Lecture 2: R Markdown

Basic manipulations

2) To create a header of type 2 one can just use `##` before text.

```
## Header Text
```

3) To perform a linear regression of ”Sepal Length” as the response variable and ”Sepal Width” as the explanatory variable using the `iris` dataset, we use the `lm()` function. Below is the code for this task.

```
# Load dataset
data(iris)

# Fit linear model
model <- lm(Sepal.Length ~ Sepal.Width, data = iris)

# Saving the fitted model
saveRDS(model, file = "linear_model.rds")
```

The function ”saveRDS” saves a model as a .rds object and it does not preserve model’s name. ”saveRDS” works in pair with ”readRDS”.

4) To write text in `monochrome` style one can use 2 symbols ”‘” before and after the text.

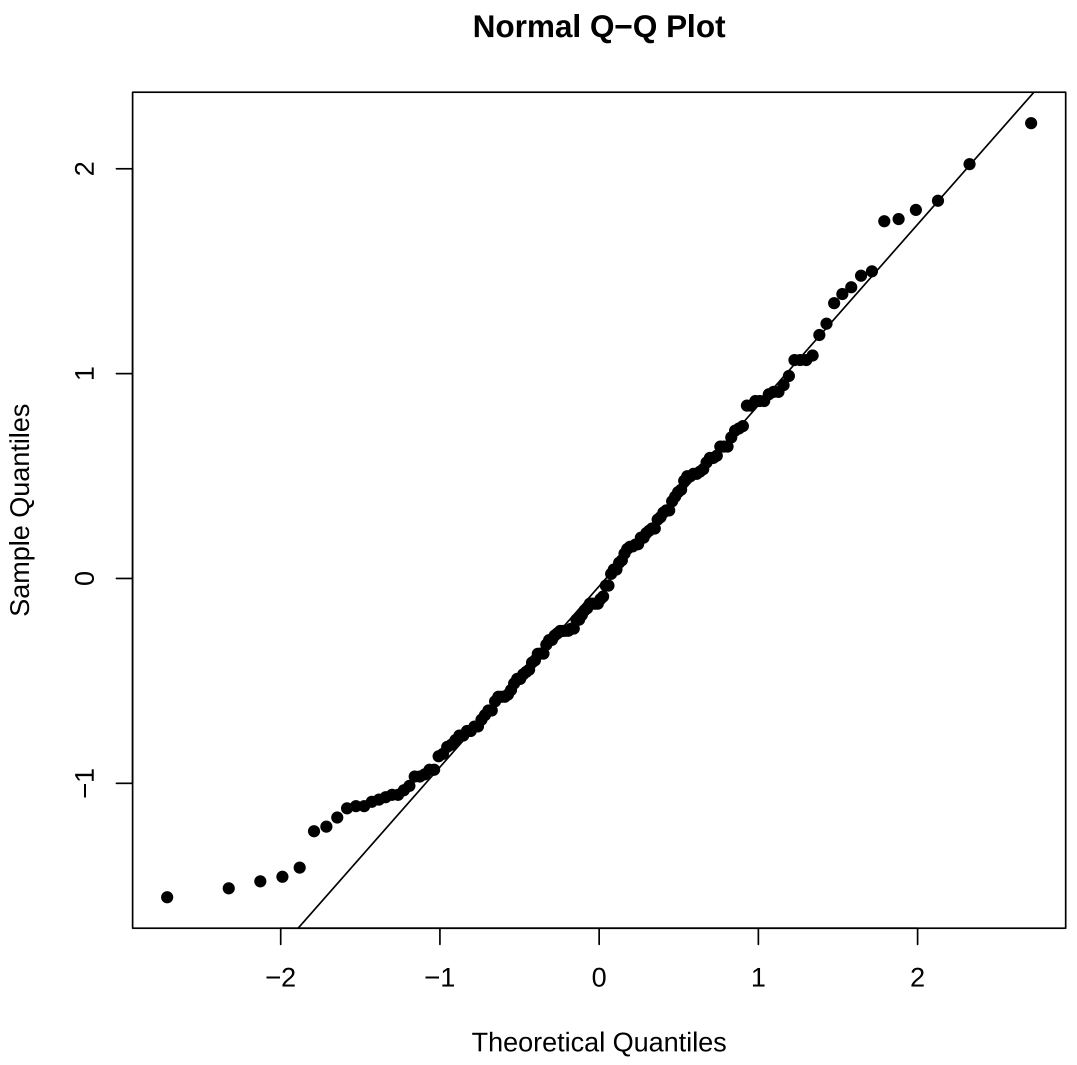
5) The summary output will display the regression coefficients, standard errors, and statistical significance.

```
# To load a model one can use the following command:
readRDS("linear_model.rds")

##
## Call:
## lm(formula = Sepal.Length ~ Sepal.Width, data = iris)
##
## Coefficients:
## (Intercept)  Sepal.Width
##      6.5262      -0.2234

# Print summary
summary(model)

##
## Call:
## lm(formula = Sepal.Length ~ Sepal.Width, data = iris)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.5561 -0.6333 -0.1120  0.5579  2.2226
```

```
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)     6.5262     0.4789   13.63   <2e-16 ***
## Sepal.Width    -0.2234     0.1551   -1.44    0.152
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8251 on 148 degrees of freedom
## Multiple R-squared:  0.01382, Adjusted R-squared:  0.007159
## F-statistic: 2.074 on 1 and 148 DF,  p-value: 0.1519
```

