

VARIABLES CRÍTICAS PARA PRONOSTICAR EL VALOR DE INMUEBLES EN CDMX

Mariana Manzano Rico

A01735770

Minería de Datos

Introducción

Determinar el precio de una casa puede convertirse en una tarea compleja, ya que implica tener un conocimiento profundo de los valores de mercado, los costos asociados a la propiedad y considerar diversos factores como el tamaño de la casa, la ubicación, las amenidades disponibles, entre otros. A pesar de la complejidad, esta tarea es esencial, ya que establecer un precio demasiado alto puede alejar a posibles compradores, prolongando innecesariamente la permanencia de la propiedad en el mercado. Por otro lado, fijar un precio adecuado es crucial para atraer compradores potenciales y asegurar una venta exitosa (BBVA, s.f.).

En este reporte, se emplean VARIABLES CRÍTICAS PARA PRONOSTICAR EL VALOR DE INMUEBLES EN CDMX. Se utilizan diversas técnicas estadísticas, como regresión lineal, árboles de decisión y redes neuronales, para determinar los precios por metro cuadrado de diversas propiedades en la Ciudad de México. Estas herramientas analíticas permiten una evaluación más precisa y fundamentada, proporcionando a Erich Zann y Asociados una ventaja estratégica en el competitivo mercado inmobiliario de la CDMX.

```
In [ ]: #Librerías
import statsmodels.api as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import cross_val_score
```

Para ello, vamos a conocer la base de datos con la que entrenaremos los modelos, que pueden ver en la siguiente imagen.

```
In [ ]: #Carga de archivos
inmu = pd.read_csv("Clusters.csv")
inmu.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 657 entries, 0 to 656
Data columns (total 27 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   Unnamed: 0        657 non-null    int64  
 1   Alcaldia          657 non-null    object  
 2   Colonia           657 non-null    object  
 3   X1                657 non-null    float64 
 4   X2                657 non-null    float64 
 5   X3                657 non-null    float64 
 6   X4                657 non-null    float64 
 7   X5                657 non-null    float64 
 8   X6                657 non-null    float64 
 9   X7                657 non-null    float64 
 10  X8                657 non-null    float64 
 11  X9                657 non-null    float64 
 12  X10               657 non-null    float64 
 13  Cocina_equip      657 non-null    int64  
 14  Gimnasio          657 non-null    int64  
 15  Amueblado         657 non-null    int64  
 16  Alberca            657 non-null    int64  
 17  Terraza            657 non-null    int64  
 18  Elevador           657 non-null    int64  
 19  m2_construido     657 non-null    float64 
 20  Baños              657 non-null    float64 
 21  Recamaras          657 non-null    int64  
 22  Lugares_estac     657 non-null    int64  
 23  Precio_m2          657 non-null    float64 
 24  Cluster Labels    657 non-null    int64  
 25  Conglomerados      657 non-null    object  
 26  NivelSocioEconomico 657 non-null    object  
dtypes: float64(13), int64(10), object(4)
memory usage: 138.7+ KB
```

Esta cuenta con las siguientes variables:

1. **Alcaldía:** Indica la demarcación administrativa dentro de la Ciudad de México donde se encuentra ubicado el inmueble.
2. **Colonia:** Hace referencia al nombre de la zona o vecindario específico donde se localiza la propiedad.

3. **Cocina Equipada:** Indica si la vivienda cuenta con una cocina equipada con electrodomésticos y utensilios básicos.
4. **Gimnasio:** Refiere a la presencia o ausencia de instalaciones deportivas o gimnasio en el inmueble.
5. **Amueblado:** Indica si la propiedad se ofrece en venta con muebles incluidos.
6. **Alberca:** Se refiere a la presencia o ausencia de una piscina en el lugar.
7. **Terraza:** Indica si la propiedad cuenta con un espacio al aire libre, como una terraza o balcón.
8. **Elevador:** Señala si el edificio o complejo inmobiliario dispone de ascensores para acceder a los diferentes niveles.
9. **m2 Construido:** Representa los metros cuadrados de construcción total de la propiedad.
10. **Baños:** Número de baños presentes en la vivienda.
11. **Recámaras:** Indica la cantidad de habitaciones destinadas para dormir en la propiedad.
12. **Lugares de Estacionamiento:** Número de espacios designados para estacionar vehículos.
13. **Precio por m2:** Costo por metro cuadrado de la propiedad, un indicador clave para la valoración inmobiliaria y nuestra variable objetivo en los modelos.
14. **Cluster Labels:** Etiquetas asignadas a los conglomerados obtenidos mediante técnicas de análisis de conglomerados.
15. **Conglomerados:** Agrupaciones resultantes del análisis de conglomerados que clasifican las propiedades según características similares.
16. **Nivel Socioeconómico:** Clasificación del nivel socioeconómico de la zona donde se encuentra la propiedad, derivada del análisis de diversas variables socioeconómicas.
17. **X1:** Poblacion de 15 años o mas analfabeta
18. **X2:** Poblacion de 6 a 14 años que no asiste a la escuela
19. **X3:** Poblacion de 15 a 24 años que no asiste a la escuela
20. **X4:** Población de 15 años o más con educación básica incompleta
21. **X5:** Población sin derechohabiencia a servicios de salud
22. **X6:** Viviendas con hacinamiento

23. **X7:** Viviendas que no disponen de agua entubada de la red pública

24. **X8:** Viviendas que no disponen de energía eléctrica

25. **X9:** Viviendas que no disponen de celular

26. **X10:** Viviendas que no disponen de computadora o laptop o tablet

Dicho esto, procedemos con los modelos matemáticos para la predicción de nuevos inmuebles:

Generación de Modelos

Regresión Lineal Multiple

La regresión lineal múltiple es una técnica estadística que extiende el concepto de regresión lineal simple al considerar múltiples variables independientes para predecir una variable dependiente. En nuestro análisis, aplicaremos la regresión lineal múltiple como una herramienta poderosa para modelar y predecir el precio por metro cuadrado de las propiedades inmobiliarias en la Ciudad de México. Al incorporar diversas características, como tamaño de construcción, número de habitaciones, servicios y ubicación, este enfoque nos permitirá capturar de manera más precisa la complejidad de los factores que influyen en la valoración de los inmuebles.

Modelo 1 (Todas las variables)

Nuestro primer modelo incluye todas las variables para determinar qué tan efectivo es el modelo y las variables que deberían ser eliminadas para obtener mejores resultados.

```
In [ ]: model = LinearRegression()
type(model)

x = inmu[['X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10", "Cocina_equi
y = inmu["Precio_m2"]

model.fit(X = x, y = y)
model.__dict__
#Coeficiente de determinación
determinacion = model.score(x, y)
correlacion = np.sqrt(determinacion)
print("Determinacion:", determinacion)
print("Correlación: ", correlacion)

# Agrega una constante al conjunto de datos (intercepto)
x_with_intercept = sm.add_constant(x)

# Ajusta el modelo
model = sm.OLS(y, x_with_intercept).fit()
```

```
# Imprime un resumen del modelo que incluye valores p
print(model.summary())
```

Determinacion: 0.7480588775333463

Correlación: 0.8649039701223173

OLS Regression Results

Dep. Variable:	Precio_m2	R-squared:	0.748
Model:	OLS	Adj. R-squared:	0.741
Method:	Least Squares	F-statistic:	99.55
Date:	Sat, 02 Dec 2023	Prob (F-statistic):	3.17e-176
Time:	17:49:13	Log-Likelihood:	-6075.7
No. Observations:	657	AIC:	1.219e+04
Df Residuals:	637	BIC:	1.228e+04
Df Model:	19		
Covariance Type:	nonrobust		
const	-1437.4161	2183.934	-0.658
X1	2867.3888	2593.414	1.106
X2	-2781.2602	525.398	-5.294
X3	1093.0076	139.017	7.862
X4	52.3560	234.869	0.223
X5	-477.5905	70.134	-6.810
X6	-1534.3042	832.801	-1.842
X7	-24.1865	101.479	-0.238
X8	2.27e+04	5124.160	4.430
X9	151.5963	459.414	0.330
X10	-549.5713	145.992	-3.764
Cocina_equip	4.0252	468.688	0.009
Gimnasio	-387.9065	355.493	-1.091
Amueblado	935.2883	704.469	1.328
Alberca	1371.2545	384.326	3.568
Terraza	-30.3337	338.209	-0.090
Elevador	233.7522	297.517	0.786
Baños	1729.5269	210.258	8.226
Recamaras	573.7772	227.433	2.523
Lugares_estac	1297.8213	200.525	6.472
Omnibus:	514.300	Durbin-Watson:	1.987
Prob(Omnibus):	0.000	Jarque-Bera (JB):	19519.111
Skew:	3.102	Prob(JB):	0.00
Kurtosis:	28.972	Cond. No.	3.63e+03

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 3.63e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Observamos que el 74% de nuestra variable objetivo es explicada por el modelo, que tienen un AIC y BIC bajos (lo cual es bueno pero debemos comparar con modelos posteriores) y que, de acuerdo al *valor p*, (*Prob F-statistics*) es significativo debido a que es menor a 0.05.

Sin embargo, observamos que algunas variables como X2, X4, X6 o X7 no son tan significativas (no impactan tanto en el modelo) por su *valor p* mayor a 0.05, por lo que serán eliminadas para el siguiente modelo.

Modelo 2 (Con significantes)

Para este segundo modelo, solo consideraremos las variables de X2, X3, X5, X8, X10, Alberca, Baños, Recamaras y Lugares de estacionamiento debido a que estas fueron significativas en el modelo anterior.

```
In [ ]: #Modelo 2 (Con significantes)
model = LinearRegression()
type(model)

x = inmu[['X2', 'X3', 'X5','X8', 'X10', 'Alberca', 'Baños', 'Recamaras', 'Lugares_e
y = inmu['Precio_m2']

model.fit(X = x, y = y)
model.__dict__
#Coeficiente de determinación
determinacion = model.score(x, y)
correlacion = np.sqrt(determinacion)
print("Determinacion:", determinacion)
print("Correlación: ", correlacion)

# Agrega una constante al conjunto de datos (intercepto)
x_with_intercept = sm.add_constant(x)

# Ajusta el modelo
model = sm.OLS(y, x_with_intercept).fit()

# Imprime un resumen del modelo que incluye valores p
print(model.summary())
```

Determinacion: 0.7367105913396992

Correlación: 0.858318467318337

OLS Regression Results

Dep. Variable:	Precio_m2	R-squared:	0.737
Model:	OLS	Adj. R-squared:	0.733
Method:	Least Squares	F-statistic:	201.2
Date:	Sat, 02 Dec 2023	Prob (F-statistic):	5.98e-181
Time:	17:49:13	Log-Likelihood:	-6090.2
No. Observations:	657	AIC:	1.220e+04
Df Residuals:	647	BIC:	1.225e+04
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
const	-1522.5496	1732.780	-0.879	0.380	-4925.101	1880.002
X2	-1034.2976	268.884	-3.847	0.000	-1562.288	-506.307
X3	729.6193	79.917	9.130	0.000	572.692	886.547
X5	-465.8834	56.872	-8.192	0.000	-577.560	-354.207
X8	1.941e+04	3930.629	4.939	0.000	1.17e+04	2.71e+04
X10	-390.5308	40.148	-9.727	0.000	-469.368	-311.694
Alberca	1251.9295	321.454	3.895	0.000	620.711	1883.148
Baños	1662.8778	204.103	8.147	0.000	1262.093	2063.662
Recamaras	781.3415	224.378	3.482	0.001	340.745	1221.938
Lugares_estac	1346.4962	195.842	6.875	0.000	961.934	1731.058

Omnibus:	516.252	Durbin-Watson:	2.028
Prob(Omnibus):	0.000	Jarque-Bera (JB):	21356.515
Skew:	3.086	Prob(JB):	0.00
Kurtosis:	30.240	Cond. No.	2.63e+03

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 2.63e+03. This might indicate that there are strong multicollinearity or other numerical problems.

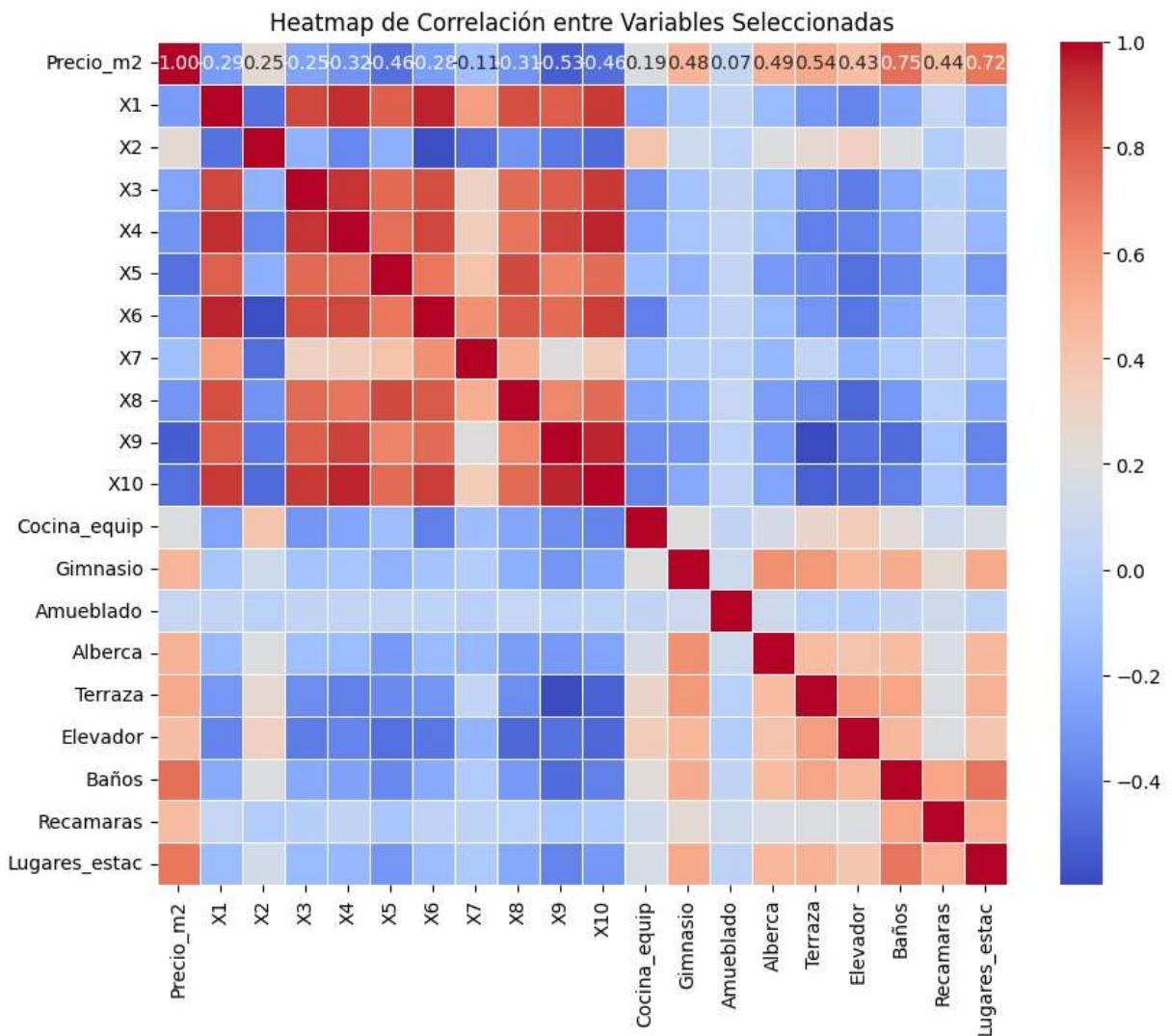
En este modelo observamos un *valor p* significativo, por lo que el modelo es funcional.

Además, el modelo explica el 73% del precio por metros cuadrados. En cuanto al BIC y AIC, vemos valores muy similares que en el modelo anterior. Finalmente, encontramos algunas variables otra vez no significativas. Sin embargo, debido a que el modelo bajó su efectividad en vez de aumentarla, procederemos a utilizar otras estrategias para definir el mejor modelo.

