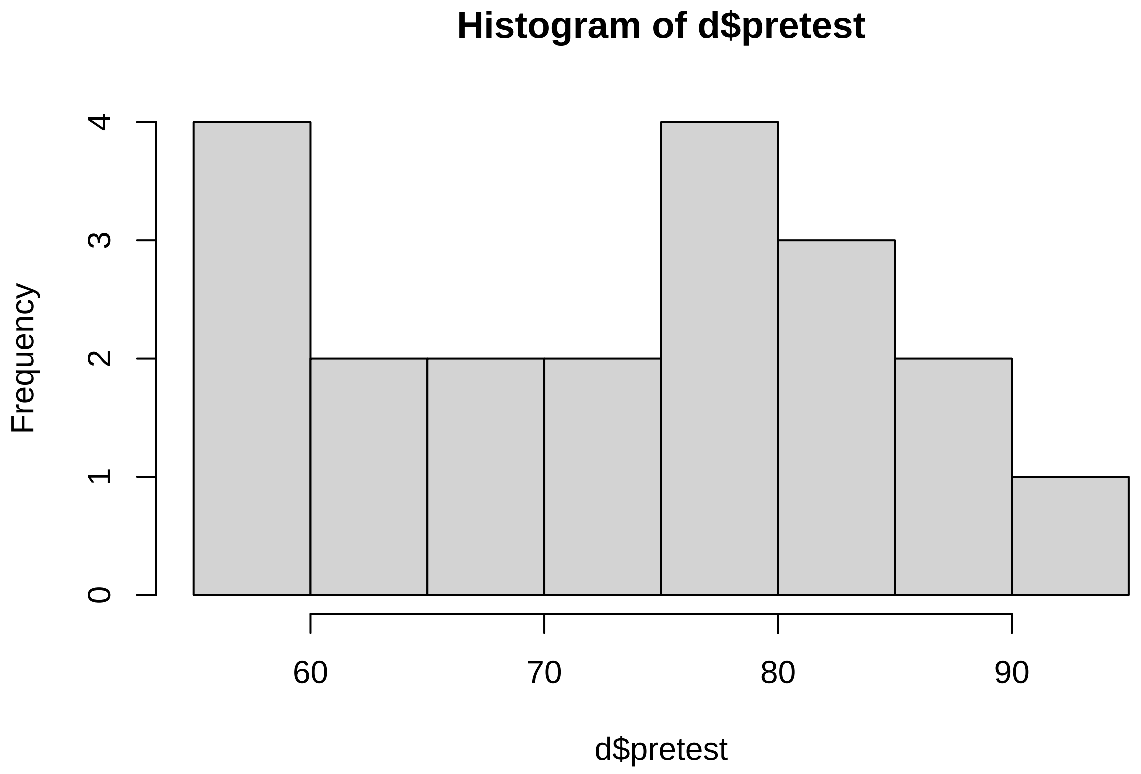
The output includes an array of six means (one for each of the majors in the treatment group and in the control group) and an array of six standard deviations.

Note that there are other ways to obtain the same results in a different format. To learn more, view the Intermediate lesson on the `dplyr` package and explore the `mutate()` and `summarize()` functions of this package.

Create a Histogram

As part of the initial data exploration process, you may wish to generate a histogram to examine the distribution of values for a variable. The most basic histogram on pretest scores, for example, is generated with this script:

