Apps Shiny.

Cómo crear aplicaciones basadas en datos en 5 minutos

Ander Fernández Jauregui

¿Quién soy?

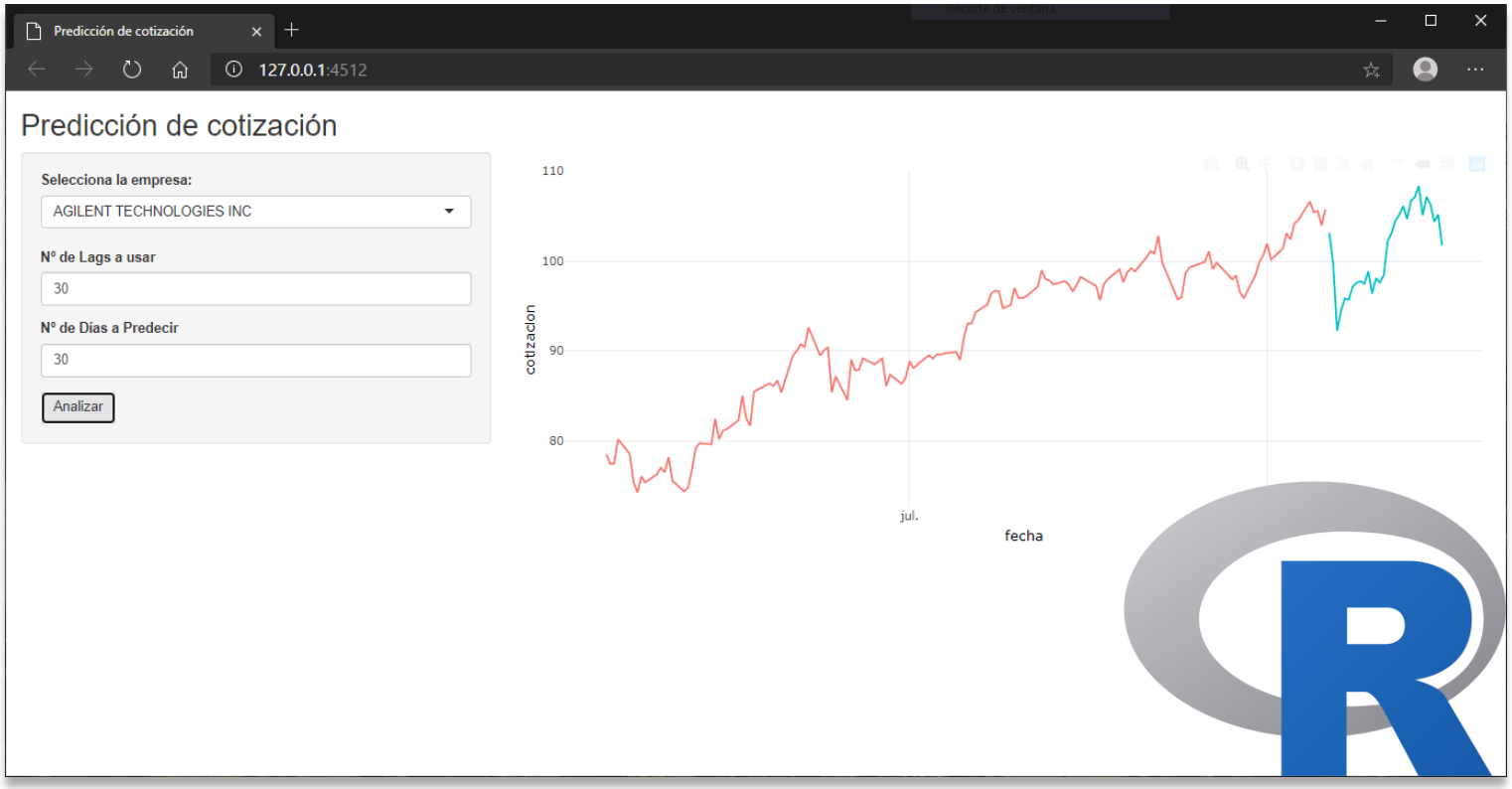
- Data Scientist en LIN3S
- Profesor en el Programa en Big Data & Business Intelligence en Deusto y en el Grado en ADE
- Ganador del Cajamar Universityhack 2020 en la categoría visualización de datos.
- Blog de Data Science: anderfernandez.com/blog

Índice

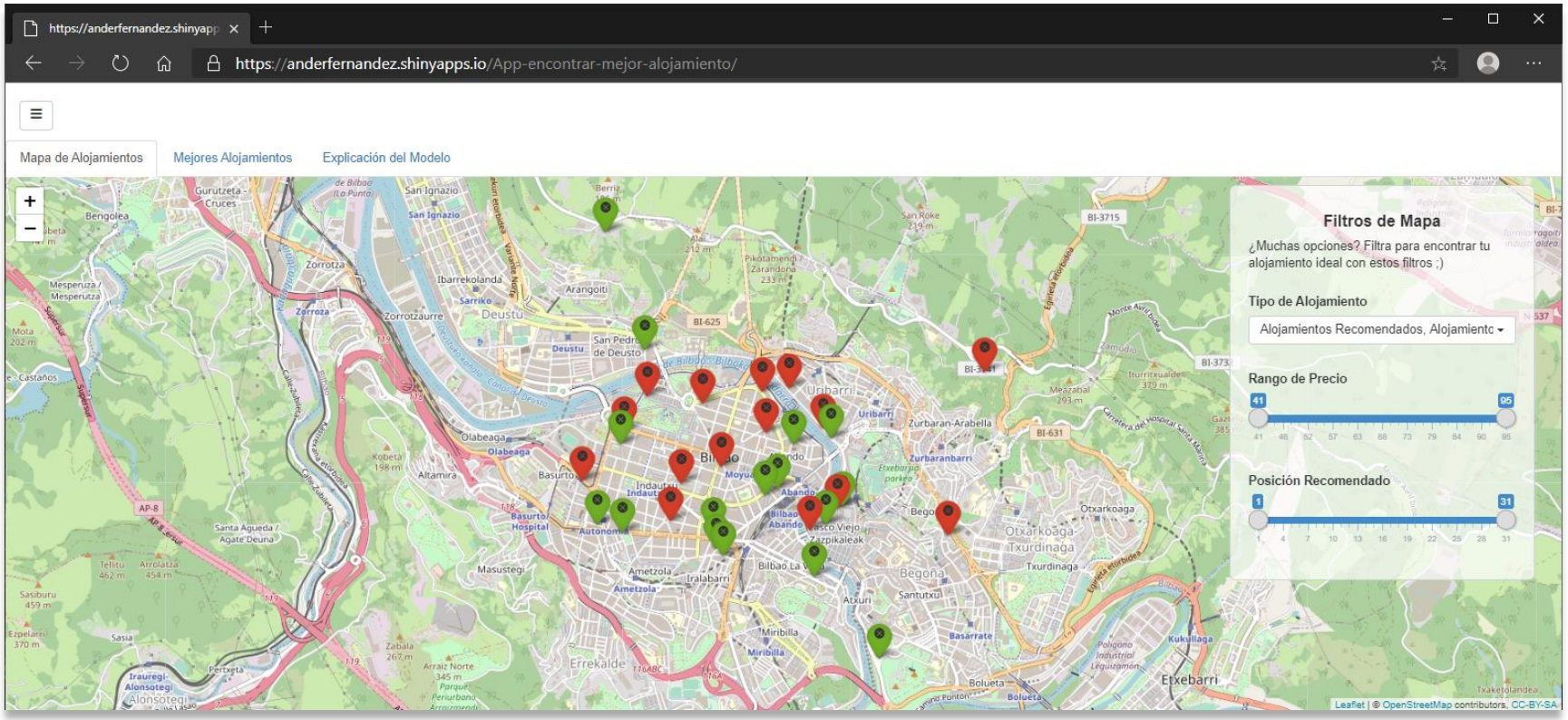
- Qué es Shiny y para qué sirve
- Cómo funciona Shiny
- Ejemplo de una app shiny
- Qué se puede llegar a hacer con Shiny

