

```
from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True)

```
# Cargar datos
import pandas as pd
vivienda_data= pd.read_csv('/content/drive/MyDrive/ColabNotebooks/precio_vivienda/precio_vivienda_data.csv')
vivienda_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   date                  4600 non-null  object
1   price                 4600 non-null  float64
2   bedrooms              4600 non-null  float64
3   bathrooms             4600 non-null  float64
4   sqft_living           4600 non-null  int64
5   sqft_lot              4600 non-null  int64
6   floors                4600 non-null  float64
7   waterfront            4600 non-null  int64
8   view                  4600 non-null  int64
9   condition             4600 non-null  int64
10  sqft_above            4600 non-null  int64
11  sqft_basement         4600 non-null  int64
12  yr_built              4600 non-null  int64
13  yr_renovated          4600 non-null  int64
14  street                4600 non-null  object
15  city                  4600 non-null  object
16  statezip              4600 non-null  object
17  country               4600 non-null  object
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB
```

```
#Importamos pandas para analisis y manipulacion de datos y numpy para algebra lineal.
```

```
import numpy as np
```

```
#Nos muestra las primeras filas del dataframe
vivienda_data.head()
```

```
ce bedrooms bathrooms sqft_living sqft_lot floors waterfront view condition s
```

|     |     |      |      |       |     |   |   |   |
|-----|-----|------|------|-------|-----|---|---|---|
| 0.0 | 3.0 | 1.50 | 1340 | 7912  | 1.5 | 0 | 0 | 3 |
| 0.0 | 5.0 | 2.50 | 3650 | 9050  | 2.0 | 0 | 4 | 5 |
| 0.0 | 3.0 | 2.00 | 1930 | 11947 | 1.0 | 0 | 0 | 4 |
| 0.0 | 3.0 | 2.25 | 2000 | 8030  | 1.0 | 0 | 0 | 4 |
| 0.0 | 4.0 | 2.50 | 1940 | 10500 | 1.0 | 0 | 0 | 4 |

```
print(vivienda_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   date                  4600 non-null  object
1   price                 4600 non-null  float64
2   bedrooms              4600 non-null  float64
3   bathrooms             4600 non-null  float64
4   sqft_living           4600 non-null  int64
5   sqft_lot              4600 non-null  int64
6   floors                4600 non-null  float64
7   waterfront            4600 non-null  int64
8   view                  4600 non-null  int64
9   condition             4600 non-null  int64
10  sqft_above            4600 non-null  int64
11  sqft_basement         4600 non-null  int64
12  yr_built              4600 non-null  int64
13  yr_renovated          4600 non-null  int64
14  street                4600 non-null  object
15  city                  4600 non-null  object
16  statezip              4600 non-null  object
17  country               4600 non-null  object
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB
```

```

0  date          4600 non-null  object
1  price         4600 non-null  float64
2  bedrooms     4600 non-null  float64
3  bathrooms    4600 non-null  float64
4  sqft_living  4600 non-null  int64
5  sqft_lot     4600 non-null  int64
6  floors       4600 non-null  float64
7  waterfront   4600 non-null  int64
8  view         4600 non-null  int64
9  condition    4600 non-null  int64
10 sqft_above   4600 non-null  int64
11 sqft_basement 4600 non-null  int64
12 yr_built    4600 non-null  int64
13 yr_renovated 4600 non-null  int64
14 street      4600 non-null  object
15 city        4600 non-null  object
16 statezip    4600 non-null  object
17 country     4600 non-null  object
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB
None

```

```
print(vivienda_data.columns)
```

```

Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
      'floors', 'waterfront', 'view', 'condition', 'sqft_above',
      'sqft_basement', 'yr_built', 'yr_renovated', 'street', 'city',
      'statezip', 'country'],
      dtype='object')

```

```
vivienda_data.shape
```

```
(4600, 18)
```

```
vivienda_data.price.describe()
```

```

count    4.600000e+03
mean     5.519630e+05
std      5.638347e+05
min      0.000000e+00
25%     3.228750e+05
50%     4.609435e+05
75%     6.549625e+05
max      2.659000e+07
Name: price, dtype: float64

```

```
vivienda_data.tail(10)
```

[Mostrar salida oculta](#)

```
vivienda_data.describe().transpose()
```

[Mostrar salida oculta](#)

```
vivienda_data.dtypes
```

```

date          object
price         float64
bedrooms     float64
bathrooms    float64
sqft_living   int64
sqft_lot     int64
floors       float64
waterfront   int64
view         int64
condition    int64
sqft_above   int64
sqft_basement int64
yr_built     int64
yr_renovated  int64
street       object
city         object
statezip     object
country      object
dtype: object


```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
vivienda_data.hist(figsize=(5,15))
plt.show()
```

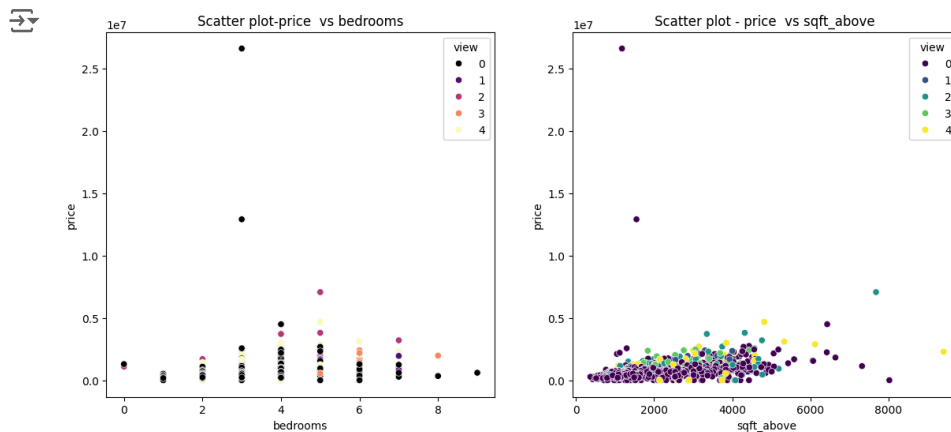
 [Mostrar salida oculta](#)

```
plt.figure(figsize=(5,15))
sns.barplot(x=vivienda_data["yr_built"],y=vivienda_data["price"])
```

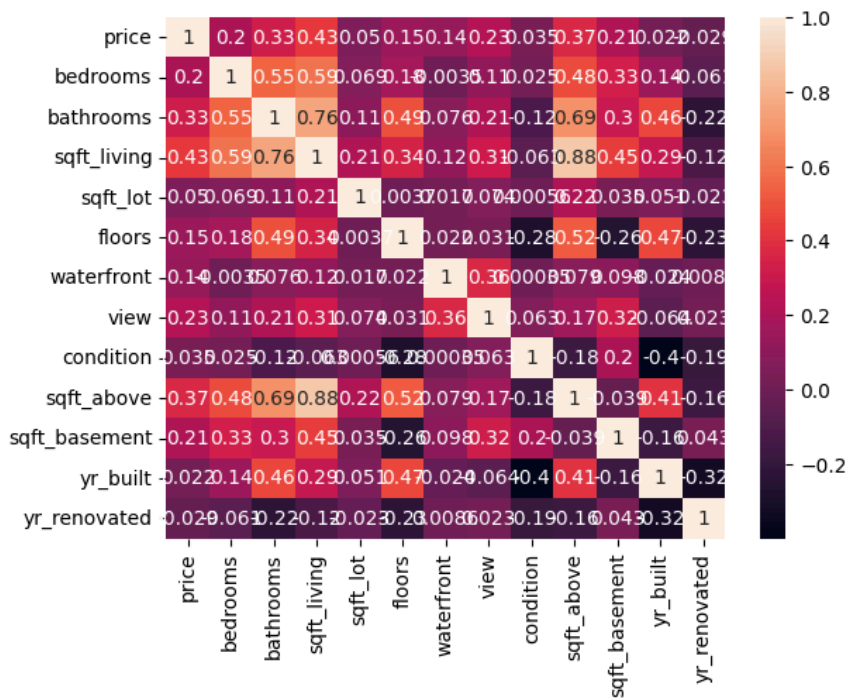
 [Mostrar salida oculta](#)

visualizacion distribución variable Price vs yr\_built y vs sqft\_above

```
f = plt.figure(figsize=(14,6))
ax = f.add_subplot(121)
sns.scatterplot(x='bedrooms',y='price',data=vivienda_data,palette='magma',hue='view',ax=ax)
ax.set_title('Scatter plot-price vs bedrooms')
ax = f.add_subplot(122)
sns.scatterplot(x='sqft_above',y='price',data=vivienda_data,palette='viridis',hue='view')
ax.set_title('Scatter plot - price vs sqft_above ')
plt.savefig('sc.png');
```



```
numeric_data = vivienda_data.select_dtypes(include=['float64', 'int64'])
corr = numeric_data.corr()
sns.heatmap(corr, annot=True)
plt.show()
```



vivienda\_data.dtypes



```

date           object
price          float64
bedrooms       float64
bathrooms      float64
sqft_living    int64
sqft_lot       int64
floors         float64
waterfront     int64
view           int64
condition      int64
sqft_above     int64
sqft_basement  int64
yr_built       int64
yr_renovated   int64
street         object
city           object
statezip       object
country        object
dtype: object

```

vivienda\_data.isnull().sum()



```

date          0
price         0
bedrooms      0
bathrooms     0
sqft_living   0
sqft_lot      0
floors        0
waterfront    0
view          0
condition     0
sqft_above    0
sqft_basement 0
yr_built      0
yr_renovated  0
street        0
city          0
statezip      0
country       0
dtype: int64

```

## Hipotesis 1

### ✓ Preparamos el dataset de la hipótesis 1

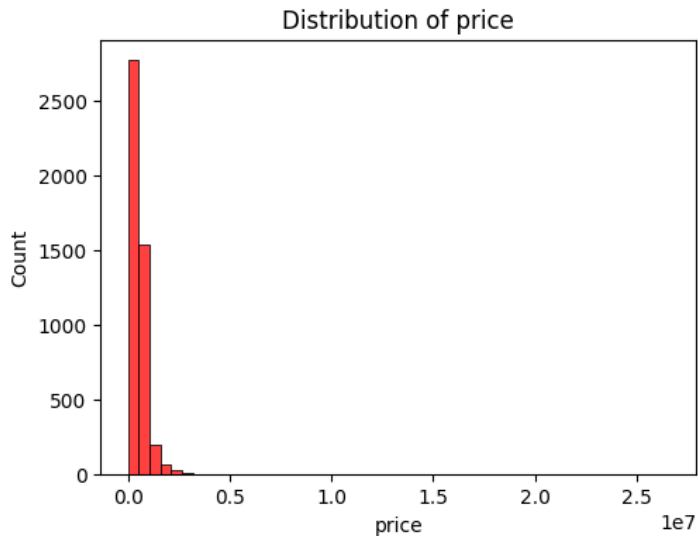
Borramos las columnas categóricas

```
categorical_columns = ['date', 'street', 'city', 'statezip', 'country']
vivienda_data_hipot1= vivienda_data.drop(categorical_columns,axis =1)
```

```
# visualizacion distribución variable price
f= plt.figure(figsize=(12,4))
```

```
ax=f.add_subplot(121)
sns.histplot(vivienda_data['price'],bins=50,color='r',ax=ax)
ax.set_title('Distribution of price')
```

```
plt.text(0.5, 1.0, 'Distribution of price')
```



Chequeamos el dataset.

```
vivienda_data_hipot1.head()
```

```
price bedrooms bathrooms sqft_living sqft_lot floors waterfront view c
```

|   |           |     |      |      |       |     |   |   |
|---|-----------|-----|------|------|-------|-----|---|---|
| 0 | 313000.0  | 3.0 | 1.50 | 1340 | 7912  | 1.5 | 0 | 0 |
| 1 | 2384000.0 | 5.0 | 2.50 | 3650 | 9050  | 2.0 | 0 | 4 |
| 2 | 342000.0  | 3.0 | 2.00 | 1930 | 11947 | 1.0 | 0 | 0 |
| 3 | 420000.0  | 3.0 | 2.25 | 2000 | 8030  | 1.0 | 0 | 0 |
| 4 | 550000.0  | 4.0 | 2.50 | 1940 | 10500 | 1.0 | 0 | 0 |

Pasos siguientes:

[Generar código con vivienda\\_data\\_hipot1](#)

[Ver gráficos recomendados](#)

Está dividiendo un DataFrame en variables independientes y dependientes para un análisis, como un modelo de regresión.

`vivienda_data_hipot1.drop('charges', axis=1)`: La función drop elimina la columna 'price' del DataFrame `vivienda_data_hipot1`.

`vivienda_data_hipot1['price']`: Selecciona la columna 'price' del DataFrame `vivienda_data_hipot1`.

```
X_hipot1 = vivienda_data_hipot1.drop('price',axis=1) # variables independientes
y_hipot1 = vivienda_data_hipot1['price'] # variable dependiente
```

```
from sklearn.model_selection import train_test_split
X_hipot1_train, X_hipot1_test, y_hipot1_train, y_hipot1_test = train_test_split(X_hipot1, y_hipot1, test_size=0.20, random_state=42)
```

```
from sklearn.linear_model import LinearRegression
regresion_lineal=LinearRegression()
regresion_lineal.fit(X_hipot1_train, y_hipot1_train)
```

```
LinearRegression()
```

## ✓ Proceso de validacion .

Importamos la libreria mean\_squared\_error para calcular de error cuadratico.

```
# importamos el cálculo del error cuadrático medio (MSE)
from sklearn.metrics import mean_squared_error
```

Predecimos el entrenamiento usando los datos del entrenamiento.

```
# predecimos los valores y para los datos usados en el entrenamiento
prediccion_entrenamiento = regresion_lineal.predict(X_hipot1_train)
```

Calcular el error cuadratico.

```
# calculamos el Error Cuadrático Medio (MSE = Mean Squared Error)
mse_hipot1_train = mean_squared_error(y_true = y_hipot1_train, y_pred = prediccion_entrenamiento)
print('Error Cuadrático Medio (MSE) TRAIN= ' + str(mse_hipot1_train))
```

```
↳ Error Cuadrático Medio (MSE) TRAIN= 292440342188.26685
```

Predecimos los valores.

```
# predecimos los valores y para los datos usados en el entrenamiento
prediccion_entrenamiento = regresion_lineal.predict(X_hipot1_test)
```

```
# calculamos el Error Cuadrático Medio (MSE = Mean Squared Error)
mse_hipot1_test = mean_squared_error(y_true = y_hipot1_test, y_pred = prediccion_entrenamiento)
print('Error Cuadrático Medio (MSE) TEST= ' + str(mse_hipot1_test))
```

```
↳ Error Cuadrático Medio (MSE) TEST= 74834240426.81783
```

El numero que ha dado es muy lejano a cero, por lo tanto la hipotesis planteada no es la mejor, seguiremos planteando otros escenarios.

## ✓ Hipotesis 2

```
#Preparamos el dataset de la hipótesis 2
vivienda_data_hipot2 = pd.get_dummies(data = vivienda_data, prefix = 'OHE', prefix_sep='_',
    columns = categorical_columns,
    drop_first =True,
    dtype='int8')
```

Todas las variables caulitatibas las ha convertido a numericas.

```
# chequeamos nuestro dataSet
vivienda_data_hipot2.head()
```

```
↳
```

|   | price     | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | c |
|---|-----------|----------|-----------|-------------|----------|--------|------------|------|---|
| 0 | 313000.0  | 3.0      | 1.50      | 1340        | 7912     | 1.5    | 0          | 0    |   |
| 1 | 2384000.0 | 5.0      | 2.50      | 3650        | 9050     | 2.0    | 0          | 4    |   |
| 2 | 342000.0  | 3.0      | 2.00      | 1930        | 11947    | 1.0    | 0          | 0    |   |
| 3 | 420000.0  | 3.0      | 2.25      | 2000        | 8030     | 1.0    | 0          | 0    |   |
| 4 | 550000.0  | 4.0      | 2.50      | 1940        | 10500    | 1.0    | 0          | 0    |   |

