BEGINNER AND INTERMEDIATE LEVEL R COURSE

Marcos Jimenez

About the workshop

Materials can be accessed at https://github.com/Marcosjnez/R_USF.

1. R as a Calculator

R can be used as a calculator:

```
3 + 2                   # Sum
```

```
[1] 5
```

```
5 - 2                   # Subtraction
```

```
[1] 3
```

```
3 * 4                   # Multiplication
```

```
[1] 12
```

```
15 / 5                  # Division
```

```
[1] 3
```

```
2^3                     # Power
```

```
[1] 8
```

```r
10 + (2 + 3) * 3^2 # Preference order for computing
```

```
[1] 55
```

1.1. Assignments

R can store values using symbolic names. There are multiple ways to assign values:

```r
a <- 3 # Best option
3 -> a # Less common...
a = 3  # Not recommended
a      # Visualize the object
```

```
[1] 3
```

1.2. Reserved Words

Some names are reserved in R and cannot be used as variables:

```r
# TRUE <- 4 # Error
?reserved # See reserved words
```

```
starting httpd help server ... done
```

```r
?objects
```

1.3. Symbolic Operations

You can use variables in arithmetic operations:

```r
a <- 3
b <- 2
c <- a + b
```

---

2. Data Types in R

R supports several basic data types.

2.1. Numeric

```r
a <- 1
b <- 2
c <- 1 + 2
```

2.2. Character Strings

```r
d <- "cat"
e <- "dog"
```

2.3. Factors

Factors are categorical variables:

```r
d <- factor("cat")
e <- "dog"
```

2.4. Logical Data

Logical operations return TRUE or FALSE:

```
A <- 5
A == A # TRUE
```

```
[1] TRUE
```

```
B <- 5
A == B # TRUE
```

```
[1] TRUE
```

```
a <- 4
A == a # FALSE: R is case-sensitive
```

```
[1] FALSE
```

```
A > a  # TRUE: A is greater than a
```

```
[1] TRUE
```

2.5. Complex Numbers

R supports complex numbers:

```
i <- -1i
i
```

```
[1] 0-1i
```

```
sqrt(i)            # Square root of -1i
```

```
[1] 0.7071068-0.7071068i
```

```r
euler <- exp(i*pi) # Euler's identity
euler
```

```
[1] -1-1.224606e-16i
```

```r
Re(euler)          # Extract the real part
```

```
[1] -1
```

2.6. Constants

R has built-in constants like `pi`:

```r
?Constants
pi
```

```
[1] 3.141593
```

2.1. Vectors

Vectors can hold elements of the same type:

```r
n <- c(1, 2, 3, 4)   # Numeric
l <- c(TRUE, FALSE)  # Logical
k <- c("cat", "dog") # Characters
x <- c(1 + 2i, 4i)   # Complex

# Preference order:
# 1. Characters
# 2. Complex
# 3. Numeric
# 4. Logical / Factors
```

2.2. Constructing Vectors

```r
# Vector of numbers from 0 to 20 by steps of 2:
y <- seq(from = 0, to = 20, by = 2)
w <- 1:4 # Vector of integers from 1 to 4

# Concatenate the vectors:
z <- c(y, w)

# Take samples from random variables:
set.seed(2025)
rbinom(n = 10, size = 5, prob = 0.5)
```

```
 [1] 3 2 3 2 3 3 4 1 3 3
```

```r
rnorm(n = 10, mean = 0, sd = 1)
```

```
 [1] -0.16285434  0.39711189 -0.07998932 -0.34496518  0.70215136 -0.39569639
 [7] -1.75505405 -0.42096376  0.76490961  1.06616211
```

```r
rt(n = 10, df = 5)
```

```
 [1] -0.2206268 -2.0056293 -0.2171123  2.5775456 -0.2399014 -1.4348883
 [7]  1.2676429 -0.6937007 -0.7065318  0.6663071
```

```r
rchisq(n = 10, df = 5)
```

```
 [1] 4.792938 4.666413 2.102163 2.458308 1.923977 7.847497 2.850527 5.289288
 [9] 6.635302 2.455811
```

```r
runif(n = 10, min = 0, max = 1)
```

```
 [1] 0.1097217 0.8087052 0.4355751 0.8042243 0.8617182 0.4337776 0.6402778
 [8] 0.9299222 0.8570572 0.1363258
```

```r
# Random letters and categories:
sample(x = letters, size = 10, replace = TRUE)
```

```
 [1] "q" "o" "m" "d" "n" "n" "l" "t" "w" "i"
```

```r
sample(c("dog", "cat"), size = 5, replace = TRUE)
```

```
[1] "dog" "dog" "dog" "cat" "dog"
```

```r
# Automatic creation of long character vectors:
paste("Sujeto", 1:20, sep = "")
```

```
 [1] "Sujeto1"  "Sujeto2"  "Sujeto3"  "Sujeto4"  "Sujeto5"  "Sujeto6"
 [7] "Sujeto7"  "Sujeto8"  "Sujeto9"  "Sujeto10" "Sujeto11" "Sujeto12"
[13] "Sujeto13" "Sujeto14" "Sujeto15" "Sujeto16" "Sujeto17" "Sujeto18"
[19] "Sujeto19" "Sujeto20"
```

2.3. Vector Indexing

```r
z <- 1:6

# Extract elements:
z[1]
```

```
[1] 1
```

```r
z[c(2, 5)]
```

```
[1] 2 5
```

```r
# Remove elements:
z[-1]
```

```
[1] 2 3 4 5 6
```

```r
z[-c(2, 5)]
```

```
[1] 1 3 4 6
```

