UNIVERSIDAD DE COLIMA

Facultad de ingeniería Mecánica y Eléctrica

Ingeniería en Computación Inteligente

“Proyecto Graficas en Go”

Moreno Olmos Luis Miguel

6°B

22/05/2021

Se desea crear una utilería para crear gráficas. Esta va a tener dos opciones:
1.- Crearlas a partir de un archivo CSV existente
2.- Crearlas a partir de valores generados de manera aleatoria.

Se deben dar al menos 3 opciones de gráficas de la página https://github.com/go-echarts/go-echarts, por ejemplo una forma de ejecutar podría ser:

go run graphgen.go --bar data.csv

y graficará los datos del archivo data.csv en una gráfica, generando la gráfica en un archivo html y el servidor para presentarla.

la otra opción podría ser:
go run graphgen --bar --generate

La cantidad de datos generados como las series van a ser fijas, por ejemplo podrían ser 2 series con 10 datos cada una. Los rangos de los valores pueden ser definidos de manera directa o enviar como argumento en el CLI.

la cual generará datos aleatorios dejando la gráfica en un archivo .html y con el servidor para presentarla.

define las opciones de gráfica que se pueden presentar: pie, dots, line, etc.

Entregables:
- código documentado
- Reporte donde se explique el sistema realizado.

Introducción:

Este proyecto consiste en un programa que genere graficas de diferente tipo, como de Barras, Lineas y de Pastel.

Desarrollo:

En el avance incluye:

Estructuras de datos:

El código tiene dos estructuras de datos, para cuando recibe 2 opciones(Para elegir el tipo de gráfica y la forma en que toma los valores) y otra para cuando el usuario seleccione la opción de crear un CSV.

```
type datosCSV struct {
    op1       string
    filas     int
    columnas  int
    nombre    string
    valMinimo int
    valMaximo int
}
```

```
type datos struct {
    op1 string
    op2 string
}
```

Main:

Primeramente, de verifican la cantidad de argumentos que ingresa el usuario, si la cantidad de argumentos fue 1, se verifican que sean 2, esto porque el primer argumento es una dirección que viene por defecto y que no utilizamos.

```
func main() {
    if len(os.Args) == 2
        data1 := datos{...}
        switch data1.op1 {...}
    } else if len(os.Args) == 3 {
        data := datos{...}
        switch data.op1 {...}
    } else if len(os.Args) == 7 {
        filas, err1 := strconv.Atoi(os.Args[2])
        columnas, err2 := strconv.Atoi(os.Args[3])
        valMaximo, err3 := strconv.Atoi(os.Args[5])
        valMinimo, err4 := strconv.Atoi(os.Args[6])
        if err != nil { //Se hacen algunas validaciones por si el usuario ingresa datos erroneos
            fmt.Println( a...: " Porfavor introduce un valor numerico, error en: '", os.Args[1], "'")
        } else if err2 != nil {
            fmt.Println( a...: " Porfavor introduce un valor numerico, error en: '", os.Args[2], "'")
        } else if err3 != nil {...} else if err4 != nil {...} else {
            data := datosCSV{...}
            switch data.op1 {
            case "--createCSV":...
            default:
                fmt.Println( a...: "Opcion no encontrada")
            }
        }
    }
}
```

Las opciones / banderas que se pueden utilizar son 5:

--showgraphs // Muestra las gráficas disponibles con números creados aleatoriamente

--bar // Muestra una grafica de barras y tiene dos opciones

--generate // Genera los datos de forma aleatoria

“Nombre”.csv // Toma el nombre del archivo CSV que se le dé y usas sus datos para hacer la gráfica

--line // Muestra una gráfica de líneas y tiene dos opciones

--generate // Genera los datos de forma aleatoria

“Nombre”.csv // Toma el nombre del archivo CSV que se le dé y usas sus datos para hacer la gráfica

--pie // Muestra una gráfica de pastel y tiene dos opciones

--generate // Genera los datos de forma aleatoria

“Nombre”.csv // Toma el nombre del archivo CSV que se le dé y usas sus datos para hacer la gráfica

--createCSV // Crea un archivo CSV que rellena con valores aleatorios

#1 #3 “Nombre”.csv // Son los datos que necesita para crear el CSV.

#1 // Filas del CSV

#2 // Columnas del CSV

“Nombre”.csv // Nombre con el que se guardará el archivo

La opción “--showgraphs” crea tres archivos html de los tres tipos de graficas que hay (barras, líneas y pastel).

Para esto primero se toman los datos y se agregan a la estructura de datos “datos”, luego se crea un switch para la primera opción, si la opción es igual a uno de los casos llama la función createHTMLgraphs() para crear los archivos HTML. Una vez creado los HTML se ponen en un servidor para poder verlas.

```
if len(os.Args) == 2 {
    data1 := datos{
        op1: os.Args[1],
    }
    switch data1.op1 {
    case "--showgraphs":
        createHTMLgraphs()
        log.Println( v...: "Servidor iniciado en http://localhost:8080/")
        http.HandleFunc( pattern: "/bar", bar)
        http.HandleFunc( pattern: "/line", line)
        http.HandleFunc( pattern: "/pie", pie)
        http.ListenAndServe( addr: ":8080",  handler: nil)
    default:
        fmt.Println( a...: "Opcion no encontrada")
    }
```

Función createHTMLgraphs()

```
func createHTMLgraphs() { //Funcion para crear los archivos HTML de la graficas
    bar := charts.NewBar()
    bar.SetGlobalOptions(charts.WithInitializationOpts(opts.Initialization{
        Theme: types.ThemeChalk,
    }),
        charts.WithTitleOpts(opts.Title{
            Title:    "GRAFICO DE BARRAS",
            Subtitle: "Con datos generados aleatoriamente",
        }))
    bar.SetXAxis([]string{"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"}).
        AddSeries( name: "Category A", generateBarItems()).
        AddSeries( name: "Category B", generateBarItems())
    a, _ := os.Create( name: "bar.html")
    bar.Render(a)

    line := charts.NewLine()
    line.SetGlobalOptions(
        charts.WithInitializationOpts(opts.Initialization{
            Theme: types.ThemeChalk,
```