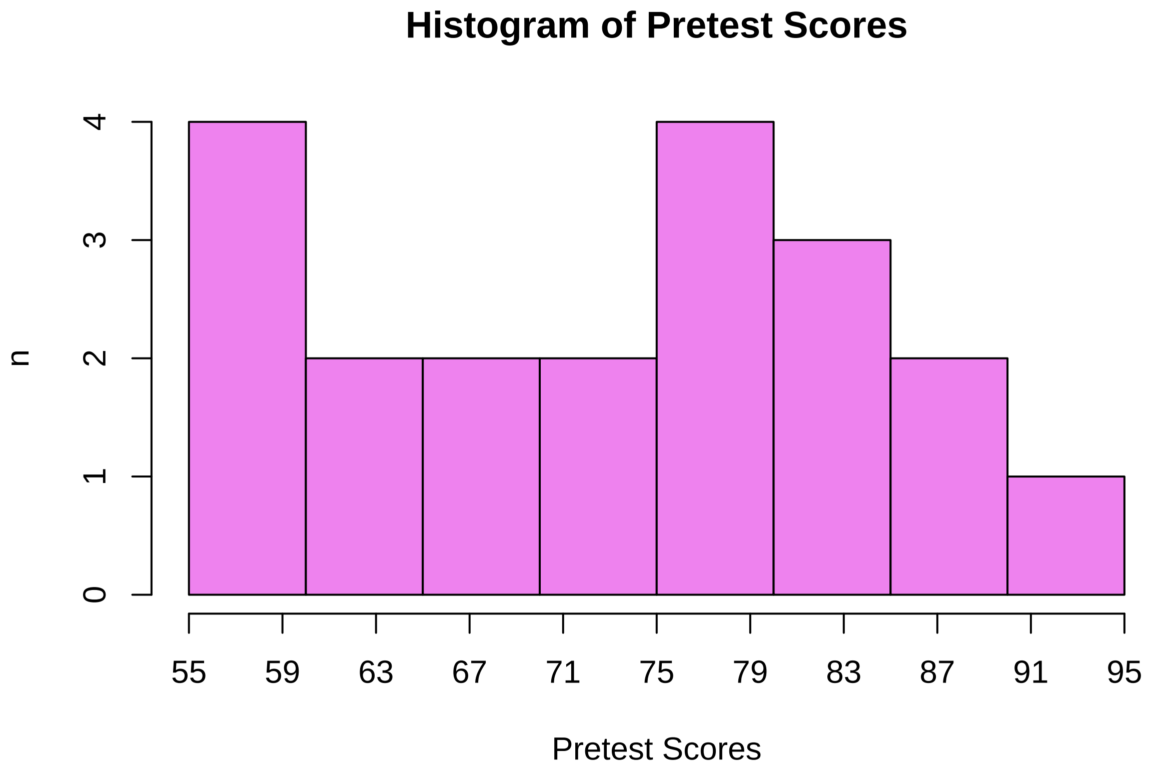
```
hist(d$pretest)
```



The plot shows the possibility of a bimodal distribution (i.e., a relatively large number of participants who scored in the lower range and a relatively large number of participants who scored in the upper range of the pretest).

Now you want to improve the aesthetic quality of the histogram by adding some arguments, including adjusting the range of values on the x axis, modifying the color, and clarifying the x and y axis labels as well as the main title of the plot.

```
hist(d$pretest,
     xaxp = c(55, 95, 10),
     col = "violet",
     xlab = "Pretest Scores",
     ylab = "n",
     main = "Histogram of Pretest Scores")
```



The additional arguments do not increase the number of bins in this case, but the actual values of the bins are clarfied. The color is more eye-catching, and the labels and titles are improved.

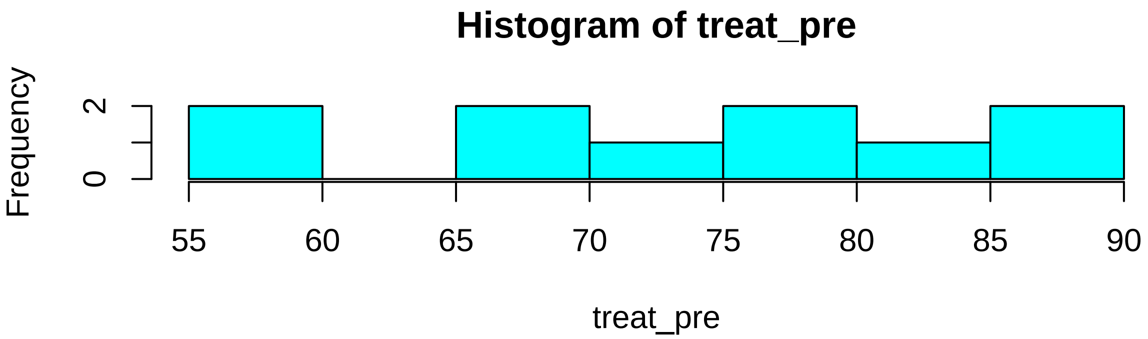
Create Histograms for Subgroups

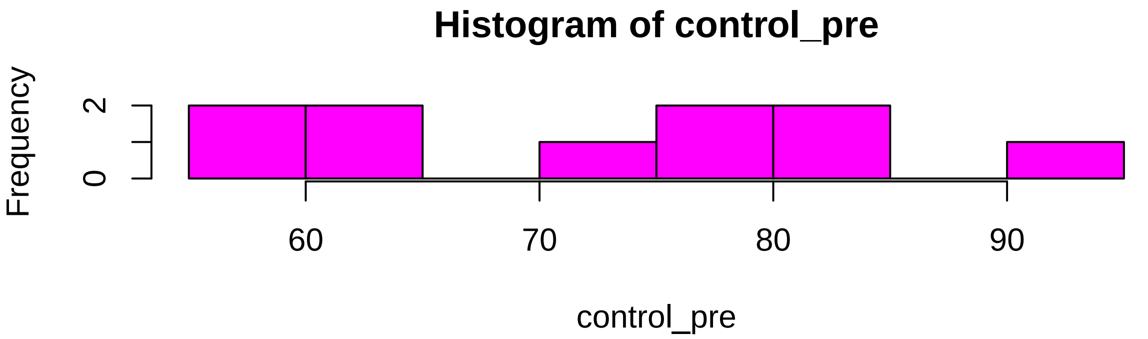
At times you will want to plot chunks of your data rather than data for the entire sample. For example, you and your research team want to look at the distribution of pretest scores for the treatment and control groups separately. You will first create two new objects (a vector of pretest scores for the treatment group and a vector of pretest scores for the control group).

```
treat_pre <- d[d$group == 1, "pretest"]
control_pre <- d[d$group == 2, "pretest"]
```

Next, you will specify coordinate parameters using `par()` so that both histograms appear in the same plot; the code for the two histograms is "sandwiched" between these parameter codes.

```
par(mfrow = c(2, 1))
hist(treat_pre, breaks = 10, col = "cyan")
hist(control_pre, breaks = 10, col = "magenta")
```





```
par(mfrow = c(1, 1))
```

