Cargamos las paquterías que vamos a utilizar:

In [56]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from statsmodels.formula.api import ols
```

In [3]:
```
#!pip install openpyxl
#esta sólo se usa si no se ha instalado para leer archivos excel
```

Se carga el documento, el equipo decidió usar el archivo "10_percent_Florida", de la opción de precios de propiedades residenciales de la carpeta Zillow

In [7]:
```
datos = pd.read_excel('Datos/10_percent_Florida.xlsx')
datos.head(10)
```

Out[7]:

	listingDataSource	zpid	city	state	homeStatus	isListingClaimedByCurrentSignedInUser	isCurrentSignedInA
0	Phoenix	42977593	tamarac	FL	for_sale	False	
1	Phoenix	43151113	fort_lauderdale	FL	for_sale	False	
2	Phoenix	155833951	miami_beach	FL	for_sale	False	
3	Phoenix	44301933	miami	FL	for_sale	False	
4	Phoenix	103464576	west_palm_beach	FL	for_sale	False	
5	Phoenix	42907000	pompano_beach	FL	for_sale	False	
6	Phoenix	46765794	delray_beach	FL	for_sale	False	
7	Phoenix	43881508	miami_beach	FL	for_sale	False	
8	Phoenix	337200955	miami	FL	for_sale	False	
9	Phoenix	71734790	fort_lauderdale	FL	for_sale	False	

10 rows × 805 columns

Limpieza de datos

In [11]:
```
#con .info () vemos todos los valores nulos dentro de nuestra tabla
datos.info()
#.describe sirve para mostrar un resumen de un estadístico descriptivo de los datos contenidos en datos.
print(datos.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2806 entries, 0 to 2805
Columns: 805 entries, listingDataSource to BUYINFEEFound_In_Columns
dtypes: bool(109), float64(311), int64(13), object(372)
memory usage: 15.2+ MB
                zpid     bedrooms    bathrooms         price   yearBuilt  \
count   2.806000e+03  2714.000000  2720.000000  2.806000e+03  2725.00000
mean    2.197555e+08     3.079587     2.799816  1.726639e+06  1985.11156
std     5.058730e+08     1.484508     1.541059  4.355494e+06    22.50474
min     4.282073e+07     0.000000     0.000000  0.000000e+00  1889.00000
25%     4.386455e+07     2.000000     2.000000  3.850000e+05  1970.00000
50%     4.663425e+07     3.000000     2.000000  6.500000e+05  1984.00000
75%     7.173174e+07     4.000000     3.000000  1.270000e+06  2003.00000
max     2.141801e+09    30.000000    18.000000  8.890000e+07  2027.00000

            zipcode  propertyUpdatePageLink  moveHomeMapLocationLink  \
count   2806.000000                     0.0                      0.0
mean   33261.277263                     NaN                      NaN
std      183.598621                     NaN                      NaN
min    33001.000000                     NaN                      NaN
25%    33129.000000                     NaN                      NaN
50%    33301.000000                     NaN                      NaN
75%    33426.000000                     NaN                      NaN
max    34145.000000                     NaN                      NaN

       propertyEventLogLink  editPropertyHistorylink  ...  \
count                   0.0                      0.0  ...
mean                    NaN                      NaN  ...
std                     NaN                      NaN  ...
min                     NaN                      NaN  ...
25%                     NaN                      NaN  ...
50%                     NaN                      NaN  ...
75%                     NaN                      NaN  ...
max                     NaN                      NaN  ...

       price_to_rent_ratio_InfoTOD  \
count                  1754.000000
mean                    183.923629
std                     205.839690
min                       9.090909
25%                     123.378471
50%                     155.755187
```

```
75%                     187.612141
max                    5190.700855

       contactFormRenderData.data.region_phone_number  \
count                                             0.0
mean                                              NaN
std                                               NaN
min                                               NaN
25%                                               NaN
50%                                               NaN
75%                                               NaN
max                                               NaN

       contactFormRenderData.data.subtitle  adTargets  \
count                                  0.0        0.0
mean                                   NaN        NaN
std                                    NaN        NaN
min                                    NaN        NaN
25%                                    NaN        NaN
50%                                    NaN        NaN
75%                                    NaN        NaN
max                                    NaN        NaN

       contactFormRenderData.data.footers     ADULTCOM   ANNUALDUES  \
count                                 0.0  2806.000000  2806.000000
mean                                  NaN     0.008197     0.000713
std                                   NaN     0.090180     0.026693
min                                   NaN     0.000000     0.000000
25%                                   NaN     0.000000     0.000000
50%                                   NaN     0.000000     0.000000
75%                                   NaN     0.000000     0.000000
max                                   NaN     1.000000     1.000000

       WAITINGPERIOD  BUYINFEE  BUYINFEEFound_In_Columns
count    2806.000000    2806.0                       0.0
mean        0.002851       0.0                       NaN
std         0.053328       0.0                       NaN
min         0.000000       0.0                       NaN
25%         0.000000       0.0                       NaN
50%         0.000000       0.0                       NaN
75%         0.000000       0.0                       NaN
max         1.000000       0.0                       NaN
```

