Data Visualization Analysis

JB

2024-12-20

Table of contents

1 Introduction

This tutorial is a concise guide to creating data visualizations in R. From basic bar charts to advanced density plots, you'll learn practical techniques to analyze and present data effectively, making complex insights easy to understand.

---

2 Preparation

To get started, we'll use two essential R packages:

- `ggplot2`: To create visualizations in this tutorial
- `gcookbook`: To provide access to datasets used throughout the tutorial

These packages form the foundation of this tutorial, as most examples and datasets provided rely on them. Be sure to load them using the following commands:

```
library(ggplot2)
library(gcookbook)
```

---

3 Labels

Before we start with the analysis, there is an important function that will be used throughout this tutorial, which is the `labs()` funtion. The `labs()` function serve as a tool to label things inside the chart provided by `ggplot2`

```
labs(title='My Chart',
     x='x-axis label',
     y='y-axis label',
     captions='Chart Captions')
```

The following is an explanation to the code above:

- `title='My Chart'`= Add text to the title (`My Chart`) of the chart
- `x= 'x-axis label'`= Changes the label in the x - axis
- `y= 'y-axis label'`= Changes the label in the y - axis
- `captions='Chart Captions'`= Add captions (`Chart Captions`) to the bottom right of the chart

---

4 Bar chart

In this section, we will draw a bar chart using `pg_mean` dataset. The dataset has two columns: `group`, `weight`.

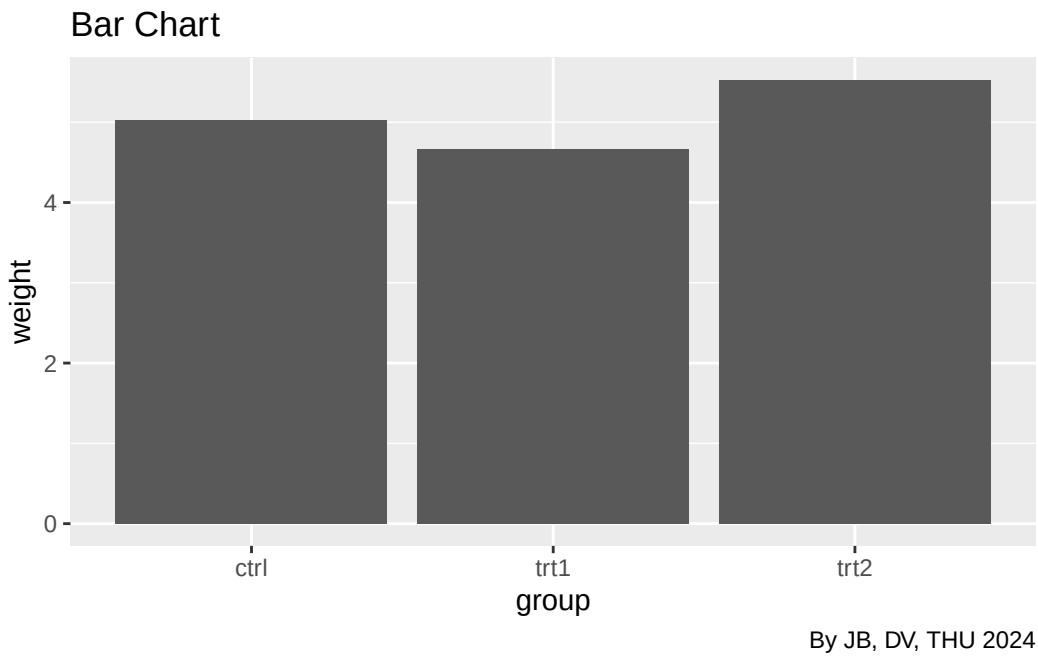
```
pg_mean
```

```
  group weight
1  ctrl  5.032
2  trt1  4.661
3  trt2  5.526
```

This dataset compares the weight across three groups:

- `ctrl`: Control group (baseline, weight = 5.032).
- `trt1`: Treatment 1 group (weight = 4.661).
- `trt2`: Treatment 2 group (weight = 5.526).

```r
ggplot(pg_mean, aes(x = group, y = weight)) +
  geom_col() +
  labs (title= "Bar Chart",
        caption = 'By JB, DV, THU 2024')
```



It initializes a ggplot with the dataset `pg_mean`.

`aes(x = group, y = weight)` specifies the aesthetics:

- `x = group`: Assign the `group` variable to the x-axis (categorical data, such as `ctrl, trt1, trt2`).
- `y = weight`: Assign the `weight` variable to the y-axis (numerical data).

`geom_col()`:

- Adds a column geometry to the plot.
- `geom_col()` creates bars where the height of each bar corresponds to the value of weight for each group.

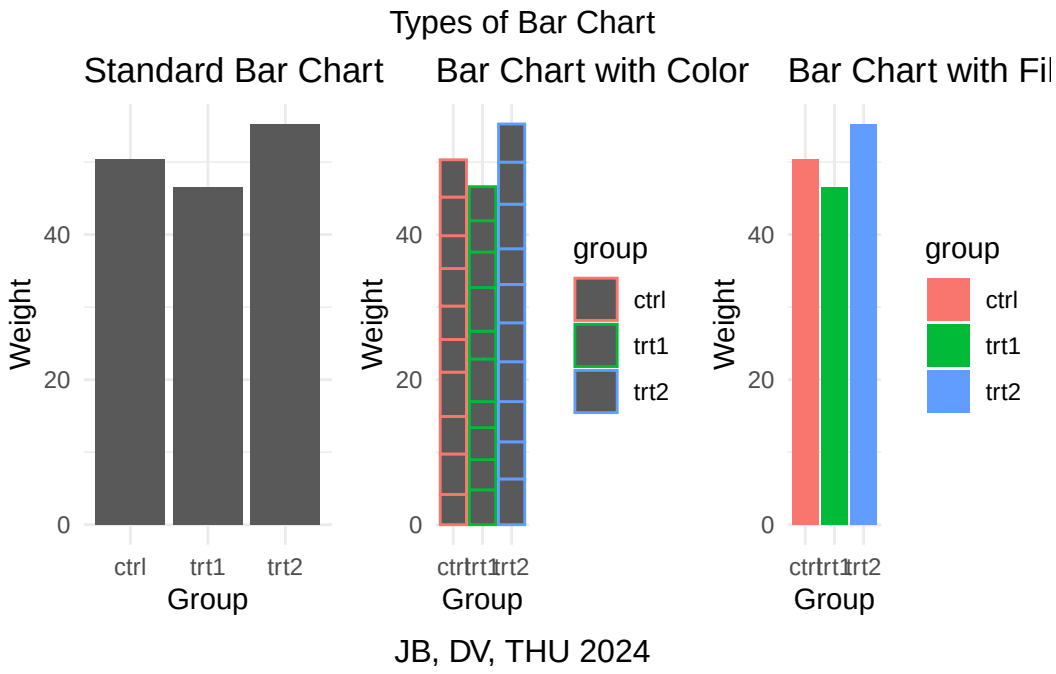
---

5 Bar chart with color

Adding colors to bar chart, there 2 types, one using `color` and other is `fill`:

- `color`: Changes the color of the outline of the bars, leaving the interior unfilled or using the default fill color.
- `fill`: Changes the color inside the bars, determining the interior fill color.

```
p1 <- ggplot(PlantGrowth, aes(x = group, y = weight)) +
  geom_col()+
  theme_minimal()+
  labs(title = 'Standard Bar Chart',
      x= 'Group',
      y= 'Weight')

p2 <- ggplot(PlantGrowth, aes(x = group, y = weight, color = group)) +
  geom_col()+
  theme_minimal()+
  labs(title = 'Bar Chart with Color',
      x= 'Group',
      y= 'Weight')

p3 <- ggplot(PlantGrowth, aes(x = group, y = weight, fill = group)) +
  geom_col()+
  theme_minimal()+
  labs(title = 'Bar Chart with Fill',
      x= 'Group',
      y= 'Weight')

grid.arrange(p1,p2,p3, ncol=3,top='Types of Bar Chart', bottom = 'JB, DV, THU 2024')
```



---

6 Line chart

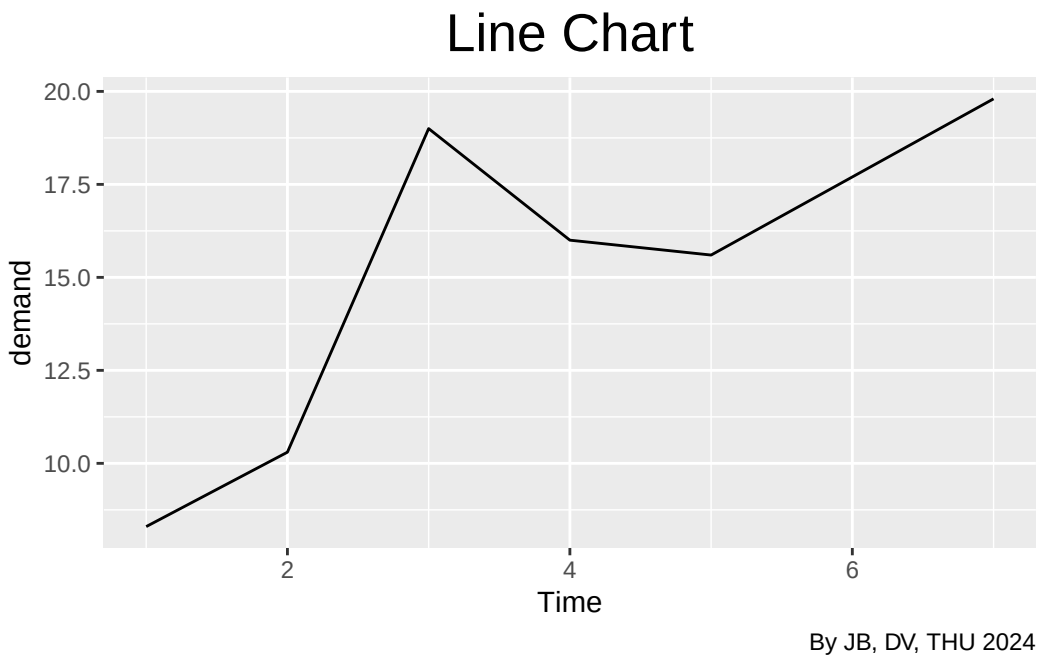
In this section, we will draw a line chart using `BOD`, The dataset has two columns: `Time` and `demmand`

```
BOD
```

```
  Time demand
1    1    8.3
2    2   10.3
3    3   19.0
4    4   16.0
5    5   15.6
6    7   19.8
```

In order to find the correlation between the Biochemical Oxygen Demand(`BOD`) over time, we can use Line chart to properly visualize the data.

```r
ggplot(BOD, aes (x = Time, y = demand)) +
  geom_line() +
  labs(title= "Line Chart",
       caption = 'By JB, DV, THU 2024') +
  theme(plot.title = element_text(hjust= 0.5, size = 20))
```



It initializes a ggplot with the dataset `BOD`

`aes(x = Time, y = demand)` specifies the aesthetics:

- `x = Time`: Assign the `Time` variable to the x-axis (numerical data, representing time in days).
- `y = demand`: Assign the `demand` variable to the y-axis (numerical data, representing biochemical oxygen demand).

