In [1]:

```
#IMPORTAMOS LIBRERIAS

import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


#IMPORTAMOS EL ARCHIVO
df = pd.read_csv("pgcf.csv")
```

In [2]:

```
df.head()
```

Out[2]:

	Comuna	Propiedad	Precio	Direccion	Tamaño en m2	Dormitorios	Baños	ID
0	Tomé	Casa	221248916	Pingueral, Tomé, Biobío	559	4	2	0
1	Tomé	Casa	291797402	Avenida Pingueral, Tomé, Chile, Pingueral, Tom...	1068	4	4	1
2	Tomé	Casa	299923405	Bella Casa Con Piscina 3d 2b T350 M2 , Tomé, T...	350	3	2	2
3	Tomé	Casa	254861021	Avenida Pingueral, Tomé, Chile, Pingueral, Tom...	560	4	3	3
4	Tomé	Casa	1244756006	Avenida Pingueral, Tomé, Chile, Pingueral, Tom...	875	6	4	4

La variable Direccion e ID serán eliminadas, la columna Direccion no tiene el mismo formato en todos los datos por lo que no será de ayuda y la variable ID no aporta nada.

In [3]:

```
df=df.drop(columns=["ID", "Direccion"])
df
```

Out[3]:

	Comuna	Propiedad	Precio	Tamaño en m2	Dormitorios	Baños
0	Tomé	Casa	221248916	559	4	2
1	Tomé	Casa	291797402	1068	4	4
2	Tomé	Casa	299923405	350	3	2
3	Tomé	Casa	254861021	560	4	3
4	Tomé	Casa	1244756006	875	6	4
...	...	...	...	...	...	...
2957	Chiguayante	Departamento	60000000	53	3	1
2958	Chiguayante	Departamento	184681900	85	2	2
2959	Chiguayante	Departamento	74000000	59	3	1
2960	Chiguayante	Departamento	350000000	145	4	3
2961	Chiguayante	Departamento	151439158	90	3	2

2962 rows × 6 columns

In [9]:

```
df.describe()
```

Out[9]:

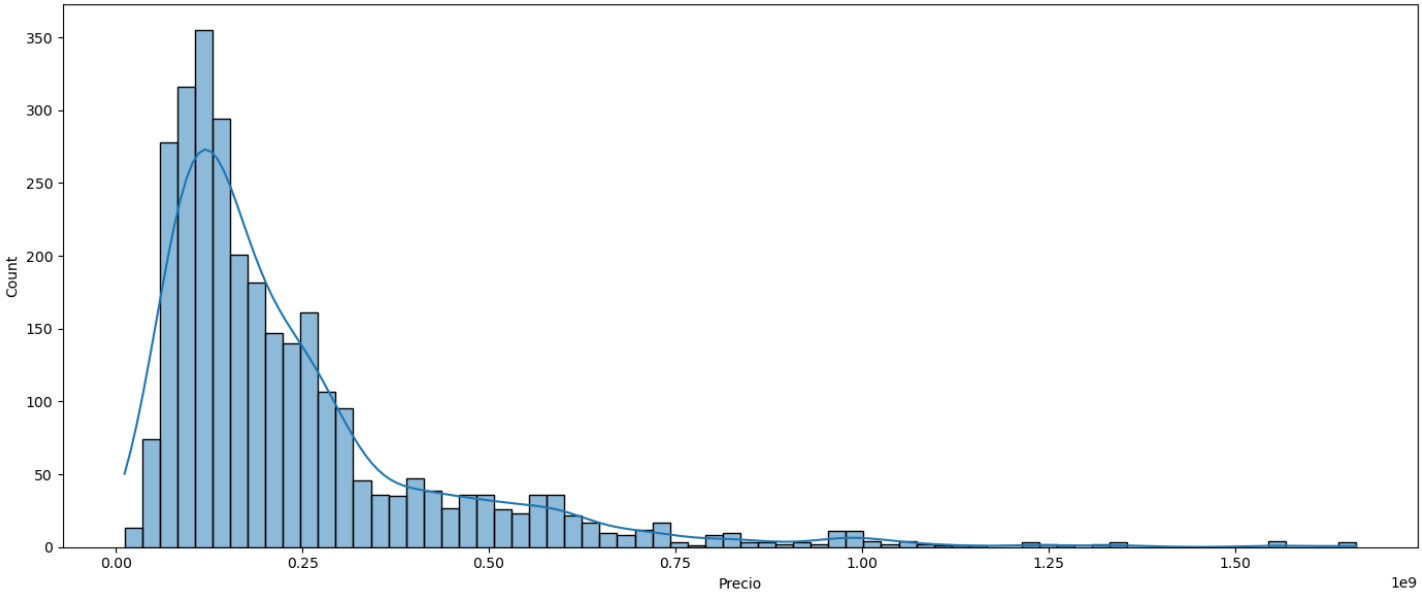
	Precio	Tamaño en m2	Dormitorios	Baños
count	2.962000e+03	2962.000000	2962.000000	2962.000000
mean	2.396467e+08	248.231600	3.210668	2.343687
std	2.062407e+08	604.934302	1.433255	1.165490
min	1.163496e+07	20.000000	1.000000	1.000000
25%	1.108091e+08	65.000000	2.000000	2.000000
50%	1.699073e+08	120.000000	3.000000	2.000000
75%	2.811320e+08	242.000000	4.000000	3.000000
max	1.662137e+09	15000.000000	18.000000	15.000000

ANALSIS EXPLORATORIO DE VARIABLES

- **VARIABLE "Precio"**

In [179]:

```
# GRAFICO VARIABLE Precio

fig, axes = plt.subplots(1, figsize=(14, 6))
sns.histplot(x="Precio", data=df, kde=True)

plt.tight_layout()
plt.show()
```

