Lecture 8: Importing, Cleaning, and Transforming Data in R

Dr. Logan Kelly

2024-09-04

Overview

- In this lecture, we'll cover:
  - How to **import datasets** into R from various file formats.
  - Essential steps for **cleaning** business datasets, including handling missing data and correcting data types.
  - Practical examples of **transforming** data for analysis.

1. Importing Data into R

R provides multiple functions to import data from common file formats such as CSV, Excel, and databases. Here, we focus on importing **CSV** and **Excel** files, which are widely used in business analytics.

Importing Data from a CSV File

You can import data from a CSV file using the `read.csv()` function.

Example: Importing Sales Data from a CSV

```
# Importing sales data from a CSV file
sales_data <- read.csv("path/to/your/data.csv", sep = ",")
head(sales_data)  # View the first few rows of the data
```

- **Explanation**: The `read.csv()` function reads the CSV file and converts it into a data frame in R. The `head()` function displays the first few rows of the dataset, making it easy to quickly inspect the data.

Importing Data from an Excel File

To import Excel files, you can use the `readxl` package and the `read_excel()` function.

Example: Importing Data from an Excel File

```
# Loading the readxl package
library(readxl)

# Importing data from an Excel file
sales_data_excel <- read_excel("path/to/your/data.xlsx", sheet = 1)
head(sales_data_excel)
```

```
# Working example
# Loading the readxl package
library(readxl)

# Importing data from an Excel file
sales_data_excel <- read_excel("../data/healthcare_sales.xlsx", sheet = 1)
head(sales_data_excel)
```

```
# A tibble: 6 x 4
  Product         Sales_Q1 Sales_Q2 Sales_Q3
  <chr>              <dbl>    <dbl>    <dbl>
1 Surgical Masks       150      200      220
2 Hand Sanitizers      120       NA      190
3 N95 Respirators      180      230      250
4 Gloves               100      150      160
5 Ventilators           90      140      150
6 Face Shields         130      180       NA
```

- **Explanation**: The `read_excel()` function imports data from the specified sheet of an Excel file, converting it into a data frame.

2. Cleaning Data in R

Once data is imported, it often requires **cleaning** to ensure accuracy and consistency. This includes: - Handling **missing values**. - Removing **duplicates**. - Correcting **data types**.

Handling Missing Values

Missing data can distort analysis, so it's important to identify and address missing values. The `is.na()` function checks for missing values.

Example: Handling Missing Values in a Data Frame

```
# Checking for missing values in the sales data
sum(is.na(sales_data_excel))
```

```
[1] 4
```

```
# Replacing missing values in the Sales column with the mean
sales_data_excel$Sales_Q2 <- ifelse(is.na(sales_data_excel$Sales_Q2),
                           mean(sales_data_excel$Sales_Q2, na.rm = TRUE),
                           sales_data_excel$Sales_Q2)

# Replacing missing values in the Sales_Q3 column with the mean of Sales_Q2
sales_data_excel$Sales_Q3[is.na(sales_data_excel$Sales_Q3)] <-
  mean(sales_data_excel$Sales_Q3, na.rm = TRUE)

head(sales_data_excel)
```

```
# A tibble: 6 x 4
  Product         Sales_Q1 Sales_Q2 Sales_Q3
  <chr>              <dbl>    <dbl>    <dbl>
1 Surgical Masks       150      200      220
2 Hand Sanitizers      120      179.     190
3 N95 Respirators      180      230      250
4 Gloves               100      150      160
5 Ventilators           90      140      150
6 Face Shields         130      180      199.
```

- **Explanation**: This code checks for missing values in the `Sales` column and replaces any missing values with the column's mean value.

Removing Duplicates

Duplicate rows can inflate your results, and it's important to remove them. You can use the `duplicated()` function to identify and remove duplicate rows.

Example: Removing Duplicates from a Data Frame

```
# Removing duplicate rows from the data frame
sales_data_excel <- sales_data_excel[!duplicated(sales_data_excel), ]
```

- **Explanation**: This code removes any duplicate rows from the `sales_data` data frame.

Correcting Data Types

It's crucial to ensure that each column in the dataset has the correct data type (e.g., numeric, character, factor). You can use the `str()` function to check the data types.

Example: Changing Data Types in a Data Frame

```
# Changing the Product column to a factor
sales_data_excel$Product <- as.factor(sales_data_excel$Product)

# Converting Sales column to numeric (if needed)
sales_data_excel$Sales_Q1 <- as.numeric(sales_data_excel$Sales_Q1)

# Viewing the structure of the data
str(sales_data_excel)
```

```
tibble [16 x 4] (S3: tbl_df/tbl/data.frame)
 $ Product : Factor w/ 16 levels "Disinfectant Wipes",..: 11 4 7 3 15 2 1 8 13 6 ...
 $ Sales_Q1: num [1:16] 150 120 180 100 90 130 110 NA 130 140 ...
 $ Sales_Q2: num [1:16] 200 179 230 150 140 ...
 $ Sales_Q3: num [1:16] 220 190 250 160 150 ...
```

- **Explanation**: This code converts the `Product` column to a factor (for categorical analysis) and ensures the `Sales` column is numeric.

3. Transforming Data in R

Data transformation is often necessary to prepare data for analysis. This includes reshaping, creating new variables, and aggregating data.

Creating New Variables

You can create new variables (columns) in a data frame by performing operations on existing columns.

Example: Calculating Total Sales Across Quarters

```
# Creating a new variable for total sales across all quarters
sales_data_excel$Total_Sales <-
  sales_data_excel$Sales_Q1 + sales_data_excel$Sales_Q2 + sales_data_excel$Sales_Q3
head(sales_data_excel)
```

```
# A tibble: 6 x 5
  Product         Sales_Q1 Sales_Q2 Sales_Q3 Total_Sales
  <fct>              <dbl>    <dbl>    <dbl>       <dbl>
1 Surgical Masks       150      200      220         570
2 Hand Sanitizers      120      179.     190         489.
3 N95 Respirators      180      230      250         660
4 Gloves               100      150      160         410
5 Ventilators           90      140      150         380
6 Face Shields         130      180      199.        509.
```

- **Explanation**: This code creates a new `Total_Sales` column that sums up sales from the first three quarters for each product.

Aggregating Data

Aggregating data helps summarize datasets and identify trends or patterns. The `group_by()` and `summarise()` functions from the `dplyr` package are useful for this.

Example: Aggregating Data by Product

```
# Loading the dplyr package
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
# Aggregating sales data by product
sales_summary <- sales_data_excel %>%
  group_by(Product) %>%
  summarise(Total_Sales = sum(Total_Sales))
sales_summary
```

```
# A tibble: 16 x 2
   Product             Total_Sales
   <fct>                     <dbl>
 1 Disinfectant Wipes         440
 2 Face Shields               509.
 3 Gloves                     410
 4 Hand Sanitizers            489.
 5 Hospital Beds              569.
 6 IV Drip Sets               550
 7 N95 Respirators            660
 8 Oxygen Tanks                NA
 9 Patient Monitors           640
10 Pulse Oximeters            410
11 Surgical Masks             570
12 Syringes                   520
13 Thermometers               490
14 Ultrasound Machines        550
15 Ventilators                380
16 Wheelchairs                390
```

- **Explanation**: This code groups the data by `Product` and calculates the total sales for each product.

Reshaping Data

Reshaping data involves changing its structure, such as converting data from wide to long format using the `pivot_longer()` function from the `tidyr` package.

Example: Reshaping Data from Wide to Long Format

```r
# Loading the tidyr package
library(tidyr)

# Reshaping sales data from wide to long format
```

```r
sales_long <- pivot_longer(sales_data_excel,
                           cols = c(Sales_Q1, Sales_Q2, Sales_Q3),
                           names_to = "Quarter",
                           values_to = "Sales")
head(sales_long)
```

```
# A tibble: 6 x 4
  Product         Total_Sales Quarter  Sales
  <fct>                 <dbl> <chr>    <dbl>
1 Surgical Masks          570  Sales_Q1  150
2 Surgical Masks          570  Sales_Q2  200
3 Surgical Masks          570  Sales_Q3  220
4 Hand Sanitizers         489. Sales_Q1  120
5 Hand Sanitizers         489. Sales_Q2  179.
6 Hand Sanitizers         489. Sales_Q3  190
```

- **Explanation**: This code reshapes the sales data from wide format (with separate columns for each quarter) to long format, where each row represents a product-quarter pair.

Key Takeaways

- You can import data from **CSV** and **Excel** files into R using `read.csv()` and `read_excel()`.
- **Cleaning** data involves handling missing values, removing duplicates, and correcting data types.
- Data **transformation** includes creating new variables, aggregating data, and reshaping datasets for better analysis.

Looking Forward

- In the next lecture, we'll explore how to **export data** from R to various file formats, allowing you to share your analysis and results.