CONSTRUYENDO UNA APLICACIÓN INTERACTIVA DE VALUACIÓN DE OBLIGACIONES

Rodrigo Meneses

Contents

Abstract

Metodología para la construcción de una aplicación interactiva de valuaciones financieras, con orientación práctica a las obligaciones que enfrentan las empresas, aunque aplicable en un ámbito más general; contiene la construcción del motor de cálculo y la interfaz de usuario con el empleo del lenguaje R.

Palabras y frases clave: Aplicación interactiva, valuación, R, predicción, obligaciones, shiny.

Introducción

En la operación de las instituciones financieras y de seguros, uno de los elementos más vitales es la toma de decisiones correctas, una combinación entre la expertiz del responsable que toma la decisión y la información que le es suministrada para que dicha decisión sea la óptima.

Muchas decisiones dependen íntegramente de los flujos de efectivo bajo los cuales opera la institución, que pueden ser dados o esperados, entendiéndose como aquellos que ya se han sentado en los registros contables y aquellos que se espera que la empresa deba afrontar en algún momento futuro, respectivamente. A lo largo del tiempo se han desarrollado múltiples instrumentos para lidiar con flujos de efectivo que para un momento dado no se han efectuado, por lo tanto, se desconocen sus valores, y para los cuales no se tiene certeza realmente que se vayan a efectuar, es decir, flujos de efectivo que presentan incertidumbre en dos dimensiones; su valor y la certeza de su realización.

Como se mencionó, los conocimientos que permiten comprender dicha incertidumbre ya están a disposición de cualquier individuo que se interese en el tema, en la teoría ya existen, por lo tanto, el presente trabajo

no pretende generar nuevo conocimiento en el tema, pero sí pretende hacer uso del mismo. Dicho uso se realizará a través de una aplicación interactiva construida con el software estadístico **R** [1], en particular con su complemento llamado **Shiny**.

La aplicación que se construirá, tendrá como finalidad **valuar obligaciones**, sin embargo, el énfasis se hará sobre el desarrollo mismo de la herramienta, como una metodología útil, reproducible y aplicable en diferentes ámbitos que permite crear herramientas intuitivas y sencillas que cualquiera puede operar, pero que en su interior poseen una poderosa carga teórica orientada a la consecución de un determinado fin.

Motivación

El conocimiento de la metodología que se presenta, tiene un impacto benéfico en dos etapas, en las cuales el interesado en el texto puede encontrarse, se habla del **ámbito educativo** y del **ámbito de negocio**.

Ámbito educativo

En el ámbito educativo el individuo puede adquirir un conocimiento ampliamente reproducible, que es de aplicabilidad académica, expresándose como una herramienta práctica en temas de valuación, además puede ser introducido o reforzar conocimientos en la programación y el cómputo estadístico.

La creación de una aplicación interactiva, es un acercamiento (como ya se mencionó) a la programación, y en dicho sentido, permite al individuo conocer el entorno del **desarrollo y la implementación de herramientas computacionales**, lo cual incentiva su creatividad y le provee un enfoque probablemente nuevo para afrontar los retos en su área de interés.

Dicha creatividad está sustentada en una gran cantidad de herramientas que provee tanto el lenguaje de programación **R**, como el complemento **Shiny**, ambos alojados en un entorno de desarrollo integrado llamado **RStudio** [2], por lo tanto, con interés y dedicación el individuo podrá dotarse de cada vez más recursos para afrontar retos gradualmente más complejos y fortalecer sus capacidades de implementación y desarrollo.

Actualmente, debe reconocerse la importancia de que el interesado en la Actuaría tenga a su disposición la capacidad necesaria para llevar a *producción* sus conocimientos, es decir, que sea capaz de llevar a cabo un desarrollo (crear una herramienta, quizá desde cero) para satisfacer un determinado fin con base en lo aprendido en sus estudios, por lo tanto, la implementación y el desarrollo de herramientas computacionales no prescinde del rigor, la lógica y las matemáticas ya fundamentales en el estudio de la Actuaría, por el contrario, parte de ellas para que en combinación con la creatividad y las herramientas contenidas en el presente texto, el interesado en la Actuaría pueda mejorar sus capacidades con el enfoque de convertirse en un elemento propositivo y álgido a la hora de afrontar los retos en su área de interés.

Ámbito de negocio

En el ámbito de negocio, la metodología que se presenta es útil dentro de las instituciones del sector público y privado, así como para trabajadores por cuenta propia y *freelance*; principalmente porque el software que se utiliza y la mayoría y más importante parte de sus herramientas es de licencia libre, por lo tanto, el interesado no tendrá que pagar en lo absoluto por el uso de las mismas sin importar el tipo de uso que les dé. Sin embargo, esa no es la mejor parte en realidad, puesto que las herramientas brindan al usuario la posibilidad de **automatizar** sus procesos.

La automatización es uno de los puntos más importantes por los cuales la metodología que se presenta resulta relevante en el ámbito de negocio, dentro de muchos entornos de trabajo (como los ya mencionados) los individuos suelen tener la responsabilidad sobre procesos cuya ejecución es recurrente y requieren un esfuerzo del individuo, estos procesos suelen tener un diseño invariante por la rigidez de la estructura donde yacen y en cierto plazo pueden volverse completamente las funciones de un individuo en una organización.

El *estancamiento* de dichos individuos en procesos recurrentes puede hallar una solución en la automatización de dichas tareas, pues la herramienta y la metodología en sí, que se utilizarán, tienen una visión encaminada a realizar desarrollos que dependan en lo mínimo necesario de la intervención del desarrollador. Por ello, es clara la ventaja de la visión que provee la metodología que se presenta, puesto que no solo implica en general un ahorro de tiempo, también representa la libertad del individuo para facilitar y organizar su quehacer proveyéndolo de independencia para innovar y ser más eficiente, con lo que se fortalece su iniciativa, creatividad y en general su potencial individual y en su caso, el potencial dentro de una organización.

Aunque no todo es gratis, pero afortunadamente, el costo que implica conocer las herramientas y la metodología no es económico, tan solo implica la inserción del individuo en una curva de aprendizaje donde los recursos están disponibles a todo público. Un costo que sí es económico y se puede suscitar es al tomar la decisión de implementar algún desarrollo, pero el beneficio a largo plazo es mayor.

En resumen, tanto si se trata de un estudiante como de un profesionista, el conocimiento que se desea transmitir en el presente texto le proveerá de una poderosa y gratuita herramienta, así como enfoque diferente para solucionar los problemas de la cotidianidad.

El entorno R

La herramienta a partir de la cual se desarrollará la metodología presente en el texto parte de un lenguaje de programación llamado **R**, el cual es también un entorno de programación en sí, sin embargo, no se hará uso del mismo dado que existe un entorno de desarrollo integrado más amigable con el usuario que depende de **R** y el cuál es también de licencia libre llamado **RStudio**.

Las distribuciones de **R** se encuentran en *The Comprehensive R Archive Network* (CRAN), una red de servidores alojados alrededor del mundo que contienen todas las versiones de **R** hasta la más actual, así como las paqueterías y documentación de las mismas, se hablará de ello más adelante.

Instalar R

Instalar R es sencillo, únicamente se debe acceder a CRAN en https://cran.r-project.org/ y seleccionar el sistema operativo del dispositivo, posteriormente seguir los pasos que se mencionan en la sección.

Instalar RStudio

Instalar RStudio es igualmente sencillo, se debe acceder a https://www.rstudio.com/products/rstudio/download/ y descargar **RStudio Desktop Open Source License**, será desplazado a una sección conteniendo los instaladores para las diferentes plataformas, se debe seleccionar la apropiada (normalmente la primera opción) e instalarse.

Dado que se ha instalado lo anterior satisfactoriamente, el dispositivo que se está utilizando ya cuenta con el software necesario, se presentan a continuación algunos pasos iniciales para comenzar a operarlo.

R: primeros pasos

El paso fundamental que se recomienda es consultar el texto *Introducción a R* [3] que se encuentra en CRAN, el cual contiene lo básico de la herramienta a utilizar y ha sido escrito por los desarrolladores principales del lenguaje, por lo tanto, no existe mejor referencia. El presente texto no contiene una guía detallada del lenguaje, por lo que se hará referencia a las páginas de [3] cuando sea necesario.

Seguido de lo anterior, el usuario debe familiarizarse con la interfaz que presenta **RStudio**.

Al iniciar el software por primera vez, podremos observar una cuadrícula con cuatro secciones y una cabecera de opciones (véase Figura 1):

1) **Ventana superior izquierda**: Encontramos en esta sección el **script** en el cual el usuario escribe, estructura y guarda los desarrollos que realice, es el lienzo del usuario y dependiendo del tipo de archivo que se desee generar tendrá ligeras modificaciones en su vista, sin embargo, en esencia es lo mismo.

2) **Ventana superior derecha**: En esta sección encontramos tres elementos; **el entorno de trabajo (*Environment*), la historia del usuario (*History*) y las conexiones (*Connections*)**.

2.1) El entorno de trabajo contiene todos aquellos objetos creados por el usuario, además de aquellos que ha importado a través de las herramientas de importación que se encuentran en la misma pestaña (*Import Dataset*).

2.2) La historia del usuario constituye un rastro de las ejecuciones del código que ha realizado el usuario, es decir, en cuanto el usuario ha plasmado algo en el script, puede ejecutarlo y dicha acción será guardada en la historia, **RStudio** guarda dicha historia en archivo temporal y puede rescatarse en cualquier momento, lo cual permite recuperar previas ejecuciones.

2.3) Las conexiones de R es cualquier interacción que se establezca con software para manejo de bases de datos, **R** permite realizar dichas conexiones con un asistente para facilitarlo.

3) **Ventana inferior izquierda**: En esta sección encontramos la **consola (*Console*)**, la cual es la *interfaz de línea de comandos* donde el usuario le dará instrucciones al software en el lenguaje **R**, además, con la secuencia de instrucciones adecuada, también es la consola la que leerá el script y ejecutará lo contenido en él.

4) **Ventana inferior derecha**: En esta sección encontramos los **archivos y las pestañas de visualización**.

4.1) En la pestaña de archivos (*files*) el usuario puede visualizar las carpetas de trabajo y su contenido tal cual podría en un explorador de archivos, pero con una vista más simplificada.

4.2) En la pestaña de gráficos (*Plots*) el usuario visualiza gráficos que haya diseñado.

4.3) En la pestaña de paquetes (*Packages*) el usuario tiene toda la información de las paqueterías que tiene instaladas, además de tener un asistente de instalación y un asistente de actualización, por ejemplo, el usuario siempre tendrá instalada la paquetería `stats`.

4.4) En la pestaña de ayuda (*Help*) el usuario puede obtener ayuda desde CRAN para conocer información acerca de paqueterías o funciones escribiendo (?) seguido de la paquetería o función directamente en la consola y presionando entrar (*Enter*), por ejemplo; para la paquetería `stats` que contiene funciones para estadística, se puede consultar información a través del comando `?stats` o la función `mean` que calcula la mediana de una serie de valores, se puede consultar su información a través de `?mean`. **Observación**: Toda la información se encuentra en inglés.

5) **Barra superior**: En la barra superior se encuentra la **cabecera de opciones** donde el usuario tiene pestañas que le permiten administrar los documentos, la vista, la sesión y las opciones de uso de Rstudio.

El usuario puede conocer más información en [3] (Páginas 2-6).

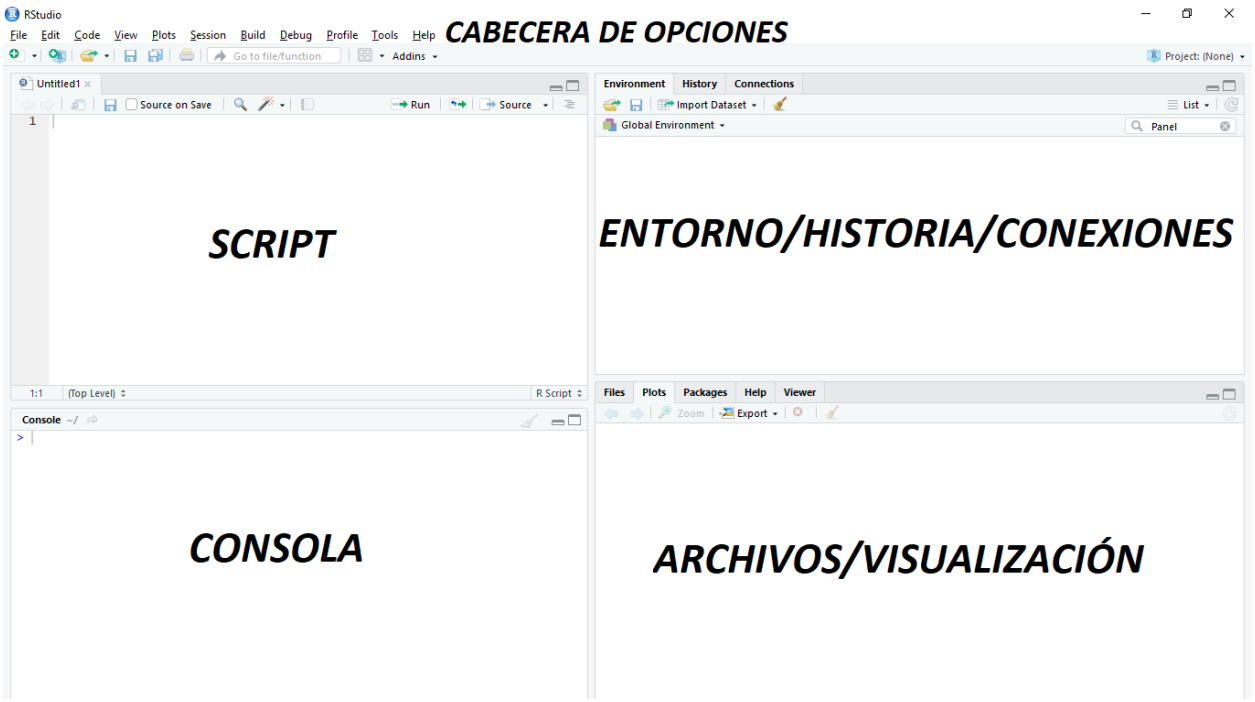

Figura 1: Interfaz de R

Construyendo la aplicación

Una metodología de pasos

La construcción de la aplicación conlleva una sucesión de pasos que el usuario debe conocer y aplicar en tanto que desee obtener los mayores beneficios de la herramienta que desarrollará, es fundamental que el usuario se encuentre afrontando un problema; entiéndase *problema* como una situación sujeta a mejoría o que requiera una solución. Por lo tanto, el usuario debe:

1) **Conocer el problema**: El usuario debe ser capaz de describir el problema y darle una solución en términos rigurosos. La herramienta no provee soluciones en sí misma, pero permite al usuario potenciar sus soluciones cuando ya las ha identificado y puede sustentarlas, no necesariamente en una teoría específica, pero mínimamente con una base rigurosa que provenga de la lógica formal. En dicho sentido, la herramienta es inútil si el usuario no tiene identificada una solución.

