R Cheatsheet

Introduction

R is a programming language used for statistical computing and plotting graphs. It is known for its rich libraries of packages and functions. It allows users to manipulate complex data, build statistical models, and create detailed graphs, making it the go-to tool for data scientists, analysts, and researchers.

Why R?

- Statistical inference
- Data analysis
- Data visualization
- Free to use

Getting Started

Installing R

To install R on your system, visit the CRAN (Comprehensive R Archive Network) website. Follow the instructions for your operating system (Windows, Mac, or Linux).

Basic Syntax

Use single or double quotes to print the text:

R
```r
"Welcome to Educative.io"
```

You can type numbers without using any quotation:

R
```r
200
```

When performing a calculation such as:

```
R
200-150
```

R will automatically print the result (50) to the console. However, if we assign the result to a variable like this:

```
R
result <- 200 - 150
```

R will only perform the calculation and store the result in the `result` variable, but it won't print anything to the console unless explicitly instructed (e.g., using `print(result)`).

Print

We can also use the `print()` function to display output.

```
R
print("Hello Educative!")
```

Type the variable name or use the `print()` function to print variable values.

```
R
y  <- "Course"
print(y)
```

Comments

Use the # symbol for single-line comments; use # at the beginning of each line to create a separate comment for multiline comments.

```
R
# This is a comment
```

Variables

R variables are used to store data values. Use <− or = operators to assign values to variables.

R

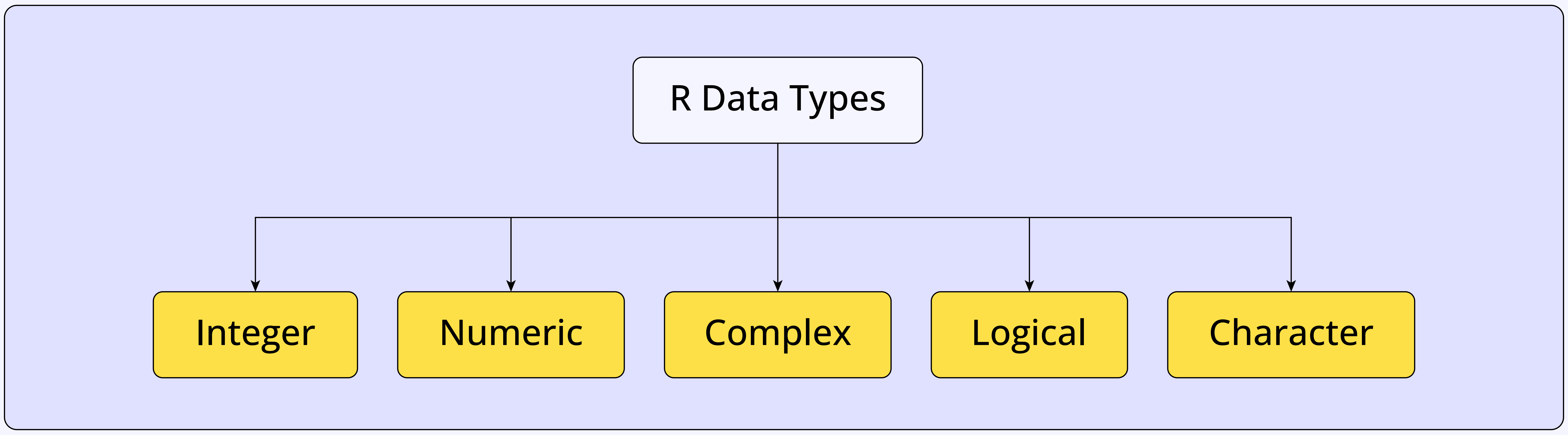
```
x <- 8       # Using <- for assignment
y  = "Course"  # Using = for assignment
```

Variable naming rules:
- It must start with a letter.
- It can include letters, numbers, and underscores (_).
- It cannot contain spaces or special characters (except _).
- It cannot use reserved words as variables.

Variable names are case sensitive!

