Introduction to R - Part 2
Data Science Skills Day 2022

Anjali Silva, PhD

Summer Undergraduate Data Science Research Program
University of Toronto
03 June 2022

a.silva@utoronto.ca anjalisilva.github.io @Silva_Anjali

Course Description

- Introduction to R - Part 2
  Data Science Skills Day

  - The vast amount of data produced by evolving information technology requires tools and skills. Among the many tools, R is a free, open-source language for data sciences. R is a programming language that can aid in the process of data analysis. This course is a beginner level, introductory course for R for data analysis. We will learn about R, RStudio (the environment use to work in R), including installation, and apply R for beginner-level data modeling and visualization. By the end of the course, you'll have a introduction to the flexibility of R, different functionalities, and understand how to apply it for basic data exploration.

  - Friday 10:00 am – 4 pm EST; online - synchronous.

Material

- Instructor Slides:
  - https://github.com/anjalisilva/DSI_IntroductionToR
  - SlideIntroR2022.pdf
- Instructor R Script:
  - https://github.com/anjalisilva/DSI_IntroductionToR
  - Script.R

Data Frames, Conditioning, Looping & Booleans

Data Frames

- Data frames are used to store tabular data in R.

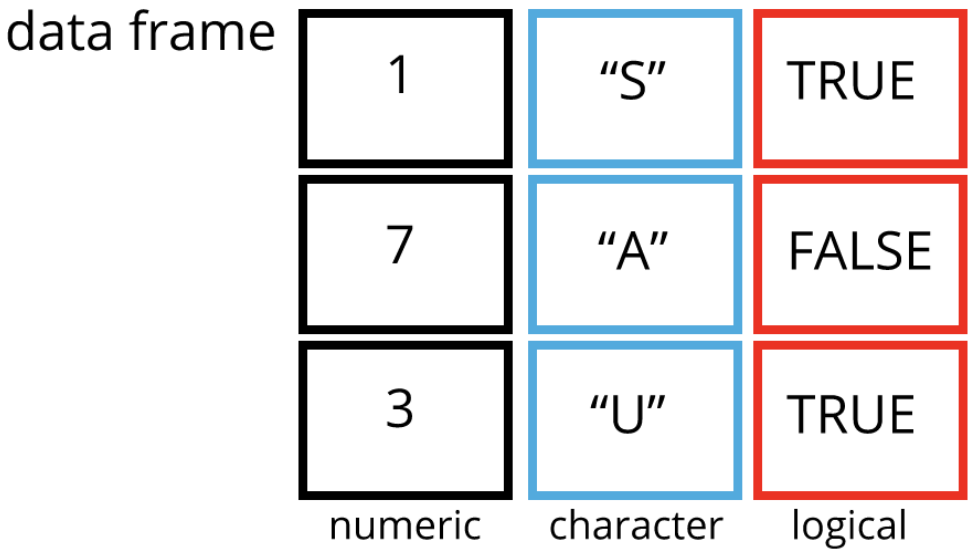


Figure: Data frames can store different classes of objects in each column. Figure from: https://datacarpentry.org/r-socialsci/02-starting-with-data/index.html

Data Frames

- Data frames are used to store tabular data in R.
- Data frames can store different classes of objects in each column.

```
dataFrameExample <- data.frame(
numbers = 1:4,
sex = c("M", "M", "F", "F"))

dataFrameExample
class(dataFrameExample) # "data.frame"
dim(dataFrameExample) # 4 2
names(dataFrameExample) # "numbers" "sex"
```

Data Frames

- Data frames can be converted to a matrix using data.matrix().

```
dataMatrix <- data.matrix(dataFrameExample)
class(dataMatrix) # "matrix"
dim(dataMatrix) # 4 2
```

Conditioning

- Conditioning in R is done using an if/else statement:

```
?runif # Uniform Distribution
x <- runif(n = 1, min = 0, max = 1)
x # differs every time (random)

if (x < 0.5) {
  print(x)
} else {
    print(-x)
}
```

Looping

- For loops:

```
numbers <- 20:26
for (i in numbers) {
  cat("\n Number is", i)
}
```

- While loops:

```
i <- 1
while (i < 10) {
  cat("\n Number is", i)
  i <- i + 1
}
```

Question:
What is the difference between cat() and print()?

Booleans

- TRUE/FALSE or T/F are called "boolean" values.

```
testValue <- FALSE
typeof(testValue)
is.logical(testValue)

! testValue
```

- "!" is the NOT operator. It returns the opposite of the argument.

Boolean Operators

- Operators $>$ or $<$:

```
a <- c(1:5)
a
a < 2 # TRUE FALSE FALSE FALSE FALSE
```

- Equivalence test:

```
a == 2 # FALSE  TRUE FALSE FALSE FALSE
```

Boolean Operators

- & symbol is the "AND" operator:

```
a <- 45
(a > 40) & (a < 50) # TRUE
```

- | symbol is the "OR" operator:

```
b <- 10
(b < 20) | (b > 5) # TRUE
```

Any questions?

Data Import/Export, Arithmetic & Text Manipulation

Data Import/Export

- To see the list of pre-loaded data in R:

```
data(package = "datasets")

AirPassengers # Example dataset
head(AirPassengers) # see first few entries
tail(AirPassengers) # see last few entries
```

Data Slicing

- Let us look at another pre-loaded datasets:

```
women # another dataset (last)
?women
dim(women) # 15  2
class(women) # "data.frame"
head(women) # height and weight information

women$height > 60 # slicing
women[women$height > 60, ] # slicing
```

Data Reading/Writing

- Files can be written using functions like write.csv(), write.table():

```
getwd() # file will be saved here
write.csv(x = women, file = "women.csv")
# saving women dataset in current working directory
```

- Txt files can be read using read.table("location of the file") or read.csv()

```
womenNew <- read.csv(file = "women.csv", row.names = 1)
womenNew

head(womenNew) # to view first part of object
tail(womenNew) # to view last part of object
dim(womenNew) #  15  2
womenNew[c(1:5), ] # to view first 5 rows
womenNew[, 1] # to view first column
```

Arithmetic

- "+" is used for addition:

```
x <- 2.5 + 2
x # 4.5

y <- 2:15
sum(y) # 119
sum(y[1:3]) # 9
```

Arithmetic

- "-" is used for substraction:

```
x <- 2.5 - 2
x # 0.5
```

- "/" is used for division:

```
x <- 2 / 2
x # 1
```

Arithmetic

- "*" is used for multiplication:

```
x <- 2 * 2
x # 4
```

- "%*% " is used for matrix multiplication:

```
a <- matrix(1:6, nrow = 2, ncol = 3) # 2 x 3 matrix
a
b <- matrix(7:12, nrow = 3, ncol = 2) # 3 x 2 matrix
b
c <- a %*% b
c # 2 x 2 matrix
```

Text Manipulation

- Use paste() or paste0() to combine strings:

```
a <- "Hello"
b <- "world"
paste(a, b)
paste(a, b, sep = "+")
places <- c("London", "Boston", "Toronto")
paste("Travelling to", places)
paste(places, collapse = " ") # combine to one string
paste0(a, b) # no space
```

- Counts the number of characters in a string using nchar():

```
nchar(places)
```

Text Manipulation

- Make strings all uppercase or lowercase:

```
tolower(places)
toupper(places)
```

Any questions?

