Module (1)

Introduction to R Programming

What is Programming?

- **Programming**: Method of conveying to a computer how to perform tasks through specific instructions.
- **Example**: To print "hello world" on a computer screen, you would use a print command like `print("hello world")` in R.

Why Learn Coding?

- **Problem Solving**: Enhances logical skills and efficiency in solving problems.
- **Job Market**: Maximizes opportunities, as many sectors require efficient programmers.
- **Creativity and Fun**: Coding allows for creative solutions and can be enjoyable.

Natural Languages vs Programming Languages

- **Natural Languages**: Ambiguous, large vocabularies (e.g., English, German).
- **Programming Languages**: Explicit, structured, and interpretable by computers. Essential for converting data into digital formats that computers can understand and process.

History of Programming

- **1938**: Modern programmable binary calculator Z1 by Konrad Zuse.
- **1945**: First operational modern computer.
- **1950**: First computer with stored programs.
- **1954**: Fortran, the first high-level programming language, developed.
- **Various Languages**: C, C++, Python, each suitable for different tasks.

R Programming Language

- **Origin**: Derived from the S programming language by John Chambers (1976). Developed by Ross Ihaka and Robert Gentleman at the University of Auckland (1991). Officially released as R 1.0.0 in 2000.
- **Purpose**: Statistical computing and graphics, essential for data science.
- **Open Source**: Released under the GNU public license, allowing freedom of use, modification, distribution, and redistribution.

Importance of R

- **Popularity**: Among the top programming languages for data science jobs (third behind Java and

Python in 2016 according to Fossbytes).

- **Modularity**: Base R installation provides a small set of recommended packages. Additional packages can be installed from CRAN (Comprehensive R Archive Network), Bioconductor, and GitHub.
- **Data Analysis and Statistical Modeling**: Extensive functions and methods, from simple linear regression to complex techniques.
- **Graphical Capabilities**: Visualization of georeferenced data, 3D plots, text mining applications, and more through specialized packages.

Examples of R Capabilities

- **Generating Random Numbers**: Using the `runif()` function to generate random numbers from a uniform distribution.

```
random <- runif(5)
print(random)
```

- **Summary Statistics**: Using the `summary()` function on datasets (e.g., cars dataset).

```
data(cars)
summary(cars)
```

- **Georeferenced Data Visualization**: Packages like `raster`, `sp`, and `colorspace` for plotting maps and spatial data.
- **Text Mining**: Packages like `tm`, `wordcloud`, and `colorspace` for classifying text and analyzing language evolution.

Communication Methods in R

- **Databases and File Types**: R can connect to various databases and handle different file types, from image files to MySQL databases.
- **Wide Range of Functions**: R's versatility makes it suitable for diverse projects and problem-solving in data science.

Conclusion

- **Overview**: Introduction to programming and the importance of learning R.

- **Next Steps**: Upcoming videos will cover practical usage of R and its packages for solving data science problems.

This lecture provides a foundational understanding of what programming is, why learning to code is beneficial, and introduces the R programming language, highlighting its history, importance, and practical applications in data science.

Installation of R and RStudio

Overview

- This video provides step-by-step instructions on how to install R and RStudio on your machine. RStudio is an Integrated Development Environment (IDE) that makes working with R more convenient.

Installation of R

- **Distribution Link**: R software can be downloaded from the Comprehensive R Archive Network (CRAN) link.

For Windows:

1. **Download R**: Get the latest base R version in `.exe` format from CRAN.
2. **Install R**: Follow the installation instructions to install R on your system.
3. **Install Rtools**: Download and install Rtools for additional functionalities and package building.

For Mac OS X:

1. **Download R**: Get the latest base R version in `.pkg` format from CRAN.
2. **Install R**: Follow the installation instructions.
3. **Install Compilers**: Download and install `clang` and `gfortran` compilers as required by CRAN.

For Linux:

1. **Download R**: Use the package manager to install the latest base R version. The package typically consists of `r-base-core` and `r-base-devel`.
2. **Follow CRAN Instructions**: Depending on your Linux distribution, follow the specific installation instructions provided on CRAN.

RStudio Installation

1. **Download RStudio**: Visit the RStudio website link and download the free RStudio Desktop version compatible with your operating system.

2. **Install RStudio**: Follow the instructions to install RStudio after installing R and Rtools (or compilers for Mac).
3. **Start RStudio**:
    - On Windows: Start via the desktop icon or programs menu.
    - On Mac/Linux: Start via the installed program link.

