proyecto2

March 9, 2025

1 Proyecto 2 “House Prices: Advanced Regression Techniques”

1.1 1. Importación de librerías y carga de datos

En esta sección importaremos las librerías necesarias y cargaremos el dataset de entrenamiento y prueba.

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier,␣
 ↪plot_tree
from sklearn.metrics import mean_squared_error, r2_score

# Para ver las gráficas de matplotlib "inline" en jupyter
%matplotlib inline

# Carga de los datos (modifica la ruta según tu entorno)
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')  # Opcional, si necesitas el dataset de prueba␣
 ↪para algún análisis adicional

# Dimensiones del dataset
print("Dimensiones del dataset de entrenamiento:", train.shape)
train.head()
```

```
Dimensiones del dataset de entrenamiento: (1460, 81)
```

[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	\
0	1	60	RL	65.0	8450	Pave	NaN	Reg	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	

```
4   5          60       RL         84.0    14260   Pave   NaN      IR1

  LandContour Utilities  … PoolArea PoolQC Fence MiscFeature MiscVal MoSold  \
0         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
1         Lvl    AllPub  …        0    NaN   NaN         NaN       0      5
2         Lvl    AllPub  …        0    NaN   NaN         NaN       0      9
3         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
4         Lvl    AllPub  …        0    NaN   NaN         NaN       0     12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000

[5 rows x 81 columns]
```

1.2 2. Revisión inicial de la estructura de los datos

En esta parte: 1. Observamos el tipo de cada columna (numérica o categórica). 2. Revisamos estadísticas descriptivas básicas de variables numéricas.

[ ]:
```
# Información sobre tipos de datos y valores nulos
train.info()

# Descripción estadística de variables numéricas
train.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
```

```
14  Condition2     1460 non-null   object
15  BldgType       1460 non-null   object
16  HouseStyle     1460 non-null   object
17  OverallQual    1460 non-null   int64
18  OverallCond    1460 non-null   int64
19  YearBuilt      1460 non-null   int64
20  YearRemodAdd   1460 non-null   int64
21  RoofStyle      1460 non-null   object
22  RoofMatl       1460 non-null   object
23  Exterior1st    1460 non-null   object
24  Exterior2nd    1460 non-null   object
25  MasVnrType     588 non-null    object
26  MasVnrArea     1452 non-null   float64
27  ExterQual      1460 non-null   object
28  ExterCond      1460 non-null   object
29  Foundation     1460 non-null   object
30  BsmtQual       1423 non-null   object
31  BsmtCond       1423 non-null   object
32  BsmtExposure   1422 non-null   object
33  BsmtFinType1   1423 non-null   object
34  BsmtFinSF1     1460 non-null   int64
35  BsmtFinType2   1422 non-null   object
36  BsmtFinSF2     1460 non-null   int64
37  BsmtUnfSF      1460 non-null   int64
38  TotalBsmtSF    1460 non-null   int64
39  Heating        1460 non-null   object
40  HeatingQC      1460 non-null   object
41  CentralAir     1460 non-null   object
42  Electrical     1459 non-null   object
43  1stFlrSF       1460 non-null   int64
44  2ndFlrSF       1460 non-null   int64
45  LowQualFinSF   1460 non-null   int64
46  GrLivArea      1460 non-null   int64
47  BsmtFullBath   1460 non-null   int64
48  BsmtHalfBath   1460 non-null   int64
49  FullBath       1460 non-null   int64
50  HalfBath       1460 non-null   int64
51  BedroomAbvGr   1460 non-null   int64
52  KitchenAbvGr   1460 non-null   int64
53  KitchenQual    1460 non-null   object
54  TotRmsAbvGrd   1460 non-null   int64
55  Functional     1460 non-null   object
56  Fireplaces     1460 non-null   int64
57  FireplaceQu    770 non-null    object
58  GarageType     1379 non-null   object
59  GarageYrBlt    1379 non-null   float64
60  GarageFinish   1379 non-null   object
61  GarageCars     1460 non-null   int64
```

```
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

[ ]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	\
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	
std	421.610009	42.300571	24.284752	9981.264932	1.382997	
min	1.000000	20.000000	21.000000	1300.000000	1.000000	
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	

	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	…	\
count	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	…	
mean	5.575342	1971.267808	1984.865753	103.685262	443.639726	…	
std	1.112799	30.202904	20.645407	181.066207	456.098091	…	
min	1.000000	1872.000000	1950.000000	0.000000	0.000000	…	
25%	5.000000	1954.000000	1967.000000	0.000000	0.000000	…	
50%	5.000000	1973.000000	1994.000000	0.000000	383.500000	…	
75%	6.000000	2000.000000	2004.000000	166.000000	712.250000	…	
max	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	…	

	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	\
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	
mean	94.244521	46.660274	21.954110	3.409589	15.060959	
std	125.338794	66.256028	61.119149	29.317331	55.757415	
min	0.000000	0.000000	0.000000	0.000000	0.000000	

```
25%       0.000000     0.000000       0.000000     0.000000     0.000000
50%       0.000000    25.000000       0.000000     0.000000     0.000000
75%     168.000000    68.000000       0.000000     0.000000     0.000000
max     857.000000   547.000000     552.000000   508.000000   480.000000

          PoolArea       MiscVal       MoSold       YrSold      SalePrice
count  1460.000000   1460.000000  1460.000000  1460.000000    1460.000000
mean      2.758904     43.489041     6.321918  2007.815753  180921.195890
std      40.177307    496.123024     2.703626     1.328095   79442.502883
min       0.000000      0.000000     1.000000  2006.000000   34900.000000
25%       0.000000      0.000000     5.000000  2007.000000  129975.000000
50%       0.000000      0.000000     6.000000  2008.000000  163000.000000
75%       0.000000      0.000000     8.000000  2009.000000  214000.000000
max     738.000000  15500.000000    12.000000  2010.000000  755000.000000

[8 rows x 38 columns]
```

1.3 3. Análisis de valores faltantes

1. Identificamos qué columnas tienen más valores nulos.
2. Evaluamos la proporción de faltantes y decidimos si imputar o eliminar.
3. Revisamos si ciertas variables usan "NA" como categoría válida (ej. "No Garage").

[ ]:
```
total_nulos = train.isnull().sum().sort_values(ascending=False)
porc_nulos = (train.isnull().sum() / train.shape[0]).
 ↪sort_values(ascending=False)

missing_data = pd.concat([total_nulos, porc_nulos], axis=1, keys=['Total',␣
 ↪'Porcentaje'])
missing_data.head(20)  # Muestra las 20 columnas con más valores nulos
```

[ ]:
```
              Total  Porcentaje
PoolQC         1453    0.995205
MiscFeature    1406    0.963014
Alley          1369    0.937671
Fence          1179    0.807534
MasVnrType      872    0.597260
FireplaceQu     690    0.472603
LotFrontage     259    0.177397
GarageYrBlt      81    0.055479
GarageCond       81    0.055479
GarageType       81    0.055479
GarageFinish     81    0.055479
GarageQual       81    0.055479
BsmtFinType2     38    0.026027
BsmtExposure     38    0.026027
BsmtQual         37    0.025342
```

```
BsmtCond          37    0.025342
BsmtFinType1      37    0.025342
MasVnrArea         8    0.005479
Electrical         1    0.000685
Id                 0    0.000000
```

1.4 4. Clasificación de variables

Separaremos las columnas en numéricas y categóricas, para tratarlas de manera distinta en nuestro análisis.

[ ]:
```
numerical_feats = train.select_dtypes(include=[np.number]).columns
categorical_feats = train.select_dtypes(include=['object']).columns

print("Variables numéricas:", numerical_feats)
print("Variables categóricas:", categorical_feats)
```

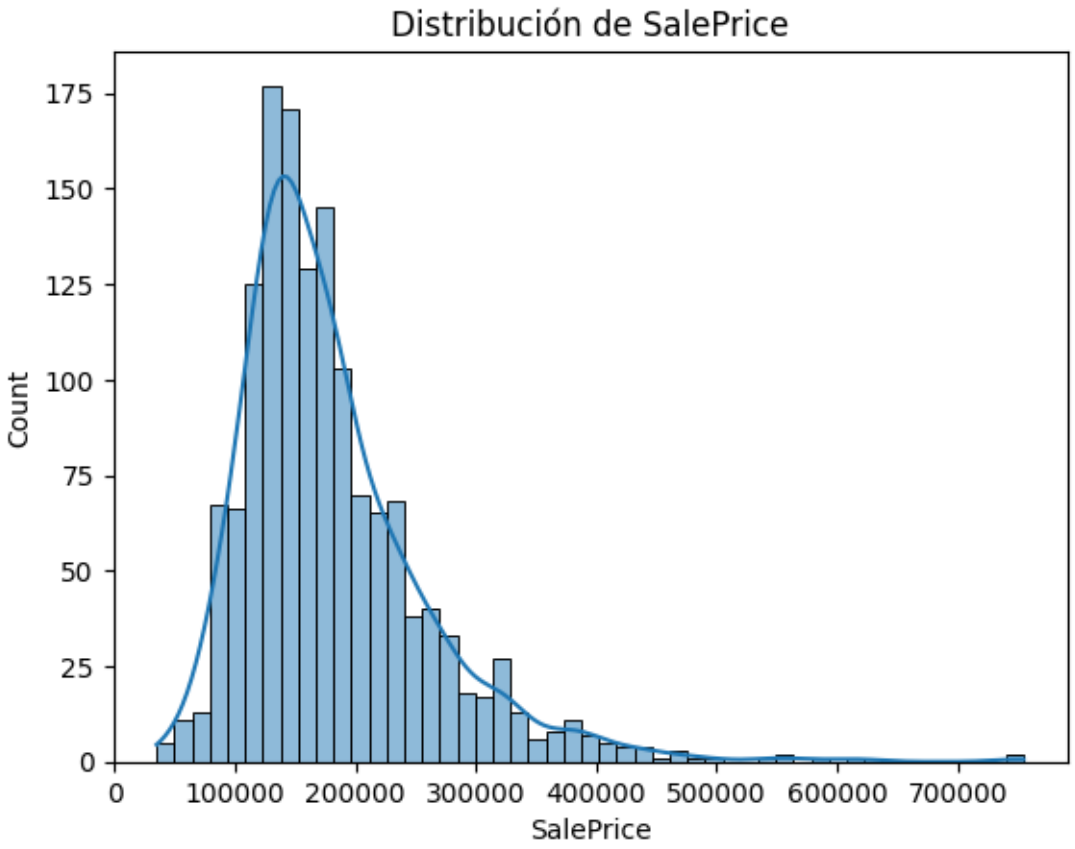
```
Variables numéricas: Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea',
'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
       'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF',
       'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea',
       'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
Variables categóricas: Index(['MSZoning', 'Street', 'Alley', 'LotShape',
'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual',
       'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

1.5 5. Análisis univariante de la variable objetivo (SalePrice)

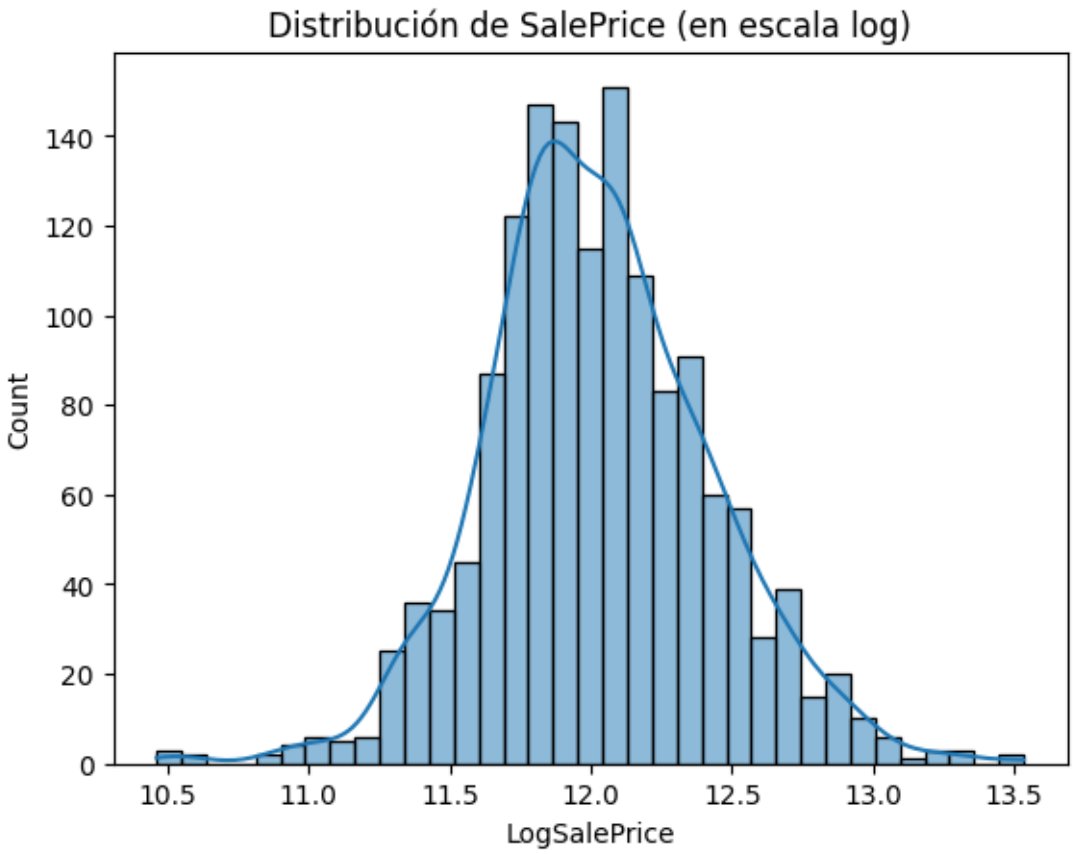
SalePrice es la variable que queremos predecir. Revisamos su distribución y outliers.

[5]:
```
# Histograma y KDE de SalePrice
sns.histplot(train['SalePrice'], kde=True)
plt.title('Distribución de SalePrice')
plt.show()
```

```python
# Estadísticos básicos
print(train['SalePrice'].describe())

# (Opcional) Transformación logarítmica para ver si se acerca más a la normal
train['LogSalePrice'] = np.log(train['SalePrice'])

sns.histplot(train['LogSalePrice'], kde=True)
plt.title('Distribución de SalePrice (en escala log)')
plt.show()
```




```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

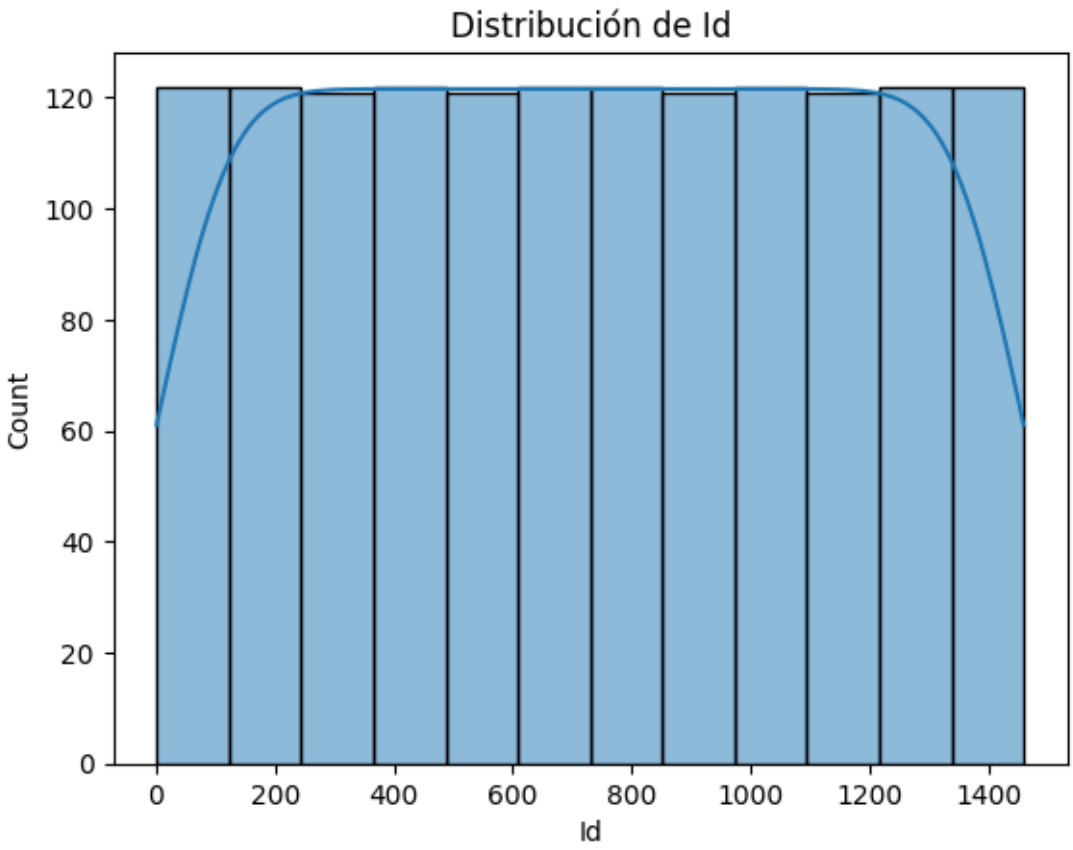


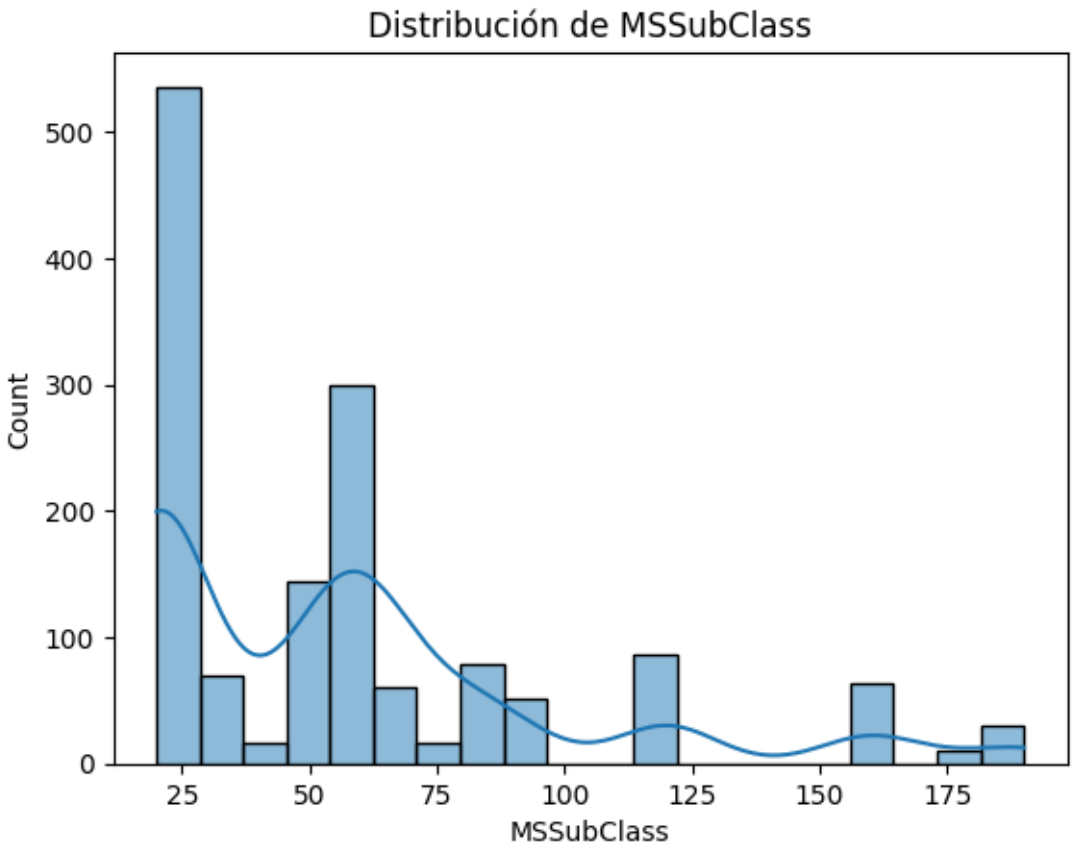
1.6 6. Análisis univariante de las demás variables

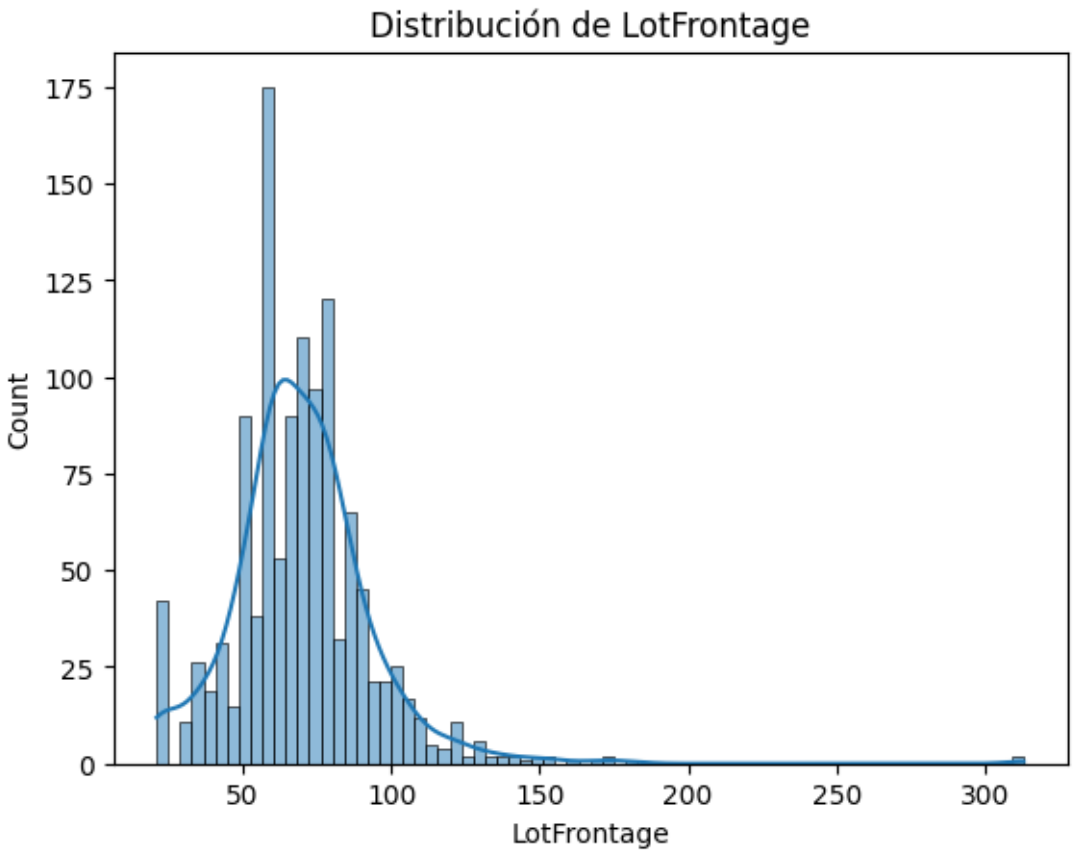
1.6.1 6.1 Variables numéricas

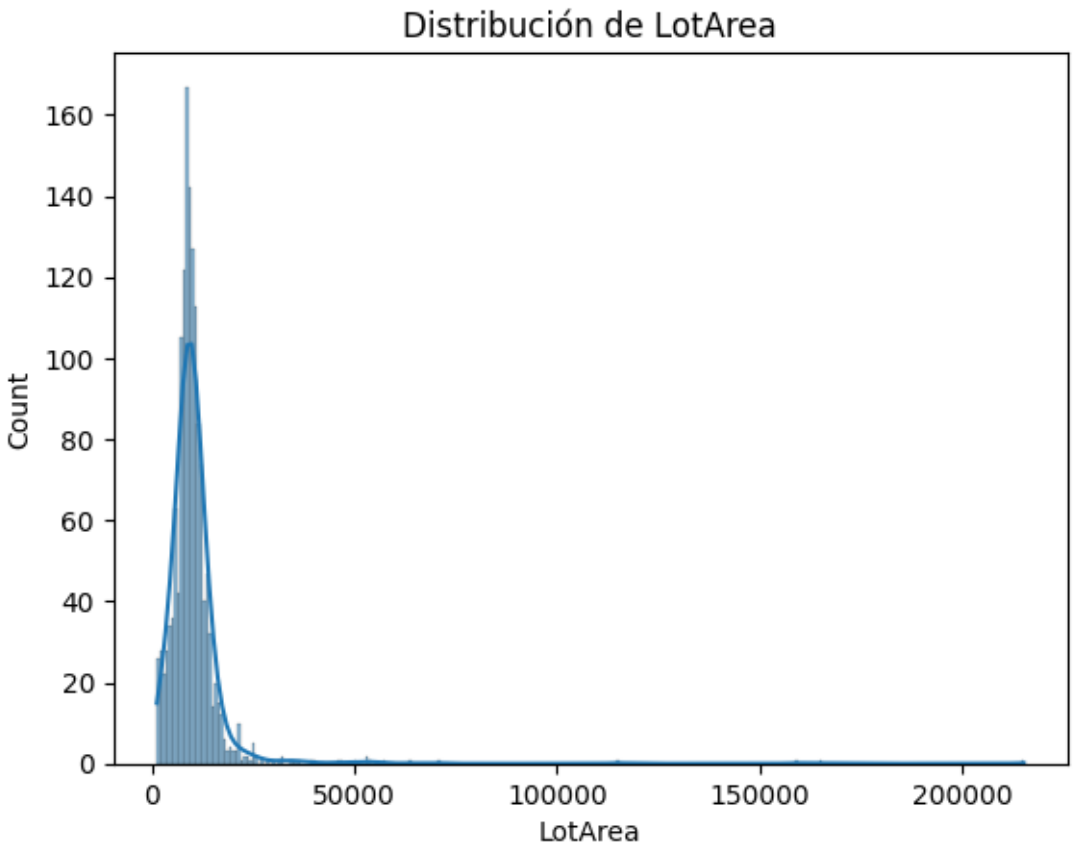
Generamos histogramas y curvas KDE para detectar asimetría, picos y presencia de outliers.