2.4. Numeric Vector Functions

```r
x <- sample(x = 1:5, size = 6, replace = TRUE)

mean(x)
```

```
[1] 3.333333
```

```r
median(x)
```

```
[1] 4
```

```r
var(x)
```

```
[1] 2.266667
```

```r
sd(x)
```

```
[1] 1.505545
```

```r
min(x)
```

```
[1] 1
```

```r
max(x)
```

```
[1] 5
```

```r
sum(x)
```

```
[1] 20
```

```r
prod(x)
```

```
[1] 640
```

```r
log(x)
```

```
[1] 1.3862944 1.3862944 0.6931472 1.6094379 0.0000000 1.3862944
```

```r
exp(x)
```

```
[1]  54.598150  54.598150   7.389056 148.413159   2.718282  54.598150
```

```r
length(x)
```

```
[1] 6
```

```r
head(x, n = 3)
```

```
[1] 4 4 2
```

```r
tail(x, n = 3)
```

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

```r
table(x)
```

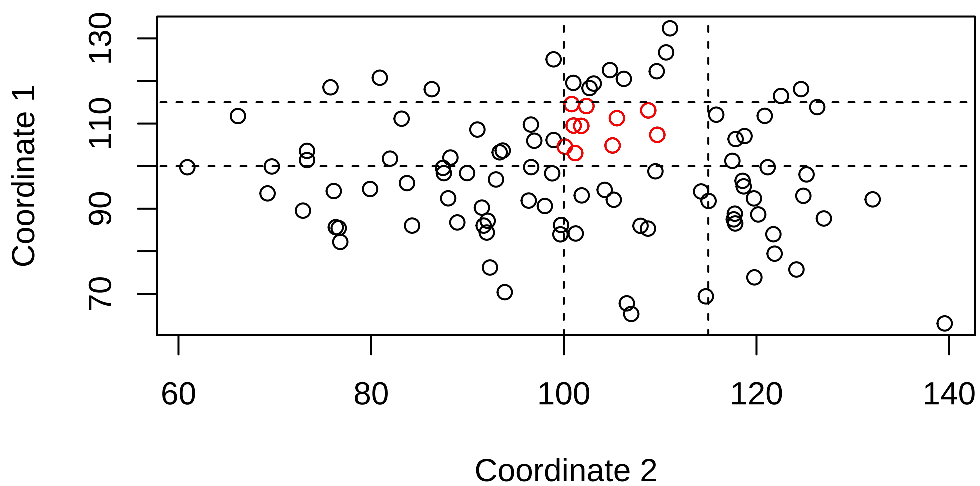
```
x
1 2 4 5
1 1 3 1
```

```r
?mean
x[1] <- NA
mean(x, na.rm = TRUE)
```

```
[1] 3.2
```

2.4.1. Plotting Two Numeric Vectors

```
x <- rnorm(n = 100, mean = 100, sd = 15)
y <- rnorm(n = 100, mean = 100, sd = 15)

plot(x, y, ylab = "Coordinate 1", xlab = "Coordinate 2")

# Highlight specific coordinates:
condition <- x > 100 & x < 115 & y > 100 & y < 115
indices <- which(condition)
points(x[indices], y[indices], col = "red")

# Draw segments:
segments(x0 = 100, x1 = 100, y0 = 0, y1 = 150, lty = "dashed")
segments(x0 = 115, x1 = 115, y0 = 0, y1 = 150, lty = "dashed")
segments(x0 = 0, x1 = 150, y0 = 100, y1 = 100, lty = "dashed")
segments(x0 = 0, x1 = 150, y0 = 115, y1 = 115, lty = "dashed")
```



2.5. Simulating Random Variables

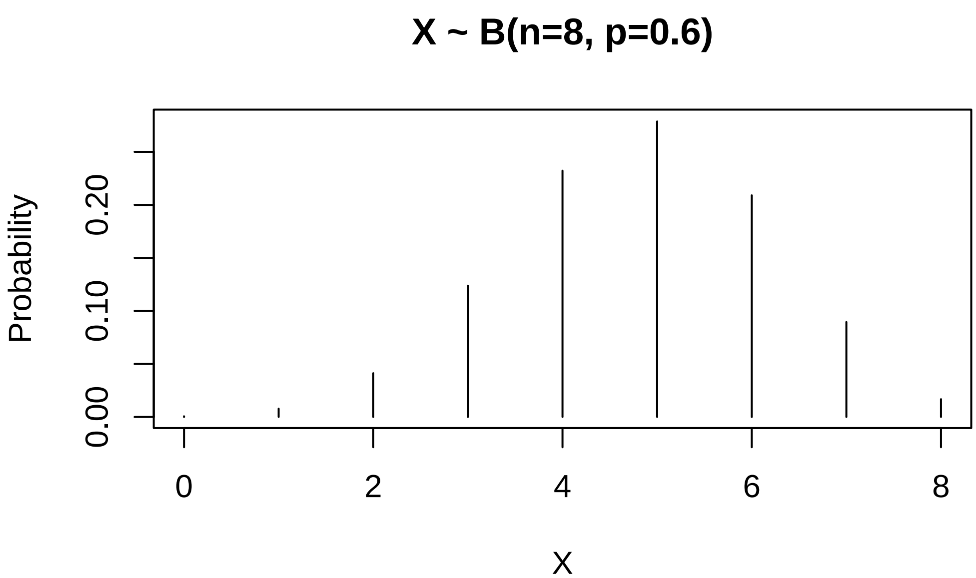
2.5.1. Binomial Distribution

```
set.seed(123)
N <- 1000
n <- 8
```

```r
prob <- 0.60
b <- rbinom(N, size = n, prob = prob)
empirical <- table(b)/1000

p <- dbinom(0:8, size = n, prob = prob)
cbind(empirical, p)
```

```
  empirical          p
0     0.001 0.00065536
1     0.007 0.00786432
2     0.044 0.04128768
3     0.112 0.12386304
4     0.235 0.23224320
5     0.287 0.27869184
6     0.206 0.20901888
7     0.092 0.08957952
8     0.016 0.01679616
```

```r
plot(0:8, p, main = paste("X ~ B(n=", n, ", p=", prob, ")", sep = ""),
     xlab = "X", ylab = "Probability", type = "h")
```



