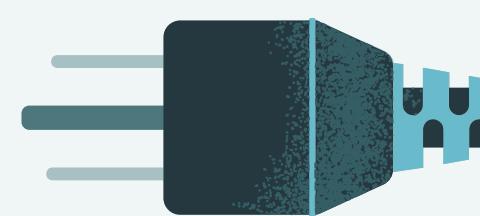
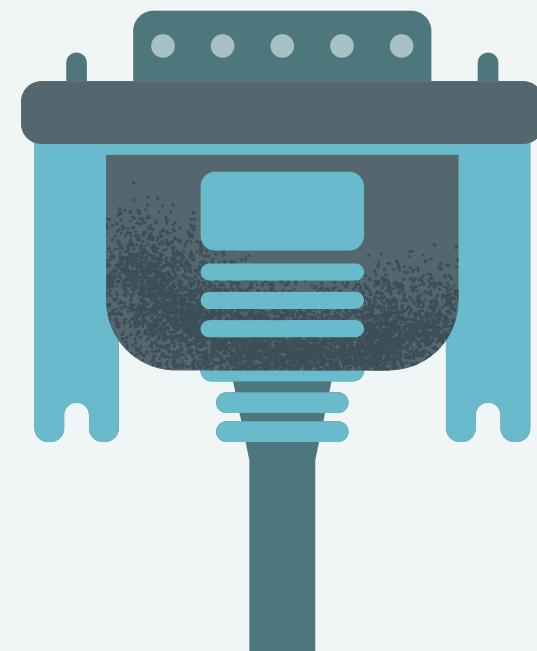
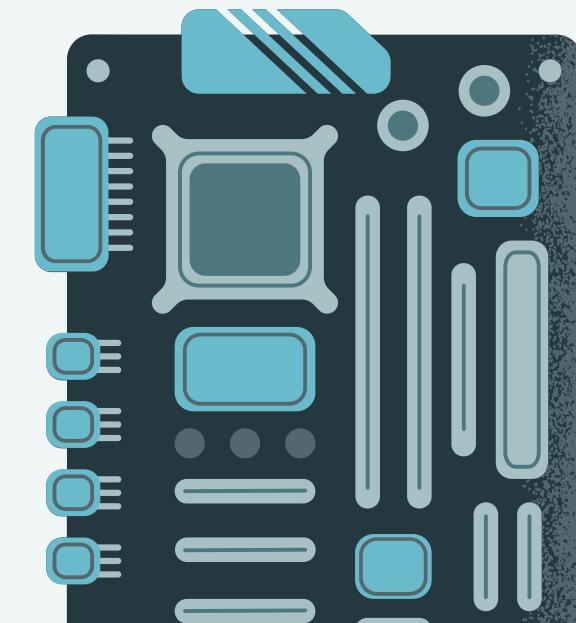
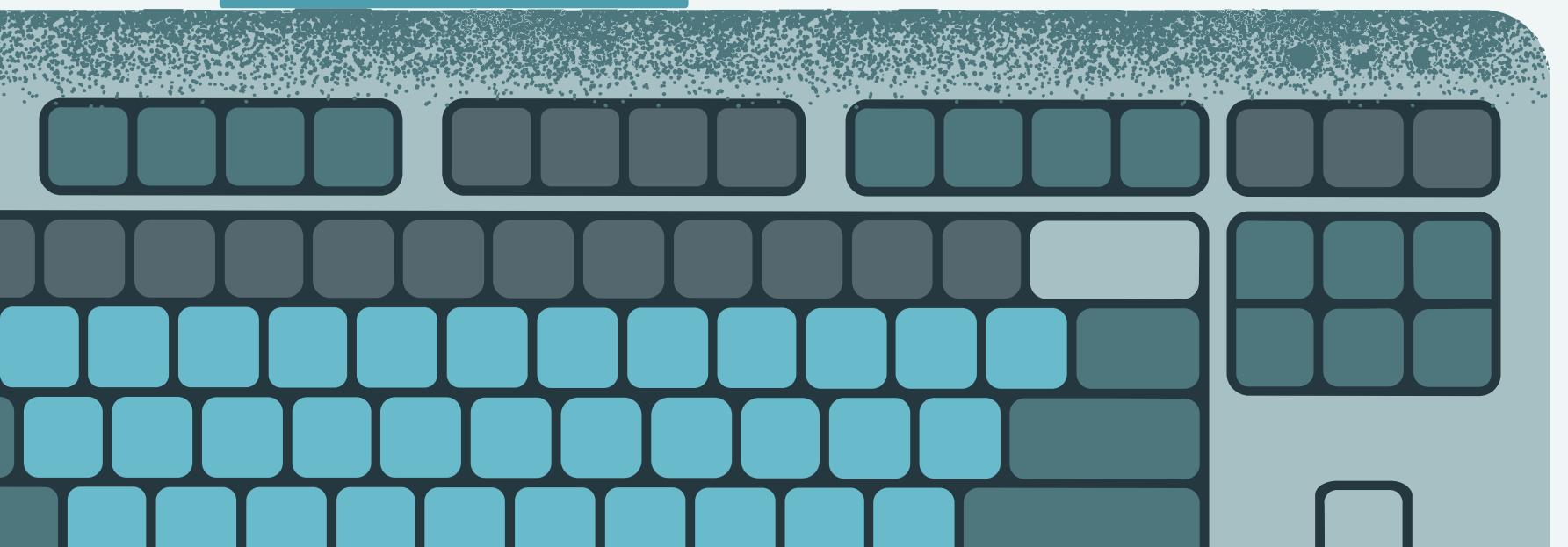