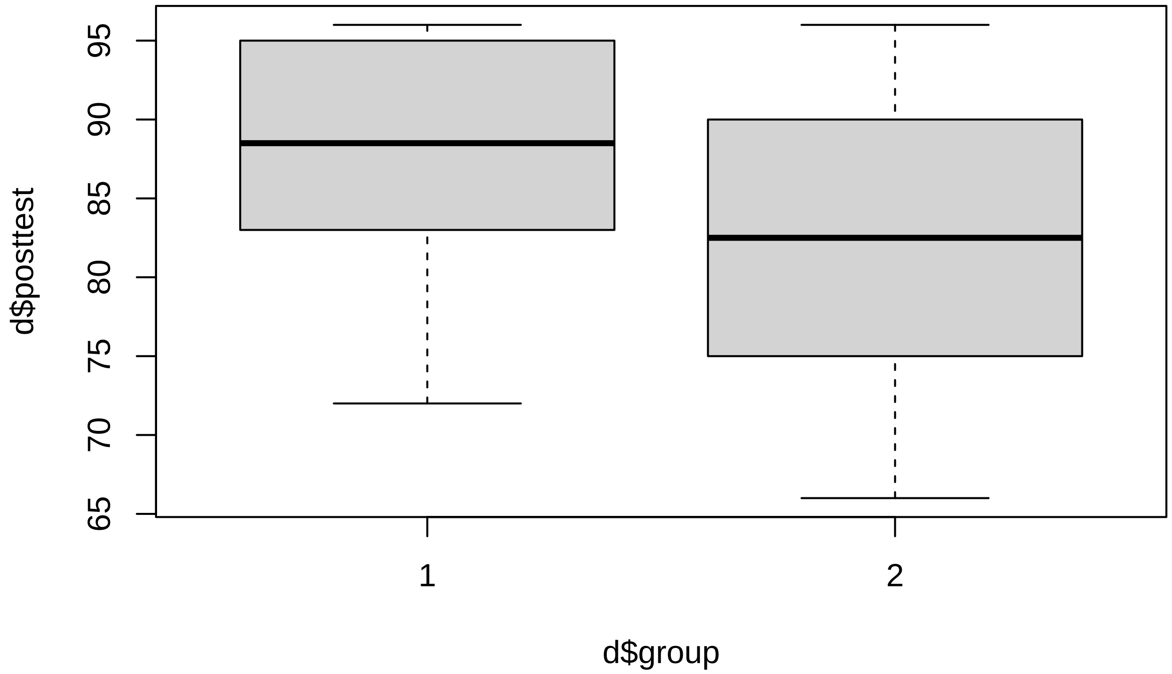
The output is two histograms; additional arguments could be added to adjust the labels and x axis, but this basic example illustrates the concept of plotting histograms for subgroups.

One note: Within the function `hist()`, the argument `breaks = 10` provides the number of bins that the data should be divided into. However, R treats this as a suggestion and will adjust the actual number of bins based on the overall number of cases and the frequency within each bin. Particularly with larger sample sizes, you may want to try out several different breaks (or different intervals within the xaxp argument) to determine which best represents the data.

Create a Boxplot

Another common exploratory visualization is a boxplot, which shows the median of a sample (or of subgroups), the interquartile range, and outliers. Begin by creating a basic boxplot of pretest scores for the two experimental groups separately:

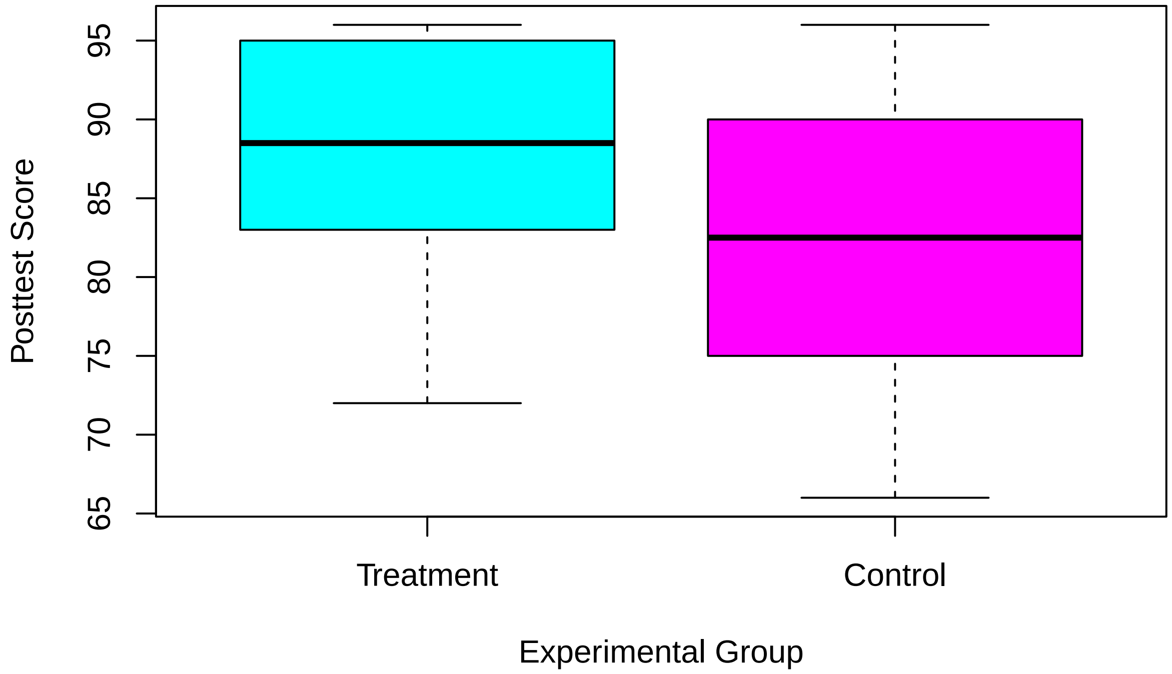
```
boxplot(d$posttest ~ d$group)
```



The output reveals that the treatment group (i.e., group 1) has a higher median posttest score than the control group (i.e., group 2).

Now add some arguments to make the boxplot more appealing:

```
boxplot(d$posttest ~ d$group,
        xlab = "Experimental Group",
        ylab = "Posttest Score",
        mean = "Statistics Posttest Scores by Group",
        col = c("cyan", "magenta"),
        names = c("Treatment", "Control"))
```



This is the same plot, but with color, group names, and improved labels and title.

