Lunch Hour Learning Guide, Sessions 9-10, Spring 2025

Basic Statistics in R - Parts 1 and 2

Sean Morey Smith

2025-04-09

Before Starting

1. Create a new, self-contained R project where you will store your work from this session. For guidance, see the instructions from Session 1.
2. Create a sub-directory (folder) called "data" in your project directory.
3. Unzip the .csvs in https://library.rice.edu/sites/default/files/materials/data.zip into the "data" sub-directory.

Session 9: Basic Statistics in R - Part 1

What You Will Learn

- Obtain descriptive statistics for continuous variables
- Create contingency tables for categorical variables
- Conduct a chi-square test
- Conduct t tests (independent and dependent)
- Conduct analysis of variance (ANOVA) and post-hoc tests

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 (i.e., how anxious someone is in general), a 30-point measure of state anxiety (i.e., how anxious someone is in the moment), and a 100-point measure of college-level math knowledge. They also indicated their major category: Humanities, Natural Sciences, or Social Sciences.

You and your team then randomly assigned participants to one of two groups:

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

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

```
scores <- read.csv("data/experiment_data.csv", stringsAsFactors = TRUE)
```

You should see the data frame scores in your Global Environment. Next, take a look at this object's structure.

Examine the Structure of the Data Frame

```
str(scores)
```

```
## 'data.frame':    20 obs. of  9 variables:
##  $ ID           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ group        : Factor w/ 2 levels "Control","Treat": 2 2 2 2 2 2 2 2 2 2 ...
##  $ 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 "H","NS","SS": 3 2 1 3 2 1 3 2 1 3 ...
##  $ 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 data frame has 20 observations of 9 variables.

Remove Unnecessary Variables

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

```
df <- scores[,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 data frame named `df`, 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 (columns) in the dataset.

```
summary(df)
```

```
##      group       pretest         posttest     trait_anxiety     difference
##  Control:10   Min.   :55.00   Min.   :66.00   Min.   : 1.00   Min.   : 4.00
##  Treat  :10   1st Qu.:63.50   1st Qu.:79.25   1st Qu.: 3.75   1st Qu.: 6.75
##               Median :75.00   Median :86.50   Median : 7.50   Median :12.00
##               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
##  major  state_anxiety    math_score
##  H :6   Min.   : 1.0   Min.   :32.00
##  NS:7   1st Qu.: 5.0   1st Qu.:52.75
##  SS:7   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(df$difference)
```

```
## [1] 11.15
```

```
median(df$difference)
```

```
## [1] 12
```

```
min(df$difference)
```

```
## [1] 4
```

```
max(df$difference)
```

```
## [1] 20
```

```
sd(df$difference)
```

```
## [1] 4.522168
```

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

Obtain Descriptive Statistics for Subgroups

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(df$difference, df$group, mean, na.rm = TRUE)
```

```
## Control   Treat
##     8.8    13.5
```

```
tapply(df$difference, df$group, sd, na.rm = TRUE)
```

```
##  Control    Treat
## 3.552777 4.275252
```

Note that the output of each `tapply` is a small array (like a vector but with any number of dimensions) – the means and standard deviations for the two groups. The columns of the array are named according to the factor level the value applies to.

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 on pretest scores 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(df$pretest, list(df$group, df$major), mean, na.rm = TRUE)
```

```
##                H    NS   SS
## Control 70.66667 77.25 70.0
## Treat   82.66667 64.00 75.5
```

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

```
##                 H      NS        SS
## Control 18.583146 9.17878  8.888194
## Treat    7.023769 7.81025 11.818065
```

In each case, `tapply` outputs an array of six means values - first the mean, then the standard deviation. Each array contains a value for each of the majors in both the treatment group and the control group. The array's rows are named by the levels of the first factor in the list, and the columns are named as the levels of the second factor.

Note that there are other ways to obtain the same results in a different format. For example, the `dplyr` package has functions called `mutate()` and `summarize()`, which perform the same calculations and save the results as a tibble (data frame). It is helpful to know more than one way to obtain the same results! According to some sources, `tapply()` is faster than the `dplyr` functions. If you are working with extremely large datasets, it may be helpful to use `tapply()`.

Create Contingency Tables

Next, you may want to know how two categorical variables relate to each other. For example, in the current experiment, are the three majors equally distributed across the treatment and control groups?

```
(group_table <- table(df$group, df$major))
```

```
##
##           H NS SS
##   Control 3  4  3
##   Treat   3  3  4
```

The entire code is wrapped in parentheses, which prints the object you have just created.

The output suggests even distribution across the two levels of group. Note that the `table()` function will ignore `NA` values. If you decide to save the output to an object, as the code here shows, the object will be a table of values.

As a bonus, here is some code to create proportion tables based on the frequency table you just created:

```
prop.table(group_table) #cell percentages
```

```
##
##              H   NS   SS
##   Control 0.15 0.20 0.15
##   Treat   0.15 0.15 0.20
```

```
prop.table(group_table, 1) #row percentages
```

```
##
##             H  NS  SS
##   Control 0.3 0.4 0.3
##   Treat   0.3 0.3 0.4
```

```
prop.table(group_table, 2) #column percentages
```

```
##
##                   H        NS        SS
##   Control 0.5000000 0.5714286 0.4285714
##   Treat   0.5000000 0.4285714 0.5714286
```

Conduct a Chi-square Test

In the table generated by the previous code, you saw that the number of H, NS, and SS majors within the two groups (Treat and Control) were almost identical. In most situations, particularly when categorical variables are naturally occurring (e.g., demographics) rather than experimentally manipulated (e.g., experimental conditions), it is unlikely that such perfectly even distribution will occur.

The question is, are the differences seen in a randomly drawn sample much larger than you might expect to see if you could sample many groups from the population and measure their differences?