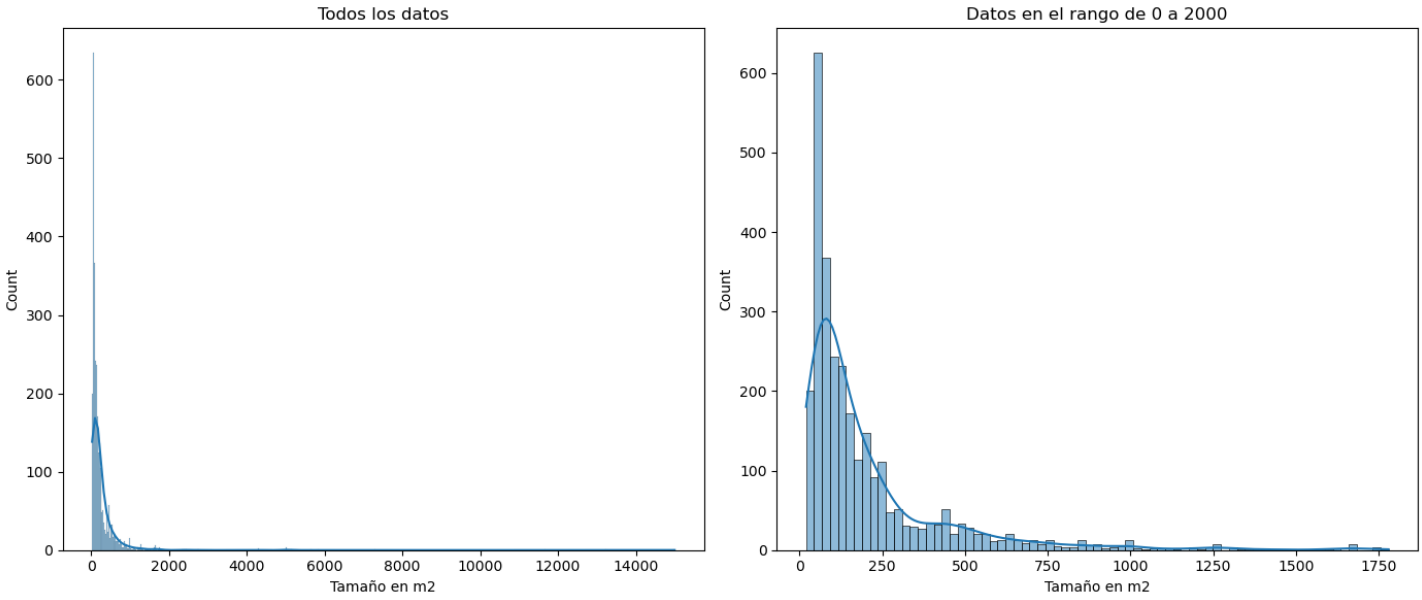


En el gráfico anterior se observa que el "Precio" NO tiene una distribución normal siendo esta asimétrica a la derecha, concentrando el 75% de los valores entre los casi 12 millones de pesos chilenos hasta los 280 millones de pesos chilenos aproximadamente.

- **VARIABLE "Tamaño en m2"**

In [180]:

```
# GRAFICO VARIABLE Tamaño en m2

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# GRAFICO DE TODOS LOS DATOS
sns.histplot(x="Tamaño en m2", data=df, kde=True, ax=axes[0])
axes[0].set_title("Todos los datos")

# GRAFICO DATOS FILTRADOS ENTRE 0 A 2000 M2
tamanof = df[df["Tamaño en m2"].between(0, 2000)]

sns.histplot(x="Tamaño en m2", data=tamanof, kde=True, ax=axes[1])
axes[1].set_title("Datos en el rango de 0 a 2000")


plt.tight_layout()
plt.show()
```



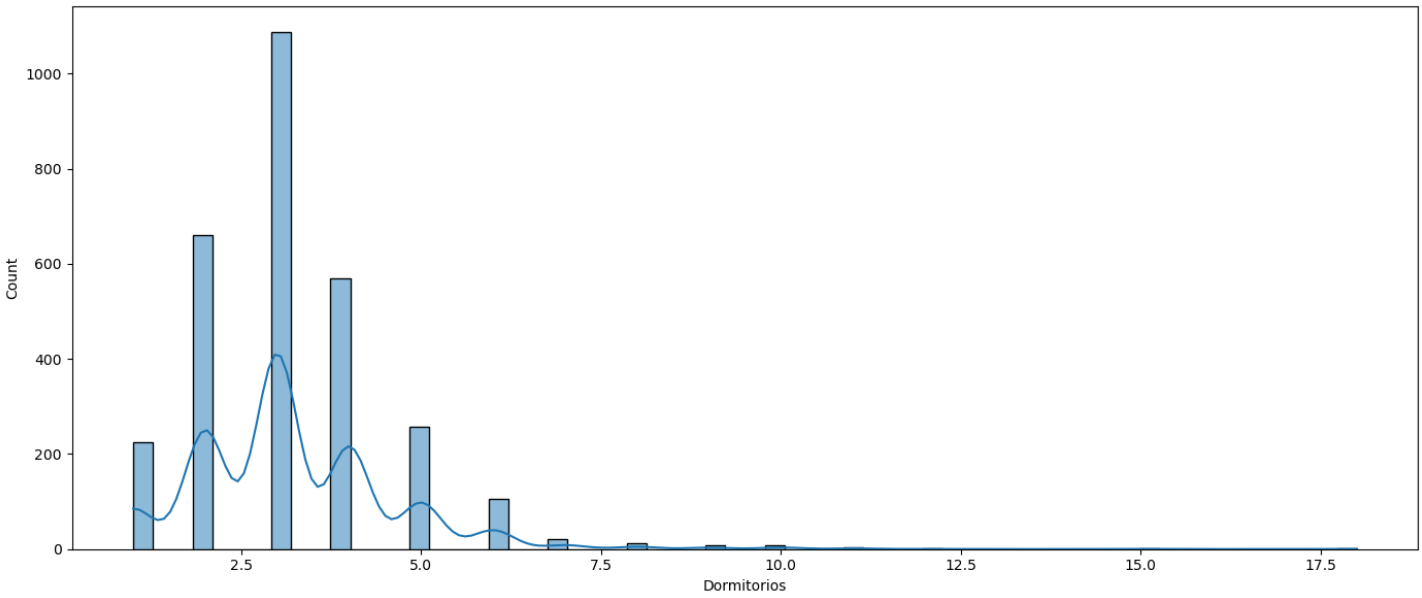
En el gráfico anterior se observa que la variable "Tamaño en m2" NO tiene una distribución normal siendo esta asimétrica a la derecha, donde en este caso hay valores mas extremos por lo que se tuvo que filtrar los datos en el rango de [0, 2000]m2 para tener una gráfica con mayor visivilidad, concentrando el 75% de los valores entre los 20 m2 hasta los 242 m2.

- **VARIABLE "Dormitorios"**

In [181]:

```
#GRAFICO VARIABLE "Dormitorios"
```

```
fig, axes = plt.subplots(1, figsize=(14, 6))
sns.histplot(x="Dormitorios", data=df, kde=True)

plt.tight_layout()
plt.show()
```

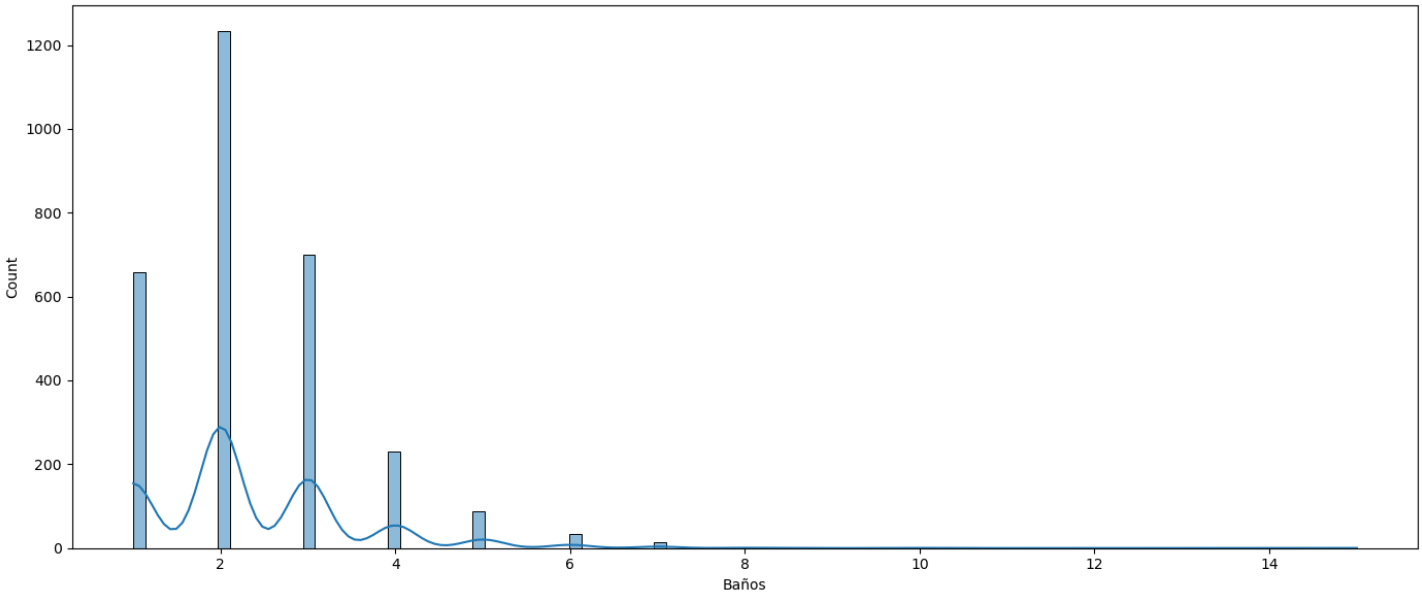


En el gráfico anterior se observa que la variable "Dormitorios" NO tiene una distribución normal siendo esta asimétrica a la derecha, concentrando el 75% de los valores entre 1 a 4 dormitorios.