Functions

Writing Functions

- Functions combine a sequence of expressions that are executed to achieve a goal.
- Can be reused without rewriting the sequence of expressions.
- Take input, *function arguments* and generate output, *return value*.
- When writing functions, ask yourself:
  - *What will the user want to modify in this function*?
  - This will help determine *function arguments*.

Function Interface

- Functions are defined using 'function' assigned to a variable name.

```
firstFunction <- function(argumentOne, argumentTwo) {
  cat("\n First argument is", argumentOne, "\n")
  cat("\n Second argument is", argumentTwo, "\n")
  argumentThree <- argumentTwo + 1
  cat("\n Second argument + 1 is", argumentThree, "\n")
  return(argumentThree)
}
```

Function Interface

- Function names cannot begin with a number.
- To run the function:

```
firstFunction(argumentOne = 2, argumentTwo = 3)
```

- To "call" or "invoke" the function, type function name:

```
firstFunction
```

Function Interface

- Default values play a vital role in R functions and influence user's behaviour.
- Default values should be assigned using "=", and not "←".

```
firstFunction <- function(argOne = 1, argTwo = 3) {
  cat("\n First argument is", argOne, "\n")
  cat("\n Second argument is", argTwo, "\n")
  argThree <- argTwo + 1
  cat("\n Second argument + 1 is", argThree, "\n")
  return(argThree)
 }
```

Local and Global Variables

- Where variables are defined matters.
- Variables defined within functions are only accessible from within the function.
- Variables declared within a function are called "local".
- Variables declared outside of functions are called "global".

Variables

- If variables are defined outside of functions, globally, they hold that value.

```
argOne <- "Hello"

anotherFunction <- function() {
  argOne <- 10
  return(argOne)
 }

anotherFunction()

argOne # What would this return?
```

Variables

- Need to pass data into functions:

```
women

# Incorrect
dataAnalysis <- function() {
    heightData <- women$height
   return(mean(heightData))
 }

# Recommended
dataAnalysis <- function(inputData) {
    heightData <- inputData$height
    meanOfHeight <- mean(heightData)
    return(meanOfHeight)
 }
```

Return Statement

- Use "return" statement to give back values from function.
- Return statement must be the last command in the function.

```
women

dataAnalysis <- function(inputData) {
    heightData <- inputData$height
    meanOfHeight <- mean(heightData)
    return(meanOfHeight)
 }

dataAnalysis(inputData = women) # 65
```

Any questions?

Non-base Functions

- Functions have been written by other authors.
- Non-base functions in R can be utilized by downloading R packages.
- To see all the currently loaded packages:

```
search()
```

Non-base Functions

- Popular repositories for Packages:
  - The Comprehensive R Archive Network (CRAN)
    - Link: https://cran.r-project.org/web/packages/
  - Bioconductor
    - Link: https://www.bioconductor.org/install/
  - GitHub
- Depending on the source of Package, downloading instructions may differ.

Example

- User wants to do a cluster analysis of the women built-in dataset.
- Assume the user is interested in performing model-based clustering using Gaussian finite mixture models.
  - Do an R search:

    ```
    ??clustering
    ```

  - Google R packages for model-based clustering.
  - Read blogs on clustering.

Example, continue...

- Get to know about R package 'mclust'.
- Visit https://cran.r-project.org/web/packages/mclust/index.html.
- Click on Reference manual to see all the functions within the package.
- Download package to R session:

```
# Install package from CRAN, case matters!
install.packages("mclust")
library("mclust") # to load and attach
```

Example, continue...

- More details on 'mclust' package:

```
vignette("mclust") # vignette for 'mclust'

?'mclust-package' # get information on package

?mclust # get information on package

lsf.str("package:mclust") # list all functions in package
```

Example, continue...

- Work with 'mclust' package:

```
# Running mclust
MclustResults <- mclust::Mclust(data = women)
str(MclustResults) # provide the structure; list of 15
names(MclustResults)
MclustResults$G # There are four clusters in the dataset

# Citing the package
citation("mclust")
```

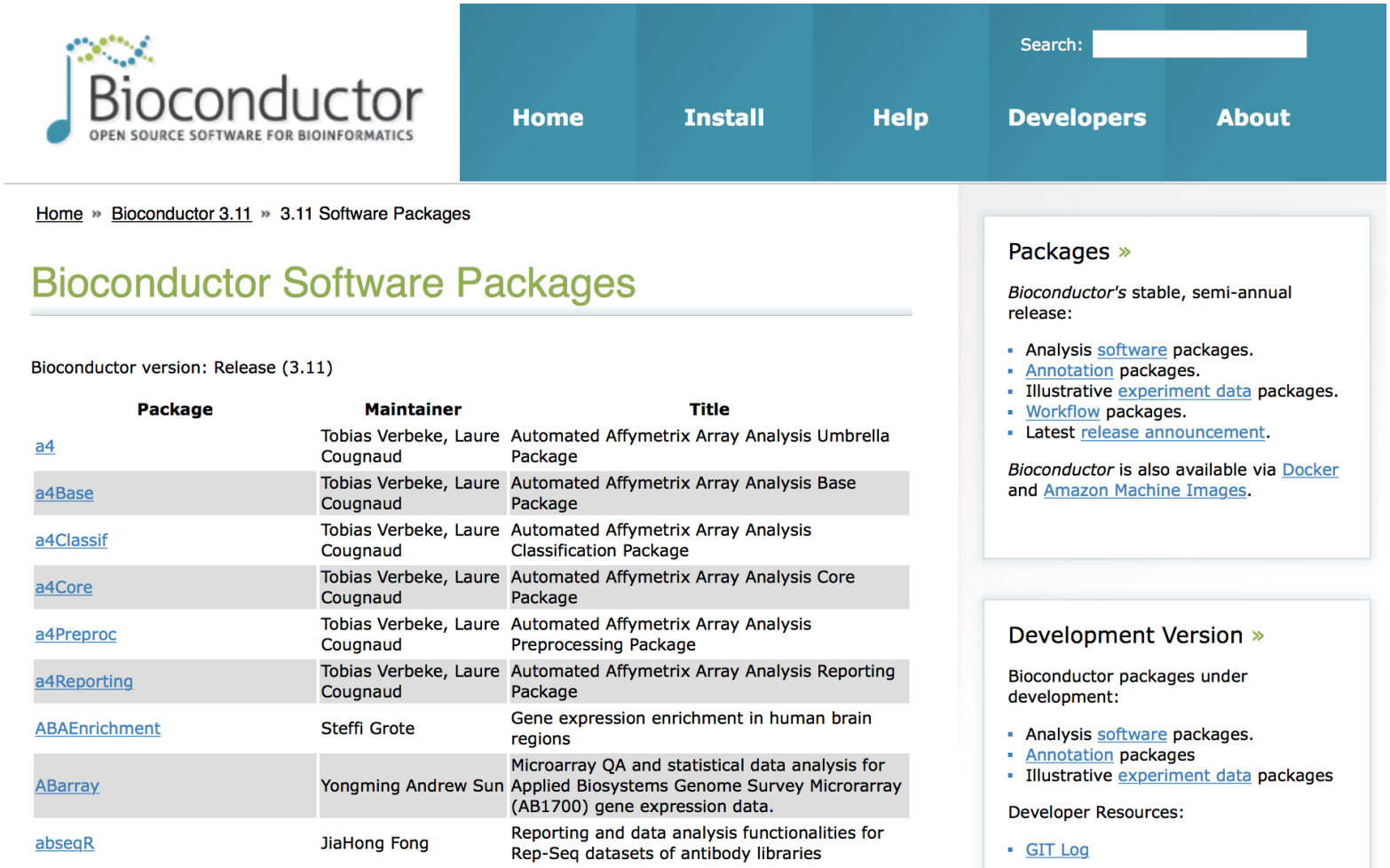
Bioconductor

- Bioconductor packages are available from bioconductor.org.
- URL: https://www.bioconductor.org/install/.

```
if (! requireNamespace("BiocManager", quietly = TRUE))
install.packages("BiocManager")
```

- Bioconductor packages are listed here: https://www.bioconductor.org/packages/release/bioc/.