2.5.2. Normal Distribution

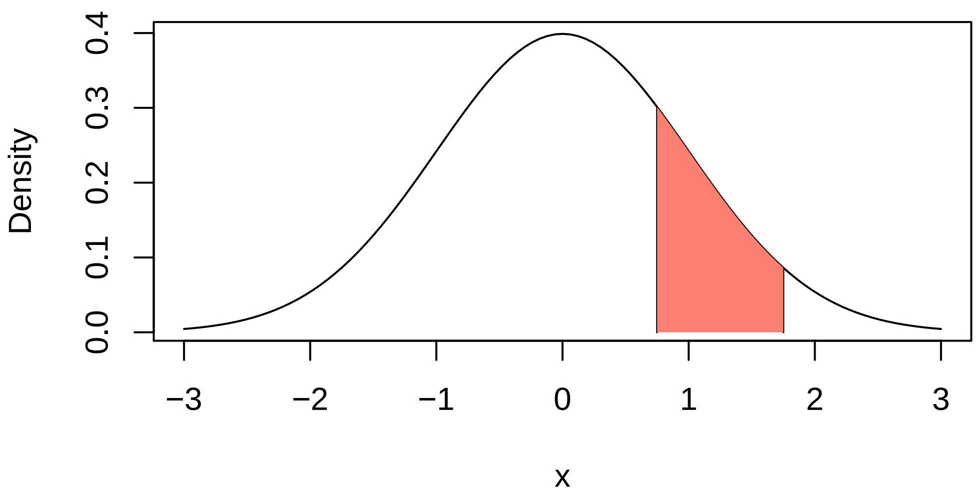
```
X <- rnorm(n = 100000, mean = 0, sd = 1)

x <- 1.25
delta <- 1
mean((X > x-delta/2 & X < x+delta/2)) / delta
```

```
[1] 0.18677
```

```
dnorm(x, mean = 0, sd = 1)
```

```
[1] 0.1826491
```

```
curve(dnorm(x, mean = 0, sd = 1), xlim = c(-3, 3), ylab = "Density")
segments(x0 = x-delta/2, x1 = x-delta/2, y0 = 0, y1 = dnorm(x-delta/2))
segments(x0 = x+delta/2, x1 = x+delta/2, y0 = 0, y1 = dnorm(x+delta/2))

x_fill <- seq(x - delta/2, x + delta/2, length.out = 100)
y_fill <- dnorm(x_fill, mean = 0, sd = 1)

polygon(c(x - delta/2, x_fill, x + delta/2),
        c(0, y_fill, 0),
        col = "salmon", border = NA)
```



2.5.3. Chi-Square Distribution

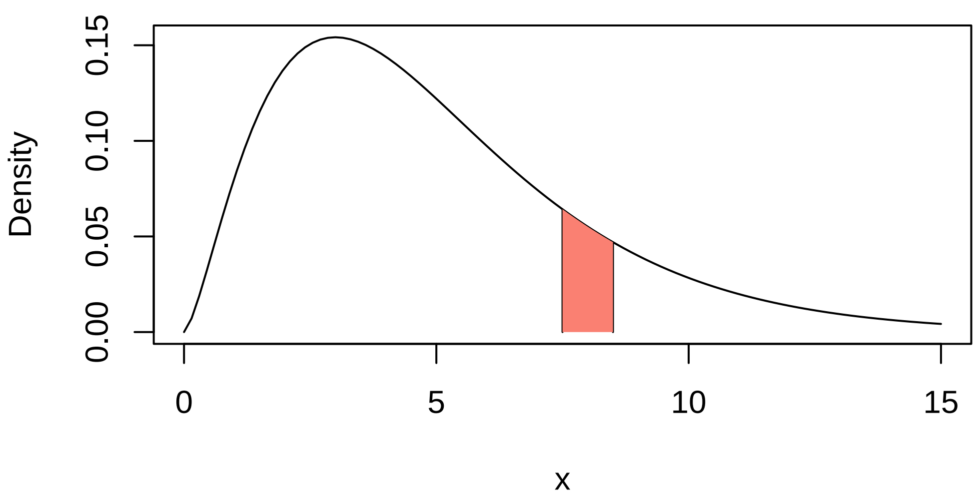
```
df <- 5
X <- rchisq(n = 100000, df = df)

x <- 8
delta <- 1
mean((X > x-delta/2 & X < x+delta/2)) / delta
```

```
[1] 0.05497
```

```
dchisq(x, df = df)
```

```
[1] 0.05511196
```

```
curve(dchisq(x, df = df), xlim = c(0, 15), ylab = "Density")
segments(x0 = x-delta/2, x1 = x-delta/2, y0 = 0, y1 = dchisq(x-delta/2, df = df))
segments(x0 = x+delta/2, x1 = x+delta/2, y0 = 0, y1 = dchisq(x+delta/2, df = df))

x_fill <- seq(x - delta/2, x + delta/2, length.out = 100)
y_fill <- dchisq(x_fill, df = df)

polygon(c(x - delta/2, x_fill, x + delta/2),
        c(0, y_fill, 0),
        col = "salmon", border = NA)
```



3.1. Creation of Matrices

```
# Create a matrix with a number of rows and columns:
N <- matrix(1:4, nrow = 2, ncol = 2)
N
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
# Create a matrix by binding vectors:
x1 <- rnorm(n = 5, mean = 0, sd = 1)
x2 <- runif(n = 5, min = 0, max = 1)
rbind(x1, x2)
```

```
         [,1]        [,2]       [,3]       [,4]      [,5]
x1 -1.0016347 -1.17737291 -0.3945986 0.07629852 1.1837093
x2  0.4718714  0.03208153  0.5716739 0.45549370 0.6932237
```

```
X <- cbind(x1, x2)
X
```

```
              x1         x2
[1,] -1.00163467 0.47187144
[2,] -1.17737291 0.03208153
[3,] -0.39459859 0.57167386
[4,]  0.07629852 0.45549370
[5,]  1.18370929 0.69322374
```

```
# Set names:
colnames(X) <- paste("Score", 1:2, sep = "_")
rownames(X) <- paste("Suject", 1:5, sep = " ")
X
```

```
             Score_1    Score_2
Suject 1 -1.00163467 0.47187144
Suject 2 -1.17737291 0.03208153
Suject 3 -0.39459859 0.57167386
Suject 4  0.07629852 0.45549370
Suject 5  1.18370929 0.69322374
```

