# Computational Bootcamp 3: Datasets in R

Ankushi Mitra

Department of Government
Georgetown University

August 16, 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. Loading in datasets

# What We'll Be Covering Today

1. Loading in datasets
2. Organizing a project

# What We'll Be Covering Today

1. Loading in datasets
2. Organizing a project
3. Basic data manipulation

# What We'll Be Covering Today

1. Loading in datasets
2. Organizing a project
3. Basic data manipulation
4. Advanced Topics?

# Common Packages in R

- tidyverse : collection of R packages designed for data science
  - dplyr : "A Grammar of Data Manipulation"
  - ggplot2 : "The Grammar of Graphics"
  - tidyr : Tools to create "tidy" data
- foreign / haven / readr / readxl: packages to allow importing datasets from file formats other than text
- stargazer : formatted regression tables
- rmarkdown / tinytex : "Dynamic Documents for R"

# Loading Datasets in R

- Datasets can be stored in various file formats. Identify the appropriate function to read the file format you are working with into R. These functions are usually included in packages like *tidyverse*, *readxl* and *haven*.

# Loading Datasets in R

- Datasets can be stored in various file formats. Identify the appropriate function to read the file format you are working with into R. These functions are usually included in packages like *tidyverse*, *readxl* and *haven*.
- Common functions for loading datasets include read_csv (for *.csv* files), read_xlsx (for *.xlsx* files), and read_dta (for *.dta* files).

# Loading Datasets in R

- Datasets can be stored in various file formats. Identify the appropriate function to read the file format you are working with into R. These functions are usually included in packages like *tidyverse*, *readxl* and *haven*.
- Common functions for loading datasets include read_csv (for *.csv* files), read_xlsx (for *.xlsx* files), and read_dta (for *.dta* files).
- Locate the dataset on your computer and determine the filepath. Pass the filepath as an argument to the function you have identified. Example: read_xlsx("filepath").

# Loading Datasets in R

- Exercise
  - Download the dataset *aid.csv* from the course website. Identify the appropriate function to read this file format.

# Loading Datasets in R

- Exercise
  - Download the dataset *aid.csv* from the course website. Identify the appropriate function to read this file format.
  - Pass the filepath for *aid.csv* to this function. Store the imported data in an object called "data".

# Loading Datasets in R

- Exercise
  - Download the dataset *aid.csv* from the course website. Identify the appropriate function to read this file format.
  - Pass the filepath for *aid.csv* to this function. Store the imported data in an object called ”data”.
  - Then view the first six observations in the dataset ”data” using the function *head()*.

```r
data <- read_csv("user/Desktop/PhD/MC/aid.csv")
head(data)
```

# Organizing a Project in R

- While you can load datasets directly from file paths, **it is better to organize your projects and use working directories.**

# Organizing a Project in R

- While you can load datasets directly from file paths, **it is better to organize your projects and use working directories.**
- A working directory acts as a home base for your project. It helps you organize all the files related to your analysis in one place. When you set a working directory, you can load datasets and access files without specifying the full file path each time.

# Organizing a Project in R

- While you can load datasets directly from file paths, **it is better to organize your projects and use working directories.**
- A working directory acts as a home base for your project. It helps you organize all the files related to your analysis in one place. When you set a working directory, you can load datasets and access files without specifying the full file path each time.
- There are many ways to organize projects in R. One way is to use the function *setwd()*.

# Organizing a Project in R

- Start with a folder on your computer where you keep all your files related to a project. This is your working directory. You use *setwd*() to tell R where this project folder is.

# Organizing a Project in R

- Start with a folder on your computer where you keep all your files related to a project. This is your working directory. You use *setwd*() to tell R where this project folder is.
- Exercise
  - Create a folder called *Math Camp* on your computer. Download the *aid.csv* to this folder.

# Organizing a Project in R

- Start with a folder on your computer where you keep all your files related to a project. This is your working directory. You use *setwd*() to tell R where this project folder is.
- Exercise
    - Create a folder called *Math Camp* on your computer. Download the *aid.csv* to this folder.
    - Tell R that *Math Camp* is the working directory for this project by passing the filepath to the *Math Camp* folder using *setwd*("").

# Organizing a Project in R

- Start with a folder on your computer where you keep all your files related to a project. This is your working directory. You use *setwd*() to tell R where this project folder is.
- Exercise
  - Create a folder called *Math Camp* on your computer. Download the *aid.csv* to this folder.
  - Tell R that *Math Camp* is the working directory for this project by passing the filepath to the *Math Camp* folder using *setwd*("").
  - Now load the dataset *aid.csv* by passing only the name of data file into the read_csv function, instead of the full filepath. Store the imported data in an object called "data" again.

