Computational Bootcamp 2: Intro to R

Ankushi Mitra

Department of Government
Georgetown University

August 15, 2023

What We'll Be Covering Overall

1. Software installation, file management
2. Basics of R: data structures, writing code, creating objects, packages
3. R: working with datasets
4. More R: data cleaning, visualization
5. LaTex: producing documents with Markdown and Overleaf

What We'll Be Covering Today

1. Basic Steps of Data Analysis

What We'll Be Covering Today

1. Basic Steps of Data Analysis
2. The R Studio Interface

What We'll Be Covering Today

1. Basic Steps of Data Analysis
2. The R Studio Interface
3. Data Structures in R

What We'll Be Covering Today

1. Basic Steps of Data Analysis
2. The R Studio Interface
3. Data Structures in R
4. Working with Objects in R

What We'll Be Covering Today

1. Basic Steps of Data Analysis
2. The R Studio Interface
3. Data Structures in R
4. Working with Objects in R
5. Packages in R

Basic Steps of Data Analysis

- Specify research question

Basic Steps of Data Analysis

- Specify research question
- Develop research design

Basic Steps of Data Analysis

- Specify research question
- Develop research design
- Collect data

Basic Steps of Data Analysis

- Specify research question
- Develop research design
- Collect data
- Preprocess data

Basic Steps of Data Analysis

- Specify research question
- Develop research design
- Collect data
- Preprocess data
  - The data you have can contain errors, be incomplete, may need additional variable construction, etc.

Basic Steps of Data Analysis

- Specify research question
- Develop research design
- Collect data
- Preprocess data
  - The data you have can contain errors, be incomplete, may need additional variable construction, etc.
  - Clean the dataset so that it can be used for analysis

Basic Steps of Data Analysis

- Specify research question
- Develop research design
- Collect data
- Preprocess data
  - The data you have can contain errors, be incomplete, may need additional variable construction, etc.
  - Clean the dataset so that it can be used for analysis
  - Summarizing/Visualizing data

Basic Steps of Data Analysis

- Specify research question
- Develop research design
- Collect data
- Preprocess data
  - The data you have can contain errors, be incomplete, may need additional variable construction, etc.
  - Clean the dataset so that it can be used for analysis
  - Summarizing/Visualizing data
  - Statistical analysis

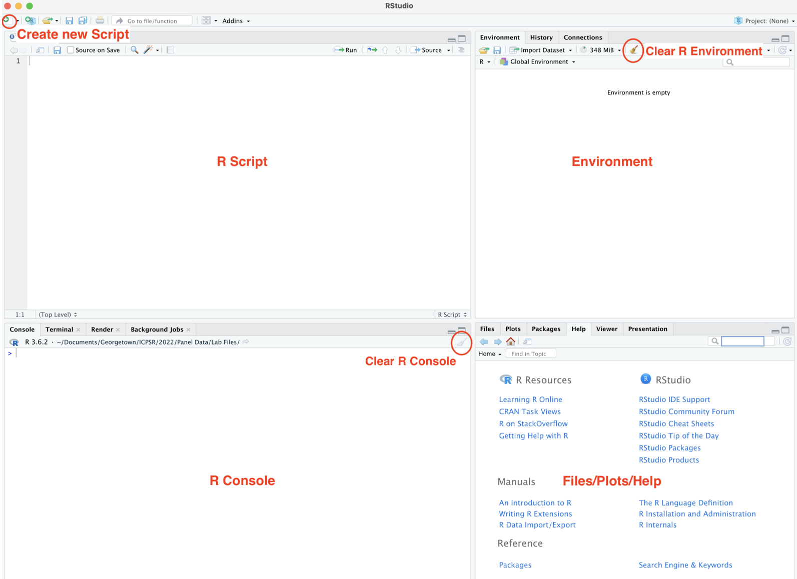
Basic Steps of Data Analysis

- Specify research question
- Develop research design
- Collect data
- Preprocess data
  - The data you have can contain errors, be incomplete, may need additional variable construction, etc.
  - Clean the dataset so that it can be used for analysis
  - Summarizing/Visualizing data
  - Statistical analysis
  - Summarizing/Visualizing results