Bioconductor



Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS

Search:

Home | Install | Help | Developers | About

Home » Bioconductor 3.11 » 3.11 Software Packages

Bioconductor Software Packages

Bioconductor version: Release (3.11)

Package	Maintainer	Title
a4	Tobias Verbeke, Laure Cougnaud	Automated Affymetrix Array Analysis Umbrella Package
a4Base	Tobias Verbeke, Laure Cougnaud	Automated Affymetrix Array Analysis Base Package
a4Classif	Tobias Verbeke, Laure Cougnaud	Automated Affymetrix Array Analysis Classification Package
a4Core	Tobias Verbeke, Laure Cougnaud	Automated Affymetrix Array Analysis Core Package
a4Preproc	Tobias Verbeke, Laure Cougnaud	Automated Affymetrix Array Analysis Preprocessing Package
a4Reporting	Tobias Verbeke, Laure Cougnaud	Automated Affymetrix Array Analysis Reporting Package
ABAEnrichment	Steffi Grote	Gene expression enrichment in human brain regions
ABarray	Yongming Andrew Sun	Microarray QA and statistical data analysis for Applied Biosystems Genome Survey Microrarray (AB1700) gene expression data.
abseqR	JiaHong Fong	Reporting and data analysis functionalities for Rep-Seq datasets of antibody libraries

Packages »

Bioconductor's stable, semi-annual release:

- Analysis software packages.
- Annotation packages.
- Illustrative experiment data packages.
- Workflow packages.
- Latest release announcement.

Bioconductor is also available via Docker and Amazon Machine Images.

Development Version »

Bioconductor packages under development:

- Analysis software packages.
- Annotation packages
- Illustrative experiment data packages

Developer Resources:

- GIT Log

Example

- User is interested in Bioconductor package GenomicFeatures.
- To load package:

```
BiocManager::install("GenomicFeatures")
library("GenomicFeatures")

# list all functions in the package
lsf.str("package:GenomicFeatures")
```

- More about installation: https://www.bioconductor.org/install/.

Bioconductor

Any questions?

R: Graphics

Applying Base Functions

- Package 'graphics' is a standard or base package.

```
library(graphics) # to load and attach

?AirPassengers
# Monthly Airline Passenger Numbers 1949-1960

# plot
plot(AirPassengers)
plot(AirPassengers, type = "p")
plot(AirPassengers, type = "p", main = "Monthly Airline
     Passenger Numbers 1949-1960") # zoom
```

Applying Base Functions

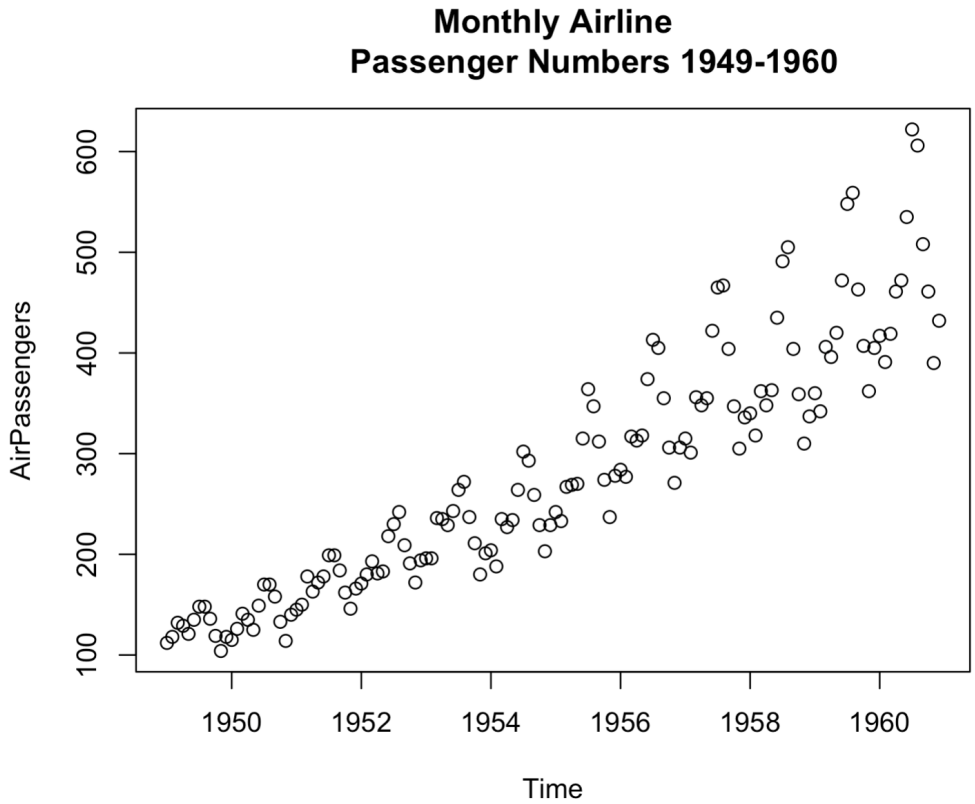


Figure: Plotting AirPassengers dataset as a scatter plot.

Applying Base Functions

- Package 'graphics' is a base package.

```
library(graphics) # to load and attach

# plot
plot(AirPassengers)
plot(AirPassengers, type = "l", main = "Monthly Airline
     Passenger Numbers 1949-1960") # zoom
```

Applying Base Functions

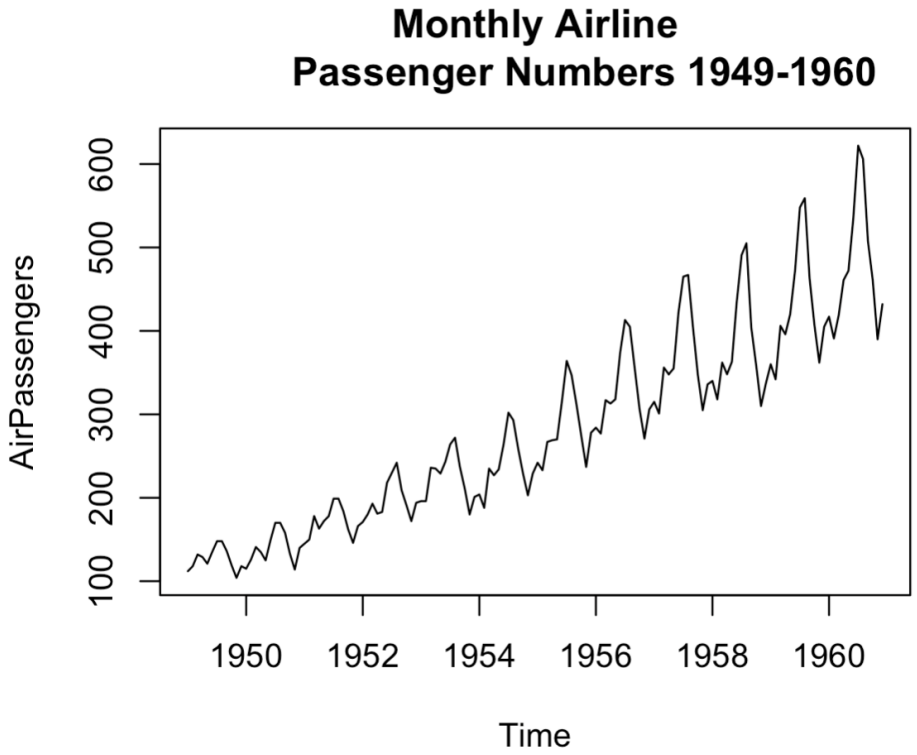


Figure: Plotting AirPassengers dataset as a line plot.