Compare Two Group Means with an Independent Samples *t* Test

Your research team wants to know if the experimental groups differed significantly on the variable `d$difference`, i.e., the difference score (posttest - pretest). An independent samples *t* test can help to answer this question.

The function `t.test()` takes two arguments: the outcome (or dependent) variable and the grouping (or independent) variable, the latter of which should be a factor. Note that the two variables are separated with ~:

```
t.test(d$difference ~ d$group)
```

```
##
##  Welch Two Sample t-test
##
## data:  d$difference by d$group
## t = 2.6737, df = 17.417, p-value = 0.0158
## alternative hypothesis: true difference in means between group 1 and group 2 is not equal to 0
## 95 percent confidence interval:
##  0.9980282 8.4019718
## sample estimates:
## mean in group 1 mean in group 2
##            13.5             8.8
```

The output shows that the two groups differed significantly in the size of their mean difference scores, with the treatment group having a mean difference of 13.5 points from pre to post and the control group having a mean difference of 8.8 points from pre to post: $t(17.42) = 2.67$, $p = .02$. If these groups were randomly sampled from the population of all statistics students, you could conclude that there is only a small chance that you would obtain this large of a difference between groups if no such difference existed in the population.

Compare Two Means from the Same Group with a Paired Samples *t* Test

Your team is also curious whether the sample as a whole improved from pre to post, so now you want to conduct a paired samples *t* test. Once again, the function `t.test()` is used; however, this time the arguments are the first measure, the second measure, and paired = TRUE:

```
t.test(d$pretest, d$posttest, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  d$pretest and d$posttest
## t = -11.027, df = 19, p-value = 1.066e-09
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  -13.26644  -9.03356
## sample estimates:
## mean difference
##          -11.15
```

The output shows a significant difference between pretest and posttest scores for the entire sample, with an overall mean difference of 11.15 points: $t(19) =$ -11.03, $p < .0001$. (Note that the actual p value is much smaller, but convention encourages you to report to the ten-thousandths place at most.) Once again, it is unlikely that you would obtain such a large difference from pretest to posttest in the sample if such a difference did not exist in the population.

Compare Multiple Group Means with a One-way ANOVA

Many comparisons involve more than two groups. For example, there were three majors involved in the study: psychology, sociology, and political science. You want to know if there were any differences among these groups in terms of the difference scores (i.e., posttest - pretest). A one-way analysis of variance (ANOVA) allows to you make such comparisons:

```
major_anova <- aov(d$difference ~ d$major)
summary(major_anova)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## d$major      2   12.9    6.43   0.291  0.751
## Residuals   17  375.7   22.10
```

In this case, there were no significant differences among the means: $F(2, 11.32) = 0.29$, $p = 0.75$. However, if there were significant differences, additional posthoc tests would be needed. One option is `TukeyHSD()`, which takes as its argument the name of the ANOVA object you created (in this case, major_anova). Note, however, that you would not actually conduct Tukey HSD tests to follow up the current ANOVA result, as the latter was not significant.

Examine Relationships with a Correlation Coefficient

At times, you may want to examine the relationship between two continuous variables. For example, you and your team want to know if there is a relationship between participants' math scores and their scores on the measure of state anxiety (i.e., how anxious they felt at the time of the test). A correlation coefficient such as Pearson's product-moment correlation coefficient (r) is a good choice if the two variables are approximately normally distributed and if the relationship between them is linear. Calculate r for the two variables using `cor.test()`:

```
cor.test(d$math_score, d$state_anxiety, method = "pearson")
```

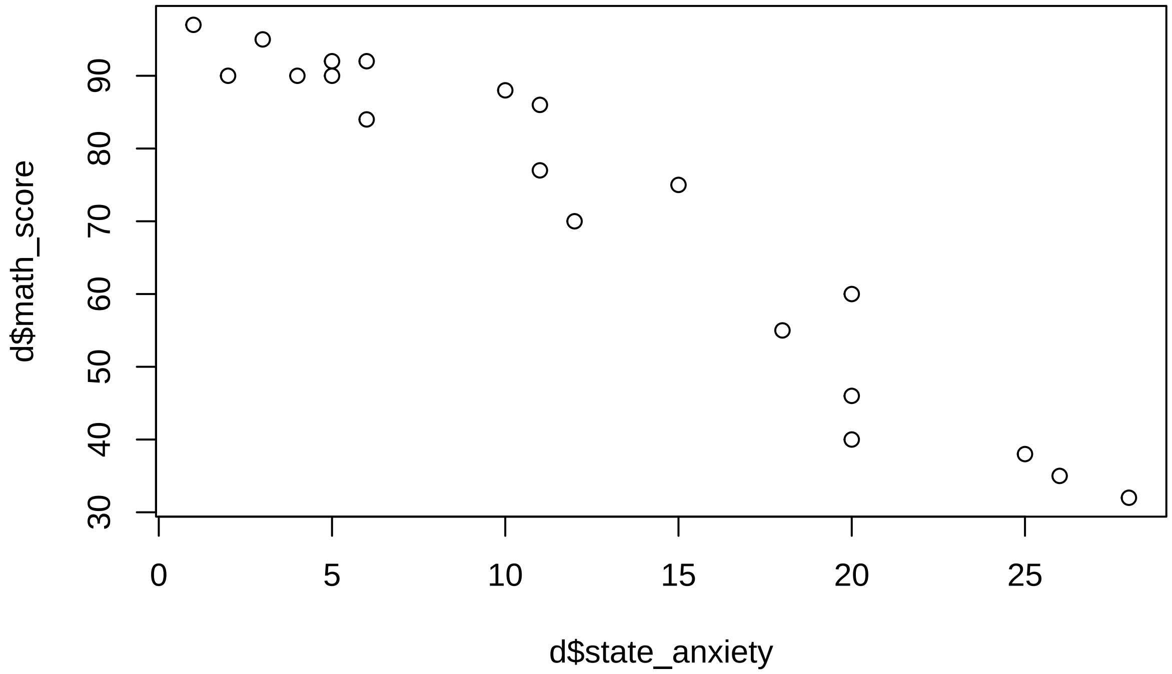
```
##
##  Pearson's product-moment correlation
##
## data:  d$math_score and d$state_anxiety
## t = -15.113, df = 18, p-value = 1.137e-11
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.9854500 -0.9064556
## sample estimates:
##        cor
## -0.9627806
```

