STATS 266 Handout - Data Visualization

Qi Wang

2025-03-01

Contents

1 Introduction

Welcome to **STATS 266: Introduction to R**. This handout provides an introduction about data visualization in R. By the end of this document, you should be able to:

- To be able to use `ggplot2` to generate publication-quality graphics.
- To apply geometry, aesthetic, and statistics layers to a `ggplot` plot.
- To manipulate the aesthetics of a plot using different colors, shapes, and lines.
- To improve data visualization through transforming scales and paneling by group.
- To save a plot created with `ggplot` to disk.

For this part, valuable materials to refer to include https://ggplot2.tidyverse.org/ and https://swcarpentry.github.io/r-novice-gapminder/08-plot-ggplot2.html.

2 Reading the Data

Before doing any visualization, we should read the data into our environment. At the very beginning of the course, I introduced working for a project, and `here::here()` function. That's for the file path to the document. With this function, you will not need to specify the file path again.

R can read a variety of data types, including structured and unstructured formats. Below are the common types of data that R can read along with the functions used to import them:

- **CSV Files**: `read.csv("file.csv")` or `readr::read_csv("file.csv")`
- **Excel Files**: `readxl::read_excel("file.xlsx")`
- **Text Files (TSV, Fixed Width, etc.)**:
  - `read.table("file.txt")`
  - `read.delim("file.txt")`
  - `readr::read_delim("file.txt", delim = "\t")`
- **JSON Files**: `jsonlite::fromJSON("file.json")`
- **XML Files**: `XML::xmlParse("file.xml")`
- **SPSS, SAS, and Stata Files**:
  - `haven::read_sav("file.sav")` (SPSS)
  - `haven::read_sas("file.sas7bdat")` (SAS)
  - `haven::read_dta("file.dta")` (Stata)
- **R Binary Files**:
  - `load("file.RData")`

    - `readRDS("file.rds")`

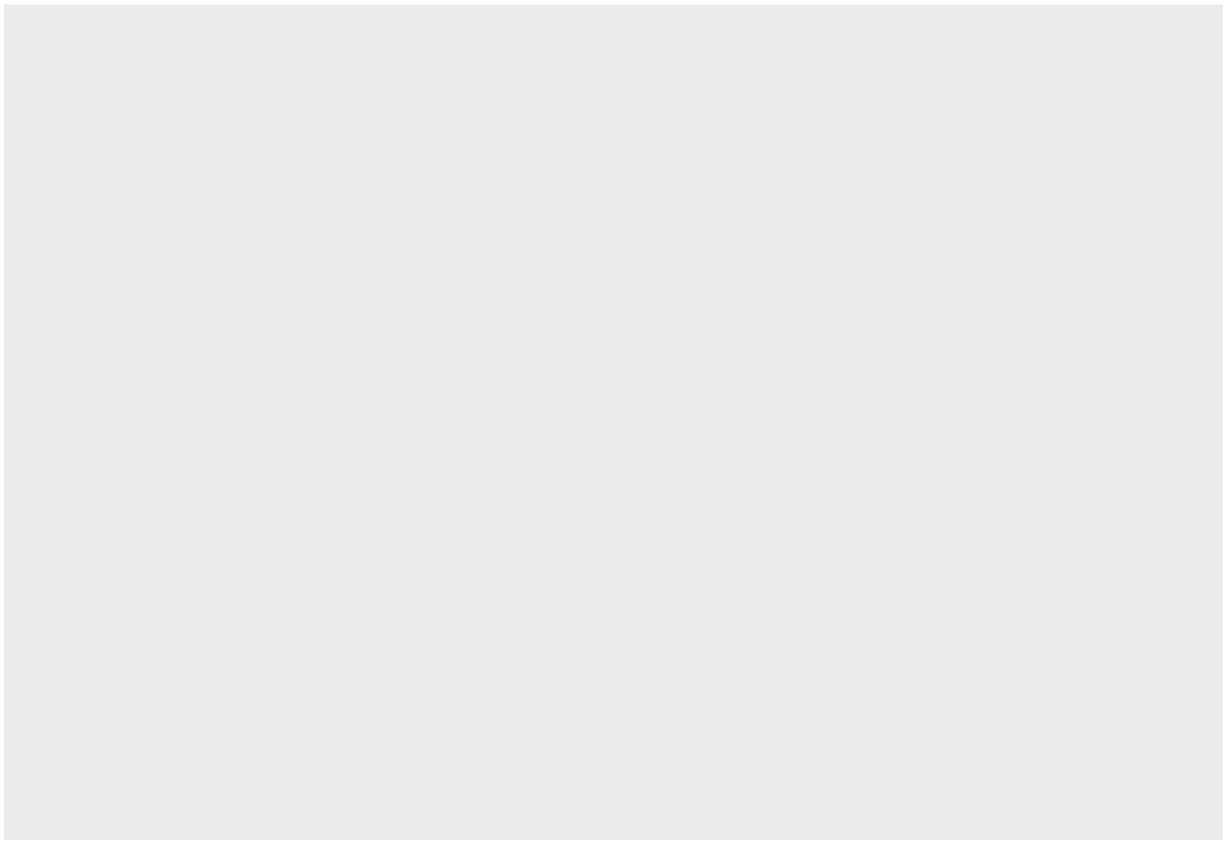
- **Database Connections**:
    - `DBI::dbReadTable(con, "table_name")`
    - `dplyr::tbl(con, "table_name")`

For efficient data handling, packages like `data.table`, `readr`, and `vroom` provide optimized functions for reading large datasets.