Basic Data Types

R has five basic data types:

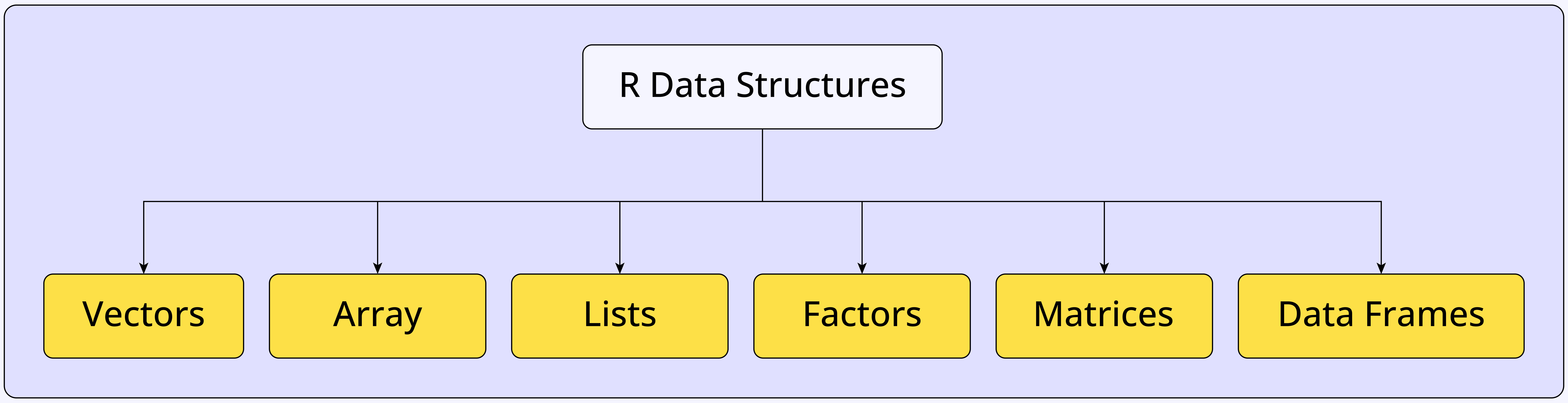


Basic Data Types

- **Integer:** 20L, 50L, and more
- **Numeric:** 1.1, 7, and more
- **Complex:** 10 + 9i
- **Logical:** TRUE and FALSE
- **Character/string:** "Educative", "205", and more

Data Structures

Data structures are essential for organizing and managing data efficiently in R. The commonly used predefined data structures are as follows:



Vectors

Vectors are one-dimensional arrays that hold elements of the same data type. To create a vector, use the `c()` method.

```R
# Creating vector "vec"
vec <- c(1, 2, 3, 4)
# Printing vec value
vec
```

Access individual elements using index positions:

```R
vec[1]
```

R indexes start at 1.

R Vectors Common Functions

`length(vectorName)`	Returns the length of the given vector.
`sum(vectorName)`	Returns the sum of the given vector.

`typeof(vectorName)`	Returns the type of the given vector.
`mean(vectorName)`	Returns the mean of the given vector.

Matrices

Matrices are two-dimensional arrays with rows and columns, holding elements of the same type. Use the following syntax to create matrices:

R

```
# Syntax
matrix(data, nrow, ncol, byrow, names)
```

- `data`: Input vector
- `nrow`: Number of rows to be created
- `ncol`: Number of columns to be created
- `byrow`: If set to `TRUE`, arranges elements row-wise
- `names`: Assign names to the rows and columns

R

```
# Example
matR <- matrix(c(1:12), nrow = 4, byrow = TRUE)
```

Access elements using row and column indexes.

R

```
matR[2,3]
```

Always assign the matrix to a variable and then access the value.

R Matrices Common Functions

`t(matrixname)`	Provides the transpose of the given matrix.
`matrixname1 + matrixname2`	Adds two matrices.
`length(matrixname)`	Returns the length of the given matrix.

`cbind(matrixname1 + matrixname2)`	Combines two matrices by adding a second matrix as a column.
`rbind(matrixname1 + matrixname2)`	Combines two matrices by adding a second matrix as a row.

Data Frames

Data frames are two-dimensional data structures that allow for columns of different types.

Name	Address	Phone Number
Alex	California	2025550167
Brian	New York	2025354137
Charles	Boston	2025339164

Read and write data to/from CSV files.