Índice

Shiny permite crear y publicar aplicaciones web que ejecutan código R con las que cualquier usuario puede interactuar



Con un solo lenguaje podemos crear aplicaciones que extraigan los datos mediante APIs, scraping o acceso a BBDD a entrenar y visualización de modelos de Machine Learning



Índice

Estructura de la App

Sección Inicial

En esta parte cargamos las librerías y datos que se mantengan estáticos desde la carga de la App.

```
library(shiny)
```

Estructura de la App

Sección Inicial

En esta parte cargamos las librerías y datos que se mantengan estáticos desde la carga de la App.

```
library(shiny)
```

User Interface

En esta parte crearemos la estructura HTML de la página, incluyendo: botones de acción, sliders, selectores, y todo lo que el usuario pueda introducir y que vayamos a usar en la página.

```
ui <- fluidPage(
  titlePanel(“Hello World!")
)
```

Estructura de la App

Sección Inicial

En esta parte cargamos las librerías y datos que se mantengan estáticos desde la carga de la App.

```
library(shiny)
```

User Interface

En esta parte crearemos la estructura HTML de la página, incluyendo: botones de acción, sliders, selectores, y todo lo que el usuario pueda introducir y que vayamos a usar en la página.

```
ui <- fluidPage(
  titlePanel(“Hello World!")
)
```

Server

Aquí es donde ejecutamos el código en R y generamos resultados y gráficos dinámicos que se visualizarán en la sección del UI.

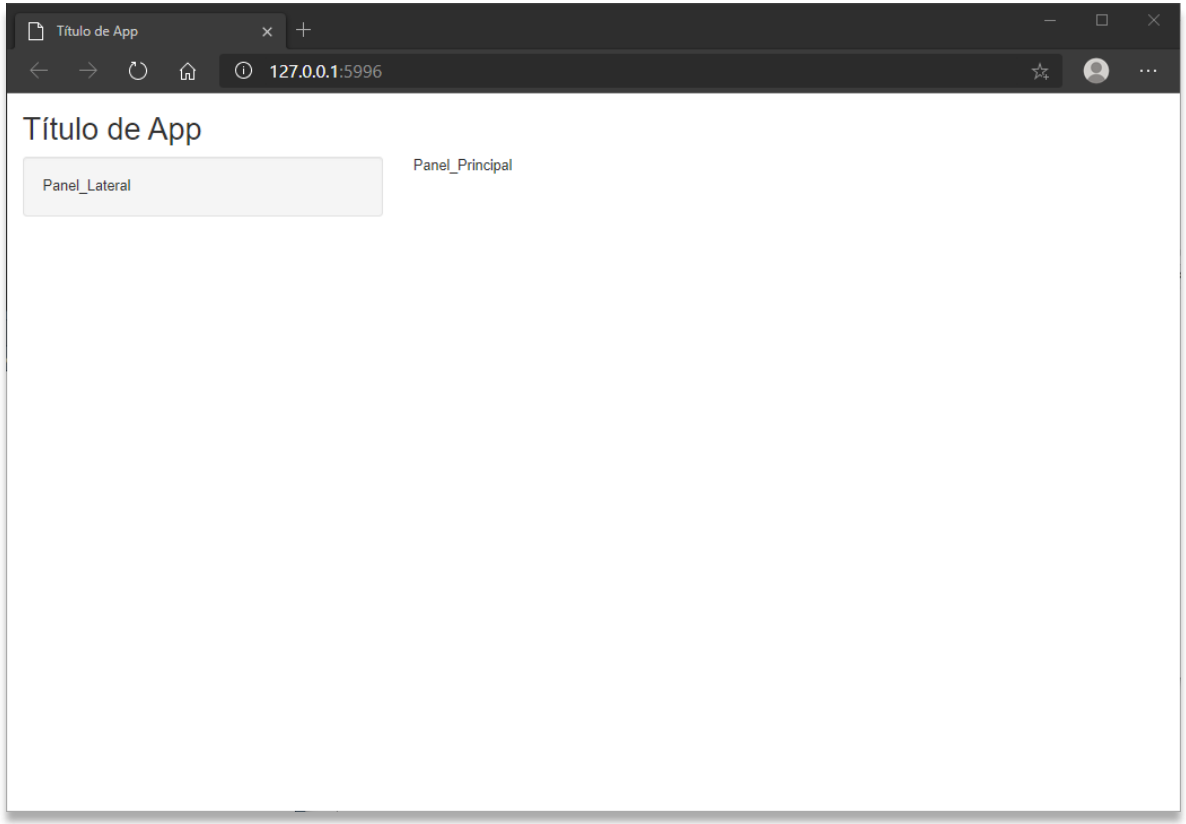
```
server <- function(input, output) {
}
```

Creando la Interfaz de Usuario

Aunque podríamos crear la estructura que quisiéramos o, incluso, cargar un documento HTML, Shiny ofrece funciones con estructuras básicas.