3 ggplot2

`ggplot2` is a package in R that can create good looking figures in R. It works like putting a layer on another layer, so we use "+" to concatenate layers together.

```r
library(ggplot2)
ggplot()
```

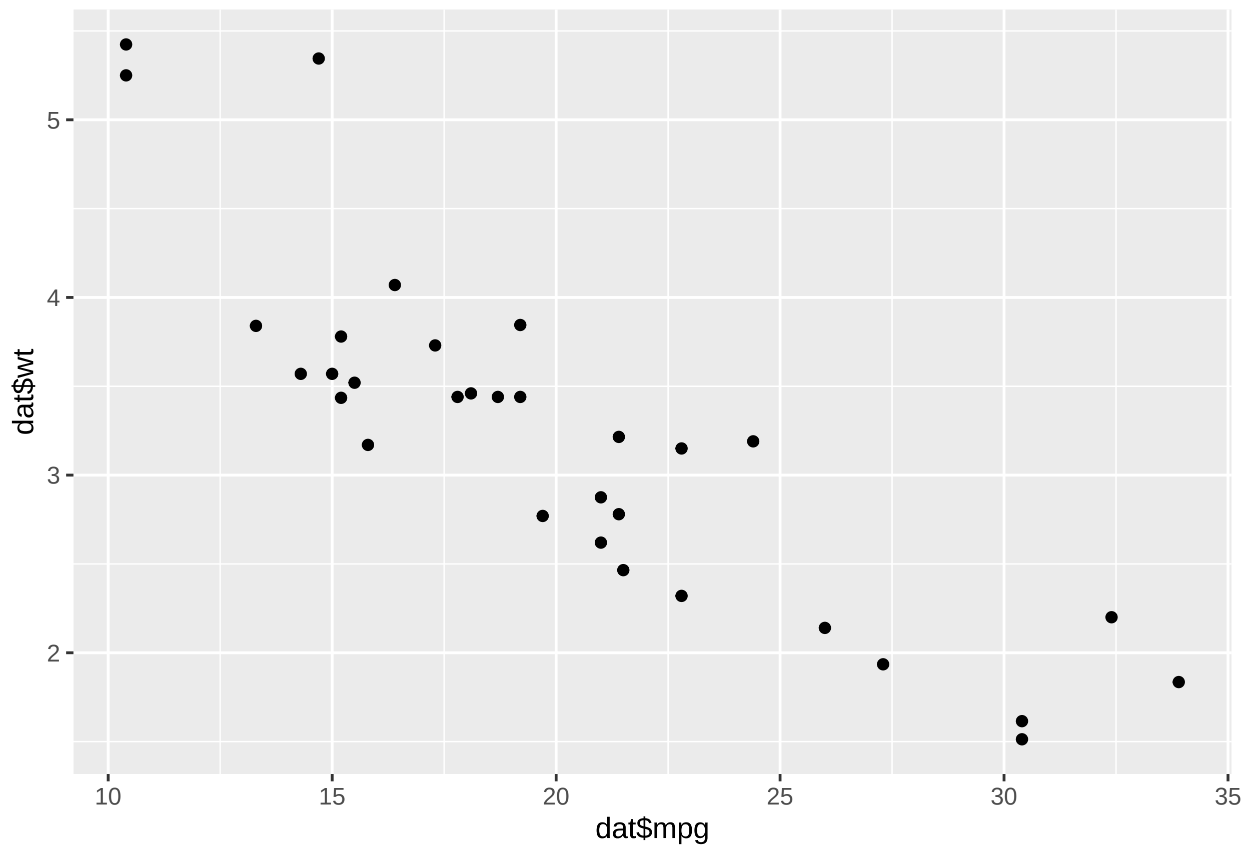


If we only run a `ggplot()`, it returns to a blank figure to us, since we only set up a background, no future plots are made. For the next steps, we will use the in-built dataset "mtcars" as examples.

