Today you’ll learn how to:

Load datasets Scrape Webpages Build REST APIs

Analyze Data and Show Statistical Summaries Visualize Data

Train a Machine Learning Model Develop Simple Web Applications

# Load datasets

To perform any sort of analysis, you first have to load in the data. With R, you can connect to any data so For a simple demonstration, we’ll see how to load in CSV data. You can find the Iris dataset in CSV form

iris <- read.csv("iris.csv") head(iris)

And here’s what the head function outputs – the first six rows:

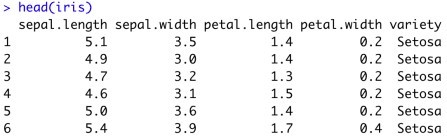


Image 1 – Iris dataset head

Did you know there’s no need to download the dataset? You can load it from the web:

iris <- read.csv("https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21398ea43cba4f7 d0e07d5ae3/iris.csv")

head(iris)

That’s all great, but what if you can’t find an appropriate dataset? That’s where web scraping comes into

# Web scraping

A good dataset is difficult to find, so sometimes you have to be creative. Web scraping is considered as o In R, the rvest package is used for the task. As some websites have strict policies against scraping, we library(rvest)

url <- "<http://books.toscrape.com/catalogue/category/books/travel_2/index.html>" titles <- read\_html(url) %>%

html\_nodes("h3") %>% html\_nodes("a") %>% html\_text()

The titles variable contains the following elements:



Image 2 – Web Scraping example in R

Yes – it’s that easy. Just don’t cross any boundaries. Check if a website has a public API first – if so, ther

# Build REST APIs

With practical machine learning comes the issue of model deployment. Currently, the best option is to wr In R, the plumber package is used to build REST APIs. Here’s the one that comes in by default when yo library(plumber)

#\* @apiTitle Plumber Example API

#\* Echo back the input

#\* @param msg The message to echo #\* @get /echo

function(msg = "") {

list(msg = paste0("The message is: '", msg, "'"))

}

#\* Plot a histogram #\* @png

#\* @get /plot function() {

rand <- rnorm(100) hist(rand)

}

#\* Return the sum of two numbers

#\* @param a The first number to add #\* @param b The second number to add #\* @post /sum

function(a, b) {

as.numeric(a) + as.numeric(b)

}

The API has three endpoints:

1. /echo – returns a specified message in the response
2. /plot – shows a histogram of 100 random normally distributed numbers
3. /sum – sums two numbers

The plumber package comes with Swagger UI, so you can explore and test your API in the web browse

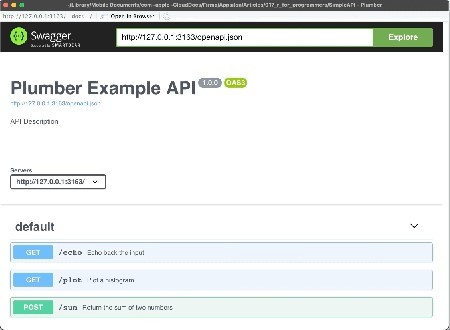


Image 3 – Plumber REST API Showcase

# Statistics and Data Analysis

This is one of the biggest reasons why R is so popular. There are entire books and courses on this topic, Most of the data manipulation in R is done with the dplyr package. Still, we need a dataset to manipulat library(dplyr)

library(gapminder)

head(gapminder)

You should see the following in the console:

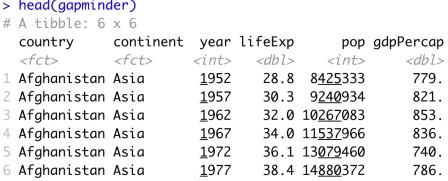


Image 4 – Head of Gapminder dataset

To perform any kind of statistical analysis, you could use R’s built-in functions such as min, max, range, summary(gapminder)

Here’s a statistical summary of the Gapminder dataset:

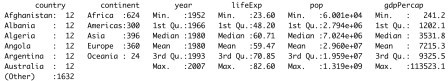


Image 5 – Statistical summary of the Gapminder dataset

With dplyr, you can drill down and keep only the data of interest. Let’s see how to show only data for P

gapminder %>%

filter(continent == "Europe", country == "Poland") %>% mutate(TotalGDP = pop \* gdpPercap)

The corresponding results are shown in the console:

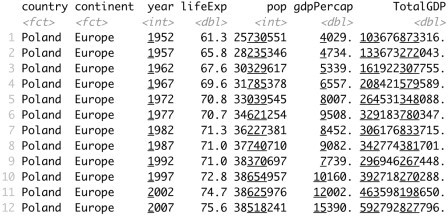


Image 6 – History data and total GDP for Poland

# Data Visualization

R is known for its impeccable data visualization capabilities. The ggplot2 package is a good starting po To start, we will create a line chart comparing the total population in Poland over time. We will need to filt

library(dplyr) library(gapminder) library(scales) library(ggplot2)

poland <- gapminder %>%

filter(continent == "Europe", country == "Poland")

ggplot(poland, aes(x = year, y = pop)) + geom\_line(size = 2, color = "#0099f9") + ggtitle("Poland population over time") + xlab("Year") +

ylab("Population") +

expand\_limits(y = c(10^6 \* 25, NA)) + scale\_y\_continuous(

labels = paste0(c(25, 30, 35, 40), "M"),

breaks = 10^6 \* c(25, 30, 35, 40)

