

Scala: Trabajo Final

Análisis exploratorio con Scala

Introducción:

Queremos realizar un análisis exploratorio de un dataset en formato csv. Este dataset suele ser usado para procesos de ML en el que se intenta predecir si los ingresos de una persona serán superiores a los 50K anuales.

Descripción:

Se solicita implementar un programa en Scala que sea capaz de hacer un análisis exploratorio básico y sacar algunas conclusiones del dataset. Para ello se hará un uso de las funciones que provee la API de las colecciones de Scala.

También se pedirá en algún apartado que se defina un companion object para poder disponer de funciones de utilidades.

Se va a proveer de un código fuente del programa del cual se tiene que partir y completarlo para su correcto funcionamiento.

En el código provisto se dispone de una función de lectura del fichero csv y la definición de funciones que se deben implementar. Así como también la definición de la clase a la que se mapea cada una de las filas del csv.

El código fuente provisto está formado por 4 ficheros .scala y 1 fichero .csv y tiene la siguiente estructura:

```
src
├── adult.data.clean.csv
├── main
│   └── scala
│       ├── AnalisisExploratorio.scala
│       ├── Analizador.scala
│       ├── Contribuyente.scala
│       └── Utilidades.scala
└── test
    └── scala
```

- AnalisisExploratorio.scala -> Es el fichero del programa principal.
- Analizador.scala -> Contiene la definición de un trait que se deberá usar para completar una de los ejercicios.
- Contribuyente.scala -> Definición de la clase Contribuyente a la que se mapea cada fila del fichero .csv.

- Utilidades.scala -> Definición de un objeto que tiene implementada la función de lectura del csv y devuelve una secuencia de instancias de la clase Contribuyente: Seq[Contribuyente].
- adult.data.clean.csv -> dataset que tiene la siguientes columnas:

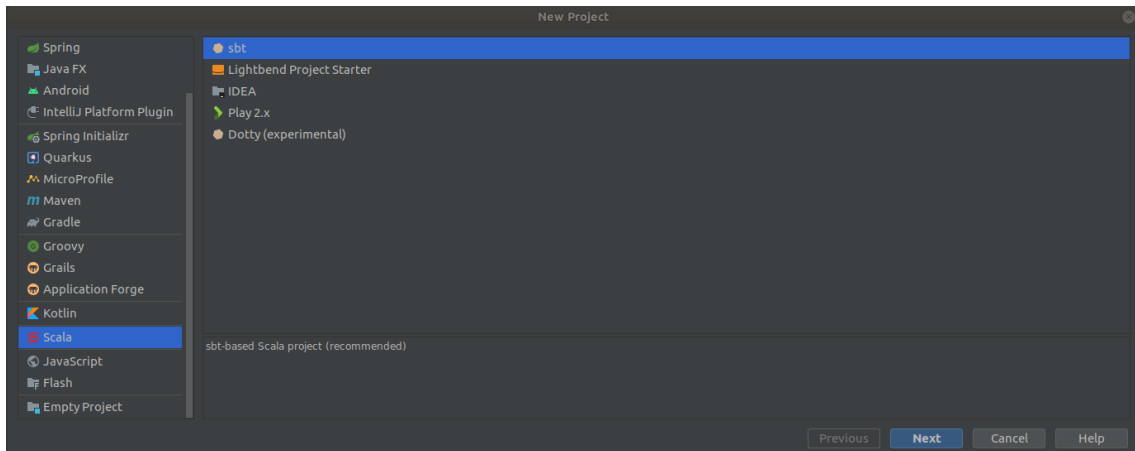
age: int.
workclass: string.
education: String
education-num: int
marital-status: string
occupation: string
relationship: string
race: string
sex: string
capital-gain: string
capital-loss: string
hours-per-week: string
native-country: string
income: string

Para poder hacer uso del código fuente que se provee, hace falta crear un proyecto nuevo de Scala en IntelliJ, lo podemos hacer desde la vista inicial:

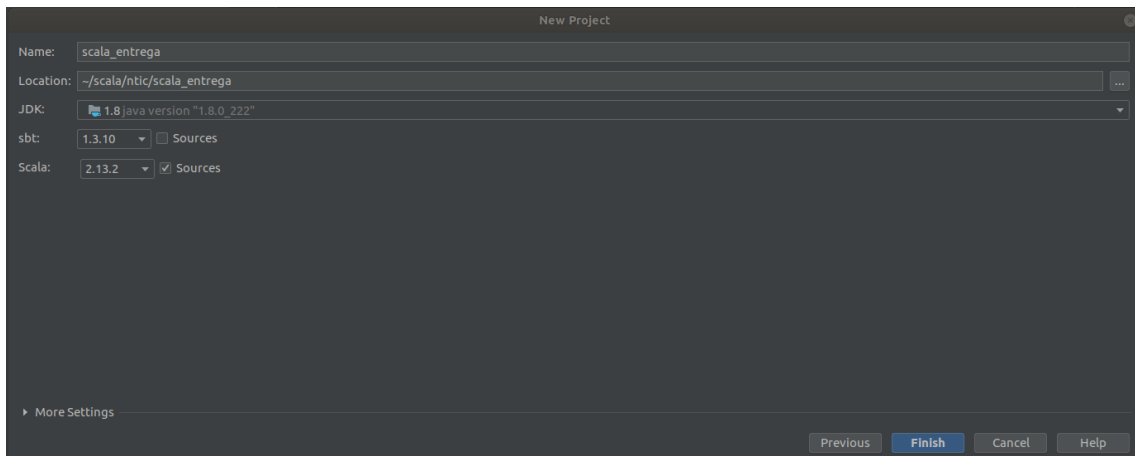


o si lo teníamos previamente abierto: File -> New -> Project.

El proyecto tiene que ser de Scala y con sbt:

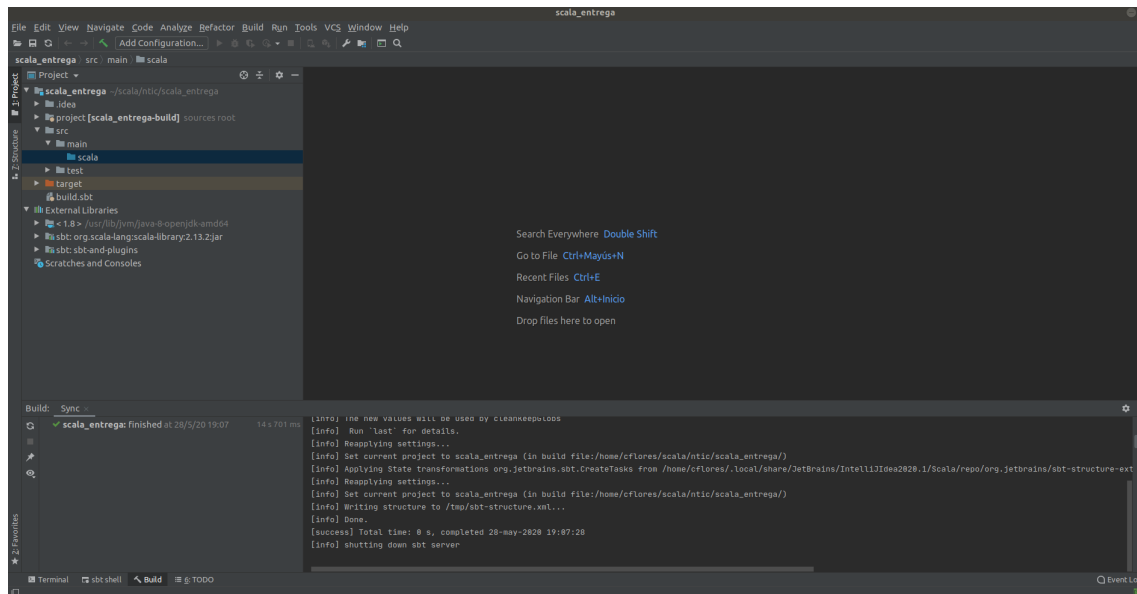


se establece un nombre al proyecto:



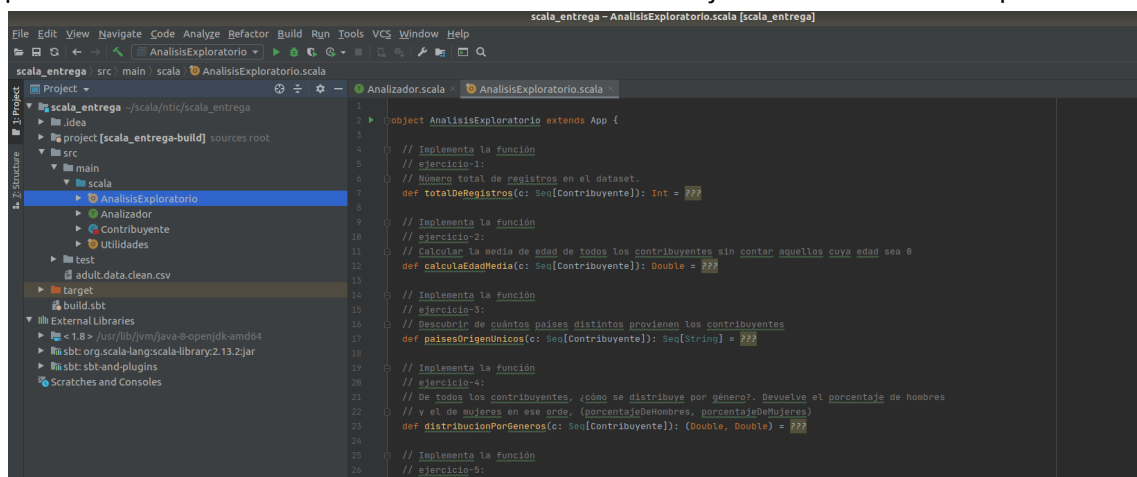
Y una vez establecido el nombre, se puede finalizar clickando sobre el botón Finish