3.1 Scatter Plots

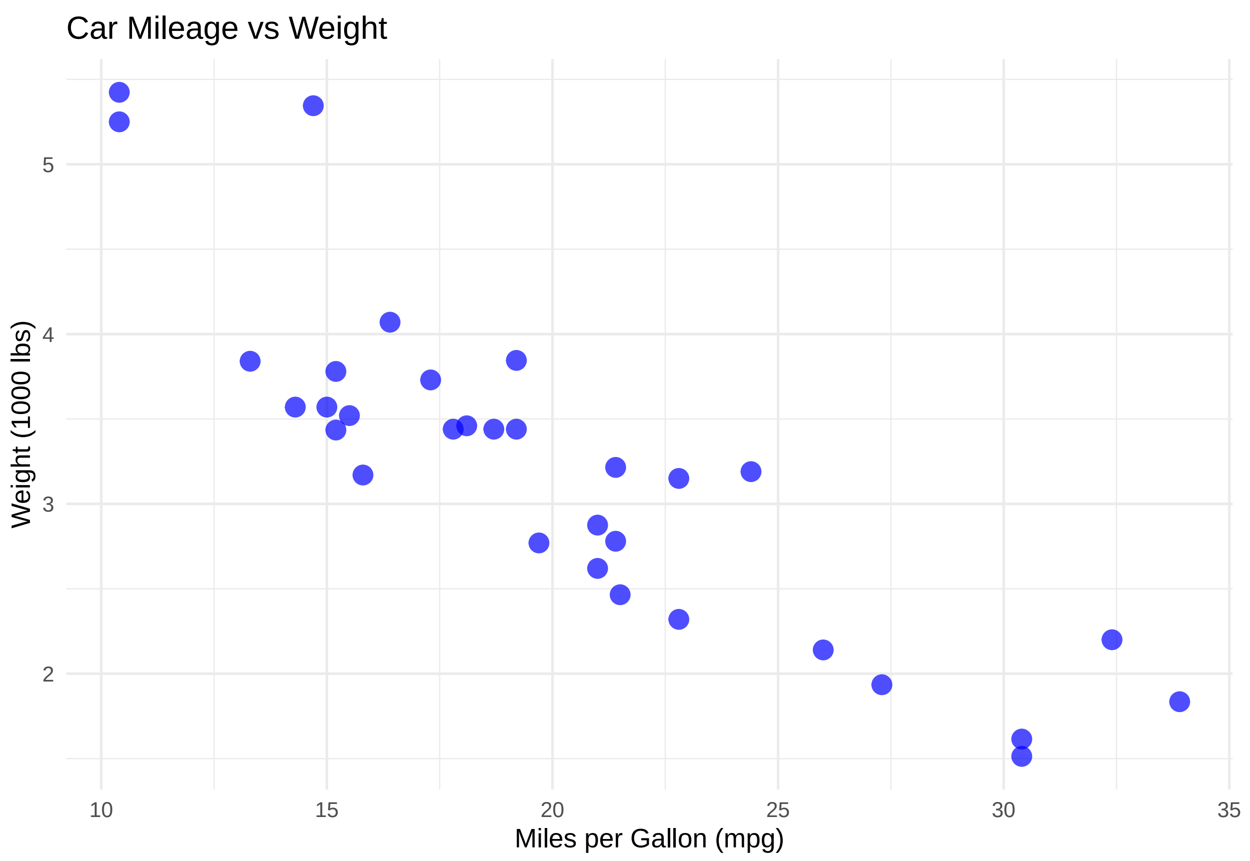
Scatter plots is one of the most commonly used plots in statistics. It visualizes the relationship between two variables. We can distinguish whether they are positively or negatively correlated based on scatter plots. It's also useful when we compare the values of two variables.

```r
dat <- mtcars
ggplot() +
  geom_point(aes(x = dat$mpg, y = dat$wt))
```



Based on the blank figure, we added one more layer called `geom_point`. This returns to a scatter plot, with `aes(x = ...,y = ...)` being the x-axis values and y-axis values. Based on this, we can do something to make the plot more readable:

```r
ggplot() +
  geom_point(data = mtcars, aes(x = mpg, y = wt), color = "blue", size = 3, alpha = 0.7) +  #
  labs(
    title = "Car Mileage vs Weight",
    x = "Miles per Gallon (mpg)",
    y = "Weight (1000 lbs)"
  ) +
  theme_minimal(base_size = 10)  # Using a clean theme
```