Basic Steps of Data Analysis

- Specify research question
- Develop research design
- Collect data
- Preprocess data
  - The data you have can contain errors, be incomplete, may need additional variable construction, etc.
  - Clean the dataset so that it can be used for analysis
  - Summarizing/Visualizing data
  - Statistical analysis
  - Summarizing/Visualizing results

R Studio Interface



Create new Script

Clear R Environment

R Script

Environment

Clear R Console

R Console

Files/Plots/Help

Environment is empty

R 3.6.2 · ~/Documents/Georgetown/ICPSR/2022/Panel Data/Lab Files/

R Studio Interface

- R Script
  - Write your code here

R Studio Interface

- R Script
  - Write your code here
  - Each line is a separate command

R Studio Interface

- R Script
  - Write your code here
  - Each line is a separate command
  - "Run" a line by highlighting it and clicking "Run" or pressing Cmd+Return (Mac) or Ctrl+Return (Windows)

R Studio Interface

- R Script
  - Write your code here
  - Each line is a separate command
  - "Run" a line by highlighting it and clicking "Run" or pressing Cmd+Return (Mac) or Ctrl+Return (Windows)
  - Code should be written so humans can easily understand what's happening. Annotate/Comment your code by prefacing a line with #

R Studio Interface

- R Script
  - Write your code here
  - Each line is a separate command
  - "Run" a line by highlighting it and clicking "Run" or pressing Cmd+Return (Mac) or Ctrl+Return (Windows)
  - Code should be written so humans can easily understand what's happening. Annotate/Comment your code by prefacing a line with #
  - The script can be saved as a .R file

R Studio Interface

- R Script
  - Write your code here
  - Each line is a separate command
  - "Run" a line by highlighting it and clicking "Run" or pressing Cmd+Return (Mac) or Ctrl+Return (Windows)
  - Code should be written so humans can easily understand what's happening. Annotate/Comment your code by prefacing a line with #
  - The script can be saved as a .R file
- Console
  - Where results are displayed

R Studio Interface

- R Script
  - Write your code here
  - Each line is a separate command
  - "Run" a line by highlighting it and clicking "Run" or pressing Cmd+Return (Mac) or Ctrl+Return (Windows)
  - Code should be written so humans can easily understand what's happening. Annotate/Comment your code by prefacing a line with #
  - The script can be saved as a .R file
- Console
  - Where results are displayed
  - You can type code directly into the console **but this is bad practice since the Console input cannot be saved**

R Studio Interface

- R Script
  - Write your code here
  - Each line is a separate command
  - "Run" a line by highlighting it and clicking "Run" or pressing Cmd+Return (Mac) or Ctrl+Return (Windows)
  - Code should be written so humans can easily understand what's happening. Annotate/Comment your code by prefacing a line with #
  - The script can be saved as a .R file
- Console
  - Where results are displayed
  - You can type code directly into the console **but this is bad practice since the Console input cannot be saved**
- Environment
  - Where objects are stored

R Studio Interface

- R Script
  - Write your code here
  - Each line is a separate command
  - "Run" a line by highlighting it and clicking "Run" or pressing Cmd+Return (Mac) or Ctrl+Return (Windows)
  - Code should be written so humans can easily understand what's happening. Annotate/Comment your code by prefacing a line with #
  - The script can be saved as a .R file
- Console
  - Where results are displayed
  - You can type code directly into the console **but this is bad practice since the Console input cannot be saved**
- Environment
  - Where objects are stored

Object Class/Type/Mode

- character: non-numeric (text)

Object Class/Type/Mode

- character: non-numeric (text)
- numeric: numbers

Object Class/Type/Mode

- character: non-numeric (text)
- numeric: numbers
  - Includes "integer" and "double" — concerns level of precision; R works this out behind the scenes

Object Class/Type/Mode

- character: non-numeric (text)
- numeric: numbers
    - Includes "integer" and "double" — concerns level of precision; R works this out behind the scenes
- logical: TRUE and FALSE

Object Class/Type/Mode

- character: non-numeric (text)
- numeric: numbers
  - Includes "integer" and "double" — concerns level of precision; R works this out behind the scenes
- logical: TRUE and FALSE
- factor: a numeric which R has categorized into "levels"

Object Class/Type/Mode

- character: non-numeric (text)
- numeric: numbers
  - Includes "integer" and "double" — concerns level of precision; R works this out behind the scenes
- logical: TRUE and FALSE
- factor: a numeric which R has categorized into "levels"

Data Structures

- Vectors
  - Data analysis is not possible without data structures for data. R has several important data structures that shape how information is stored and processed.