# Organizing a Project in R

- Start with a folder on your computer where you keep all your files related to a project. This is your working directory. You use *setwd*() to tell R where this project folder is.
- Exercise
  - Create a folder called *Math Camp* on your computer. Download the *aid.csv* to this folder.
  - Tell R that *Math Camp* is the working directory for this project by passing the filepath to the *Math Camp* folder using *setwd*("").
  - Now load the dataset *aid.csv* by passing only the name of data file into the read_csv function, instead of the full filepath. Store the imported data in an object called "data" again.

```
setwd("user/Desktop/PhD/math_camp")
data <- read_csv("aid.csv")
head(data)
```

# Organizing a Project in R

- Once you set the working directory, R knows where to look for your files. You can read multiple datasets and scripts, run analyses, and do all sorts of things without having to type long file paths every time.

# Organizing a Project in R

- Once you set the working directory, R knows where to look for your files. You can read multiple datasets and scripts, run analyses, and do all sorts of things without having to type long file paths every time.
- As projects get more complex or you work with collaborators, a more flexible and reproducible approach is to use relative paths and directories. Here are some additional resources for setting relative file paths in R. However, as a beginner, working with setwd() is adequate.

## Organizing a Project in R

- Once you set the working directory, R knows where to look for your files. You can read multiple datasets and scripts, run analyses, and do all sorts of things without having to type long file paths every time.
- As projects get more complex or you work with collaborators, a more flexible and reproducible approach is to use relative paths and directories. Here are some additional resources for setting relative file paths in R. However, as a beginner, working with setwd() is adequate.
- Other useful functions to know:
    - getwd() returns the current working directory

# Organizing a Project in R

- Once you set the working directory, R knows where to look for your files. You can read multiple datasets and scripts, run analyses, and do all sorts of things without having to type long file paths every time.
- As projects get more complex or you work with collaborators, a more flexible and reproducible approach is to use relative paths and directories. Here are some additional resources for setting relative file paths in R. However, as a beginner, working with setwd() is adequate.
- Other useful functions to know:
  - getwd() returns the current working directory
  - dir() lists the contents of the directory

# Special Characters in R

- NA: Not Available (i.e. missing values)

# Special Characters in R

- NA: Not Available (i.e. missing values)
- NaN: Not a Number (e.g. 0/0)

# Special Characters in R

- NA: Not Available (i.e. missing values)
- NaN: Not a Number (e.g. 0/0)
- Inf: Infinity

# Special Characters in R

- NA: Not Available (i.e. missing values)
- NaN: Not a Number (e.g. 0/0)
- Inf: Infinity
- -Inf: Minus Infinity. For instance 0 divided by 0 gives a NaN, but 1 divided by 0 gives Inf.

# Relational Syntax

| | |
|---|---|
| < | less than |
| <= | less than or equal to |
| > | greater than |
| >= | greater than or equal to |
| == | equal to |
| ! = | not equal to |
| & | and |
| \| | or |
| NA | missing |
| is.na | is missing |
| !is.na | is not missing |

# Basic Data Manipulation

- Summarizing data: The *tidyverse* package provides powerful tools for summarizing data.

# Basic Data Manipulation

- Summarizing data: The *tidyverse* package provides powerful tools for summarizing data.
  - You can use the *select()* function to create a new dataset containing only the columns you are interested in.
    - Use select(data, column) to select columns you want to keep, or select(data, -column) to drop certain columns. Remember to assign the function to an object to save your work.

# Basic Data Manipulation

- Summarizing data: The *tidyverse* package provides powerful tools for summarizing data.
  - You can use the *select()* function to create a new dataset containing only the columns you are interested in.
    - Use select(data, column) to select columns you want to keep, or select(data, -column) to drop certain columns. Remember to assign the function to an object to save your work.
  - The *filter()* function allows you to “filter out” the rows that you’re not interested in. Rows need to be filtered based on logical conditions.

## Basic Data Manipulation

- Summarizing data: The *tidyverse* package provides powerful tools for summarizing data.
  - You can use the *select()* function to create a new dataset containing only the columns you are interested in.
    - Use select(data, column) to select columns you want to keep, or select(data, -column) to drop certain columns. Remember to assign the function to an object to save your work.
  - The *filter()* function allows you to "filter out" the rows that you're not interested in. Rows need to be filtered based on logical conditions.
    - ==, <, >, <=, >=, !=, `%in%`, and `is.na()` are all operators that can be used for logical conditions. ! can be used to negate a condition, and & and | can be used to combine conditions. | means or.

## Basic Data Manipulation

- Summarizing data: The *tidyverse* package provides powerful tools for summarizing data.
  - You can use the *select()* function to create a new dataset containing only the columns you are interested in.
    - Use select(data, column) to select columns you want to keep, or select(data, -column) to drop certain columns. Remember to assign the function to an object to save your work.
  - The *filter()* function allows you to "filter out" the rows that you're not interested in. Rows need to be filtered based on logical conditions.
    - ==, <, >, <=, >=, !=, `%in%`, and `is.na()` are all operators that can be used for logical conditions. ! can be used to negate a condition, and & and | can be used to combine conditions. | means or.
    - You can use filter(data, condition) to specify which rows to keep using these conditional operators. For example: to keep only observations where the variable *year* equals *2020* in a dataset called *data*, you can use filter(data, year==2020).