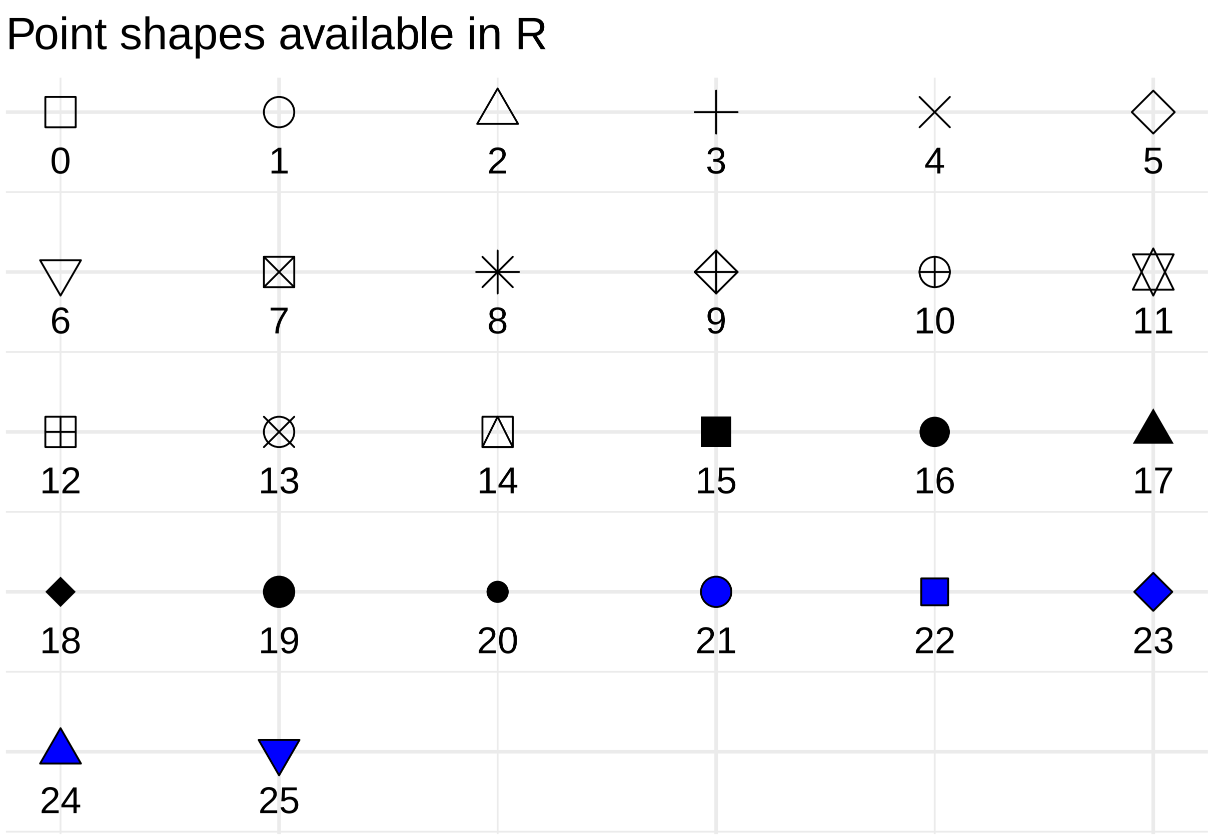


In the `labs()`, we can set the legend title, figure title, and xy axis labels. In the `geom_point` we can set the size of the point, shape of the point, and the color of the point. There are 25 different styles of points in R:

```r
library(ggplot2)

# Create a dataframe with shape IDs
point_shapes <- data.frame(
  x = rep(1:6, length.out = 26),  # Adjust x positions for 26 points
  y = rep(5:1, each = 6, length.out = 26),  # Adjust y positions for 26 points
  shape = 0:25  # Shape IDs (0 to 25)
)

# Create the plot
ggplot(point_shapes, aes(x = x, y = y)) +
  geom_point(aes(shape = shape), size = 5, fill = "blue") +  # Shape varies, filled for 21-25
  scale_shape_identity() +  # Use shape IDs directly
  geom_text(aes(label = shape), nudge_y = -0.3, size = 5) +  # Label each point with its shape
  labs(title = "Point shapes available in R") +
  theme_minimal(base_size = 14) +
  theme(axis.title = element_blank(), axis.text = element_blank(), axis.ticks = element_blank(
```



3.2 Line Chart / Time Series Plot

Sometimes, if the data is a time series:

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
library(lubridate)  # Ensure proper date handling
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
# Set English locale for months
# Sys.setlocale("LC_TIME", "C")

# Simulated time series data
set.seed(123)
time_series_data <- data.frame(
  date = seq(as.Date("2023-01-01"), by = "month", length.out = 12),
  series_A = cumsum(rnorm(12, mean = 5, sd = 2)),
  series_B = cumsum(rnorm(12, mean = 3, sd = 1.5)),
  series_C = cumsum(rnorm(12, mean = 4, sd = 1.8))
)
time_series_data
```

```
##          date  series_A  series_B  series_C
## 1  2023-01-01  3.879049  3.601157  2.874929
## 2  2023-02-01  8.418694  6.767181  3.838881
## 3  2023-03-01 16.536110  8.933420  9.346898
## 4  2023-04-01 21.677127 14.613789 13.622970
## 5  2023-05-01 26.935703 18.360565 15.574323
## 6  2023-06-01 35.365833 18.410639 21.831190
## 7  2023-07-01 41.287665 22.462673 26.598826
## 8  2023-08-01 43.757543 24.753486 30.067697
## 9  2023-09-01 47.383837 26.151750 35.678923
## 10 2023-10-01 51.492513 28.824788 41.259563
## 11 2023-11-01 58.940676 30.285781 46.738409
## 12 2023-12-01 64.660304 32.192445 51.977962
```

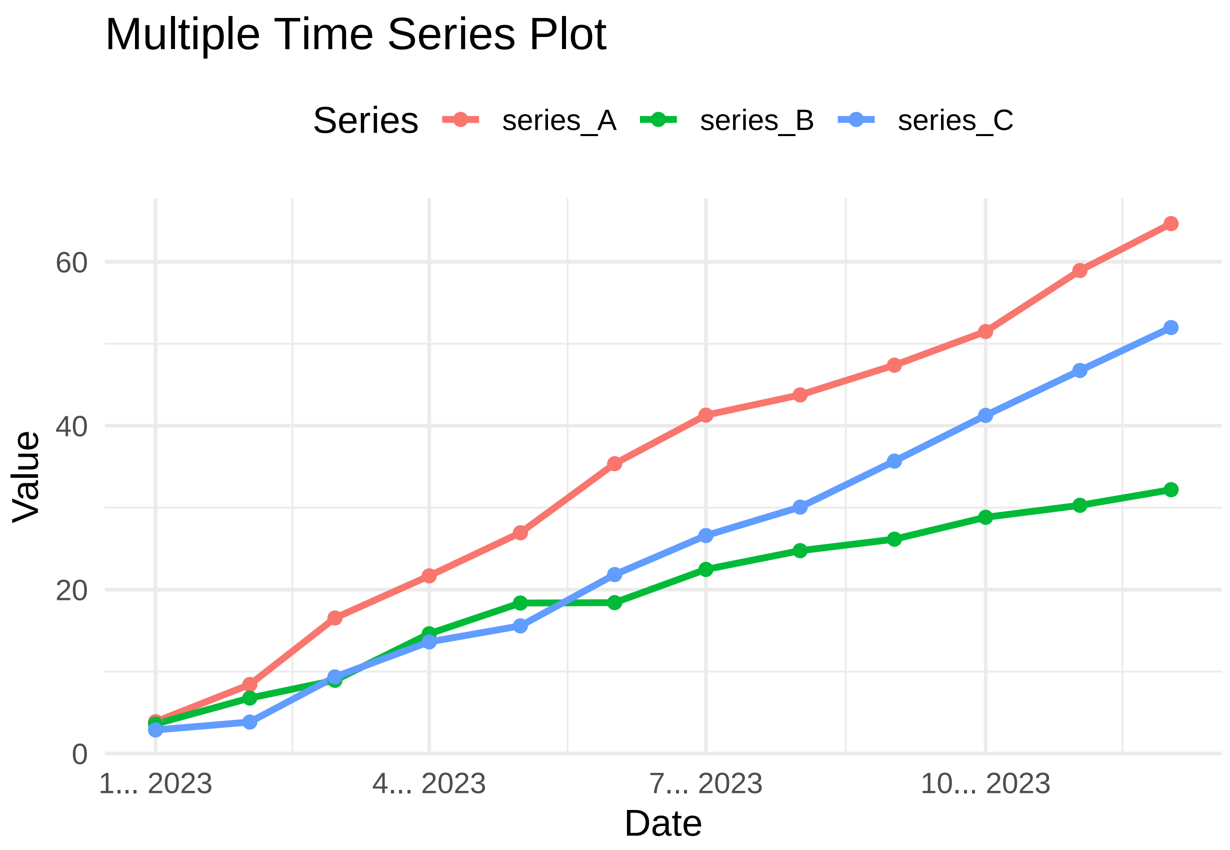
The data includes three time series. So we can use a line chart to plot them together. In this format of data, we call them wide format since they are combined together with a same date at each row. However, ggplot cannot recognize this type of data, we need to transform them to a long format.

