Pácticas de Aprendizaje Automático con Julia

Alfredo Sánchez Alberca
asalber@ceu.es
https://aprendeconalf.es

Tabla de contenidos

Prefacio

¡Bienvenido a Prácticas de Aprendizaje Automático con Julia!

Este libro presenta una recopilación de prácticas de Aprendizaje Automático (Machine Learning) con el lenguaje de programación Julia.

No es un libro para aprender a programar con Julia, ya que solo enseña el uso del lenguaje y de algunos de sus paquetes para implementar los algoritmos más comunes de Aprendizaje Automático. Para quienes estén interesados en aprender a programar en este Julia, os recomiendo leer este manual de Julia.

Licencia

Nada en esta licencia menoscaba o restringe los derechos morales del autor.

1 Introducción

La gran potencia de cálculo alcanzada por los ordenadores en las últimas décadas ha convertido a los mismos en poderosas herramientas al servicio de todas aquellas disciplinas que, como las matemáticas, requieren cálculos largos y complejos.

Julia es un lenguaje de programación especialmente orientado al cálculo numérico y el análisis de datos. Julia permite además realizar cálculos simbólicos y dispone de una gran biblioteca de paquetes con aplicaciones en muy diversas áreas de las Matemáticas como Cálculo, Álgebra, Geometría, Matemática Discreta o Estadística.

La ventaja de Julia frente a otros programas habituales de cálculo como Mathematica, MATLAB o Sage radica en su potencia de cálculo y su velocidad (equiparable al lenguaje C), lo que lo hace ideal para manejar grandes volúmenes de datos o realizar tareas que requieran largos y complejos cálculos. Además, es software libre por lo que resulta ideal para introducirlo en el aula como soporte computacional para los modelos matemáticos sin coste alguno.

En el siguiente enlace se explica el procedimiento de instalación de Julia.

Existen también varios entornos de desarrollo online que permiten ejecutar código en Julia sin necesidad de instalarlo en nuestro ordenador, como por ejemplo Replit, Cocalc o Codeanywhere.

El objetivo de esta práctica es introducir al alumno en la utilización de este lenguaje, enseñándole a realizar las operaciones básicas más habituales en Cálculo.

1.1 El REPL de Julia

Para arrancar el REPL^(REPL es el acrónimo de Read, Evaluate, Print and Loop, que describe el funcionamiento del compilador de Julia) de julia basta con abrir una terminal y teclear `julia`.

```
prompt> julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.7.3 (2022-05-06)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia>
```

1.2 El gestor de paquetes de Julia

Julia viene con varios paquetes básicos preinstalados, como por ejemplo el paquete `LinearAlgebra` que define funciones básicas del Álgebra Lineal, pero en estas prácticas utilizaremos otros muchos paquetes que añaden más funcionalidades que no vienen instalados por defecto y tendremos que instalarlos aparte. Julia tiene un potente gestor de paquetes que facilita la búsqueda, instalación, actualización y eliminación de paquetes.

Por defecto el gestor de paquetes utiliza el repositorio de paquetes oficial pero se pueden instalar paquetes de otros repositorios.

Para entrar en el modo de gestión de paquetes hay que teclear `]`. Esto produce un cambio en el *prompt* del REPL de Julia.

Los comandos más habituales son:

- `add p`: Instala el paquete `p` en el entorno activo de Julia.
- `update`: Actualiza los paquetes del entorno activo de Julia.
- `status`: Muestra los paquetes instalados y sus versiones en el entorno activo de Julia.

- `remove p`: Elimina el paquete `p` del entorno activo de Julia.

> **Ejemplo**
>
> Para instalar el paquete `SymPy` para cálculo simbólico basta con teclear `add Sympy`.

```
(@v1.7) pkg> add SymPy
    Updating registry at `~/.julia/registries/General.toml`
   Resolving package versions...
    Updating `~/.julia/environments/v1.7/Project.toml`
  [24249f21] + SymPy v1.1.6
    Updating `~/.julia/environments/v1.7/Manifest.toml`
  [3709ef60] + CommonEq v0.2.0
  [38540f10] + CommonSolve v0.2.1
  [438e738f] + PyCall v1.93.1
  [24249f21] + SymPy v1.1.6
```

1.3 Operadores aritméticos.

El uso más simple de Julia es la realización de operaciones aritméticas como en una calculadora. En Julia se utilizan los siguientes operadores.

Operador	Descripción
`x + y`	Suma
`x - y`	Resta
`x * y`	Producto
`x / y`	División
`x ÷ y`	Cociente división entera
`x % y`	Resto división entera
`x ^ y`	Potencia

1.4 Operadores de comparación

Operador	Descripción
`==`	Igualdad
`!=`,	Desigualdad
`<`	Menor que
`<=`,	Menor o igual que

Operador	Descripción
`>`	Mayor que
`>=`,	Mayor o igual que

1.5 Operadores booleanos

Operador	Descripción
`!x`	Negación
`x && y`	Conjunción (y)
`x || y`	Disyunción (o)

Existen también un montón de funciones predefinidas habituales en Cálculo.

1.6 Funciones de redondeo

Función	Descripción
`round(x)`	Devuelve el entero más próximo a `x`
`round(x, digits = n)`	Devuelve al valor más próximo a `x` con `n` decimales
`floor(x)`	Redondea `x` al próximo entero menor
`ceil(x)`	Redondea `x` al próximo entero mayor
`trunc(x)`	Devuelve la parte entera de `x`

Ejemplo

```
julia> round(2.7)
3.0

julia> floor(2.7)
2.0

julia> floor(-2.7)
-3.0

julia> ceil(2.7)
3.0

julia> ceil(-2.7)
-2.0

julia> trunc(2.7)
2.0

julia> trunc(-2.7)
-2.0

julia> round(2.5)
2.0

julia> round(2.786, digits = 2)
2.79
```

1.7 Funciones de división

Función	Descripción
`div(x,y)`, `x÷y`	Cociente de la división entera
`fld(x,y)`	Cociente de la división entera redondeado hacia abajo
`cld(x,y)`	Cociente de la división entera redondeado hacia arriba
`rem(x,y)`, `x%y`	Resto de la división entera. Se cumple `x == div(x,y)*y + rem(x,y)`
`mod(x,y)`	Módulo con respecto a `y`. Se cumple `x == fld(x,y)*y + mod(x,y)`
`gcd(x,y...)`	Máximo común divisor positivo de `x`, `y`,...
`lcm(x,y...)`	Mínimo común múltiplo positivo de `x`, `y`,...

Ejemplo

```
julia> div(5,3)
1

julia> cld(5,3)
2

julia> 5%3
2

julia> -5%3
-2

julia> mod(5,3)
2

julia> mod(-5,3)
1

julia> gcd(12,18)
6

julia> lcm(12,18)
36
```

1.8 Funciones para el signo y el valor absoluto

Función	Descripción
`abs(x)`	Valor absoluto de `x`
`sign(x)`	Devuelve -1 si `x` es positivo, -1 si es negativo y 0 si es 0.

Ejemplo

```
julia> abs(2.5)
2.5

julia> abs(-2.5)
2.5

julia> sign(-2.5)
-1.0

julia> sign(0)
0

julia> sign(2.5)
1.0
```

1.9 Raíces, exponenciales y logaritmos

Función	Descripción
`sqrt(x)`, √x	Raíz cuadrada de `x`
`cbrt(x)`, ∛x	Raíz cúbica de `x`
`exp(x)`	Exponencial de `x`
`log(x)`	Logaritmo neperiano de `x`
`log(b,x)`	Logaritmo en base `b` de `x`
`log2(x)`	Logaritmo en base 2 de `x`
`log10(x)`	Logaritmo en base 10 de `x`

Ejemplo

```
julia> sqrt(4)
2.0

julia> cbrt(27)
3.0

julia> exp(1)
2.718281828459045

julia> exp(-Inf)
0.0

julia> log(1)
0.0

julia> log(0)
-Inf

julia> log(-1)
ERROR: DomainError with -1.0:
log will only return a complex result if called with a complex argument.
...

julia> log(-1+0im)
0.0 + 3.141592653589793im

julia> log2(2^3)
3.0
```

1.10 Funciones trigonométricas

Función	Descripción
`hypot(x,y)`	Hipotenusa del triángulo rectángulo con catetos `x` e `y`
`sin(x)`	Seno del ángulo `x` en radianes
`sind(x)`	Seno del ángulo `x` en grados
`cos(x)`	Coseno del ángulo `x` en radianes
`cosd(x)`	Coseno del ángulo `x` en grados
`tan(x)`	Tangente del ángulo `x` en radianes

Función	Descripción
`tand(x)`	Tangente del ángulo `x` en grados
`sec(x)`	Secante del ángulo `x` en radianes
`csc(x)`	Cosecante del ángulo `x` en radianes
`cot(x)`	Cotangente del ángulo `x` en radianes

Ejemplo

```
julia> sin( /2)
1.0

julia> cos( /2)
6.123233995736766e-17

julia> cosd(90)
0.0

julia> tan( /4)
0.9999999999999999

julia> tand(45)
1.0

julia> tan( /2)
1.633123935319537e16

julia> tand(90)
Inf

julia> sin( /4)^2 + cos( /4)^2
1.0
```

1.11 Funciones trigonométricas inversas

Función	Descripción
`asin(x)`	Arcoseno (inversa del seno) de `x` en radianes
`asind(x)`	Arcoseno (inversa del seno) de `x` en grados
`acos(x)`	Arcocoseno (inversa del coseno) de `x` en radianes
`acosd(x)`	Arcocoseno (inversa del coseno) de `x` en grados

Función	Descripción
`atan(x)`	Arcotangente (inversa de la tangente) de `x` en radianes
`atand(x)`	Arcotangente (inversa de la tangente) de `x` en grados
`asec(x)`	Arcosecante (inversa de la secante) de `x` en radianes
`acsc(x)`	Arcocosecante (inversa de la cosecante) de `x` en radianes
`acot(x)`	Arcocotangente (inversa de la cotangente) de `x` en radianes

Ejemplo

```
julia> asin(1)
1.5707963267948966

julia> asind(1)
90.0

julia> acos(-1)
3.141592653589793

julia> atan(1)
0.7853981633974483

julia> atand(tan( /4))
45.0
```

1.12 Precedencia de operadores

A la hora de evaluar una expresión aritmética, Julia evalúa los operadores según el siguiente orden de prioridad (de mayor a menor prioridad).

Categoría	Operadores	Asociatividad
Funciones	`exp`, `log`, `sin`, etc.	
Exponenciación	`^`	Derecha
Unarios	`+ - √`	Derecha
Fracciones	`//`	Izquierda
Multiplicación	`* / % & \ ÷`	Izquierda
Adición	`+ - |`	Izquierda
Comparaciones	`> < >= <= == != !==`	
Asignaciones	`= += -= *= /= //= ^= ÷= %= |= &=`	Derecha

Cuando se quiera evaluar un operador con menor prioridad antes que otro con mayor prioridad, hay que utilizar paréntesis.

Ejemplo

```
julia> 1 + 4 ^ 2 / 2 - 3
6.0

julia> (1 + 4 ^ 2) / 2 - 3
5.5

julia> (1 + 4) ^ 2 / 2 - 3
9.5

julia> 1 + 4 ^ 2 / (2 - 3)
-15.0

julia> (1 + 4 ^ 2) / (2 - 3)
-17.0
```

1.13 Definición de variables

Para definir variables se pueden utilizar cualquier carácter Unicode. Los nombres de las variables pueden contener más de una letra y, en tal caso, pueden usarse también números, pero siempre debe comenzar por una letra. Así, para Julia, la expresión `xy`, no se interpreta como el producto de la variable x por la variable y, sino como la variable xy. Además, se distingue entre mayúsculas y minúsculas, así que no es lo mismo xy que xY.

2 Preprocesamiento de datos

Esta práctica contiene ejercicios que muestran como preprocesar un conjunto de datos con Julia. El preprocesamiento de datos es una tarea fundamental en la construcción de modelos de aprendizaje automático que consiste en la limpieza, transformación y preparación de los datos para que puedan alimentar el proceso de entrenamiento de los modelos, así como para la evaluación de su rendimiento. El preprocesamiento de datos incluye tareas como

- Limpieza de datos.
- Imputación de valores perdidos.
- Recodificación de variables.
- Creación de nuevas variables.
- Transformación de variables.
- Selección de variables.
- Fusión de datos.
- Reestructuración del conjunto de datos.
- División del conjunto de datos en subconjuntos de entrenamiento y prueba.

2.1 Ejercicios Resueltos

Para la realización de esta práctica se requieren los siguientes paquetes:

```
using CSV  # Para la lectura de archivos CSV.
using DataFrames  # Para el manejo de datos tabulares.
using PrettyTables  # Para mostrar tablas formateadas.
using Plots  # Para el dibujo de gráficas.
using Makie  # Para obtener gráficos interactivos.
```

Ejercicio 2.1. La siguiente tabla contiene los ingresos y gastos de una empresa durante el primer trimestre del año.

a. Crear un data frame con los datos de la tabla.

> **Ayuda**
>
> Utilizar la función `DataFrame` del paquete `DataFrames` para partir el rango de valores en intervalos y asociar a cada intervalo una categoría.

Solución

```
using DataFrames
df = DataFrame(
    Mes = ["Enero", "Febrero", "Marzo", "Abril"],
    Ingresos = [45000, 41500, 51200, 49700],
    Gastos = [33400, 35400, 35600, 36300],
    Impuestos = [6450, 6300, 7100, 6850]
    )
```

	Mes	Ingresos	Gastos	Impuestos
	String	Int64	Int64	Int64
1	Enero	45000	33400	6450
2	Febrero	41500	35400	6300
3	Marzo	51200	35600	7100
4	Abril	49700	36300	6850

b. Crear una nueva columna con los beneficios de cada mes (ingresos - gastos - impuestos).