The output shows that there is a very large (almost perfect) negative correlation between these two variables and that the relationship is significant: r = -.96, t(18) = -15.11, p < .0001. Such a large correlation would be unlikely in a real study, but it illustrates the concept of a strong relationship between two variables.

Create a Scatterplot

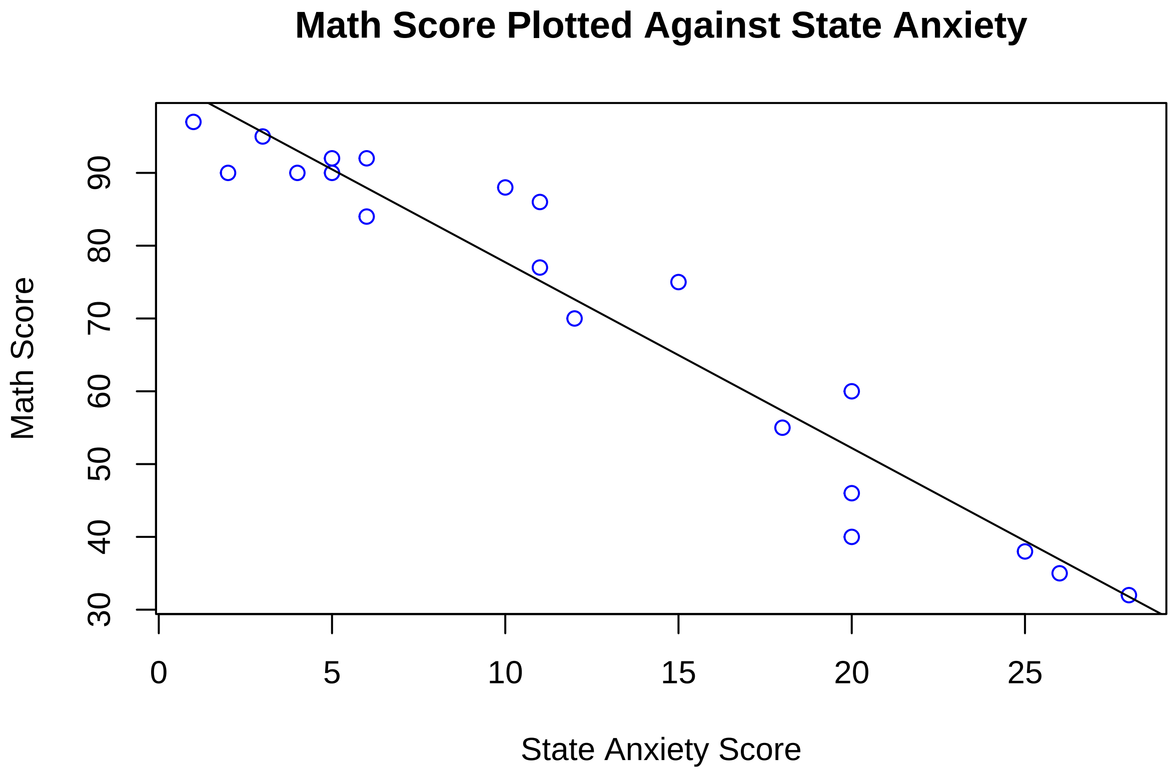
It is helpful to visualize the relationship between the variables to ensure that it is indeed linear and to look for outliers. In base R, the function `plot()` creates a scatterplot of two variables.

```
plot(d$math_score ~ d$state_anxiety)
```



The output shows a linear and strongly negative relationship between the two variables (as indicated by the correlation coefficient). Now add some arguments and a regression line to make the plot more informative:

```
plot(d$math_score ~ d$state_anxiety,
     col = "blue",
     main = "Math Score Plotted Against State Anxiety",
     xlab = "State Anxiety Score",
     ylab = "Math Score")
abline(lm(d$math_score ~ d$state_anxiety))
```



Note that the order of additional arguments (point color, main title, and axis labels) within `plot()` does not matter. The output is the same plot, with minor aesthetic improvements and the regression line that allows you to visualize the extent to which the dots deviate from the line.

Calculate Correlations Among Multiple Variables

So far, you have been working with the entire datarame, `d`. There are some functions that operate on the entire dataframe, so you will need to create a new dataframe that includes only the needed variables. For example, if you want to obtain a correlation matrix of all continuous variables, you will need to create an object that omits the factor variables.