```
[8 rows x 324 columns]
```

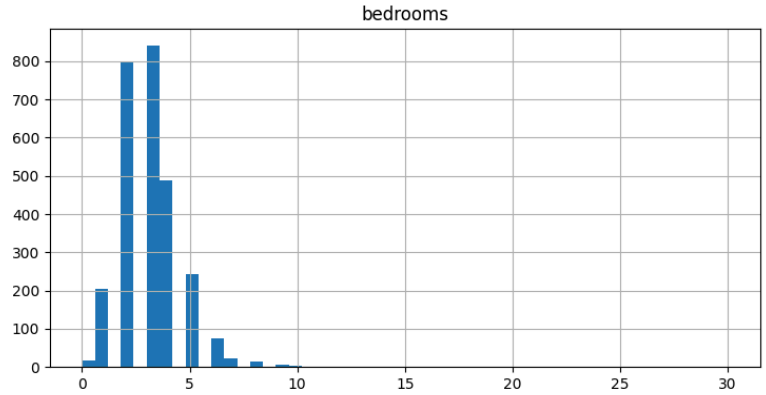
In [13]:
```
#nos muestra las columnas
datos.columns
```

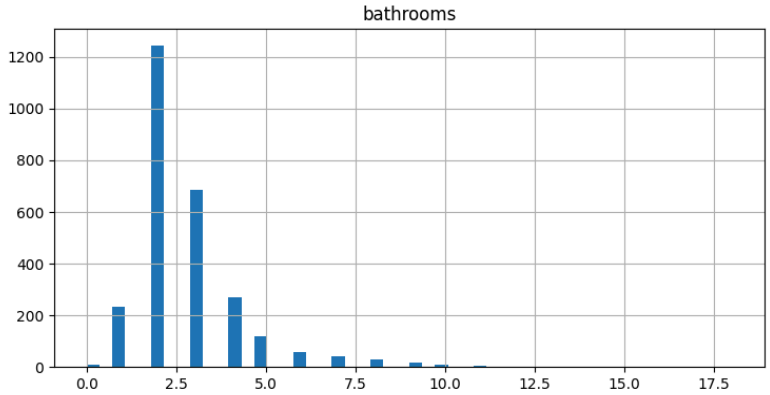
Out[13]:
```
Index(['listingDataSource', 'zpid', 'city', 'state', 'homeStatus',
       'isListingClaimedByCurrentSignedInUser',
       'isCurrentSignedInAgentResponsible', 'bedrooms', 'bathrooms', 'price',
       ...
       'image_urls', 'source_file', 'ADULTCOM', 'ADULTCOMFound_In_Columns',
       'ANNUALDUES', 'ANNUALDUESFound_In_Columns', 'WAITINGPERIOD',
       'WAITINGPERIODFound_In_Columns', 'BUYINFEE',
       'BUYINFEEFound_In_Columns'],
      dtype='object', length=805)
```

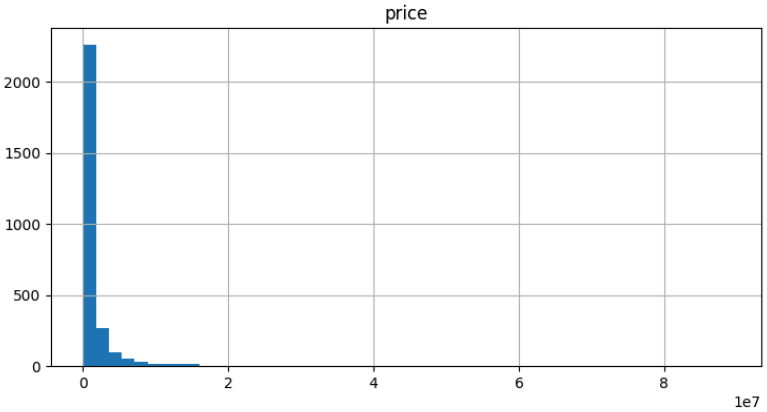
Con estos visuales podemos ver los datos a analizar

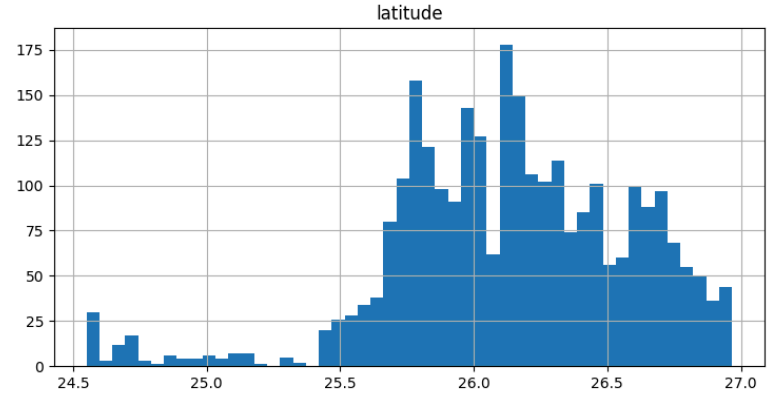
Ahora haremos un histograma, sólo contemplaremos los datos que nos parezcan relevantes de la tabla

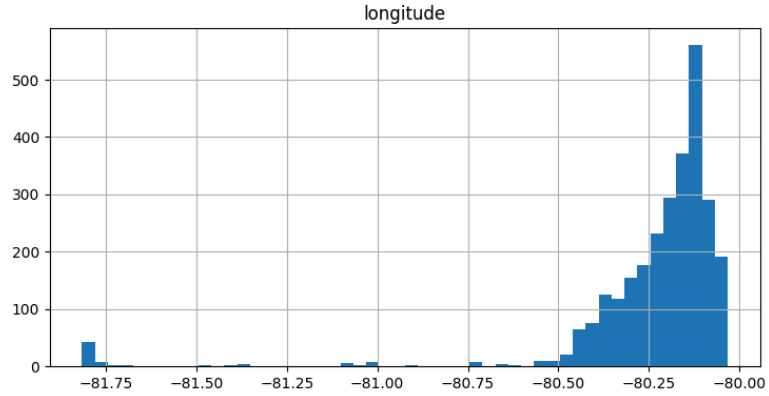
In [17]:
```
datos[['bedrooms', 'bathrooms', 'price', 'latitude', 'longitude']].hist(bins = 50, figsize = (20, 15))
plt.show()
```









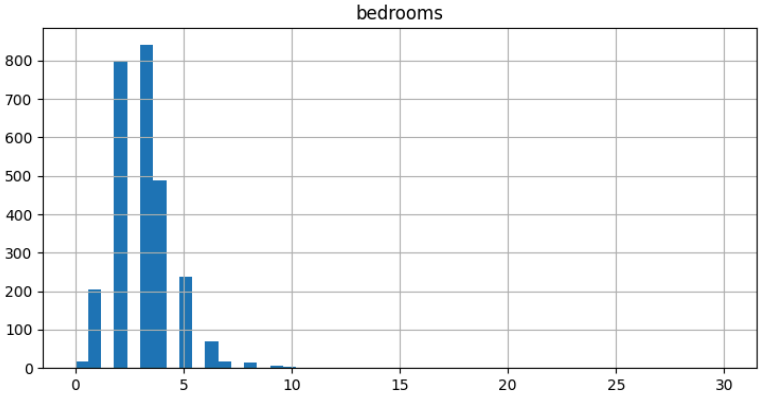


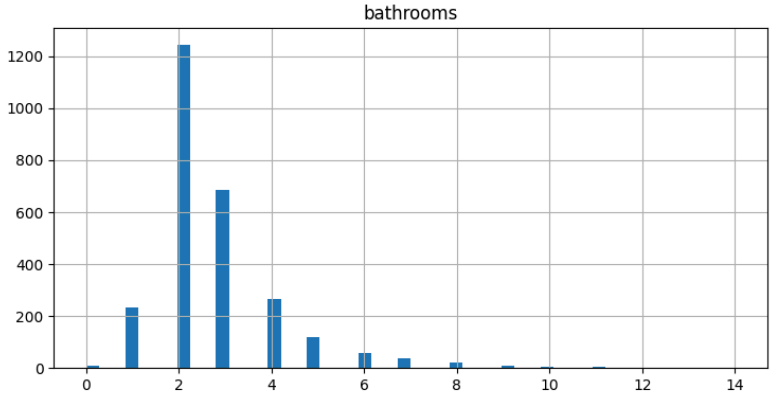
Las variables que no se vieron representadas en los gráficos fueron: 'city', 'state', 'homeStatus', esto es debido a que son textos para ello debemos trasformarlas con ayuda de las dummies con ayuda del comando pd.get_dummies.