- Use the `read_csv(file)` method to import a CSV into a data frame.
- Use the `df.to_csv(file)` method to write the data frame to a CSV file.

R

```
df <- read_csv('data.csv') # Read data
df.to_csv('data.csv') # Save Data Frame to CSV
```

Accessing columns and rows by position or name in the data frame.

R

```
df$Score # Access the Score column
df[2, 1] # Accesses element at row 2, column 1
```

Lists

Lists are ordered collections that can hold elements of different types.
Use the `list()` function to create a list:

R

```
newList <- list(1, 1+1i, "a", TRUE)
```

`list()` works similar to the `c()` funtion.

Use `[[ ]]` to access elements of a list:

R

```
print(newList[[2]]) # returns 1+1i
# Modify list
myList[[1]] = 2
```

Arrays

Arrays are n-dimensional and can hold data of the same type.

- **Creation:** Use the `array()` function to create an array. Here, `data` represents elements to be included in the array, and `dim` represents the array's dimensions.

R

```
array(data, dim)

# Example
newVec <- c (1, 2, 3, 4, 5, 6)
newArr <- array (newVec, dim = c(3, 2))

# Expected output
1      4
2      5
3      6
```

- **Accessing elements:** Use square brackets `[]` and specify the index for each array dimension.

R

```
arrayName[2,2]
```

- **Array length:** Use the `length()` function.

R

```
length(arrayName)
```

Factors

Factors represent categorical data with a fixed number of unique values called **levels**.
Use the `factor()` function to create a factor as follows:

```r
fact <- factor(c("C++", "R", "C++", "C"))
fact
```

This will return the following output:

```
[1] C++ R   C++ C
Levels: C C++ R
```

Use `levels()` to print only levels of a factor and indexing to access a specific element.

```r
levels(fact) # returns [1] "C"   "C++" "R"
levels(fact)[1] # returns [1] "C"
```

Data Manipulation

The `dplyr` Package

- `dplyr` is an R package used for data manipulation.
- Provides simple functions to perform operations like filtering, selecting, transforming, and summarizing data

The `dplyr` Package Common Functions

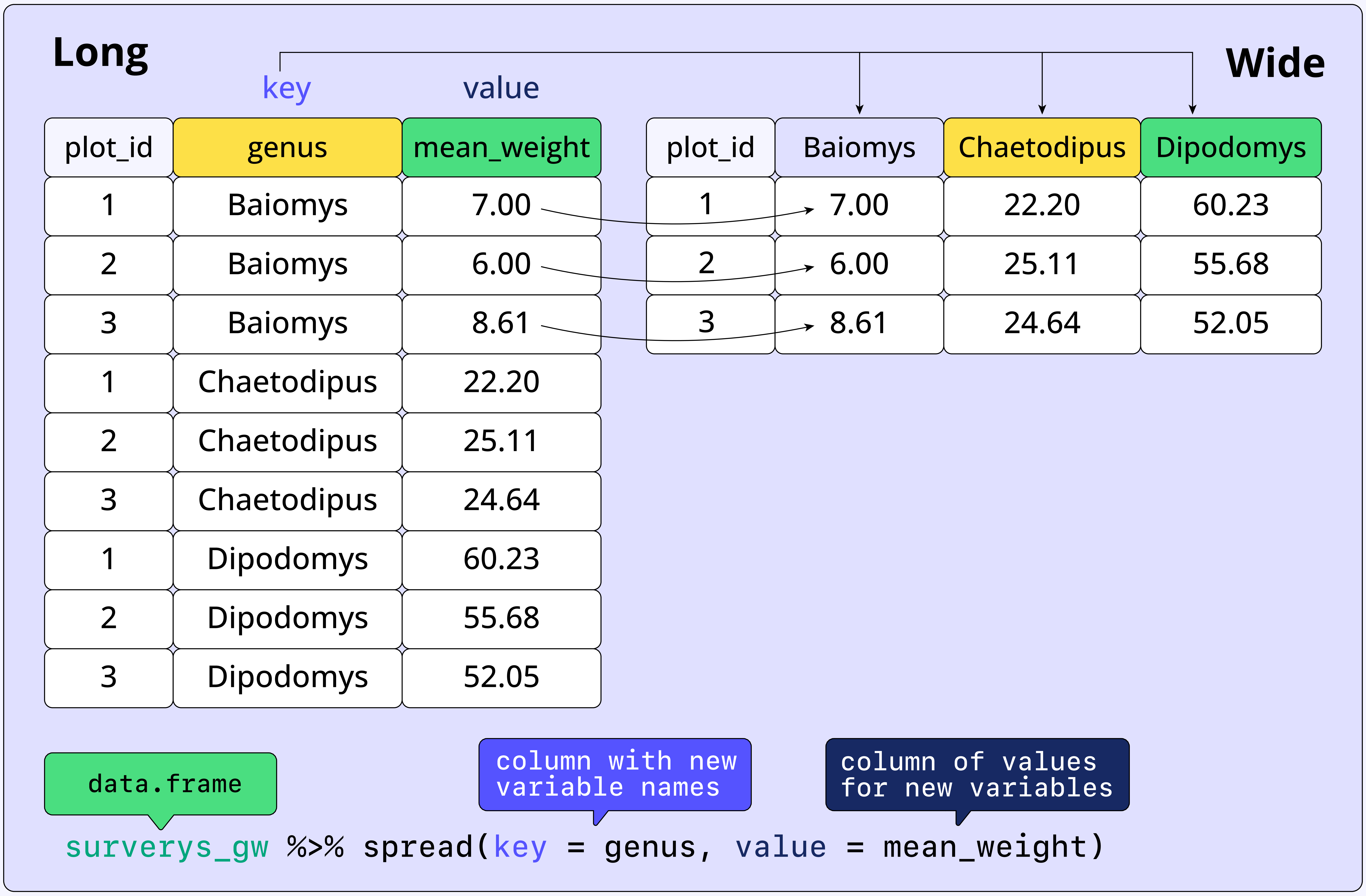
Function name	Description	Example
`filter(data, condition)`	Filters rows in a data frame based on conditions.	`filter(df, score > 20)`
`select(data, column1, column2, …)`	Selects specific columns from a data frame.	`select(df, studentName, score)`

