

## Tutorial III.III - DataFrames in Julia

Applied Optimization with Julia

# DataFrames in Julia

This interactive Julia script is your comprehensive guide to understanding DataFrames in Julia. Work on creating DataFrames, accessing, modifying, and filtering data, alongside basic data manipulation techniques. Follow the instructions, input your code in the designated areas, and verify your implementations with `@assert` statements. Note: Ensure you have the DataFrames package installed to effectively follow this tutorial!

```
# Import the DataFrames package
using DataFrames
```

## Section 1: Creating DataFrames

A DataFrame in Julia is akin to a table in SQL or a spreadsheet - each column can have its own type, making it highly versatile. A DataFrame can be created using the DataFrame constructor and passing key-value pairs where the key is the column name and the value is an array of data. For more help, use `?` in the REPL and type `DataFrame`.

### Exercise 1.1

Create and Test a DataFrame. Create a DataFrame named `employees` with the columns `Name`, `Age`, and `Salary`, and populate it with the specified data: John is 28 years old and earns 50000, Mike is 23 years old and earns 62000. Frank is 37 years old and earns 90000.

```
# Create and Test a DataFrame. Create a DataFrame named 'employees' with the columns
→ 'Name', 'Age', and 'Salary', and populate it with the specified data: John is 28
→ years old and earns 50000, Mike is 23 years old and earns 62000. Frank is 37 years
→ old and earns 90000.
# YOUR CODE BELOW

# Test your answer
@assert employees == DataFrame(Name = ["John", "Mike", "Frank"],
    Age = [28, 23, 37], Salary = [50000, 62000, 90000])
println("DataFrame created successfully!")
println(employees)
```

#### Tip

Remember, for more help, use `?` in the REPL and type `DataFrame`.

## Section 2: Accessing and Modifying Data

Accessing columns in a DataFrame can be done using the dot syntax, while rows can be accessed via indexing. Modification of data is straightforward; just assign a new value to the desired cell. To access the column 'name' in our DataFrame with `employees`, we could do:

```
employees.Name
```

To access the third name specifically, we could do:

```
employees.Name[3]
```

### Exercise 2.1

Access the `Age` column from the DataFrame and save it in a new variable `ages`.

```
# Access the 'Age' column from the DataFrame and save it in a new variable 'ages'.
# YOUR CODE BELOW

# Test your answer
@assert ages == [28, 23, 37]
println("Correct, the Ages column is: ", ages)
```

### Exercise 2.2

Update John's salary to 59000.

```
# Update John's salary to 59000.
# YOUR CODE BELOW

# Test your answer
@assert employees.Salary[1] == 59000
println("Modified DataFrame: ")
println(employees)
```

## Section 3: Filtering Data

Logical indexing can be used to filter rows in a DataFrame based on conditions. To filter the DataFrame to include only employees names "Frank" we could do:

```
allFranks = employees[employees.Name .== "Frank", :]
```

Alternatively, the `filter` function provides a powerful tool to extract subsets of data based on a condition:

```
allFranks = filter(row -> row.Name == "Frank", employees)
```

### Exercise 3.1:

Filter the DataFrame to include only employees with salaries above 60000. Save the resulting employees in the DataFrame `high_earners`.

```
# Filter the DataFrame to include only employees with salaries above 60000. Save
# the resulting employees in the DataFrame 'high_earners'.
# YOUR CODE BELOW

# Test your answer
@assert nrow(high_earners) == 2
println("High earners: ")
println(high_earners)
```

## Section 4: Basic Data Manipulation

Julia provides functions for basic data manipulation tasks, including sorting, grouping, and joining DataFrames. The `sort` function can be used to order the rows in a DataFrame based on the values in one or more columns. To see how to use the function, type `?` into the REPL (terminal) and type `sort`.

### Exercise 4.1

Sort the DataFrame based on the `Age` column and save it as `sorted_df`.

```
# Sort the DataFrame based on the 'Age' column and save it as 'sorted_df'.
# YOUR CODE BELOW

# Test your answer
@assert sorted_df.Age[1] == 23
println("DataFrame sorted by age: ")
println(sorted_df)
```

#### Tip

If you have more complicated data structures, take a look at JSON files which can be used to work with all kind of differently structured data sets.

## Conclusion

Fantastic work! You've completed the tutorial on DataFrames in Julia. You've seen how to create DataFrames and access, modify and filter data. Continue to the next file to learn more.