Ejemplo:

```
ui <- fluidPage(
   titlePanel('Título de App'),
   sidebarLayout(
        sidebarPanel('Panel_Lateral'),
        mainPanel('Panel_Principal')
)
)
```

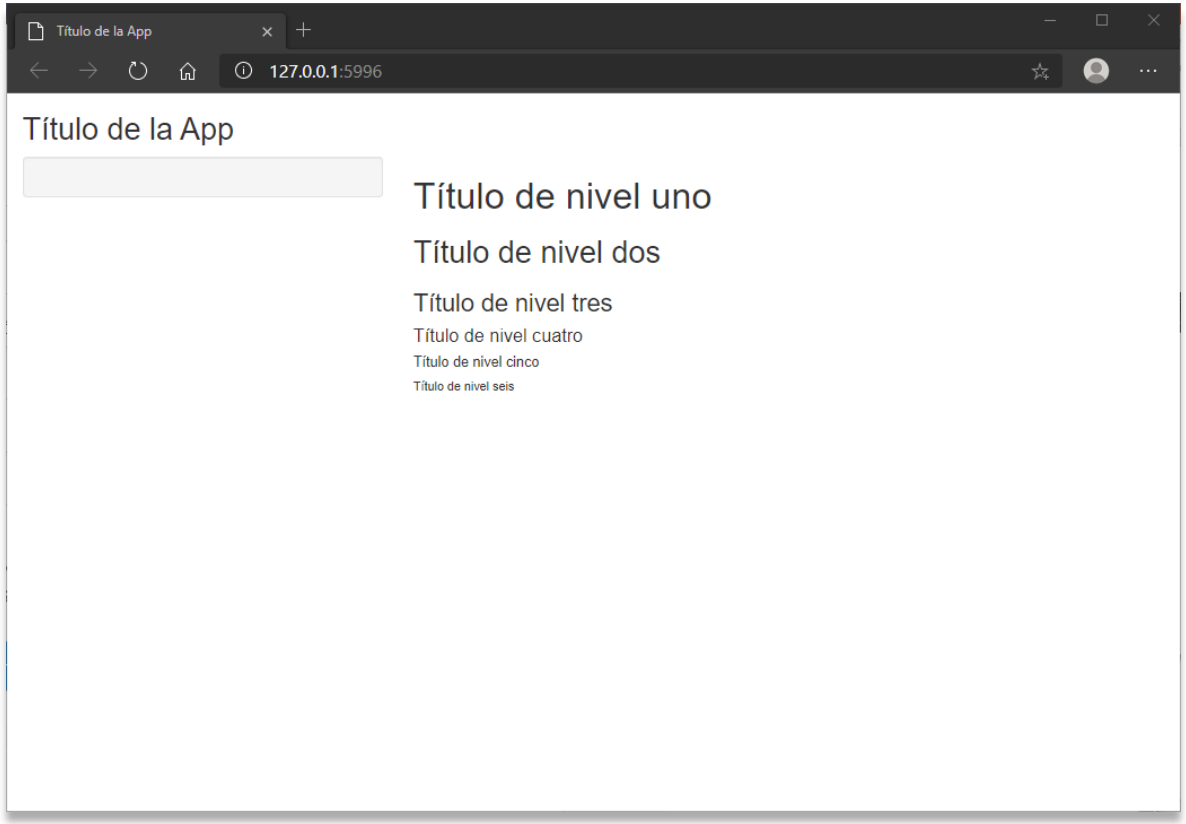


Creando la Interfaz de Usuario

Además, por si no sabemos de HTML, ofrece muchas funciones que permiten crear la UI en HTML5 directamente desde R.

Ejemplo:

```
ui <- fluidPage(
    titlePanel("Título de la App"),
    sidebarLayout(
        sidebarPanel(),
        mainPanel(
     h1("Título de nivel uno"),
     h2("Título de nivel dos"),
     h3("Título de nivel tres"),
     h4("Título de nivel cuatro"),
     h5("Título de nivel cinco"),
     h6("Título de nivel seis")
        )))
```

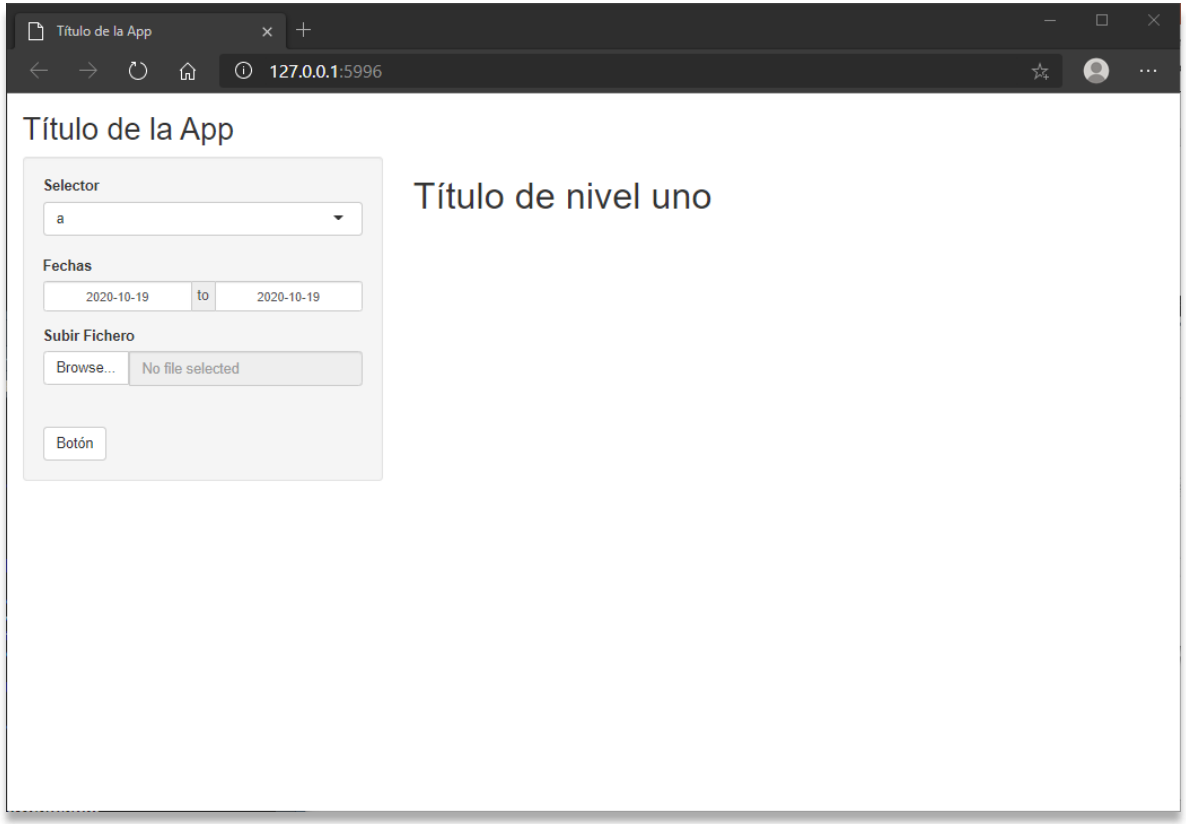


Creando la Interfaz de Usuario

Asimismo, podemos crear también elementos que sirvan como inputs del usuario, tales como: un botón de acción, un selector

Ejemplo:

```
ui <- fluidPage(
    titlePanel("Título de la App"),
    sidebarLayout(
        sidebarPanel(
selectInput("a","Selector",letters),
dateRangeInput("b","Fechas"),
fileInput("c","Subir Fichero"),
actionButton("d","Botón")
        ),
        mainPanel(
            h1("Título de nivel uno")
        )
```