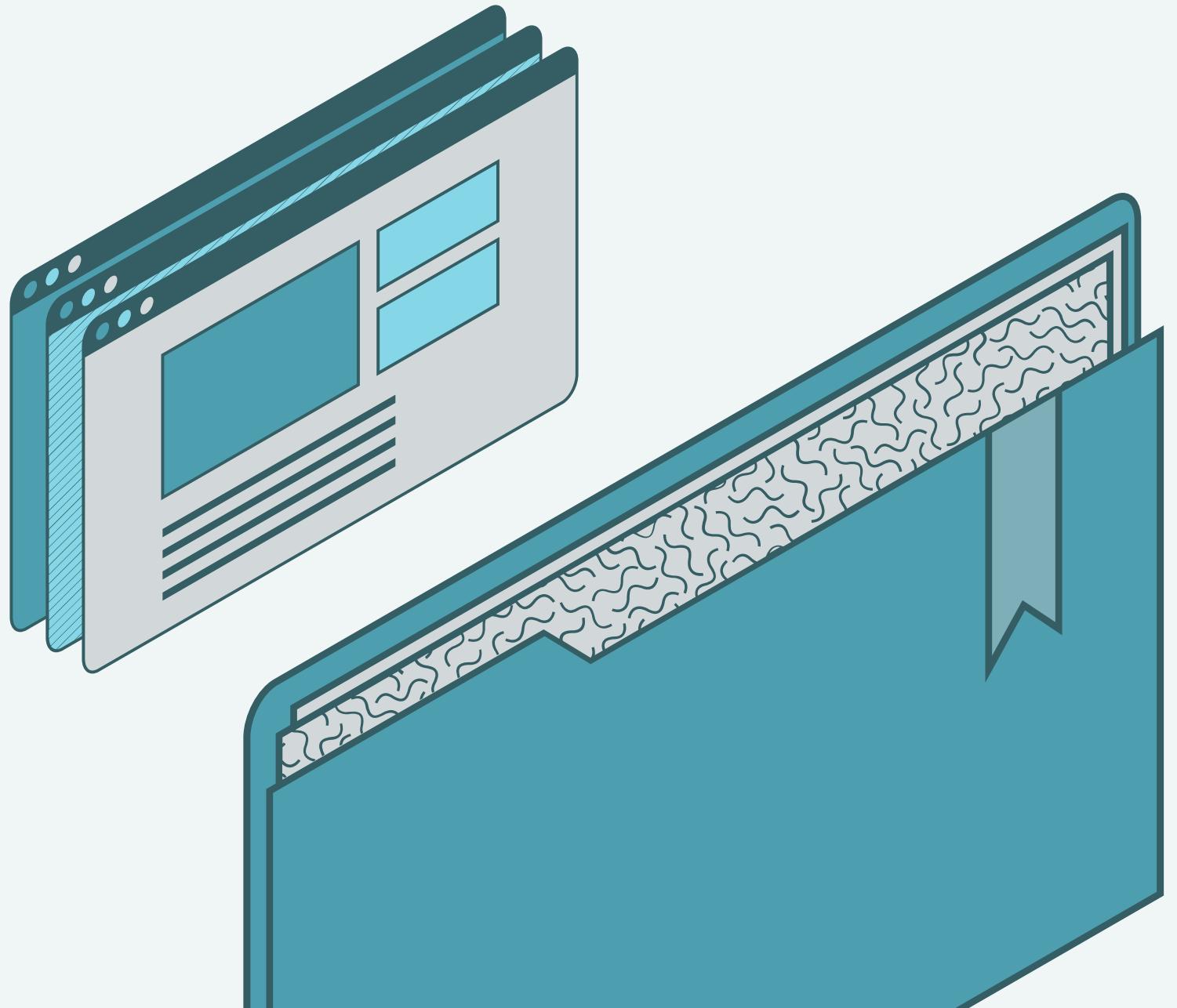
Pontificia Universidad
JAVERIANA
Bogotá

PROYECTO C++

Juan David Delgado Burbano
Nicolas Martinez
Camila Ariza



íNDIC



01. Introducción

02. Análisis de datos

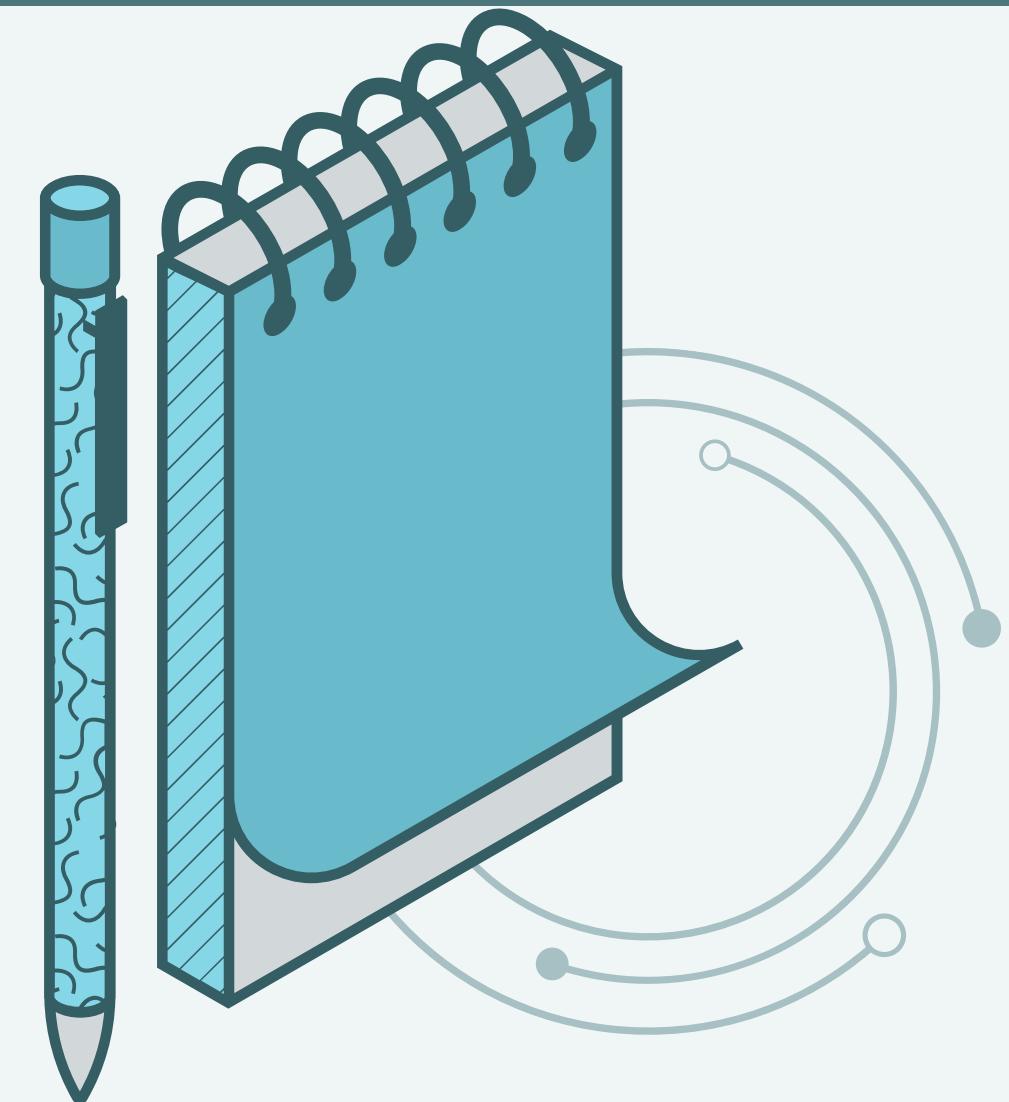
03. Código

04. Validación del modelo

05. Conclusiones

INTRODUCCIÓN

- el proyecto de c++ trataba de elaborar un código en c++, en conjunto de archivos.h, que lea un CSV con datos hacerca de venta de casas, departamentos, ect. con sus respectivos datos como pais, ciudad, barrio, #habitaciones, superficie total, precio metro cuadrado, con el objetovo de leer CSV, para crear una regrecion lineal basica que prediscá el precio de una vivienda a base del numero de habitaciones de esta.