Modelo 3 (Con correlaciones más altas)

Este nuevo modelo considerará aquellas variables mejor correlacionadas con la variable dependiente, por lo que primero generaremos un mapa de calor para encontrar aquellas mejor relacionadas, como puede observarse en el siguiente gráfico.

```
In [ ]: #Modelo 3 (Con correlaciones más altas)
data_selected = inmu[["Precio_m2", "X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8",
#
# Calcula la matriz de correlación
correlation_matrix = data_selected.corr()
#
# Crea el heatmap con seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=
plt.title("Heatmap de Correlación entre Variables Seleccionadas")
plt.show()
```



Vemos que las variables Gimnasio, Alberca, Terraza, Elevador, Baños, Recámaras y Lugares de Estacionamiento tienen las relaciones más fuertes, por lo que serán usadas en el siguiente modelo:

```
In [ ]: model = LinearRegression()
type(model)

x = inmu[["Gimnasio", "Alberca", "Terraza", "Elevador", "Baños", "Recamaras", "Luga
y = inmu["Precio_m2"]
```

```

model.fit(X = x, y = y)
model.__dict__
#Coeficiente de determinación
determinacion = model.score(x, y)
correlacion = np.sqrt(determinacion)
print("Determinacion:", determinacion)
print("Correlación: ", correlacion)

# Agrega una constante al conjunto de datos (intercepto)
x_with_intercept = sm.add_constant(x)

# Ajusta el modelo
model = sm.OLS(y, x_with_intercept).fit()

# Imprime un resumen del modelo que incluye valores p
print(model.summary())

```

Determinacion: 0.6483537987961022

Correlación: 0.8052041969563387

OLS Regression Results

Dep. Variable:	Precio_m2	R-squared:	0.648			
Model:	OLS	Adj. R-squared:	0.645			
Method:	Least Squares	F-statistic:	170.9			
Date:	Sat, 02 Dec 2023	Prob (F-statistic):	1.01e-142			
Time:	17:49:15	Log-Likelihood:	-6185.2			
No. Observations:	657	AIC:	1.239e+04			
Df Residuals:	649	BIC:	1.242e+04			
Df Model:	7					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	-3983.4077	493.184	-8.077	0.000	-4951.836	-3014.979
Gimnasio	-788.7465	384.514	-2.051	0.041	-1543.788	-33.705
Alberca	1938.8306	412.111	4.705	0.000	1129.599	2748.062
Terraza	1226.8637	336.896	3.642	0.000	565.326	1888.402
Elevador	147.8290	298.111	0.496	0.620	-437.550	733.208
Baños	2375.8266	234.258	10.142	0.000	1915.831	2835.822
Recamaras	246.4787	252.463	0.976	0.329	-249.264	742.222
Lugares_estac	1935.9446	222.752	8.691	0.000	1498.543	2373.346
Omnibus:	454.628	Durbin-Watson:	2.062			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	13589.184			
Skew:	2.642	Prob(JB):	0.00			
Kurtosis:	24.645	Cond. No.	16.9			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

En este observamos una r^2 (eficacia del modelo), del 64%, es servible el modelo e igualmente un AIC y BIC similar a los anteriores. Nuevamente vemos algunas variables no

significativas como Elevador y Recámaras, las cuales serán eliminadas para generar un nuevo modelo.

Modelo 4 (Con significantes de la anterior)

Este último modelo considera las variables significantes del modelo anterior, con el cual obtenemos los siguientes resultados.

```
In [ ]: #Modelo 4 (Con significantes de la anterior)
model = LinearRegression()
type(model)

x = inmu[["Gimnasio", "Alberca", "Terraza", "Baños", "Lugares_estac"]]
y = inmu["Precio_m2"]

model.fit(X = x, y = y)
model.__dict__
#Coeficiente de determinación
determinacion = model.score(x, y)
correlacion = np.sqrt(determinacion)
print("Determinacion:", determinacion)
print("Correlación: ", correlacion)

# Agrega una constante al conjunto de datos (intercepto)
x_with_intercept = sm.add_constant(x)

# Ajusta el modelo
model = sm.OLS(y, x_with_intercept).fit()

# Imprime un resumen del modelo que incluye valores p
print(model.summary())
```

Determinacion: 0.6477140502689188

Correlación: 0.8048068403467498

OLS Regression Results

Dep. Variable:	Precio_m2	R-squared:	0.648
Model:	OLS	Adj. R-squared:	0.645
Method:	Least Squares	F-statistic:	239.4
Date:	Sat, 02 Dec 2023	Prob (F-statistic):	7.59e-145
Time:	17:49:15	Log-Likelihood:	-6185.8
No. Observations:	657	AIC:	1.238e+04
Df Residuals:	651	BIC:	1.241e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
const	-3565.6455	276.410	-12.900	0.000	-4108.408	-3022.883
Gimnasio	-773.0224	383.135	-2.018	0.044	-1525.352	-20.693
Alberca	1930.1871	408.522	4.725	0.000	1128.007	2732.367
Terraza	1233.2721	311.408	3.960	0.000	621.786	1844.758
Baños	2474.0428	215.768	11.466	0.000	2050.357	2897.729
Lugares_estac	1980.0743	217.340	9.111	0.000	1553.303	2406.845

Omnibus:	454.458	Durbin-Watson:	2.065
Prob(Omnibus):	0.000	Jarque-Bera (JB):	13630.454
Skew:	2.639	Prob(JB):	0.00
Kurtosis:	24.681	Cond. No.	11.1

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

El modelo es servible, explica nuevamente el 64% del precio por metro cuadrado de las casas y el BIC y AIC es parecido a los demás.

En conclusión, se selecciona el primer modelo de regresión lineal múltiple como punto de referencia por su eficacia, evidenciada por un coeficiente de determinación (*R-squared*) del 74%. Este valor proporciona una medida sólida de la capacidad del modelo para explicar la variabilidad en los datos. Dado que el *R-squared* es el más alto entre los modelos considerados, hemos decidido seleccionarlo como el punto de partida para la comparación con otros enfoques, como los modelos de árboles de regresión y redes neuronales. La robustez y la interpretabilidad inherentes a la regresión lineal múltiple respaldan su elección como la primera opción para entender y predecir los precios por metro cuadrado de propiedades en la Ciudad de México.

Para ver una gráfica de cómo se comportan los datos, podemos ver la siguiente imagen:

```
In [ ]: model = LinearRegression()
type(model)

x = inmu[["X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10", "Cocina_equi
y = inmu["Precio_m2"]
```

```
model.fit(X = x, y = y)
model.__dict__

# Agrega una constante al conjunto de datos (intercepto)
x_with_intercept = sm.add_constant(x)

# Ajusta el modelo
model = sm.OLS(y, x_with_intercept).fit()

# Predicciones
predictions = model.predict(x_with_intercept)

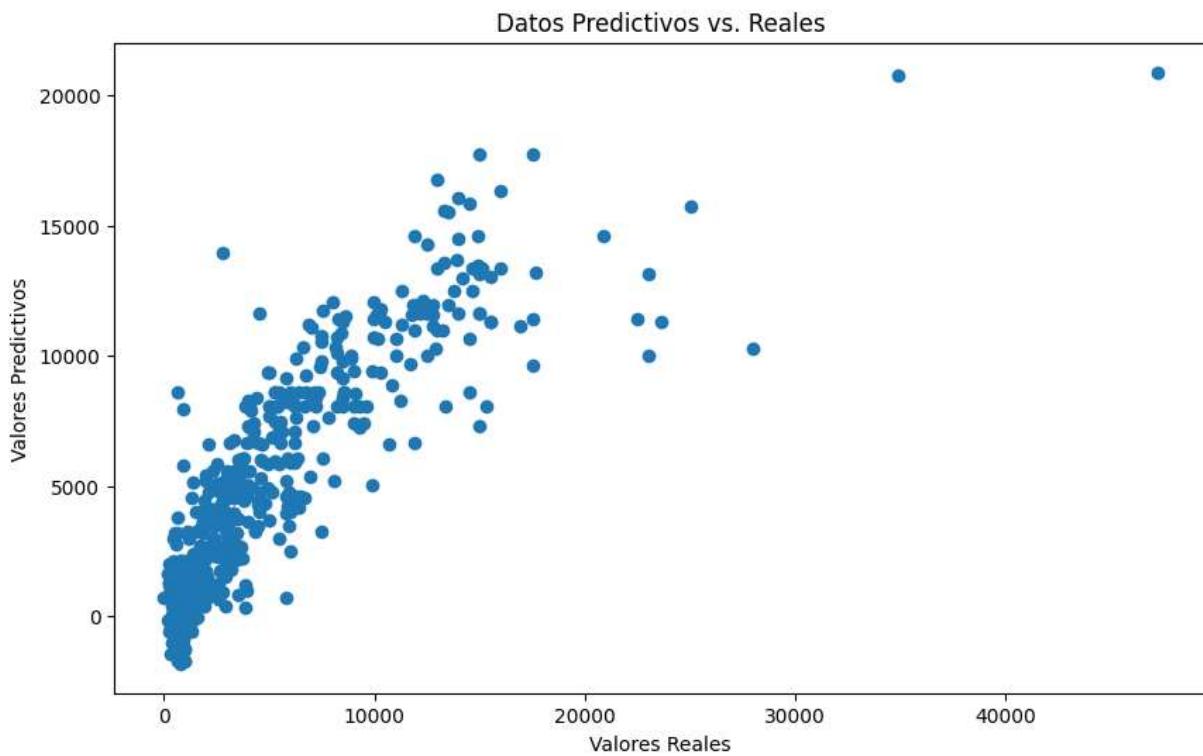
# Gráfica de datos predictivos vs. reales
plt.figure(figsize=(10, 6))
plt.scatter(y, predictions)
plt.title('Datos Predictivos vs. Reales')
plt.xlabel('Valores Reales')
plt.ylabel('Valores Predictivos')
plt.show()

# Errores MAE y MSE
mae = mean_absolute_error(y, predictions)
mse = mean_squared_error(y, predictions)

print(f'Error Absoluto Medio (MAE): {mae:.2f}')
print(f'Error Cuadrático Medio (MSE): {mse:.2f}')

# Estadísticas descriptivas de las predicciones
stats = model.predict(x_with_intercept).describe()
print(stats)

# Estadísticas descriptivas de los valores reales
stats_reales = y.describe()
print("\nEstadísticas descriptivas de los valores reales:")
print(stats_reales)
```



Error Absoluto Medio (MAE): 1502.21

Error Cuadrático Medio (MSE): 6308371.07

```
count      657.000000
mean      4094.677207
std       4331.195045
min      -1801.755824
25%       702.043274
50%      2343.845438
75%      7073.761952
max      20888.522690
dtype: float64
```

Estadísticas descriptivas de los valores reales:

```
count      657.000000
mean      4094.677207
std       5007.717845
min       1.250000
25%      860.657000
50%      1918.080000
75%      5524.000000
max      47200.000000
Name: Precio_m2, dtype: float64
```

También podemos ver los estadísticos del modelo así como sus errores, los cuales serán comparados posteriormente con otras técnicas para seleccionar el mejor modelo.

Arboles de Decisión

En conclusión, se selecciona el primer modelo de regresión lineal múltiple como punto de referencia por su eficacia, evidenciada por un coeficiente de determinación (*R-squared*) del 74%. Este valor proporciona una medida sólida de la capacidad del modelo para explicar la

variabilidad en los datos. Dado que el R-squared es el más alto entre los modelos considerados, hemos decidido seleccionarlo como el punto de partida para la comparación con otros enfoques, como los modelos de árboles de regresión y redes neuronales. La robustez y la interpretabilidad inherentes a la regresión lineal múltiple respaldan su elección como la primera opción para entender y predecir los precios por metro cuadrado de propiedades en la Ciudad de México.

Modelo 1

Este primer modelo contiene todas las variables y será calculado para ver su comportamiento.

```
In [ ]: inmu.dropna(inplace=True)

X = inmu.select_dtypes(include=['float64', 'int64']).drop(columns=['Precio_m2', 'Clu
#x = inmu.iloc[:,1:] # todas las columnas posteriores a la primera
y = inmu.iloc[:,23] # la primera columna

#Entrenamos nuestro modelo con las variables seleccionadas
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=.3)

dtree = DecisionTreeRegressor(random_state=44)
dtree.fit(x_train, y_train) #entrenamiento: covars y respuesta
pred = dtree.predict(x_test) #predicción con covars no entrenadas.
print(pred)

print("Mean Squared Error:", mean_squared_error(y_test, pred))
print("R-squared:", r2_score(y_test, pred))
feature_importance = dtree.feature_importances_
feature_names = X.columns

# Para una mejor visualización, crearemos un data frame nuevo con los nombres de la
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': featu
# Lo ordenamos en orden descendiente
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Mostramos los resultados
print(feature_importance_df)
```

[8.12000000e+02 1.12500000e+03 3.90024900e+03 5.15140000e+03
 3.60000000e+02 1.10000000e+03 4.77174000e+02 4.95000000e+03
 3.98239333e+02 1.10000000e+04 3.78680000e+03 1.15700000e+03
 4.50000000e+03 6.35760000e+02 1.38000000e+03 5.69500000e+02
 8.36000000e+02 4.10000000e+03 1.84000000e+03 2.00100000e+03
 9.57050000e+02 1.35000000e+04 1.29000000e+04 1.17000000e+03
 4.51777000e+02 3.62000000e+03 7.00000000e+02 3.50994000e+02
 3.50994000e+02 1.10000000e+04 1.52000000e+03 9.68000000e+02
 1.40000000e+03 3.62000000e+03 5.54780000e+03 5.43428000e+03
 1.19000000e+04 1.58000000e+03 7.99130000e+02 6.50000000e+02
 2.25000000e+04 2.35999900e+03 2.50000000e+04 1.49000000e+03
 1.58000000e+03 1.10000000e+03 1.72000000e+03 5.50777000e+02
 2.01200000e+03 2.35100000e+03 1.74750300e+03 7.80000000e+02
 8.38953333e+02 7.63560000e+02 1.01800000e+03 6.50000000e+02
 1.00750000e+03 1.16000000e+03 1.06757000e+03 4.51777000e+02
 1.40000000e+03 6.68265000e+02 1.35000000e+04 6.60000000e+02
 1.69500000e+03 5.61400000e+02 9.69000500e+02 8.50000000e+02
 4.44900000e+03 4.51777000e+02 1.50000000e+04 2.95000000e+03
 1.82292500e+03 1.69500000e+03 3.29500000e+03 1.13129780e+04
 1.46720000e+04 1.38000000e+03 1.23000000e+04 8.90000000e+03
 8.50000000e+03 1.07500000e+03 9.99000000e+03 4.15440150e+03
 1.09545000e+03 1.28000000e+04 3.29500000e+03 4.16000000e+03
 1.18500000e+04 3.30000000e+02 9.68000000e+02 9.68000000e+02
 1.32000000e+03 3.32300000e+03 5.20000000e+03 4.94500000e+03
 9.93600000e+03 7.20000000e+02 1.46662930e+04 1.25000000e+04
 1.69500000e+03 9.99000000e+03 1.21666667e+03 1.75000000e+04
 7.85000000e+02 2.70000000e+03 1.25000000e+04 1.29000000e+04
 1.17500000e+03 6.99300000e+02 1.38000000e+03 3.84000000e+03
 3.85000000e+03 8.91000000e+02 7.20000000e+02 2.95100000e+03
 5.30000000e+02 2.70000000e+03 1.05947000e+03 4.77174000e+02
 5.49500000e+03 7.91528000e+02 1.90000000e+03 5.20000000e+03
 6.00000000e+03 5.30000000e+02 1.17000000e+03 1.13129780e+04
 1.12500000e+03 1.80783800e+03 6.13060000e+02 1.69500000e+03
 9.50000000e+02 8.08704500e+03 3.50994000e+02 6.95000000e+02
 1.16000000e+03 1.23000000e+04 1.69500000e+03 1.60000000e+04
 6.11250000e+02 1.40000000e+03 1.46720000e+04 3.30000000e+02
 1.91808000e+03 7.80000000e+02 1.09444000e+03 7.99130000e+02
 3.50994000e+02 5.11500000e+03 4.51777000e+02 1.50000000e+04
 4.25000000e+03 1.90000000e+03 1.38000000e+03 7.63560000e+02
 2.74500000e+03 2.50000000e+04 7.85000000e+02 1.35000000e+04
 4.52219100e+03 1.07500000e+03 5.79000000e+03 1.55000000e+04
 1.35000000e+04 1.89000000e+03 6.99300000e+02 4.10000000e+03
 1.19500000e+03 1.84000000e+03 1.27680000e+04 1.74750300e+03
 1.60000000e+04 7.90000000e+02 9.99000000e+03 8.90000000e+02
 5.61400000e+02 1.25000000e+00 8.50000000e+02 1.19500000e+03
 4.52219100e+03 9.69000500e+02 4.04348000e+02 4.16000000e+03
 3.95000000e+02 1.84000000e+03 1.91808000e+03 9.57050000e+02
 1.29000000e+04 3.90000000e+03 8.86680000e+02 1.38000000e+03
 5.72600000e+02 9.70428750e+02 7.00000000e+02 3.28695200e+03
 7.99000000e+03 4.51777000e+02]

Mean Squared Error: 4981756.32544859

R-squared: 0.7823230393557081

	Feature	Importance
16	m2_construido	0.719555
8	X9	0.090160
12	Amueblado	0.064311

```

4          X5  0.034051
17         Baños  0.024874
18     Recamaras  0.011800
2          X3  0.010963
15     Elevador  0.006677
19  Lugares_estac  0.005558
1          X2  0.005511
11    Gimnasio  0.005256
7          X8  0.005255
0          X1  0.004357
3          X4  0.003515
6          X7  0.002279
14    Terraza  0.002148
13    Alberca  0.001547
10  Cocina_equip  0.001511
5          X6  0.000493
9          X10  0.000180

```

Observamos que tiene muchas variables que no son tan importantes (Lo podemos ver en la tabla de importancia), por lo que decidimos tomar sólo las primeras tres más importantes.