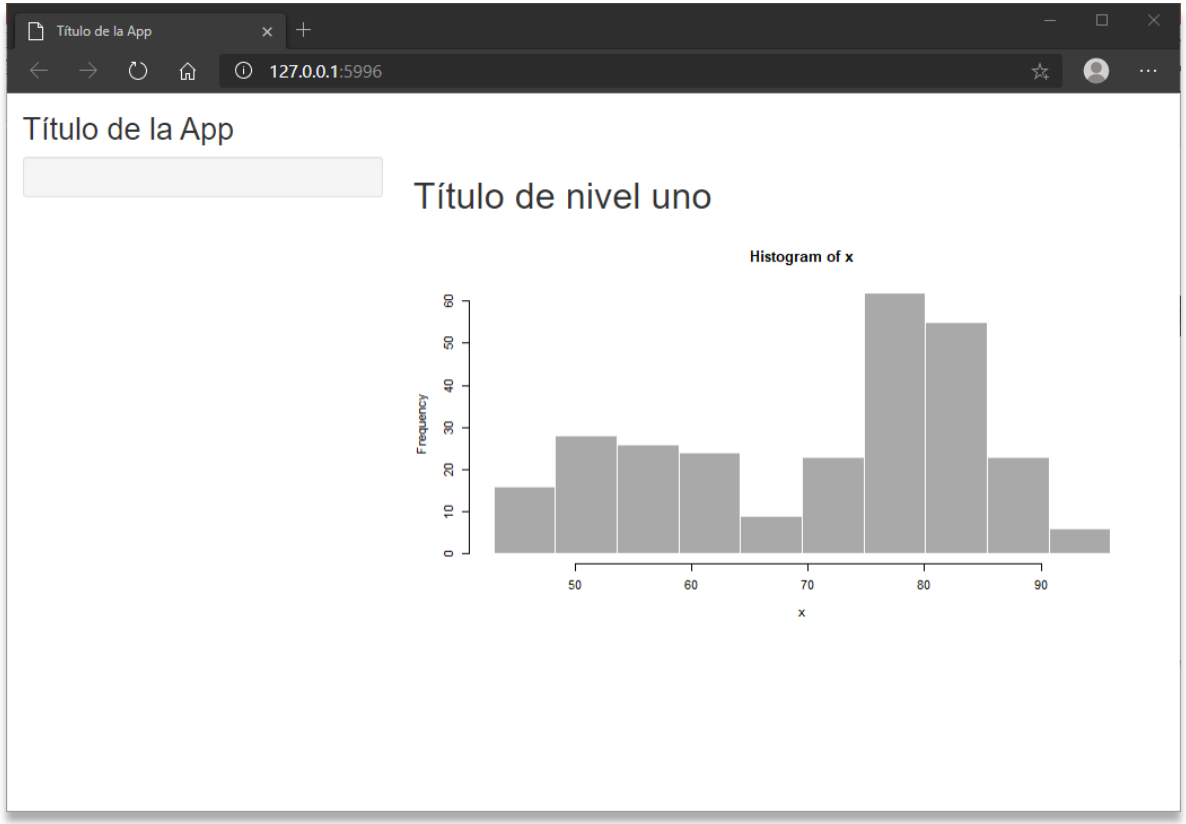


Creando la Interfaz de Usuario

Por último, tenemos que indicar dónde irán los elementos que creemos en el server. Esto lo haremos con las funciones de Output:

Ejemplo:

```
ui <- fluidPage(
    titlePanel("Título de la App"),
    sidebarLayout(
        sidebarPanel(),
        mainPanel(
            h1("Título de nivel uno"),
            plotOutput("distPlot")
        )
    )
)
```



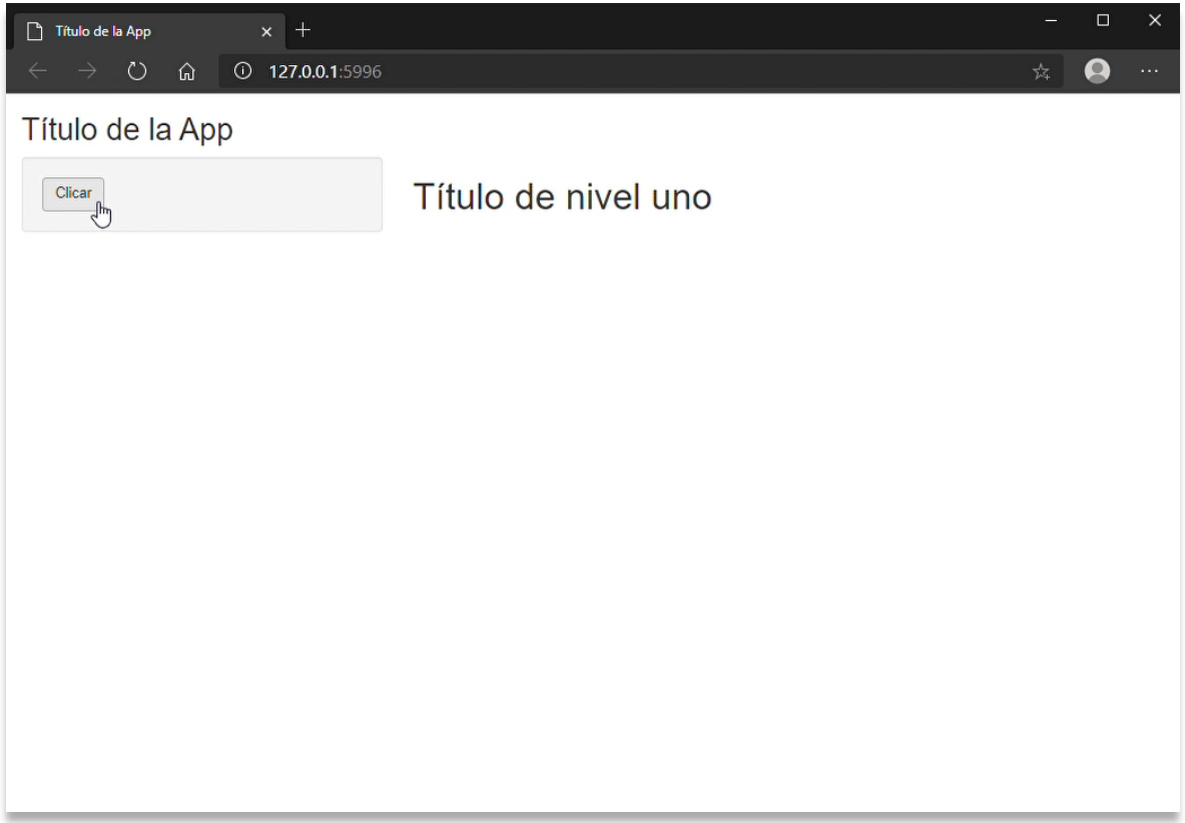
Creando el Servidor

- **Elementos reactivos**: son elementos que se actualizan cada vez que exista cualquier cambio en la UI u otro elemento reactivo o cuando cambie un elemento que nosotros indiquemos.

```
datos = eventReactive(input$accion,{
        x = sample(50)
})
```

