

DOCUMENTACIÓN VISUALIZADOR ACCIONES COLOMBIA – VERSIÓN 1.0

Este proyecto busca establecer una página con un visualizador que muestre información sobre los dividendos que se pagan en Colombia y otros pocos países que están presentes en la base de datos. La información es obtenida del archivo con las fechas Ex-dividendo que se encuentra en la página de la Bolsa de Valores de Colombia (BVC).

Todos los recursos se encuentran en el github del laboratorio (Fintrade2020 - <https://github.com/Fintrade2020?tab=repositories>), en el repositorio “Visualizador-Acciones-1.0 “. Acá encontrará los archivos de R empleados, y una carpeta con código de Python para descargar información de la Web y otros códigos para crear dashboards, y la carpeta R-Studio en donde encontrará todo el código con el cual se desarrolló el dashboard.

Archivos Excel.

Son los archivos con extensión .xlsx y .csv, estos son descargados y modificados haciendo uso de códigos de python y de R, los cuales serán explicados más adelante.

- **2022-06-10:** Archivo con las fechas Ex-dividendo de los años 2020 a 2022. Cada año se encuentra en una hoja diferente. Fue descargado el 10 de junio de 2022.
- **2020:** Contiene solo la información de las fechas Ex-dividendo del año 2020. La información fue extraída del archivo **2022-06-10**.
- **2021:** Contiene solo la información de las fechas Ex-dividendo del año 2021. La información fue extraída del archivo **2022-06-10**.
- **2022:** Contiene solo la información de las fechas Ex-dividendo del año 2022. La información fue extraída del archivo **2022-06-10**.
- **Compilado:** Contiene la información de los años 2021 a 2022 en la misma hoja. Con la información de este archivo se está trabajando, sin embargo, los datos para el dashboard se están tomando de otro sitio.
- **Sectores:** Contiene información de los sectores a los que pertenece cada emisor disponible en la base de datos.

Pasos a seguir:

Antes de pasar a explicar los códigos se hará un paso a paso de lo que se hizo, y de cómo se debería ejecutar un código ante una actualización de la base de datos con las fechas Ex-dividendo.

- **Paso 1:** Abrir el repositorio de la página
https://github.com/Fintrade2020/Dashboard_Colcap
- **Paso 2:** Ingresar a la carpeta Python y abrir el archivo “Websc”, archivo con el cual se descargará y limpiará la base de datos.
- **Paso 3:** Descargue o copie y pegue el código en una IDE para trabajar Python.
- **Paso 4:** Ingresar a la página de la BVC, busque la sección de las fechas Exdividendo y presione el ícono de Excel para descargar el archivo. (Ruta: BVC versión antigua, Mercado local, Informes bursátiles, buscar Fechas Exdividendo.)
- **Paso 5:** Visualizar el archivo, mire detalladamente columnas. y observaciones.
- **Paso 6:** Ingrese al archivo donde tiene el código “Websc” y cambie la ruta de destino que se encuentra en la línea 12. Aquí deberá colocar la ruta o la dirección donde guardará el archivo de Excel por medio de python. (Para esto puede crear una carpeta en su computador)
- **Paso 7:** Ingrese al archivo donde tiene el código “Websc” y ejecute los códigos de la línea 1 a la línea 20.

```

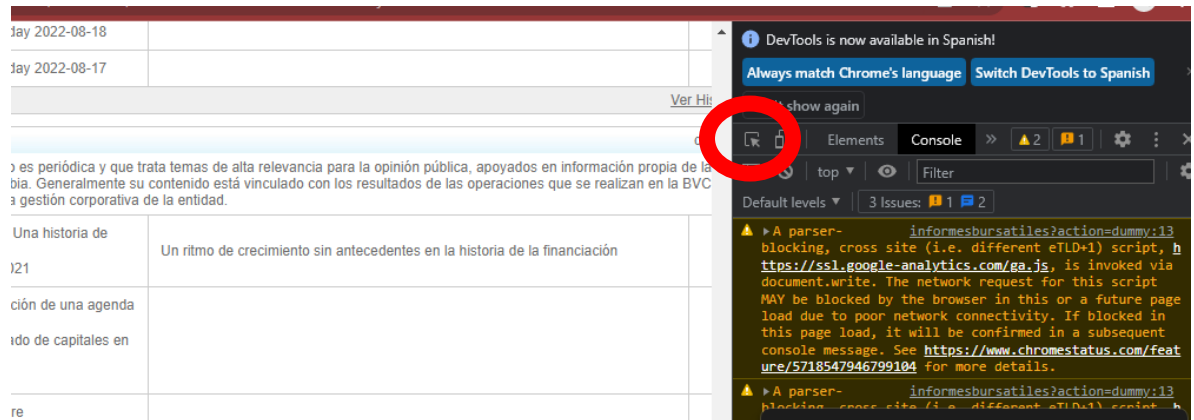
Websc.py > ...
1  import numpy as np
2  import pandas as pd
3  import requests
4  from datetime import date
5  import os
6  import openpyxl
7
8
9  url = "https://www.bvc.com.co/pps/tibco/portalbvc/Home/historicoInfBursatiles?com.tibco.ps.pagesvc.renderParams.sub76930762_11df8ad6f89_-77357f"
10
11 archivo = requests.get(url,verify=False) #requests.get permite extraer informacion sobre la pagina, ya sea texto, imagenes o archivos.
12 ubicacion = "C:/Users/Laura/Desktop/AccionesCol" #Especifica carpeta o ruta para guardar
13
14 fecha = str(date.today()) #Se guarda un objeto con la fecha de hoy
15 nombre = fecha+".xlsx" #Se guarda un objeto con la fecha y la extecion xlsx
16
17 ruta = os.path.join(ubicacion,nombre) #Permite unir las rutas, agrega el nombre del archivo a la ruta de ubicacion
18
19 with open (ruta, "wb") as output: #permite guardar el archivo en la ubicacion especificada, "write bytes"
20     output.write(archivo.content) #Lo que se va a guardar,para este caso el contenido de la url
21
22
23
24 ##### 2022 #####
25

