Interested in **a verified certificate or transfer credit and accreditation**?

CS50's Introduction to Programming with R

OpenCourseWare

Donate

Carter Zenke
carter@cs50.harvard.edu

David J. Malan
malan@harvard.edu

Menu

Lecture 5

Welcome!

- Welcome back to CS50's Introduction to Programming with R!
- Today, we will be learning about visualizing data. A good visualization can help us interpret and understand data in a whole new way.

ggplot2

- The `plot` in `ggplot` means that we are going to *plot* our data.
- The `gg` in `ggplot` references a *grammar of graphics* where individual components of graphics can be gathered together to visualize data.
- There are many components that make up this grammar of graphics, starting with *data*.
- Another component is *geometries*. These are the various types of graphical representation options for plots. These include columns, points, and lines.
- Finally, *aesthetic mappings* are the relationships between the data and visual features of our plot. For example, in a plot, a horizontal `x` axis may represent each candidate. Then, a vertical `y` axis may be associated with the number of votes for each candidate. It is through this relationship between data and geometries that we are able to visualize and understand the aesthetic map of our plot. You may be able to imagine times when poorly designed plots have been shown to you: when the mapping is incorrect, data is more challenging to interpret and understand.
- Download the lecture's source files and run `library("tidyverse")` in the R console such that the `tidyverse` is loaded into memory. Then, create a visualization as follows:

  ```
  # Create a blank visualization

  votes <- read.csv("votes.csv")

  ggplot()
  ```

  Notice how `votes.csv` is loaded into `votes`. When `ggplot` is run, nothing is currently visualized.
- We can provide inputs to `ggplot` as follows:

  ```
  # Supply data

  votes <- read.csv("votes.csv")

  ggplot(votes)
  ```

  Notice how `votes` is provided to `ggplot`. Still, nothing is visualized.
- We need to tell `ggplot` what type of plot we want:

  ```
  # Add first geometry

  votes <- read.csv("votes.csv")

  ggplot(votes) +
    geom_col()
  ```

  Notice that `geom_col` specifies that the data should be visualized with a column geometry. However, at this point, an error will result. The error indicates that we need to designate aesthetic mappings.
- Notice, too, that the `+` operator has a new meaning: using the `+` operator adds a layer on

top of the base layer of your plot.

- To designate aesthetic mappings, we can define them as follows:

```
# Add x and y aesthetics

votes <- read.csv("votes.csv")

ggplot(votes, aes(x = candidate, y = votes)) +
  geom_col()
```

Notice how various aesthetic mappings, designated by `aes`, are defined within the parenthesis. For example, `x = candidate` and `y = votes` are both aesthetic mappings. Now, `ggplot` knows which data maps to which aesthetic features of our plot.

- Running the above code, our first visualization finally appears!

Scales

- Notice how `ggplot` has decided that the values of the `votes` axis range from `0` to `200`. What if we wanted to provide more headroom, such that we could visualize up to `250`? Let's learn about **Scales**.
- Scales can be *continuous*, ranging from one number to another, or *discrete*, which means categorical.
- Continuous scales have *limits*. For example, the data provided in `votes` ranges from `0` to `200`. Hence, we can modify these limits as follows:

```
# Adjust y scale

votes <- read.csv("votes.csv")

ggplot(votes, aes(x = candidate, y = votes)) +
  geom_col() +
  scale_y_continuous(limits = c(0, 250))
```

Notice how the scale of `y` is modified via `scale_y_continuous` to range from `0` to `250`. This, again, is provided via a new layer with the `+` operator.

Labels

- Additionally, one can add labels to the plot. Consider the following:

```
# Add labels

votes <- read.csv("votes.csv")

ggplot(votes, aes(x = candidate, y = votes)) +
  geom_col() +
  scale_y_continuous(limits = c(0, 250)) +
```

```
  labs(
    x = "Candidate",
    y = "Votes",
    title = "Election Results"
  )
```

Notice how labels are provided for `x`, `y`, and `title`. These are added as a new layer via the `+` operator.