- **Elementos de Render**: consiste en renderizar un elemento que después mostraremos en la UI. En el caso anterior, si cuando usamos `plotOutput` se visualizó un gráfico es porque ese gráfico se había renderizado en el server.

```
output$texto = renderText(
         datos()
    )
```



Código del Ejemplo Anterior

```
library(shiny)

ui <- fluidPage(
    titlePanel("Título de la App"),
    sidebarLayout(
        sidebarPanel(
            actionButton("accion","Clicar")
        ),
        mainPanel(
            h1("Título de nivel uno"),
            textOutput("texto")
        )
    )
)
```

```
server <- function(input, output) {

    datos = eventReactive(input$accion,{
        x = sample(50)
    })

    output$texto = renderText(
        datos()
    )
shinyApp(ui = ui, server = server)
```

Índice

- Qué es Shiny y para qué sirve
- Cómo funciona Shiny
- **Ejemplo de una app shiny**
- Qué se puede llegar a hacer con Shiny

Conocimientos Previos

- Conocimiento de hacer peticiones a APIs desde R para acceder a los datos financieros utilizaremos la API de Finnhub.
  Si no sabes cómo funcionan las APIs, puedes aprender a utilizarlas en este webinar.
- Manejo del paquete de manipulación de datos `dplyr` y el paquete de visualización de datos `ggplot2`.

Estructura de la App

Parte Previa

- Cargamos librerías.
- Hacemos una llamada a la API para extraer los valores bursátiles que nos ofrecen.
- Nos quedamos con aquellos valores que tienen el nombre de la empresa relleno.

UI

- Creamos un selector que tenga como opciones los valores que hemos extraído.
- Creamos un selector numérico para personalizar el número de lags a tener en cuenta.
- Creamos un selector numérico para personalizar el número de días a predecir.
- Incluimos el output de la visualización del gráfico que vamos a mostrar

Server

- Extraemos los datos de la cotización para el valor elegido en el selector y lo convertimos en reactivo.
- Creamos un objeto reactivo que usa los datos de cotización, hace la predicción y devuelve los datos previos y los predichos.
- Creamos y renderizamos una visualización usando el objeto reactivo previo.

Estructura de la App

Parte Previa

- Cargamos librerías.
- Hacemos una llamada a la API para extraer los valores bursátiles que nos ofrecen.
- Nos quedamos con aquellos valores que tienen el nombre de la empresa relleno.

```
# Cargamos librerías
libs = c("httr","shiny","jsonlite","ggplot2","plotly","forecast","dplyr")
for(i in libs){if (!i %in% installed.packages()) install.packages(i)}
sapply(libs, require, character.only = TRUE)

# Obtenemos las empresas
url = "https://finnhub.io/api/v1/stock/symbol?exchange=US&token=XXXXX"
resp = GET(url)
codigos = fromJSON(content(resp, type = "text", encoding = "UTF-8"))
codigos = codigos[codigos$description!= "",]
```

Estructura de la App

UI

- Creamos un selector que tenga como opciones los valores que hemos extraído.
- Creamos un selector numérico para personalizar el número de lags a tener en cuenta.
- Creamos un selector numérico para personalizar el número de días a predecir.
- Incluimos el output de la visualización del gráfico que vamos a mostrar.

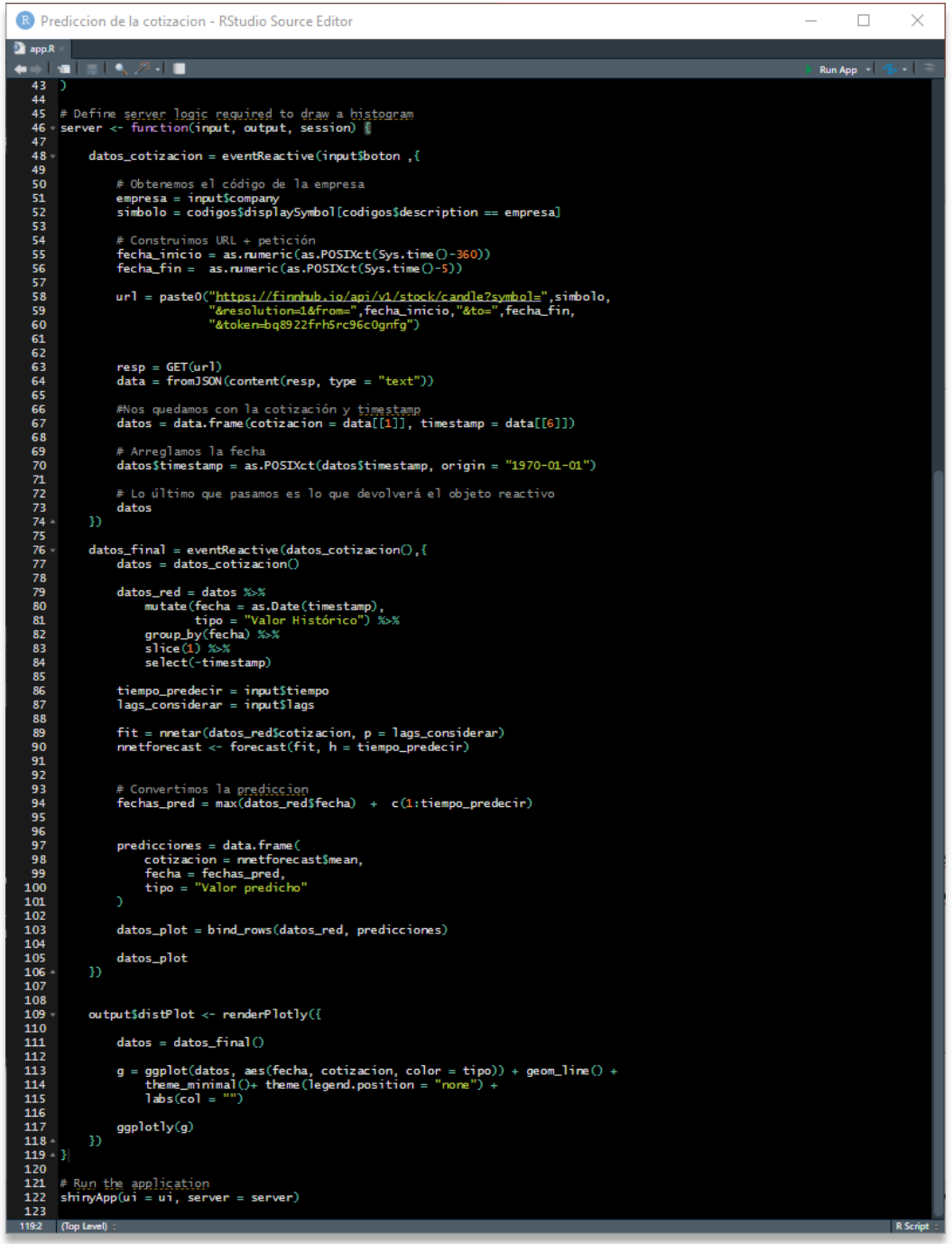
```
ui <- fluidPage(
titlePanel("Predicción de cotización"),
sidebarLayout(


# Creamos los selectores
sidebarPanel(
selectInput("company","Selecciona la empresa:",choices = codigos$description),
numericInput("lags","Nº de Lags a usar",min = 1, max = 30,value = 30),
numericInput("tiempo","Nº de Días a Predecir",value = 30, min = 1, max = 100),
actionButton("boton",label = "Analizar")
),
# Mostramos el gráfico
mainPanel(
           plotlyOutput("distPlot")
        )
    )
)
```

Estructura de la App

Server

- Extraemos los datos de la cotización para el valor elegido en el selector y lo convertimos en reactivo.
- Creamos un objeto reactivo que usa los datos de cotización, hace la predicción y devuelve los datos previos y los predichos.
- Creamos y renderizamos una visualización usando el objeto reactivo previo.