2) **Parámetros del problema**: El usuario debe ser capaz de extraer todas aquellas variables que determinen completamente el problema, es decir, la información necesaria y suficiente para que el problema tenga una solución en los términos que ya se han planteado en el punto anterior. Por ejemplo, si el problema es **sumar dos números reales**, los parámetros serán dos valores numéricos reales que pueden variar (los cuales van a sumarse), si además se desea variar la operación binaria, podría tenerse un tercer parámetro definido como *la operación binaria* (suma, resta, multiplicación, etcétera).

2.1) El usuario debe definir el **dominio** de los parámetros, haciendo referencia al ejemplo anterior de la suma, los parámetros a sumar podrían variar en los números reales, o en los enteros o en algún subconjunto de los anteriores, mientras que para el parámetro de la operación binaria se podría definir una lista de valores {suma, resta, multiplicación, división} como el dominio del parámetro. En realidad, el dominio del parámetro

muchas veces lo sugiere la naturaleza misma de los parámetros, aunque el usuario puede definirlos con base en sus necesidades.

2.2) *Definición.* Cuando para un problema determinado se han definido los parámetros del mismo, se dice que el problema es un **problema parametrizado**.

3) **Motor de cálculo**: El usuario debe ser capaz de extraer del punto 1) la estructura lógica-matemática que con el uso de los parámetros genera la solución al problema relativo y posteriormente el usuario debe ser capaz de resumir en reglas directas y ordenadas el flujo natural del problema parametrizado hasta la generación de la solución, lo cual en resumen es un *algoritmo*, por lo tanto, el usuario parte de una estructura lógica-matemática que da solución al problema parametrizado en cuestión y la disgrega en unidades que puedan ser empleadas en la construcción de un algoritmo.

3.1) En el vocabulario de RStudio el motor de cálculo recibirá el nombre de **Servidor (*Server*)**.

4) **Interfaz de usuario**: El usuario debe determinar cómo se interactúa con el problema parametrizado y cómo se muestra la solución, debe de comprender el funcionamiento del motor de cálculo y cómo el usuario interactuará con el para variar los parámetros, además de definirse el estilo y presentación de la solución para mejorar la visualización y su manejo. Por ejemplo, continuando con la suma, el usuario debería tener la opción de definir los parámetros a sumar, y sería adecuado que se pueda visualizar la operación que se está realizando, además de mostrar el resultado en un tamaño de fuente mayor para que sea más sencilla la lectura.

4.1) La interfaz del usuario debe estar orientada a la optimización de la *experiencia de usuario* véase [4] (páginas 9-34) , por lo tanto, el usuario debe poder interactuar con el problema de la manera más sencilla y amigable que sea posible cuidando la calidad de la solución y su presentación.

4.2) La interfaz de usuario está compuesta de tres elementos:

- Los datos de entrada (*input*): valores asignados a los parámetros del problema parametrizado provistos por el usuario.
- Los datos de salida (*output*): valores que devuelve el motor de cálculo tras procesar el problema parametrizado con los valores de los parámetros que ha asignado el usuario.
- La vista: conjunto de elementos gráficos.

4.3) En el vocabulario de RStudio la interfaz de usuario se expresa con la abreviatura en inglés **UI (*User Interface*)**.

4.4) *Definición.* La terna {problema parametrizado, motor de cálculo, interfaz de usuario} recibe el nombre de **problema de programación**.

5) **Entorno de desarrollo**: El usuario debe desarrollar el problema de programación en el entorno de programación que seleccione, en nuestro caso haremos uso de R (que a su vez es el lenguaje de programación) y del entorno de desarrollo integrado RStudio, aunque este paso no limita al usuario a emplear el software que decida se advierte que en lo subsecuente se empleará el software mencionado.

5.1) *Definición.* El desarrollo del problema de programación dentro del entorno de desarrollo recibe el nombre de **programa** y se debe entender como la escritura en código (relativo al lenguaje de programación empleado) del motor de cálculo y la interfaz de usuario.

6) **Desarrollo interactivo**: El usuario debe desarrollar la estructura interactiva del programa, con la finalidad de generar el producto final que permita al usuario manipular el problema de programación variando los parámetros a través de un dispositivo.

6.1) El desarrollo interactivo en el presente texto se hará con la paquetería `Shiny` de R; cada software podrá o no tener un soporte de desarrollo interactivo.

6.2) *Definición.* La estructura interactiva desarrollada para un programa en específico recibe el nombre de **aplicación interactiva**.

6.3) La construcción de la aplicación interactiva constituye un trabajo dual donde el usuario deberá ser capaz de integrar el motor de cálculo y la interfaz de usuario en una estructura interactiva.

7) **Depuración**: El usuario debe depurar la aplicación interactiva para prevenir que produzca resultados desfavorables.

7.1) Existe una gran variedad de errores que pueden presentarse durante el desarrollo de la aplicación interactiva, en el presente texto se comentará muy poco sobre depuración debido a lo extenso que sería informar bien al lector sobre el tema, sin embargo, se puede buscar ayuda para resolver muchos errores en foros como **Stackoverflow**.

Un ejemplo sencillo

En esta sección se desarrollará como problema a *las operaciones binarias aritméticas* aplicando paso a paso la metodología expuesta y comentando, para que el lector se familiarice con la misma, más adelante se abordará un caso práctico más complejo.

1) Para fines del ejemplo se hará uso de definiciones sencillas, por lo tanto, se definen a las operaciones binarias aritméticas como sigue:

Sean $a, b, c \epsilon \mathbb{R}$, donde $\mathbb{R}$ es el conjunto de los números reales [5] (páginas 37-84) y X el producto cartesiano [5] (página 20), se definen las operaciones:

- +: $\mathbb{R}$ X $\mathbb{R} \rightarrow \mathbb{R}$ tal que a + b = c llamada *suma.*
- *: $\mathbb{R}$ X $\mathbb{R} \rightarrow \mathbb{R}$ tal que a * b = c llamada *producto.*

Las propiedades de dichas operaciones binarias se pueden encontrar en [5] (SECCIÓN 2.1 Las propiedades algebraicas de R, páginas 38-42), por simplicidad se asumirá que el lector las conoce.

El problema esencial radica en poder facilitar el uso de las operaciones binarias aritméticas a través de una aplicación interactiva (similar a una calculadora) donde se represente la operación simbólicamente junto con el resultado y se muestre un gráfico que exprese la solución de forma geométrica.

2) El lector debería ser capaz de identificar fácilmente los parámetros del problema, en este caso, el problema queda completamente determinado cuando se conocen los valores **a**, **b** y la operación binaria **+** o ***** . Además, los rangos de los parámetros son bastante naturales, pues $a, b \epsilon \mathbb{R}$ y la operación binaria solo tiene dos opciones posibles como valor: {+,*}.

Por lo tanto, el problema ya tiene una solución en términos de las propiedades mismas de las operaciones y toma como parámetros {a, b, operación binaria}.

3) El servidor puede ser fácilmente obtenido:

El usuario ingresa los parámetros:

a, b, operación binaria

Si se cumple que:

operación binaria = +

Entonces se ejecuta:

a + b

En caso que no se cumpla, se ejecuta:

a * b

Luego el servidor calcula el valor de la operación y devuelve (dependiendo del caso):

a + b = c o **a * b = c**

Luego el servidor grafica los puntos a, b, c en una región tridimensional y los une con líneas rectas

Finalmente, el servidor termina la ejecución y regresa al inicio

---

La estructura anterior es similar a un *pseudocódigo* aunque con un estilo libre. El lector puede seguir una metodología específica para escribir un pseudocódigo si así lo decide, para ello puede consultar [6], sin embargo, la idea es que el lector con base en los conocimientos que ya ha adquirido en sus estudios previos sea capaz de escribir sentencias tan sencillas como las mostradas para desagregar problemas complejos.

Puede que no siempre sea necesario escribir las estructuras dada la sencillez misma del problema y que el lector pueda retenerlas en la memoria, o que sea posible reducirlas, resumirlas, emplear una menor especificidad o usar abreviaturas, todo depende del lector y es libre de plasmar sus ideas como desee con el único objetivo de tenerlas en orden y facilitar el trabajo de crear la aplicación interactiva.

4) Con base en la estructura del servidor, podemos idear la interfaz como sigue:

- Inputs:

1) [Parámetro 1] - Rango: $\mathbb{R}$

2) [Parámetro 2] - Rango: $\mathbb{R}$

3) [Parámetro 3] - Rango: +, *

- Outputs:

1) [Parámetro 1] [Parámetro 3] [Parámetro 2] = [Resultado]

2) <Gráfico tridimensional>

- Vista: se realizará un panel con dos secciones divididas verticalmente.

1) **Sección izquierda**

Ingresar valor de los parámetros:

[Parámetro 1]

[Parámetro 2]

[Parámetro 3]

2) **Sección derecha**

Se muestra el gráfico:

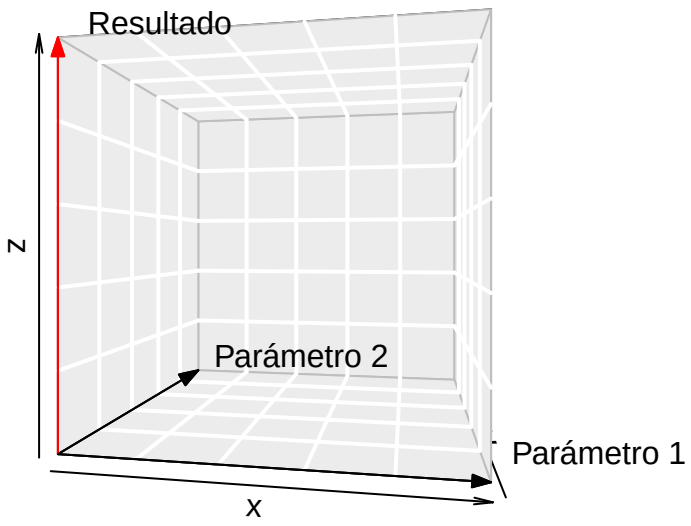

Figura 2: Vista del gráfico tridimensional

Observación: la finalidad del gráfico es solo para fines del ejemplo, se comprende que el lector no encuentre utilidad en el mismo, además el lector puede tan solo realizar un boceto en papel, no es necesario realizar algo tan complejo como lo que se muestra. Finalmente, la idea es que el lector plasme de forma ordenada sus ideas.

5) Como se mencionó, se usará el lenguaje de programación R dentro del entorno de desarrollo integrado RStudio, para ello es importante que el usuario conozca el lenguaje, dado que el presente texto no contiene una guía. Se recomienda al lector no familiarizado con el lenguaje que consulte [3] (páginas 7-54).

En primera instancia, se sentencian los parámetros a través de valores iniciales arbitrarios, esto nos permite trabajar con el Servidor y la UI sin integrar aun un entorno interactivo, probando primero la funcionalidad en un caso fijo, por lo tanto, se debe abrir un script nuevo y desarrollar el problema de programación como sigue.

```
# Sentencia de valores arbitrarios para los parámetros

parametro1<-round(runif(1,1,2),2)
parametro2<-round(runif(1,1,2),2)
parametro3<-c("Suma","Producto")
```

Luego, desarrollamos el servidor, usaremos la paquetería `plot3D` para realizar el gráfico mientras que lo demás se encuentra en la base de R. La cabecera que se muestra sirve para instalar las paqueterías necesarias si no se encuentran instaladas, además de activar las necesarias con la función `require`.

```
# Cabecera de paqueterías

list.of.packages <- c("plot3D")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()
                                   [,"Package"])]
if(length(new.packages)) install.packages(new.packages)

require(plot3D)

# Generamos una función

servidor<-function(parametro1,parametro2,parametro3){
  if(parametro3=="Suma"){
    resultado<-sum(parametro1,parametro2)
    x0 <- c(0, 0, 0)
    y0 <- c(0, 0, 0)
    z0 <- c(0, 0, 0)
    x1 <- c(parametro1,0,0)
    y1 <- c(0,parametro2,0)
    z1 <- c(0,0,resultado)

grafico<-arrows3D(x0, y0, z0, x1, y1, z1,col = c("black","black","red"),
                  bty = "g",phi=0,theta = 10) +text3D(x1+0.05, y1+0.05,
z1+0.05, c(parametro1,parametro2,resultado), colvar = NULL, add=TRUE)

return(list(resultado,grafico))

  } else{
    resultado<-prod(parametro1,parametro2)
    x0 <- c(0, 0, 0)
    y0 <- c(0, 0, 0)
    z0 <- c(0, 0, 0)
    x1 <- c(parametro1,0,0)
    y1 <- c(0,parametro2,0)
    z1 <- c(0,0,resultado)

grafico<-arrows3D(x0, y0, z0, x1, y1, z1,col = c("black","black","red"),
                  bty = "g",phi=0,theta = 10)+text3D(x1+0.05, y1+0.05,
z1+0.05, c(parametro1,parametro2,resultado), colvar = NULL, add=TRUE)

return(list(resultado,grafico))
  }
}
```

Luego, realizamos una ejecución de muestra para que el lector observe el resultado de lo desarrollado.

```
resultado<-servidor(parametro1 = parametro1,
                    parametro2 = parametro2,parametro3 = parametro3[1])
```

```
# Obtenemos el resultado de la operación

resultado[[1]]
```

```
## [1] 3.35
```

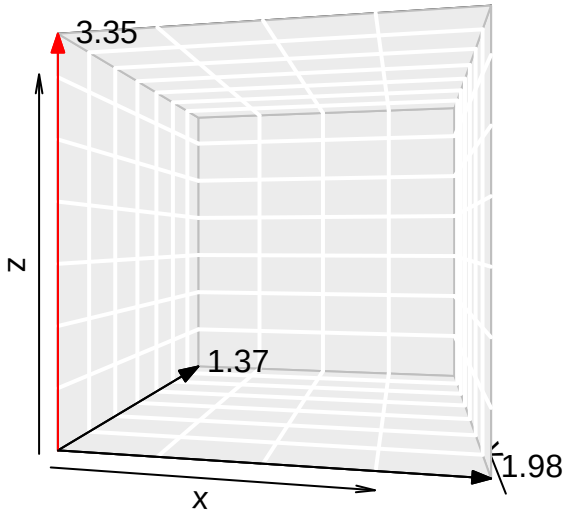

Figura 3: Ejemplo de ejecución

6) Para realizar esta sección, se debe abrir un script interactivo, como se muestra a continuación.

a) Seleccione File > New File > Shiny Web App...