RStudio Interface

- **Starting RStudio**: Upon starting, you will see the RStudio interface.
- **Features**:
    - **Toolbars and Windows**: Multiple toolbars and windows for various functionalities.
    - **Script and Console**: Write code in the script editor and run it in the console.
    - **Variables Window**: View and manage variables.
- **Customization**:
    - **Appearance**: Change the editor theme by navigating to `Tools > Global Options > Appearance > Editor theme`. This can help reduce eye strain.
    - **Auto-Completion**: Enable or disable auto-completion from `Tools > Global Options > Code > Insert matching parenthesis/quotes`.

Conclusion

- The installation process for both R and RStudio is straightforward and essential for working effectively in this course. More features and functionalities of RStudio will be explored in upcoming videos.

First Steps in RStudio

Introduction to RStudio

- **Overview**: RStudio is an Integrated Development Environment (IDE) for R programming. It features a user-friendly interface with multiple sections to facilitate coding and data analysis.
- **Interface Sections**:
    - **Code Editor/Script Section**: Write and run R scripts.
    - **Interactive R Console**: Execute commands interactively.
    - **Environment/History Section**: View variables and command history.
    - **Files/Plots/Packages/Help Section**: Manage files, visualize plots, manage packages, and access help resources.

Basic Commands and Calculations

- **Simple Calculations**: R can be used as a calculator.
  - Example: `5 * 2` yields `10`.
  - **Variables**: Store results in variables for reuse.
    - Example: `b = 5^2` stores `25` in `b`.
- **Sequences**:
  - Create sequences with specified steps: `seq(1, 10, by = 2)` results in `1, 3, 5, 7, 9`.
  - Element-wise operations: `c^2` squares each element of `c`.

Visualization

- **Basic Plots**:
  - Use `plot(variable)` to create plots.
  - Example: `plot(d)` where `d` is a sequence of squared numbers.

Script Management

- **Creating and Running Scripts**:
  - Create a new R Script: Use the green plus icon or `Ctrl + Shift + N`.
  - Save the script with a `.R` extension.
  - Run commands line by line: `Ctrl + Enter` or `Cmd + Return` for Mac.
  - Run the entire script using `source("path/to/script.R")`.

Managing Working Directory

- **Get Current Directory**: `getwd()`.
- **Set Working Directory**: `setwd("path/to/directory")`.
- **Tools**: Set working directory through the toolbar or using console commands.

Multi-line Commands

- **Writing Multi-line Commands**: Use `Shift + Enter` to break commands into multiple lines for better readability.

Working with Environment and Console

- **Environment**: View variables and their values.
- **Console**: Clear console with the clear console button. Clear environment using the clear objects button.
- clear console: ctrl + L

Practical Tips

- **Saving and Executing Code**: Regularly save your script to prevent data loss.
- **Comments**: Use comments (`#`) to annotate code for clarity.
- **Help and Packages**: Access help and manage packages through the corresponding sections in RStudio.

Conclusion

- **Further Learning**: More features and advanced usage of RStudio will be explored in upcoming videos and weeks.

This overview should help you get started with RStudio and understand its basic functionalities. Keep practicing to become more familiar with the environment and its features.

Conclusion

Summary of Week 1

- **Introduction to R for Data Science**:
  - **Importance of R**: Understanding the role and significance of R programming in data science.
  - **Brief History and Features**: Learning about the development and unique features of R.
- **Installing R and RStudio**:
  - **Installation Guide**: Steps to install R and the RStudio IDE on your machine.
- **Exploring RStudio**:
  - **Key Features**:
    - Code Editor and Script Section
    - Interactive R Console
    - Environment and History Section
    - Files, Plots, Packages, and Help Section
- **Basic R Commands and Operations**:
  - **Arithmetic Operations**: Using R as a basic calculator.
  - **Variable Assignment**: Storing and manipulating data in variables.
  - **Sequences and Calculations**: Generating sequences and performing element-wise operations.
- **Visualization**:
  - **Basic Plotting**: Creating simple plots using the `plot` function.
- **Working with Scripts**:
  - **Creating and Saving R Scripts**: Writing, saving, and running R scripts.

    - **Managing Working Directory**: Setting and getting the working directory in RStudio.
- **Practical Tips and Multi-line Commands**:
    - **Efficient Coding**: Writing multi-line commands and managing scripts efficiently.
    - **Console and Environment Management**: Clearing the console and environment, managing variables and commands.

Looking Ahead

- **Upcoming Topics**:
    - Further exploration of R programming features.
    - Introduction to more tools and functionalities in RStudio.
    - Learning about various packages and libraries available in R.

Final Note

- **Encouragement**: This week provided a foundational understanding of R and RStudio. Continue learning and practicing to build on this knowledge in the upcoming modules.

Introduction to the Lab Environment

- **Navigating the Lab**:
    - Access the lab through the Coursera portal.
    - Look for the "Ungraded Lab" section.