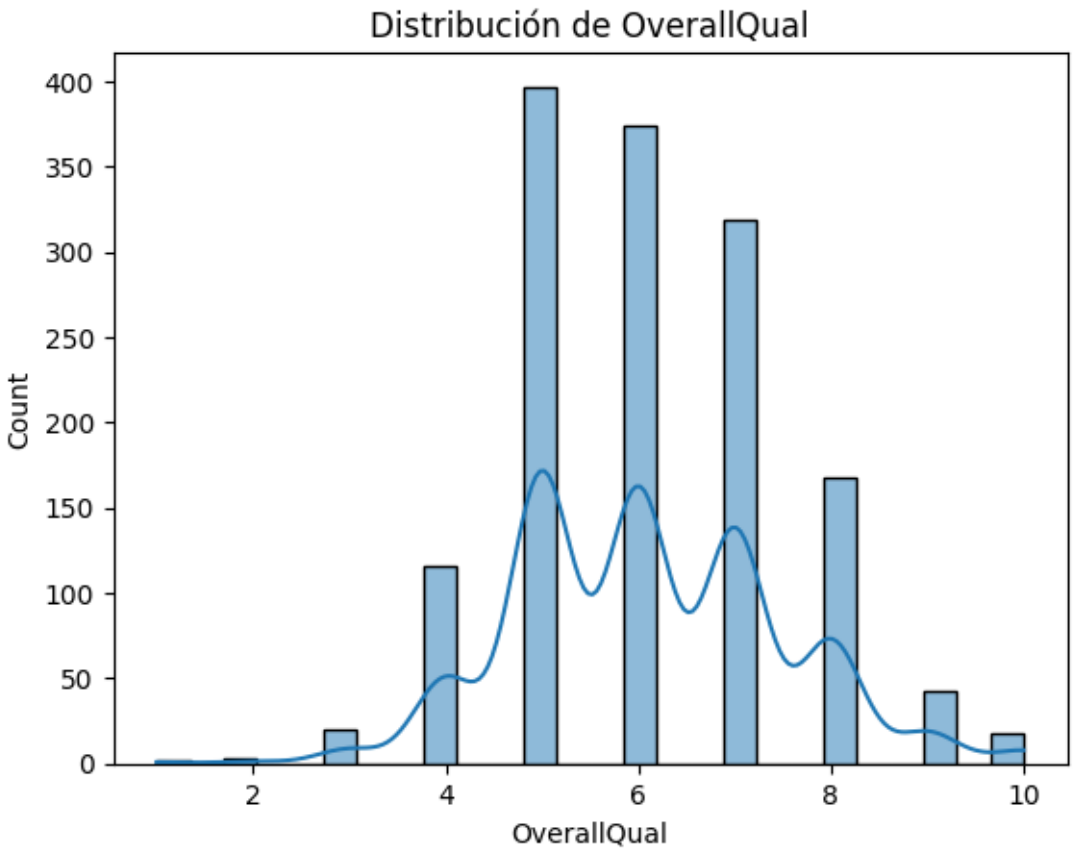
[6]:
```
for col in numerical_feats:
    plt.figure()
    # Eliminamos NaN con dropna()
    sns.histplot(train[col].dropna(), kde=True)
    plt.title(f'Distribución de {col}')
    plt.show()
```

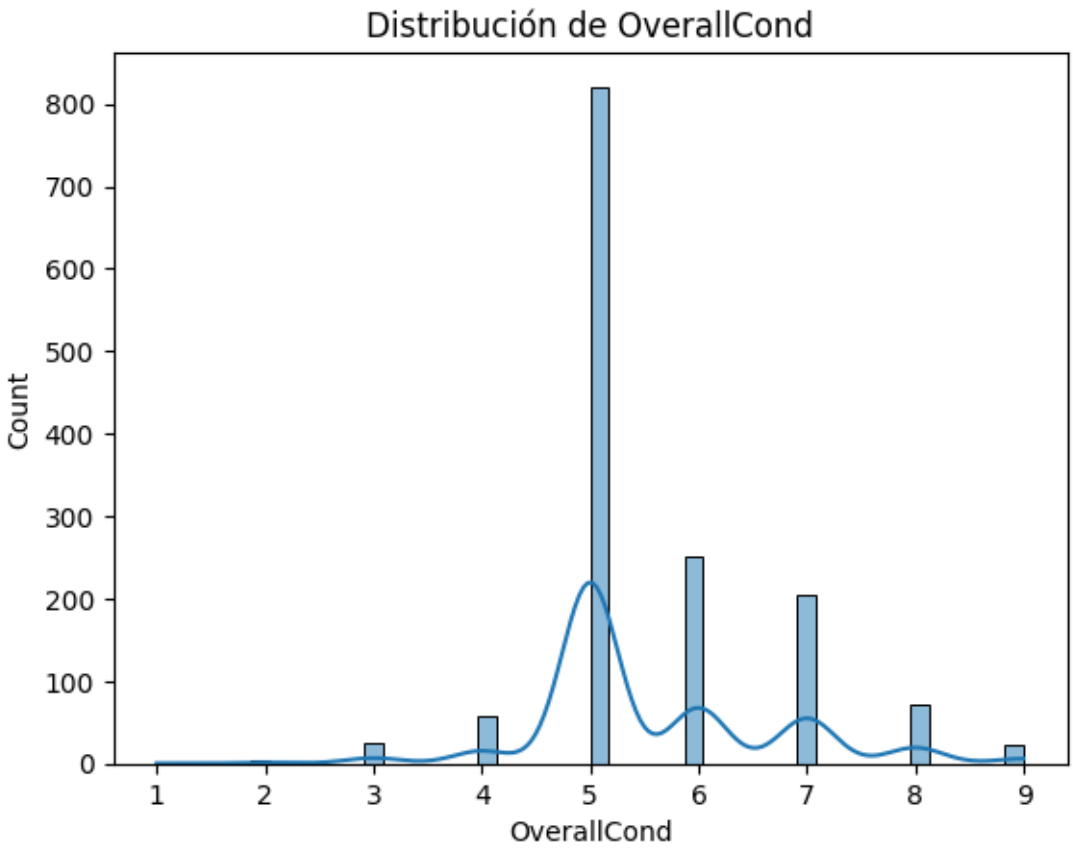

Distribución de Id
Count
Id
0
20
40
60
80
100
120
0
200
400
600
800
1000
1200
1400

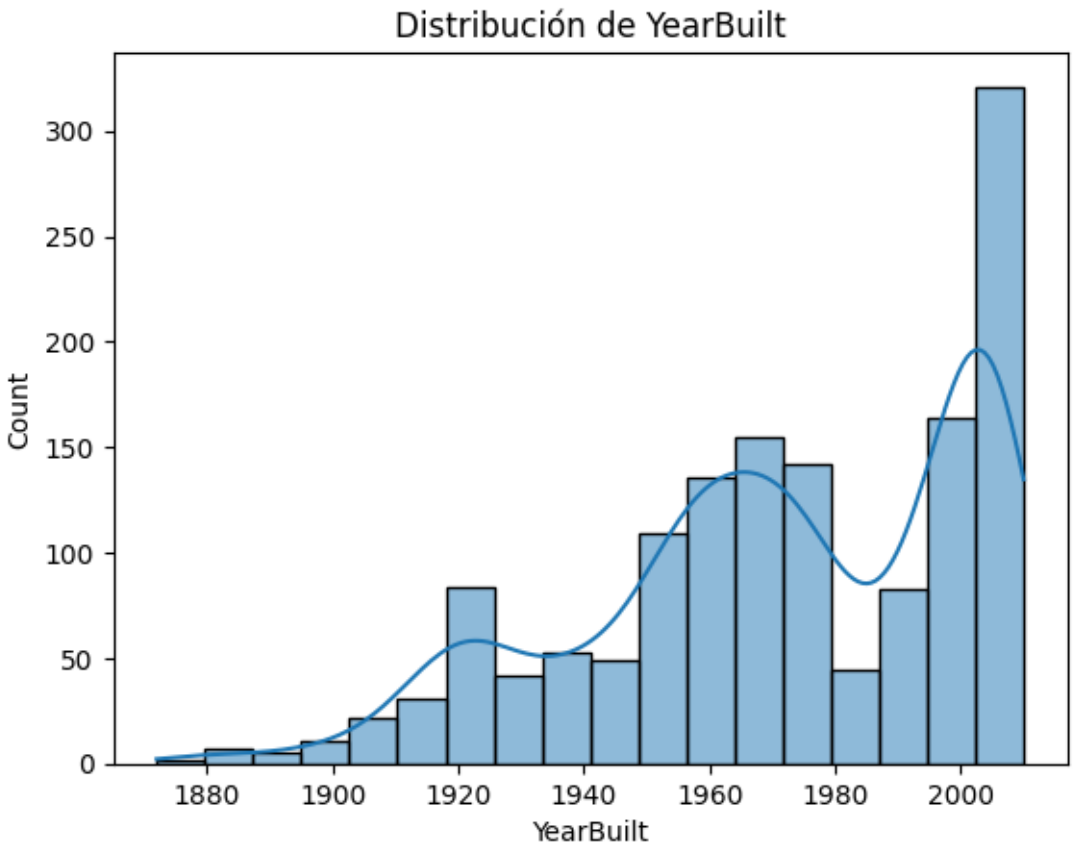

Distribución de MSSubClass
Count
MSSubClass
0
100
200
300
400
500
25
50
75
100
125
150
175

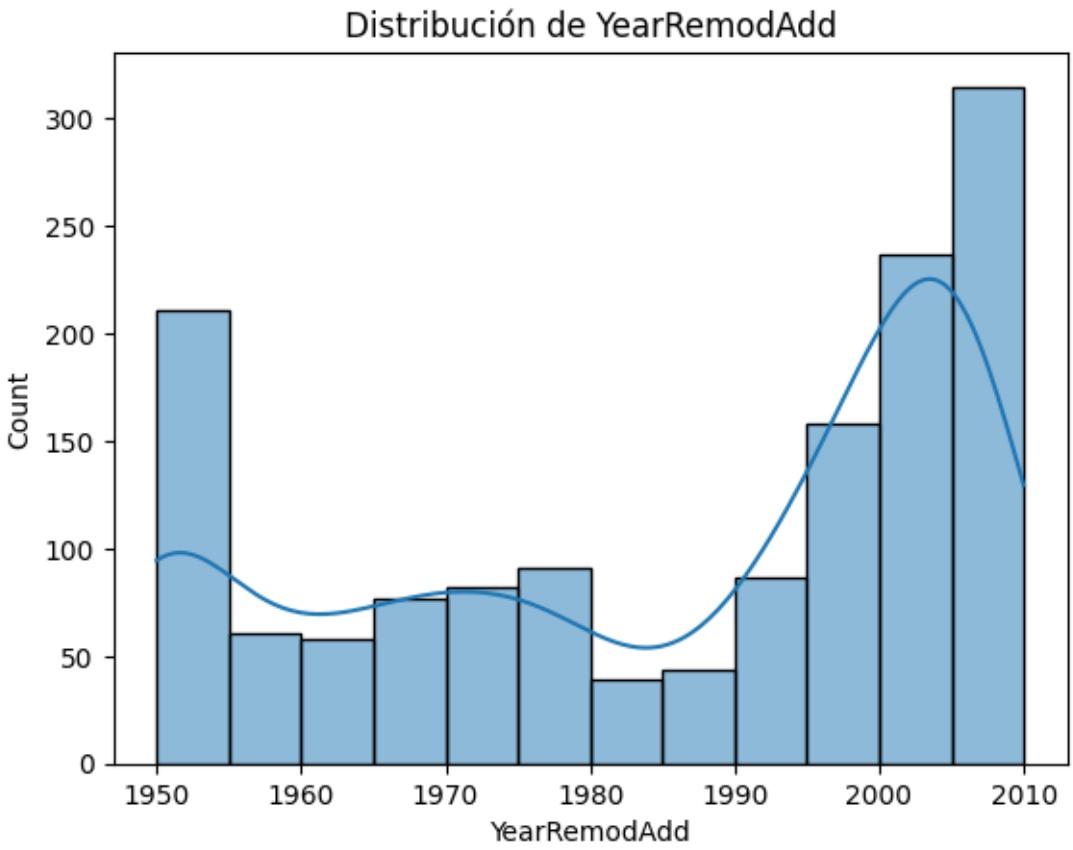

Distribución de LotFrontage
Count
LotFrontage
0
25
50
75
100
125
150
175
50
100
150
200
250
300

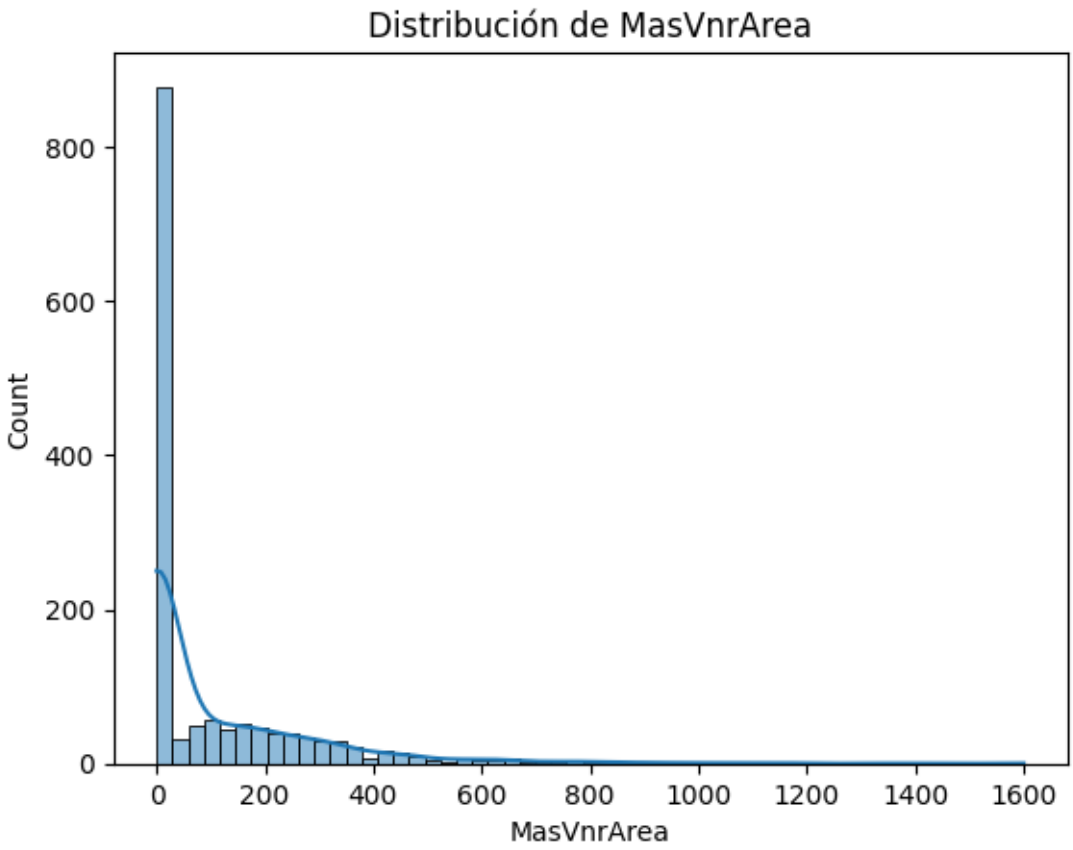

Distribución de LotArea
Count
LotArea
0
20
40
60
80
100
120
140
160
0
50000
100000
150000
200000

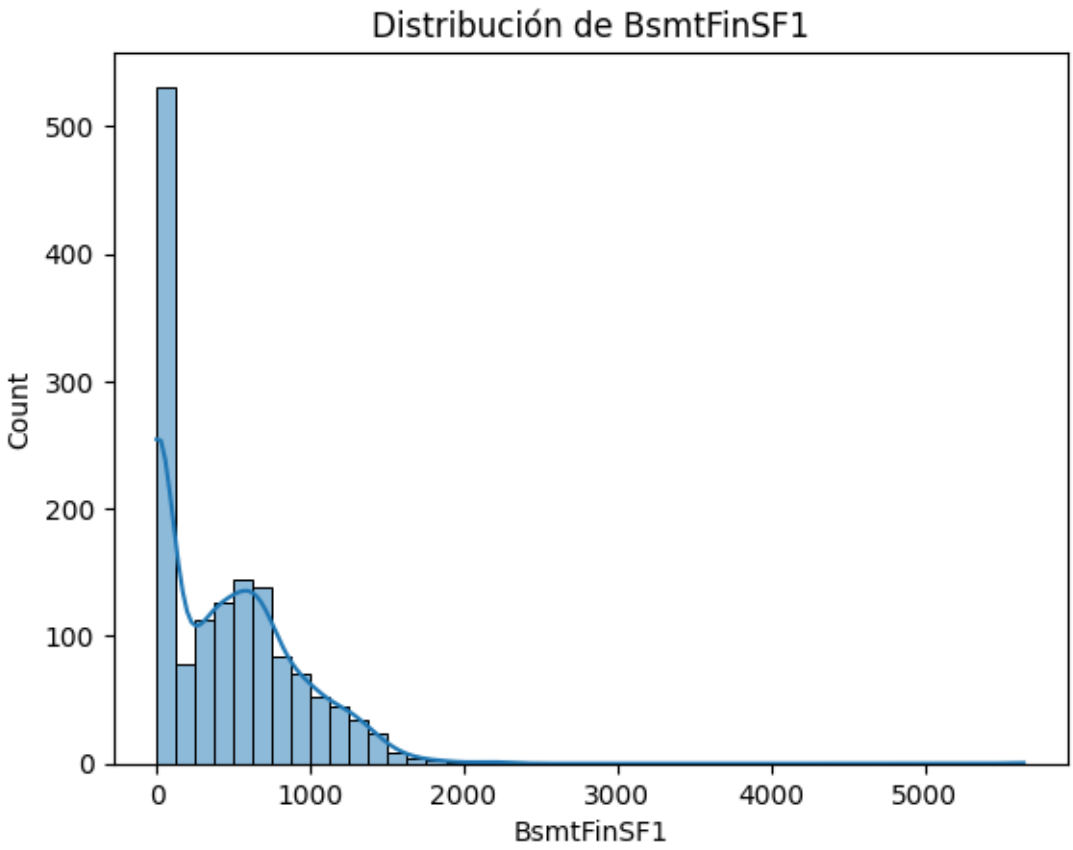

Distribución de OverallQual
Count
OverallQual
0
50
100
150
200
250
300
350
400
2
4
6
8
10

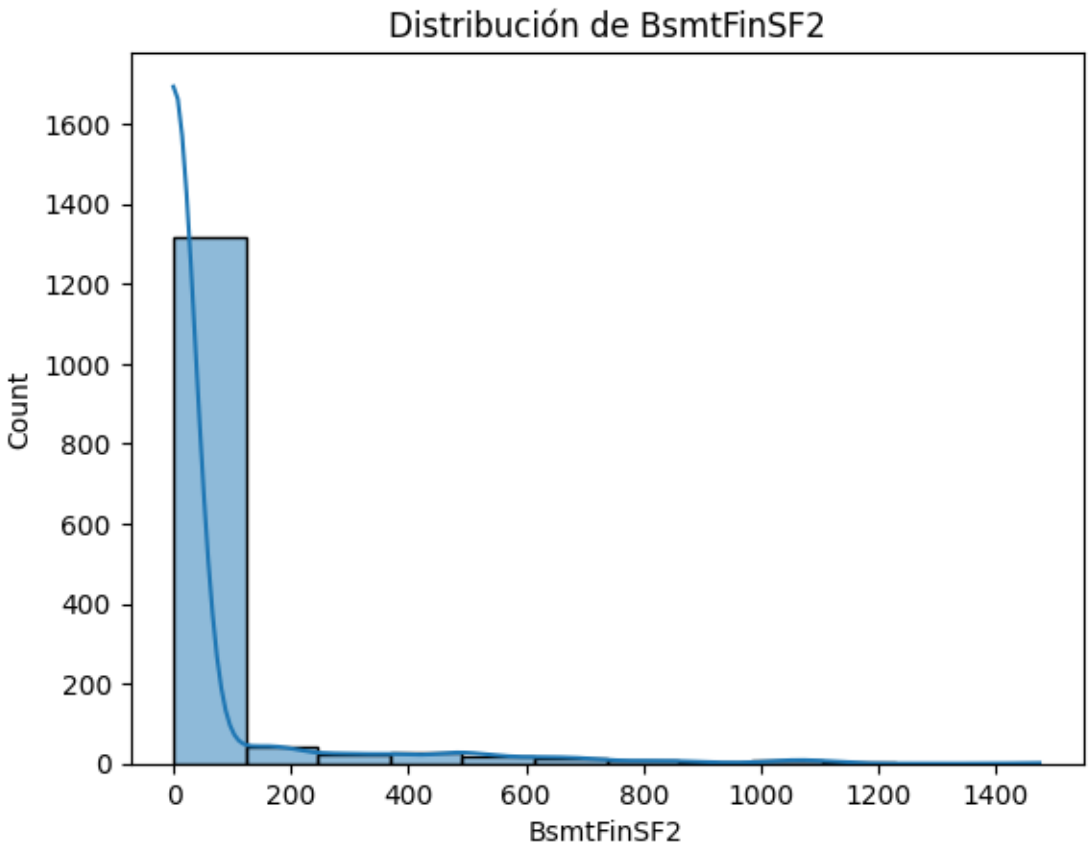

Distribución de OverallCond
Count
OverallCond
0
100
200
300
400
500
600
700
800
1
2
3
4
5
6
7
8
9

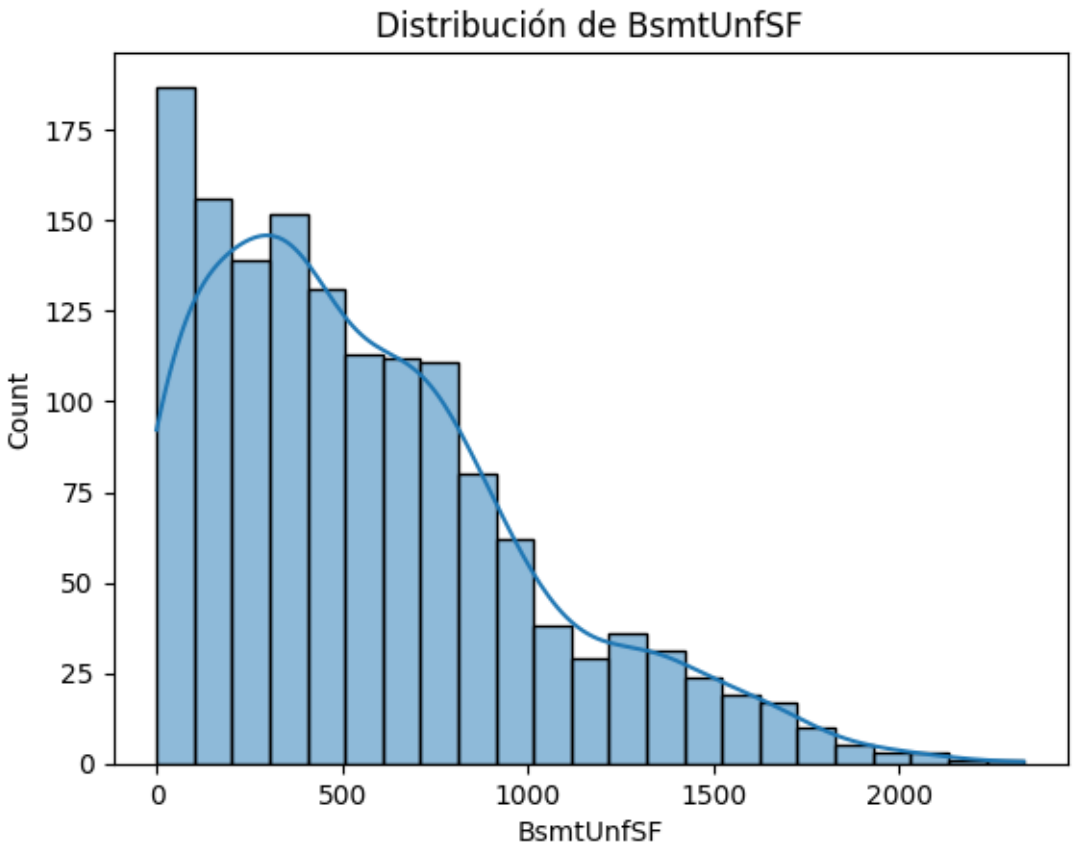

Distribución de YearBuilt
Count
YearBuilt
0
50
100
150
200
250
300
1880
1900
1920
1940
1960
1980
2000

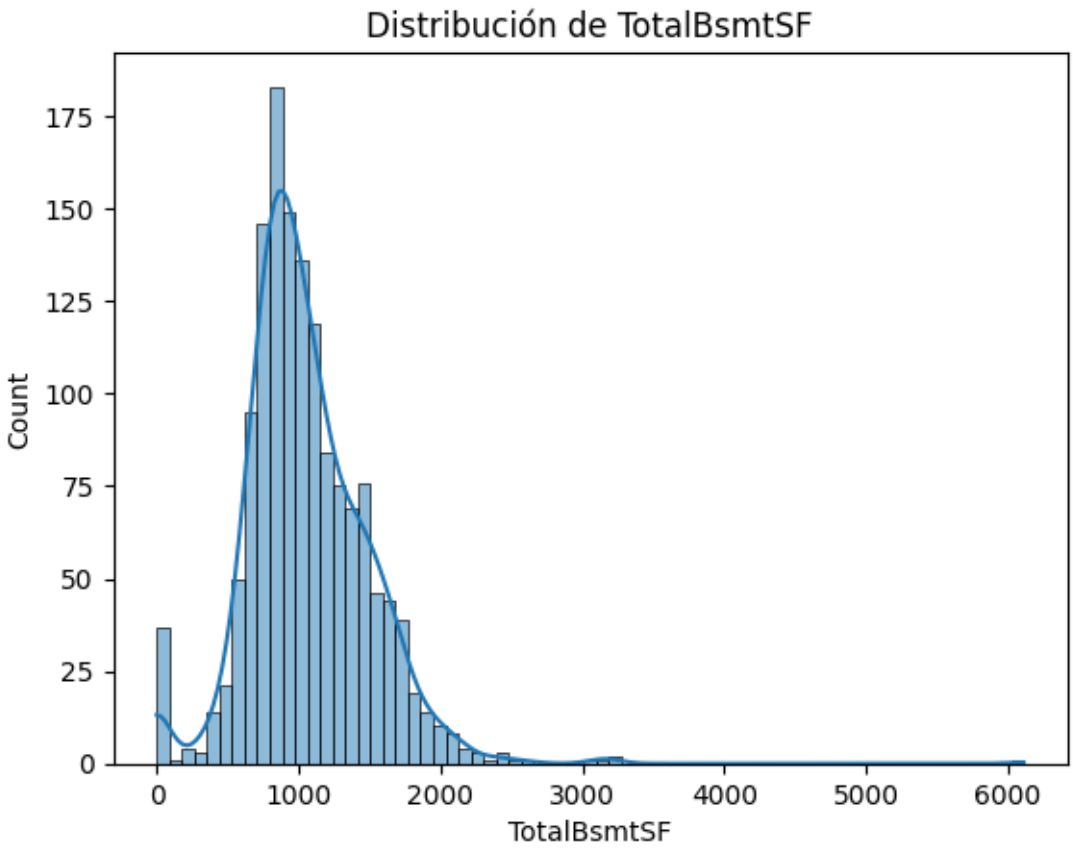

Distribución de YearRemodAdd
Count
YearRemodAdd
0
50
100
150
200
250
300
1950
1960
1970
1980
1990
2000
2010

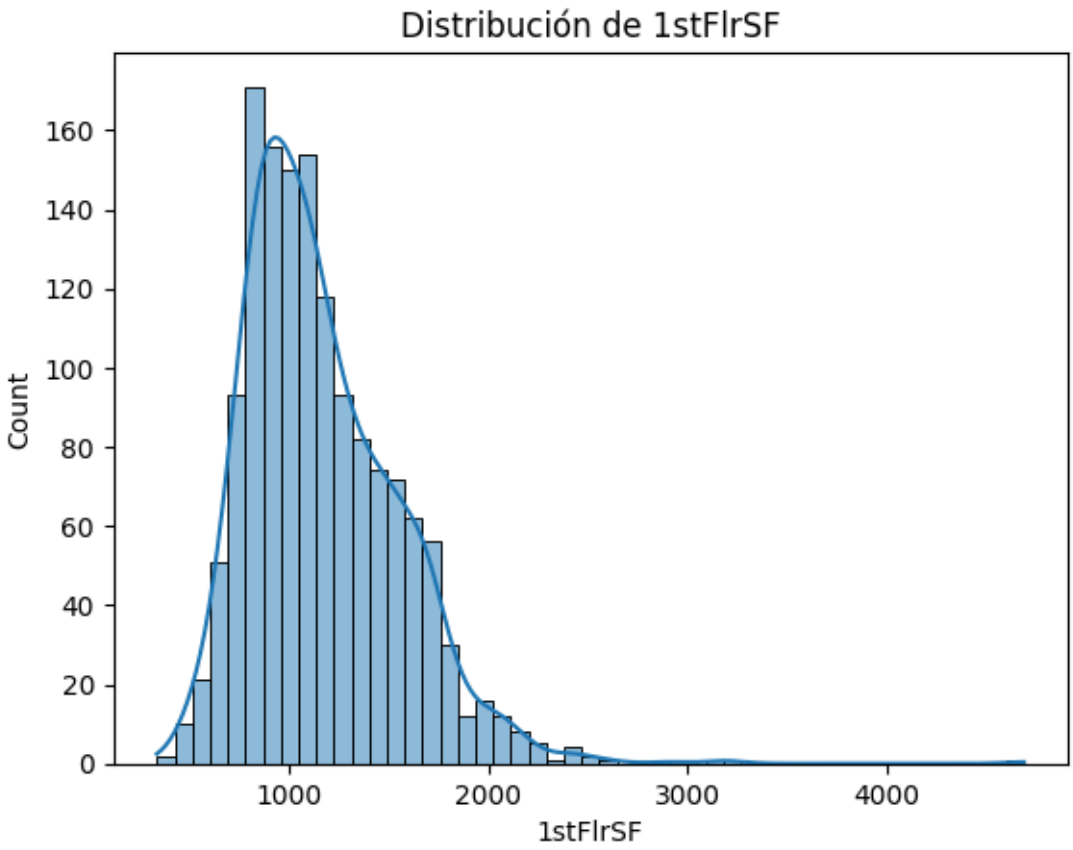

Distribución de MasVnrArea
Count
MasVnrArea
0
200
400
600
800
1000
1200
1400
1600