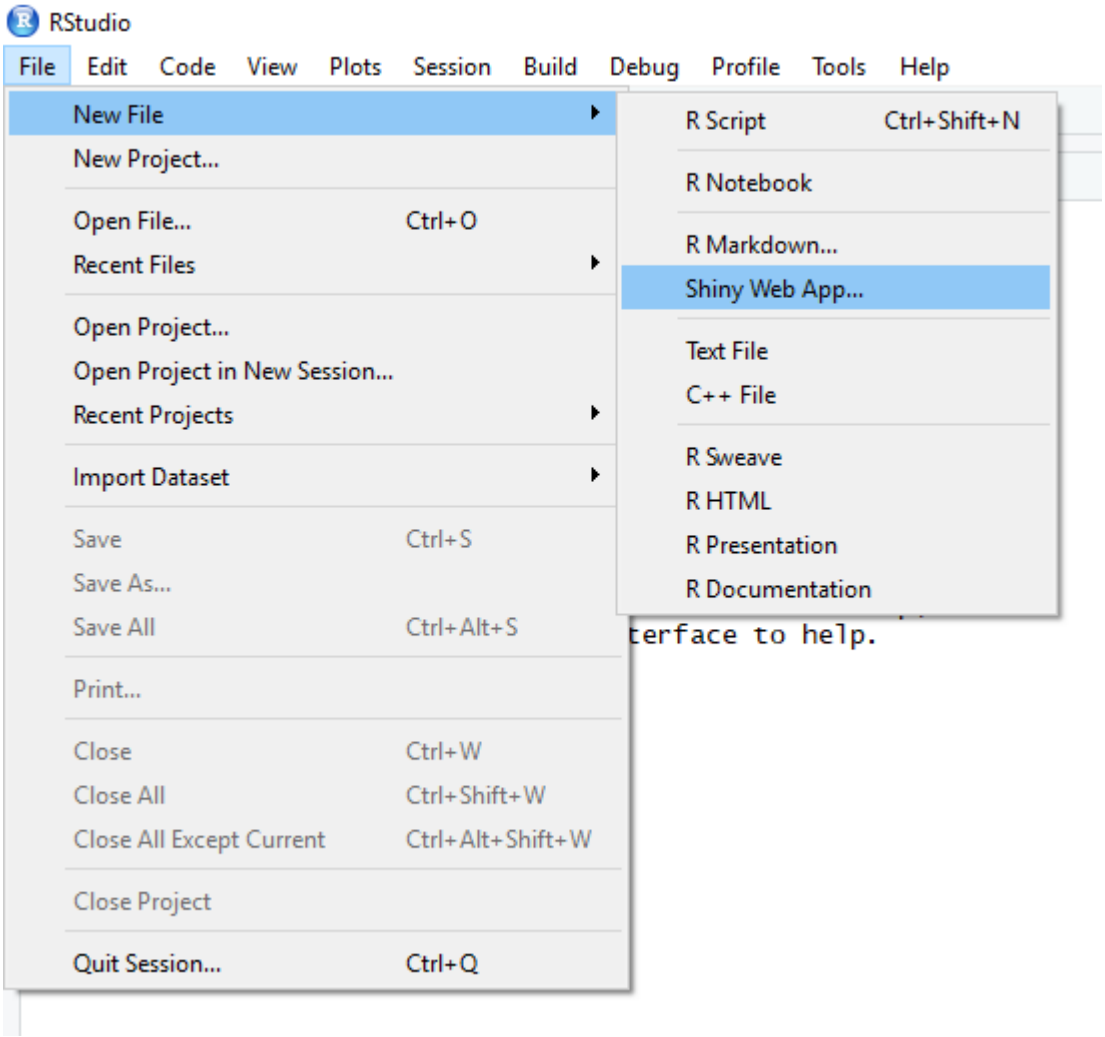

Figura 4: Iniciando el script interactivo a)

b) Podrá observar la siguiente ventana. El tipo de aplicación seleccionado (y el cual es el formato con el que se trabajará) es de **archivo único (*Single File*)**, lo cual significa que en un solo script estará contenido todo el programa.

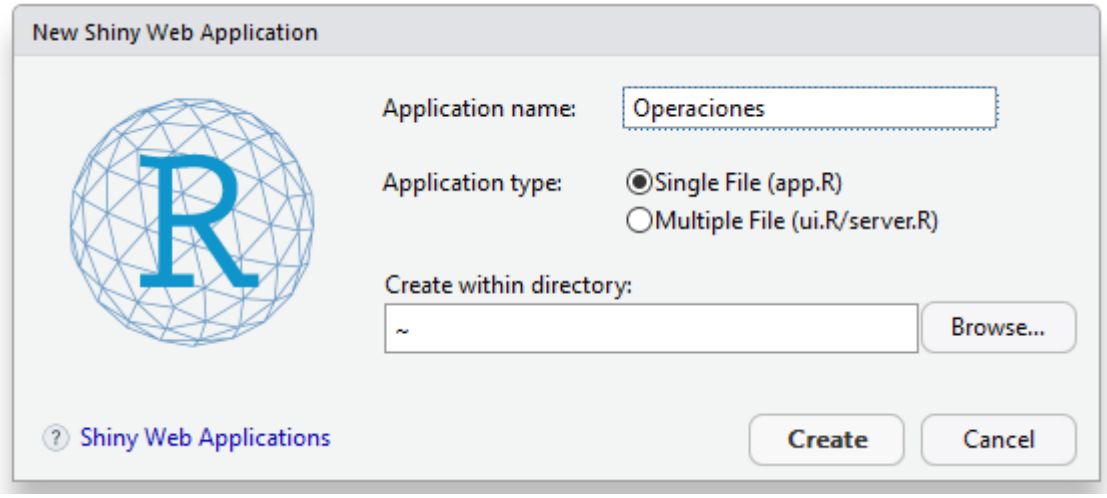

Figura 5: Iniciando el script interactivo b)

c) Como puede observarse, se genera un único script el cual siempre tendrá como nombre **app.R**, en el que puede observarse una aplicación interactiva escrita, la cual es predeterminada.

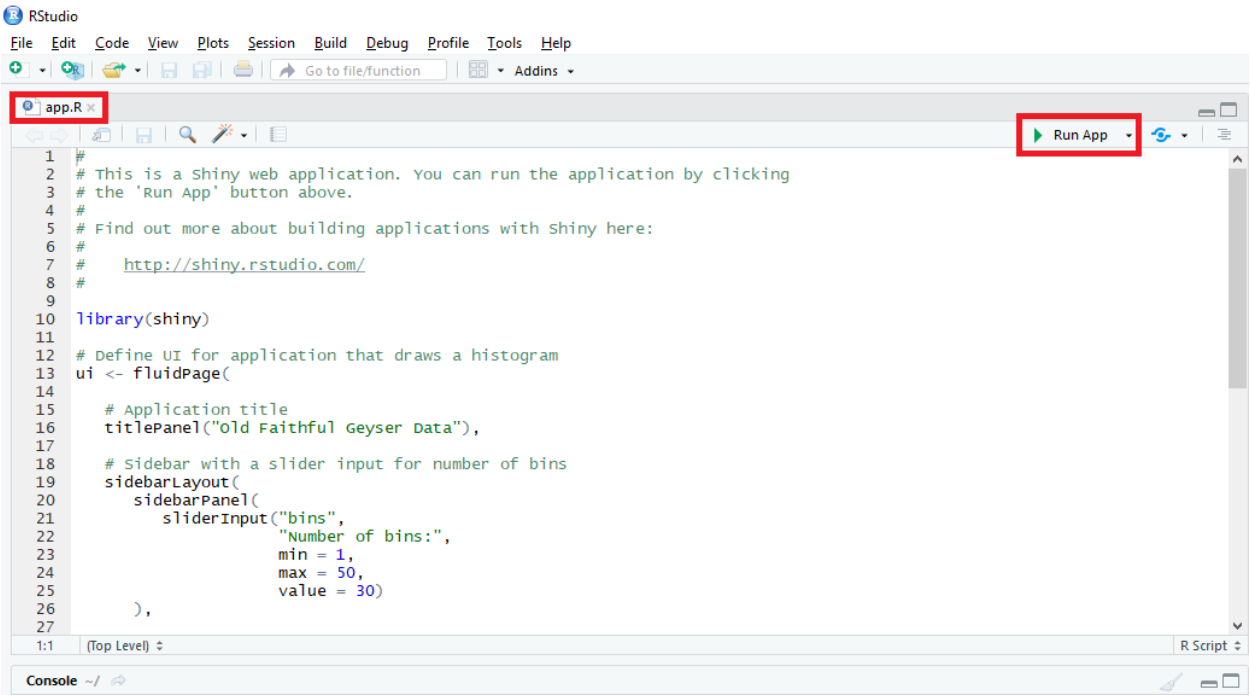

Figura 6: Iniciando el script interactivo c)

d) Puede ejecutarse dicha aplicación interactiva predeterminada con el botón **Run App**, se selecciona la opción **Run in Window**, aunque también se puede seleccionar la opción **Run External** (se invita al lector a hacerlo) y abrir la aplicación en el explorador web predeterminado (Internet Explorer, Chrome, Safari, Opera, entre otros).

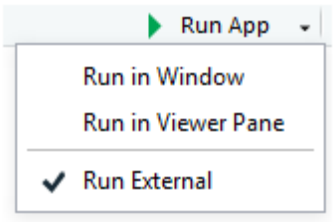

Figura 7: Iniciando el script interactivo d)

e) Finalmente, se abre la aplicación interactiva y el usuario puede interactuar con ella.

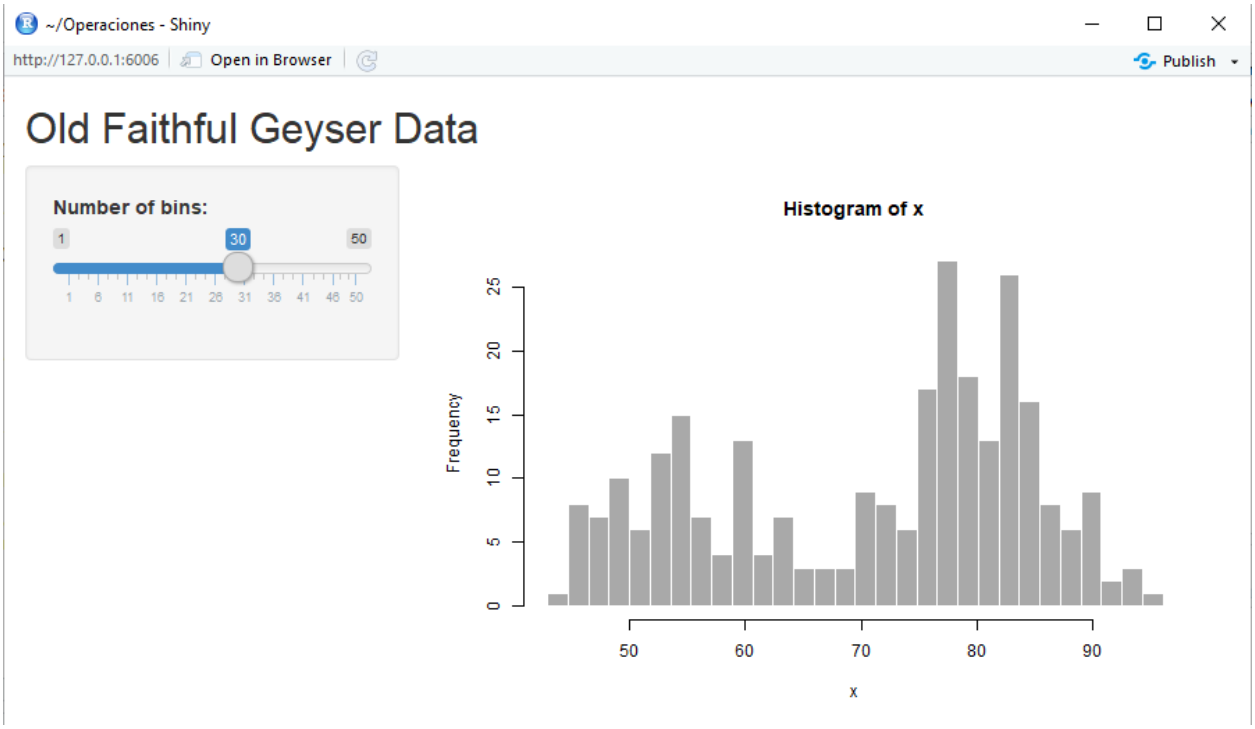

Figura 8: Iniciando el script interactivo e)

En virtud de lo anterior, desarrollamos la aplicación interactiva del programa escrito en los pasos anteriores, se recomienda visitar el sitio https://shiny.rstudio.com/tutorial/ para obtener mayor información sobre *Shiny*.