`geom_line()`

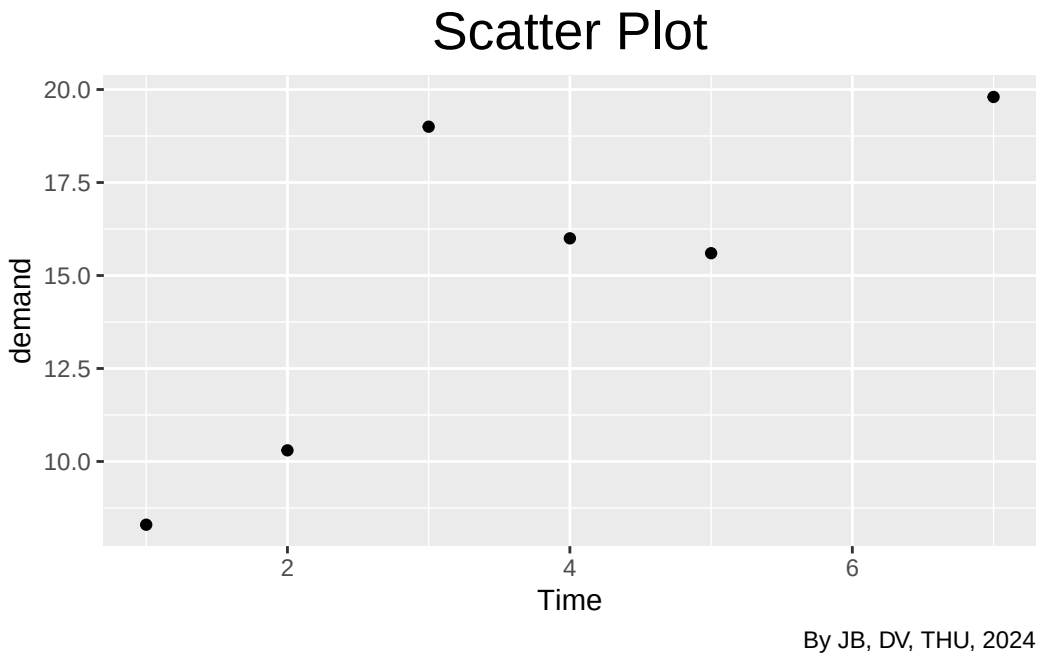
- Adds a line geometry to the plot.
- `geom_line()` connects the data points sequentially with straight lines, illustrating how oxygen demand changes over time.

---

7 Scatter Plot

Alternatively, we can use `geom_point` to also create a scatter plot for the `BOD` dataset

```
ggplot(BOD, aes (x = Time, y = demand)) +
  geom_point() +
  labs(title= "Scatter Plot",
       caption = 'By JB, DV, THU, 2024') +
  theme(plot.title = element_text(hjust= 0.5, size = 20))
```



`geom_point()`:

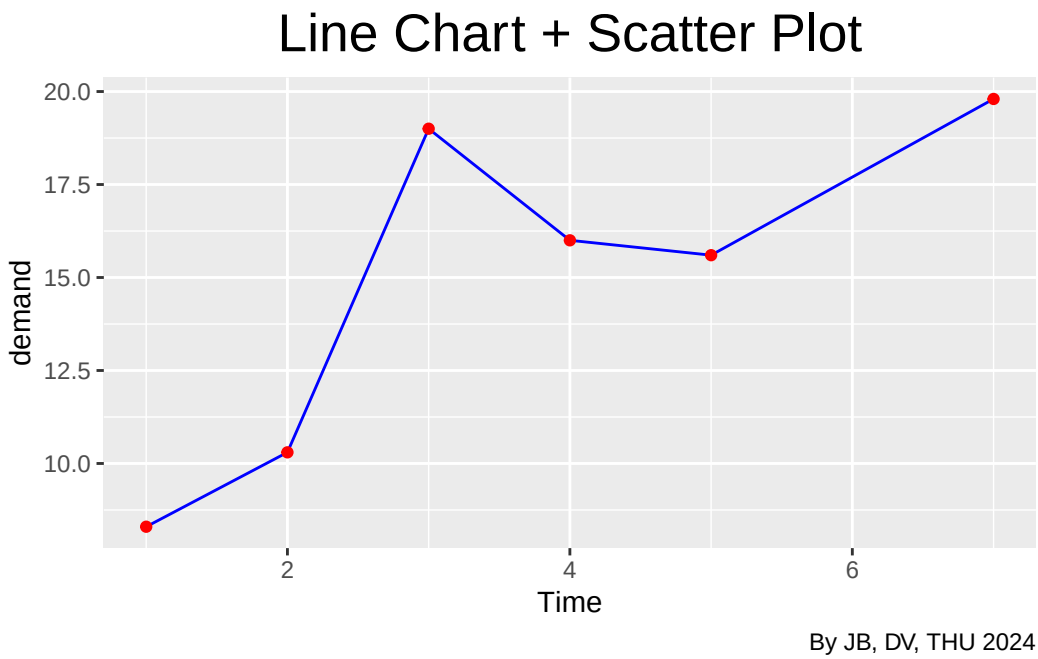
- Adds a point geometry to the plot.
- `geom_point()` plots individual data points, making it useful for visualizing the distribution of data or highlighting specific values in a dataset. _____

8 Combining Line Chart and Scatter Plot

In this section, you can combine `geom_point` and `geom_line` in a single plot to create a more informative and visually appealing chart.

While `geom_line` shows the trend or connection between data points, `geom_point` highlights the individual data values, making it easier to identify specific observations.

```r
ggplot(BOD, aes (x = Time, y = demand)) +
  geom_line(color = 'blue')+
  geom_point(color = 'red') +
  labs(title= "Line Chart + Scatter Plot",
       caption = 'By JB, DV, THU 2024') +
  theme(plot.title = element_text(hjust= 0.5, size = 20))
```



This combined chart will display a `blue` line representing the trend of `demand` over `Time` and `red` points to indicate each data point.

The result is a clearer representation of both the overall pattern and specific values in the dataset.

---

9 Multiple Variable Bar Chart

A `multivariable bar chart` is a type of `bar chart` that displays data for multiple `variables` simultaneously. It allows you to compare different `groups` side-by-side or stacked within each `category`, making it useful for visualizing relationships and patterns among variables.

In order to properly make the variable bar chart, necessary R packages:

- `dplyr`: Provides `%>%` for clean and readable code
- `tidyr`: Provides `pivot_longer` to reshape the dataset into a format suitable for `ggplot`

```
library(dplyr)
library(tidyr)
```

In this section, we will draw a multivariable bar chart using the `iris` dataset, The dataset have provided 3 different species (`setosa`, `versicolor`, `virginica`) which we can use to compare

```
iris
```

The `iris` dataset consists of the following columns:

- `Sepal.Length`: Length of the sepal (in cm).
- `Sepal.Width`: Width of the sepal (in cm).
- `Petal.Length`: Length of the petal (in cm).
- `Petal.Width`: Width of the petal (in cm).
- `Species`: The species of the flower, which can be one of three categories: `setosa`, `versicolor`, or `virginica`.

```
df <- iris

iris_long <- df %>% pivot_longer(cols=c('Sepal.Length', 'Sepal.Width'),
                                         names_to="Sizes",
                                         values_to='Count')
```

This reshapes the `iris` dataset to create a long-format dataset suitable for grouping by `Sizes`;

- `df <- iris`: Creates a temporary dataset to be modified without affecting the original dataset `iris`
- `pivot_longer`: A function in R is used to transform data from a wide format to a long format. It collapses multiple columns into two: one for the column names and one for their corresponding values.
- `%>%`: It takes the result of the expression on the left and passes it as the first argument to the function on the right.

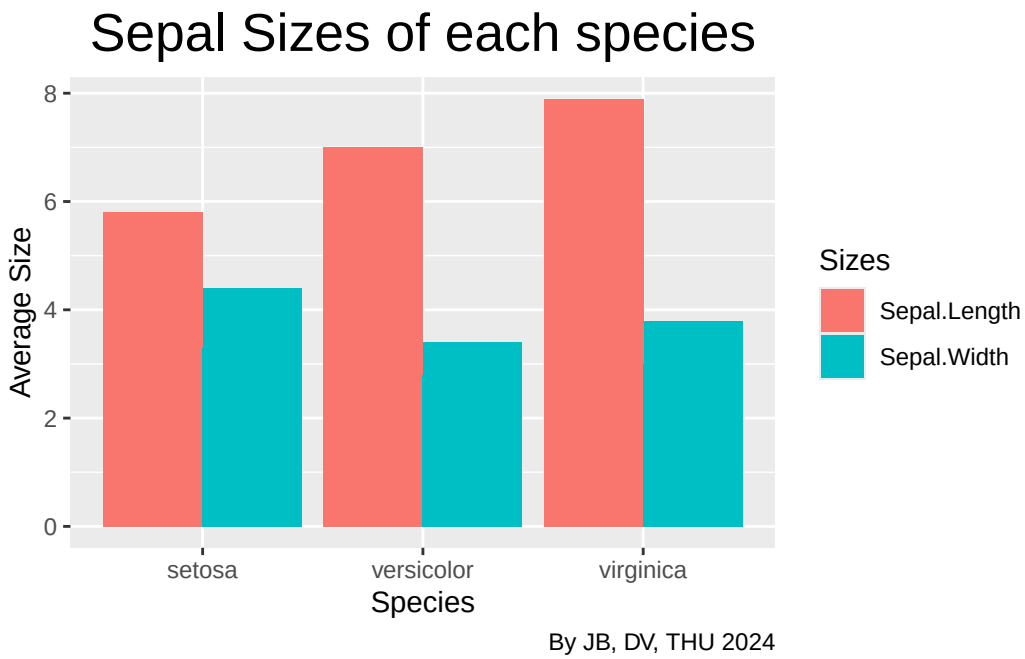
After `pivot_longer`

```
iris_long
```

```
# A tibble: 300 x 5
   Petal.Length Petal.Width Species Sizes        Count
          <dbl>       <dbl> <fct>   <chr>        <dbl>
 1          1.4         0.2 setosa  Sepal.Length   5.1
 2          1.4         0.2 setosa  Sepal.Width    3.5
 3          1.4         0.2 setosa  Sepal.Length   4.9
 4          1.4         0.2 setosa  Sepal.Width    3
 5          1.3         0.2 setosa  Sepal.Length   4.7
 6          1.3         0.2 setosa  Sepal.Width    3.2
 7          1.5         0.2 setosa  Sepal.Length   4.6
 8          1.5         0.2 setosa  Sepal.Width    3.1
 9          1.4         0.2 setosa  Sepal.Length   5
10          1.4         0.2 setosa  Sepal.Width    3.6
# i 290 more rows
```

As you can see from the `iris_long` dataset, after using `pivot_longer` the column `Sepal.Length` and `Sepal.Width` have been converted into `Sizes`, this way we can differentiate between `length` and `width` of the `Sepal` in a plot, instead of treating them as separate variables.

```
ggplot(iris_long, aes(x = as.factor(Species), y = Count, fill = Sizes)) +
  labs(title="Sepal Sizes of each species",
       x= 'Species',
       y= 'Average Size',
       caption = 'By JB, DV, THU 2024') +
  theme(plot.title = element_text(hjust= 0.5, size = 20)) +
  geom_bar(position = 'dodge', stat= 'identity')
```



It initalizes a ggplot with the dataset `iris`

`aes(x = as.factor(Species), y = Count)` specifies the aesthetics:

- `x = as.factor(Sepcies):` Assigns the `Species` variable to the x-axis. It is converted into a factor so each species (`Setosa`, `Versicolor`, and `Virginica`) is treated as a distinct category.
- `y = Count:` Assigns the `Count` variable (the values from `Sepal.Length` or `Sepal.Width`) to the y-axis.

