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

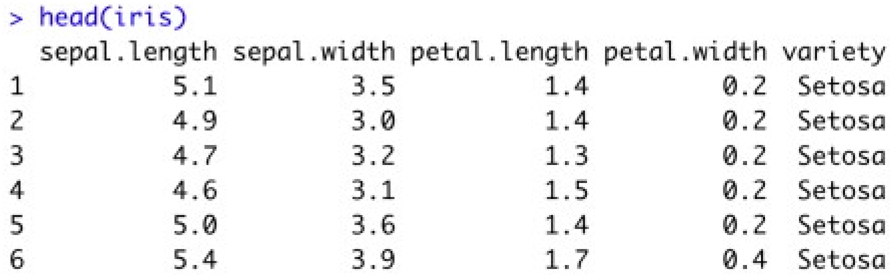
Load datasets

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 forma

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

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



```
> head(iris)
  sepal.length sepal.width petal.length petal.width variety
1          5.1         3.5          1.4         0.2  Setosa
2          4.9         3.0          1.4         0.2  Setosa
3          4.7         3.2          1.3         0.2  Setosa
4          4.6         3.1          1.5         0.2  Setosa
5          5.0         3.6          1.4         0.2  Setosa
6          5.4         3.9          1.7         0.4  Setosa
```

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

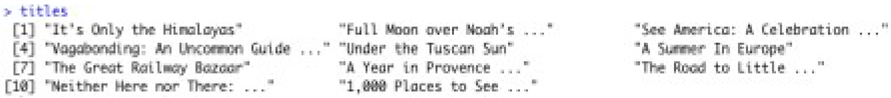
Web scraping

A good dataset is difficult to find, so sometimes you have to be creative. Web scraping is considered as c

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:



```
> titles
 [1] "It's Only the Himalayas"          "Full Moon over Noah's ..."        "See America: A Celebration ..."
 [4] "Vagabonding: An Uncommon Guide ..." "Under the Tuscan Sun"           "A Summer In Europe"
 [7] "The Great Railway Bazaar"         "A Year in Provence ..."           "The Road to Little ..."
[10] "Neither Here nor There: ..."      "1,000 Places to See ..."
```

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, there

Build REST APIs

With practical machine learning comes the issue of model deployment. Currently, the best option is to wra