A chi-square test is an inferential statistical test of whether the distribution of one categorical variable significantly differs according to the levels of another categorical variable. It is based on the difference between the observed proportion of observations in a each category and the expected proportion of observations.

To illustrate this concept with the current data, first you will recode one of the continuous variables to be categorical. For example, you decide to code scores of 80 or above on the posttest as Pass and those below 80 as Fail.

```
df$post_cat <- ifelse(df$posttest >= 80, "Pass", "Fail")
```

Next, check the contingency table of this variable against experimental group. Save the output to a new table object.

```
post_cat_table <- table(df$group, df$post_cat)
post_cat_table
```

```
##
##           Fail Pass
##   Control    4    6
##   Treat      1    9
```

The results show that, among control group participants, 4 failed and 6 passed; among treatment group participants, 1 failed and 9 passed. The control group had a 60% pass rate, while the treatment group had a 90% pass rate. These numbers (and their associated proportions) are clearly different from each other in this sample. But how different are they? In other words, if you could sample repeatedly from the larger null population of statistics students (null meaning that in this hypothetical population, the two groups did not differ in their pass rates) and perform the same measures, how often would you see this big of a difference in the samples?

You are now ready to conduct the chi-squared test to answer this question!

```
chisq.test(post_cat_table, correct = TRUE)
```

```
## Warning in chisq.test(post_cat_table, correct = TRUE): Chi-squared
## approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  post_cat_table
## X-squared = 1.0667, df = 1, p-value = 0.3017
```

We call the function `chisq.test()` on the table object and include the argument `correct = TRUE` because we have very small cell sizes (i.e., number of observations in each cell of the table). The additional argument will include a continuity correction.

In the output, `X-squared` is the Chi-square, `df` are the degrees of freedom, and `p-value` is the probability value, a measure of how likely our data would have occurred under the null hypothesis.

Here, the p-value of 0.3017 is pretty high (generally we want <= 0.05), so the result is a non-significant chi-squared value, indicating that in such small samples as these, it is not uncommon to see large differences in distributions. However, the warning message indicates that the chi-squared approximation may be incorrect. This is because a small sample may provide inaccurate results. For example, one of the cells has only one observation. Chi-squared does not perform well with such small cell sizes.

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

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

The function `t.test()` takes a formula as an argument with the syntax `dependent_var ~ independent_var`. Note that there's a tilde (~) in between the two variables, that the outcome (or dependent) variable is on the left, that the grouping (or independent) variable is on the right, and that the grouping variable should be a factor.

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

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

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.

Note that the output of the t test does not give you standard deviations, but you can quickly use `tapply()` to obtain that information for each group.

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 we don't give it a formula argument. Instead, the arguments are the first measure, the second measure, and `paired = TRUE`:

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

```
##
##  Paired t-test
##
## data:  df$pretest and df$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 major disciplines involved in the study: humanities, natural science, and social 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(df$difference ~ df$major)
summary(major_anova)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## df$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.

Nonetheless, it is useful to see the code and output for this function!