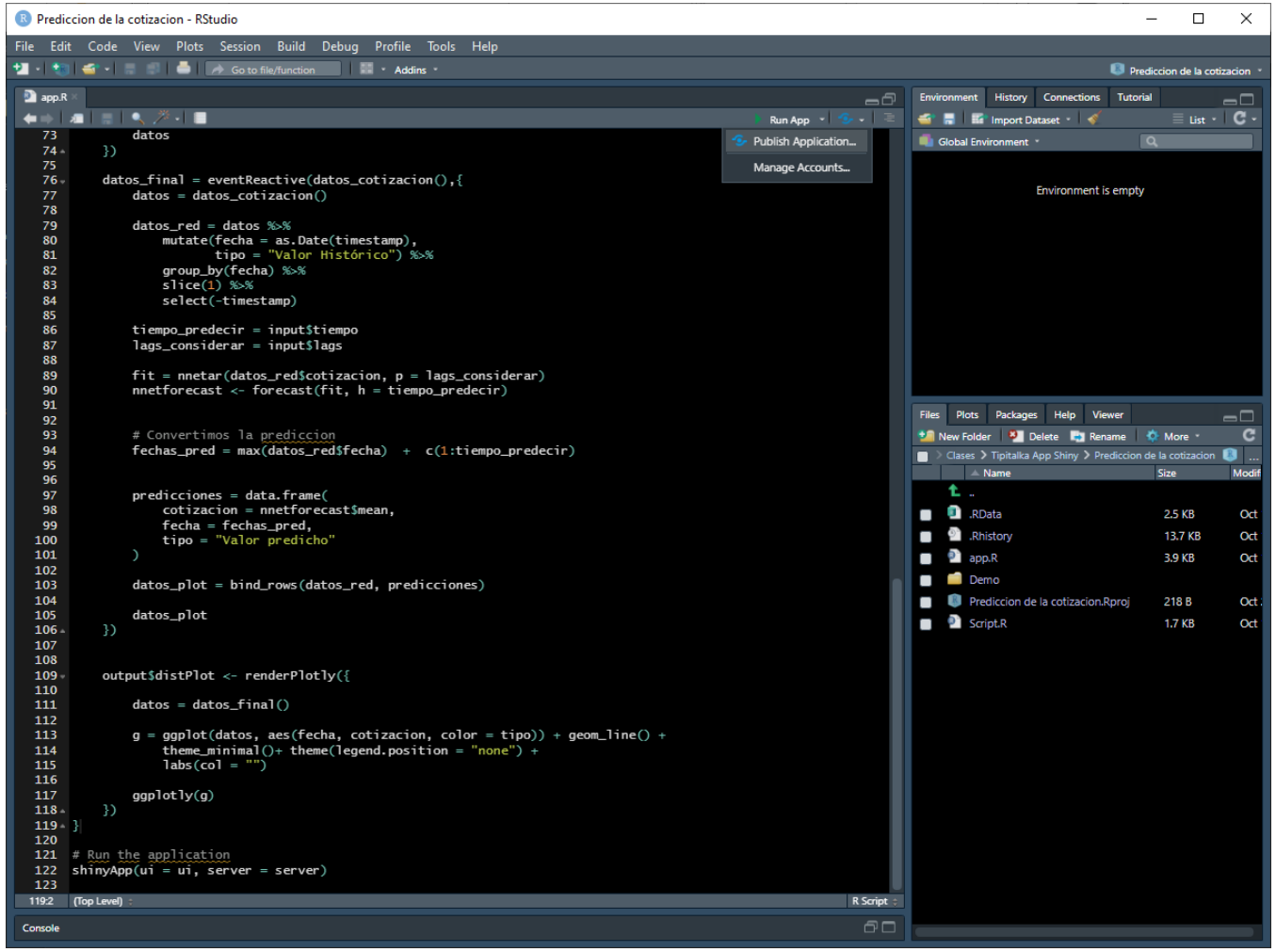


Cómo poner la App en Producción

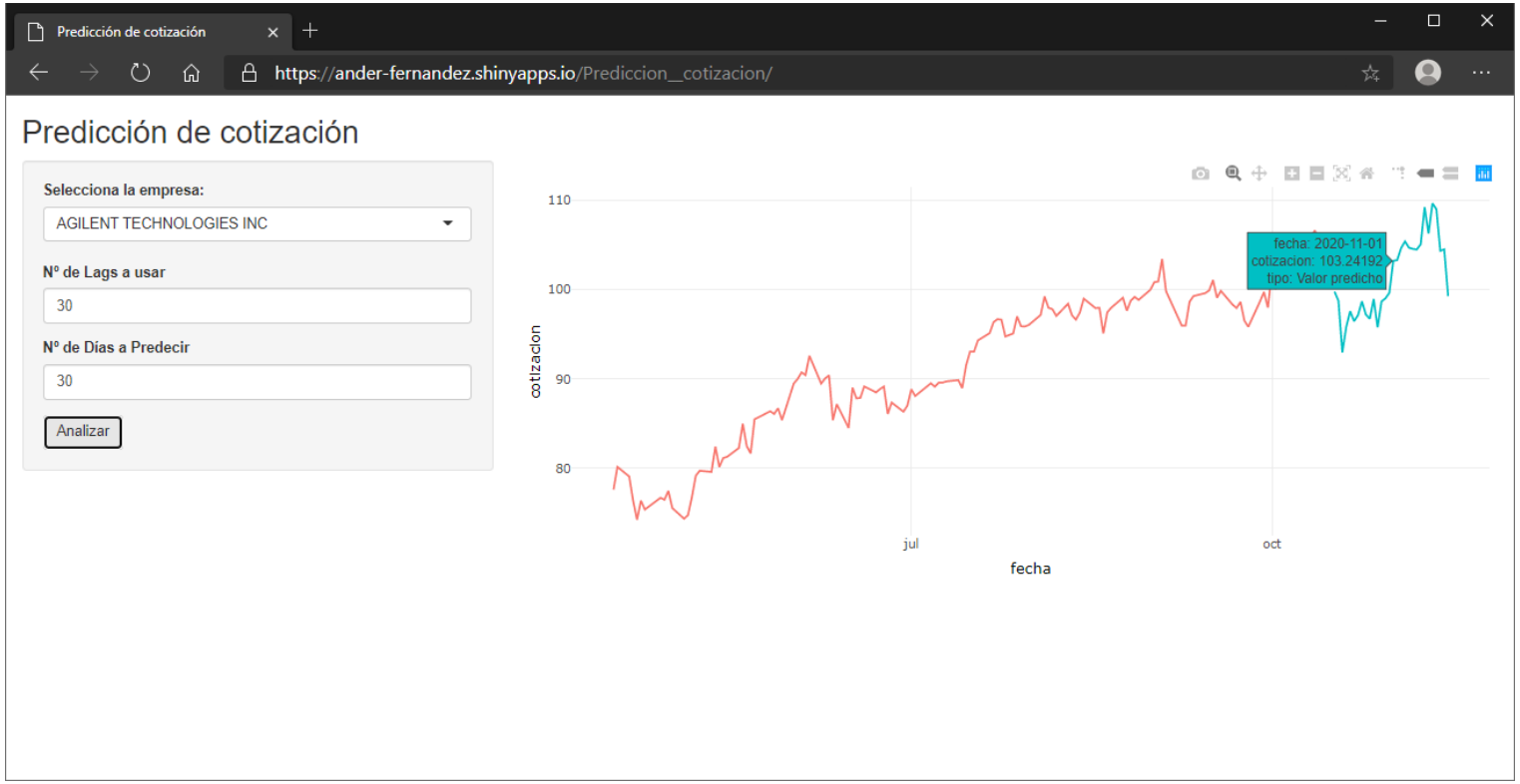
Shinyapps.io
Las aplicaciones Shiny creadas en Rstudio pueden subirse al servicio de hosting de aplicaciones de Rstudio shinyapps.io.

