1. Introduction to R

Emmanuel Masavo DJEGOU

2025-04-06

Contents

1 Launching R

R can be run in two modes: **interactive** and **batch**. Here's a brief overview of both:

1.1 Interactive Mode

In interactive mode, you can run commands directly, and R will display results immediately in the console. This mode is useful for experimenting and quickly testing your ideas.

You can launch an R session in the following ways:

- On Linux or macOS, open a terminal and type `R`, then press Enter.
- On Windows, start R by double-clicking the R shortcut icon on your desktop or in the Start menu.

```
# Example of Interactive Mode: Creating a numeric vector and calculating its mean
y_data <- abs(rnorm(100))
mean(y_data)  # Output shown directly in the console
```

```
## [1] 0.7851292
```

The code generates 100 random values, takes their absolute values, and computes the mean. The `[1]` in the output shows the position of the first item in the line—useful for reading long outputs, where each line is numbered by its starting item.

R commands can be saved in a file, usually with a `.R` or `.r` extension. To run the code in a file like `Script.R`, use the command:

```r
source("Script.R")
```

```
## [1] "The mean of y_data is: 0.752387571161506"
```

1.2 Batch Mode

You can automate R scripts by running them in **batch mode**, avoiding manual interaction. For example, save the following code in a file called `Graph-Making.R`:

- pdf("histogram.pdf") # Save the next plot to a PDF file named "histogram.pdf"
- hist(rnorm(100)) # Create a histogram of 100 random values from a standard normal distribution
- dev.off() # Finish and close the PDF file

Run the script from the command line with:

```
R CMD BATCH Graph-Making.R
```

2 Exploring Your First R Session

In this section, we will explore how to create and manipulate vectors, compute summary statistics, and visualize data using R.

```r
# Creating a numeric vector
x <- c(1, 2, 4)

# Creating a new vector by repeating x and adding 8
q <- c(x, x, 8)

# Accessing the third element of x
print(x[3])
```

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

```r
# Subsetting: extracting the first two elements of x
subset_x <- x[1:2]
print(subset_x)
```

```
## [1] 1 2
```

```r
# Calculating the mean of x
mean_x <- mean(x)
print(mean_x)
```

```
## [1] 2.333333
```

```r
# Calculating the standard deviation of x
std_x <- sd(x)
print(std_x)
```

```
## [1] 1.527525
```

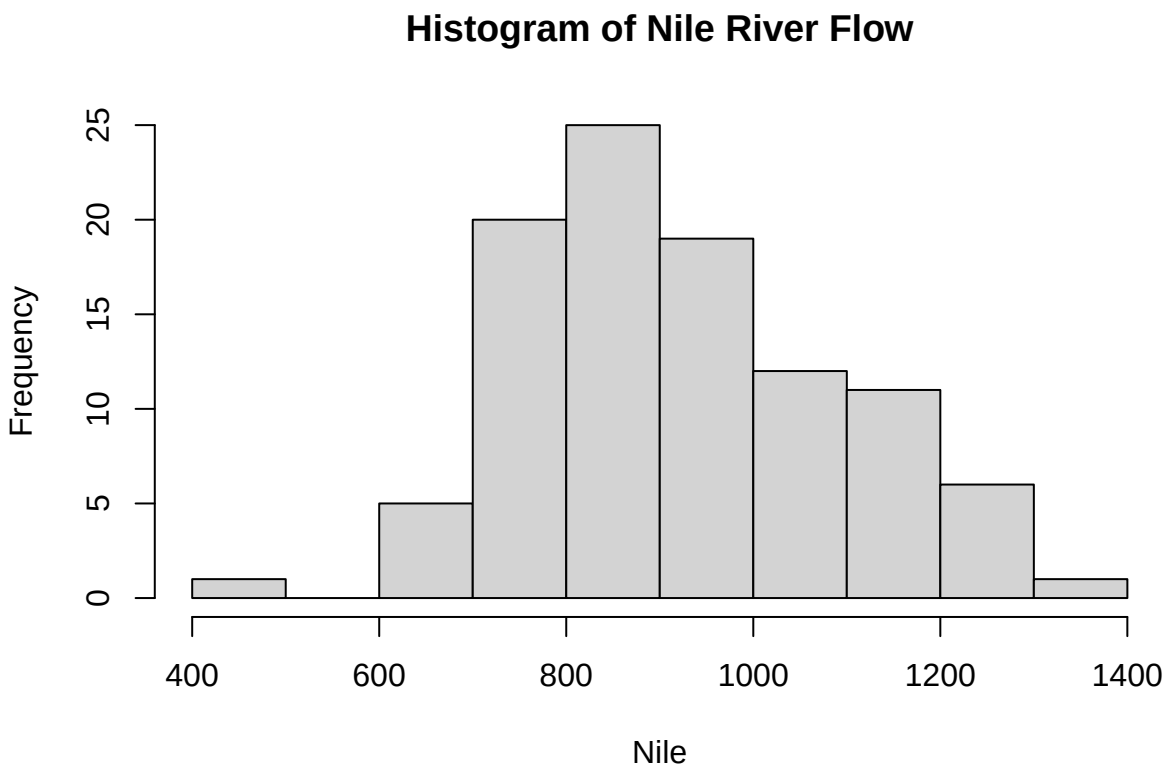
```r
# Listing available datasets
data()

# Working with the "Nile" dataset
print(mean(Nile))      # Mean of the dataset
```

```
## [1] 919.35
```