3.2. Matrix Indexation

```
X <- matrix(rnorm(2*4), nrow = 2, ncol = 4)
X
```

```
            [,1]       [,2]        [,3]       [,4]
[1,] -0.62904648 -0.4053884 -0.04479366 -0.9214312
[2,] -0.03305434  0.4058965 -0.66945112 -0.1832950
```

```
X[2, 4] # Element in row 2, column 4
```

```
[1] -0.183295
```

```
X[2, ]  # All elements in row 2
```

```
[1] -0.03305434  0.40589648 -0.66945112 -0.18329500
```

```
X[, 4]  # All elements in column 4
```

```
[1] -0.9214312 -0.1832950
```

3.3. Matrix Operations

```
# Matrix by vector multiplication:
set.seed(123)
N <- 20
p <- 4
b <- rnorm(n = p, mean = 0.5, sd = 2)
X <- matrix(rnorm(N*p), nrow = N, ncol = p-1)
```

```
Warning in matrix(rnorm(N * p), nrow = N, ncol = p - 1): data length [80] is
not a sub-multiple or multiple of the number of columns [3]
```

```
X <- cbind(1, X)

e <- rnorm(N, mean = 0, sd = 1.5)
y <- X %*% b + e

fit <- lm(y ~ 0 + X)
summary(fit)
```

```
Call:
lm(formula = y ~ 0 + X)

Residuals:
    Min      1Q  Median      3Q     Max 
-2.3744 -0.5936 -0.2121  0.7138  2.3042 

Coefficients:
   Estimate Std. Error t value Pr(>|t|)    
X1 -0.19338    0.27095  -0.714    0.486    
X2  0.02804    0.28091   0.100    0.922    
X3  3.91972    0.29229  13.410 4.05e-10 ***
X4 -0.10644    0.34319  -0.310    0.760    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.201 on 16 degrees of freedom
Multiple R-squared:  0.9193,    Adjusted R-squared:  0.8991 
F-statistic: 45.58 on 4 and 16 DF,  p-value: 1.5e-08
```

```
fit$coefficients
```

```
         X1          X2          X3          X4 
-0.19338130  0.02804353  3.91971708 -0.10644232 
```

```
solve(t(X) %*% X) %*% t(X) %*% y
```

```
            [,1]
[1,] -0.19338130
[2,]  0.02804353
[3,]  3.91971708
[4,] -0.10644232
```

3.3. Matrix by Matrix Multiplication

```
Z <- matrix(rnorm(6), nrow = 3, ncol = 2)
Y <- matrix(rnorm(6), nrow = 2, ncol = 3)
ZY <- Z %*% Y
ZY
```

```
           [,1]       [,2]       [,3]
[1,] -0.4665383 1.63228136 -0.9965839
[2,] -0.2052576 0.09397571 -0.1378942
[3,]  1.0103096 1.21882211 -0.1309261
```

```
Z[, 1, drop = FALSE] %*% Y[1, , drop = FALSE] +
  Z[, 2, drop = FALSE] %*% Y[2, , drop = FALSE]
```

```
           [,1]       [,2]       [,3]
[1,] -0.4665383 1.63228136 -0.9965839
[2,] -0.2052576 0.09397571 -0.1378942
[3,]  1.0103096 1.21882211 -0.1309261
```

3.3. Matrix Utilities

```
ncol(ZY)
```

```
[1] 3
```

```
nrow(ZY)
```

```
[1] 3
```

```r
rowSums(ZY)
```

```
[1]  0.1691592 -0.2491761  2.0982055
```

```r
colMeans(ZY)
```

```
[1]  0.1128379  0.9816931 -0.4218014
```

```r
# Apply a custom function to each column
X <- matrix(runif(16), nrow = 4, ncol = 4)

geomean <- function(x) {
  n <- length(x)
  result <- prod(x)^(1/n)
  return(result)
}

apply(X, MARGIN = 2, FUN = geomean)
```

```
[1] 0.2393893 0.3770300 0.4454315 0.5436993
```

3.4. Arrays

```r
A <- array(rnorm(2*3*4), dim = c(2, 3, 4))
A
```

```
, , 1

           [,1]       [,2]        [,3]
[1,]  1.8438620 0.23538657 -0.96185663
[2,] -0.6519499 0.07796085 -0.07130809

, , 2

          [,1]        [,2]      [,3]
[1,] 1.4445509  0.04123292 -2.053247
```

```
[2,] 0.4515041 -0.42249683  1.131337

, , 3

           [,1]      [,2]       [,3]
[1,] -1.4606401  1.909104  0.7017843
[2,]  0.7399475 -1.443893 -0.2621975

, , 4

          [,1]       [,2]       [,3]
[1,] -1.572144 -1.6015362 -1.4617556
[2,] -1.514668 -0.5309065  0.6879168
```

```
apply(A, MARGIN = 2, FUN = mean)      # Across dimensions 1 and 3
```

```
[1] -0.08994217 -0.21689360 -0.28616584
```

```
apply(A, MARGIN = c(1, 2), FUN = sum) # Across slices
```

```
           [,1]       [,2]      [,3]
[1,]  0.2556286  0.5841869 -3.775075
[2,] -0.9751660 -2.3193357  1.485748
```

