DOCUMENTACIÓN SIMULADOR DIVIDENDOS SHINY.

Contenido

La idea de este proyecto es dejar una página con un simulador 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 "Dashboard_Colcap" https://github.com/Fintrade2020/Dashboard_Colcap. Acá encontrará tres carpetas, una con los archivos empleados, otra con código de Python para descargar información de la Web y otros códigos para crear dashboards, y la carpeta R-Rstudio 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.

- **2021-11-23:** Archivo con las fechas Ex-dividendo de los años 2021 y 2020. Cada año se encuentra en una hoja diferente. Fue descargado el 23 de noviembre de 2021.
- **Archivo2020:** Contiene solo la información de las fechas Ex-dividendo del año 2020. La información fue extraída del archivo **2021-11-23.**
- **Archivo2021:** Contiene solo la información de las fechas Ex-dividendo del año 2021. La información fue extraída del archivo **2021-11-23.**
- **Completo:** Contiene la información de los años 2021 y 2020 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.
- **ACCIONES:** Acá encontrará histórico de precios de 4 acciones.

Pasos a seguir.

Antes de pasar a explicar lo códigos ser 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.

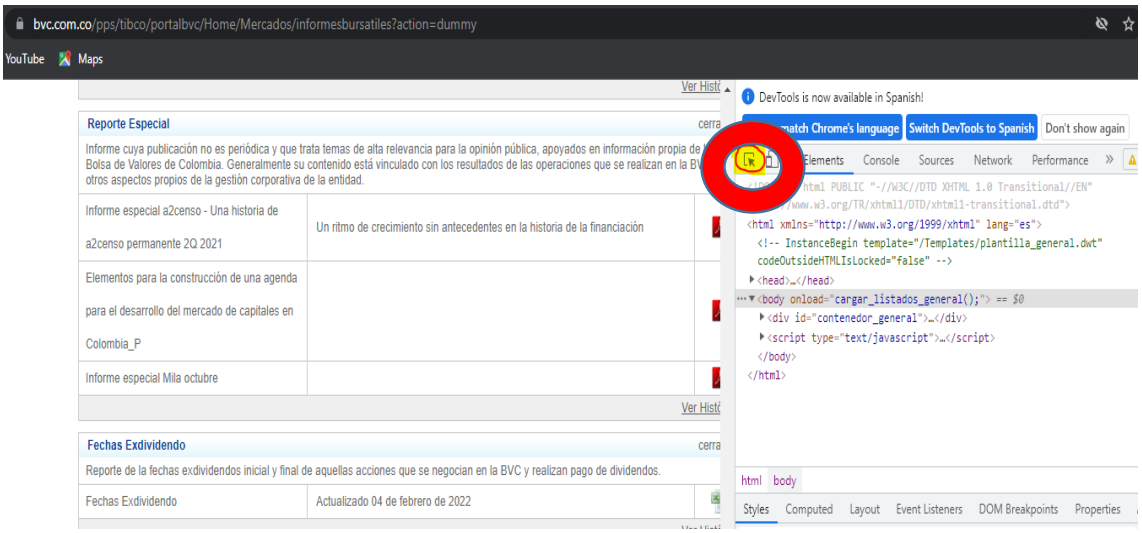
- **Paso 1:** Abrir el repositorio de la página. (https://github.com/Fintrade2020/Dashboard_Colcap)
- **Paso 2:** Ingresar a la carpeta Python y abrir el archivo "Websc", archivo con el cual se descargará y limpiará la base de datos.
- **Paso 3:** Descargue o copie y pegue el código en una IDE para trabajar Python.
- **Paso 3:** 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 4:** 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. Acá deberá colocar la ruta o la dirección donde guardará el archivo de Excel por medio de python.
- **Paso 5:** Ingrese al archivo donde tiene el código "Websc" y ejecute los códigos de la línea 1 a la línea 20.

```python
import numpy as np
import pandas as pd
import requests
from datetime import date
import os
import openpyxl


url = "https://www.bvc.com.co/pps/tibco/portalbvc/Home/Mercados/informesbursatiles?com.tibco.ps.pagesvc.action=updateRenderState&rp.currentDocumentID=-566aa413_16cde9c48ee_293

archivo = requests.get(url)     #requests.get permite extraer informacion sobre la pagina, ya sea texto, imagenes o archivos.
ubicacion = "C:/Users/JosePinzon/Documents/FINTRADE/Python" #Especifica carpeta o ruta para guardar

fecha = str(date.today()) #Se guarda un objeto con la fecha de hoy
nombre = fecha+".xlsx" #Se guarda un objeto con la fecha y la extecion xlsx

ruta = os.path.join(ubicacion,nombre) #Permite unir las rutas, agrega el nombre del archivo a la ruta de ubicacion

with open (ruta, "wb") as output: #permite guardar el archivo en la ubicacion especificada, "write bytes"
  output.write(archivo.content) #Lo que se va a guardar,para este caso el contenido de la url