Modelo 2

Esta solamente contiene las de metro cuadrado construido y dos que describen la zona donde se encuentra la casa, obtenemos el siguiente resultado.

```

In [ ]: variables_seleccionadas = ['m2_construido', 'X9', 'X5']
X2 = inmu[variables_seleccionadas]
y = inmu.iloc[:,23]      # nuestra variable dependiente

#Entrenamos nuestro modelo con las variables seleccionadas
x2_train, x2_test, y2_train, y2_test = train_test_split(X2, y, test_size=.3)

#Corremos nuestro modelo:
dtree2 = DecisionTreeRegressor(random_state=44)
dtree2.fit(x2_train, y2_train) #entrenamiento: covars y respuesta
pred2 = dtree2.predict(x2_test) #predicción con covars no entrenadas.
print(pred2)

#Evaluamos nuestro modelo
print("Mean Squared Error:", mean_squared_error(y2_test, pred2))
print("R-squared:", r2_score(y2_test, pred2))

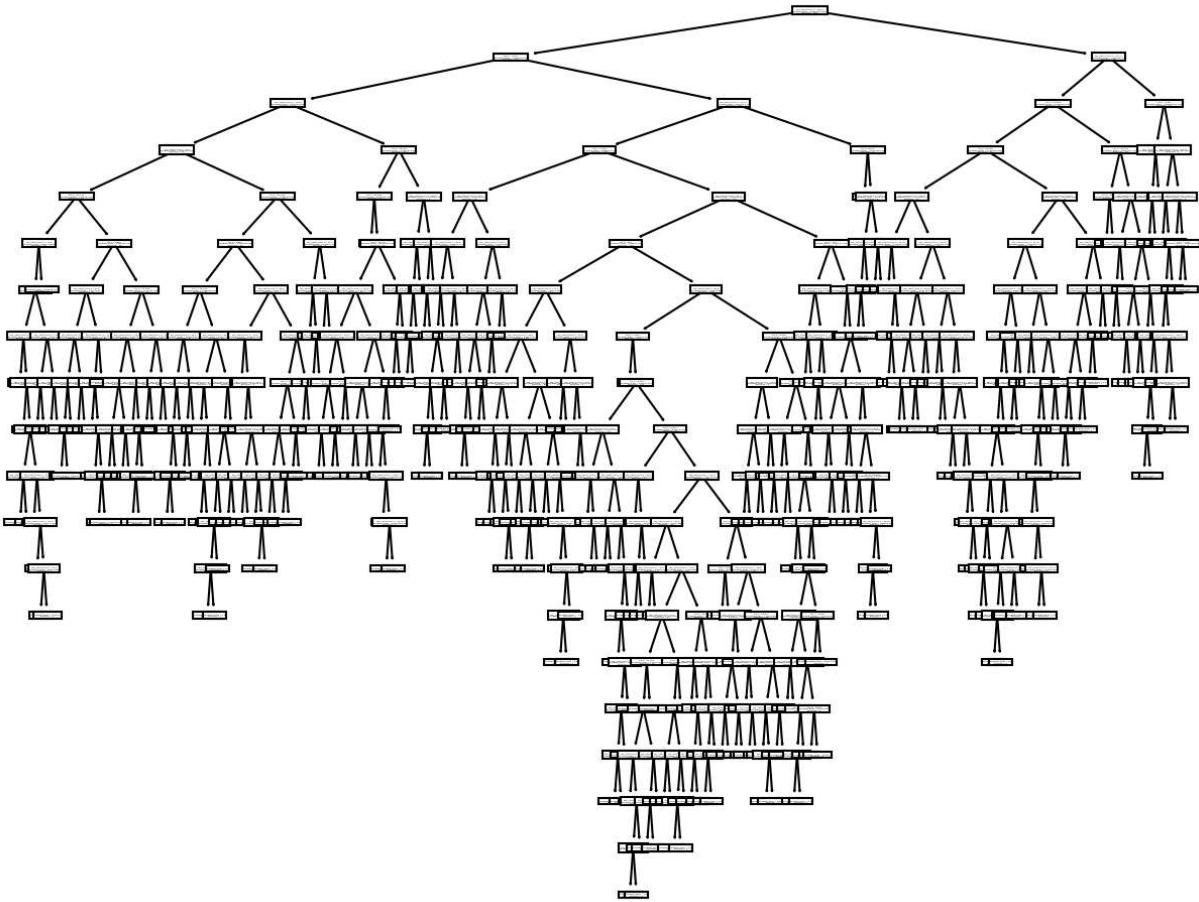
plt.figure(figsize=(10,8), dpi=150)
plot_tree(dtree2, feature_names=X2.columns);

```

[285.192	1600.	7517.	1380.
1175.	284.426	11000.	1156.	
3300.	570.	1007.5	705.06	
1046.4	3500.	3300.	3800.	
605.46245455	4449.	978.97857143	525.5	
695.	11850.	1180.	13500.	
978.97857143	15263.	783.43	749.40266667	
978.97857143	749.40266667	1145.	428.	
540.	705.06	6500.	1046.4	
47200.	2951.	783.43	10100.	
836.	490.	6200.	3465.	
812.	2093.	7517.	1242.5	
2817.	4250.	13000.	9100.	
1595.	780.	1247.624875	995.43333333	
2817.	5434.28	7790.	978.97857143	
4258.60833333	5791.8	5254.799	12500.	
745.886	650.	7790.	1084.6	
4578.22	750.	605.46245455	5791.8	
1242.5	6400.	1242.5	3200.	
11312.978	6000.	440.123	593.6	
11950.	3500.	605.46245455	749.40266667	
2730.	650.	870.	5750.	
1247.624875	2150.	13000.	6200.	
813.467	722.385	1900.	3500.	
746.419	6950.	12200.	3450.	
5411.0955	1900.	995.43333333	1900.	
15500.	3350.	525.5	650.	
3539.	1130.	11850.	8900.	
990.	7150.	7150.	1077.5	
990.	580.	836.	3627.173	
7500.	905.	1242.5	614.076	
5254.799	1007.5	8500.	8500.	
2150.	3800.	676.52066667	11950.	
3653.205	580.	890.	3850.	
750.	749.40266667	2783.354	12500.	
4900.	5550.	4950.	7790.	
12500.	5800.	1845.	990.	
954.	1736.	3653.205	2950.	
750.	695.	2800.	2349.569	
680.7	1247.624875	1500.	1175.	
3295.	12500.	749.40266667	899.	
1915.	5434.28	976.36	540.	
750.	553.045	2001.	990.	
1450.	990.	3500.	3300.	
899.	9100.	635.76	12500.	
28000.	1900.	1600.	1007.5	
890.	668.265	13000.	4990.	
2874.	976.36	490.	705.06	
695.	15500.	580.	749.40266667	
523.	13295.263]		

Mean Squared Error: 8552042.40845578

R-squared: 0.5972527657872242



Sin embargo observamos un Overfitting en nuestro arbol, por lo que será necesario llevar a cabo técnicas de poda. Por otro lado, observamos una R^2 de 81%, lo cual es una muy buena cifra para nuestro modelo.

```
In [ ]: dtree_poda = DecisionTreeRegressor(max_depth=6, min_samples_split=9, min_samples_leaf=1)
dtree_poda.fit(x2_train, y2_train)

y_pred_poda = dtree_poda.predict(x2_test)

# Calculamos el error cuadrático medio (MSE) después de la poda
mse_poda = mean_squared_error(y2_test, y_pred_poda)
print(f'MSE después de la poda: {mse_poda}')
print("R-squared:", r2_score(y2_test, pred2))

plt.figure(figsize=(10,8), dpi=150)
plot_tree(dtree_poda, feature_names=X2.columns);

# Realizamos La validación cruzada con 5 divisiones (k=5)
cross_val_scores = cross_val_score(dtree2, X2, y, cv=5, scoring='r2')

# Imprimimos Los puntajes R-squared en cada pliegue
print("R-squared en cada pliegue:", cross_val_scores)
```

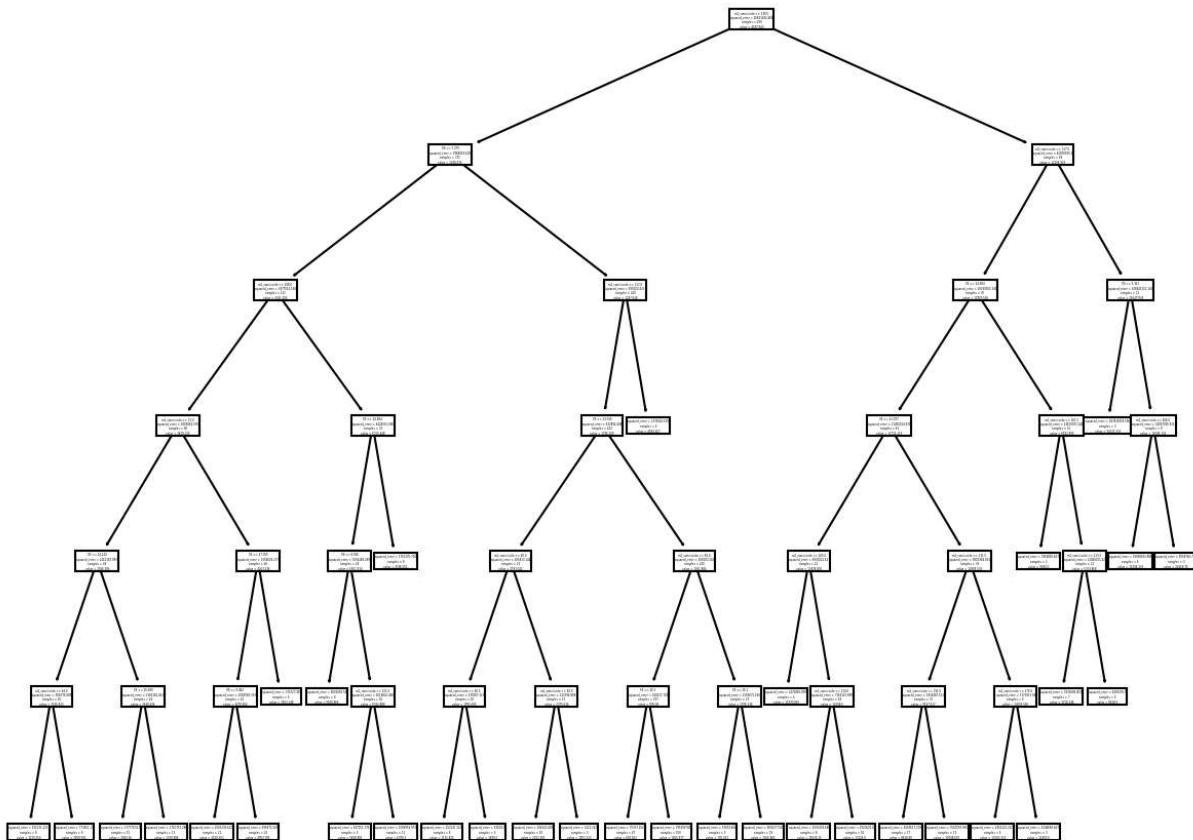
```
# Imprimimos el R-squared promedio
print("R-squared promedio:", cross_val_scores.mean())
```

MSE después de la poda: 4529472.42155807

R-squared: 0.5972527657872242

R-squared en cada pliegue: [0.73309841 0.85260213 0.02437481 0.57751551 0.75084187]

R-squared promedio: 0.5876865471568692



En este observamos un árbol con menos overfitting, así que es el seleccionado para comparar con otras técnicas. Sin embargo, también observamos que tiene una R^2 de 58%, lo que ya no es tan bueno para comparar con otras técnicas.

Redes Neuronales ANN

En esta fase del análisis, hemos decidido emplear redes neuronales para realizar una regresión lineal y, así, determinar el valor por metro cuadrado de los inmuebles. Las redes neuronales son modelos flexibles que pueden capturar patrones complejos en los datos, y la elección de una regresión lineal dentro de la red nos permite mantener la interpretabilidad asociada con este enfoque clásico. Este enfoque nos permitirá evaluar el rendimiento del modelo neuronal en comparación con otros métodos de regresión, contribuyendo así a la comprensión integral de la relación entre las características de los inmuebles y sus precios por metro cuadrado.

En la arquitectura de la red neuronal para regresión que hemos definido, se utilizan tres capas densas (fully connected). La primera capa tiene 35 unidades y utiliza la función de activación ReLU (Rectified Linear Unit). La segunda capa, también con 35 unidades y función de activación ReLU, sigue procesando la información aprendida en la capa anterior. Finalmente, la tercera capa consiste en una única unidad con activación lineal, encargada de realizar la predicción continua del valor por metro cuadrado.

El modelo se compila utilizando el optimizador Adam y la función de pérdida 'mean_squared_error' para guiar el proceso de aprendizaje. Posteriormente, el modelo se entrena durante 100 épocas, con un tamaño de lote de 32 y una validación del 10% de los datos de entrenamiento para evaluar su desempeño durante el proceso de entrenamiento.

Es importante mencionar que los primeros cuatro modelos fueron basados en los modelos de regresión lineal múltiple, por lo que se ingresarán las mismas variables para cada uno. El último modelo es con variables que consideramos como mejores opciones. Además, sólo imprimiremos los errores y R^2 para comparar entre modelos de Redes Neuronales y finalmente escogeremos el mejor modelo que será comparado con técnicas como Regresión Lineal Múltiple y Árboles de Regresión, para el cual detallaremos más información.

Modelo 1

En este primer modelo ingresaremos 19 variables, todas las numéricas de nuestra base de datos.

```
In [ ]: x = inmu[['X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10", "Cocina_equi  
y = inmu["Precio_m2"]  
  
# Dividir los datos en conjuntos de entrenamiento y prueba  
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_st  
  
# Escalar los datos  
scaler = StandardScaler()  
X_train_scaled = scaler.fit_transform(X_train)  
X_test_scaled = scaler.transform(X_test)  
  
# Definir la arquitectura de la red neuronal para regresión  
model = Sequential()  
model.add(Dense(units=35, activation='relu', input_dim=19))  
model.add(Dense(units=35))  
  
model.add(Dense(units=1, activation='linear'))  
  
# Compilar el modelo para regresión  
model.compile(optimizer='adam', loss='mean_squared_error')  
  
# Entrenar el modelo para regresión  
model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_split=0.1)  
  
# Evaluar el modelo en el conjunto de prueba  
loss = model.evaluate(X_test_scaled, y_test)
```

```
print(f'Error Cuadrático Medio en el conjunto de prueba: {loss:.2f}')
```

Realizar predicciones en el conjunto de prueba

```
predictions = model.predict(X_test_scaled)
```

Visualizar algunas predicciones

```
for i in range(10):
    print(f'Predicción: {predictions[i][0]:.2f}, Valor Real: {y_test.iloc[i]:.2f}')
```

Calcular métricas

```
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
r2 = r2_score(y_test, predictions)
```