3.5. Data Frames

```
N <- 100
x1 <- sample(c("dog", "cat"), size = N, replace = TRUE)
x2 <- sample(1:4, size = N, replace = TRUE)

df <- data.frame(pet = x1, score = x2)
df$pet
```

```
  [1] "cat" "cat" "dog" "dog" "cat" "cat" "dog" "cat" "dog" "dog" "cat" "dog"
 [13] "dog" "dog" "cat" "dog" "cat" "dog" "cat" "cat" "cat" "cat" "dog" "dog"
 [25] "dog" "cat" "dog" "dog" "cat" "dog" "dog" "cat" "dog" "cat" "cat" "cat"
```

```
[37] "dog" "dog" "dog" "dog" "cat" "dog" "dog" "dog" "dog" "cat" "cat" "dog"
[49] "cat" "dog" "dog" "dog" "cat" "dog" "dog" "dog" "cat" "dog" "dog" "cat"
[61] "dog" "dog" "dog" "cat" "cat" "dog" "dog" "dog" "cat" "dog" "cat" "dog"
[73] "cat" "dog" "dog" "dog" "cat" "cat" "dog" "dog" "cat" "dog" "dog" "cat"
[85] "dog" "cat" "cat" "cat" "cat" "cat" "cat" "cat" "cat" "dog" "cat" "dog"
[97] "dog" "dog" "dog" "dog"
```

```r
df$score
```

```
  [1] 3 3 4 2 3 3 3 1 1 3 2 3 2 1 4 2 2 2 4 1 4 1 1 3 3 1 1 3 1 3 4 2 1 1 3 4 2
 [38] 3 2 1 2 3 2 3 3 1 2 4 1 2 2 4 2 4 2 1 4 2 4 1 4 2 1 4 4 3 2 1 3 2 4 3 2 3
 [75] 3 1 4 2 3 1 1 3 2 3 2 2 2 4 1 4 2 1 4 4 1 4 1 4 2 2
```

```r
fit <- lm(score ~ pet, data = df)
summary(fit)
```

```
Call:
lm(formula = score ~ pet, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.4386 -0.6836 -0.4186  0.5814  1.5814 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  2.41860    0.16769   14.42   <2e-16 ***
petdog       0.01999    0.22211    0.09    0.928    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.1 on 98 degrees of freedom
Multiple R-squared:  8.266e-05,	Adjusted R-squared:  -0.01012 
F-statistic: 0.008101 on 1 and 98 DF,  p-value: 0.9285
```

3.6. Lists

```r
L <- list(1, 1:5, matrix(1:16, nrow = 4, ncol = 4))
L
```

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

[[2]]
[1] 1 2 3 4 5

[[3]]
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

```r
# Multiply and sum rank-1 matrices using lapply
f <- function(i) {
  x <- Z[, i, drop = FALSE] %*% Y[i, , drop = FALSE]
  return(x)
}
ZY <- lapply(1:ncol(Z), FUN = f)
ZY
```

```
[[1]]
           [,1]       [,2]        [,3]
[1,] 0.54751085 1.53960722 -0.49427754
[2,] 0.02590656 0.07284958 -0.02338772
[3,] 0.45159240 1.26988337 -0.40768504

[[2]]
           [,1]        [,2]       [,3]
[1,] -1.0140492  0.09267413 -0.5023063
[2,] -0.2311642  0.02112613 -0.1145065
[3,]  0.5587172 -0.05106126  0.2767589
```

```r
ZY[[1]] + ZY[[2]]
```

```
           [,1]       [,2]       [,3]
[1,] -0.4665383 1.63228136 -0.9965839
```

```
[2,] -0.2052576 0.09397571 -0.1378942
[3,]  1.0103096 1.21882211 -0.1309261
```

3.7. Objects in the Environment

```r
objects()
```

```
 [1] "a"          "A"          "b"          "B"          "c"          "condition"
 [7] "d"          "delta"      "df"         "e"          "empirical"  "euler"
[13] "f"          "fit"        "geomean"    "i"          "indices"    "k"
[19] "l"          "L"          "n"          "N"          "p"          "prob"
[25] "w"          "x"          "X"          "x_fill"     "x1"         "x2"
[31] "y"          "Y"          "y_fill"     "z"          "Z"          "ZY"
```

```r
ls()
```

```
 [1] "a"          "A"          "b"          "B"          "c"          "condition"
 [7] "d"          "delta"      "df"         "e"          "empirical"  "euler"
[13] "f"          "fit"        "geomean"    "i"          "indices"    "k"
[19] "l"          "L"          "n"          "N"          "p"          "prob"
[25] "w"          "x"          "X"          "x_fill"     "x1"         "x2"
[31] "y"          "Y"          "y_fill"     "z"          "Z"          "ZY"
```