```
TukeyHSD(major_anova)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = df$difference ~ df$major)
##
## $`df$major`
##             diff       lwr      upr     p adj
## NS-H   1.9761905 -4.733237 8.685618 0.7344083
## SS-H   0.8333333 -5.876094 7.542761 0.9457514
## SS-NS -1.1428571 -7.589067 5.303352 0.8929417
```

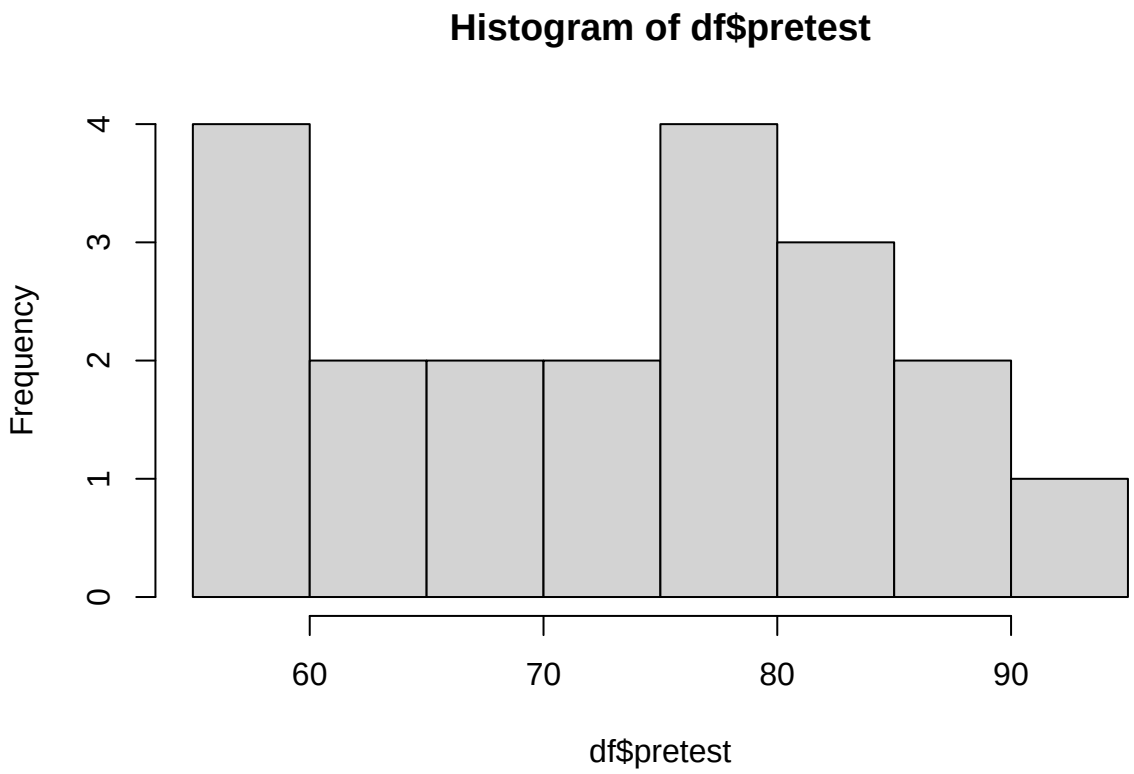
As expected for the post-hoc tests on a non-significant omnibus ANOVA model, the pair-wise comparisons were also non-significant.

Bonus: Simple Exploratory Data Visualization

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:

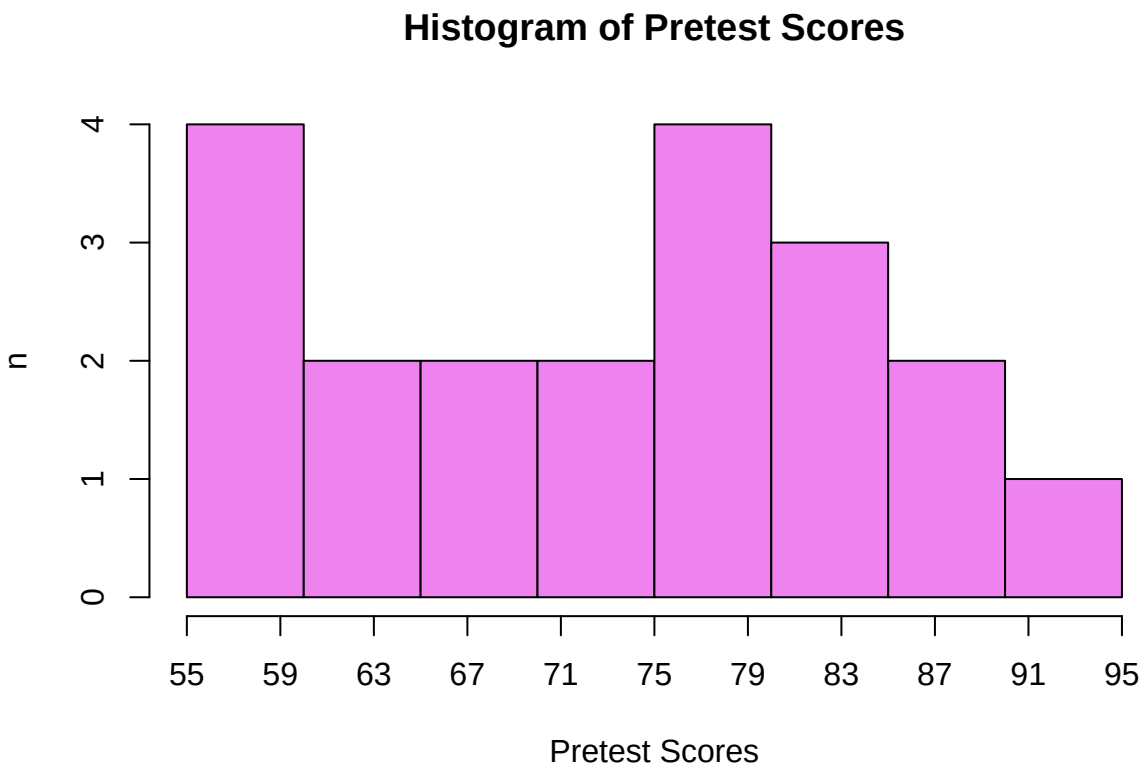
```
hist(df$pretest)
```



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(df$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.

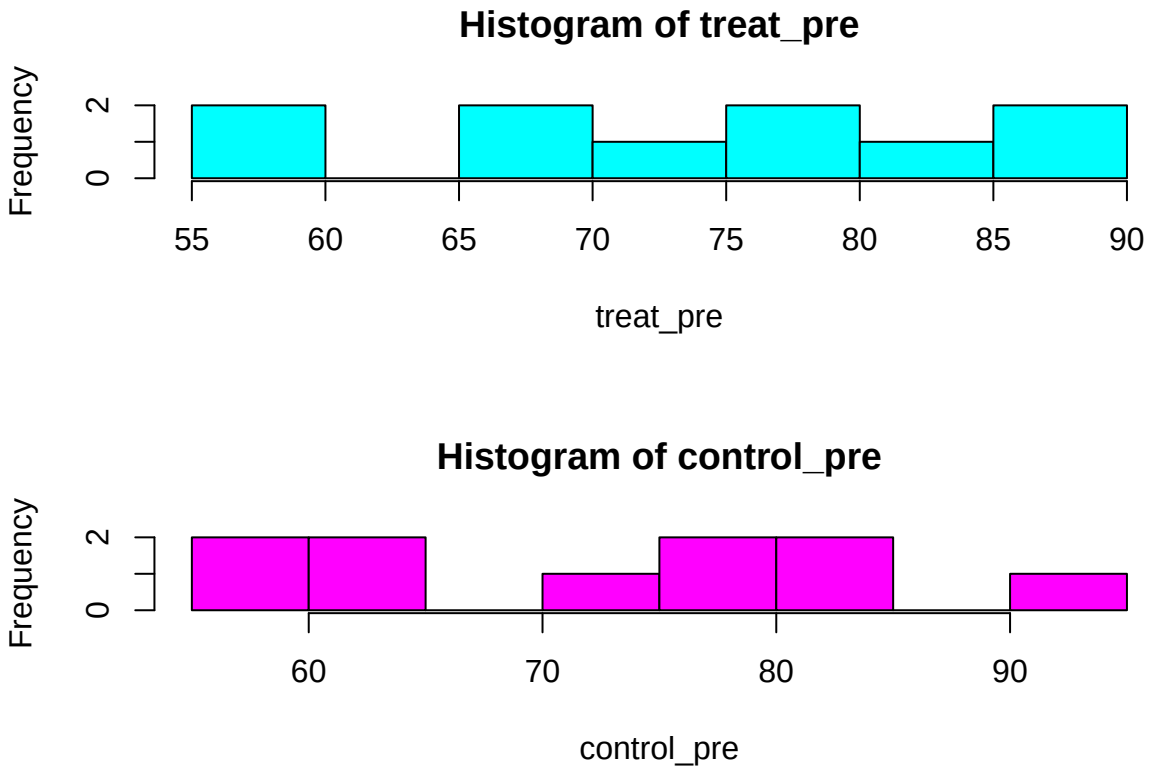
Create Histograms for Subgroups

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 <- df[df$group == "Treat", "pretest"]
control_pre <- df[df$group == "Control", "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))
```