Data Structures

- Vectors
    - Data analysis is not possible without data structures for data. R has several important data structures that shape how information is stored and processed.
    - *Vectors* hold a sequence of values of the same data type, like numbers, characters, or logical values. They are one-dimensional arrays or lists of values that contain one and only one type of data. Vector types in R include: logical, integer, double, character, complex, raw.

Data Structures

- Vectors
  - Data analysis is not possible without data structures for data. R has several important data structures that shape how information is stored and processed.
  - *Vectors* hold a sequence of values of the same data type, like numbers, characters, or logical values. They are one-dimensional arrays or lists of values that contain one and only one type of data. Vector types in R include: logical, integer, double, character, complex, raw.
- Matrices
  - *Matrices* are two-dimensional data structures in R that consists of rows and columns of values. Every element is of the same type. It's essentially a collection of vectors of the same length, organized in rows and columns.

Data Structures

- Vectors
  - Data analysis is not possible without data structures for data. R has several important data structures that shape how information is stored and processed.
  - *Vectors* hold a sequence of values of the same data type, like numbers, characters, or logical values. They are one-dimensional arrays or lists of values that contain one and only one type of data. Vector types in R include: logical, integer, double, character, complex, raw.
- Matrices
  - *Matrices* are two-dimensional data structures in R that consists of rows and columns of values. Every element is of the same type. It's essentially a collection of vectors of the same length, organized in rows and columns.
  - Most data contains at least numeric information and character information. That is why we don't usually use matrices.

Data Structures

- Data Frames
  - *Data Frames* and *Tibbles* are what you will usually use. They are a tabular data structure in R, similar to a spreadsheet or a database table. They can store different types of data (e.g., numeric, character) in columns. Each column is a vector, and all columns must have the same length.

Data Structures

- Data Frames
  - *Data Frames* and *Tibbles* are what you will usually use. They are a tabular data structure in R, similar to a spreadsheet or a database table. They can store different types of data (e.g., numeric, character) in columns. Each column is a vector, and all columns must have the same length.
  - Most times, each column will be of one type while a given row will contain many different types. We usually refer to columns as *variables* and rows as *observations*.

Data Structures

- Data Frames
  - *Data Frames* and *Tibbles* are what you will usually use. They are a tabular data structure in R, similar to a spreadsheet or a database table. They can store different types of data (e.g., numeric, character) in columns. Each column is a vector, and all columns must have the same length.
  - Most times, each column will be of one type while a given row will contain many different types. We usually refer to columns as *variables* and rows as *observations*.
- Example
  - Inspect R's built-in dataset, *mtcars*

Objects and Assignment

- In R, to work with values/data, we *assign* them to *objects*

Objects and Assignment

- In R, to work with values/data, we *assign* them to *objects*
  - Object: named "box" or "container" to store values/data. Think of it as a box where you can store numbers, text, or other types of information. You can give the box a name, and then you can use that name to access and manipulate the data inside.