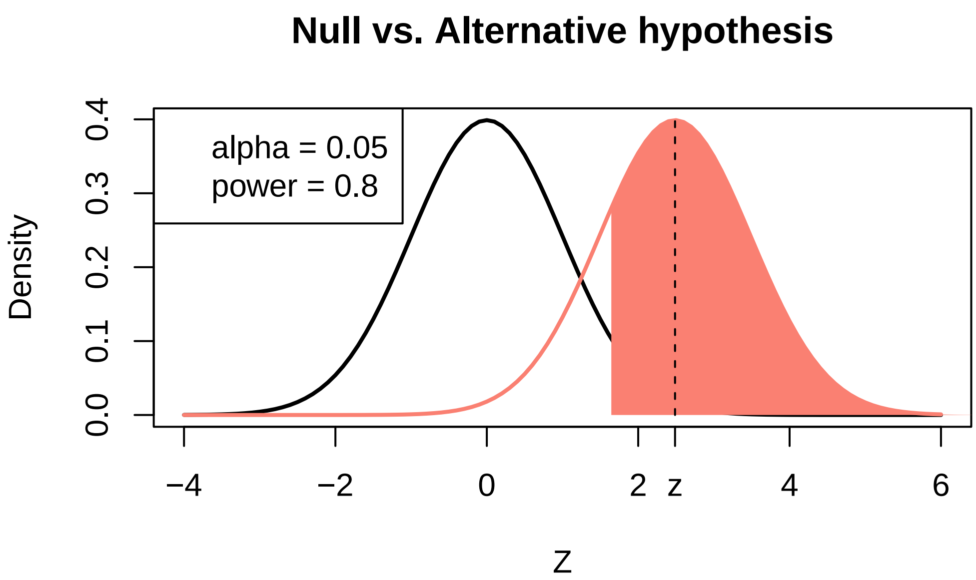
4.1. Simulation Setup for the Linear Model

```r
set.seed(123)

# Setup:
N <- 20                               # Sample size
x <- rnorm(N, mean = 0, sd = 1)  # Predictor
sigma <- 1.5                          # Standard deviation of the errors
true_se <- sigma / sqrt(N)            # True standard error of b
power <- 0.80                         # Statistical power
```

```r
alpha <- 0.05                    # Significance level
threshold <- qnorm(1 - alpha)   # One-sided threshold

# Get the b that gives the desired statistical power:
b <- qnorm(power, mean = threshold, sd = 1) * true_se
```

4.2. Visual Insight into Statistical Power

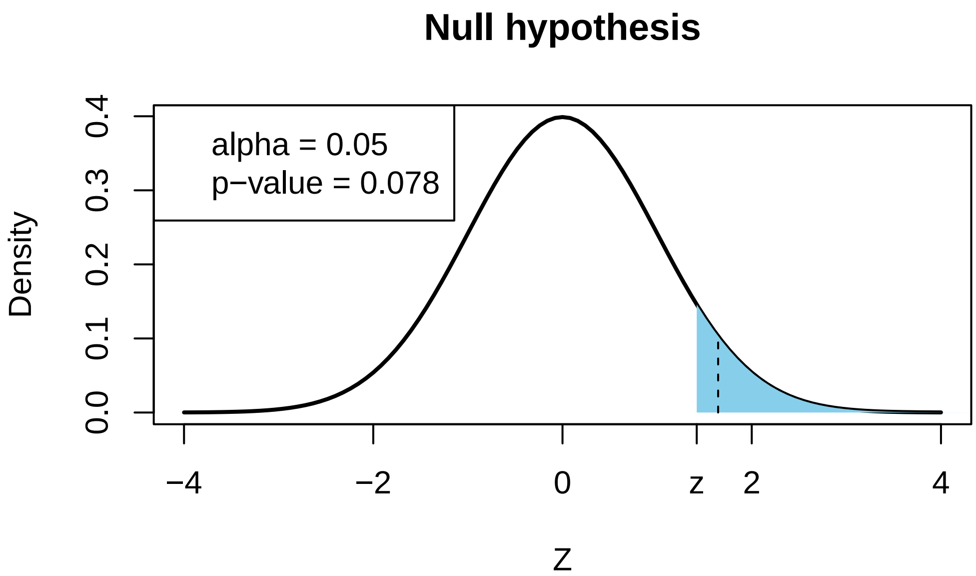
```r
z_statistic <- b / true_se

# Visualize the null hypothesis:
curve(dnorm(x, mean = 0, sd = 1), xlim = c(-4, 6), lwd = 2,
      xlab = "Z", ylab = "Density", main = "Null vs. Alternative hypothesis")

# Visualize the alternative hypothesis:
curve(dnorm(x, mean = z_statistic, sd = 1), col = "salmon", lwd = 2, add = TRUE)

# Shaded area under the alternative curve beyond the threshold:
x_fill <- seq(threshold, 10, length.out = 100)
y_fill <- dnorm(x_fill, mean = z_statistic, sd = 1)

polygon(c(threshold, x_fill, 10),
        c(0, y_fill, 0),
        col = "salmon", border = NA)

# Highlight the true z-statistic:
segments(x0 = z_statistic,
         x1 = z_statistic,
         y0 = 0,
         y1 = dnorm(z_statistic, mean = z_statistic),
         lty = "dashed")

axis(1, at = z_statistic, label = "z")
legend(x = "topleft", legend = c(paste("alpha =", alpha),
                                 paste("power =", power)))
```



4.3. Visual Insight into the P-value

```
# Generate a random z-statistic under the alternative hypothesis:
z_statistic <- rnorm(1, mean = b / true_se, sd = 1)
pval <- round(1 - pnorm(z_statistic), 3)

curve(dnorm(x, mean = 0, sd = 1), xlim = c(-4, 4), lwd = 2,
      xlab = "Z", ylab = "Density", main = "Null hypothesis")

# Shade the p-value region:
x_fill <- seq(z_statistic, 10, length.out = 100)
y_fill <- dnorm(x_fill, mean = 0, sd = 1)

polygon(c(z_statistic, x_fill, 10),
        c(0, y_fill, 0),
        col = "skyblue", border = NA)

segments(x0 = threshold,
         x1 = threshold,
         y0 = 0,
         y1 = dnorm(threshold, mean = 0, sd = 1),
         lty = "dashed")
```

```
axis(1, at = z_statistic, label = "z")
legend(x = "topleft", legend = c(paste("alpha =", alpha),
                                 paste("p-value =", pval)))
```



4.4. Simulating Power

```
# Simulation setup:
nsim <- 1000
pval <- vector(length = nsim)

for(i in 1:nsim) {
  e <- rnorm(N, mean = 0, sd = sigma)
  y <- x * b + e
  fit <- lm(y ~ 0 + x)
  z_statistic <- fit$coefficients / true_se
  pval[i] <- 1 - pnorm(z_statistic, mean = 0, sd = 1)
}

# Empirical power estimate:
mean(pval < alpha)
```

```
[1] 0.789
```

```
power
```

```
[1] 0.8
```