- **VARIABLE "Baños"**

In [182]:

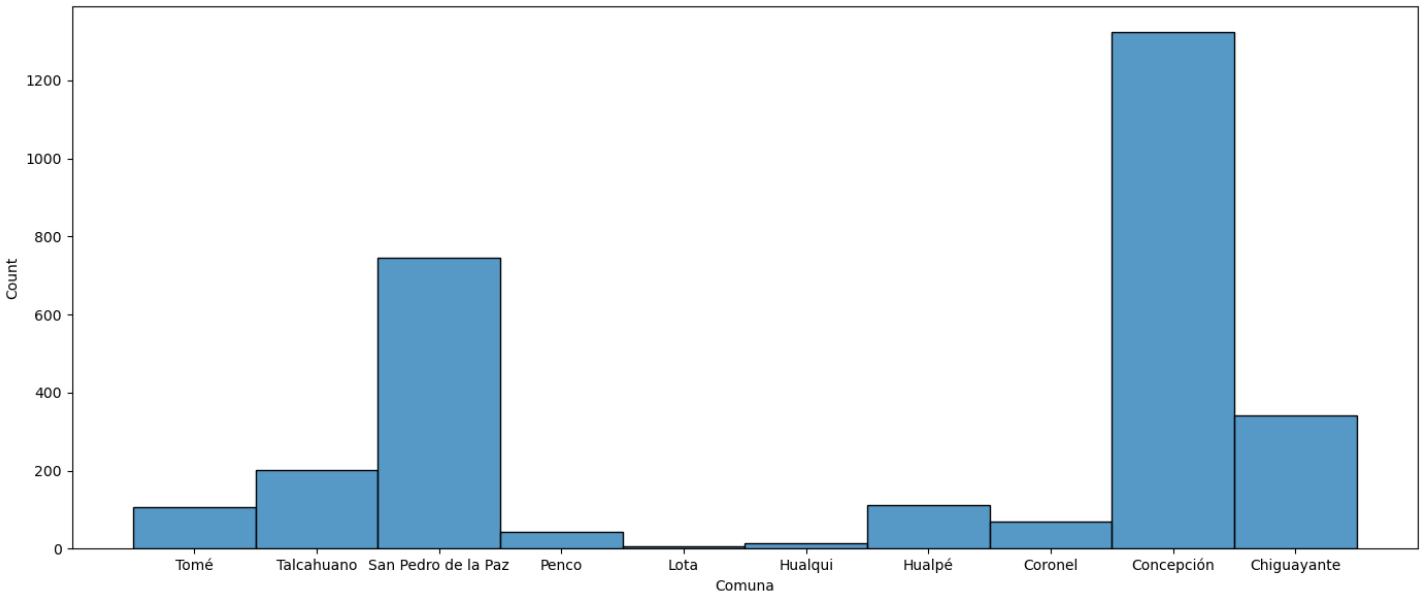
```
#GRAFICO VARIABLE Baños

fig, axes = plt.subplots(1, figsize=(14, 6))
sns.histplot(x="Baños", data=df, kde=True)

plt.tight_layout()
plt.show()
```



- **VARIABLE "Comuna"**

In [183]:

```
#GRAFICO VARIABLE Comuna

fig, axes = plt.subplots(1, figsize=(14, 6))
sns.histplot(x="Comuna", data=df)

plt.tight_layout()
plt.show()
```



In [184]:

```
conteoc = df['Comuna'].value_counts()

print(conteoc)
```

```
Concepción             1324
San Pedro de la Paz     745
Chiguayante             341
Talcahuano              201
Hualpé                  112
Tomé                    106
Coronel                  69
Penco                    42
Hualqui                  15
Lota                      7
Name: Comuna, dtype: int64
```

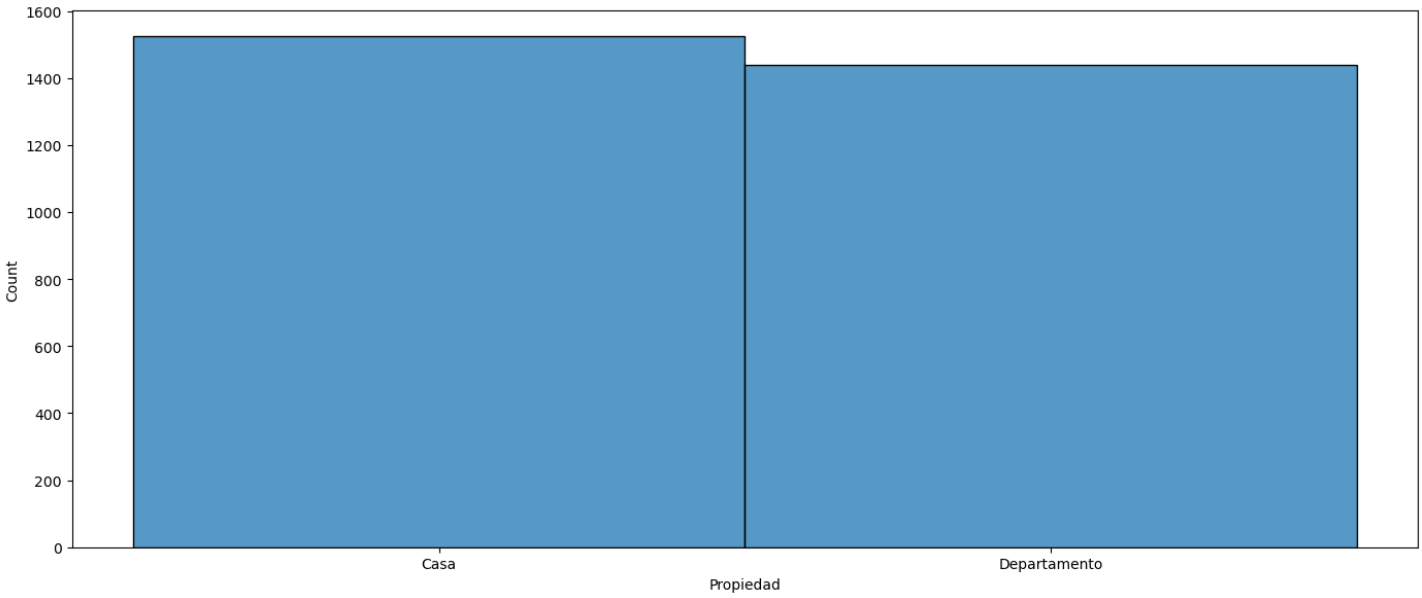
En el gráfico anterior se observa la distribución de los datos en la variable "Comuna" donde casi el 70% de los datos los concentran las comunas de Concepción y San Pedro de la Paz.