6) For better visualization, we can create a QQ plot of residuals:

```
# QQ plot with filled dots
qqnorm(residuals(model), pch = 16)
qqline(residuals(model))
```



7) To display the first few rows of the `iris` dataset using `kable`, we proceed as follows:

```r
# Install and load knitr
library(knitr)

# Print the head of the dataset using kable
kable(head(iris), row.names = FALSE)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

8) To remove the period from the column names, we can rename the columns:

```r
# Rename columns by removing periods
colnames(iris) <- gsub("\\.", " ", colnames(iris))
kable(head(iris), row.names = FALSE)
```

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

More advanced manipulations

3) To cite the paper arXiv:math/0303109, we need to include a BibTeX entry in the RMarkdown file, to make a BibTex file one can use Google Scholar. Add the reference in the bibliography section and use `@perelman2003ricci` for in-text citation.

Lecture 3: GitHub

Solution: Collaborative Workflow in GitHub

3)

1. Person A and Person B should collaborate using the following Git commands:

```
# For Person B: Make changes, commit, and push
git pull
git add .
git commit -m "Changes by Person B"
git push

# For Person A: Pull the changes made by Person B
git pull
```

2. If both modify different sections without conflicts, they can simply merge without issues. However, in the event of conflicts (both editing the same section), they need to resolve the conflict manually:

```
# Pull changes after a conflict
git pull
# Resolve the conflicts manually in the editor
git add .
git commit -m "Resolved conflict"
git push
```

Lecture 4: Data Structures

1. Matrix Dimensions and Products

The matrices A and B have dimensions 10×2. Their transpose products are calculated as follows:

```
# Create matrices A and B
set.seed(1)
A <- matrix(rnorm(20), ncol = 2)
B <- matrix(rnorm(20), ncol = 2)