`mutate(data, expr)`	Adds or modifies new variables in a data frame based on `expr`.	`mutate(df, newColumn = score + 10)`
`summarise(data, newCol = summaryFunc(column))`	Summarizes data by applying a function to one or more columns (mean, sum, count).	`summarise(df, avg_score = mean(score))`
`arrange(data, column)`	Sorts rows in a data frame by one or more columns.	`arrange(df, score)`
Piping(`%>%`)	Chains multiple `dplyr` operations.	`df %>% filter(score > 40) %>% select( studentName, score) %>% arrange(score)`

The `tidyr` Package

The `tidyr` package in R helps tidy and reshape data, making it easier to manipulate and analyze.

- `spread(data, key, value)`: Spreads key-value pairs across different columns.

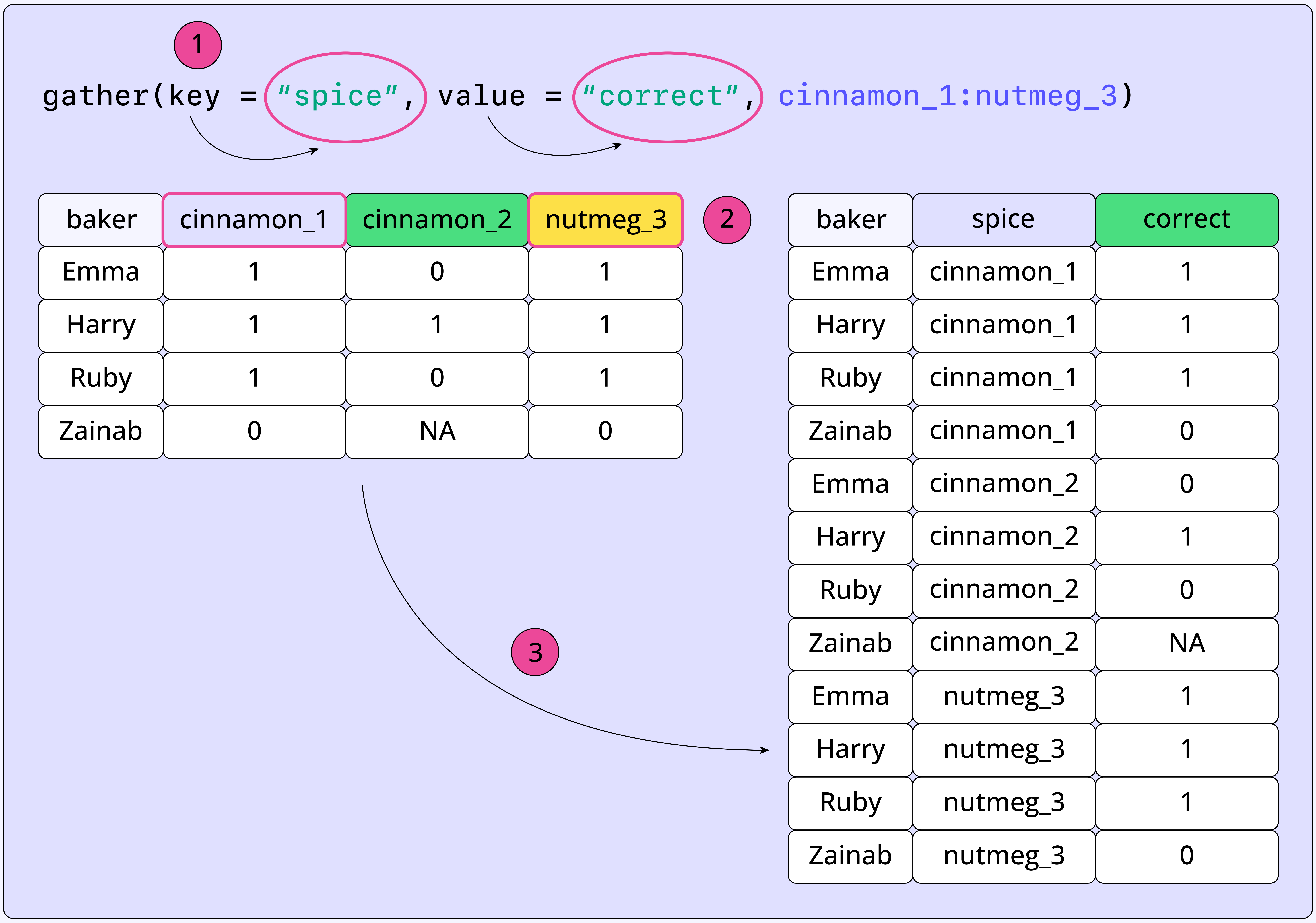

Long
key
value
Wide
data.frame
column with new variable names
column of values for new variables

plot_id	genus	mean_weight
1	Baiomys	7.00
2	Baiomys	6.00
3	Baiomys	8.61
1	Chaetodipus	22.20
2	Chaetodipus	25.11
3	Chaetodipus	24.64
1	Dipodomys	60.23
2	Dipodomys	55.68
3	Dipodomys	52.05

plot_id	Baiomys	Chaetodipus	Dipodomys
1	7.00	22.20	60.23
2	6.00	25.11	55.68
3	8.61	24.64	52.05

```
surverys_gw %>% spread(key = genus, value = mean_weight)
```

- `gather(data, key, value, …, na.rm = FALSE)`: Gathers multiple columns and collapses them into key-value pairs.