Applying Base Functions

- Package 'graphics' is a base package.

```
library(graphics) # to load and attach

?AirPassengers
# Monthly Airline Passenger Numbers 1949-1960

# To have multiple plots in one overall plot using
par(mfrow = c(1, 2))
hist(AirPassengers) # histogram
boxplot(AirPassengers) # boxplot
```

Applying Base Functions

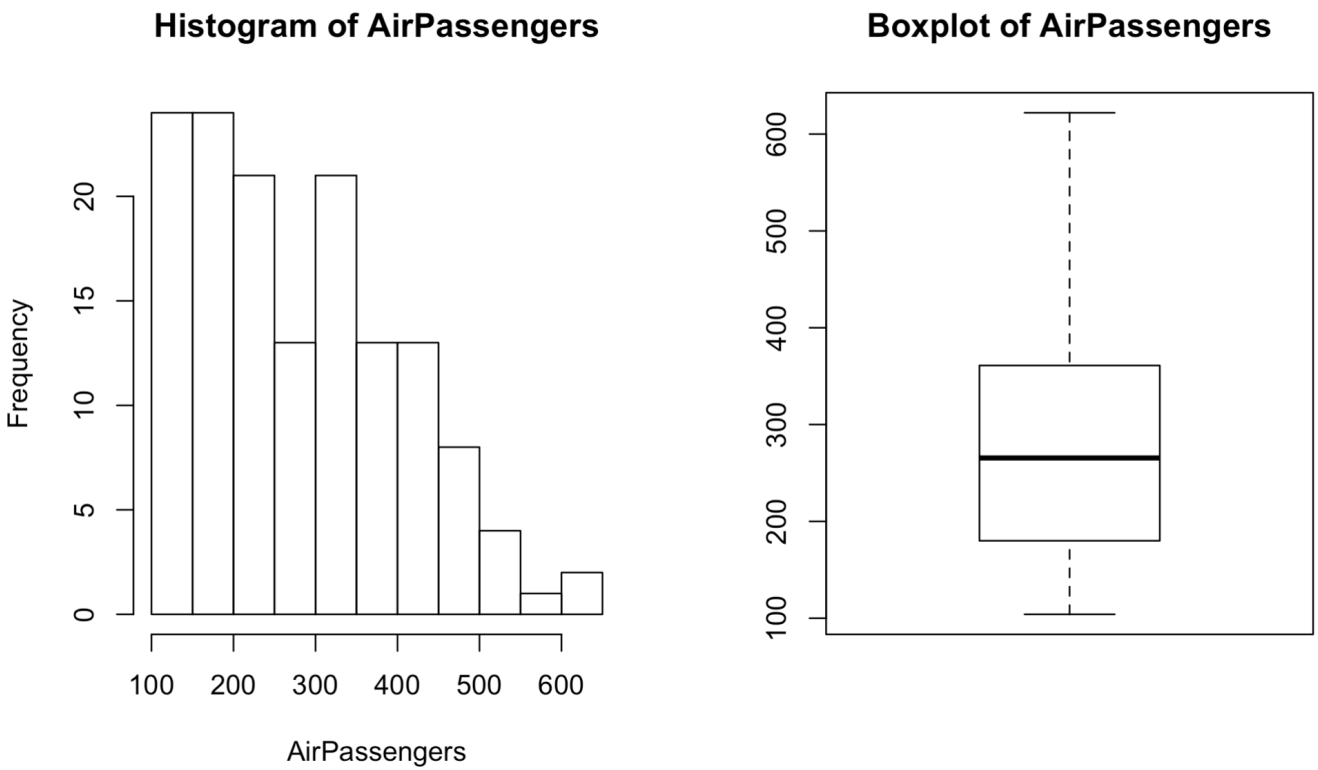


Figure: Output from plotting AirPassengers dataset.

Applying Base Functions

- Package 'graphics' is a base package.

```
library(graphics) # to load and attach

?iris
?pairs

pairs(iris[, c(1:4)], col = iris$Species,
    main = "Scatter plot of iris dataset")
```

Applying Base Functions

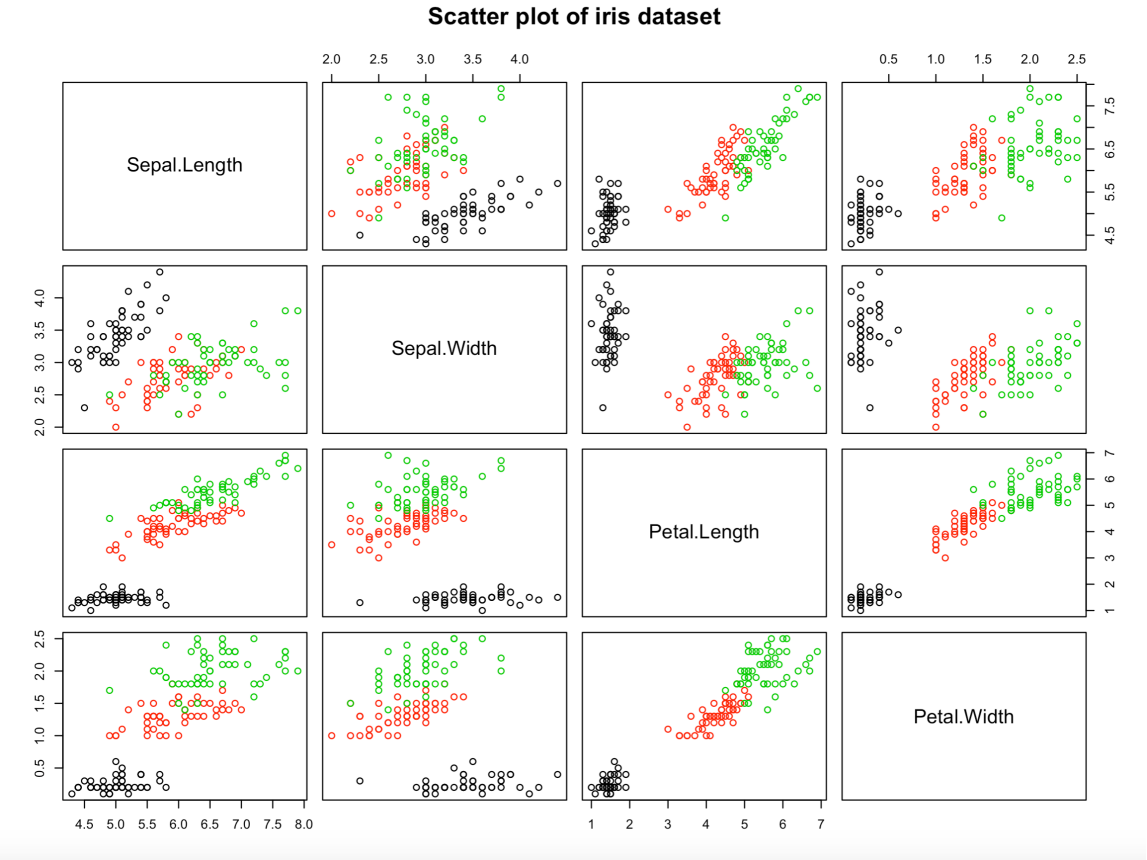


Figure: Output from plotting iris dataset as a pairs plot.