5.1. Linear Model with Multiple Predictors

```
set.seed(123)
N <- 20
p <- 4
b <- rnorm(n = p, mean = 0.5, sd = 2)
X <- matrix(rnorm(N*p), nrow = N, ncol = p-1)
```

```
Warning in matrix(rnorm(N * p), nrow = N, ncol = p - 1): data length [80] is
not a sub-multiple or multiple of the number of columns [3]
```

```
X <- cbind(1, X)

nsim <- 1000
sigma <- 1.5
true_se <- sqrt(diag(sigma^2 * solve(t(X) %*% X)))
power <- c(0.80, 0.60, 0.40, 0.05)
alpha <- 0.05
threshold <- qt(1 - alpha, df = N - p)

f <- function(power, threshold) {
  suppressWarnings(uniroot(\(x) (1 - pt(threshold, df = N - p, ncp = x)) - power,
                           interval = c(-6, 6))$root)
}
t_stat <- sapply(power, FUN = f, threshold = threshold)
b <- t_stat * true_se

# Initialize result storage:
coefs <- matrix(NA, nsim, p)
se <- matrix(NA, nsim, p)
ts <- matrix(NA, nsim, p)
```

```r
pval <- matrix(NA, nsim, p)
upper <- matrix(NA, nsim, p)
lower <- matrix(NA, nsim, p)
error <- vector(length = nsim)

for (i in 1:nsim) {
  e <- rnorm(N, mean = 0, sd = sigma)
  y <- X %*% b + e
  fit <- lm(y ~ 0 + X)
  coefs[i, ] <- coefficients(fit)
  error[i] <- sum(resid(fit)^2) / (N - p)
  se[i, ] <- sqrt(diag(error[i] * solve(t(X) %*% X)))
  ts[i, ] <- coefs[i, ] / se[i, ]
  upper[i, ] <- coefficients(fit) + qnorm(1 - alpha/2) * se[i, ]
  lower[i, ] <- coefficients(fit) + qnorm(alpha/2) * se[i, ]
  pval[i, ] <- (1 - pt(ts[i, ], df = N - p))
}
```

5.1. Checking Simulation Accuracy

```r
cbind(colMeans(coefs), b)     # Linear coefficients
```

```
                         b
[1,]  0.87159473  8.793200e-01
[2,]  0.69604900  6.956320e-01
[3,]  0.52666068  5.304025e-01
[4,] -0.02259479 -7.209672e-08
```

```r
cbind(colMeans(se), true_se)  # Standard errors
```

```
                true_se
[1,] 0.3328807 0.3383593
[2,] 0.3451118 0.3507917
[3,] 0.3590934 0.3650034
[4,] 0.4216360 0.4285753
```

```r
cbind(colMeans(ts), t_stat)    # t statistics
```

```
                        t_stat
[1,]  2.69519451  2.598776e+00
[2,]  2.08044868  1.983035e+00
[3,]  1.51452823  1.453144e+00
[4,] -0.05401576 -1.682241e-07
```

```r
apply(pval, MARGIN = 2, FUN = \(x) mean(x < alpha)) # Empirical power
```

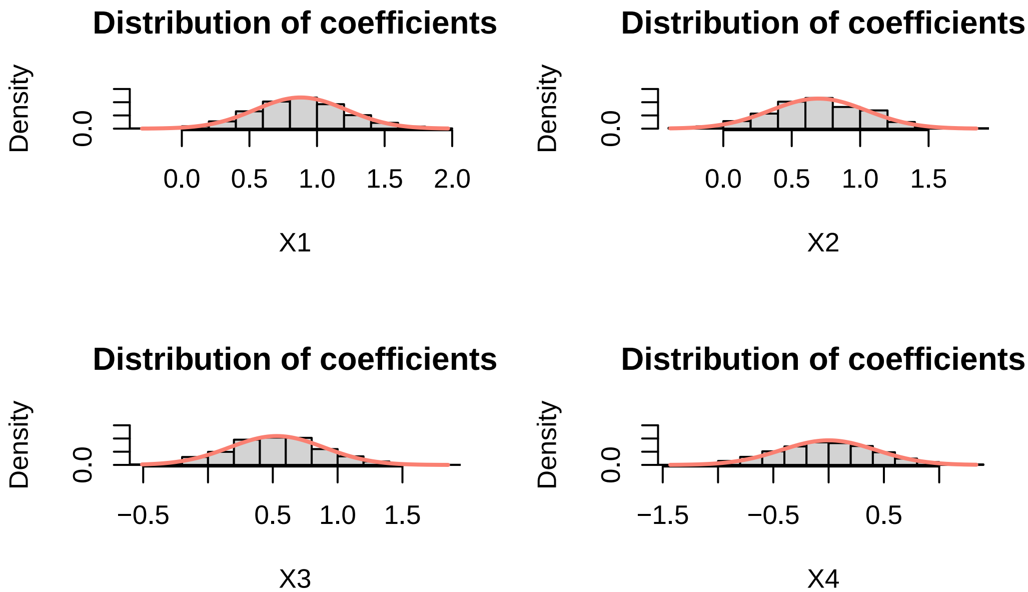
```
[1] 0.799 0.585 0.398 0.045
```

```r
power
```

```
[1] 0.80 0.60 0.40 0.05
```