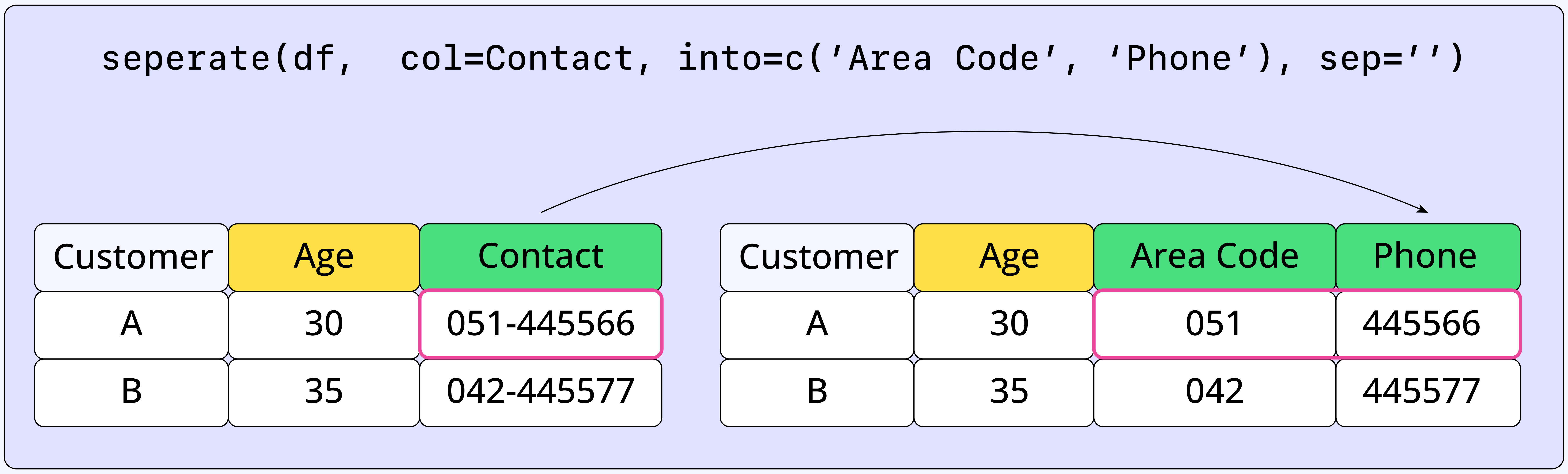


```
gather(key = "spice", value = "correct", cinnamon_1:nutmeg_3)
```

baker	cinnamon_1	cinnamon_2	nutmeg_3
Emma	1	0	1
Harry	1	1	1
Ruby	1	0	1
Zainab	0	NA	0

baker	spice	correct
Emma	cinnamon_1	1
Harry	cinnamon_1	1
Ruby	cinnamon_1	1
Zainab	cinnamon_1	0
Emma	cinnamon_2	0
Harry	cinnamon_2	1
Ruby	cinnamon_2	0
Zainab	cinnamon_2	NA
Emma	nutmeg_3	1
Harry	nutmeg_3	1
Ruby	nutmeg_3	1
Zainab	nutmeg_3	0

- `separate(data, col, into, sep = " ", remove = TRUE)`: Splits a single column into multiple columns based on a separator.