Applying Base Functions

- Package 'graphics' is a base package.

```
library(graphics) # to load and attach

barplot(as.matrix(iris[, c(1:4)]),
        legend.text = TRUE,
        main = "Bar plot of iris dataset")

# Calculate column sums of iris dataset
colSums(as.matrix(iris[, c(1:4)]))

# Sepal.Length  Sepal.Width Petal.Length  Petal.Width
# 876.5         458.6         563.7         179.9
```

Applying Base Functions

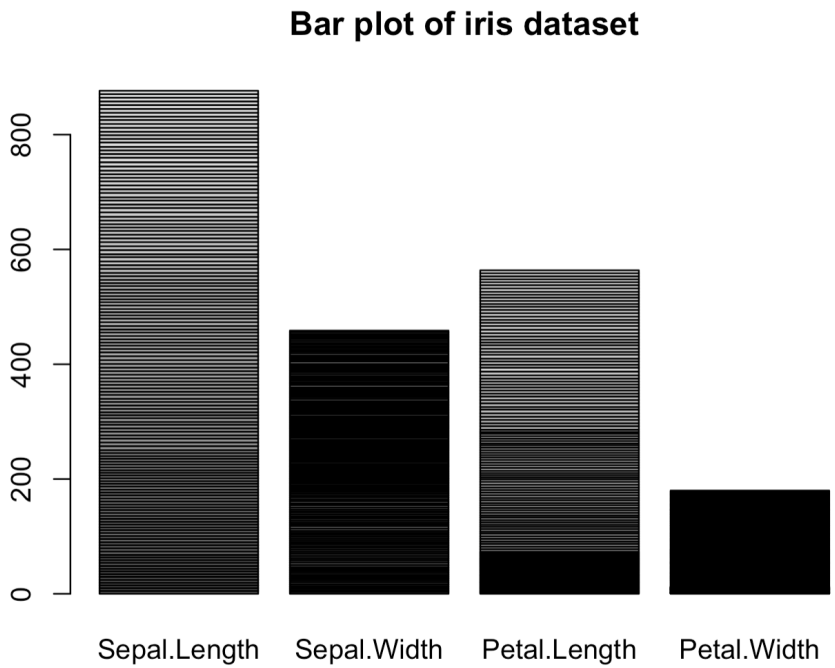


Figure: Output from plotting iris dataset as a bar plot.

R Packages for Plotting

- "tidyverse" is a collection of R packages designed for data science.
- "ggplot2" is a popular plotting package.
- "reshape" is an R package for restructure and aggregation of data.

```
install.packages("tidyverse")
library("tidyverse")
library("ggplot2")

install.packages("reshape")
library(reshape)
```

R Packages for Plotting

```
library(reshape)

meltIrisData <- reshape::melt(iris)

# Plot iris dataset using ggplot
library("ggplot2")

ggplot2::ggplot(meltIrisData, aes(x = variable,
                                  y = value, col = Species)) +
  geom_point() +
  labs(x = "Variable",  y = "Measurements (cm)") +
  ggtitle("Scatter plot of iris dataset") +
  theme_bw() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

R Packages for Plotting

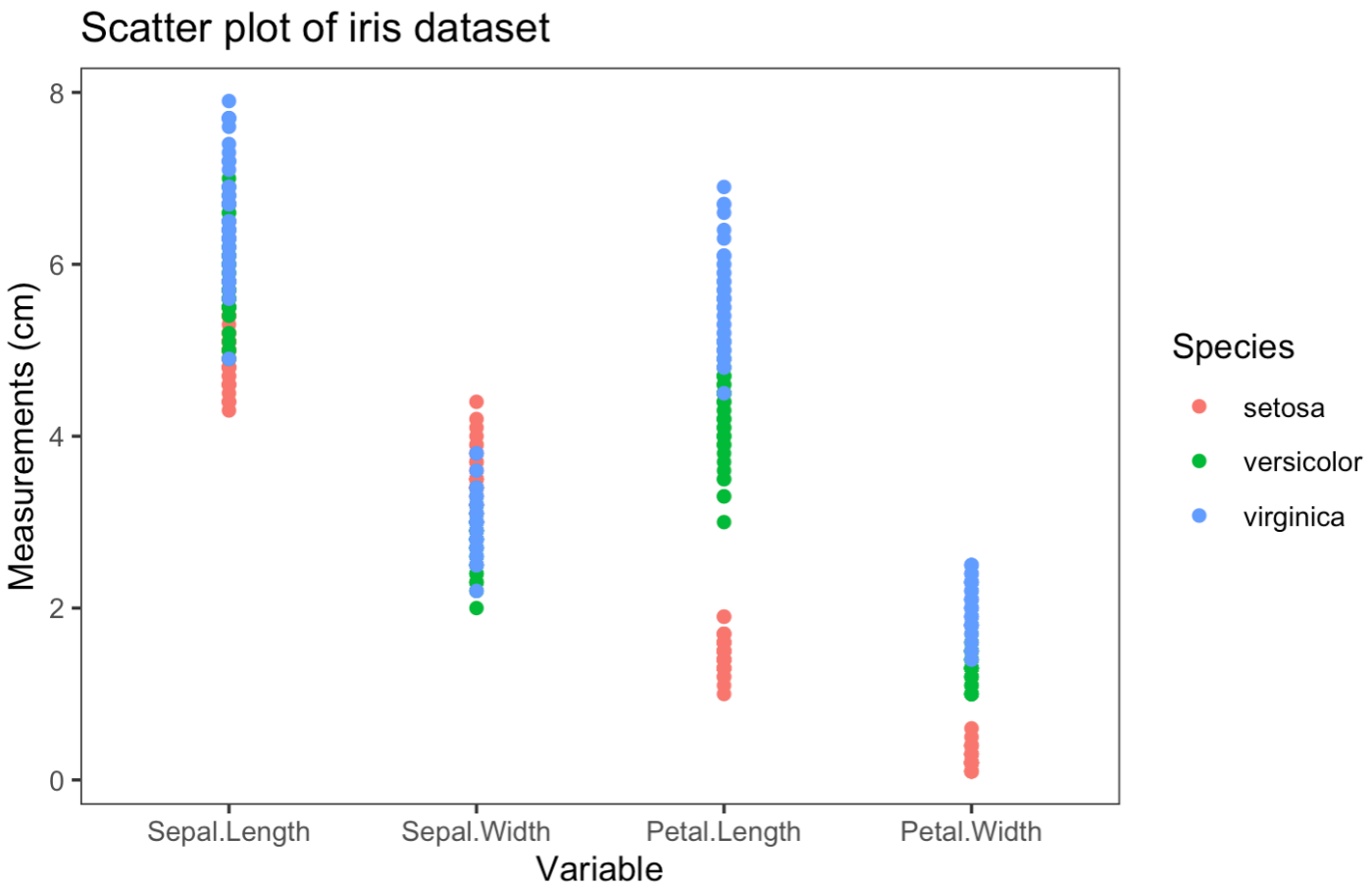


Figure: Output from plotting iris dataset as a scatter plot via ggplot2::ggplot().

R Packages for Plotting