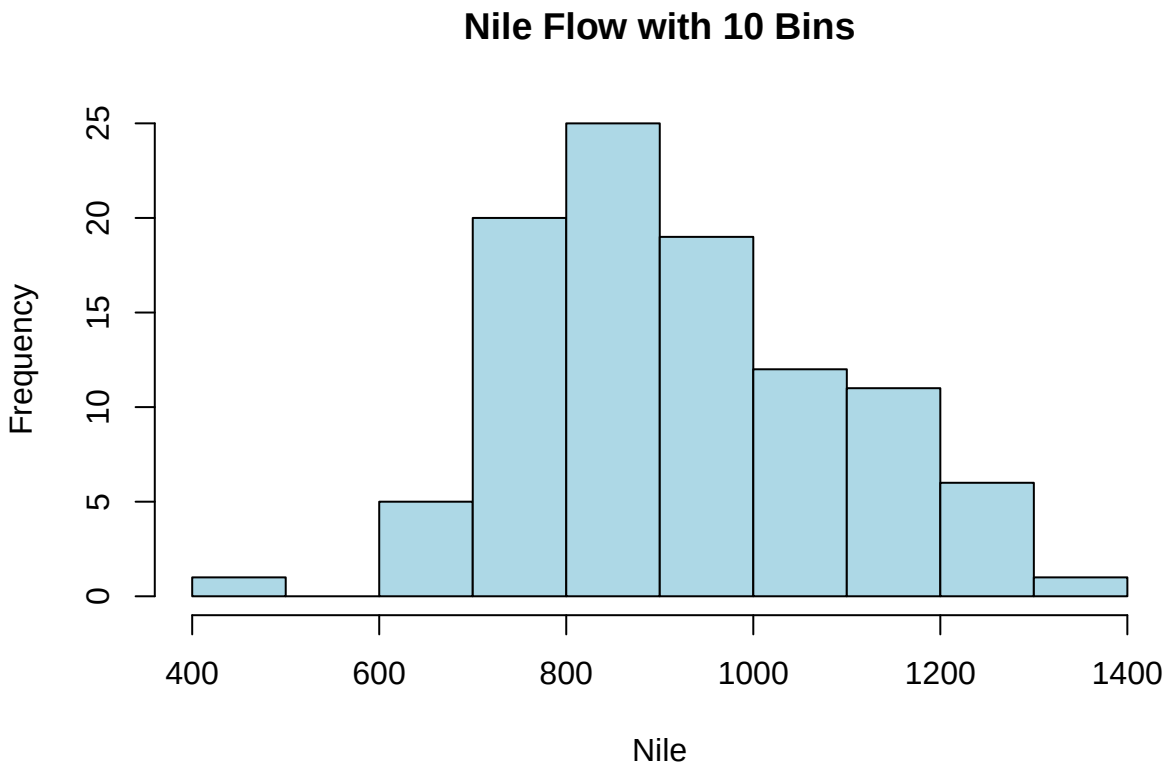
```r
print(sd(Nile))        # Standard deviation
```

```
## [1] 169.2275
```

```r
# Plotting histogram
hist(Nile, main = "Histogram of Nile River Flow")
```




```r
hist(Nile, breaks = 10, col = "lightblue", main = "Nile Flow with 10 Bins")
```



3 Creating and Using Functions in R

```
# Function to count the number of odd numbers in a vector
oddcount <- function(x) {
     k <- 0
     for (n in x) {
          if (n %% 2 == 1) k <- k + 1
     }
     return(k)
}

# Testing the oddcount function
y <- c(1, 2, 3, 7, 9)
print(oddcount(y))
```

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

```
# Modulo operator example
print(38 %% 7)
```

```
## [1] 3
```

```
# Global variable example
f <- function(x) return(z + y)
z <- 3
print(f(z))
```

```
## [1]  4  5  6 10 12
```

```r
# Function with default arguments
g <- function(x, y = 2, z = TRUE) {
    print(paste("x:", x, "| y:", y, "| z:", z))
}

g(12, z = FALSE)
```

```
## [1] "x: 12 | y: 2 | z: FALSE"
```

```r
# To quit R (uncomment if needed)
# q()
```

4 Key R Data Structures

```r
# Vectors
# A vector in R is a collection of elements that must all have the same data type (mode).

# Scalar = one-element vector of mode numeric
x <- 8        # Scalar, a single numeric value
x             # Display the value of x
```

```
## [1] 8
```

```r
# Character String = one-element vector of mode character
x <- c(5, 12, 13)  # Vector of numeric values
x                  # Display the vector
```

```
## [1]  5 12 13
```

```r
length(x)          # Returns the number of elements in the vector (3)
```

```
## [1] 3
```

```r
mode(x)            # Returns the mode of the vector, which is "numeric"
```

```
## [1] "numeric"
```

```r
y <- "cpt"         # Single character string, mode is character
length(y)          # Length of the string (1)
```

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

```r
mode(y)            # Mode is "character"
```

```
## [1] "character"
```

```r
z <- c("cpt", "29 88")  # Character vector with two elements
length(z)               # Length of the vector (2)
```

```
## [1] 2
```

```r
mode(z)                 # Mode is "character"
```

```
## [1] "character"
```

```r
u <- paste("cpt","opt","f1")  # Concatenate strings into one character string
u                             # Display the concatenated string
```

```
## [1] "cpt opt f1"
```

```r
v <- strsplit(u, " ")  # Split the string into a list based on space
v                      # Display the result (list of substrings)
```

```
## [[1]]
## [1] "cpt" "opt" "f1"
```

```r
# Matrices = A two-dimensional array of numbers (rectangular array)
# Matrices are essentially vectors with two additional attributes: row and column numbers

m <- rbind(c(4, 5), c(0, 1))  # Create a matrix by binding rows
m                             # Display the matrix
```

```
##      [,1] [,2]
## [1,]    4    5
## [2,]    0    1
```

```r
n <- cbind(c(2, 8), c(3, 8))  # Create a matrix by binding columns
n                             # Display the matrix
```

```
##      [,1] [,2]
## [1,]    2    3
## [2,]    8    8
```

```r
# Matrix-multiplication operator
m %*% c(2, 3)  # Perform matrix multiplication with a vector (resulting in a matrix)
```

```
##      [,1]
## [1,]   23
## [2,]    3
```

```r
# Indexing matrices
m[1,1]  # Access element in the 1st row, 1st column of the matrix
```

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

```r
m[2,1]  # Access element in the 2nd row, 1st column of the matrix
```

```
## [1] 0
```

```r
# Extracting Submatrices
m[1,]   # Extract the 1st row of the matrix
```

```
## [1] 4 5
```

```r
m[,2]   # Extract the 2nd column of the matrix
```

```
## [1] 5 1
```