Solución

```
df.Beneficios = df.Ingresos - df.Gastos - df.Impuestos
df
```

	Mes	Ingresos	Gastos	Impuestos	Beneficios
	String	Int64	Int64	Int64	Int64
1	Enero	45000	33400	6450	5150
2	Febrero	41500	35400	6300	-200
3	Marzo	51200	35600	7100	8500
4	Abril	49700	36300	6850	6550

c. Crear una nueva columna con la variable `Balance` con dos posibles categorías: `positivo` si ha habido beneficios y `negativo` si ha habido pérdidas.

Solución

```
df.Balance = ifelse.(df.Beneficios .> 0, "positivo", "negativo")
df
```

	Mes	Ingresos	Gastos	Impuestos	Beneficios	Balance
	String	Int64	Int64	Int64	Int64	String
1	Enero	45000	33400	6450	5150	positivo
2	Febrero	41500	35400	6300	-200	negativo
3	Marzo	51200	35600	7100	8500	positivo
4	Abril	49700	36300	6850	6550	positivo

d. Filtrar el conjunto de datos para quedarse con los nombres de los meses y los beneficios de los meses con balance positivo.

Solución

```
df[df.Balance .== "positivo", [:Mes, :Beneficios]]
```

	Mes	Beneficios
	String	Int64
1	Enero	5150
2	Marzo	8500
3	Abril	6550

Ejercicio 2.2. El fichero `colesterol.csv` contiene información de una muestra de pacientes donde se han medido la edad, el sexo, el peso, la altura y el nivel de colesterol, además de su nombre.

a. Crear un data frame con los datos de todos los pacientes del estudio a partir del fichero `colesterol.csv`.

Ayuda

Utilizar la función `CSV.read` del paquete `CSV` para partir el rango de valores en intervalos y asociar a cada intervalo una categoría.

Solución

```
using CSV
df = CSV.read("datos/colesterol.csv", DataFrame)
```

	nombre	edad	sexo	peso	altura	colesterol
	String	Int64	String1	Float64?	Float64	Float64?
1	José Luis Martínez Izquierdo	18	H	85.0	1.79	182.0
2	Rosa Díaz Díaz	32	M	65.0	1.73	232.0
3	Javier García Sánchez	24	H	*missing*	1.81	191.0
4	Carmen López Pinzón	35	M	65.0	1.7	200.0
5	Marisa López Collado	46	M	51.0	1.58	148.0
6	Antonio Ruiz Cruz	68	H	66.0	1.74	249.0
7	Antonio Fernández Ocaña	51	H	62.0	1.72	276.0
8	Pilar Martín González	22	M	60.0	1.66	*missing*
9	Pedro Gálvez Tenorio	35	H	90.0	1.94	241.0
10	Santiago Reillo Manzano	46	H	75.0	1.85	280.0
11	Macarena Álvarez Luna	53	M	55.0	1.62	262.0
12	José María de la Guía Sanz	58	H	78.0	1.87	198.0
13	Miguel Angel Cuadrado Gutiérrez	27	H	109.0	1.98	210.0
14	Carolina Rubio Moreno	20	M	61.0	1.77	194.0

b. Crear una nueva columna con el índice de masa corporal, usando la siguiente fórmula

$$\text{IMC} = \frac{\text{Peso (kg)}}{\text{Altura (cm)}^2}$$

Solución

```
df.imc = df.peso ./ (df.altura .^ 2)
df
```

	nombre	edad	sexo	peso	altura	colesterol	
	String	Int64	String1	Float64?	Float64	Float64?	
1	José Luis Martínez Izquierdo	18	H	85.0	1.79	182.0	...
2	Rosa Díaz Díaz	32	M	65.0	1.73	232.0	...
3	Javier García Sánchez	24	H	*missing*	1.81	191.0	...
4	Carmen López Pinzón	35	M	65.0	1.7	200.0	...
5	Marisa López Collado	46	M	51.0	1.58	148.0	...
6	Antonio Ruiz Cruz	68	H	66.0	1.74	249.0	...
7	Antonio Fernández Ocaña	51	H	62.0	1.72	276.0	...
8	Pilar Martín González	22	M	60.0	1.66	*missing*	...
9	Pedro Gálvez Tenorio	35	H	90.0	1.94	241.0	...
10	Santiago Reillo Manzano	46	H	75.0	1.85	280.0	...
11	Macarena Álvarez Luna	53	M	55.0	1.62	262.0	...
12	José María de la Guía Sanz	58	H	78.0	1.87	198.0	...
13	Miguel Angel Cuadrado Gutiérrez	27	H	109.0	1.98	210.0	...
14	Carolina Rubio Moreno	20	M	61.0	1.77	194.0	...

c. Crear una nueva columna con la variable `obesidad` recodificando la columna `imc` en las siguientes categorías.

Rango IMC	Categoría
Menor de 18.5	Bajo peso
De 18.5 a 24.5	Saludable
De 24.5 a 30	Sobrepeso
Mayor de 30	Obeso

Ayuda

Utilizar la función `cut` del paquete `CategoricalArrays` para partir el rango de valores en intervalos y asociar a cada intervalo una categoría.

Solución

```
using CategoricalArrays
df.obesidad = cut(df.imc, [0, 18.5, 24.5, 30, Inf],
                  labels=["Bajo peso", "Saludable", "Sobrepeso", "Obeso"],
                  extend=true)
df
```

	nombre	edad	sexo	peso	altura	colesterol	
	String	Int64	String1	Float64?	Float64	Float64?	
1	José Luis Martínez Izquierdo	18	H	85.0	1.79	182.0	...
2	Rosa Díaz Díaz	32	M	65.0	1.73	232.0	...
3	Javier García Sánchez	24	H	*missing*	1.81	191.0	...
4	Carmen López Pinzón	35	M	65.0	1.7	200.0	...
5	Marisa López Collado	46	M	51.0	1.58	148.0	...
6	Antonio Ruiz Cruz	68	H	66.0	1.74	249.0	...
7	Antonio Fernández Ocaña	51	H	62.0	1.72	276.0	...
8	Pilar Martín González	22	M	60.0	1.66	*missing*	...
9	Pedro Gálvez Tenorio	35	H	90.0	1.94	241.0	...
10	Santiago Reillo Manzano	46	H	75.0	1.85	280.0	...
11	Macarena Álvarez Luna	53	M	55.0	1.62	262.0	...
12	José María de la Guía Sanz	58	H	78.0	1.87	198.0	...
13	Miguel Angel Cuadrado Gutiérrez	27	H	109.0	1.98	210.0	...
14	Carolina Rubio Moreno	20	M	61.0	1.77	194.0	...

d. Seleccionar las columnas `nombre`, `sexo` y `edad`.

Solución

```
df[:, [:nombre, :sexo, :edad]]
```

	nombre	sexo	edad
	String	String1	Int64
1	José Luis Martínez Izquierdo	H	18
2	Rosa Díaz Díaz	M	32
3	Javier García Sánchez	H	24
4	Carmen López Pinzón	M	35
5	Marisa López Collado	M	46
6	Antonio Ruiz Cruz	H	68
7	Antonio Fernández Ocaña	H	51
8	Pilar Martín González	M	22
9	Pedro Gálvez Tenorio	H	35
10	Santiago Reillo Manzano	H	46
11	Macarena Álvarez Luna	M	53
12	José María de la Guía Sanz	H	58
13	Miguel Angel Cuadrado Gutiérrez	H	27
14	Carolina Rubio Moreno	M	20

e. Anonimizar los datos eliminando la columna `nombre`.

> **Ayuda**
>
> Utilizar la función `select` del paquete `DataFrames` para seleccionar las columnas deseadas y eliminar las columnas no deseadas. Existe también la función `select!` que modifica el data frame original eliminando las columnas no seleccionadas.

> **Solución**

```
select(df, Not(:nombre))
```

	edad	sexo	peso	altura	colesterol	imc	obesidad
	Int64	String1	Float64?	Float64	Float64?	Float64?	Cat…?
1	18	H	85.0	1.79	182.0	26.5285	Sobrepeso
2	32	M	65.0	1.73	232.0	21.7181	Saludable
3	24	H	*missing*	1.81	191.0	*missing*	*missing*
4	35	M	65.0	1.7	200.0	22.4913	Saludable
5	46	M	51.0	1.58	148.0	20.4294	Saludable
6	68	H	66.0	1.74	249.0	21.7994	Saludable
7	51	H	62.0	1.72	276.0	20.9573	Saludable
8	22	M	60.0	1.66	*missing*	21.7738	Saludable
9	35	H	90.0	1.94	241.0	23.9133	Saludable
10	46	H	75.0	1.85	280.0	21.9138	Saludable
11	53	M	55.0	1.62	262.0	20.9572	Saludable
12	58	H	78.0	1.87	198.0	22.3055	Saludable
13	27	H	109.0	1.98	210.0	27.8033	Sobrepeso
14	20	M	61.0	1.77	194.0	19.4708	Saludable

f. Reordenar las columnas poniendo la columna `sexo` antes que la columna `edad`.

> **Solución**

```
select(df, Cols(:sexo, :edad, Not(:sexo, :edad)))
```

	sexo	edad	nombre	peso	altura	colesterol	
	String1	Int64	String	Float64?	Float64	Float64?	
1	H	18	José Luis Martínez Izquierdo	85.0	1.79	182.0	...
2	M	32	Rosa Díaz Díaz	65.0	1.73	232.0	...
3	H	24	Javier García Sánchez	*missing*	1.81	191.0	...
4	M	35	Carmen López Pinzón	65.0	1.7	200.0	...
5	M	46	Marisa López Collado	51.0	1.58	148.0	...
6	H	68	Antonio Ruiz Cruz	66.0	1.74	249.0	...
7	H	51	Antonio Fernández Ocaña	62.0	1.72	276.0	...
8	M	22	Pilar Martín González	60.0	1.66	*missing*	...
9	H	35	Pedro Gálvez Tenorio	90.0	1.94	241.0	...
10	H	46	Santiago Reillo Manzano	75.0	1.85	280.0	...
11	M	53	Macarena Álvarez Luna	55.0	1.62	262.0	...
12	H	58	José María de la Guía Sanz	78.0	1.87	198.0	...
13	H	27	Miguel Angel Cuadrado Gutiérrez	109.0	1.98	210.0	...
14	M	20	Carolina Rubio Moreno	61.0	1.77	194.0	...

g. Filtrar el data frame para quedarse con las mujeres.

Solución

```
df[df.sexo .== "M", :]
```

	nombre	edad	sexo	peso	altura	colesterol	imc	
	String	Int64	String1	Float64?	Float64	Float64?	Float64?	
1	Rosa Díaz Díaz	32	M	65.0	1.73	232.0	21.7181	...
2	Carmen López Pinzón	35	M	65.0	1.7	200.0	22.4913	...
3	Marisa López Collado	46	M	51.0	1.58	148.0	20.4294	...
4	Pilar Martín González	22	M	60.0	1.66	*missing*	21.7738	...
5	Macarena Álvarez Luna	53	M	55.0	1.62	262.0	20.9572	...
6	Carolina Rubio Moreno	20	M	61.0	1.77	194.0	19.4708	...

h. Filtrar el data frame para quedarse con los hombres mayores de 30 años.

Solución

```
df[(df.sexo .== "H") .& (df.edad .> 30), :]
```

	nombre	edad	sexo	peso	altura	colesterol	imc	
	String	Int64	String1	Float64?	Float64	Float64?	Float64?	
1	Antonio Ruiz Cruz	68	H	66.0	1.74	249.0	21.7994	..
2	Antonio Fernández Ocaña	51	H	62.0	1.72	276.0	20.9573	..
3	Pedro Gálvez Tenorio	35	H	90.0	1.94	241.0	23.9133	..
4	Santiago Reillo Manzano	46	H	75.0	1.85	280.0	21.9138	..
5	José María de la Guía Sanz	58	H	78.0	1.87	198.0	22.3055	..

i. Filtrar el data frame para quedarse con las filas sin valores perdidos.

> **Ayuda**
>
> Utilizar la función `dropmissing` del paquete `DataFrames` para eliminar las filas con valores perdidos.

Solución

```
dropmissing(df)
```

	nombre	edad	sexo	peso	altura	colesterol	imc
	String	Int64	String1	Float64	Float64	Float64	Float6
1	José Luis Martínez Izquierdo	18	H	85.0	1.79	182.0	26.528
2	Rosa Díaz Díaz	32	M	65.0	1.73	232.0	21.718
3	Carmen López Pinzón	35	M	65.0	1.7	200.0	22.491
4	Marisa López Collado	46	M	51.0	1.58	148.0	20.429
5	Antonio Ruiz Cruz	68	H	66.0	1.74	249.0	21.799
6	Antonio Fernández Ocaña	51	H	62.0	1.72	276.0	20.957
7	Pedro Gálvez Tenorio	35	H	90.0	1.94	241.0	23.913
8	Santiago Reillo Manzano	46	H	75.0	1.85	280.0	21.913
9	Macarena Álvarez Luna	53	M	55.0	1.62	262.0	20.957
10	José María de la Guía Sanz	58	H	78.0	1.87	198.0	22.305
11	Miguel Angel Cuadrado Gutiérrez	27	H	109.0	1.98	210.0	27.803
12	Carolina Rubio Moreno	20	M	61.0	1.77	194.0	19.470

j. Filtrar el data frame para eliminar las filas con datos perdidos en la columna `colesterol`.

> **Ayuda**
>
> Utilizar la función `dropmissing`, col donde `col` es el nombre de la columna que contiene los valores perdidos.