```
library(reshape)

meltIrisData <- reshape::melt(iris)

# Plot iris dataset using ggplot
library("ggplot2")

ggplot2::ggplot(meltIrisData, aes(x = variable,
                                  y = value, col = Species)) +
  geom_boxplot() +
  labs(x = "Variable",  y = "Measurements (cm)") +
  ggtitle("Scatter plot of iris dataset") +
  theme_bw() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

R Packages for Plotting

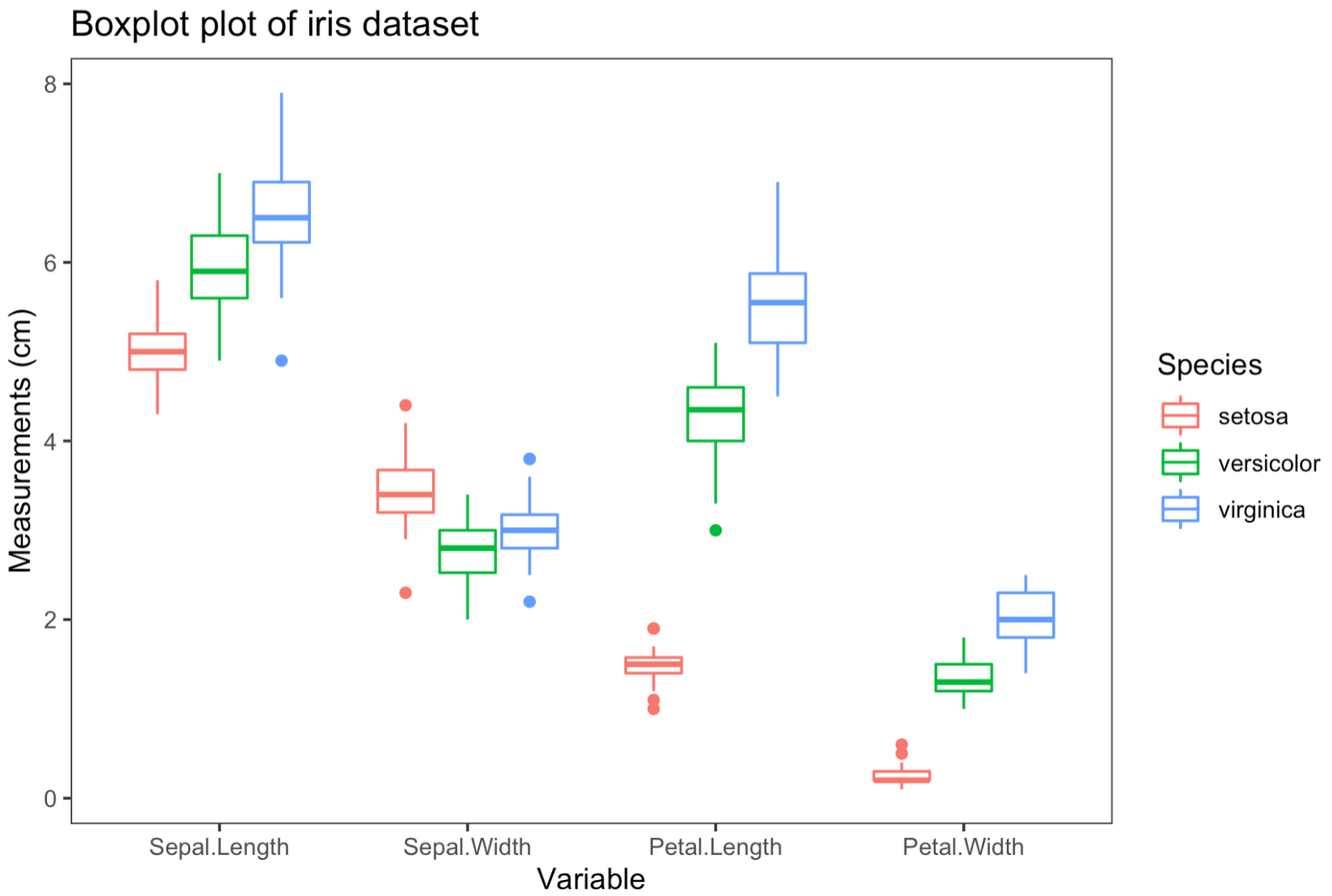


Figure: Output from plotting iris dataset as a boxplot via ggplot2::ggplot().

R Packages for Plotting

```
# Violin plot of iris dataset
library("ggplot2")

ggplot2::ggplot(meltIrisData, aes(x = variable, y = value,
                                  col = Species, fill = Species)) +
  geom_violin() +
  labs(x = "Variable",  y = "Measurements (cm)") +
  ggtitle("Violin plot of iris dataset") +
  theme_bw() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

R Packages for Plotting

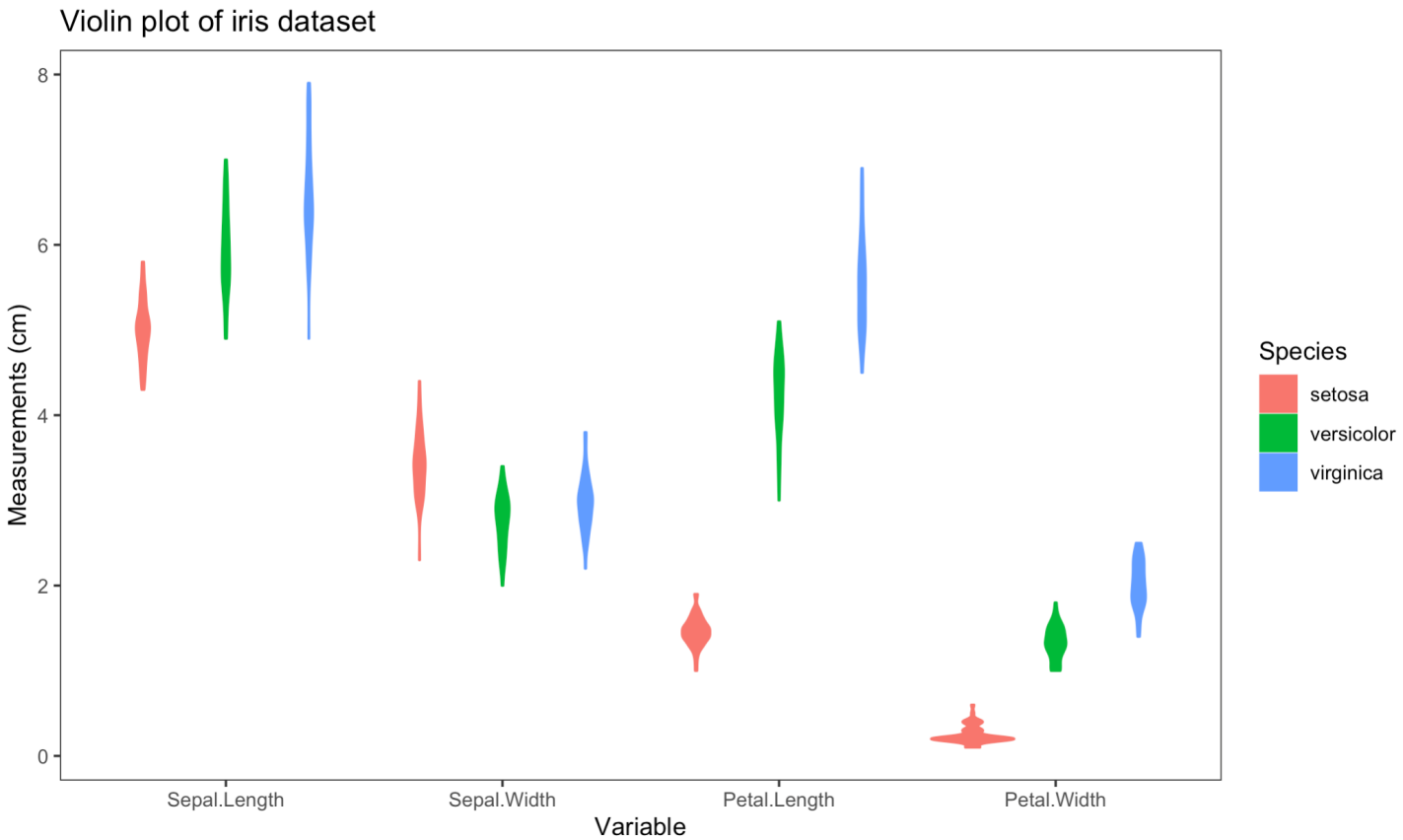


Figure: Output from plotting iris dataset as a violin plot via ggplot2::ggplot().

R Packages for Plotting