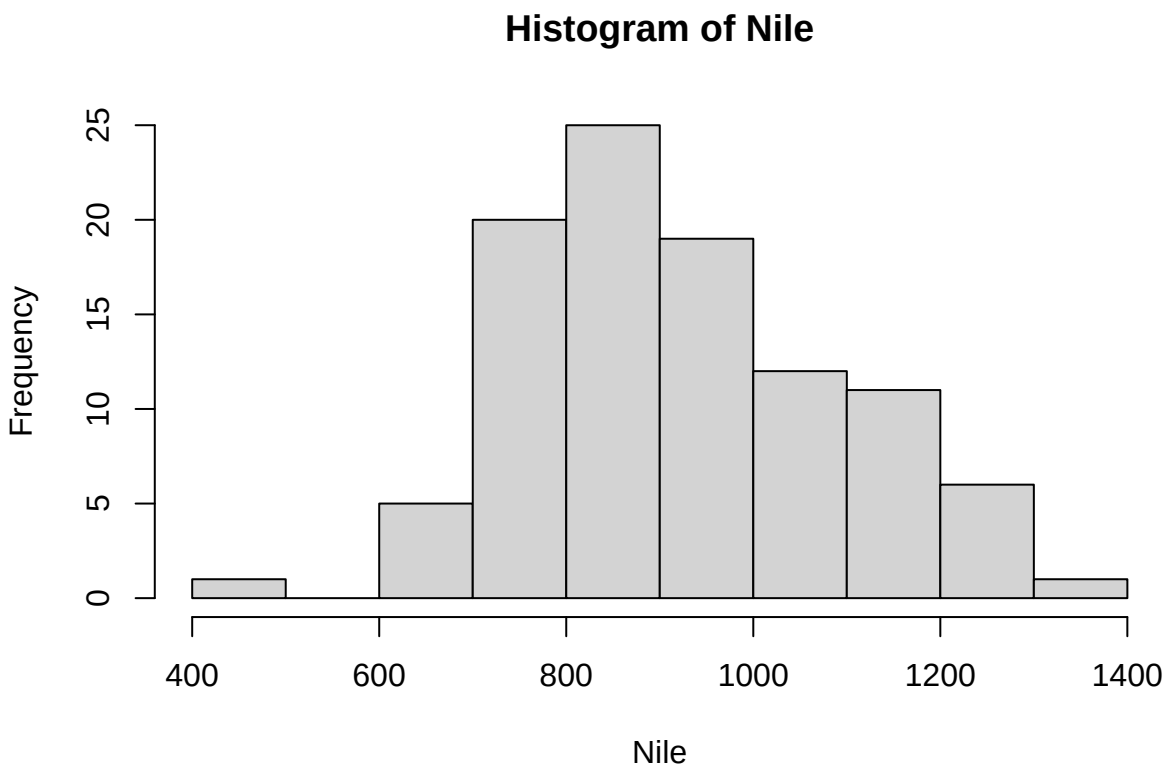
```r
# Lists = Containers that can hold multiple values of different types
x <- list(u=2, v="cpt")  # Create a list with numeric and character elements
x                        # Display the list
```

```
## $u
## [1] 2
##
## $v
## [1] "cpt"
```

```r
x$u                      # Access the value of element 'u' in the list
```

```
## [1] 2
```

```r
# Example of histogram
hist(Nile)               # Create a histogram of the Nile dataset
```




```r
hist_value <- hist(Nile) # Store the histogram object in hist_value
print(hist_value)        # Display the stored histogram object
```

```
## $breaks
##  [1]  400  500  600  700  800  900 1000 1100 1200 1300 1400
##
## $counts
##  [1]  1  0  5 20 25 19 12 11  6  1
##
## $density
##  [1] 0.0001 0.0000 0.0005 0.0020 0.0025 0.0019 0.0012 0.0011 0.0006 0.0001
##
## $mids
##  [1]  450  550  650  750  850  950 1050 1150 1250 1350
##
## $xname
## [1] "Nile"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

```r
# str() function displays the internal structure of any R object
str(hist_value)          # Display the structure of the histogram object
```

```
## List of 6
##  $ breaks  : int [1:11] 400 500 600 700 800 900 1000 1100 1200 1300 ...
##  $ counts  : int [1:10] 1 0 5 20 25 19 12 11 6 1
##  $ density : num [1:10] 0.0001 0 0.0005 0.002 0.0025 0.0019 0.0012 0.0011 0.0006 0.0001
##  $ mids    : num [1:10] 450 550 650 750 850 950 1050 1150 1250 1350
##  $ xname   : chr "Nile"
##  $ equidist: logi TRUE
##  - attr(*, "class")= chr "histogram"
```

```r
# Data Frame = A special type of list where each component is a column vector
# Create a data frame with columns "kids" and "ages"
df <- data.frame(list(kids = c("Easton", "Emma"), ages = c(25, 32)))
df                       # Display the data frame
```

```
##     kids ages
## 1 Easton   25
## 2   Emma   32
```

```r
df$kids                 # Access the "kids" column
```

```
## [1] "Easton" "Emma"
```

```r
df$age          # Access the "ages" column (Note: Should be df$ages, not df$age)
```

```
## [1] 25 32
```

R is an object-oriented language, meaning it uses **objects** that belong to **classes**. In R, many objects use **S3 classes**, which are just regular lists with an extra label (the class name).

For example, the result of `hist()` is a list with components like `breaks` and `counts`, and its class is `"histogram"`.

```r
# Printing hist_value
print(hist_value)
```

```
## $breaks
##  [1]  400  500  600  700  800  900 1000 1100 1200 1300 1400
##
## $counts
##  [1]  1  0  5 20 25 19 12 11  6  1
##
## $density
##  [1] 0.0001 0.0000 0.0005 0.0020 0.0025 0.0019 0.0012 0.0011 0.0006 0.0001
##
## $mids
##  [1]  450  550  650  750  850  950 1050 1150 1250 1350
##
## $xname
## [1] "Nile"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

Classes are used because they work with **generic functions** like `summary()` and `plot()`. These functions behave differently depending on the object's class.

- `summary()` gives useful summaries tailored to each type of object (e.g., histogram, regression).
- `plot()` knows how to make the right kind of plot based on the object's class.

In short, **S3 classes and generic functions** help R handle different types of data in a consistent and flexible way.

