The Scenario

Assume that you have built a Machine Learning model and you other people to be able to interact with it. So, let's say that you want to build a tool that **takes as an input a CSV file** of the data that you want to get predictions and the **output will be the predicted values**.

Build the Model

For exhibition purposes, we will work with the `iris` dataset by building a `multinomial logistic regression` model. Once we build the model, we will save it and we will use it later.

```
library(nnet)

irisModel<-multinom(Species~Sepal.Length+Sepal.Width+
Petal.Length+Petal.Width,data = iris)

saveRDS(irisModel, "irisModel.rds")
```

So, we built the logistic regression model and we saved it as "irisModel.rds".

Build the Shiny App

The Shiny App will do the following things:

- Load the `irisModel.rds` model
- Ask the users to upload a csv file of the data that they want to predict
- Run the predictions on the data
- Return the predictions on the UI and give them the opportunity to download them in a csv format.

On purpose, the Shiny App will be as simple as possible. In another post, we will provide tutorials of how you can build advanced Shiny Apps.

For now, we can open the R Studio, **File–>New File–>Shiny Web App**. Then, you should choose a name for your Shiny Web App and it will create a folder and a sample code file. Within the folder, store the `irisModel.rds` object. The code is the following:

```
library(shiny)
library(DT)
library(tidyverse)
irisModel <- readRDS("irisModel.rds")

# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("Iris Dataset Predictions"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            # Input: Select a file ----
            fileInput("file1", "upload csv file here",
                      multiple = FALSE,
                      accept = c("text/csv",
                                 "text/comma-separated-values,
text/plain",
                                 ".csv")),


            # Button
            downloadButton("downloadData", "Download the Predictions")
        ),

        # Show the table with the predictions
        mainPanel(
            DT::dataTableOutput("mytable")
        )
    )
)

# Define server logic required to draw a histogram
server <- function(input, output) {


    reactiveDF<-reactive({
        req(input$file1)
        df <- read.csv(input$file1$datapath, stringsAsFactors = TRUE)

        df$predictions<-predict(irisModel, newdata = iris, type
="class")
        return(df)

    })

    output$mytable = DT::renderDataTable({
        req(input$file1)

        return(DT::datatable(reactiveDF(),  options = list(pageLength =
100), filter = c("top")))
    })


    # Downloadable csv of selected dataset ----
    output$downloadData <- downloadHandler(
        filename = function() {
            paste("data-", Sys.Date(), ".csv", sep="")
        },
        content = function(file) {
            write.csv(reactiveDF(), file, row.names = FALSE)
        }
    )
```

```
}

# Run the application
shinyApp(ui = ui, server = server)
```

Explanation of the Code

ui

Within the `sidebarLayout` we have the `sidebarPanel` and the `mainPanel`. Let’s have a closer look at these parts:

`sidebarPanel`

We have the `fileInput` that allows us to upload a CSV file and the `downloadButton` which is the button to download the predictions.

`mainPanel`

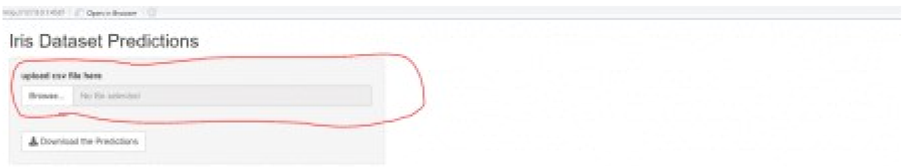
In the main panel we return in the UI the predictions as a DT table.

sever

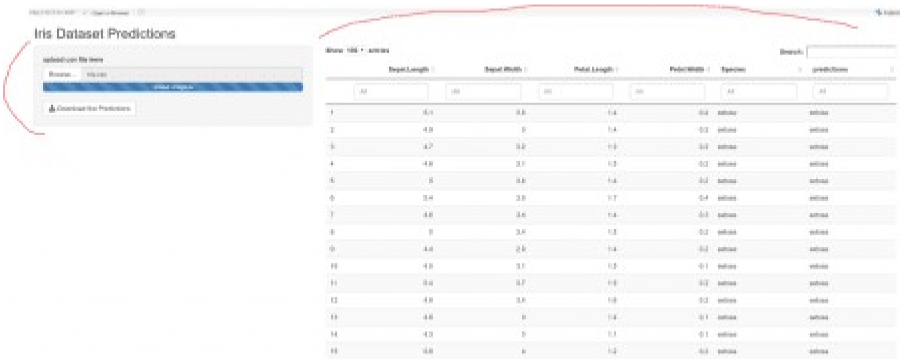
We have the `reactiveDF` which is a reactive object and starts working once there is an input file, that is why we used the command `req(input$file1)`. Then we read the CSV file and we add the predictions in an extra column. Then we have to return this reactive object and that is why we created the `output$mytable`, Notice that the `mytable` was defined in the UI part. Finally, we define the `output$downloadData` object which downloads the `reactiveDF` once we press the `downloadButton`. Notice that we the `downloadData` was defined in the UI part.

Get the Predictions

I will upload the `iris.csv`. Notice that this file contains already the species. It does not matter since our model gets as input only the features and its adds an extra column for the predictions. So, it will help us to see if it returns right predictions.


Iris Dataset Predictions

Once we upload the file we get the predictions:



Iris Dataset Predictions

upload csv file here

Browse... iris.csv

Upload complete

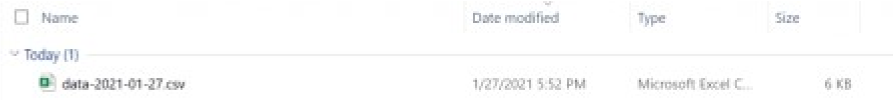
Download the Predictions

Show 100 entries

Search:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	predictions
	All	All	All	All	All	All
1	5.1	3.5	1.4	0.2	setosa	setosa
2	4.9	3	1.4	0.2	setosa	setosa
3	4.7	3.2	1.3	0.2	setosa	setosa
4	4.6	3.1	1.5	0.2	setosa	setosa
5	5	3.6	1.4	0.2	setosa	setosa
6	5.4	3.9	1.7	0.4	setosa	setosa
7	4.6	3.4	1.4	0.3	setosa	setosa
8	5	3.4	1.5	0.2	setosa	setosa
9	4.4	2.9	1.4	0.2	setosa	setosa
10	4.9	3.1	1.5	0.1	setosa	setosa
11	5.4	3.7	1.5	0.2	setosa	setosa
12	4.8	3.4	1.6	0.2	setosa	setosa
13	4.8	3	1.4	0.1	setosa	setosa
14	4.3	3	1.1	0.1	setosa	setosa
15	5.8	4	1.2	0.2	setosa	setosa

And if we click on the **Download the Predictions** button we get the predictions in a csv file.



□ Name	Date modified	Type	Size
Today (1)			
data-2021-01-27.csv	1/27/2021 5:52 PM	Microsoft Excel C...	6 KB

Deployment

This Shiny App can be deployed to ShinyApps.io or if you have a server, you can run the R Studio on it. Using AWS is relatively easy to run R Studio…