- **VARIABLE "Propiedad"**

In [185]:

```
#GRAFICO VARIABLE Propiedad

fig, axes = plt.subplots(1, figsize=(14, 6))
sns.histplot(x="Propiedad", data=df)
```

```
plt.tight_layout()
plt.show()
```



In [186]:

```
conteop = df['Propiedad'].value_counts()

print(conteop)
```

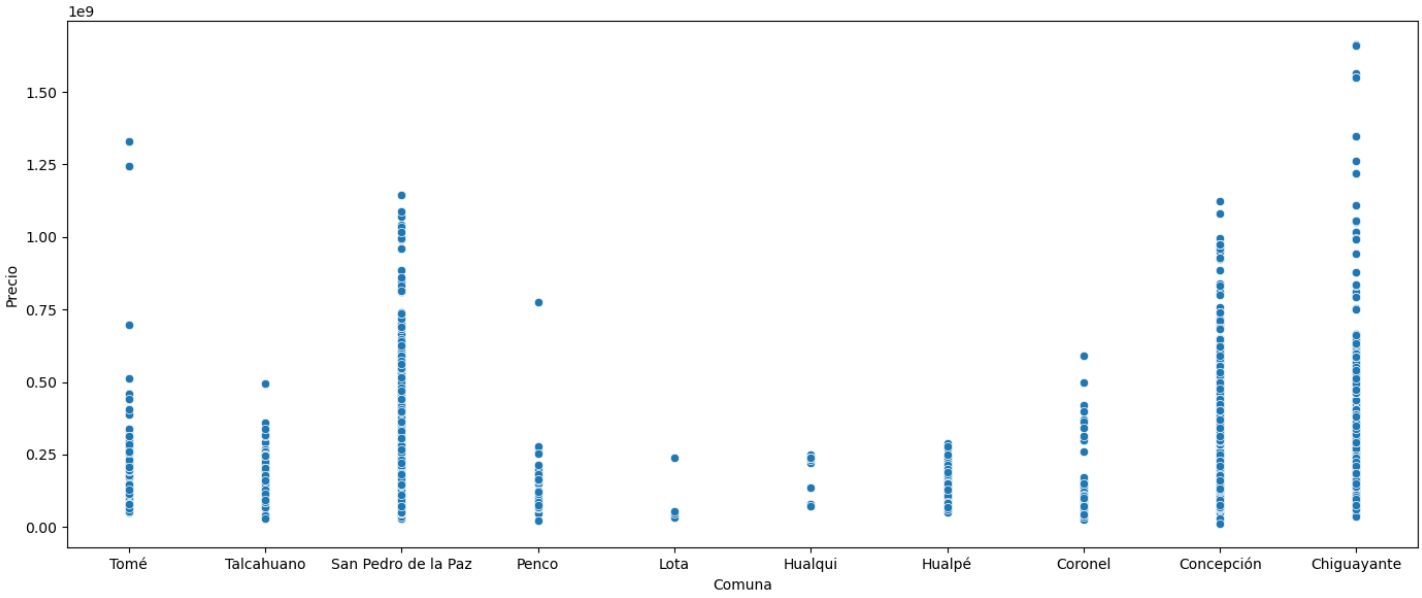
```
Casa            1524
Departamento    1438
Name: Propiedad, dtype: int64
```

En el gráfico anterior se observa la distribución de los datos en la variable "Propiedad" donde se distribuyen casi de manera uniforme siendo el 51,4% propiedades de tipo Casa y 48,6% para propiedades de tipo Departamento

- **VARIABLES PRECIO Y COMUNA**

In [187]:

```
#PRECIO Y COMUNA
fig, axes = plt.subplots(1, figsize=(14, 6))
sns.scatterplot(x="Comuna", y = "Precio", data = df )
plt.tight_layout()
plt.show()
```



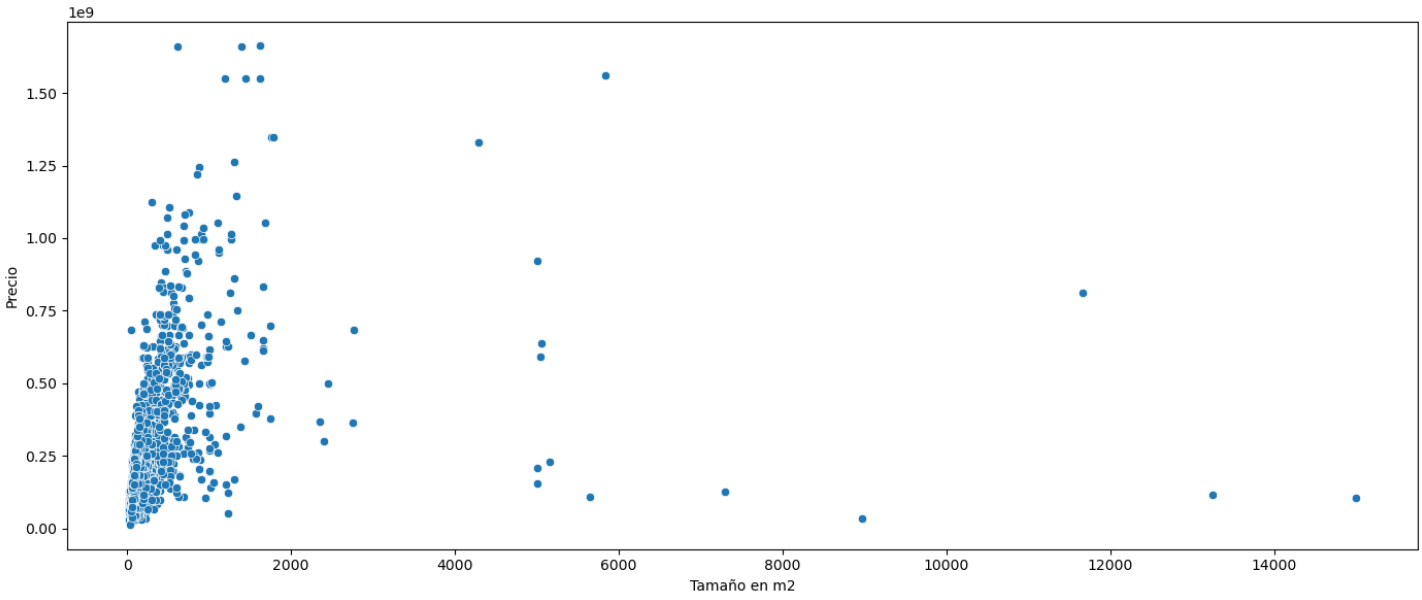
En este gráfico podemos ver la relación de la variable "Precio" y la variable "Comuna", destacando la comuna de

Chiguayante con los valores más altos seguida por San Pedro de la Paz, Concepción y Tomé con algunos valores superiores a los mil millones, por el lado contrario las comunas con los valores más bajos son Lota, Hualqui, Hualpén y Penco esta ultima teniendo un valor que se escapa de la distribución general de sus valores, Talcahuano y Coronel poseen algunos valores un poco mas elevados pero que no tienen gran diferencia.

- **VARIABLES PRECIO Y TAMAÑO EN M2**

In [188]:

```
#PRECIO Y TAMAÑO
fig, axes = plt.subplots(1, figsize=(14, 6))
sns.scatterplot(x="Tamaño en m2", y = "Precio", data = df )
plt.tight_layout()
plt.show()
```



In [189]:

```
from scipy.stats import pearsonr

pearsont, _= pearsonr(df['Tamaño en m2'], df['Precio'])

print("Coeficiente de correlación:", pearsont)
```

```
Coeficiente de correlación: 0.35534998130003664
```

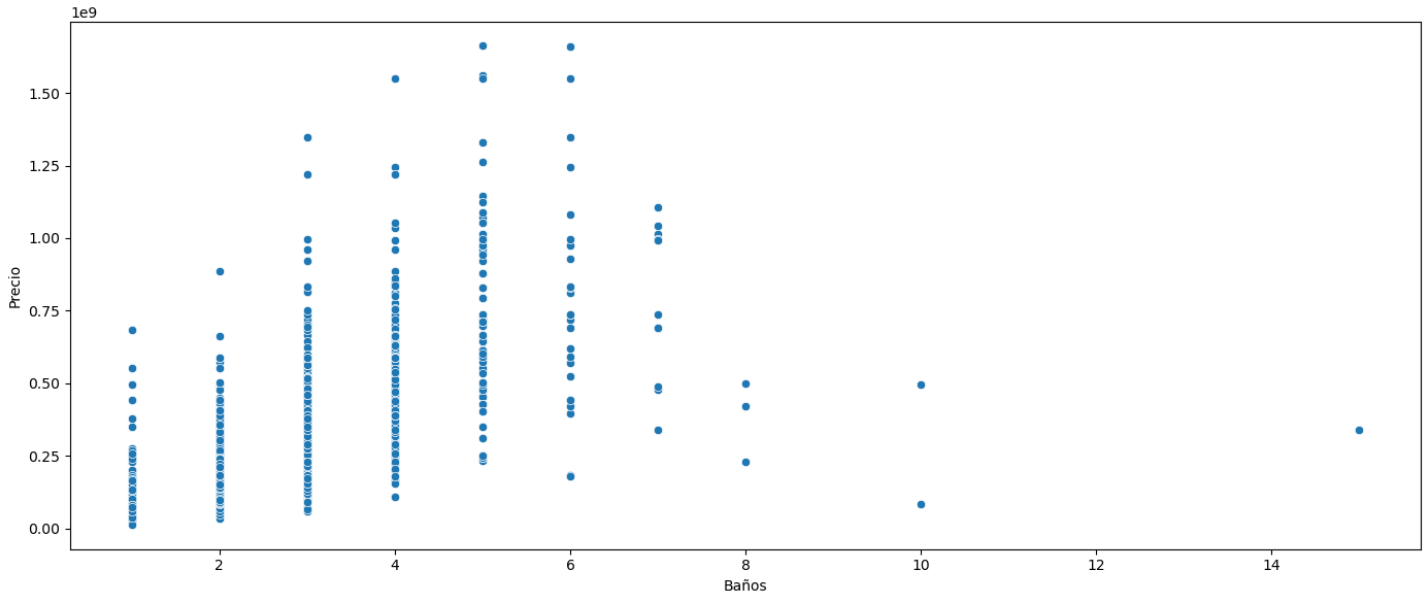
Observando el gráfico podemos observar que existe muy poca correlación entre ambas variables, ya que los datos están bastante dispersos, esto lo podemos corroborar calculando el coeficiente de Pearson el cual es 0.35 estando mas cercano a 0 que a 1

- **PRECIO Y BAÑOS**

In [190]:

```
#PRECIO Y BAÑOS
fig, axes = plt.subplots(1, figsize=(14, 6))
```

```
sns.scatterplot(x="Baños", y = "Precio", data = df )
plt.tight_layout()
plt.show()
```



In [191]:

```
pearsonb, _= pearsonr(df['Baños'], df['Precio'])

print("Coeficiente de correlación:", pearsonb)
```

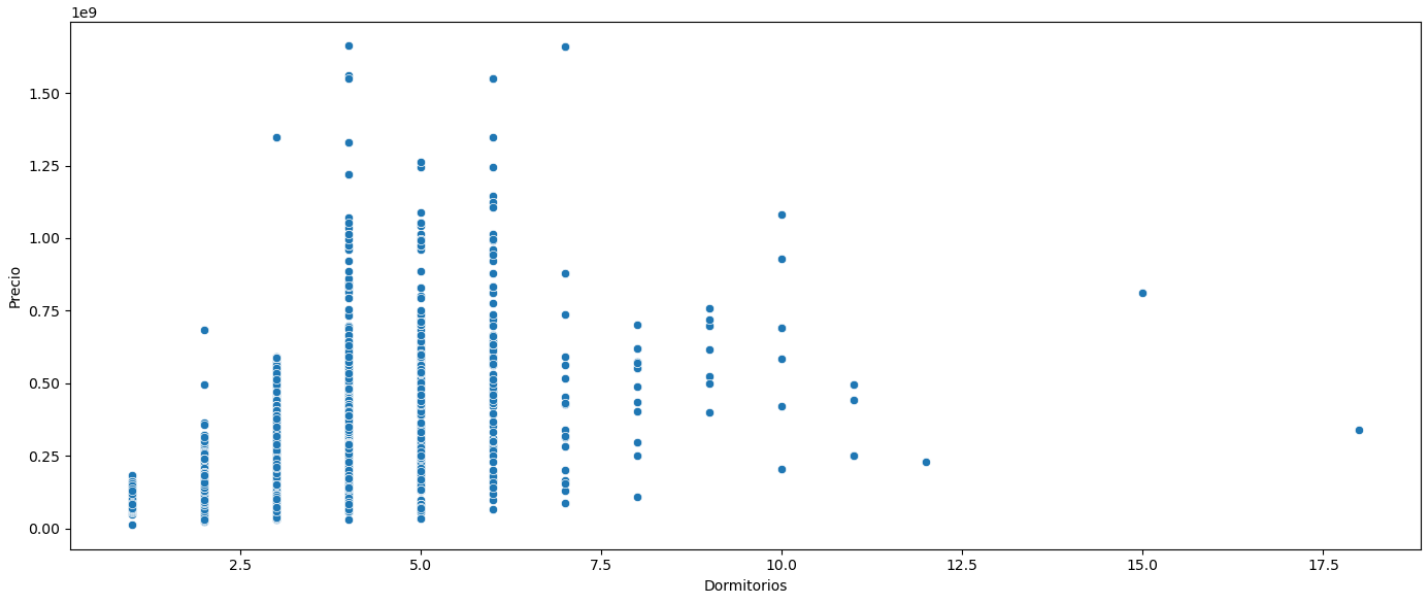
```
Coeficiente de correlación: 0.735685384899389
```

En esta gráfico podemos observar una mayor relación, donde al aumentar la cantidad de baños el precio "base" de las propiedades aumenta levemente y también aumenta el precio máximo que alcanzan estas, al calcular el coeficiente de Pearson observamos que este aumenta siendo mucho mas cercano a 1.