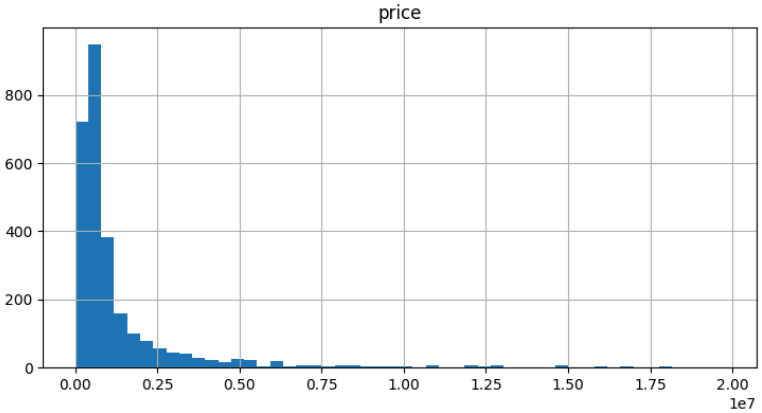
Filtro

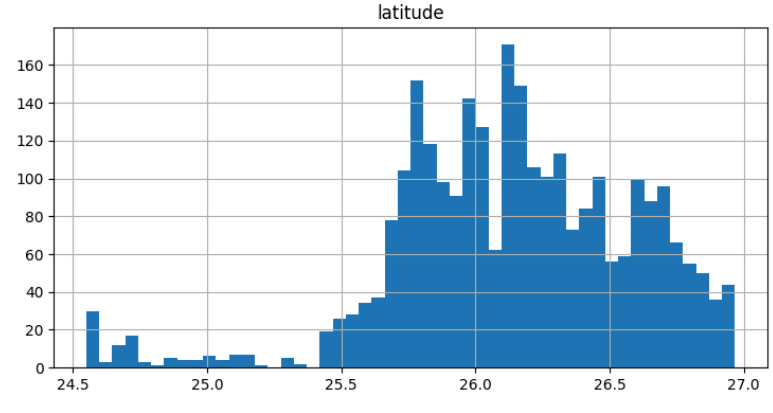
Hacemos un filtro de los datos basándonos en el histograma anterior, en este vamos a contemplar los inmuebles que esten arriba de 1000 dólares, pero por debajo de 20000000 de dólares, no contemplaremos datos más grandes a estos puesto que son pocos y causan ruido inecesario en el modelo

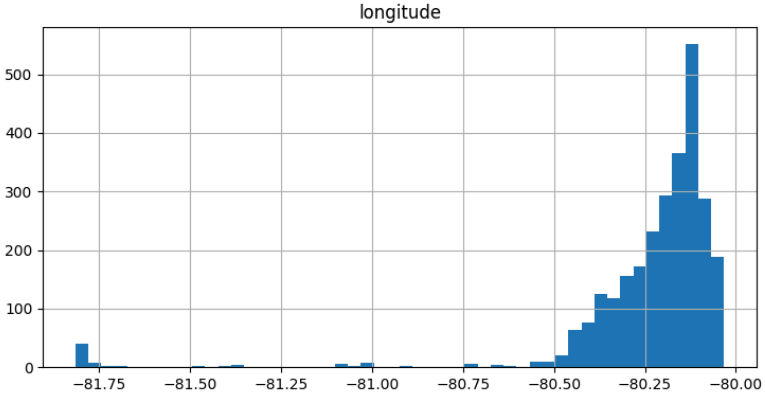
In [20]:
```
datos_1 = datos[(datos['price'] > 1000) & (datos['price'] < 20000000)]
datos_1[['bedrooms','bathrooms', 'price', 'latitude', 'longitude']].hist(bins = 50, figsize = (20, 15))
plt.show()
```





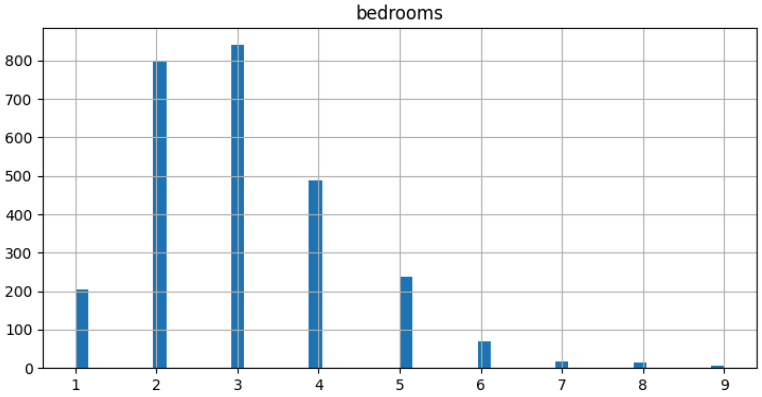


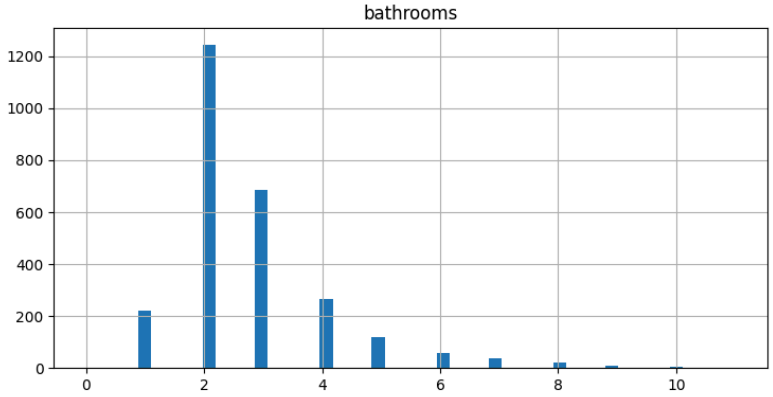


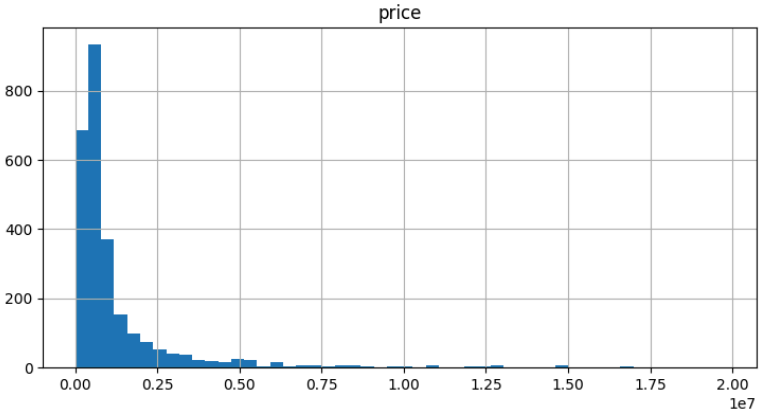


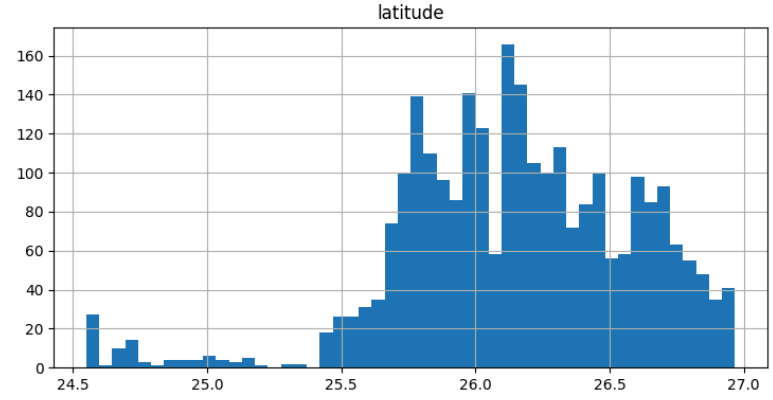
En este histograma haremos un filtro de las recamaras escogeremos que sea mayor a 0 y menor a 10, no contemplaremos datos más grandes a estos puesto que son pocos y causan ruido inecesario en el modelo

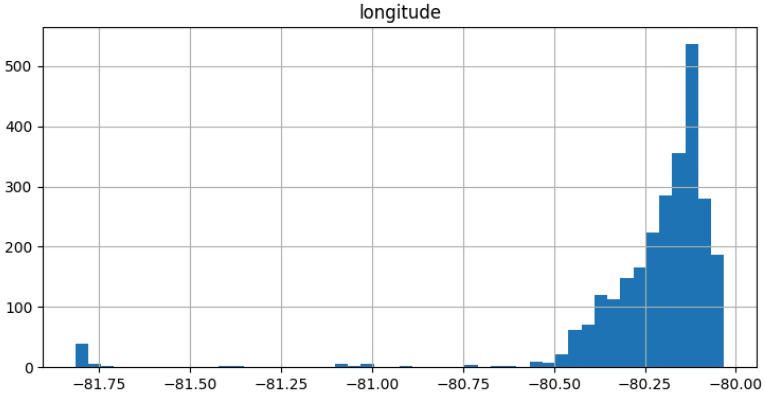
In [23]:
```
datos_1 = datos_1[(datos_1['bedrooms'] > 0) & (datos_1['bedrooms'] < 10)]
datos_1[['bedrooms','bathrooms', 'price', 'latitude', 'longitude']].hist(bins = 50, figsize = (20, 15))