```r
# Transform data into long format for ggplot
long_data <- time_series_data %>%
  pivot_longer(cols = -date, names_to = "series", values_to = "value")
long_data
```

```
## # A tibble: 36 x 3
##    date       series   value
##    <date>     <chr>    <dbl>
##  1 2023-01-01 series_A  3.88
##  2 2023-01-01 series_B  3.60
##  3 2023-01-01 series_C  2.87
##  4 2023-02-01 series_A  8.42
##  5 2023-02-01 series_B  6.77
##  6 2023-02-01 series_C  3.84
##  7 2023-03-01 series_A 16.5
##  8 2023-03-01 series_B  8.93
##  9 2023-03-01 series_C  9.35
## 10 2023-04-01 series_A 21.7
## # i 26 more rows
```

We add one more colum called "series", it's kind of a index of which series this observation is from. Based on this:

```r
# Plot with English x-axis labels
ggplot(long_data, aes(x = date, y = value, color = series)) +
  geom_line(size = 1.2) +
  geom_point(size = 2) +
  scale_x_date(date_labels = "%b %Y") +
  labs(
    title = "Multiple Time Series Plot",
    x = "Date",
    y = "Value",
    color = "Series"
  ) +
  theme_minimal(base_size = 14) +
  theme(legend.position = "top")
```



This example adds the data directly to the ggplot() function, that means for the following geom_ functions, by default we are using the same data, same mapping, same color settings. In the `aes()` function, x axis is the date, y axis is the value, and we set the color of the line/point is the series variable in the long format data. In the `theme()` function, we can adjust a lot of personalized preferences. Refer to: https://www.rdocumentation.org/packages/ggplot2/versions/3.5.0/topics/theme.

Time series plots can visualize the trends, like stock market and population.

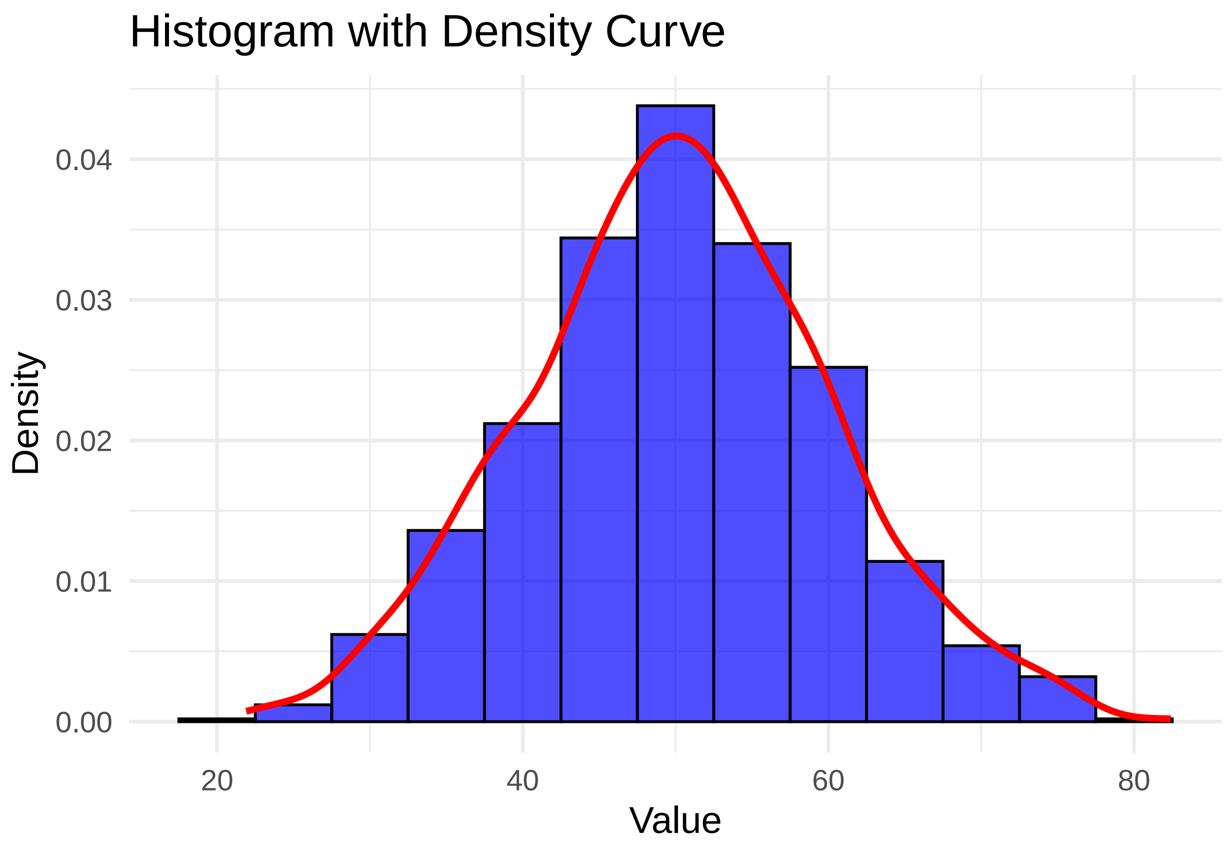
3.3 Histogram and Density

To visualize a distribution, or how the data are distributed, we can use a histogram in ggplot2. Below, we are using normal distribution samples as an example:

```r
library(ggplot2)

# Generate random samples from a normal distribution
set.seed(123)  # For reproducibility
data <- data.frame(value = rnorm(1000, mean = 50, sd = 10))

# Create histogram with a density line
ggplot(data, aes(x = value)) +
  geom_histogram(aes(y = ..density..), binwidth = 5, fill = "blue", color = "black", alpha = 0
  geom_density(color = "red", size = 1.2) +  # Add a smooth density line
  labs(
    title = "Histogram with Density Curve",
    x = "Value",
    y = "Density"
  ) +
  theme_minimal(base_size = 14)
```



We first drew a blue histogram, and then set a density curve above it.

3.4 Boxplot

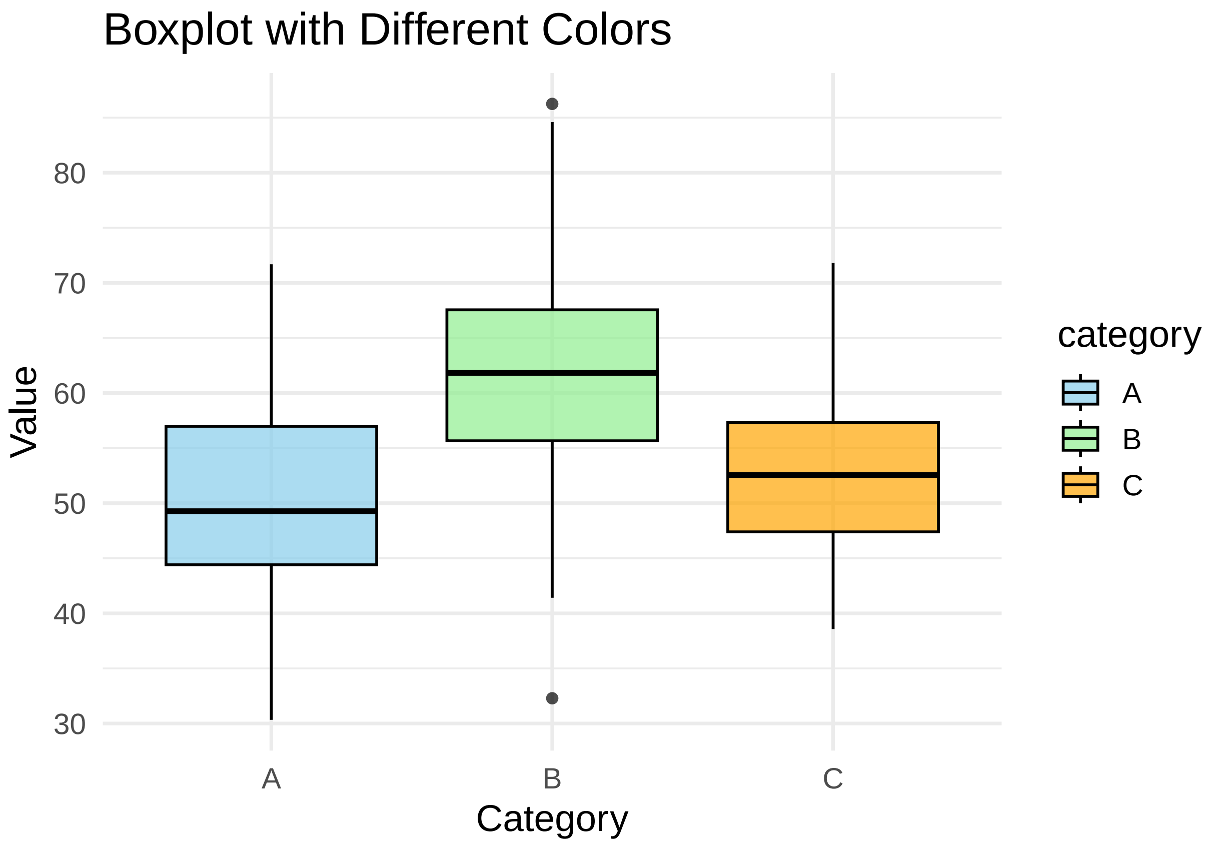
Boxplots are used when we have multiple groups to compare. We want to compare the distribution of them, for example:

```
library(ggplot2)

# Create sample data
set.seed(123)
data <- data.frame(
  category = rep(c("A", "B", "C"), each = 50),
  value = c(rnorm(50, mean = 50, sd = 10),
            rnorm(50, mean = 60, sd = 12),
            rnorm(50, mean = 55, sd = 8))
```

```r
)

# Boxplot with different colors for each category
ggplot(data, aes(x = category, y = value, fill = category)) +  # Color by category
  geom_boxplot(color = "black", alpha = 0.7) +  # Add black borders
  scale_fill_manual(values = c("A" = "skyblue", "B" = "lightgreen", "C" = "orange")) +  # Cust
  labs(title = "Boxplot with Different Colors", x = "Category", y = "Value") +
  theme_minimal(base_size = 14)
```



The top edge of the box describes the third quartile of the data, the thick black line in the middle describes the median of the data, and the bottom edge of the box is the first quartile. From the plot, we can see the group B has an overall higher value than the group A and C.

3.5 Heatmap and Contour Plot

Some spatial dataset, has a location together with a value. The location is usually given in a 2-d coordinate. In this case, we can use a heatmap or a contour plot.

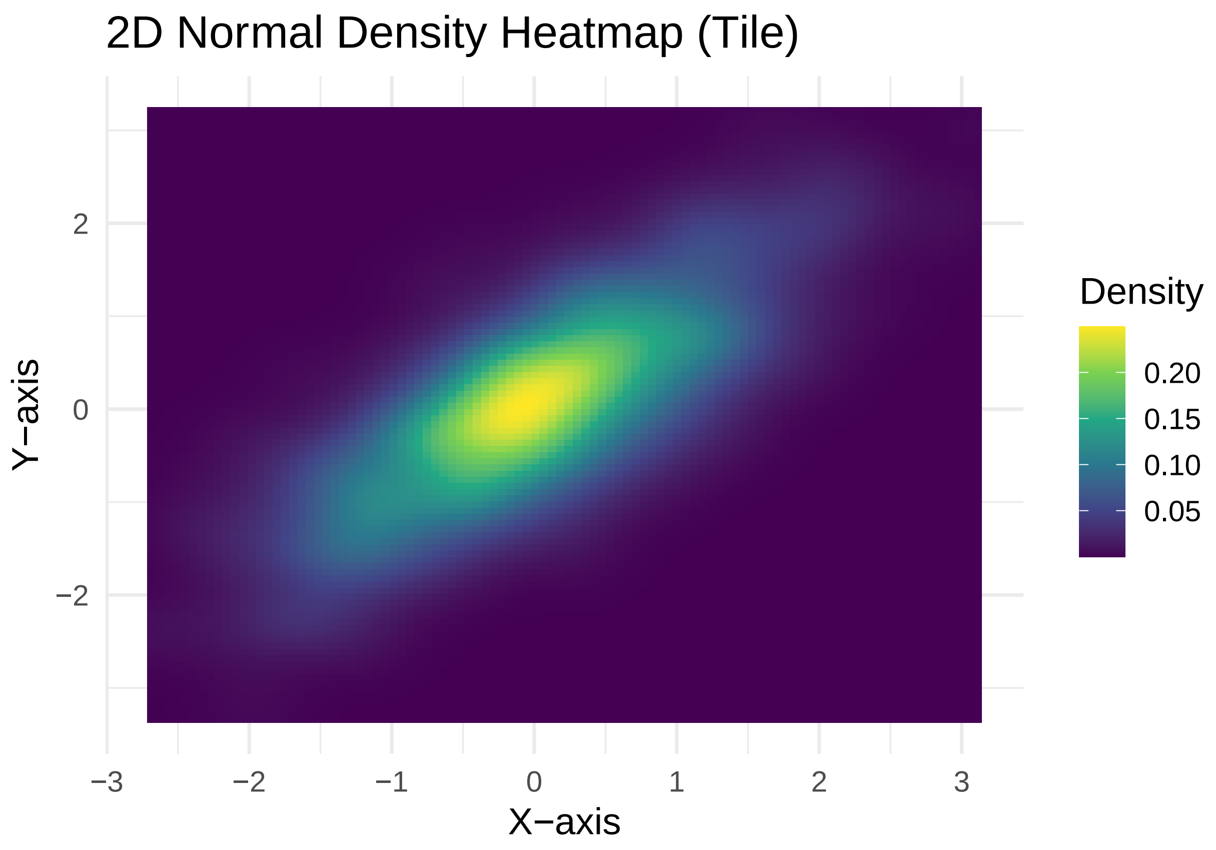
```r
library(ggplot2)
library(MASS)  # For multivariate normal distribution
```