Mentioned Reading Material

https://bookdown.org/daniel_dauber_io/r4np_book/the-rstudio-interface.html
https://www.educative.io/blog/r-tutorial-beginners-guide
https://www.analytixlabs.co.in/blog/r-programming-language-basics

Steps to Access the Lab

1. **Opening the Lab**:
    - Click on the "Open Lab" button to start the lab session.
    - A new page will load, preparing your lab environment.
2. **Waiting for Setup**:
    - The setup process may take a few moments.

Using the RStudio Interface in Coursera

- **Familiar Interface**:
  - The in-built RStudio platform resembles the standalone version.
  - Key sections include the console, environment, file browser, plots, and more.
- **Basic Operations in RStudio**:
  - **Clearing the Console**: Demonstrating how to clear the console.
  - **Running Commands**: Example of simple print commands like `print(4)` and `print("hello")`.
  - **Variable Assignment**: Storing values in variables and observing them in the environment section.
  - **Creating and Saving Scripts**: Writing, saving, and running R scripts (e.g., `test.R`).
  - **Checking Working Directory**: Using commands like `getwd()` to determine the current directory.

Practical Example

- **Sample Problem**:
  - **Task**: Calculate and print the square of the sum of 3 and 5.
  - **Implementation**:

```
# Problem: Calculate and print the square of the sum of 3 and 5
result <- (3 + 5)^2
print(result)
# Output: 64
```

  - **Comments**: Emphasize the importance of commenting for clarity and documentation.

Tips for Lab Work

- **Instructions and Task Details**:
  - Detailed instructions for tasks will be provided within the lab.
  - Verify that your code meets the specified requirements and produces the desired results.
- **Debugging and Testing**:
  - Use the console for quick checks and debugging.
  - Ensure the code in your script functions as expected.

Conclusion

- **Getting Comfortable**:

- Spend time exploring and familiarizing yourself with the platform during Lab-1.
- Practice and experiment to become proficient with the RStudio environment in Coursera.

All the best and happy coding!

Exercises covered in LAB

Familiarizing Yourself with RStudio Interface

Before you dive into the exercises, get comfortable with the RStudio interface:

- **Console**: Where you can directly interact with R. You can type commands and see immediate results.
- **Script Editor**: Where you can write, save, and run your R scripts.
- **Environment/History**: This panel shows the objects in your current R session and keeps a history of your commands.
- **Plots/Packages/Help**: Useful for data visualization, managing R packages, and accessing documentation.

Exercise 1: Basic Arithmetic Operations

1. Open a new script file by going to `File` > `New File` > `R Script`.
2. Type the following lines into the script editor:

```
# Addition
2 + 3
# Subtraction
5 - 2
# Multiplication
4 * 6
# Division
10 / 2
```

3. Place the cursor on each line and press `Ctrl + Enter` to run the line and observe the results in the console.

Exercise 2: Variables and Assignments

1. Continue in the same script or open a new one.
2. Declare variables and perform operations:

```
# Declare variables
x <- 5
y <- 3

# Perform operations
sum <- x + y
difference <- x - y

# View the results
sum
difference
```

3. Run the code by placing the cursor on each line and pressing `Ctrl + Enter`.
4. Modify the values of `x` and `y` and rerun the code to see how the output changes.

Exercise 3: Data Types

1. In the same script or a new one, explore different data types:

```
# Numeric
num_var <- 10

# Character
char_var <- "Hello, R!"

# Logical
bool_var <- TRUE

# View the data types
class(num_var)
class(char_var)
class(bool_var)
```

2. Run each line and observe the output. The `class()` function will show you the type of each variable.
3. Experiment by assigning different values to these variables and checking their data types again.

Exercise 4: Simple Plot

1. Generate a simple scatter plot by adding the following code to your script:

```
# Generate data
x <- 1:10
```

```
y <- x^2

# Plot
plot(x, y, main = "Simple Scatter Plot", xlab = "X", ylab = "Y")
```

2. Run the code and observe the plot in the `Plots` tab.
3. Experiment with modifying the data or adding additional customization to the plot. For example:

```
plot(x, y, main = "Customized Scatter Plot", xlab = "X Axis", ylab = "Y Axis", col = "blue", pch = 19)
```

   - `col` sets the color of the points.
   - `pch` sets the type of the plot points.

Use `?plot` in the console to access the help documentation for more customization options.

Notes

- **Saving Work**: Save your script frequently by going to `File` > `Save` or pressing `Ctrl + S`.
- **Running Code**: `Ctrl + Enter` runs the current line or selected lines in the script editor.
- **Modifications**: Feel free to change the values, try different operations, and customize your plots to better understand how R works.
  By following these steps, you should become more comfortable with basic R operations, variables, data types, and plotting. Good luck with your lab!