

# MEMORIA PROYECTO

## Curso Data & Desarrollo Python

HouseParator



Nueva Consulta



### Calle de Los Limones

- Área: 56
- Garajes: 1
- Chimeneas: 1
- Baños: 2
- ciudad: 2
- Pisos: 1
- Tipo Marmol: Blanco

**Incluye :** - Fibra telefónica - Jardín - Piscina - Puertas de cristal - Placas solares

Precio promedio: 59600.00 €

**Precio de venta: 80000.00 €**

Editar

Borrar



### Calle del Kiwi

- Área: 50
- Garajes: 1
- Chimeneas: 0
- Baños: 1
- ciudad: 1
- Pisos: 1
- Tipo Marmol: Negro

**Incluye :** - Jardín - Placas solares

Precio promedio: 28750.00 €

**Precio de venta: 25000.00 €**

Editar

Borrar



### Calle del Bambú

- Área: 56
- Garajes: 1
- Chimeneas: 0
- Baños: 1
- ciudad: 3
- Pisos: 1
- Tipo Marmol: Blanco

**Incluye :** - Red eléctrica - Fibra telefónica - Piscina

Precio promedio: 57650.00 €

**Precio de venta: 30000.00 €**

Editar

Borrar



### Calle del Ficus

- Área: 50
- Garajes: 1
- Chimeneas: 0
- Baños: 1
- ciudad: 2
- Pisos: 0
- Tipo Marmol: Indio

No incluye extras

Precio promedio: 12000.00 €

**Precio de venta: 40000.00 €**

Editar

Borrar

## Dataset:

<https://www.kaggle.com/datasets/greenwing1985/housepricing>

## Aplicación disponible en replit:

backhouses.jose-luislui964.repl.co

fronthouses.jose-luislui964.repl.co

**Fecha Entrega: 28/09/23**



## 1. - Memoria Descriptiva

### 1.1 Objetivo

El objetivo del proyecto ha sido realizar una aplicación que dadas las características de una vivienda se prediga un precio, comparándolo con un conjunto de datos de casas, y se informe al usuario de si el precio que va a pagar es mayor o menor que las viviendas que hay en el mercado con características similares.

El proyecto se compone de tres grandes ramas:

- Un modelo de regresión lineal que estudia las propiedades de las casas del conjunto de datos y entrena un modelo que dadas las características de un inmueble infiere el precio.
- Una página web donde el usuario introduce las características de la vivienda que quiere consultar y puede comparar el precio del inmueble con el generado por el modelo. Además de guardar, editar y borrar todas las consultas que haga.
- Un servidor que contesta las peticiones de la página web, almacena las consultas del usuario y utiliza el modelo de regresión lineal para inferir el precio.

El proyecto cuenta con un modelo de data en Python, un servidor backend en Python y un frontend con HTML, CSS y JavaScript.

### 1.2 Análisis de Datos

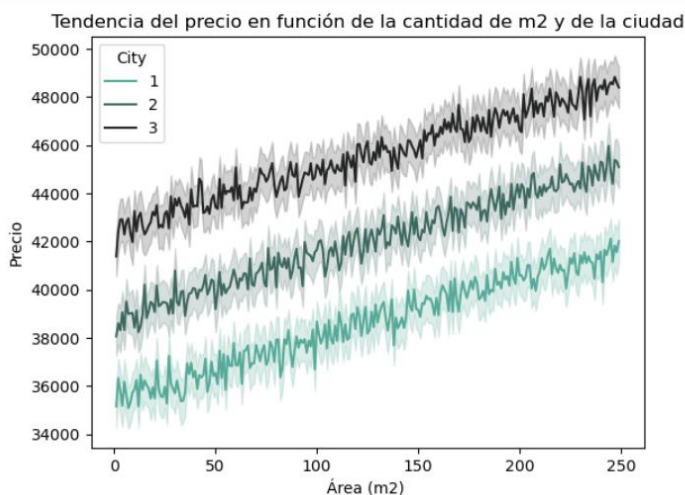
En el modelo de datos primero se hace un estudio de los datos que nos permite limpiarlos y prepararlos para el modelo. En este caso el conjunto de datos eran 500.000 entradas de casas de la India.

```
#Vemos una muestra de los datos
houses.head()
```