5 Application to Regression Analysis

```r
# Create the dataset
examsquiz <- data.frame(list(Exam1 = c(2.0, 3.3, 4.0, 2.3, 2.3, 3.3, 3.7, 4.0, 3.0, 2.7), Exam2 = c(3.3

# View the first few rows
head(examsquiz)
```

```
##   Exam1 Exam2 Quiz
## 1   2.0   3.3  4.0
## 2   3.3   2.0  3.7
## 3   4.0   3.7  4.0
## 4   2.3   0.0  3.3
## 5   2.3   1.0  3.3
## 6   3.3   3.7  4.0
```

```r
# Fit a simple linear regression model where Exam2 is predicted by Exam1
# Model: Exam2 = a_0 + a_1 * Exam1
lin_model1 <- lm(examsquiz$Exam2 ~ examsquiz$Exam1)

# Alternatively, using column indices instead of names
# This assumes Exam1 is in column 1 and Exam2 is in column 2
lin_model2 <- lm(examsquiz[, 2] ~ examsquiz[, 1])

# View the attributes (components) of the linear model object
# This shows what parts are stored inside lin_model1, such as coefficients, residuals, etc.
attributes(lin_model1)
```

```
## $names
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
##
## $class
## [1] "lm"
```

```r
# Get a structured and detailed summary of all components in the model
# This is helpful for exploring the internal structure of the lm object
str(lin_model1)
```

```
## List of 12
##  $ coefficients : Named num [1:2] 0.658 0.645
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "examsquiz$Exam1"
##  $ residuals    : Named num [1:10] 1.353 -0.785 0.464 -2.14 -1.14 ...
##   ..- attr(*, "names")= chr [1:10] "1" "2" "3" "4" ...
##  $ effects      : Named num [1:10] -8.317 -1.398 0.637 -2.676 -1.676 ...
##   ..- attr(*, "names")= chr [1:10] "(Intercept)" "examsquiz$Exam1" "" "" ...
```

```
##  $ rank         : int 2
##  $ fitted.values: Named num [1:10] 1.95 2.78 3.24 2.14 2.14 ...
##   ..- attr(*, "names")= chr [1:10] "1" "2" "3" "4" ...
##  $ assign       : int [1:2] 0 1
##  $ qr           :List of 5
##   ..$ qr   : num [1:10, 1:2] -3.162 0.316 0.316 0.316 0.316 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:10] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:2] "(Intercept)" "examsquiz$Exam1"
##   .. ..- attr(*, "assign")= int [1:2] 0 1
##   ..$ qraux: num [1:2] 1.32 1.23
##   ..$ pivot: int [1:2] 1 2
##   ..$ tol  : num 1e-07
##   ..$ rank : int 2
##   ..- attr(*, "class")= chr "qr"
##  $ df.residual  : int 8
##  $ xlevels      : Named list()
##  $ call         : language lm(formula = examsquiz$Exam2 ~ examsquiz$Exam1)
##  $ terms        :Classes 'terms', 'formula'  language examsquiz$Exam2 ~ examsquiz$Exam1
##   .. ..- attr(*, "variables")= language list(examsquiz$Exam2, examsquiz$Exam1)
##   .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:2] "examsquiz$Exam2" "examsquiz$Exam1"
##   .. .. .. ..$ : chr "examsquiz$Exam1"
##   .. ..- attr(*, "term.labels")= chr "examsquiz$Exam1"
##   .. ..- attr(*, "order")= int 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(examsquiz$Exam2, examsquiz$Exam1)
##   .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. ..- attr(*, "names")= chr [1:2] "examsquiz$Exam2" "examsquiz$Exam1"
##  $ model        :'data.frame':   10 obs. of  2 variables:
##   ..$ examsquiz$Exam2: num [1:10] 3.3 2 3.7 0 1 3.7 3 2.3 3.3 4
##   ..$ examsquiz$Exam1: num [1:10] 2 3.3 4 2.3 2.3 3.3 3.7 4 3 2.7
##   ..- attr(*, "terms")=Classes 'terms', 'formula'  language examsquiz$Exam2 ~ examsquiz$Exam1
##   .. .. ..- attr(*, "variables")= language list(examsquiz$Exam2, examsquiz$Exam1)
##   .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. .. ..$ : chr [1:2] "examsquiz$Exam2" "examsquiz$Exam1"
##   .. .. .. .. ..$ : chr "examsquiz$Exam1"
##   .. .. ..- attr(*, "term.labels")= chr "examsquiz$Exam1"
##   .. .. ..- attr(*, "order")= int 1
##   .. .. ..- attr(*, "intercept")= int 1
##   .. .. ..- attr(*, "response")= int 1
##   .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. .. ..- attr(*, "predvars")= language list(examsquiz$Exam2, examsquiz$Exam1)
##   .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. .. ..- attr(*, "names")= chr [1:2] "examsquiz$Exam2" "examsquiz$Exam1"
##  - attr(*, "class")= chr "lm"
```

```r
# Access the estimated regression coefficients directly
# This gives you the intercept (a_0) and the slope (a_1)
lin_model1$coefficients
```

```
##     (Intercept) examsquiz$Exam1
##       0.6576531       0.6445578
```

```r
# Alternatively, printing the model itself displays the coefficients
print(lin_model1)
```

```
##
## Call:
## lm(formula = examsquiz$Exam2 ~ examsquiz$Exam1)
##
## Coefficients:
##     (Intercept)  examsquiz$Exam1
##          0.6577           0.6446
```

R only printed part of `lin_model1` because `print()` is a generic function. For `lm` objects, it calls `print.lm()`, which shows only a summary, not all components.

```r
# Using the generic function summary()
summary(lin_model1)
```

```
##
## Call:
## lm(formula = examsquiz$Exam2 ~ examsquiz$Exam1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.1401 -0.8981  0.2108  0.8637  1.6020
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.6577     1.8662   0.352    0.734
## examsquiz$Exam1   0.6446     0.5951   1.083    0.310
##
## Residual standard error: 1.291 on 8 degrees of freedom
## Multiple R-squared:  0.1279, Adjusted R-squared:  0.01888
## F-statistic: 1.173 on 1 and 8 DF,  p-value: 0.3103
```

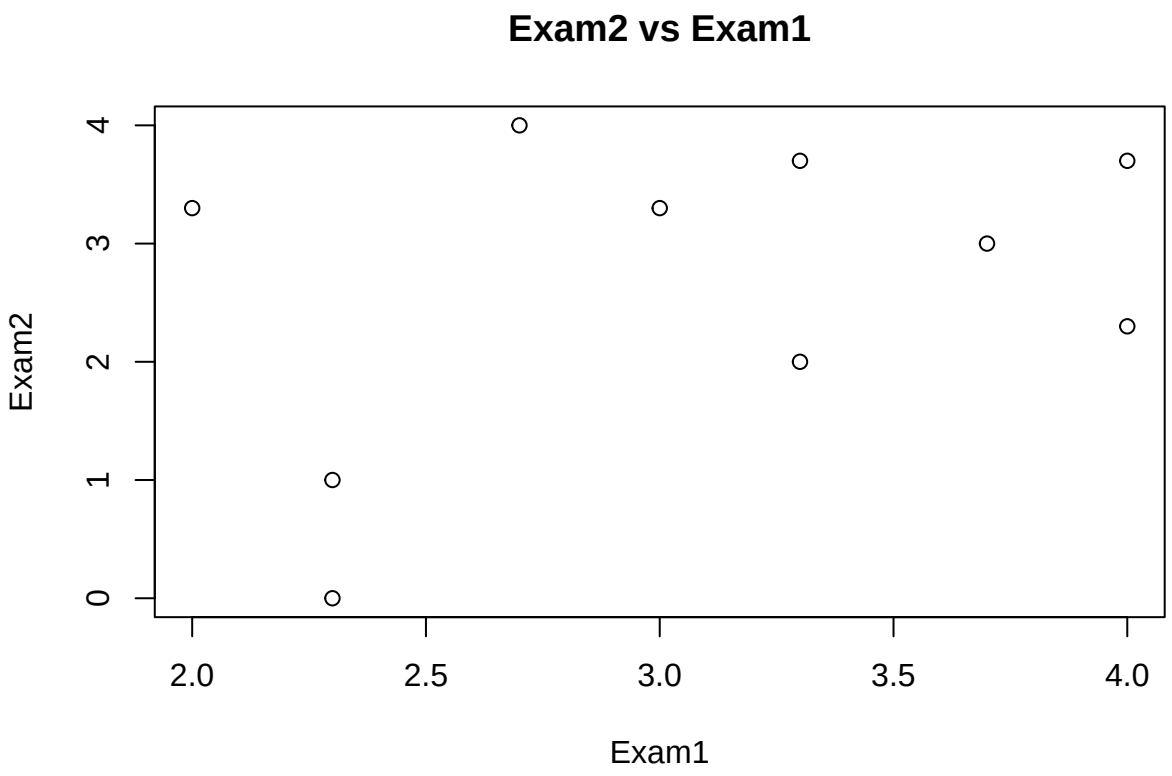
The p-value for the predictor `examsquiz$Exam1` is **0.310**, which is greater than the common significance level (e.g., 0.05). This suggests that **Exam1 is not a statistically significant predictor of Exam2** in this model. In other words, there's **not enough evidence** to conclude a linear relationship between Exam1 and Exam2.