```

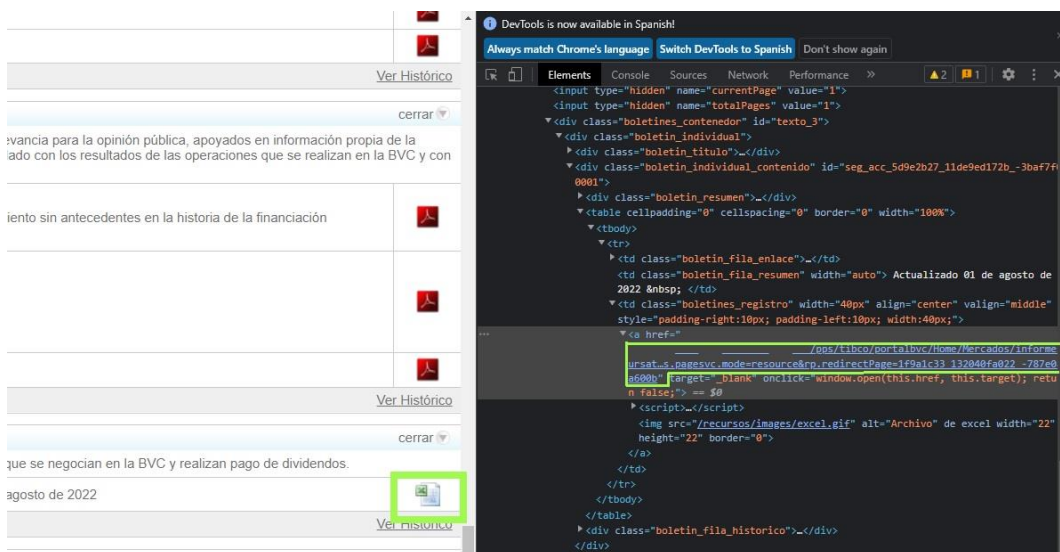
- **Paso 8:** Busque el archivo que guardó en la ruta especificada y ábralo. Tenga en cuenta que el nombre del archivo tendrá la fecha del día en el cual usted ejecute el código. (Esto se especifica en las líneas 14 y 15).
- **Paso 9:** Verifique que se encuentre la misma la información en el archivo descargado de la BVC y el archivo guardado con Python.

EN CASO DE QUE LA INFORMACIÓN DE LOS ARCHIVOS SEA DIFERENTE Y NECESITE ACTUALIZAR LOS DATOS HAGA LOS PASOS DEL 10 AL 14

- **Paso 10:** Vuelva a la página de la BVC donde está el archivo de las fechas Exdividendo, oprima F12 y saldrá un menú al costado derecho



- **Paso 11:** Oprima el botón que se encuentra en el círculo rojo (ver imagen anterior)
- **Paso 12:** Luego busque y haga clic en el ícono de Excel para descargar las fechas Exdividendo, en el menú del lado derecho deberá buscar un enlace para la descarga del archivo. Observe la siguiente imagen



- **Paso 13:** Copie el enlace para la descarga del archivo y péguelo en la línea 9 del archivo "Websc". De esta manera se obtendrá el archivo actualizado.
- **Paso 14:** Vuelva a ejecutar el código de la línea 1 a la 20 para guardar el archivo nuevo.
- **Paso 15:** Una vez descargado el archivo actualizado por medio de Python se procede a limpiarlo y organizarlo de tal manera que se facilite la manipulación e interpretación de los datos. Esto se hace con el código de las filas 28 a la 54.

Tenga en cuenta que debe cambiar todas las rutas de los archivos. En la fila 28 debe ir la dirección de donde se encuentra ubicado el archivo a modificar, y en la fila 53 debe especificar la dirección o ruta de donde quiere guardar el archivo una vez este sea modificado.

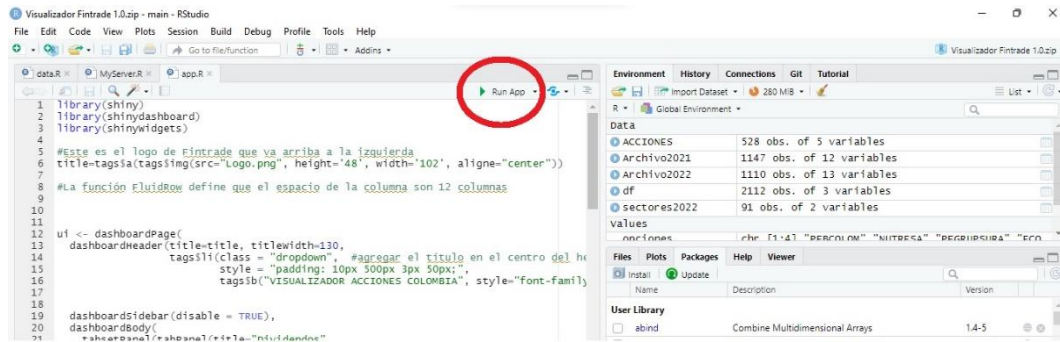
De las filas 65 a 91 se hace exactamente lo mismo pero seleccionado la segunda hoja del archivo, y de la 100 a la 126 con la tercera hoja del archivo.