---

10 Histogram

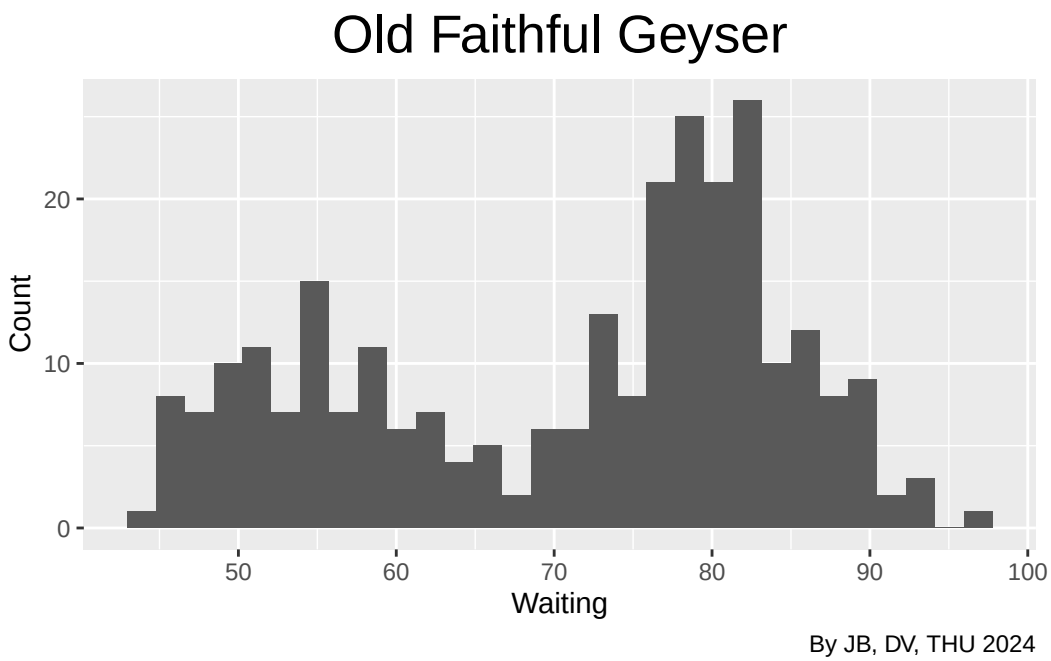
In this section, we will draw a histogram using the `faithful` dataset. The dataset has one numeric column: `waiting`.

```
faithful
```

This dataset contains measurements of the waiting times between eruptions of the Old Faithful geyser in Yellowstone National Park. The `waiting` column represents the waiting time (in minutes) between eruptions.

```r
ggplot(faithful, aes (x=waiting))+
  geom_histogram()+
  labs(title = "Old Faithful Geyser",
       x = 'Waiting',
       y= 'Count',
       caption= "By JB, DV, THU 2024") +
  theme(plot.title = element_text(hjust= 0.5, size = 20))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



It initializes a ggplot with the dataset `faithful`.

- `aes(x = waiting)` Assign the aesthetic of waiting variable to the x-axis (numerical data representing the waiting times between eruptions).
- `geom_histogram()` is a function provided by `ggplot2` to create histogram.

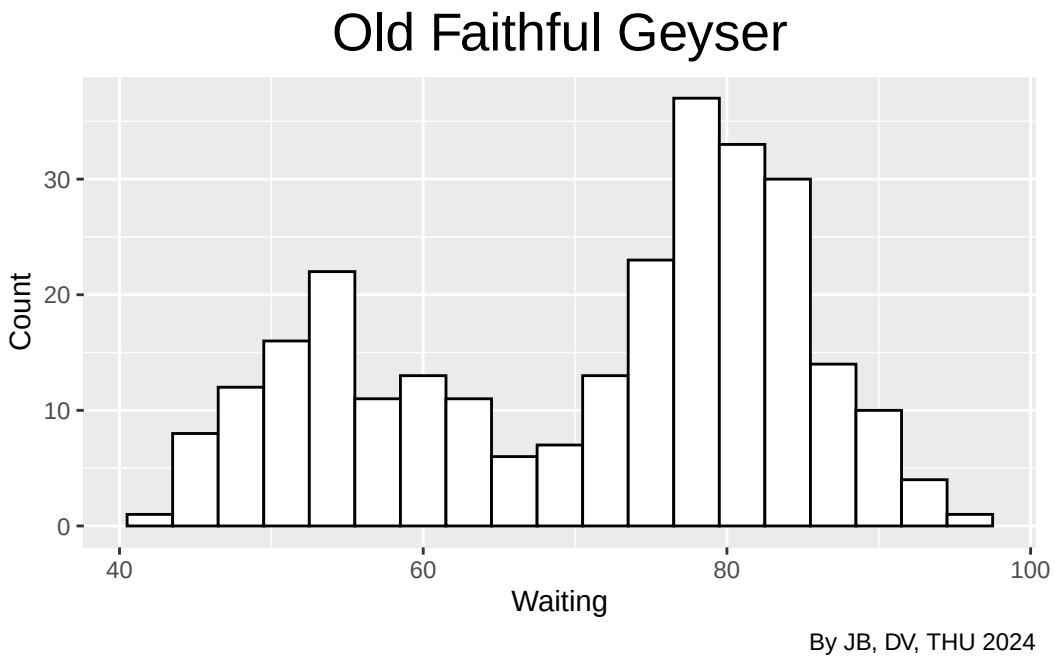
---

11 Changing the Appearance of the Histogram

Changing the binsize of the histogram:

`binwidth` specifies the width of the bins (intervals) to group the waiting data. In this case, each bin represents 3 minutes of waiting time.

```
ggplot(faithful, aes (x=waiting))+
  geom_histogram(binwidth=3, fill='white', color='black')+
  labs(title = "Old Faithful Geyser",
       x = 'Waiting',
       y= 'Count',
       caption= "By JB, DV, THU 2024") +
  theme(plot.title = element_text(hjust= 0.5, size = 20))
```



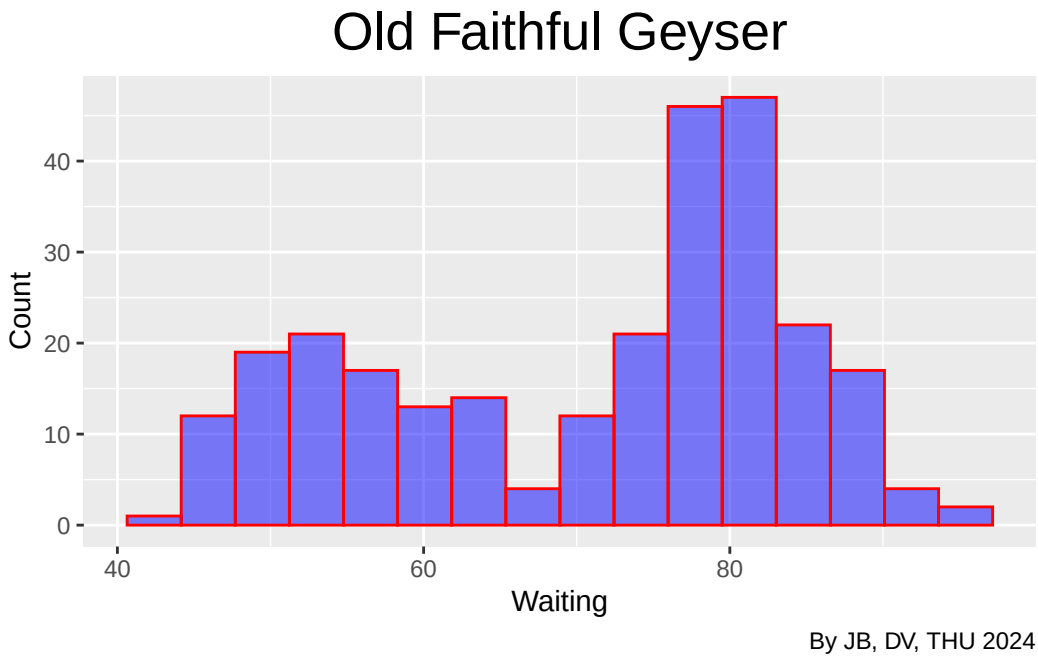
Finding the perfect binsize:

The right bin size strikes a balance between showing enough detail to reveal the distribution and keeping the plot simple and interpretable. It helps in accurately representing the data's shape and ensures meaningful comparisons.

To find the ideal `binsize`, we need to find the `range`, which is `Max` - `Min` divided by the desired the ideal number of `bin`

- `range(faithful$waiting)` : calculates the minimum and maximum values of the `waiting` column in the `faithful` dataset.
- `diff(range(faithful$waiting))/15`: calculates the difference between the maximum and minimum values of the range and `/15` the division will give the ideal bin width (or bin size) for creating, in this case, 15 bars in the histogram.

```r
binsize <- diff(range(faithful$waiting))/15

ggplot(faithful, aes (x=waiting))+
  geom_histogram(binwidth=binsize, fill='blue', color='red', alpha= .5)+
  labs(title = "Old Faithful Geyser",
       x = 'Waiting',
       y= 'Count',
       caption= "By JB, DV, THU 2024") +
  theme(plot.title = element_text(hjust= 0.5, size = 20))
```



Adding colors and Changing transparency:

- `alpha`: Changes the transparency of the chart
- `fill`: Changes the color inside the chart
- `color`: Changes the outline of the chart

---

12 Facets

In this section, we will discuss on how to display multiple chart using `facets`

Firstly, necessary R packages need to be loaded:

- `MASS`: Provides the appropriate dataset for this section, please note that the `MASS` package is not necessarily provided in R Studio, you can run `install.packages(MASS)` in order to properly load the package
- `dplyr`: Provides `recode_factor` function to change data types inside the dataset

```
library(MASS)
library(dplyr)
```

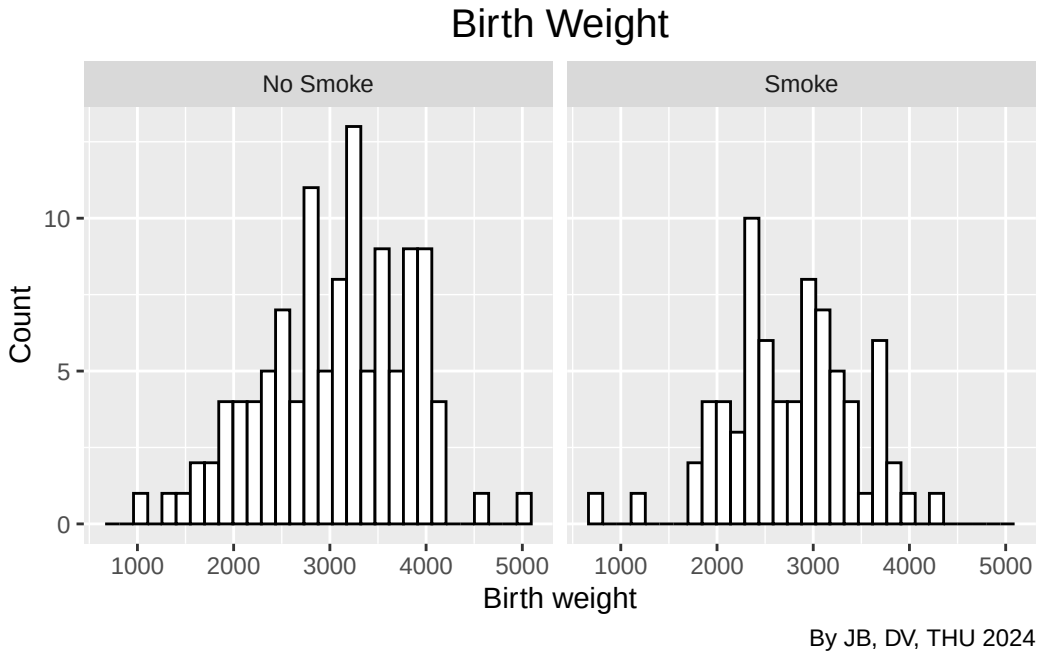
Once loaded, we are now set to use the `birthwt` dataset, which contains information about the birth weights of infants and various maternal and pregnancy-related factors.

This dataset includes the following variables:

- `low`: A binary factor indicating whether the birth weight is low (< 2500 grams).
- `lwt`: The weight of the mother in pounds at the last prenatal visit.
- `race`: The race of the mother.
- `Levels`: 0 `(non-smoker)`, 1 `(smoker)`.
- `ptl`: The `number` of previous premature labors (labors before 37 weeks of gestation).
- `ht`: Whether the mother has a history of hypertension.
- `ui`: Whether the mother has uterine irritability.
- `birweight`: The `weight` of the infant at birth in grams

```
birthwt_mod <- birthwt
birthwt_mod$smoke <- recode_factor(birthwt_mod$smoke, '0' = 'No Smoke', '1' = 'Smoke')

ggplot(birthwt_mod, aes(x = bwt)) +
  geom_histogram(fill='white', color='black')+
  facet_grid(. ~ smoke) +
  labs(title="Birth Weight",
       x= 'Birth weight',
       y = 'Count',
       caption = 'By JB, DV, THU 2024') +
  theme(plot.title = element_text(hjust= 0.5, size = 15))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



- `birthwt_mod <- birthwt`: Creates temporary dataset without altering the original dataset, `birthwt`
- `recode_factor`: A function to recode the levels of a factor variable. In this case, it is used to change the values of the `smoke` variable from numeric values (`0` and `1`) to more meaningful labels (`No Smoke` and `Smoke`).
- `facet_grid()`: A function in `ggplot2` that is used to create faceted plots. Faceting allows you to split a plot into multiple subplots (or panels), based on the values of one or more variables.

`facet_grid( . ~ smoke)` is used to facet the plot by the `smoke` variable while leaving the rows empty (i.e., no faceting by rows);

- `.` (`dot`): This represents no faceting by rows. It tells `ggplot2` to create only columns, without splitting the data into rows based on any variable.
- `smoke`: This is the categorical variable used to facet the data along the columns. Since the smoke variable now has two levels (`No Smoke` and `Smoke`), `facet_grid(. ~ smoke)` will create two columns: one for the `No Smoke` group and one for the `Smoke` group.

---

13 Grid Arrange

Alternatively to `facet_grid()` which is provided by `ggplot2`, we also can use `grid.arrange()` which is provided by the `gridExtra` package

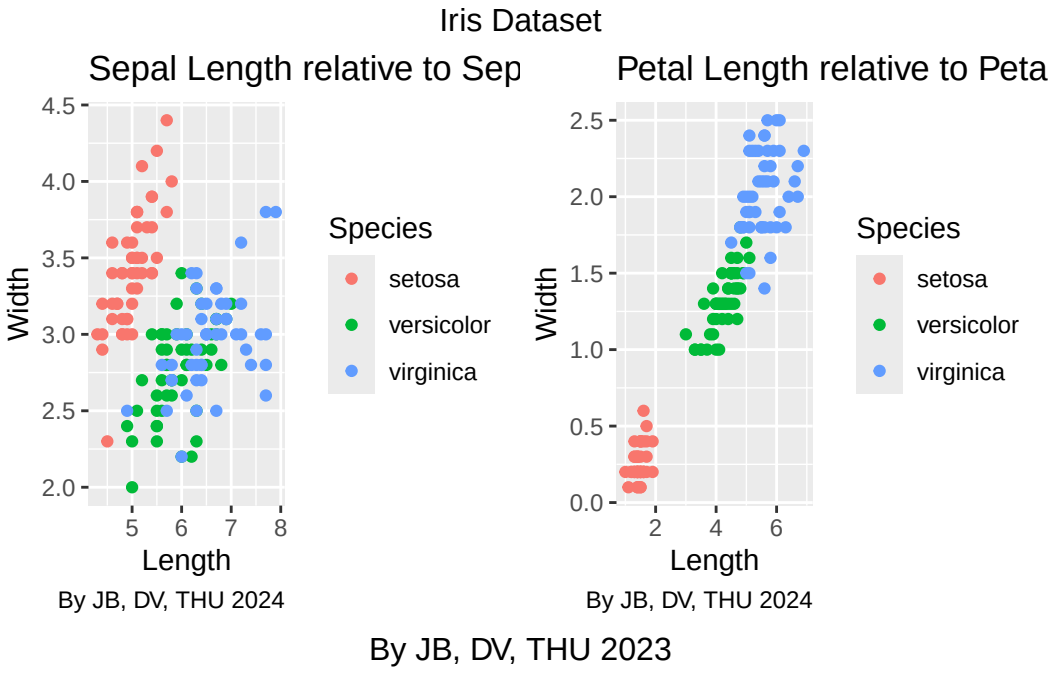
```
library(gridExtra)
```

To use the `grid.arrange` function, we have to have our chart. In this case, we will be using the `iris` dataset

```
Sepal <- ggplot(iris, aes(x=Sepal.Length ,y=Sepal.Width, color=Species ))+
  geom_point()+
  labs(title='Sepal Length relative to Sepal Width',
       x='Length',
       y='Width',
       captions='By JB, DV, THU 2024')

Petal <- ggplot(iris, aes(x=Petal.Length ,y=Petal.Width, color=Species ))+
  geom_point()+
  labs(title='Petal Length relative to Petal Width',
       x='Length',
       y='Width',
       captions='By JB, DV, THU 2024')
```

Once we have created the chart, we can create a temporary dataset `Sepal` and `Petal` to store and be used for the next step

```
grid.arrange(Sepal, Petal, ncol=2, top= 'Iris Dataset', bottom='By JB, DV, THU 2023')
```



Using the `grid.arrange` function, we are able to display multiple chart from the same or different dataset - `ncol`: To determine the amount of column needed in the grid - `top`: To add `text` on the `top` section of the grid - `bottom`: To add `text` on the `bottom` section of the grid

---

14 Data Manipulation

In this section, we will discuss on how Data are analyzed and processed through Data Manipulation

Preparation Before performing data manipulation, the `dplyr` package is loaded to utilize its functions, and a dataset (`student_scores.csv`) is imported into R as a data frame called `score`.

```
library(dplyr)
score <- read.csv('student_scores.csv')
```

- Data Filtering In order to filter a data, we can simply use `filter()`

```
score %>%
  filter(English > 60 & Math > 70 & class_year==2)
```

```
  English Math Science class_year
1      88   76      66          2
2      70   96      89          2
3      93   96      67          2
```

The code above filters the dataset to retain only the rows where:

- `English > 60`: The English score is greater than 60.
- `Math > 70`: The Math score is greater than 70.
- `class_year == 2`: The student is in the second class year.

This allows us to focus on specific subsets of the data that meet certain conditions.

- Calculating Average and Add new Column Here we are going to calculate the average and create new column named `avg`

```
score_avg <- score %>%
  mutate(avg = (English + Math + Science)/3)
```

In order to add a new column, we can use the following function:

- `mutate()`: To add a new column to the dataset `score_avg`
- `avg=`: The name of the new column
- `(English + Math + Science)/3)`: The line to calculate the average score across three subjects: `English`, `Math`, and `Science`.

The resulting data frame, `score_avg`, includes all the original columns plus the new `avg` column, making it easier to analyze overall student performance.

- Calculating Group Average To further our skills in Data Manipulation, we can group the data by variables and calculate group average

```
score_avg %>%
  group_by(class_year) %>%
  summarize(avg_English = mean(English),
            avg_Math = mean(Math))
```

```
# A tibble: 3 x 3
  class_year avg_English avg_Math
       <int>       <dbl>    <dbl>
1          1        72.2     69.1
2          2        84.5     84.2
3          3        72.9     75.5
```

This code groups the data by the `class_year` variable and calculates:

- `avg_English`: The average English score for each class year.
- `avg_Math`: The average Math score for each class year.
- `group_by`: Organizes the data into groups based on `class_year`
- `summarize`: Reduces a data frame to a summary of just one vector or value.
- `mean()`: Calculates the average of the variable

---

15 Correlation chart

In this section, we are going to make a correlation chart. The dataset that we are going to use is `mpg`.

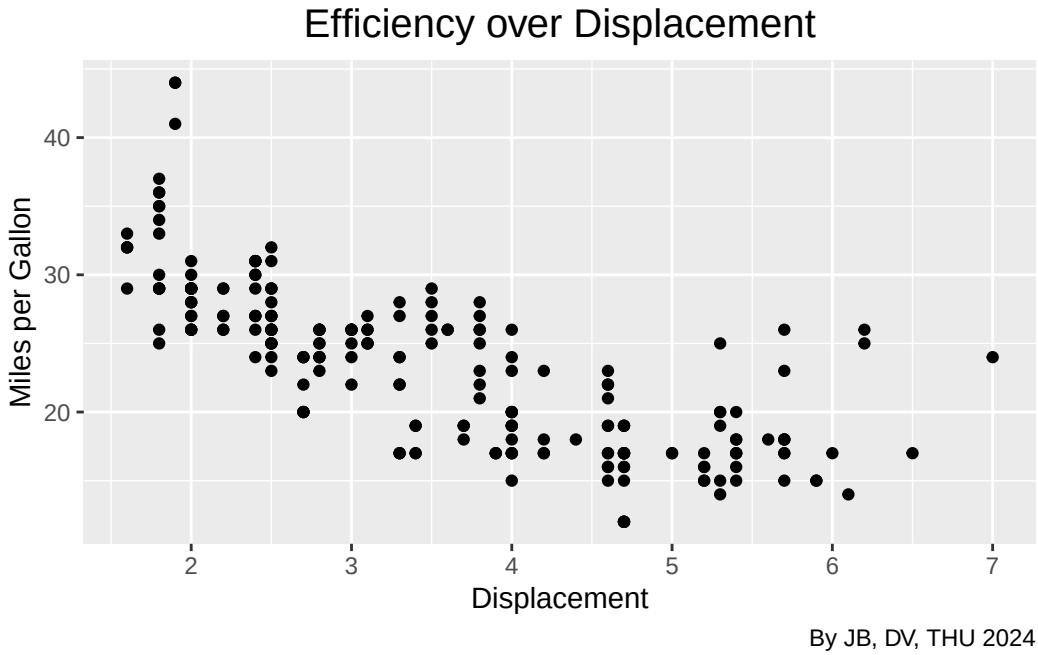
```
mpg
```

The `mpg` dataset is a Fuel Economy data from 1999 to 2008 for 38 popular models of cars provided in the `ggplot2` package.

This dataset includes the following variables:

- `manufacturer`: manufacturer name
- `model`: model name
- `displ`: engine displacement, in litres
- `year`: year of manufacture
- `cyl`: number of cylinders
- `trans`: type of transmission
- `drv`: the type of drive train, where f = front-wheel drive, r = rear wheel drive, 4 = 4wd
- `cty`: city miles per gallon
- `hwy`: highway miles per gallon
- `fl`: fuel type
- `class`: "type" of car

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  labs(title='Efficiency over Displacement',
       x='Displacement',
       y='Miles per Gallon',
       captions='By JB, DV, THU 2024')+
  theme(plot.title = element_text(hjust= 0.5, size = 15))
```



It initializes a ggplot with the dataset `mpg`.

`aes(x = displ, y = hwy)` specifies the aesthetics:

- `x = displ`: Assign the engine displacement variable to the x-axis (numerical data).
- `y = hwy`: Assign the highway miles per gallon variable to the y-axis (numerical data).

`geom_point()`:

- Adds points to the plot to represent individual observations in the dataset.
- Each point corresponds to a vehicle, plotted based on its engine displacement and highway efficiency.

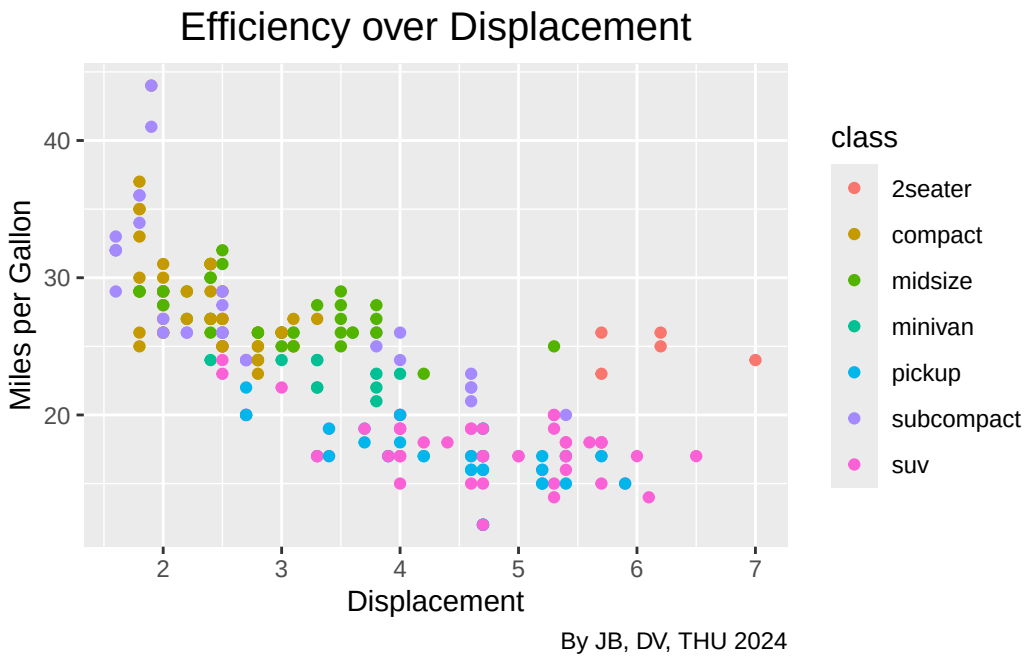
---

16 Correlation chart: Color by group

From the previous sectionn, we are able to plot a correlation chart between the Engine Displacement `displ` and the Mileage of the cars `hwy`.

To make the chart more informative, we can color the points by vehicle class (class), allowing us to observe how different types of vehicles vary in the relationship between Engine Displacement `displ` and the Mileage of the cars `hwy`.

```r
ggplot(mpg, aes(x = displ, y = hwy, color=class)) +
  geom_point() +
  labs(title='Efficiency over Displacement',
       x='Displacement',
       y='Miles per Gallon',
       captions='By JB, DV, THU 2024')+
  theme(plot.title = element_text(hjust= 0.5, size = 15))
```



It initializes a ggplot with the dataset `mpg`.

`aes(x = displ, y = hwy, color = class)` specifies the aesthetics:

- `x = displ`: Assign the engine displacement variable to the x-axis.
- `y = hwy`: Assign the highway miles per gallon variable to the y-axis.
- `color = class`: Assign the vehicle class to the color aesthetic, differentiating points by their vehicle type.

`geom_point()`:

- Adds points to the plot, with colors representing different vehicle classes.
- This chart provides additional insights by showing how vehicle type influences the relationship between engine displacement and fuel efficiency.

---

17 Multigroup histogram

In this section, we are creating a multigroup histogram to visualize the distribution of two variables on the same plot. This is done by overlaying two histograms with different datasets, allowing us to compare their distributions.

```
# Sample data: Two variables with similar distributions
Variable1 <- rnorm(1000)
Variable2 <- rnorm(1000, mean = 2)
```

Here, we generate two variables:

- `Variable1:` 1000 random values drawn from a normal distribution with a mean of 0 and standard deviation of 1 (`rnorm(1000)`).
- `Variable2`: 1000 random values drawn from a normal distribution but with a mean of 2 (`rnorm(1000, mean = 2)`).

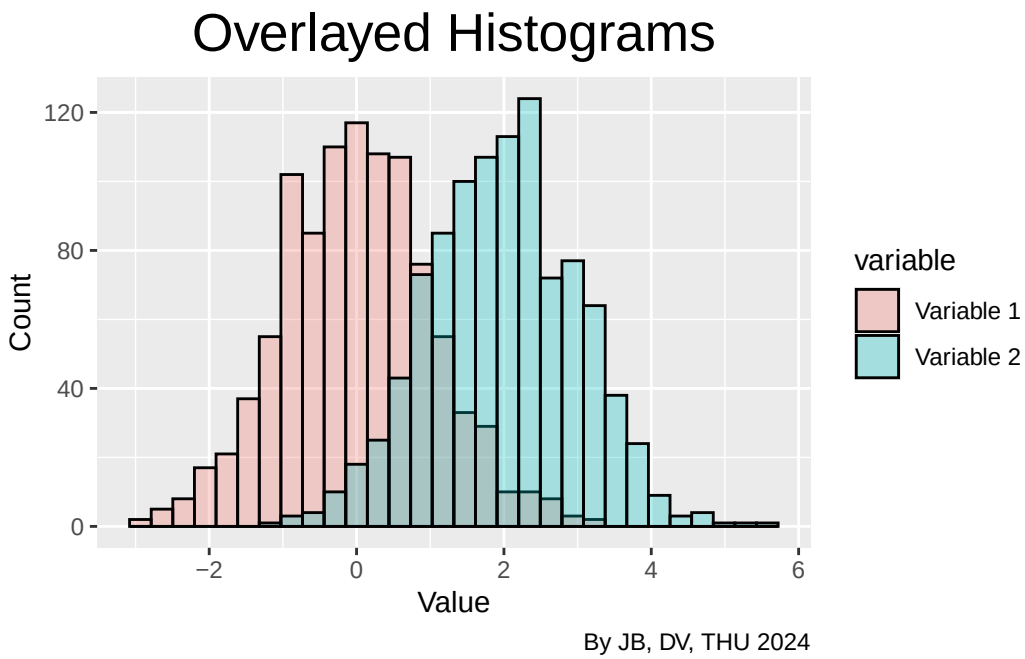
These two variables have similar distributions but are centered at different means, allowing us to visualize how they compare.

```
# Combine the data into a data frame
df <- data.frame(
  value = c(Variable1, Variable2),
  variable = rep(c('Variable 1', 'Variable 2'), each = 1000))
```

We then combine `Variable1` and `Variable2` into a single data frame `df`:

- The `value` column contains all the data points from both `Variable1` and `Variable2`.
- The variable column is used to label which variable the data point belongs to, with "`Variable 1`" assigned to the first 1000 points (from `Variable1`), and "`Variable 2`" assigned to the next 1000 points (from `Variable2`).

```
# Create the plot with overlapping histograms
ggplot(df, aes(x = value, fill = variable)) +
  geom_histogram(position = 'identity', alpha = 0.3, bins = 30, color = 'black') +
  labs(title = 'Overlayed Histograms',
       x = 'Value',
       y = 'Count',
       legend.title='Variables',
       captions='By JB, DV, THU 2024') +
  theme(plot.title = element_text(hjust= 0.5, size = 20))
```



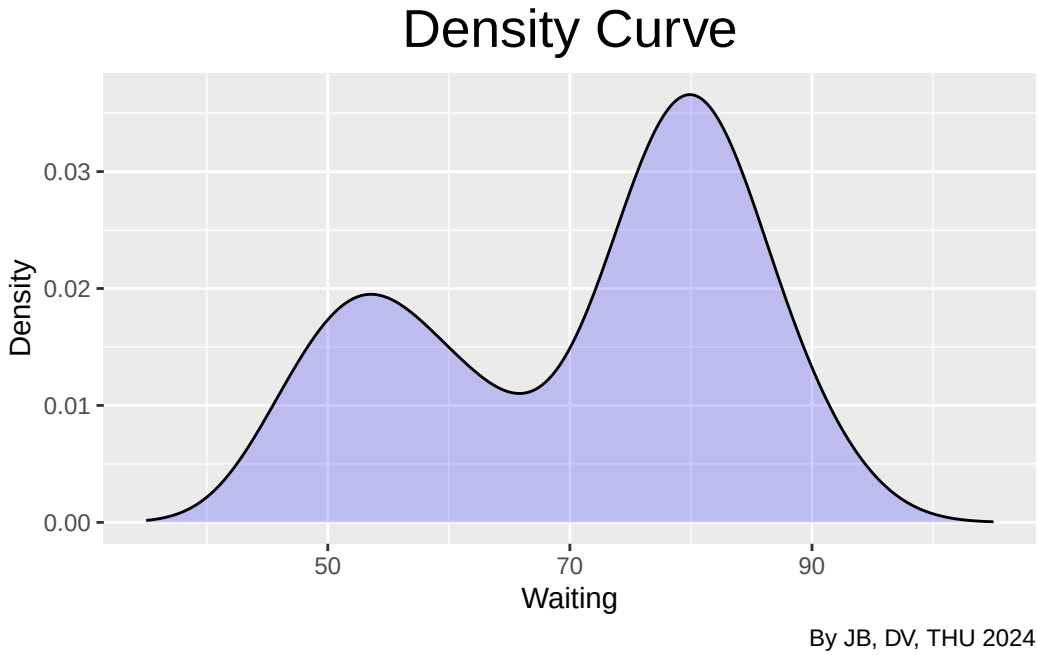
By using `position = identity` we are able to make the histograms to overlap each other rather than stacking them. Hence producing a multigroup histogram

---

18 Density chart

In this section, we will use the `faithful` dataset to visualize the distribution of the `waiting` variable using a density curve.

```
ggplot(faithful, aes(x=waiting)) +
   geom_density(fill = "blue", alpha = .2) +
  xlim(35,105) +
  labs(title="Density Curve",
       x= 'Waiting',
       y='Density',
       caption = 'By JB, DV, THU 2024') +
  theme(plot.title = element_text(hjust= 0.5, size = 20))
```



Unlike the other chart, we can see that a Density curves does not include a `y-axis`, this is because density curve displays a smooth estimate of the data's distribution, offering insights into its shape and spread.

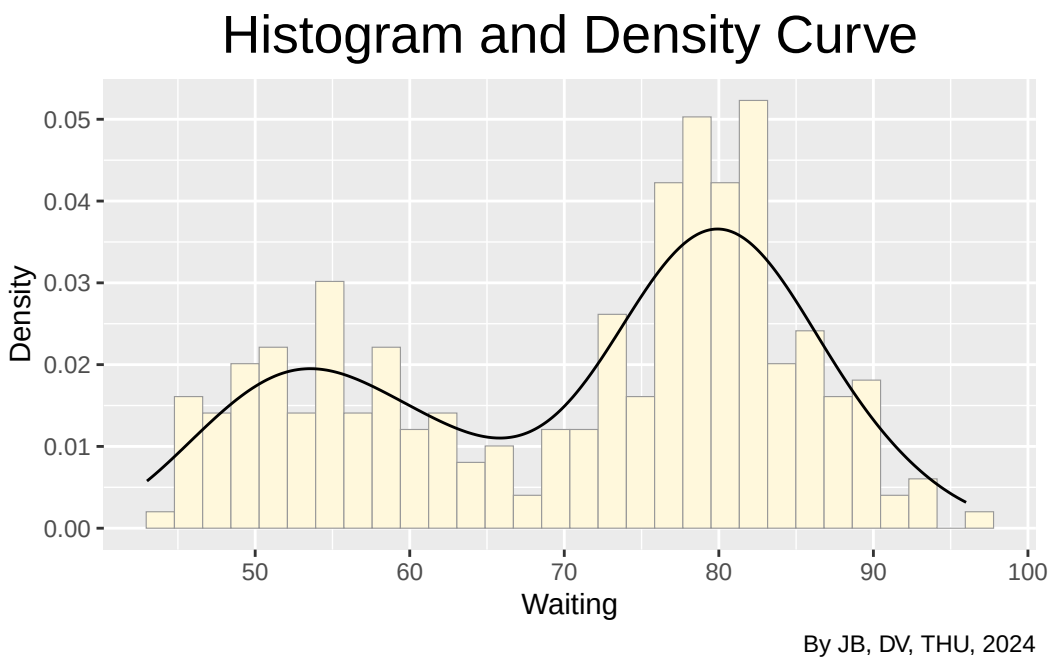
- `ggplot(faithful, aes(x = waiting))`: Initializes a ggplot with the `faithful` dataset, mapping the waiting variable to the x-axis.
- `geom_density(fill = "blue", alpha = .2)`: Adds a density curve to the plot, with a blue fill and 20% transparency for better visualization.
- `xlim(35, 105)`: Limits the x-axis to a range between 35 and 105. _____

19 Histogram and Density chart

After learning the histogram and density chart, we can combine a histogram and a density curve for better visualization. The histogram shows the raw frequency of observations, while the density curve overlays this with a smooth approximation.

```
ggplot(faithful, aes(x=waiting, y= ..density..)) +
  geom_histogram(fill = "cornsilk", color = "grey60", size = .2) +
  geom_density()+
  labs(title="Histogram and Density Curve",
       x= 'Waiting',
       y='Density',
```

```
       caption = 'By JB, DV, THU, 2024') +
  theme(plot.title = element_text(hjust= 0.5, size = 20))
```



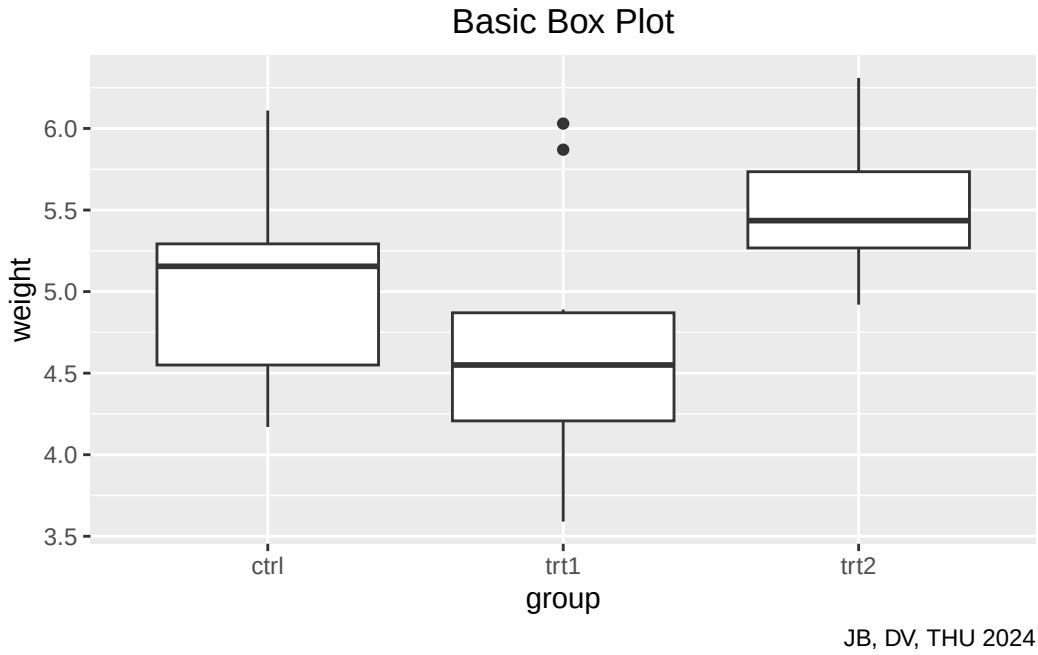
Similarly when combining multiple geometries (`geom_`) we can add `geom_histogram` before `geom_density` in order for the density curve to be displayed in front of histogram

- `aes(x = waiting, y = ..density..)`: Maps waiting to the `x-axis` and ensures the `y-axis` represents the density of data points.
- `geom_histogram(fill = "cornsilk", color = "grey60", size = .2)`: Adds a histogram with a cornsilk fill, grey outlines, and thin bars.

20 Box plot

In this section, we create a box plot to visualize the distribution of `weight` across different group categories in the `PlantGrowth` dataset.

```
ggplot(PlantGrowth, aes(x = group, y = weight)) +
  geom_boxplot() +
  labs(title = 'Basic Box Plot', caption = 'JB, DV, THU 2024') +
  theme(plot.title = element_text(hjust=0.5))
```



It initializes a ggplot with the dataset `PlantGrowth`.

`aes(x = group, y = weight)` specifies the aesthetics:

- `x = group`: Assign the experimental treatment.
- `y = weight`: Assign the weight of the plant

`geom_boxplot()` is used to create the box plot, displaying the summary statistics of the weight variable for each group.

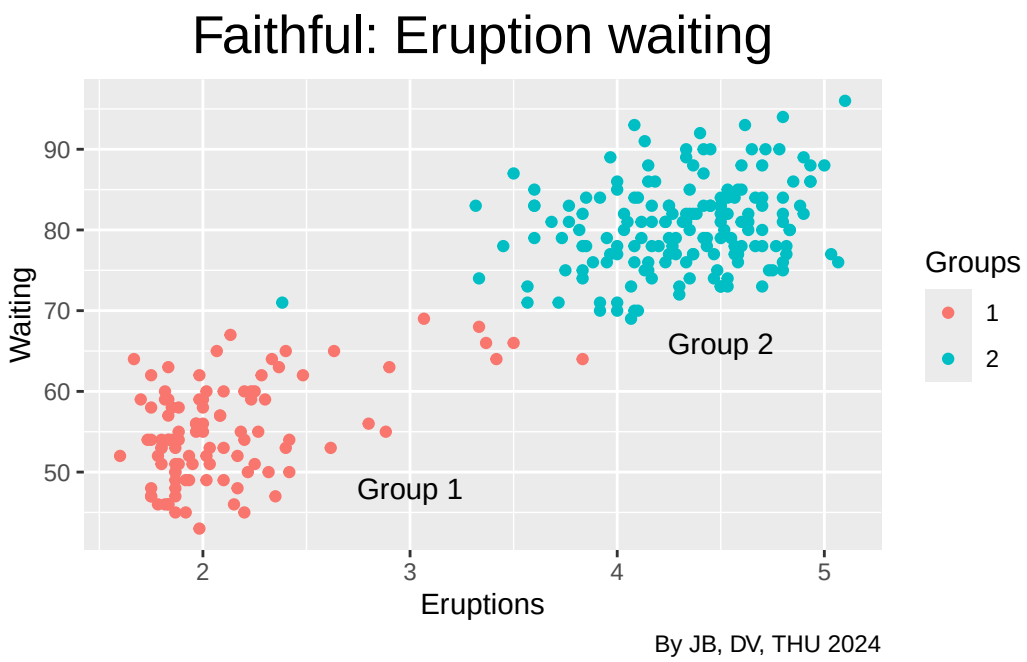
---

21 Adding Annotations

Annotations in a chart are additional text, shapes, or markers used to highlight specific points, provide context, or explain key elements of the data. They enhance the chart's clarity and help the audience understand important insights or trends at a glance.

In this section we will be using the `faithful` dataset to display the different groups of the faithful eruption waiting

```r
my_faithful <- faithful

my_faithful$group <- ifelse(my_faithful$waiting < 70 & my_faithful$eruption < 4, 1,2)
my_faithful$group = as.factor(my_faithful$group)

ggplot(my_faithful, aes(x = eruptions, y = waiting, color = group)) +
  geom_point() +
  labs(title="Faithful: Eruption waiting",
       x = 'Eruptions',
       y = 'Waiting',
       color = "Groups",
       caption = 'By JB, DV, THU 2024')+
  theme(plot.title = element_text(hjust = 0.5, size=20))+
  annotate("text", x = 3, y = 48, label = "Group 1") +
  annotate( "text", x = 4.5, y = 66, label = "Group 2")
```



The graph shown wants to seperate the data into `Group 1` and `Group 2`. However the data for categorizing the data into 2 groups is not provided in the `faithful` dataset.

By using the `my_faithful$waiting < 70 & my_faithful$eruption < 4, 1,2)` from the Data Manipulation section, we are able to seperate the data into 2 groups.

- `my_faithful$waiting < 70`: `Group 1` being the waiting time less than 70 and `Group 2` being the waiting time 70 and more

- `my_faithful$eruption < 4`: `Group 1` being eruption less than 4 and `Group 2` being eruption 4 and more

To add the colors into 2 groups, we have to make sure that newly made column `group` is a `factor` by using `as.factor(my_faithful$group)`

Finally, we are able to use `annotate()` in order to highlight the different groups in the chart

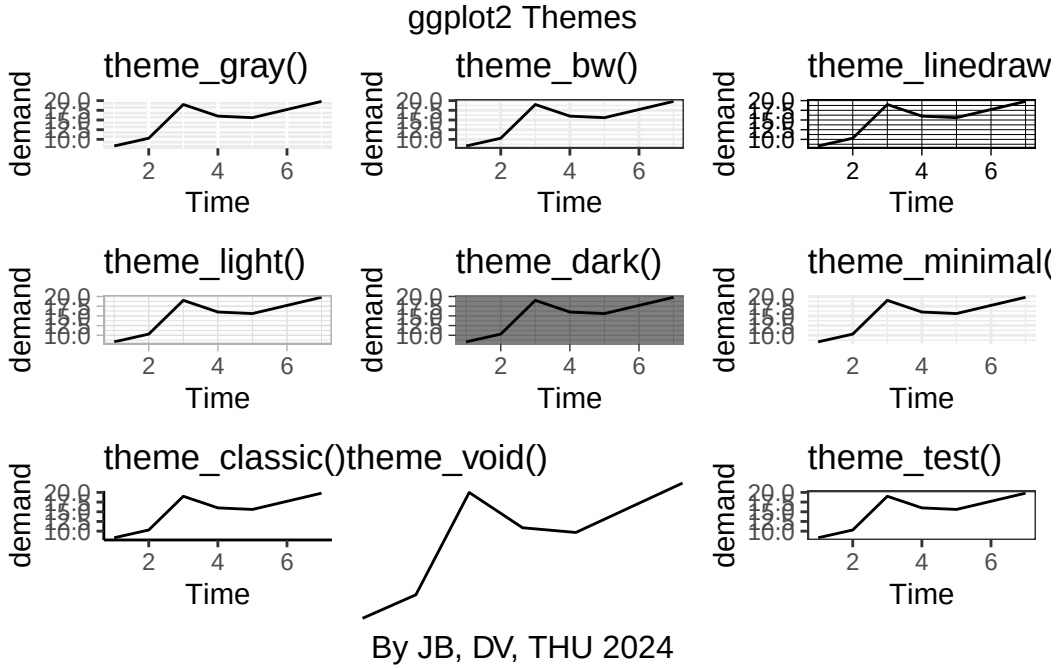
- `"text"`: Specifies that the annotation will be a text label.
- `x`: Determines the horizontal `(x-axis)` position of the annotation.
- `y`: Determines the vertical `(y-axis)` position of the annotation.
- `label`: The content of the annotation—what the text will display.

---

22 Themes

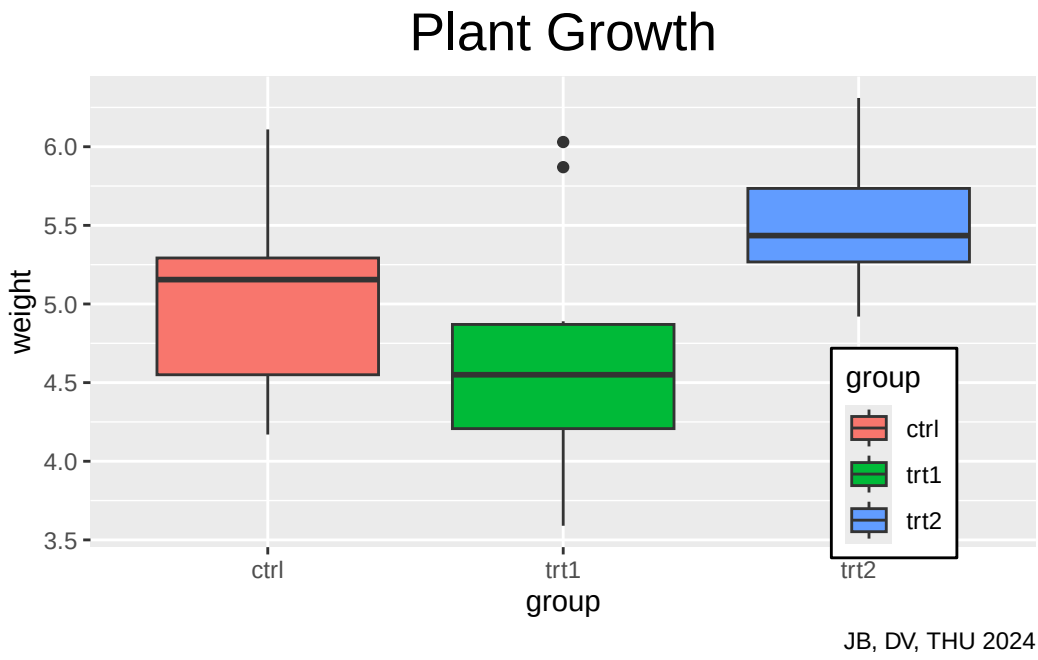
Themes in `ggplot2` control the overall appearance of the plot, including the background, gridlines, text styles, legend, and axis formatting. They allow for customization of the visual elements to make charts more professional or tailored to specific needs.

Themes are applied using the `theme_*()` functions in `ggplot2`. The package provides nine built-in themes, each suited for different presentation purposes.



- `theme_gray()` (default):
  - The default theme with a gray background and white gridlines.
  - Suitable for making gridlines easily visible, which helps in interpreting data.
- `theme_bw()` (black and white):
  - A minimalistic theme with a white background and black gridlines.
  - Useful for printing or reports where colored backgrounds might not be desirable.
- `theme_linedraw()`:
  - A theme with thin black gridlines and a white background.
  - Inspired by technical line drawings, offering a precise and clean look.
- `theme_light()`:
  - A theme with a light gray background and white gridlines.
  - Offers a subtle contrast, making it visually appealing while retaining clarity.
- `theme_dark()`:
  - A theme with a dark gray background and white gridlines.
  - Suitable for presentations, particularly when displayed on a screen.
- `theme_minimal()`:
  - A clean, modern theme with no background and minimal gridlines.
  - Best for presentations where simplicity and a polished look are important.
- `theme_classic()`:
  - A traditional theme with a white background and no gridlines.
  - Emphasizes the plot area and is reminiscent of classic statistical graphs.
- `theme_void()`:
  - A blank slate theme with no axes, text, or gridlines.
  - Ideal for overlays or creating customized charts where the plot elements are designed manually.
- `theme_test()`:
  - A diagnostic theme used for debugging.
  - Displays all the visual elements of a plot to help identify alignment and layout issues.

```r
ggplot(PlantGrowth, aes(x = group, y = weight, fill = group)) +
  geom_boxplot() +
  labs(title= 'Plant Growth',
    caption = 'JB, DV, THU 2024') +
  theme(plot.title = element_text(size= 20, hjust=0.5),
        legend.position = c(0.85, 0.2),
        legend.background = element_rect(fill = 'white',
                                         color = 'black'))
```



`theme()` function is not limited to what is provided by `ggplot2`. In this example, we are able to change the following:

- `plot.title`: Changes the appearance of the chart title
  - `size`: Changes the font size
  - `hjust`: Changes the position of the text
- `legend.position=c(0.85, 0.2)`: Changes the legend position to `x = 0.85` and `y = 0.2`
- `legend.background`: Changes the background of the legend `-element_rect(fill='white', color='black')`: Changes the color inside the rectangle and the outline

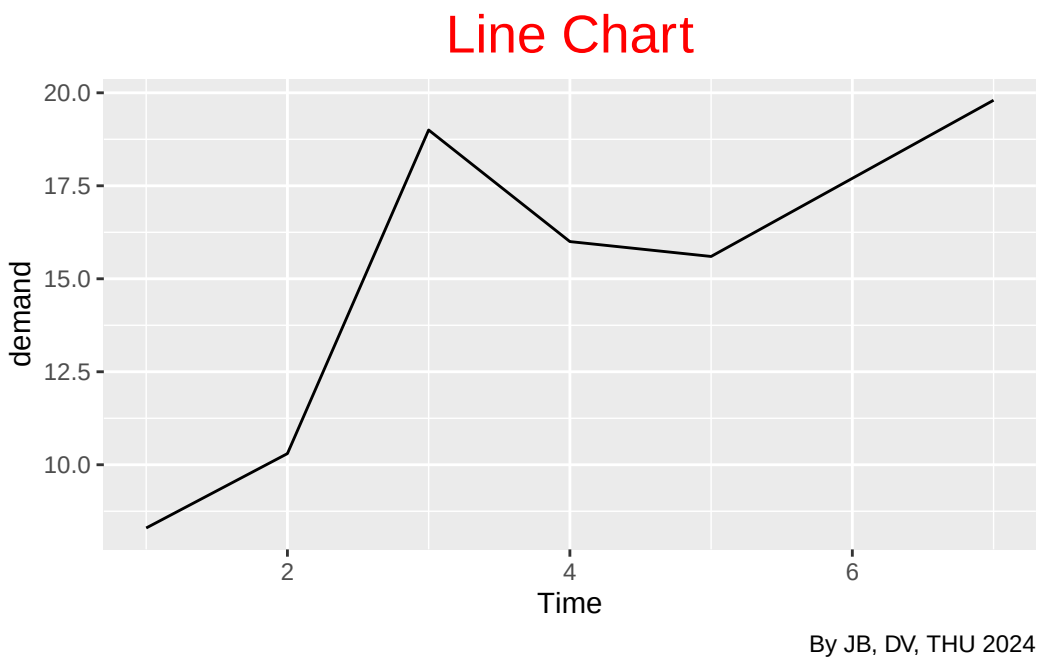
---

23 Creating your own Themes

The nine built-in themes provided by `ggplot2` is just a pre-written line of function that are provided in the package. In this section, we will learn on how to make our own theme

Let's use the chart from previous section

```
ggplot(BOD, aes (x = Time, y = demand)) +
  geom_line() +
  labs(title= "Line Chart",
       caption = 'By JB, DV, THU 2024') +
  theme(plot.title = element_text(hjust= 0.5, size = 20, colour= 'red'))
```

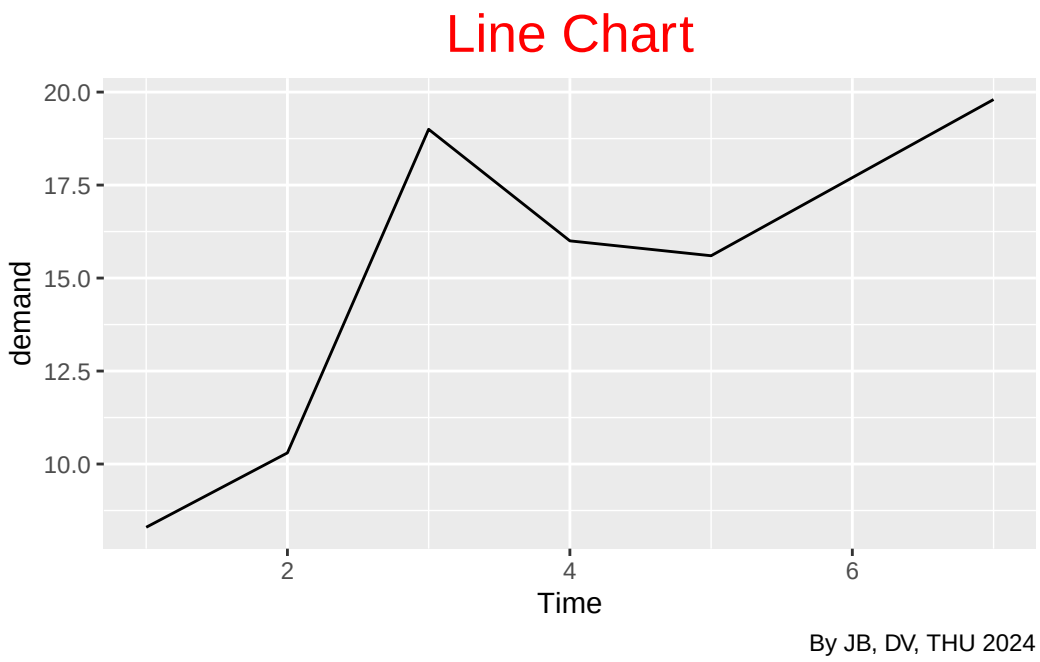


Here as we can see that we have added `theme(plot.title = element_text(hjust= 0.5, size = 20, colour= 'red')` to our preferance, and we want to apply to other chart as well, but we have to write the function again.

In big projects, this could be time consuming and produces a long line, we can however store the functions into a dataset in which we can use, so instead we can

```
ct <-  theme(plot.title = element_text(hjust= 0.5, size = 20, colour= 'red'))

BOD <- ggplot(BOD, aes (x = Time, y = demand)) +
```

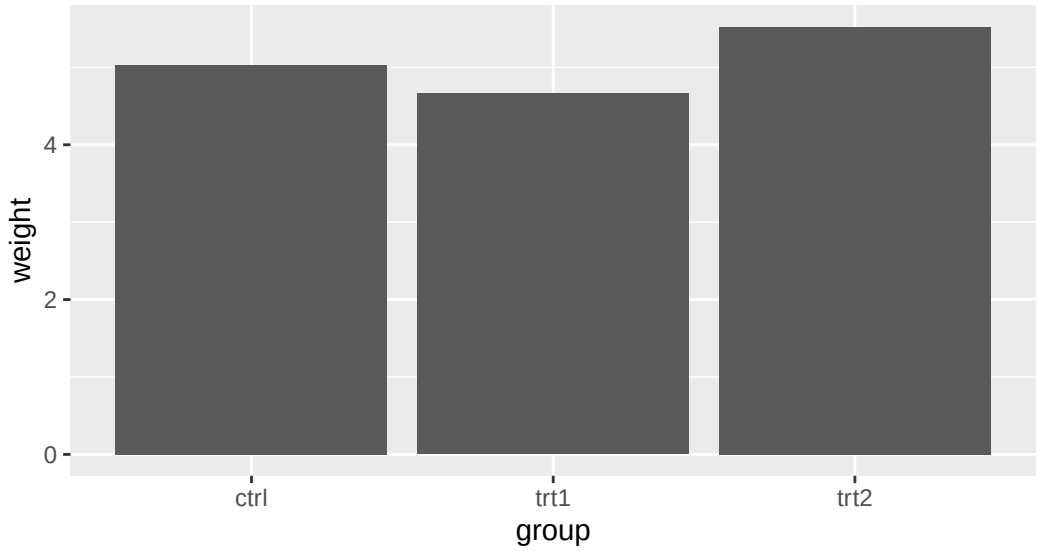
```
  geom_line() +
  labs(title= "Line Chart",
       caption = 'By JB, DV, THU 2024')

BOD + ct
```



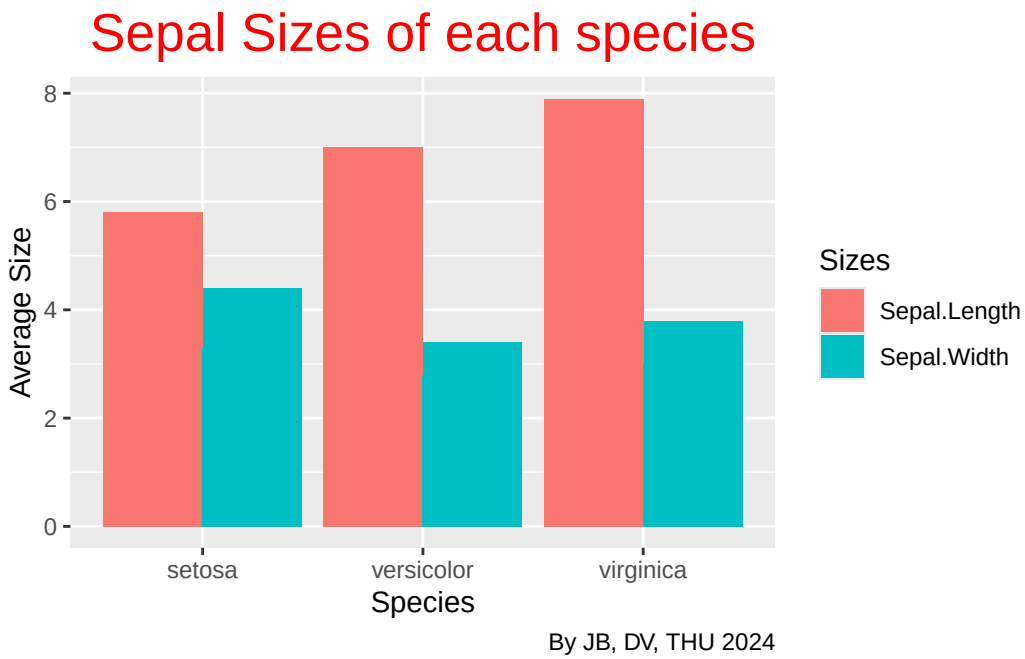
And now, what we have instead of writing the functions all over again, we can simply use `ct`. Let's try with a different plot

```
ggplot(pg_mean, aes(x = group, y = weight)) +
  geom_col() +
  labs (title= "Bar Chart",
       caption = 'By JB, DV, THU 2024') + ct
```

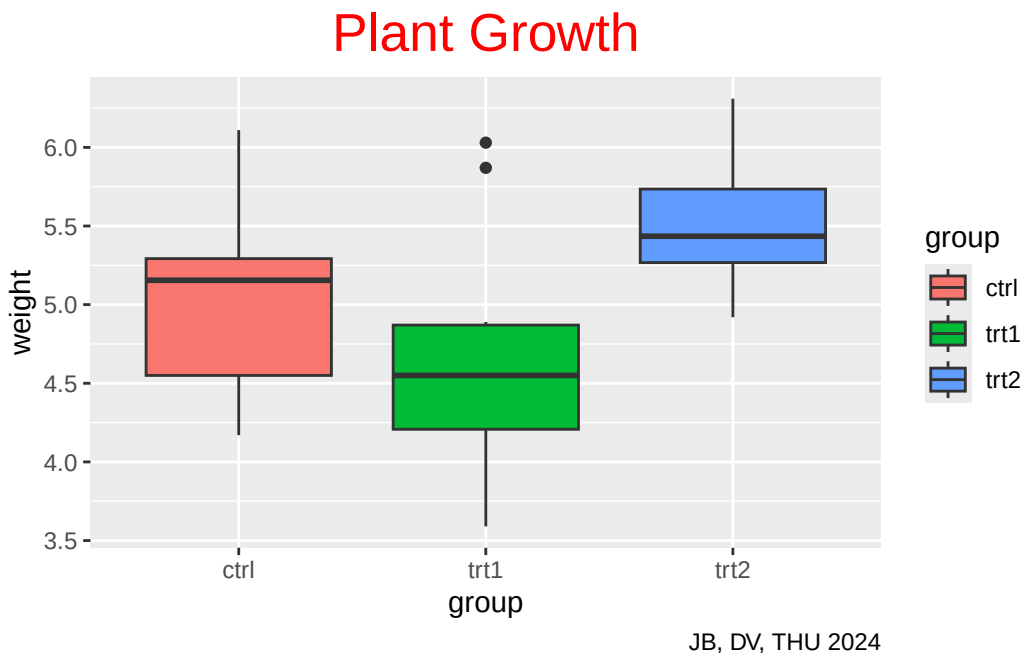
Bar Chart




```
ggplot(iris_long, aes(x = as.factor(Species), y = Count, fill = Sizes)) +
  labs(title="Sepal Sizes of each species",
       x= 'Species',
       y= 'Average Size',
       caption = 'By JB, DV, THU 2024') +
  geom_bar(position = 'dodge', stat= 'identity') + ct
```




```
ggplot(PlantGrowth, aes(x = group, y = weight, fill = group)) +
  geom_boxplot() +
  labs(title= 'Plant Growth',
    caption = 'JB, DV, THU 2024') + ct
```



As you can see, all the text in 3 chart is `size = 20` and are `red`, indicating that you have successfully made your own themes.