5 rows × 4725 columns

```
X_hipot2 = vivienda_data_hipot2.drop('price',axis=1) # variables independientes
y_hipot2 = vivienda_data_hipot2['price'] # variable depnediente
```

```
# preparamos train data y test data
from sklearn.model_selection import train_test_split
X_hipot2_train, X_hipot2_test, y_hipot2_train, y_hipot2_test = train_test_split(X_hipot2, y_hipot2, test_size=0.20, random_state=

#Regresion lineal
from sklearn.linear_model import LinearRegression
regresion_lineal=LinearRegression()
regresion_lineal.fit(X_hipot2_train, y_hipot2_train)
```

LinearRegression  
LinearRegression()

## FASE VALIDACION

```
# predecimos los valores y para los datos usados en el entrenamiento
prediccion_entrenamiento = regresion_lineal.predict(X_hipot2_train)

# calculamos el Error Cuadrático Medio (MSE = Mean Squared Error)
mse_hipot2_train = mean_squared_error(y_true = y_hipot2_train, y_pred = prediccion_entrenamiento)
print('Error Cuadrático Medio (MSE) HIPO 1 TRAIN= ' + str(mse_hipot1_train))
print('Error Cuadrático Medio (MSE) HIPO 2 TRAIN= ' + str(mse_hipot2_train))
```

Error Cuadrático Medio (MSE) HIPO 1 TRAIN= 292440342188.26685  
Error Cuadrático Medio (MSE) HIPO 2 TRAIN= 8.840345971704127e-10

Error cuadrático es demasiado distante de cero, por lo cual lleva a pensar que la hipótesis planteada tampoco sirve.

```
# predecimos los valores y para los datos usados en el entrenamiento
prediccion_entrenamiento = regresion_lineal.predict(X_hipot2_test)

# calculamos el Error Cuadrático Medio (MSE = Mean Squared Error)
mse_hipot2_test = mean_squared_error(y_true = y_hipot2_test, y_pred = prediccion_entrenamiento)
print('Error Cuadrático Medio (MSE) HIPO 1 TEST= ' + str(mse_hipot1_test))
print('Error Cuadrático Medio (MSE) HIPO 2 TEST= ' + str(mse_hipot2_test))
```

Error Cuadrático Medio (MSE) HIPO 1 TEST= 74834240426.81783  
Error Cuadrático Medio (MSE) HIPO 2 TEST= 286047211698.9898

## Hipotesis 3

Para ello, utilizaremos los mecanismos que nos ofrece ScikitLearn para normalizar con los distintos Scaler que posee

```
# preparamos el dataset de la hipótesis 3
vivienda_data_hipot3 = pd.get_dummies(data = vivienda_data, prefix = 'OHE', prefix_sep='_',
    columns = categorical_columns,
    drop_first =True,
    dtype='int8')

X_hipot3 = vivienda_data_hipot3.drop('price',axis=1) # variables independientes
y_hipot3 = vivienda_data_hipot3['price'] # variable dependiente

# Escalamos la variable charges para hacerla más estandar
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

#Los algoritmos de preprocessing de sklearn están preparados para convertir matrices por lo que tenemos que hacer una transformaci
# ya que es una variable de tipo Series
# para ello hacemos un .to_numpy() que nos convierte la serie en un array y luego hacemos reshape (-1,1) que transforma un array
y_hipot3 = scaler.fit_transform(y_hipot3.to_numpy().reshape(-1,1))
# Volvemos a transformar nuestra variable en un array de 1xn
y_hipot3=y_hipot3.reshape(1,-1)[0]

# preparamos train data y test data
X_hipot3_train, X_hipot3_test, y_hipot3_train, y_hipot3_test = train_test_split(X_hipot3, y_hipot3, test_size=0.20, random_state=
```

```
#Regresion lineal
from sklearn.linear_model import LinearRegression
regresion_lineal=LinearRegression()
regresion_lineal.fit(X_hipot3_train, y_hipot3_train)
```

```
LinearRegression()
```