Una vez IntelliJ ha terminado de generar el proyecto:

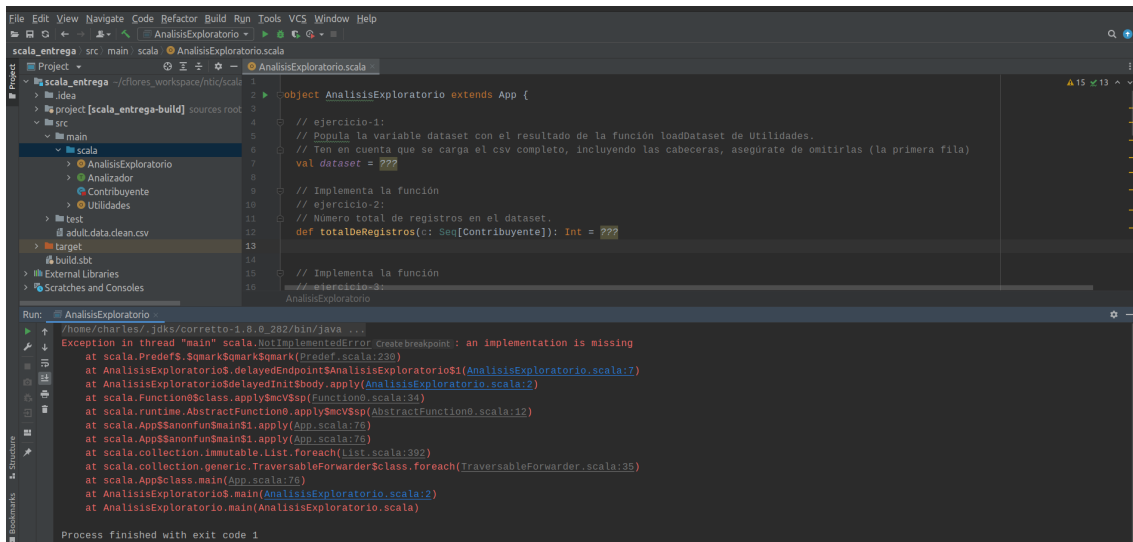


IntelliJ por defecto nos crea una carpeta **src** vacía, esta carpeta debemos reemplazarla por la que se proporciona dentro del fichero src.zip

Una vez reemplazada la carpeta **src** por la que se encuentra dentro del src.zip, podemos hacer una prueba rápida para comprobar que se ha cargado correctamente, para eso abrimos el objeto **AnalisisExploratorio**:



Y si lo ejecutamos, para ello podemos darle al botón de Run (situado a la derecha del número de línea), situado a la izquierda de la definición del objeto. Al darle a Run, es probable que aparezcan varias opciones de las cuales debemos elegir Run, esto intentará ejecutar el programa terminará dejando una salida de error como la siguiente:



```
object AnalisisExploratorio extends App {  
  // ejercicio-1:  
  // Popula la variable dataset con el resultado de la función loadDataset de Utilidades.  
  // Ten en cuenta que se carga el csv completo, incluyendo las cabeceras, asegurate de omitirlas (la primera fila)  
  val dataset = ???  
  // Implementa la función  
  // ejercicio-2:  
  // Número total de registros en el dataset.  
  def totalDeRegistros(c: Seq[Contribuyente]): Int = ???  
  // Implementa la función  
  // ejercicio-3:  
  AnalisisExploratorio  
}
```

Run: AnalisisExploratorio
/home/charles/.jdk/corretto-1.8.0_282/bin/java ...
Exception in thread "main" scala.NotImplementedError: create breakpoint: an implementation is missing
at scala.Predef\$.ScalaSquarkSquark(Predef.scala:288)
at AnalisisExploratorio\$.delayedInit\$body.apply(AnalisisExploratorio.scala:7)
at AnalisisExploratorio\$.delayedInit\$body.apply(AnalisisExploratorio.scala:2)
at scala.Function0\$class.apply\$mcV\$sp(Function0.scala:34)
at scala.runtime.AbstractFunction0.apply\$mcV\$sp(AbstractFunction0.scala:12)
at scala.App\$anonfun\$main\$1\$.apply(App.scala:76)
at scala.App\$anonfun\$main\$1\$.apply(App.scala:76)
at scala.collection.immutable.List.foreach(List.scala:392)
at scala.collection.generic.TraversableForwarder\$class.foreach(TraversableForwarder.scala:36)
at scala.App\$class.main(App.scala:76)
at AnalisisExploratorio\$.main(AnalisisExploratorio.scala:2)
at AnalisisExploratorio.main(AnalisisExploratorio.scala)
Process finished with exit code 1

Este error se debe a que hay una expresión que se va a evaluar y no tiene su implementación (¡¡es la que debes completar!!). Llegados hasta aquí, estamos listos, ya se puede empezar a desarrollar.