	Area	Garage	FirePlace	Baths	White Marble	Black Marble	Indian Marble	Floors	City	Solar	Electric	Fiber	Glass Doors	Swimming Pool	Garden	Prices
0	164	2	0	2	0	1	0	0	3	1	1	1	1	0	0	43800
1	84	2	0	4	0	0	1	1	2	0	0	0	1	1	1	37550
2	190	2	4	4	1	0	0	0	2	0	0	1	0	0	0	49500
3	75	2	4	4	0	0	1	1	1	1	1	1	1	1	1	50075
4	148	1	4	2	1	0	0	1	2	1	0	0	1	1	1	52400

```
#El área al ser una variable continua la vamos a estudiar con un gráfico de líneas, comparando las distintas ciudades
sns.lineplot(x='Area', y='Prices', data=houses, hue='City', palette="dark:#5A9_r")

plt.xlabel('Área (m2)')
plt.ylabel('Precio')
plt.title('Tendencia del precio en función de la cantidad de m2 y de la ciudad')
plt.show()
```



Una vez se han estudiado los datos se hace el modelo de regresión lineal y lo entrenamos:

```
#Asignamos las variables independientes y la dependiente(el precio depende de Las demás características)
X = houses.drop('Prices', axis=1)
y = houses['Prices']

#Creación del conjunto de entrenamiento, usamos el 20% de los datos para realizar los test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

#Creamos el modelo de regresión lineal
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()

# Entrenamos el modelo con los datos de entrenamiento (80%)
lin_reg.fit(X_train, y_train)

LinearRegression()

# Hacemos las predicciones utilizando solo las variables independientes de los casos de prueba
y_pred = lin_reg.predict(X_test)
```

Comprobamos que el nivel de error sea bajo y el modelo haga predicciones correctas:

## Mettricas

**RMSE** (Root Mean Squared Error):

```
[26]: from sklearn import metrics
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
print("RMSE Value: ",rmse)
```

RMSE Value: 7.320020373718509e-11

Como podemos ver el valor es muy cercano a cero por lo que podemos afirmar que el modelo se ajusta muy bien a los datos

**MSE** (Mean Squared Error):

```
[27]: print("MSE Value: ", metrics.mean_squared_error(y_test,y_pred))
```

MSE Value: 5.358269827165406e-21

Un valor bajo de MSE, como el que tiene este modelo, indica que tiene un bajo error cuadrático medio.

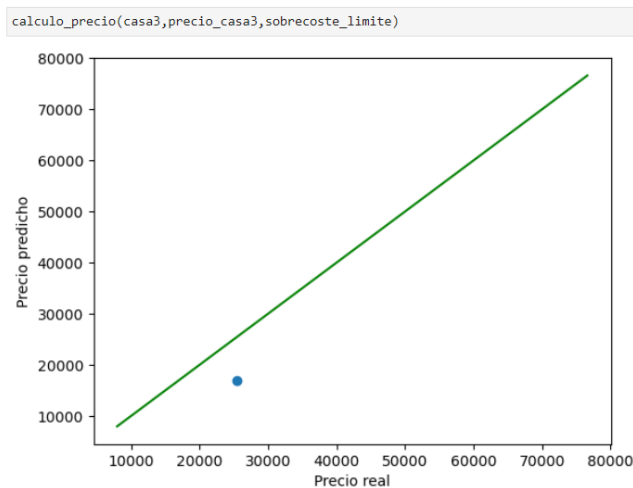
**Accuracy** (Precisión):

```
[28]: print("Accuracy Value: ", 1 - metrics.mean_absolute_error(y_test,y_pred)/np.mean(y))
```

Accuracy Value: 0.99999999999999986

Mide la capacidad del modelo para predecir correctamente los valores reales. Un valor alto de precisión, como el de nuestro modelo, indica que tiene un alto porcentaje de acierto

Una vez entrenado podemos exportar el modelo para posteriormente usarlo en el backend y predecir el precio de las consultas que nos hagan desde la web.