Este servicio ofrece una cuenta gratuita limitada en el número de aplicaciones, horas de uso, potencial del servidor y almacenamiento, entre otros, pero seguramente sea útil para vuestras aplicaciones.

Otra opción sería Dockerizar la aplicación para subirlo a una plataforma Cloud, ya sea montándolo en una máquina virtual o contenedores serverless.

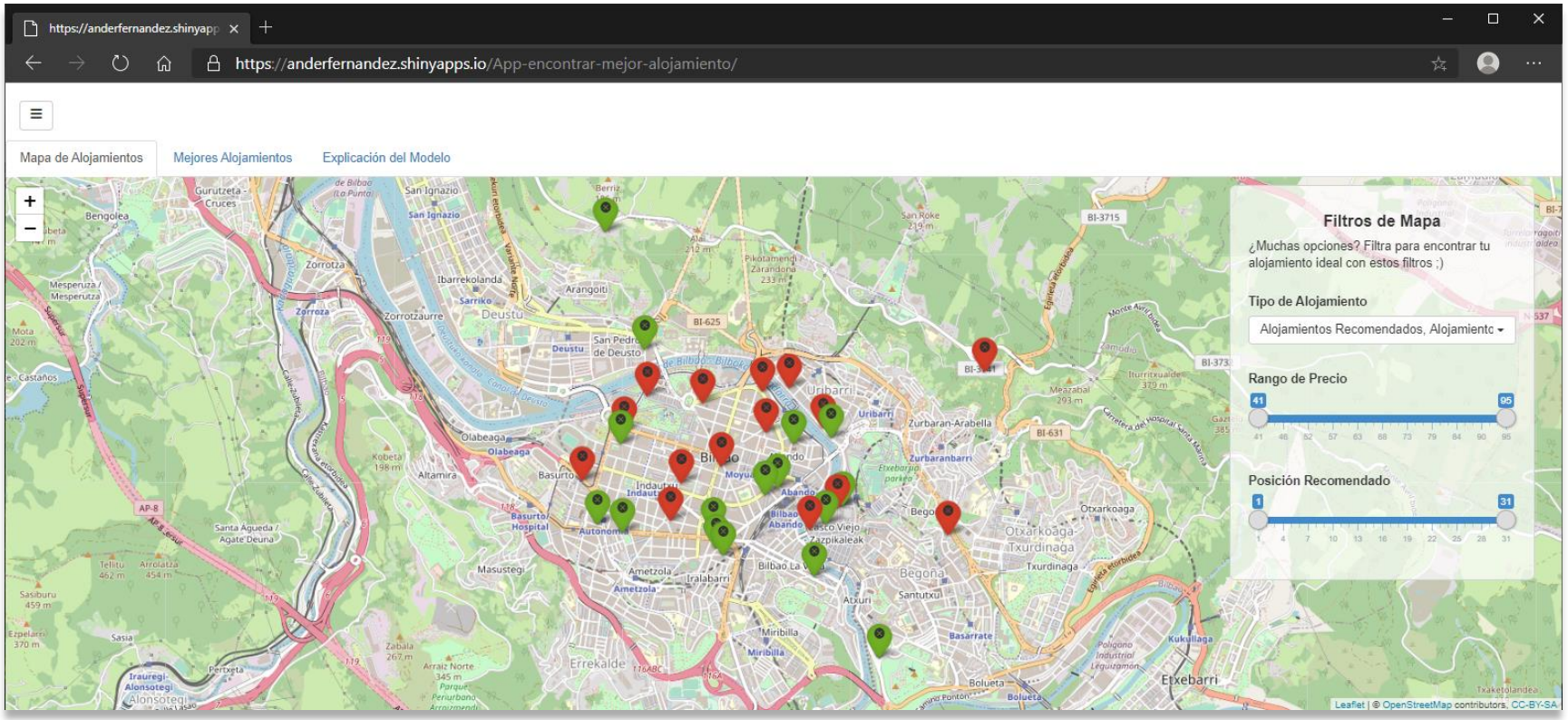


Al publicar la aplicación, automáticamente se nos abrirá una venta en la que podamos ver nuestra aplicación funcionar