- **Paso 16:** Cuando tenga completamente lista la base de datos, deberá subir la información a una hoja de Excel en Google Drive.
- **Paso 17:** En la primera hoja del archivo de google drive creado combine la información de todos los años (2020,2021,2022).
- **Paso 18:** Posteriormente debe crear un enlace de descarga de tipo CSV para ese archivo. Para esto vaya a archivo y seleccione la opción publicar en la web. Este enlace será utilizado en el código de R como base de datos para la aplicación creada.
- **Paso 19:** Vuelva al repositorio del visualizador, abra la carpeta R-Rstudio y luego seleccione el archivo data.R. Descargue el archivo o copie y pegue el código en Rstudio.
- **Paso 20:** En la fila 4, reemplace el link que aparece por el enlace de descarga que usted creó en google drive, de tal manera que se tome la base de datos nueva.

Nota: La base de datos que creó no cuenta con la variable “sectores”, pues en el archivo original de la bvc no se encuentra esta información. Por esto, en la línea 12 del código se encuentra un link de descarga con esta variable.

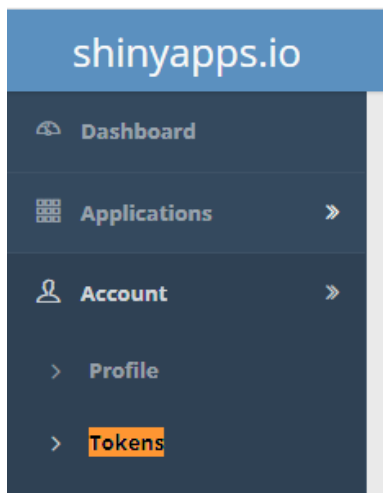
Descargue el archivo y confirme que se encuentra la información de todos los emisores que aparecen en su nueva base de datos. De lo contrario agregue la información faltante en un nuevo archivo y reemplace el enlace de descarga.

- **Paso 21:** Una vez los enlaces de descarga estén actualizados, vuelva al repositorio y descargue o copie y pegue en scripts diferentes en Rstudio los archivos “app.R” y “MyServer.R”
- **Paso 22:** Luego de tener los tres archivos en R, puede ejecutar el código de la página haciendo clic en el botón “Run App” desde el archivo “app.R”. Observe la siguiente imagen

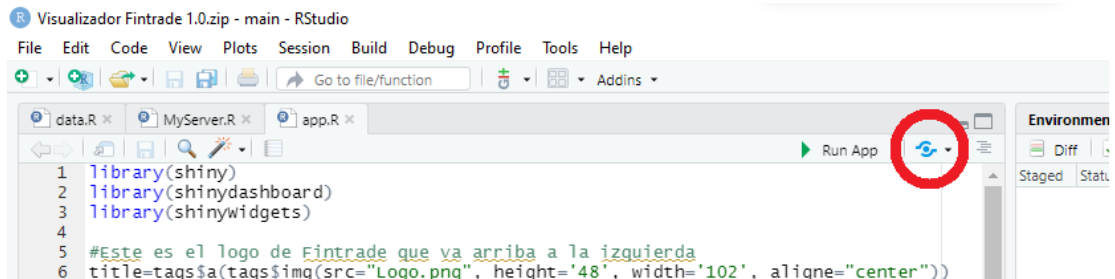


- **Paso 23:** Luego de dar clic en el botón “Run App” el código será ejecutado y en unos segundos aparecerá en una ventana emergente la página creada.
- **Paso 24:** Si considera que todos los ajustes fueron realizados, y necesita subir la nueva versión de la página a la web debe iniciar la sesión de ShinyApps de Fintrade en su R.

Para esto, inicie abra la cuenta en <https://www.shinyapps.io/>, vaya a tokens como se muestra en la siguiente imagen, copie el token y ejecútelo en R.



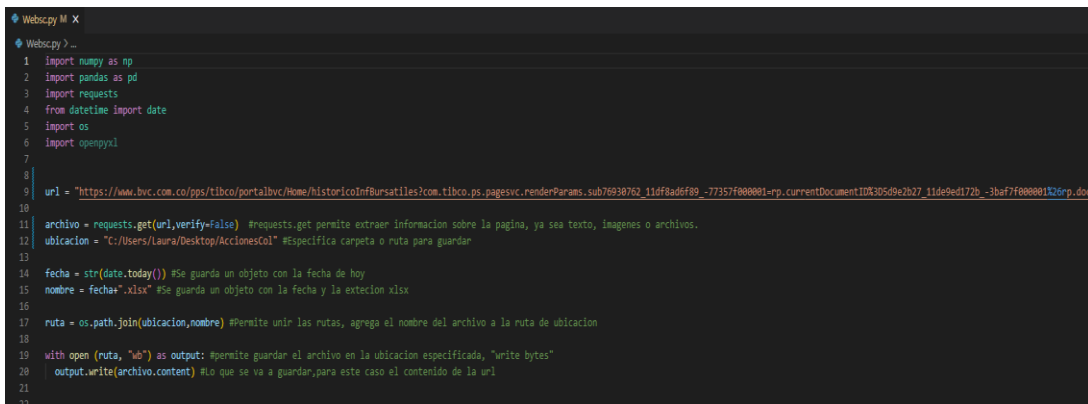
Una vez la cuenta esté vinculada, haga clic en el botón publicar que aparece en la esquina superior izquierda, como está señalado en la siguiente imagen:



PYTHON.

Los archivos con extensión .py, son aquellos que tienen código o programación en lenguaje Python, estos materiales se encuentran en la carpeta “Python”.