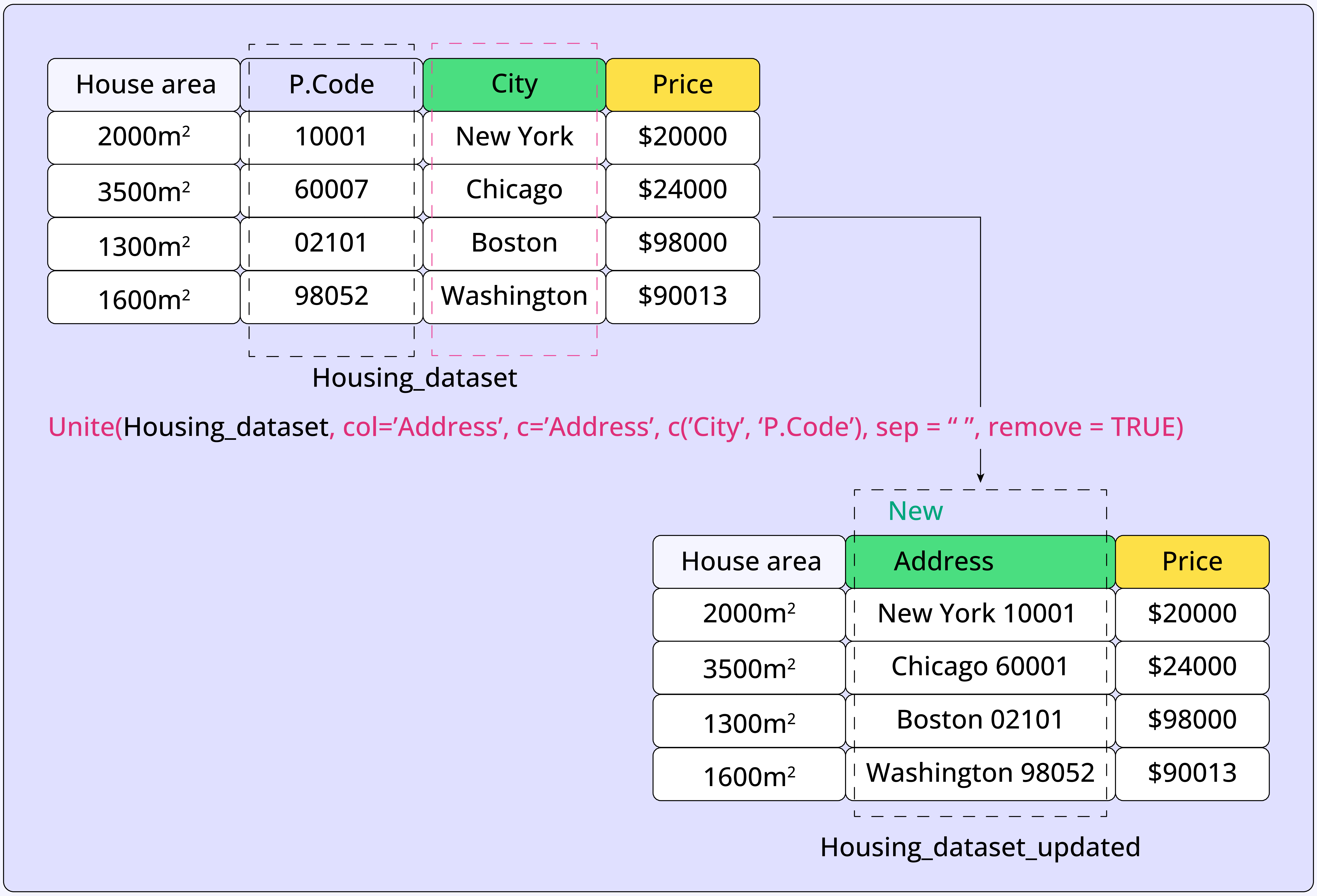


```
seperate(df,  col=Contact, into=c('Area Code', 'Phone'), sep='')
```

Customer	Age	Contact
A	30	051-445566
B	35	042-445577

Customer	Age	Area Code	Phone
A	30	051	445566
B	35	042	445577

- `unite(data, col, ..., sep=",", remove=TRUE)`: Combines multiple columns into a single column.



House area	P.Code	City	Price
2000m²	10001	New York	$20000
3500m²	60007	Chicago	$24000
1300m²	02101	Boston	$98000
1600m²	98052	Washington	$90013

Housing_dataset

Unite(Housing_dataset, col='Address', c='Address', c('City', 'P.Code'), sep = " ", remove = TRUE)

House area	Address	Price
2000m²	New York 10001	$20000
3500m²	Chicago 60001	$24000
1300m²	Boston 02101	$98000
1600m²	Washington 98052	$90013

Housing_dataset_updated

Data Visualization

R provides default functions for visualization.

- Use `plot()` for the creation of various types of charts.
- Use `pie()` for pie charts.
- Use `barplot()` for bar charts.
- Use `hist()` for the histogram.
- Use `boxplot()` for boxplots.

The `ggplot2` Package

The `ggplot2` package is used for data visualization, which provides a systematic approach to building plots.

- **Basic structure:** Specifies the dataset you want to visualize, where `<variable_name>` is the data frame or dataset.
    - The `aes()` function maps data columns to visual elements.
    - `x` and `y` represent the dataset columns for the x and y axes.

```r
ggplot(data = <variable_name>, aes(x = <column1>, y = <column2>))
```

- **Common geoms:**
    - `geom_point()`: Creates scatter plot points.
    - `geom_line()`: Creates lines (e.g., for time series).
    - `geom_bar()`: Creates bar charts.

- **Themes and customization:**
  `theme_…()`: Customizes plot appearance such as background color, legend location, object style, and font size in the chart.

List of common theme functions:
- `theme_light()`: A light theme with a white background and gray grid lines.
- `theme_gray()`: A gray background and white grid lines.
- `theme_bw()`: White background with black grid lines.
- `theme_minimal()`: Minimalistic theme with no gridlines.