Solución

```
dropmissing(df, :colesterol)
```

	nombre	edad	sexo	peso	altura	colesterol	
	String	Int64	String1	Float64?	Float64	Float64	
1	José Luis Martínez Izquierdo	18	H	85.0	1.79	182.0	...
2	Rosa Díaz Díaz	32	M	65.0	1.73	232.0	...
3	Javier García Sánchez	24	H	*missing*	1.81	191.0	...
4	Carmen López Pinzón	35	M	65.0	1.7	200.0	...
5	Marisa López Collado	46	M	51.0	1.58	148.0	...
6	Antonio Ruiz Cruz	68	H	66.0	1.74	249.0	...
7	Antonio Fernández Ocaña	51	H	62.0	1.72	276.0	...
8	Pedro Gálvez Tenorio	35	H	90.0	1.94	241.0	...
9	Santiago Reillo Manzano	46	H	75.0	1.85	280.0	...
10	Macarena Álvarez Luna	53	M	55.0	1.62	262.0	...
11	José María de la Guía Sanz	58	H	78.0	1.87	198.0	...
12	Miguel Angel Cuadrado Gutiérrez	27	H	109.0	1.98	210.0	...
13	Carolina Rubio Moreno	20	M	61.0	1.77	194.0	...

k. Imputar los valores perdidos en la columna `colesterol` con la media de los valores no perdidos.

Ayuda

Utilizar la función `coalesce` para reemplazar los valores perdidos por otros valores.

Solución

```
using Statistics
media_colesterol = mean(skipmissing(df.colesterol))
df.colesterol = coalesce.(df.colesterol, media_colesterol)
df
```

	nombre	edad	sexo	peso	altura	colesterol	
	String	Int64	String1	Float64?	Float64	Float64	
1	José Luis Martínez Izquierdo	18	H	85.0	1.79	182.0	...
2	Rosa Díaz Díaz	32	M	65.0	1.73	232.0	...
3	Javier García Sánchez	24	H	*missing*	1.81	191.0	...
4	Carmen López Pinzón	35	M	65.0	1.7	200.0	...
5	Marisa López Collado	46	M	51.0	1.58	148.0	...
6	Antonio Ruiz Cruz	68	H	66.0	1.74	249.0	...
7	Antonio Fernández Ocaña	51	H	62.0	1.72	276.0	...
8	Pilar Martín González	22	M	60.0	1.66	220.231	...
9	Pedro Gálvez Tenorio	35	H	90.0	1.94	241.0	...
10	Santiago Reillo Manzano	46	H	75.0	1.85	280.0	...
11	Macarena Álvarez Luna	53	M	55.0	1.62	262.0	...
12	José María de la Guía Sanz	58	H	78.0	1.87	198.0	...
13	Miguel Angel Cuadrado Gutiérrez	27	H	109.0	1.98	210.0	...
14	Carolina Rubio Moreno	20	M	61.0	1.77	194.0	...

l. Ordenar el data frame según la columna `nombre`.

Ayuda

Utilizar la función `sort` para ordenar las filas del data frame según los valores de una o varias columnas. Utilizar el parámetro `rev` para especificar mediante un vector de booleanos si el orden es ascendente o descendente.

Solución

```
sort(df, :nombre)
```

	nombre	edad	sexo	peso	altura	colesterol	
	String	Int64	String1	Float64?	Float64	Float64	
1	Antonio Fernández Ocaña	51	H	62.0	1.72	276.0	...
2	Antonio Ruiz Cruz	68	H	66.0	1.74	249.0	...
3	Carmen López Pinzón	35	M	65.0	1.7	200.0	...
4	Carolina Rubio Moreno	20	M	61.0	1.77	194.0	...
5	Javier García Sánchez	24	H	*missing*	1.81	191.0	...
6	José Luis Martínez Izquierdo	18	H	85.0	1.79	182.0	...
7	José María de la Guía Sanz	58	H	78.0	1.87	198.0	...
8	Macarena Álvarez Luna	53	M	55.0	1.62	262.0	...
9	Marisa López Collado	46	M	51.0	1.58	148.0	...
10	Miguel Angel Cuadrado Gutiérrez	27	H	109.0	1.98	210.0	...
11	Pedro Gálvez Tenorio	35	H	90.0	1.94	241.0	...
12	Pilar Martín González	22	M	60.0	1.66	220.231	...
13	Rosa Díaz Díaz	32	M	65.0	1.73	232.0	...
14	Santiago Reillo Manzano	46	H	75.0	1.85	280.0	...

m. Ordenar el data frame ascendentemente por la columna `sexo` y descendentemente por la columna `edad`.

Solución

```
sort(df, [:sexo, :edad], rev=[false, true])
```

	nombre	edad	sexo	peso	altura	colesterol	
	String	Int64	String1	Float64?	Float64	Float64	
1	Antonio Ruiz Cruz	68	H	66.0	1.74	249.0	...
2	José María de la Guía Sanz	58	H	78.0	1.87	198.0	...
3	Antonio Fernández Ocaña	51	H	62.0	1.72	276.0	...
4	Santiago Reillo Manzano	46	H	75.0	1.85	280.0	...
5	Pedro Gálvez Tenorio	35	H	90.0	1.94	241.0	...
6	Miguel Angel Cuadrado Gutiérrez	27	H	109.0	1.98	210.0	...
7	Javier García Sánchez	24	H	*missing*	1.81	191.0	...
8	José Luis Martínez Izquierdo	18	H	85.0	1.79	182.0	...
9	Macarena Álvarez Luna	53	M	55.0	1.62	262.0	...
10	Marisa López Collado	46	M	51.0	1.58	148.0	...
11	Carmen López Pinzón	35	M	65.0	1.7	200.0	...
12	Rosa Díaz Díaz	32	M	65.0	1.73	232.0	...
13	Pilar Martín González	22	M	60.0	1.66	220.231	...
14	Carolina Rubio Moreno	20	M	61.0	1.77	194.0	...

Ejercicio 2.3. El fichero `notas-curso2.csv` contiene información de las notas de los alumnos de un curso.

a. Crear un data frame con los datos de los alumnos del curso a partir del fichero notas-curso2.csv.

Solución

```
using CSV, DataFrames
df = CSV.read("datos/notas-curso2.csv", DataFrame; missingstring="NA")
```

	sexo	turno	grupo	trabaja	notaA	notaB	notaC	notaD	notaE
	String7	String7	String1	String1	Float64	Float64?	Float64?	Float64?	Float64
1	Mujer	Tarde	C	N	5.2	6.3	3.4	2.3	2.0
2	Hombre	Mañana	A	N	5.7	5.7	4.2	3.5	2.7
3	Hombre	Mañana	B	N	8.3	8.8	8.8	8.0	5.5
4	Hombre	Mañana	B	N	6.1	6.8	4.0	3.5	2.2
5	Hombre	Mañana	A	N	6.2	9.0	5.0	4.4	3.7
6	Hombre	Mañana	A	S	8.6	8.9	9.5	8.4	3.9
7	Mujer	Mañana	A	N	6.7	7.9	5.6	4.8	4.2
8	Mujer	Tarde	C	S	4.1	5.2	1.7	0.3	1.0
9	Hombre	Tarde	C	N	5.0	5.0	3.3	2.7	6.0
10	Hombre	Tarde	C	N	5.3	6.3	4.8	3.6	2.3
11	Mujer	Mañana	A	N	7.8	*missing*	6.5	6.7	2.8
12	Hombre	Mañana	A	N	6.5	8.0	5.0	3.2	3.3
13	Hombre	Mañana	B	N	6.6	7.6	5.3	4.0	1.0
14	Hombre	Mañana	B	N	6.2	6.7	5.3	4.7	4.7
15	Hombre	Mañana	B	N	5.2	4.1	5.8	5.0	1.9
16	Hombre	Mañana	B	S	8.7	*missing*	7.6	6.3	9.3
17	Mujer	Mañana	B	N	6.7	6.3	6.8	5.3	2.8
18	Mujer	Tarde	C	N	3.1	4.8	2.7	1.8	1.0
19	Hombre	Mañana	B	N	7.1	10.0	5.9	4.9	2.5
20	Hombre	Mañana	B	S	5.3	4.8	2.8	0.9	4.2
21	Hombre	Mañana	A	N	4.4	6.1	2.9	1.9	2.4
22	Mujer	Tarde	C	S	5.7	6.4	4.2	3.3	1.0
23	Mujer	Tarde	C	S	4.5	3.9	3.5	2.6	2.2
24	Mujer	Tarde	C	N	4.9	4.4	5.2	3.7	0.4
25	Hombre	Tarde	C	N	5.1	4.5	6.4	5.6	0.5
26	Hombre	Mañana	B	N	3.5	3.9	4.0	3.7	1.7
27	Mujer	Mañana	A	N	7.3	6.8	6.1	5.4	1.8
28	Hombre	Mañana	B	N	6.2	7.9	3.5	1.9	3.1
29	Mujer	Mañana	A	N	4.3	7.4	2.9	1.7	0.2
30	Hombre	Mañana	B	N	7.9	7.6	7.0	6.1	0.4
...	...	...	...	...	...	...	...	...	...

b. Obtener el número de datos perdidos en cada columna.

Solución

```
describe(df)[:, [:variable, :nmissing]]
```

	variable	nmissing
	Symbol	Int64
1	sexo	0
2	turno	0
3	grupo	0
4	trabaja	0
5	notaA	0
6	notaB	5
7	notaC	1
8	notaD	2
9	notaE	2

c. Recodificar la variable `grupo` en una colección de columnas binarias.

Ayuda

Utilizar la función `onehotbatch` del paquete OneHotArrays para recodificar una variable categórica en una colección de columnas binarias.

Solución

```
using OneHotArrays
codificacion = permutedims(onehotbatch(df.grupo, unique(df.grupo)))
hcat(df, DataFrame(codificacion, :auto))
```

	sexo	turno	grupo	trabaja	notaA	notaB	notaC	notaD	notaE
	String7	String7	String1	String1	Float64	Float64?	Float64?	Float64?	Float64
1	Mujer	Tarde	C	N	5.2	6.3	3.4	2.3	2.0
2	Hombre	Mañana	A	N	5.7	5.7	4.2	3.5	2.7
3	Hombre	Mañana	B	N	8.3	8.8	8.8	8.0	5.5
4	Hombre	Mañana	B	N	6.1	6.8	4.0	3.5	2.2
5	Hombre	Mañana	A	N	6.2	9.0	5.0	4.4	3.7
6	Hombre	Mañana	A	S	8.6	8.9	9.5	8.4	3.9
7	Mujer	Mañana	A	N	6.7	7.9	5.6	4.8	4.2
8	Mujer	Tarde	C	S	4.1	5.2	1.7	0.3	1.0
9	Hombre	Tarde	C	N	5.0	5.0	3.3	2.7	6.0
10	Hombre	Tarde	C	N	5.3	6.3	4.8	3.6	2.3
11	Mujer	Mañana	A	N	7.8	*missing*	6.5	6.7	2.8
12	Hombre	Mañana	A	N	6.5	8.0	5.0	3.2	3.3
13	Hombre	Mañana	B	N	6.6	7.6	5.3	4.0	1.0
14	Hombre	Mañana	B	N	6.2	6.7	5.3	4.7	4.7
15	Hombre	Mañana	B	N	5.2	4.1	5.8	5.0	1.9
16	Hombre	Mañana	B	S	8.7	*missing*	7.6	6.3	9.3
17	Mujer	Mañana	B	N	6.7	6.3	6.8	5.3	2.8
18	Mujer	Tarde	C	N	3.1	4.8	2.7	1.8	1.0
19	Hombre	Mañana	B	N	7.1	10.0	5.9	4.9	2.5
20	Hombre	Mañana	B	S	5.3	4.8	2.8	0.9	4.2
21	Hombre	Mañana	A	N	4.4	6.1	2.9	1.9	2.4
22	Mujer	Tarde	C	S	5.7	6.4	4.2	3.3	1.0
23	Mujer	Tarde	C	S	4.5	3.9	3.5	2.6	2.2
24	Mujer	Tarde	C	N	4.9	4.4	5.2	3.7	0.4
25	Hombre	Tarde	C	N	5.1	4.5	6.4	5.6	0.5
26	Hombre	Mañana	B	N	3.5	3.9	4.0	3.7	1.7
27	Mujer	Mañana	A	N	7.3	6.8	6.1	5.4	1.8
28	Hombre	Mañana	B	N	6.2	7.9	3.5	1.9	3.1
29	Mujer	Mañana	A	N	4.3	7.4	2.9	1.7	0.2
30	Hombre	Mañana	B	N	7.9	7.6	7.0	6.1	0.4
...	...	...	...	...	...	...	...	...	...

2.2 Ejercicios propuestos

Ejercicio 2.4. El fichero `vinos.csv` contiene información sobre las características de una muestra de vinos portugueses de la denominación "Vinho Verde". Las variables que contiene son:

Variable	Descripción	Tipo (unidades)
tipo	Tipo de vino	Categórica (blanco, tinto)
meses.barrica	Mesesde envejecimiento en barrica	Numérica(meses)
acided.fija	Cantidadde ácidotartárico	Numérica(g/dm3)
acided.volatil	Cantidad de ácido acético	Numérica(g/dm3)
acido.citrico	Cantidad de ácidocítrico	Numérica(g/dm3)
azucar.residual	Cantidad de azúcarremanente después de la fermentación	Numérica(g/dm3)
cloruro.sodico	Cantidad de clorurosódico	Numérica(g/dm3)
dioxido.azufre.libre	Cantidad de dióxido de azufreen formalibre	Numérica(mg/dm3)
dioxido.azufre.total	Cantidadde dióxido de azufretotal en forma libre o ligada	Numérica(mg/dm3)
densidad	Densidad	Numérica(g/cm3)
ph	pH	Numérica(0-14)
sulfatos	Cantidadde sulfato de potasio	Numérica(g/dm3)
alcohol	Porcentajede contenidode alcohol	Numérica(0-100)
calidad	Calificación otorgada porun panel de expertos	Numérica(0-10)

a. Crear un data frame con los datos de los vinos a partir del fichero `vinos.csv`.

b. Obtener el número de valores perdidos en cada columna.

c. Imputar los valores perdidos del alcohol con la media de los valores no perdidos para cada tipo de vino.

d. Crear la variable categórica `Envejecimiento` recodificando la variable `meses.barrica` en las siguientes categorías.