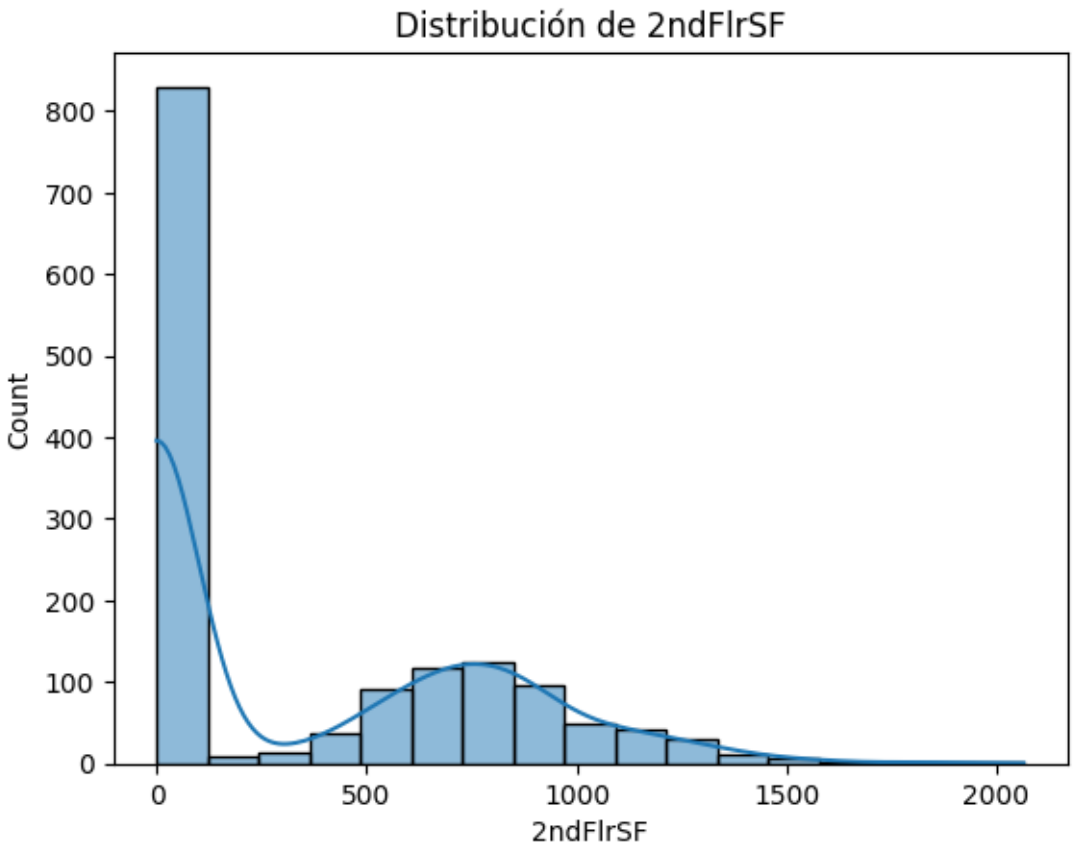

Distribución de BsmtFinSF1
Count
BsmtFinSF1
0
100
200
300
400
500
0
1000
2000
3000
4000
5000

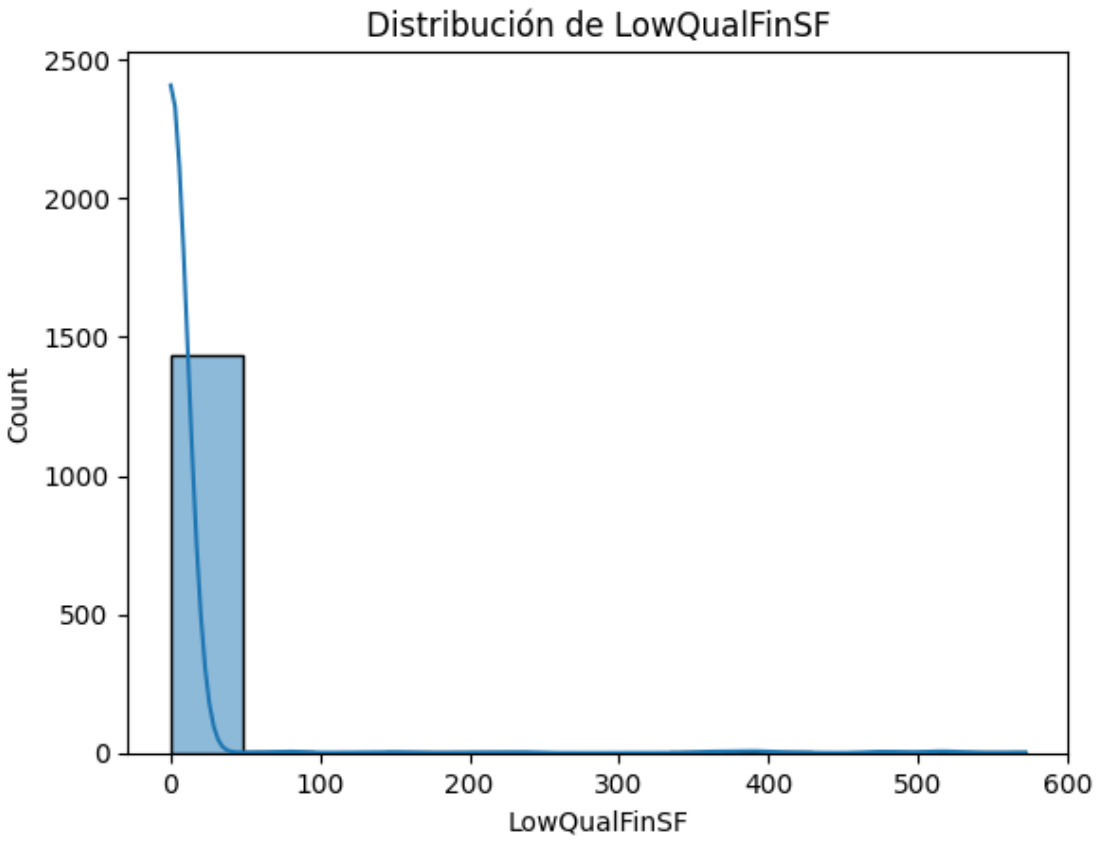

Distribución de BsmtFinSF2
Count
BsmtFinSF2
0
200
400
600
800
1000
1200
1400
1600

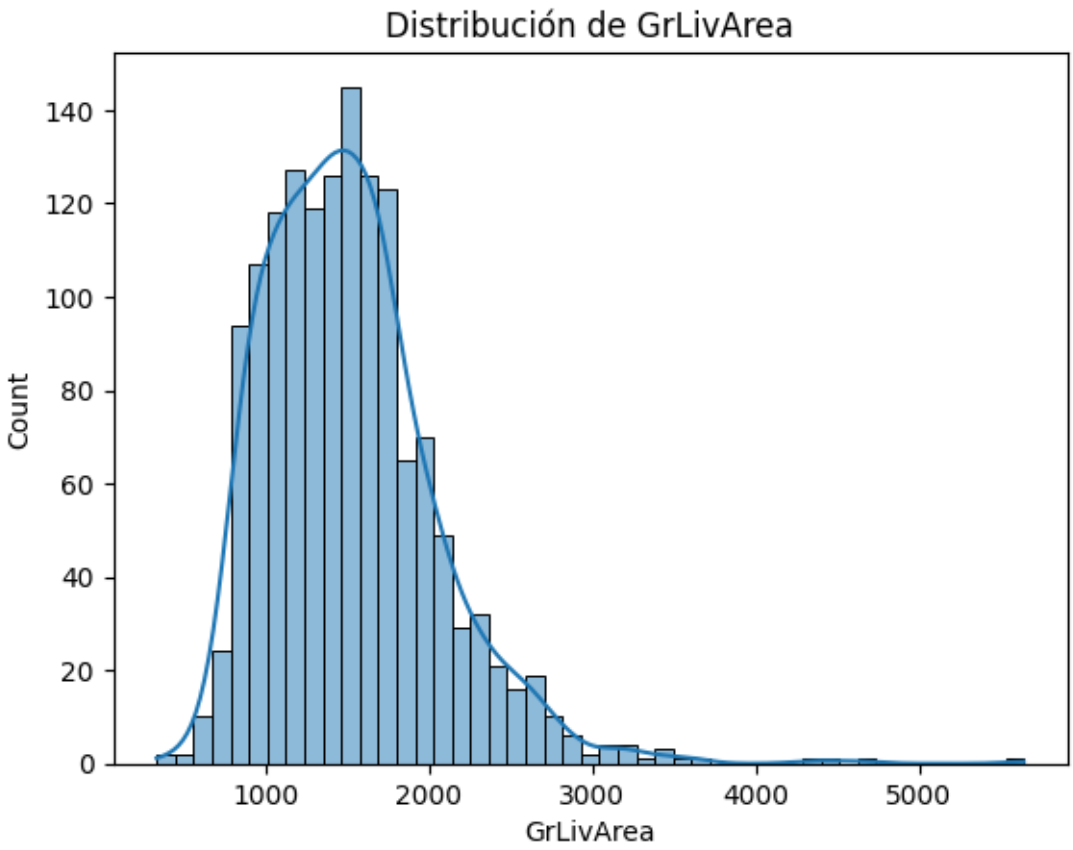

Distribución de BsmtUnfSF
Count
BsmtUnfSF
0
25
50
75
100
125
150
175
0
500
1000
1500
2000

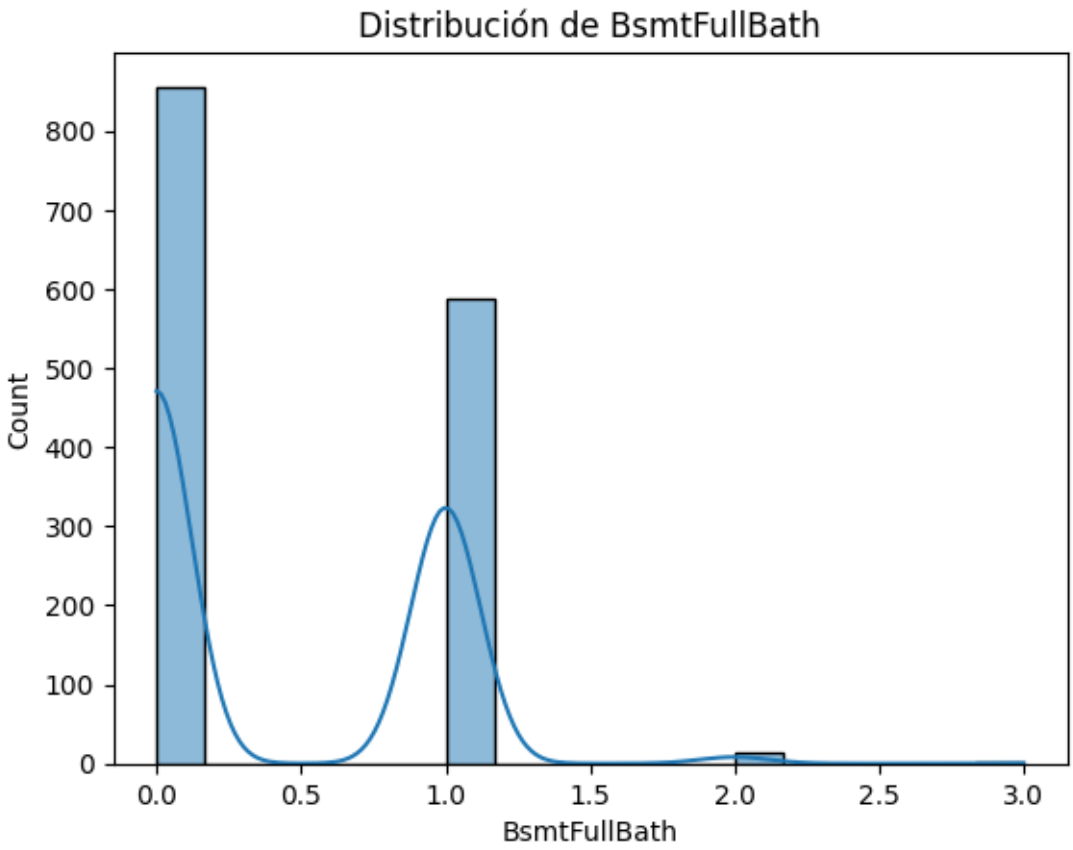

Distribución de TotalBsmtSF
Count
TotalBsmtSF
0
25
50
75
100
125
150
175
0
1000
2000
3000
4000
5000
6000

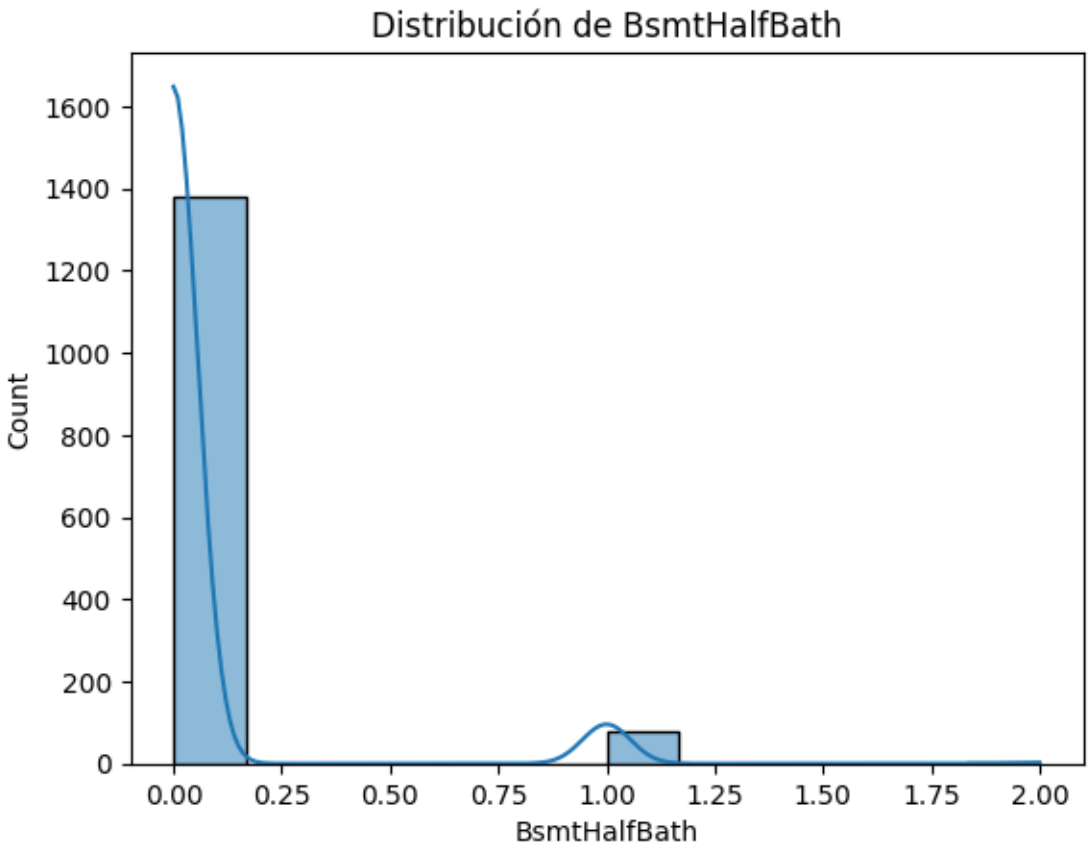

Distribución de 1stFlrSF
Count
0
20
40
60
80
100
120
140
160
1000
2000
3000
4000
1stFlrSF

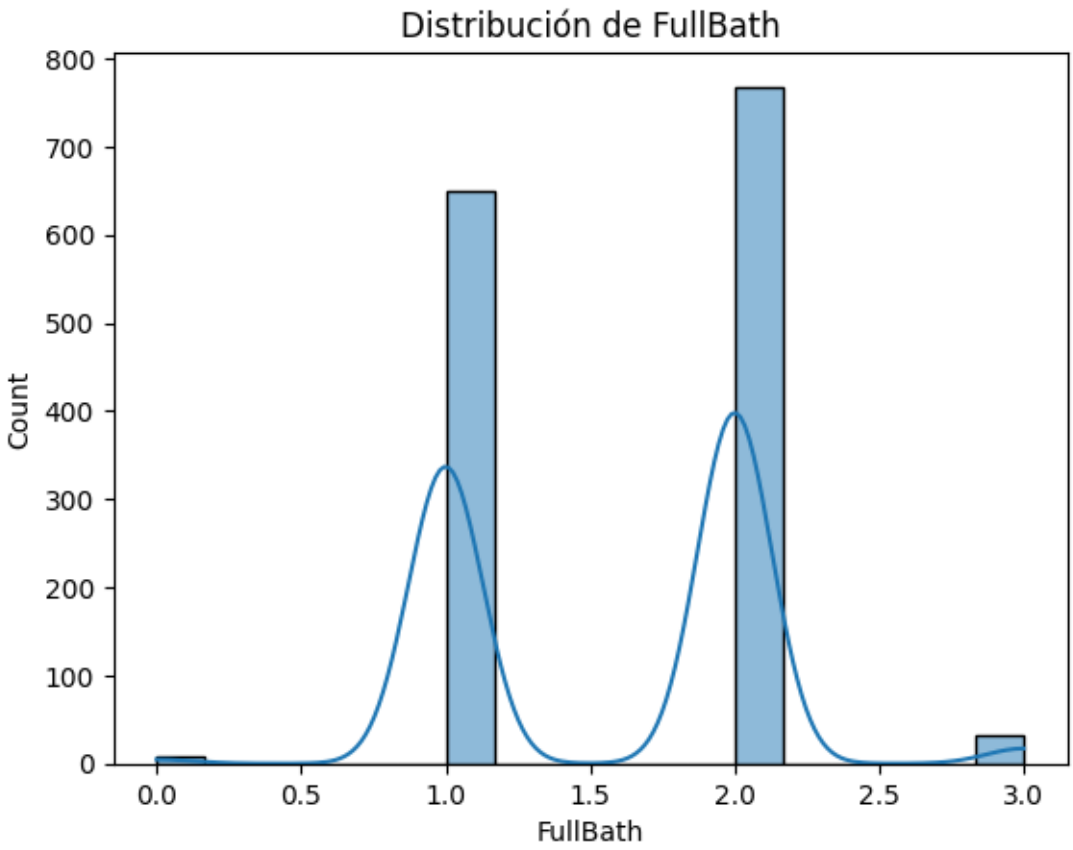

Distribución de 2ndFlrSF
Count
2ndFlrSF
0
100
200
300
400
500
600
700
800
0
500
1000
1500
2000

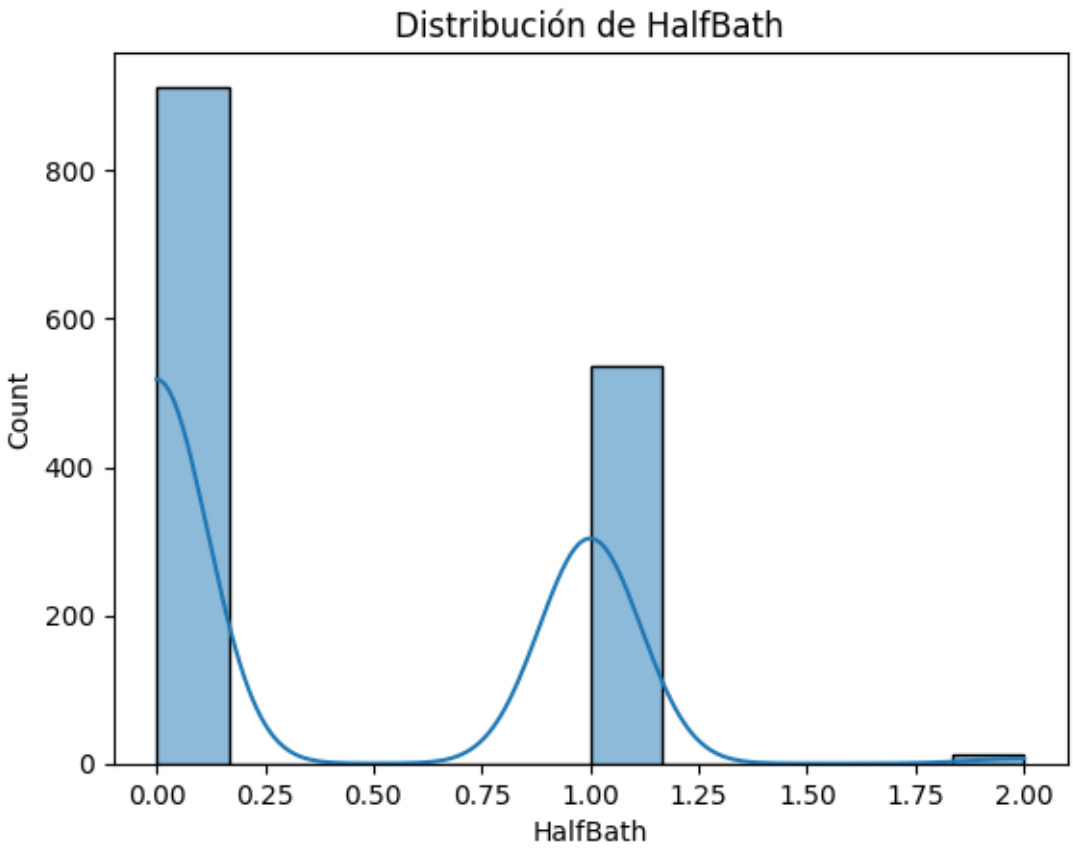

Distribución de LowQualFinSF
2500
2000
1500
1000
500
0
Count
0
100
200
300
400
500
600
LowQualFinSF

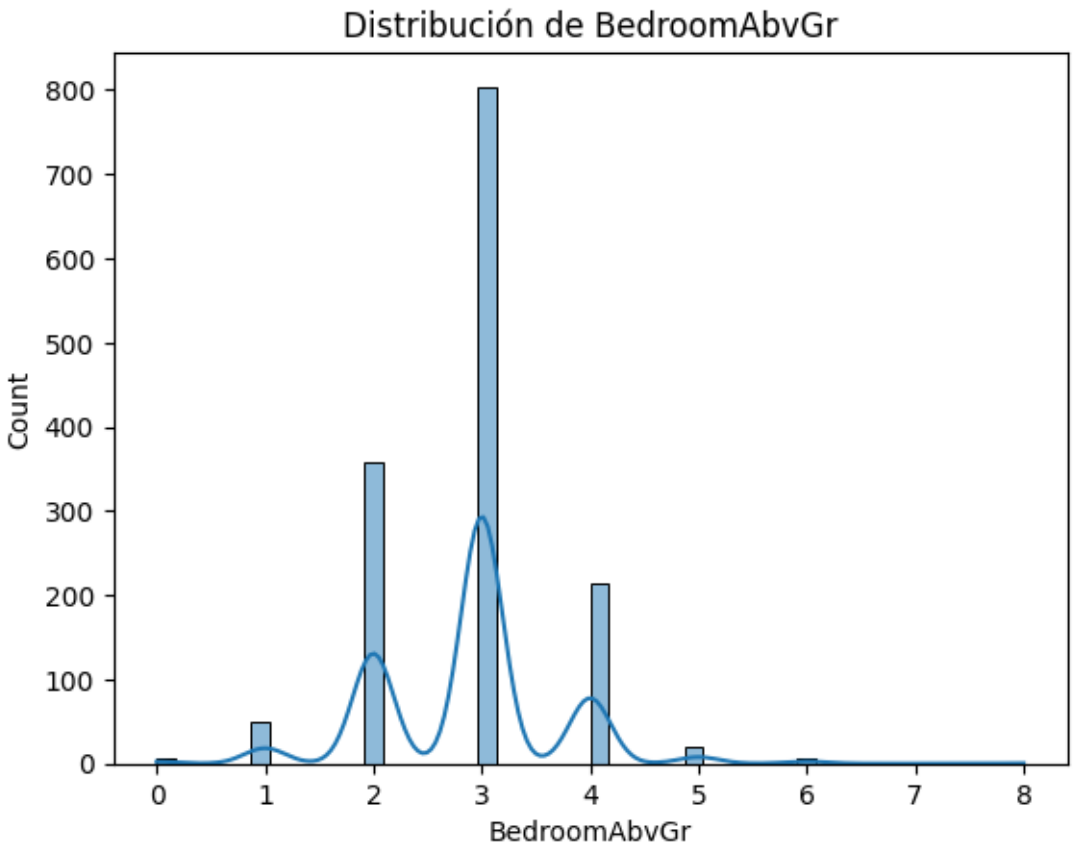

Distribución de GrLivArea
Count
GrLivArea
0
20
40
60
80
100
120
140
1000
2000
3000
4000
5000

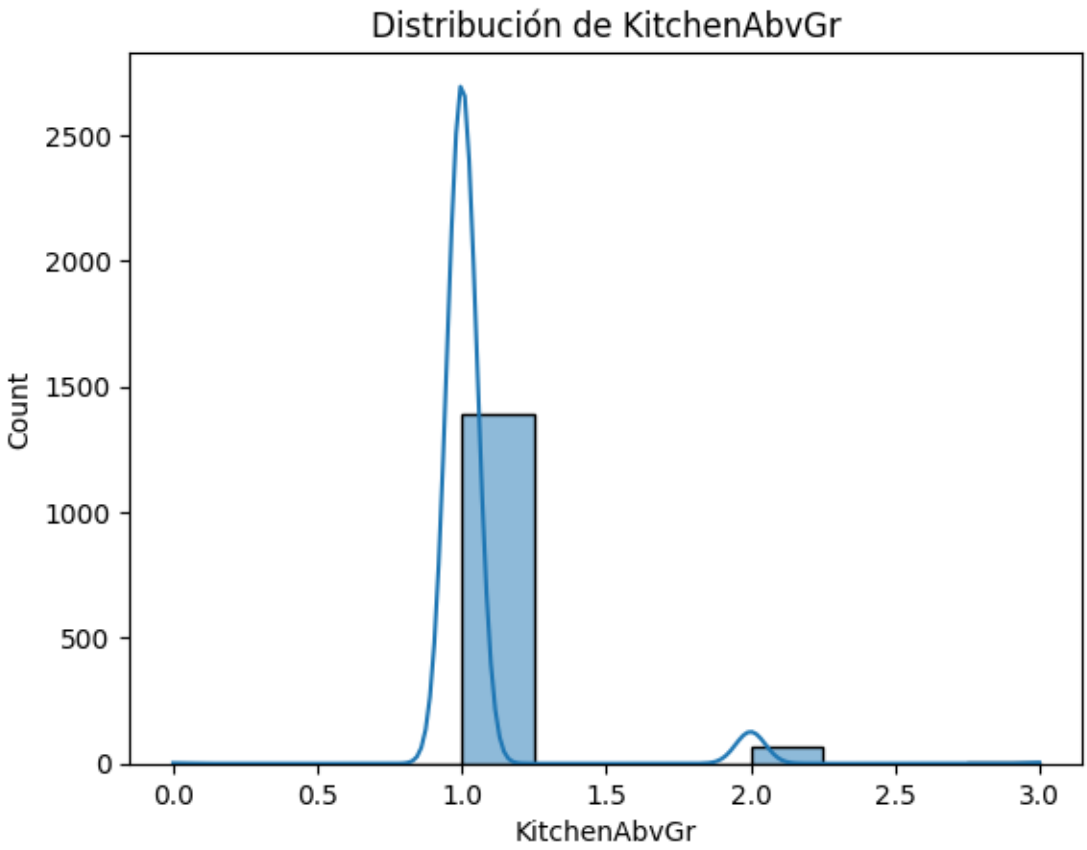

Distribución de BsmtFullBath
Count
BsmtFullBath
0
100
200
300
400
500
600
700
800
0.0
0.5
1.0
1.5
2.0
2.5
3.0