Mostrar las métricas

```
print(f'Error Cuadrático Medio (MSE): {mse:.2f}')
print(f'Error Absoluto Medio (MAE): {mae:.2f}')
print(f'Coeficiente de Determinación (R^2): {r2:.2f}'')
```

```
WARNING:tensorflow:From c:\Users\maria\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From c:\Users\maria\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/100
WARNING:tensorflow:From c:\Users\maria\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

15/15 [=====] - 5s 66ms/step - loss: 44931852.0000 - val_loss: 37582820.0000
Epoch 2/100
15/15 [=====] - 0s 14ms/step - loss: 44910984.0000 - val_loss: 37568940.0000
Epoch 3/100
15/15 [=====] - 0s 10ms/step - loss: 44881680.0000 - val_loss: 37548616.0000
Epoch 4/100
15/15 [=====] - 0s 14ms/step - loss: 44839068.0000 - val_loss: 37516312.0000
Epoch 5/100
15/15 [=====] - 0s 14ms/step - loss: 44768004.0000 - val_loss: 37465932.0000
Epoch 6/100
15/15 [=====] - 0s 12ms/step - loss: 44661980.0000 - val_loss: 37389620.0000
Epoch 7/100
15/15 [=====] - 0s 10ms/step - loss: 44498364.0000 - val_loss: 37279736.0000
Epoch 8/100
15/15 [=====] - 0s 12ms/step - loss: 44278864.0000 - val_loss: 37116760.0000
Epoch 9/100
15/15 [=====] - 0s 16ms/step - loss: 43957988.0000 - val_loss: 36907384.0000
Epoch 10/100
15/15 [=====] - 0s 18ms/step - loss: 43553592.0000 - val_loss: 36632420.0000
Epoch 11/100
15/15 [=====] - 0s 17ms/step - loss: 43010632.0000 - val_loss: 36286148.0000
Epoch 12/100
15/15 [=====] - 0s 14ms/step - loss: 42354228.0000 - val_loss: 35859552.0000
Epoch 13/100
15/15 [=====] - 0s 30ms/step - loss: 41601028.0000 - val_loss: 35330180.0000
Epoch 14/100
15/15 [=====] - 0s 16ms/step - loss: 40630012.0000 - val_loss: 34742288.0000
Epoch 15/100
15/15 [=====] - 0s 29ms/step - loss: 39576508.0000 - val_loss:
```

```
ss: 34050880.0000
Epoch 16/100
15/15 [=====] - 0s 23ms/step - loss: 38355628.0000 - val_lo
ss: 33260940.0000
Epoch 17/100
15/15 [=====] - 0s 18ms/step - loss: 37023600.0000 - val_lo
ss: 32386120.0000
Epoch 18/100
15/15 [=====] - 0s 20ms/step - loss: 35547344.0000 - val_lo
ss: 31454146.0000
Epoch 19/100
15/15 [=====] - 0s 15ms/step - loss: 33894476.0000 - val_lo
ss: 30495664.0000
Epoch 20/100
15/15 [=====] - 0s 16ms/step - loss: 32231622.0000 - val_lo
ss: 29425184.0000
Epoch 21/100
15/15 [=====] - 0s 17ms/step - loss: 30405420.0000 - val_lo
ss: 28332012.0000
Epoch 22/100
15/15 [=====] - 0s 16ms/step - loss: 28570218.0000 - val_lo
ss: 27194310.0000
Epoch 23/100
15/15 [=====] - 0s 24ms/step - loss: 26743750.0000 - val_lo
ss: 26023310.0000
Epoch 24/100
15/15 [=====] - 0s 17ms/step - loss: 24828620.0000 - val_lo
ss: 24950826.0000
Epoch 25/100
15/15 [=====] - 0s 14ms/step - loss: 23125936.0000 - val_lo
ss: 23824242.0000
Epoch 26/100
15/15 [=====] - 0s 13ms/step - loss: 21452606.0000 - val_lo
ss: 22784864.0000
Epoch 27/100
15/15 [=====] - 0s 9ms/step - loss: 19893118.0000 - val_lo
ss: 21840570.0000
Epoch 28/100
15/15 [=====] - 0s 16ms/step - loss: 18518880.0000 - val_lo
ss: 20978254.0000
Epoch 29/100
15/15 [=====] - 0s 17ms/step - loss: 17294522.0000 - val_lo
ss: 20190754.0000
Epoch 30/100
15/15 [=====] - 0s 18ms/step - loss: 16267553.0000 - val_lo
ss: 19454634.0000
Epoch 31/100
15/15 [=====] - 0s 17ms/step - loss: 15314379.0000 - val_lo
ss: 18833052.0000
Epoch 32/100
15/15 [=====] - 0s 19ms/step - loss: 14484884.0000 - val_lo
ss: 18350736.0000
Epoch 33/100
15/15 [=====] - 0s 16ms/step - loss: 13909082.0000 - val_lo
ss: 17811326.0000
Epoch 34/100
```

```
15/15 [=====] - 0s 18ms/step - loss: 13327834.0000 - val_loss: 17380314.0000
Epoch 35/100
15/15 [=====] - 0s 22ms/step - loss: 12866587.0000 - val_loss: 16999754.0000
Epoch 36/100
15/15 [=====] - 0s 27ms/step - loss: 12456112.0000 - val_loss: 16680152.0000
Epoch 37/100
15/15 [=====] - 0s 18ms/step - loss: 12159178.0000 - val_loss: 16345002.0000
Epoch 38/100
15/15 [=====] - 0s 19ms/step - loss: 11861058.0000 - val_loss: 16049386.0000
Epoch 39/100
15/15 [=====] - 0s 17ms/step - loss: 11561626.0000 - val_loss: 15807795.0000
Epoch 40/100
15/15 [=====] - 0s 15ms/step - loss: 11331171.0000 - val_loss: 15567908.0000
Epoch 41/100
15/15 [=====] - 0s 19ms/step - loss: 11113290.0000 - val_loss: 15359441.0000
Epoch 42/100
15/15 [=====] - 0s 17ms/step - loss: 10927915.0000 - val_loss: 15119478.0000
Epoch 43/100
15/15 [=====] - 0s 17ms/step - loss: 10720594.0000 - val_loss: 14921414.0000
Epoch 44/100
15/15 [=====] - 0s 17ms/step - loss: 10543667.0000 - val_loss: 14747495.0000
Epoch 45/100
15/15 [=====] - 0s 20ms/step - loss: 10385529.0000 - val_loss: 14557947.0000
Epoch 46/100
15/15 [=====] - 0s 13ms/step - loss: 10210881.0000 - val_loss: 14407231.0000
Epoch 47/100
15/15 [=====] - 0s 24ms/step - loss: 10097264.0000 - val_loss: 14238679.0000
Epoch 48/100
15/15 [=====] - 0s 19ms/step - loss: 9931850.0000 - val_loss: 14094621.0000
Epoch 49/100
15/15 [=====] - 0s 19ms/step - loss: 9807143.0000 - val_loss: 13959358.0000
Epoch 50/100
15/15 [=====] - 0s 18ms/step - loss: 9696755.0000 - val_loss: 13819958.0000
Epoch 51/100
15/15 [=====] - 0s 18ms/step - loss: 9576443.0000 - val_loss: 13700115.0000
Epoch 52/100
15/15 [=====] - 0s 18ms/step - loss: 9480010.0000 - val_loss: 13575559.0000
```

Epoch 53/100
15/15 [=====] - 0s 17ms/step - loss: 9370885.0000 - val_loss:
s: 13474443.0000
Epoch 54/100
15/15 [=====] - 0s 20ms/step - loss: 9281631.0000 - val_loss:
s: 13361366.0000
Epoch 55/100
15/15 [=====] - 0s 18ms/step - loss: 9181974.0000 - val_loss:
s: 13262457.0000
Epoch 56/100
15/15 [=====] - 0s 19ms/step - loss: 9105200.0000 - val_loss:
s: 13164249.0000
Epoch 57/100
15/15 [=====] - 0s 16ms/step - loss: 9020449.0000 - val_loss:
s: 13074802.0000
Epoch 58/100
15/15 [=====] - 0s 15ms/step - loss: 8954543.0000 - val_loss:
s: 12991657.0000
Epoch 59/100
15/15 [=====] - 0s 20ms/step - loss: 8879193.0000 - val_loss:
s: 12928710.0000
Epoch 60/100
15/15 [=====] - 0s 17ms/step - loss: 8825253.0000 - val_loss:
s: 12852351.0000
Epoch 61/100
15/15 [=====] - 0s 18ms/step - loss: 8747985.0000 - val_loss:
s: 12778457.0000
Epoch 62/100
15/15 [=====] - 0s 22ms/step - loss: 8692157.0000 - val_loss:
s: 12708113.0000
Epoch 63/100
15/15 [=====] - 0s 13ms/step - loss: 8646391.0000 - val_loss:
s: 12642678.0000
Epoch 64/100
15/15 [=====] - 0s 18ms/step - loss: 8577646.0000 - val_loss:
s: 12586605.0000
Epoch 65/100
15/15 [=====] - 0s 14ms/step - loss: 8531106.0000 - val_loss:
s: 12523369.0000
Epoch 66/100
15/15 [=====] - 0s 13ms/step - loss: 8479733.0000 - val_loss:
s: 12473380.0000
Epoch 67/100
15/15 [=====] - 0s 12ms/step - loss: 8431764.0000 - val_loss:
s: 12423982.0000
Epoch 68/100
15/15 [=====] - 0s 12ms/step - loss: 8389963.0000 - val_loss:
s: 12381459.0000
Epoch 69/100
15/15 [=====] - 0s 16ms/step - loss: 8342568.5000 - val_loss:
s: 12336220.0000
Epoch 70/100
15/15 [=====] - 0s 14ms/step - loss: 8314186.5000 - val_loss:
s: 12291468.0000
Epoch 71/100
15/15 [=====] - 0s 16ms/step - loss: 8266570.5000 - val_loss:

```
s: 12254474.0000
Epoch 72/100
15/15 [=====] - 0s 14ms/step - loss: 8230643.0000 - val_los
s: 12210975.0000
Epoch 73/100
15/15 [=====] - 0s 14ms/step - loss: 8193472.0000 - val_los
s: 12186977.0000
Epoch 74/100
15/15 [=====] - 0s 12ms/step - loss: 8159138.5000 - val_los
s: 12143937.0000
Epoch 75/100
15/15 [=====] - 0s 17ms/step - loss: 8132102.5000 - val_los
s: 12111448.0000
Epoch 76/100
15/15 [=====] - 0s 16ms/step - loss: 8105455.5000 - val_los
s: 12086052.0000
Epoch 77/100
15/15 [=====] - 0s 16ms/step - loss: 8063259.0000 - val_los
s: 12062049.0000
Epoch 78/100
15/15 [=====] - 0s 14ms/step - loss: 8032135.0000 - val_los
s: 12032091.0000
Epoch 79/100
15/15 [=====] - 0s 21ms/step - loss: 8002481.0000 - val_los
s: 12008512.0000
Epoch 80/100
15/15 [=====] - 0s 14ms/step - loss: 7974699.5000 - val_los
s: 11978933.0000
Epoch 81/100
15/15 [=====] - 0s 13ms/step - loss: 7947423.0000 - val_los
s: 11950737.0000
Epoch 82/100
15/15 [=====] - 0s 13ms/step - loss: 7924993.5000 - val_los
s: 11932468.0000
Epoch 83/100
15/15 [=====] - 0s 16ms/step - loss: 7900250.0000 - val_los
s: 11902773.0000
Epoch 84/100
15/15 [=====] - 0s 16ms/step - loss: 7865925.5000 - val_los
s: 11882468.0000
Epoch 85/100
15/15 [=====] - 0s 14ms/step - loss: 7847598.5000 - val_los
s: 11865531.0000
Epoch 86/100
15/15 [=====] - 0s 14ms/step - loss: 7823246.0000 - val_los
s: 11843003.0000
Epoch 87/100
15/15 [=====] - 0s 14ms/step - loss: 7804309.5000 - val_los
s: 11825451.0000
Epoch 88/100
15/15 [=====] - 0s 12ms/step - loss: 7787302.5000 - val_los
s: 11806038.0000
Epoch 89/100
15/15 [=====] - 0s 15ms/step - loss: 7760451.5000 - val_los
s: 11793493.0000
Epoch 90/100
```

```
15/15 [=====] - 0s 14ms/step - loss: 7757454.0000 - val_loss:  
s: 11772293.0000  
Epoch 91/100  
15/15 [=====] - 0s 14ms/step - loss: 7715395.5000 - val_loss:  
s: 11755353.0000  
Epoch 92/100  
15/15 [=====] - 0s 21ms/step - loss: 7708790.0000 - val_loss:  
s: 11741842.0000  
Epoch 93/100  
15/15 [=====] - 0s 13ms/step - loss: 7670828.5000 - val_loss:  
s: 11734393.0000  
Epoch 94/100  
15/15 [=====] - 0s 14ms/step - loss: 7659217.5000 - val_loss:  
s: 11721757.0000  
Epoch 95/100  
15/15 [=====] - 0s 16ms/step - loss: 7636732.0000 - val_loss:  
s: 11705454.0000  
Epoch 96/100  
15/15 [=====] - 0s 14ms/step - loss: 7627970.0000 - val_loss:  
s: 11690390.0000  
Epoch 97/100  
15/15 [=====] - 0s 17ms/step - loss: 7602041.0000 - val_loss:  
s: 11681925.0000  
Epoch 98/100  
15/15 [=====] - 0s 18ms/step - loss: 7584359.5000 - val_loss:  
s: 11674277.0000  
Epoch 99/100  
15/15 [=====] - 0s 17ms/step - loss: 7568575.0000 - val_loss:  
s: 11660667.0000  
Epoch 100/100  
15/15 [=====] - 0s 17ms/step - loss: 7552993.5000 - val_loss:  
s: 11649171.0000  
5/5 [=====] - 0s 4ms/step - loss: 5165522.5000  
Error Cuadrático Medio en el conjunto de prueba: 5165522.50  
5/5 [=====] - 0s 4ms/step  
Predicción: 9289.23, Valor Real: 5115.00  
Predicción: 481.43, Valor Real: 505.84  
Predicción: 1321.89, Valor Real: 2350.00  
Predicción: 1141.93, Valor Real: 572.60  
Predicción: 885.50, Valor Real: 920.00  
Predicción: 5790.83, Valor Real: 3300.00  
Predicción: 9667.73, Valor Real: 6400.00  
Predicción: 143.08, Valor Real: 965.00  
Predicción: 33.10, Valor Real: 357.60  
Predicción: 12405.64, Valor Real: 12200.00  
Error Cuadrático Medio (MSE): 5165522.45  
Error Absoluto Medio (MAE): 1502.25  
Coeficiente de Determinación (R^2): 0.69
```

Observamos los valores antes mencionados. Los errores MSE y MAE serán comparados con otros modelos y observamos que nuestra variable objetivo se explica un 69% con este modelo, lo cual es un buen número pero podemos mejorarlo.

Modelo 2

En este ingresamos las mismas variables que las de regresión lineal multiple, algunas sobre la descripción del lugar donde se encuentran las casas y otras como si cuentan con alberca y numeros de baños, récamaras y lugares de estacionamiento. En esta, observamso los siguientes resultados.

```
In [ ]: X = inmu[['X2", "X3", "X5", "X8", "X10", "Alberca", "Baños", "Recamaras", "Lugares_e  
y = inmu["Precio_m2"]  
  
# Dividir los datos en conjuntos de entrenamiento y prueba  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st  
  
# Escalar los datos  
scaler = StandardScaler()  
X_train_scaled = scaler.fit_transform(X_train)  
X_test_scaled = scaler.transform(X_test)  
  
# Definir la arquitectura de la red neuronal para regresión  
model = Sequential()  
model.add(Dense(units=35, activation='relu', input_dim=9))  
model.add(Dense(units=35))  
  
model.add(Dense(units=1, activation='linear'))  
  
# Compilar el modelo para regresión  
model.compile(optimizer='adam', loss='mean_squared_error')  
  
# Entrenar el modelo para regresión  
model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_split=0.1)  
  
# Evaluar el modelo en el conjunto de prueba  
loss = model.evaluate(X_test_scaled, y_test)  
print(f'Error Cuadrático Medio en el conjunto de prueba: {loss:.2f}')  
  
# Realizar predicciones en el conjunto de prueba  
predictions = model.predict(X_test_scaled)  
  
# Visualizar algunas predicciones  
for i in range(10):  
    print(f'Predicción: {predictions[i][0]:.2f}, Valor Real: {y_test.iloc[i]:.2f}')  
  
# Calcular métricas  
mse = mean_squared_error(y_test, predictions)  
mae = mean_absolute_error(y_test, predictions)  
r2 = r2_score(y_test, predictions)  
  
# Mostrar las métricas  
print(f'Error Cuadrático Medio (MSE): {mse:.2f}')  
print(f'Error Absoluto Medio (MAE): {mae:.2f}')  
print(f'Coeficiente de Determinación (R^2): {r2:.2f}')
```

Epoch 1/100
15/15 [=====] - 3s 65ms/step - loss: 44938268.0000 - val_loss: 37587240.0000
Epoch 2/100
15/15 [=====] - 0s 14ms/step - loss: 44924000.0000 - val_loss: 37576220.0000
Epoch 3/100
15/15 [=====] - 0s 17ms/step - loss: 44908148.0000 - val_loss: 37561480.0000
Epoch 4/100
15/15 [=====] - 0s 17ms/step - loss: 44885532.0000 - val_loss: 37541624.0000
Epoch 5/100
15/15 [=====] - 0s 13ms/step - loss: 44852496.0000 - val_loss: 37512152.0000
Epoch 6/100
15/15 [=====] - 0s 12ms/step - loss: 44801916.0000 - val_loss: 37467992.0000
Epoch 7/100
15/15 [=====] - 0s 18ms/step - loss: 44727160.0000 - val_loss: 37403720.0000
Epoch 8/100
15/15 [=====] - 0s 16ms/step - loss: 44617612.0000 - val_loss: 37315368.0000
Epoch 9/100
15/15 [=====] - 0s 17ms/step - loss: 44475936.0000 - val_loss: 37192836.0000
Epoch 10/100
15/15 [=====] - 0s 17ms/step - loss: 44274524.0000 - val_loss: 37037960.0000
Epoch 11/100
15/15 [=====] - 0s 13ms/step - loss: 44005996.0000 - val_loss: 36846444.0000
Epoch 12/100
15/15 [=====] - 0s 24ms/step - loss: 43688004.0000 - val_loss: 36596512.0000
Epoch 13/100
15/15 [=====] - 0s 15ms/step - loss: 43288384.0000 - val_loss: 36286180.0000
Epoch 14/100
15/15 [=====] - 0s 21ms/step - loss: 42796028.0000 - val_loss: 35923724.0000
Epoch 15/100
15/15 [=====] - 0s 14ms/step - loss: 42222032.0000 - val_loss: 35490968.0000
Epoch 16/100
15/15 [=====] - 0s 21ms/step - loss: 41542288.0000 - val_loss: 34982352.0000
Epoch 17/100
15/15 [=====] - 0s 16ms/step - loss: 40722352.0000 - val_loss: 34421908.0000
Epoch 18/100
15/15 [=====] - 0s 15ms/step - loss: 39845512.0000 - val_loss: 33762404.0000
Epoch 19/100
15/15 [=====] - 0s 17ms/step - loss: 38831500.0000 - val_loss:

```
ss: 33033448.0000
Epoch 20/100
15/15 [=====] - 0s 15ms/step - loss: 37733732.0000 - val_lo
ss: 32242538.0000
Epoch 21/100
15/15 [=====] - 0s 15ms/step - loss: 36495860.0000 - val_lo
ss: 31405450.0000
Epoch 22/100
15/15 [=====] - 0s 14ms/step - loss: 35218636.0000 - val_lo
ss: 30497824.0000
Epoch 23/100
15/15 [=====] - 0s 14ms/step - loss: 33839488.0000 - val_lo
ss: 29532568.0000
Epoch 24/100
15/15 [=====] - 0s 24ms/step - loss: 32395680.0000 - val_lo
ss: 28510526.0000
Epoch 25/100
15/15 [=====] - 0s 13ms/step - loss: 30875616.0000 - val_lo
ss: 27463130.0000
Epoch 26/100
15/15 [=====] - 0s 17ms/step - loss: 29245340.0000 - val_lo
ss: 26418342.0000
Epoch 27/100
15/15 [=====] - 0s 16ms/step - loss: 27736696.0000 - val_lo
ss: 25270376.0000
Epoch 28/100
15/15 [=====] - 0s 19ms/step - loss: 26031434.0000 - val_lo
ss: 24195166.0000
Epoch 29/100
15/15 [=====] - 0s 14ms/step - loss: 24419706.0000 - val_lo
ss: 23091350.0000
Epoch 30/100
15/15 [=====] - 0s 22ms/step - loss: 22888382.0000 - val_lo
ss: 21958710.0000
Epoch 31/100
15/15 [=====] - 0s 19ms/step - loss: 21249798.0000 - val_lo
ss: 20922078.0000
Epoch 32/100
15/15 [=====] - 0s 17ms/step - loss: 19736388.0000 - val_lo
ss: 19945240.0000
Epoch 33/100
15/15 [=====] - 0s 13ms/step - loss: 18286046.0000 - val_lo
ss: 19049414.0000
Epoch 34/100
15/15 [=====] - 0s 16ms/step - loss: 16994562.0000 - val_lo
ss: 18130970.0000
Epoch 35/100
15/15 [=====] - 0s 16ms/step - loss: 15847100.0000 - val_lo
ss: 17211832.0000
Epoch 36/100
15/15 [=====] - 0s 18ms/step - loss: 14629669.0000 - val_lo
ss: 16482762.0000
Epoch 37/100
15/15 [=====] - 0s 19ms/step - loss: 13714419.0000 - val_lo
ss: 15763665.0000
Epoch 38/100
```

```
15/15 [=====] - 0s 16ms/step - loss: 12792142.0000 - val_loss: 15179968.0000
Epoch 39/100
15/15 [=====] - 0s 24ms/step - loss: 12071565.0000 - val_loss: 14639086.0000
Epoch 40/100
15/15 [=====] - 0s 13ms/step - loss: 11418955.0000 - val_loss: 14188864.0000
Epoch 41/100
15/15 [=====] - 0s 14ms/step - loss: 10905101.0000 - val_loss: 13766747.0000
Epoch 42/100
15/15 [=====] - 0s 16ms/step - loss: 10449780.0000 - val_loss: 13403780.0000
Epoch 43/100
15/15 [=====] - 0s 18ms/step - loss: 10066583.0000 - val_loss: 13122306.0000
Epoch 44/100
15/15 [=====] - 0s 16ms/step - loss: 9784706.0000 - val_loss: 12872635.0000
Epoch 45/100
15/15 [=====] - 0s 15ms/step - loss: 9535181.0000 - val_loss: 12659618.0000
Epoch 46/100
15/15 [=====] - 0s 16ms/step - loss: 9332142.0000 - val_loss: 12484677.0000
Epoch 47/100
15/15 [=====] - 0s 14ms/step - loss: 9172891.0000 - val_loss: 12331144.0000
Epoch 48/100
15/15 [=====] - 0s 14ms/step - loss: 9042041.0000 - val_loss: 12202905.0000
Epoch 49/100
15/15 [=====] - 0s 15ms/step - loss: 8942889.0000 - val_loss: 12085898.0000
Epoch 50/100
15/15 [=====] - 0s 14ms/step - loss: 8835502.0000 - val_loss: 11993938.0000
Epoch 51/100
15/15 [=====] - 0s 15ms/step - loss: 8753705.0000 - val_loss: 11915568.0000
Epoch 52/100
15/15 [=====] - 0s 15ms/step - loss: 8699087.0000 - val_loss: 11830946.0000
Epoch 53/100
15/15 [=====] - 0s 16ms/step - loss: 8635685.0000 - val_loss: 11759556.0000
Epoch 54/100
15/15 [=====] - 0s 15ms/step - loss: 8569163.0000 - val_loss: 11712803.0000
Epoch 55/100
15/15 [=====] - 0s 16ms/step - loss: 8529264.0000 - val_loss: 11649529.0000
Epoch 56/100
15/15 [=====] - 0s 15ms/step - loss: 8486084.0000 - val_loss: 11593943.0000
```

Epoch 57/100
15/15 [=====] - 0s 13ms/step - loss: 8434004.0000 - val_loss:
s: 11557668.0000
Epoch 58/100
15/15 [=====] - 0s 26ms/step - loss: 8400202.0000 - val_loss:
s: 11509923.0000
Epoch 59/100
15/15 [=====] - 0s 15ms/step - loss: 8361135.5000 - val_loss:
s: 11473480.0000
Epoch 60/100
15/15 [=====] - 0s 16ms/step - loss: 8334807.5000 - val_loss:
s: 11431260.0000
Epoch 61/100
15/15 [=====] - 0s 15ms/step - loss: 8300175.5000 - val_loss:
s: 11398187.0000
Epoch 62/100
15/15 [=====] - 0s 17ms/step - loss: 8281504.0000 - val_loss:
s: 11358056.0000
Epoch 63/100
15/15 [=====] - 0s 16ms/step - loss: 8240864.5000 - val_loss:
s: 11343288.0000
Epoch 64/100
15/15 [=====] - 0s 15ms/step - loss: 8214365.5000 - val_loss:
s: 11305988.0000
Epoch 65/100
15/15 [=====] - 0s 16ms/step - loss: 8186107.0000 - val_loss:
s: 11281971.0000
Epoch 66/100
15/15 [=====] - 0s 16ms/step - loss: 8163162.0000 - val_loss:
s: 11254905.0000
Epoch 67/100
15/15 [=====] - 0s 15ms/step - loss: 8138268.5000 - val_loss:
s: 11234735.0000
Epoch 68/100
15/15 [=====] - 0s 15ms/step - loss: 8116971.5000 - val_loss:
s: 11209928.0000
Epoch 69/100
15/15 [=====] - 0s 15ms/step - loss: 8099524.0000 - val_loss:
s: 11183911.0000
Epoch 70/100
15/15 [=====] - 0s 12ms/step - loss: 8077190.5000 - val_loss:
s: 11162047.0000
Epoch 71/100
15/15 [=====] - 0s 18ms/step - loss: 8058106.0000 - val_loss:
s: 11145162.0000
Epoch 72/100
15/15 [=====] - 0s 17ms/step - loss: 8039034.5000 - val_loss:
s: 11133155.0000
Epoch 73/100
15/15 [=====] - 1s 62ms/step - loss: 8024187.0000 - val_loss:
s: 11113401.0000
Epoch 74/100
15/15 [=====] - 1s 37ms/step - loss: 8007630.0000 - val_loss:
s: 11093637.0000
Epoch 75/100
15/15 [=====] - 0s 33ms/step - loss: 7985775.5000 - val_loss:

```
s: 11076858.0000
Epoch 76/100
15/15 [=====] - 0s 18ms/step - loss: 7969185.5000 - val_los
s: 11063218.0000
Epoch 77/100
15/15 [=====] - 0s 16ms/step - loss: 7957118.5000 - val_los
s: 11047198.0000
Epoch 78/100
15/15 [=====] - 0s 18ms/step - loss: 7961197.5000 - val_los
s: 11043207.0000
Epoch 79/100
15/15 [=====] - 0s 16ms/step - loss: 7918495.5000 - val_los
s: 11035070.0000
Epoch 80/100
15/15 [=====] - 0s 13ms/step - loss: 7907745.0000 - val_los
s: 11011446.0000
Epoch 81/100
15/15 [=====] - 0s 18ms/step - loss: 7901755.0000 - val_los
s: 10996316.0000
Epoch 82/100
15/15 [=====] - 0s 18ms/step - loss: 7880261.5000 - val_los
s: 10988631.0000
Epoch 83/100
15/15 [=====] - 0s 12ms/step - loss: 7881061.0000 - val_los
s: 10983803.0000
Epoch 84/100
15/15 [=====] - 0s 13ms/step - loss: 7866113.0000 - val_los
s: 10969846.0000
Epoch 85/100
15/15 [=====] - 0s 14ms/step - loss: 7842904.5000 - val_los
s: 10966134.0000
Epoch 86/100
15/15 [=====] - 0s 15ms/step - loss: 7844429.0000 - val_los
s: 10958325.0000
Epoch 87/100
15/15 [=====] - 0s 15ms/step - loss: 7820168.5000 - val_los
s: 10944367.0000
Epoch 88/100
15/15 [=====] - 0s 15ms/step - loss: 7807132.0000 - val_los
s: 10937352.0000
Epoch 89/100
15/15 [=====] - 0s 14ms/step - loss: 7795363.5000 - val_los
s: 10930492.0000
Epoch 90/100
15/15 [=====] - 0s 16ms/step - loss: 7786456.5000 - val_los
s: 10926990.0000
Epoch 91/100
15/15 [=====] - 0s 14ms/step - loss: 7776986.5000 - val_los
s: 10919359.0000
Epoch 92/100
15/15 [=====] - 0s 19ms/step - loss: 7759338.0000 - val_los
s: 10913282.0000
Epoch 93/100
15/15 [=====] - 0s 15ms/step - loss: 7758054.5000 - val_los
s: 10907107.0000
Epoch 94/100
```

```

15/15 [=====] - 0s 17ms/step - loss: 7745410.0000 - val_loss: 10897724.0000
Epoch 95/100
15/15 [=====] - 0s 16ms/step - loss: 7731247.5000 - val_loss: 10889764.0000
Epoch 96/100
15/15 [=====] - 0s 18ms/step - loss: 7727614.5000 - val_loss: 10888657.0000
Epoch 97/100
15/15 [=====] - 0s 17ms/step - loss: 7708734.5000 - val_loss: 10880799.0000
Epoch 98/100
15/15 [=====] - 0s 23ms/step - loss: 7705732.0000 - val_loss: 10872318.0000
Epoch 99/100
15/15 [=====] - 0s 15ms/step - loss: 7689031.0000 - val_loss: 10871294.0000
Epoch 100/100
15/15 [=====] - 0s 15ms/step - loss: 7679919.5000 - val_loss: 10862041.0000
5/5 [=====] - 0s 5ms/step - loss: 5543079.0000
Error Cuadrático Medio en el conjunto de prueba: 5543079.00
5/5 [=====] - 0s 4ms/step
Predicción: 8993.00, Valor Real: 5115.00
Predicción: 876.25, Valor Real: 505.84
Predicción: 2209.62, Valor Real: 2350.00
Predicción: 1623.95, Valor Real: 572.60
Predicción: 1309.15, Valor Real: 920.00
Predicción: 6958.67, Valor Real: 3300.00
Predicción: 9027.41, Valor Real: 6400.00
Predicción: 579.28, Valor Real: 965.00
Predicción: 212.60, Valor Real: 357.60
Predicción: 12193.16, Valor Real: 12200.00
Error Cuadrático Medio (MSE): 5543078.91
Error Absoluto Medio (MAE): 1540.54
Coeficiente de Determinación (R^2): 0.66

```

Valores cercanos de errores a los vistos anteriormente una R² de 65%, por lo que es un modelo menos eficiente que el anterior.

Modelo 3

El siguiente modelo contempla variables como gimnasio, alberca, terraza, elevador, baños, récamaras y lugares de estacionamiento. Observamos los siguientes valores:

```

In [ ]: X = inmu[['Gimnasio', 'Alberca', 'Terraza', 'Elevador', 'Baños', 'Recamaras', 'Lugares_de_estacionamiento']
y = inmu['Precio_m2']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Escalar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

```

```
X_test_scaled = scaler.transform(X_test)

# Definir la arquitectura de la red neuronal para regresión
model = Sequential()
model.add(Dense(units=35, activation='relu', input_dim=7))
model.add(Dense(units=35))

model.add(Dense(units=1, activation='linear'))

# Compilar el modelo para regresión
model.compile(optimizer='adam', loss='mean_squared_error')

# Entrenar el modelo para regresión
model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_split=0.1)

# Evaluar el modelo en el conjunto de prueba
loss = model.evaluate(X_test_scaled, y_test)
print(f'Error Cuadrático Medio en el conjunto de prueba: {loss:.2f}')

# Realizar predicciones en el conjunto de prueba
predictions = model.predict(X_test_scaled)

# Visualizar algunas predicciones
for i in range(10):
    print(f'Predicción: {predictions[i][0]:.2f}, Valor Real: {y_test.iloc[i]:.2f}')

# Calcular métricas
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

# Mostrar las métricas
print(f'Error Cuadrático Medio (MSE): {mse:.2f}')
print(f'Error Absoluto Medio (MAE): {mae:.2f}')
print(f'Coeficiente de Determinación (R^2): {r2:.2f}'')
```

Epoch 1/100
15/15 [=====] - 3s 35ms/step - loss: 44939664.0000 - val_loss: 37586664.0000
Epoch 2/100
15/15 [=====] - 0s 11ms/step - loss: 44925132.0000 - val_loss: 37573628.0000
Epoch 3/100
15/15 [=====] - 0s 21ms/step - loss: 44906848.0000 - val_loss: 37556916.0000
Epoch 4/100
15/15 [=====] - 0s 11ms/step - loss: 44883792.0000 - val_loss: 37531708.0000
Epoch 5/100
15/15 [=====] - 0s 19ms/step - loss: 44845276.0000 - val_loss: 37495720.0000
Epoch 6/100
15/15 [=====] - 0s 16ms/step - loss: 44795568.0000 - val_loss: 37439764.0000
Epoch 7/100
15/15 [=====] - 0s 18ms/step - loss: 44712236.0000 - val_loss: 37364480.0000
Epoch 8/100
15/15 [=====] - 0s 18ms/step - loss: 44602336.0000 - val_loss: 37253196.0000
Epoch 9/100
15/15 [=====] - 0s 17ms/step - loss: 44446008.0000 - val_loss: 37106312.0000
Epoch 10/100
15/15 [=====] - 0s 13ms/step - loss: 44239800.0000 - val_loss: 36920296.0000
Epoch 11/100
15/15 [=====] - 0s 17ms/step - loss: 43971100.0000 - val_loss: 36675816.0000
Epoch 12/100
15/15 [=====] - 0s 23ms/step - loss: 43633764.0000 - val_loss: 36367388.0000
Epoch 13/100
15/15 [=====] - 0s 15ms/step - loss: 43229220.0000 - val_loss: 35995444.0000
Epoch 14/100
15/15 [=====] - 0s 21ms/step - loss: 42732528.0000 - val_loss: 35556704.0000
Epoch 15/100
15/15 [=====] - 0s 24ms/step - loss: 42134252.0000 - val_loss: 35057220.0000
Epoch 16/100
15/15 [=====] - 0s 17ms/step - loss: 41456216.0000 - val_loss: 34458384.0000
Epoch 17/100
15/15 [=====] - 0s 13ms/step - loss: 40681916.0000 - val_loss: 33765408.0000
Epoch 18/100
15/15 [=====] - 0s 18ms/step - loss: 39750092.0000 - val_loss: 33031832.0000
Epoch 19/100
15/15 [=====] - 0s 17ms/step - loss: 38810232.0000 - val_loss:

```
ss: 32153958.0000
Epoch 20/100
15/15 [=====] - 0s 17ms/step - loss: 37674284.0000 - val_lo
ss: 31266482.0000
Epoch 21/100
15/15 [=====] - 0s 12ms/step - loss: 36505364.0000 - val_lo
ss: 30294960.0000
Epoch 22/100
15/15 [=====] - 0s 15ms/step - loss: 35302576.0000 - val_lo
ss: 29237378.0000
Epoch 23/100
15/15 [=====] - 0s 16ms/step - loss: 33954064.0000 - val_lo
ss: 28174362.0000
Epoch 24/100
15/15 [=====] - 0s 15ms/step - loss: 32613532.0000 - val_lo
ss: 27036024.0000
Epoch 25/100
15/15 [=====] - 0s 11ms/step - loss: 31182814.0000 - val_lo
ss: 25875992.0000
Epoch 26/100
15/15 [=====] - 0s 13ms/step - loss: 29675796.0000 - val_lo
ss: 24740414.0000
Epoch 27/100
15/15 [=====] - 0s 16ms/step - loss: 28267844.0000 - val_lo
ss: 23527142.0000
Epoch 28/100
15/15 [=====] - 0s 12ms/step - loss: 26778976.0000 - val_lo
ss: 22355318.0000
Epoch 29/100
15/15 [=====] - 0s 16ms/step - loss: 25308226.0000 - val_lo
ss: 21235316.0000
Epoch 30/100
15/15 [=====] - 0s 16ms/step - loss: 23921504.0000 - val_lo
ss: 20102408.0000
Epoch 31/100
15/15 [=====] - 0s 27ms/step - loss: 22547502.0000 - val_lo
ss: 19037426.0000
Epoch 32/100
15/15 [=====] - 0s 15ms/step - loss: 21261266.0000 - val_lo
ss: 18019370.0000
Epoch 33/100
15/15 [=====] - 0s 12ms/step - loss: 19969158.0000 - val_lo
ss: 17167298.0000
Epoch 34/100
15/15 [=====] - 0s 13ms/step - loss: 18976608.0000 - val_lo
ss: 16259473.0000
Epoch 35/100
15/15 [=====] - 0s 16ms/step - loss: 17854564.0000 - val_lo
ss: 15556993.0000
Epoch 36/100
15/15 [=====] - 0s 16ms/step - loss: 16968382.0000 - val_lo
ss: 14921769.0000
Epoch 37/100
15/15 [=====] - 0s 15ms/step - loss: 16197629.0000 - val_lo
ss: 14323735.0000
Epoch 38/100
```

```
15/15 [=====] - 0s 15ms/step - loss: 15508349.0000 - val_loss: 13828086.0000
Epoch 39/100
15/15 [=====] - 0s 12ms/step - loss: 14883620.0000 - val_loss: 13450617.0000
Epoch 40/100
15/15 [=====] - 0s 13ms/step - loss: 14429899.0000 - val_loss: 13091486.0000
Epoch 41/100
15/15 [=====] - 0s 15ms/step - loss: 13980431.0000 - val_loss: 12819763.0000
Epoch 42/100
15/15 [=====] - 0s 13ms/step - loss: 13652432.0000 - val_loss: 12586728.0000
Epoch 43/100
15/15 [=====] - 0s 13ms/step - loss: 13351372.0000 - val_loss: 12403757.0000
Epoch 44/100
15/15 [=====] - 0s 12ms/step - loss: 13107268.0000 - val_loss: 12265058.0000
Epoch 45/100
15/15 [=====] - 0s 17ms/step - loss: 12911269.0000 - val_loss: 12150638.0000
Epoch 46/100
15/15 [=====] - 0s 23ms/step - loss: 12736847.0000 - val_loss: 12060110.0000
Epoch 47/100
15/15 [=====] - 0s 17ms/step - loss: 12611573.0000 - val_loss: 11963853.0000
Epoch 48/100
15/15 [=====] - 0s 15ms/step - loss: 12468130.0000 - val_loss: 11902635.0000
Epoch 49/100
15/15 [=====] - 0s 14ms/step - loss: 12403935.0000 - val_loss: 11840041.0000
Epoch 50/100
15/15 [=====] - 0s 13ms/step - loss: 12274945.0000 - val_loss: 11800155.0000
Epoch 51/100
15/15 [=====] - 0s 14ms/step - loss: 12193247.0000 - val_loss: 11762879.0000
Epoch 52/100
15/15 [=====] - 0s 14ms/step - loss: 12124120.0000 - val_loss: 11725981.0000
Epoch 53/100
15/15 [=====] - 0s 12ms/step - loss: 12053067.0000 - val_loss: 11695385.0000
Epoch 54/100
15/15 [=====] - 0s 14ms/step - loss: 12008291.0000 - val_loss: 11662276.0000
Epoch 55/100
15/15 [=====] - 0s 13ms/step - loss: 11938274.0000 - val_loss: 11634070.0000
Epoch 56/100
15/15 [=====] - 0s 17ms/step - loss: 11875185.0000 - val_loss: 11609897.0000
```