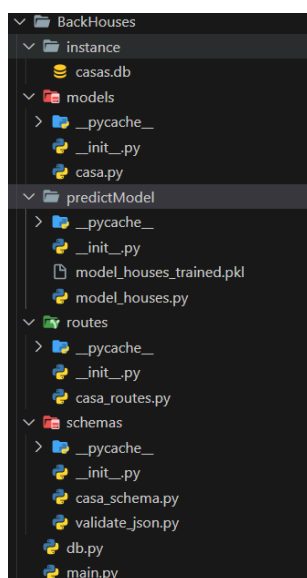


El precio promedio de una casa de esas características es: 25500.0  
Es 8500.0 euros más barato  
Opinión final: Es muy buena compra, su precio es más bajo que el mercado actual

### 1.3 Servidor Python

El código del servidor se ha organizado en varias carpetas donde se realiza una función distinta:

- Instance: Guarda la base de datos
- Models: Contiene la clase Casa donde se describe que forma tendrán los datos de este tipo en la base de datos SQLAlchemy
- PredictModel: Carga el modelo de regresión lineal y calcula el precio en función de las características
- Routes: Almacena los endpoint de Flask que permiten acceder desde fuera del servidor a las funcionalidades que implementa. Estas son el CRUD básico de una API
- Schemas: Contiene esquemas de validación de los datos que entran e impide que se creen con formatos inadecuados
- Main.py: Aquí se encuentra el proceso principal que coordina y ejecuta el proyecto



El modelo de datos describe las propiedades que tiene un objeto de tipo Casa en la base de datos SQLAlchemy (izquierda) y el modelo predictivo carga el modelo anteriormente generado y convierte los datos de entrada a un formato compatible con el modelo:

```
@dataclass
class Casa(db.Model):
    __tablename__ = "casa"
    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    nombre: Mapped[str] = mapped_column(String(255))
    area: Mapped[float] = mapped_column(Numeric(10,2))
    garaje: Mapped[float] = mapped_column(Numeric(10,2))
    chimenea: Mapped[float] = mapped_column(Numeric(10,2))
    aseos: Mapped[float] = mapped_column(Numeric(10,2))
    marmol_blanco: Mapped[float] = mapped_column(Numeric(10,2))
    marmol_negro: Mapped[float] = mapped_column(Numeric(10,2))
    marmol_indio: Mapped[float] = mapped_column(Numeric(10,2))
    pisos: Mapped[float] = mapped_column(Numeric(10,2))
    ciudad: Mapped[float] = mapped_column(Numeric(10,2))
    solar: Mapped[float] = mapped_column(Numeric(10,2))
    electricidad: Mapped[float] = mapped_column(Numeric(10,2))
    fibra: Mapped[float] = mapped_column(Numeric(10,2))
    puertas_cristal: Mapped[float] = mapped_column(Numeric(10,2))
    piscina: Mapped[float] = mapped_column(Numeric(10,2))
    jardin: Mapped[float] = mapped_column(Numeric(10,2))
    precio_estimado: Mapped[float] = mapped_column(Numeric(10,2))
    precio_venta: Mapped[float] = mapped_column(Numeric(10,2))
```

```
4 model = joblib.load('predictModel/model_houses_trained.pkl')
5
6 def predecir_precio(json_casa):
7
8     casa = convertir_datos(json_casa)
9     x_new = pd.DataFrame([casa])
10    return model.predict(x_new)[0]
11
12 def convertir_datos(json):
13
14    return {'Area': json["area"],
15            'Garage': json["garaje"],
16            'FirePlace': json["chimenea"],
17            'Baths': json["aseos"],
18            'White Marble': json["marmol_blanco"],
19            'Black Marble': json["marmol_negro"],
20            'Indian Marble': json["marmol_indio"],
21            'Floors': json["pisos"],
22            'City': json["ciudad"],
23            'Solar': json["solar"],
24            'Electric': json["electricidad"],
25            'Fiber': json["fibra"],
26            'Glass Doors': json["puertas_cristal"],
27            'Swiming Pool': json["piscina"],
28            'Garden': json["jardin"]}
```