The **high p-value (0.310)** and the **low R-squared (0.1279)** both suggest that a **simple linear regression** is **not a good fit** for the data. It means the model does **not explain much of the variability** in `Exam2` using `Exam1`.

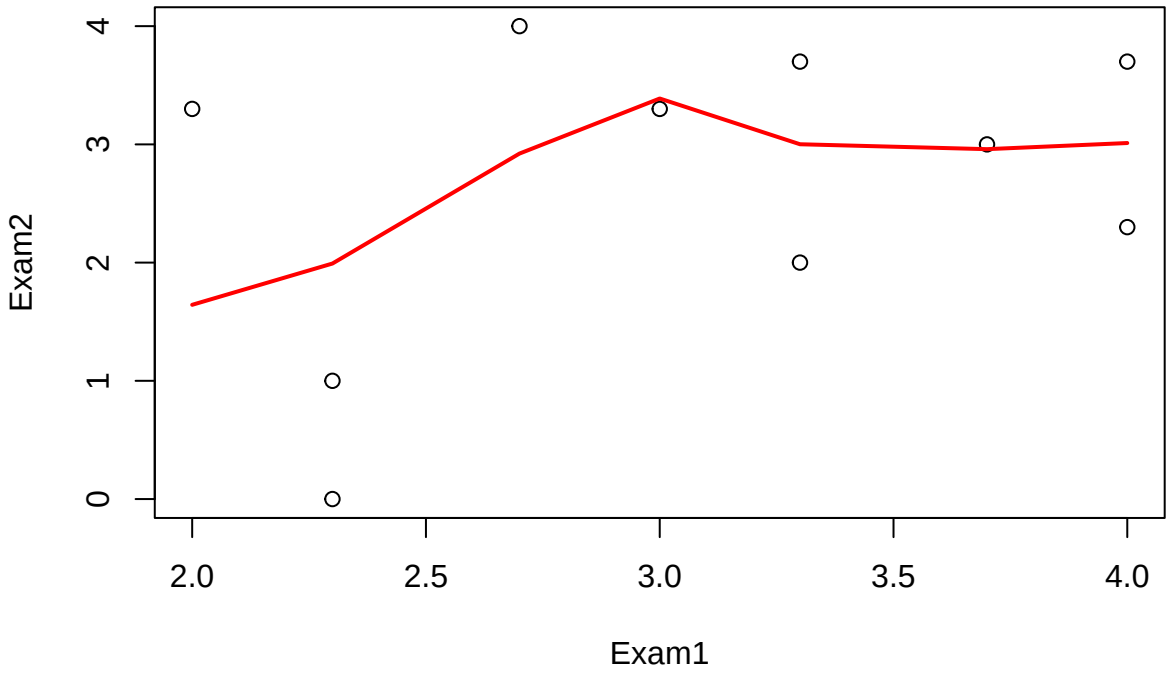
That said, it doesn't mean there's no relationship at all—just that this **linear** model isn't capturing it well. You might consider:

- Checking for **nonlinear relationships**
- Adding **more predictors**
- Looking for **outliers** or **influential points**

```r
# Scatterplot
plot(examsquiz$Exam1, examsquiz$Exam2,
     main = "Exam2 vs Exam1",
     xlab = "Exam1", ylab = "Exam2")
```



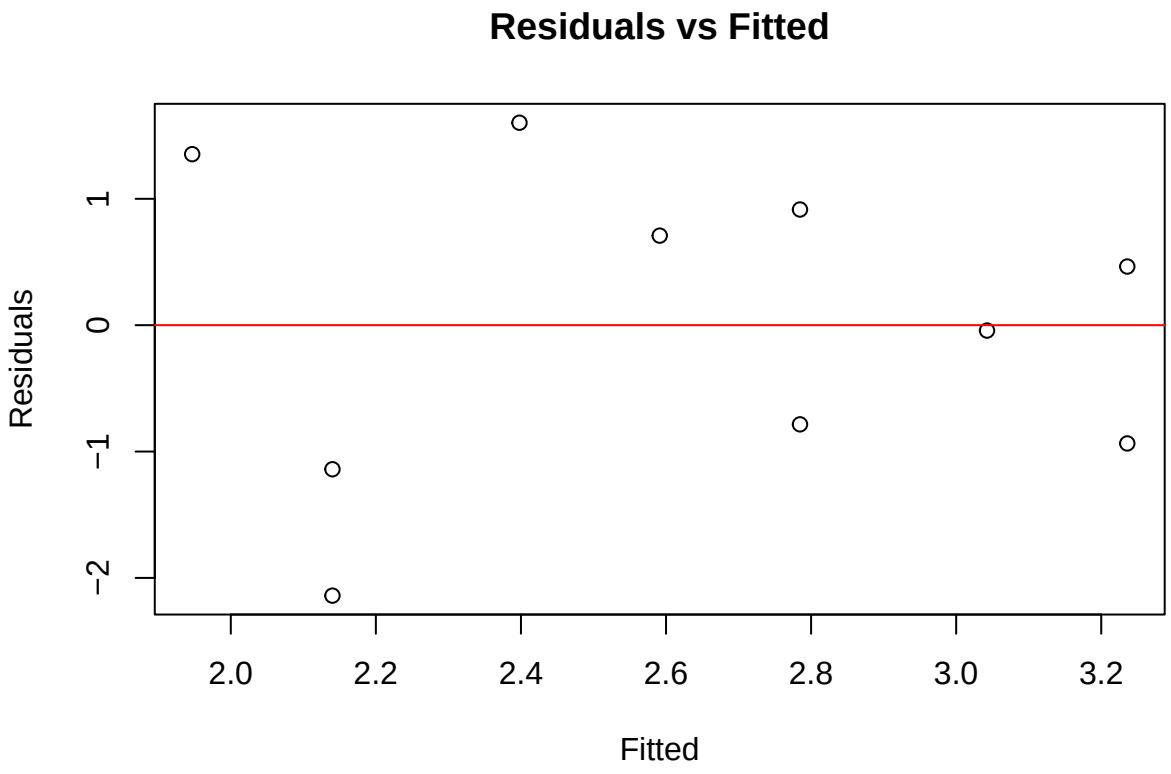

```
# Add a Smoother (like LOESS)
plot(examsquiz$Exam1, examsquiz$Exam2,
     xlab = "Exam1", ylab = "Exam2")
lines(lowess(examsquiz$Exam1, examsquiz$Exam2), col = "red", lwd = 2)
```




```
# Fit a Polynomial Regression
poly_model <- lm(examsquiz$Exam2 ~ examsquiz$Exam1 + I(examsquiz$Exam1^2))
summary(poly_model)
```

```
##
## Call:
## lm(formula = examsquiz$Exam2 ~ examsquiz$Exam1 + I(examsquiz$Exam1^2))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.1076 -0.8999  0.2307  0.7251  1.6018
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -2.8880    10.1244  -0.285    0.784
## examsquiz$Exam1        3.1002     6.9058   0.449    0.667
## I(examsquiz$Exam1^2)  -0.4036     1.1302  -0.357    0.732
##
## Residual standard error: 1.367 on 7 degrees of freedom
## Multiple R-squared:  0.1435, Adjusted R-squared:  -0.1012
## F-statistic: 0.5864 on 2 and 7 DF,  p-value: 0.5815
```

```r
# Residual Plot
plot(fitted(lin_model1), resid(lin_model1),
     main = "Residuals vs Fitted", xlab = "Fitted", ylab = "Residuals")
abline(h = 0, col = "red")
```