) +

theme\_bw()

Here is the corresponding output:

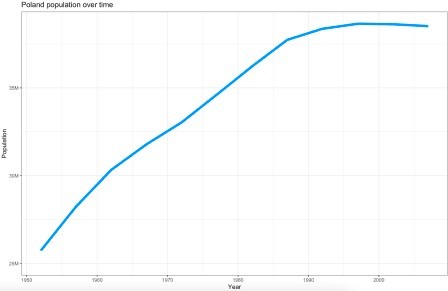


Image 7 – Poland population over time

You can get a similar visualization with the first two code lines – the others are added for styling.

The ggplot2 package can display almost any data visualization type, so let’s explore bar charts next. W

europe\_2007 <- gapminder %>%

filter(continent == "Europe", year == 2007)

ggplot(europe\_2007, aes(x = reorder(country, -lifeExp), y = lifeExp)) + geom\_bar(stat = "identity", fill = "#0099f9") +

geom\_text(aes(label = lifeExp), color = "white", hjust = 1.3) + ggtitle("Average life expectancy in Europe countries in 2007") + xlab("Country") +

ylab("Life expectancy (years)") + coord\_flip() +

theme\_bw()

Here’s how the chart looks like:

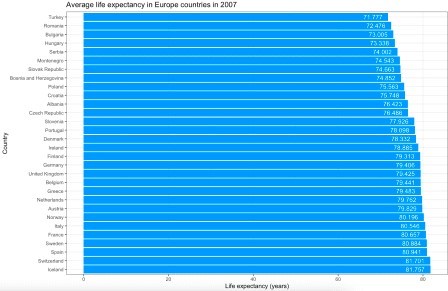


Image 8 – Average life expectancy in European countries in 2007

Once again, the first two code lines for the visualization will produce similar output. The rest are here to

# Training a Machine Learning Model

Yet another area that R handles with ease. The rpart package is great for machine learning, and we wil Here’s how to load in the libraries, perform the train/test split, fit and visualize the model:

library(caTools) library(rpart) library(rpart.plot)

set.seed(42)

sample <- sample.split(iris, SplitRatio = 0.75) iris\_train = subset(iris, sample == TRUE) iris\_test = subset(iris, sample == FALSE)

model <- rpart(Species ~., data = iris\_train, method = "class" rpart.plot(model)

The snippet shouldn’t take more than a second or two to execute. Once done, you’ll be presented with th

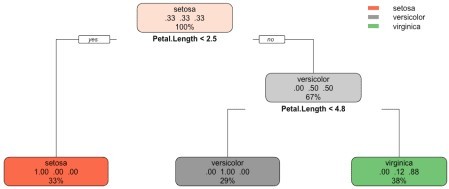


Image 9 – Decision tree visualization for Iris dataset

The above figure tells you everything about the decision-making process of the algorithm. We can now e

preds <- predict(model, iris\_test, type = "class")

confusion\_matrix <- table(iris\_test$Species, preds) print(confusion\_matrix)

accuracy <- sum(diag(confusion\_matrix)) / sum(confusion\_matrix) print(accuracy)

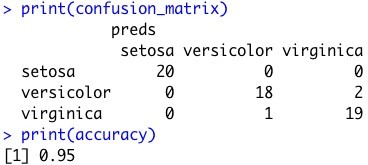


Image 10 – Confusion matrix and accuracy on the test subset

As you can see, we got a 95% accurate model with only a couple of lines of code.

# Develop Simple Web Applications

Here is a script for the Shiny app:

library(shiny) library(ggplot2)

ui <- fluidPage( sidebarPanel(

width = 3, tags$h4("Select"), varSelectInput(

inputId = "x\_select", label = "X-Axis", data = mtcars

),

varSelectInput(

inputId = "y\_select", label = "Y-Axis", data = mtcars

)

),

mainPanel(

plotOutput(outputId = "scatter")

)

)

server <- function(input, output) {

output$scatter <- renderPlot({ col1 <- sym(input$x\_select) col2 <- sym(input$y\_select)

ggplot(mtcars, aes(x = !!col1, y = !!col2)) + geom\_point(size = 6, color = "#0099f9") + ggtitle("MTCars Dataset Explorer") + theme\_bw()

})

}

shinyApp(ui = ui, server = server)

And here’s the corresponding Shiny app:

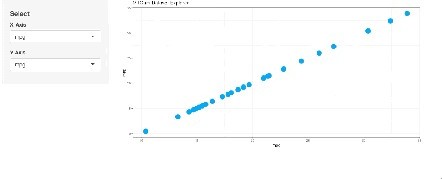


Image 11 – MTCars Shiny app

This dashboard is as simple as they come, but that doesn’t mean you can’t develop beautiful-looking app

# Conclusion

To conclude – R can do almost anything that a general-purpose programming language can do.