El Routes se encuentran las rutas para acceder al servidor. En estas llamadas también tenemos el código que realiza las operaciones en la base de datos.

```

7  casas = Blueprint('casas', __name__)
8
9  @casas.get("/")
10 def get_casas():
11     select = db.select(Casa)
12     casas = db.session.execute(select).scalars().all()
13     return jsonify(casas)
14
15 @casas.get("/<int:id>")
16 def get_casa(id: int):
17     casa = db.get_or_404(Casa, id)
18     return jsonify(casa)
19
20 @casas.post("/")
21 @validate_json(CasaSchema)
22 def add_casa():
23     json = request.json
24     prediccion = predecir_precio(json)
25     casa = Casa(nombre=json["nombre"],
26                 area=json["area"],
27                 garaje=json["garaje"],
28                 chimenea=json["chimenea"],
29                 aseos=json["aseos"],
30                 marmol_blanco=json["marmol_blanco"],
31                 marmol_negro=json["marmol_negro"],
32                 marmol_indio=json["marmol_indio"],
33                 pisos=json["pisos"],
34                 ciudad=json["ciudad"],
35                 solar=json["solar"],
36                 electricidad=json["electricidad"],
37                 fibra=json["fibra"],
38                 puertas_cristal=json["puertas_cristal"],
39                 piscina=json["piscina"],
40                 jardin=json["jardin"],
41                 precio_estimado=prediccion,
42                 precio_venta=json["precio_venta"])
43     db.session.add(casa)
44     db.session.commit()
45     return jsonify(casa), 201

```

```

47
48 @casas.put("/<int:id>")
49 def update_casa(id: int):
50     casa = db.get_or_404(Casa, id)
51     json = request.json
52     prediccion = predecir_precio(json)
53     casa.nombre = json['nombre']
54     casa.area = json["area"]
55     casa.garaje = json["garaje"]
56     casa.chimenea = json["chimenea"]
57     casa.aseos = json["aseos"]
58     casa.marmol_blanco = json["marmol_blanco"]
59     casa.marmol_negro = json["marmol_negro"]
60     casa.marmol_indio = json["marmol_indio"]
61     casa.pisos = json["pisos"]
62     casa.ciudad = json["ciudad"]
63     casa.solar = json["solar"]
64     casa.electricidad = json["electricidad"]
65     casa.fibra = json["fibra"]
66     casa.puertas_cristal = json["puertas_cristal"]
67     casa.piscina = json["piscina"]
68     casa.jardin = json["jardin"]
69     casa.precio_estimado = prediccion
70     casa.precio_venta = json["precio_venta"]
71     db.session.commit()
72     return jsonify(casa)
73
74
75 @casas.delete("/<int:id>")
76 def delete_casa(id: int):
77     casa = db.get_or_404(Casa, id)
78     db.session.delete(casa)
79     db.session.commit()
80     return "", 204
81