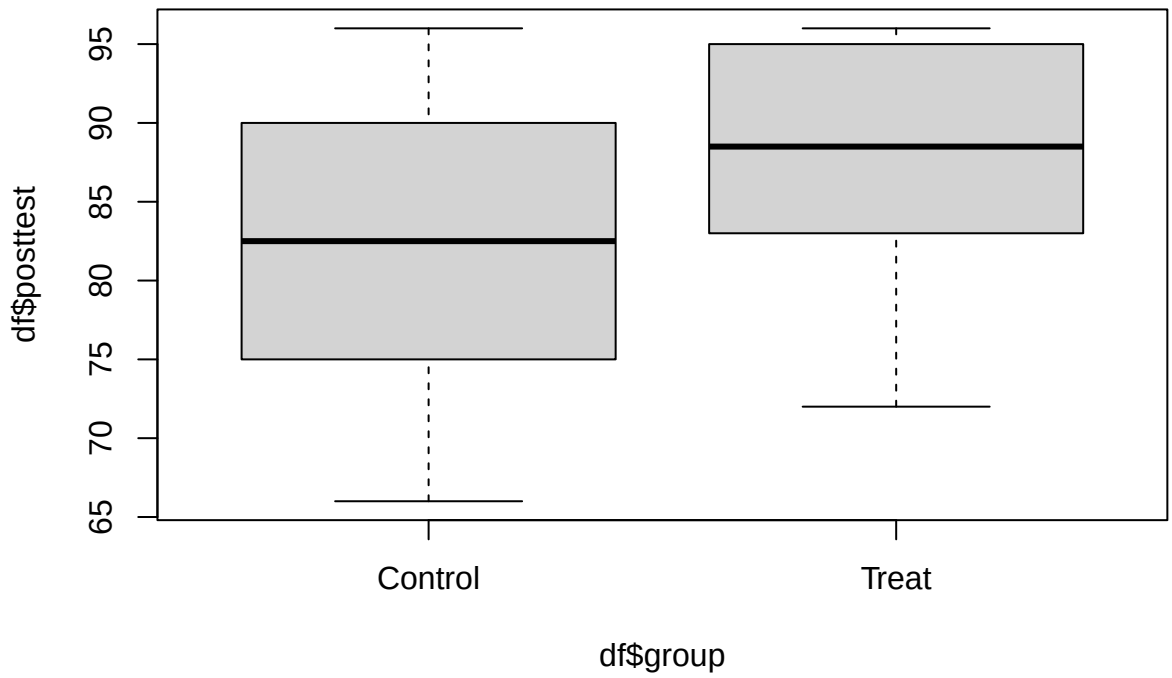
The output is two histograms; additional arguments could be added to adjust the labels and x axis, but this basic example illustrates the concept of plotting histograms for subgroups.

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:

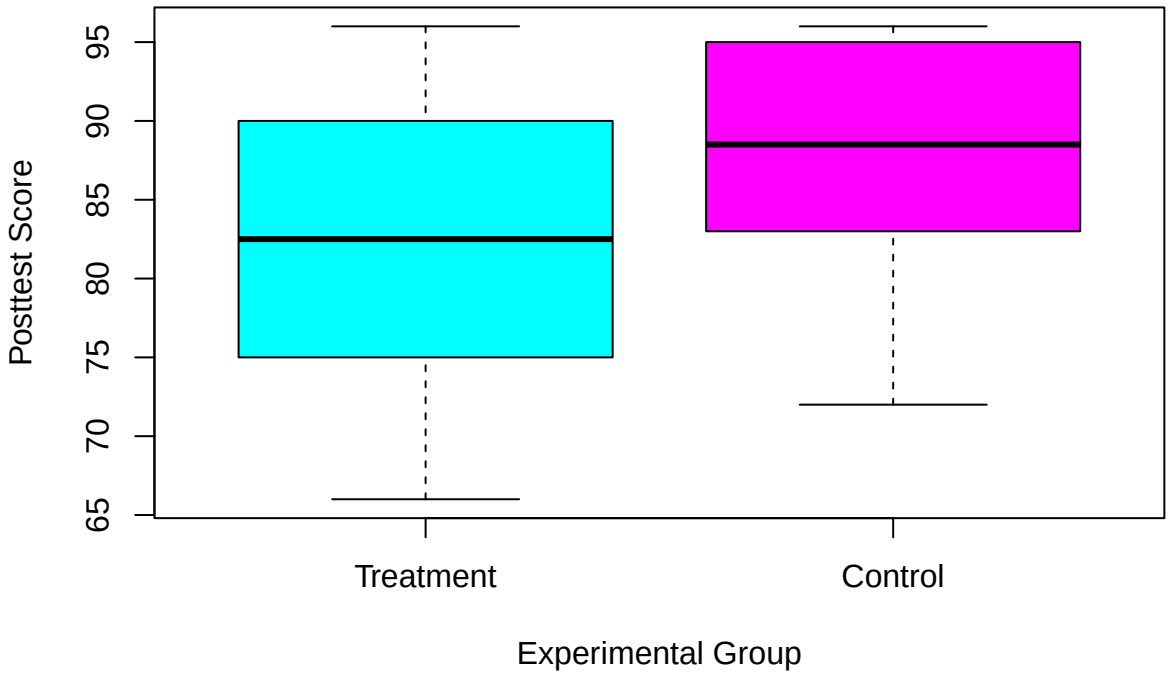
```
boxplot(df$posttest ~ df$group)
```



The output reveals that the treatment group has a higher median posttest score than the control group.