Lo que escribiremos a continuación debe ser sobre escrito dentro del archivo de *Shiny Web App* que se ha abierto, por lo tanto, el primer paso será dejar limpio el script borrando el código de la aplicación interactiva predeterminada.

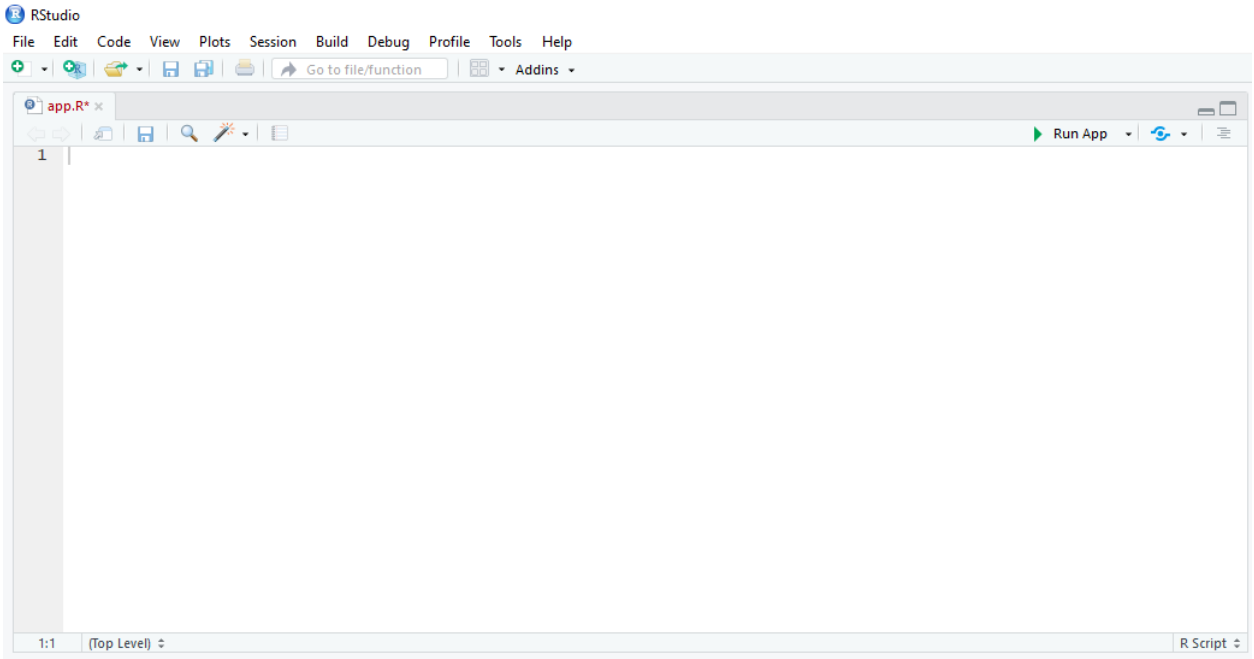

Figura 9: Script interactivo en blanco

Listo lo anterior, se procede a explicar los tres pilares fundamentales de *Shiny*, previo a la escritura del código.

Dualidad

Shiny trabaja de forma dual, integra el servidor y la UI en una sola herramienta (la aplicación interactiva), por lo que durante la creación de la aplicación interactiva se debe tener en cuenta dicha dependencia y explotarla para lograr los mejores resultados. En otro sentido, esta relación implica que algunas veces la modificación de ciertos elementos del servidor puede afectar al funcionamiento de la UI y viceversa. Por lo tanto, se debe tener cuidado y anticipar afectaciones en alguna de las partes cuando se modifica la otra.

Reactividad

Shiny trabaja de forma reactiva, lo cual es en parte consecuencia de la dualidad, pues un cambio dentro de una de las partes (Server, UI) afecta a la otra, sin embargo, la reactividad va más allá y nos dice que no solo se produce un cambio, si no que este es inmediato siempre que se empleen los objetos reactivos.

Dichos objetos reactivos se pueden agrupar en:

1) *Valores reactivos*: variables donde se almacenan datos sujetos a cambiar en cuanto el usuario de la aplicación interactiva cambie los inputs, es decir, ingrese valores distintos para los parámetros.

2) *Observadores*: los observadores son *piezas claves* en el funcionamiento de una aplicación interactiva, son funciones que detectan cuando un valor reactivo ha cambiado de valor (detectan cuando un parámetro toma otro valor) y activan una acción en concreto, por lo cual también reciben el nombre de *detonadores*. Serán de gran utilidad para activar al servidor cuando sea necesario realizar un cálculo.

3) *Output reactivo*: cuando un observador detecta el cambio de valor de un valor reactivo, se detona la ejecución de un conjunto de líneas de código que pueden detonar otro proceso o bien *producir un resultado que se muestra en la UI* el cual recibe el nombre de output reactivo (pues es el producto de una cadena de sucesos).

Cada uno de los objetos reactivos será utilizado a través una función específica o familia de funciones en **R** que serán conocidas a lo largo del ejemplo.

Lenguaje de marcado

Shiny trabaja con **HTML**, un lenguaje de marcado que será de utilidad para construir la vista de la UI. El uso que se hace de dicho lenguaje en el presente texto es mínimo y de rápido aprendizaje, el lector puede consultar más información sobre el lenguaje HTML en [7].

Con dicha información, será central que el lector identifique los tres elementos en el desarrollo, la jerarquía de construcción de la aplicación interactiva es la misma que la relativa a la metodología.

En este punto se asume que el usuario ha abierto satisfactoriamente el script interactivo con los pasos mencionados, se comienza con la escritura en código de la estructura interactiva del programa.

Se hará uso de la paquetería `shinydashboard` para dar funcionalidades extras a Shiny y una vista predeterminada como dashboard, la cual se compone de un `header (cabecera)`, una `sidebar (barra lateral)` y un 'body (cuerpo), una estructura sencilla, se pide al lector preste atención a las funciones que se utilizan y a los comentarios que están escritos con el símbolo #.

Observación: Es preferible no utilizar acentos ni símbolos especiales en el script interactivo de Shiny, puesto que la codificación del mismo es **UTF-8** y no trabaja correctamente con dicha escritura, aunque se puede modificar la codificación es recomendable no hacerlo.

```
# CABECERA DE PAQUETERÍAS

list.of.packages<-c("plot3D","shinydashboard","shiny")
new.packages<-list.of.packages[!(list.of.packages %in% installed.packages()
                                 [,"Package"])]
if(length(new.packages)) install.packages(new.packages)

require(plot3D)
require(shinydashboard)
require(shiny)

header<-dashboardHeader(title = "Operaciones binarias") # Se crea el cabecero y se asigna
                                                        # el nombre de la aplicacion

sidebar<-dashboardSidebar(disable = TRUE) # Se desactiva la barra lateral dado que no
                                          # será de utilidad
# Se genera el cuerpo
body<-dashboardBody(
  fluidRow(                                 # fluidRow genera un espacio donde "acomodar"
                                            # objetos visuales
    column(6,                               # Column define el orden en el que se acomodan
                                            # los objetos "en columna"
           numericInput("par1","Parametro 1"),
           numericInput("par2","Parametro 2"), # numericInput permite al usuario escribir
                                               # numeros y restringe el campo a numeros
           selectInput("par3","Parametro 3",
      choices = c("Suma","Producto")) # Select input permite al usuario escoger el valor
                                      # de un parametro en una lista de valores
           )))

# dashboardPage integra los elementos anteriores y genera una UI

ui<-dashboardPage(header = header,sidebar = sidebar,body = body)

# Por el momento el servidor se encuentra vacío, pero se sentencia
```

```
# para que sea posible observar la aplicacion

server<-function(input,output){}

# La funcion shinyApp integra la UI y el Servidor, y despliega la aplicacion web

shinyApp(ui,server)
```

Luego, el usuario debe hacer clic sobre el botón **Run App** para observar el resultado.

Se prosigue con la colocación del servidor, el cual se encuentra en la página 11. Se realizan algunos ajustes que se comentan a continuación.

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

observeEvent(c(input$parametro1, # Se añade un observador para detectar
input$parametro2,input$parametro3), # la entrada de inputs y activar el servidor
     {

output$operacion<-renderText({ # El prefijo "render" expresa que un output reactivo sera
if(input$parametro3=="Suma"){  # producido y el sufijo significa el tipo de resultado

 paste(input$parametro1,"+",input$parametro2)

} else {
 paste(input$parametro1,"*",input$parametro2)
 }
})

       output$grafico<-renderPlot({ # renderPlot genera como output reactivo un grafico

# Note las diferencias con el servidor escrito en la pagina 11, se ha reemplazado
# "parametro1" por "input$parametro1" para hacer referencia a que se trata de un
# input y en otro sentido, es un parámetro del problema.

        if(input$parametro3=="Suma"){
resultado<-sum(input$parametro1,input$parametro2)
x0 <- c(0, 0, 0)
y0 <- c(0, 0, 0)
z0 <- c(0, 0, 0)
x1 <- c(input$parametro1,0,0)
y1 <- c(0,input$parametro2,0)
z1 <- c(0,0,resultado)
grafico<-arrows3D(x0, y0, z0, x1, y1, z1,col = c("black","black","red"),
bty = "g",phi=0,theta = 10) +text3D(x1+0.05, y1+0.05,
z1+0.05, c(input$parametro1,input$parametro2,resultado), colvar = NULL, add=TRUE)
} else{
resultado<-prod(input$parametro1,input$parametro2)
x0 <- c(0, 0, 0)
y0 <- c(0, 0, 0)
z0 <- c(0, 0, 0)
x1 <- c(input$parametro1,0,0)
```

```
y1 <- c(0,input$parametro2,0)
z1 <- c(0,0,resultado)
grafico<-arrows3D(x0, y0, z0, x1, y1, z1,col = c("black","black","red"),
bty = "g",phi=0,theta = 10)+text3D(x1+0.05, y1+0.05,
z1+0.05, c(input$parametro1,input$parametro2,resultado), colvar = NULL, add=TRUE)
}
             })

output$resultado<-renderText(if(input$parametro3=="Suma"){
  resultado<-sum(input$parametro1,input$parametro2)
  paste(resultado)
} else{
  resultado<-prod(input$parametro1,input$parametro2)
  paste(resultado)
  }
  )
           })

}
```

Ahora, por *dualidad*, dado que generamos outputs reactivos en el servidor, debemos generar un espacio en la UI que permita al usuario visualizarlos, en el siguiente bloque se agrupa toda la aplicación y se realizan las modificaciones necesarias.

```
list.of.packages<-c("plot3D","shinydashboard","shiny")
new.packages<-list.of.packages[!(list.of.packages %in% installed.packages()
                                 [,"Package"])]
if(length(new.packages)) install.packages(new.packages)

require(plot3D)
require(shinydashboard)
require(shiny)

header<-dashboardHeader(title = "Operaciones binarias")
sidebar<-dashboardSidebar(disable = TRUE)
body<-dashboardBody(
  fluidRow(
    column(3, # Se ha cambiado el tamanio de la columna para mejorar la visualizacion
           numericInput("parametro1","Parametro 1",1),
           numericInput("parametro2","Parametro 2",1),
           selectInput("parametro3","Parametro 3", choices = c("Suma","Producto"))
           ),
    column(6, #Se añade una columna extra que permita organizar el output reactivo
           box("Operacion:",h2(textOutput("operacion")),width = 6),
           box("Resultado:",h2(textOutput("resultado")),width = 6),
           box("Grafico:",plotOutput("grafico"),width = 12,height = 500)

           # h2 es una utilidad de HTML que se ha adaptado a R como una funcion y
              # es la abreviatura de "Header tipo 2", permite agrandar el texto
              # para darle estilo de titulo.

              # La funcion box permite generar "cajas flotantes" alrededor del output
```

```
              # reactivo, lo que da una presentacion mas ordenada, el primer argument
              # es el titulo de la caja.