plt.show()
```





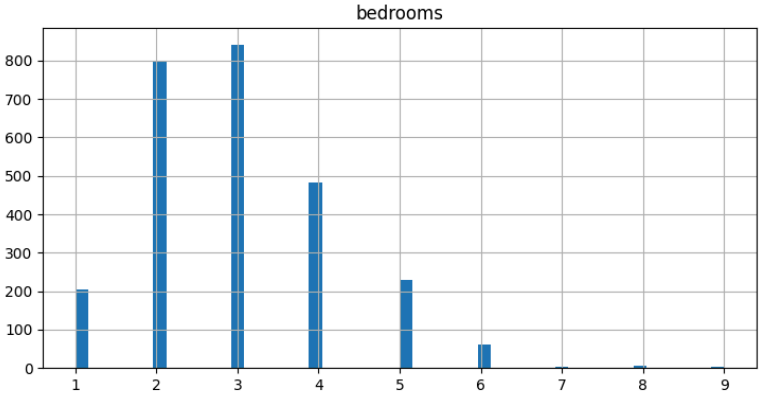


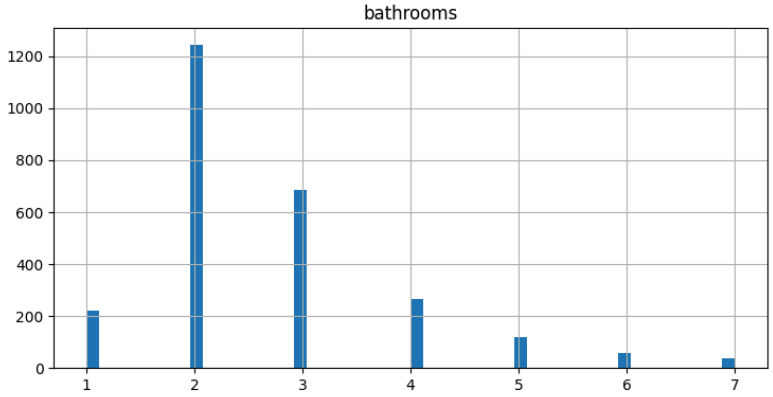


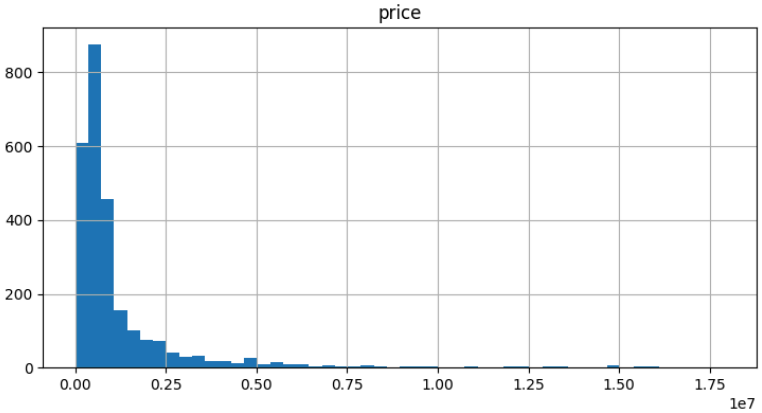


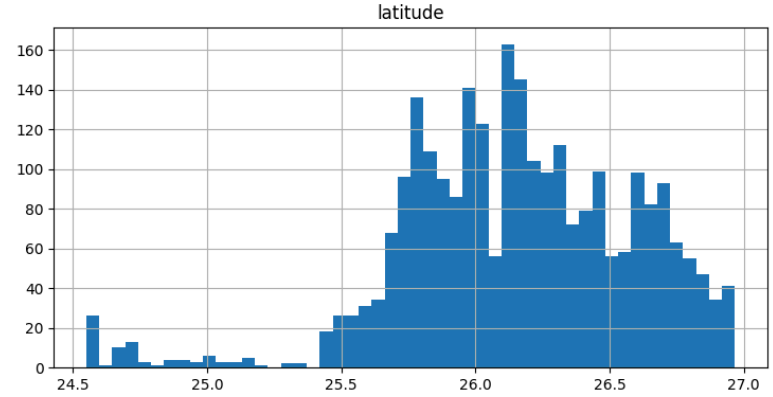
En este histograma haremos un filtro de las recamaras escogeremos que sea mayor a 0 y menor a 10, no contemplaremos datos más grandes a estos puesto que son pocos y causan ruido inecesario en el modelo

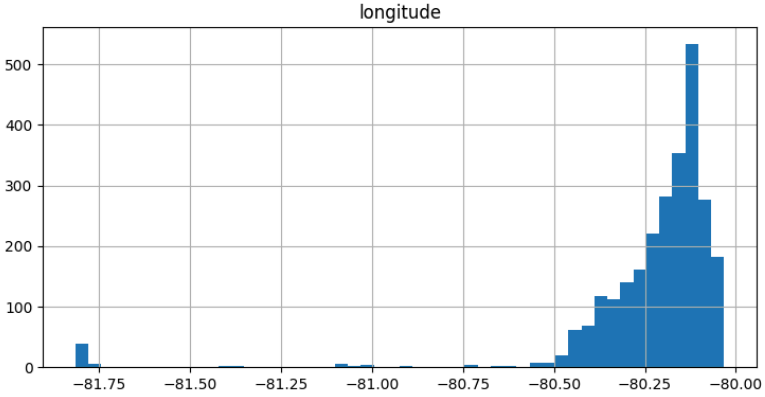
In [27]:
```
datos_1 = datos_1[(datos_1['bathrooms'] > 0) & (datos_1['bathrooms'] < 8)]
datos_1[['bedrooms','bathrooms', 'price', 'latitude', 'longitude']].hist(bins = 50, figsize = (20, 15))
plt.show()
```





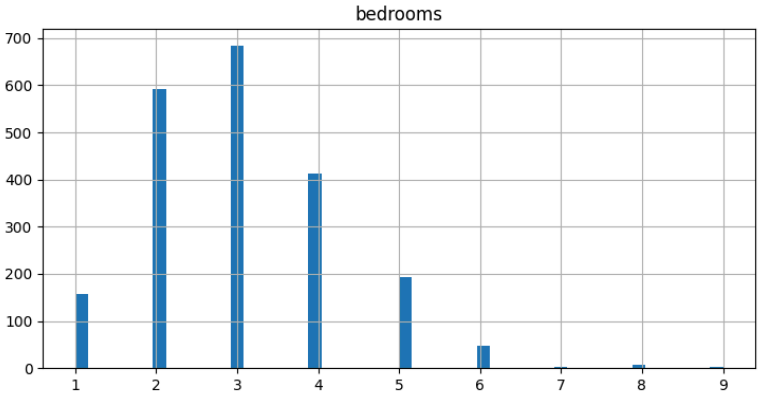


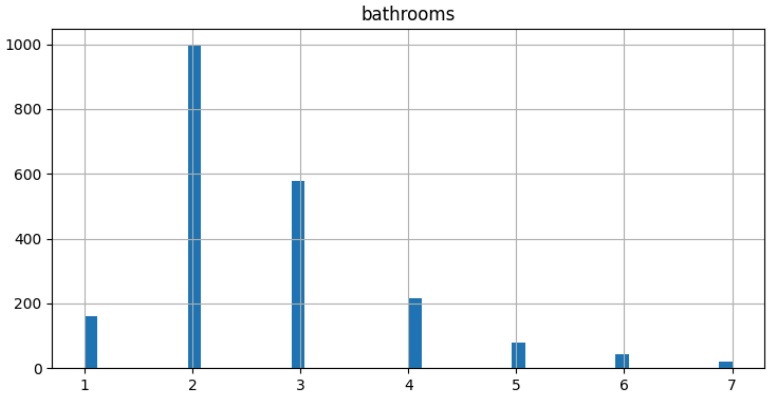


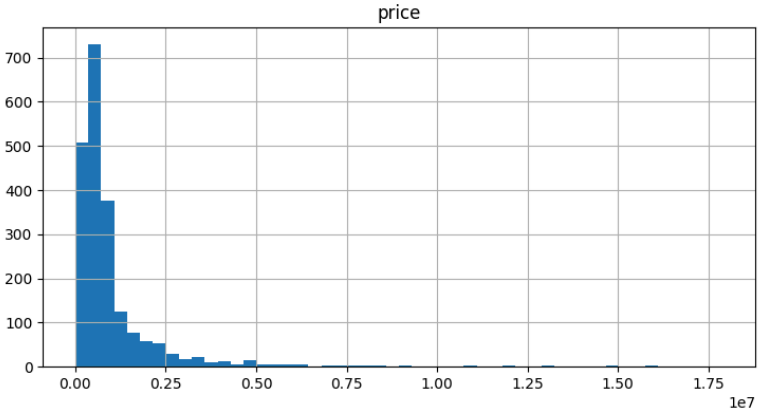


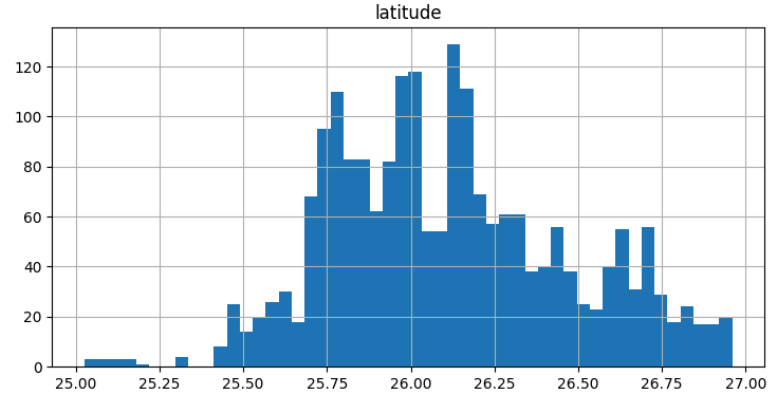
Meteremos un filtro para latitud y longitud tomando en cuenta la concentración de los datos

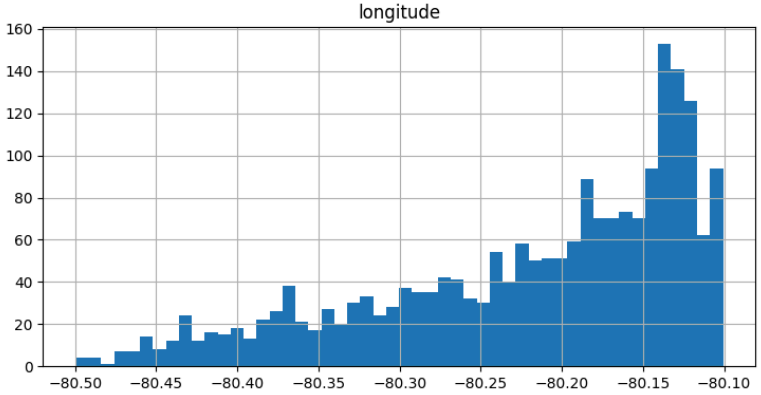
In [30]:
```
datos_1 = datos_1[
    (datos_1['latitude'] > 24.800000) & (datos_1['latitude'] < 26.967148) &
    (datos_1['longitude'] > -80.500000) & (datos_1['longitude'] < -80.100655)
]
datos_1[['bedrooms','bathrooms', 'price', 'latitude', 'longitude']].hist(bins=50, figsize=(20, 15))
plt.show()
```


bedrooms
700
600
500
400
300
200
100
0
1