ANALISIS DATA

2

ANALISIS DATA

- El archivo original de csv tendría (414.120) datos, de los cuales se veían algo así:

ANALISIS DATA

- con los siguientes datos, separados por comas:

- Fecha de inicio,
- fecha terminando
- ,creado en ,
- lat,
- lon ,
- pais,
- ciudad,
- barrio ,
- #habitaciones
- #baños,
- #alcobas,
- superficie total,
- superficie cubierta ,
- precio ,
- usd ,
- nombre
- ,descripción de la propiedad ,

- de los cuales todos esos datos eran completamente innecesarios para la elaboración del modelo, como fechas, descripciones, lat, lon, superficie cubierta, nombre de la propiedad,

ERRORES EN CSV

- Además el csv contenía errores, que provocaban una mala lectura del csv, como títulos en filas, o datos faltantes, o caracteres en variables numéricas:

84956	18/01/20	31/12/99	18/01/20	-34.499.037	-584.845.681.227 Argentina	Bs.As. G.B.
84957	18/01/20	6/02/20	18/01/20	-3.477.667.279.999.990	-584.891.578 Argentina	Bs.As. G.B.
84958	 			-346.410.239	-584.236.034 Argentina	Bs.As. G.B.
84959	 Oficina Barracas: Av. Martin García 560 Local 5 -				-58.447.872.100.000.000 Argentina	Capital Fed
84960	 Oficina San Telmo: Carlos Calvo 691 -					
84961	 Oficina La Boca/Catalinas Sur: 20 de Septiembre 314 -					
84962	 WhatsApp: 11					
84963	 					
84964	 La descripción expensas impuestos servicios y medidas han sido proporcionados por el producto del paso del tiempo. Previo a la realización de cualquier tipo de interacción requiriendo					
84965	18/01/20	31/12/99	18/01/20	-346.100.742	-58.480.362 Argentina	Capital Fed
84966	18/01/20	31/12/99	18/01/20	-344.813.853.289	-585.562.483.261 Argentina	Bs.As. G.B.
84967	 					
84968	 SAN ISIDRO: 11-					
84969	 BANCALARIO/ ND: 11-					
84970	 PILAR: 11-					
84971	 PILAR DELESTE: 11-					
84972	 VILLA NUEVA: 11-					
84973	 ESCOBAR / VN: 11-					
84974	 					
84975	 					
84976	 Martilleros titulares:					
84977	 Julia E Local comercio Venta					
84978	18/01/20	31/12/99	18/01/20	-34.634.085.799.999.900	-5.836.068.829.999.990 Argentina	Capital Fed
84979	18/01/20	31/12/99	18/01/20	-345.611.493	-584.809.377 Argentina	Capital Fed
84980	18/01/20	31/12/99	18/01/20	-346.956.019	-583.185.760 Argentina	Bs.As. G.B.
84981	18/01/20	31/12/99	18/01/20	-346.580.423	-586.918.208 Argentina	Bs.As. G.B.
84982	18/01/20	31/12/99	18/01/20	-344.733.829	-585.525.425 Argentina	Bs.As. G.B.
84983	 					
84984	 Matrícula CMPCSI 5964					
84985	 Matrícula PH Venta					
84986	18/01/20	31/12/99	18/01/20	-3.459.132.520.000.000	-58.508.053 Argentina	Capital Fed
84987	 					
84988	 Contactenos:					
84989	 					

62	228000
71	245000
63	175000
197	6,00E+05
45	89000
52	120000

	Palermo	3	73	310000	
	Palermo	3	69	265000	
	Palermo	3	63	155000	
	Palermo	3	58	158000	
	Palermo	3	87	220000	
	Palermo	3	68	335000	
	Palermo	3	66	184900	
	Palermo	3	55	240000	
	Palermo	3	70	220000	
	Palermo	3	NA	236000	
	Palermo	3	66	184900	
	Palermo	3	65	134000	
	Palermo	3	90	399000	
	Palermo	3	64	145000	
	Palermo	3	82	334000	
	Palermo	3	213	318000	
	Palermo	3	66	174000	
	Palermo	3	72	215000	
	Palermo	3	56	189000	
	Palermo	3	78	2,00E+05	
		79		2,00E+05	

ERRORES EN CSV

- por lo que se elaboro un código en R para arreglar la base de datos:

*Este código busca eliminar, en las columnas los elementos NA y los números con notación científica.

```
8 library(dplyr)
9
10 file_path <- "/Users/nicolasmartinezdelgadillo/Desktop/Limpia/DS_Proyecto_01_Datos_Properati.csv"
11 data <- read.csv(file_path, stringsAsFactors = FALSE)
12
13 data <- data %>%
14   mutate(across(everything(), ~ trimws(.)))
15
16 data <- data %>% select(property_type, l1, l2, l3, rooms, surface_total, price)
17
18 output_file <- "/Users/nicolasmartinezdelgadillo/Desktop/Limpia/DS_Proyecto_01_Datos_Properati_Limpio.csv"
19 write.csv(data, output_file, row.names = FALSE)
20
21
```

```
4:14 (Top Level) ▾
Console Terminal × Background Jobs ×
R 4.4.0 · ~/Desktop/Datos/ ↵
cannot open file '/Users/nicolasmartinezdelgadillo/Desktop/DS_Proyecto_01_Datos_Properati.csv': no such file or directory
library(dplyr)

# Cargar el archivo CSV
file_path <- "/Users/nicolasmartinezdelgadillo/Desktop/Limpia/DS_Proyecto_01_Datos_Properati.csv"
data <- read.csv(file_path, stringsAsFactors = FALSE)

# Limpiar los espacios en blanco de las columnas
data <- data %>%
  mutate(across(everything(), ~ trimws(.)))

# Seleccionar solo las columnas necesarias
data <- data %>% select(property_type, l1, l2, l3, rooms, surface_total, price)

# Guardar el archivo limpio
output_file <- "/Users/nicolasmartinezdelgadillo/Desktop/Limpia/DS_Proyecto_01_Datos_Properati_Limpio.csv"
write.csv(data, output_file, row.names = FALSE)
```