# Basic Data Manipulation

- Exercise: Summarizing data
  - In *aid.csv*, drop the column *id* using the *select()* function

# Basic Data Manipulation

- Exercise: Summarizing data
  - In *aid.csv*, drop the column *id* using the *select()* function

```
data <- select(data, -id)
```

# Basic Data Manipulation

- Exercise: Summarizing data
  - In *aid.csv*, drop the column *id* using the *select()* function

    ```
    data <- select(data, -id)
    ```

  - Only keep observations where *status* is "Completion" using the *filter()* function

# Basic Data Manipulation

- Exercise: Summarizing data
  - In *aid.csv*, drop the column *id* using the *select()* function

    ```
    data <- select(data, -id)
    ```

  - Only keep observations where *status* is "Completion" using the *filter()* function

    ```
    data <- filter(data, status == "Completion")
    ```

# Basic Data Manipulation

- Exercise: Summarizing data
  - In *aid.csv*, drop the column *id* using the *select()* function

    ```
    data <- select(data, -id)
    ```

  - Only keep observations where *status* is "Completion" using the *filter()* function

    ```
    data <- filter(data, status == "Completion")
    ```

  - You can chain these commands together using piping, which is done with the %>%. Piping is used to apply a series of operations to your data step by step.

# Basic Data Manipulation

- Exercise: Summarizing data
  - In *aid.csv*, drop the column *id* using the *select()* function

    ```
    data <- select(data, -id)
    ```

  - Only keep observations where *status* is "Completion" using the *filter()* function

    ```
    data <- filter(data, status == "Completion")
    ```

  - You can chain these commands together using piping, which is done with the %>%. Piping is used to apply a series of operations to your data step by step.

    ```
    data <- data %>%
            select(-id) %>%
            filter(status == "Completion")
    ```

# Basic Data Manipulation

- Summarizing data
  - You can use the *group_by()* and *summarize()* functions to inspect the distribution of the data and create summary statistics based on specific variables or groups.

# Basic Data Manipulation

- Summarizing data
  - You can use the *group_by()* and *summarize()* functions to inspect the distribution of the data and create summary statistics based on specific variables or groups.
    - Use the *group_by()* function to group your data by one or more variables. Use the *summarize()* function to take the grouped data and apply a summarizing function to each group. The summarizing function could be something like *count()*, *mean()*, *sum()*, *min()*, *max()*, etc.

# Basic Data Manipulation

- For example, to calculate the mean of variable *gini_score* by *year* in a given dataset *data*:

# Basic Data Manipulation

- For example, to calculate the mean of variable *gini_score* by *year* in a given dataset *data*:

```
data %>%
group_by(year) %>%
summarize(mean(gini_score))
```

# Basic Data Manipulation

- For example, to calculate the mean of variable *gini_score* by *year* in a given dataset *data*:

  ```
  data %>%
  group_by(year) %>%
  summarize(mean(gini_score))
  ```

- Exercise: Summarizing data
  - In *aid.csv*, count the number of observations by *recipient* using *group_by()*, *summarize()*, and *count*.

# Basic Data Manipulation

- For example, to calculate the mean of variable *gini_score* by *year* in a given dataset *data*:

  ```
  data %>%
  group_by(year) %>%
  summarize(mean(gini_score))
  ```

- Exercise: Summarizing data
  - In *aid.csv*, count the number of observations by *recipient* using *group_by()*, *summarize()*, and *count*.

    ```
    df %>%
    group_by(recipient) %>%
    summarize(count = n())
    ```

# Basic Data Manipulation

- The *mutate()* function in R is used to create new columns in a data frame by applying some operation or calculation to the existing columns.

## Basic Data Manipulation

- The *mutate()* function in R is used to create new columns in a data frame by applying some operation or calculation to the existing columns.
- Often, you will have data in two separate datasets that you'd like to combine based on common variables. You can join one dataframe to columns from another dataframe by matching values common in both dataframes using functions like *left_join()*, *inner_join()*, *full_join()*, and *anti_join()*.

# Basic Data Manipulation

- The *mutate()* function in R is used to create new columns in a data frame by applying some operation or calculation to the existing columns.
- Often, you will have data in two separate datasets that you'd like to combine based on common variables. You can join one dataframe to columns from another dataframe by matching values common in both dataframes using functions like *left_join()*, *inner_join()*, *full_join()*, and *anti_join()*.
- You can use *pivot_wider()* and *pivot_longer()* to switch between wide and long formats of the data.This is useful when you want to transform long-format data into a wider format, or vice-versa.

# Resources for Learning R

- Introduction to data analysis using R. I recommend starting here.

# Resources for Learning R

- Introduction to data analysis using R. I recommend starting here.
- Complete introduction to base R.

# Resources for Learning R

- Introduction to data analysis using R. I recommend starting here.
- Complete introduction to base R.
- Cheatsheets on various topics like data transformation and data visualization.
- Advanced R. I recommend reading when you are further along.