Fill

- The fill color can also be changed, depending on the `candidate` name. Consider the following:

```
# Add fill aesthetic mapping for geom_col

votes <- read.csv("votes.csv")

ggplot(votes, aes(x = candidate, y = votes)) +
  geom_col(aes(fill = candidate)) +
  scale_y_continuous(limits = c(0, 250)) +
  labs(
    x = "Candidate",
    y = "Votes",
    title = "Election Results"
  )
```

Notice that the `fill` is dependent upon the `candidate` via the `aes` function.

- We may wish to adjust the `fill` color to be friendly for color blindness. We can do so as follows:

```
# Use viridis scale to design for color blindness

votes <- read.csv("votes.csv")

ggplot(votes, aes(x = candidate, y = votes)) +
  geom_col(aes(fill = candidate)) +
  scale_fill_viridis_d("Candidate") +
  scale_y_continuous(limits = c(0, 250)) +
  labs(
    x = "Candidate",
    y = "Votes",
    title = "Election Results"
  )
```

Notice how the `viridis` scale is provided via `scale_fill_viridis_d` function.

Themes

- One can also modify the themes being used by `ggplot`. You can do so as follows:

```r
# Adjust ggplot theme

votes <- read.csv("votes.csv")

ggplot(votes, aes(x = candidate, y = votes)) +
  geom_col(aes(fill = candidate)) +
  scale_fill_viridis_d("Candidate") +
  scale_y_continuous(limits = c(0, 250)) +
  labs(
    x = "Candidate",
    y = "Votes",
    title = "Election Results"
  ) +
  theme_classic()
```

Notice that `theme_classic` is provided. ggplot offers several themes.

Saving Your Plot

- Finally, plots can be saved.

```r
# Save file

votes <- read.csv("votes.csv")

p <- ggplot(votes, aes(x = candidate, y = votes)) +
  geom_col(aes(fill = candidate)) +
  scale_fill_viridis_d("Candidate") +
  scale_y_continuous(limits = c(0, 250)) +
  labs(
    x = "Candidate",
    y = "Votes",
    title = "Election Results"
  ) +
  theme_classic()

ggsave(
  "votes.png",
  plot = p,
  width = 1200,
  height = 900,
  units = "px"
)
```

Notice how the entire plot is designated as `p`. Then, `ggsave` is employed, naming the file name, the plot (in this case, `p`), height, width, and the units.

- By executing this code, you have saved your first plot. Congratulations!

Point

- Now, let's look at a new type of geometry called `point`.

- Imagine data where candy's price percentile and sugar percentile are represented.
- You can imagine how the `sugar` percentile can be mapped on the `y` axis while the `price` percentile can be noted on the `x` axis.
- This can be realized in code form as follows:

```
# Introduce geom_point

load("candy.RData")

ggplot(
  candy,
  aes(x = price_percentile, y = sugar_percentile)
) +
  geom_point()
```

  Notice how the data `candy` is provided to the `ggplot` function. Then, the aesthetic mappings are set with the `aes` function. `price_percentile`, for example, is assigned to the `x` axis. Finally, the `geom_point` function is run.
- Running this code results in points being represented in a plot.
- Labels can be added as follows:

```
# Add labels and theme

load("candy.RData")

ggplot(
  candy,
  aes(x = price_percentile, y = sugar_percentile)
) +
  geom_point() +
  labs(
    x = "Price",
    y = "Sugar",
    title = "Price and Sugar"
  ) +
  theme_classic()
```

  Notice how `labs` (labels) for `x`, `y`, and `title` are provided. Also, a theme is named.
- Now, a number of points do overlap. `jitter` can be used to help visualize points that overlap:

```
# Introduce geom_jitter

ggplot(
  candy,
  aes(x = price_percentile, y = sugar_percentile)
) +
  geom_jitter() +
  labs(
    x = "Price",
    y = "Sugar",
    title = "Price and Sugar"
```

```
  ) +
  theme_classic()
```

Notice how `geom_point` is replaced with `geom_jitter`. This allows for the visualization of points that overlap.