ERRORES EN CSV

- por lo que se elaboro un código en R para arreglar la base de datos:

Este código busca en el csv en la carpeta del escritorio, luego limpia la base de datos eliminando las columnas y filas vacías y luego elimina todas las columnas , menos las columnas de los datos escogidos:

property_type	I1	I2	I3	rooms	surface_total	price
Departamento	Argentina	Capital Federal	San Cristobal	7	140	153000

```
library(dplyr)
file_path <- "/Users/nicolasmartinezdelgadillo/Desktop/Limpia/DS_Proyecto_01_Datos_Properati_Limpio.csv"
df <- read.csv(file_path, stringsAsFactors = FALSE)
df$price <- as.numeric(df$price)
df_cleaned <- df %>%
  na.omit() %>%
  filter(price < 1e6)
cleaned_file_path <- "/Users/nicolasmartinezdelgadillo/Desktop/Limpia/DS_Proyecto_01_Datos_Properati_Limpio2.csv"
write.csv(df_cleaned, cleaned_file_path, row.names = FALSE)
```

ERRORES EN CSV

- Por ultimo, la tabla ya esta limpia, cada fila tendría 7 variables separadas por comas, y tendría una cantidad de 123517 filas el csv en total; cullos datos son usables para el modelo:

property_type,l1,l2,l3,rooms,surface_total,price
Departamento,Argentina,Capital Federal,Boedo,2,68,149000
PH,Argentina,Capital Federal,Boedo,2,70,159000
PH,Argentina,Capital Federal,Palermo,2,45,125000
PH,Argentina,Capital Federal,Palermo,2,85,295000
PH,Argentina,Bs. As. G.B.A. Zona Sur,La Plata,2,50,40000
PH,Argentina,Capital Federal,Villa Crespo,2,56,150000
PH,Argentina,Capital Federal,Villa Crespo,2,70,159500
PH,Argentina,Capital Federal,Villa Crespo,2,70,159500
PH,Argentina,Capital Federal,Parque Patricios,1,45,89000
PH,Argentina,Capital Federal,Parque Patricios,1,45,89000
PH,Argentina,Capital Federal,Villa Pueyrredón,2,66,170000
Departamento,Argentina,Capital Federal,Boedo,2,68,149000
Departamento,Argentina,Capital Federal,Boedo,2,50,115000