- Websc: Con este código se descarga el archivo de las fechas Ex-dividendo, se organiza la información y se crean los archivos por años. En este archivo, luego de cada código aparece una breve descripción de lo que hace esa línea de código, sin embargo, acá se explicarán algunos aspectos de mayor complejidad que debe tener en cuenta.



```
1 import numpy as np
2 import pandas as pd
3 import requests
4 from datetime import date
5 import os
6 import openpyxl
7
8
9 url = "https://www.bvc.com.co/pps/tibco/portalbvc/Home/HistoryCoinBursatiles?com.tibco.ps.pagesvc.renderParams.sub76938762_11dffa0df89_-77357f080001+rp.currentDocumentIDK305d9e2b27_11de9ed172b_-3ba7f080001%26rp.doc"
10
11 archivo = requests.get(url,verify=False) #requests.get permite extraer informacion sobre la pagina, ya sea texto, imagenes o archivos.
12 ubicacion = "C:/Users/Laura/Desktop/AccionesCol" #Especifica carpeta o ruta para guardar
13
14 fecha = str(date.today()) #Se guarda un objeto con la fecha de hoy
15 nombre = fecha+".xlsx" #Se guarda un objeto con la fecha y la extencion xlsx
16
17 ruta = os.path.join(ubicacion,nombre) #Permite unir las rutas, agrega el nombre del archivo a la ruta de ubicacion
18
19 with open (ruta, "wb") as output: #permite guardar el archivo en la ubicacion especificada, "write bytes"
20     output.write(archivo.content) #Lo que se va a guardar,para este caso el contenido de la url
21
22
```

Al igual que en R, primero se deben llamar las librerías que se emplearán, esto se hace desde la línea de código 1 hasta la 6. En Python, siempre que usted vaya a emplear una de estas librerías debe llamarla primero antes de la función que va a hacer.

En la línea 9 se guarda un objeto con el nombre “url”, el cual contiene el enlace de descarga del archivo con la información de los dividendos.

La línea 11 se crea un objeto con el nombre de “archivo”, el cual extrae la información del enlace ubicado en el objeto “url” haciendo uso de la función get, que hace parte de la librería requests, empleada principalmente para hacer Webscraping o bajar información de internet.

La línea 19, se indica la ruta con en la cual va a quedar guardado el archivo, luego aparece “wb”, esto hace referencia Write Bytes. En la fila 20 se indica el archivo o enlace con la información a guardar, seguido por un punto y luego la opción que se quiere guardar,

esto depende de si el contenido es un archivo, es un texto, o imagen, para este archivo que se va a descargar se deja “.content”.

Luego de esta parte, se encuentra el código para solucionar algunos aspectos que pueden generar problemas más adelante, como lo son los valores faltantes de las celdas combinadas, algunos caracteres especiales, eliminación de las primeras filas, nombres de las columnas, finalmente, se separa la información de los años en archivos diferentes (Archivo2020, Archivo2021, y Archivo2022).

R.

La página <https://fintrade.shinyapps.io/VisualizadorAcciones/> está elaborada en su totalidad en R. Estas aplicaciones se pueden crear por medio de algunos paquetes como *shiny*, para hacer páginas con estructuras básicas / sencillas, y *shinydashboard*, la cual da una estructura un poco más elaborada y amigable con el usuario en comparación con shiny.

Para iniciar debe tener en cuenta que la programación de una página se divide en 2, está la interfaz o la cara que ve el usuario (Front end), esta se guarda en un objeto llamado “ui” y se encuentra en el archivo “app.R”, allí se definen la interfaz, todos los temas de organización, contenido a mostrar, colores, etc.

La otra parte es la encargada de ejecutar cálculos internos y que toda la lógica de una página funcione (Back end), esto último se guarda en un objeto llamado “server”, cuyo código se encuentra en el archivo “MyServer.R”

Es importante que estas dos partes se conecten para que la página funcione correctamente, para esto se usa la función *Source*.

```
1 library(dplyr)
2 library(shiny)
3 library(readxl)
4 library(ggplot2)
5 library(plotly)
6 library(ggiraph)
7
8
9 server <- function(input, output, session) {
10   source("data.R")
11
12   ##### PRIMER MENU #####
13 }
```

El código de la página debe terminar especificando donde se encuentran estos dos componentes, por defecto, se dejan los nombres predeterminados, por lo cual la última fila

de código en el archivo “app.R ”debería terminar en source(“MyServer.R”) y shinyApp(ui , server)

```
61     ),
62     skin = "yellow" #Tema de la pagina amarillo.
63   )
64 )
65
66 source("MyServer.R")
67
68 shinyApp(ui, server)
```

CÓDIGOS PARA EL DESARROLLO DE LA PÁGINA CON R

- data.R

En primer lugar, se deben llamar los paquetes que se van a usar para desarrollar la aplicación, luego se carga la base de datos. Como se mencionó anteriormente, los datos son llamados desde un enlace de Google drive, esto debido a que cuando uno carga el archivo de Excel desde el computador inicialmente la aplicación corre bien, pero cuando esta se va a publicar se originan algunos problemas, la manera de corregirlo es tomar toda la información desde la Web.

Posteriormente se ajustan los nombres de algunas variables, junto con el tipo de dato que se necesita, de las filas 15 a la 17, se indica que se va a trabajar con fechas, en la fila 18 se establece esa variable contiene datos de tipo numérico

```
1 library(dplyr)
2 library(readxl)
3
4 Archivo2021<-read.csv("https://docs.google.com/spreadsheets/d/e/2PACX-1vRE4a_Z6cvVLrc46w4L26FhF_zEp3crU8NCEnoKx_dqIH5FqPthQd4Z
5
6 names(Archivo2021)[7] <- "FECHA.FINAL"
7 names(Archivo2021)[8] <- "TOTAL"
8 names(Archivo2021)[9] <- "CUOTA"
9 names(Archivo2021)[11] <- "TOTAL.ENTREGADO.EN.DIVIDENDOS"
10
11 #agregar la variable sectores
12 sectores2022<-read.csv("https://docs.google.com/spreadsheets/d/e/2PACX-1vSuC1u7x6hNZt1MC1At-KuGXM3vMrRE4V5miGW6NN100PKyLgCPGKY
13 Archivo2022<-merge(x=Archivo2021, y= sectores2022, all.y = TRUE)
14
15 Archivo2022$FECHA.ASAMBLEA <- as.Date(Archivo2022$FECHA.ASAMBLEA) #Convertir tipo de dato a fecha.
16 Archivo2022$FECHA.INICIAL <- as.Date(Archivo2022$FECHA.INICIAL) #Convertir tipo de dato a fecha.
17 Archivo2022$FECHA.FINAL <- as.Date(Archivo2022$FECHA.FINAL) #Convertir tipo de dato a fecha.
18 Archivo2022$CUOTA <- as.numeric(Archivo2022$CUOTA) #Convertir tipo de dato a numérico.
```

En este punto ya se tienen cargadas las bases de datos y están organizadas, con lo cual se procederá con el código para programar la aplicación.

- **app.R**

Esta parte inicia programando el “UI”. En la línea 6 se crea un objeto con el logo del laboratorio que se encuentra en la carpeta www, para posteriormente ubicarlo en el encabezado de la página.

```
1 library(shiny)
2 library(shinydashboard)
3 library(shinyWidgets)
4
5 #Este es el logo de Fintrade que va arriba a la izquierda
6 title=tags$img(src="Logo.png", height="48", width="102", align="center")
7
```

Como se está haciendo uso de la librería “*shinydashboard*”, todos los componentes de la interfaz (Encabezado, menú, cuerpo, tema) deben ir adentro del argumento “*dashboardpage*”.

- **dashboardHeader:** En esta parte se coloca el logo que anteriormente se nombró como “*title*” y título o nombre que va a tener la página. Los comandos principales para esta sección son “*style*”, utilizado para modificar la ubicación, tipo de letra, color, tamaño y “*width*”, que sirve para determinar el ancho del espacio que va a ocupar su título.

```
11
12 ui <- dashboardPage(
13   dashboardHeader(title=title, titleWidth=130,
14     tags$li(class = "dropdown", #agregar el título en el centro del header
15       style = "padding: 10px 500px 3px 50px;",
16       tags$b("VISUALIZADOR ACCIONES COLOMBIA", style="font-family:Tahoma; color:#002956; text-align:center; font-size:20px))),
17
18
```

- **dashboardSidebar:** Por defecto, esta función de shiny ubica el menú de la página en la parte lateral izquierda. Sin embargo para optimizar el espacio de la interfaz, se decide deshabilitar esta función utilizando el comando *disable=TRUE* en la línea 19 del código.
- **dashboardBody:** Aquí se indican todos los contenidos que tendrá la página, las opciones con las que interactuará el usuario, los resultados que se van a mostrar, ya sean bases de datos, salidas de cálculos específicos, gráficos, etc. Si usted desea personalizar la página, cambiando colores, tipos de letras, entre otras propiedades de estilo, debe aprender a programar en CSS y/o HTML
- **tabsetPanel:** Para sustituir el menú lateral entonces, se utiliza el argumento *tabsetPanel*, que permite crear diferentes pestañas con lo que se quiere mostrar en la página, en este caso los gráficos de los dividendos y sus filtros.
- **tabPanel:** Este argumento permite crear cada una de las pestañas u opciones del menú, y debe ir dentro de *tabsetPanel*. Allí se especificará cada cosa que debe ir en

la página. En esta página solo hay una pestaña llamada “Dividendos” con los filtros para modificar la base de datos y los gráficos.

```
20 dashboardBody(  
21   tabsetPanel(tabPanel(title="Dividendos",  
22     fluidRow(column(2,dateRangeInput("IN_Fechas", "Seleccione el rango de fechas", #Creacion de input para la fecha de asamblea.  
23       start = "2020-01-01",end = "2022-12-31", #Se indican fechas seleccionadas por defecto.  
24       min = "2020-01-01",max = "2024-12-31", #Limites de seleccion.  
25       format = "yyyy-mm-dd", #Formato de impresion.  
26     ),  
27   ),  
28   column(2,pickerInput("IN_Sector", label = "Sector Economico", #Creacion de input para seleccionar el sector economico.  
29     choices = NULL,multiple = T, #No se establecen opciones, se pueden seleccionar multiples opciones.  
30     options = list('actions-box' = TRUE)  
31   ), #Permite seleccionar y anular la seleccion de TODO.  
32   ),  
33   column(2,pickerInput("IN_Moneda", label = "Moneda", #Creacion de input para seleccionar la moneda.  
34     choices = NULL,multiple = T,  
35     options = list('actions-box' = TRUE)  
36   ),  
37   ),  
38   column(2,pickerInput("IN_Emisor",label = "Emisor", #Creacion de input para seleccionar el Emisor.  
39     choices = NULL,multiple = T,  
40     options = list('actions-box' = TRUE)  
41   ),  
42   ),  
43   column(2,pickerInput("IN_Nemotecnico",label = "Nemotécnico", #Creacion de input para seleccionar el Nemotécnico.  
44     choices = NULL,multiple = T,  
45     options = list('actions-box' = TRUE, 'live-search'=TRUE)  
46   ),  
47   )  
48 )
```

- fluidrow: Se utiliza para ubicar elementos en una misma fila, de tal manera que todo quede organizado. El tamaño o ancho de la página es de “12”, de tal manera que usted puede establecer varias columnas dentro de la fila, especificando el tamaño de cada columna (Siempre y cuando la suma de estos no pase de 12), esto se hace con el argumento “column”. El tamaño de la fila dependerá de cuantos elementos coloque en una columna. En la imagen anterior, se utilizó una “fluidrow”, para ubicar un menú interactivo para seleccionar fechas, esto será colocado en una columna de tamaño 2, luego de eso viene el código del menú. Si no se especifican las columnas se toma todo el espacio como uno solo.
- Tabbbbox: Permite la creación de cajas o espacios para colocar los elementos, esta herramienta resulta útil debido a que puede tener varias pestañas, logrando tener de forma organizada algunos contenidos. En la siguiente imagen se encuentra un ejemplo de una tabBox que tendrá un ancho de 12. Para colocar varias pestañas o subsecciones dentro de esa caja, deberá utilizar el comando “tabPanel”, indicando el nombre que recibirá la pestaña y el contenido de la misma. Para este caso, hay dos pestañas, una que se llama “Dividendos por nemotécnico”, y otra que tiene el nombre de “Dividendos por Moneda”, las dos tendrán como contenido un gráfico. Esto se explicará más adelante

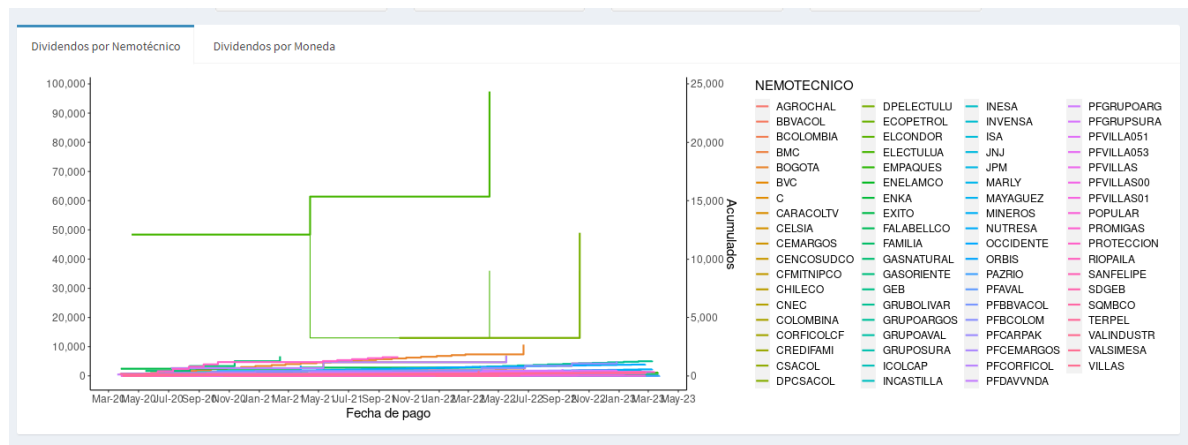
```

fluidRow(
  tabBox(width = 12, #Se crea una caja de tamaño 12 con para los gráficos.

    tabPanel("Dividendos por Nemotécnico", plotOutput("Grafico_D")), #Se crea un espacio o pestaña llamada "Dividendos", en donde se muestra el
    tabPanel("Dividendos por Moneda", plotOutput("Grafico_A0")) #Se crea un espacio o pestaña llamada "Dividendos acumulados", en donde se muestr
  )

```

Como resultado se obtendrá lo siguiente, donde en la parte superior izquierda se encuentran las pestañas, y lo de abajo es el contenido



- **dateRangeInput:** Con este comando se agrega el input o herramienta para seleccionar un rango de fechas.

```

dateRangeInput("IN_Fechas", "Seleccione el rango de fechas", #Creacion de input para la fecha de asamblea.
  start = "2020-01-01", end = "2022-12-31", #Se indican fechas seleccionadas por defecto.
  min = "2020-01-01", max = "2024-12-31", #Limites de seleccion.
  format = "yyyy-mm-dd", #Formato de impresion.

```

Al igual que todo lo anterior, lo primero que se coloca es el ID, para este caso es "IN_Fechas" (En esta página todos los inputs empiezan por IN_, con el objetivo de identificarlos fácilmente. Luego se coloca el encabezado o texto que se mostrará en el interfaz. El argumento "start" sirve para establecer una de inicio fecha por defecto, que sería "2020-01-01", "end" establece la fecha final por defecto. "min" es la fecha más antigua que los usuarios pueden seleccionar, para este caso, no se puede seleccionar nada que esté antes de la fecha "2020-01-01", "max" sirve para especificar la fecha máxima que los usuarios pueden seleccionar, en este fragmento se indica que no se podrá seleccionar nada que esté después del "2024-12-31". "Format" se utiliza para especificar el formato con el cual aparecerá la fecha, esto es totalmente personalizable, puede poner fecha larga, fecha corta, puede variar con el orden de los meses, días años, etc. Por último, el argumento width indica el ancho que va a tener este input.

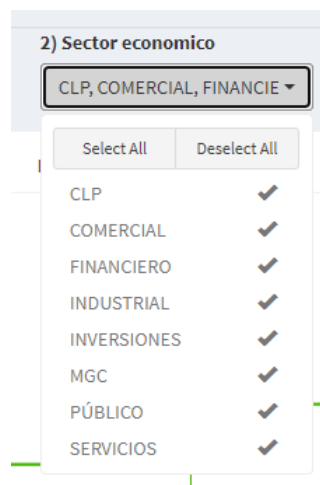
Resultado: Muestra la etiqueta especificada y los rangos que se pusieron por defecto con los argumentos “start” y “end”.



- pickerInput: PicerInput sirve para ubicar una lista desplegable en el cuerpo de la página, esta también cuenta con ID, con etiqueta o texto que aparecerá en el output, note que este texto aparece diferente en el código y en resultado, esto es debido a que este input se actualiza dependiendo de la selección del usuario al anterior Input, esto será explicado más adelante. En “choices”, se indican las opciones que el usuario podrá seleccionar, “multiple” permite determinar si el usuario puede seleccionar más de una sola respuesta, si coloca F o FALSE, la selección está limitada a una sola respuesta, T o TRUE, permiten la selección de varias opciones. “options” es un argumento especial de este input, debido a que este agrega botones que permiten seleccionar o quitar la selección de todas las opciones, esto se hace colocando “list(`actions-box` = TRUE)”, dentro de este argumento.

```
pickerInput("IN_Sector", label = "Sector Economico", #Creacion de input para seleccionar el sector economico.  
            choices = NULL, multiple = T, #No se establecen opciones, se pueden seleccionar multiples opciones.  
            options = list(`actions-box` = TRUE))
```

Como resultado se obtiene una lista con las opciones establecidas para facilitar su selección



- Personalización HTML: Como ya se indicó, la gran parte que se desarrolla desde Shiny se puede personalizar, para esto se usa el

lenguaje HTML. Para conocer todas estas herramientas puede acceder al siguiente enlace. <https://shiny.rstudio.com/articles/tag-glossary.html>.

Esos fueron casi todos los elementos empleados en el desarrollo de la interfaz de la primera pestaña gráfica

Output:

Cuando usted coloque un resultado de R en su página debe indicar que tipo de resultado es, si es gráfico debe usar "PlotOutput", si es un dataframe o una tabla de datos puede utilizar "DataTableOutput", si usted desea mostrar un resultado de una operación, modelo que sea texto utilice "verbatimTextOutput". Hay muchos más tipos de outputs, pero estos son lo más básicos y los que se usaron en la construcción de la página:

- Plououtput: Gráficos
- DataTableOutput: Dataframes
- Estos Outputs, deben ir relacionados con el Render, que se coloca en el server. Nuevamente, se explicará más adelante.

MyServer.R

Si se da cuenta, en la interfaz no se hace ningún calculo o acción relacionada con las bases de datos, debido a que todo eso se realiza en el server. El server siempre debe iniciar con el siguiente comando, la llave del final queda abierta debido a que todo lo que se coloca en esta parte debe quedar adentro de esa llave.

```
8
9  server <- function(input, output, session) {
10    source("data.R")
11  }
```

- Funciones reactivas (reactive): Las funciones reactivas se usan para ejecutar acciones repetitivas o procesos que se van a emplear en varias partes del código, se usan para hacer un código más limpio y eficiente. El resultado del objeto que se guarda acá solo se ejecuta una vez y se guarda el resultado, tan pronto el programa percibe un cambio actualiza de manera automáticamente el proceso. En la siguiente imagen se presenta una función reactiva, en esta parte se filtra el dataframe dependiendo de los filtros que la persona haya seleccionado, con el resultado de esos filtros luego se realizan algunos gráficos y cálculos adicionales. El dataframe que resulta de esto también se imprime o se muestra al final de la página.

Estas funciones inician con un nombre, puede ser cualquiera, en este caso es `datn`, luego viene `reactive({})`, es que donde se indica que se va a trabajar con una función reactiva.

Esta función luego se puede hacer uso se esta función llamándola por su nombre y con paréntesis `“datn()”`.

```
42 datn<-reactive({ #Funcion reactiva para la creacion de un df con todos los filtros aplicados sobre el df original.
43   Archivo2022 %>% filter(SECTOR %in% input$IN_Sector &
44                         MONEDA %in% input$IN_Moneda &
45                         EMISOR %in% input$IN_Emisor &
46                         NEMOTECNICO %in% input$IN_Nemo)%>%
47   select(-FECHA.INGRESO, -TOTAL.ENTREGADO.EN.DIVIDENDOS, #Eliminación de columnas.
48         -DESCRIPCION.PAGO.PDU, -MODULO.DE.PAGO)
49 }
```

Actualización del menú

- **Observe:** En shiny, además de la función reactiva (`reactive`), también hay algunos otros comandos sirven para ejecutar cambios y actualizaciones de datos según los criterios seleccionados en los inputs y elementos de la aplicación, estos son `“observe”`, `“observeEvent”`, `“eventReactive”`, `“reactiveValue”`, cada uno tiene una función específica, es importante que los investigue y entienda la diferencia de cada uno de estos. Puede que lo vaya a necesitar si necesitan hacer más modificaciones sobre filtros. En el desarrollo de esta página solo se usó la función reactiva para guardar objetos, y `“observe”` para actualizar las opciones y valores de los inputs según las opciones seleccionadas del usuario que haga uso de la página.

En primer lugar se abre el objeto `“observe({})”`, adentro se crea un nuevo dataframe, en donde se filtran los datos según la selección de los inputs, en la siguiente imagen, se muestra que se crea el objeto `dat0`, que realiza un filtro en la variable `“Fecha.Asamblea”` sobre el archivo base (`Archivo2022`).

Para indicar que se debe hacer uso de alguna de las selecciones de los filtros se debe poner `“input$”` y luego del signo pesos el ID del input, en este caso, se quiere obtener el dato del Input llamado `“IN_Fechas”`. Como este filtro tiene dos datos (Fecha inicial y fecha final), se debe indicar cual dato se va a usar, por eso en primer lugar está `“input$IN_Fechas[1]”` y luego `“input$IN_Fechas[2]”`. De esta manera se crea un dataframe llamado `dat0`, que contiene solo la información filtrada y que se actualiza cada vez que se hace un cambio sobre el input que se especificó en el código.

- **updatePickerInput:** Dentro de la misma función de `observe`, luego de crear un nuevo dataframe con la información seleccionada, se usa `“updatePickerInput”`, para actualizar uno de los filtros o inputs. En esta función siempre se debe colocar `“session”` de primero, luego se selecciona el input que se va a actualizar, como luego

del input de la fecha viene el input del sector, se actualizará el filtro del sector, en la etiqueta o “label”, deberá colocar el texto que desea que aparezca arriba del input, en “choices” debe especificar las opciones que el usuario podrá usar. Note que el código de este argumento es “sort(unique(dat0\$SECTOR))”. Vamos a explicar esta parte desde adentro.

- **dat0\$SECTOR:** Se selecciona solo los datos de la columna SECTOR del dataframe dat0 (Recuerde que dat0 fue el dataframe creado en el paso anterior),
- **unique:** Sirve para identificar los valores únicos que se encuentran en una columna.
- **Sort:** ordena los valores o datos que se seleccionaron por orden alfabético.