A curved pattern in residuals indicates nonlinearity.

6 Working Directory

```r
# Checking your current working directory
getwd()
```

```
## [1] "/cloud/project"
```

```
# Setting your working directory
setwd("/cloud/project")
```

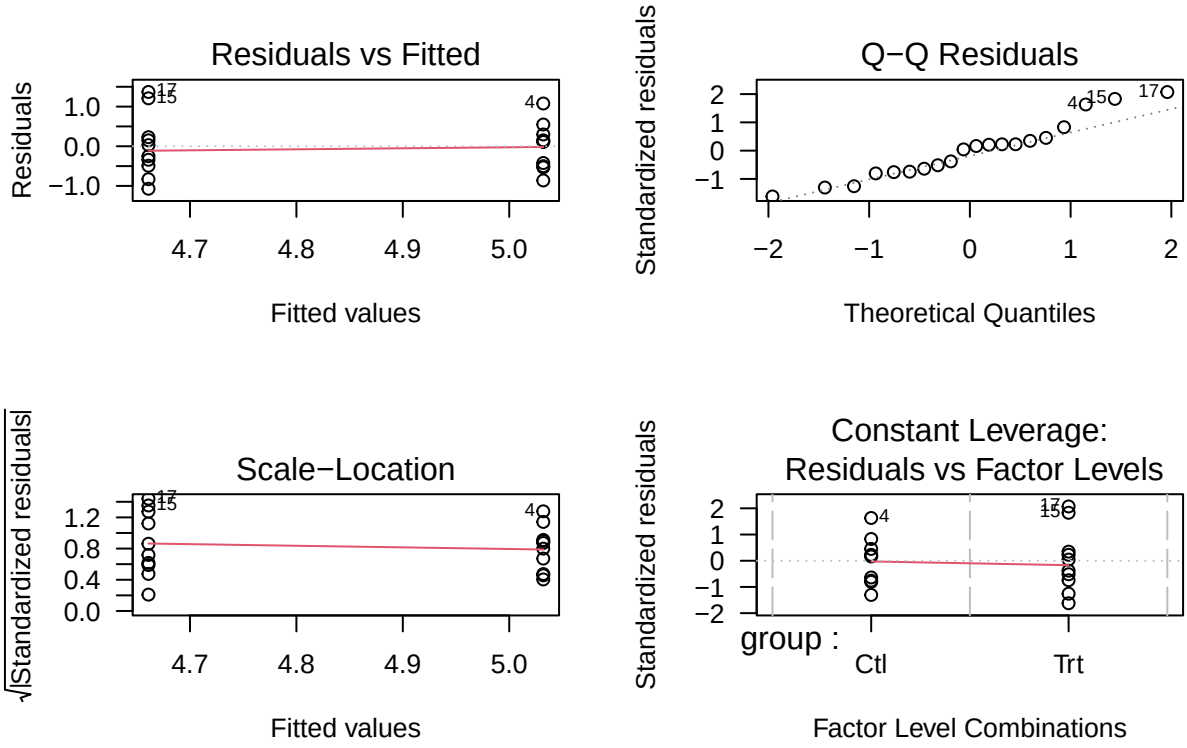
The `.RData` file stores your R workspace and is saved in your working directory (Linux) or R installation folder (Windows). Use the `.Rhistory` file to recall the commands used to create it.

7 Getting Help

There are plenty of resources available to help you deepen your understanding of R.

```
# Get help documentation for the lm() function, which fits linear models
help(lm)

# Shortcut to access the same help page
?lm

# Get help on operators like "<"
?"<"

# Run a built-in example to see how lm() works in practice
example(lm)
```

```
##
## lm> require(graphics)
##
## lm> ## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## lm> ## Page 9: Plant Weight Data.
## lm> ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
##
## lm> trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
##
## lm> group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
##
## lm> weight <- c(ctl, trt)
##
## lm> lm.D9 <- lm(weight ~ group)
##
## lm> lm.D90 <- lm(weight ~ group - 1) # omitting intercept
##
## lm> ## No test:
## lm> ##D anova(lm.D9)
## lm> ##D summary(lm.D90)
## lm> ## End(No test)
## lm> opar <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
##
## lm> plot(lm.D9, las = 1)      # Residuals, Fitted, ...
```

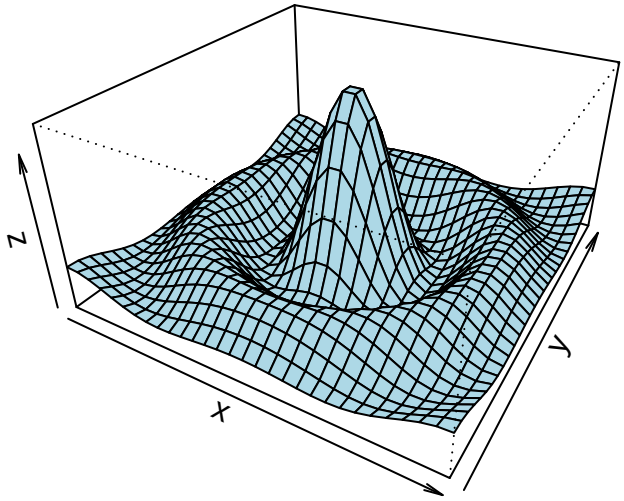



```
##
## lm> par(opar)
##
## lm> ## Don't show:
## lm> ## model frame :
## lm> stopifnot(identical(lm(weight ~ group, method = "model.frame"),
## lm+                     model.frame(lm.D9)))
##
## lm> ## End(Don't show)
## lm> ### less simple examples in "See Also" above
## lm>
## lm>
## lm>
```