2
3
4
5
6
7
8
9


bathrooms
1000
800
600
400
200
0
1
2
3
4
5
6
7


price
700
600
500
400
300
200
100
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e7


latitude
120
100
80
60
40
20
0
25.00
25.25
25.50
25.75
26.00
26.25
26.50
26.75
27.00


longitude
160
140
120
100
80
60
40
20
0
−80.50
−80.45
−80.40
−80.35
−80.30
−80.25
−80.20
−80.15
−80.10

En suma

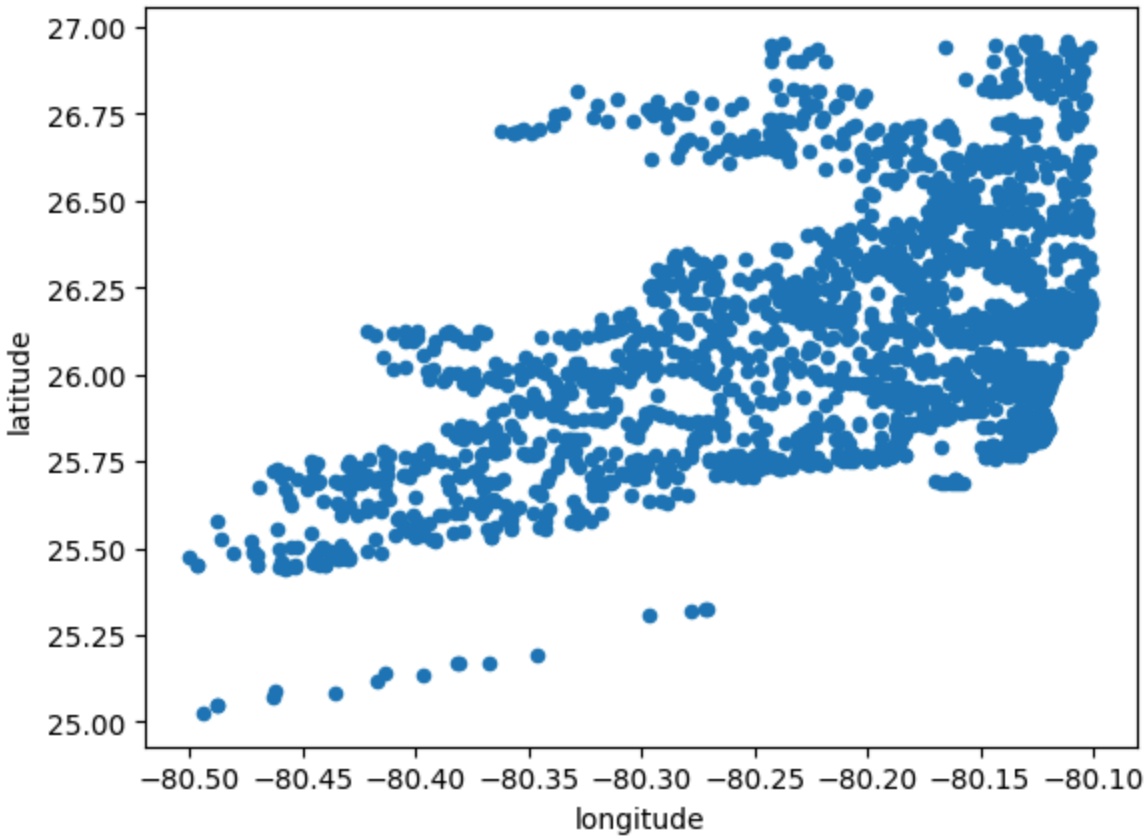
In [33]:
```
datos_1 = datos[
    (datos['price'] > 1000) & (datos['price'] < 20000000) &
    (datos['bedrooms'] > 0) & (datos['bedrooms'] < 10) &
    (datos['bathrooms'] > 0) & (datos['bathrooms'] < 8) &
    (datos['latitude'] > 24.800000) & (datos['latitude'] < 26.967148) &
    (datos['longitude'] > -80.500000) & (datos['longitude'] < -80.100655)
]
```

In [35]:
```
datos.shape, datos_1.shape
```

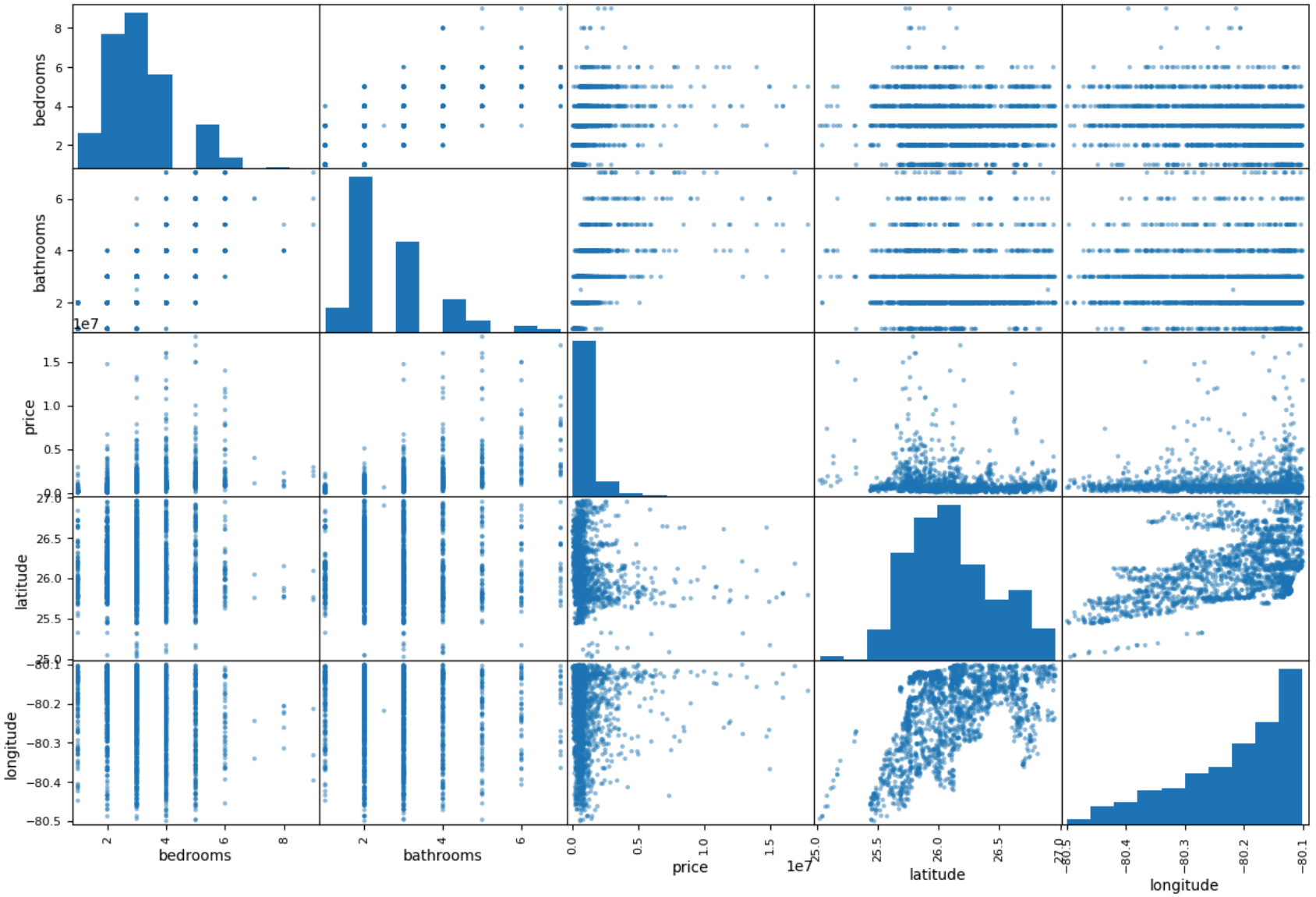
Out[35]:
```
((2806, 805), (2098, 805))
```

Visualización geográfica de la latitud y longitud

In [38]:
```
datos_1.plot(kind = 'scatter', x = 'longitude', y = 'latitude')
plt.show()
```



In [40]:
```
attributes = ['bedrooms', 'bathrooms', 'price', 'latitude', 'longitude']
scatter_matrix(datos_1[attributes], figsize=(15, 10), alpha=0.5, diagonal='hist')
plt.show()
```



Regresión Líneal Multiple

El modelo es una regresión lineal múltiple porque tiene diferentes variables independientes.

Usamos "C()" para crear la regresión lineal múltiple de manera manual, porque estamos usando variablescategoricas y numéricas. Las variables numéricas del modelo son:

- price
- bedrooms
- badrooms
- latitude
- longitude

Las variables de texto son:

- city
- state
- homeStatus

Como no son iguales nuestras variables porque son de texto y númericas, la función C() crea la codificación automáticamente de las variables de texto. Además statsmodels usa la librería Patsy para convertir variables categóricas en variables automáticamente.

Explicación de la composición del modelo

Se va a predicir el precio de una propiedad en función de sus:

- Características físicas (número de recámaras y baños)
- Ubicación geográfica (latitud y longitud)
- Ubicación administrativa (ciudad y estado)
- El estatus de publicación (vendida, en venta, pendiente, etc.).

El C(...) le dice al modelo que esas variables son categóricas (texto), y que debe convertirlas automáticamente en variables dummy. Así, por ejemplo, puedes saber si estar en “Miami” o tener status “Sold” tiene un impacto positivo o negativo en el precio.

Dummies

In [46]:
```
# 5. Agrupar categorías poco frecuentes (city, state)
frecuentes_ciudades = datos_1['city'].value_counts().nlargest(20).index
```