Epoch 57/100
15/15 [=====] - 0s 14ms/step - loss: 11820598.0000 - val_loss: 11588088.0000
Epoch 58/100
15/15 [=====] - 0s 17ms/step - loss: 11772991.0000 - val_loss: 11565194.0000
Epoch 59/100
15/15 [=====] - 0s 18ms/step - loss: 11721093.0000 - val_loss: 11541836.0000
Epoch 60/100
15/15 [=====] - 0s 15ms/step - loss: 11696303.0000 - val_loss: 11522254.0000
Epoch 61/100
15/15 [=====] - 0s 22ms/step - loss: 11624590.0000 - val_loss: 11499656.0000
Epoch 62/100
15/15 [=====] - 0s 14ms/step - loss: 11586841.0000 - val_loss: 11475534.0000
Epoch 63/100
15/15 [=====] - 0s 13ms/step - loss: 11530779.0000 - val_loss: 11458583.0000
Epoch 64/100
15/15 [=====] - 0s 15ms/step - loss: 11491376.0000 - val_loss: 11439519.0000
Epoch 65/100
15/15 [=====] - 0s 16ms/step - loss: 11458821.0000 - val_loss: 11425082.0000
Epoch 66/100
15/15 [=====] - 0s 17ms/step - loss: 11408474.0000 - val_loss: 11402083.0000
Epoch 67/100
15/15 [=====] - 0s 16ms/step - loss: 11373950.0000 - val_loss: 11383446.0000
Epoch 68/100
15/15 [=====] - 0s 16ms/step - loss: 11338530.0000 - val_loss: 11372573.0000
Epoch 69/100
15/15 [=====] - 0s 18ms/step - loss: 11292310.0000 - val_loss: 11356544.0000
Epoch 70/100
15/15 [=====] - 0s 14ms/step - loss: 11258525.0000 - val_loss: 11335430.0000
Epoch 71/100
15/15 [=====] - 0s 14ms/step - loss: 11219158.0000 - val_loss: 11319339.0000
Epoch 72/100
15/15 [=====] - 0s 12ms/step - loss: 11182391.0000 - val_loss: 11308666.0000
Epoch 73/100
15/15 [=====] - 0s 16ms/step - loss: 11145050.0000 - val_loss: 11288173.0000
Epoch 74/100
15/15 [=====] - 0s 15ms/step - loss: 11115075.0000 - val_loss: 11275461.0000
Epoch 75/100
15/15 [=====] - 0s 14ms/step - loss: 11074847.0000 - val_loss:

```
ss: 11258233.0000
Epoch 76/100
15/15 [=====] - 0s 24ms/step - loss: 11049237.0000 - val_lo
ss: 11248029.0000
Epoch 77/100
15/15 [=====] - 0s 15ms/step - loss: 11017911.0000 - val_lo
ss: 11227961.0000
Epoch 78/100
15/15 [=====] - 0s 14ms/step - loss: 10978215.0000 - val_lo
ss: 11216972.0000
Epoch 79/100
15/15 [=====] - 0s 15ms/step - loss: 10950261.0000 - val_lo
ss: 11202512.0000
Epoch 80/100
15/15 [=====] - 0s 16ms/step - loss: 10921528.0000 - val_lo
ss: 11188252.0000
Epoch 81/100
15/15 [=====] - 0s 11ms/step - loss: 10890069.0000 - val_lo
ss: 11177744.0000
Epoch 82/100
15/15 [=====] - 0s 14ms/step - loss: 10858421.0000 - val_lo
ss: 11167669.0000
Epoch 83/100
15/15 [=====] - 0s 14ms/step - loss: 10834631.0000 - val_lo
ss: 11156228.0000
Epoch 84/100
15/15 [=====] - 0s 18ms/step - loss: 10818868.0000 - val_lo
ss: 11146041.0000
Epoch 85/100
15/15 [=====] - 0s 17ms/step - loss: 10778833.0000 - val_lo
ss: 11126227.0000
Epoch 86/100
15/15 [=====] - 0s 20ms/step - loss: 10757323.0000 - val_lo
ss: 11118312.0000
Epoch 87/100
15/15 [=====] - 0s 15ms/step - loss: 10722580.0000 - val_lo
ss: 11103399.0000
Epoch 88/100
15/15 [=====] - 0s 13ms/step - loss: 10707017.0000 - val_lo
ss: 11086875.0000
Epoch 89/100
15/15 [=====] - 0s 17ms/step - loss: 10675115.0000 - val_lo
ss: 11078258.0000
Epoch 90/100
15/15 [=====] - 0s 20ms/step - loss: 10650884.0000 - val_lo
ss: 11068889.0000
Epoch 91/100
15/15 [=====] - 0s 13ms/step - loss: 10625943.0000 - val_lo
ss: 11053359.0000
Epoch 92/100
15/15 [=====] - 0s 16ms/step - loss: 10607324.0000 - val_lo
ss: 11038105.0000
Epoch 93/100
15/15 [=====] - 0s 13ms/step - loss: 10582123.0000 - val_lo
ss: 11029638.0000
Epoch 94/100
```

```

15/15 [=====] - 0s 13ms/step - loss: 10562202.0000 - val_lo
ss: 11020749.0000
Epoch 95/100
15/15 [=====] - 0s 13ms/step - loss: 10546270.0000 - val_lo
ss: 11016601.0000
Epoch 96/100
15/15 [=====] - 0s 16ms/step - loss: 10510313.0000 - val_lo
ss: 11003614.0000
Epoch 97/100
15/15 [=====] - 0s 13ms/step - loss: 10494648.0000 - val_lo
ss: 10994500.0000
Epoch 98/100
15/15 [=====] - 0s 15ms/step - loss: 10477073.0000 - val_lo
ss: 10970882.0000
Epoch 99/100
15/15 [=====] - 0s 15ms/step - loss: 10457134.0000 - val_lo
ss: 10973609.0000
Epoch 100/100
15/15 [=====] - 0s 12ms/step - loss: 10436203.0000 - val_lo
ss: 10962219.0000
5/5 [=====] - 0s 8ms/step - loss: 5196798.5000
Error Cuadrático Medio en el conjunto de prueba: 5196798.50
5/5 [=====] - 0s 4ms/step
Predicción: 5241.20, Valor Real: 5115.00
Predicción: 52.68, Valor Real: 505.84
Predicción: 77.89, Valor Real: 2350.00
Predicción: 72.41, Valor Real: 572.60
Predicción: 52.68, Valor Real: 920.00
Predicción: 4154.68, Valor Real: 3300.00
Predicción: 9493.38, Valor Real: 6400.00
Predicción: 77.89, Valor Real: 965.00
Predicción: 77.89, Valor Real: 357.60
Predicción: 15919.26, Valor Real: 12200.00
Error Cuadrático Medio (MSE): 5196798.14
Error Absoluto Medio (MAE): 1705.34
Coeficiente de Determinación (R^2): 0.68

```

Este se presenta errores menores a los anteriores, lo que es un buen indicio, y un coeficiente de determinación del 74%. En conclusión tenemos un mejor modelo que los anteriores.

Modelo 4

El último modelo que se empleó en regresión lineal múltiple contiene las variables como gimnasio, alberca, terraza, baños y lugares de estacionamiento, por lo que utilizaremos esas mismas variables en este mismo.

```

In [ ]: X = inmu[["Gimnasio", "Alberca", "Terraza", "Baños", "Lugares_estac"]]
y = inmu["Precio_m2"]

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
# Escalar los datos
scaler = StandardScaler()

```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Definir la arquitectura de la red neuronal para regresión
model = Sequential()
model.add(Dense(units=35, activation='relu', input_dim=5))
model.add(Dense(units=35))

model.add(Dense(units=1, activation='linear'))

# Compilar el modelo para regresión
model.compile(optimizer='adam', loss='mean_squared_error')

# Entrenar el modelo para regresión
model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_split=0.1)

# Evaluar el modelo en el conjunto de prueba
loss = model.evaluate(X_test_scaled, y_test)
print(f'Error Cuadrático Medio en el conjunto de prueba: {loss:.2f}')

# Realizar predicciones en el conjunto de prueba
predictions = model.predict(X_test_scaled)

# Visualizar algunas predicciones
for i in range(10):
    print(f'Predicción: {predictions[i][0]:.2f}, Valor Real: {y_test.iloc[i]:.2f}')

# Calcular métricas
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

# Mostrar las métricas
print(f'Error Cuadrático Medio (MSE): {mse:.2f}')
print(f'Error Absoluto Medio (MAE): {mae:.2f}')
print(f'Coeficiente de Determinación (R^2): {r2:.2f}')
```

Epoch 1/100
15/15 [=====] - 3s 54ms/step - loss: 44939560.0000 - val_loss: 37588048.0000
Epoch 2/100
15/15 [=====] - 0s 17ms/step - loss: 44928040.0000 - val_loss: 37577232.0000
Epoch 3/100
15/15 [=====] - 0s 21ms/step - loss: 44914524.0000 - val_loss: 37562732.0000
Epoch 4/100
15/15 [=====] - 0s 14ms/step - loss: 44894704.0000 - val_loss: 37542572.0000
Epoch 5/100
15/15 [=====] - 0s 16ms/step - loss: 44865672.0000 - val_loss: 37514116.0000
Epoch 6/100
15/15 [=====] - 0s 13ms/step - loss: 44825716.0000 - val_loss: 37473148.0000
Epoch 7/100
15/15 [=====] - 0s 13ms/step - loss: 44770120.0000 - val_loss: 37415352.0000
Epoch 8/100
15/15 [=====] - 0s 15ms/step - loss: 44691872.0000 - val_loss: 37339676.0000
Epoch 9/100
15/15 [=====] - 0s 15ms/step - loss: 44586076.0000 - val_loss: 37239800.0000
Epoch 10/100
15/15 [=====] - 0s 16ms/step - loss: 44450496.0000 - val_loss: 37106992.0000
Epoch 11/100
15/15 [=====] - 0s 13ms/step - loss: 44273324.0000 - val_loss: 36938340.0000
Epoch 12/100
15/15 [=====] - 0s 15ms/step - loss: 44049904.0000 - val_loss: 36726464.0000
Epoch 13/100
15/15 [=====] - 0s 15ms/step - loss: 43777288.0000 - val_loss: 36469548.0000
Epoch 14/100
15/15 [=====] - 0s 15ms/step - loss: 43446016.0000 - val_loss: 36167072.0000
Epoch 15/100
15/15 [=====] - 0s 17ms/step - loss: 43047236.0000 - val_loss: 35819060.0000
Epoch 16/100
15/15 [=====] - 0s 19ms/step - loss: 42594272.0000 - val_loss: 35412972.0000
Epoch 17/100
15/15 [=====] - 0s 14ms/step - loss: 42071200.0000 - val_loss: 34949916.0000
Epoch 18/100
15/15 [=====] - 0s 21ms/step - loss: 41471012.0000 - val_loss: 34429580.0000
Epoch 19/100
15/15 [=====] - 0s 17ms/step - loss: 40794404.0000 - val_loss:

```
ss: 33855284.0000
Epoch 20/100
15/15 [=====] - 0s 12ms/step - loss: 40060836.0000 - val_lo
ss: 33204938.0000
Epoch 21/100
15/15 [=====] - 0s 15ms/step - loss: 39242948.0000 - val_lo
ss: 32491708.0000
Epoch 22/100
15/15 [=====] - 0s 13ms/step - loss: 38339828.0000 - val_lo
ss: 31744780.0000
Epoch 23/100
15/15 [=====] - 0s 12ms/step - loss: 37405376.0000 - val_lo
ss: 30916286.0000
Epoch 24/100
15/15 [=====] - 0s 12ms/step - loss: 36368268.0000 - val_lo
ss: 30046222.0000
Epoch 25/100
15/15 [=====] - 0s 17ms/step - loss: 35227664.0000 - val_lo
ss: 29176890.0000
Epoch 26/100
15/15 [=====] - 0s 15ms/step - loss: 34106200.0000 - val_lo
ss: 28207010.0000
Epoch 27/100
15/15 [=====] - 0s 13ms/step - loss: 32957040.0000 - val_lo
ss: 27162846.0000
Epoch 28/100
15/15 [=====] - 0s 14ms/step - loss: 31635752.0000 - val_lo
ss: 26178164.0000
Epoch 29/100
15/15 [=====] - 0s 13ms/step - loss: 30365128.0000 - val_lo
ss: 25162214.0000
Epoch 30/100
15/15 [=====] - 0s 16ms/step - loss: 29042102.0000 - val_lo
ss: 24127700.0000
Epoch 31/100
15/15 [=====] - 0s 14ms/step - loss: 27743408.0000 - val_lo
ss: 23036990.0000
Epoch 32/100
15/15 [=====] - 0s 13ms/step - loss: 26427206.0000 - val_lo
ss: 21939192.0000
Epoch 33/100
15/15 [=====] - 0s 13ms/step - loss: 25033658.0000 - val_lo
ss: 20913594.0000
Epoch 34/100
15/15 [=====] - 0s 23ms/step - loss: 23690930.0000 - val_lo
ss: 19945284.0000
Epoch 35/100
15/15 [=====] - 0s 12ms/step - loss: 22487162.0000 - val_lo
ss: 18913614.0000
Epoch 36/100
15/15 [=====] - 0s 12ms/step - loss: 21203612.0000 - val_lo
ss: 17978744.0000
Epoch 37/100
15/15 [=====] - 0s 17ms/step - loss: 20043030.0000 - val_lo
ss: 17073972.0000
Epoch 38/100
```