Now add some arguments to make the boxplot more visually interesting:

```
boxplot(df$posttest ~ df$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.

Session 10: Basic Statistics in R - Part 2

What You Will Learn

- Calculate correlation coefficients
- Conduct a linear multiple regression analysis
- Examine assumptions of regression
- Conduct a logistic regression analysis

Import the Dataset

If you're starting with Part 2 and haven't loaded the data already, load it and remove an extra "ID" variable:

```
scores <- read.csv("data/experiment_data.csv", stringsAsFactors = TRUE)
df <- scores[, c(2:9)]
```

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(df$math_score, df$state_anxiety, method = "pearson")
```

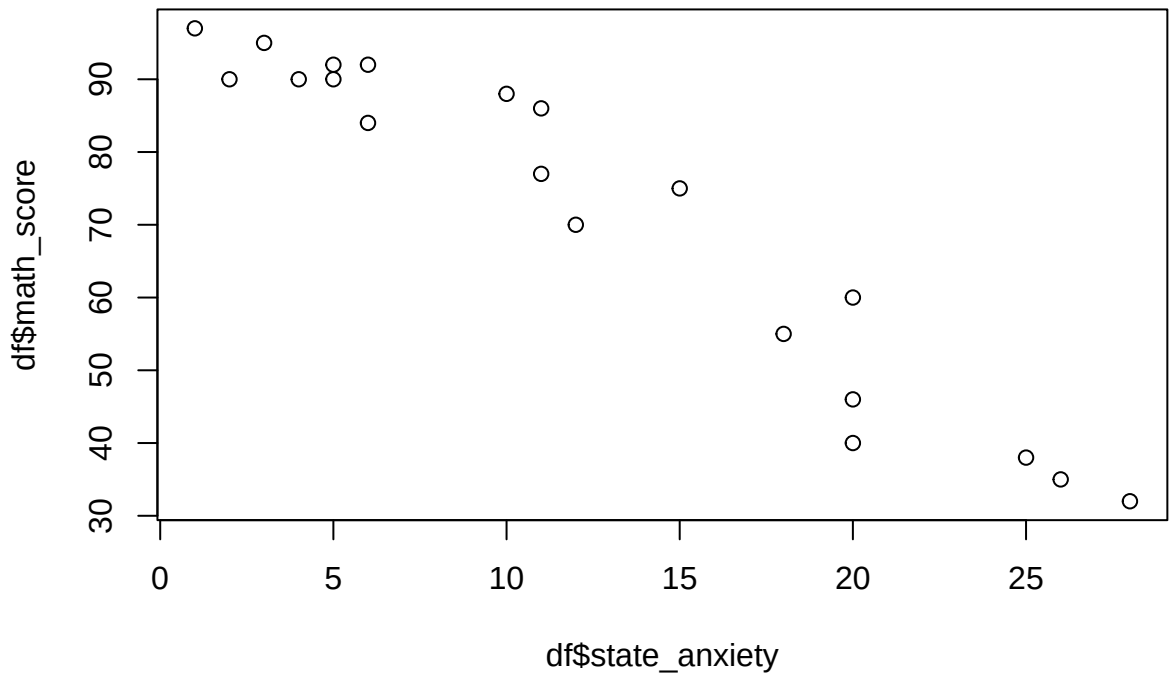
```
##
##  Pearson's product-moment correlation
##
## data:  df$math_score and df$state_anxiety
## t = -15.113, df = 18, p-value = 1.137e-11
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.9854500 -0.9064556
## sample estimates:
##        cor
##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

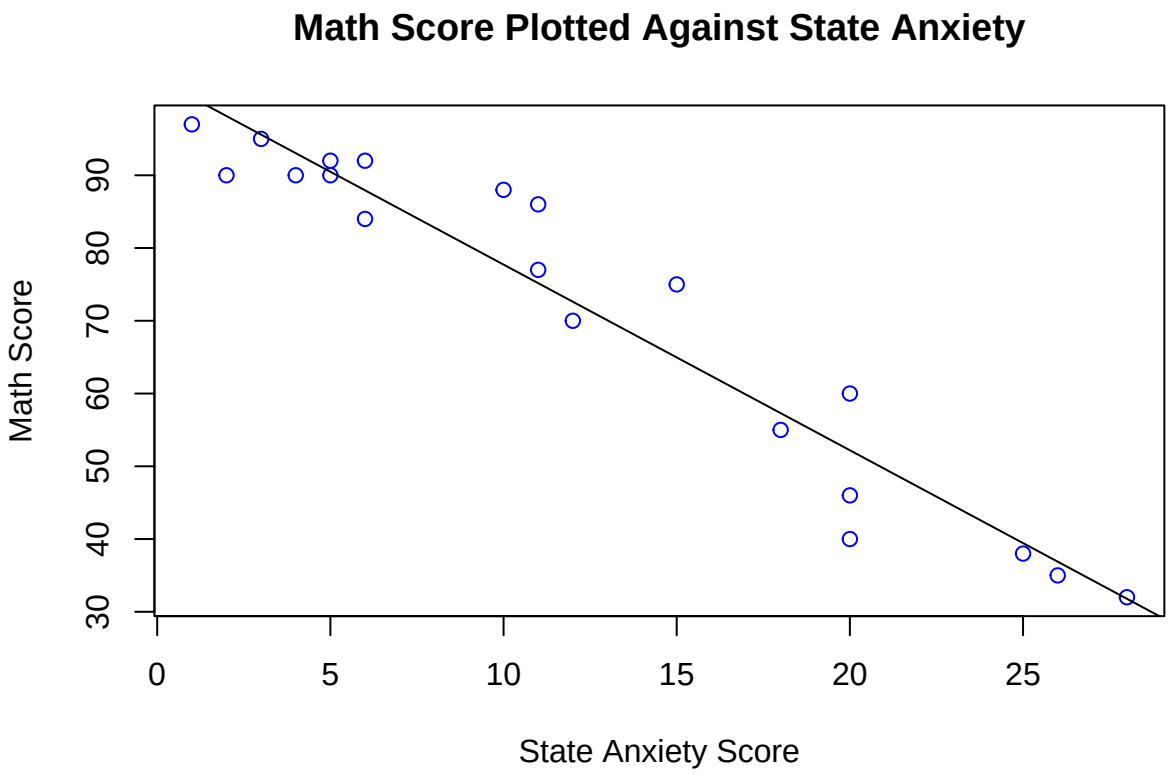
It is helpful to visualize the relationship between the variables to ensure that it is indeed linear and to look for outliers. In base R, the function `plot()` creates a scatterplot of two variables.

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



Note that the order of additional arguments (point color, main title, and axis labels) within `plot()` does not matter since we used their names. 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 data frame, `df`, but using a formula (`df$math_score ~ df$state_anxiety`) to specify which variables to compare. There are some functions that operate on the *entire* data frame, so if you only want to compare some of the variables, you will need to create a new data frame 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, `df2`.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
df2 <- select(df, -group, - major)
# if you continued from Part 1 and didn't load the data at the start of Part 2,
# you'll also need to remove post_cat:
#df2 <- select(df2, -post_cat)
```