Entonces, lo que se hace con “sort(unique(dat0\$SECTOR))”, es extraer los valores únicos de la columna sector perteneciente al dataframe dat0, luego se ordenan esos valores, de tal manera que en la lista desplegable salgan ordenados y sean más fáciles de ubicar.

Por último, se tiene el argumento “selected”, con el cual usted indica cuales valores se seleccionan por defecto antes de que el usuario haga su propia selección. En este caso, se indica que se deben seleccionar todos los datos que hay en “dat0\$SECTOR”.

```
14   observe({
15     dat0<-filter(Archivo2022,FECHA.ASAMBLEA >= input$IN_Fechas[1] & #Crea dat0 con filtro de la seleccion en el input
16                  FECHA.ASAMBLEA <= input$IN_Fechas[2])
17     updatePickerInput(session, "IN_Sector", label = "2) Sector economico", #Se actualiza el input del sector IN_Sect
18                       choices = sort(unique(dat0$SECTOR)),selected = unique(dat0$SECTOR)) #con los valores únicos de
19   })
20
```

Otra manera de crear un objeto para la actualización de un input sin la necesidad de usar dplyr, es con Rbase, los filtros diferentes a fechas, que solo tienen un dato fueron programados de esta manera.

Lo primero que se hace es especificar la base de datos y la variable, en este caso “Archivo2022\$MONEDA”, se abre un corchete en donde se especifica el filtro que se quiere aplicar sobre la base de datos. Como se va a actualizar el filtro de la variable MONEDA con la selección del filtro SECTOR, entonces se pone “Archivo2022\$SECTOR %in% IN_Sector”. Donde se indica que se debe filtrar la columna SECTOR con la información seleccionada en el input.

El update del input funciona de la misma manera, solo que para este caso no se está creando un dataframe con múltiples variables, simplemente se crea una lista de la columna seleccionada con los filtros aplicados. Por lo que al momento de poner las opciones y los elementos seleccionados solo se pone el objeto creado, sin necesidad de especificar base y columna como se hizo en el ejemplo anterior.

```

22   observe({
23     dat00<-Archivo2022$MONEDA[Archivo2022$SECTOR%in%input$IN_Sector] #Crea dat00, un
24     updatePickerInput(session, "IN_Moneda", label = "3) Moneda", #Se actualiza el i
25                           choices = sort(unique(dat00)),selected = unique(dat00)) #con l
26   })
27

```

Render

Cuando en el server se crea una salida, se debe especificar que tipo de output es, ya sea el resultado de una operación matemática, la creación de una lista, dataframe, grafico normal o interactivo. Esto se hace al momento de CREAR EL OUTPUT. Lo primero que se debe hacer es escribir “output\$” y luego del signo pesos debe escribir el nombre que tendrá ese output, el cual es el mismo que se coloca en el UI. Luego de esto se pone el símbolo de asignación “<-” y se indica el render con el tipo de salida que es. Observe los comandos que se usaron en la creación de la aplicación. Todo render abre con ({}), dentro de los corchetes debe ir toda la información.

- **renderDataTable:** Se usa para imprimir dataframes, se asignó como nombre de la salida “Base” y luego se indico el tipo de salida que es “renderDataTable({})”. En este caso lo que se hizo fue crear el objeto o dataframe Base_M1 con la información de la función reactiva datn(). Luego se indica que se debe imprimir Base_M1. Cuando se usa este comando se debe relacionar el nombre de la salida con el comando dataTableOutput, vuelva a revisar la imagen de los outputs en el UI.

```

52   output$Base <- renderDataTable({ #SE CREA EL OUTPUT BASE, QUE SEGUN EL RE
53     #Contenido de Base.
54     Base_M1 <- datn() #Se guarda el resultado de datn en el objeto Base_M1
55     Base_M1 #Imprime Base_M1.
56   })

```

- **renderPlot:** Si usted desea imprimir un gráfico, ya sea con R base o ggplot2 puede usar el renderPlot. En la siguiente imagen se indica que es un output de nombre “Grafico_D”, luego se indica el tipo de output “renderPlot({})”. Dentro de esta función va el código del gráfico que usted quiere desarrollar. En este caso se realizó un ggplot de tipo “step” para graficar los dividendos recibidos en cada fecha. Cuando se usa este comando se debe relacionar el nombre de la salida con el comando plotOutput, vuelva a revisar la imagen de los outputs en el UI


```

79   output$Grafico_AD <- renderPlot({ #Se crea Grafico_AD, el cual contiene un grafico de ggplot2.
80     Dividendos <- Base_AD() #Se establece el df Dividendos con la información de la función reactiva Base_AD()
81     ggplot()+ #Selección de base de datos, columnas de los ejes x, y, color dependiendo el tipo de moneda.
82
83     geom_step_interactive(Dividendos, mapping= aes(x=FECHA.INICIAL,y=Total, color = MONEDA))+
84     geom_step_interactive(Dividendos, mapping= aes(x=FECHA.INICIAL,y=T_div, color = MONEDA),size=1)+ #Tipo de grafico
85

```

- **renderPrint:** Por último, está la salida para imprimir resultados de operaciones matemáticas o modelos. Para este ejemplo se creó el output llamado “base1” y se indicó que es un `renderPrint({})`. Lo que se hace acá es definir el objeto “Reac_b” con la información de la función reactiva “Base_R”. Luego se imprime el resultado de sumar todos los valores de la columna CUOTA. Esta salida debe ir relacionada en el UI con `verbatimTextOutput`.

```

157   output$base2 <- renderPrint({ #Creación salida u output base2.
158     Reac_b <- Retorno_Precio() #Establece df Reac_b con la información
159     tail(Reac_b$Precio,1) - head(Reac_b$Precio,1) #Resta el valor de 1
160   })

```