Statistical Analysis

Statistics helps analyze data to identify patterns and inform decisions.

Descriptive Statistics

Descriptive statistics summarizes and organizes data using measures of central tendency (mean, median, mode) and dispersion (range, variance, standard deviation).

Common Descriptive Statistics Functions

`mean(data)`	Calculates the mean or average of data.
`media(data)`	Calculates the median or middle value of data.
`sd(data)`	Calculates the standard deviation of data.
`var(data)`	Calculates the variance of data.

Inferential Statistics

Inferential statistics helps predict a population based on a sample of data.

Common Inferential Statistics Tests

Tests	Description
`t.test()`	Performs a t-test for comparing means.
`aov()`	Determines if the means of the dependent variables are affected by the independent variable.
`cor.test()`	Measures the relationship between two variables and indicates the direction and strength of this relationship
`chisq.test()`	Determines statistically significant differences between observed and expected data.
`lm()` (Linear regression)	Examines relationships between a dependent variable and independent variables.

Programming with R

Control Structures

Control structures execute code based on whether a condition is true or false.

`if` **statement:** If the condition is true, execute the if statement's code block.

```
R

if (condition)
{ # code to execute }
```

`else if` **statement:** Checks conditions sequentially and executes the first true condition code block.

```
R

if (condition)
{ # code to execute }
else if (condition)
{ # code to execute }
else
 { # code to execute }
```

for loop: Execute the code a specified number of times.

R

```r
if (x in 1:20)
{ # code to execute }
```

while loop: Executes code until a condition is true.

R

```r
while (condition)
{ # code to execute }
```

Functions

Reusable code block that performs a specific task and only runs when called.

- **Built-in functions**: By default, provided by R
- **User-defined functions**: Custom functions defined by the user

Function Syntax

R

```r
functionName <- function()
{
# code to execute
}
```

Calling Functions

R

```r
functionName();
```

Arguments

R

```r
functionName <- function(argument1, argument2) # function with arguments
{
 # function body
}

result <- functionName(5, 5); # Calling the function with arguments
```

Returning Values

R

```
mulFunc <- function(a, b)
{
return(a * b)
}
result <- sum_function(2, 2)  # result = 4
```

Working with Dates and Times

In R, dates and times can be parsed, extracted, modified, and manipulated using various functions and packages.

The Lubridate Package

Lubridate simplifies date-time handling with parsing, extraction, and manipulation functions.

Common Parsing Date-Time Functions	
`ymd()`	Parses a date in "YYYY-MM-DD" format.
`dmy()`	Parses a date in "DD-MM-YYYY" format.
`mdy()`	Parses a date in "MM-DD-YYYY" format.
`ymd_hms()`	Parses a date-time in "YYYY-MM-DD HH:MM:SS" format.
`mdy_hms()`	Parses a date-time in "MM-DD-YYYY HH:MM:SS" format.

Common Extracting Date-Time Functions	
`year()`	Extracts the year from a date-time object.
`month()`	Extracts the month from a date-time object.
`days_in_month()`	Extracts the number of days in a given month.
`hour()`	Extracts the hour from a date-time object.
`minute()`	Extracts the minute from a date-time object.

Common Modifying Date-Time Functions	
`year(datetime) <- newYearValue`	Modifies the year of a date-time object.
`month(datetime) <- newMonthValue`	Modifies the month of a date-time object.
`day(datetime) <- newDayValue`	Modifies the day of a date-time object.
`hour(datetime) <- newHourValue`	Modifies the hour of a date-time object.
`minute(datetime) <- newMinuteValue`	Modifies the minute of a date-time object.

Some Useful R Functions

Function	Description
`apply(data, margin, functName)`	Applies a function (mean, median, etc.) on each vector or list element.
`str(objectName)`	Displays the structure of an object.
`summary(dataframe)`	Summarizes the statistics of the data.
`subset(dataframe, condition)`	Subsets the data based on a condition.
`sort(vecName)`	Sorts vector or factor.
`log(x)`	Returns the logarithm of a given value.
`str_dup(str, n)`	Prints the string provided `n` times.