```

El Schema se encarga de permitir solo datos que cumplan con las condiciones indicadas para que no se introduzcan datos erróneos en la base de datos:

```

3  class CasaSchema(Schema):
4      nombre = fields.Str(
5          required=True,
6          error_messages={"required": "El nombre es obligatorio"},
7          validate=validate.Length(min=4, error="El nombre debe tener al menos 4 letras"),)
8
9      area = fields.Number(
10         required=True,
11         error_messages={"required": "El area es obligatoria"},
12         validate=validate.Range(min=0, error="El precio no puede ser negativo"),)
13
14     garaje = fields.Number(
15         required=True,
16         error_messages={"required": "El número de garajes es obligatorio"},
17         validate=validate.Range(min=0, error="El garaje no puede ser negativo"),)
18
19     chimenea = fields.Number(
20         required=True,
21         error_messages={"required": "El número de chimeneas es obligatorio"},
22         validate=validate.Range(min=0, error="La chimenea no puede ser negativo"),)
23
24     aseos = fields.Number(
25         required=True,
26         error_messages={"required": "El número de aseos es obligatorio"},
27         validate=validate.Range(min=0, error="Los aseos no pueden ser negativos"),)
28

```

## 1.4 FrontEnd – Aplicación web

El Front está compuesto de 3 archivos: un css, un js y un html.

### HTML

El HTML está compuesto por una cabecera que permite una nueva consulta, un panel central donde se muestran las casas consultadas y una ventana modal que se superpone a la ventana principal con un formulario para crear y actualizar casas. Además se ha utilizado Bootstrap para el estilo:

Cabecera:

```
18 <header>
19 <nav class="navegacion">
20 <div id="navLogo" class="d-flex ms-5 ps-5">
21 <a id="nom" href="#"><b>HouseParator</b></a>
22 <a id="nom" href="#">
23 
24 </a>
25 </div>
26 <div class="me-5 pe-5">
27 <a id="home">
28 <button id="nueva" type="button" class="btn btn-link" data-bs-toggle="modal" data-bs-target="#staticBackdrop">
29 Nueva Consulta
30 </button>
31 </a>
32 </div>
33 </nav>
34 </header>
```

Panel central que se rellenará automáticamente con javascript:

```
36 <section class="ms-5 me-5 mt-5 ps-4 row" id="panel">
37 </section>
```

Modal:

```
39 <!-- Modal -->
40 <div class="modal fade" id="staticBackdrop" data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1"
41 aria-labelledby="staticBackdropLabel" aria-hidden="true">
42 <div class="modal-dialog modal-lg">
43 <div class="modal-content">
44 <div class="modal-header">
45 <h5 class="modal-title" id="staticBackdropLabel">Rellena la información del inmueble</h5>
46 <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="close"></button>
47 </div>
48 <div class="modal-body m-3">
49 <form class="row needs-validation" id="form">
50 <div class="col-md-5 mt-3">
51 <label for="nombre" class="form-label">Nombre:</label>
52 <input type="text" id="nombre" placeholder="Nombre del inmueble" class="form-control" minlength="2"
53 maxlength="100" required>
54 </div>
55 <div class="col-md-2 mt-3">
56 <label for="area" class="form-label">Área (m²):</label>
57 <input type="number" id="area" placeholder="100 m²" class="form-control" value="50" min="1" max="10000"
58 required>
59 </div>
60 <div class="col-md-2 mt-3">
61 <label for="garaje" class="form-label">Nº Garajes:</label>
62 <input type="number" id="garaje" class="form-control" value="1" min="0" max="10" required>
63 </div>
64 <div class="col-md-3 mt-3">
65 <label for="chimenea" class="form-label">Nº Chimeneas:</label>
66 <input type="number" id="chimenea" class="form-control" value="0" min="0" max="10" required>
67 </div>
68 <div class="col-md-2 mt-3">
69 <label for="aseos" class="form-label">Nº Aseos:</label>
70 <input type="number" id="aseos" class="form-control" value="1" min="0" max="10" required>
71 </div>
72 <div class="col-md-4 mt-3 form-group">
73 <label for="marmol" class="form-label">Tipo de mármol:</label>
74 <select class="form-control mt-2" id="marmol">
75 <option value="0">Mármol blanco</option>
76 <option value="1">Mármol negro</option>
77 <option value="2">Mármol indio</option>
```