Rango en meses	Categoría
Menos de 3	Joven
Entre 3 y 12	Crianza
Entre 12 y 18	Reserva
Más de 18	Gran reserva

e. Crear la variable categórica `Dulzor` recodificando la variable `azucar.residual` en las siguientes categorías.

Rango azúcar	Categoría
Menos de 4	Seco
Más de 4 y menos de 12	Semiseco
Más de 12 y menos de 45	Semidulce
Más de 45	Dulce

f. Filtrar el conjunto de datos para quedarse con los vinos Reserva o Gran Reserva con una calidad superior a 7 y ordenar el data frame por calidad de forma descendente.

g. ¿Cuántos vinos blancos con un contenido en alcohol superior al 12% y una calidad superior a 8 hay en el conjunto de datos?

h. ¿Cuáles son los 10 mejores vinos tintos crianza secos?

3 Regresión

Los modelos de aprendizaje basados en regresión son modelos bastante simples que pueden utilizarse para predecir variables cuantitativas (regresión lineal) o cualitativas (regresión logística). Esta práctica contiene ejercicios que muestran como construir modelos de aprendizaje de regresión lineal y regresión logística con Julia.

3.1 Ejercicios Resueltos

Para la realización de esta práctica se requieren los siguientes paquetes:

```
using CSV  # Para la lectura de archivos CSV.
using DataFrames  # Para el manejo de datos tabulares.
using PrettyTables  # Para mostrar tablas formateadas.
using Plots  # Para el dibujo de gráficas.
using GLMakie  # Para obtener gráficos interactivos.
```

Ejercicio 3.1. El conjunto de datos `viviendas.csv` contiene información sobre el precio de venta de viviendas en una ciudad.

a. Cargar los datos del archivo `viviendas.csv` en un data frame.

Solución

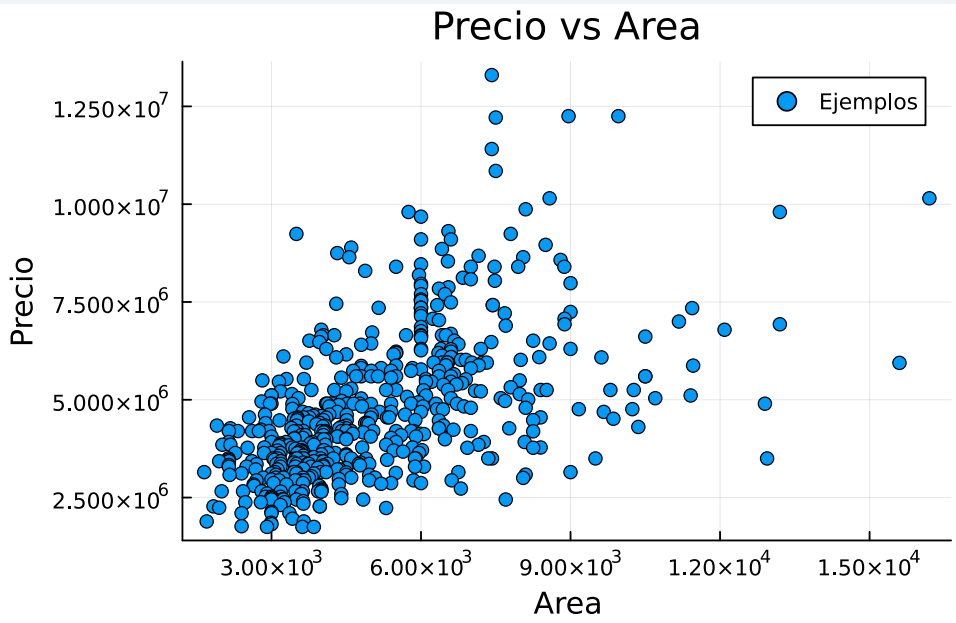
```
using CSV, DataFrames
df = CSV.read("datos/viviendas.csv", DataFrame)
first(df, 5)
```

	precio	area	dormitorios	baños	habitaciones	calleprincipal	huespedes	sotano
	Int64	Int64	Int64	Int64	Int64	String3	String3	String3
1	13300000	7420	4	2	3	si	no	no
2	12250000	8960	4	4	4	si	no	no
3	12250000	9960	3	2	2	si	no	si
4	12215000	7500	4	2	2	si	no	si
5	11410000	7420	4	1	2	si	si	si

b. Dibujar un diagrama de dispersión entre el precio y el area de las viviendas.

Solución

```
using Plots
plt = scatter(df.area, df.precio, xlabel="Area", ylabel="Precio", title="Precio vs
```



c. Definir un modelo lineal que explique el precio en función del área de las viviendas.

Ayuda

Un modelo lineal tiene encuación $y = \theta_1 + \theta_2 x$.

Solución

```
precio(area, ) = [1] .+ [2] * area
```

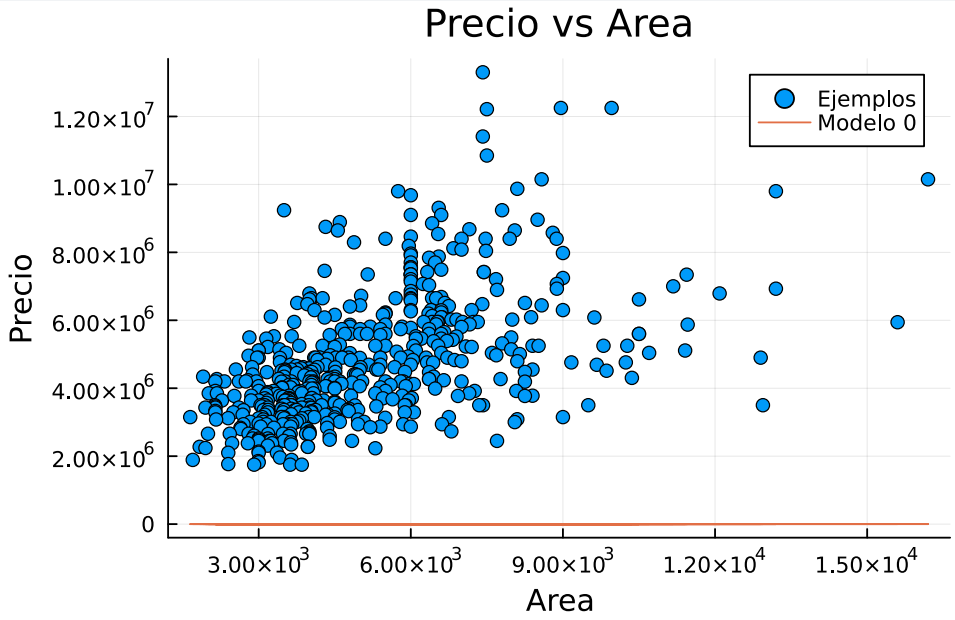
```
precio (generic function with 1 method)
```

Observa que la función precio está vectorizada, lo que significa que puede recibir un vector de áreas y devolver un vector de precios.

d. Inicializar los parámetros del modelo lineal con valores nulos y dibujar el modelo sobre el diagrama de dispersión.

Solución

```
  = [0.0, 0.0]
plot!(df.area, precio(df.area,  ), label = "Modelo 0")
```



e. Definir una función de costo para el modelo lineal y evaluar el coste para el modelo lineal construido con los parámetros iniciales. A la vista del coste obtenido, ¿cómo de bueno es el modelo?

Ayuda

La función de coste para un modelo lineal es el error cuadrático medio.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

donde h_θ es el modelo, $h_\theta(x^{(i)})$ es la predicción del modelo para el ejemplo i-ésimo, $y^{(i)}$ es el valor real observado para el ejemplo i-ésimo, y m es el número de ejemplos.

Solución

```
function coste( , X, Y)
    m = length(Y)
    return sum((precio(X,  ) .- Y).^2) / (2 * m)
end

coste( , df.area, df.precio)
```

```
1.3106916364659266e13
```

La función de coste nos da una medida de lo lejos que están las predicciones del modelo de los valores reales observados. En este caso, el coste es muy alto, lo que indica que el modelo no es bueno.

f. ¿En qué dirección debemos modificar los parámetros del modelo para mejorar el modelo?

Solución

Para minimizar la función de coste, debemos modificar los parámetros del modelo en la dirección opuesta al gradiente de la función de coste, ya que el gradiente de una función indica la dirección de mayor crecimiento de la función.

g. Crear una función para modificar los pesos del modelo lineal mediante el algoritmo del gradiente descendente, y aplicarla a los parámetros actuales tomando una tasa de aprendizaje de 10^{-8}. ¿Cómo han cambiado los parámetros del modelo? Dibujar el modelo actualizado sobre el diagrama de dispersión. ¿Cómo ha cambiado el coste?

Ayuda

El algoritmo del gradiente descendente actualiza los parámetros del modelo de acuerdo a la siguiente regla:

$$\theta_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

donde α es la tasa de aprendizaje y $\frac{\partial J(\theta)}{\partial \theta_j}$ es la derivada parcial de la función de coste con respecto al parámetro θ_j.

Solución

```
function gradiente_descendente!( , X, Y,  )
    # Calculamos el número de ejemplos
    m = length(Y)
    # Actualizamos el término independiente del modelo lineal.
     [1] -=   * sum(precio(X,  ) - Y) / m
    # Actualizamos la pendiente del modelo lineal.
     [2] -=   * sum((precio(X,  ) - Y) .* X) / m
    return
end
```

```
gradiente_descendente! (generic function with 1 method)
```

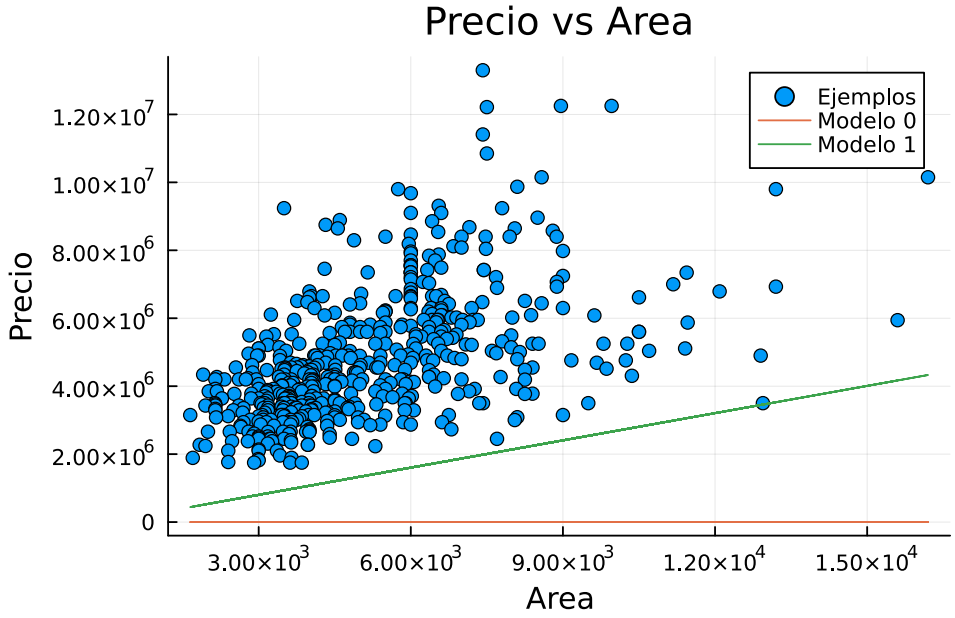
Aplicamos la función a los parámetros del modelo actual y mostramos los nuevos parámetros.

```
gradiente_descendente!( , df.area, df.precio, 1e-8)
```

```
2-element Vector{Float64}:
   0.04766729247706422
 267.22919804579385
```

Dibujamos el nuevo modelo.

```
plot!(df.area, precio(df.area,  ), label = "Modelo 1")
```



Se observa que ahora la recta está más cerca de la nube de puntos, por lo que el modelo ha mejorado. Calculamos el coste del nuevo modelo.

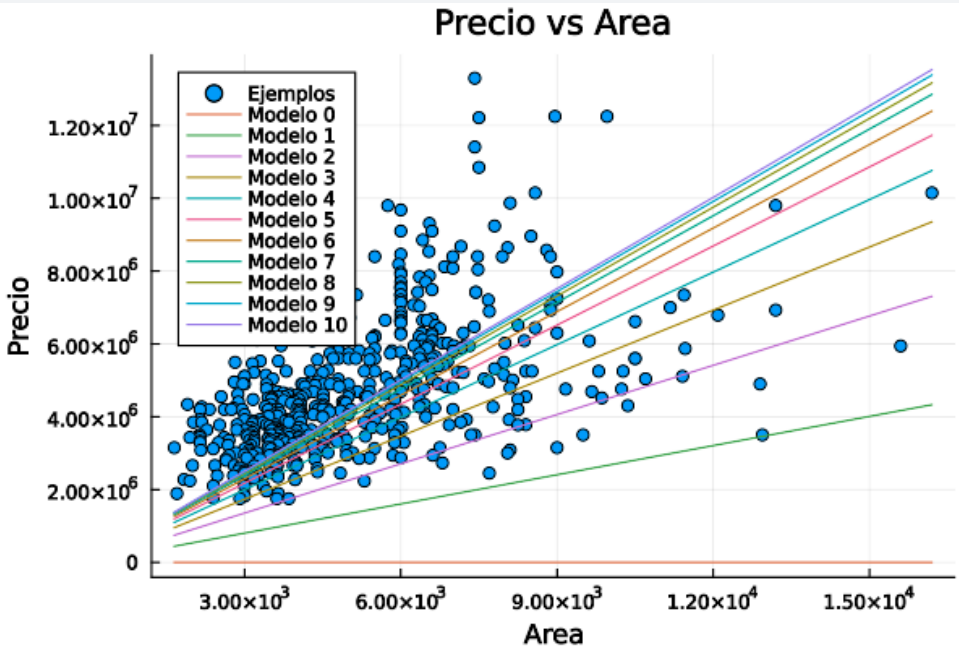
```
coste( , df.area, df.precio)
```

```
7.080823787113201e12
```

h. Repetir el proceso de actualización de los parámetros del modelo mediante el algoritmo del gradiente descendente durante 9 iteraciones más y dibujar los modelos actualizados.