# Matrix multiplication
A_t_B <- t(A) %*% B
A_B_t <- A %*% t(B)

# Display results
A_t_B

##           [,1]      [,2]
## [1,] -4.982433 -1.228744
## [2,]  5.223403  1.668847

A_B_t

##              [,1]       [,2]        [,3]       [,4]       [,5]        [,6]
##  [1,]  1.4783293 -0.6453648  0.53936312  1.1648955 -2.4701049 -0.59221890
##  [2,]  0.6984361  0.1035630  0.16482452 -0.3863067 -0.4230105 -0.17209049
##  [3,] -1.6119907 -0.5897196 -0.30314597  1.6957851  0.3375411  0.30471424
##  [4,] -1.5430405  1.4753710 -0.73962419 -3.0544126  4.0385698  0.82954731
##  [5,]  1.8312308  0.1420909  0.46067352 -0.7160338 -1.3448595 -0.48533507
##  [6,] -0.8150423 -0.6370995 -0.07859770  1.6346178 -0.4466712  0.06469906
##  [7,]  0.4259389  0.3829001  0.03006863 -0.9687967  0.3244160 -0.02063991
##  [8,]  1.9608747  0.4804558  0.42095167 -1.5195706 -0.8420860 -0.43312813
##  [9,]  1.6449065  0.3659280  0.36129727 -1.1896174 -0.7739864 -0.37312021
## [10,]  0.5262766 -0.2999011  0.20746740  0.5755700 -1.0071251 -0.22932475
##             [,7]       [,8]       [,9]      [,10]
##  [1,] -0.4984814  0.8316896  1.9625366  0.8919336
##  [2,] -0.1823221 -0.2932168  0.3410284  0.3742711
##  [3,]  0.3751361  1.2658507 -0.2838245 -0.8233597
##  [4,]  0.6246963 -2.2149017 -3.1990097 -1.0234711
##  [5,] -0.4948848 -0.5513478  1.0798984  0.9962350
##  [6,]  0.1455422  1.2093710  0.3428789 -0.3772001
##  [7,] -0.0695556 -0.7159271 -0.2508739  0.1913608
##  [8,] -0.4871728 -1.1418750  0.6852138  1.0288895
##  [9,] -0.4135034 -0.8955412  0.6280543  0.8673791
## [10,] -0.1865912  0.4139244  0.7993280  0.3256166
```

2. Combine Matrices

Combining A and B row-wise to create C:

```r
# Combine matrices row-wise
C <- rbind(A, B)
```

3. Covariance Matrix

The unbiased estimator of the covariance matrix is given by $\frac{1}{n-1}D^T D$. Here is how to compute it and compare with `cov(C)`:

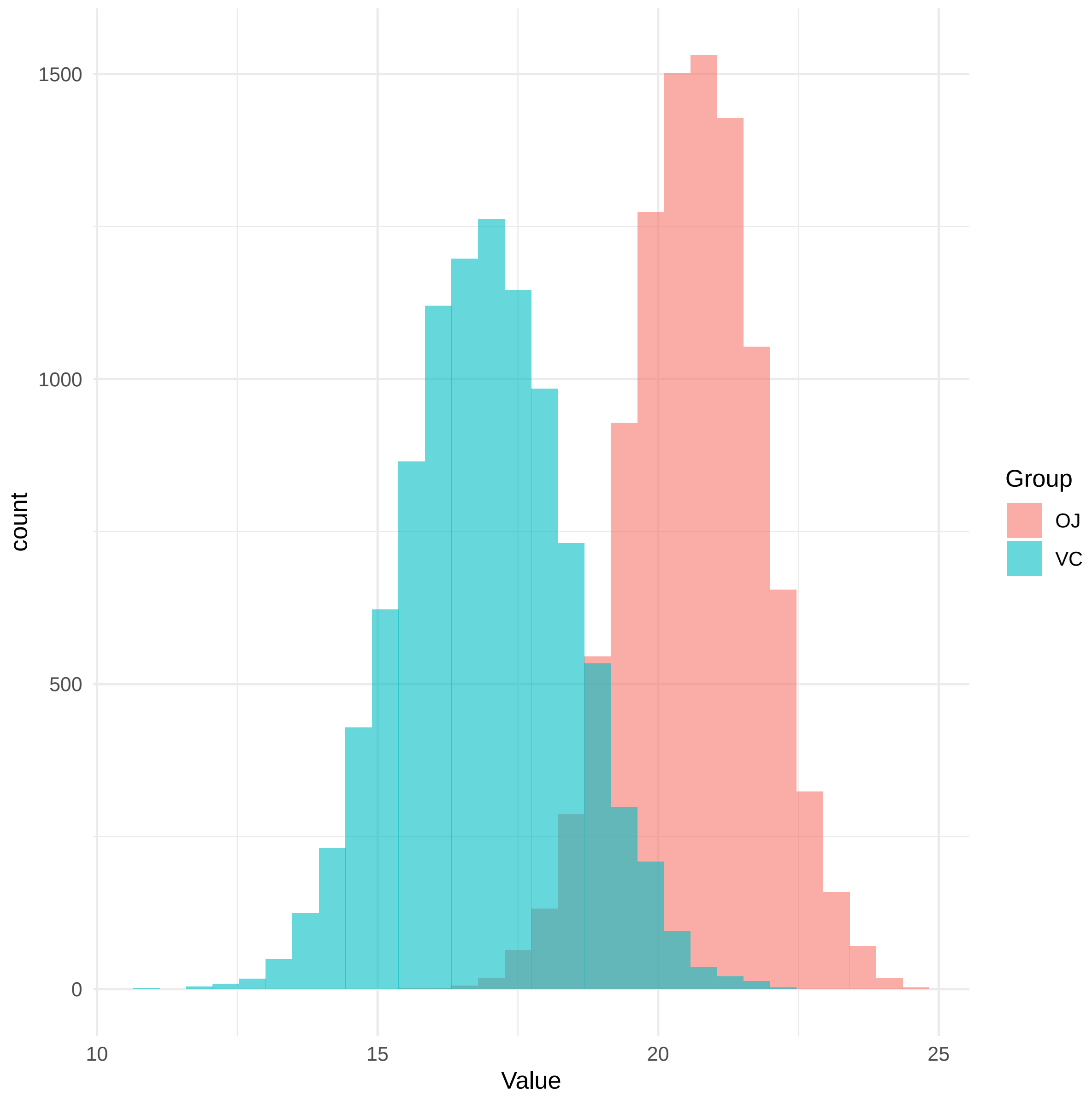
```r
# Center matrix C column-wise
D <- scale(C, center = TRUE, scale = FALSE)