```
15/15 [=====] - 0s 15ms/step - loss: 18898608.0000 - val_loss: 16257643.0000
Epoch 39/100
15/15 [=====] - 0s 14ms/step - loss: 17852706.0000 - val_loss: 15488238.0000
Epoch 40/100
15/15 [=====] - 0s 15ms/step - loss: 16875428.0000 - val_loss: 14791781.0000
Epoch 41/100
15/15 [=====] - 0s 14ms/step - loss: 15960043.0000 - val_loss: 14191277.0000
Epoch 42/100
15/15 [=====] - 0s 14ms/step - loss: 15181010.0000 - val_loss: 13632414.0000
Epoch 43/100
15/15 [=====] - 0s 16ms/step - loss: 14499928.0000 - val_loss: 13123089.0000
Epoch 44/100
15/15 [=====] - 0s 16ms/step - loss: 13793118.0000 - val_loss: 12731824.0000
Epoch 45/100
15/15 [=====] - 0s 14ms/step - loss: 13284918.0000 - val_loss: 12355760.0000
Epoch 46/100
15/15 [=====] - 0s 12ms/step - loss: 12764587.0000 - val_loss: 12069695.0000
Epoch 47/100
15/15 [=====] - 0s 11ms/step - loss: 12374147.0000 - val_loss: 11810314.0000
Epoch 48/100
15/15 [=====] - 0s 19ms/step - loss: 12024376.0000 - val_loss: 11600604.0000
Epoch 49/100
15/15 [=====] - 0s 22ms/step - loss: 11705951.0000 - val_loss: 11444650.0000
Epoch 50/100
15/15 [=====] - 0s 11ms/step - loss: 11485547.0000 - val_loss: 11284110.0000
Epoch 51/100
15/15 [=====] - 0s 15ms/step - loss: 11251994.0000 - val_loss: 11175419.0000
Epoch 52/100
15/15 [=====] - 0s 11ms/step - loss: 11096868.0000 - val_loss: 11073613.0000
Epoch 53/100
15/15 [=====] - 0s 13ms/step - loss: 10928629.0000 - val_loss: 11000004.0000
Epoch 54/100
15/15 [=====] - 0s 17ms/step - loss: 10813028.0000 - val_loss: 10930053.0000
Epoch 55/100
15/15 [=====] - 0s 13ms/step - loss: 10707009.0000 - val_loss: 10871797.0000
Epoch 56/100
15/15 [=====] - 0s 18ms/step - loss: 10609973.0000 - val_loss: 10821061.0000
```

Epoch 57/100
15/15 [=====] - 0s 20ms/step - loss: 10530700.0000 - val_loss: 10777332.0000
Epoch 58/100
15/15 [=====] - 0s 16ms/step - loss: 10457838.0000 - val_loss: 10735352.0000
Epoch 59/100
15/15 [=====] - 0s 14ms/step - loss: 10400816.0000 - val_loss: 10689925.0000
Epoch 60/100
15/15 [=====] - 0s 15ms/step - loss: 10346658.0000 - val_loss: 10657234.0000
Epoch 61/100
15/15 [=====] - 0s 11ms/step - loss: 10282875.0000 - val_loss: 10623063.0000
Epoch 62/100
15/15 [=====] - 0s 25ms/step - loss: 10235319.0000 - val_loss: 10587079.0000
Epoch 63/100
15/15 [=====] - 0s 13ms/step - loss: 10189491.0000 - val_loss: 10551326.0000
Epoch 64/100
15/15 [=====] - 0s 17ms/step - loss: 10154383.0000 - val_loss: 10526212.0000
Epoch 65/100
15/15 [=====] - 0s 15ms/step - loss: 10112617.0000 - val_loss: 10492139.0000
Epoch 66/100
15/15 [=====] - 0s 17ms/step - loss: 10080184.0000 - val_loss: 10457963.0000
Epoch 67/100
15/15 [=====] - 0s 16ms/step - loss: 10039952.0000 - val_loss: 10435295.0000
Epoch 68/100
15/15 [=====] - 0s 13ms/step - loss: 10017027.0000 - val_loss: 10404831.0000
Epoch 69/100
15/15 [=====] - 0s 15ms/step - loss: 9980442.0000 - val_loss: 10388354.0000
Epoch 70/100
15/15 [=====] - 0s 15ms/step - loss: 9945839.0000 - val_loss: 10360499.0000
Epoch 71/100
15/15 [=====] - 0s 11ms/step - loss: 9916818.0000 - val_loss: 10335652.0000
Epoch 72/100
15/15 [=====] - 0s 18ms/step - loss: 9901141.0000 - val_loss: 10304378.0000
Epoch 73/100
15/15 [=====] - 0s 16ms/step - loss: 9867973.0000 - val_loss: 10287594.0000
Epoch 74/100
15/15 [=====] - 0s 13ms/step - loss: 9841622.0000 - val_loss: 10258233.0000
Epoch 75/100
15/15 [=====] - 0s 17ms/step - loss: 9816768.0000 - val_loss:

```
s: 10237276.0000
Epoch 76/100
15/15 [=====] - 0s 15ms/step - loss: 9794244.0000 - val_los
s: 10225350.0000
Epoch 77/100
15/15 [=====] - 0s 21ms/step - loss: 9777015.0000 - val_los
s: 10197803.0000
Epoch 78/100
15/15 [=====] - 0s 15ms/step - loss: 9777507.0000 - val_los
s: 10185007.0000
Epoch 79/100
15/15 [=====] - 0s 16ms/step - loss: 9740518.0000 - val_los
s: 10161303.0000
Epoch 80/100
15/15 [=====] - 0s 17ms/step - loss: 9719660.0000 - val_los
s: 10142402.0000
Epoch 81/100
15/15 [=====] - 0s 11ms/step - loss: 9698570.0000 - val_los
s: 10124388.0000
Epoch 82/100
15/15 [=====] - 0s 16ms/step - loss: 9693112.0000 - val_los
s: 10102788.0000
Epoch 83/100
15/15 [=====] - 0s 12ms/step - loss: 9672004.0000 - val_los
s: 10095430.0000
Epoch 84/100
15/15 [=====] - 0s 14ms/step - loss: 9648141.0000 - val_los
s: 10079321.0000
Epoch 85/100
15/15 [=====] - 0s 15ms/step - loss: 9633684.0000 - val_los
s: 10068329.0000
Epoch 86/100
15/15 [=====] - 0s 13ms/step - loss: 9618580.0000 - val_los
s: 10043317.0000
Epoch 87/100
15/15 [=====] - 0s 18ms/step - loss: 9604521.0000 - val_los
s: 10030980.0000
Epoch 88/100
15/15 [=====] - 0s 15ms/step - loss: 9589694.0000 - val_los
s: 10012778.0000
Epoch 89/100
15/15 [=====] - 0s 24ms/step - loss: 9575598.0000 - val_los
s: 9997758.0000
Epoch 90/100
15/15 [=====] - 0s 10ms/step - loss: 9565163.0000 - val_los
s: 9987144.0000
Epoch 91/100
15/15 [=====] - 0s 16ms/step - loss: 9554814.0000 - val_los
s: 9981221.0000
Epoch 92/100
15/15 [=====] - 0s 13ms/step - loss: 9541278.0000 - val_los
s: 9969394.0000
Epoch 93/100
15/15 [=====] - 0s 17ms/step - loss: 9531109.0000 - val_los
s: 9951222.0000
Epoch 94/100
```

```

15/15 [=====] - 0s 18ms/step - loss: 9521163.0000 - val_loss: 9933144.0000
Epoch 95/100
15/15 [=====] - 0s 13ms/step - loss: 9519415.0000 - val_loss: 9930287.0000
Epoch 96/100
15/15 [=====] - 0s 15ms/step - loss: 9505472.0000 - val_loss: 9926337.0000
Epoch 97/100
15/15 [=====] - 0s 16ms/step - loss: 9491713.0000 - val_loss: 9914297.0000
Epoch 98/100
15/15 [=====] - 0s 15ms/step - loss: 9488354.0000 - val_loss: 9904965.0000
Epoch 99/100
15/15 [=====] - 0s 18ms/step - loss: 9480409.0000 - val_loss: 9878209.0000
Epoch 100/100
15/15 [=====] - 0s 18ms/step - loss: 9462691.0000 - val_loss: 9869380.0000
5/5 [=====] - 0s 7ms/step - loss: 3903322.0000
Error Cuadrático Medio en el conjunto de prueba: 3903322.00
5/5 [=====] - 0s 4ms/step
Predicción: 6649.37, Valor Real: 5115.00
Predicción: 1071.24, Valor Real: 505.84
Predicción: 1071.24, Valor Real: 2350.00
Predicción: 1071.24, Valor Real: 572.60
Predicción: 1071.24, Valor Real: 920.00
Predicción: 3229.84, Valor Real: 3300.00
Predicción: 8299.86, Valor Real: 6400.00
Predicción: 1071.24, Valor Real: 965.00
Predicción: 1071.24, Valor Real: 357.60
Predicción: 16406.89, Valor Real: 12200.00
Error Cuadrático Medio (MSE): 3903322.02
Error Absoluto Medio (MAE): 1350.96
Coeficiente de Determinación (R^2): 0.76

```

Este modelo tiene errores todavía menores a los anteriores y una R^2 de 0.76, por lo que el 76% del precio por metro cuadrado es explicado por el modelo. Este presenta mejores resultados que los anteriores.

Modelo 5

Para este último modelo utilizamos algunas variables que consideramos mejor para determinar el precio de una casa, como los metros cuadrados construidos, número de baños, reácamaras y lugares de estacionamiento. Observamos los siguientes resultados.

```

In [ ]: df = inmu

x = df[["m2_construido", "Baños", "Recamaras", "Lugares_estac"]]
y = df["Precio_m2"]

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

```

```

# Escalar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Definir la arquitectura de la red neuronal para regresión
model = Sequential()
model.add(Dense(units=35, activation='relu', input_dim=4))
model.add(Dense(units=35))

model.add(Dense(units=1, activation='linear'))

# Compilar el modelo para regresión
model.compile(optimizer='adam', loss='mean_squared_error')

# Entrenar el modelo para regresión
model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_split=0.1)

# Evaluar el modelo en el conjunto de prueba
loss = model.evaluate(X_test_scaled, y_test)
print(f'Error Cuadrático Medio en el conjunto de prueba: {loss:.2f}')

# Realizar predicciones en el conjunto de prueba
predictions = model.predict(X_test_scaled)

# Visualizar algunas predicciones
for i in range(10):
    print(f'Predicción: {predictions[i][0]:.2f}, Valor Real: {y_test.iloc[i]:.2f}')

# Calcular métricas
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

# Mostrar las métricas
print(f'Error Cuadrático Medio (MSE): {mse:.2f}')
print(f'Error Absoluto Medio (MAE): {mae:.2f}')
print(f'Coeficiente de Determinación (R^2): {r2:.2f}')

```

Epoch 1/100
15/15 [=====] - 3s 44ms/step - loss: 44941264.0000 - val_loss: 37591772.0000
Epoch 2/100
15/15 [=====] - 0s 12ms/step - loss: 44934176.0000 - val_loss: 37585532.0000
Epoch 3/100
15/15 [=====] - 0s 13ms/step - loss: 44926352.0000 - val_loss: 37577768.0000
Epoch 4/100
15/15 [=====] - 0s 11ms/step - loss: 44916060.0000 - val_loss: 37566956.0000
Epoch 5/100
15/15 [=====] - 0s 12ms/step - loss: 44900720.0000 - val_loss: 37551464.0000
Epoch 6/100
15/15 [=====] - 0s 13ms/step - loss: 44878612.0000 - val_loss: 37527720.0000
Epoch 7/100
15/15 [=====] - 0s 12ms/step - loss: 44842524.0000 - val_loss: 37493560.0000
Epoch 8/100
15/15 [=====] - 0s 12ms/step - loss: 44792844.0000 - val_loss: 37442256.0000
Epoch 9/100
15/15 [=====] - 0s 11ms/step - loss: 44720676.0000 - val_loss: 37368768.0000
Epoch 10/100
15/15 [=====] - 0s 12ms/step - loss: 44619180.0000 - val_loss: 37270080.0000
Epoch 11/100
15/15 [=====] - 0s 11ms/step - loss: 44474972.0000 - val_loss: 37142032.0000
Epoch 12/100
15/15 [=====] - 0s 17ms/step - loss: 44309916.0000 - val_loss: 36974132.0000
Epoch 13/100
15/15 [=====] - 0s 17ms/step - loss: 44067892.0000 - val_loss: 36785508.0000
Epoch 14/100
15/15 [=====] - 0s 15ms/step - loss: 43807960.0000 - val_loss: 36542352.0000
Epoch 15/100
15/15 [=====] - 0s 14ms/step - loss: 43477164.0000 - val_loss: 36249360.0000
Epoch 16/100
15/15 [=====] - 0s 24ms/step - loss: 43078432.0000 - val_loss: 35909172.0000
Epoch 17/100
15/15 [=====] - 0s 14ms/step - loss: 42634652.0000 - val_loss: 35504148.0000
Epoch 18/100
15/15 [=====] - 0s 16ms/step - loss: 42107656.0000 - val_loss: 35050528.0000
Epoch 19/100
15/15 [=====] - 0s 15ms/step - loss: 41538956.0000 - val_loss:

```
ss: 34529000.0000
Epoch 20/100
15/15 [=====] - 0s 17ms/step - loss: 40855492.0000 - val_lo
ss: 33967468.0000
Epoch 21/100
15/15 [=====] - 0s 16ms/step - loss: 40115828.0000 - val_lo
ss: 33343846.0000
Epoch 22/100
15/15 [=====] - 0s 16ms/step - loss: 39325448.0000 - val_lo
ss: 32661846.0000
Epoch 23/100
15/15 [=====] - 0s 15ms/step - loss: 38436104.0000 - val_lo
ss: 31938290.0000
Epoch 24/100
15/15 [=====] - 0s 11ms/step - loss: 37497068.0000 - val_lo
ss: 31145328.0000
Epoch 25/100
15/15 [=====] - 0s 17ms/step - loss: 36452908.0000 - val_lo
ss: 30323324.0000
Epoch 26/100
15/15 [=====] - 0s 17ms/step - loss: 35419552.0000 - val_lo
ss: 29379526.0000
Epoch 27/100
15/15 [=====] - 0s 16ms/step - loss: 34236272.0000 - val_lo
ss: 28434832.0000
Epoch 28/100
15/15 [=====] - 0s 13ms/step - loss: 33037488.0000 - val_lo
ss: 27443900.0000
Epoch 29/100
15/15 [=====] - 0s 14ms/step - loss: 31793604.0000 - val_lo
ss: 26412196.0000
Epoch 30/100
15/15 [=====] - 0s 21ms/step - loss: 30491022.0000 - val_lo
ss: 25342406.0000
Epoch 31/100
15/15 [=====] - 0s 13ms/step - loss: 29118984.0000 - val_lo
ss: 24280544.0000
Epoch 32/100
15/15 [=====] - 0s 13ms/step - loss: 27821186.0000 - val_lo
ss: 23165972.0000
Epoch 33/100
15/15 [=====] - 0s 18ms/step - loss: 26413850.0000 - val_lo
ss: 22110974.0000
Epoch 34/100
15/15 [=====] - 0s 15ms/step - loss: 25163826.0000 - val_lo
ss: 20975522.0000
Epoch 35/100
15/15 [=====] - 0s 15ms/step - loss: 23734002.0000 - val_lo
ss: 19976918.0000
Epoch 36/100
15/15 [=====] - 0s 15ms/step - loss: 22485916.0000 - val_lo
ss: 18962172.0000
Epoch 37/100
15/15 [=====] - 0s 14ms/step - loss: 21236014.0000 - val_lo
ss: 17956588.0000
Epoch 38/100
```

```
15/15 [=====] - 0s 14ms/step - loss: 19987896.0000 - val_loss: 17013536.0000
Epoch 39/100
15/15 [=====] - 0s 18ms/step - loss: 18819384.0000 - val_loss: 16099384.0000
Epoch 40/100
15/15 [=====] - 0s 11ms/step - loss: 17726982.0000 - val_loss: 15200374.0000
Epoch 41/100
15/15 [=====] - 0s 18ms/step - loss: 16646968.0000 - val_loss: 14414765.0000
Epoch 42/100
15/15 [=====] - 0s 15ms/step - loss: 15718499.0000 - val_loss: 13647990.0000
Epoch 43/100
15/15 [=====] - 0s 14ms/step - loss: 14792150.0000 - val_loss: 12983766.0000
Epoch 44/100
15/15 [=====] - 0s 12ms/step - loss: 13978218.0000 - val_loss: 12364933.0000
Epoch 45/100
15/15 [=====] - 0s 20ms/step - loss: 13213832.0000 - val_loss: 11825307.0000
Epoch 46/100
15/15 [=====] - 0s 14ms/step - loss: 12562824.0000 - val_loss: 11314432.0000
Epoch 47/100
15/15 [=====] - 0s 15ms/step - loss: 11965721.0000 - val_loss: 10866495.0000
Epoch 48/100
15/15 [=====] - 0s 12ms/step - loss: 11452840.0000 - val_loss: 10466971.0000
Epoch 49/100
15/15 [=====] - 0s 16ms/step - loss: 10962098.0000 - val_loss: 10141907.0000
Epoch 50/100
15/15 [=====] - 0s 15ms/step - loss: 10608043.0000 - val_loss: 9824990.0000
Epoch 51/100
15/15 [=====] - 0s 17ms/step - loss: 10236949.0000 - val_loss: 9582392.0000
Epoch 52/100
15/15 [=====] - 0s 14ms/step - loss: 9929740.0000 - val_loss: 9386524.0000
Epoch 53/100
15/15 [=====] - 0s 16ms/step - loss: 9673855.0000 - val_loss: 9219550.0000
Epoch 54/100
15/15 [=====] - 0s 18ms/step - loss: 9483964.0000 - val_loss: 9038175.0000
Epoch 55/100
15/15 [=====] - 0s 12ms/step - loss: 9278605.0000 - val_loss: 8904473.0000
Epoch 56/100
15/15 [=====] - 0s 18ms/step - loss: 9139490.0000 - val_loss: 8784659.0000
```

Epoch 57/100
15/15 [=====] - 0s 14ms/step - loss: 9007929.0000 - val_loss:
s: 8684810.0000
Epoch 58/100
15/15 [=====] - 0s 17ms/step - loss: 8890205.0000 - val_loss:
s: 8596317.0000
Epoch 59/100
15/15 [=====] - 0s 23ms/step - loss: 8794384.0000 - val_loss:
s: 8520405.0000
Epoch 60/100
15/15 [=====] - 0s 14ms/step - loss: 8710169.0000 - val_loss:
s: 8458547.0000
Epoch 61/100
15/15 [=====] - 0s 15ms/step - loss: 8639484.0000 - val_loss:
s: 8400298.0000
Epoch 62/100
15/15 [=====] - 0s 16ms/step - loss: 8572827.0000 - val_loss:
s: 8349225.0000
Epoch 63/100
15/15 [=====] - 0s 17ms/step - loss: 8516983.0000 - val_loss:
s: 8293030.0000
Epoch 64/100
15/15 [=====] - 0s 14ms/step - loss: 8471486.0000 - val_loss:
s: 8244769.0000
Epoch 65/100
15/15 [=====] - 0s 12ms/step - loss: 8414273.0000 - val_loss:
s: 8204026.0000
Epoch 66/100
15/15 [=====] - 0s 19ms/step - loss: 8370575.0000 - val_loss:
s: 8160431.0000
Epoch 67/100
15/15 [=====] - 0s 13ms/step - loss: 8332552.5000 - val_loss:
s: 8125835.0000
Epoch 68/100
15/15 [=====] - 0s 14ms/step - loss: 8298865.5000 - val_loss:
s: 8085986.0000
Epoch 69/100
15/15 [=====] - 0s 18ms/step - loss: 8254772.0000 - val_loss:
s: 8046321.5000
Epoch 70/100
15/15 [=====] - 0s 15ms/step - loss: 8214858.5000 - val_loss:
s: 8011840.0000
Epoch 71/100
15/15 [=====] - 0s 14ms/step - loss: 8178952.0000 - val_loss:
s: 7977877.5000
Epoch 72/100
15/15 [=====] - 0s 24ms/step - loss: 8149871.0000 - val_loss:
s: 7948297.0000
Epoch 73/100
15/15 [=====] - 0s 17ms/step - loss: 8112236.0000 - val_loss:
s: 7916840.0000
Epoch 74/100
15/15 [=====] - 0s 12ms/step - loss: 8086279.5000 - val_loss:
s: 7886975.0000
Epoch 75/100
15/15 [=====] - 0s 15ms/step - loss: 8053817.5000 - val_loss:

s: 7854820.0000
Epoch 76/100
15/15 [=====] - 0s 12ms/step - loss: 8027787.5000 - val_los
s: 7823728.5000
Epoch 77/100
15/15 [=====] - 0s 14ms/step - loss: 7996765.5000 - val_los
s: 7796313.0000
Epoch 78/100
15/15 [=====] - 0s 14ms/step - loss: 7969770.5000 - val_los
s: 7768556.5000
Epoch 79/100
15/15 [=====] - 0s 20ms/step - loss: 7952247.0000 - val_los
s: 7744204.0000
Epoch 80/100
15/15 [=====] - 0s 17ms/step - loss: 7917653.0000 - val_los
s: 7714312.5000
Epoch 81/100
15/15 [=====] - 0s 18ms/step - loss: 7895210.5000 - val_los
s: 7684025.0000
Epoch 82/100
15/15 [=====] - 0s 16ms/step - loss: 7868697.5000 - val_los
s: 7659535.0000
Epoch 83/100
15/15 [=====] - 0s 20ms/step - loss: 7845041.5000 - val_los
s: 7631484.0000
Epoch 84/100
15/15 [=====] - 0s 12ms/step - loss: 7818999.5000 - val_los
s: 7610652.0000
Epoch 85/100
15/15 [=====] - 0s 16ms/step - loss: 7804326.0000 - val_los
s: 7586057.5000
Epoch 86/100
15/15 [=====] - 0s 22ms/step - loss: 7776127.5000 - val_los
s: 7569628.0000
Epoch 87/100
15/15 [=====] - 0s 13ms/step - loss: 7765804.5000 - val_los
s: 7541817.0000
Epoch 88/100
15/15 [=====] - 0s 13ms/step - loss: 7734023.5000 - val_los
s: 7520861.5000
Epoch 89/100
15/15 [=====] - 0s 16ms/step - loss: 7721086.0000 - val_los
s: 7496393.5000
Epoch 90/100
15/15 [=====] - 0s 14ms/step - loss: 7708090.5000 - val_los
s: 7472303.0000
Epoch 91/100
15/15 [=====] - 0s 14ms/step - loss: 7683439.0000 - val_los
s: 7455098.5000
Epoch 92/100
15/15 [=====] - 0s 17ms/step - loss: 7663751.5000 - val_los
s: 7434519.5000
Epoch 93/100
15/15 [=====] - 0s 14ms/step - loss: 7647729.5000 - val_los
s: 7410121.5000
Epoch 94/100

```

15/15 [=====] - 0s 14ms/step - loss: 7629638.0000 - val_loss
s: 7388962.0000
Epoch 95/100
15/15 [=====] - 0s 13ms/step - loss: 7625313.0000 - val_loss
s: 7364743.0000
Epoch 96/100
15/15 [=====] - 0s 14ms/step - loss: 7602986.5000 - val_loss
s: 7352383.5000
Epoch 97/100
15/15 [=====] - 0s 15ms/step - loss: 7587848.5000 - val_loss
s: 7331386.0000
Epoch 98/100
15/15 [=====] - 0s 13ms/step - loss: 7569185.5000 - val_loss
s: 7307651.5000
Epoch 99/100
15/15 [=====] - 0s 20ms/step - loss: 7550754.5000 - val_loss
s: 7290155.5000
Epoch 100/100
15/15 [=====] - 0s 13ms/step - loss: 7538056.5000 - val_loss
s: 7268489.0000
5/5 [=====] - 0s 9ms/step - loss: 3458916.7500
Error Cuadrático Medio en el conjunto de prueba: 3458916.75
5/5 [=====] - 0s 9ms/step
Predicción: 4283.47, Valor Real: 5115.00
Predicción: 1295.99, Valor Real: 505.84
Predicción: 1295.99, Valor Real: 2350.00
Predicción: 499.19, Valor Real: 572.60
Predicción: 1290.68, Valor Real: 920.00
Predicción: 4913.85, Valor Real: 3300.00
Predicción: 6861.63, Valor Real: 6400.00
Predicción: 1290.68, Valor Real: 965.00
Predicción: 1290.68, Valor Real: 357.60
Predicción: 16457.77, Valor Real: 12200.00
Error Cuadrático Medio (MSE): 3458916.81
Error Absoluto Medio (MAE): 1219.63
Coeficiente de Determinación (R^2): 0.79

```

Este tiene mucho mejores resultados que todos los anteriores, pues los errores son menores y presenta una R² de 79%, por lo que este será el modelo final para comparar con otras técnicas estadísticas presentadas con anterioridad.

Podemos observar en la siguiente gráfica cómo se comporta el modelo y vemos que los datos predecidos se acercan mucho a la linea de referencia.

```

In [ ]: # Visualizar los resultados de la predicción con la Línea de referencia
plt.scatter(y_test, predictions, label='Datos')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--', color='red',
plt.title('Predicciones vs. Valores Reales')
plt.xlabel('Valor Real')
plt.ylabel('Predicción')
plt.legend()
plt.show()

# Calcular estadísticas descriptivas de las predicciones
mean_prediction = np.mean(predictions)

```

```

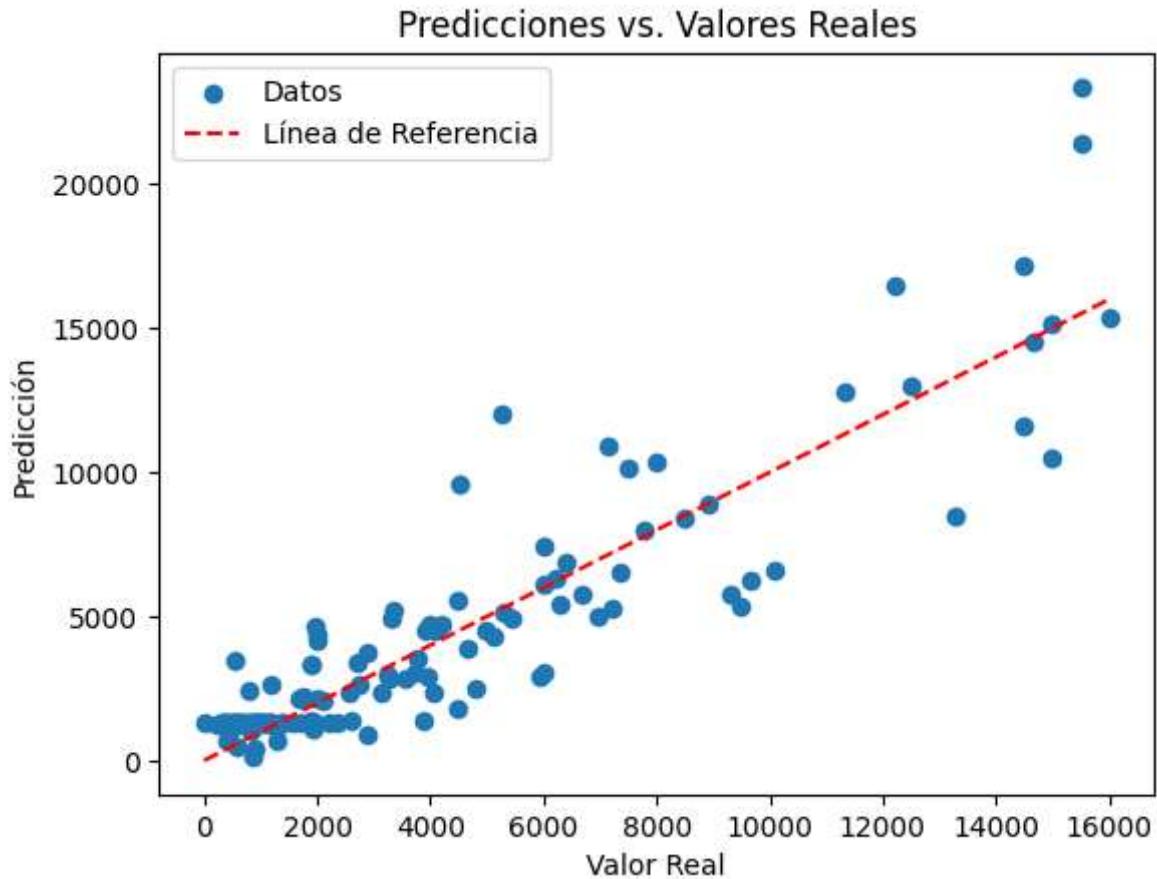
median_prediction = np.median(predictions)
std_dev_prediction = np.std(predictions)
min_prediction = np.min(predictions)
max_prediction = np.max(predictions)

mean_real = np.mean(y)
median_real = np.median(y)
std_dev_real = np.std(y)
min_real = np.min(y)
max_real = np.max(y)

# Mostrar las estadísticas
print(" ")
print(f'Media de las Predicciones: {mean_prediction:.2f}')
print(f'Mediana de las Predicciones: {median_prediction:.2f}')
print(f'Desviación Estándar de las Predicciones: {std_dev_prediction:.2f}')
print(f'Mínimo de las Predicciones: {min_prediction:.2f}')
print(f'Máximo de las Predicciones: {max_prediction:.2f}')

print(" ")
print(f'Media de datos reales: {mean_real:.2f}')
print(f'Mediana de datos reales: {median_real:.2f}')
print(f'Desviación Estándar de datos reales: {std_dev_real:.2f}')
print(f'Mínimo de datos reales: {min_real:.2f}')
print(f'Máximo de datos reales: {max_real:.2f}')

```



Media de las Predicciones: 4180.60
Mediana de las Predicciones: 2399.80
Desviación Estándar de las Predicciones: 4345.73
Mínimo de las Predicciones: 106.76
Máximo de las Predicciones: 23364.01

Media de datos reales: 4094.68
Mediana de datos reales: 1918.08
Desviación Estándar de datos reales: 5003.91
Mínimo de datos reales: 1.25
Máximo de datos reales: 47200.00

En cuanto a las estadísticas del modelo, observamos que se asemejan en algunos puntos, principalmente en la media pero aún así hay información que no es parecida en lo absoluto, como los valores mínimos y máximos.

Comparación de los tres modelos

Finalmente obtuvimos los siguientes resultados de los modelos seleccionados de cada técnica:

REGRESION MULTIPLE

Coeficiente de Determinación (R^2): 0.78

Error Absoluto Medio (MAE): 1502.21

Error Cuadrático Medio (MSE): 6308371.07

Predicciones

count 657.000000

mean 4094.677207

std 4331.195045

min -1801.755824

25% 702.043274

50% 2343.845438

75% 7073.761952

max 20888.522690

Estadísticas descriptivas de los valores reales:

count 657.000000

mean 4094.677207

std 5007.717845

min 1.250000

25% 860.657000

50% 1918.080000

75% 5524.000000

max 47200.000000

ARBOLES DE DECISION

R-squared promedio: 0.5876865471568692

Mean Squared Error: 3843461.1098892237

REDES NEURONALES

Error Cuadrático Medio (MSE): 3483201.91

Error Absoluto Medio (MAE): 1214.97

Coeficiente de Determinación (R^2): 0.79

Predicciones

Media de las Predicciones: 4132.02

Mediana de las Predicciones: 2391.04

Desviación Estándar de las Predicciones: 4399.41

Mínimo de las Predicciones: 339.94

Máximo de las Predicciones: 23405.99

Datos reales

Media de datos reales: 4094.68

Mediana de datos reales: 1918.08

Desviación Estándar de datos reales: 5003.91

Mínimo de datos reales: 1.25

Máximo de datos reales: 47200.00

En comparación con los tres modelos utilizados para predecir el precio por metro cuadrado de las propiedades inmobiliarias en la Ciudad de México, la regresión múltiple presenta un Coeficiente de Determinación (R^2) de 0.78, indicando que explica aproximadamente el 78% de la variabilidad en los datos. Además, tiene un Error Cuadrático Medio (MSE) de 6308371.07 y un Error Absoluto Medio (MAE) de 1502.21. Estas métricas sugieren que la regresión múltiple ofrece un buen ajuste al conjunto de datos, capturando la relación entre las variables predictoras y la variable objetivo.

En comparación, los árboles de decisión muestran un R-squared promedio de 0.59 y un Mean Squared Error de 3843461.11. Mientras que las redes neuronales presentan un Coeficiente de Determinación (R^2) de 0.79, un Error Cuadrático Medio (MSE) de 3483201.91 y un Error Absoluto Medio (MAE) de 1214.97.

Aunque las métricas son similares, la regresión múltiple y las redes neuronales destacan por su capacidad para explicar la variabilidad en los datos y minimizar los errores de predicción. La elección entre estos modelos dependerá de otros factores, como la interpretabilidad del modelo y la complejidad computacional. En este contexto, la regresión múltiple podría ser preferible si se valora la interpretación directa de los coeficientes, mientras que las redes neuronales podrían ser elegidas si se busca un modelo más flexible y capaz de manejar relaciones no lineales.

Sin embargo, debido a que el modelo de redes neuronales no ocupa tanta memoria ni tiempo de procesamiento y es más flexible que una regresión lineal, consideramos que este último modelo es el mejor y el que debería ser usado para predecir el valor de bienes inmuebles en la CDMX.

Conclusión

Elegir un modelo de predicción adecuado es crucial para el éxito de una empresa, ya que impacta directamente en la precisión de las decisiones basadas en datos. La selección cuidadosa de un modelo influye en la calidad de las predicciones, lo que a su vez impulsa la efectividad de las estrategias comerciales. Un modelo preciso no solo facilita la anticipación de tendencias y comportamientos del mercado, sino que también contribuye a la optimización de recursos y la toma de decisiones fundamentadas. En un entorno empresarial dinámico, donde la competitividad y la eficiencia son clave, contar con un modelo de predicción adecuado se convierte en un elemento estratégico para alcanzar el éxito y la sostenibilidad a largo plazo.

- BBVA. s.f. *¿Cómo saber el valor de una casa?*. BBVA. <https://www.bbva.mx/educacion-financiera/blog/como-saber-el-valor-de-una-casa.html#:~:text=Determinar%20el%20valor%20de%20una%20casa%20es%20clave,propie>