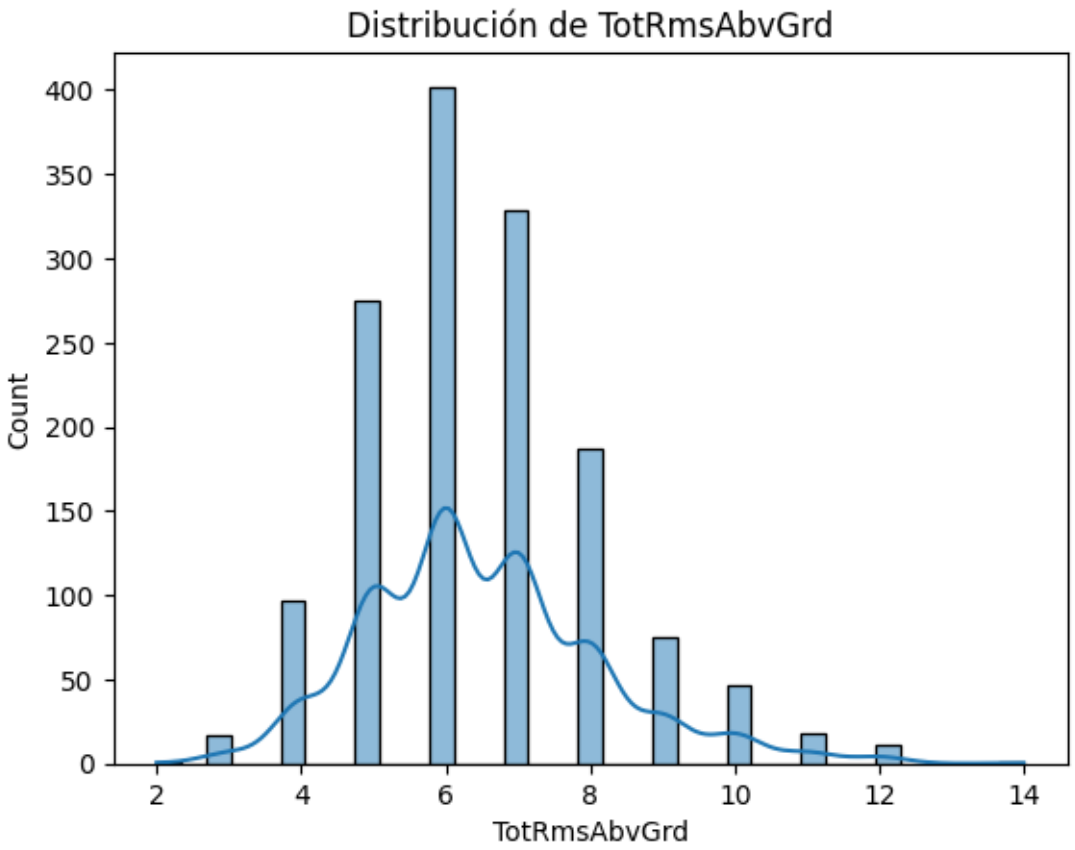

Distribución de BsmtHalfBath
Count
BsmtHalfBath
0
200
400
600
800
1000
1200
1400
1600
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

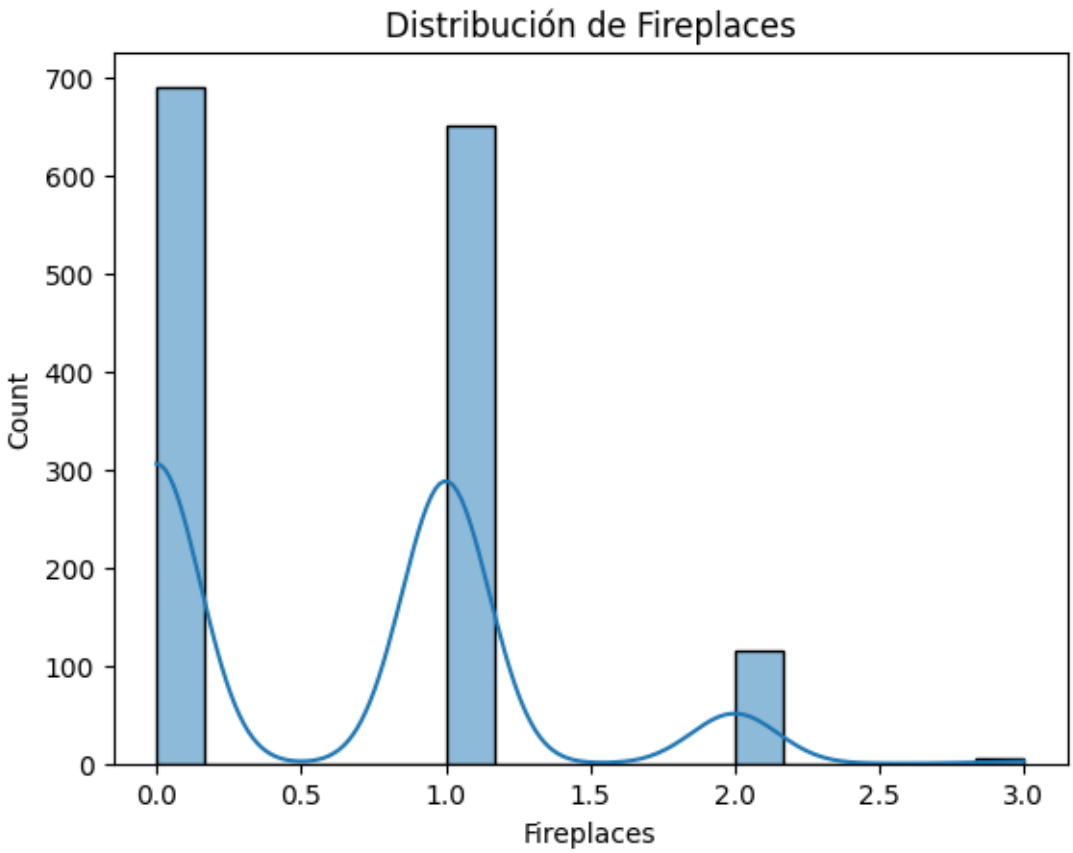

Distribución de FullBath
800
700
600
500
400
300
200
100
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
3.0
FullBath

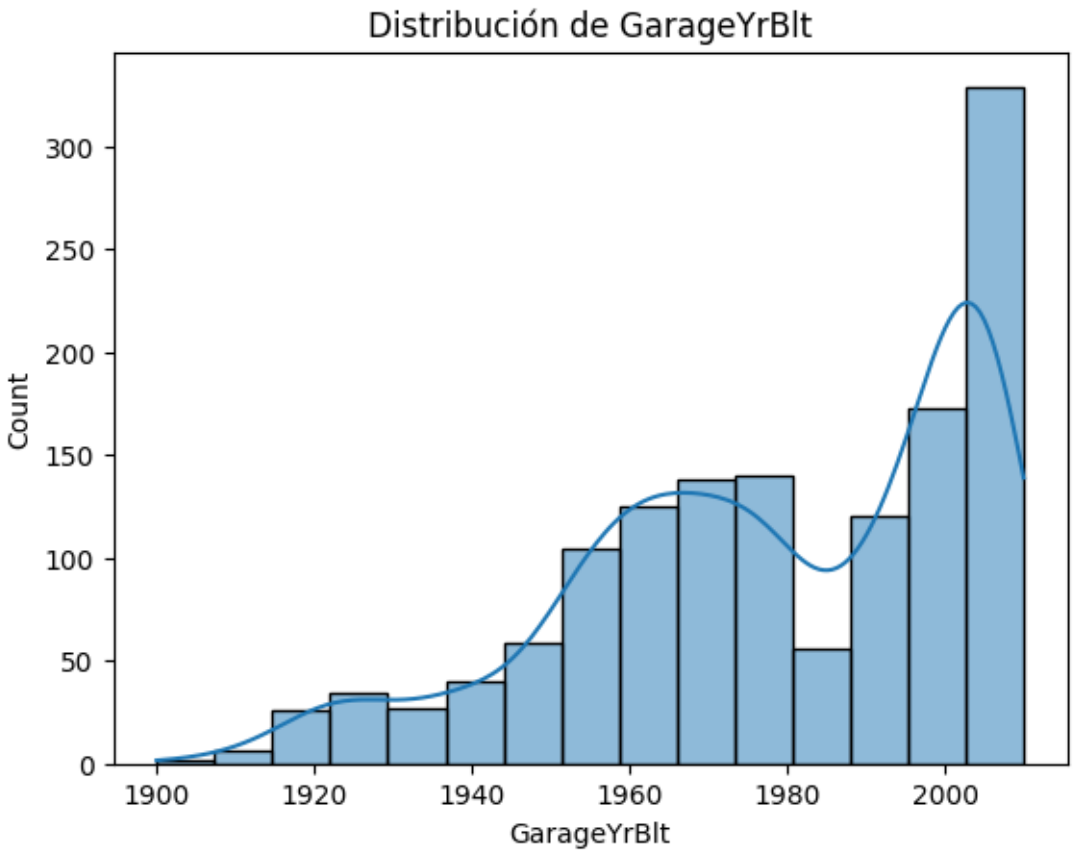

Distribución de HalfBath
Count
800
600
400
200
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
HalfBath

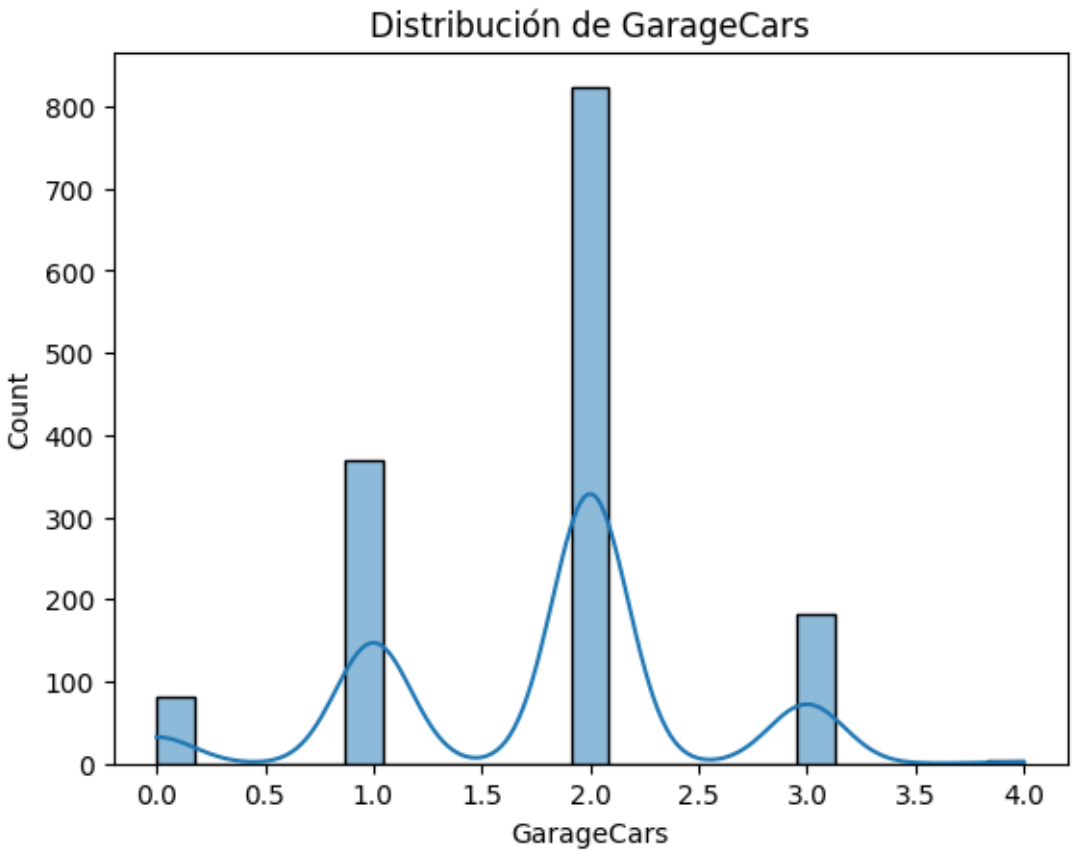

Distribución de BedroomAbvGr
Count
BedroomAbvGr
0
100
200
300
400
500
600
700
800
0
1
2
3
4
5
6
7
8

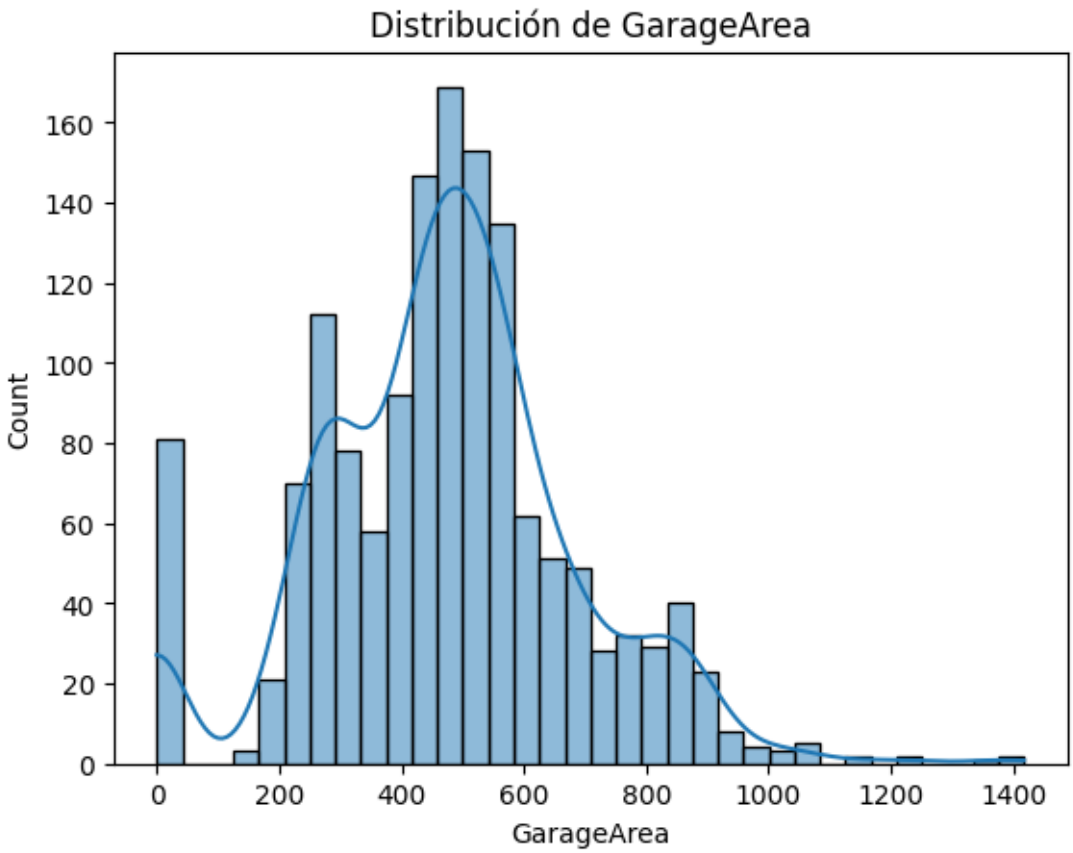

Distribución de KitchenAbvGr
Count
2500
2000
1500
1000
500
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
KitchenAbvGr

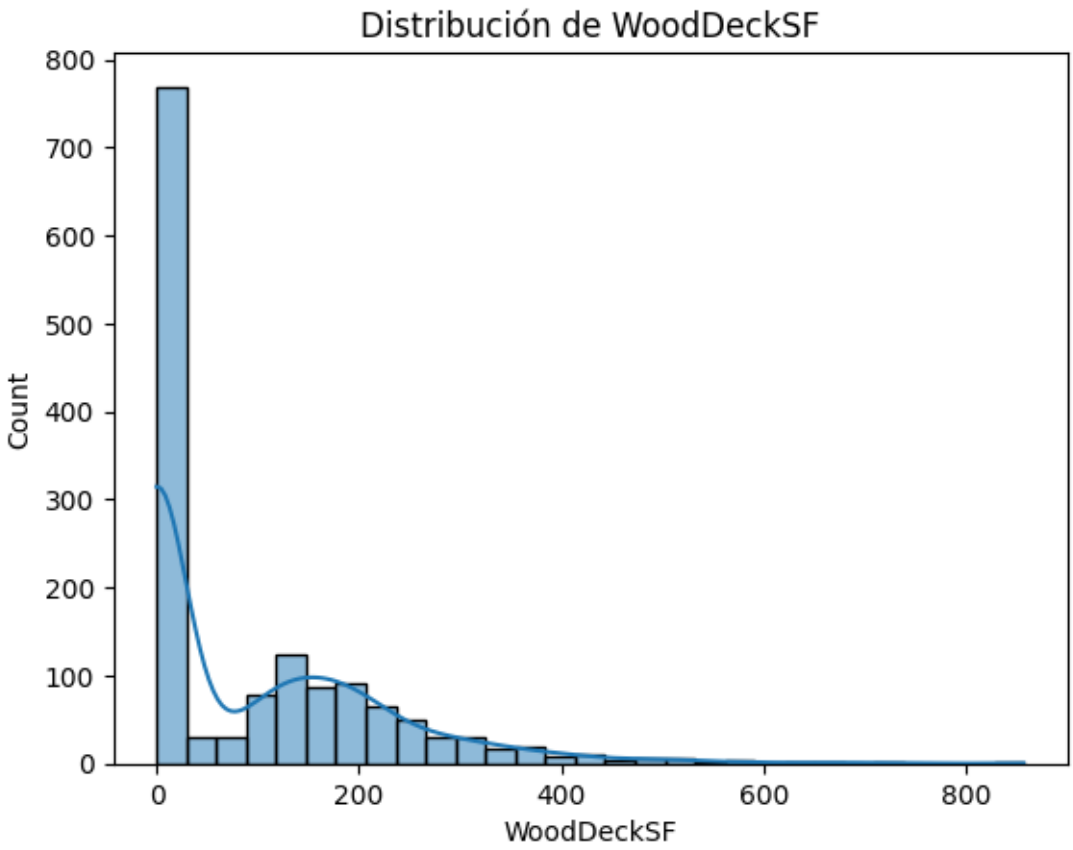

Distribución de TotRmsAbvGrd
Count
TotRmsAbvGrd
0
50
100
150
200
250
300
350
400
2
4
6
8
10
12
14

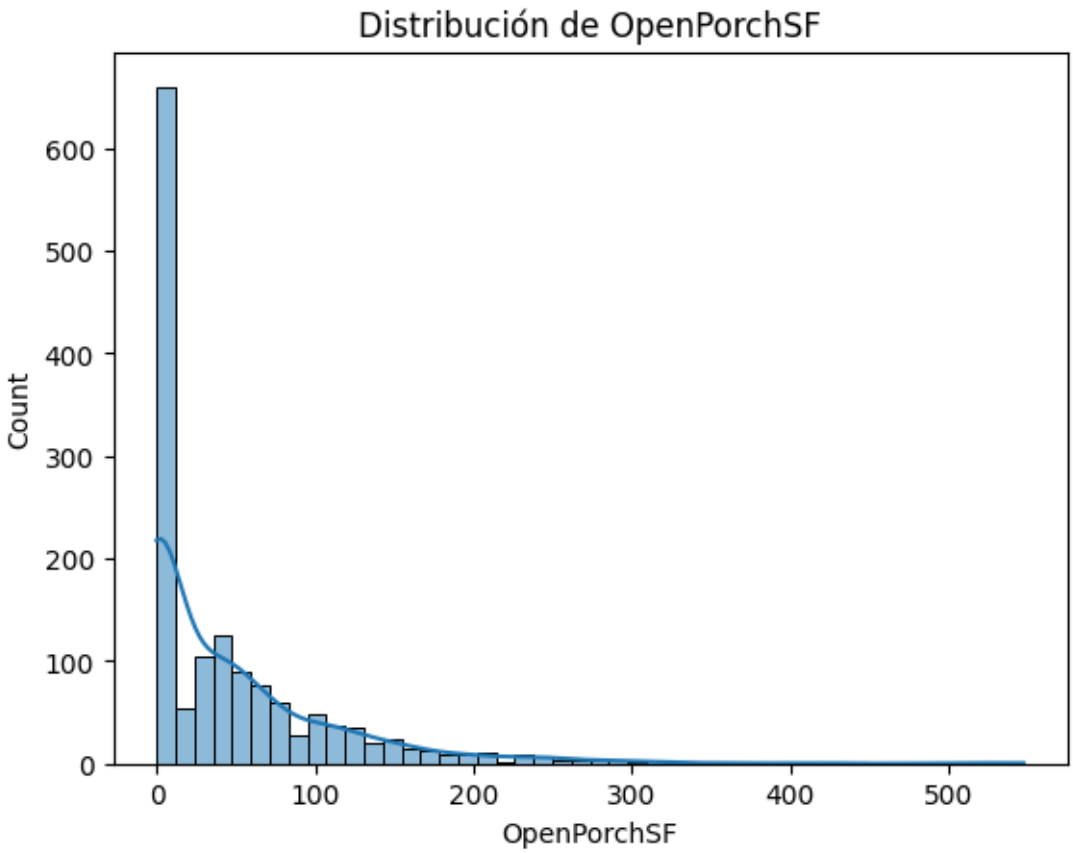

Distribución de Fireplaces
Count
Fireplaces
0
100
200
300
400
500
600
700
0.0
0.5
1.0
1.5
2.0
2.5
3.0

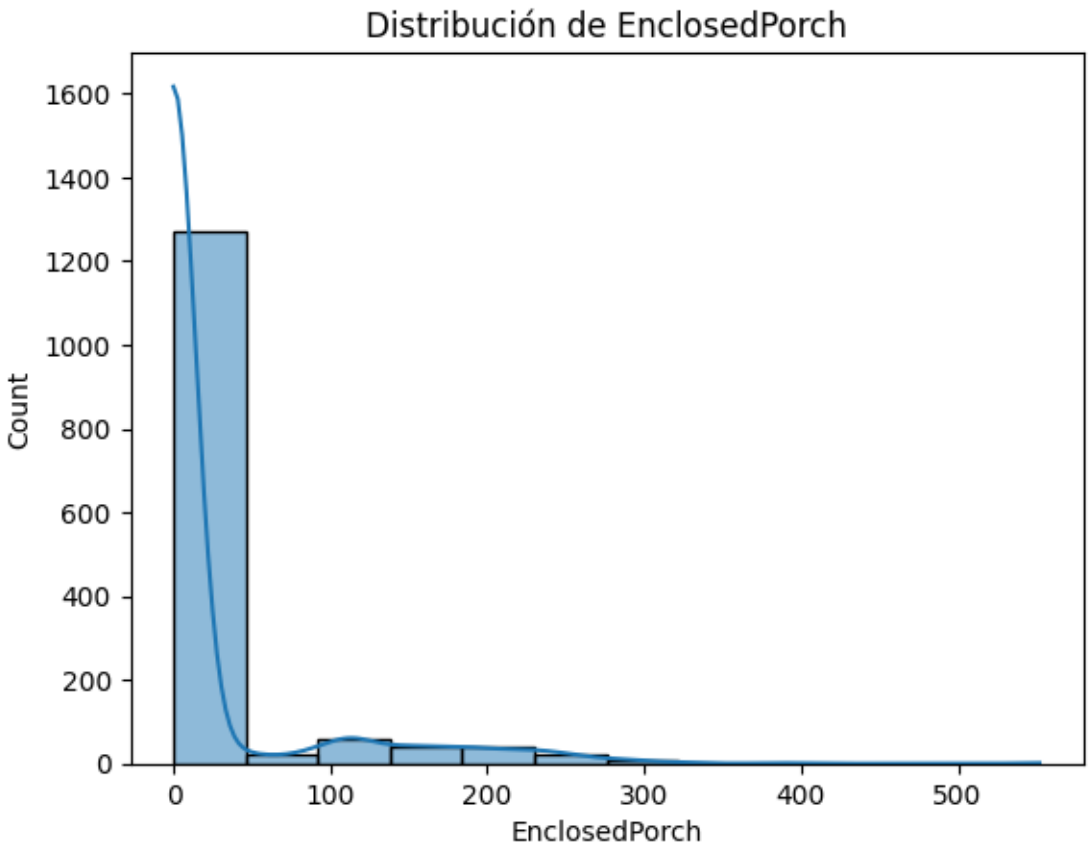

Distribución de GarageYrBlt
Count
GarageYrBlt
0
50
100
150
200
250
300
1900
1920
1940
1960
1980
2000

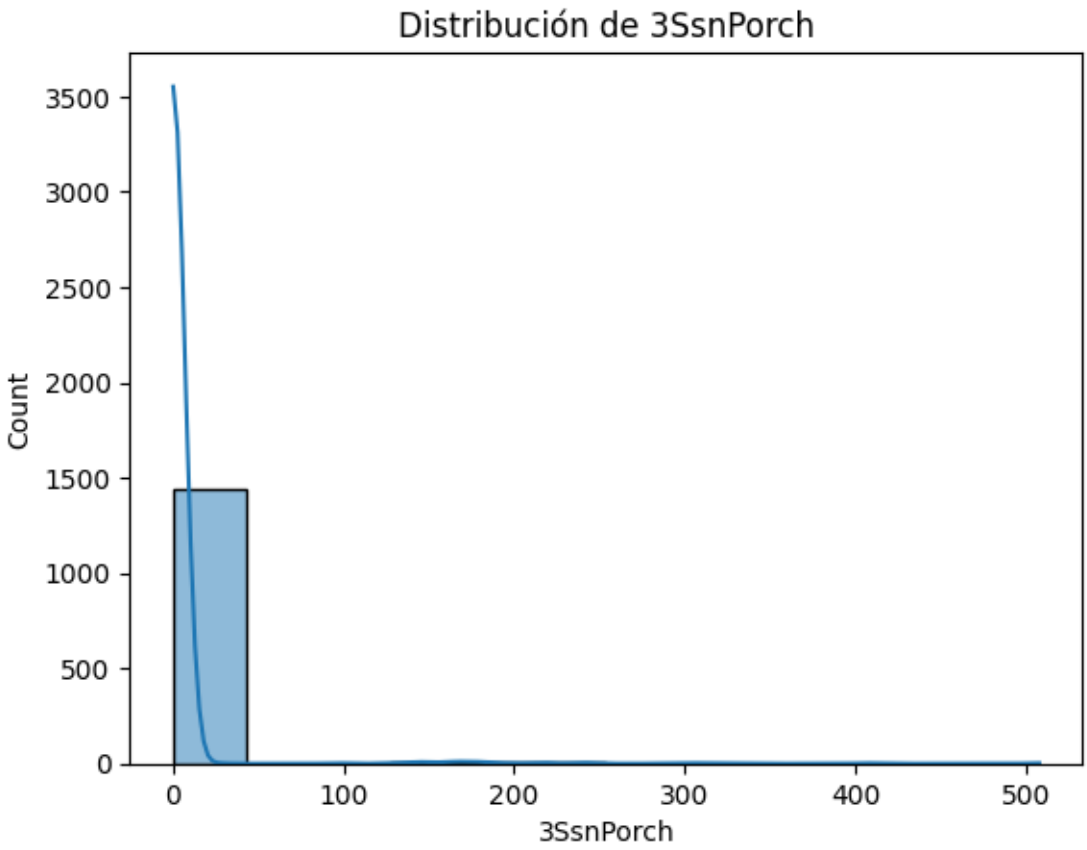

Distribución de GarageCars
Count
GarageCars
0
100
200
300
400
500
600
700
800
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

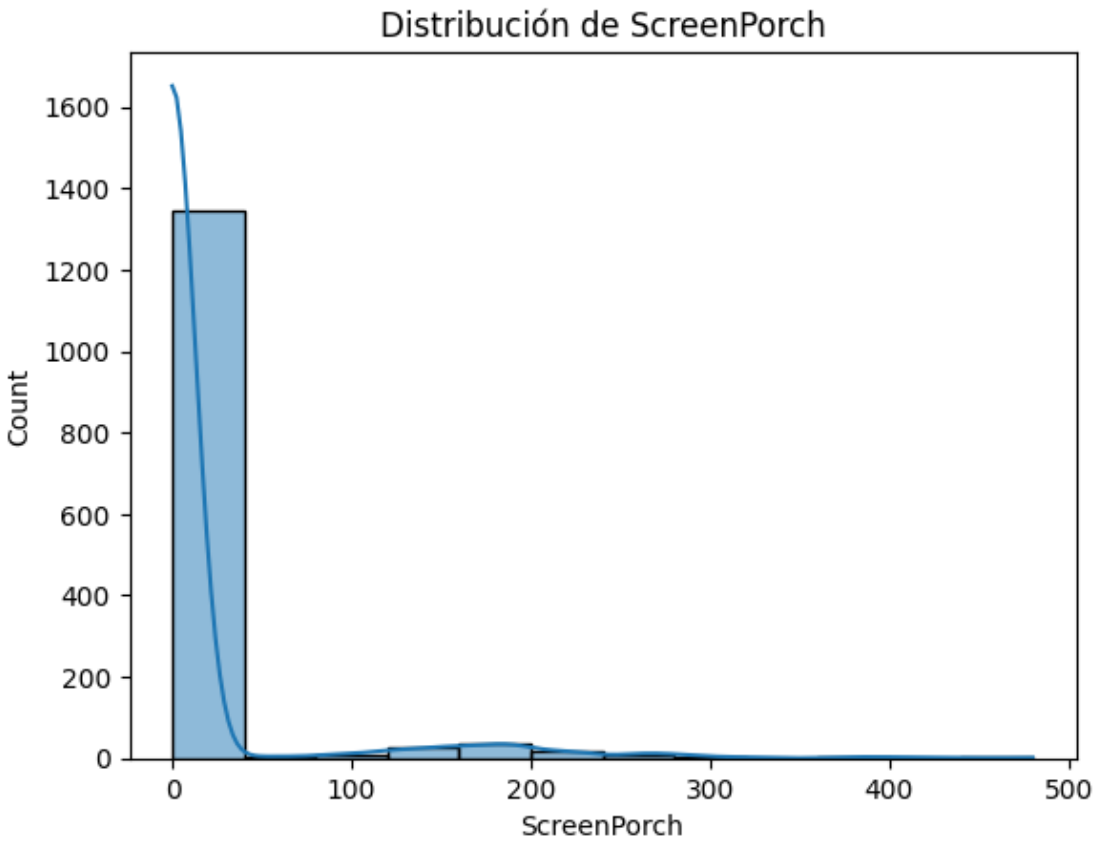

Distribución de GarageArea
Count
GarageArea
0
20
40
60
80
100
120
140
160
0
200
400
600
800
1000
1200
1400