```
##
## Attaching package: 'MASS'
## The following object is masked from 'package:dplyr':
##
##     select
```

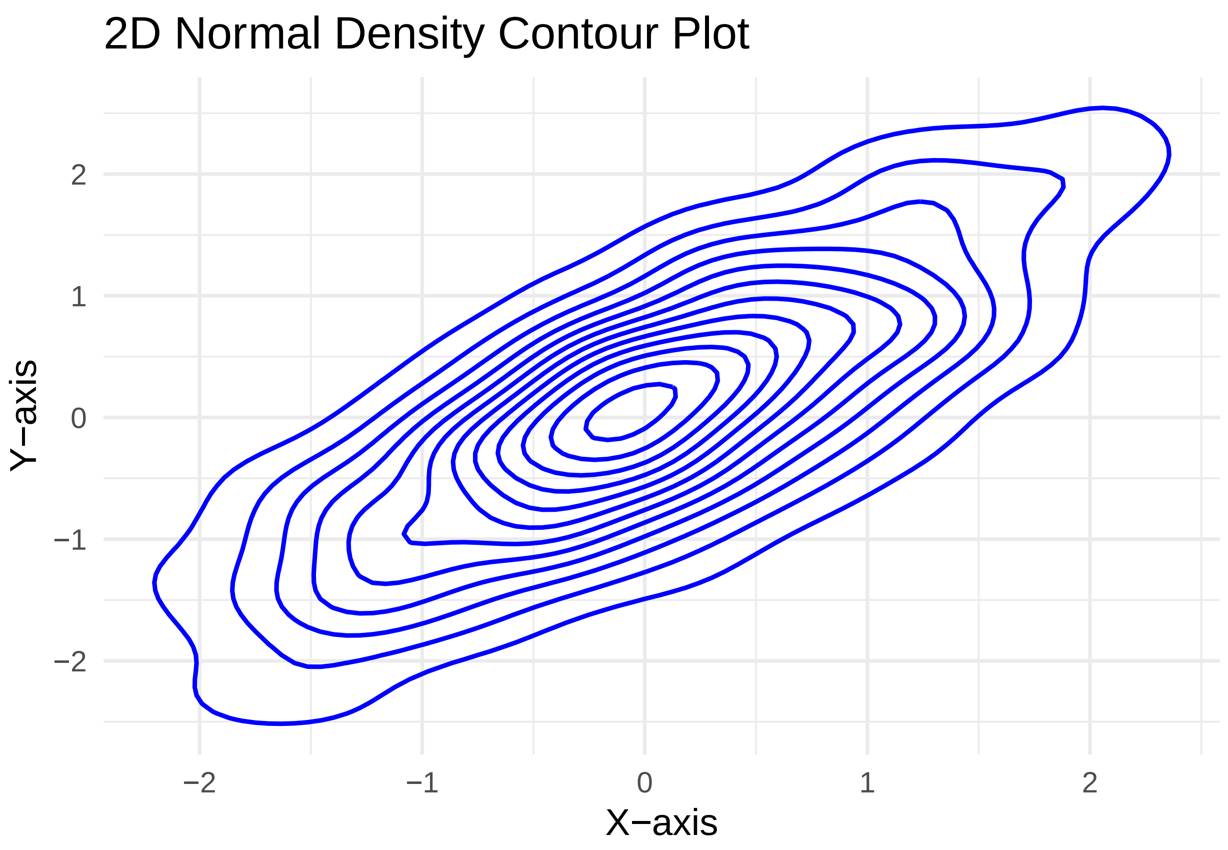
```r
# Generate 2D normal distribution data
set.seed(123)
mu <- c(0, 0)  # Mean vector
sigma <- matrix(c(1, 0.8, 0.8, 1), ncol = 2)  # Covariance matrix
samples <- mvrnorm(n = 1000, mu = mu, Sigma = sigma)  # Generate samples
data <- data.frame(x = samples[,1], y = samples[,2])  # Convert to dataframe

# Compute 2D density
density_data <- with(data, MASS::kde2d(x, y, n = 100))  # Estimate density
density_df <- data.frame(expand.grid(x = density_data$x, y = density_data$y),
                         density = as.vector(density_data$z))  # Convert to dataframe

# 2D heatmap with geom_tile()
ggplot(density_df, aes(x = x, y = y, fill = density)) +
  geom_tile() +  # Heatmap using tiles
  scale_fill_viridis_c() +  # Color scale
  labs(title = "2D Normal Density Heatmap (Tile)",
       x = "X-axis", y = "Y-axis", fill = "Density") +
  theme_minimal(base_size = 14)
```



```r
ggplot(density_df, aes(x = x, y = y, z = density)) +
  geom_contour(color = "blue", size = 0.8) +  # Contour lines
  labs(title = "2D Normal Density Contour Plot",
       x = "X-axis", y = "Y-axis") +
  theme_minimal(base_size = 14)
```



4 More..

ggplot2 is a very powerful package in R that can almost visualize any kind of data. Please go to https://ggplot2.tidyverse.org/articles/ggplot2.html for further information if in the future you want to do some fancy plots in R.

5 Ackowledgement

This teaching material is adapted from the previous material of this course made by Marcela Alfaro-Córdoba and Sheng Jiang.