Objects and Assignment

- In R, to work with values/data, we *assign* them to *objects*
    - Object: named "box" or "container" to store values/data. Think of it as a box where you can store numbers, text, or other types of information. You can give the box a name, and then you can use that name to access and manipulate the data inside.
    - Using the assignment operator: <-

Objects and Assignment

- In R, to work with values/data, we *assign* them to *objects*
  - Object: named "box" or "container" to store values/data. Think of it as a box where you can store numbers, text, or other types of information. You can give the box a name, and then you can use that name to access and manipulate the data inside.
  - Using the assignment operator: <-
    - "Less than" (Shift+comma) Hyphen

Objects and Assignment

- In R, to work with values/data, we *assign* them to *objects*
    - Object: named "box" or "container" to store values/data. Think of it as a box where you can store numbers, text, or other types of information. You can give the box a name, and then you can use that name to access and manipulate the data inside.
    - Using the assignment operator: <-
        - "Less than" (Shift+comma) Hyphen
    - object <- values/data

Objects and Assignment

- In R, to work with values/data, we *assign* them to *objects*
  - Object: named "box" or "container" to store values/data. Think of it as a box where you can store numbers, text, or other types of information. You can give the box a name, and then you can use that name to access and manipulate the data inside.
  - Using the assignment operator: <-
    - "Less than" (Shift+comma) Hyphen
  - object <- values/data
- Exercise

Objects and Assignment

- In R, to work with values/data, we *assign* them to *objects*
  - Object: named "box" or "container" to store values/data. Think of it as a box where you can store numbers, text, or other types of information. You can give the box a name, and then you can use that name to access and manipulate the data inside.
  - Using the assignment operator: <-
    - "Less than" (Shift+comma) Hyphen
  - object <- values/data
- Exercise
  - Write two arithmetic operations in R and assign them to unique names. Then perform arithmetic operations using the named results.

Objects and Assignment

- In R, to work with values/data, we *assign* them to *objects*
    - Object: named "box" or "container" to store values/data. Think of it as a box where you can store numbers, text, or other types of information. You can give the box a name, and then you can use that name to access and manipulate the data inside.
    - Using the assignment operator: <-
        - "Less than" (Shift+comma) Hyphen
    - object <- values/data
- Exercise
    - Write two arithmetic operations in R and assign them to unique names. Then perform arithmetic operations using the named results.

```
a <− 5 + 5 + 5
b <− 6 − 6 − 6
a + b
```

Objects, Functions, and Arguments

- In R, we use *commands* (or *functions*) to perform tasks on these *objects*

Objects, Functions, and Arguments

- In R, we use *commands* (or *functions*) to perform tasks on these *objects*
    - Data analysis requires a lot more than just basic arithmetic. R contains many *functions* that can perform mathematical operations, process data, run analyses, and more.

Objects, Functions, and Arguments

- In R, we use *commands* (or *functions*) to perform tasks on these *objects*
    - Data analysis requires a lot more than just basic arithmetic. R contains many *functions* that can perform mathematical operations, process data, run analyses, and more.
    - A function is like a recipe that tells R to perform a specific task. You give the function some input (called arguments), and it gives you an output based on the task it's designed to do.

Objects, Functions, and Arguments

- In R, we use *commands* (or *functions*) to perform tasks on these *objects*
  - Data analysis requires a lot more than just basic arithmetic. R contains many *functions* that can perform mathematical operations, process data, run analyses, and more.
  - A function is like a recipe that tells R to perform a specific task. You give the function some input (called arguments), and it gives you an output based on the task it's designed to do.
  - Type your function, then your object and other arguments in parentheses. For example: command(object)

Objects, Functions, and Arguments

- In R, we use *commands* (or *functions*) to perform tasks on these *objects*
    - Data analysis requires a lot more than just basic arithmetic. R contains many *functions* that can perform mathematical operations, process data, run analyses, and more.
    - A function is like a recipe that tells R to perform a specific task. You give the function some input (called arguments), and it gives you an output based on the task it's designed to do.
    - Type your function, then your object and other arguments in parentheses. For example: command(object)
    - Exercise: calculate the average of the 2 objects you created earlier using the mean() function.

Objects, Functions, and Arguments

- In R, we use *commands* (or *functions*) to perform tasks on these *objects*
  - Data analysis requires a lot more than just basic arithmetic. R contains many *functions* that can perform mathematical operations, process data, run analyses, and more.
  - A function is like a recipe that tells R to perform a specific task. You give the function some input (called arguments), and it gives you an output based on the task it's designed to do.
  - Type your function, then your object and other arguments in parentheses. For example: command(object)
  - Exercise: calculate the average of the 2 objects you created earlier using the mean() function.

```
mean(a, b)
```

Objects, Functions, and Arguments

- R functions typically contain many *arguments*. Your object is usually one argument, other arguments can be thought of as "options". For example, mean() has x, trim, and na.rm.