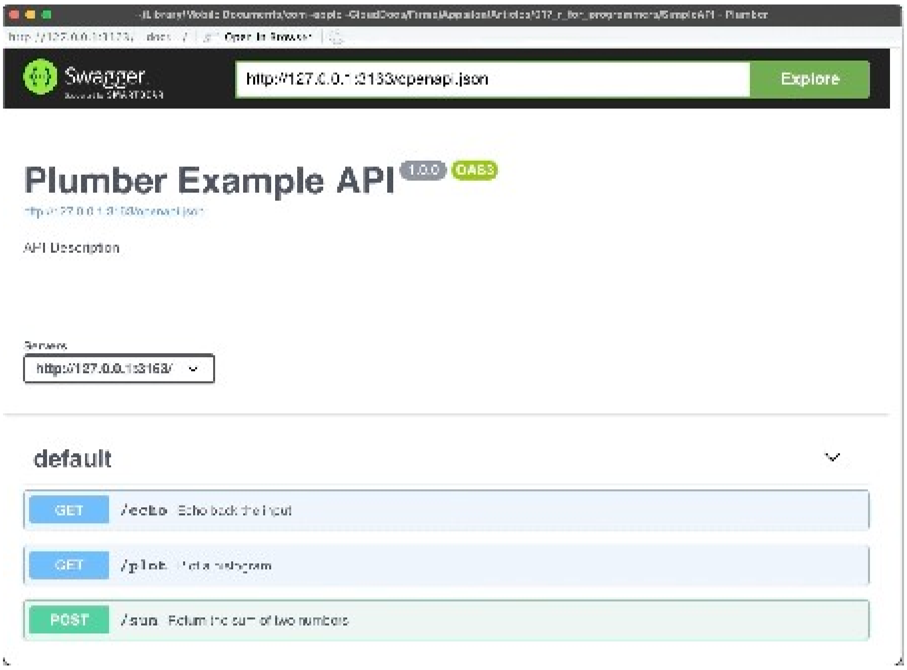
In R, the `plumber` package is used to build REST APIs. Here’s the one that comes in by default when yo

```
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



Plumber Example API

API Description

/echo Echo back the input

/plot Plot a histogram

/sum Return the sum of two numbers

Image 3 – Plumber REST API Showcase

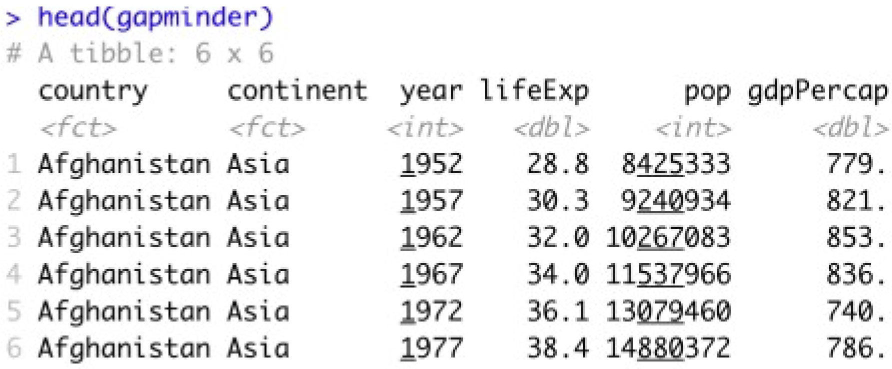
Statistics and Data Analysis

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:



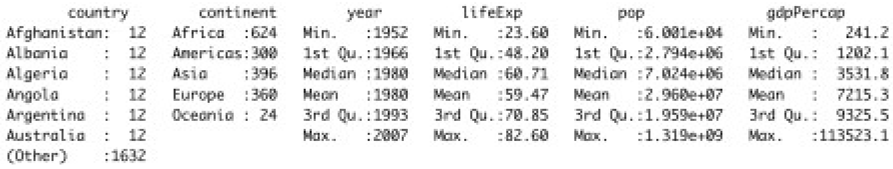
```
> head(gapminder)
# A tibble: 6 x 6
  country     continent  year lifeExp      pop gdpPercap
  <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
1 Afghanistan Asia       1952    28.8  8425333      779.
2 Afghanistan Asia       1957    30.3  9240934      821.
3 Afghanistan Asia       1962    32.0 10267083      853.
4 Afghanistan Asia       1967    34.0 11537966      836.
5 Afghanistan Asia       1972    36.1 13079460      740.
6 Afghanistan Asia       1977    38.4 14880372      786.
```

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:



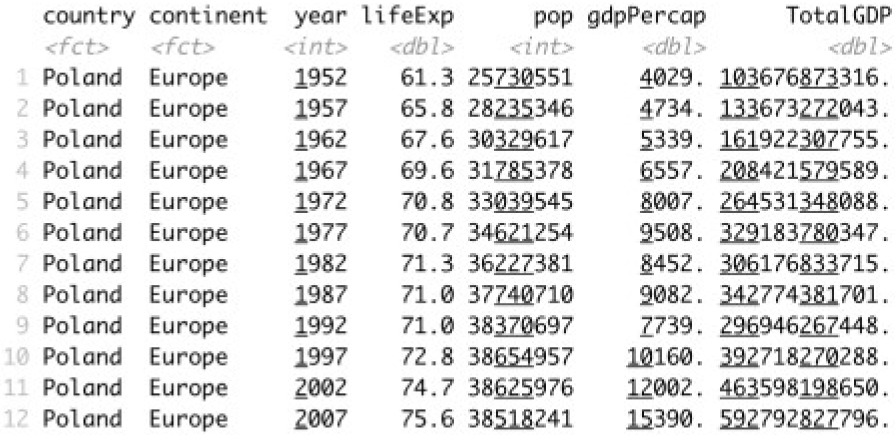
```
      country        continent        year          lifeExp           pop              gdpPercap
 Afghanistan:  12   Africa  :624   Min.   :1952   Min.   :23.60   Min.   :6.001e+04   Min.   :   241.2
 Albania    :  12   Americas:300   1st Qu.:1966   1st Qu.:48.20   1st Qu.:2.794e+06   1st Qu.:  1202.1
 Algeria    :  12   Asia    :396   Median :1980   Median :60.71   Median :7.024e+06   Median :  3531.8
 Angola     :  12   Europe  :360   Mean   :1980   Mean   :59.47   Mean   :2.960e+07   Mean   :  7215.3
 Argentina  :  12   Oceania : 24   3rd Qu.:1993   3rd Qu.:70.85   3rd Qu.:1.959e+07   3rd Qu.:  9325.5
 Australia  :  12                  Max.   :2007   Max.   :82.60   Max.   :1.319e+09   Max.   :113523.1
 (Other)    :1632
```

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 Po

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

The corresponding results are shown in the console:



```
   country continent  year lifeExp      pop gdpPercap      TotalGDP
   <fct>   <fct>     <int>   <dbl>    <int>     <dbl>         <dbl>
 1 Poland  Europe     1952    61.3 25730551     4029. 103676873316.
 2 Poland  Europe     1957    65.8 28235346     4734. 133673272043.
 3 Poland  Europe     1962    67.6 30329617     5339. 161922307755.
 4 Poland  Europe     1967    69.6 31785378     6557. 208421579589.
 5 Poland  Europe     1972    70.8 33039545     8007. 264531348088.
 6 Poland  Europe     1977    70.7 34621254     9508. 329183780347.
 7 Poland  Europe     1982    71.3 36227381     8452. 306176833715.
 8 Poland  Europe     1987    71.0 37740710     9082. 342774381701.
 9 Poland  Europe     1992    71.0 38370697     7739. 296946267448.
10 Poland  Europe     1997    72.8 38654957    10160. 392718270288.
11 Poland  Europe     2002    74.7 38625976    12002. 463598198650.
12 Poland  Europe     2007    75.6 38518241    15390. 592792827796.
```

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 filte

```
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:

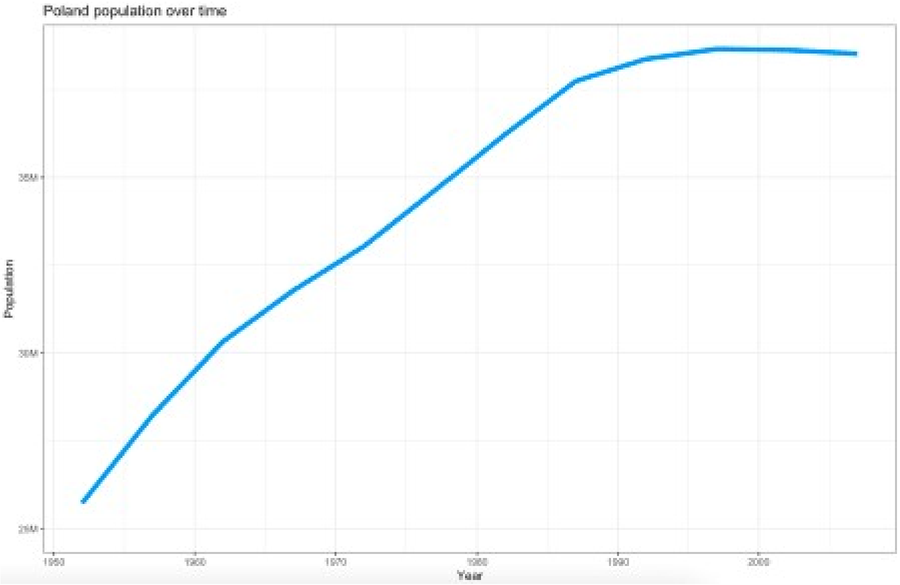


Image 7 – Poland population over time

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:

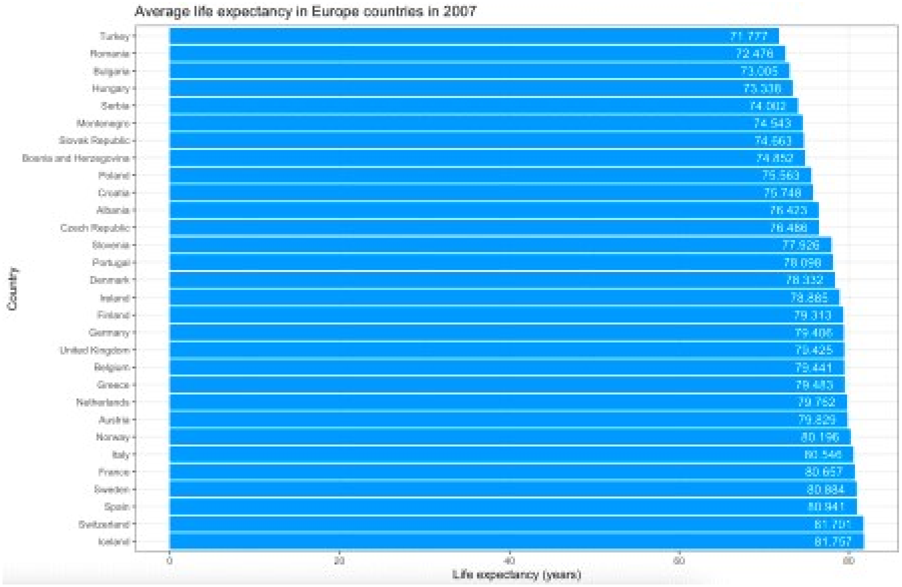


Image 8 – Average life expectancy in European countries in 2007

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

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

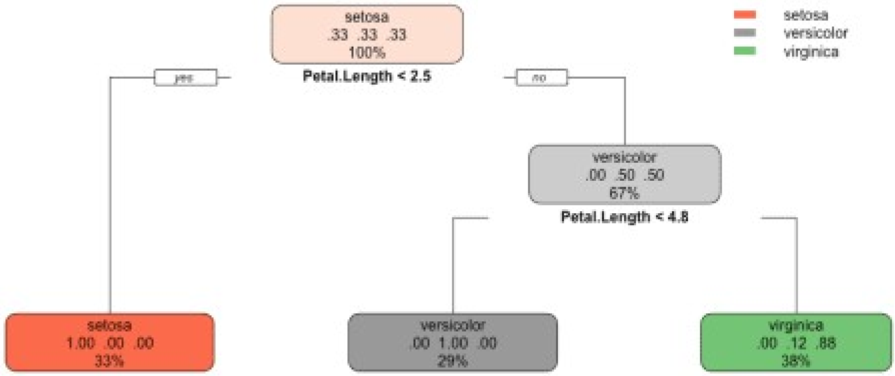
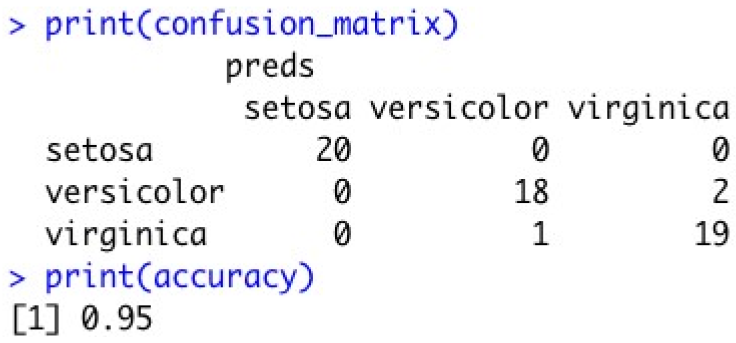


Image 9 – Decision tree visualization for Iris dataset

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

```
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)
```



```
> print(confusion_matrix)
            preds
             setosa versicolor virginica
  setosa         20          0         0
  versicolor      0         18         2
  virginica       0          1        19
> print(accuracy)
[1] 0.95
```

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

At Appsilon, we are global leaders in R Shiny, and we've developed some of the world's most advanced R

For the web app example, we'll see how to make simple interactive dashboards that displays a scatter pl

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:

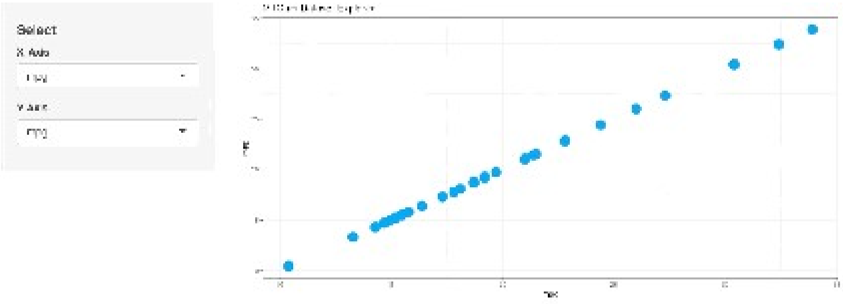


Image 11 – MTCars Shiny app

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

> Looking for inspiration? Take a look at our Shiny App Demo Gallery.

Conclusion

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