              # Para agregar el output reactivo a la UI, se añade como prefijo el tipo
              # de salida (que debe coincidir con el sufijo de "render") seguido del
              # sufijo "Output", dentro del mismo se coloca el identificador clave, el
              # cual comparten la UI y el Servidor, observe Output$operacion y
              # textOutput("operacion"), este par es la escencia de la dualidad y la
              # forma de transferir información del servidor a la UI.
            )))
ui<-dashboardPage(header = header,sidebar = sidebar,body = body)

server<-function(input,output){
  data<-reactiveValues(contenido=NULL)
  observeEvent(c(input$parametro1,input$parametro2,input$parametro3),
               { output$operacion<-renderText({

                   if(input$parametro3=="Suma"){

                     paste(input$parametro1,"+",input$parametro2)

                   } else {
                     paste(input$parametro1,"*",input$parametro2)
                     }
                 })

                 output$grafico<-renderPlot({
                  if(input$parametro3=="Suma"){
    resultado<-sum(input$parametro1,input$parametro2)
    x0 <- c(0, 0, 0)
    y0 <- c(0, 0, 0)
    z0 <- c(0, 0, 0)
    x1 <- c(input$parametro1,0,0)
    y1 <- c(0,input$parametro2,0)
    z1 <- c(0,0,resultado)
    grafico<-arrows3D(x0, y0, z0, x1, y1, z1,col = c("black","black","red"),
    bty = "g",phi=0,theta = 10) +text3D(x1, y1+0.05,
    z1+0.05, c(input$parametro1,input$parametro2,resultado), colvar = NULL, add=TRUE)
  } else{
    resultado<-prod(input$parametro1,input$parametro2)
    x0 <- c(0, 0, 0)
    y0 <- c(0, 0, 0)
    z0 <- c(0, 0, 0)
    x1 <- c(input$parametro1,0,0)
    y1 <- c(0,input$parametro2,0)
    z1 <- c(0,0,resultado)
    grafico<-arrows3D(x0, y0, z0, x1, y1, z1,col = c("black","black","red"),
    bty = "g",phi=0,theta = 10)+text3D(x1, y1+0.05,
    z1+0.05, c(input$parametro1,input$parametro2,resultado), colvar = NULL, add=TRUE)
  }
                 })
    output$resultado<-renderText(if(input$parametro3=="Suma"){
      resultado<-sum(input$parametro1,input$parametro2)
```

```
      paste(resultado)
    } else{
      resultado<-prod(input$parametro1,input$parametro2)
      paste(resultado)
      })})}