Nota: fijémonos que las funciones están definidas tal que:

```
def totalDeRegistros(c: Seq[Contribuyente]): Int = ???
```

fijémonos que la función evaluará y devolverá (el cuerpo de la función) es: ???, esta sintaxis es únicamente útil para dejar la función sin implementar y que el compilador no lo detecte como un error. La implementación de la función debe reemplazar los ??? e ir entre '{' y '}' si procede.

Si exploramos el código de Análisis exploratorio y nos centramos en las líneas de la 39 a la 44, vemos que las llamadas a la función println están comentadas. Estas líneas pueden ser descomentadas en el momento de desarrollo para evaluar las invocaciones a las funciones implementadas.

Ejercicios:

1. (0.5) Carga del dataset. Usa la función *loadDataset* del objeto Singleton para obtener el dataset cargado y asignar el valor retornado en la variable *dataset*

del objeto *AnalisisExploratorio*. La función *loadDataset* carga el csv completo, incluyendo las cabeceras, asegúrate de omitir las cabeceras (primer elemento de la colección que devuelve *loadDataset*).

2. (0.5 puntos) Número total de registros en el dataset.
3. (0.5 puntos) Calcular la media de edad de todos los contribuyentes.
4. (0.5 puntos) Calcular la media de edad de todos los contribuyentes sin contar aquellos que nunca se han casado: marital-status = Never-married.
5. (1 punto) Descubrir los países únicos de donde provienen los contribuyentes.
6. (1 punto) De todos los contribuyentes, cómo se distribuye por género? ¿cuál es el porcentaje de hombres? y ¿el de mujeres?. Esta función debe devolver una tupla, un par de Double, donde la primera posición debe contener el porcentaje de hombres y en la segunda posición el porcentaje de mujeres.
7. (2 puntos) Detectar cuál es el tipo de trabajo (workclass) mejor remunerado. El trabajo mejor remunerado es aquel trabajo donde el porcentaje de los contribuyentes que perciben ingresos (income) superiores a ">50K" es mayor que los contribuyentes cuyos ingresos son "<50K".
8. (2 puntos)Cuál es la media de años de educación (educationNum) de aquellos contribuyentes cuyo país de origen no es United-States.
9. (2 puntos, 1 cada sub-apartado) Dada la clase 'Contribuyente' (viene en el código fuente del que se parte) y que mapea cada columna del csv, se pide que se cree su companion object y se definan las funciones dentro de este companion object:
 - La función **imprimeDatos** que tome como parámetro de entrada un objeto de tipo Contribuyente y muestre por consola (*println*) los atributos del parámetro de entrada en el siguiente formato: "\$workclass - \$occupation - \$nativeCountry - \$income". Y cuya firma sea:

```
def imprimeDatos(c: Contribuyente): Unit = ???
```

- **apply**, que no reciba ningún parámetro y que devuelva una instancia de la clase Contribuyente con aquellos campos que sean del tipo Int inicializados a -1 y los del tipo String inicializado a "desconocido". Además, cuya firma sea:

```
def apply(): Contribuyente = ???
```

10. (0.5 puntos) Hacer que el objeto *AnalisisExploratorio* extienda del trait *Analizador* (provisto en el código) en lugar de *App*.

11. (1 punto) Al extender Analizador, es necesario que se implemente la función *imprimeContribuyente*, esta función tiene que imprimir cada instancia de Contribuyente que contenga la colección que recibe por parámetro, con el siguiente formato: "\$workclass - \$occupation - \$nativeCountry - \$income". Se invita a hacer uso de la función definida en el ejercicio 9.1
12. (0.5 punto) Llama a la función *imprimeContribuyente* al final del programa pasándole los 5 primeros elementos de *dataset*

Entrega:

Se enviará el contenido de la carpeta **src** del proyecto con la implementación de las funcionalidades pedidas para cada apartado en un fichero zip cuyo nombre tiene que seguir el siguiente formato: PrimerApellidoSegundoApellidoNombre.zip, por ejemplo: FloresEspinozaCharles.zip

Hay 12 puntos en total en todos los apartados, pero la evaluación será sobre 10.