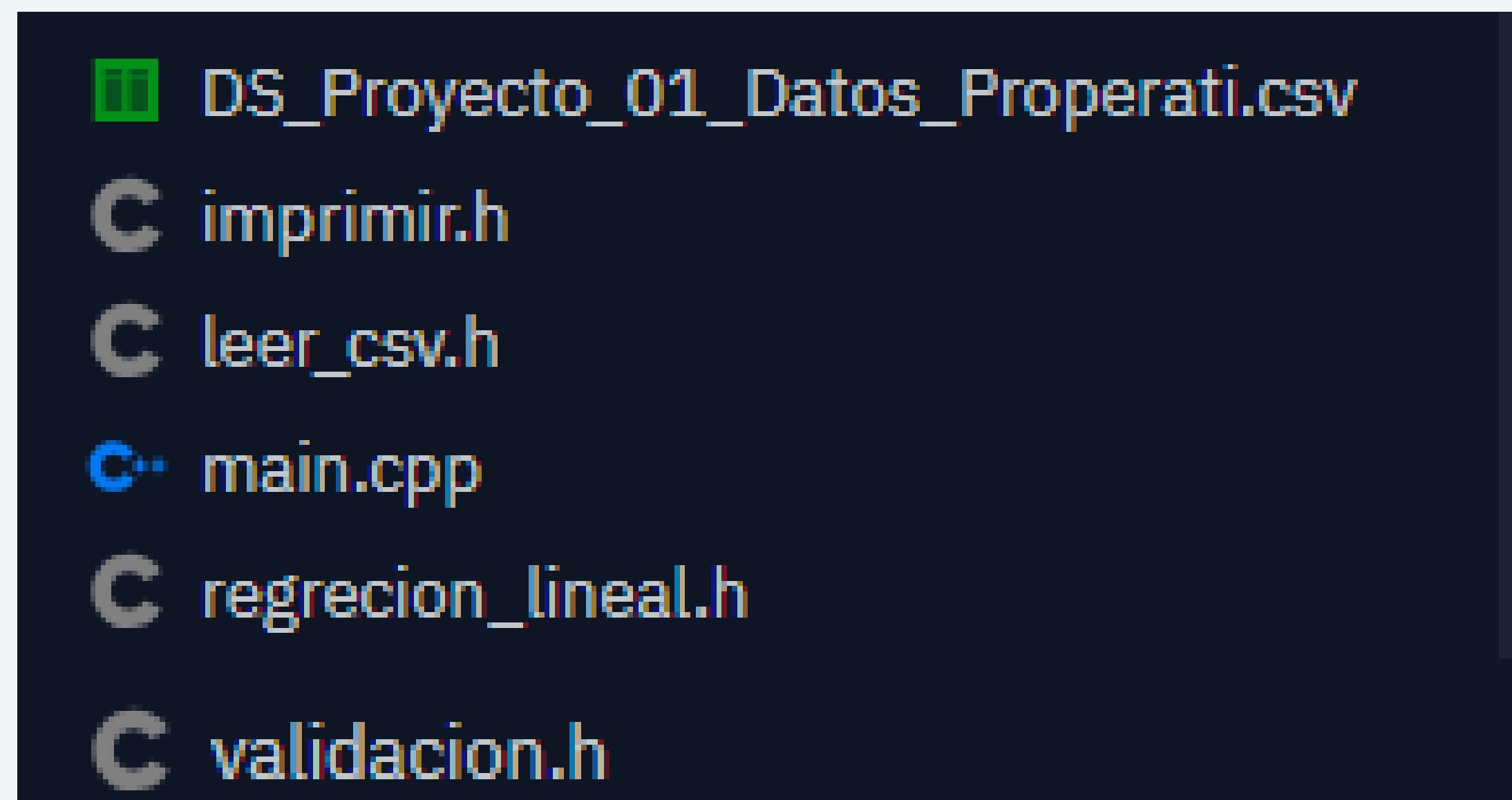


CODIGO

3



FORMATO



LÉER CSV

```
1 #ifndef LEER_CSV_H
2 #define LEER_CSV_H
3
4 #include <iostream>
5 #include <fstream>
6 #include <sstream>
7
8 using namespace std;
```

```
// Función para contar el número de líneas en un archivo CSV
int contarLineasCSV(const char* nombreArchivo) {
    ifstream archivo(nombreArchivo);
    int n = 0;
    char c;

    // Leer el archivo carácter por carácter
    while (archivo.get(c)) {
        if (c == '\n') {
            n++;
        }
    }

    // Añadir una línea si el archivo no termina con un salto de línea
    archivo.clear();
    archivo.seekg(0, ios::end);
    if (archivo.tellg() > 0) { // Si el archivo no está vacío
        archivo.seekg(-1, ios::end);
        if (archivo.get() != '\n') {
            n++;
        }
    }
}

return n;
}
```

Esta función cuenta las líneas que existen en el csv, leyendo carácter por carácter, y un contador (n) aumenta cada que hay un salto de linea, y se agrega un espacio mas al final

LÉER CSV

```
struct todo {  
    string id;  
    string l1;  
    string l2;  
    string l3;  
    float habitaciones;  
    float superficie;  
    float precio;  
};
```

Se crea una estructura de tipo todo, en donde se ubican cada variable que utilizaremos en el proyecto

```
int a = contarLineasCSV("DS_Proyecto_01_Datos_Properati.csv");  
todo *arr = new todo[a];
```

a una variable (INT) llamada a, se da el valor de #de filas csv, para crear un arreglo dinámico de tipo todo.

LÉER CSV

Se crea una función para llenar el arreglo dinámico (arr) creado anteriormente, leyendo coma por coma el csv, y retorna el arreglo lleno; además convierte las variables de string a numéricas con stoi y stof:

```
todo funcionllenar(todo *arr, int a)
{
    ifstream archivo("DS_Proyecto_01_Datos_Properati.csv");

    if (!archivo.is_open()) {
        cout << "Error al abrir el archivo." << endl;
    }

    string linea;
    int i = 0;

    // Saltar la primera línea que contiene los encabezados
    getline(archivo, linea);

    // Leer el archivo línea por línea
    while (getline(archivo, linea) && i < a) {
        stringstream ss(linea);
        string valor;
```

```
// Leer y asignar cada valor separado por comas
getline(ss, arr[i].id, ',');                                // Leer id (tipo de propiedad)
getline(ss, arr[i].l1, ',');                                // Leer l1
getline(ss, arr[i].l2, ',');                                // Leer l2
getline(ss, arr[i].l3, ',');                                // Leer l3

// Convertir los valores numéricos
-----
getline(ss, valor, ',');                                  // Leer (habitaciones)
arr[i].habitaciones = stof(valor);

getline(ss, valor, ',');                                  // Leer (superficie)
arr[i].superficie = stof(valor);

getline(ss, valor, ',');                                  // Leer (precio)
arr[i].precio = stof(valor);
-----

i++;
}

archivo.close();

94     return *arr;
95
96
97 }
98
99
100
101
102 #endif
```

REGRESIÓN LINEAL

```
#ifndef REGRESION_LINEAL_H
#define REGRESION_LINEAL_H

#include "imprimir.h"
#include "leer_csv.h"
#include <iostream>
#include <cmath>
#include <iomanip> // Para formatear la salida

using namespace std;
```

- crea arreglos dinamicos de variable independiente (# habitaciones) y variable dependiente (precio), y los llena con el arreglo (arr) en un bucle.

- Es una unica funcion de regrecion lineal, la cual recibe por parametro el arreglo llenado anteriormente de tipo todo (arr) y (n) que es el numero de filas del csv.

```
void regresion(todo* arr, int n) {
    // Crear arreglos dinámicos para almacenar las habitaciones y precios
    float* habi = new float[n];
    float* preci = new float[n];

    for (int r = 0; r < n; r++) {
        habi[r] = arr[r].habitaciones;
        preci[r] = arr[r].precio;
    }
```

REGRESIÓN LINEAL

```
// Normalizar los datos (entre 0 y 1)
float max_habi = habi[0], max_preci = preci[0];
float min_habi = habi[0], min_preci = preci[0];

for (int i = 1; i < n; ++i) {
    if (habi[i] > max_habi) max_habi = habi[i];
    if (preci[i] > max_preci) max_preci = preci[i];
    if (habi[i] < min_habi) min_habi = habi[i];
    if (preci[i] < min_preci) min_preci = preci[i];
}

for (int i = 0; i < n; ++i) {
    habi[i] = (habi[i] - min_habi) / (max_habi - min_habi); // Normalización
    preci[i] = (preci[i] - min_preci) / (max_preci - min_preci);
}
```

- Primero, se buscan los valores máximos y mínimos de los datos de habitaciones y precios.
- Luego, se realiza la normalización: en donde cada valor se transforma según la fórmula:

$$\text{valor_normalizado} = \frac{\text{valor}-\text{mínimo}}{\text{máximo}-\text{mínimo}}$$

Esto ajusta los datos para que todos los valores estén entre 0 y 1.

- LA Normalización:
.se hace para trabajar con las mismas escalas de datos.

REGRESIÓN LINEAL

```
// Parámetros iniciales de la regresión
float m = 0.0; // Pendiente
float b = 0.0; // Intercepto
float alpha = 0.001; // Tasa de aprendizaje ajustada
float mse = 0.0; // Error cuadrático medio
float mse_objetivo = 0.001; // Umbral para detener el descenso de gradiente
int max_iteraciones = 20000; // Límite máximo de épocas
int iteraciones = 0;
```

La regresión lineal busca una relación de la forma $y = m \cdot x + b$, donde m es la pendiente y b es el intercepto eje y .