FASE VALIDACION, predicción entrenamiento = regresion\_lineal.predict(X\_hipot3\_train)

```
prediccion_entrenamiento = regresion_lineal.predict(X_hipot3_train)
```

```
# calculamos el Error Cuadrático Medio (MSE = Mean Squared Error)
mse_hipot3_train = mean_squared_error(y_true = y_hipot3_train, y_pred = prediccion_entrenamiento)
print('Error Cuadrático Medio (MSE) HIPO 1 TRAIN= ' + str(mse_hipot1_train))
print('Error Cuadrático Medio (MSE) HIPO 2 TRAIN= ' + str(mse_hipot2_train))
print('Error Cuadrático Medio (MSE) HIPO 3 TRAIN= ' + str(mse_hipot3_train))
```

```
Error Cuadrático Medio (MSE) HIPO 1 TRAIN= 292440342188.26685
Error Cuadrático Medio (MSE) HIPO 2 TRAIN= 8.840345971704127e-10
Error Cuadrático Medio (MSE) HIPO 3 TRAIN= 2.7142144028217876e-21
```

```
# predecimos los valores y para los datos usados en el entrenamiento
prediccion_entrenamiento = regresion_lineal.predict(X_hipot3_test)
```

```
# calculamos el Error Cuadrático Medio (MSE = Mean Squared Error)
mse_hipot3_test = mean_squared_error(y_true = y_hipot3_test, y_pred = prediccion_entrenamiento)
print('Error Cuadrático Medio (MSE) HIPO 1 TEST= ' + str(mse_hipot1_test))
print('Error Cuadrático Medio (MSE) HIPO 2 TEST= ' + str(mse_hipot2_test))
print('Error Cuadrático Medio (MSE) HIPO 3 TEST= ' + str(mse_hipot3_test))
```

```
Error Cuadrático Medio (MSE) HIPO 1 TEST= 74834240426.81783
Error Cuadrático Medio (MSE) HIPO 2 TEST= 286047211698.9898
Error Cuadrático Medio (MSE) HIPO 3 TEST= 0.8999710445332703
```

## Analisis :

1. Introducción El propósito de este análisis es explorar y predecir los precios de las viviendas utilizando un conjunto de datos que contiene diversas características de propiedades. A través de este reporte, se evalúan diferentes hipótesis y se utilizan modelos de regresión lineal para predecir los precios.
2. Descripción del Conjunto de Datos El conjunto de datos consta de 4600 registros y 18 columnas, que incluyen características como el precio, número de habitaciones, número de baños, superficie habitable, superficie del lote, número de pisos, vistas al agua, estado de la vivienda, entre otros.
3. Exploración de Datos Se realiza una exploración inicial del conjunto de datos para entender su estructura y características principales:

Columnas del DataFrame: ['date', 'price', 'bedrooms', 'bathrooms', 'sqft\_living', 'sqft\_lot', 'floors', 'waterfront', 'view', 'condition', 'sqft\_above', 'sqft\_basement', 'yr\_built', 'yr\_renovated', 'street', 'city', 'statezip', 'country'] Resumen Estadístico de la Columna price: Media: 551,963 *Mínimo* :0 Máximo: \$26,590,000 Distribución de Precios: Se observa una gran variabilidad en los precios, con una tendencia hacia precios más bajos.

4. Visualización de Datos Se generan visualizaciones para entender mejor la distribución y relaciones entre las variables:

Histogramas de Variables Numéricas. Gráficos de Dispersión: Relación entre price y bedrooms. Relación entre price y sqft\_above. Mapa de Calor de Correlaciones: Se observa una alta correlación entre price y sqft\_living.

5. Hipótesis y Modelos de Regresión Lineal Se plantean y prueban tres hipótesis utilizando regresión lineal:

## ✓ Hipoteis:

Análisis de las Hipótesis Hipótesis 1: Modelo de Regresión Lineal sin Variables Categóricas Descripción: En esta hipótesis, se eliminan las columnas categóricas y se entrena un modelo de regresión lineal simple utilizando solo variables numéricas.