```

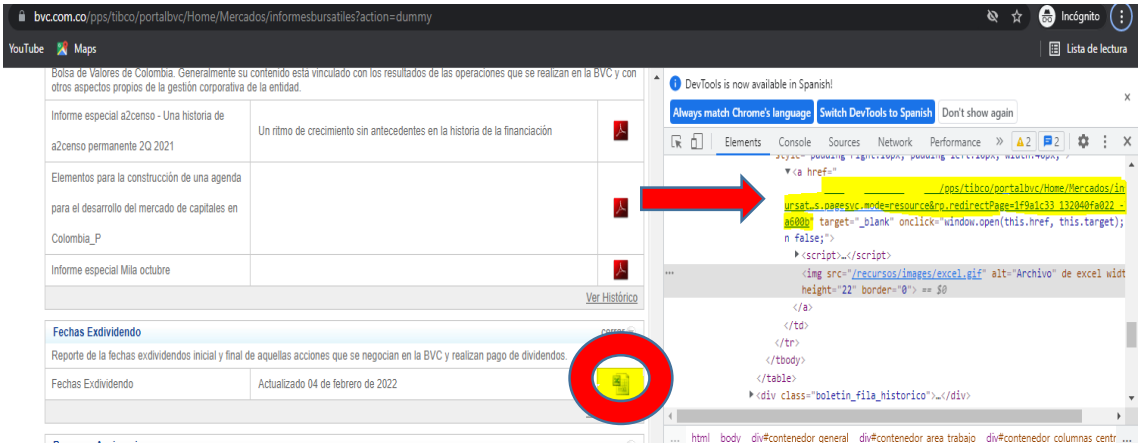
- **Paso 6:** Busque el archivo que guardó en la ruta especificada y ábralo. Tenga en cuenta que el nombre del archivo tendrá la fecha del día en el cual usted ejecute el código. (Esto se especifica en las líneas 14 y 15).
- **Paso 7:** Verifique que se encuentre la misma la información en el archivo descargado de la BVC y el archivo guardado con Python.

  EN CASO DE QUE LA INFORMACIÓN DE LOS ARCHIVOS SEA DIFERENTE HAGA LOS PASOS DEL 8 AL 11.

- **Paso 8:** Vuelva a la página de la BVC donde está el archivo de las fechas Ex-dividendo, oprima F12 y saldrá un menú al costado derecho.



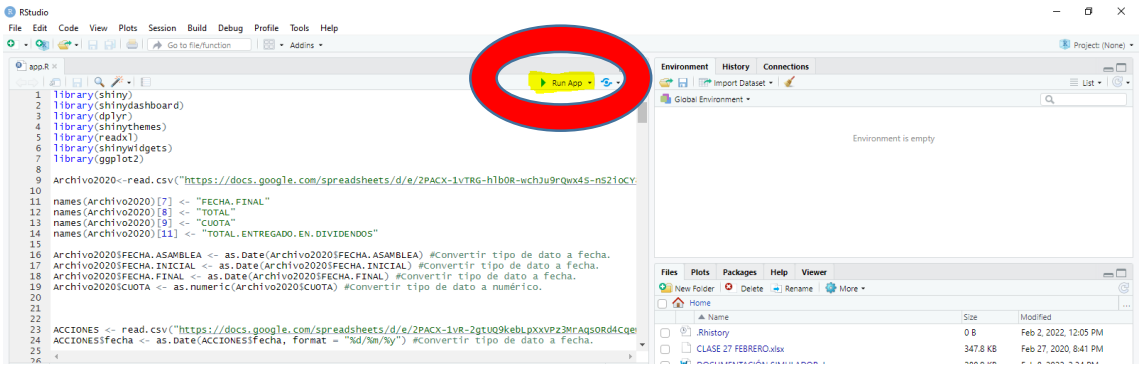
- **Paso 9:** Oprima el botón que se encuentra resaltado en amarillo y encerrado en círculos rojos. Luego de eso busque el ícono de Excel para descargar el archivo de las fechas Exdividendo, en el menú del lado derecho deberá buscar un enlace para la descarga del archivo. Mire la siguiente imagen.