REGRESIÓN LINEAL

```
// Descenso de gradiente con criterio de convergencia basado en MSE
do {
    float gradiente_m = 0.0;
    float gradiente_b = 0.0;
    mse = 0.0;

    for (int i = 0; i < n; ++i) {
        float prediccion = m * habi[i] + b;
        float error = preci[i] - prediccion;
        mse += error * error; // Sumar el cuadrado de los errores
        gradiente_m += -2 * habi[i] * error;
        gradiente_b += -2 * error;
    }
}
```

El bucle for recorre todos los puntos de datos (habitaciones y precios) y realiza las siguientes operaciones para cada dato:

- **Predicción:** Para cada dato i , se realiza una predicción usando la ecuación de la recta:
$$\text{prediccion} = m \cdot \text{habi}[i] + b$$

 $\text{habi}[i]$ es el número de habitaciones para el dato i , m es la pendiente actual y b es el intercepto actual.

- **Cálculo del error:** El error es la diferencia entre el valor real del precio $\text{preci}[i]$ y la predicción. Esto nos indica cuánto se desvía la predicción del valor real.
- **MSE (Error cuadrático medio):** Se suma el cuadrado del error al MSE. Esto se hace para que el error tenga siempre un valor positivo y penalice los errores más grandes de manera más significativa. Al final del bucle, el MSE contendrá la suma de todos los errores al cuadrado.
- **Gradiente de m y B :** Se actualiza el gradiente de m sumando $-2 \cdot \text{habi}[i] \cdot \text{error}$. Este valor indica cuánto debe ajustarse m para reducir el error en la siguiente iteración, lo mismo para B .

REGRESIÓN LINEAL

```
// Calcular el MSE promedio  
mse /= n;  
  
// Actualizar los parámetros m y b  
m -= (gradiente_m / n) * alpha;  
b -= (gradiente_b / n) * alpha;  
  
iteraciones++;
```

- Al dividir el **error cuadrático acumulado** entre **n** (el número de datos), se obtiene el **MSE promedio**. Esto nos da una medida de cuán buena o mala es la predicción en general, y el objetivo del descenso de gradiente es minimizar este valor.
- Después de calcular los gradientes para **m** y **b**, se ajustan estos parámetros restándoles una fracción de los gradientes.

- El factor **alpha** es la tasa de **aprendizaje**, que controla cuánto cambian **m** y **b** en cada iteración. Si **alpha** es muy grande, los ajustes serán abruptos y podrían no converger; si es muy pequeño, el algoritmo podría tardar mucho en encontrar la solución óptima.
- La división por **n** es para promediar los gradientes, de modo que la actualización no dependa del tamaño del **dataset**.
- Se incrementa el número de **iteraciones (epochas)** en cada paso del ciclo. Esto permite controlar el número de veces que el algoritmo ajusta **m** y **b**.

```
} while (mse > mse_objetivo && iteraciones < max_iteraciones);
```

el bucle continuara hasta que haga 20.000 interacciones (epochas) o hasta que el mse sea menor que el mse objetivo

REGRESIÓN LINEAL

```
// Desnormalizar los parámetros m y b  
m = m * (max_preci - min_preci) / (max_habi - min_habi);  
b = b * (max_preci - min_preci) + min_preci - m * min_habi;
```

los datos de las habitaciones (habi) y los precios (preci) se normalizados para facilitar el proceso de regresión lineal utilizando descenso de gradiente.

Una vez que el modelo ha ajustado los parámetros m (pendiente) y b (intercepto), es necesario desnormalizar los resultados para que las predicciones y la ecuación final de la regresión tengan sentido en el contexto real.

IMPRÉCION

```
// Imprimir resultados de la regresión en formato de tabla
cout << "Desea ver los resultados de la regresión lineal como: " << endl;
cout << "#epocas, ecuación recta, error cuadrático?" << endl;
cout << "si o no: ";
string respuesta2;
cin >> respuesta2;

if (respuesta2 == "si" || respuesta2 == "SI" || respuesta2 == "Si") {
    imprimirTablaRegresion(iteraciones, m, b, mse);
}
```

- se pregunta si el usuario desea ver los resultados de la regresión como #epocas, ecuación recta, error cuadrático.

Resultados de la regresión lineal:

Epocas	10000
Ecuación de la recta:	
y = 56676.4375 * habitaciones + 37471.1172	
Error cuadrático medio (MSE): 16765711360.0000	

- Imprecision:

IMPRÉCION

```
#ifndef imprimir_h
#define imprimir_h

#include "regresion_lineal.h"
#include "leer_csv.h"
#include <iostream>
#include <cmath>
#include <iomanip> // Para formatear la salida
|
using namespace std;

void imprimirTablaRegresion(int epochas, float m, float b, float mse) {
    cout << "Resultados de la regresión lineal:" << endl;
    cout << "_____ | " << epochas << endl;
    cout << "Epocas | " << endl;
    cout << "Ecuación de la recta: " << endl;
    cout << "y = " << fixed << setprecision(4) << m << " * habitaciones + " << b << endl;
    cout << "_____ | " << endl;
    cout << "Error cuadrático medio (MSE): " << mse << endl;
    cout << "_____ | " << endl << endl;
}
```

- Utilizando imprecision.h y llamando a la función (imprimirtablaregresion) se imprime de la manera que se vio anteriormente.

IMPRÉCION

```
// Preguntar al usuario cuántas predicciones quiere hacer
int num_predicciones;
cout << "¿Cuántas predicciones de precio deseas realizar?: ";
cin >> num_predicciones;

// Arreglos para almacenar habitaciones y predicciones
float* habitaciones_pred = new float[num_predicciones];
float* precios_pred = new float[num_predicciones];

// Hacer las predicciones para el número de habitaciones ingresado
for (int i = 0; i < num_predicciones; ++i) {
    cout << "Ingrese el número de habitaciones para la predicción " << i + 1 << ": ";
    cin >> habitaciones_pred[i];

    // Predecir precio y desnormalizar el resultado
    precios_pred[i] = m * habitaciones_pred[i] + b;
}
```

- se pregunta al usuario, cuantas predicciones desea hacer.
con ello se hacen dos arreglos dinamicos, para hacer la predicción, y se remplaza en la ecuación de la recta ya encontrada.

IMPRÉCION

```
// Imprimir las predicciones en formato de tabla
imprimirTablaPredicciones(num_predicciones, habitaciones_pred, precios_pred);

// Liberar la memoria de los arreglos dinámicos
delete[] habi;
delete[] preci;
delete[] habitaciones_pred;
delete[] precios_pred;
}

#endif // REGRESION_LINEAL_H
```

- se imprime el resultado de las predicciones, y se elimina memoria, de los arreglos dinamicos utilizados.