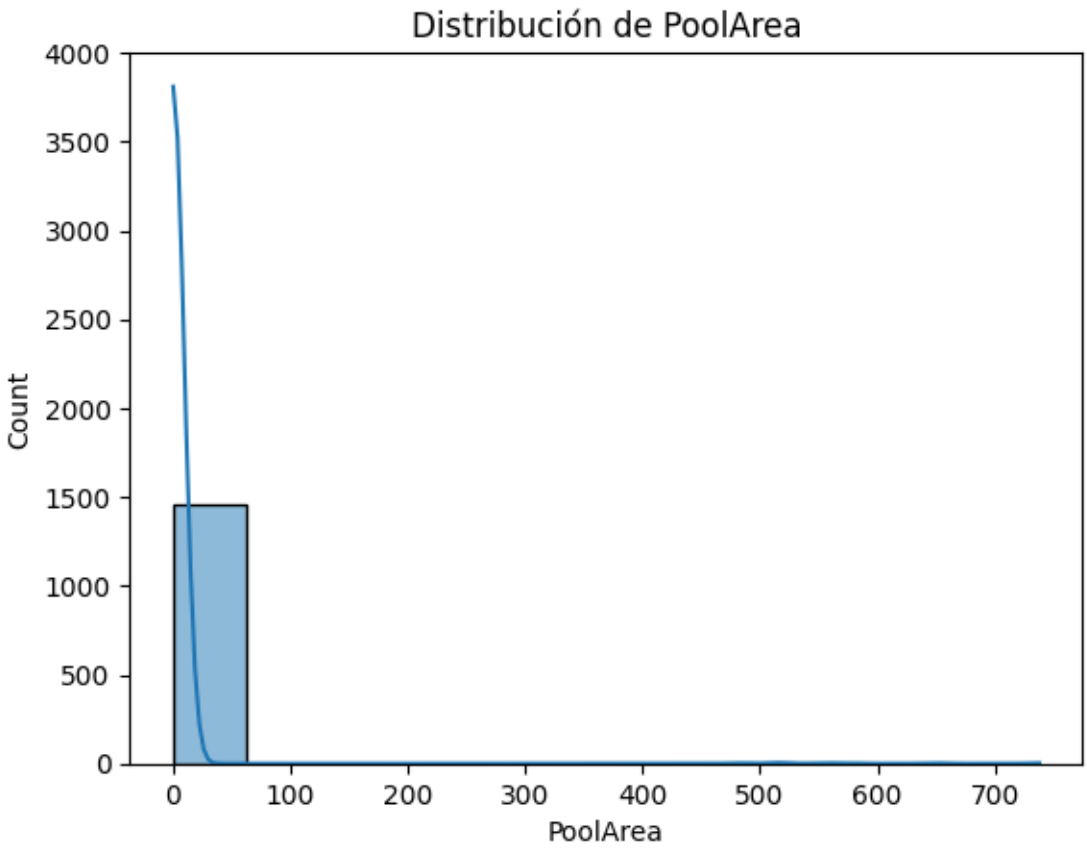

Distribución de WoodDeckSF
800
700
600
500
400
300
200
100
0
Count
0
200
400
600
800
WoodDeckSF

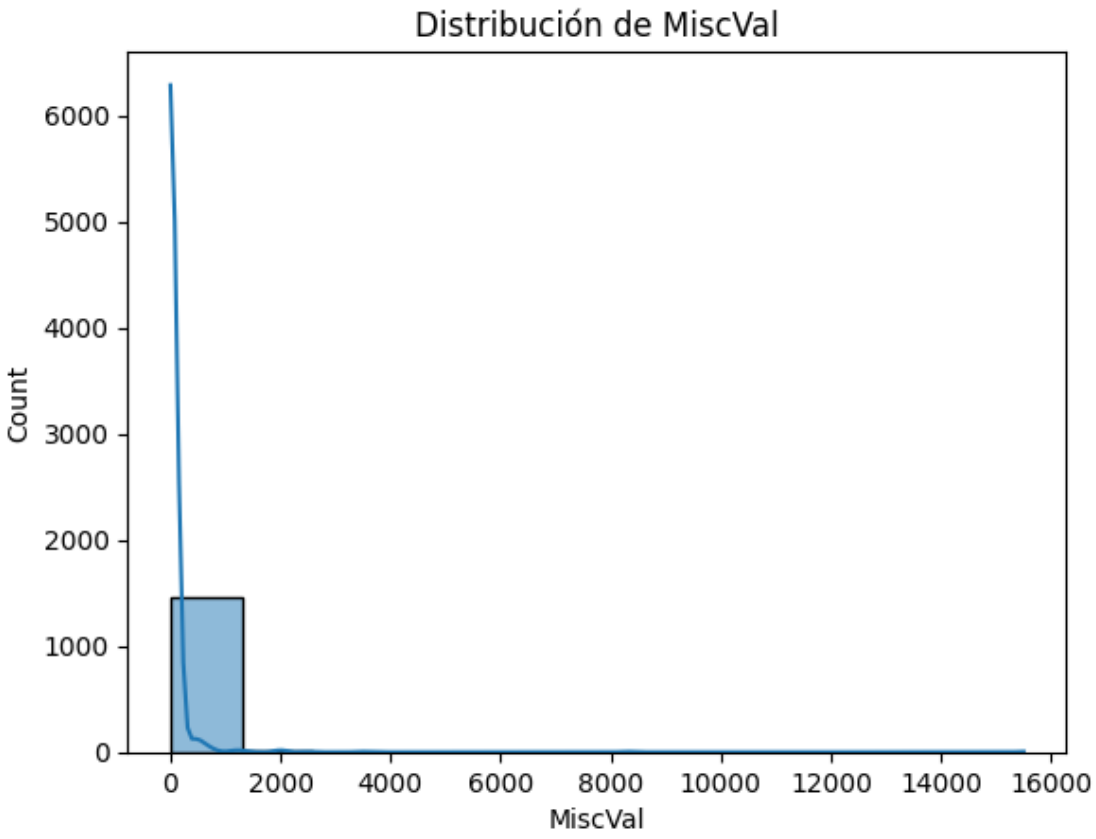

Distribución de OpenPorchSF
Count
OpenPorchSF
0
100
200
300
400
500
600

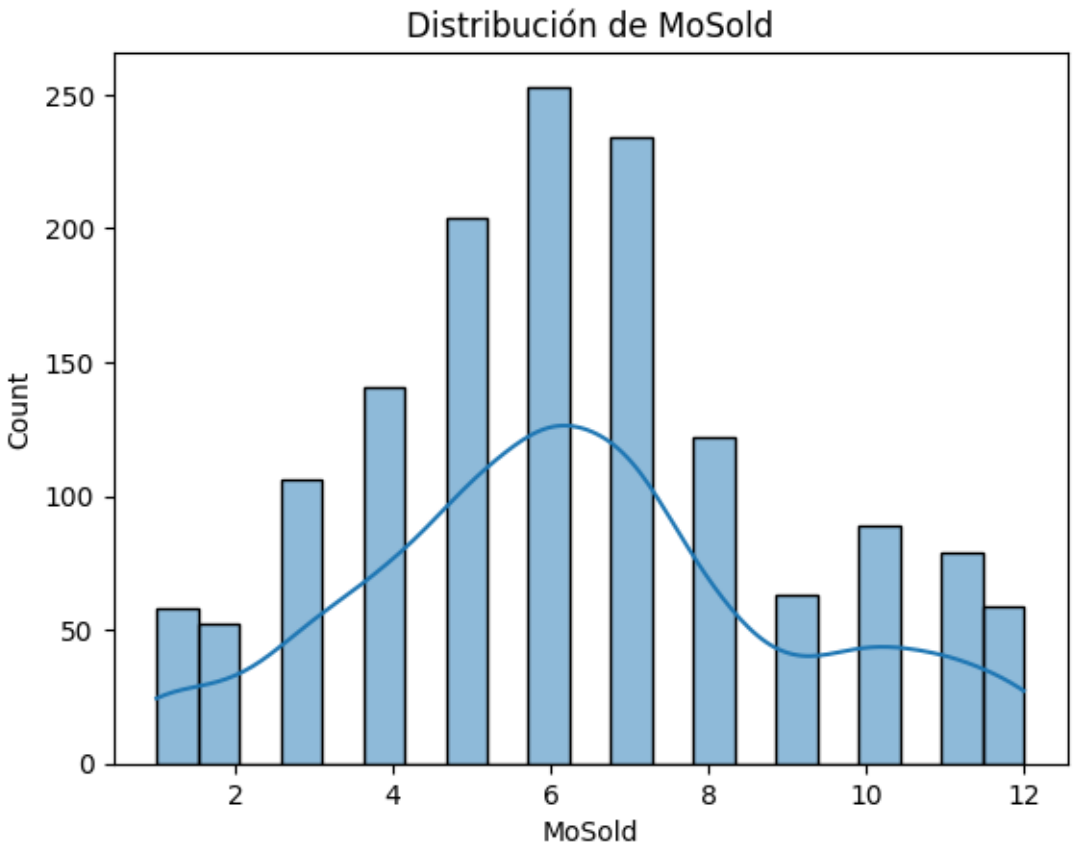

Distribución de EnclosedPorch
Count
EnclosedPorch
0
200
400
600
800
1000
1200
1400
1600
0
100
200
300
400
500

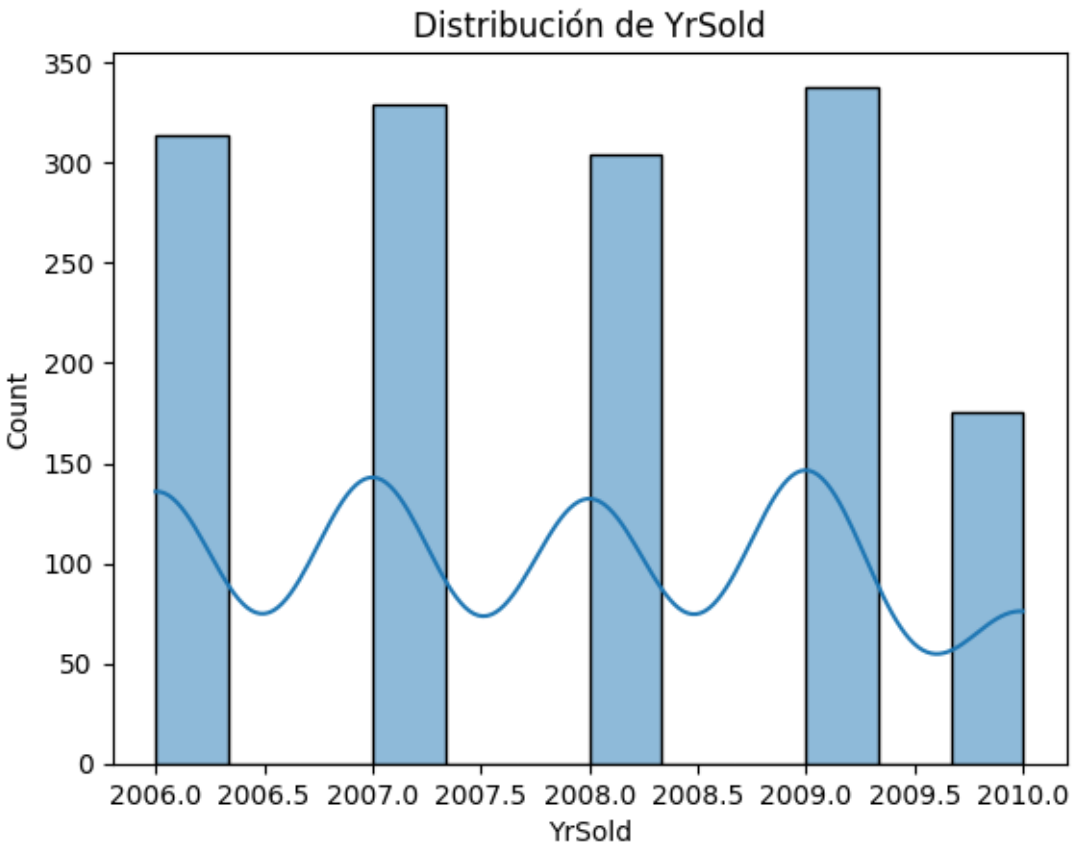

Distribución de 3SsnPorch
Count
3SsnPorch
0
500
1000
1500
2000
2500
3000
3500
0
100
200
300
400
500

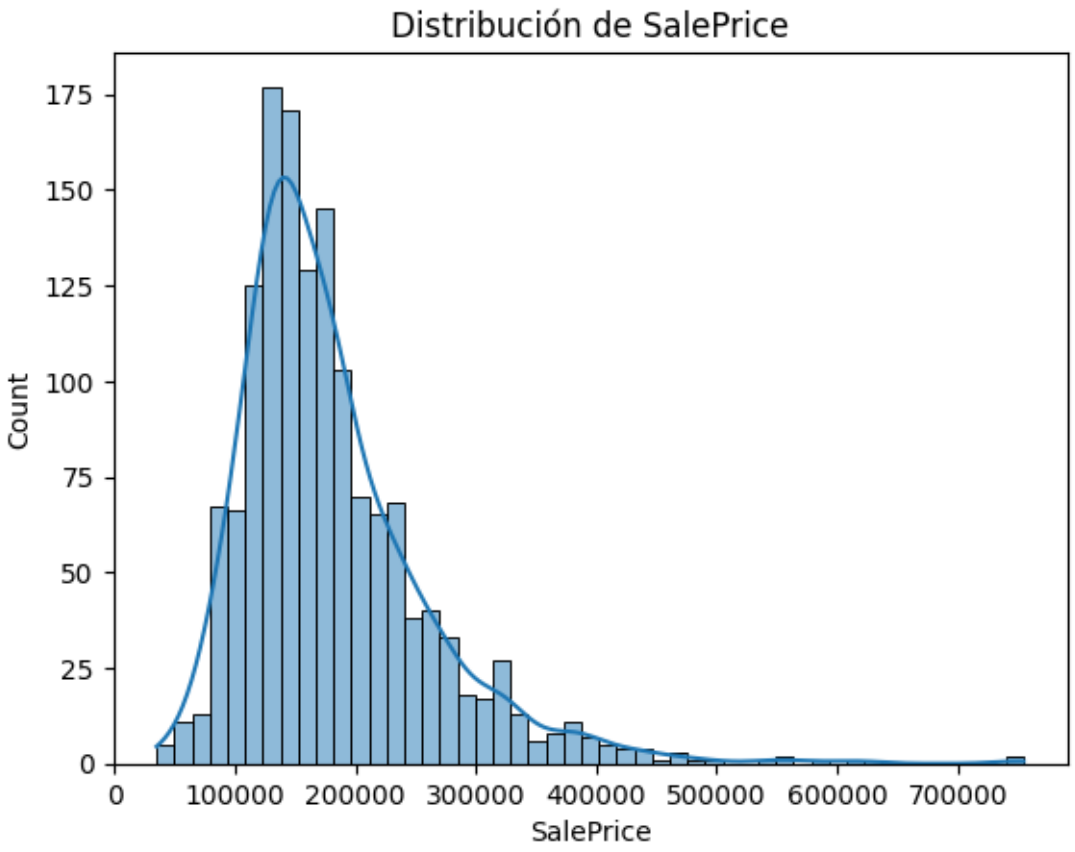

Distribución de ScreenPorch
Count
ScreenPorch
0
200
400
600
800
1000
1200
1400
1600
0
100
200
300
400
500


Distribución de PoolArea
Count
PoolArea
4000
3500
3000
2500
2000
1500
1000
500
0
0
100
200
300
400
500
600
700


Distribución de MiscVal
Count
MiscVal
0
1000
2000
3000
4000
5000
6000
0
2000
4000
6000
8000
10000
12000
14000
16000


Distribución de MoSold
Count
MoSold
0
50
100
150
200
250
2
4
6
8
10
12

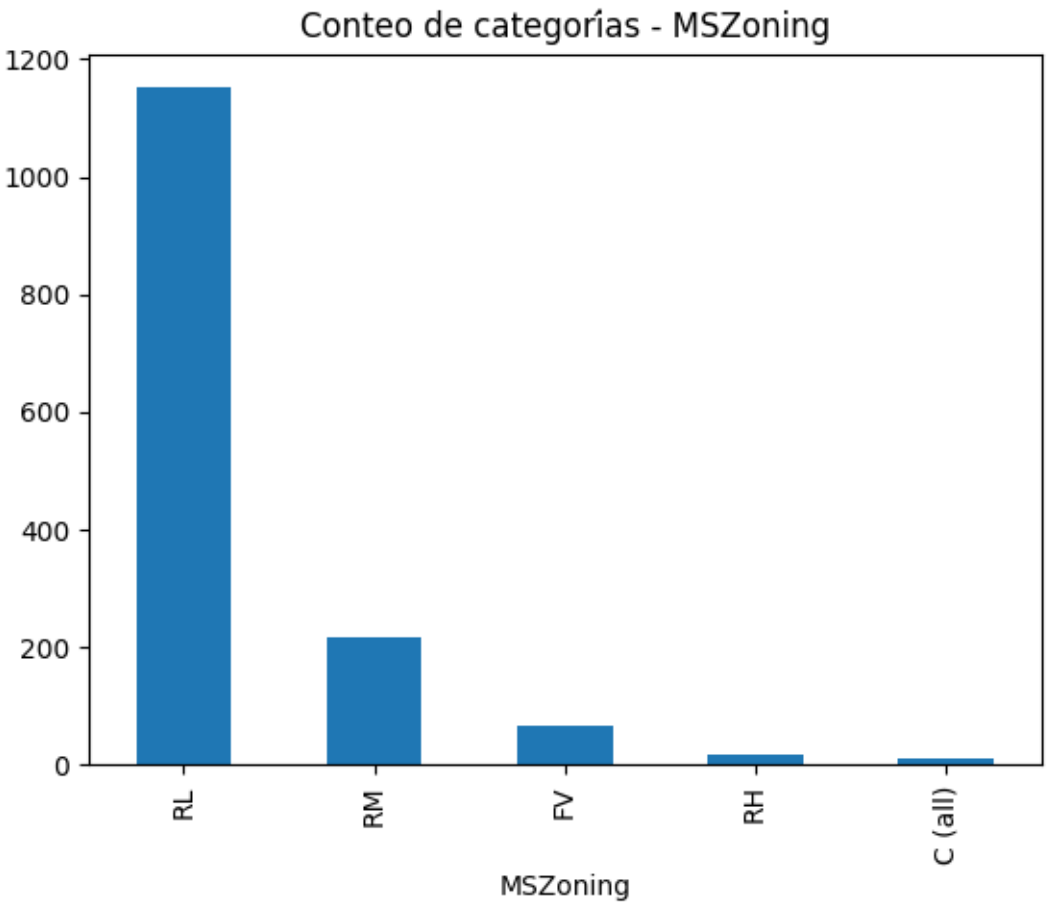

Distribución de YrSold
Count
YrSold
0
50
100
150
200
250
300
350
2006.0
2006.5
2007.0
2007.5
2008.0
2008.5
2009.0
2009.5
2010.0



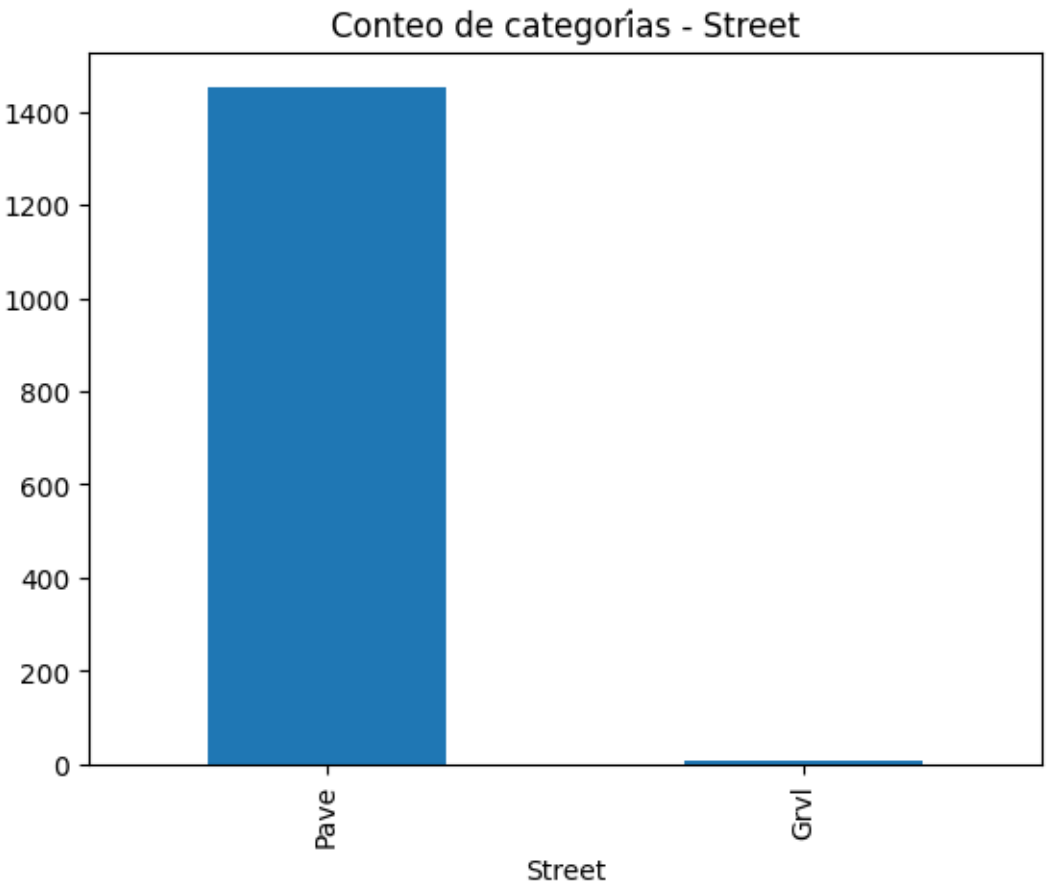
1.6.2 6.2 Variables categóricas

Mostramos cuántas entradas hay para cada categoría. Podemos usar gráficos de barras o tablas.

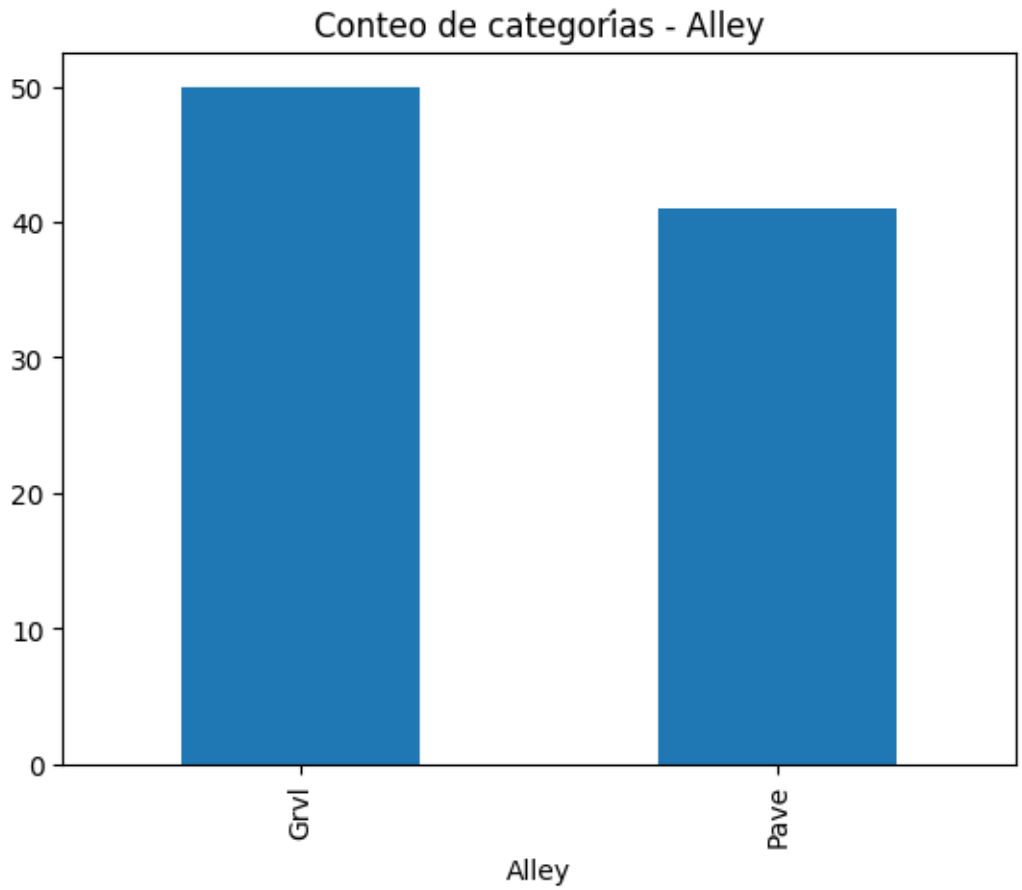
[7]:
```
for col in categorical_feats:
    plt.figure()
    train[col].value_counts().plot(kind='bar')
    plt.title(f'Conteo de categorías - {col}')
    plt.show()

    # Si deseas ver la tabla numérica:
    display(train[col].value_counts())
```



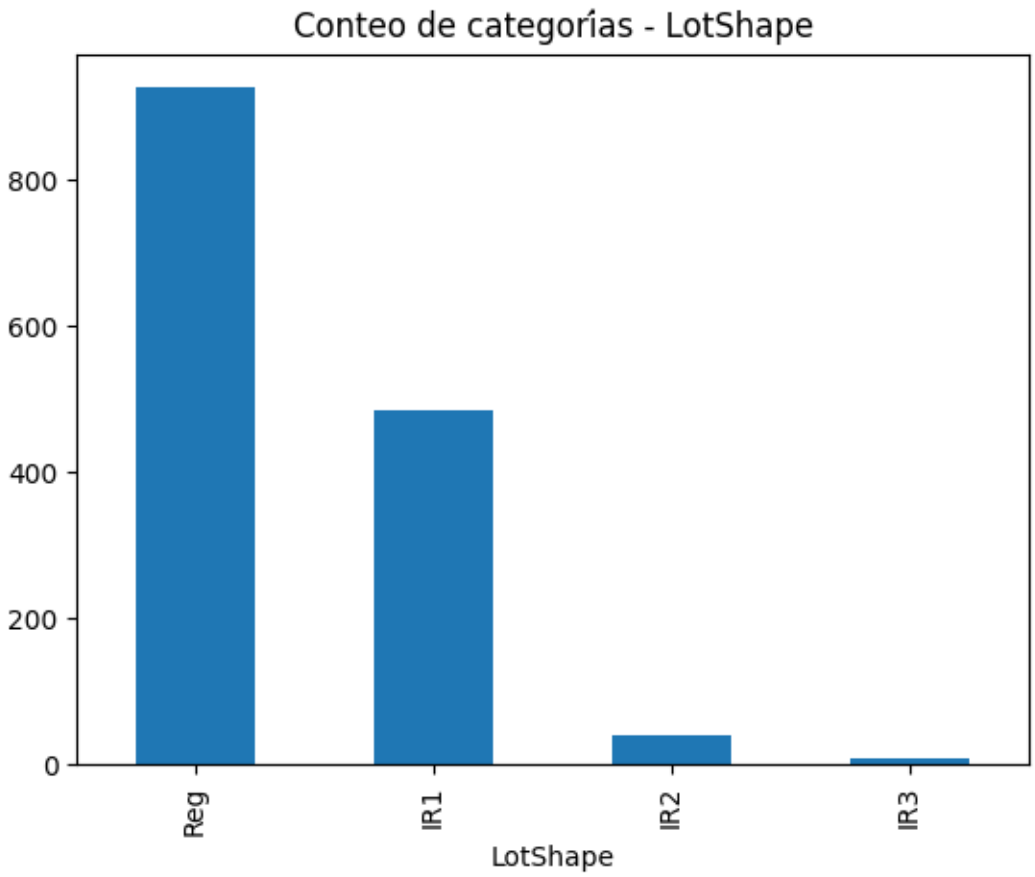

```
MSZoning
RL         1151
RM          218
FV           65
RH           16
C (all)      10
Name: count, dtype: int64
```