Prepare the Data for Additional Analyses

Ensure that you have installed the `tidyverse` package with `install.packages()`. Then load the package and create a new object, `d2`.

```
library(tidyverse)
d2 <- select(d, -group, - major)
```

Create a Correlation Matrix

The function `cor()` is called upon an entire dataframe; because you have prepared the dataframe to include only continuous variables, you can call `cor()` on `d2`:

```
cor(d2)
```

```
##                 pretest   posttest trait_anxiety difference state_anxiety
## pretest        1.0000000  0.9211587    -0.7691706 -0.6123984    -0.8664105
## posttest       0.9211587  1.0000000    -0.8065450 -0.2564444    -0.9217842
## trait_anxiety -0.7691706 -0.8065450     1.0000000  0.2719390     0.6771587
## difference    -0.6123984 -0.2564444     0.2719390  1.0000000     0.2793539
## state_anxiety -0.8664105 -0.9217842     0.6771587  0.2793539     1.0000000
## math_score     0.8045917  0.8642849    -0.5539325 -0.2426224    -0.9627806
##               math_score
## pretest        0.8045917
## posttest       0.8642849
## trait_anxiety -0.5539325
## difference    -0.2426224
## state_anxiety -0.9627806
## math_score     1.0000000
```

The output is a matrix of all intercorrelations. Note that the values above and below the diagonal are identical. There are very large correlations among many of the variables (a phenomenon called multicollinearity), which could be problematic in later analyses. This is something to keep in mind.

Predict an Outcome with Two Predictors

The final task is to conduct a regression analysis, predicting an outcome variable with two predictor variables. You and your team are noticed that math score was related to both trait anxiety and state anxiety, with a particularly large correlation between math score and state anxiety. You are curious whether trait anxiety adds anything to the prediction of math score. Multiple regression can be used to anxer this question.

The function `lm()`, which you saw earlier in the context of creating a regression line for a scatterplot, will fit a linear model onto the data in `d2`. (Note that you can use the larger dataframe, `d`, if you prefer.) When you save the results as an object, you can obtain the statistics for the regression.

```
model <- lm(d2$math_score ~ d2$state_anxiety + d2$trait_anxiety)
summary(model)
```

```
##
## Call:
## lm(formula = d2$math_score ~ d2$state_anxiety + d2$trait_anxiety)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -9.946 -3.010 -1.240  3.368 10.092
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      100.9716     2.4618  41.016  < 2e-16 ***
## d2$state_anxiety  -2.8771     0.2055 -14.002 9.18e-11 ***
## d2$trait_anxiety   0.7880     0.3374   2.335    0.032 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.651 on 17 degrees of freedom
## Multiple R-squared:  0.9447, Adjusted R-squared:  0.9382
## F-statistic: 145.2 on 2 and 17 DF,  p-value: 2.059e-11
```

The output shows the coefficients for the intercept and each predictor, along with their significance. Although `state_anxiety` is clearly the best predictor of `math_score`, `trait_anxiety` is still a significant predictor in this model, suggesting the possibility of a unique contribution of trait anxiety to the variance in math score.

In addition, the output includes multiple R-squared and adjusted R-squared, along with the F test of the linear model.

Check Regression Assumptions

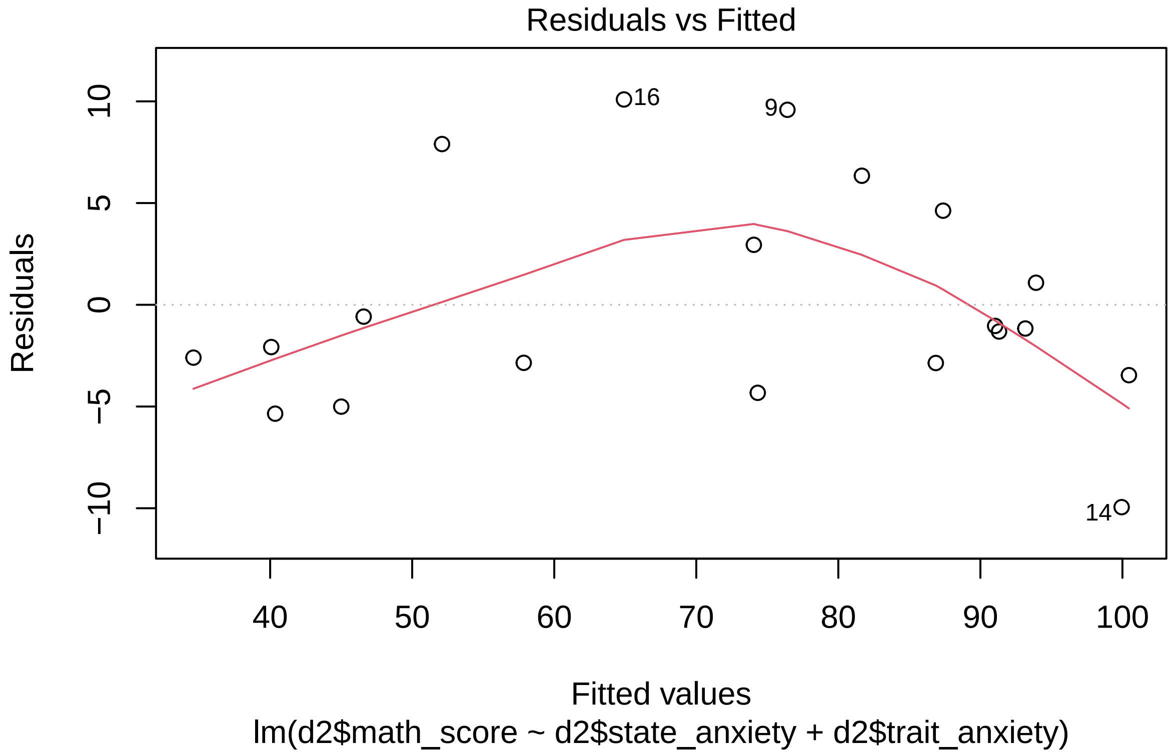
The final step is to examine the assumptions of regression:

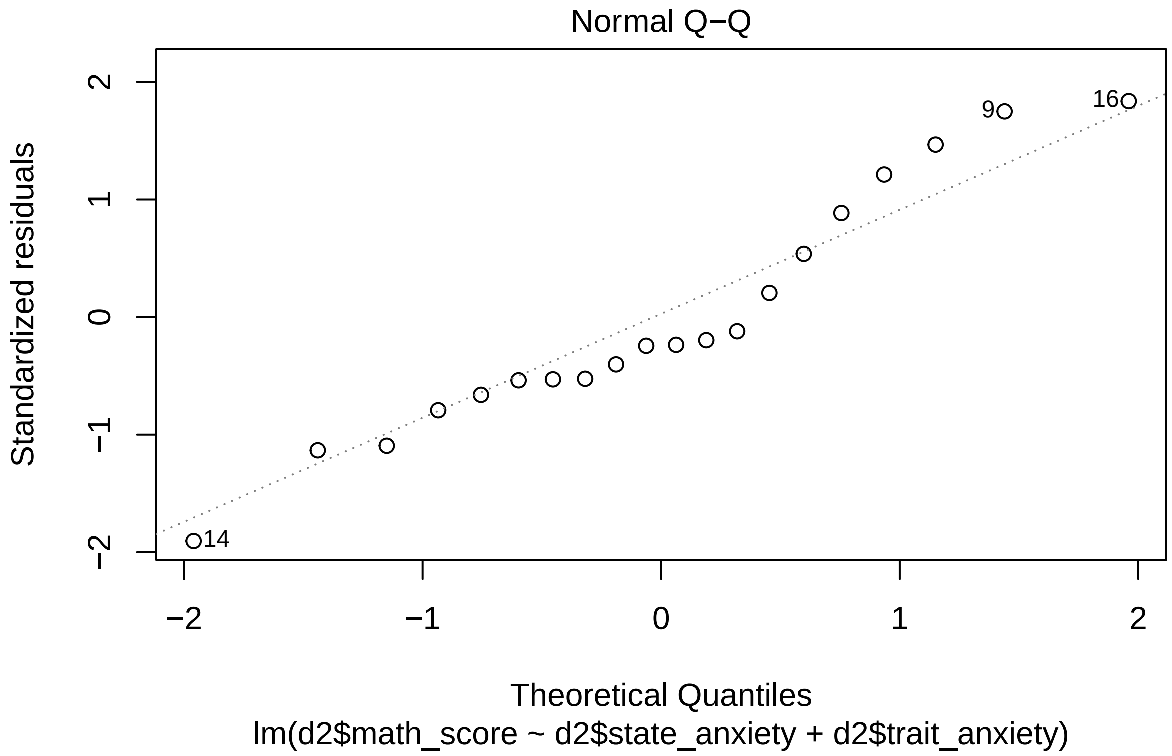
1. Linearity: Is the relationship between each predictor and the outcome linear?
2. Homoscedasticity: Is the variance of the residuals the same for any value of X?
3. Normality: For any value of X, is Y normally distributed?

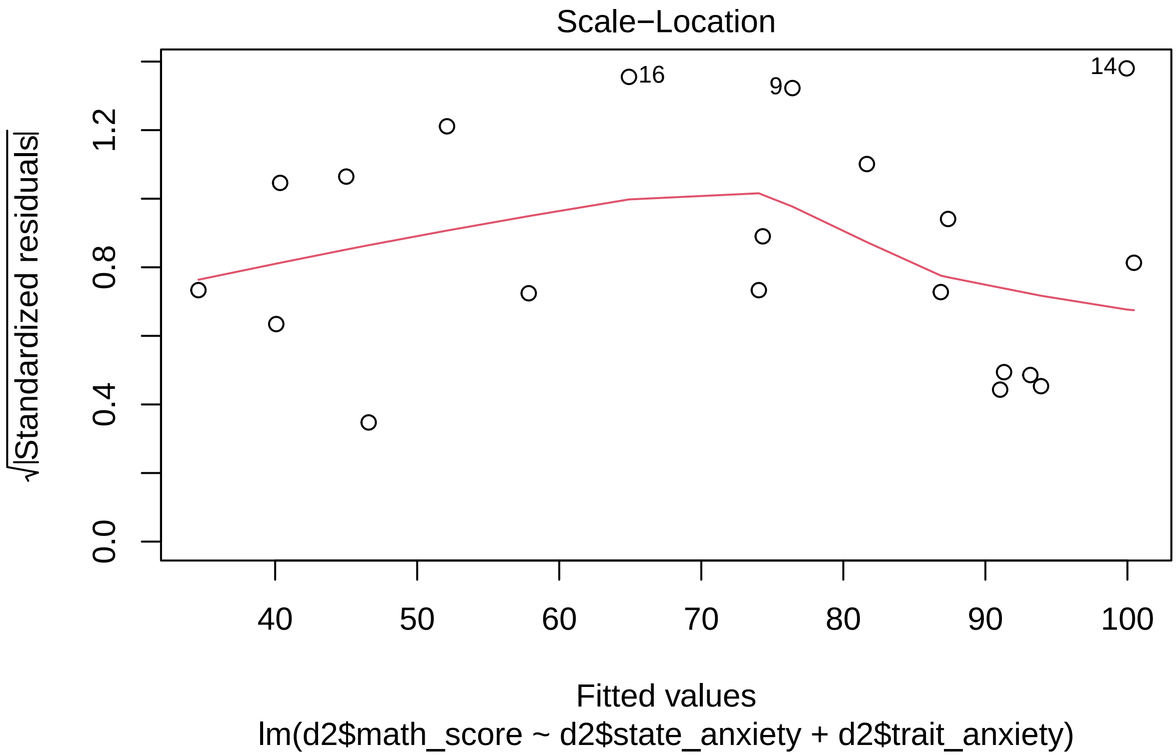
Additional assumptions include independence of observations and random sampling, which you would need to confirm with a review of your research methods.