IMPRECION

```
void imprimirTablaPredicciones(int num_predicciones, float* habitaciones, float* precios) {
    cout << "Predicciones de precio:" << endl;
    cout << "-----" << endl;
    cout << "Habitaciones | Precio estimado ($)" << endl;
    cout << "-----" << endl;
    for (int i = 0; i < num_predicciones; ++i) {
        cout << fixed << setprecision(2) << habitaciones[i] << " | $" << precios[i] << endl;
    }
    cout << "-----" << endl << endl;
}

#endif // imprimir.h
```

- se llama a la función de imprimir predicciones,

Predicciones de precio:

Habitaciones	Precio estimado (\$)
1.00	\$ 94147.55
2.00	\$ 150824.00
3.00	\$ 207500.44
4.00	\$ 264176.88
5.00	\$ 320853.31
6.00	\$ 377529.75
7.00	\$ 434206.19

- imprecision:

MAIN.CPP

```
2 #include "leer_csv.h"
3 #include "regrecion_lineal.h"
4 #include "validacion.h"

5
6 #include <iostream>
7 #include <fstream>
8 #include <cmath>
9
0 #include <vector>
1 #include <iomanip>

2
3 using namespace std;
4
```

```
int main( )
{
    // Especifica el nombre del archivo CSV
    const char* nombreArchivo = "DS_Proyecto_01_Datos_Properati.csv";

    // Llama a la función para contar las líneas y almacena el resultado en n
    int n = contarLineasCSV(nombreArchivo);

    // Imprime el número de líneas
    cout << "El archivo tiene " << n << " líneas." << endl;

    // Aquí puedes crear arreglos dinámicos utilizando n

    // Llamar a la función para llenar el arreglo
    *arr = funcionllenar(arr, n);
```

MAIN.CPP

```
// Verificar que el arreglo haya sido llenado correctamente
cout << "DESEA VER SI EL ARREGLO SE LLENO CORRECTAMENTE ??????"<<endl;
cout << "si o no"<<endl;
string respuesta1, respuesta2;
cin>> respuesta1;
if (respuesta1== "si" || respuesta1 == "SI" || respuesta1 == "Si")
{
    if (arr != nullptr) {
        for (int i = 0; i < 1; i++) {
            cout << "Propiedad " << i + 1 << ":" << arr[i].id << ", "
                << arr[i].l1 << ", " << arr[i].l2 << ", " << arr[i].l3 << ", "
                << arr[i].habitaciones << " habitaciones, "
                << arr[i].superficie << " m2, $"
                << arr[i].precio << endl;
        }
    }
}
cout << " "<
```

```
// -----
//-----
//-----
// parte regrecion lineal:
regresion(arr, n);

delete[] arr; // Liberar memoria al final
}
```

MAIN.CPP

```
// Verificar que el arreglo haya sido llenado correctamente
cout << "DESEA VER SI EL ARREGLO SE LLENO CORRECTAMENTE ??????"<<endl;
cout << "si o no"<<endl;
string respuesta1, respuesta2;
cin>> respuesta1;
if (respuesta1== "si" || respuesta1 == "SI" || respuesta1 == "Si")
{
    if (arr != nullptr) {
        for (int i = 0; i < 1; i++) {
            cout << "Propiedad " << i + 1 << ":" << arr[i].id << ", "
                << arr[i].l1 << ", " << arr[i].l2 << ", " << arr[i].l3 << ", "
                << arr[i].habitaciones << " habitaciones, "
                << arr[i].superficie << " m2, $"
                << arr[i].precio << endl;
        }
    }
}
cout << " "<
```

```
// -----
//-----
//-----
// parte regrecion lineal:
regresion(arr, n);

delete[] arr; // Liberar memoria al final
}
```

MAIN.CPP

```
cout << "¿Deseas dividir el dataset para probar el modelo? (si/no): ";
string respuesta3;
cin >> respuesta3;

if (respuesta3 == "si" || respuesta3 == "SI" || respuesta3 == "Si") {
    todo* entrenamiento = nullptr;
    todo* prueba = nullptr;
    int tam_entrenamiento = 0;
    int tam_prueba = 0;

    // Dividir el dataset en 80% entrenamiento y 20% prueba
    dividirDataset(arr, n, 0.8, entrenamiento, prueba, tam_entrenamiento, tam_prueba);

    // Realizar la regresión con los datos de entrenamiento
    regresion(entrenamiento, tam_entrenamiento);

    // Probar el modelo con los datos de prueba
    probarModelo(prueba, tam_prueba);

    // Liberar memoria
    delete[] entrenamiento;
    delete[] prueba;
}
```

- se llama a la función de dividir dataset que divide la cantidad de datos 80% para entrenamiento, y 20% prueba.
- se llama a la función de regresión mandando como parámetro los datos de prueba, y por último se llama a la función de probar modelo, donde se manda por parámetro los arreglos de prueba.

VALIDACION MODELO

4

VALIDACION

C `validacion.h`

- **Por ultimo se valida el modelo de regrecion linea al:**

Dividir el dataset en un conjunto de entrenamiento y un conjunto de prueba, dividir los datos originales (habitaciones y precios) en dos conjuntos:

- Conjunto de entrenamiento: Se usa para entrenar el modelo, es decir, ajustar los valores de mmm y bbb utilizando el descenso de gradiente.
- Conjunto de prueba: Se usa para probar el modelo después de que ha sido entrenado. Esto te permite validar si el modelo puede hacer predicciones correctas para datos que no ha visto antes.
- usar el 80% de los datos para entrenamiento y el 20% restante para prueba.

