Exercises for second session (1)

Part 1: [Warm-up] Foundational Operations & Transformations

Exercise 1: Personal Finance Calculator

This exercise will test your basic skills with numbers, strings, and vectors to calculate and summarize monthly expenses.

Tasks:

1. Create a numeric vector named `expenses` containing the following weekly costs: `85.50, 210.00, 45.25, 120.75`.
2. Calculate the total monthly expense by summing the elements in the `expenses` vector and store it in a variable called `total_expenses`. (Assume a 4-week month).
3. Create a character variable `user_name` with the value "Maria".
4. Create a summary sentence using `paste()` that says: "Maria, your total expenses for the month are: [amount]". Store this in a variable called `summary_message`.
5. Print the `summary_message`.

Exercise 2: Sales Performance Matrix

In this exercise, you will create and manipulate a matrix representing sales data for different products across several regions. This will involve matrix creation, subsetting, and applying calculations to rows and columns.

Setup:

First, create a vector of sales data:
`sales_data <- c(250, 300, 210, 400, 550, 420, 180, 220, 190, 310, 330, 290)`

Tasks:

1. Create a 4×3 matrix named `sales_matrix` from `sales_data`. The matrix should represent 4 products (rows) and 3 regions (columns). Fill the matrix **by row.**
2. Assign meaningful names to the rows ("Product A", "Product B", "Product C", "Product D") and columns ("North", "South", "West"). (Hint: use `rownames()` and

`colnames()`).

3. Calculate the total sales for "Product C" across all regions.
4. Calculate the total sales for the "West" region across all products.
5. Find the product with the highest total sales across all regions. (Hint: `rowSums()` will be useful).
6. Create a new vector called `average_sales_per_product` that contains the average sales for each of the four products.

Exercise 3: Weather Data Cleanup and Transformation

This exercise simulates a real-world task: reading messy data from a CSV file, cleaning it, and performing transformations to make it usable for analysis.

Dataset Preparation:

1. Open a plain text editor.
2. Copy and paste the following text exactly as it is shown below:Extrait de code

```
Date,Temperature_F,WindSpeed_MPH,Event
2023-01-15,45,12mph,Rain
2023-01-16,32,8 mph,Snow
2023-01-17,51,,Sun
2023-01-18,48,15mph,Rain
2023-01-19,55,7mph,
```

3. Save this file as `weather_data.csv` in your R working directory.

Data Analysis Tasks:

1. Load the `weather_data.csv` file into a data frame called `weather_df`. Make sure that empty strings are treated as missing values (`NA`). (Hint: look at the `na.strings` argument in `read.csv()`).
2. The `WindSpeed_MPH` column is a character type because it contains "mph". Create a new column called `wind_speed_clean` by removing the "mph" suffix from the values. (Hint: `gsub()` is a good function for this).
3. Convert the `wind_speed_clean` column from a character type to a numeric type.
4. Create a new column called `temperature_c` that converts the Fahrenheit temperature to Celsius. The formula is: $C = (F - 32) * 5/9$.

5. Calculate the average temperature in Celsius for the days it rained.

Part 2: [NEW] Data Frames - Simple Access and Filters

For this section, we will primarily use the built-in `mtcars` dataset.

Exercise 4: Basic Data Frame Inspection

This exercise focuses on the fundamental commands for exploring a data frame.

Tasks:

1. Load the `mtcars` dataset into your R environment (it's built-in, so you just need to use its name).
2. Display the first 6 rows of the `mtcars` data frame.
3. What are the dimensions (rows and columns) of the dataset?
4. Display the names of all the columns in the dataset.
5. Extract and display only the `mpg` (Miles/(US) gallon) column using the `$` operator.

Exercise 5: Single-Condition Filtering

This exercise introduces the most basic form of subsetting a data frame based on a logical condition.

Tasks:

1. From the `mtcars` dataset, create a new data frame called `six_cyl_cars` that contains only the cars with exactly 6 cylinders (`cyl`).
2. Display the `six_cyl_cars` data frame.
3. How many cars have exactly 8 cylinders? (You don't need to create a new data frame, just find the number).
4. Create a new data frame called `efficient_cars` containing only cars that get more than 25 miles per gallon (`mpg`).

Exercise 6: Multi-Condition Filtering

This exercise requires combining multiple logical conditions to perform more specific filtering.

Tasks:

1. Create a data frame named `powerful_and_efficient` that contains cars from `mtcars` with more than 150 horsepower (`hp`) **AND** more than 14 miles per gallon (`mpg`).
2. How many cars fit these criteria?
3. Create a data frame named `light_or_fast` that contains cars that weigh less than 2 tons (`wt` < 2) **OR** have a quarter-mile time (`qsec`) of less than 16 seconds.
4. From the `powerful_and_efficient` data frame you created, select only the `mpg`, `hp`, and `wt` columns.

Exercise 7: Filtering with a Set of Values

This exercise introduces the `%in%` operator, which is very useful for checking if a value is part of a specific set.