- **Paso 10:** Copie el enlace para la descarga del archivo y péguelo en la línea 9 en el código del archivo “Websc”. De esta manera se reemplaza el enlace de descarga y se obtiene el archivo actualizado.
- **Paso 11:** Vuelva a ejecutar el código de la línea 1 a la 20 para volver a guardar el archivo nuevo.
- **Paso 12:** 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.
- **Paso 13:** Cuando tenga completamente lista la base de datos, deberá subir la información a una hoja de Excel en Google Drive.
- **Paso 14:** En la primera hoja del archivo de google drive creado combine la información de todos los años (2020,2021,2022).
- **Paso 15:** Posteriormente debe crear un enlace de descarga de tipo CSV para ese archivo. Este enlace será utilizado en el código de R como base de datos para la aplicación creada.
- **Paso 16:** Vuela al repositorio del simulador, abra la carpeta R-Rstudio y luego seleccione el archivo app.R. Descargue el archivo o copie y pegue el código en Rstudio.
- **Paso 17:** En la fila 9, 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.
- **Paso 18:** Abra el enlace de la línea 23, luego de esto se descargará un nuevo archivo. Cree otro archivo de Excel en google drive y suba la información que se encuentra en el archivo descargado en este paso.
- **Paso 19:** Genere otro enlace de descarga de tipo CSV para este último archivo. Luego copie y pegue ese enlace en la fila 23. (Borre el anterior y ponga ahí el enlace nuevo).
- **Paso 20:** Una vez actualizados los enlaces de descarga en el archivo de R, puede ejecutar el código de la página. Esto lo puede hacer dando clic en el botón de “Run App”. Observe la siguiente imagen.



- **Paso 21:** 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.

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.

98 lines (48 sloc) | 3.74 KB — Raw | Blame

```
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/Mercados/informesbursatiles?com.tibco.ps.pagesvc.action=updateRenderState&rp.currentDo
10
11 archivo = requests.get(url)     #requests.get permite extraer informacion sobre la pagina, ya sea texto, imagenes o archivos.
12 ubicacion = "C:/Users/JosePinzon/Documents/FINTRADE/Python" #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
```

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 y Archivo2021).

R.

La aplicación o página sobre los dividendos (https://jose-pinzon.shinyapps.io/Simulador/), está elaborada en su totalidad en R. Estas aplicaciones se pueden crear por medio de algunos paquetes como sh*iny*, 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", es donde se definen 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".

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 debería terminar en shinyApp(ui = ui, server = server)

CODIGO INICIAL DESARROLLO APLICACIÓN CON R.

```
1  library(shiny)
2  library(shinydashboard)
3  library(dplyr)
4  library(shinythemes)
5  library(readxl)
6  library(shinyWidgets)
7  library(ggplot2)
8
9  Archivo2020<-read.csv("https://docs.google.com/spreadsheets/d/e/2PACX-1vTRG-hlb0R-wchJu9rQwx4S-nS2ioCY8RsyPigRLLH1D1scB_
10
11 names(Archivo2020)[7] <- "FECHA.FINAL"
12 names(Archivo2020)[8] <- "TOTAL"
13 names(Archivo2020)[9] <- "CUOTA"
14 names(Archivo2020)[11] <- "TOTAL.ENTREGADO.EN.DIVIDENDOS"
15
16 Archivo2020$FECHA.ASAMBLEA <- as.Date(Archivo2020$FECHA.ASAMBLEA)
17 Archivo2020$FECHA.INICIAL <- as.Date(Archivo2020$FECHA.INICIAL)
18 Archivo2020$FECHA.FINAL <- as.Date(Archivo2020$FECHA.FINAL)
19 Archivo2020$CUOTA <- as.numeric(Archivo2020$CUOTA)
```

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 16 a la 18, se indica que se va a trabajar con fechas, en la fila 19 se establece esa variable contiene datos de tipo numérico.

```
23 ACCIONES <- read.csv("https://docs.google.com/spreadsheets/d/e/2PACX-1vR-2gtUQ9kebLpXxVPz3MrAqsORd4CqewaRxP_QU8i4oAWOipv
24 ACCIONES$fecha <- as.Date(ACCIONES$fecha, format = "%d/%m/%y")
25
26 df <- tidyr::gather(ACCIONES, key = "Accion", value = "Precio",
27                     PFBCOLOM, NUTRESA, PFGRUPSURA,ECOPETROL)
28
29 opciones <- c("PFBCOLOM", "NUTRESA", "PFGRUPSURA","ECOPETROL")
```

El archivo “Acciones”, tiene un formato o estilo que no es el apropiado para manejar en estos aplicativos, por lo cual se debe ajustar a lo que se llama “tidy data”, esto se hace con el código de las filas 26 y 27. La función gather, de la librería tidyr, organiza la información de varias columnas en una sola. El primer argumento que se debe poner es la base de datos, “Key”, hace referencia al nombre de la columna que va a contener los nombres de las columnas que se organizarán, el argumento “value”, permite especificar el nombre de la columna que contendrá los valores que estaban ubicados en las filas, finalmente, se especifican las variables que serán sometidas al código.

En la fila 29 se hace una lista con los nombres de las acciones que hacen parte de esta base de datos, esto debido a que serán utilizados más adelante como opciones para una lista desplegable.

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.

UI

Esta parte inicia programando el “UI”, 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 título o nombre que va a tener la página, por defecto, esta aparece al lado izquierdo en la esquina superior. Los comandos principales para esta sección son “style”, utilizado para modificar tipo de letra, color, tamaño y “width”, que sirve para determinar el ancho del espacio que va a ocupar su título.

```
dashboardHeader(title = span("Simulador Utilidades Colcap",
                             style = "font-size: 16px"),
                titleWidth = 250 ),
```

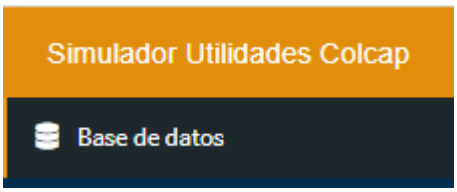
- dashboardSidebar: Luego del encabezado se personalizan las características del menú que aparecerá al lado izquierdo debajo del título. El primer argumento que aparece en el código es “width”, para especificar el ancho de ese menú.
    - sidebarMenu: Si usted va a crear varias pestañas desde el menú, debe usar este comando. Recuerde que estos objetos casi siempre tienen in id, importante al momento de trabajar o modificar estos, esto tiene muchas opciones de personalización y herramientas que se pueden agregar, pero para este caso solo se utilizó “menuItem”.
        - menuItem: Este comando se emplea para dar nombre a las pestañas donde el usuario podrá acceder. En esta página solo hay dos pestañas, una con los filtros para modificar bases de datos y gráficos, y otra pestaña para calcular los retornos de las acciones por medio de dividendos y de retornos del mercado de renta variable.

```
menuItem("Base de datos", tabName = "DataF",
         icon = icon("database", lib = "font-awesome")),
```

Lo primero que debe especificar es el nombre de la pestaña, seguido por el “tabName”, que hace referencia al ID, si usted quiere agregar un ícono puede hacerlo con el argumento

"icon", especificando el nombre del ícono que va a poner y la fuente de ese ícono. Esos logos se pueden bajar de "Font-awesome" y "Glyphicons".

El resultado del **dashboardHeader** y **dashboardSidebar** es el siguiente.