## CSS

El CSS junto al Bootstrap da formato a la página:

```
1  nav {
2    display: flex;
3    flex-direction: row;
4    flex-wrap: wrap;
5    justify-content: space-between;
6    background-color: #242424;
7    padding: 20px;
8  }
9
10 #home, #nom, #nueva {
11   color: white;
12   text-decoration: none;
13   margin-right: 15px;
14   size: 100px;
15   font-size: 18px;
16 }
17
18 #nom{
19   color: #e86d82;
20   font-size: 20px;
21 }
22
23 label{
24   white-space: nowrap;
25 }
26
27 img {
28   max-width: 50%;
29   max-height: 50%;
30   margin-left: 25%;
31   margin-top: 3%;
32 }
33
34 #logo{
35   max-width: 100%;
36   max-height: 100%;
37 }
```

## JavaScript

El JS tiene las funciones para acceder al servidor de Python y para mostrar dinámicamente el contenido de las respuestas de la base de datos:

```
function obtenerCasa(id){
  fetch(`${url}${id}`, { method: 'GET' })
    .then(respuesta => respuesta.json())
    .then(datos => mostrarActualizar(datos))
    .catch();
}

function getProductos() {
  fetch(url)
    .then(respuesta => respuesta.json())
    .then(datos => muestraProductos(datos))
    .catch(() => muestraError());
}

function postProducto() {
  casa = obtenerDatos();
  const opciones = {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(casa)
  };
  fetch(url, opciones)
    .then(respuesta => respuesta.json())
    .then(() => cerrarModal())
    .catch(() => muestraError());
}

function putProducto(id) {
  casa = obtenerDatos();
  const opciones = {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(casa)
  };
}
```

```
let color = ''
color = Math.round(casa['precio_estimado']) < Math.round(casa['precio_venta']) ? 'red':'green';
document.getElementById("panel").innerHTML +=

<div class="card m-1 col-4 min" style="width: 20rem;">
  
  <div class="card-body">
    <h5 class="card-title text-center"> ${casa['nombre']} </h5>
    <p class="card-text">
      <ul>
        <li>Área: ${Math.round(casa['area'])}</li>
        <li>Garajes: ${Math.round(casa['garaje'])}</li>
        <li>Chimeneas: ${Math.round(casa['chimenea'])}</li>
        <li>Baños: ${Math.round(casa['aseos'])}</li>
        <li>Ciudad: ${Math.round(casa['ciudad'])}</li>
        <li>Pisos: ${Math.round(casa['pisos'])}</li>
        <li>Tipo Marmol: ${marmol}</li>
      </ul>
    <p style="min-height: 70px;"> ${extras}</p>
    <p>
      Precio promedio: ${casa['precio_estimado']} € <br>
      <b style="color: ${color};"> Precio de venta: ${casa['precio_venta']} € </b>
    </p>
    <p class="d-flex justify-content-center">
      <button class="btn btn-outline-info me-3 ps-4 pe-4"
        onclick="obtenerCasa(${casa.id})"> Editar</button>
      <button class="btn btn-outline-danger ps-4 pe-4"
        onclick="deleteProducto(${casa.id}, '${casa.nombre}')"> Borrar</button>
    </p>
  </div>
</div>
);
```



### Aspecto

El JS tiene las funciones para acceder al servidor de Python y para mostrar dinámicamente el contenido de las respuestas de la base de datos:

Pantalla principal:

HouseParator
Nueva Consulta

**Calle de Los Limones**

- Área: 56
- Garajes: 1
- Chimeneas: 1
- Baños: 2
- ciudad: 2
- Pisos: 1
- Tipo Marmol: Blanco

**Incluye :** - Fibra telefónica - Jardín - Piscina - Puertas de cristal - Placas solares

Precio promedio: 59600.00 €  
**Precio de venta: 80000.00 €**

[Editar](#)
[Borrar](#)

**Calle del Kiwi**

- Área: 50
- Garajes: 1
- Chimeneas: 0
- Baños: 1
- ciudad: 1
- Pisos: 1
- Tipo Marmol: Negro