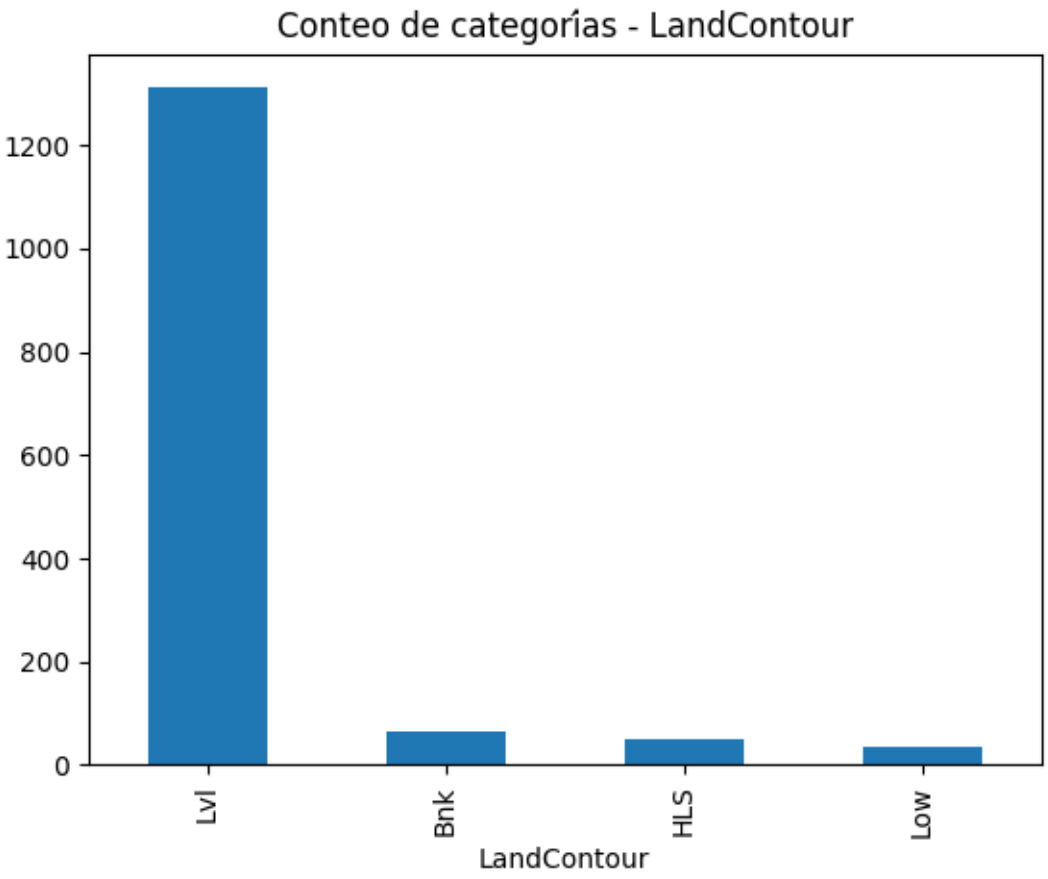

```
Street
Pave    1454
Grvl       6
Name: count, dtype: int64
```



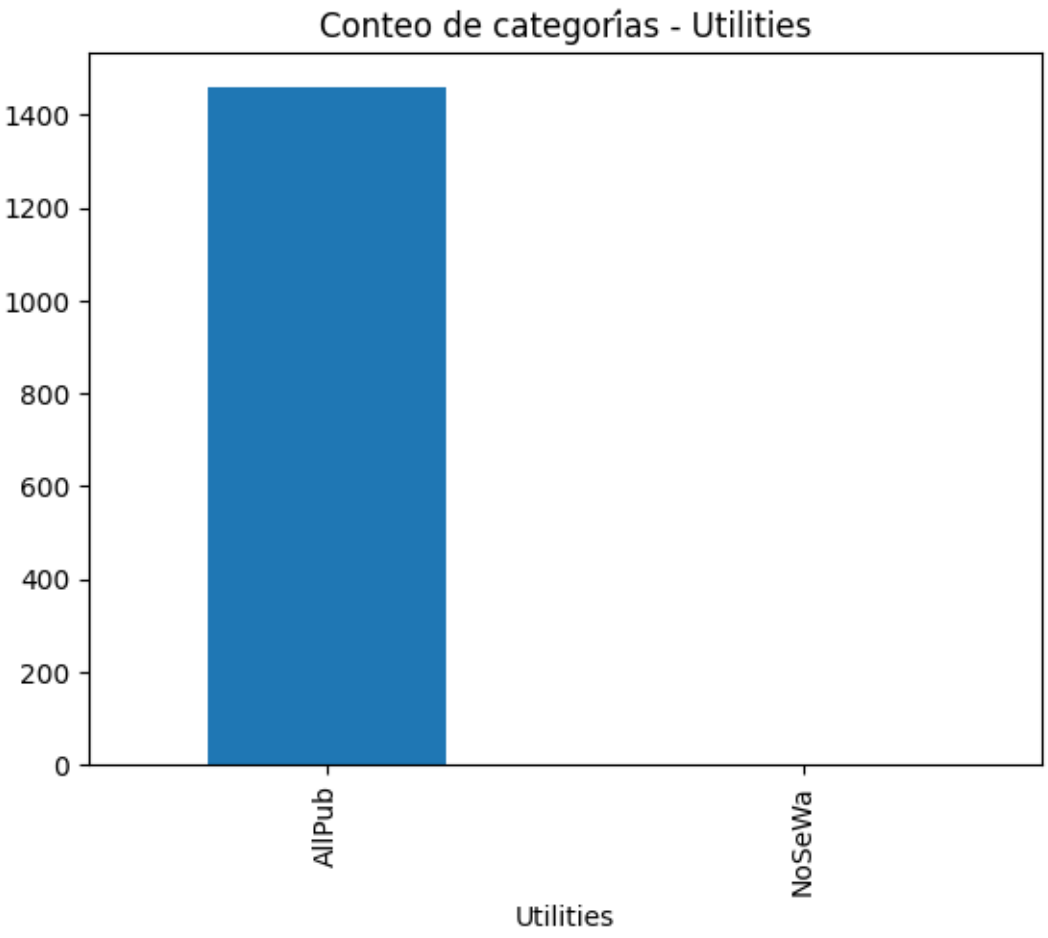

```
Alley
Grvl    50
Pave    41
Name: count, dtype: int64
```



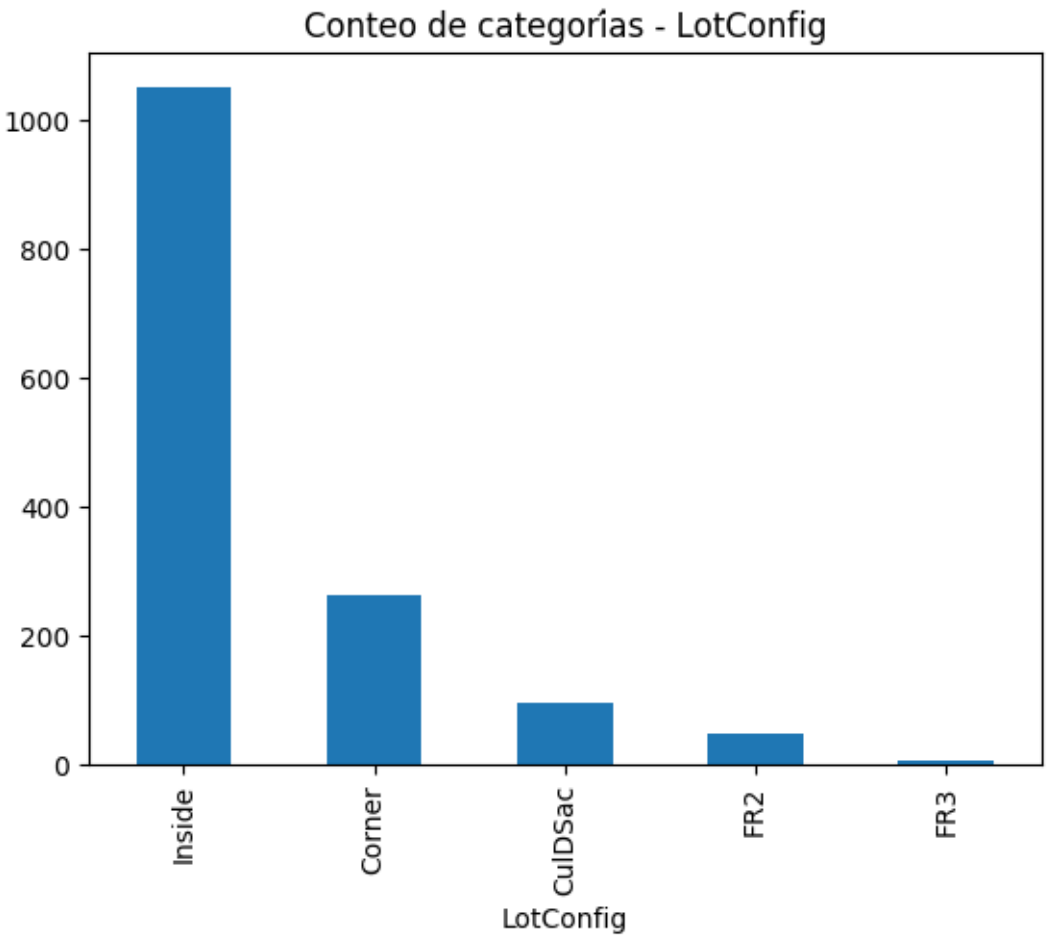

```
LotShape
Reg    925
IR1    484
IR2     41
IR3     10
Name: count, dtype: int64
```



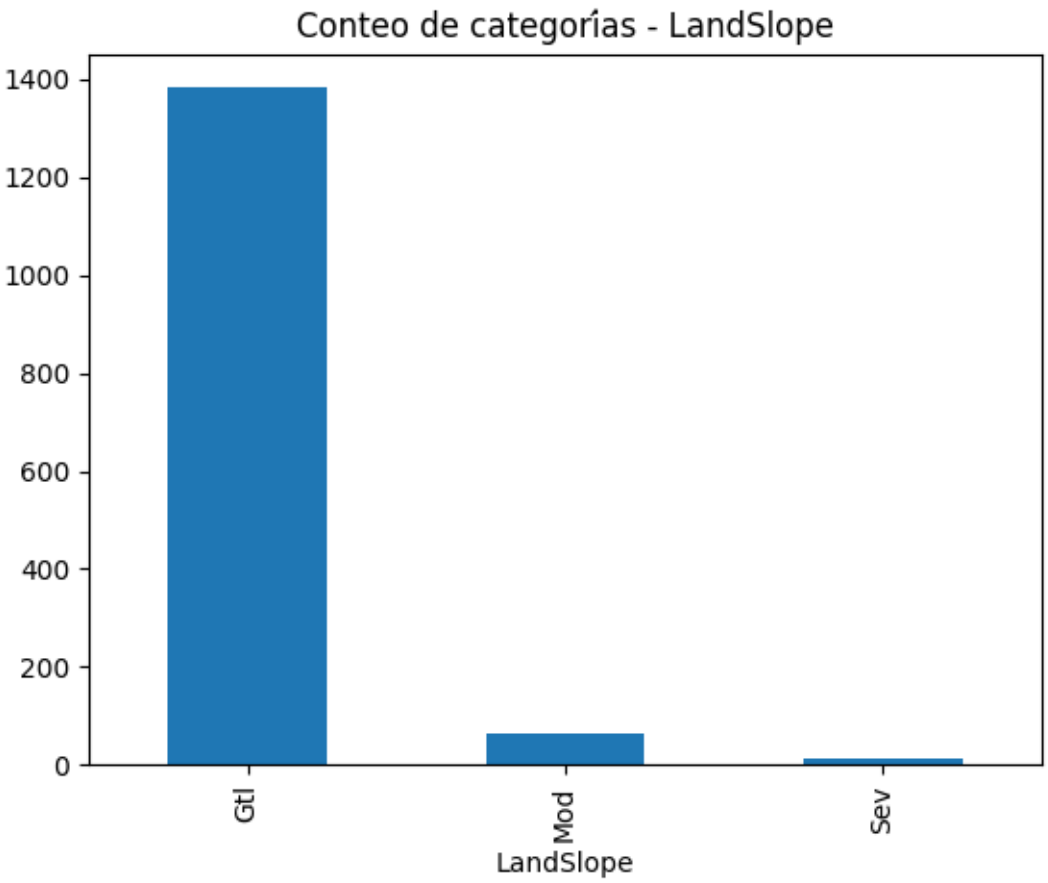

```
LandContour
Lvl    1311
Bnk      63
HLS      50
Low      36
Name: count, dtype: int64
```



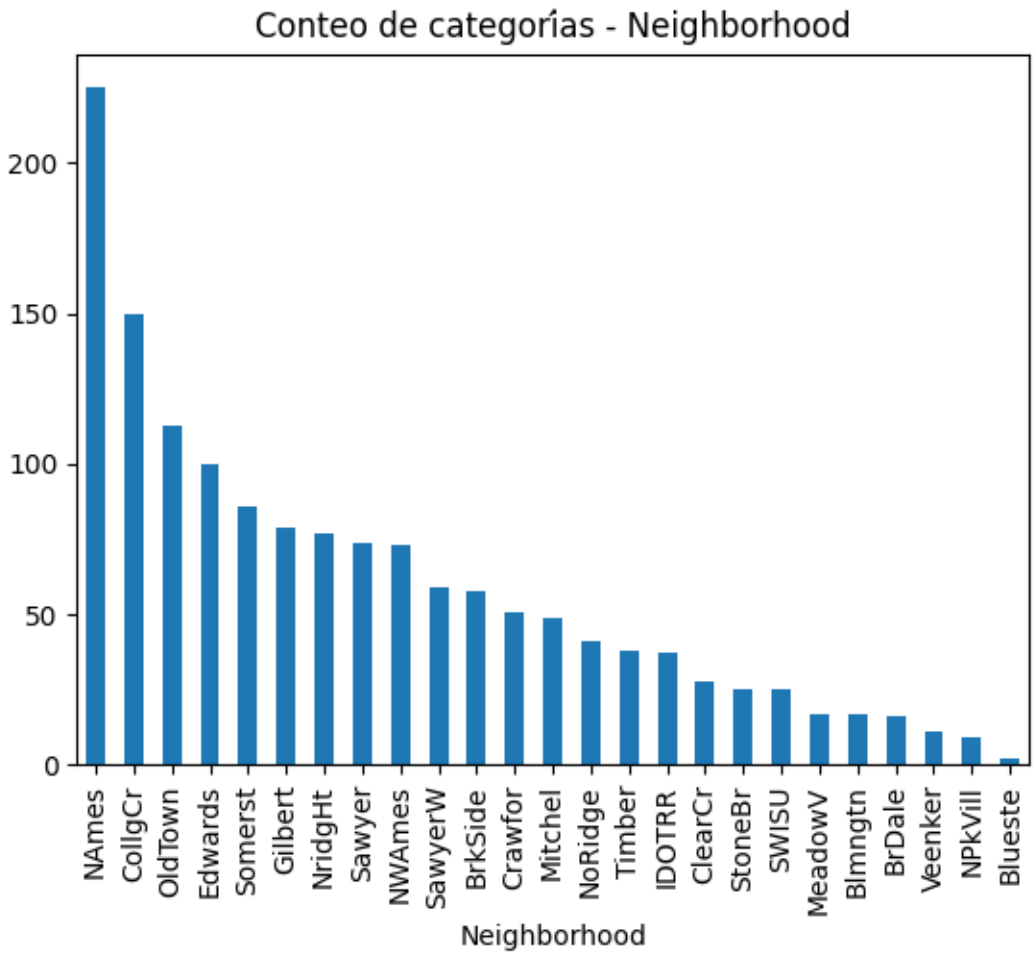

```
Utilities
AllPub    1459
NoSeWa       1
Name: count, dtype: int64
```




```
LotConfig
Inside     1052
Corner      263
CulDSac      94
FR2          47
FR3           4
Name: count, dtype: int64
```

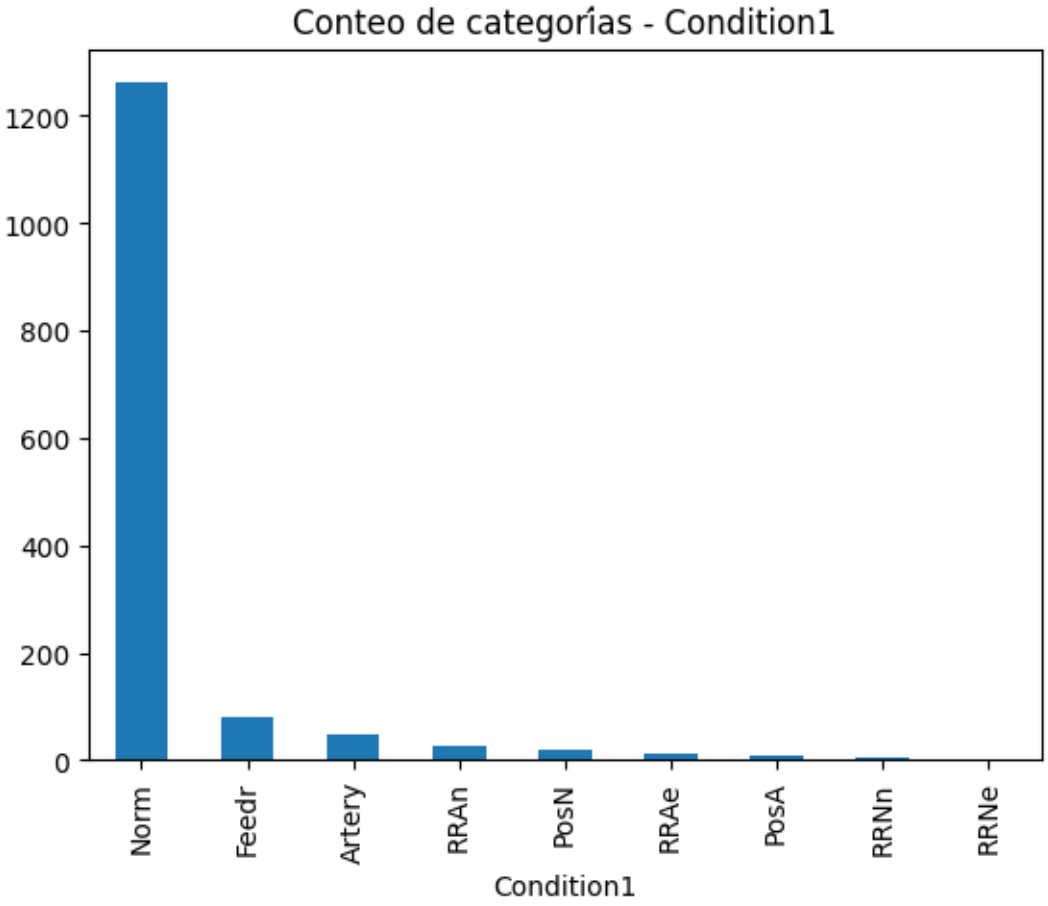



```
LandSlope
Gtl    1382
Mod      65
Sev      13
Name: count, dtype: int64
```



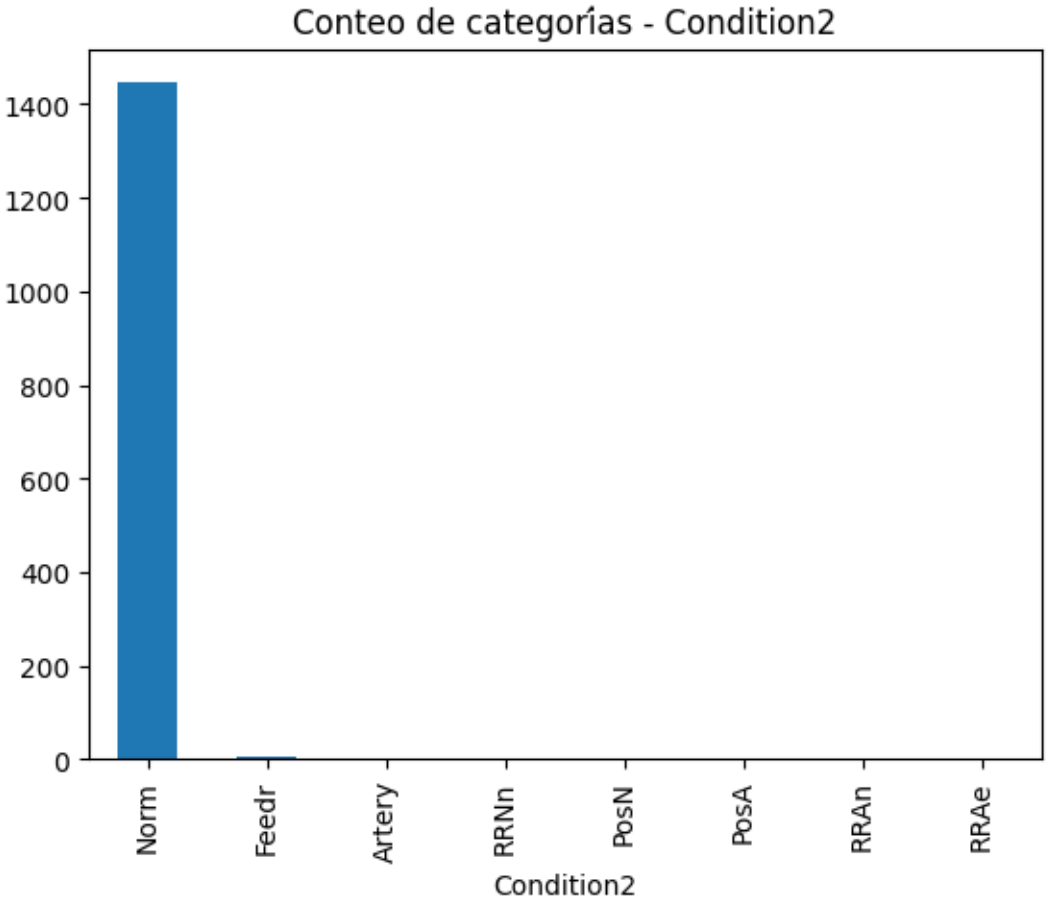

```
Neighborhood
NAmes      225
CollgCr    150
OldTown    113
Edwards    100
Somerst     86
Gilbert     79
NridgHt     77
Sawyer      74
NWAmes      73
SawyerW     59
BrkSide     58
Crawfor     51
Mitchel     49
NoRidge     41
Timber      38
IDOTRR      37
```

```
ClearCr     28
StoneBr     25
SWISU       25
MeadowV     17
Blmngtn     17
BrDale      16
Veenker     11
NPkVill      9
Blueste      2
Name: count, dtype: int64
```

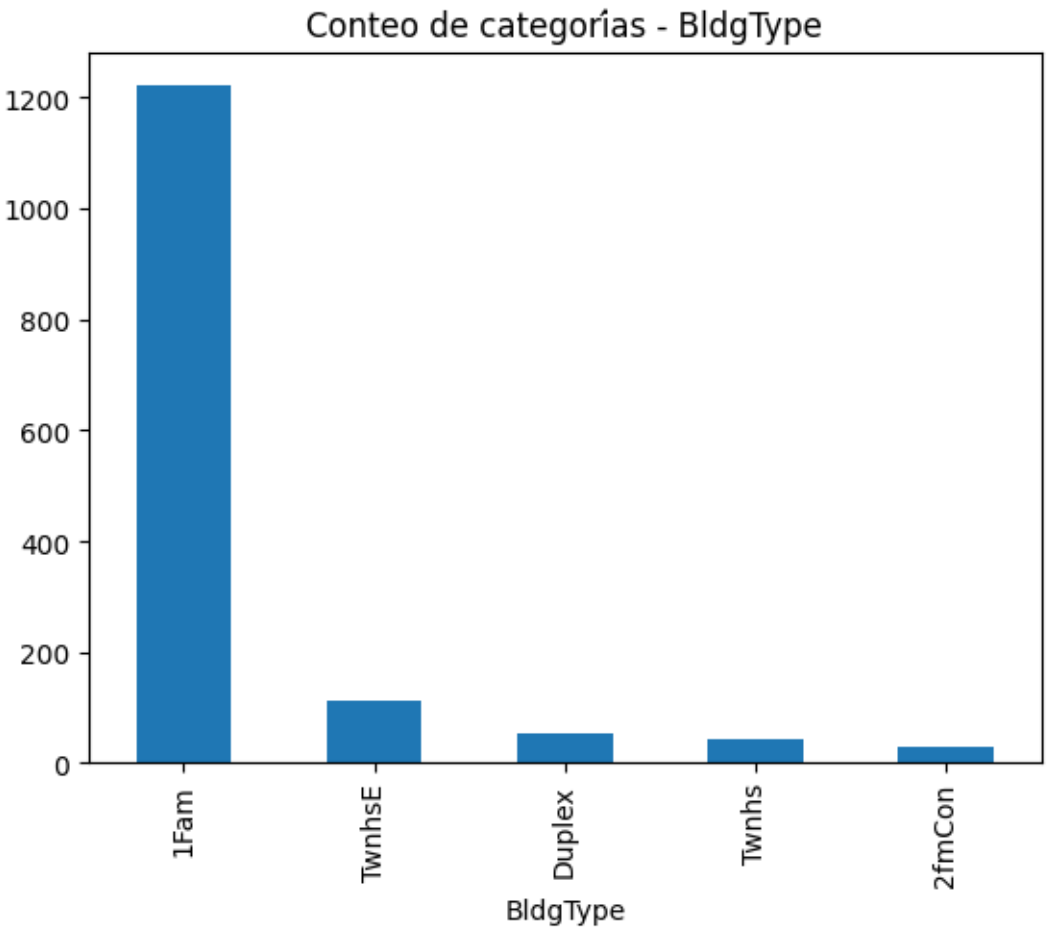



```
Condition1
Norm      1260
Feedr       81
Artery      48
RRAn        26
PosN        19
RRAe        11
PosA         8
```

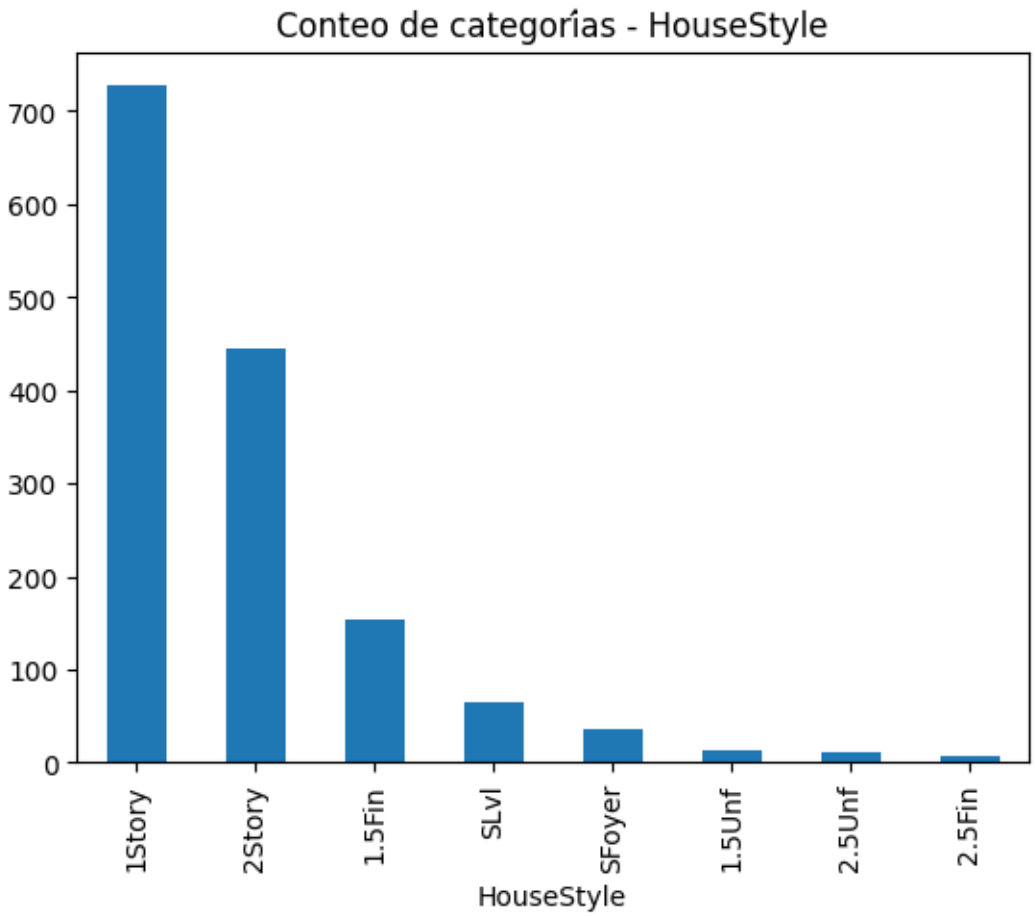
```
RRNn          5
RRNe          2
Name: count, dtype: int64
```



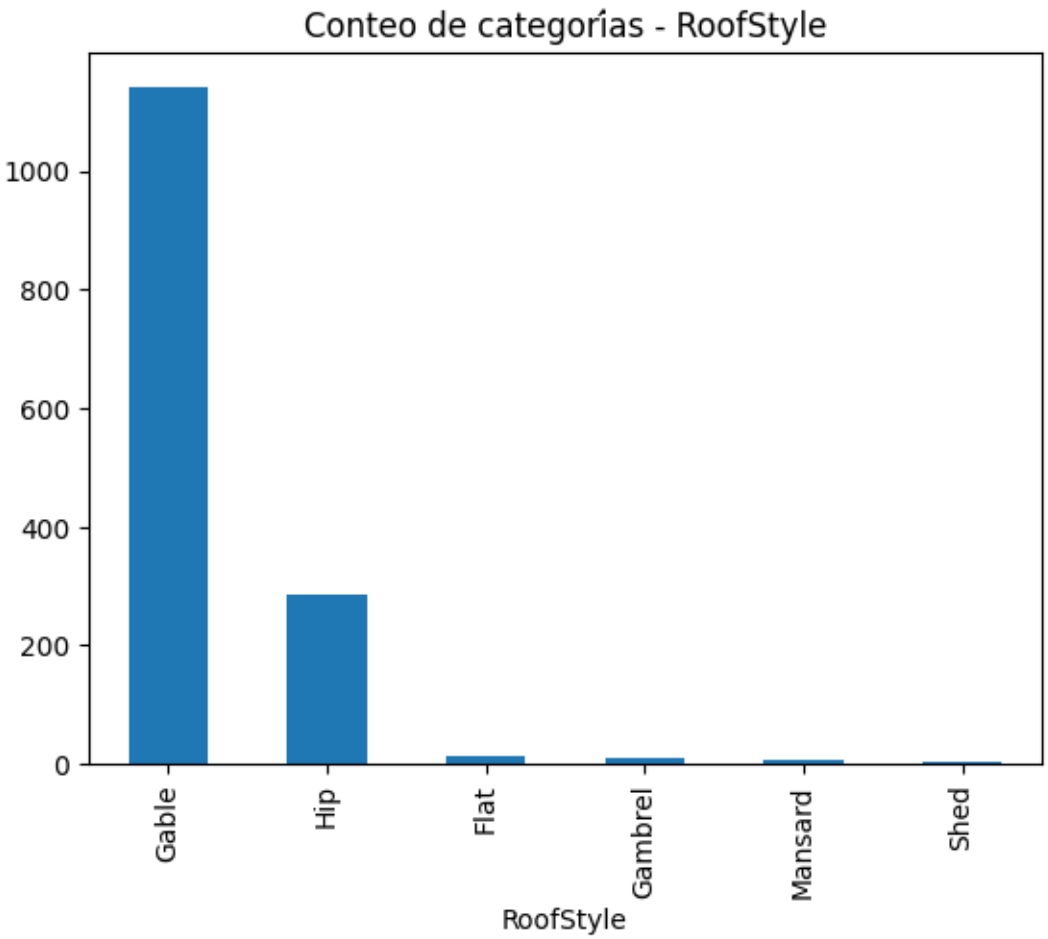

```
Condition2
Norm      1445
Feedr        6
Artery       2
RRNn         2
PosN         2
PosA         1
RRAn         1
RRAe         1
Name: count, dtype: int64
```



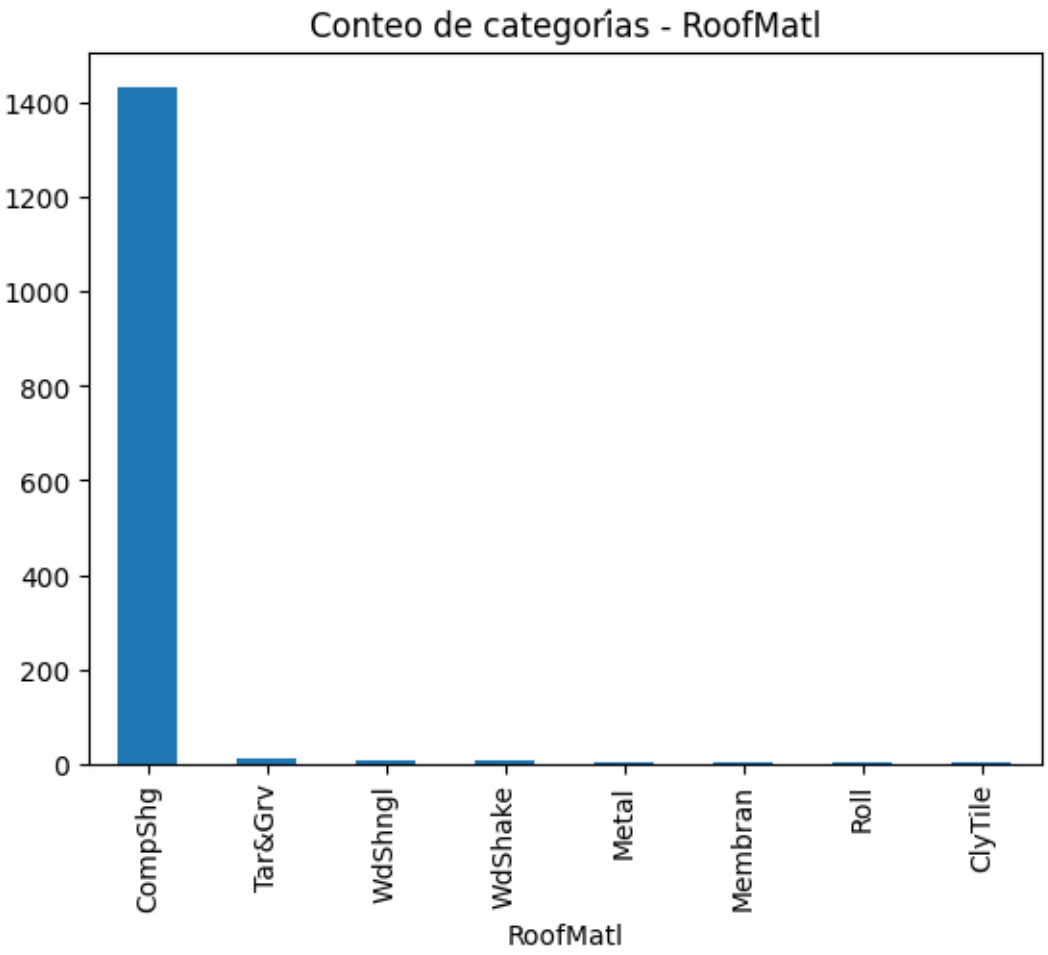

```
BldgType
1Fam      1220
TwnhsE     114
Duplex      52
Twnhs       43
2fmCon      31
Name: count, dtype: int64
```



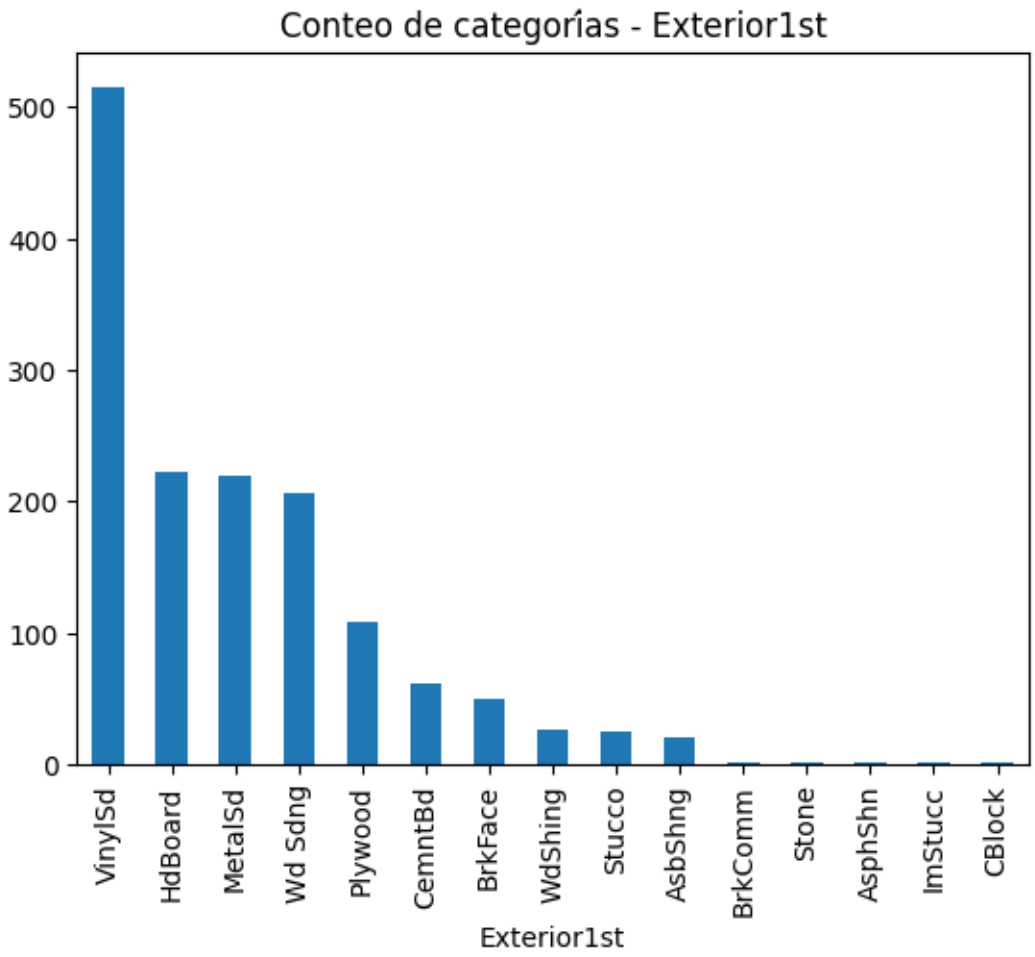

```
HouseStyle
1Story    726
2Story    445
1.5Fin    154
SLvl       65
SFoyer     37
1.5Unf     14
2.5Unf     11
2.5Fin      8
Name: count, dtype: int64
```



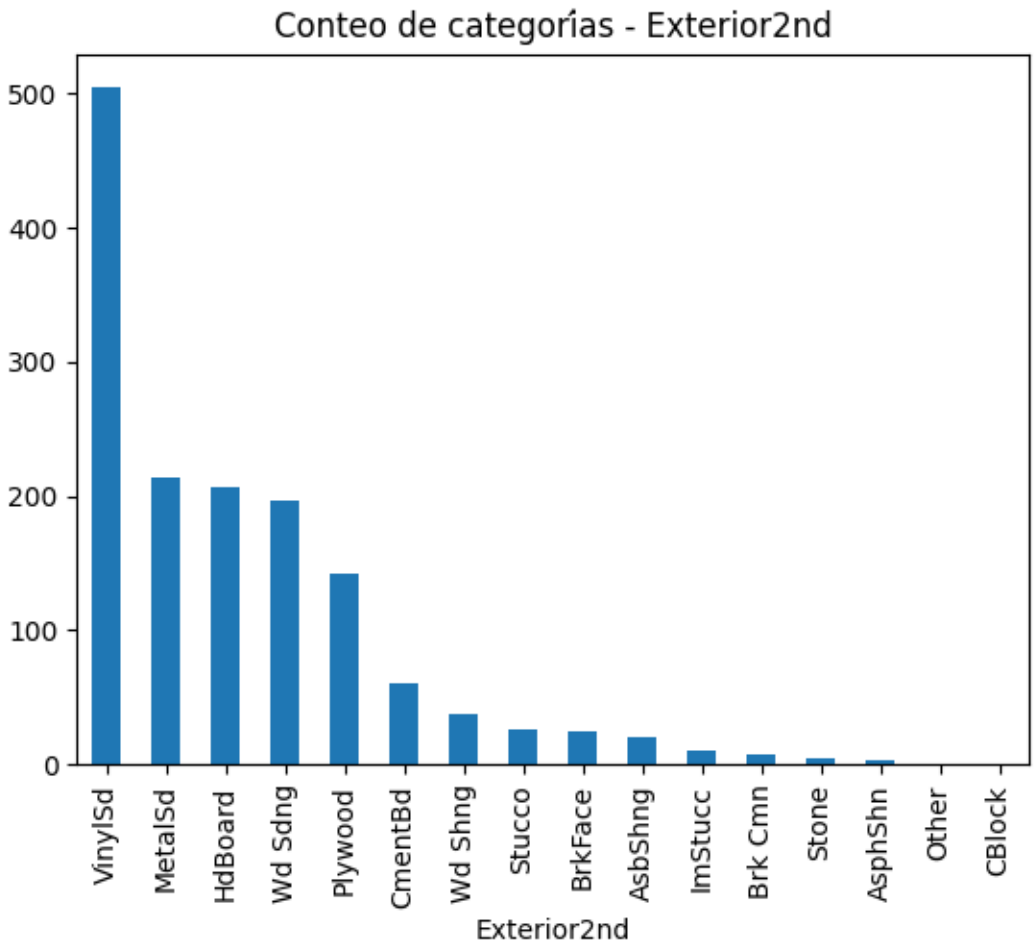

```
RoofStyle
Gable      1141
Hip         286
Flat         13
Gambrel      11
Mansard       7
Shed          2
Name: count, dtype: int64
```




```
RoofMatl
CompShg    1434
Tar&Grv      11
WdShngl       6
WdShake       5
Metal         1
Membran       1
Roll          1
ClyTile       1
Name: count, dtype: int64
```

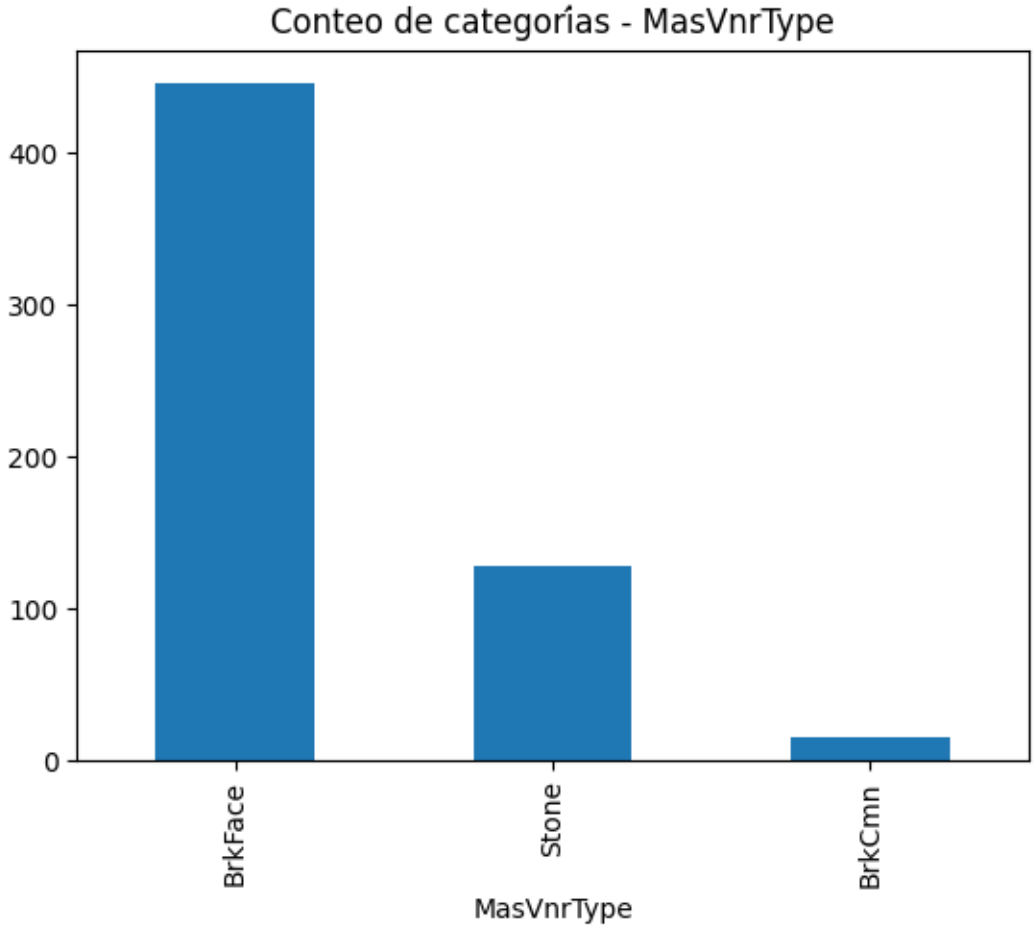



```
Exterior1st
VinylSd    515
HdBoard    222
MetalSd    220
Wd Sdng    206
Plywood    108
CemntBd     61
BrkFace     50
WdShing     26
Stucco      25
AsbShng     20
BrkComm      2
Stone        2
AsphShn      1
ImStucc      1
CBlock       1
Name: count, dtype: int64
```

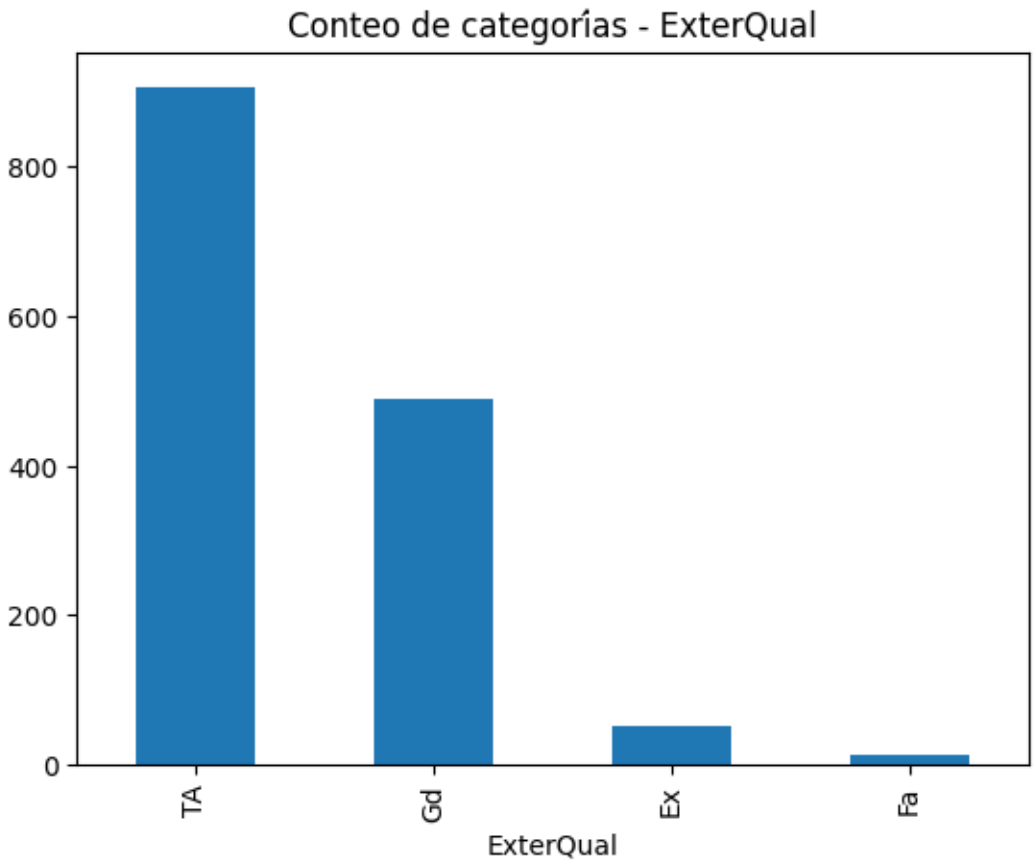



```
Exterior2nd
VinylSd    504
MetalSd    214
HdBoard    207
Wd Sdng    197
Plywood    142
CmentBd     60
Wd Shng     38
Stucco      26
BrkFace     25
AsbShng     20
ImStucc     10
Brk Cmn      7
Stone        5
AsphShn      3
Other        1
CBlock       1
```

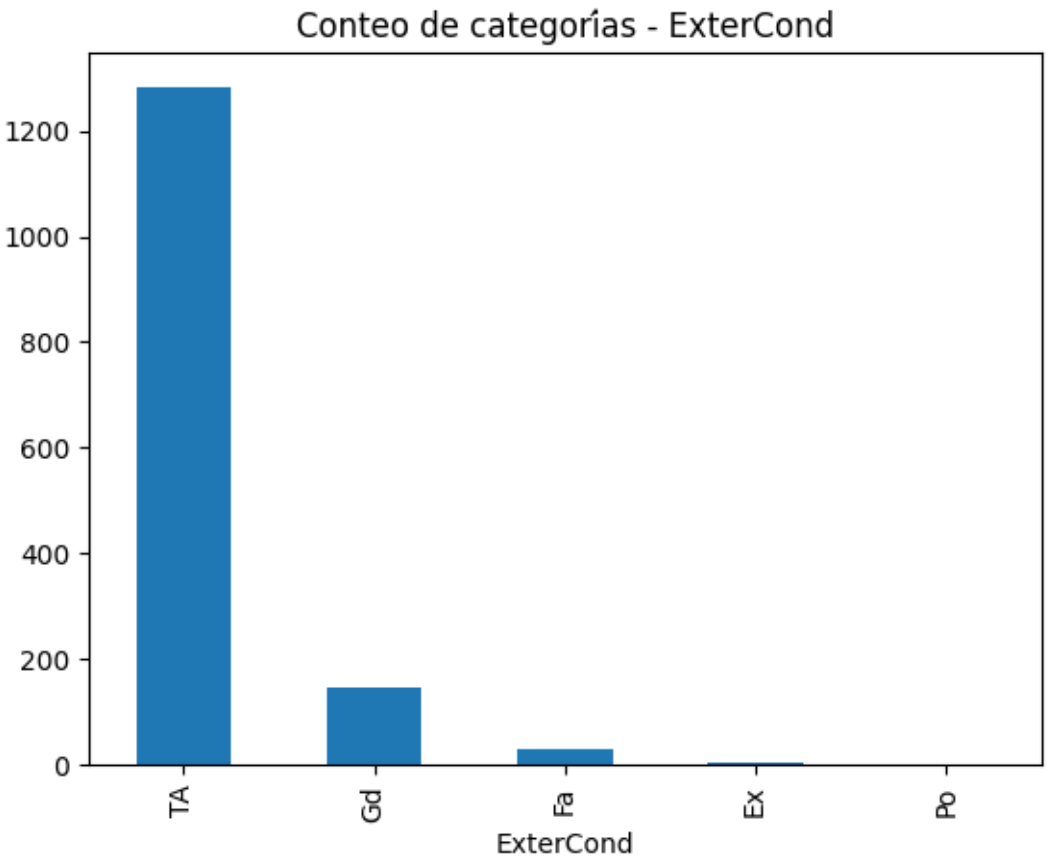
```
Name: count, dtype: int64
```



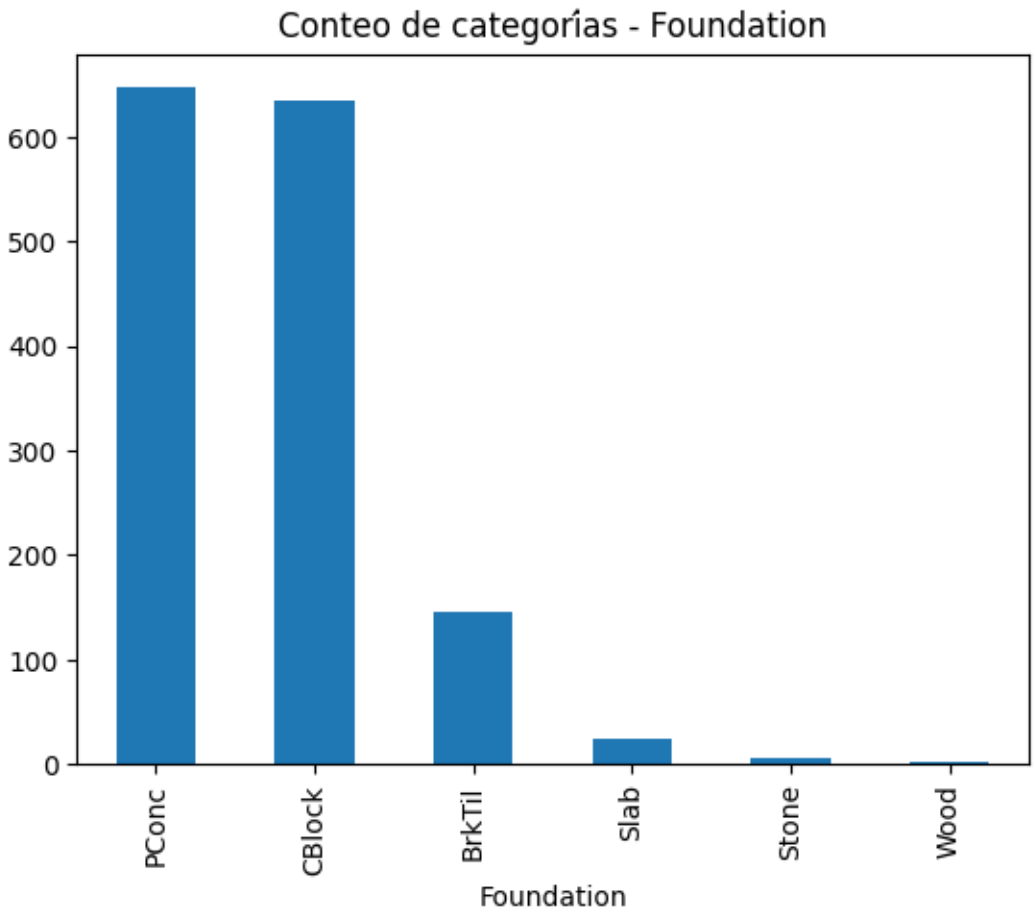

```
MasVnrType
BrkFace    445
Stone      128
BrkCmn      15
Name: count, dtype: int64
```



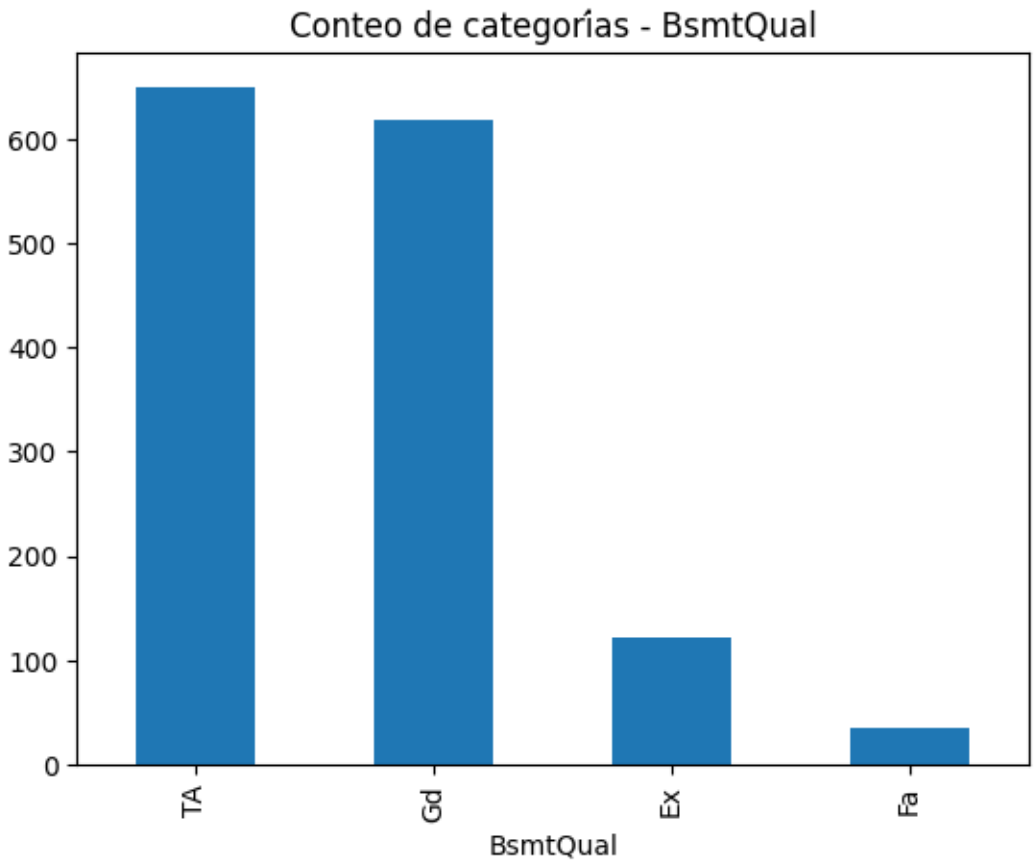

```
ExterQual
TA    906
Gd    488
Ex     52
Fa     14
Name: count, dtype: int64
```