shinyApp(ui,server)
```

Observación: Puede ejecutarse el código escrito en un script interactivo pegándolo y borrando los comentarios (aquellos que comienzan con #).

Con lo anterior, estará lista la aplicación.

Puede observarse:

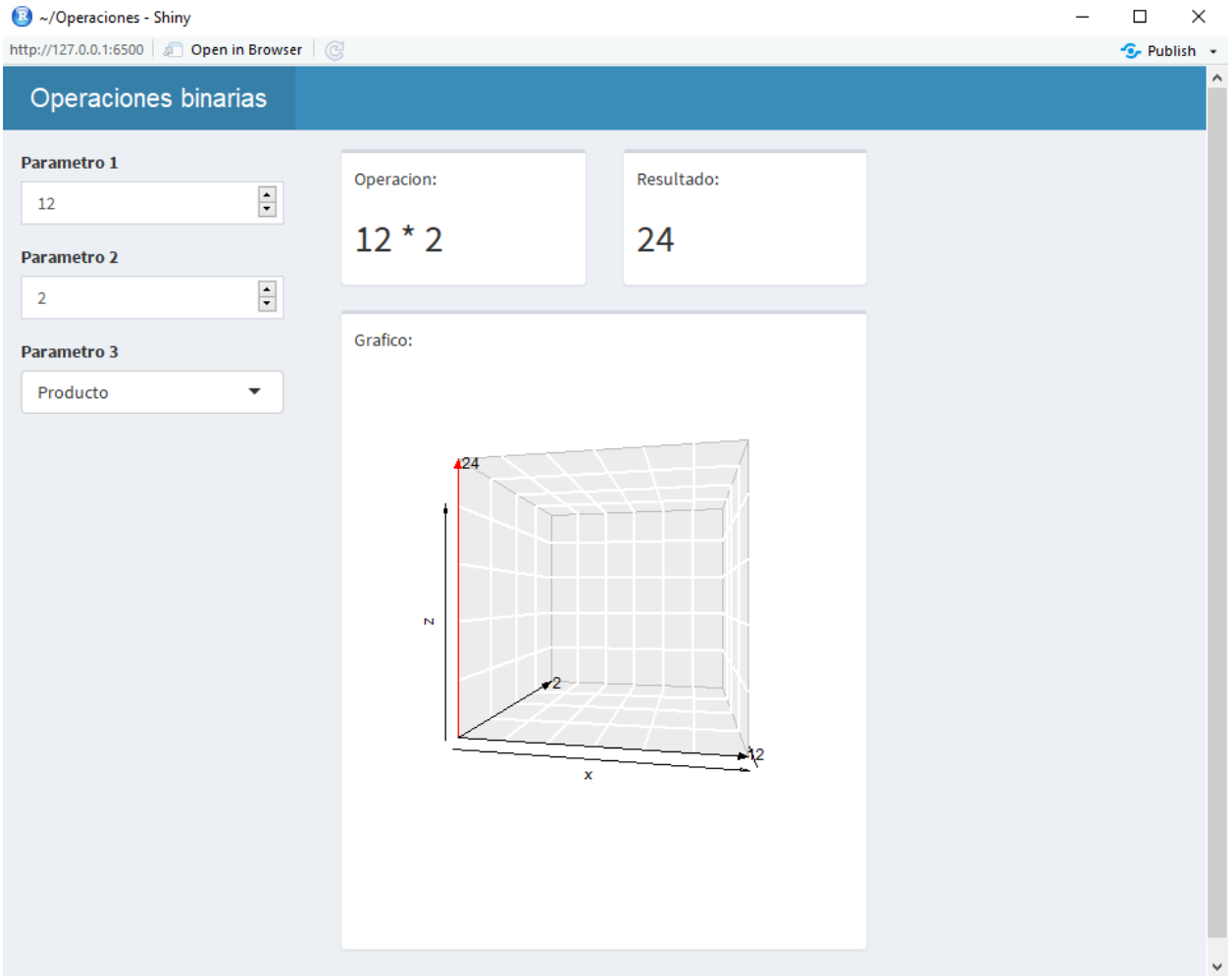

Figura 10: Aplicación interactiva - Operaciones

Caso práctico

El ejemplo que se ha desarrollado en la sección anterior ha tenido como objetivo que el lector comprenda una lógica y una filosofía de trabajo probablemente nueva, sin embargo, todo ese esfuerzo resulta insatisfactorio si no se presenta una meta que sea de provecho para el lector, con base en ello, se presenta un **Caso práctico** que como obra en el título del presente texto tiene como nombre *Valuación de obligaciones*.

Panorama general

En las finanzas de toda empresa existen responsabilidades monetarias que la empresa debe afrontar en un determinado periodo de tiempo, dichas responsabilidades implican el pago de una suma monetaria a otra entidad o el incremento de la deuda de la empresa, estas responsabilidades las denominaremos **obligaciones**.

Realizar un correcto seguimiento de las obligaciones eleva directamente la salud financiera de la empresa, pues permite una mejor administración de los recursos en términos de solvencia [8] (páginas 64-68).

Como administrador se debe contar con una herramienta que permita realizar dicha tarea y que tenga una sólida base matemática que permita fiarse de la misma, además debe ser de fácil uso y rápido aprendizaje, lo que permite que pueda compartirse a individuos no especializados en la programación o incluso en los modelos que se encuentran detrás y, aun así, pueda ser de su provecho. En otras palabras, que, pese a la complejidad del desarrollo y la matemática de fondo, la herramienta pueda ser operada fácilmente y por cualquiera, lo que permite independizar al desarrollador de proveer la información que genera el proceso plasmado en la herramienta y que la información sea de *Autoservicio*.

La herramienta de la que hablamos es una aplicación interactiva y se desarrolla más adelante.

Preliminares

Para abordar el problema (recuerde el primer paso de la metodología), se enuncia un conjunto de **definiciones matemáticas** sencillas relativas a los conceptos que se utilizarán y se recurre a la teoría de las probabilidades y la teoría de los procesos estocásticos; en especial de las series de tiempo, haciendo uso de algunos resultados.

Definición (Obligación):

Responsabilidad contraída por una empresa bajo la cual se compromete al pago de una suma monetaria en un determinado periodo de tiempo, denotaremos una obligación con la letra O, mientras que al conjunto de todas las obligaciones lo denotaremos con la letra D.

Definición (Cartera de obligaciones):

Una cartera de obligaciones es una sucesión de obligaciones, se denota por $\mathbf{O}=\{O^1, O^2, ...O^n\}$ donde O^k es la k-ésima obligación con $k \in [1, n]$.

Definición (Valor de la obligación) :

Sea $V{:}D \to \mathbb{R}^+$ una función tal que para cada obligación en D asigna un valor real positivo o cero, se llamará a V el *valor de la obligación*.

Temporalidad de las obligaciones

La responsabilidad del pago de una obligación puede estar sujeta a recurrencia, es decir, una obligación puede implicar más de un pago, cada uno efectuado en diferentes periodos de tiempo, esta característica dota a la obligación de una dimensión temporal. Dicha dimensión temporal solo puede ser definida a partir de una base de conteo que permita definir periodos en los que al interior la obligación se encuentre en curso y en los extremos (inicio y fin del periodo) se efectúen los pagos asociados para cada periodo. La base de conteo

puede estar dada por un determinado número de días naturales, por una base semanal, mensual, semestral, entre otras.

El manejo de dicha temporalidad se hará a través de las series de tiempo, se mencionan ciertas definiciones de utilidad provenientes de [9].

Definición (Serie de tiempo discreta):

Una *serie de tiempo discreta* es un conjunto de observaciones de una variable X denotadas como X_{t_i} con $t_i \in T$, donde cada observación es realizada en el tiempo t_i y $T \subset \mathbb{N} \cup \{0\}$. En adición, se llamará a T el *conjunto temporal*.

La definición anterior permitirá conocer la evolución de la obligación a través del tiempo, sin embargo, la obligación en sí misma es solo el nombre asignado a la responsabilidad contraída, por lo que el análisis se realizará sobre el valor de la obligación, es por ello que es necesario reajustar la definición del valor de la obligación.

Definición (Valor temporal de la obligación) :

Sea V_{t_i}:D x $T \to \mathbb{R}^+$ una función tal que para cada obligación en D y cada tiempo en T asigna un valor real positivo o cero, se llamará a V_{t_i} el *valor temporal de la obligación en* t_i.

Definición (Punto de corte):

Sea T un conjunto temporal asociado a una obligación O, se define $t_0 := max\{t_k \in T : k \in \mathbb{N}\}$ llamado el *punto de corte de la obligación*.

La definición implica que después del punto de corte no existe un valor temporal conocido para la obligación.

Definición (Historia)

Sea O una obligación, t_0 un punto de corte relativo a la obligación y T el conjunto temporal de la obligación, se define la sucesión H:=$\{V_{t_k}(O) : t_k \leq t_0\}$ llamada la *historia* de la obligación.

Por definición, la historia de la obligación es una serie de tiempo discreta del valor temporal de la obligación en cada t_k.

Esta sucesión almacena todas las observaciones dadas del valor temporal de la obligación.

Definición (Valuación):

Sea O una obligación, t_0 un punto de corte relativo a la obligación y T el conjunto temporal de la obligación, se define la sucesión de variables aleatorias $\{\hat{V}_k\}_{k\in\mathbb{N}}$ tal que para cada $t_k \in T$ que cumpla que $t_k \geq t_0$ entonces $\hat{V}_k$=$V_{t_k}(O)$. El k-ésimo elemento de la sucesión recibe el nombre de *valuación* de la obligación al tiempo t_k.

La sucesión anterior representa un proceso estocástico [10]..

Resulta ineficiente obtener una distribución del proceso estocástico, por lo que se hará uso de la historia de la obligación para que mediante modelos de series de tiempo se pueda llegar a definir correctamente una valuación o un conjunto de ellas, con ello se hace referencia a poder obtener una estimación aceptable del valor real que tendría el valor de la obligación o conjunto de valores, si ya se encontraran en la historia de la obligación.

Para dicho fin, el lector puede hacer uso de la teoría que desee, en el presente Caso se hará uso del modelo de series de tiempo **ARIMA**.

El modelo ARIMA

El Modelo Autorregresivo Integrado de Media Móvil (ARIMA, por sus siglas en inglés) es un modelo muy útil cuando se desea usar datos conocidos para estimar valores futuros. Una definición rigurosa puede encontrarse en [9] (páginas 180 y 181).

Dicho modelo está compuesto por tres parámetros:

- Componente autorregresiva: **p**
- Componente integrada: **d**
- Componente de media móvil: **q**

Se recomienda al lector consultar la bibliografía mencionada [9] para mayores referencias, dado que no se hará un tratamiento riguroso del modelo, únicamente se utilizarán resultados ya probados y podrían asumirse sin mención alguna.

Integración en carteras

Típicamente la empresa no cuenta con una obligación, más bien con un conjunto de ellas, dicho conjunto se ha definido como una cartera de obligaciones **O**. La determinación de valuaciones se realiza obligación por obligación de la cartera y el modelo que se utiliza es el mismo, aunque el ajuste varíe, sin embargo, hay algo que se debe considerar llamado la *componente de no pago*.

Para una determinada base de conteo (semanal, mensual, semestral, entre otras) pueden existir periodos donde al final de los mismos no se lleve a cabo el pago de la obligación, esta condición será llamada **no pago** y se puede deber a diversas razones como las condiciones mismas de la obligación.

Dicha condición no puede incluirse en la historia de la obligación como el valor de la obligación en un determinado momento, pues el razonamiento sería incorrecto ya que no se trata en principio de un valor. Por lo tanto, dicha componente debe servir para definir otro proceso.

Definición (Proceso de pagos):

Sea O una obligación y T el conjunto temporal asociado a la obligación, se define una sucesión de indicadoras $P=\{P_{t_k}\}_{t_k \epsilon T}$ llamado el *proceso de pagos*, donde el k-ésimo elemento de P es P_{t_k} y toma el valor de 1 si y solo si $V_{t_k}(O)$ existe (no necesariamente distinto de cero) y 0 en otro caso.

Definición (Historia de pagos):

Sea O una obligación, T el conjunto temporal asociado a la obligación y t_0 el punto de corte de la obligación, entonces el subconjunto $H_p := \{P_{t_k} : t_k \leq t_0\}$ recibe el nombre de la *historia de pagos*.

Definición (Probabilidad de pago)

Sea O una obligación y H_p la historia de pagos asociada, se define.

$$p := \frac{\sum_{t_k \epsilon H_p} P_{t_k}}{card(H_p)}$$

con *card* siendo la cardinalidad del conjunto. Se llama a p la *probabilidad de pago* de la obligación.

Definición (Probabilidad de ningún pago)

Sea **O** una cartera de n obligaciones, se llamará *probabilidad de ningún pago* a la probabilidad de que ninguna obligación se pague en el tiempo t, con $t \geq t_0$ donde t es un punto de corte común de todas las obligaciones.

Se define como

$$p_0 := \prod_{k=1}^{n} (1 - p_k)$$

En cuanto se tenga determinada una o múltiples valuaciones para cada obligación de la cartera, se procede a integrarla para un tiempo t_k, lo que implica poder definir el valor total de la cartera, para ello recurrimos a la siguiente definición.

Definición (Valor de la cartera):

Sea $\mathbf{O}=\{O^1, O^2, ...O^n\}$ una cartera de n obligaciones y t un punto de corte común para todas las obligaciones de la cartera, se define

$$VC_{t_k} := \sum_{k=1}^{n} V_{t_k}(O^k)$$

llamado el *valor de la cartera* al tiempo t_k con $t_k \leq t_0$.

Recordando que, tras el punto de corte el valor de las obligaciones debe ser estimado, entonces un resultado directo de la definición anterior es que tras el punto de corte, el valor de la cartera es una suma de variables aleatorias.

Definición (Pronóstico de la cartera de obligaciones):

Sea $\mathbf{O}$ una cartera de obligaciones y t_0 un punto de corte común para todas las obligaciones de la cartera, entonces el valor de la cartera para tiempos posteriores al punto de corte sujeto a posibles condiciones de no pago se denomina un *pronóstico de la cartera de obligaciones* y se denota por $\widehat{VC}$.

Se hará uso de dos tipos de pronósticos.

Pronósticos

Sea $\mathbf{O}$ una cartera de n obligaciones, t_0 un punto de corte común para todas las obligaciones de la cartera y t algún tiempo tal que $t \geq t_0$ se definen:

Definición (Pronóstico ponderado):

Cuando se conoce con certeza si el pago de una o más obligaciones será realizado o no para el tiempo t, entonces la probabilidad de pago p de cada obligación en la mencionada situación únicamente puede tomar alguno de los valores $\{0, 1\}$ y sustituye a la probabilidad de pago propia de la definición, esta capacidad de decidir sobre la certeza de pago mejora la estimación del pronóstico de la cartera de obligaciones, se define

$${}^{p}\widehat{VC}_t := \sum_{k=1}^{n} V_t(O^k) * p_k$$

llamado el *pronóstico ponderado* de la cartera de obligaciones al tiempo t.

Definición (Pronóstico esperado):

Sea $\{V_t(O^k)\}_{k=1}^{n}$ el conjunto de valores de las obligaciones de la cartera, μ la media de dicha sucesión de valores (o la mediana si resulta más conveniente) y $\{p_k\}_{k=1}^{n}$ las probabilidades de pago para cada obligación, defina la variable aleatoria N como el número de obligaciones totales a pagar, entonces N sigue una distribución Binomial-Poisson de parámetros $\{p_k\}_{k=0}^{n}$ (véase [11]).

La función de densidad f de la variable aleatoria N expresa, por lo tanto, la probabilidad de que N tome algún valor entre 0 y n entendido como el total de obligaciones a pagar.

Para fines de cálculo, se usará la función de densidad expresada a través de la Transformada Discreta de Fourier, el lector puede conocer mayores detalles en [11].

Sea $\hat{k} := max\{k\epsilon[0,n] : f(k) \geq f(j)\}$ para toda $j \neq k$ y $j \epsilon [0,n]$, se define

$$^{e}\widehat{VC}_t := \hat{k} * \mu$$

llamado el *pronóstico esperado*.

Con ello se concluye la sección de definiciones, las cuales serán útiles para manejar de manera sencilla el problema propuesto.

Construyendo la aplicación interactiva

La aplicación que se construirá tendrá como finalidad valuar obligaciones, a través de las definiciones mencionadas se comprendió que valuar la obligación finalmente implica obtener el pronóstico de un valor futuro incierto, pues la valuación de obligaciones que ya se han pagado es completamente determinista.

Dicho lo anterior, se prosigue con el desarrollo de la metodología.

1) El problema de valuar obligaciones y más aún, de encontrar un valor único para toda una cartera de obligaciones halla su solución en los pronósticos que se han definido, los cuales serán de utilidad para que el administrador tenga cierta idea del valor real de la obligación y la pueda afrontar de manera correcta. Más aún, será útil contar con gráficos para comprender mejor la evolución de las obligaciones, lo cual permitirá respaldar de manera gráfica las soluciones que ofrezca el modelo.

Para dar solución al problema, se supondrá que existe una cartera de obligaciones **O** que consta de n obligaciones para las cuales se tiene como único supuesto que existen al menos tres observaciones en la historia H de cada obligación, posteriormente se deberá ajustar un modelo ARIMA para cada obligación, con el modelo se podrán determinar los valores de las obligaciones $V_t(O^1), V_t(O^2), ..., V_t(O^n)$ para algún tiempo t superior al punto de corte t_0 común de las obligaciones, luego se obtendrán las probabilidades de pago $p_1, ..., p_n$ para cada obligación y con todo lo anterior se obtiene como producto un par de pronósticos, el esperado y el ponderado para la cartera completa.

Será además de utilidad contar con los valores de las obligaciones obtenidos a través del modelo ARIMA y gráficas de la evolución del valor de cada obligación y de la cartera completa, por lo cual también será necesario el cálculo del valor de la cartera para la historia de las obligaciones.

2) Los parámetros del problema serán:

Parámetro 1: La información de la historia de las obligaciones, conteniendo los valores de las obligaciones, la cual almacenaremos en una sola hoja de datos, dicho parámetro tendrá como dominio una hoja de datos con alguna de las disposiciones dadas en el parámetro 4.

Parámetro 2: Tipo de pronóstico con dominio en la lista de valores {Esperado,Ponderado}.

Parámetro 3: Tipo de probabilidades de pago por obligación con dominio en la lista de valores {Definicion,Asignable}, en el caso de que sea de tipo asignable, se depende también del siguiente parámetro.

Parámetro 3.1 (Opcional): Valor de la probabilidad de pago en tipo de probabilidad asignable con dominio en el intervalo [0,1] por obligación. Contenidas en una hoja de datos, ordenadas como obligación en la primera columna, probabilidad en la siguiente columna, vacío si no se desea reemplazar la probabilidad por definición.

Con dichos parámetros el problema tendrá una solución con base en lo definido.

3) El servidor será de la siguiente manera:

El usuario ingresa el parámetro 1:

Importa una hoja de datos en formato CSV

El usuario ingresa el parámetro 2:

Selecciona el tipo de pronóstico

El usuario ingresa el parámetro 3:

Selecciona el tipo de probabilidades

Si se cumple que

parámetro 3 = Asignable

Entonces el usuario ingresa una hoja de datos con las probabilidades y las visualiza

En otro caso

El parámetro 3.1 se ignora

El usuario confirma los parámetros anteriores con un botón

> El servidor obtiene los valores de cartera para la historia de las obligaciones presente en la hoja de datos

> El servidor ajusta un modelo ARIMA por obligación obteniendo un valor de la obligación que continúe la serie de tiempo

> El servidor calcula el valor de la cartera que continúe la serie de tiempo (pronóstico) basado en el tipo de pronóstico y con uso de las probabilidades de pago

> El servidor grafica la serie de tiempo del valor de las obligaciones y la serie relativa al valor de la cartera

> El servidor genera un output reactivo conteniendo una hoja de datos en formato CSV similar a la ingresada pero extendida con los datos calculados

El servidor termina la ejecución y envía un mensaje de éxito

El servidor se encuentra listo para una nueva ejecución

---

4) La interfaz de usuario será como sigue:

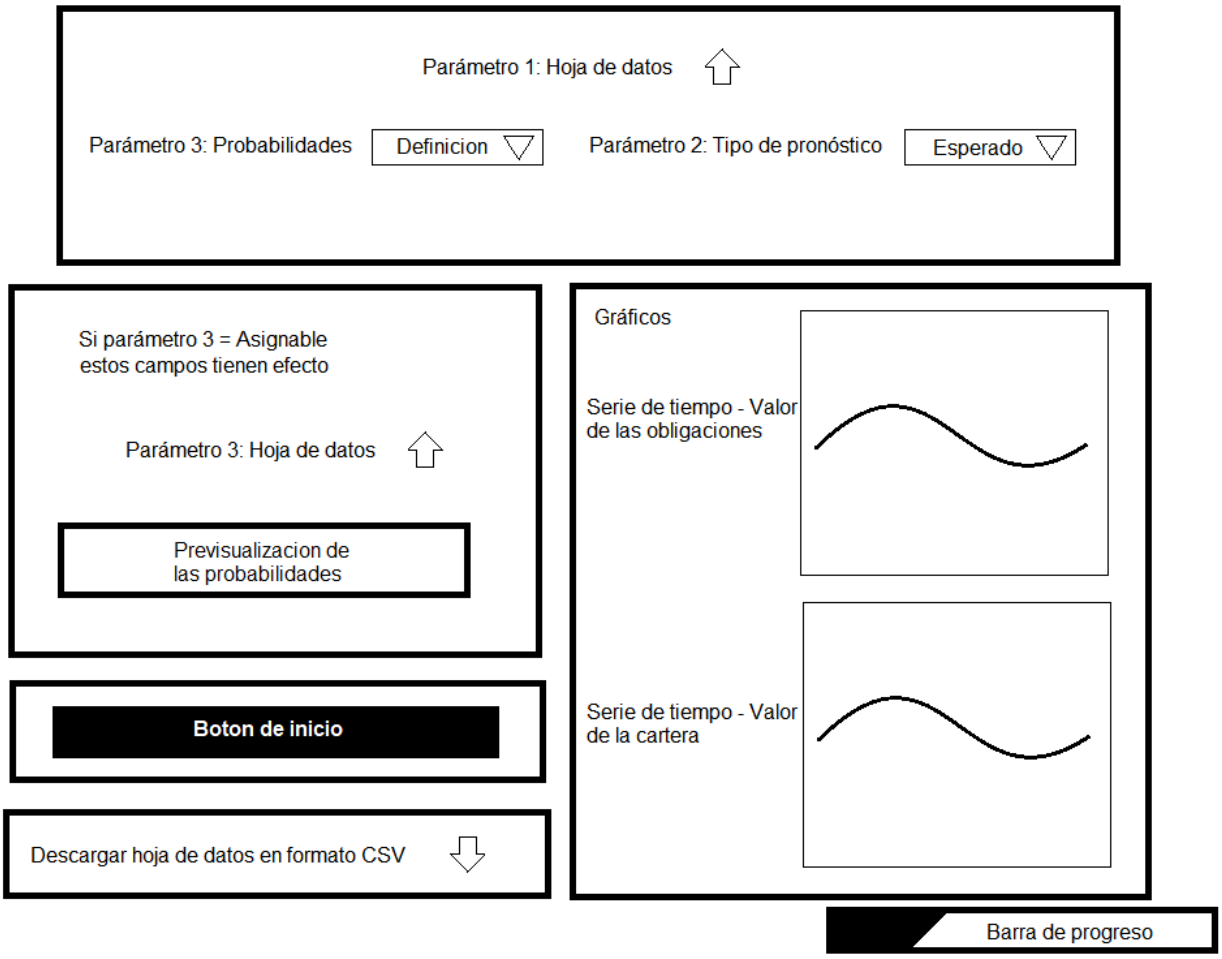

Figura 11: Interfaz de usuario - Caso práctico a)

Y se desplegarán dos mensajes, al apretar el botón de inicio se pedirá confirmar la acción, luego, al terminar la ejecución se enviará un mensaje de éxito, se incluye además una barra de progreso:

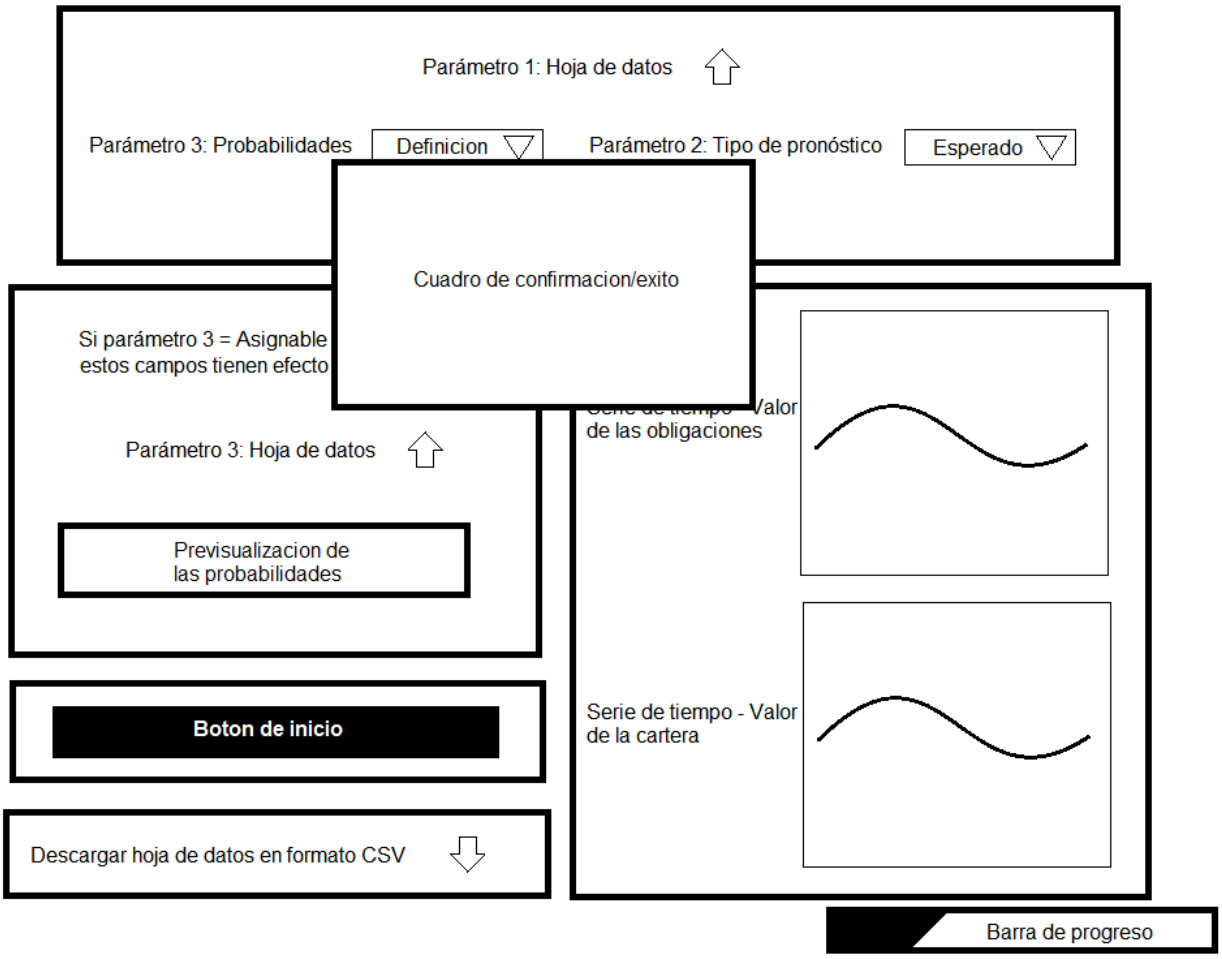

Figura 12: Interfaz de usuario - Caso práctico b)

5) Se sentencian valores arbitrarios para realizar pruebas de funcionalidad previo a la integración en la estructura interactiva.

Para ello se hará uso de las hojas de datos que se muestran.

En la primera se encuentra un ejemplo de una cartera de obligaciones, dicha disposición tendrá que respetarse a la hora de ingresar a la aplicación nuevas hojas de datos:

Obligacion	Valor	Anio	Mes
Servicios	216000	2017	1
Servicios	218900	2017	2
Servicios	220300	2017	3
Servicios	225600	2017	4
Servicios	228900	2017	5
Servicios	230400	2017	6
Nomina	150000	2017	1
Nomina	155000	2017	2
Nomina	155000	2017	3
Nomina	165000	2017	4
Nomina	167000	2017	5
Nomina	180000	2017	6
Mantenimiento	15000	2017	1
Mantenimiento	18000	2017	2
Mantenimiento	19000	2017	3
Mantenimiento	19200	2017	4
Mantenimiento	18000	2017	5
Mantenimiento	18699	2017	6
Insumos de oficina	NA	2017	1
Insumos de oficina	7998	2017	2
Insumos de oficina	NA	2017	3
Insumos de oficina	9520	2017	4
Insumos de oficina	NA	2017	5
Insumos de oficina	11299	2017	6

Los valores `NA` hacen referencia a la condición de no pago.

En la segunda, se encuentra una muestra de probabilidades de pago asignables:

Obligacion	Probabilidad
Servicios	NA
Nomina	NA
Mantenimiento	NA
Insumos de oficina	1

Los valores `NA` hacen referencia a que la probabilidad de dicha obligación será por definición.

Se abre un script nuevo de RStudio y se escribe lo siguiente.

```
#Librerías a utilizar
list.of.packages <- c("forecast","tseries","zoo","lubridate","ggplot2")
new.packages<-list.of.packages[!(list.of.packages %in% installed.packages()
                                 [,"Package"])]
if(length(new.packages)) install.packages(new.packages)

require(tseries) # La libreria tseries genera objetos de series de tiempo
require(forecast) # La libreria forecast contiene el modelo ARIMA
require(zoo) #Libreria de manejo de fechas
require(lubridate) # Libreria de manejo de fechas
require(ggplot2) # Permite realizar gráficos
```

```r
# Sentencia de valores arbitrarios para los parametros

setwd(choose.dir()) # Importante definir el directorio de trabajo
                    # como la carpeta donde se encuentre la hoja de datos

parametro1<-read.csv("./datos.csv",sep = ",") # Importacion de la tabla de datos
      # sep (separador) puede ser {, ; :}, depende de la configuracion del equipo
# Se hará uso del pronóstico esperado

parametro2<-"Esperado"

# Se hará uso de probabilidades asignables

parametro3<-"Asignable"

# Se construye la serie de tiempo

obligaciones<-levels(as.factor(parametro1$Obligacion)) # Obligaciones

# Lista de sub hojas de datos por obligacion

obslist<-list()
j<-1
for(i in obligaciones){
  obslist[[j]]<-parametro1[which(parametro1$Obligacion==i),]
  j<-j+1}

# Serie de tiempo por obligación

tslist<-list()
for (i in 1:length(obslist)) {
  tslist[[i]]<-ts(obslist[[i]][,2],start = c(min(obslist[[i]][,3]),
                          min(obslist[[i]][,4])),frequency = 12 )}

# Ajuste de ARIMA por obligación sobre todos los modelos posibles
# sin realizar aproximaciones para los criterios de información

arimalist<-list()
for (i in 1:length(obslist)) {
  arimalist[[i]]<-auto.arima(tslist[[i]],stepwise = FALSE, approximation = FALSE)}

# Valores estimados de las obligaciones con un 95% de confianza

vals<-list()
for (i in 1:length(obslist)) {
  vals[[i]]<-forecast(arimalist[[i]],level = 95, h=1)$mean}

# Se extraen las fechas de corte

fechaslist<-list()
for (i in 1:length(obslist)) {
  fechaslist[[i]]<-as.yearmon(paste0(max(as.numeric(obslist[[i]][,3])),"-"
                                     ,max(as.numeric(obslist[[i]][,4]))))}
```

```r
# Nuevas fechas

nuevasfec<-list()
for (i in 1:length(obslist)) {
nuevasfec[[i]]<-as.Date(as.yearmon(fechaslist[[i]])+.1)}

# Se completa la serie de tiempo del valor de las obligaciones

nuevats<-list()
for (i in 1:length(obslist)) {
  nuevats[[i]]<-rbind(obslist[[i]],c(obligaciones[i],
 as.vector(vals[[i]]),year(nuevasfec[[i]]),month(nuevasfec[[i]])))}

# Se calculan las probabilidades de pago por obligacion

probslist<-list()
for (i in 1:length(obslist)) {
  probslist[[i]]<-sum(!is.na(obslist[[i]][,2]))/nrow(obslist[[i]])}

# Probabilidad de ningún pago

p0<-1
for (i in 1:length(obslist)) {
  p0<-prod(1,1-probslist[[i]])}

# Unico vector de probabilidades

probs<-c(p0)
for (i in 1:length(obslist)) {
  probs<-c(probs,probslist[[i]])
}

# Serie de tiempo de valores de cartera en historia

carvals<-c()
z<-1
for (i in levels(as.factor(parametro1$Anio))) {
  for (j in levels(as.factor(parametro1$Mes))){
carvals[z]<-sum(parametro1$Valor[which(parametro1$Anio==i
                  & parametro1$Mes==j)],na.rm = TRUE)
z<-z+1
  }
}

# k gorro

kdf<-data.frame(Valores=0:length(obslist),
Probabilidades=dpoisbinom(0:length(obslist),probs)) #Distribucion Binomial-Poisson
k.hat<-kdf$Valores[which(kdf$Probabilidades==max(kdf$Probabilidades))]

# Se genera el pronóstico

if(parametro2=="Esperado"){
```

```
  valor<-c()
  j<-0
  for (i in 1:length(obslist)) {
    valor<-sum(valor,as.vector(vals[[i]]),na.rm = TRUE)
    j<-j+1}
  media<-valor/j
  pronostico<-k.hat*media # Pronostico esperado
  } else {
  valores<-c()
  probabilidades<-c()
  for (i in 1:length(obslist)) {
    valores[i]<-as.vector(vals[[i]])
    probabilidades[i]<-probslist[[i]]
  }
  pronostico<-sum(valores*probabilidades,na.rm = TRUE) #Pronostico ponderado
  }

# Se continua la serie de tiempo del valor de cartera

fechas<-nuevats[[1]][,c(3,4)]

nuevacarval<-data.frame(fechas,Valores=c(carvals,pronostico))

# GRAFICOS
# Se colapsan las nuevas series de tiempo en una sola hoja de datos

df<-nuevats[[1]]
for (i in 2:length(obslist)) {
df<-rbind(df,nuevats[[i]])}

fecha_corta<-c()
for (i in 1:nrow(df)) {
  fecha_corta[i]<-ifelse(as.numeric(df$Mes[i])<10,
paste0(df$Anio[i],"-0",df$Mes[i]),paste0(df$Anio[i],"-",df$Mes[i]))}

df<-cbind(df,Fecha_corta=fecha_corta)

# Grafico de valores de obligaciones

plot.obs<-ggplot(df, aes(x = df$Fecha_corta, y = as.numeric(df$Valor),
  group=df$Obligacion),na.rm=TRUE) +
  geom_line(aes(color = df$Obligacion))+geom_point(aes(color=df$Obligacion))+theme_bw()+
  labs(title = "Serie de tiempo del valor de las obligaciones",
       x="Fecha",y="Valor de la obligacion",color="Obligacion")+
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1))

# Grafico de valores de la cartera

df1<-cbind(nuevacarval,Fecha_corta=fecha_corta[1:nrow(nuevacarval)])

plot.car<-ggplot(df1, aes(x = df1$Fecha_corta, y = as.numeric(df1$Valor)),na.rm=TRUE) +
  geom_line(aes(group=1),col="blue")+geom_point(aes(group=1))+theme_bw()+
  labs(title = "Serie de tiempo del valor de la cartera",
```

```
      x="Fecha",y="Valor de la cartera")+
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1))
```

De donde se observan los valores estimados de las obligaciones y de la cartera para Julio-2017, así como los gráficos (se mostrará al final del desarrollo).

6) Dado que lo anterior ha tenido un resultado favorable, se procede a la integración en una estructura interactiva como se aprecia a continuación.

```
list.of.packages <- c("ggplot2", "shiny","shinydashboard","kableExtra","lubridate")
new.packages<-list.of.packages[!(list.of.packages %in% installed.packages()
                                 [,"Package"])]
if(length(new.packages)) install.packages(new.packages)

library(shiny)
library(shinydashboard)
library(ggplot2)
library(lubridate)
require(kableExtra)

header<-dashboardHeader(title = "Valuacion de obligaciones",titleWidth = 300)
sidebar<-dashboardSidebar(disable = TRUE)

body<-dashboardBody(
  box(h2("Parametros"),align="center",width = 12,
    box(icon("database"),width = 12,background = "green",
    fileInput("par1","Hoja de datos de obligaciones",
    placeholder = "Archivo CSV con valores de las obligaciones", buttonLabel = "Buscar")),
    box(width = 6,background = "green",
    selectInput("par3","Tipo de probabilidades",choices = c("Asignable","Definicion"),
                selected = "Definicion")),
    box(width = 6,background = "green",
    selectInput("par2","Tipo de pronostico",choices = c("Esperado","Ponderado"),
                selected = "Esperado"))),
  fluidRow(width=12,
  box(h2("Probabilidades de pago"),width = 6,align="center",
             box(width = 12,background = "green",
                 fileInput("par3.1","Hoja de datos de probabilidades",
placeholder = "Archivo CSV con probabilidades",
buttonLabel = "Buscar",
            accept = c("text/csv","text/comma-separated-values,text/plain",".csv"))),
      box("Previsualizacion",width = 12,background = "navy",tableOutput("previs")),
      box(h2("Resultados"),h2(icon("file-excel")),width = 12,
          box(width = 12,background = "green",actionButton("inicio",
                                        "Comenzar los calculos",icon = icon("play"))),
          box(h5("Serie de tiempo del valor de las obligaciones"),width = 12,
              background = "green",downloadButton("obs",icon("download"),
                                                  label = "Descargar resultados")),
          box(align="center",h5("Serie de tiempo del valor de la cartera"),width = 12,
              background = "green",downloadButton("car",icon("download"),
                                                  label = "Descargar resultados")))),
  box(h2("Graficos"),width = 6,
      box(h5("Valor de las obligaciones"),align="center",
          width = 12,background = "navy",plotOutput("plotobs")),
      box(h5("Valor de la cartera"),align="center",
```