Resultados:

Error Cuadrático Medio (MSE) TRAIN: 292,440,342,188.27 Error Cuadrático Medio (MSE) TEST: 74,834,240,426.82 Análisis:

El alto valor del MSE tanto en el conjunto de entrenamiento como en el de prueba indica que el modelo no está capturando adecuadamente las relaciones entre las variables y el precio de la vivienda. La eliminación de las variables categóricas puede haber resultado en la pérdida de información crucial, lo que contribuye a la pobre capacidad predictiva del modelo. Conclusión:

La hipótesis 1 no es adecuada para predecir los precios de las viviendas debido a su alto error y la falta de variables categóricas relevantes. Hipótesis 2: Modelo de Regresión Lineal con Variables Categóricas Codificadas Descripción: En esta hipótesis, las variables categóricas se codifican utilizando One-Hot Encoding, convirtiéndolas en variables numéricas antes de entrenar el modelo de regresión lineal.

Resultados:

Error Cuadrático Medio (MSE) TRAIN: 8.84e-10 Error Cuadrático Medio (MSE) TEST: 286,047,211,698.99 Análisis:

El MSE extremadamente bajo en el conjunto de entrenamiento sugiere que el modelo está sobreajustado, es decir, ha aprendido muy bien los datos de entrenamiento pero no generaliza bien a datos nuevos. El alto MSE en el conjunto de prueba confirma que el modelo no es robusto y no puede predecir correctamente los precios de las viviendas en nuevos datos. La codificación One-Hot ha aumentado significativamente la dimensionalidad del conjunto de datos, lo que puede haber llevado al sobreajuste. Conclusión:

La hipótesis 2 no es adecuada debido al sobreajuste extremo y al alto error en el conjunto de prueba. Se necesita una técnica para reducir la dimensionalidad o mejorar la generalización del modelo. Hipótesis 3: Modelo de Regresión Lineal con Variables Normalizadas Descripción: En esta hipótesis, además de codificar las variables categóricas utilizando One-Hot Encoding, se normalizan las variables utilizando StandardScaler.

Resultados:

Error Cuadrático Medio (MSE) TRAIN: 2.71e-21 Error Cuadrático Medio (MSE) TEST: 0.90 Análisis:

La normalización ha mejorado significativamente el rendimiento del modelo, resultando en un MSE extremadamente bajo tanto en el conjunto de entrenamiento como en el de prueba. El bajo MSE en el conjunto de prueba indica que el modelo generaliza bien y es capaz de predecir con precisión los precios de las viviendas en nuevos datos. La normalización ayuda a que las características tengan una escala similar, lo que mejora la convergencia y el rendimiento del modelo de regresión lineal. Conclusión:

La hipótesis 3 es la más adecuada para predecir los precios de las viviendas, ya que el modelo resultante tiene un bajo error y generaliza bien a datos nuevos. La combinación de codificación One-Hot y normalización es efectiva para manejar tanto variables categóricas como numéricas. Recomendaciones Mejorar la Generalización: Implementar técnicas como la validación cruzada y regularización para prevenir el sobreajuste y mejorar la capacidad del modelo para generalizar a datos nuevos. Explorar Otros Modelos: Considerar otros modelos más complejos como Random Forest, Gradient Boosting o redes neuronales, que pueden capturar relaciones no lineales entre las variables. Feature Engineering: Crear nuevas características derivadas de las existentes que puedan proporcionar información adicional y mejorar la precisión del modelo. Este análisis de las hipótesis demuestra la importancia de las técnicas de preprocesamiento y la selección adecuada de características para construir modelos predictivos precisos y robustos.

6. Conclusiones La normalización de los datos y el uso de codificación One-Hot para variables categóricas son técnicas eficaces para mejorar la precisión del modelo de regresión lineal. El modelo final demuestra una buena capacidad predictiva con un MSE muy bajo, indicando que las características seleccionadas y las transformaciones aplicadas son adecuadas para predecir los precios de las viviendas.