- **PRECIO Y DORMITORIOS**

In [192]:

```
#PRECIO Y DORMITORIOS

fig, axes = plt.subplots(1, figsize=(14, 6))
sns.scatterplot(x="Dormitorios", y = "Precio", data = df )
plt.tight_layout()
plt.show()
```



In [193]:

```
pearsond, _= pearsonr(df['Dormitorios'], df['Precio'])

print("Coeficiente de correlación:", pearsond)
```

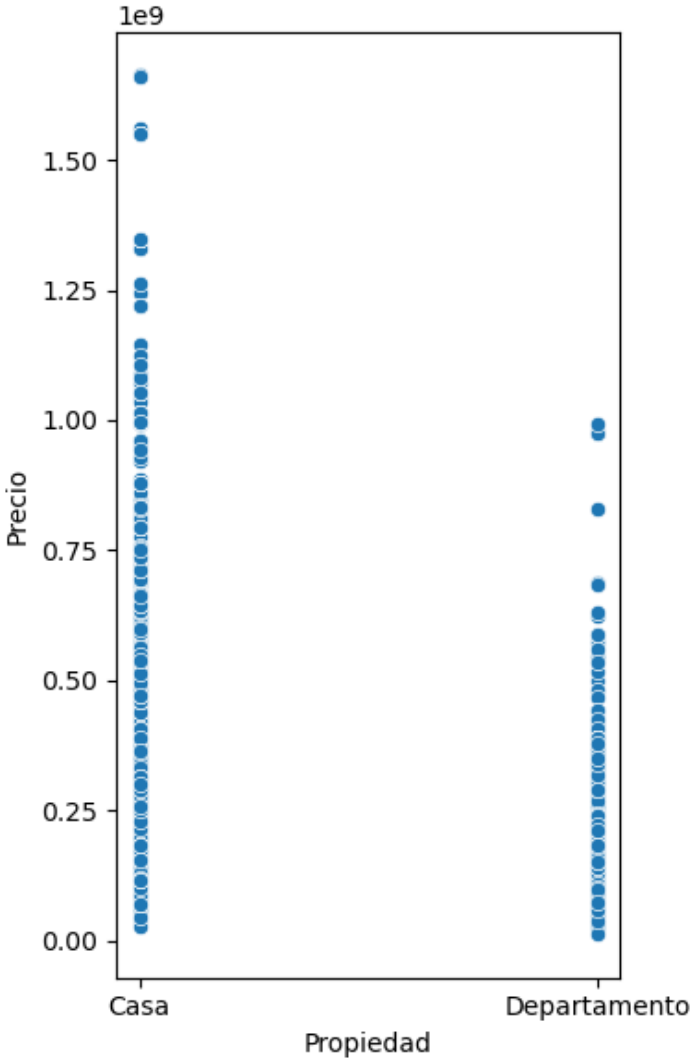
```
Coeficiente de correlación: 0.5657825610664862
```

En esta gráfica también se puede observar cierta relación, donde desde 1 a 6 dormitorios los precios máximos que pueden alcanzar aumentan progresivamente y desde 6 dormitorios en adelante los precios "base" de las propiedades aumentan levemente, al calcular el coeficiente de Pearson obtenemos 0,56 teniendo una fuerza de relación mediana.

- **PRECIO Y PROPIEDAD**

In [194]:

```
#PRECIO Y PROPIEDAD

fig, axes = plt.subplots(1, figsize=(4, 6))
sns.scatterplot(x="Propiedad", y = "Precio", data = df )
plt.tight_layout()
plt.show()
```



Mirando esta gráfica se puede observar que en este data frame los precios de las casas llegan a valores más altos que los de departamentos

DIVISION EN SET ENTRENAMIENTO Y TESTEO

In [195]:

```
from sklearn.model_selection import train_test_split
```

```
X = df.drop(['Precio'], axis = 1)
y = df['Precio']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=20
24)
```

PREPROCESAMIENTO N°2, PASAR VARIABLES CATEGORICAS A DUMMYS Y ESTANDARIZAR LAS VARIABLES

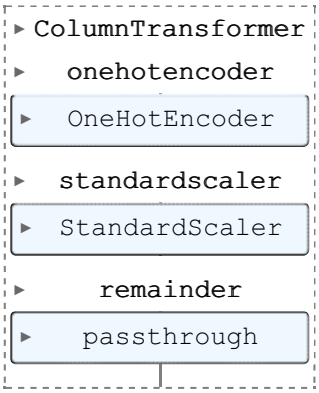
In [196]:

```
#SEPARACIÓN DE VARIABLES

variables_categoricas = X_train.columns[X_train.dtypes == "object"]
variables_numericas   = X_train.columns[X_train.dtypes != "object"]
```

In [197]:

```
#INSTANCIAMOS LAS CLASES, CREAMOS EL TRANSFORMER Y REALIZAMOS EL FIT PARA AJUSTAR EL PREP
ROCESAMIENTO EN EL SET DE ENTRENAMIENTO

from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

one_hot = OneHotEncoder(drop='first')
scaler  = StandardScaler()

transformer = make_column_transformer(
    (one_hot, variables_categoricas),
    (scaler,  variables_numericas),
    remainder                 = "passthrough",
    verbose_feature_names_out = False
)

transformer.fit(X_train)
```

Out[197]:



In [198]:

```
#APLICAMOS EL PREPROCESAMIENTO

X_train_prep = pd.DataFrame(
    data=transformer.transform(X_train),
    columns=transformer.get_feature_names_out(),
    index=X_train.index
)

X_test_prep = pd.DataFrame(
    data=transformer.transform(X_test),
    columns=transformer.get_feature_names_out(),
    index=X_test.index
)
```

MODELO ARBOL DE DECISION

In [199]:

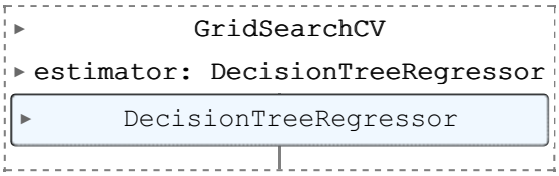
```
#GRILA DE HIPERPARAMETROS

hiperparametros_arbol= {
    "min_samples_split": np.arange(2,10, 1),  # OBSERVACIONES MINIMAS PARA DIVISION
    "ccp_alpha": np.logspace(-4,1,25),  # COSTO DE COMPLEJIDAD
    "max_depth": np.arange(4, 5, 1),  # PROFUNDIDAD MAXIMA DEL ARBOL
    "min_samples_leaf":np.arange(2,10, 1), # OBSERVACIONES MINIMAS POR CADA NODO HIJO
    "max_features":np.arange(2,10, 1) # CANTIDAD MAXIMA DE CARACTERISTICAS PARA DIVIDIR
UN NODO
    }
```

In [200]:

```
#BUSQUEDA DE MEJORES HIPERPARAMETROS CON VALIDACION CRUZADA OPTIMIZANDO LA METRICA R2

from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV, KFold

configuraciones = KFold(n_splits=5, shuffle=True, random_state=2024)
modelo_arbol          = DecisionTreeRegressor( random_state=2024)

modelos_arbol = GridSearchCV(
    estimator  = modelo_arbol,
    param_grid = hiperparametros_arbol,
    cv         = configuraciones,
    scoring    = "r2",
    n_jobs     = -1,
    verbose    = 1
)
#modelos_arbol.param_grid['max_depth'] = [4]

modelos_arbol.fit(X_train_prep, y_train)
```

```
Fitting 5 folds for each of 12800 candidates, totalling 64000 fits
```

Out[200]:



In [201]:

```
#MEJORES PARAMETROS ENCONTRADOS

modelos_arbol.best_params_
```

Out[201]:

```
{'ccp_alpha': 0.0001,
 'max_depth': 4,
 'max_features': 9,
 'min_samples_leaf': 8,
 'min_samples_split': 2}
```

In [202]:

```
#PREDICCIONES EN CONJUNTO DE ENTRENAMIENTO Y TEST

y_train_predar = modelos_arbol.predict(X_train_prep)
y_test_predar = modelos_arbol.predict(X_test_prep)
```

MODELO RANDOM FOREST

In [203]:

```
hiperparametros_rf = {
    'n_estimators': np.arange(300, 401, 20),  # NUMERO DE ARBOLES
```

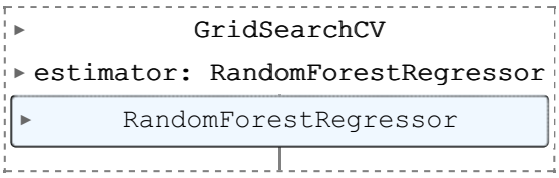
```
    'max_features': ['sqrt', 'log2', None], # NUMERO DE PREDICTORES MUESTREADOS PARA CAD
A ARBOL
    'min_samples_split': np.arange(2, 10, 1),  # OBSERVACIONES MINIMAS PARA DIVISION
    'max_depth': np.arange(4, 5, 1), # PROFUNDIDAD MAXIMA DE CADA ARBOL
    'min_samples_leaf':(2, 10, 1) # OBSERVACIONES MINIMAS POR CADA NODO HIJO
}
```

In [204]:

```
#BUSQUEDA DE MEJORES HIPERPARAMETROS CON VALIDACION CRUZADA OPTIMIZANDO LA METRICA R2 PAR
A RANDOM FOREST

from sklearn.ensemble import RandomForestRegressor

configuraciones = KFold(n_splits=5, shuffle=True, random_state=2024)
modelo_rf          = RandomForestRegressor( random_state=2024)

modelos_rf = GridSearchCV(
    estimator  = modelo_rf,
    param_grid = hiperparametros_rf,
    cv         = configuraciones,
    scoring    = "r2",
    n_jobs     = -1,
    verbose    = 1
)
modelos_rf.fit(X_train_prep, y_train)
```

```
Fitting 5 folds for each of 432 candidates, totalling 2160 fits
```

Out[204]:



In [205]:

```
#MEJORES PARAMETROS ENCONTRADOS

modelos_rf.best_params_
```

Out[205]:

```
{'max_depth': 4,
 'max_features': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 400}
```

In [206]:

```
#PREDICCIONES EN CONJUNTO DE ENTRENAMIENTO Y TEST

y_train_predrf = modelos_rf.predict(X_train_prep)
y_test_predrf = modelos_rf.predict(X_test_prep)
```

MODELO XGBoost

In [207]:

```
hiperparametros_xgb={
    'objective': ['reg:squarederror'],  # función objetivo para regresión
    'eval_metric': ['rmse'],             # métrica de evaluación: raíz del error cuadrát
ico medio
    'max_depth': np.arange(2, 3, 1),                    # profundidad máxima del árbol
    'learning_rate': [0.1],              # tasa de aprendizaje
    'n_estimators': np.arange(300, 401, 20),                # número de árboles en el mo
delo
```

```
    'min_child_weight': np.arange(1, 12, 1),
    #'subsample': np.arange(0.1, 1.1, 0.1),
}
```

In [208]:

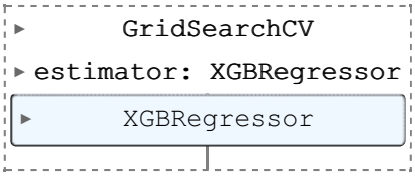
```
from xgboost import XGBRegressor

configuraciones = KFold(n_splits=5, shuffle=True, random_state=2024)
modelo_xgb = XGBRegressor(random_state=2024, objective=['reg:squarederror'])

modelos_xgb = GridSearchCV(
    estimator=modelo_xgb,
    param_grid=hiperparametros_xgb,
    cv=configuraciones,
    scoring='r2',
    n_jobs=-1,
    verbose=1
)
```

In [209]:

```
modelos_xgb.fit(X_train_prep, y_train)
```

```
Fitting 5 folds for each of 66 candidates, totalling 330 fits
```

Out[209]:



In [210]:

```
#MEJORES PARAMETROS ENCONTRADOS

modelos_xgb.best_params_
```

Out[210]:

```
{'eval_metric': 'rmse',
 'learning_rate': 0.1,
 'max_depth': 2,
 'min_child_weight': 2,
 'n_estimators': 380,
 'objective': 'reg:squarederror'}
```

In [211]:

```
#PREDICCIONES SET DE ENTRENAMIENTO Y TEST

y_train_predxgb = modelos_xgb.best_estimator_.predict(X_train_prep)

y_test_predxgb = modelos_xgb.best_estimator_.predict(X_test_prep)
```

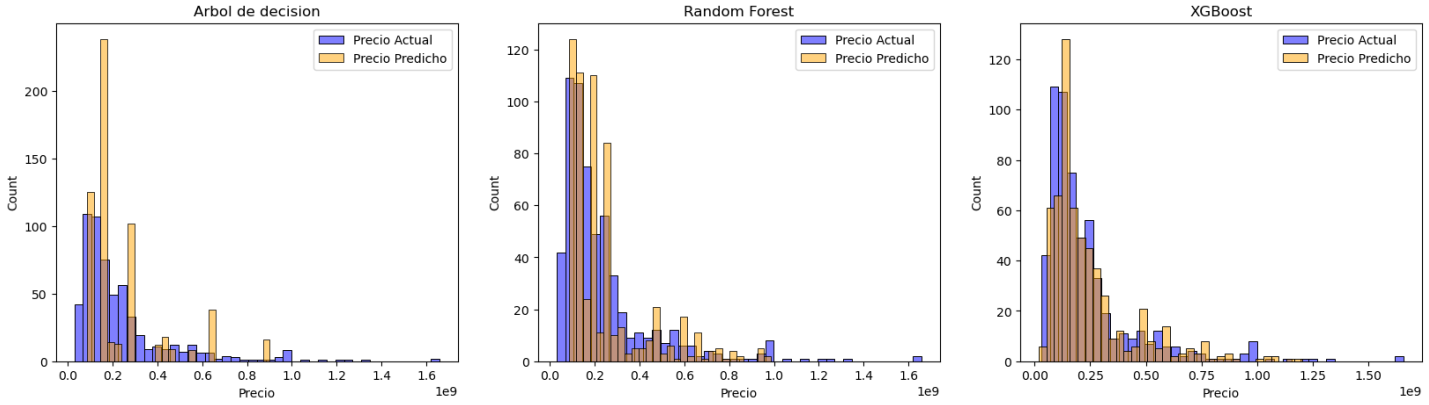
EVALUACION MODELOS

Para la evaluación de los modelo se utilizarán las métricas de R2 para ver que tan bien se ajusta el modelo a los datos observados y los errores de predicción calculando: MSE, RMSE y MAE

In [221]:

```
#GRAFICO DE DISTRIBUCIÓN DE LOS VALORES ACTUALES Y LOS VALORES PREDICHOS

fig, ax = plt.subplots(1, 3, figsize=(20, 5))

# Arbol de decision
```

```
sns.histplot(y_test, ax=ax[0], color='blue', alpha=0.5)
sns.histplot(y_test_predar, ax=ax[0], color='orange', alpha=0.5)

# Random Forest
sns.histplot(y_test, ax=ax[1], color='blue', alpha=0.5)
sns.histplot(y_test_predrf, ax=ax[1], color='orange', alpha=0.5)

# XGBoost
sns.histplot(y_test, ax=ax[2], color='blue', alpha=0.5)
sns.histplot(y_test_predxgb, ax=ax[2], color='orange', alpha=0.5)


ax[0].legend(['Precio Actual', 'Precio Predicho'])
ax[1].legend(['Precio Actual', 'Precio Predicho'])
ax[2].legend(['Precio Actual', 'Precio Predicho'])

ax[0].set_title('Arbol de decision')
ax[1].set_title('Random Forest')
ax[2].set_title('XGBoost')

plt.show()
```



In [241]:

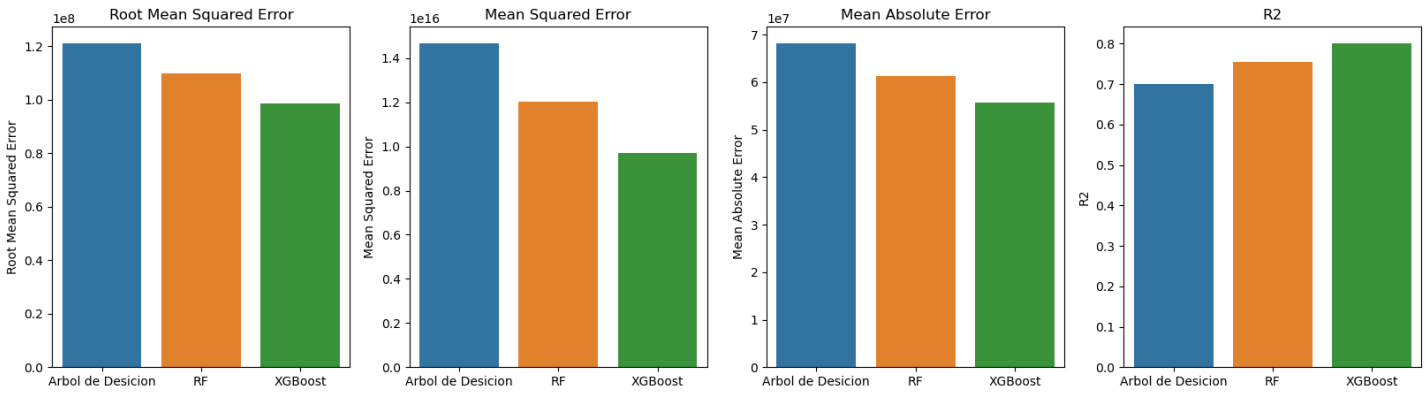
```
#CALCULO DE METRICAS PARA LA COMPARACION DE LOS MODELOS


from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score



print("Arbol de regresión")
print("Mean Squared Error: ", mean_squared_error(y_test, y_test_predar))
print("Root Mean Squared Error: ", mean_squared_error(y_test, y_test_predar, squared=Fals
e))
print("Mean Absolute Error: ", mean_absolute_error(y_test, y_test_predar))
print("R2 Score: ", r2_score(y_test, y_test_predar))
print("\n")
print("Random Forest Regressor")
print("Mean Squared Error: ", mean_squared_error(y_test, y_test_predrf))
print("Root Mean Squared Error: ", mean_squared_error(y_test, y_test_predrf, squared=Fals
e))
print("Mean Absolute Error: ", mean_absolute_error(y_test, y_test_predrf))
print("R2 Score: ", r2_score(y_test, y_test_predrf))
print("\n")
print("XGBoost")
print("Mean Squared Error: ", mean_squared_error(y_test, y_test_predxgb))
print("Root Mean Squared Error: ", mean_squared_error(y_test, y_test_predxgb, squared=Fal
se))
print("Mean Absolute Error: ", mean_absolute_error(y_test, y_test_predxgb))
print("R2 Score: ", r2_score(y_test, y_test_predxgb))
```

```
Arbol de regresión
Mean Squared Error:  1.4687445639852108e+16
Root Mean Squared Error:  121191772.16235477
Mean Absolute Error:  68250839.70057805
R2 Score:  0.6992869692191291
```

```
Random Forest Regressor
Mean Squared Error:  1.2029039703099012e+16
Root Mean Squared Error:  109676978.91125107
Mean Absolute Error:  61311435.38698953
R2 Score:  0.7537155830087039


XGBoost
Mean Squared Error:  9699080534608636.0
Root Mean Squared Error:  98483910.029043
Mean Absolute Error:  55638013.67959528
R2 Score:  0.8014195269301245
```

In [246]:

```
modelos = ['Arbol de Desicion', 'RF', 'XGBoost']
rmse= [mean_squared_error(y_test, y_test_predar, squared=False),mean_squared_error(y_test
, y_test_predrf, squared=False), mean_squared_error(y_test, y_test_predxgb, squared=Fals
e)]
mse = [mean_squared_error(y_test, y_test_predar), mean_squared_error(y_test, y_test_predr
f), mean_squared_error(y_test, y_test_predxgb)]
mae = [mean_absolute_error(y_test, y_test_predar), mean_absolute_error(y_test, y_test_pre
drf), mean_absolute_error(y_test, y_test_predxgb)]
r2 = [r2_score(y_test, y_test_predar),r2_score(y_test, y_test_predrf),r2_score(y_test, y
_test_predxgb)]

fig, axs = plt.subplots(1, 4, figsize=(20, 5))

# GRAFICO COMPARACION DE ROOT MEAN SQUARED ERROR
axs[0].set_ylabel('Root Mean Squared Error')
axs[0].set_title('Root Mean Squared Error')

# GRAFICO COMPARACION MEAN SQUARED ERROR
sns.barplot(x=modelos, y=mse, ax=axs[1])
axs[1].set_ylabel('Mean Squared Error')
axs[1].set_title('Mean Squared Error')

# GRAFICO COMPARACION MEAN ABSOLUTE ERROR
sns.barplot(x=modelos, y=mae, ax=axs[2])
axs[2].set_ylabel('Mean Absolute Error')
axs[2].set_title('Mean Absolute Error')

# GRAFICO COMPARACION R2
sns.barplot(x=modelos, y=r2, ax=axs[3])
axs[3].set_ylabel('R2')
axs[3].set_title('R2')

plt.show()
```



CONCLUSION

Del análisis exploratorio de variables se observa que ninguna de las variables numéricas posee un distribucion normal siento todas asimétricas a la derecha. Esto se refleja en valores "atípicos" en el extremo superior de los datos, también podemos ver cierto desbalance con la cantidad de datos por comunas ya que hay 2 en particular que concentran mas de la mitad de los datos totales por lo que el mercado inmobiliario es mayor en ellas (Concepción y San Pedro de la Paz) y finalmente la variable propiedad se encuentra dispuesta casi de manera uniforme entre Casas y Departamentos que estan en venta. Las variables Dormitorios y Baños tienen una mayor correlación positiva con la variable respuesta Precio y la variable Tamaño en m2 tiene una correlación bastante débil pero positiva igualmente. Es importante mencionar que los datos son bastante "generales" y para tomar decisiones o llegar a mas concluciones sería importante tener datos mas específicos de las viviendas como por ejemplo una dirección mas precisa dentro de cada comuna, aun que en un principio existía una variable llamada Dirección esta no poseía en la mayoría de los casos un formato adecuado para ser trabajada, también podría ser interesante tener detalles de materiales de construcción, si posee estacionamiento o no, etc.

En cuanto a los modelos de aprendizaje automático se entrenó un Arbol de Decision, un Random Forest y un XGBoost. Como se mencionó anteriormente se calcularon las metricas de R2, MSE, RMSE y MAE para ser comparadas entre los modelos, en este caso en particular la elección del modelo es sencilla ya que se condice que el modelo que tiene un mayor R2 también presenta los errores mas bajos siendo el XGBoost.