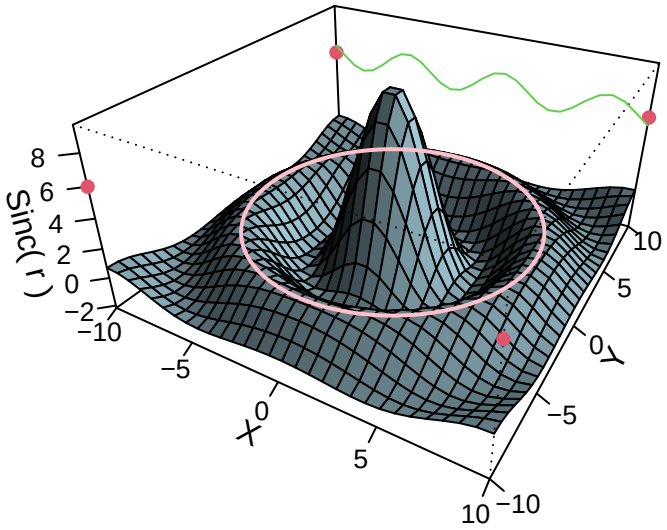
```
# Another example for the persp() function (3D perspective plots)
example(persp)
```

```
##
## persp> require(grDevices) # for trans3d
##
## persp> ## More examples in  demo(persp) !!
## persp> ##                   -----------
## persp>
## persp> # (1) The Obligatory Mathematical surface.
## persp> #     Rotated sinc function.
## persp>
## persp> x <- seq(-10, 10, length.out = 30)
##
```

```
## persp> y <- x
##
## persp> f <- function(x, y) { r <- sqrt(x^2+y^2); 10 * sin(r)/r }
##
## persp> z <- outer(x, y, f)
##
## persp> op <- par(bg = "white")
##
## persp> persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue")
```

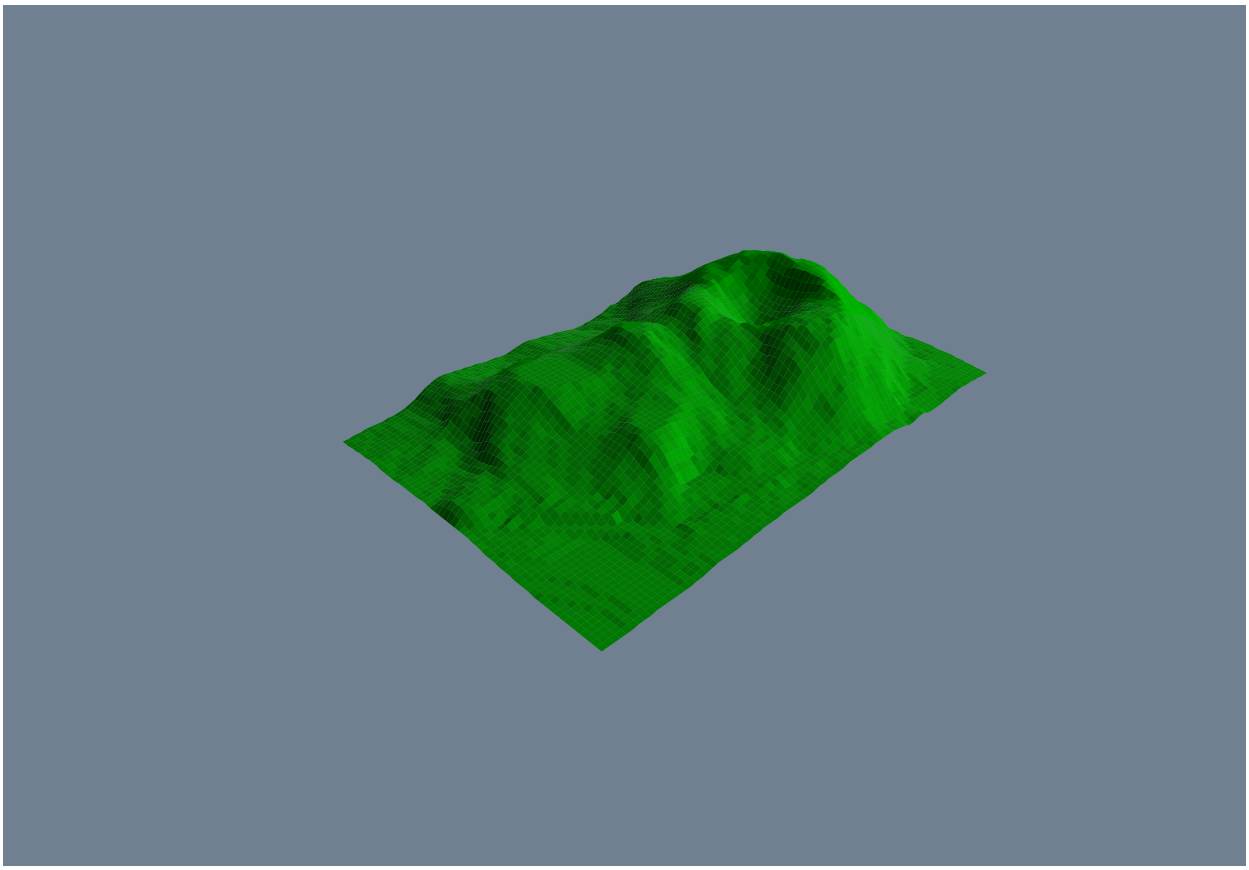



```
##
## persp> persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue",
## persp+       ltheta = 120, shade = 0.75, ticktype = "detailed",
## persp+       xlab = "X", ylab = "Y", zlab = "Sinc( r )", cex.axis = 0.8
## persp+ ) -> res
```