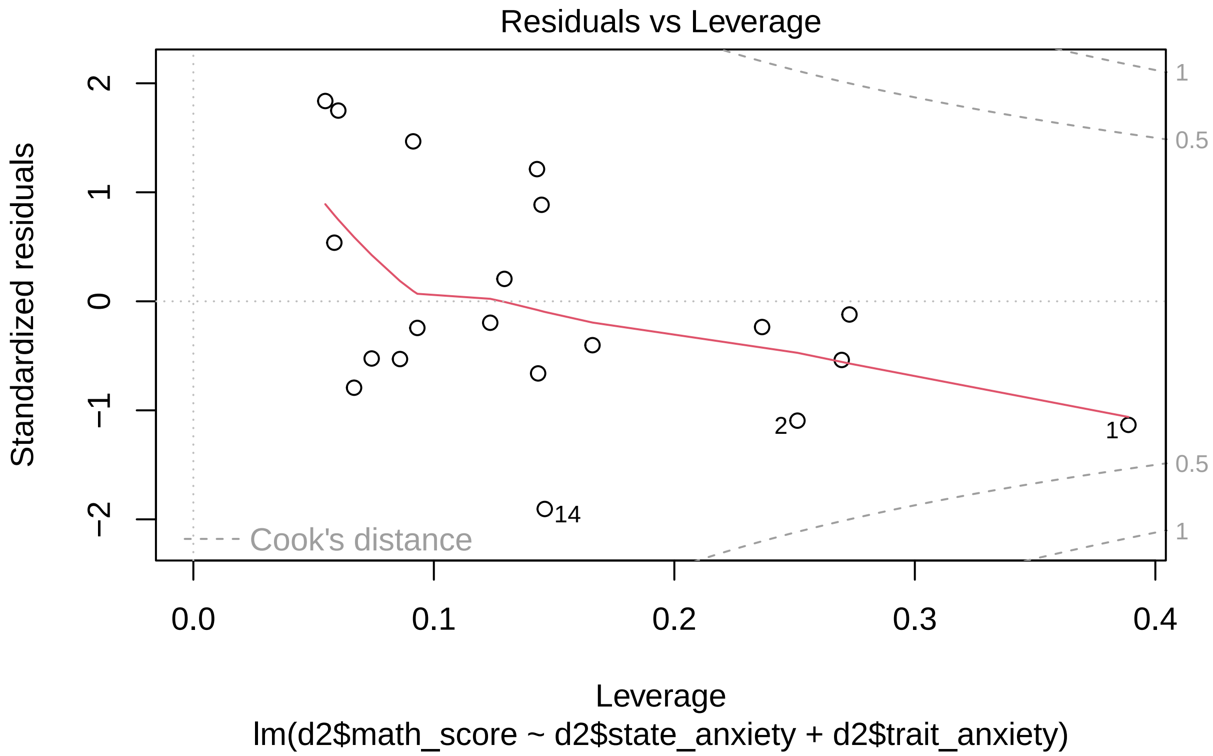
There are four built-in plots for checking regression assumptions and diagnosing the quality of the model. These plots can be called with `plot()` on the name of the model (in this case, `model`).

```
plot(model)
```




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(d2$math_score ~ d2$state_anxiety + d2$trait_anxiety)
−2
−1
0
1
2
−2
−1
0
1
2
14
9
16


Scale−Location
√|Standardized residuals|
0.0
0.4
0.8
1.2
40
50
60
70
80
90
100
16
9
14
Fitted values
lm(d2$math_score ~ d2$state_anxiety + d2$trait_anxiety)



Interpretation of Plots and Conclusions

The first plot regresses the residuals on the fitted values. In a "good" model, the points would be randomly scattered above and below the horizontal line; in the current model, however, there is a slightly parabolic pattern to the residuals that suggests a potential problem with linearity.

The second plot is a Q-Q plot of the standardized residuals, which addresses normality. The points should closely adhere to the regression line. In contrast, many of the points in the current model are rather far from the the line. Note that R will label points that may be particularly influential in a problematic way.

The third plot, called a scale-location or spread-location plot, examines homoscedasticity–whether the residuals are spread equally along all values of the predictors. A roughly horizontal line with evenly distributed points is desirable. In the current model, the points appear to follow a curve, suggesting lack of homoscedasticity.

The final plot examines leverage, or the degree to which outliers influence the regression model. In this plot, points that are in the upper right or upper left corners (outside the dashed line, which indicates Cook's distance, a measure of leverage) are potentially problematic. In the current model, no points fall within these areas, so this assumption is not violated.

What can you conclude from these diagnostics? Although the two predictors account for a very large proportion of the variance in outcome (math score), this model may not be the best fit. Additional data exploration and transformation may be needed.

References and Recommended Reading

- Kim, B. (2015, September 21). *Understanding diagnostic plots for linear regression analysis.* University of Virginia Library - Research Data Services + Sciences. https://data.library.virginia.edu/diagnostic-plots/
- R Project for Statistical Computing. (n.d.). https://www.r-project.org/
- Teetor, P. (2011). *R cookbook.* O'Reilly.

Contact Information

Thank you for participating in this lesson. If you have questions, please reach out to cb88@rice.edu.