VALIDACION

```
#ifndef VALIDACION_H
#define VALIDACION_H

#include <iostream>
#include <cmath>
#include "regrecion_lineal.h"

using namespace std;
```

Reciviendo como parametro el arreglo lleno con los datos ya leidos del csv (arr) , tambien el numero de filas del csv, y recibe los arreglos (punteros) de entrenamiento y prueba, para hacer arreglos dinamicos de ellos, y llenarlos con un ford con el 20% y el 80% de los datos,

```
void dividirDataset(todo* arr, int n, float porcentaje_entrenamiento, todo*& entrenamiento, todo*& prueba, int& tam_entrenamiento, int& tam_prueba)
{
    tam_entrenamiento = static_cast<int>(n * porcentaje_entrenamiento);
    tam_prueba = n - tam_entrenamiento;

    entrenamiento = new todo[tam_entrenamiento];
    prueba = new todo[tam_prueba];

    for (int i = 0; i < tam_entrenamiento; ++i) {
        entrenamiento[i] = arr[i];
    }

    for (int i = 0; i < tam_prueba; ++i) {
        prueba[i] = arr[tam_entrenamiento + i];
    }
}
```

VALIDACION

```
#ifndef VALIDACION_H
#define VALIDACION_H

#include <iostream>
#include <cmath>
#include "regrecion_lineal.h"

using namespace std;
```

Reciviendo como parametro el arreglo lleno con los datos ya leidos del csv (arr) , tambien el numero de filas del csv, y recibe los arreglos (punteros) de entrenamiento y prueba, para hacer arreglos dinamicos de ellos, y llenarlos con un ford con el 20% y el 80% de los datos,

```
void dividirDataset(todo* arr, int n, float porcentaje_entrenamiento, todo*& entrenamiento, todo*& prueba, int& tam_entrenamiento, int& tam_prueba)
{
    tam_entrenamiento = static_cast<int>(n * porcentaje_entrenamiento);
    tam_prueba = n - tam_entrenamiento;

    entrenamiento = new todo[tam_entrenamiento];
    prueba = new todo[tam_prueba];

    for (int i = 0; i < tam_entrenamiento; ++i) {
        entrenamiento[i] = arr[i];
    }

    for (int i = 0; i < tam_prueba; ++i) {
        prueba[i] = arr[tam_entrenamiento + i];
    }
}
```

VALIDACION

imprime los datos de entrenamiento haciendo su regresión lineal.

```
// Realizar la regresión con los datos de entrenamiento  
regresion(entrenamiento, tam_entrenamiento);  
  
// Probar el modelo con los datos de prueba  
probarModelo(prueba, tam_prueba);
```

imprime los datos de prueba haciendo su regresión lineal.

```
void probarModelo(todo* prueba, int tam_prueba) {  
    cout << "Probando el modelo con los datos de prueba..." << endl;  
  
    // Aquí se realiza la regresión con los datos de prueba  
    regresion(prueba, tam_prueba);  
  
    cout << "Prueba completada." << endl;  
}  
  
#endif // VALIDACION_H
```

VALIDACION

regresión original

```
Epochs | 20000  
Ecuación de la recta:  
y = 3791.6887 * habitaciones + 200955.9219  
Error cuadrático medio (MSE): 0.0219
```

Habitaciones	Precio estimado (\$)
7.00	\$ 227497.75

regresión prueba

```
Epochs | 20000  
Ecuación de la recta:  
y = 7857.1094 * habitaciones + 201363.5781  
Error cuadrático medio (MSE): 0.0240
```

Habitaciones	Precio estimado (\$)
7.00	\$ 256363.34

regresión entrenamiento

```
Epochs | 20000  
Ecuación de la recta:  
y = 3835.5237 * habitaciones + 197617.8906  
Error cuadrático medio (MSE): 0.0214
```

Habitaciones	Precio estimado (\$)
7.00	\$ 224466.56

IMPRÉCION FINAL

```
El archivo tiene 123517 líneas.  
DESEA VER SI EL ARREGLO SE LLENO CORRECTAMENTE ??????  
si o no  
si  
Propiedad 1: Departamento, Argentina, Capital Federal, San Cristobal, 7 habitaciones, 140 m2, $1  
53000  
  
Desea ver los resultados de la regresión lineal como:  
#epochas, ecuación recta, error cuadrático?  
si o no: si  
Resultados de la regresión lineal:  
-----  
Epochs | 20000  
-----  
Ecuación de la recta:  
y = 3791.6887 * habitaciones + 200955.9219  
-----  
Error cuadrático medio (MSE): 0.0219  
-----
```

IMPRECCION FINAL

```
¿Cuántas predicciones de precio deseas realizar?: 7
Ingrrese el número de habitaciones para la predicción 1: 1
Ingrrese el número de habitaciones para la predicción 2: 2
Ingrrese el número de habitaciones para la predicción 3: 3
Ingrrese el número de habitaciones para la predicción 4: 4
Ingrrese el número de habitaciones para la predicción 5: 5
Ingrrese el número de habitaciones para la predicción 6: 6
Ingrrese el número de habitaciones para la predicción 7: 7
Predicciones de precio:
```

Habitaciones	Precio estimado (\$)
1.00	\$ 204747.61
2.00	\$ 208539.30
3.00	\$ 212330.98
4.00	\$ 216122.67
5.00	\$ 219914.36
6.00	\$ 223706.06
7.00	\$ 227497.75

IMPRÉCION POR LA SHELL

UTILIZAMOS EL SIGUIENTE
COMANDO PARA COMPILAR EL
ARCHIVO MAIN.CPP

```
~/taller-4$ g++ -o proyecto main.cpp
~/taller-4$ ./proyecto
El archivo tiene 123517 líneas.
DESEA VER SI EL ARREGLO SE LLENO CORRECTAMENTE ??????
sí o no
sí
Propiedad 1: Departamento, Argentina, Capital Federal, San Cristobal, 7 habita-
ciones, 140 m2, $153000

Desea ver los resultados de la regresión lineal como:
#epocas, ecuación recta, error cuadrático?
sí o no: sí
Resultados de la regresión lineal:

Epocas | 20000

Ecuación de la recta:
y = 3791.6887 * habitaciones + 200955.9219

Error cuadrático medio (MSE): 0.0219

¿Cuántas predicciones de precio deseas realizar?: 1
Ingrese el número de habitaciones para la predicción 1: 7
Predicciones de precio:

Habitaciones | Precio estimado ($)

7.00 | $ 227497.75

¿Deseas dividir el dataset para probar el modelo? (si/no): sí
Desea ver los resultados de la regresión lineal como:
```

```
~/taller-4$ g++ -o proyecto main.cpp
~/taller-4$ ./proyecto
```

CONCLUSIONES

5

CONCLUSIONES

- Para concluir, el precio de la propiedad, segun # de habitaciones en toda argentina, se puede predecir con el modelo de regrecion lineal.
- El modelo puede mejorarse, al predecir los precios segun ciudades, o incluso barrios, de argentina, para tener una predicción mas precisa, con el modelo.

PROYECTO C++

VENTA DE INMUEBLES.

```
|   └── data
|       └── DS_Proyecto_01_Datos_Properati.csv
|
|   └── src
|       └── main.cpp
|
|   └── README.md
|
|   └── resultados
|       └── reporte_analisis.txt
|
└── presentacion
    └── presentacion_proyecto.pdf
```