- We can add color aesthetics to our points:

```
# Introduce size and color aesthetic

ggplot(
  candy,
  aes(x = price_percentile, y = sugar_percentile)
) +
  geom_jitter(
    color = "darkorchid",
    size = 2
  ) +
  labs(
    x = "Price",
    y = "Sugar",
    title = "Price and Sugar"
  ) +
  theme_classic()
```

Notice how all points are changed to one color.

- Further, we can change the size and shape of our points:

```
# Introduce point shape and fill color

ggplot(
  candy,
  aes(x = price_percentile, y = sugar_percentile)
) +
  geom_jitter(
    color = "darkorchid",
    fill = "orchid",
    shape = 21,
    size = 2
  ) +
  labs(
    x = "Price",
    y = "Sugar",
    title = "Price and Sugar"
  ) +
  theme_classic()
```

Notice how `shape` and `size` are changed. You can reference the documentation to learn more about which numbers correspond to which shapes.

Visualizing Over Time

- You can imagine how data can be represented over time.

- For example, consider how data regarding Hurricane Anita may be represented over time.
- We could plot as we did prior with points:

  ```
  # Visualize with geom_point

  load("anita.RData")

  ggplot(anita, aes(x = timestamp, y = wind)) +
    geom_point()
  ```

  Notice how `timestamp` and `wind` speed are placed in points over time.

- While this visualization is useful, it could be more useful to present with lines showing whether wind speed increased or decreased. Each point can be connected with a line as follows:

  ```
  # Introduce geom_line

  load("anita.RData")

  ggplot(anita, aes(x = timestamp, y = wind)) +
    geom_line()
  ```

  Notice `geom_line` is employed as a new layer.

- What results is a series of lines that change direction at each timestamp. What if we could combine both `point` and `line`? Well, indeed, we can!

  ```
  # Combine geom_line and geom_point

  load("anita.RData")

  ggplot(anita, aes(x = timestamp, y = wind)) +
    geom_line() +
    geom_point(color = "deepskyblue4")
  ```

  Notice how a layer with lines is added via `geom_line`. Then, `geom_point` is added as a layer using `deepskyblue4`.

- Aesthetics can be modified in various ways:

  ```
  # Experiment with geom_line and geom_point aesthetics

  load("anita.RData")

  ggplot(anita, aes(x = timestamp, y = wind)) +
    geom_line(
      linetype = 1,
      linewidth = 0.5
    ) +
    geom_point(
      color = "deepskyblue4",
      size = 2
    )
  ```

Notice how the `linetype` and `linewidth` are modified. Then, the `size` of the points is changed. You can reference the documentation to learn more about various line types.

- As with our prior plots today, we can add labels and a theme:

```
# Add labels and adjust theme

load("anita.RData")

ggplot(anita, aes(x = timestamp, y = wind)) +
  geom_line(
    linetype = 1,
    linewidth = 0.5
  ) +
  geom_point(
    color = "deepskyblue4",
    size = 2
  ) +
  labs(
    y = "Wind Speed (Knots)",
    x = "Date",
    title = "Hurricane Anita"
  ) +
  theme_classic()
```

Notice how `labs` allows us to designate labels for `y`, `x`, and `title`. Then, `theme_classic` is enabled.

- As a final flourish, we can also add a horizontal line to demarcate the hurricane status. When did Hurricane Anita become a hurricane?

```
# Add horizontal line to demarcate hurricane status

load("anita.RData")

ggplot(anita, aes(x = timestamp, y = wind)) +
  geom_line(
    linetype = 1,
    linewidth = 0.5
  ) +
  geom_point(
    color = "deepskyblue4",
    size = 2
  ) +
  geom_hline(
    linetype = 3,
    yintercept = 64
  ) +
  labs(
    y = "Wind Speed (Knots)",
    x = "Date",
    title = "Hurricane Anita"
  ) +
  theme_classic()
```

Notice how a new layer is added to display a line at `yintercept = 64`, to designate that anything above 65 or higher is considered a hurricane. The `linetype` is designated as `3` or dotted.

Summing Up

In this lesson, you learned how to visualize data in R. Specifically, you learned about:

- ggplot2
- Scales
- Labels
- Fill
- Themes
- Point
- Visualizing over time

See you next time when we discuss how to test our programs.