# Compute covariance estimator
cov_estimator <- (t(D) %*% D) / (nrow(D) - 1)

# Compare with built-in cov(C)
cov_estimator

##            [,1]       [,2]
## [1,]  0.7397870 -0.1096401
## [2,] -0.1096401  0.8558324

cov(C)

##            [,1]       [,2]
## [1,]  0.7397870 -0.1096401
## [2,] -0.1096401  0.8558324
```

Lecture 5: Control Structures

Solution: Bootstrap Distribution

Using the `ToothGrowth` dataset, we can compute the bootstrap distribution for each vector and plot the histograms.

```r
# Load dataset
data("ToothGrowth")

# Create vectors for OJ and VC factors
OJ_length <- ToothGrowth$len[ToothGrowth$supp == "OJ"]
VC_length <- ToothGrowth$len[ToothGrowth$supp == "VC"]

# Compute means
mean(OJ_length)

## [1] 20.66333

mean(VC_length)

## [1] 16.96333

# Bootstrap function using replicate function
bootstrap <- function(x, B = 10000) {
  replicate(B, mean(sample(x, replace = TRUE)))
}

# Alternatively, one can create the function using for loop:
bootstrap_for <- function(x, B = 10000) {
  y <- rep(NA, B)
  for (i in 1:B){
    y[i] <- mean(sample(x, replace = TRUE))
  }
  return(y)
}


# Generate bootstrap distributions
set.seed(123)
bootstrap_OJ <- bootstrap(OJ_length)
bootstrap_VC <- bootstrap(VC_length)

# Plot histograms using ggplot2
library(ggplot2)
df <- data.frame(Value = c(bootstrap_OJ, bootstrap_VC),
                 Group = rep(c("OJ", "VC"), each = 10000))

ggplot(df, aes(x = Value, fill = Group)) +
  geom_histogram(alpha = 0.6, position = "identity", bins = 30) +
  theme_minimal()
```



Lecture 6: Functions

Solution 1: Function Return Value

The following code returns 4:

```r
x <- 2
f1 <- function(x) {
  function() {
    x + 3
  }
}
f1(1)()  # returns 4 because x inside f1 is set to 1

## [1] 4
```

Solution 2: Simplifying Expressions

The following code expressions can be simplified:

```
# Simplified
1 + 2 * 3

## [1] 7

3 * (2 + 1)

## [1] 9
```

Solution 3: Improving Readability

This function call can be made more readable:

```
mean(x = c(seq(10), rep(NA, 3)), na.rm = TRUE)

## [1] 5.5
```

Solution 4: Error Handling in Function

The following code throws an error because the first argument of `f2()` is evaluated first, leading to an error in the second call:

```
f2 <- function(a, b) {
  a * 3
}
f2(3, stop("This is an error!"))  # No error

## [1] 9

f2(stop("This is an error!"), 3)  # Error because stop is called

## Error in f2(stop("This is an error!"), 3):  This is an error!
```

Solution 5: Infix Function

An example of an infix function in R could be for string concatenation:

```
`%concat%` <- function(a, b) {
  paste(a, b, sep = "")
}

# Usage
"Hello" %concat% " World"  # returns "Hello World"

## [1] "Hello World"
```