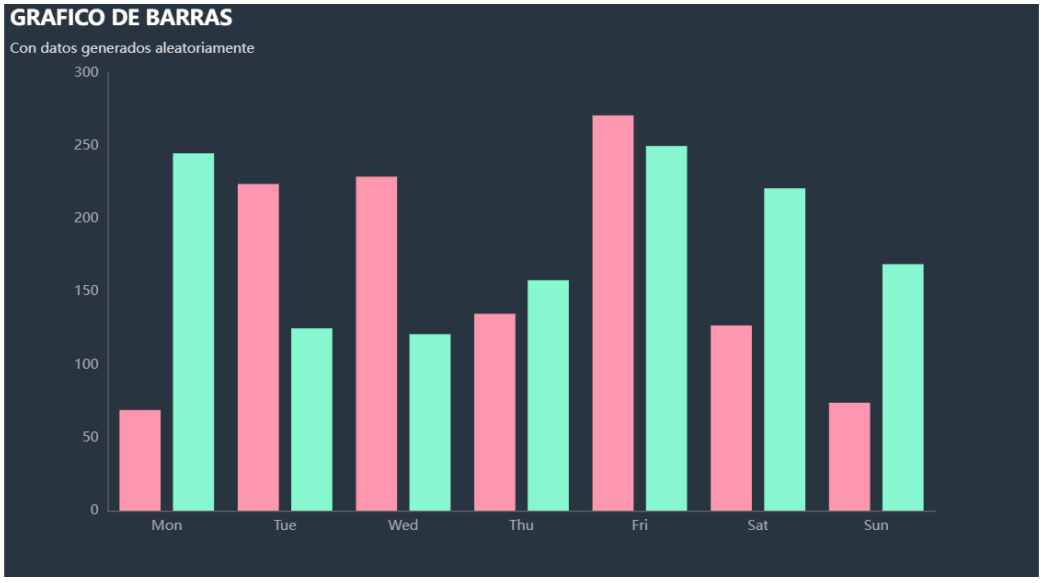
Para publicar los archivos en el servidor.

```
var plantilla = template.Must(template.ParseGlob( pattern: "*.html"))

func bar(w http.ResponseWriter, r *http.Request) {
    plantilla.ExecuteTemplate(w,  name: "bar.html",  data: nil)
}

func line(w http.ResponseWriter, r *http.Request) {
    plantilla.ExecuteTemplate(w,  name: "line.html",  data: nil)
}

func pie(w http.ResponseWriter, r *http.Request) {
    plantilla.ExecuteTemplate(w,  name: "pie.html",  data: nil)
}
```

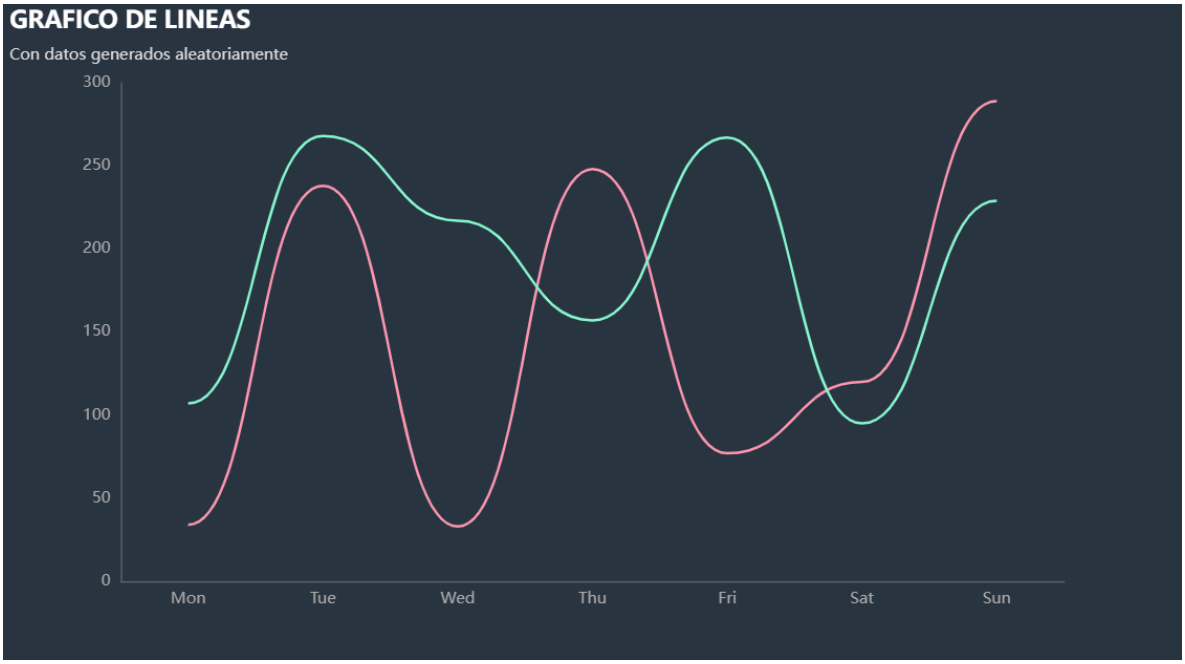
Resultados

```
go run graficas.go --showgraphs
```