**Incluye :** - Jardín - Placas solares

Precio promedio: 28750.00 €  
**Precio de venta: 25000.00 €**

[Editar](#)
[Borrar](#)

**Calle del Bambú**

- Área: 56
- Garajes: 1
- Chimeneas: 0
- Baños: 1
- ciudad: 3
- Pisos: 1
- Tipo Marmol: Blanco

**Incluye :** - Red eléctrica - Fibra telefónica - Piscina

Precio promedio: 57650.00 €  
**Precio de venta: 30000.00 €**

[Editar](#)
[Borrar](#)

**Calle del Ficus**

- Área: 50
- Garajes: 1
- Chimeneas: 0
- Baños: 1
- ciudad: 2
- Pisos: 0
- Tipo Marmol: Indio

No incluye extras

Precio promedio: 12000.00 €  
**Precio de venta: 40000.00 €**

[Editar](#)
[Borrar](#)

Modal de Nueva consulta:

HouseParator
Nueva Consulta

**Calle de Los Limones**

- Área: 56
- Garajes: 1
- Chimeneas: 1
- Baños: 2
- ciudad: 2
- Pisos: 1
- Tipo Marmol: Blanco

**Incluye :** - Fibra telefónica - Jardín - Piscina - Puertas de cristal - Placas solares

Precio promedio: 59600.00 €  
**Precio de venta: 80000.00 €**

**Calle del Ficus**

- Área: 50
- Garajes: 1
- Chimeneas: 0
- Baños: 1
- ciudad: 2
- Pisos: 0
- Tipo Marmol: Indio

No incluye extras

Precio promedio: 12000.00 €  
**Precio de venta: 40000.00 €**

Rellena la información del inmueble


Nombre: Calle del Cerezo
Área (m²): 50
Nº Garajes: 1
Nº Chimeneas: 2
Nº Aseos: 1
Tipo de mármol: Mármol blanco
Ciudad: Ciudad 1
Pisos: 1
☐ Placas solares
☒ Red eléctrica
☒ Fibra telefónica
☐ Puertas de cristal
☐ Piscina
☒ Jardín
Precio de venta: 50000

Cerrar
Consultar

### Modal de Actualización de datos:

Precio de venta: 80000.00 €

Editar Borrar



**Calle del cerezo**

- Área: 50
- Garajes: 1
- Chimeneas: 1
- Baños: 1
- ciudad: 1
- Pisos: 1
- Tipo Marmol: Blanco

Incluye : - Red eléctrica - Jardín

Precio promedio: 39500.00 €  
**Precio de venta: 50000.00 €**

Editar Borrar

**Rellena la información del inmueble**

Nombre:  Área (m²):  N° Garajes:  N° Chimeneas:

N° Aseos:  Tipo de mármol:  Ciudad:  Pisos:

☐ Placas solares ☒ Red eléctrica ☐ Fibra telefónica ☐ Puertas de cristal

☐ Piscina ☒ Jardín

Precio de venta:

Cerrar Actualizar

### Alerta de confirmación de borrado:

Precio promedio: 39500.00 €  
**Precio de venta: 80000.00 €**

Editar Borrar

Precio promedio: 25000.00 €  
**Precio de venta: 25000.00 €**


Editar Borrar

Precio promedio: 30000.00 €  
**Precio de venta: 30000.00 €**

Editar Borrar

Precio promedio: 40000.00 €  
**Precio de venta: 40000.00 €**

Editar Borrar




**Calle del cerezo**

- Área: 50
- Garajes: 1
- Chimeneas: 1
- Baños: 1
- ciudad: 1
- Pisos: 1
- Tipo Marmol: Blanco

Incluye : - Red eléctrica - Jardín

Precio promedio: 39500.00 €  
**Precio de venta: 50000.00 €**

Editar Borrar



**¿Estás seguro de borrar la casa 'Calle del cerezo'?**

Esta acción no se puede deshacer

Borrar Cancelar