Objects, Functions, and Arguments

- R functions typically contain many *arguments*. Your object is usually one argument, other arguments can be thought of as "options". For example, mean() has x, trim, and na.rm.
- Many arguments have default values and don't need to be included in function calls.

Objects, Functions, and Arguments

- R functions typically contain many *arguments*. Your object is usually one argument, other arguments can be thought of as "options". For example, mean() has x, trim, and na.rm.
- Many arguments have default values and don't need to be included in function calls.
- Functions are recognizable because they end in (). For example: command(object, other arguments)

Objects, Functions, and Arguments

- R functions typically contain many *arguments*. Your object is usually one argument, other arguments can be thought of as "options". For example, mean() has x, trim, and na.rm.
- Many arguments have default values and don't need to be included in function calls.
- Functions are recognizable because they end in (). For example: command(object, other arguments)
- Documentation for functions can be easily accessed by prefacing the function name with ? and dropping the (). The documentation typically includes a description, a list of the arguments, references, a list of related functions, and examples. The examples are incredibly useful. For example: ?mean

Packages

- Opening RStudio automatically loads base R, a fundamental collection of code and functions.

Packages

- Opening RStudio automatically loads base R, a fundamental collection of code and functions.
- R is an extensible programming language. Users contribute millions of lines of code that can be used by other users without condition or compensation. The main mode of contribution are R packages.

Packages

- Opening RStudio automatically loads base R, a fundamental collection of code and functions.
- R is an extensible programming language. Users contribute millions of lines of code that can be used by other users without condition or compensation. The main mode of contribution are R packages.
- A package is a collection of functions, data, and documentations which is publicly shared to enhance the functionality of R.

Packages

- Opening RStudio automatically loads base R, a fundamental collection of code and functions.
- R is an extensible programming language. Users contribute millions of lines of code that can be used by other users without condition or compensation. The main mode of contribution are R packages.
- A package is a collection of functions, data, and documentations which is publicly shared to enhance the functionality of R.

Packages

- Installing packages

Packages

- Installing packages
    - Packages can be installed using install.packages(""). Include the name of the desired package in quotes as the only argument to the function.

Packages

- Installing packages
  - Packages can be installed using install.packages(""). Include the name of the desired package in quotes as the only argument to the function.
  - Example: Install the package, *tidyverse*

Packages

- Installing packages
  - Packages can be installed using install.packages(""). Include the name of the desired package in quotes as the only argument to the function.
  - Example: Install the package, *tidyverse*
  - It is customary to never include install.packages() in a .R script.

Packages

- Installing packages
    - Packages can be installed using install.packages(""). Include the name of the desired package in quotes as the only argument to the function.
    - Example: Install the package, *tidyverse*
    - It is customary to never include install.packages() in a .R script.
- Using packages
    - After installation, packages need to be loaded once per R session using the library() function. Include the name of the desired package without quotes as the only argument to the function.

Packages

- Installing packages
  - Packages can be installed using install.packages(""). Include the name of the desired package in quotes as the only argument to the function.
  - Example: Install the package, *tidyverse*
  - It is customary to never include install.packages() in a .R script.
- Using packages
  - After installation, packages need to be loaded once per R session using the library() function. Include the name of the desired package without quotes as the only argument to the function.
  - Example: Load the package, *tidyverse*

Packages

- Installing packages
  - Packages can be installed using install.packages(""). Include the name of the desired package in quotes as the only argument to the function.
  - Example: Install the package, *tidyverse*
  - It is customary to never include install.packages() in a .R script.
- Using packages
  - After installation, packages need to be loaded once per R session using the library() function. Include the name of the desired package without quotes as the only argument to the function.
  - Example: Load the package, *tidyverse*
  - It is a good idea to include library() statements at the top of scripts for each package used in the script. This way it is obvious at the top of the script which packages are necessary.