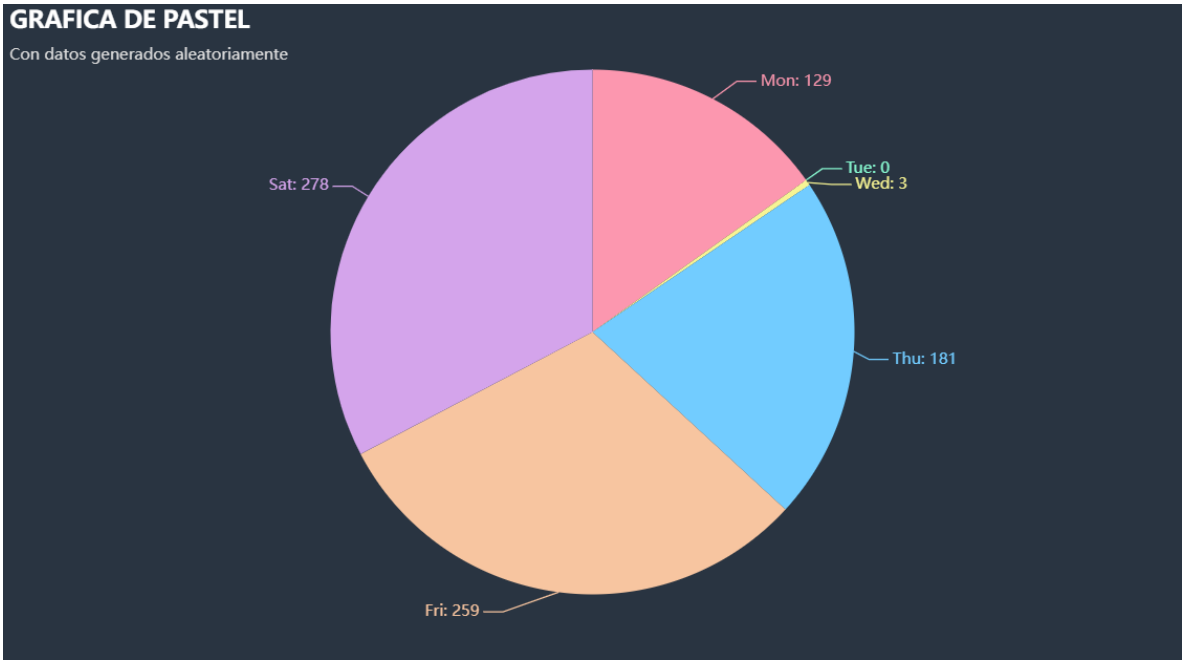
Grafica de barras: localhost:8080/bar



Grafica de líneas: localhost:8080/line



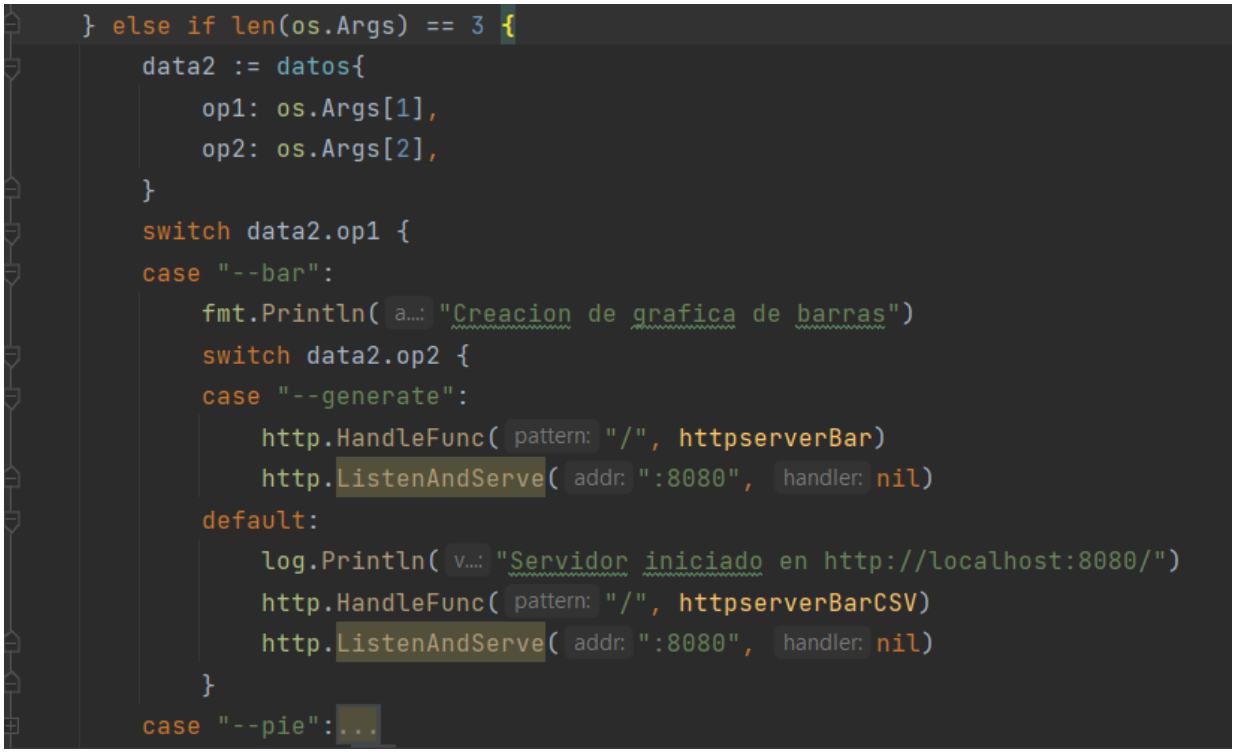
Grafica de pastel: localhost:8080/pie



Dentro de la validación de los 2 elementos se encuentran las opciones para poder mostrar un tipo de grafica con un servidor.

Primero se meten los argumentos a la estructura de datos y después se hace un switch con el primer argumento, cada case corresponde a un tipo de grafica.

Dentro de cada case existe otro para dar la opción de que la gráfica tome datos aleatorios o de un archivo CSV, en caso de ser archivo debes de colocar el nombre del archivo con su extensión, ejemplo “Datos.csv”.



Las dos opciones mandan a llamar la función que corresponde de cada forma de agarrar los datos y mostrarlo en un servidor.

Cada grafica tiene dos funciones correspondientes, similares en su estructura, pero diferente grafica.

```
func generateBarItems() []opts.BarData {...}

func archivoCSVBarItems(op2 string, fila int) []opts.BarData {...}

//------------------------------------------------------------------------//

func generateLineItems() []opts.LineData {...}

func archivoCSVLineItems(op2 string, fila int) []opts.LineData {...}
```

```
func generatePieItems() []opts.PieData {...}

func archivoCSVPieItems(op2 string, fila int) []opts.PieData {...}

//------------------------------------------------------------------------//

func httpserverBar(f http.ResponseWriter, _ *http.Request) {...}

func httpserverBarCSV(f http.ResponseWriter, _ *http.Request) {...}

//------------------------------------------------------------------------//

func httpserverLine(f http.ResponseWriter, _ *http.Request) {...}

func httpserverLineCSV(f http.ResponseWriter, _ *http.Request) {...}

//------------------------------------------------------------------------//

func httpserverPie(f http.ResponseWriter, _ *http.Request) {...}

func httpserverPieCSV(f http.ResponseWriter, _ *http.Request) {...}

//------------------------------------------------------------------------//
```

Para datos generados automáticamente:

```go
func httpserverBar(f http.ResponseWriter, _ *http.Request) {
    bar := charts.NewBar()
    bar.SetGlobalOptions(charts.WithInitializationOpts(opts.Initialization{
        Theme: types.ThemeChalk,
    }),
        charts.WithTitleOpts(opts.Title{
            Title:    "GRAFICO DE BARRAS",
            Subtitle: "Con datos generados aleatoriamente",
        }))
    bar.SetXAxis([]string{"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"}).
        AddSeries( name: "Category A", generateBarItems()).
        AddSeries( name: "Category B", generateBarItems())
    os.Create( name: "bar.html")
    bar.Render(f)
}
```

Para generar los datos se manda a llamar la función generate”Tipo de grafica”Items()

```go
func generateBarItems() []opts.BarData {
    items := make([]opts.BarData, 0)
    for i := 0; i < 7; i++ {
        items = append(items, opts.BarData{Value: rand.Intn( n: 300)})
    }
    return items
}
```

Para datos generados a partir de un archivo CSV:

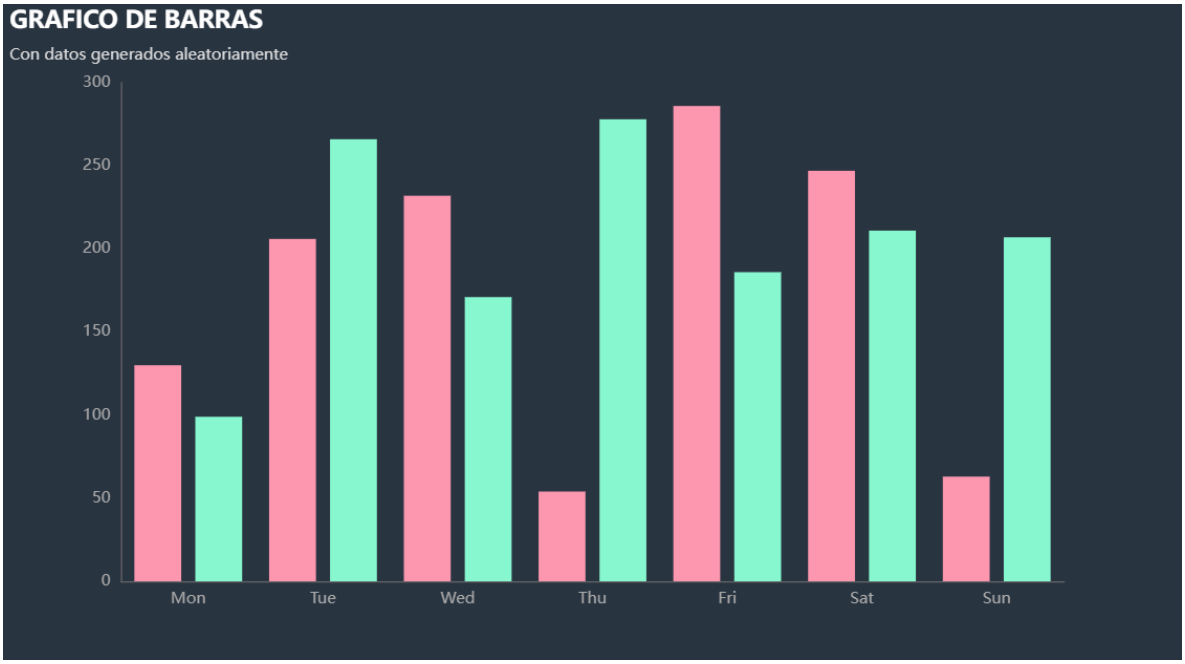
```go
func httpserverBarCSV(f http.ResponseWriter, _ *http.Request) {
    data := datos{
        op2: os.Args[2],
    }
    fileName := data.op2
    fs1, _ := os.Open(fileName)
    r1 := csv.NewReader(fs1)
    content, err := r1.ReadAll()
    SliceLetras := []string{"A", "B", "C", "D", "E", "F", "G", "H", "I", "j", "K"}
    if err != nil {
        log.Fatalf( format: "No se pudo leer el archivo, el error es: %+v", err)
    }
    bar := charts.NewBar()
    bar.SetGlobalOptions(charts.WithInitializationOpts(opts.Initialization{
        Theme: types.ThemeChalk,
    }),
        charts.WithTitleOpts(opts.Title{
            Title:    "GRAFICA DE BARRAS",
            Subtitle: "Con datos tomados de un archivo CSV",
        }))

    for ix, _ := range content {
        bar.SetXAxis([]string{"Lunes", "Martes", "Miercoles", "Jueves", "Viernes", "Sabado", "Domingo"}).
            AddSeries("Category"+SliceLetras[ix], archivoCSVBarItems(data.op2, ix))
    }
    os.Create( name: "barCSV.html")
    bar.Render(f)
}
```

Para generar los datos se manda a llamar la función archivoCSV” Tipo de grafica ”Items(), esta requiere de 2 argumentos, uno para darle el nombre del archivo CSV y el otro es el numero de la fila que va a tomar los datos.

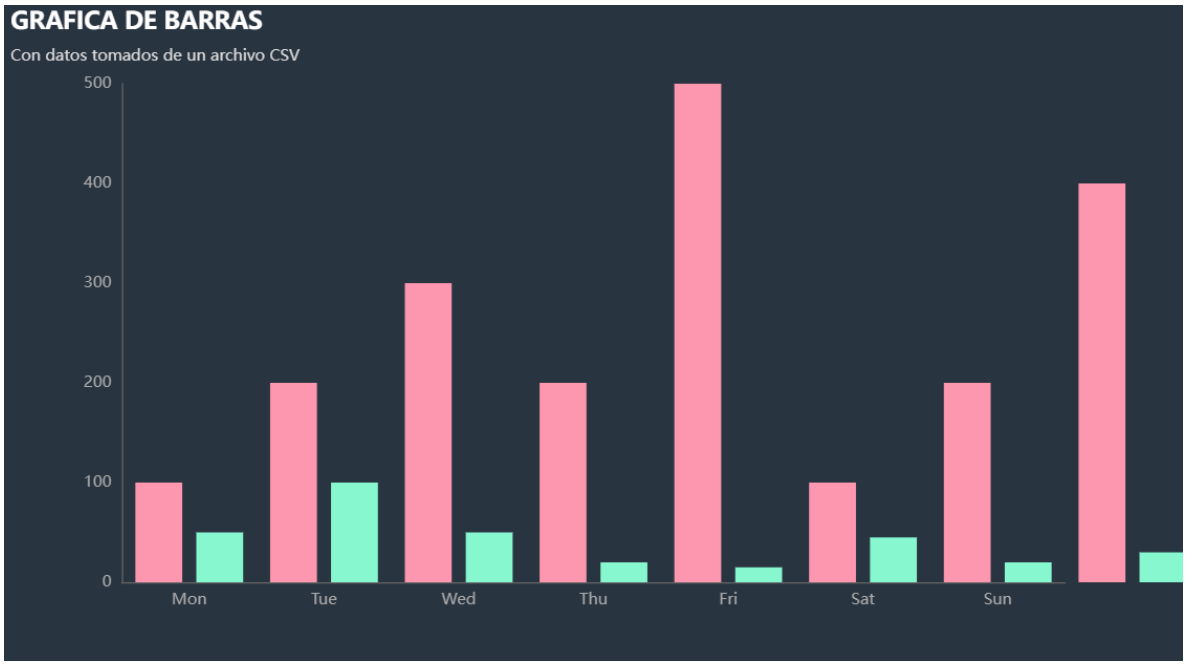
```go
func archivoCSVBarItems(op2 string, fila int) []opts.BarData {
    fileName := op2
    fs1, _ := os.Open(fileName)
    r1 := csv.NewReader(fs1)
    content, err := r1.ReadAll()
    if err != nil {
        log.Fatalf( format: "can not readall, err is %+v", err)
    }
    items := make([]opts.BarData, 0)
    for i := 0; i < 7; i++ {
        for ix, row := range content {
            if ix == fila {
                for _, valor := range row {
                    items = append(items, opts.BarData{Value: valor})
                }
            }
        }
    }
    return items
}
```

Graficas resultantes:

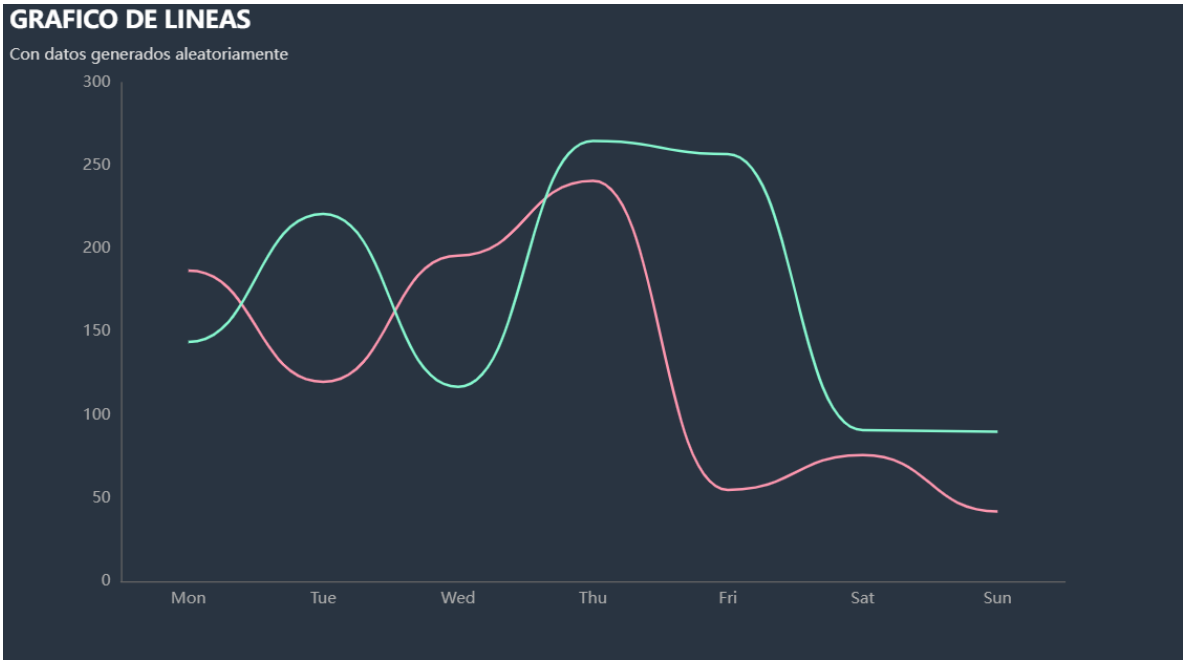
```
go run graficas.go --bar --generate
```