Create a Correlation Matrix

The function `cor()` is called upon an entire data frame; because you have prepared the data frame to include only continuous variables, you can call `cor()` on `df2`:

```
cor(df2)
```

```
##                  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 next 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 answer 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 `df`. When you save the results as an object, you can obtain the statistics for the regression.

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

```
##
## Call:
## lm(formula = df$math_score ~ df$state_anxiety + df$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 ***
## df$state_anxiety  -2.8771     0.2055 -14.002 9.18e-11 ***
## df$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

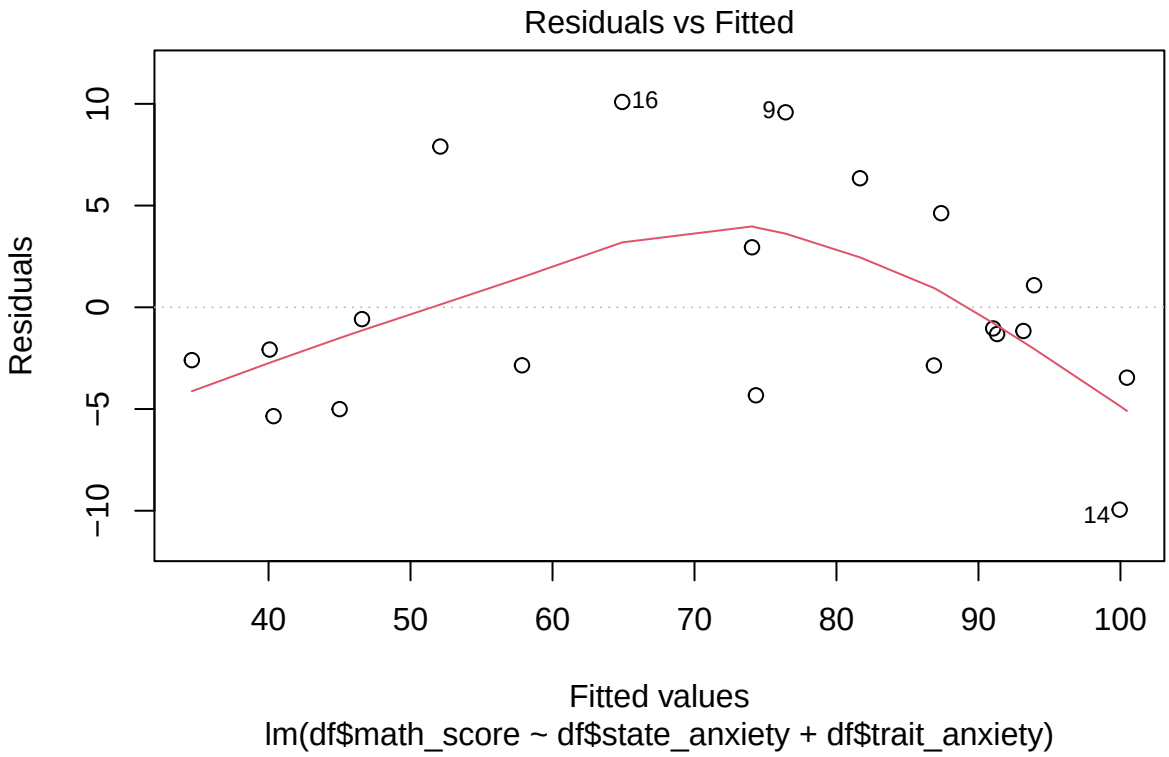
The final step is to examine the assumptions of regression:

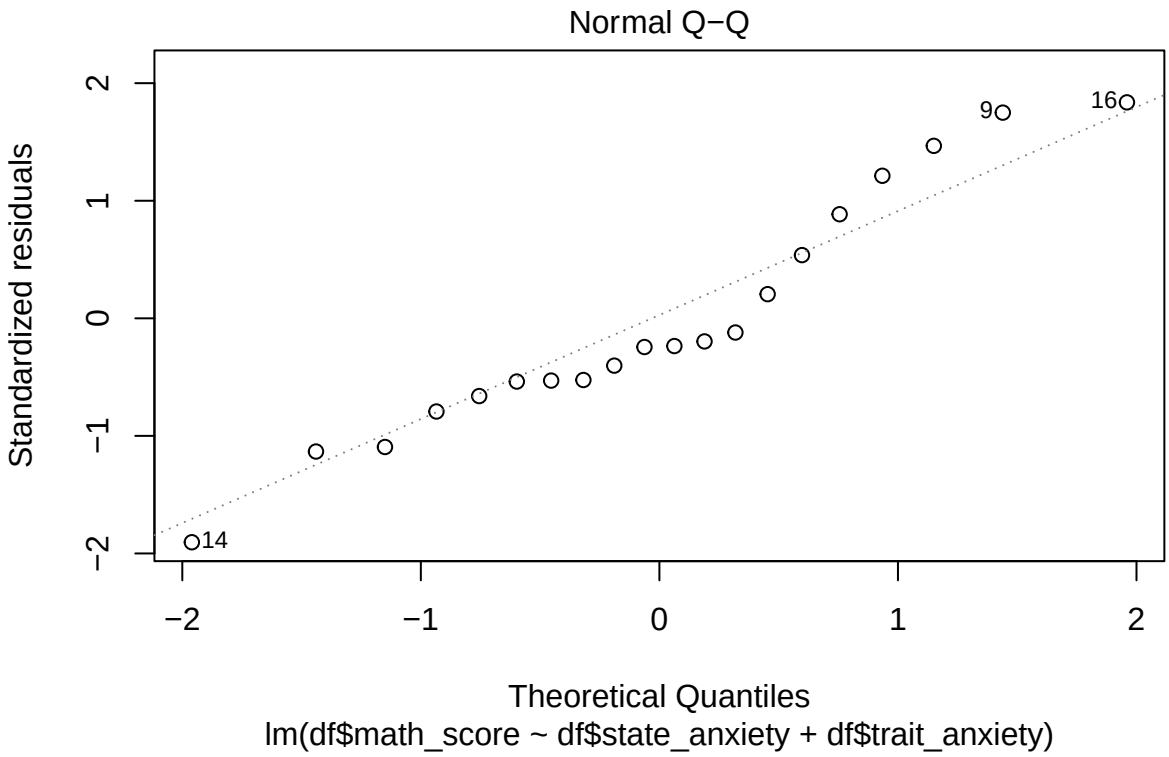
1. Linearity: Is the relationship between each predictor and the outcome linear?
2. Homoscedasticity: Is the variance of the residuals the same for any value of X?
3. Normality: For any value of X, is Y normally distributed?

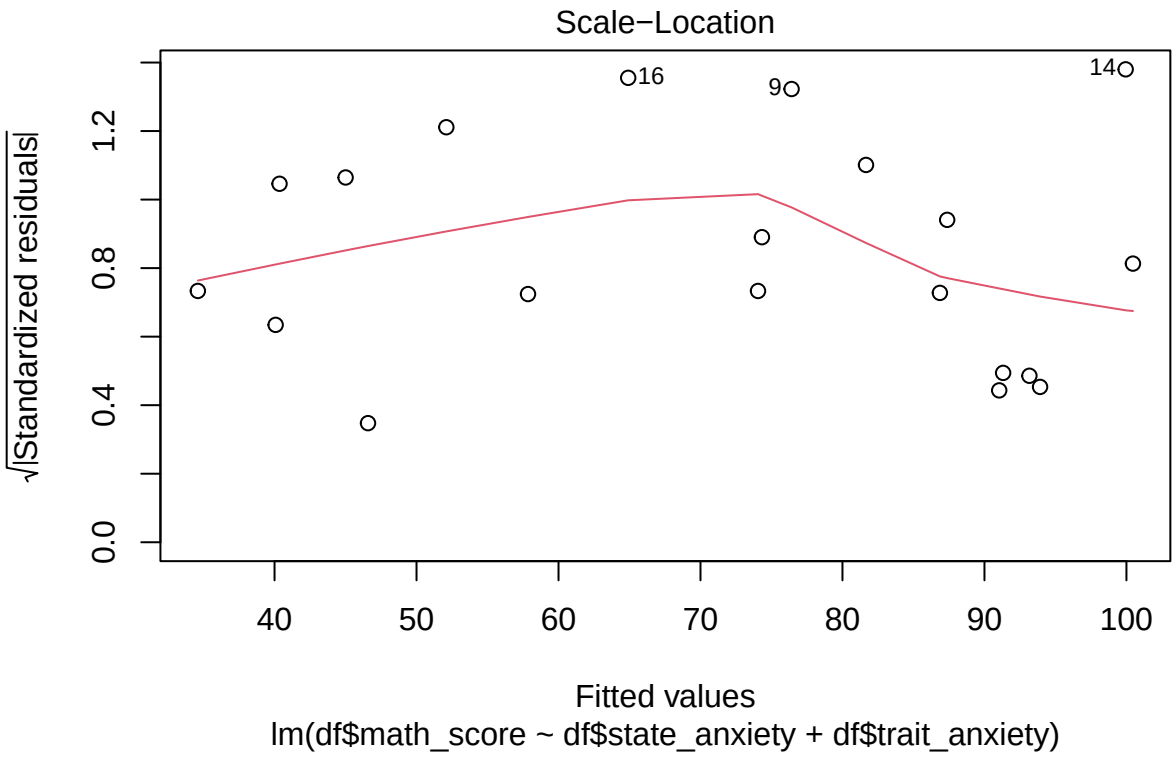
Additional assumptions include independence of observations and random sampling, which you would need to confirm with a review of your research methods.

