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 el factor `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 `ismissing` en la condición del filtro.

Solución

```
df[.!ismissing.(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. Calcular el décimo término de la sucesión $\left(\frac{3n^2+n}{6n^2-1}\right)_{n=1}^{\infty}$.

*Hint: *

Introducir hasta 5 decimales

Ejercicio 2.5. Los ficheros `vinos-blancos.xls` y `vinos-tintos.csv` contienen información sobre las características de vinos blancos y tintos portugueses de la denominación "Vinho Verde". Las variables almacenadas en estos archivos son las siguientes:

Variable	Descripción	Tipo (unidades)
tipo	Tipo de vino	Factor (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 blancos partir del fichero de Excel `vinos-blancos.xlsx`.

b. Crear un data frame con los datos de los vinos tintos partir del fichero csv `vinos-tintos.csv`.

c. Fusionar los datos de los vinos blancos y tintos en un nuevo data frame.

d. Convertir el tipo de vino en un factor.

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

f. Crear un factor `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

g. Crear un factor `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

h. 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.

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

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

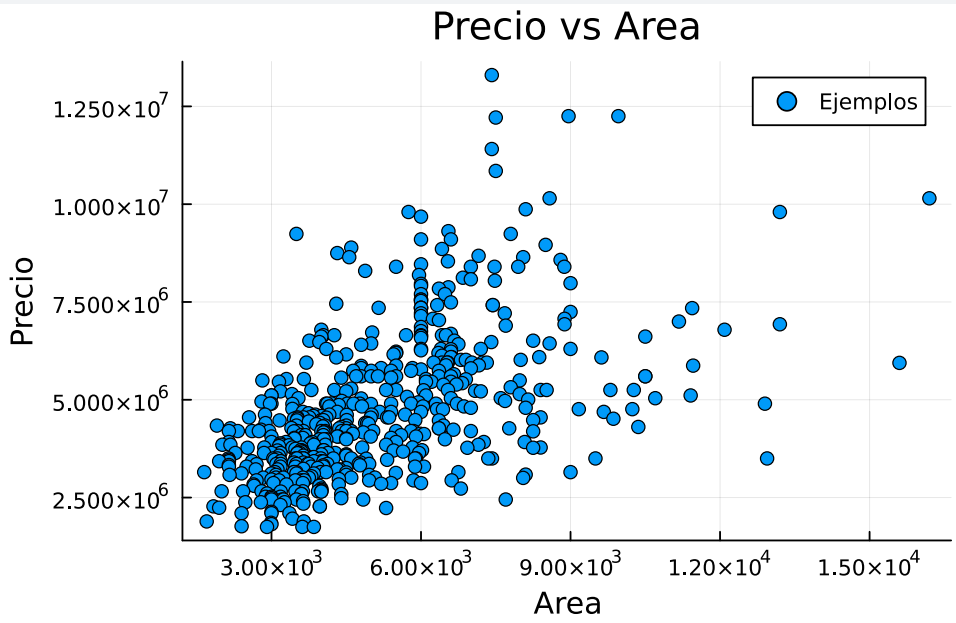
```
using CSV, DataFrames
df = CSV.read("datos/viviendas.csv", DataFrame)
first(df, 5)
```

	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
```

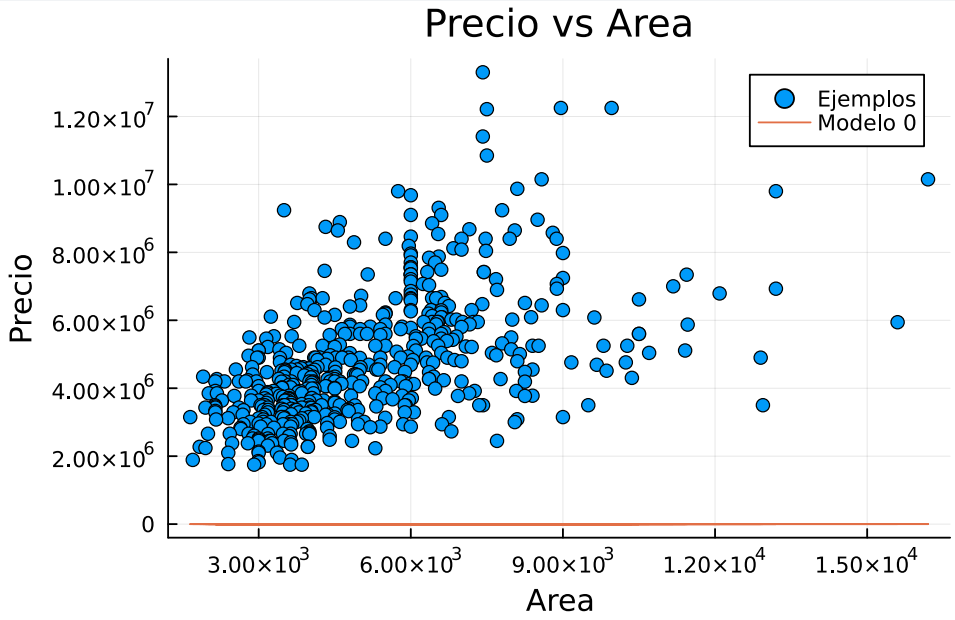
```
precio (generic function with 1 method)
```