Tasks:

1. Create a data frame named `four_or_six_cyl` that contains all cars from `mtcars` that have either 4 or 6 cylinders. Use the `%in%` operator for this.
2. Create a vector of car names you are interested in: `target_cars ← c("Mazda RX4", "Honda Civic", "Ford Pantera L")`.
3. Create a new data frame `my_favorite_cars` that contains only the rows from `mtcars` corresponding to the names in `target_cars`. (Hint: car names are the row names of `mtcars`).
4. How many cars in the dataset have a number of forward gears (`gear`) that is **NOT** 3 or 5?

Exercise 8: Handling Missing Data

This exercise simulates working with an incomplete dataset, a common challenge in data analysis.

Setup:

First, let's create a slightly "broken" version of the `iris` dataset.

R

`iris_with_na ← iris`
`iris_with_na ← NA`
`iris_with_na ← NA`

```
# Setup: Create the data frame with NA values
iris_with_na ← iris
# Manually insert some NA values for the exercise
iris_with_na[1, 1] ← NA  # Set Sepal.Length of row 1 to NA
iris_with_na[2, 3] ← NA
```

Tasks:

1. Using the `iris_with_na` data frame, find the total number of `NA` (missing) values in the entire dataset. (Hint: `is.na()` and `sum()`).
2. Create a new data frame called `clean_iris` that contains only the rows from `iris_with_na` that have **no missing values at all**. (Hint: `complete.cases()`).
3. Verify that `clean_iris` has no missing values.
4. Calculate the mean of the `Sepal.Length` column in the original `iris_with_na` data frame. You will need to tell the `mean()` function to ignore the `NA` values. (Hint: check the help file for `mean`).

Part 3: Data Frames & Tidyverse Methods

For this section, you must first install and load the Tidyverse library.

`install.packages("tidyverse")`

`library(tidyverse)`

We will use the `starwars` dataset, which is included in `dplyr`.

Exercise 9: (Simple) - Selecting and Filtering Data

This exercise introduces the two most fundamental `dplyr` verbs: `select()` and `filter()`.

Tasks:

1. Load the `starwars` dataset.

2. Create a new tibble (a modern data frame) called `jedi_characters` that contains only the characters from the "Jedi" order. (Hint: this information is not directly available, so filter for characters whose `name` is "Luke Skywalker", "Anakin Skywalker", or "Obi-Wan Kenobi").
3. From the full `starwars` dataset, create a new tibble that includes only the `name`, `height`, and `mass` columns.
4. Combine these two operations: create a tibble that contains the `name`, `homeworld`, and `species` for all droids.

Exercise 10: Arranging (Sorting) Data

This exercise focuses on `arrange()`, the `dplyr` verb for sorting data frames.

Tasks:

1. Arrange the `starwars` dataset by `height` in ascending order. Display the first 10 rows.
2. Arrange the `starwars` dataset by `mass` in descending order. Display the first 10 rows. (Hint: use the `desc()` helper function).
3. Arrange the dataset first by `species` (alphabetically) and then by `birth_year` (from youngest to oldest) within each species.

Exercise 11: Creating New Columns with Mutate

This exercise uses `mutate()` to add new columns based on existing data.

Tasks:

1. The `height` in `starwars` is in centimeters. Create a new column named `height_m` that contains the height in meters.
2. The `mass` is in kilograms. Create a new column named `bmi` (Body Mass Index) calculated as: mass/(height_m)2.
3. Create a new column `name_length` that contains the number of characters in each character's name. (Hint: use the `nchar()` function).

Exercise 12: Summarizing Grouped Data

This exercise introduces the powerful combination of `group_by()` and `summarise()` to calculate summary statistics for different groups.

Tasks:

1. Group the `starwars` dataset by `species`.
2. For each species, calculate the number of characters, the average `height`, and the average `mass`. Name these new columns `character_count`, `avg_height`, and `avg_mass`. Make sure to handle missing values (`NA`) in your calculations.
3. Find the homeworld with the highest average height. (You will need to group by `homeworld`, calculate the average height, and then arrange the results).

Exercise 13: Chaining Operations with the Pipe

This exercise requires you to chain multiple `dplyr` verbs together using the pipe (`%>%`) to create a complete analysis workflow in a single, readable command.

Tasks:

1. Start with the `starwars` dataset.
2. Filter to include only species that have more than one character.
3. Then, group the filtered data by `species`.
4. Next, calculate the average `birth_year` for each species.
5. Finally, arrange the result to show the species with the oldest average `birth_year` first.
   Your final output should be a small tibble with two columns: `species` and the calculated average birth year.

Exercise 14: Conditional Transformations

This exercise uses the `case_when()` function inside `mutate()` to perform more complex, conditional logic—a very common and powerful technique.

Tasks:

1. Create a new column in the `starwars` dataset called `mass_category`. Use `case_when()` to assign values based on the `mass` column:

    - "light" if mass is less than 60.
    - "medium" if mass is between 60 and 100 (inclusive).
    - "heavy" if mass is greater than 100.
    - "unknown" if mass is `NA`.

2. How many characters fall into each `mass_category`? (Hint: use `count()` or `group_by()` and `summarise()`).
3. Create another column named `origin_galaxy`. If a character's `homeworld` is "Tatooine", "Naboo", or "Alderaan", assign "Core Worlds". For all other homeworlds, assign "Outer Rim".
4. Find the average `height` of "heavy" characters from the "Outer Rim".