Solución

```
for i = 2:10
    gradiente_descendente!( , df.area, df.precio, 1e-8)
    plot!(df.area, precio(df.area,  ), label = "Modelo $i", legend = true)
end
plt
```



i. Dibujar un gráfico con la evolución del coste del modelo a lo largo de las iteraciones. ¿Cómo se comporta el coste a lo largo de las iteraciones?

Solución

```
costes = Float64[]
for i = 1:10
    gradiente_descendente!( , df.area, df.precio, 1e-8)
    push!(costes, coste( , df.area, df.precio))
end
costes
```

```
10-element Vector{Float64}:
 4.230808760870044e12
 2.882906194020343e12
 2.2454213686913755e12
 1.9439256128790886e12
 1.8013344680594421e12
 1.7338965877160208e12
 1.7020021263374993e12
 1.6869177748236997e12
 1.6797836937723748e12
 1.6764096595632322e12
```

El coste del modelo disminuye en cada iteración, lo que indica que el modelo está mejorando. Esto se debe a que el algoritmo del gradiente descendente modifica los parámetros del modelo en la dirección que minimiza la función de coste.

j. ¿Hasta qué iteración habrá que llegar para conseguir un reducción del coste menor de un 0.0001%?

Solución

```
 = [0.0, 0.0]
costes = [0, coste( , df.area, df.precio)]
i = 1
while abs(costes[end] - costes[end-1]) / costes[end-1] > 0.000001
    i += 1
    gradiente_descendente!( , df.area, df.precio, 1e-8)
    push!(costes, coste( , df.area, df.precio))
end
i
```

```
23
```

En este caso, el algoritmo del gradiente descendente converge en 1000 iteraciones.

k. ¿Qué sucede si se utiliza una tasa de aprendizaje $\alpha = 0.0001$? ¿Cómo afecta al coste y a la convergencia del modelo?

Solución

```
 = [0.0, 0.0]
costes = [coste( , df.area, df.precio)]
for i = 1:10
    gradiente_descendente!( , df.area, df.precio, 0.0001)
    push!(costes, coste( , df.area, df.precio))
end
costes
```

```
11-element Vector{Float64}:
 1.3106916364659266e13
 1.114133369099188e20
 1.0856750832581238e27
 1.05794371802143e34
 1.0309206941949286e41
 1.004587918634273e48
 9.789277603492545e54
 9.539230386975057e61
 9.29557011881276e68
 9.058133657380397e75
 8.826762028174244e82
```

Si la tasa de aprendizaje es demasiado grande, el algoritmo del gradiente descendente puede no converger y el coste puede oscilar en lugar de disminuir. En este caso, el coste aumenta en cada iteración, lo que indica que la tasa de aprendizaje es demasiado grande.

4 Árboles de decisión

Los árboles de decisión son modelos de aprendizaje simples e intuitivos que pueden utilizarse para tanto para predecir variables cuantitativas (regresión) como categóricas (clasificación). Esta práctica contiene ejercicios que muestran como construir modelos de aprendizaje basados en árboles de decisión con Julia.

4.1 Ejercicios Resueltos

Para la realización de esta práctica se requieren los siguientes paquetes:

```
using CSV  # Para la lectura de archivos CSV.
using DataFrames  # Para el manejo de datos tabulares.
using Tidier # Para el preprocesamiento de datos.
using PrettyTables  # Para mostrar tablas formateadas.
using Plots  # Para el dibujo de gráficas.
using GLMakie  # Para obtener gráficos interactivos.
using AlgebraOfGraphics # Para generar gráficos mediante la gramática de gráficos.
using DecisionTree # Para construir árboles de decisión.
using GraphMakie # Para la visualización de árboles de decisión.
```

Ejercicio 4.1. El conjunto de datos `tenis.csv` contiene información sobre las condiciones meteorológicas de varios días y si se pudo jugar al tenis o no.

a. Cargar los datos del archivo `tenis.csv` en un data frame.

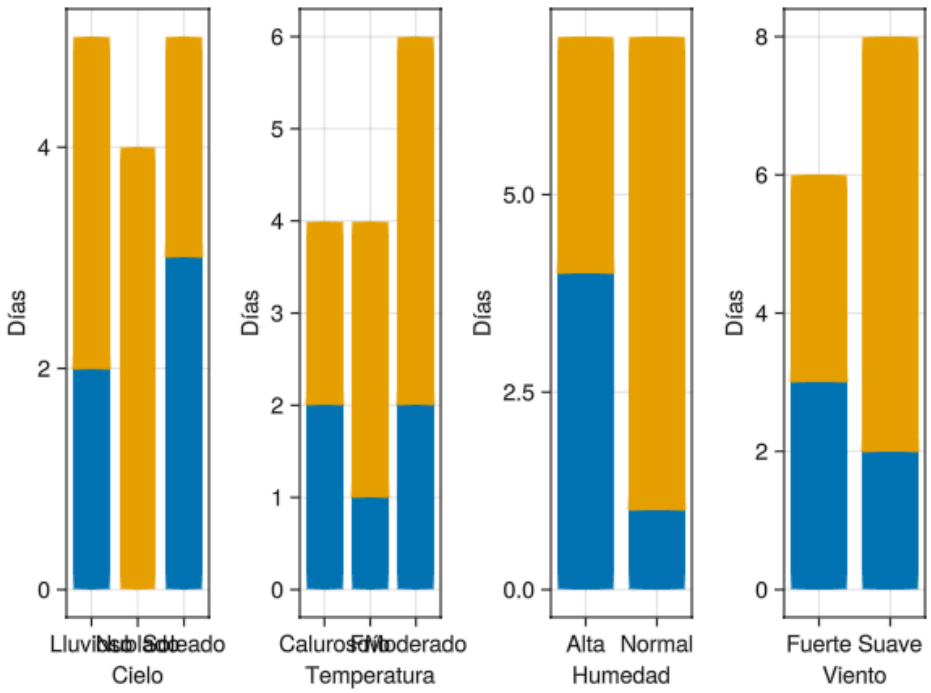
> Solución
>
> ```
> using CSV, DataFrames
> df = CSV.read("datos/tenis.csv", DataFrame)
> ```

	Cielo	Temperatura	Humedad	Viento	Tenis
	String15	String15	String7	String7	String3
1	Soleado	Caluroso	Alta	Suave	No
2	Soleado	Caluroso	Alta	Fuerte	No
3	Nublado	Caluroso	Alta	Suave	Sí
4	Lluvioso	Moderado	Alta	Suave	Sí
5	Lluvioso	Frío	Normal	Suave	Sí
6	Lluvioso	Frío	Normal	Fuerte	No
7	Nublado	Frío	Normal	Fuerte	Sí
8	Soleado	Moderado	Alta	Suave	No
9	Soleado	Frío	Normal	Suave	Sí
10	Lluvioso	Moderado	Normal	Suave	Sí
11	Soleado	Moderado	Normal	Fuerte	Sí
12	Nublado	Moderado	Alta	Fuerte	Sí
13	Nublado	Caluroso	Normal	Suave	Sí
14	Lluvioso	Moderado	Alta	Fuerte	No

b. Crear un diagrama de barras que muestre la distribución de frecuencias de cada variable meteorológica según si se pudo jugar al tenis o no. ¿Qué variable meteorológica parece tener más influencia en la decisión de jugar al tenis?

Solución

```
using GLMakie, AlgebraOfGraphics

function frecuencias(df::DataFrame, var::Symbol)
    # Calculamos el número de días de cada clase que se juega al tenis.
    frec = combine(groupby(df, [var, :Tenis]), nrow => :Días)
    # Dibujamos el diagrama de barras.
    plt = data(frec) *
    mapping(var, :Días, stack = :Tenis, color = :Tenis, ) *
    visual(BarPlot)
    # Devolvemos el gráfico.
    return plt
end

fig = Figure()
draw!(fig[1, 1], frecuencias(df, :Cielo))
draw!(fig[1, 2], frecuencias(df, :Temperatura))
draw!(fig[1, 3], frecuencias(df, :Humedad))
draw!(fig[1, 4], frecuencias(df, :Viento))
fig
```



A la vista de las frecuencias de cada variable, las variable `Cielo` y `Humedad` parecen ser las que más influye en la decisión de jugar al tenis.

c. Calcular la impureza del conjunto de datos utilizando el índice de Gini. ¿Qué variable meteorológica parece tener más influencia en la decisión de jugar al tenis?

Ayuda

El índice de Gini se calcula mediante la fórmula

$$GI = 1 - \sum_{i=1}^{n} p_i^2$$

donde p_i es la proporción de cada clase en el conjunto de datos y n es el número de clases.
El índice de Gini toma valores entre 0 y $1 - \frac{1}{n}$ (0.5 en el caso de clasificación binaria), donde 0 indica que todas las instancias pertenecen a una sola clase (mínima impureza) y $1 - \frac{1}{n}$ indica que las instancias están distribuidas uniformemente entre todas las clases (máxima impureza).

Solución

```
function gini(df::DataFrame, var::Symbol)
    # Calculamos el número de ejemplos.
    n = nrow(df)
    # Calculamos las frecuencias absolutas de cada clase.
    frec = combine(groupby(df, var), nrow => :ni)
    # Calculamos la proporción de cada clase.
    frec.p = frec.ni ./ n
    # Calculamos el índice de Gini.
    gini = 1 - sum(frec.p .^ 2)
    return gini
end

g0 = gini(df, :Tenis)
```

```
0.4591836734693877
```

d. ¿Qué reducción del índice Gini se obtiene si dividimos el conjunto de ejemplos según la variable `Humedad`? ¿Y si dividimos el conjunto con respecto a la variable `Viento`?

Ayuda

La reducción del índice de Gini se calcula como la diferencia entre el índice de Gini del conjunto original y el índice de Gini del conjunto dividido.

$$\Delta GI = GI_{original} - GI_{dividido}$$

donde el índice de Gini del conjunto dividido es la media ponderada de los índices de Gini de los subconjuntos resultantes de la división.

Solución

Calculamos primero la reducción del índice de Gini al dividir el conjunto de ejemplos según la variable `Humedad`.

```
using Tidier
# Dividimos el conjunto de ejemplos según la variable Humedad.
df_humedad_alta = @filter(df, Humedad == "Alta")
df_humedad_normal = @filter(df, Humedad == "Normal")
# Calculamos los tamaños de los subconjuntos de ejemplos.
n = nrow(df_humedad_alta), nrow(df_humedad_normal)
# Calculamos el índice de Gini de cada subconjunto.
gis = gini(df_humedad_alta, :Tenis), gini(df_humedad_normal, :Tenis)
# Calculamos media ponderada de los índices de Gini de los subconjuntos
g_humedad = sum(gis .* n) / sum(n)
# Calculamos la reducción del índice de Gini.
g0 - g_humedad
```

```
0.09183673469387743
```

Calculamos ahora la reducción del índice de Gini al dividir el conjunto de ejemplos según la variable `Viento`.

```
# Dividimos el conjunto de ejemplos según la variable `Viento`
df_viento_fuerte = @filter(df, Viento == "Fuerte")
df_viento_suave = @filter(df, Viento == "Suave")
# Calculamos los tamaños de los subconjuntos de ejemplos
n = nrow(df_viento_fuerte), nrow(df_viento_suave)
# Calculamos el índice de Gini de cada subconjunto
gis = gini(df_viento_fuerte, :Tenis), gini(df_viento_suave, :Tenis)
# Calculamos media ponderada de los índices de Gini de los subconjuntos
g_viento = sum(gis .* n) / sum(n)
# Calculamos la reducción del índice de Gini
g0 - g_viento
```

```
0.030612244897959162
```

Como se puede observar, la reducción del índice de Gini al dividir el conjunto de ejemplos según la variable `Humedad` es mayor que la reducción del índice de Gini al dividir el conjunto con respecto a la variable `Viento`. Por lo tanto, la variable `Humedad` parece tener más influencia en la decisión de jugar al tenis y sería la variable que se debería elegir para dividir el conjunto de ejemplos.

e. Construir un árbol de decisión que explique si se puede jugar al tenis en función de las variables meteorológicas.

> **Ayuda**
>
> Usar la función `DecisionTreeClassifier` del paquete `DecisionTree.jl`.

Solución

```
using DecisionTree, CategoricalArrays
# Variables predictoras.
X = Matrix(select(df, Not(:Tenis)))
# Variable objetivo.
y = df.Tenis
# Convertir las variables categóricas a enteros.
X = hcat([levelcode.(categorical(X[:, j])) for j in 1:size(X, 2)]...)
# Convertir la variable objetivo a enteros.
y = levelcode.(categorical(y))
tree = DecisionTreeClassifier(max_depth=3)
fit!(tree, X, y)
```

```
DecisionTreeClassifier
max_depth:                3
min_samples_leaf:         1
min_samples_split:        2
min_purity_increase:      0.0
pruning_purity_threshold: 1.0
n_subfeatures:            0
classes:                  [1, 2]
root:                     Decision Tree
Leaves: 6
Depth:  3
```

f. Visualizar el árbol de decisión construido.

Ayuda

Usar la función `plot_tree` del paquete `DecisionTree.jl`.

Solución

```
print_tree(tree, feature_names=names(df)[1:end-1])
```

```
Feature 3: "Humedad" < 2.0 ?
  Feature 1: "Cielo" < 3.0 ?
      Feature 4: "Viento" < 2.0 ?
          2 : 1/2
          2 : 2/2
      1 : 3/3
  Feature 1: "Cielo" < 2.0 ?
```

```
    Feature 4: "Viento" < 2.0 ?
        1 : 1/1
        2 : 2/2
    2 : 4/4
```

Ejercicio 4.2. El conjunto de datos pingüinos.csv contiene un conjunto de datos sobre tres eEspecie de pingüinos con las siguientes variables:

- Especie: Especie de pingüino, comúnmente Adelie, Chinstrap o Gentoo.
- Isla: Isla del archipiélago Palmer donde se realizó la observación.
- Longitud_pico: Longitud del pico en mm.
- Profundidad_pico: Profundidad del pico en mm
- Longitud_ala: Longitud de la aleta en mm.
- Peso: Masa corporal en gramos.
- Sexo: Sexo

a. Cargar los datos del archivo `pinguïnos.csv` en un data frame.