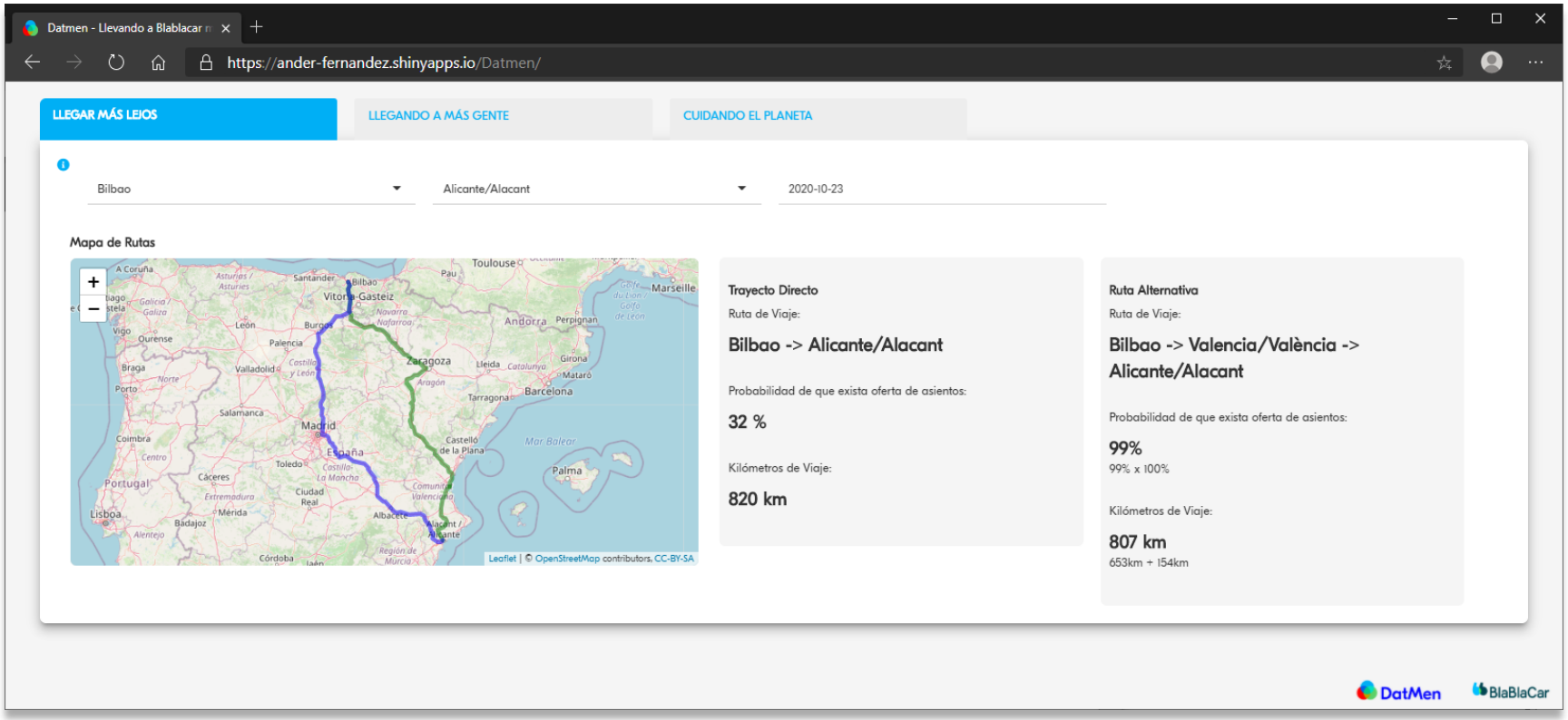
Índice

Extracción dinámica de alojamientos, limpieza de datos según los datos extraídos y entrenamiento *ad hoc* de un modelo para encontrar alojamientos con precios infraestimados



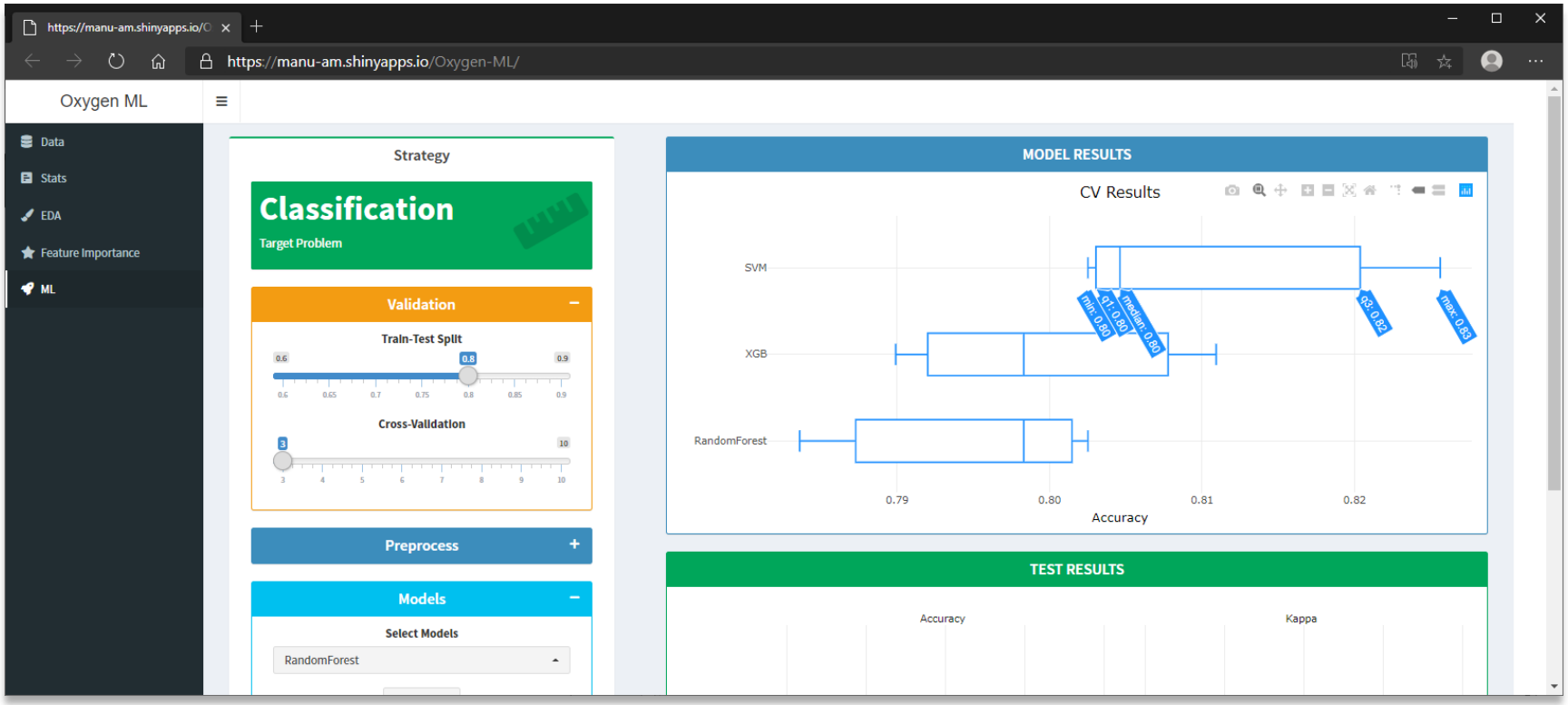
Autor: **Jon Osés** y **Ander Fernández**

Propuesta de nuevo servicio a Blablacar que usa los grafos para encontrar las rutas alternativas más cortas y probables, además de decirte la probabilidad de que se de tu viaje.



Autor: **Alexander Seoane** y **Ander Fernández**

Una herramienta de AutoML que realiza de forma automática y sin programar un análisis descriptivo de los datos, así como el entrenamiento de los modelos que elijas.



Autor: **Manuel Alda Martín-Mora**

¡Muchas gracias!

Web: **andernfernandez.com**
LinkedIn: **linkedin.com/in/ander-fernandez/**
Email: **afernandez@lin3s.com**