R Programming: Data Import/Export

1 Introduction

Data import and export are fundamental skills in R programming. This document provides an overview of how to handle text files, Excel files, and databases in R, with practical examples, exercises, and tips to streamline the process.

2 Data Import/Export in R

2.1 Importing and Exporting Text Files

Theoretical Part: Text files are a common format for storing tabular data. R provides several functions to work with these files:

- `read.table()` and `read.csv()`: Used for reading text files with tabular data.
- `write.table()` and `write.csv()`: Used for writing data frames to text files.

Practical Examples:
Importing Text Files

```
# Reading a CSV file
data <- read.csv("data.csv")
head(data)

# Reading a tab-delimited text file
data <- read.table("data.txt", sep = "\t", header = TRUE)
head(data)
```

Exporting Text Files

```
# Writing to a CSV file
write.csv(data, "output.csv", row.names = FALSE)

# Writing to a tab-delimited text file
write.table(data, "output.txt", sep = "\t", row.names = FALSE)
```

2.2 Importing and Exporting Excel Files

Theoretical Part: Excel files are widely used in data storage and sharing. R supports importing and exporting Excel files via the following packages:

- `readxl`: For reading Excel files.
- `writexl`: For writing Excel files.
- `openxlsx`: Comprehensive package for reading and writing Excel files.

Practical Examples:
Importing Excel Files

```
# Install and load the readxl package
install.packages("readxl")
library(readxl)

# Read an Excel file
data <- read_excel("data.xlsx")
head(data)

# Specify a sheet
data <- read_excel("data.xlsx", sheet = "Sheet1")
```

Exporting Excel Files

```
# Install and load the writexl package
install.packages("writexl")
library(writexl)

# Write data to an Excel file
write_xlsx(data, "output.xlsx")
```

2.3 Importing and Exporting Data from Databases

Theoretical Part: R provides tools to interact with databases, enabling efficient querying and data retrieval:

- `DBI`: Provides a standardized interface for database interaction.
- `RSQLite`: A lightweight database driver for SQLite.
- `RODBC` and `odbc`: For connecting to other databases like MySQL, PostgreSQL, etc.

Practical Examples:
Connecting to a SQLite Database

```r
# Install and load necessary packages
install.packages("DBI")
install.packages("RSQLite")
library(DBI)
library(RSQLite)

# Connect to a SQLite database
con <- dbConnect(RSQLite::SQLite(), dbname = "my_database.sqlite")

# List tables
dbListTables(con)

# Read a table into R
data <- dbReadTable(con, "table_name")

# Execute a custom SQL query
query_result <- dbGetQuery(con, "SELECT * FROM table_name WHERE column1
```

Exporting Data to a SQLite Database

```r
# Create a new database and write a data frame to it
con <- dbConnect(RSQLite::SQLite(), dbname = "new_database.sqlite")
dbWriteTable(con, "new_table", data)

# Disconnect from the database
dbDisconnect(con)
```

2.4 Practice Questions and Exercises

Questions:

1. What function would you use to read a tab-delimited text file in R?
2. How can you export a data frame as an Excel file using the `writexl` package?
3. Explain the process of connecting to a SQLite database in R.

Exercises:

1. Import a CSV file, manipulate the data (e.g., filter rows where a specific column is greater than a threshold), and save the result as a new CSV file.

2. Read an Excel file with multiple sheets, combine data from two sheets into a single data frame, and write it back to an Excel file.

3. Connect to a database, retrieve a subset of data using a SQL query, and save the result as a text file.

2.5 Tips and Tricks

- Always specify the `header` argument in `read.table()` or `read.csv()` to avoid misinterpreting the first row.
- Use `row.names = FALSE` when writing data to avoid including row numbers unnecessarily.
- For Excel, the `openxlsx` package offers more customization options compared to `writexl`.
- When working with large databases, retrieve data in chunks using `dbFetch()` to avoid memory issues.