Solución

```julia
using CSV, DataFrames
df = CSV.read("datos/pingüinos.csv", DataFrame, missingstring="NA")
```

	Especie	Isla	Longitud_pico	Profundidad_pico	Longitud_ala	Peso	Sex
	String15	String15	Float64?	Float64?	Int64?	Int64?	Strin
1	Adelie	Torgersen	39.1	18.7	181	3750	mac
2	Adelie	Torgersen	39.5	17.4	186	3800	hem
3	Adelie	Torgersen	40.3	18.0	195	3250	hem
4	Adelie	Torgersen	*missing*	*missing*	*missing*	*missing*	*miss*
5	Adelie	Torgersen	36.7	19.3	193	3450	hem
6	Adelie	Torgersen	39.3	20.6	190	3650	mac
7	Adelie	Torgersen	38.9	17.8	181	3625	hem
8	Adelie	Torgersen	39.2	19.6	195	4675	mac
9	Adelie	Torgersen	34.1	18.1	193	3475	*miss*
10	Adelie	Torgersen	42.0	20.2	190	4250	*miss*
11	Adelie	Torgersen	37.8	17.1	186	3300	*miss*
12	Adelie	Torgersen	37.8	17.3	180	3700	*miss*
13	Adelie	Torgersen	41.1	17.6	182	3200	hem
14	Adelie	Torgersen	38.6	21.2	191	3800	mac
15	Adelie	Torgersen	34.6	21.1	198	4400	mac
16	Adelie	Torgersen	36.6	17.8	185	3700	hem
17	Adelie	Torgersen	38.7	19.0	195	3450	hem
18	Adelie	Torgersen	42.5	20.7	197	4500	mac
19	Adelie	Torgersen	34.4	18.4	184	3325	hem
20	Adelie	Torgersen	46.0	21.5	194	4200	mac
21	Adelie	Biscoe	37.8	18.3	174	3400	hem
22	Adelie	Biscoe	37.7	18.7	180	3600	mac
23	Adelie	Biscoe	35.9	19.2	189	3800	hem
24	Adelie	Biscoe	38.2	18.1	185	3950	mac
25	Adelie	Biscoe	38.8	17.2	180	3800	mac
26	Adelie	Biscoe	35.3	18.9	187	3800	hem
27	Adelie	Biscoe	40.6	18.6	183	3550	mac
28	Adelie	Biscoe	40.5	17.9	187	3200	hem
29	Adelie	Biscoe	37.9	18.6	172	3150	hem
30	Adelie	Biscoe	40.5	18.9	180	3950	mac
...	...	...	...	...	...	...	...

b. Hacer un análisis de los datos perdidos en el data frame.

Solución

```
describe(df, :nmissing)
```

	variable	nmissing
	Symbol	Int64
1	Especie	0
2	Isla	0
3	Longitud_pico	2
4	Profundidad_pico	2
5	Longitud_ala	2
6	Peso	2
7	Sexo	11

c. Eliminar del data frame los casos con valores perdidos.

Solución

```
dropmissing!(df)
```

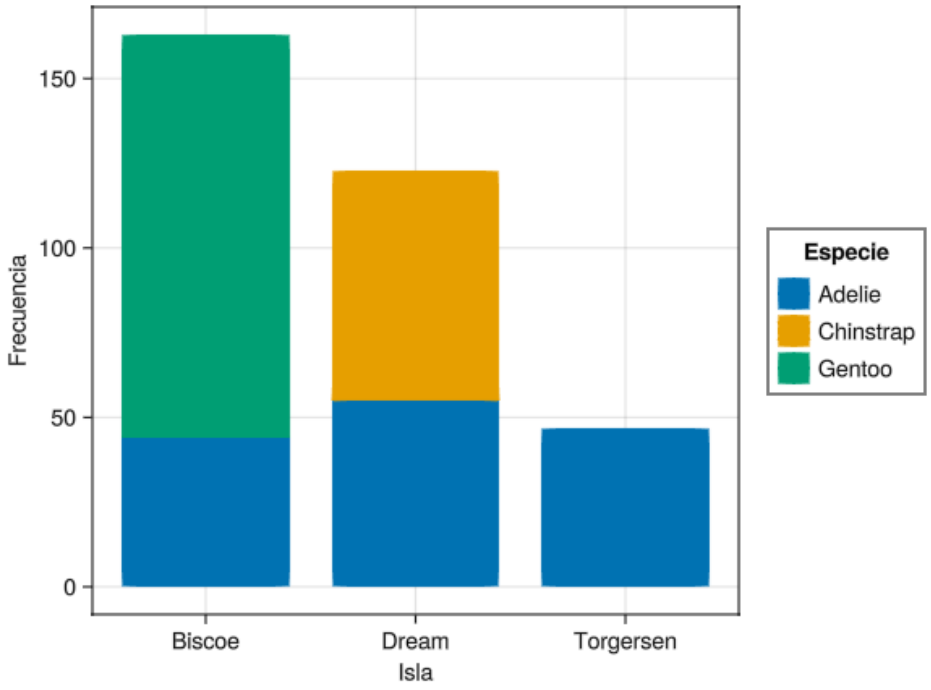
	Especie	Isla	Longitud_pico	Profundidad_pico	Longitud_ala	Peso	Sexo
	String15	String15	Float64	Float64	Int64	Int64	String7
1	Adelie	Torgersen	39.1	18.7	181	3750	macho
2	Adelie	Torgersen	39.5	17.4	186	3800	hembra
3	Adelie	Torgersen	40.3	18.0	195	3250	hembra
4	Adelie	Torgersen	36.7	19.3	193	3450	hembra
5	Adelie	Torgersen	39.3	20.6	190	3650	macho
6	Adelie	Torgersen	38.9	17.8	181	3625	hembra
7	Adelie	Torgersen	39.2	19.6	195	4675	macho
8	Adelie	Torgersen	41.1	17.6	182	3200	hembra
9	Adelie	Torgersen	38.6	21.2	191	3800	macho
10	Adelie	Torgersen	34.6	21.1	198	4400	macho
11	Adelie	Torgersen	36.6	17.8	185	3700	hembra
12	Adelie	Torgersen	38.7	19.0	195	3450	hembra
13	Adelie	Torgersen	42.5	20.7	197	4500	macho
14	Adelie	Torgersen	34.4	18.4	184	3325	hembra
15	Adelie	Torgersen	46.0	21.5	194	4200	macho
16	Adelie	Biscoe	37.8	18.3	174	3400	hembra
17	Adelie	Biscoe	37.7	18.7	180	3600	macho
18	Adelie	Biscoe	35.9	19.2	189	3800	hembra
19	Adelie	Biscoe	38.2	18.1	185	3950	macho
20	Adelie	Biscoe	38.8	17.2	180	3800	macho
21	Adelie	Biscoe	35.3	18.9	187	3800	hembra
22	Adelie	Biscoe	40.6	18.6	183	3550	macho
23	Adelie	Biscoe	40.5	17.9	187	3200	hembra
24	Adelie	Biscoe	37.9	18.6	172	3150	hembra
25	Adelie	Biscoe	40.5	18.9	180	3950	macho
26	Adelie	Dream	39.5	16.7	178	3250	hembra
27	Adelie	Dream	37.2	18.1	178	3900	macho
28	Adelie	Dream	39.5	17.8	188	3300	hembra
29	Adelie	Dream	40.9	18.9	184	3900	macho
30	Adelie	Dream	36.4	17.0	195	3325	hembra
...	...	...	...	...	...	...	...

d. Crear diagramas que muestren la distribución de frecuencias de cada variable según la especie de pingüino. ¿Qué variable parece tener más influencia en la especie de pingüino?

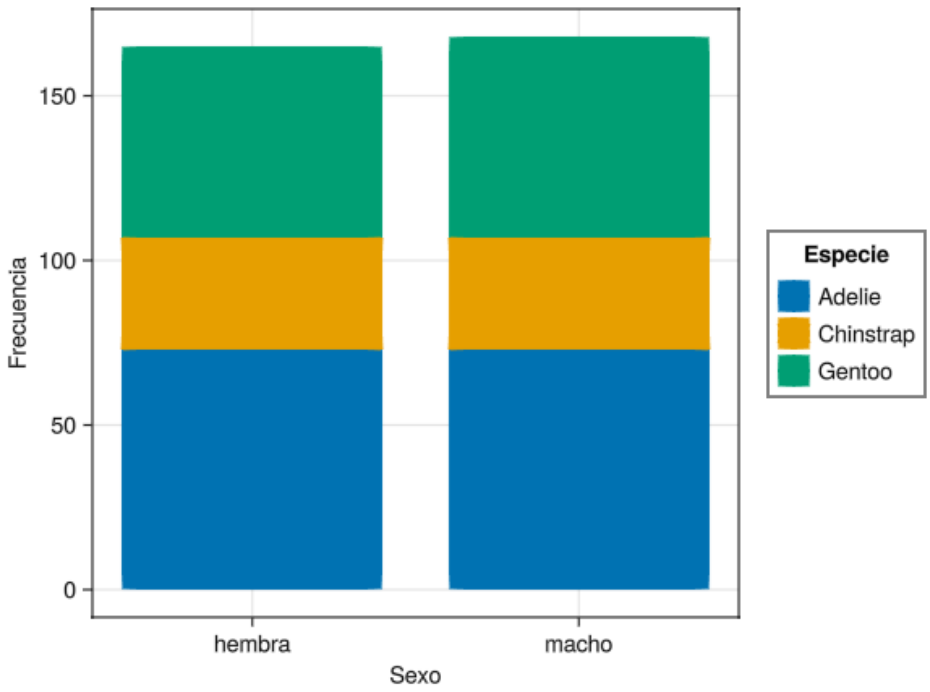
Solución

Para las variables cualitativas dibujamos diagramas de barras.

```
using GLMakie, AlgebraOfGraphics

frec_isla = combine(groupby(df, [:Isla, :Especie]), nrow => :Frecuencia)
data(frec_isla) *
    mapping(:Isla, :Frecuencia, stack = :Especie, color =:Especie) *
    visual(BarPlot) |> draw
```

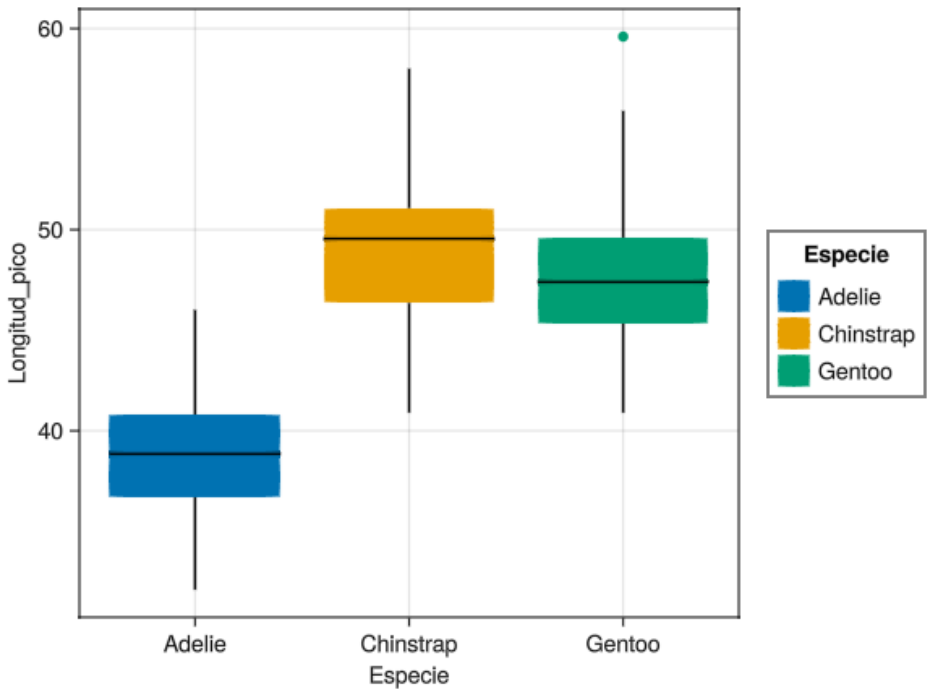



```
frec_sexo = combine(groupby(df, [:Sexo, :Especie]), nrow => :Frecuencia)
data(frec_sexo) *
    mapping(:Sexo, :Frecuencia, stack = :Especie, color =:Especie) *
    visual(BarPlot) |> draw
```

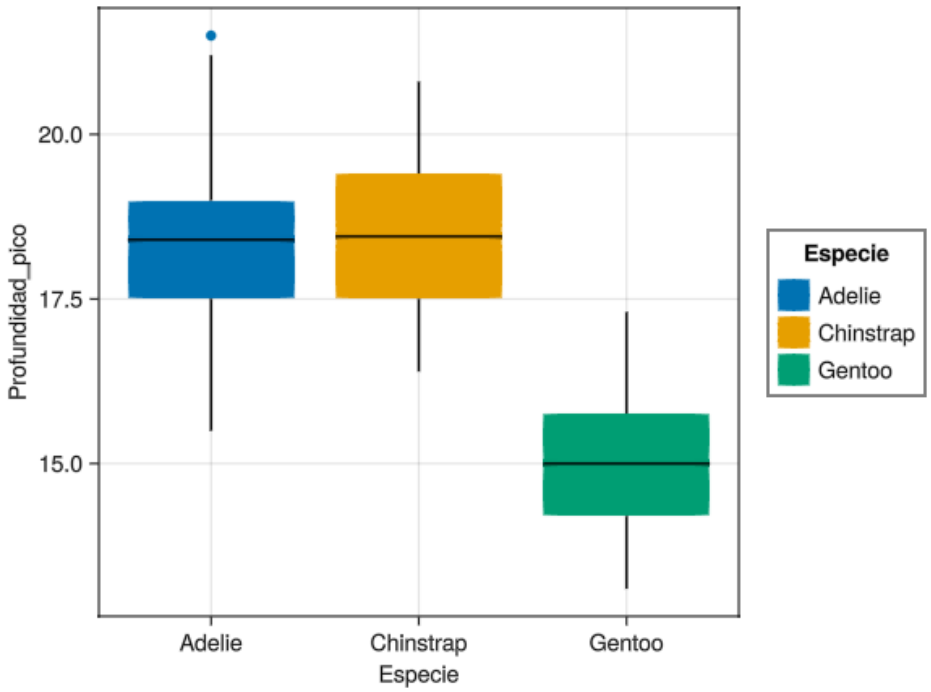


Para las variables cuantitativas dibujamos diagramas de cajas.