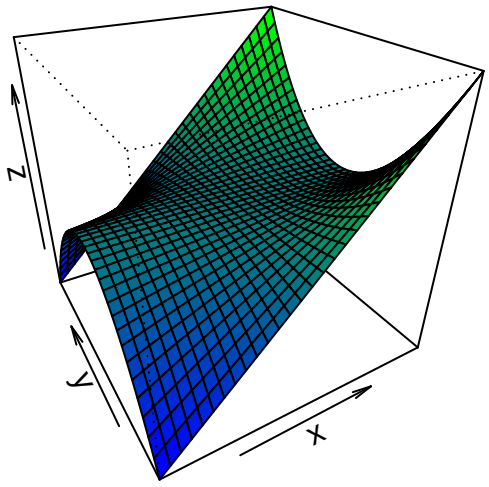

```
##
## persp> round(res, 3)
##       [,1]   [,2]   [,3]   [,4]
## [1,] 0.087 -0.025  0.043 -0.043
## [2,] 0.050  0.043 -0.075  0.075
```

```
## [3,] 0.000  0.074  0.042 -0.042
## [4,] 0.000 -0.273 -2.890  3.890
##
## persp> # (2) Add to existing persp plot - using trans3d() :
## persp>
## persp> xE <- c(-10,10); xy <- expand.grid(xE, xE)
##
## persp> points(trans3d(xy[,1], xy[,2], z = 6,          pmat = res), col = 2, pch = 16)
##
## persp> lines (trans3d(x,      y = 10, z = 6 + sin(x), pmat = res), col = 3)
##
## persp> phi <- seq(0, 2*pi, length.out = 201)
##
## persp> r1 <- 7.725 # radius of 2nd maximum
##
## persp> xr <- r1 * cos(phi)
##
## persp> yr <- r1 * sin(phi)
##
## persp> lines(trans3d(xr,yr, f(xr,yr), res), col = "pink", lwd = 2)
##
## persp> ## (no hidden lines)
## persp>
## persp> # (3) Visualizing a simple DEM model
## persp>
## persp> z <- 2 * volcano        # Exaggerate the relief
##
## persp> x <- 10 * (1:nrow(z))   # 10 meter spacing (S to N)
##
## persp> y <- 10 * (1:ncol(z))   # 10 meter spacing (E to W)
##
## persp> ## Don't draw the grid lines :  border = NA
## persp> par(bg = "slategray")
##
## persp> persp(x, y, z, theta = 135, phi = 30, col = "green3", scale = FALSE,
## persp+       ltheta = -120, shade = 0.75, border = NA, box = FALSE)
```



```
##
## persp> # (4) Surface colours corresponding to z-values
## persp>
## persp> par(bg = "white")
##
## persp> x <- seq(-1.95, 1.95, length.out = 30)
##
## persp> y <- seq(-1.95, 1.95, length.out = 35)
##
## persp> z <- outer(x, y, function(a, b) a*b^2)
##
## persp> nrz <- nrow(z)
##
## persp> ncz <- ncol(z)
##
## persp> # Create a function interpolating colors in the range of specified colors
## persp> jet.colors <- colorRampPalette( c("blue", "green") )
##
## persp> # Generate the desired number of colors from this palette
## persp> nbcol <- 100
##
## persp> color <- jet.colors(nbcol)
##
## persp> # Compute the z-value at the facet centres
## persp> zfacet <- z[-1, -1] + z[-1, -ncz] + z[-nrz, -1] + z[-nrz, -ncz]
##
```

```
## persp> # Recode facet z-values into color indices
## persp> facetcol <- cut(zfacet, nbcol)
##
## persp> persp(x, y, z, col = color[facetcol], phi = 30, theta = -30)
```




```
##
## persp> par(op)
```

```
# If you're not sure of the exact function name, search the help system
help.search("multivariate normal")

# Shortcut for help.search()
??"multivariate normal"

# List documentation for all functions in the 'MASS' package
help(package = MASS)

# Get help on the 'files' topic (useful for file-related functions)
?files
```

```
# To view available options for the R CMD INSTALL command (used to install packages from source)
R CMD INSTALL --help
```

```
## Usage: R CMD INSTALL [options] pkgs
##
## Install the add-on packages specified by pkgs.  The elements of pkgs can
## be relative or absolute paths to directories with the package
## sources, or to gzipped package 'tar' archives.  The library tree
## to install to can be specified via '--library'.  By default, packages are
## installed in the library tree rooted at the first directory in
## .libPaths() for an R session run in the current environment.
##
## Options:
##   -h, --help     print short help message and exit
##   -v, --version      print INSTALL version info and exit
##   -c, --clean        remove files created during installation
##       --preclean remove files created during a previous run
##   -d, --debug        turn on debugging messages
##   -l, --library=LIB  install packages to library tree LIB
##       --no-configure    do not use the package's configure script
```

```
##       --no-docs      do not install HTML, LaTeX or examples help
##       --html     build HTML help
##       --no-html      do not build HTML help
##       --latex        install LaTeX help
##       --example      install R code for help examples
##       --fake     do minimal install for testing purposes
##       --no-lock      install on top of any existing installation
##          without using a lock directory
##       --lock     use a per-library lock directory (default)
##       --pkglock      use a per-package lock directory
##                  (default for a single package)
##       --build        build binaries of the installed package(s)
##       --install-tests    install package-specific tests (if any)
##       --no-R, --no-libs, --no-data, --no-help, --no-demo, --no-exec,
##       --no-inst
##          suppress installation of the specified part of the
##          package for testing or other special purposes
##       --no-multiarch build only the main architecture
##       --libs-only    only install the libs directory
##       --data-compress=   none, gzip (default), bzip2 or xz compression
##          to be used for lazy-loading of data
##       --resave-data  re-save data files as compactly as possible
##       --compact-docs re-compress PDF files under inst/doc
##       --with-keep.source
##       --without-keep.source
##          use (or not) 'keep.source' for R code
##       --with-keep.parse.data
##       --without-keep.parse.data
##          use (or not) 'keep.parse.data' for R code
##       --byte-compile byte-compile R code
##       --no-byte-compile  do not byte-compile R code
##       --staged-install   install to a temporary directory and then move
##                      to the target directory (default)
##       --no-staged-install    install directly to the target directory
##       --no-test-load skip test of loading installed package
##       --no-clean-on-error    do not remove installed package on error
##       --merge-multiarch  multi-arch by merging (from a single tarball only)
##       --use-vanilla  do not read any Renviron or Rprofile files
##       --use-LTO          use Link-Time Optimization
##       --no-use-LTO       do not use Link-Time Optimization
##       --use-C17          use a C standard at most C17 (also C90, C99)
##       --use-C23          use a C standard at least C23
##
## for Unix
##       --configure-args=ARGS
##          set arguments for the configure scripts (if any)
##       --configure-vars=VARS
##          set variables for the configure scripts (if any)
##       --strip            strip shared object(s)
##       --strip-lib        strip static/dynamic libraries under lib/
##       --dsym             (macOS only) generate dSYM directory
##       --built-timestamp=STAMP
##                      set timestamp for Built: entry in DESCRIPTION
##
```

```
## Which of --html or --no-html is the default depends on the build of R:
## for this one it is --no-html.
##
## Report bugs at <https://bugs.R-project.org>.
```

Internet Help: To locate R scripts focusing on permutations, use the query: `filetype:R permutations -rebol`. This command targets `.R` files related to permutations and omits any results pertaining to Rebol.