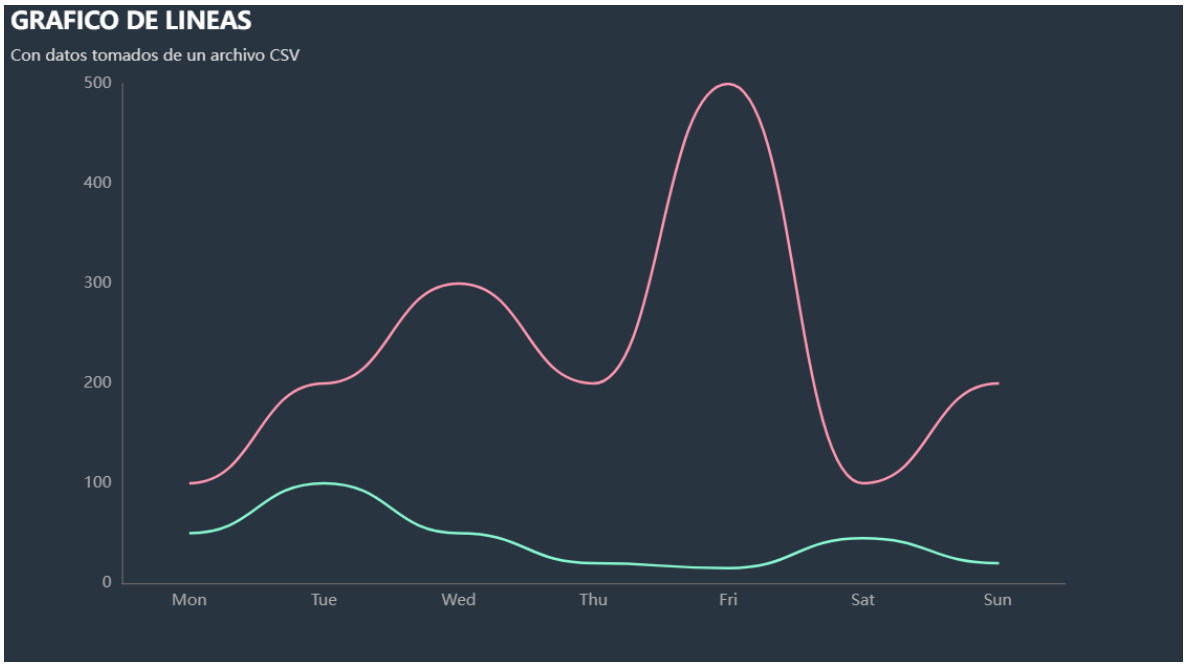

```
go run graficas.go --bar Datos.csv
```



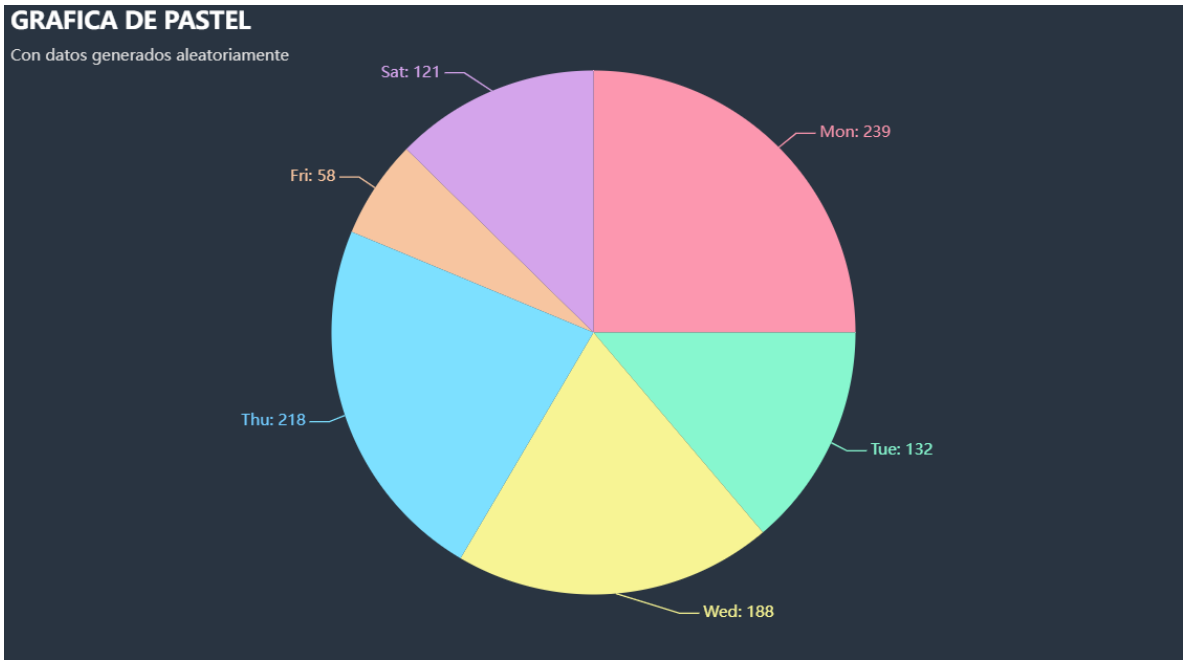
```
go run graficas.go --line --generate
```



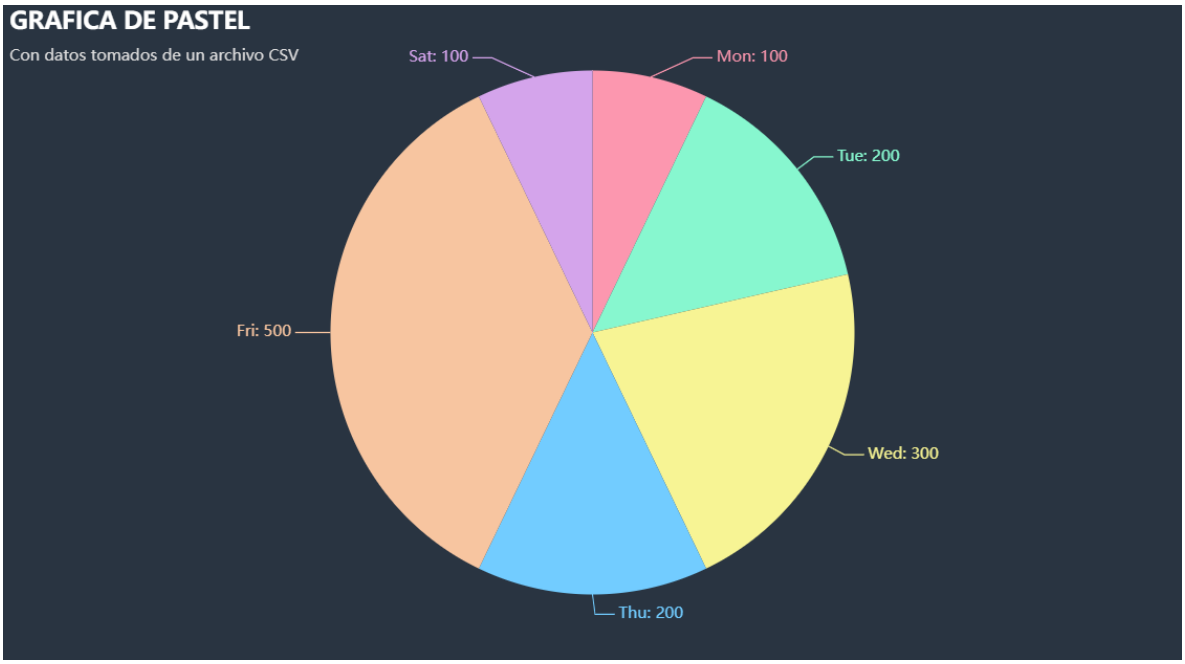

```
go run graficas.go --line Datos.csv
```



```
go run graficas.go --pie --generate
```



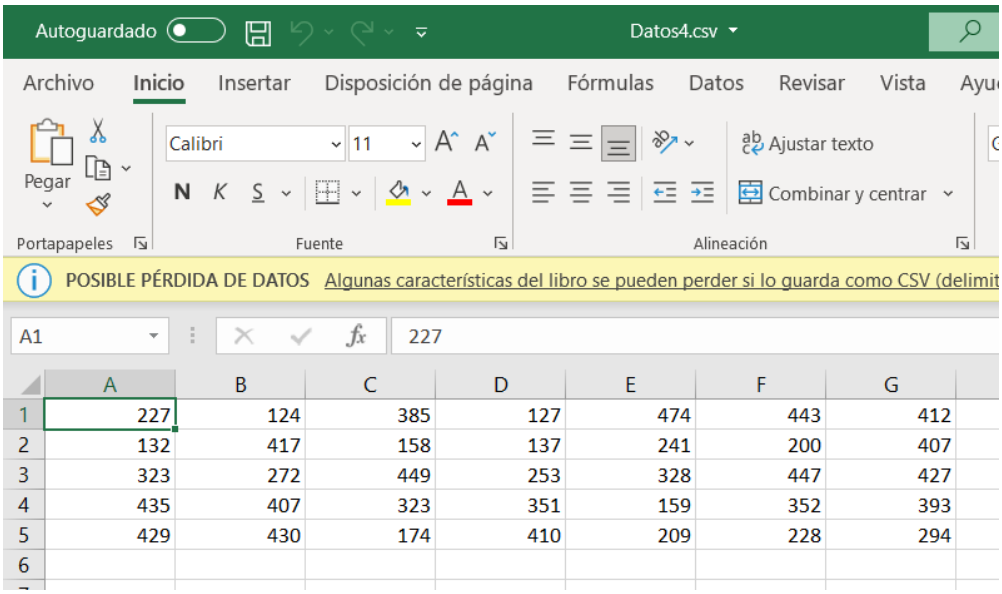

```
go run graficas.go --pie Datos.csv
```



Por último, la última validación es para crear los archivos CSV, para poder crearlos primero los datos se ponen en la segunda estructura de datos, la cual es “datosCSV”. Después se hace un switch con un único case para la primera opción, la cual es “--createCSV”.

```go
switch data.op1 {
case "--createCSV":
    var miSlice = make([][]string, filas) //Se crea el slice 2d
    for i := 0; i < filas; i++ {
        miSlice[i] = make([]string, columnas)
    }
    rand.Seed(time.Now().UnixNano())
    for i := 0; i < filas; i++ {
        for j := 0; j < columnas; j++ { //Ciclo para rellener el resto de columnas
            numeros := valMinimo + rand.Intn(valMaximo-valMinimo)
            strr := strconv.Itoa(numeros) //Se convierten a string para ser ingresados al slice
            miSlice[i][j] = strr
        }
    }
    file, err := os.Create(data.nombre) //Se crea el archivo
    if err != nil {
        log.Fatal( v...: " Ha ocurrido un error", err)
    }
    archivo := csv.NewWriter(file)
    defer archivo.Flush()
    for _, rows := range miSlice {
        archivo.Write(rows)
    }
    fmt.Println( a...: "Archivo CSV creado exitosamente")
default:
    fmt.Println( a...: "Opcion no encontrada")
}
```

```
PS D:\6to Semestre\Análisis y Visualización de Datos (Walter)\ProyectoFinal> go run graficas.go --createCSV 5 7 Datos4.csv 500 100
Archivo CSV creado exitosamente
```



	A	B	C	D	E	F	G
1	227	124	385	127	474	443	412
2	132	417	158	137	241	200	407
3	323	272	449	253	328	447	427
4	435	407	323	351	159	352	393
5	429	430	174	410	209	228	294
6							