```
datos_1.loc[:, 'city'] = datos_1['city'].where(datos_1['city'].isin(frecuentes_ciudades), other='Otros')

frecuentes_estados = datos_1['state'].value_counts().nlargest(10).index
datos_1.loc[:, 'state'] = datos_1['state'].where(datos_1['state'].isin(frecuentes_estados), other='Otros')
```

In [48]:
```
# 6. Crear variables dummy para city, state y homeStatus
datos_encoded = pd.get_dummies(datos_1, columns=['city', 'state', 'homeStatus'], drop_first=True)
```

In [50]:
```
# 7. Separar variables predictoras y objetivo
X = datos_encoded.drop(columns=['price'])
y = datos_encoded['price']
```

In [58]:
```
# 8. Imputar valores faltantes en variables numéricas
X_numeric = X.select_dtypes(include=[np.number])
X_numeric = X_numeric.dropna(axis=1, how='all')  # eliminar columnas con solo NaNs

imputer = SimpleImputer(strategy='mean')
X_numeric_imputed = imputer.fit_transform(X_numeric)
```

In [60]:
```
# 9. Escalar variables numéricas
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_numeric_imputed)
```

In [62]:
```
# 10. Dividir en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

In [64]:
```
# 11. Entrenamiento de modelos
lr = LinearRegression().fit(X_train, y_train)
ridge = Ridge(alpha=1.0).fit(X_train, y_train)
lasso = Lasso(alpha=0.1, max_iter=10000).fit(X_train, y_train)
```