```
# Plot a histogram of Sepal.Width

library("ggplot2")

ggplot2::ggplot(data = iris,
  mapping = aes(x = Sepal.Width, fill = Species)) +
  geom_histogram(binwidth = 0.2) +
  facet_grid(Species ~ .) +
  ggtitle("Histogram of Sepal.Width in iris dataset") +
  xlab("Sepal Width (cm)") +
  theme_bw() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

R Packages for Plotting

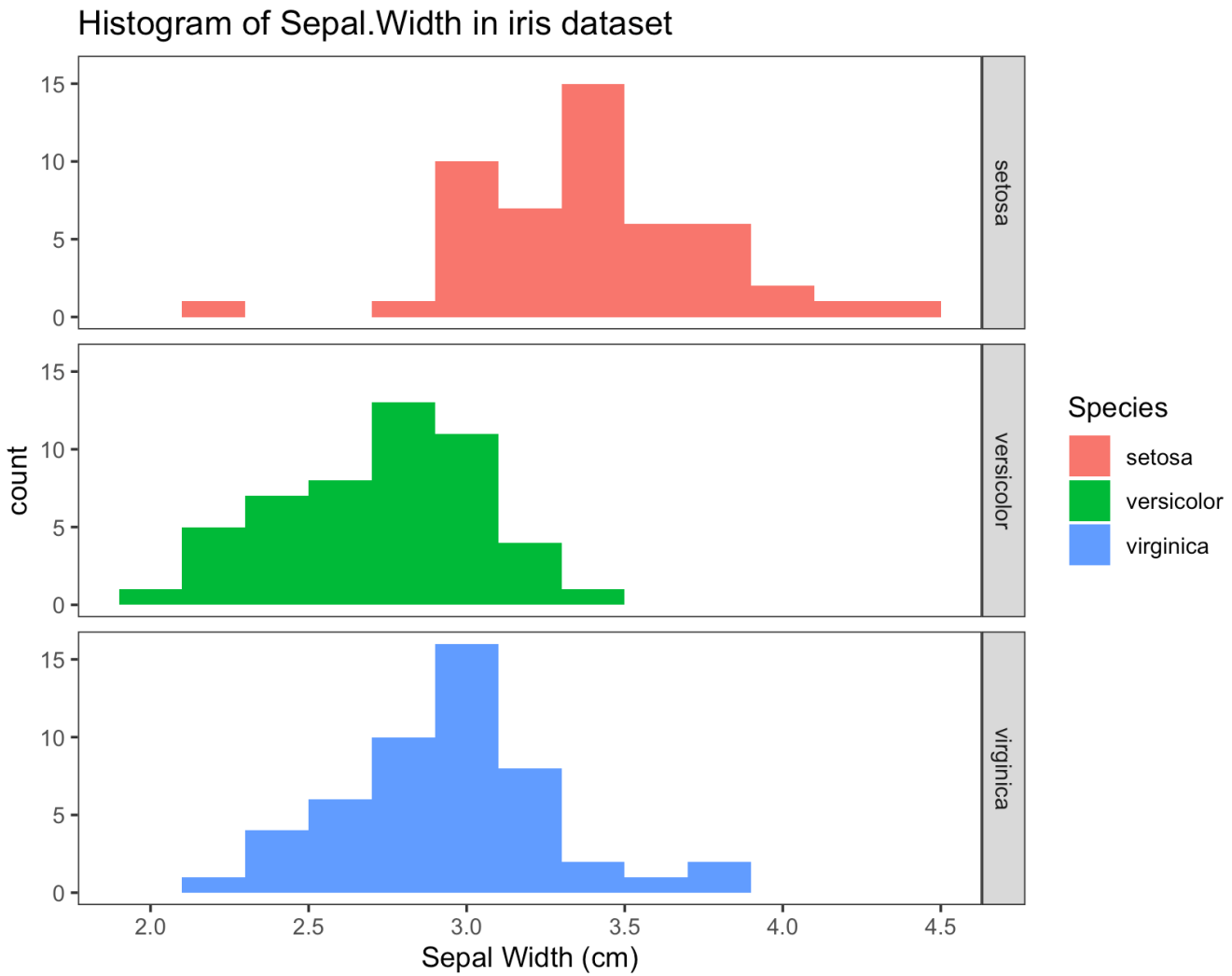


Figure: Output from plotting iris dataset as a histogram via ggplot2::ggplot().

R Packages for Plotting

```
# An example of annotating plots
?faithful

plotRegular <- ggplot2::ggplot(faithful,
                  aes(x = eruptions, y = waiting)) +
                  geom_point() +
                  ggtitle("Plot of old faithful geyser data") +
                  theme_bw() +
                  theme(panel.grid.major = element_blank(),
                        panel.grid.minor = element_blank())

plotRegular +
  annotate("text", x = 3, y = 48, label = "Group 1") +
  annotate("text", x = 4.5, y = 66, label = "Group 2")
```

