4.1 Preparing Data using Tidyr Package

1. Introduction to `tidyr`

`tidyr` helps to organize and clean data so that it is easy to analyze. It focuses on transforming data into a “tidy” format, where:

- Each variable is a column.

- Each observation is a row.

- Each value is a single cell.

---

2. Installing and Loading `tidyr`

```r
# Install the package if you haven't already
install.packages("tidyr")
```

```
Installing package into '/home/me/R/x86_64-pc-linux-gnu-library/4.4'
(as 'lib' is unspecified)
```

```r
# Load tidyr
library(tidyr)
```

---

3. Common Functions in `tidyr`

Reshaping Data, Long and Wide Formats:

Long format data is common in various real-life scenarios, especially when dealing with time series, repeated measures, or categorical data. Here are a few examples:

Real-Life Examples of Long Format Data:

1. **Healthcare Data:**

   - **Example:** Patient data often comes in long format when tracking vital signs, medication doses, or symptoms over time. Each row might represent a single measurement for a specific patient at a particular time point.
   - **Why:** This format makes it easier to analyze trends, apply statistical models, and visualize changes over time.

Patient_ID	Date	Measurement	Value
001	2024-08-01	BloodPressure	120/80
001	2024-08-02	BloodPressure	130/85
002	2024-08-01	HeartRate	70
002	2024-08-02	HeartRate	72

2. **Survey Data:**
   - **Example:** When analyzing survey responses, each respondent might have multiple answers for different questions. Instead of having separate columns for each question, long format organizes it by question and response.
   - **Why:** Long format is useful for summarizing, visualizing, and performing statistical tests across different questions or groups of respondents.

Respondent	Question	Response
101	Q1	Yes
101	Q2	No
102	Q1	No
102	Q2	Yes

3. **Time Series Data:**
   - **Example:** Financial data often tracks metrics like stock prices, sales, or revenue over time. Each entry in the long format represents a single observation at a specific time point.
   - **Why:** This format is essential for time series analysis, forecasting, and modeling temporal trends.

Date	Company	Metric	Value
2024-08-01	A	Revenue	1000
2024-08-01	B	Revenue	1500
2024-08-02	A	Revenue	1100
2024-08-02	B	Revenue	1600

Why Are R Functions Like `pivot_longer()` and `pivot_wider()` Important?

1. **Data Preparation for Analysis:**
   - Many statistical models, especially those for repeated measures, mixed-effects models, or time series analysis, require data in long format.

- Visualization tools like `ggplot2` in R often prefer data in long format for plotting.

2. **Flexibility in Data Transformation:**
    - Having the ability to switch between wide and long formats allows you to adapt your data structure to the needs of different analyses, making your workflow more efficient.
    - `pivot_longer()` and `pivot_wider()` automate this process, saving time and reducing the potential for manual errors.
3. **Interoperability:**
    - Different tools and libraries might expect data in different formats. By converting between wide and long formats, you can ensure compatibility across tools, whether you're doing machine learning, statistical analysis, or data visualization.

3.1 pivot_longer()

Converts data from wide to long format.

Converts wide data into long data. It's useful when you want to convert several columns into key-value pairs.

```
# Example: Converting sales data from wide to long format

library(tidyr)

# Original wide data frame
wide_data <- data.frame(
  Student = c("Alice", "Bob", "Carol"),
  Math_Score = c(85, 90, 75),
  English_Score = c(78, 88, 82),
  Science_Score = c(92, 85, 80),
  History_Score = c(88, 90, 78),
  Art_Score = c(79, 86, 85),
  Music_Score = c(84, 90, 83),
  PE_Score = c(91, 88, 82)
)

print(wide_data)
```

```
  Student Math_Score English_Score Science_Score History_Score Art_Score
1   Alice         85            78            92            88        79
2     Bob         90            88            85            90        86
3   Carol         75            82            80            78        85
  Music_Score PE_Score
1          84       91
2          90       88
3          83       82
```

```
# Convert to long format
long_data <- pivot_longer(
  wide_data,
  cols = starts_with("Math_Score"):starts_with("PE_Score"),
  names_to = "Course",
  values_to = "Score"
)

# Print long format data
print(long_data)
```

```
# A tibble: 21 × 3
   Student Course          Score
   <chr>   <chr>           <dbl>
 1 Alice   Math_Score         85
 2 Alice   English_Score      78
 3 Alice   Science_Score      92
 4 Alice   History_Score      88
 5 Alice   Art_Score          79
 6 Alice   Music_Score        84
 7 Alice   PE_Score           91
 8 Bob     Math_Score         90
 9 Bob     English_Score      88
10 Bob     Science_Score      85
# ℹ 11 more rows
```

We can also explicitly specify the columns

```
# Convert to long format using specific column names
long_data <- pivot_longer(
  wide_data,
  cols = c(Math_Score, English_Score, Science_Score, History_Score, Art_Score, Music_Scor
  names_to = "Course",
  values_to = "Score"
)

# Print long format data
print(long_data)
```

```
# A tibble: 21 × 3
   Student Course          Score
   <chr>   <chr>           <dbl>
 1 Alice   Math_Score         85
 2 Alice   English_Score      78
 3 Alice   Science_Score      92
 4 Alice   History_Score      88
 5 Alice   Art_Score          79
 6 Alice   Music_Score        84
 7 Alice   PE_Score           91
 8 Bob     Math_Score         90
 9 Bob     English_Score      88
10 Bob     Science_Score      85
# ℹ 11 more rows
```

3.2 pivot_wider():

Converts long data into wide data. It's the inverse of pivot_longer().

```
# Example: Converting long sales data back to wide format

wide_data_again <- pivot_wider(
  long_data,
  names_from = Course,
  values_from = Score
)

# Print wide format data
print(wide_data_again)
```

```
# A tibble: 3 × 8
  Student Math_Score English_Score Science_Score History_Score Art_Score
  <chr>        <dbl>         <dbl>         <dbl>         <dbl>     <dbl>
1 Alice           85            78            92            88        79
2 Bob             90            88            85            90        86
3 Carol           75            82            80            78        85
# i 2 more variables: Music_Score <dbl>, PE_Score <dbl>
```

3.3 separate()

Splits one column into multiple columns.

Example: Split a column containing “Date-Time” into separate “Date” and “Time” columns.

```
# Example dataset
data <- data.frame(
  ID = 1:3,
  DateTime = c("2024-11-01 10:30", "2024-11-02 14:45", "2024-11-03 18:00")
)

# Separate DateTime into Date and Time
separated_data <- data %>%
  separate(DateTime, into = c("Date", "Time"), sep = " ")

print(separated_data)
```

```
  ID       Date  Time
1  1 2024-11-01 10:30
2  2 2024-11-02 14:45
3  3 2024-11-03 18:00
```

Output:

```
  ID       Date  Time
1  1 2024-11-01 10:30
2  2 2024-11-02 14:45
3  3 2024-11-03 18:00
```

3.4 unite()

Combines multiple columns into one column.

Example: Combine “First” and “Last” name columns.

```
# Example dataset
data <- data.frame(
  First = c("John", "Jane", "Jake"),
  Last = c("Doe", "Smith", "Johnson")
)

# Unite First and Last into FullName
united_data <- data %>%
  unite("FullName", First, Last, sep = " ")

print(united_data)
```

```
      FullName
1     John Doe
2   Jane Smith
3 Jake Johnson
```

Output:

```
         FullName
1        John Doe
2      Jane Smith
3 Jake Johnson
```

3.5 drop_na()

Removes rows with missing values.

Example: Drop rows where any value is missing.

```
# Example dataset
data <- data.frame(
  Name = c("Alice", "Bob", NA),
  Age = c(25, 30, 35)
)

# Drop rows with NA
clean_data <- data %>%
  drop_na()

print(clean_data)
```

```
   Name Age
1 Alice  25
2   Bob  30
```

Output:

```
   Name Age
1 Alice  25
2   Bob  30
```

3.6 fill()

Fills missing values with the last non-missing value.

Example: Fill down missing values.

```
# Example dataset
data <- data.frame(
  Group = c("A", NA, NA, "B", NA),
  Value = c(10, 20, 30, 40, 50)
)

# Fill missing Group values
filled_data <- data %>%
  fill(Group, .direction = "down")

print(filled_data)
```

```
  Group Value
1     A    10
2     A    20
3     A    30
4     B    40
5     B    50
```

Output:

```
  Group Value
1     A    10
2     A    20
3     A    30
4     B    40
5     B    50
```

3.7 replace_na()

Replaces missing values with a specified value.

Example: Replace missing values in “Score” with 0.

```r
# Example dataset
data <- data.frame(
  Name = c("Tom", "Jerry", "Spike"),
  Score = c(95, NA, 88)
)

# Replace NA with 0
replaced_data <- data %>%
  replace_na(list(Score = 0))

print(replaced_data)
```

```
   Name Score
1   Tom    95
2 Jerry     0
3 Spike    88
```

Output:

```
   Name Score
1   Tom    95
2 Jerry     0
3 Spike    88
```

4. Combining `tidyr` with dplyr

You can combine `tidyr` with `dplyr` for powerful data manipulation.

Example: Tidy data and calculate summary statistics.

```r
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
# Example dataset
data <- data.frame(
  Name = c("Alice", "Bob", "Alice", "Bob"),
  Year = c(2020, 2020, 2021, 2021),
  Value = c(50, 60, 70, 80)
)

# Pivot longer and calculate total Value per Name
summary <- data %>%
  pivot_longer(cols = Value, names_to = "Metric", values_to = "Score") %>%
  group_by(Name) %>%
  summarise(Total = sum(Score))

print(summary)
```

```
# A tibble: 2 × 2
  Name  Total
  <chr> <dbl>
1 Alice   120
2 Bob     140
```

Output:

```
  Name Total
1 Alice   120
2   Bob   140
```

5. Summary Table of Functions

Function	Purpose
`pivot_longer`	Convert wide data to long format
`pivot_wider`	Convert long data to wide format
`separate`	Split one column into multiple columns
`unite`	Combine multiple columns into one column
`drop_na`	Remove rows with missing values
`fill`	Fill missing values with previous/next one
`replace_na`	Replace missing values with specific value

With these tools, you'll be able to tidy and reshape your data efficiently using `tidyr`. Let me know if you need help with specific examples!