In [66]:
```
# 12. Evaluación de modelos
def evaluar_modelo(nombre, modelo):
    y_pred = modelo.predict(X_test)
    print(f'\n--- {nombre} ---')
    print(f'MAE:  {mean_absolute_error(y_test, y_pred):,.2f}')
    print(f'MSE:  {mean_squared_error(y_test, y_pred):,.2f}')
    print(f'RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):,.2f}')
    print(f'R²:   {r2_score(y_test, y_pred):.4f}')
```

```
evaluar_modelo('Regresión Lineal', lr)
evaluar_modelo('Ridge', ridge)
evaluar_modelo('Lasso', lasso)
```

```
--- Regresión Lineal ---
MAE:  0.00
MSE:  0.00
RMSE: 0.00
R²:   1.0000

--- Ridge ---
MAE:  6,682.00
MSE:  394,945,519.39
RMSE: 19,873.24
R²:   0.9995

--- Lasso ---
MAE:  4,913.59
MSE:  1,257,538,304.26
RMSE: 35,461.79
R²:   0.9986
```

In [70]:
```
# 13. Modelo con statsmodels (opcional)
# Puedes usar solo algunas variables si deseas claridad
formula = 'price ~ bedrooms + bathrooms + latitude + longitude + yearBuilt + price_to_rent_ratio_InfoTOD + pageViewCo
modelo = ols(formula, data=datos_1).fit()
modelo_stats = ols(formula, data=datos_1).fit()
print(modelo_stats.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.531
Model:                            OLS   Adj. R-squared:                  0.528
Method:                 Least Squares   F-statistic:                     182.0
Date:                Fri, 18 Jul 2025   Prob (F-statistic):          3.02e-205
Time:                        23:55:49   Log-Likelihood:                -19791.
No. Observations:                1294   AIC:                         3.960e+04
Df Residuals:                    1285   BIC:                         3.965e+04
Df Model:                           8
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                     2.155e+08   3.44e+07      6.266      0.000    1.48e+08    2.83e+08
bedrooms                     -2.944e+05   4.07e+04     -7.235      0.000   -3.74e+05   -2.15e+05
bathrooms                     3.331e+05   5.01e+04      6.655      0.000    2.35e+05    4.31e+05
latitude                     -7.783e+05   1.02e+05     -7.610      0.000   -9.79e+05   -5.78e+05
longitude                     2.412e+06    4.2e+05      5.738      0.000    1.59e+06    3.24e+06
yearBuilt                    -1366.2966   1758.327     -0.777      0.437   -4815.803    2083.210
price_to_rent_ratio_InfoTOD  3269.7612    339.528      9.630      0.000    2603.671    3935.852
pageViewCount                  49.3853     50.999      0.968      0.333     -50.665     149.435
livingArea                    818.9176     57.390     14.269      0.000     706.330     931.505
==============================================================================
Omnibus:                     1601.820   Durbin-Watson:                   2.013
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           270636.762
Skew:                           6.336   Prob(JB):                         0.00
Kurtosis:                      72.706   Cond. No.                     3.38e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.38e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [ ]: