Computational Bootcamp 4: Data Manipulation and Visualization in R

Ankushi Mitra

Department of Government

Georgetown University

August 17, 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 manipulation, visualization
5. LaTex: producing documents with Markdown and Overleaf

What We'll Be Covering Today

1. Basic data manipulation

What We'll Be Covering Today

1. Basic data manipulation
2. Basic data visualization

Basic Data Manipulation

- The *mutate()* function is used to create new variables or modify existing ones within a dataset.

Basic Data Manipulation

- The *mutate()* function is used to create new variables or modify existing ones within a dataset.
- The basic syntax of the *mutate()* function is mutate(data, new column = calculation/transformation).

Basic Data Manipulation

- The *mutate()* function is used to create new variables or modify existing ones within a dataset.
- The basic syntax of the *mutate()* function is mutate(data, new column = calculation/transformation).
- For example, let's say you have a dataset of students' test scores on different subjects and you want to create a column *total* based on columns *math* and *science*: mutate(data, total = math + science)

Basic Data Manipulation: Exercise

- Set your *Math Camp* folder as the working directory and save the script. Assign the built-in R dataset *storms* to an object *data*

Basic Data Manipulation: Exercise

- Set your *Math Camp* folder as the working directory and save the script. Assign the built-in R dataset *storms* to an object *data*

```
data <- storms
```

Basic Data Manipulation: Exercise

- Set your *Math Camp* folder as the working directory and save the script. Assign the built-in R dataset *storms* to an object *data*

```
data <- storms
```

- *storms* has a column, *category*, which categorizes storms from levels 1-5. Using *mutate()*, create a new variable called *level_5*

Basic Data Manipulation: Exercise

- Set your *Math Camp* folder as the working directory and save the script. Assign the built-in R dataset *storms* to an object *data*

```
data <- storms
```

- *storms* has a column, *category*, which categorizes storms from levels 1-5. Using *mutate()*, create a new variable called *level_5*
- *level_5* should equal 1 if the column *category* equals 5 and 0 for all other categories. *Hint: if_else()* is useful and works like the IF command in Microsoft Excel.

Basic Data Manipulation: Exercise

- Set your *Math Camp* folder as the working directory and save the script. Assign the built-in R dataset *storms* to an object *data*

```
data <- storms
```

- *storms* has a column, *category*, which categorizes storms from levels 1-5. Using *mutate()*, create a new variable called *level_5*
- *level_5* should equal 1 if the column *category* equals 5 and 0 for all other categories. *Hint: if_else()* is useful and works like the IF command in Microsoft Excel.

```
data <- data %>%
mutate(level_5 = if_else(
       category == 5, 1, 0))
```

Basic Data Manipulation: Exercise

- Set your *Math Camp* folder as the working directory and save the script. Assign the built-in R dataset *storms* to an object *data*

```
data <- storms
```

- *storms* has a column, *category*, which categorizes storms from levels 1-5. Using *mutate()*, create a new variable called *level_5*
- *level_5* should equal 1 if the column *category* equals 5 and 0 for all other categories. *Hint: if_else()* is useful and works like the IF command in Microsoft Excel.

```
data <- data %>%
mutate(level_5 = if_else(
       category == 5, 1, 0))
```

- Drop observations with missing (NA) values using function *drop_na()*

Basic Data Manipulation: Exercise

- Set your *Math Camp* folder as the working directory and save the script. Assign the built-in R dataset *storms* to an object *data*

```
data <- storms
```

- *storms* has a column, *category*, which categorizes storms from levels 1-5. Using *mutate()*, create a new variable called *level_5*

- *level_5* should equal 1 if the column *category* equals 5 and 0 for all other categories. *Hint: if_else()* is useful and works like the IF command in Microsoft Excel.

```
data <- data %>%
mutate(level_5 = if_else(
         category == 5, 1, 0))
```

- Drop observations with missing (NA) values using function *drop_na()*

```
data <- drop_na(data)
```

Why Data Visualization

1. Exploratory analysis

Why Data Visualization

1. Exploratory analysis
2. Diagnosis and validation

Why Data Visualization

1. Exploratory analysis
2. Diagnosis and validation
3. Communication

Why Data Visualization

1. Exploratory analysis
2. Diagnosis and validation
3. Communication
4. Flexibility, reproducibility, scalability

Working with ggplot

- *ggplot2* is an R package for data visualization. Install and load the package in R.

Working with ggplot

- *ggplot2* is an R package for data visualization. Install and load the package in R.
- You build up a plot by adding layers that represent different aspects of your data. Examples of layers include scatter points, lines, labels, titles, and so on.

Working with ggplot

- *ggplot2* is an R package for data visualization. Install and load the package in R.
- You build up a plot by adding layers that represent different aspects of your data. Examples of layers include scatter points, lines, labels, titles, and so on.
- This layering approach allows you to build complex visualizations step by step while maintaining clarity and customization.

Working with ggplot

- *ggplot2* is an R package for data visualization. Install and load the package in R.
- You build up a plot by adding layers that represent different aspects of your data. Examples of layers include scatter points, lines, labels, titles, and so on.
- This layering approach allows you to build complex visualizations step by step while maintaining clarity and customization.
- There are eight main ingredients to ggplot visualizations.

Working with ggplot

- Data are the values represented in the visualization.

Working with ggplot

- Data are the values represented in the visualization.

```
ggplot(data = ) or data %>% ggplot()
```

Working with ggplot

- Data are the values represented in the visualization.

```
    ggplot(data = ) or data %>% ggplot()
```

- Aesthetic mappings are directions for how data are mapped in a plot in a way that we can perceive. Aesthetic mappings link variables to the x-position, y-position, color, fill, shape, transparency, and size.

Working with ggplot

- Data are the values represented in the visualization.

  ```
  ggplot(data = ) or data %>% ggplot()
  ```

- Aesthetic mappings are directions for how data are mapped in a plot in a way that we can perceive. Aesthetic mappings link variables to the x-position, y-position, color, fill, shape, transparency, and size.

  ```
  aes(x = , y = , color = )
  ```

Working with ggplot

- Data are the values represented in the visualization.

  ```
  ggplot(data = ) or data %>% ggplot()
  ```

- Aesthetic mappings are directions for how data are mapped in a plot in a way that we can perceive. Aesthetic mappings link variables to the x-position, y-position, color, fill, shape, transparency, and size.

  ```
  aes(x = , y = , color = )
  ```

- Geometric objects are representations of the data, including points, lines, and polygons.

Working with ggplot

- Data are the values represented in the visualization.

  ```
  ggplot(data = ) or data %>% ggplot()
  ```

- Aesthetic mappings are directions for how data are mapped in a plot in a way that we can perceive. Aesthetic mappings link variables to the x-position, y-position, color, fill, shape, transparency, and size.

  ```
  aes(x = , y = , color = )
  ```

- Geometric objects are representations of the data, including points, lines, and polygons.

  ```
  geom_bar(), geom_col(), geom_line(), geom_point(),
  geom_histogram(), geom_smooth()
  ```

Working with ggplot

- Theme controls the visual style of plot with font types, font sizes, background colors, margins, and positioning.

Working with ggplot

- Theme controls the visual style of plot with font types, font sizes, background colors, margins, and positioning.

```
theme_bw(), theme_minimal(), theme_classic()
```

Working with ggplot

- Theme controls the visual style of plot with font types, font sizes, background colors, margins, and positioning.

  `theme_bw(), theme_minimal(), theme_classic()`

- Scales turn data values into aesthetic values. This includes the x-axis and y-axis, and ranges of sizes, shapes, and colors of aesthetics. Unless otherwise specified, *ggplot()* will use default scales.

Working with ggplot

- Theme controls the visual style of plot with font types, font sizes, background colors, margins, and positioning.

  `theme_bw(), theme_minimal(), theme_classic()`

- Scales turn data values into aesthetic values. This includes the x-axis and y-axis, and ranges of sizes, shapes, and colors of aesthetics. Unless otherwise specified, *ggplot()* will use default scales.

  `scale_x_continuous(), scale_x_discrete()`

Working with ggplot

- Theme controls the visual style of plot with font types, font sizes, background colors, margins, and positioning.

  `theme_bw(), theme_minimal(), theme_classic()`

- Scales turn data values into aesthetic values. This includes the x-axis and y-axis, and ranges of sizes, shapes, and colors of aesthetics. Unless otherwise specified, *ggplot()* will use default scales.

  `scale_x_continuous(), scale_x_discrete()`

- Other ingredients include coordinate systems, facets, and statistical transformations.

Working with ggplot

- Theme controls the visual style of plot with font types, font sizes, background colors, margins, and positioning.

  `theme_bw(), theme_minimal(), theme_classic()`

- Scales turn data values into aesthetic values. This includes the x-axis and y-axis, and ranges of sizes, shapes, and colors of aesthetics. Unless otherwise specified, *ggplot()* will use default scales.

  `scale_x_continuous(), scale_x_discrete()`

- Other ingredients include coordinate systems, facets, and statistical transformations.
- You use + to chain different layers together in *ggplot()*.

Working with ggplot

- Exercise: A barplot using *storms*

Working with ggplot

- Exercise: A barplot using *storms*
    - Make a barplot of the number of Level 5 storms, *level_5*, by *year* in *data* using *ggplot2*. Put *year* on the x-axis and *level_5* on the y-axis.

Working with ggplot

- Exercise: A barplot using *storms*
  - Make a barplot of the number of Level 5 storms, *level_5*, by *year* in *data* using *ggplot2*. Put *year* on the x-axis and *level_5* on the y-axis.

```
data %>%
ggplot(aes(x = year, y = level_5)) +
geom_col()
```

Working with ggplot

- Exercise: A barplot using *storms*
  - Make a barplot of the number of Level 5 storms, *level_5*, by *year* in *data* using *ggplot2*. Put *year* on the x-axis and *level_5* on the y-axis.

    ```
    data %>%
    ggplot(aes(x = year, y = level_5)) +
    geom_col()
    ```

  - Use *theme_minimal()*

Working with ggplot

- Exercise: A barplot using *storms*
  - Make a barplot of the number of Level 5 storms, *level_5*, by *year* in *data* using *ggplot2*. Put *year* on the x-axis and *level_5* on the y-axis.

```
data %>%
ggplot(aes(x = year, y = level_5)) +
geom_col()
```

  - Use *theme_minimal()*

```
data %>%
ggplot(aes(x = year, y = level_5)) +
geom_col() +
theme_minimal()
```

Working with ggplot

- Clean up the axis labels and add a title using function *labs()*.

Working with ggplot

- Clean up the axis labels and add a title using function *labs()*.

```
data %>%
ggplot(aes(x = year, y = level_5)) +
geom_col() +
theme_minimal() +
labs(title = "Level 5 Storms by Year",
     y = "Number of Level-5 Storms")
```

Working with ggplot

- You can further clean up the x-axis labels by making them vertical.

Working with ggplot

- You can further clean up the x-axis labels by making them vertical.

```
data %>%
ggplot(aes(x = year, y = level_5)) +
geom_col() +
theme_minimal() +
labs(title = "Level 5 Storms by Year",
     y = "Number of Level-5 Storms") +
theme(axis.text.x =
      element_text(angle = 90, hjust = 1))
```

Working with ggplot

- Exercise: A scatterplot using the *economics* dataset

Working with ggplot

- Exercise: A scatterplot using the *economics* dataset
  - Make a scatterplot of the median duration of unemployment, *uempmed*, by *date* in *economics* using *ggplot2*. Put *date* on the x-axis and *uempmed* on the y-axis.

Working with ggplot

- Exercise: A scatterplot using the *economics* dataset
  - Make a scatterplot of the median duration of unemployment, *uempmed*, by *date* in *economics* using *ggplot2*. Put *date* on the x-axis and *uempmed* on the y-axis.

```
economics %>%
ggplot(aes(x = date, y = uempmed)) +
  geom_point() +
  labs(title = "Median Unemployment Duration by Date",
       x = "Date",
       y = "Median Unemployment Duration") +
       theme_bw()
```

Working with ggplot

- Try adding *geom_line()* as another layer.

Working with ggplot

- Try adding *geom_line()* as another layer.

```
economics %>%
ggplot(aes(x = date, y = uempmed)) +
  geom_point() +
  geom_line()+
  labs(title = "Median Unemployment Duration by Date",
       x = "Date",
       y = "Median Unemployment Duration") +
  theme_bw()
```

Resources for Data Visualization in R

- Elegant graphics for data analysis

Resources for Data Visualization in R

- Elegant graphics for data analysis
- R graphics cookbook

Resources for Data Visualization in R

- Elegant graphics for data analysis
- R graphics cookbook
- The complete ggplot tutorial

Resources for Data Visualization in R

- Elegant graphics for data analysis
- R graphics cookbook
- The complete ggplot tutorial
- R graph gallery