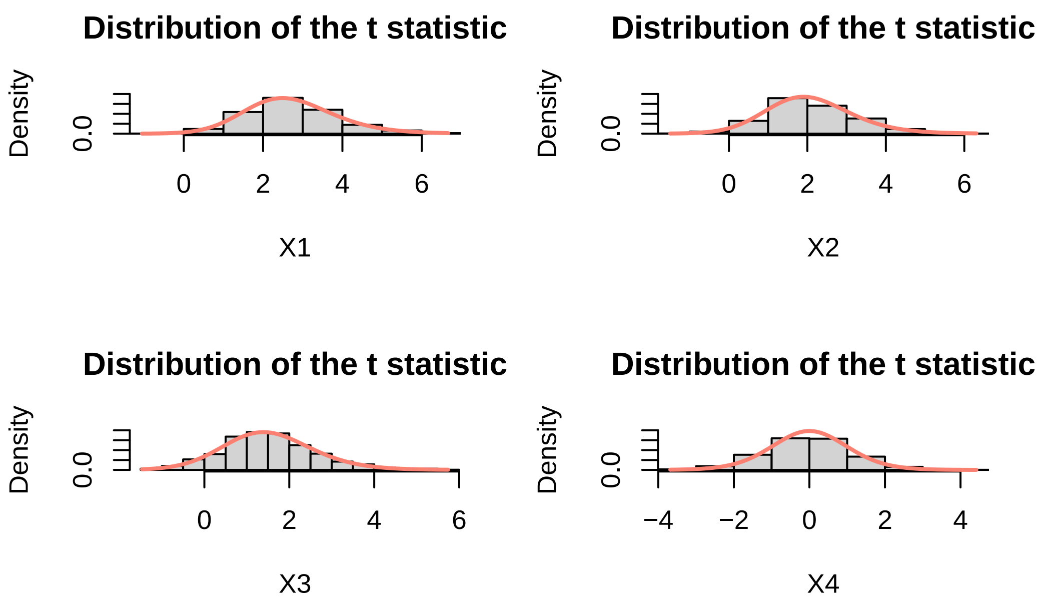
5.2. Distribution of Coefficients

```r
par(mfrow = c(2, 2))
for (i in 1:p) {
  hist(coefs[, i], main = "Distribution of coefficients", freq = FALSE,
       xlim = range(coefs[, i]), ylim = c(0, 1.5),
       xlab = names(coefficients(fit))[i])
  curve(dnorm(x, b[i], sd = true_se[i]), lwd = 2, col = "salmon", add = TRUE)
}
```



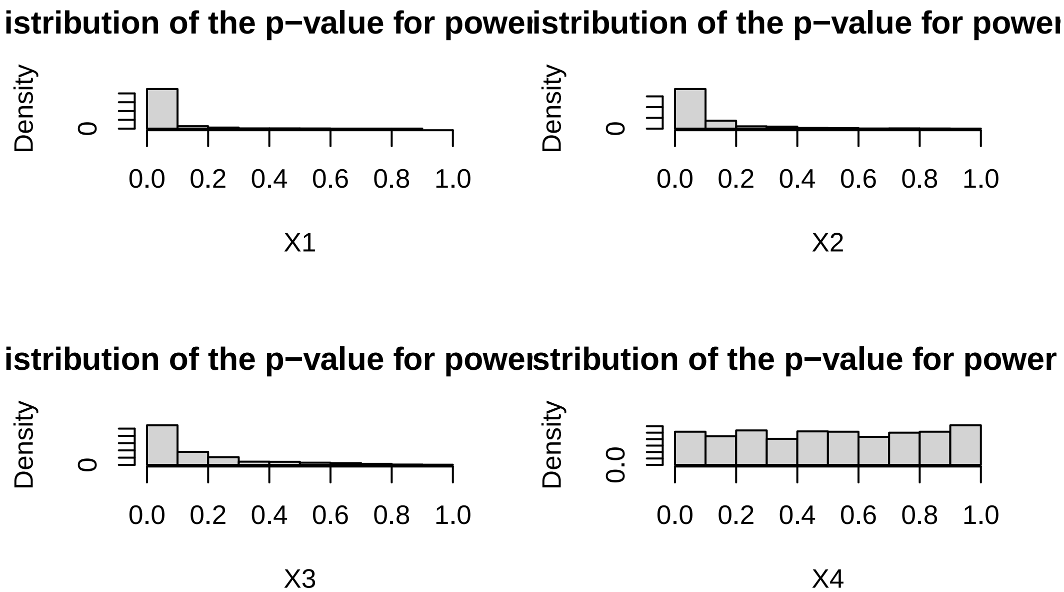
5.3. Distribution of t Statistics

```
par(mfrow = c(2, 2))
for (i in 1:p) {
  hist(ts[, i], main = "Distribution of the t statistic", freq = FALSE,
       xlim = range(ts[, i]), ylim = c(0, 0.4),
       xlab = names(coefficients(fit))[i])
  curve(dt(x, df = N - p, ncp = t_stat[i]), lwd = 2, col = "salmon", add = TRUE)
}
```



5.4. P-curve

```
par(mfrow = c(2, 2))
for (i in 1:p) {
  title <- paste("Distribution of the p-value for power =", power[i])
  hist(pval[, i], main = title, freq = FALSE,
       xlim = c(0, 1), xlab = names(coefficients(fit))[i])
}
```



5.5. The Dance of Confidence Intervals

```
par(mfrow = c(1, 1))
success <- rep(0, p)
failure <- rep(0, p)
nsim <- 20

for (i in 1:nsim) {
  plot(b, 1:p, xlab = "Value", ylab = "Coefficient",
       ylim = c(-1, 4.5), xlim = c(-2, 5),
       main = "The Dance of confidence intervals",
       yaxt = "n")
  axis(2, labels = paste(1:p), at = 1:p)

  for (j in 1:p) {
    segments(x0 = upper[i, j], x1 = lower[i, j], y0 = j, y1 = j)

    if (b[j] > lower[i, j] & b[j] < upper[i, j]) {
      points(b[j], j, bg = "skyblue", pch = 21)
      success[j] <- success[j] + 1
    } else {
```

```
      points(b[j], j, bg = "red", pch = 21)
      failure[j] <- failure[j] + 1
    }

    text(x = 4.5, y = j, labels = paste(success[j], failure[j], sep = "/"))
  }

  text(x = 1, y = 0, paste("Iteration", i, sep = "="))
  Sys.sleep(2)
}
```