# Introduction to R

Tan Sein Jone

University of British Columbia

July 26, 2024

# Table of contents

1. Data Visualization

2. Tips for Tidyverse

3. Functions Best Practices

# Table of Contents

1. Data Visualization

2. Tips for Tidyverse

3. Functions Best Practices

# Data Visualization

- Data visualization is the process of representing data graphically.
- R has a wide range of functions and packages that make data visualization easier.
- The most common types of data visualizations are scatter plots, bar charts, and line charts.

## Base Plotting

- Base plotting is the default plotting system in R.
- Base plotting is simple and easy to use.
- Base plotting is good for creating simple plots.

```
x <- c(1, 2, 3, 4, 5)
y <- c(2, 4, 6, 8, 10)

plot(x, y)
```

## ggplot2

- ggplot2 is a popular plotting package in R.
- ggplot2 is based on the grammar of graphics.
- ggplot2 is good for creating complex plots.

```
library(ggplot2)

data <- data.frame(
    x = c(1, 2, 3, 4, 5),
    y = c(2, 4, 6, 8, 10)
)

ggplot(data, aes(x = x, y = y)) + geom_point()
```

## Plotly

- Plotly is an interactive plotting package in R.
- Plotly is based on the Plotly.js library.
- Plotly is good for creating interactive plots.

```
library(plotly)

data <- data.frame(
    x = c(1, 2, 3, 4, 5),
    y = c(2, 4, 6, 8, 10)
)

plot_ly(
    data, x = ~x, y = ~y,
    type = "scatter",
    mode = "markers"
)
```

# Table of Contents

1. Data Visualization

2. Tips for Tidyverse

3. Functions Best Practices

## Tips for Tidyverse

- The Tidyverse is a collection of R packages designed for data science.
- The Tidyverse is based on the principles of tidy data.
- The Tidyverse is good for data manipulation and visualization.

## Pipes

- Pipes are a way to chain R functions together.
- Pipes make it easy to read and write code.
- Pipes are good for data manipulation and visualization.

```
library(dplyr)

data <- data.frame(
    x = c(1, 2, 3, 4, 5),
    y = c(2, 4, 6, 8, 10)
)

data %>%
    filter(x > 2) %>%
    ggplot(aes(x = x, y = y)) +
    geom_point()
```

## Tibbles

- Tibbles are a modern version of data frames.
- Tibbles are easier to read and write than data frames.
- Tibbles are good for data manipulation and visualization.

```
library(dplyr)

data <- tibble(
    x = c(1, 2, 3, 4, 5),
    y = c(2, 4, 6, 8, 10)
)

data %>%
    filter(x > 2) %>%
    ggplot(aes(x = x, y = y)) +
    geom_point()
```

## Grouping

- Grouping is a way to split data into groups.
- Grouping is good for data manipulation and visualization.
- Grouping is good for summarizing data.

```
library(dplyr)

data <- data.frame(
    x = c(1, 2, 3, 4, 5),
    y = c(2, 4, 6, 8, 10),
    group = c("A", "A", "B", "B", "B")
)

data %>%
    group_by(group) %>%
    summarize(mean_y = mean(y))
```

# Table of Contents

1. Data Visualization

2. Tips for Tidyverse

3. Functions Best Practices

## Functions Best Practices

- Functions are a way to organize code in R.
- Functions are good for code reuse and readability.
- Functions are good for data manipulation and visualization.

## Function Basics

- A function is a block of code that performs a specific task.
- A function takes input, processes it, and returns output.
- A function is defined using the function keyword.

```
add <- function(x, y){
    return(x + y)
}

add(5, 3)
```

## Function Arguments

- A function can take zero or more arguments.
- Arguments are the input values that a function uses to perform its task.
- Arguments can have default values.

```
add <- function(x, y = 0){
    return(x + y)
}

add(5, 3)
add(5)
```

## Higher Order Functions

- Higher-order functions are functions that can either take other functions as arguments or return them as results.
- This is possible because functions are first-class citizens.
- Higher-order functions allow us to abstract over actions, not just values.

## Tips for Higher Order Functions and Modularity

- Higher-order functions are functions that can take other functions as arguments or return them as results.
- Higher-order functions allow us to abstract over actions, not just values.
- Higher-order functions are good for modularity and code reuse.

```
add <- function(x, y){
    return(x + y)
}

subtract <- function(x, y){
    return(x - y)
}

operate <- function(func, x, y){
    return(func(x, y))
}

operate(add, 5, 3)
operate(subtract, 5, 3)
```

```
add <- function(x, y){
    return(x + y)
}

subtract <- function(x, y){
    return(x - y)
}

create_operator <- function(op){
    if(op == "add"){
        return(add)
    } else if(op == "subtract"){
        return(subtract)
    }
}

operator <- create_operator("add")
operator(5, 3)
```