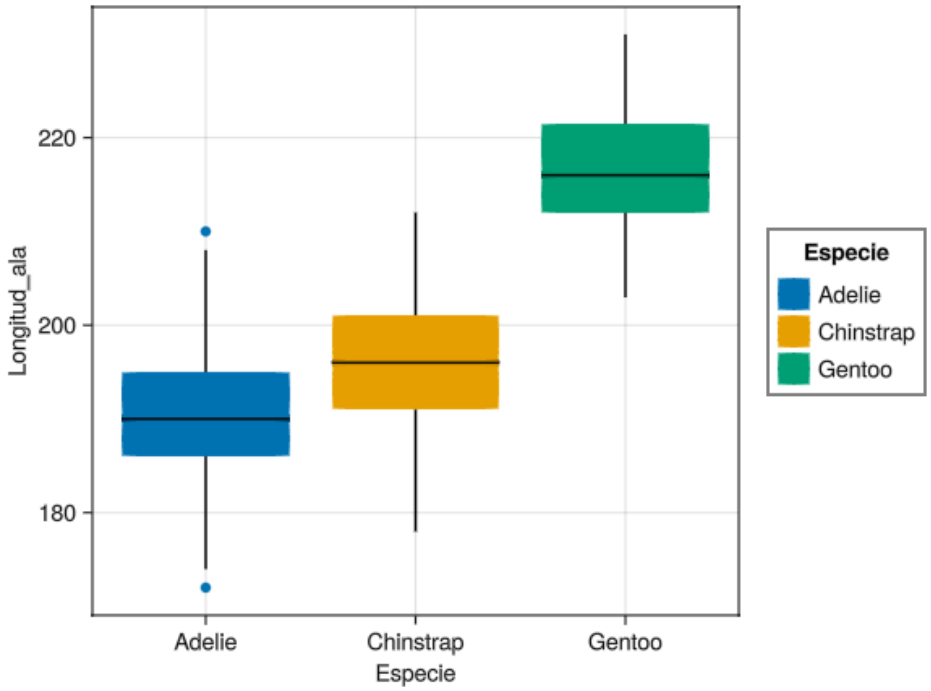
```
function cajas(df, var, clase)
    data(df) *
        mapping(clase, var, color = clase) *
        visual(BoxPlot) |>
        draw
end

cajas(df, :Longitud_pico, :Especie)
```



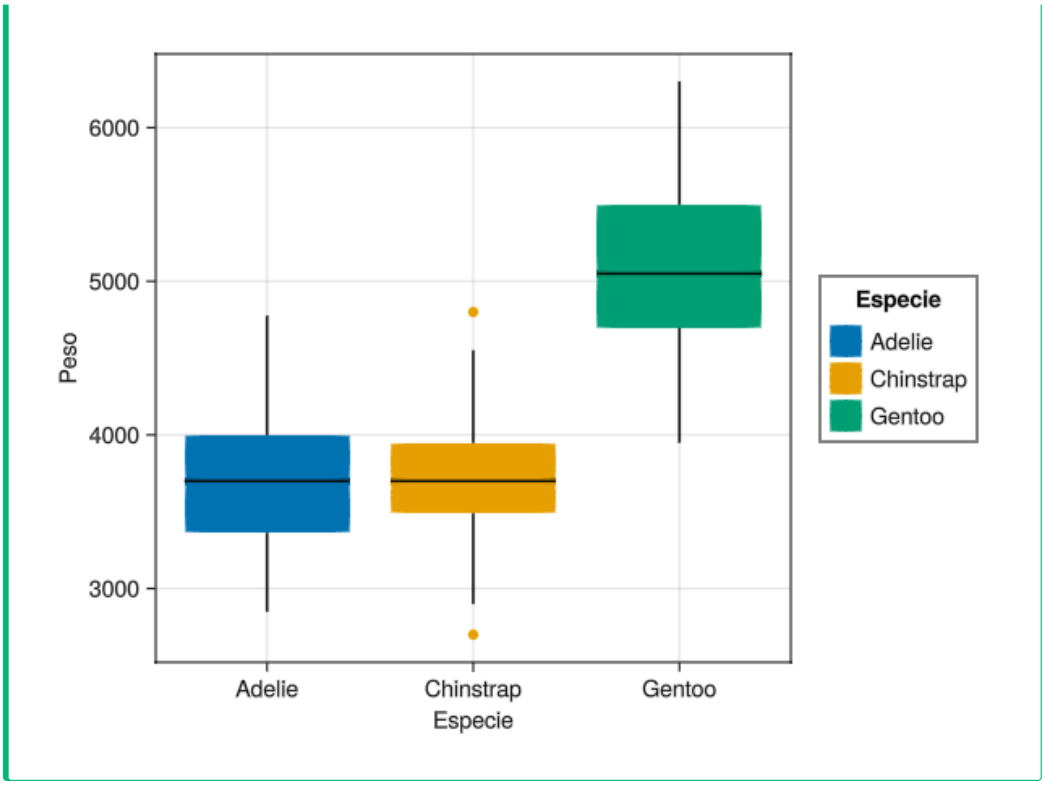

```
cajas(df, :Profundidad_pico, :Especie)
```



```
cajas(df, :Longitud_ala, :Especie)
```




```
cajas(df, :Peso, :Especie)
```



e. ¿Cuál es la reducción de la impureza del conjunto de datos si dividimos el conjunto de datos en dos conjuntos según si la longitud del pico es mayor o menor que 44 mm?

Solución

```
using Tidier
function gini(df::DataFrame, var::Symbol)
    n = nrow(df)
    frec = combine(groupby(df, var), nrow => :ni)
    frec.p = frec.ni ./ n
    gini = 1 - sum(frec.p .^ 2)
    return gini
end

function reduccion_impureza(df::DataFrame, var::Symbol, val::Number)
    # Dividimos el conjunto de ejemplos según la longitud del pico es menor de 44.
    df_menor = @eval @filter($df, $var <= $val)
    df_mayor = @eval @filter($df, $var > $val)
    # Calculamos los tamaños de los subconjuntos de ejemplos.
    n = nrow(df_menor), nrow(df_mayor)
    # Calculamos el índice de Gini de cada subconjunto.
    gis = gini(df_menor, :Especie), gini(df_mayor, :Especie)
    # Calculamos media ponderada de los índices de Gini de los subconjuntos.
    g1 = sum(gis .* n) / sum(n)
    # Calculamos la reducción del índice de Gini.
    gini(df, :Especie) - g1
end

reduccion_impureza(df, :Longitud_pico, 44)
```

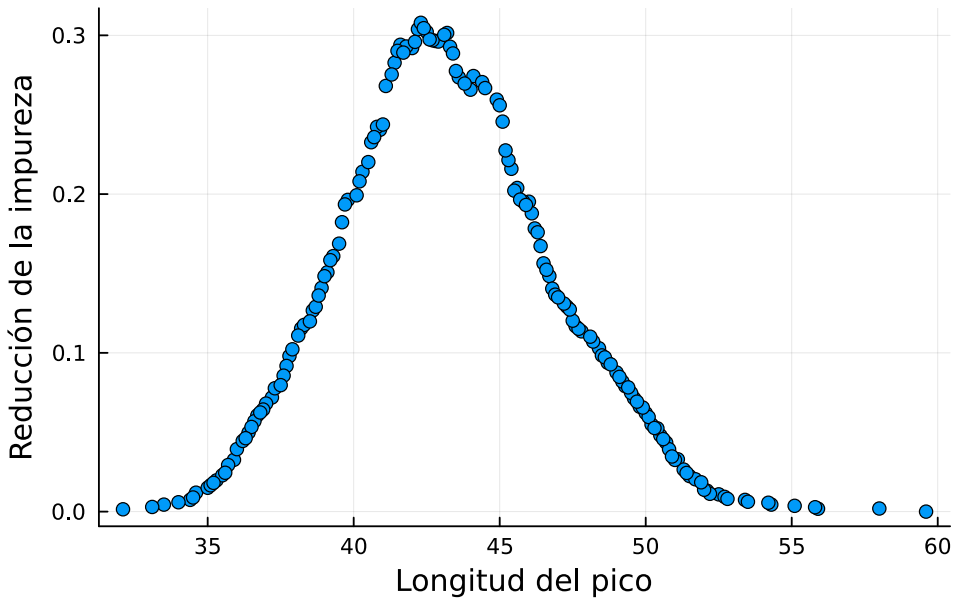
```
0.26577182779353914
```

f. Determinar el valor óptimo de división del conjunto de datos según la longitud del pico. Para ello, calcular la reducción de la impureza para cada valor de longitud del pico y dibujar el resultado.

Solución

Dibujamos la reducción de la impureza en función de la longitud del pico.

```
using Plots
# Valores únicos de longitud del pico.
valores = unique(df.Longitud_pico)
# Reducción de la impureza para cada valor.
reducciones = [reduccion_impureza(df, :Longitud_pico, val) for val in valores]
# Graficamos el resultado.
Plots.scatter(valores, reducciones, xlabel = "Longitud del pico", ylabel = "Reducc
```



Y ahora obtenemos el valor óptimo de división del conjunto de datos según la longitud del pico.

```
val_optimo = valores[argmax(reducciones)]
```

```
42.3
```

g. Dividir aleatoriamente el dataframe en un conjunto de entrenamiento y un conjunto de test con proporciones 3/4 y 1/4 respectivamente.

Ayuda

Utilizar la función `shuffle` del paquete `Random` para barajar el dataframe y luego dividirlo en dos subconjuntos.

Solución

```
using Random
# Establecemos la semilla para la reproducibilidad.
Random.seed!(1234)
# Barajamos el dataframe.
df = shuffle(df)
# Dividimos el dataframe en un conjunto de entrenamiento y un conjunto de test.
n = nrow(df)
df_test = df[1:div(n, 4), :]
df_train = df[div(n, 4)+1:end, :]
```

	Especie	Isla	Longitud_pico	Profundidad_pico	Longitud_ala	Peso	Sexo
	String15	String15	Float64	Float64	Int64	Int64	String
1	Adelie	Dream	39.0	18.7	185	3650	mach
2	Chinstrap	Dream	52.8	20.0	205	4550	mach
3	Chinstrap	Dream	55.8	19.8	207	4000	mach
4	Adelie	Torgersen	35.1	19.4	193	4200	mach
5	Adelie	Torgersen	34.6	21.1	198	4400	mach
6	Gentoo	Biscoe	50.0	15.2	218	5700	mach
7	Chinstrap	Dream	50.6	19.4	193	3800	mach
8	Chinstrap	Dream	43.5	18.1	202	3400	hemb
9	Adelie	Dream	36.9	18.6	189	3500	hemb
10	Adelie	Dream	36.6	18.4	184	3475	hemb
11	Chinstrap	Dream	46.6	17.8	193	3800	hemb
12	Gentoo	Biscoe	50.8	17.3	228	5600	mach
13	Chinstrap	Dream	52.2	18.8	197	3450	mach
14	Adelie	Dream	39.6	18.8	190	4600	mach
15	Adelie	Torgersen	42.8	18.5	195	4250	mach
16	Adelie	Biscoe	36.5	16.6	181	2850	hemb
17	Gentoo	Biscoe	49.1	14.8	220	5150	hemb
18	Chinstrap	Dream	43.2	16.6	187	2900	hemb
19	Gentoo	Biscoe	43.3	13.4	209	4400	hemb
20	Gentoo	Biscoe	49.5	16.1	224	5650	mach
21	Adelie	Biscoe	37.8	20.0	190	4250	mach
22	Gentoo	Biscoe	50.4	15.3	224	5550	mach
23	Adelie	Biscoe	45.6	20.3	191	4600	mach
24	Chinstrap	Dream	45.4	18.7	188	3525	hemb
25	Adelie	Dream	39.2	18.6	190	4250	mach
26	Gentoo	Biscoe	48.4	14.4	203	4625	hemb
27	Adelie	Torgersen	35.2	15.9	186	3050	hemb
28	Gentoo	Biscoe	48.4	16.3	220	5400	mach
29	Adelie	Dream	33.1	16.1	178	2900	hemb
30	Adelie	Dream	36.8	18.5	193	3500	hemb
...	...	...	...	...	...	...	...

h. Construir un árbol de decisión con el conjunto de entrenamiento sin tener en cuenta la variable `Isla` y visualizarlo.