```
          width = 12,background = "navy",plotOutput("plotcar")))))

ui<-dashboardPage(header = header,sidebar = sidebar,body = body)

server<-function(input,output){

  observeEvent(input$par3.1,{
    output$previs<-renderTable({

      inFile <- input$par3.1

      if (is.null(inFile))
        return(NULL)

      read.csv(inFile$datapath)})})

  jeez<-modalDialog("Proceso completado",footer=tagList(actionButton('ok2',"Cerrar")))

  dataModal <- function(failed = FALSE) {
    modalDialog(
      "Estas seguro?",
      footer = tagList(
        modalButton("No"),
        actionButton("ok", "Si")))}

  observeEvent(input$inicio, {
    showModal(dataModal())})

  observeEvent(input$ok, {
    removeModal()})

  observeEvent(input$ok,{
    list.of.packages<-c("forecast","tseries","zoo","lubridate","ggplot2",
                        "poisbinom","dplyr")
    new.packages<-list.of.packages[!(list.of.packages %in% installed.packages()
                                     [,"Package"])]
    if(length(new.packages)) install.packages(new.packages)

    require(tseries)
    require(forecast)
    require(zoo)
    require(lubridate)
    require(ggplot2)
    require(poisbinom)
    require(dplyr)

    progress<-shiny::Progress$new()
    on.exit(progress$close())
    progress$set(message = "Haciendo ciencia...",value = 0.2)

    inFile1 <- input$par1
    if (is.null(inFile1))
      return(NULL)
```

```
parametro1<-read.csv(inFile1$datapath)
obligaciones<-levels(as.factor(parametro1$Obligacion))

obslist<-list()
j<-1
for(i in obligaciones){
  obslist[[j]]<-parametro1[which(parametro1$Obligacion==i),]
  j<-j+1}

tslist<-list()
for (i in 1:length(obslist)) {
  tslist[[i]]<-ts(obslist[[i]][,2],start = c(min(obslist[[i]][,3]),
                  min(obslist[[i]][,4])),frequency = 12 )}

arimalist<-list()
for (i in 1:length(obslist)) {
  arimalist[[i]]<-auto.arima(tslist[[i]],stepwise = FALSE, approximation = FALSE)}

vals<-list()
for (i in 1:length(obslist)) {
  vals[[i]]<-forecast(arimalist[[i]],level = 95, h=1)$mean}

fechaslist<-list()
for (i in 1:length(obslist)) {
  fechaslist[[i]]<-as.yearmon(paste0(max(as.numeric(obslist[[i]][,3])),"-"
                                     ,max(as.numeric(obslist[[i]][,4]))))}

nuevasfec<-list()
for (i in 1:length(obslist)) {
  nuevasfec[[i]]<-as.Date(as.yearmon(fechaslist[[i]])+.1)}

nuevats<-list()
for (i in 1:length(obslist)) {
  nuevats[[i]]<-rbind(obslist[[i]],c(obligaciones[i],
as.vector(vals[[i]]),year(nuevasfec[[i]]),month(nuevasfec[[i]])))}

probslist<-list()
for (i in 1:length(obslist)) {
  probslist[[i]]<-sum(!is.na(obslist[[i]][,2]))/nrow(obslist[[i]])}

if(input$par3=="Asignable"){

  inFile <- input$par3.1

  if (is.null(inFile))
    return(NULL)

  probs<-read.csv(inFile$datapath)

ps<-c()
for (i in 1:length(obslist)) {
  ps[i]<-probslist[[i]]}
```

```
probas<-data.frame(Obligacion=obligaciones,ps)

union<-left_join(probas,probs)

seleccion<-which(!is.na(union$Probabilidad))

for(i in seleccion){
  probslist[[i]]<-union[i,3]}}

p0<-1
for (i in 1:length(obslist)) {
  p0<-prod(p0,1-probslist[[i]])}

probs<-c(p0)
for (i in 1:length(obslist)) {
  probs<-c(probs,probslist[[i]])}

# Serie de tiempo de valores de cartera en historia

carvals<-c()
z<-1
for (i in levels(as.factor(parametro1$Anio))) {
  for (j in levels(as.factor(parametro1$Mes))){
    carvals[z]<-sum(parametro1$Valor[which(parametro1$Anio==i
                            & parametro1$Mes==j)],na.rm = TRUE)
    z<-z+1}}

kdf<-data.frame(Valores=0:length(obslist),
                Probabilidades=dpoisbinom(0:length(obslist),probs))
kdf$Valores<-as.numeric(kdf$Valores)
kdf$Probabilidades<-round(kdf$Probabilidades,10)
k.hat<-max(kdf$Valores[which(kdf$Probabilidades==max(kdf$Probabilidades))])

if(input$par2=="Esperado"){

  valor<-c()
  j<-0
  for (i in 1:length(obslist)) {
    valor<-sum(valor,as.vector(vals[[i]]),na.rm = TRUE)
    j<-j+1}
  media<-valor/j
  pronostico<-k.hat*media
  } else {
  valores<-c()
  probabilidades<-c()
  for (i in 1:length(obslist)) {
    valores[i]<-as.vector(vals[[i]])
    probabilidades[i]<-probslist[[i]]}
  pronostico<-sum(valores*probabilidades,na.rm = TRUE)}

fechas<-nuevats[[1]][,c(3,4)]

nuevacarval<-data.frame(fechas,Valores=c(carvals,pronostico))
```

```
    df<-nuevats[[1]]
    for (i in 2:length(obslist)) {
      df<-rbind(df,nuevats[[i]])}

    fecha_corta<-c()
    for (i in 1:nrow(df)) {
      fecha_corta[i]<-ifelse(as.numeric(df$Mes[i])<10,paste0(df$Anio[i],
                          "-0",df$Mes[i]),paste0(df$Anio[i],"-",df$Mes[i]))}

    df<-cbind(df,Fecha_corta=fecha_corta)

    output$plotobs<-renderPlot({
      ggplot(df, aes(x = df$Fecha_corta, y = as.numeric(df$Valor),
      group=df$Obligacion),na.rm=TRUE) +
      geom_line(aes(color = df$Obligacion))+geom_point(aes(color=df$Obligacion))
      +theme_bw()+
      labs(title = "Serie de tiempo del valor de las obligaciones",
           x="Fecha",y="Valor de la obligacion",color="Obligacion")+
      theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1))})

    df1<-cbind(nuevacarval,Fecha_corta=fecha_corta[1:nrow(nuevacarval)])

    output$plotcar<-renderPlot({
      ggplot(df1, aes(x = df1$Fecha_corta, y = as.numeric(df1$Valor)),na.rm=TRUE) +
      geom_line(aes(group=1),col="blue")+geom_point(aes(group=1))+theme_bw()+
      labs(title = "Serie de tiempo del valor de la cartera",
           x="Fecha",y="Valor de la cartera")+
      theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1))})

        output$obs <- downloadHandler(
          filename = function() {
            paste0("Serie_de_tiempo_obligaciones",
                   df$Fecha_corta[nrow(df)],".csv")},
          content = function(file) {
            write.csv(df, file, row.names = FALSE)})

        output$car <- downloadHandler(
          filename = function() {
            paste0("Serie_de_tiempo_cartera",
                   df$Fecha_corta[nrow(df)],".csv")},
          content = function(file) {
            write.csv(df1, file, row.names = FALSE)})

    progress$inc(1,detail="Haciendo ciencia...")
    showModal(jeez)})

  observeEvent(input$ok2,{
    removeModal()})
}

shinyApp(ui,server = server)
```

7) No se han encontrado errores que requieran depuración.

Con ello se finaliza la aplicación interactiva, se sugiere al lector ejecutar la aplicación con **Run App** y la opción **Run External**, así como seguir el desarrollo mismo. Se ofrece una vista de la aplicación:

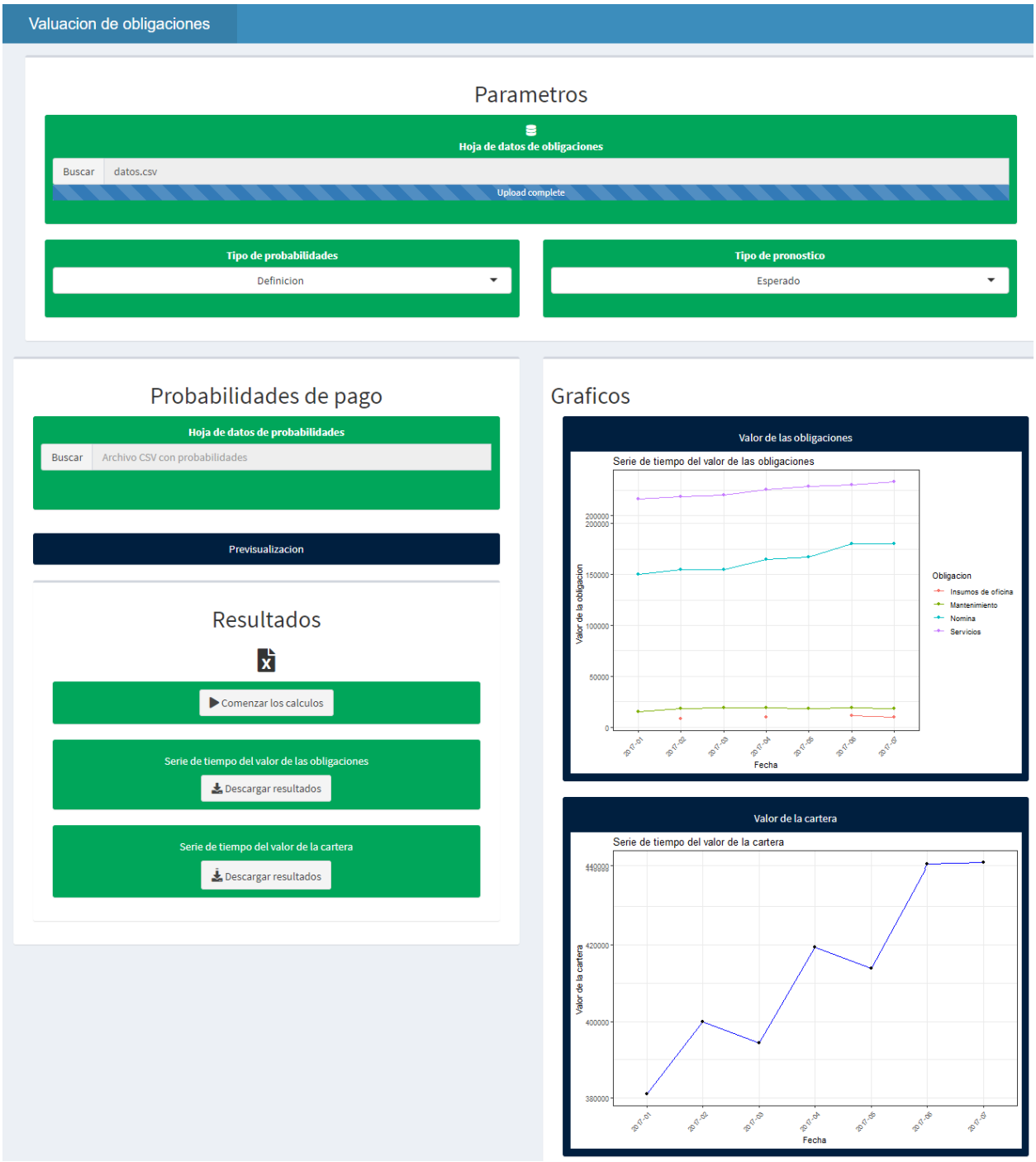

Figura 13: Aplicación interactiva - Caso práctico

Funcionalidades

El usuario de la aplicación podrá utilizarla para pronosticar valores futuros con información histórica, ingresando un conjunto de datos previamente estructurado. El uso de la aplicación no implica conocimiento alguno de los fundamentos teóricos detrás de la misma, así como tampoco implica el conocimiento previo de la estructura en código, pues la misma funciona a través de botones y cuadros de diálogo que cualquier persona con un conocimiento básico de computación puede manipular.

Lo anterior implica que como administrador no se necesita poseer el conocimiento específico de la herramienta para utilizarla, simplemente conocer la finalidad de la misma y la naturaleza o implicación de los parámetros, con ello se pueden obtener resultados satisfactorios y confiables a través de la misma.

El desarrollador implicado en la creación de la herramienta puede dedicarse a mejorar sus capacidades teóricas y de programación, generando herramientas cada vez más robustas y privándose de la tarea de suministrar información, pues las herramientas permiten que la información sea de *autoservicio*, con ello en el largo plazo se puede tener la confianza de que el responsable del desarrollo genere mejores herramientas y que la información siempre esté a la mano para quien la necesite a través de las mismas.

Bajo el supuesto de que se hace uso de la herramienta presente en el **Caso práctico**, suponga que desea conocer los flujos en términos de obligaciones que deberá pagar en el próximo cierre de mes, además suponga que no cuenta con los conocimientos ni la experiencia para valuar dichas obligaciones y que el actuario responsable de realizar dicha tarea se encuentra incapacitado y no puede atenderle. En ese caso, sería adecuado conocer el proceso que utiliza el actuario responsable para generar dicha información, más aún, sería adecuado poder generar dicha información usted mismo; es ahí donde radica el valor de la herramienta, en ofrecer dicha posibilidad sin la necesidad de poseer el conjunto de conocimientos del experto, pero con la confianza de que la información es correcta.

Finalmente, una herramienta similar puede diseñarse para todo tipo de valor futuro o para otros procesos, por lo tanto, no se limita a lo presente en el texto, todo depende de la creatividad y del objetivo que se planteen. A través de la metodología mostrada existe una infinidad de desarrollos posibles.

Con ello, se finaliza el texto, sin antes remarcar la importancia de seguir el desarrollo por uno mismo e intentar comprenderlo, pueden existir fallos o dudas, sin embargo, el fruto del conocimiento tiene dicha naturaleza.

Comentarios finales

- El lector con interés en compartir la aplicación, debe saber que la aplicación interactiva puede ser hospedada en un sitio web como una aplicación web y ser de acceso público, o para una red de trabajo, si se encuentra interesado en ello, se le invita a conocer el proyecto Shiny Server https://www.rstudio.com/products/shiny/shiny-server/.
- El lector con interés en el diseño de reportes, artículos científicos, documentación, etc. puede acercarse a la herramienta RMarkdown y combinarla con las aplicaciones interactivas para generar reportes automáticos https://rmarkdown.rstudio.com/.
- ???El lector puede encontrarse con múltiples dudas que desafortunadamente el texto no puede cubrir, sin embargo, existen foros como StackOverFlow donde el lector puede resolver sus dudas relativas a la programación ¡con expertos de todo el mundo! https://stackoverflow.com/.

Referencias

[1] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2018.

[2] RStudio Team. *RStudio: Integrated Development Environment for R.* RStudio, Inc., Boston, MA, 2015.

[3] R Core Team. *Introducción a R.* Vienna, Austria, Mayo 2000.

[4] Hassan Montero Yusef. *Experiencia de Usuario: Principios y Métodos.* Granada, España, 2015.

[5] Robert Bartle. *Introducción al Análisis Matemático de una Variable.* Limusa Wiley, México, 2004.

[6] Martínez Gil Francisco. *Introducción a la programación estructurada en C.* Valencia, España, 2003.

[7] Ferrer Jorge; García Victor; García Rodrigo. *Curso completo de HTML.*

[8] Díaz Llanes Miguel. *Análisis contable con un enfoque empresarial.* 2012.

[9] Peter Brockwell; Richard Davis. *Introduction to Time Series and Forecasting.* Nueva York, EEUU, 2002.

[10] Rincón Luis. *Introducción a los procesos estocásticos.* 2012.

[11] Y. H. Wang. *On the number of successes in independent trials.* Quebéc, Canadá, 1993.