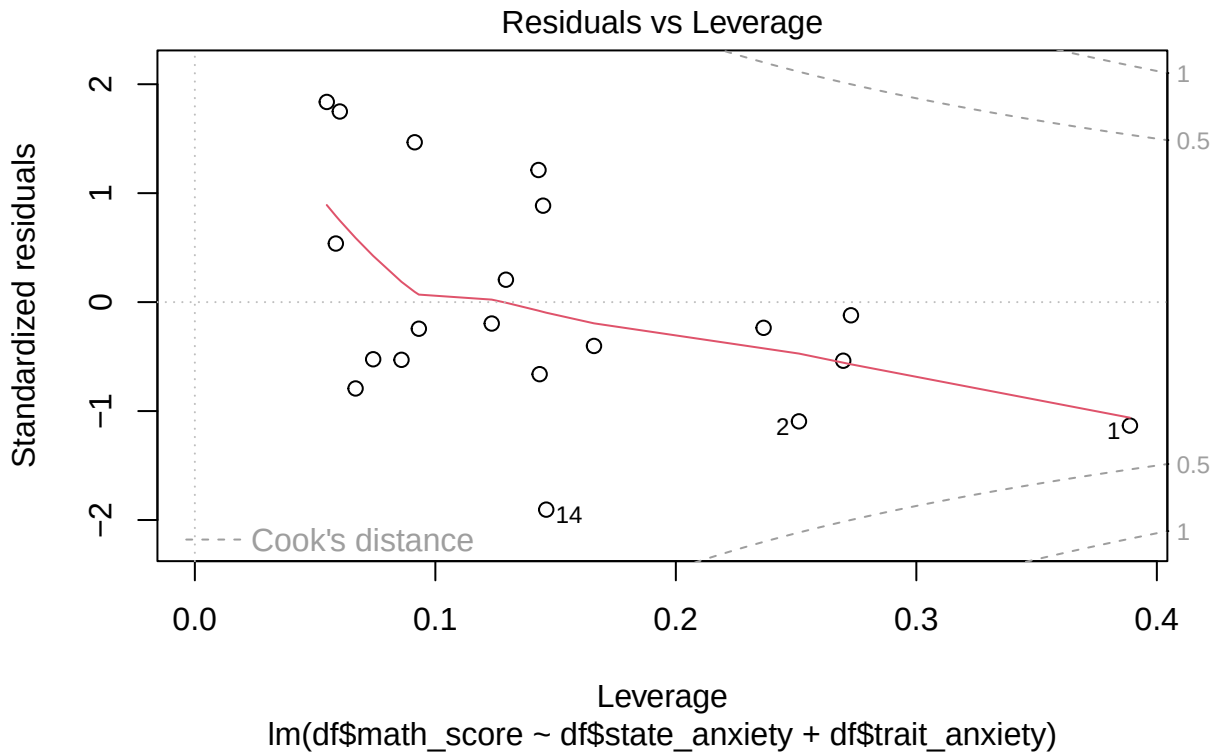
There are four built-in plots for checking regression assumptions and diagnosing the quality of the model. These plots can be called with `plot()` on the name of the model (in this case, `model`).

```
plot(model)
```


Residuals vs Fitted
Residuals
10
5
0
−5
−10
16
9
14
40
50
60
70
80
90
100
Fitted values
lm(df$math_score ~ df$state_anxiety + df$trait_anxiety)


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


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

Conduct Logistic Regression Analysis

The final topic is conducting logistic regression. This involves predicting a binary outcome, such as "pass" vs. "fail."

First, you will create a binary outcome variable called `df$math_cat` based on math scores. Note that the data type for the outcome must be numeric or logical, so rather than using "pass" and "fail" as the outcome values, you will use 1 (pass) for scores above 70 and 0 (fail) for scores equal to or less than 70. You will also need to convert this new variable to a factor.

```
df$math_cat <- ifelse(df$math_score > 70, 1, 0)
df$math_cat <- factor(df$math_cat)
```

Next, build the model. You want to predict `df$math_cat` status from `df$trait_anxiety` and `df$group`. Note that you will use the function `glm()` instead of `lm()` to signify the *generalized* linear model. This function takes an additional argument: the family of regression methods (e.g., binomial or poisson).

```
math_logit <- glm(math_cat ~ group + trait_anxiety,
                  data = df,
                  family = "binomial")
```

Then look at the summary.

```
summary(math_logit)
```

```
##
## Call:
## glm(formula = math_cat ~ group + trait_anxiety, family = "binomial",
##     data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2202  -0.7464   0.4463   0.9005   1.2464
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)     1.9338     1.2139   1.593   0.1111
## groupTreat      0.9074     1.0804   0.840   0.4010
## trait_anxiety  -0.2327     0.1224  -1.902   0.0572 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 26.920  on 19  degrees of freedom
## Residual deviance: 21.223  on 17  degrees of freedom
## AIC: 27.223
##
## Number of Fisher Scoring iterations: 4
```

The output shows a marginally significant effect of trait anxiety, which may be worth investigating in a larger sample. However, group was not a significant predictor of math outcome. To translate this output into log-odds language, we could say that for every one unit increase in trait anxiety, the log odds of passing the math test *decrease* by 0.23.

References and Recommended Reading

- Kim, B. (2015). "Understanding diagnostic plots for linear regression analysis". *UVA Library StatLab*. https://library.virginia.edu/data/articles/diagnostic-plots
- R Project for Statistical Computing. (n.d.). https://www.r-project.org/
- Teetor, P. (2011). *R cookbook*. O'Reilly.
- UCLA: Statistical Consulting Group. (2021). "Logit regression: R data analysis examples". https://stats.oarc.ucla.edu/r/dae/logit-regression/