R Packages for Plotting

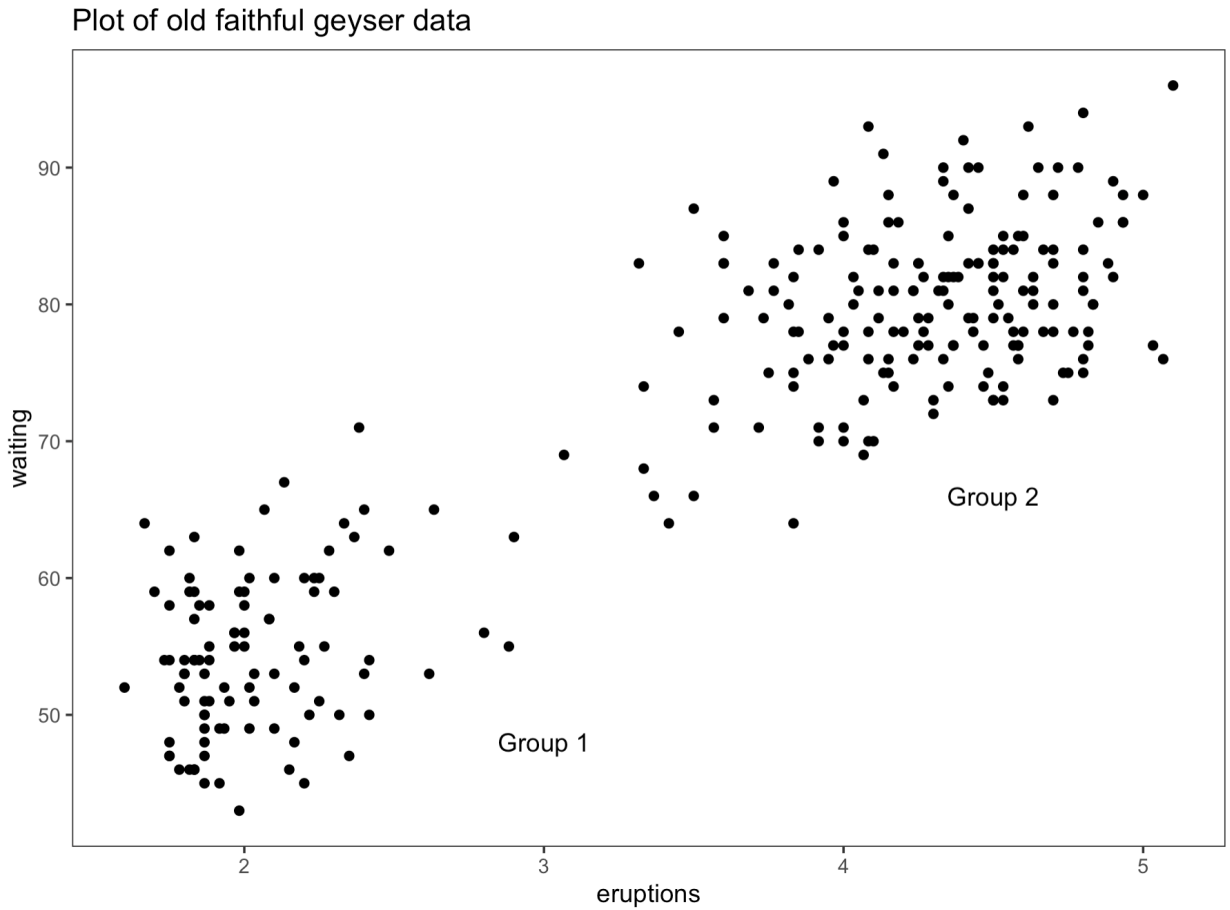


Figure: Output from plotting old faithful geyser dataset via ggplot2::ggplot().

R Packages for Plotting

- "ggpubr" is a package for publication ready plots.

```
# install.packages("ggpubr")
library(ggpubr)

# install.packages("EnvStats")
library(EnvStats)

meltIrisData %>%  # setosa species only
  dplyr::filter(Species == "setosa") %>%
  ggpubr::ggboxplot(x = "variable", y = "value",
                    fill = "variable", add = "jitter",
                    ylab = "Measurements (cm)",
                    font.label = list(size = 20, color = "black")) +
  ggtitle("Setosa species measurements") +
  ggpubr::stat_compare_means() +
  EnvStats::stat_n_text()
```

R Packages for Plotting

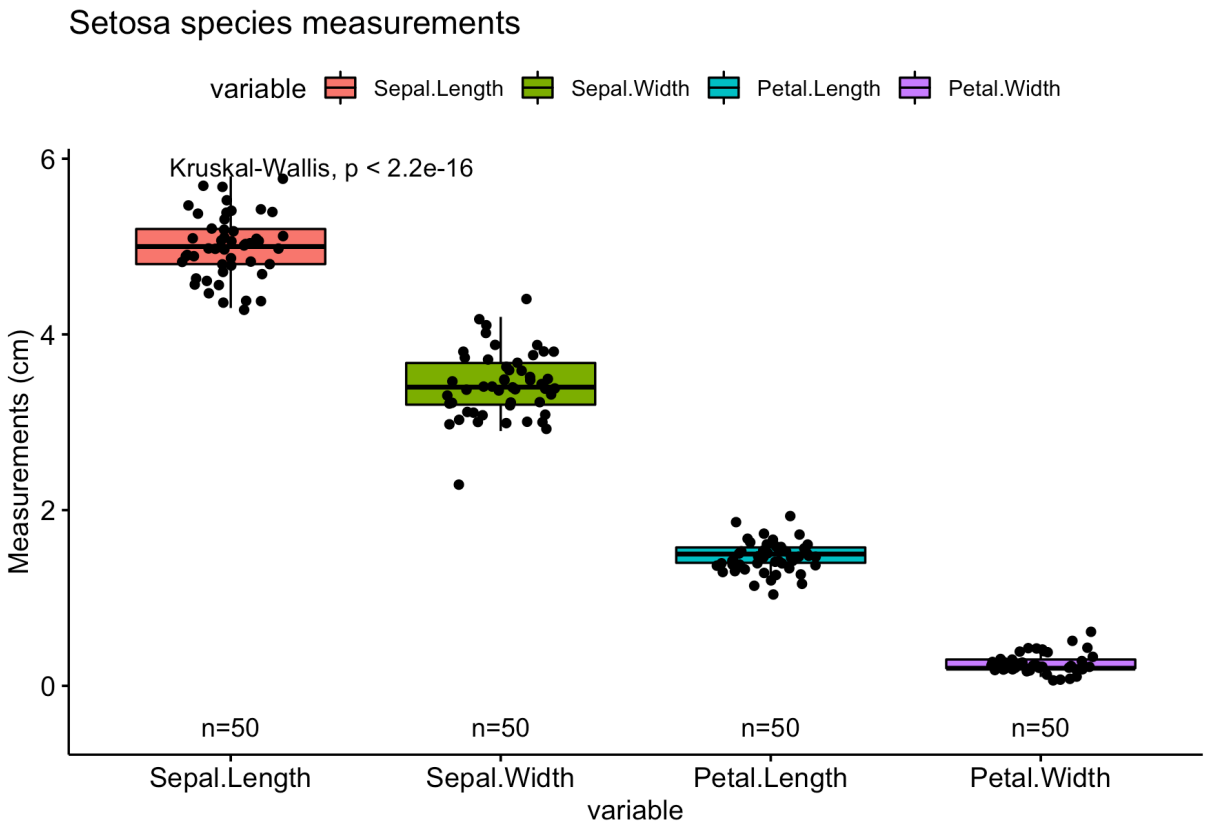


Figure: Output from plotting iris dataset via ggpubr::ggboxplot().

R Packages for Plotting

- More resources:
  - R Graphics Cookbook: https://r-graphics.org/.
  - sthda: http://www.sthda.com/english/wiki/ggplot2-essentials.

Next Steps

Any questions?

Next Steps

Resources

- The R Journal: https://journal.r-project.org/
- R for Data Science textbook (free): https://r4ds.had.co.nz/.
- Bioconductor Workshops: https://www.bioconductor.org/help/events/
- Coursera offers courses via both paid + free options
  - https://www.coursera.org/courses?query=bioinformatics

Resources

R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

Figure: Tidyverse R Packages designed for data science. Figure from https://www.tidyverse.org/.