Solución

```
using DecisionTree, CategoricalArrays
# Variables predictivas.
X_train = Matrix(select(df_train, Not(:Isla, :Especie)))
# Variable objetivo.
y_train = df_train.Especie
# Convertir las variables categóricas a enteros.
X_train = hcat([levelcode.(categorical(X_train[:, j])) for j in 1:size(X_train, 2)
# Convertir la variable objetivo a enteros
y_train = levelcode.(categorical(y_train))

# Construimos el árbol de decisión con profundidad máxima 3.
tree = DecisionTreeClassifier(max_depth = 3)
fit!(tree, X_train, y_train)
print_tree(tree, feature_names=names(df)[3:end])
```

```
Feature 3: "Longitud_ala" < 29.0 ?
  Feature 1: "Longitud_pico" < 62.0 ?
      1 : 96/96
      Feature 1: "Longitud_pico" < 87.0 ?
          2 : 10/20
          2 : 37/38
  Feature 2: "Profundidad_pico" < 46.0 ?
      3 : 90/90
      Feature 1: "Longitud_pico" < 109.0 ?
          1 : 2/2
          2 : 4/4
```

i. Predecir la especie de los pingüinos del conjunto de test y calcular la matriz de confusión de las predicciones.

Ayuda

Utilizar la función `confmat` del paquete `StatisticalMeaures` para barajar el dataframe y luego dividirlo en dos subconjuntos.

Solución

```
using StatisticalMeasures
# Variables predictivas
X_test = Matrix(select(df_test, Not(:Isla, :Especie)))
# Variable objetivo
y_test = df_test.Especie
# Convertir las variables categóricas a enteros
X_test = hcat([levelcode.(categorical(X_test[:, j])) for j in 1:size(X_test, 2)]..
# Convertir la variable objetivo a enteros
y_test = levelcode.(categorical(y_test))
# Predecimos la especie de pingüino del conjunto de test
y_pred = predict(tree, X_test)
# Calculamos la precisión del modelo
confmat(y_pred, y_test)
```

```
            Ground Truth

Predicted  1    2    3

    1      38   11   9

    2      0    6    0

    3      0    0    19
```

j. Calcular la precisión del modelo.

Ayuda

La precisión es la proporción de predicciones correctas sobre el total de predicciones.
Utilizar la función `accuracy` del paquete `StatisticalMeaures` para calcular la precisión del modelo.

Solución

```
# Calculamos la precisión del modelo
accuracy(y_pred, y_test)
```

```
0.7590361445783133
```

Ejercicio 4.3. El fichero `vinos.csv` contiene información sobre las características de una muestra de vinos portugueses de la denominación "Vinho Verde". Las variables que contiene son:

Variable	Descripción	Tipo (unidades)
tipo	Tipo de vino	Categórica (blanco, tinto)
meses.barrica	Mesesde envejecimiento en barrica	Numérica(meses)
acided.fija	Cantidadde ácidotartárico	Numérica(g/dm3)
acided.volatil	Cantidad de ácido acético	Numérica(g/dm3)
acido.citrico	Cantidad de ácidocítrico	Numérica(g/dm3)
azucar.residual	Cantidad de azúcarremanente después de la fermentación	Numérica(g/dm3)
cloruro.sodico	Cantidad de clorurosódico	Numérica(g/dm3)
dioxido.azufre.libre	Cantidad de dióxido de azufreen formalibre	Numérica(mg/dm3)
dioxido.azufre.total	Cantidadde dióxido de azufretotal en forma libre o ligada	Numérica(mg/dm3)
densidad	Densidad	Numérica(g/cm3)
ph	pH	Numérica(0-14)
sulfatos	Cantidadde sulfato de potasio	Numérica(g/dm3)
alcohol	Porcentajede contenidode alcohol	Numérica(0-100)
calidad	Calificación otorgada porun panel de expertos	Numérica(0-10)

a. Crear un data frame con los datos de los vinos a partir del fichero `vinos.csv`.

Solución

```
using CSV, DataFrames
df = CSV.read("datos/vinos.csv", DataFrame)
```

	tipo	meses_barrica	acided_fija	acided_volatil	acido_citrico	azucar_residual	
	String7	Int64	Float64	Float64	Float64	Float64	
1	blanco	0	7.0	0.27	0.36	20.7	.
2	blanco	0	6.3	0.3	0.34	1.6	.
3	blanco	0	8.1	0.28	0.4	6.9	.
4	blanco	0	7.2	0.23	0.32	8.5	.
5	blanco	0	6.2	0.32	0.16	7.0	.
6	blanco	0	8.1	0.22	0.43	1.5	.
7	blanco	0	8.1	0.27	0.41	1.45	.
8	blanco	0	8.6	0.23	0.4	4.2	.
9	blanco	0	7.9	0.18	0.37	1.2	.
10	blanco	0	6.6	0.16	0.4	1.5	.
11	blanco	0	8.3	0.42	0.62	19.25	.
12	blanco	0	6.6	0.17	0.38	1.5	.
13	blanco	0	6.3	0.48	0.04	1.1	.
14	blanco	0	6.2	0.66	0.48	1.2	.
15	blanco	0	7.4	0.34	0.42	1.1	.
16	blanco	0	6.5	0.31	0.14	7.5	.
17	blanco	0	6.4	0.31	0.38	2.9	.
18	blanco	0	6.8	0.26	0.42	1.7	.
19	blanco	0	7.6	0.67	0.14	1.5	.
20	blanco	0	6.6	0.27	0.41	1.3	.
21	blanco	0	7.0	0.25	0.32	9.0	.
22	blanco	0	6.9	0.24	0.35	1.0	.
23	blanco	0	7.0	0.28	0.39	8.7	.
24	blanco	0	7.4	0.27	0.48	1.1	.
25	blanco	0	7.2	0.32	0.36	2.0	.
26	blanco	0	8.5	0.24	0.39	10.4	.
27	blanco	0	8.3	0.14	0.34	1.1	.
28	blanco	0	7.4	0.25	0.36	2.05	.
29	blanco	0	6.2	0.12	0.34	1.5	.
30	blanco	0	5.8	0.27	0.2	14.95	.
...	...	...	...	...	...	...	

b. Mostrar los tipos de cada variable del data frame.

> **Ayuda**
>
> Usar la función `schema` del paquete `MLJ`.

Solución

```
using MLJ
schema(df)
```

```
WARNING: using MLJ.fit! in module Main conflicts with an existing identifier.
WARNING: using MLJ.predict in module Main conflicts with an existing identifier.
```

names	scitypes	types
tipo	Textual	String7
meses_barrica	Count	Int64
acided_fija	Continuous	Float64
acided_volatil	Continuous	Float64
acido_citrico	Continuous	Float64
azucar_residual	Continuous	Float64
cloruro_sodico	Continuous	Float64
dioxido_azufre_libre	Continuous	Float64
dioxido_azufre_total	Continuous	Float64
densidad	Continuous	Float64
ph	Continuous	Float64
sulfatos	Continuous	Float64
alcohol	Continuous	Float64
calidad	Count	Int64

c. Hacer un análisis de los datos perdidos en el data frame.

Solución

```
describe(df, :nmissing)
```

	variable	nmissing
	Symbol	Int64
1	tipo	0
2	meses_barrica	0
3	acided_fija	0
4	acided_volatil	0
5	acido_citrico	0
6	azucar_residual	0
7	cloruro_sodico	0
8	dioxido_azufre_libre	0
9	dioxido_azufre_total	0
10	densidad	0
11	ph	0
12	sulfatos	0
13	alcohol	0
14	calidad	0

d. Se considera que un vino es bueno si tiene una puntuación de calidad mayor que 6.5. Recodificar la variable `calidad` en una variable categórica que tome el valor 1 si la calidad es mayor que 6.5 y 0 en caso contrario.

Solución

```
using CategoricalArrays
# Recodificamos la variable calidad.
df.calidad = cut(df.calidad, [0, 6.5, 10], labels = [0, 1])
```

```
5320-element CategoricalArray{Int64,1,UInt32}:
 0
 0
 0
 0
 0
 0
 0
 0
 0
 1
 0
 1
 0

 0
```

```
0
0
0
0
0
0
0
0
0
0
0
0
```

e. Descomponer el data frame en un data frame con las variables predictivas y un vector con la variable objetivo `bueno`.

Solución

```
y, X = unpack(df, ==(:calidad), rng = 123)
```

```
(CategoricalValue{Int64, UInt32}[0, 0, 0, 0, 0, 1, 0, 0, 0, 0  …  0, 0, 0, 0, 0, 0
  Row   tipo     meses_barrica  acided_fija  acided_volatil  acido_citrico  az
        String7  Int64          Float64      Float64         Float64        Fl

    1   blanco               0          6.7           0.5           0.36
    2   blanco               0          6.3           0.2           0.3
    3   blanco               0          6.2          0.35          0.03
    4   tinto                3          8.0          0.39           0.3
    5   blanco               0          7.9         0.255          0.26
    6   blanco               0          6.1          0.31          0.37
    7   blanco               0          6.8          0.28          0.36
    8   blanco               0          8.2          0.34          0.49
    9   tinto                0          6.7          0.48          0.02
   10   blanco               0          7.4          0.35           0.2
   11   tinto                5          7.5          0.53          0.06

 5311   blanco               0          7.2          0.14          0.35
 5312   tinto                3          7.6          0.41          0.24
 5313   tinto                0          7.3           0.4           0.3
 5314   tinto                4          7.1          0.48          0.28
 5315   blanco               0          6.4          0.29           0.2
 5316   blanco               0          9.4          0.24          0.29
 5317   blanco               0          6.3          0.25          0.27
 5318   blanco               0          5.5          0.16          0.26
```

```
5319   blanco                0          7.4            0.36            0.32
5320   blanco                0          7.6            0.51            0.24
                                                  8 columns and 5299 rows omitted)
```

f. Para poder entrenar un modelo de un arbol de decisión, las variables predictivas deben ser cuantitativas. Transmformar las variables categóricas en variables numéricas.

Solución

```
# Convertir las variables categóricas a enteros.
coerce!(X, :tipo => OrderedFactor, :meses_barrica => Continuous)
schema(X)
```

```
names                 scitypes           types

tipo                  OrderedFactor{2}   CategoricalValue{String7, UInt32}
meses_barrica         Continuous         Float64
acided_fija           Continuous         Float64
acided_volatil        Continuous         Float64
acido_citrico         Continuous         Float64
azucar_residual       Continuous         Float64
cloruro_sodico        Continuous         Float64
dioxido_azufre_libre  Continuous         Float64
dioxido_azufre_total  Continuous         Float64
densidad              Continuous         Float64
ph                    Continuous         Float64
sulfatos              Continuous         Float64
alcohol               Continuous         Float64
```

g. Definir un modelo de árbol de decisión con profundidad máxima 3.

Ayuda

Cargar el modelo `DecisionTreeClassifier` del paquete `DecisionTree` con la macros `@iload`.

Solución

```
Tree = @iload DecisionTreeClassifier pkg = "DecisionTree"
tree = Tree(max_depth = 3, rng = 123)
```

```
[ Info: For silent loading, specify `verbosity=0`.
import MLJDecisionTreeInterface
DecisionTreeClassifier(
  max_depth = 3,
  min_samples_leaf = 1,
  min_samples_split = 2,
  min_purity_increase = 0.0,
  n_subfeatures = 0,
  post_prune = false,
  merge_purity_threshold = 1.0,
  display_depth = 5,
  feature_importance = :impurity,
  rng = 123)
```

h. Evaluar el modelo mediante validación cruzada usando las métricas de la pérdida de entropía cruzada estratificada, la matriz de confusión, la tasa de verdaderos positivos, la tasa de verdaderos negativos, el valor predictivo positivo, el valor predictivo negativo y la precisión. ¿Es un buen modelo?

Ayuda

Usar la función `evaluate` del paquete `MLJ` para evaluar el modelo.
Para indicar que se utilice como método de muestreo la validación cruzada se utiliza el parámetro `resampling = CV(shuffle=true)`, mientras que para usar validación cruzada estratificada se utiliza `resampling = StratifiedCV(shuffle=true)`.
Para indicar las métricas a utilizar se utiliza el parámetro `measures = [cross_entropy, confusion_matrix, true_positive_rate, true_negative_rate, ppv, npv, accuracy]`.

Solución

```
evaluate(tree, X, y, resampling = StratifiedCV(shuffle=true), measures=[cross_entr
```

```
Evaluating over 6 folds:  33%[========>                ]  ETA: 0:00:09Evaluating c
PerformanceEvaluation object with these fields:
  model, measure, operation,
  measurement, per_fold, per_observation,
  fitted_params_per_fold, report_per_fold,
  train_test_rows, resampling, repeats
Extract:
```

```
    measure                    operation      measurement

A   LogLoss(                   predict        0.376
      tol = 2.22045e-16)
B   ConfusionMatrix(           predict_mode   ConfusionMatrix{2}([3819762 77
      levels = nothing,
      perm = nothing,
      rev = nothing,
      checks = true)
C   TruePositiveRate(          predict_mode   0.129
      levels = nothing,
      rev = nothing,
      checks = true)
D   TrueNegativeRate(          predict_mode   0.999
      levels = nothing,
      rev = nothing,
      checks = true)
E   PositivePredictiveValue(   predict_mode   0.978
      levels = nothing,
      rev = nothing,
      checks = true)
F   NegativePredictiveValue(   predict_mode   0.831
      levels = nothing,
      rev = nothing,

                                                  1 column and 2 rows omitted

    per_fold

A   [0.364, 0.399, 0.409, 0.363, 0.361, 0.362]
B   ConfusionMatrix{2, true, CategoricalValue{Int64, UInt32}}[ConfusionMatri
C   [0.179, 0.149, 0.107, 0.101, 0.137, 0.101]
D   [1.0, 1.0, 0.999, 1.0, 0.997, 1.0]
E   [1.0, 1.0, 0.947, 1.0, 0.92, 1.0]
F   [0.839, 0.834, 0.827, 0.825, 0.832, 0.826]
G   [0.844, 0.839, 0.83, 0.829, 0.834, 0.83]

                                                                2 columns omitted
```

La precisión del modelo es de 0.834 que no está mal, pero si consdieramos la tasa de verdadero positivos, que es 0.13 y la tasa de verdaderos negativos, que es prácticamente 1, el modelo tiene un buen rendimiento en la clasificación

de los vinos malos, pero un mal rendimiento en la clasificación de los vinos malos. Por lo tanto, el modelo no es un buen modelo.