Código:

```
package main

import (
   "encoding/csv"
   "fmt"
   "github.com/go-echarts/go-echarts/v2/charts"
   "github.com/go-echarts/go-echarts/v2/opts"
   "github.com/go-echarts/go-echarts/v2/types"
   "log"
   "math/rand"
   "net/http"
   "os"
   "strconv"
   "text/template"
   "time"
)

type datos struct {
   op1 string
   op2 string
}

type datosCSV struct {
   op1       string
   filas     int
   columnas  int
   nombre    string
   valMinimo int
   valMaximo int
}

func createHTMLgraphs() { //Funcion para crear los archivos HTML de la
graficas
   bar := charts.NewBar()

bar.SetGlobalOptions(charts.WithInitializationOpts(opts.Initialization{
      Theme: types.ThemeChalk,
   }),
      charts.WithTitleOpts(opts.Title{
         Title:    "GRAFICO DE BARRAS",
         Subtitle: "Con datos generados aleatoriamente",
      }))
   bar.SetXAxis([]string{"Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
"Sun"}).
      AddSeries("Category A", generateBarItems()).
      AddSeries("Category B", generateBarItems())
   a, _ := os.Create("bar.html")
   bar.Render(a)

   line := charts.NewLine()
   line.SetGlobalOptions(
      charts.WithInitializationOpts(opts.Initialization{
         Theme: types.ThemeChalk,
      }),
      charts.WithTitleOpts(opts.Title{
```

```
            Title:    "GRAFICO DE LINEAS",
            Subtitle: "Con datos generados aleatoriamente",
        }),
    )
    line.SetXAxis([]string{"Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
"Sun"}).
        AddSeries("Category A", generateLineItems()).
        AddSeries("Category B", generateLineItems()).
        SetSeriesOptions(charts.WithLineChartOpts(opts.LineChart{Smooth:
true}))
    b, _ := os.Create("line.html")
    line.Render(b)

    pie := charts.NewPie()

pie.SetGlobalOptions(charts.WithInitializationOpts(opts.Initialization{
        Theme: types.ThemeChalk,
    }),
        charts.WithTitleOpts(
            opts.Title{
                Title:    "GRAFICA DE PASTEL",
                Subtitle: "Con datos generados aleatoriamente",
            },
        ),
    )
    pie.SetSeriesOptions()
    pie.AddSeries("Monthly revenue",
        generatePieItems()).
        SetSeriesOptions(
            charts.WithPieChartOpts(
                opts.PieChart{
                    Radius: 200,
                },
            ),
            charts.WithLabelOpts(
                opts.Label{
                    Show:      true,
                    Formatter: "{b}: {c}",
                },
            ),
        )
    c, _ := os.Create("pie.html")
    pie.Render(c)

}

//------------------------------------------------------------------------
------//

func generateBarItems() []opts.BarData {
    items := make([]opts.BarData, 0)
    for i := 0; i < 7; i++ {
        items = append(items, opts.BarData{Value: rand.Intn(300)})
    }
    return items
}
```

```
func archivoCSVBarItems(op2 string, fila int) []opts.BarData {
   fileName := op2
   fs1, _ := os.Open(fileName)
   r1 := csv.NewReader(fs1)
   content, err := r1.ReadAll()
   if err != nil {
      log.Fatalf("can not readall, err is %+v", err)
   }
   items := make([]opts.BarData, 0)
   for i := 0; i < 6; i++ {
      for ix, row := range content {
         if ix == fila {
            for _, valor := range row {
               items = append(items, opts.BarData{Value: valor})
            }
         }
      }
   }
   return items
}

//------------------------------------------------------------------------
------//

func generateLineItems() []opts.LineData {
   items := make([]opts.LineData, 0)
   for i := 0; i < 7; i++ {
      items = append(items, opts.LineData{Value: rand.Intn(300)})
   }
   return items
}

func archivoCSVLineItems(op2 string, fila int) []opts.LineData {
   fileName := op2
   fs1, _ := os.Open(fileName)
   r1 := csv.NewReader(fs1)
   content, err := r1.ReadAll()
   if err != nil {
      log.Fatalf("can not readall, err is %+v", err)
   }
   items := make([]opts.LineData, 0)
   for i := 0; i < 7; i++ {
      for ix, row := range content {
         if ix == fila {
            for _, valor := range row {
               items = append(items, opts.LineData{Value: valor})
            }
         }
      }
   }
   return items
}

//------------------------------------------------------------------------
------//

func generatePieItems() []opts.PieData {
```

```
    dias := []string{"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"}
    items := make([]opts.PieData, 0)
    for i := 0; i < 6; i++ {
        items = append(items, opts.PieData{
            Name:  dias[i],
            Value: rand.Intn(300)})
    }
    return items
}

func archivoCSVPieItems(op2 string, fila int) []opts.PieData {
    fileName := op2
    fs1, _ := os.Open(fileName)
    r1 := csv.NewReader(fs1)
    content, err := r1.ReadAll()
    if err != nil {
        log.Fatalf("can not readall, err is %+v", err)
    }
    dias := []string{"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"}
    items := make([]opts.PieData, 0)
    for i := 0; i < 6; i++ {
        items = append(items, opts.PieData{
            Name:  dias[i],
            Value: content[fila][i]})
    }
    return items
}

//------------------------------------------------------------------------
------//

func httpserverBar(f http.ResponseWriter, _ *http.Request) {
    bar := charts.NewBar()

bar.SetGlobalOptions(charts.WithInitializationOpts(opts.Initialization{
        Theme: types.ThemeChalk,
    }),
        charts.WithTitleOpts(opts.Title{
            Title:    "GRAFICO DE BARRAS",
            Subtitle: "Con datos generados aleatoriamente",
        }))
    bar.SetXAxis([]string{"Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
"Sun"}).
        AddSeries("Category A", generateBarItems()).
        AddSeries("Category B", generateBarItems())
    os.Create("bar.html")
    bar.Render(f)
}

func httpserverBarCSV(f http.ResponseWriter, _ *http.Request) {
    data := datos{
        op2: os.Args[2],
    }
    fileName := data.op2
    fs1, _ := os.Open(fileName)
    r1 := csv.NewReader(fs1)
    content, err := r1.ReadAll()
```

```
   SliceLetras := []string{"A", "B", "C", "D", "E", "F", "G", "H", "I",
"j", "K"}
   if err != nil {
      log.Fatalf("No se pudo leer el archivo, el error es: %+v", err)
   }
   bar := charts.NewBar()

bar.SetGlobalOptions(charts.WithInitializationOpts(opts.Initialization{
      Theme: types.ThemeChalk,
   }),
      charts.WithTitleOpts(opts.Title{
         Title:    "GRAFICA DE BARRAS",
         Subtitle: "Con datos tomados de un archivo CSV",
      }))

   for ix, _ := range content {
      bar.SetXAxis([]string{"Lunes", "Martes", "Miercoles", "Jueves",
"Viernes", "Sabado", "Domingo"}).
         AddSeries("Category"+SliceLetras[ix],
archivoCSVBarItems(data.op2, ix))
   }
   os.Create("barCSV.html")
   bar.Render(f)
}

//------------------------------------------------------------------------
------//

func httpserverLine(f http.ResponseWriter, _ *http.Request) {
   line := charts.NewLine()
   line.SetGlobalOptions(
      charts.WithInitializationOpts(opts.Initialization{
         Theme: types.ThemeChalk,
      }),
      charts.WithTitleOpts(opts.Title{
         Title:    "GRAFICO DE LINEAS",
         Subtitle: "Con datos generados aleatoriamente",
      }),
   )
   line.SetXAxis([]string{"Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
"Sun"}).
      AddSeries("Category A", generateLineItems()).
      AddSeries("Category B", generateLineItems()).
      SetSeriesOptions(charts.WithLineChartOpts(opts.LineChart{Smooth:
true}))
   os.Create("line.html")
   line.Render(f)
}

func httpserverLineCSV(f http.ResponseWriter, _ *http.Request) {
   data := datos{op2: os.Args[2]}
   fileName := data.op2
   fs1, _ := os.Open(fileName)
   r1 := csv.NewReader(fs1)
   content, err := r1.ReadAll()
   SliceLetras := []string{"A", "B", "C", "D", "E", "F", "G", "H", "I",
"j", "K"}
```

```
   if err != nil {
      log.Fatalf("No se pudo leer el archivo, el error es: %+v", err)
   }
   line := charts.NewLine()
   line.SetGlobalOptions(
      charts.WithInitializationOpts(opts.Initialization{
         Theme: types.ThemeChalk,
      }),
      charts.WithTitleOpts(opts.Title{
         Title:    "GRAFICO DE LINEAS",
         Subtitle: "Con datos tomados de un archivo CSV",
      }),
   )
   for ix, _ := range content {
      line.SetXAxis([]string{"Mon", "Tue", "Wed", "Thu", "Fri", "Sat",
"Sun"}).
         AddSeries("Category "+SliceLetras[ix],
archivoCSVLineItems(data.op2, ix)).
         SetSeriesOptions(charts.WithLineChartOpts(opts.LineChart{Smooth:
true}))
   }
   os.Create("lineCSV.html")
   line.Render(f)
}

//------------------------------------------------------------------------
------//

func httpserverPie(f http.ResponseWriter, _ *http.Request) {
   pie := charts.NewPie()

pie.SetGlobalOptions(charts.WithInitializationOpts(opts.Initialization{
      Theme: types.ThemeChalk,
   }),
      charts.WithTitleOpts(
         opts.Title{
            Title:    "GRAFICA DE PASTEL",
            Subtitle: "Con datos generados aleatoriamente",
         },
      ),
   )
   pie.SetSeriesOptions()
   pie.AddSeries("Monthly revenue",
      generatePieItems()).
      SetSeriesOptions(
         charts.WithPieChartOpts(
            opts.PieChart{
               Radius: 200,
            },
         ),
         charts.WithLabelOpts(
            opts.Label{
               Show:      true,
               Formatter: "{b}: {c}",
            },
         ),
      )
```

```
   pie.Render(f)
}

func httpserverPieCSV(f http.ResponseWriter, _ *http.Request) {
   data := datos{
      op2: os.Args[2],
   }
   pie := charts.NewPie()
   pie.SetGlobalOptions(
      charts.WithInitializationOpts(opts.Initialization{
         Theme: types.ThemeChalk,
      }),
      charts.WithTitleOpts(
         opts.Title{
            Title:    "GRAFICA DE PASTEL",
            Subtitle: "Con datos tomados de un archivo CSV",
         },
      ),
   )
   pie.SetSeriesOptions()
   pie.AddSeries("Monthly revenue",
      archivoCSVPieItems(data.op2, 0)).
      SetSeriesOptions(
         charts.WithPieChartOpts(
            opts.PieChart{
               Radius: 200,
            },
         ),
         charts.WithLabelOpts(
            opts.Label{
               Show:      true,
               Formatter: "{b}: {c}",
            },
         ),
      )
   pie.Render(f)
}

//------------------------------------------------------------------------
------//

var plantilla = template.Must(template.ParseGlob("*.html"))

func bar(w http.ResponseWriter, r *http.Request) {
   plantilla.ExecuteTemplate(w, "bar.html", nil)
}

func line(w http.ResponseWriter, r *http.Request) {
   plantilla.ExecuteTemplate(w, "line.html", nil)
}

func pie(w http.ResponseWriter, r *http.Request) {
   plantilla.ExecuteTemplate(w, "pie.html", nil)
}

func main() {
   if len(os.Args) == 2 {
```

```
    data1 := datos{
        op1: os.Args[1],
    }
    switch data1.op1 {
    case "--showgraphs":
        createHTMLgraphs()
        log.Println("Servidor iniciado en http://localhost:8080")
        http.HandleFunc("/bar", bar)
        http.HandleFunc("/line", line)
        http.HandleFunc("/pie", pie)
        http.ListenAndServe(":8080", nil)
    default:
        fmt.Println("Opcion no encontrada")
    }
} else if len(os.Args) == 3 {
    data2 := datos{
        op1: os.Args[1],
        op2: os.Args[2],
    }
    switch data2.op1 {
    case "--bar":
        fmt.Println("Creacion de grafica de barras")
        switch data2.op2 {
        case "--generate":
            http.HandleFunc("/", httpserverBar)
            http.ListenAndServe(":8080", nil)
        default:
            log.Println("Servidor iniciado en http://localhost:8080/")
            http.HandleFunc("/", httpserverBarCSV)
            http.ListenAndServe(":8080", nil)
        }
    case "--pie":
        fmt.Println("Grafcia de pastel")
        switch data2.op2 {
        case "--generate":
            log.Println("Servidor iniciado en http://localhost:8080/")
            http.HandleFunc("/", httpserverPie)
            http.ListenAndServe(":8080", nil)
        default:
            log.Println("Servidor iniciado en http://localhost:8080/")
            http.HandleFunc("/", httpserverPieCSV)
            http.ListenAndServe(":8080", nil)
        }
    case "--line":
        fmt.Println("Grafcia de lineas")
        switch data2.op2 {
        case "--generate":
            log.Println("Servidor iniciado en http://localhost:8080/")
            http.HandleFunc("/", httpserverLine)
            http.ListenAndServe(":8080", nil)
        default:
            log.Println("Servidor iniciado en http://localhost:8080/")
            http.HandleFunc("/", httpserverLineCSV)
            http.ListenAndServe(":8080", nil)
        }
    default:
        fmt.Println("Opcion no encontrada")
```

```
      }
   } else if len(os.Args) == 7 {
      filas, err := strconv.Atoi(os.Args[2])
      columnas, err2 := strconv.Atoi(os.Args[3])
      valMaximo, err3 := strconv.Atoi(os.Args[5])
      valMinimo, err4 := strconv.Atoi(os.Args[6])
      if err != nil { //Se hacen algunas validaciones por si el usuario
ingresa datos erroneos
         fmt.Println(" Porfavor introduce un valor numerico, error en:
'", os.Args[1], "'")
      } else if err2 != nil {
         fmt.Println(" Porfavor introduce un valor numerico, error en:
'", os.Args[2], "'")
      } else if err3 != nil {
         fmt.Println(" Porfavor introduce un valor numerico, error en:
'", os.Args[3], "'")
      } else if err4 != nil {
         fmt.Println(" Porfavor introduce un valor numerico, error en:
'", os.Args[4], "'")
      } else {
         data := datosCSV{
            op1:       os.Args[1],
            filas:     filas,
            columnas:  columnas,
            nombre:    os.Args[4],
            valMaximo: valMaximo,
            valMinimo: valMinimo,
         }
         switch data.op1 {
         case "--createCSV":
            var miSlice = make([][]string, filas) //Se crea el slice 2d
            for i := 0; i < filas; i++ {
               miSlice[i] = make([]string, columnas)
            }
            rand.Seed(time.Now().UnixNano())
            for i := 0; i < filas; i++ {
               for j := 0; j < columnas; j++ { //Ciclo para rellener el
resto de columnas
                  numeros := valMinimo + rand.Intn(valMaximo-valMinimo)
                  strr := strconv.Itoa(numeros) //Se convierten a string
para ser ingresados al slice
                  miSlice[i][j] = strr
               }
            }
            file, err := os.Create(data.nombre) //Se crea el archivo
            if err != nil {
               log.Fatal(" Ha ocurrido un error", err)
            }
            archivo := csv.NewWriter(file)
            defer archivo.Flush()
            for _, rows := range miSlice {
               archivo.Write(rows)
            }
            fmt.Println("Archivo CSV creado exitosamente")
         default:
            fmt.Println("Opcion no encontrada")
         }
```

```
        }
    }
}
```