- dashboardBody: Acá 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.
  - Personalización con CSS: Para asemejar los colores de la página con los institucionales, se programó una pequeña parte del código en este lenguaje, para poner el menú de color azul y cambiar el fondo de gris a blanco.

```
dashboardBody(
  tags$head(tags$style(HTML('
  /* main sidebar */
      .skin-yellow .main-sidebar {
                              background-color: #042e4f;
                              }

  /* body */
  .content-wrapper, .right-side {
  background-color: #fafafa;
  }
  '))),
```

  Siempre que empiece a trabajar con CSS, debe agregar la siguiente concatenación de códigos "tags$head(tags$style(HTML()))" dentro de los paréntesis debe especificar la parte que quiere cambiar y lo que quiere obtener.
  Lo primero que se modificó fue el color del menú, donde se especifica que el color de fondo será de color azul. (Los colores deben ser puestos con código también). Luego de esto se modificó el color del cuerpo de la página, donde se especifica un color blanco.
  - tabItems: Cuando se usan varias pestañas u opciones del menú, se deben agrupar todas desde el argumento tabItems, dentro de este debe colocar otro "tabItem", en donde especificará cada cosa de la página.
    - tabItem: En primer lugar, se indica el ID de la opción del menú sobre el cual se va a trabajar. En la siguiente imagen, se colocarán los contenidos para la opción con el ID "DataF", luego de esto se procede a ubicar todos los elementos del cuerpo de

la pestaña. Para este caso, solo aparece el código para agregar la primera herramienta interactiva de la página. Para terminar una pestaña y seguir con los contenidos de la siguiente, debe cerrar los paréntesis del comando “tabItem”, luego poner una coma, volver a abrir otro argumento de “tabItem” y especificar el ID de la otra pestaña. RECUERDE QUE TODO DEBE QUEDAR DENTRO DEL “tabItems”.

Los comandos fluidRow, y dateRangeInput serán explicados más adelante.

```
tabItems(
  tabItem(
    tabName = "DataF",
    fluidRow(
      column(3,
             dateRangeInput("IN_Fechas", "1) Seleccione el rango de fechas",
                            start = "2020-01-01",end = "2022-12-31",
                            min = "2020-01-01",max = "2024-12-31",
                            format = "yyyy-mm-dd",
                            width = "200px"),
```

En términos generales, esos son todos los comandos empleados para establecer la estructura de la página.

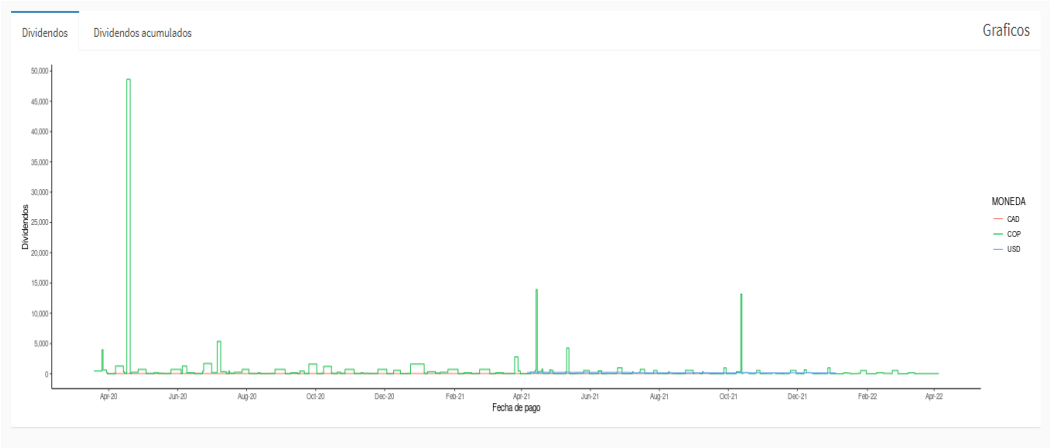
Otros elementos:

- 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 3, luego de eso viene el código del menú. Si no se especifican las columnas se toma todo el espacio como uno solo.
- tabbox: 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 y su título será “Gráficos”. 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”, y otra que tiene el nombre de “dividendos acumulados”, las dos tendrán como contenido un gráfico. Esto se explicará más adelante.

```
fluidRow(
  tabBox(width = 12,title = "Graficos",

         tabPanel("Dividendos", plotOutput("Grafico_D")),
         tabPanel("Dividendos acumulados", plotOutput("Grafico_AD")))
),
```

Resultado: En la parte superior izquierda se encuentran las pestañas, lo de abajo es el contenido. Y en la parte superior derecha queda el título de la caja.



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

```
dateRangeInput("IN_Fechas", "1) Seleccione el rango de fechas",
               start = "2020-01-01",end = "2022-12-31",
               min = "2020-01-01",max = "2024-12-31",
               format = "yyyy-mm-dd",
               width = "200px"),
```

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",
            choices = NULL, multiple = T,
            options = list(`actions-box` = TRUE)),
```

  Resultado: Lista con las opciones establecidas y los botones para facilitar la selección de todas las opciones.

- 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.

- PlotOutput: Gráficos.
- DataTableOutput: DataFrames.
- verbatimTextOutput: Salidas de texto.

```
fluidRow(
  tabBox(width = 12,title = "Graficos",

         tabPanel("Dividendos", plotOutput("Grafico_D")),
         tabPanel("Dividendos acumulados", plotOutput("Grafico_AD")))
),

fluidRow(
  column(12,
         h2(("BASES DE DATOS FECHAS EX-DIVIDENDO"),
            align = "center", style = 'font-size:30px'))
),
br(),
fluidRow(
  tabBox(width = 12,title = "Bases de datos",

         tabPanel("Base original", dataTableOutput("Base")),
         tabPanel("Resumen por dia", dataTableOutput("Base_d")))
),
),
```

Estos Outputs, deben ir relacionados con el Render, que se coloca en el server. Nuevamente, se explicará más adelante.

SERVER:

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.

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

- 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()”.

```
datn<-reactive({
  Archivo2020 %>% filter(SECTOR %in% input$IN_Sector &
                         MONEDA %in% input$IN_Moneda &
                         EMISOR %in% input$IN_Emisor &
                         NEMOTECNICO %in% input$IN_Nemo)%>%
    select(-FECHA.INGRESO, -TOTAL.ENTREGADO.EN.DIVIDENDOS,
           -DESCRIPCION.PAGO.PDU,-MODO.DE.PAGO)

})
```

Actualización de 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 (Archivo2020).
  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”.

```
observe({
  dat0<-filter(Archivo2020,FECHA.ASAMBLEA >= input$IN_Fechas[1] &
                 FECHA.ASAMBLEA <= input$IN_Fechas[2])
  updatePickerInput(session, "IN_Sector", label = "2) Sector economico",
                    choices = sort(unique(dat0$SECTOR)),selected = unique(dat0$SECTOR))
})
```

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 “Archivo2020$MONEDA”, se abre un corchete en donde se especifica el filtro que se quiere aplicar sobre la base de datos. Como se va a actualizar el filtro de la variable MONEDA con la selección del filtro SECTOR, entonces se pone “Archivo2020$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.

```
observe({
  dat00<-Archivo2020$MONEDA[Archivo2020$SECTOR%in%input$IN_Sector]
  updatePickerInput(session, "IN_Moneda", label = "3) Moneda",
                    choices = sort(unique(dat00)),selected = unique(dat00))
})
```

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.

```
output$Base <- renderDataTable({

  Base_M1 <- datn()
  Base_M1
})
```

- 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

```
output$Grafico_D <- renderPlot({
  Dividendos <- Base_AD()
  ggplot(Dividendos, aes(x=FECHA.INICIAL,y=Total, color = MONEDA))+
    geom_step()+
    scale_y_continuous(labels = scales::label_comma(),
                       breaks = scales::breaks_extended(n = 10))+
    scale_x_date(date_breaks = "2 months",
                 date_labels = "%b-%y")+
    labs(x = "Fecha de pago", y = "Dividendos")+
    theme_classic()

})
```

- 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.

```
output$base1 <- renderPrint({

  Reac_b <- Base_R()
  sum(Reac_b$CUOTA)

})
```

En esta página puede encontrar información complementaria:

https://rstudio.github.io/shinydashboard/structure.html

Si tiene dudas, su mejor amigo es StackOverFlow, información en español es muy poca, si necesita investigar algo toca que lo busque todo en inglés y le será mucho más fácil.