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)
```

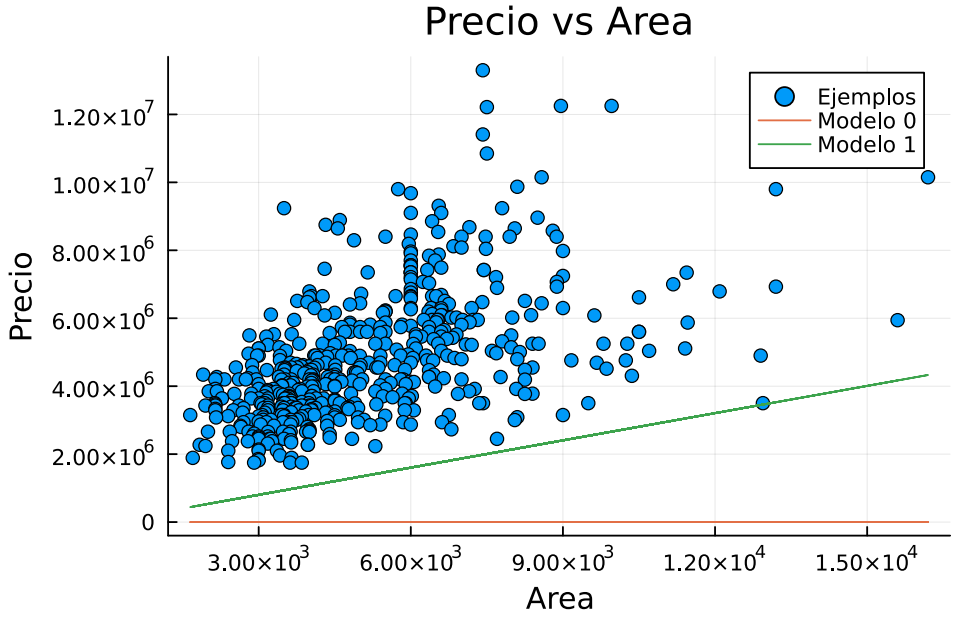
Aplicamos la función a los parámetros del modelo actual y mostramos los nuevos parámetros.

```
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.

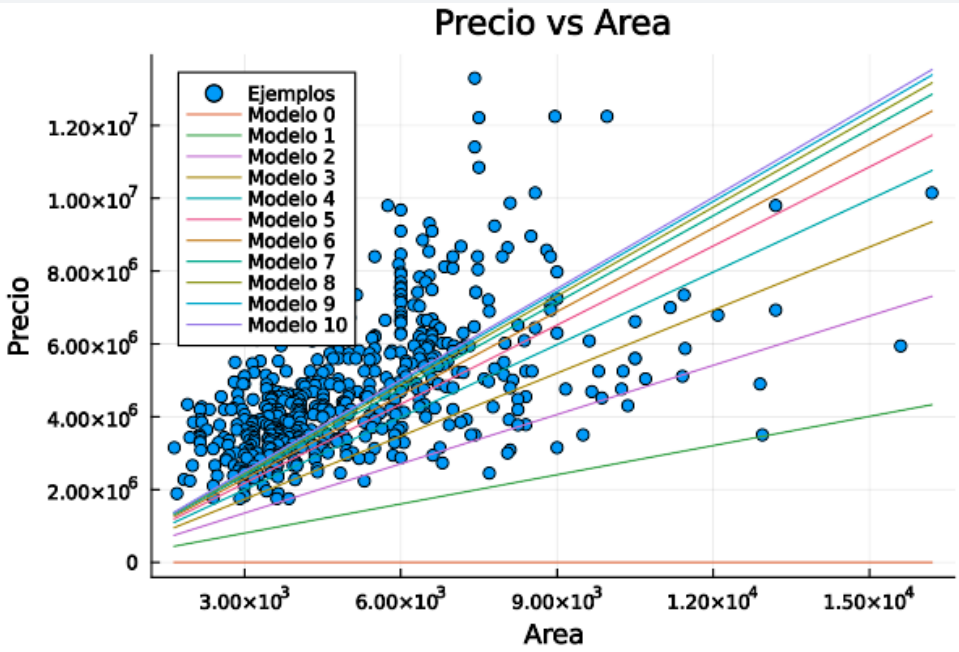
```
coste( , df.area, df.precio)
```

```
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.