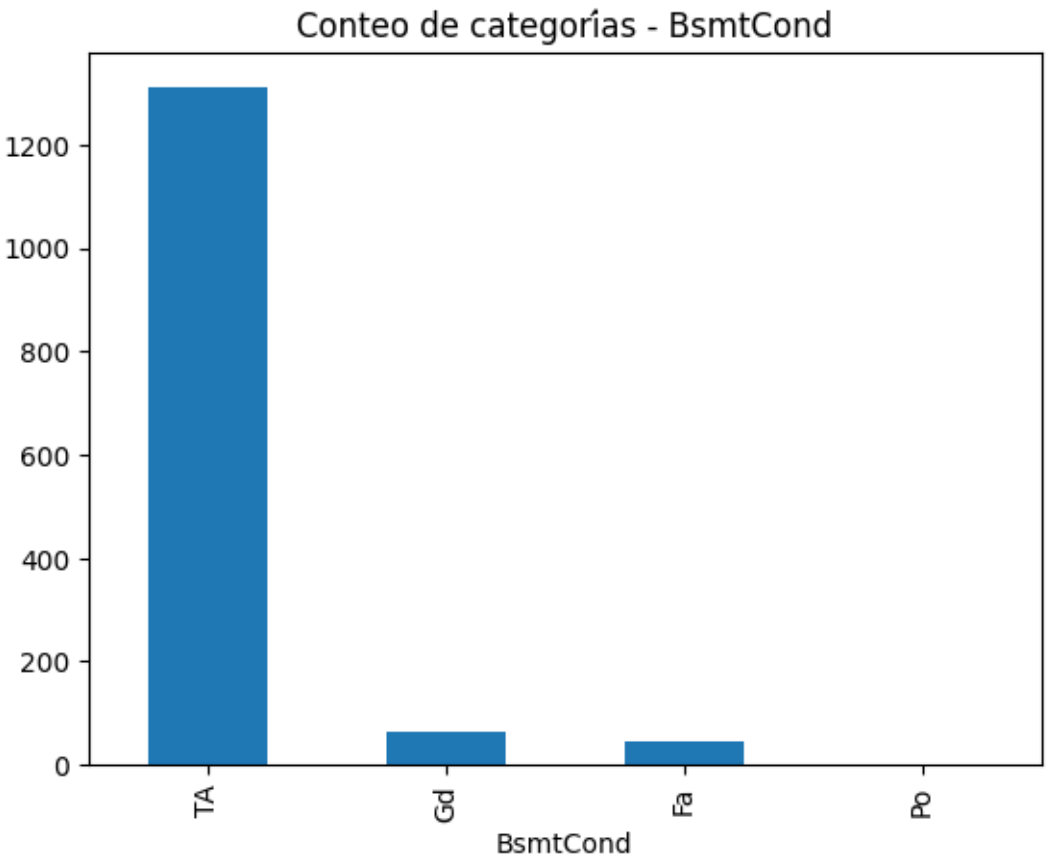

```
ExterCond
TA    1282
Gd     146
Fa      28
Ex       3
Po       1
Name: count, dtype: int64
```



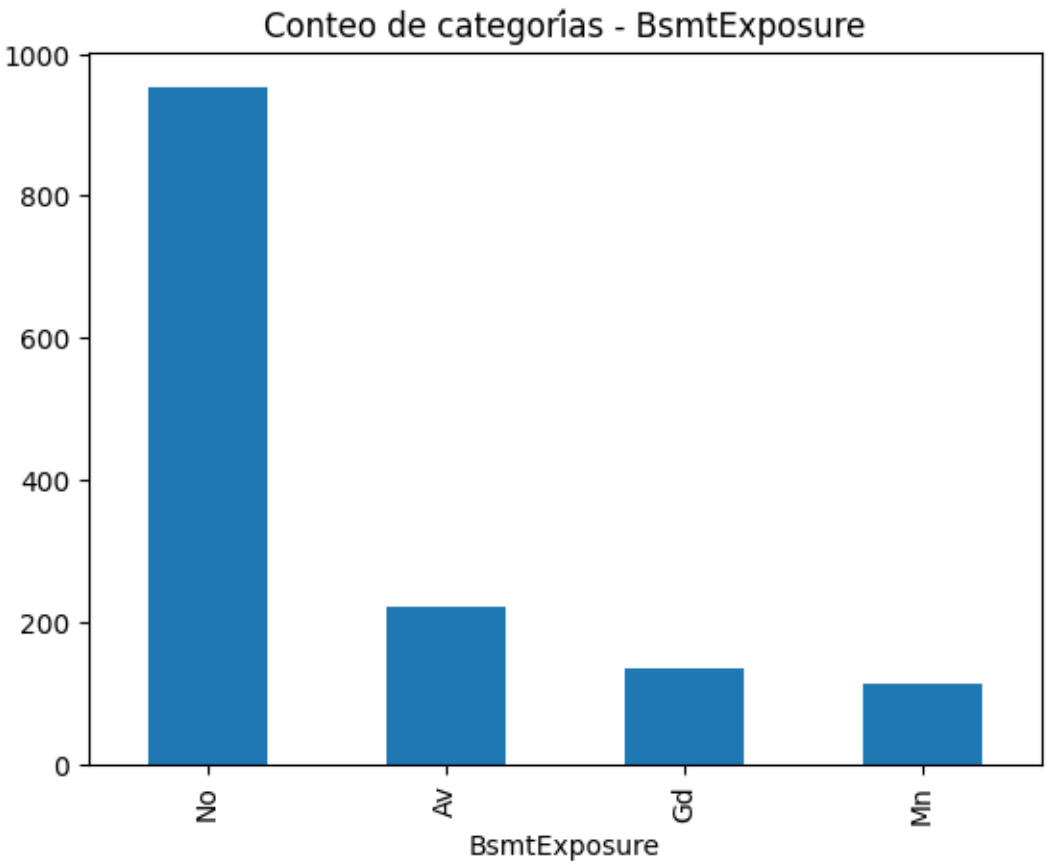

```
Foundation
PConc     647
CBlock    634
BrkTil    146
Slab       24
Stone       6
Wood        3
Name: count, dtype: int64
```



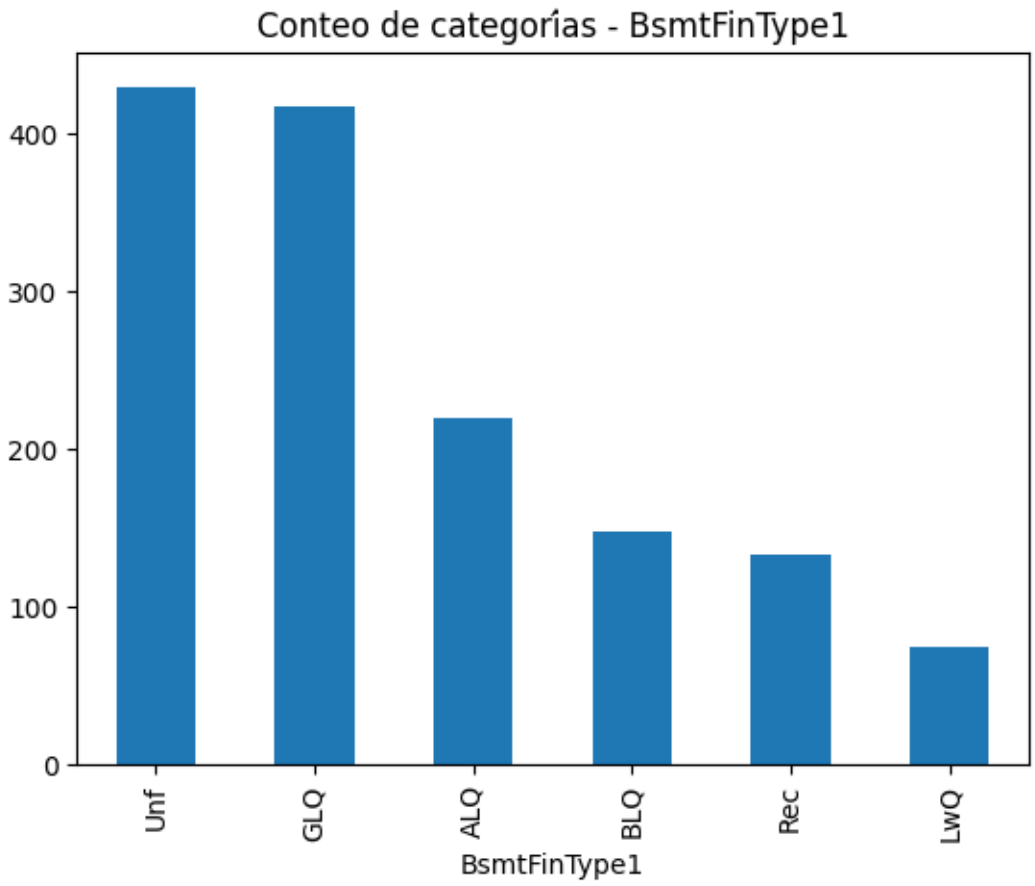

```
BsmtQual
TA    649
Gd    618
Ex    121
Fa     35
Name: count, dtype: int64
```



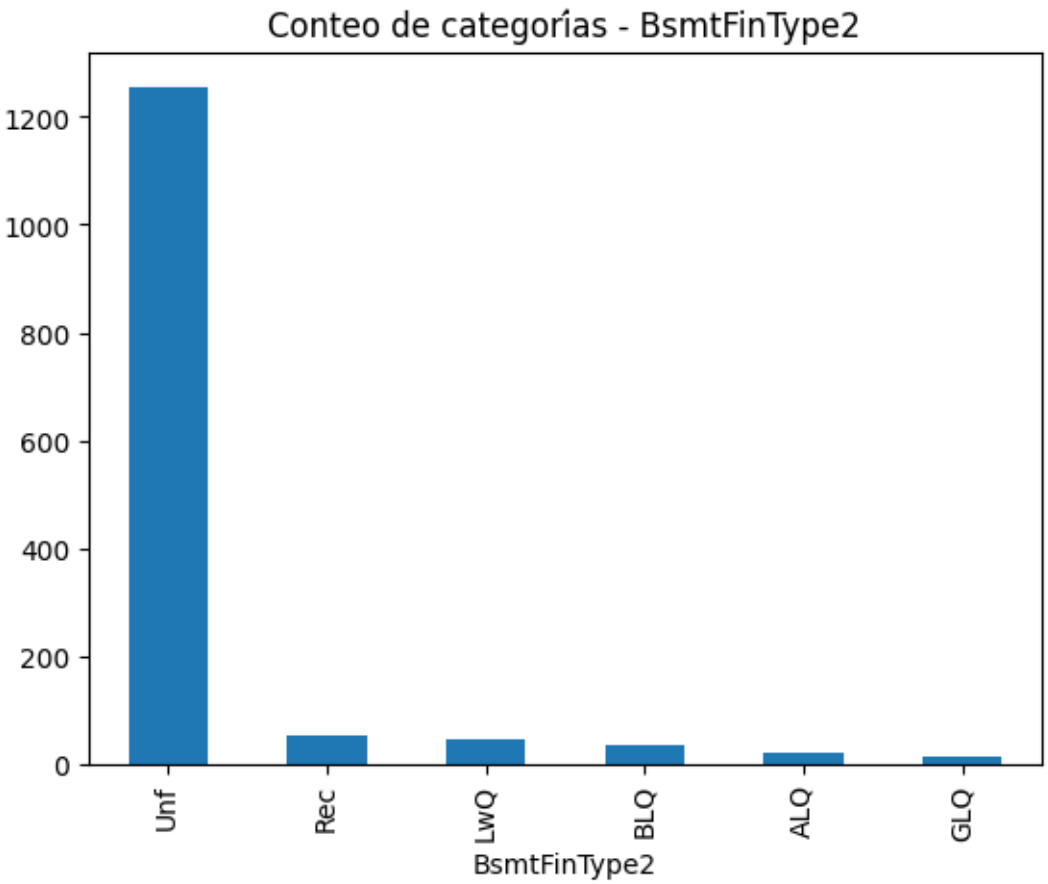

```
BsmtCond
TA    1311
Gd      65
Fa      45
Po       2
Name: count, dtype: int64
```



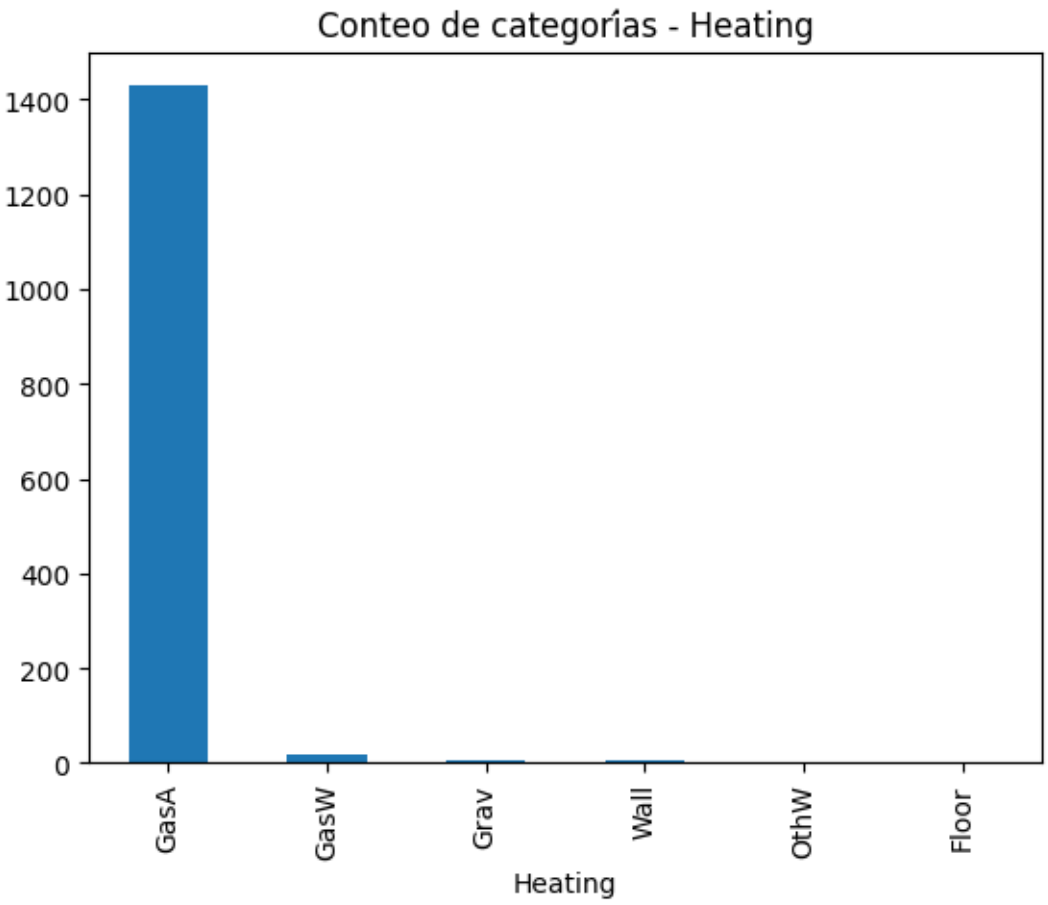

```
BsmtExposure
No    953
Av    221
Gd    134
Mn    114
Name: count, dtype: int64
```



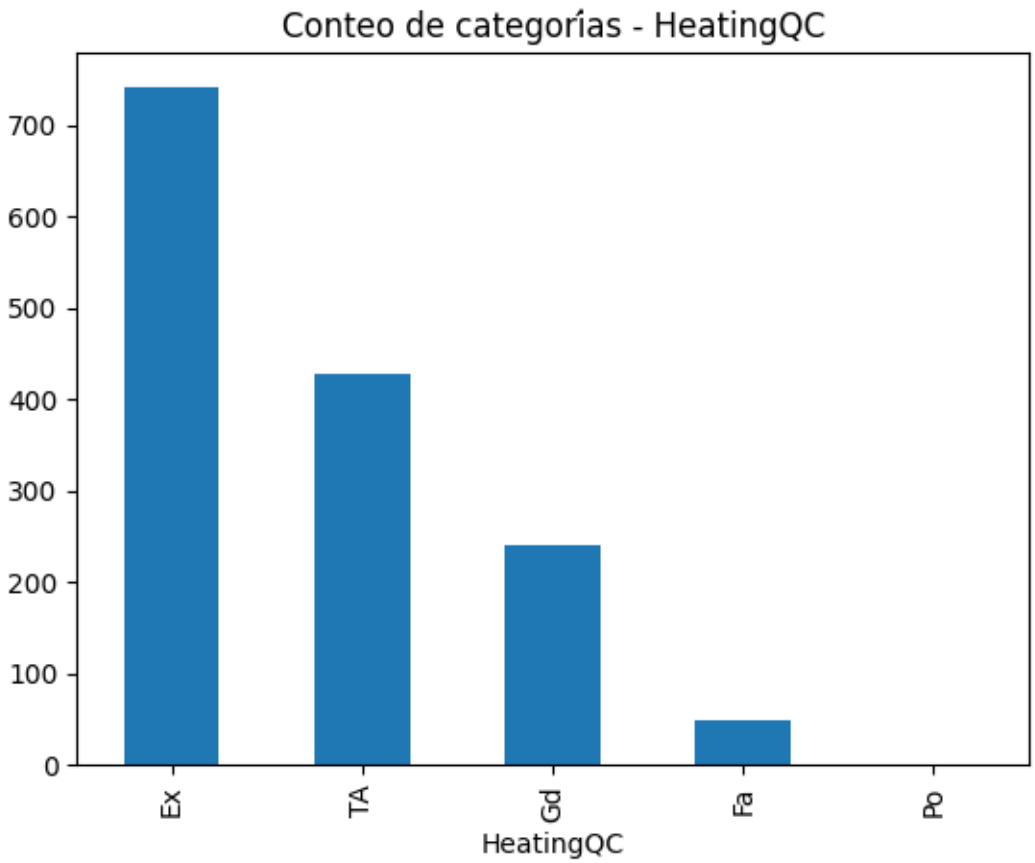

```
BsmtFinType1
Unf    430
GLQ    418
ALQ    220
BLQ    148
Rec    133
LwQ     74
Name: count, dtype: int64
```



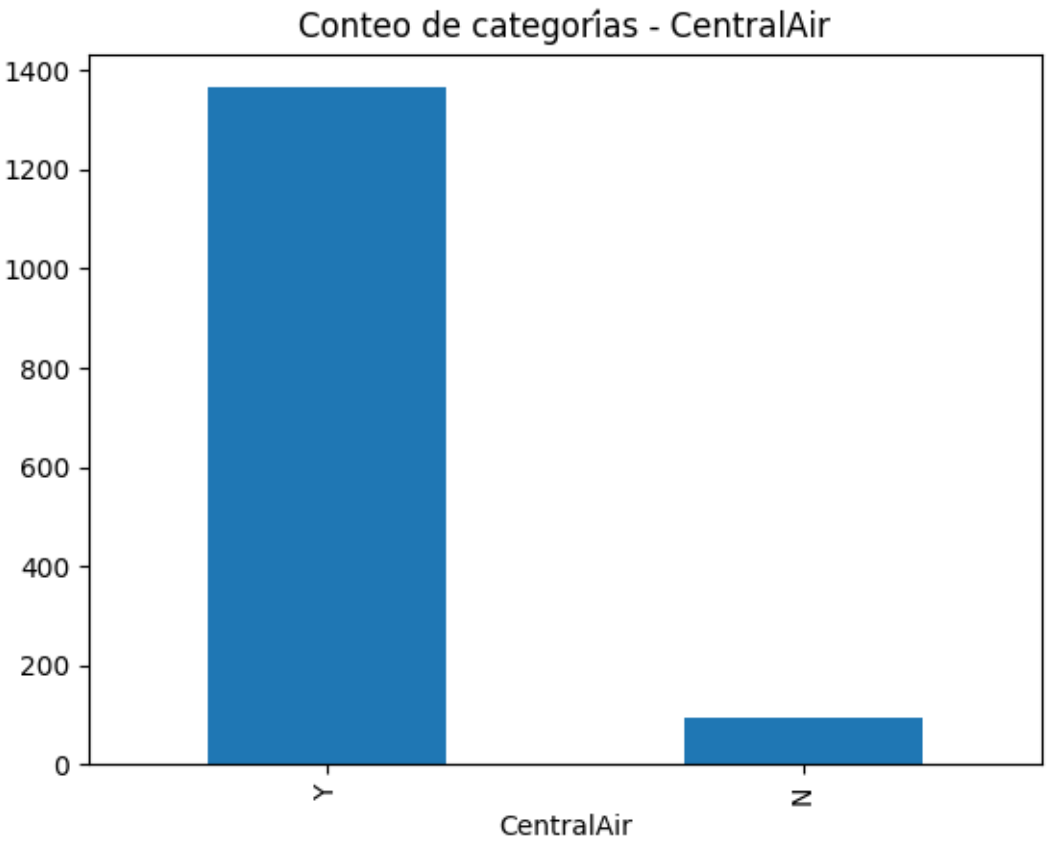

```
BsmtFinType2
Unf    1256
Rec      54
LwQ      46
BLQ      33
ALQ      19
GLQ      14
Name: count, dtype: int64
```



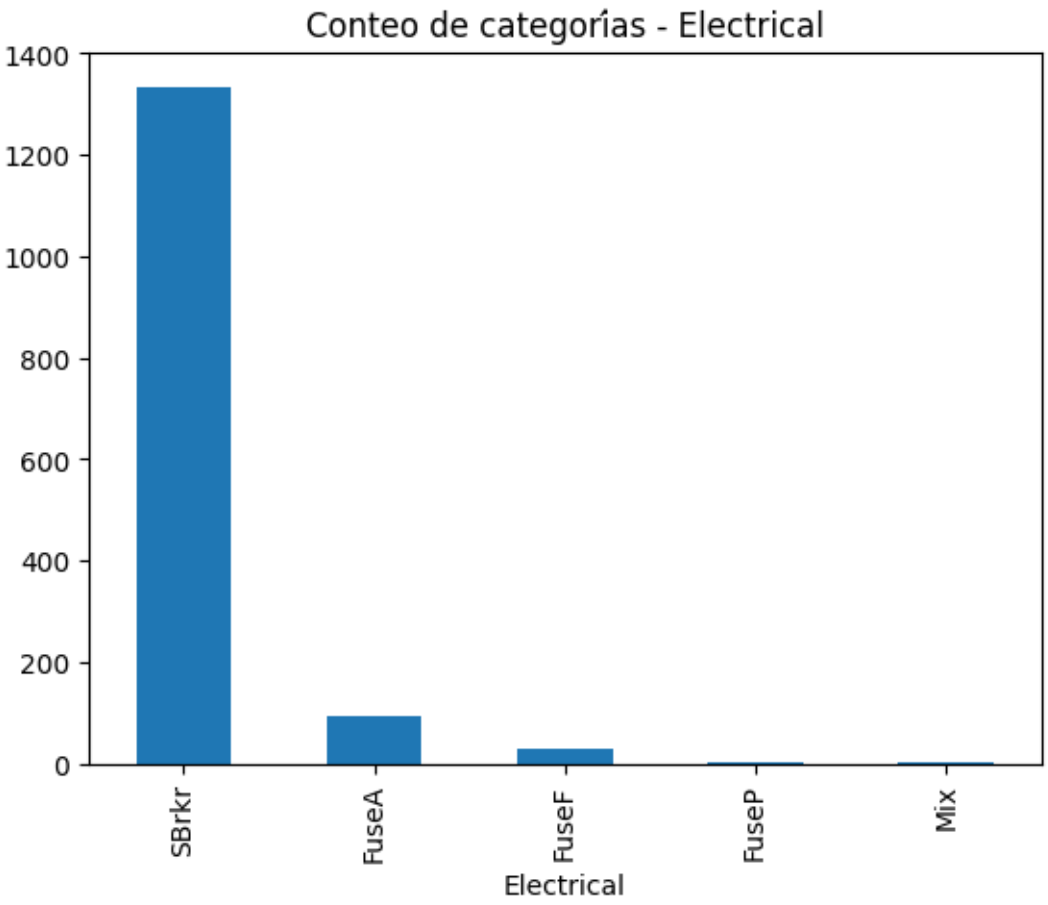

```
Heating
GasA     1428
GasW       18
Grav        7
Wall        4
OthW        2
Floor       1
Name: count, dtype: int64
```



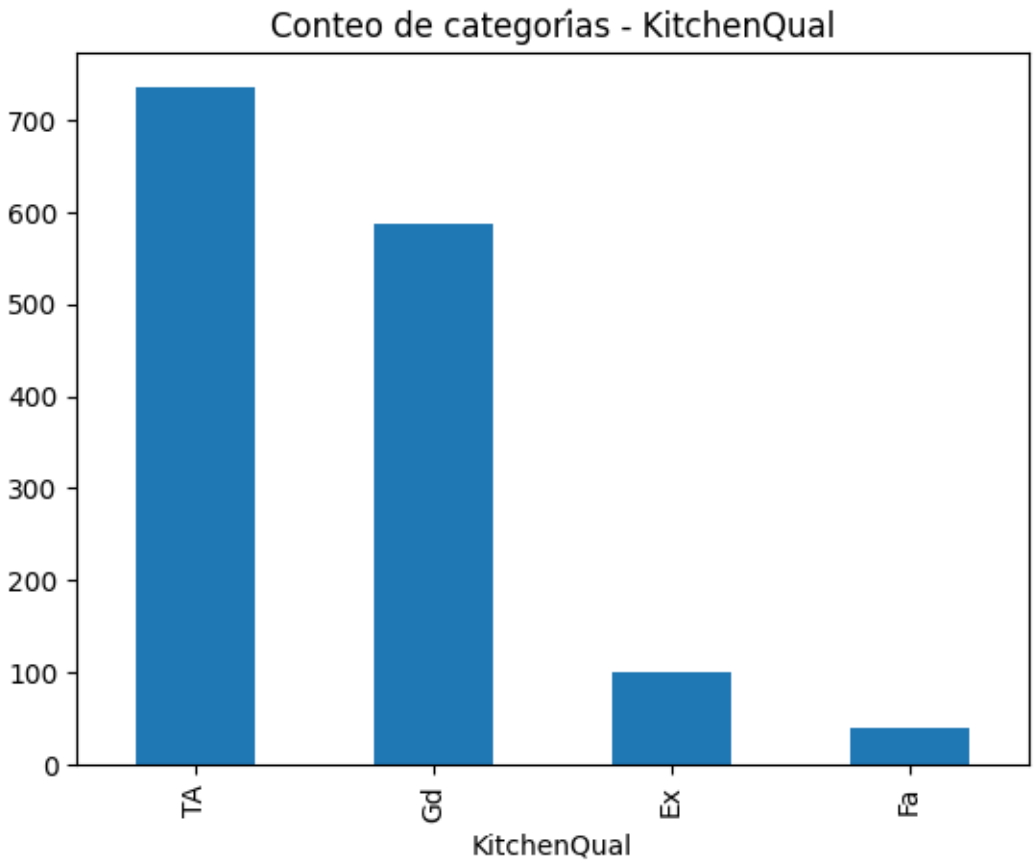

```
HeatingQC
Ex    741
TA    428
Gd    241
Fa     49
Po      1
Name: count, dtype: int64
```



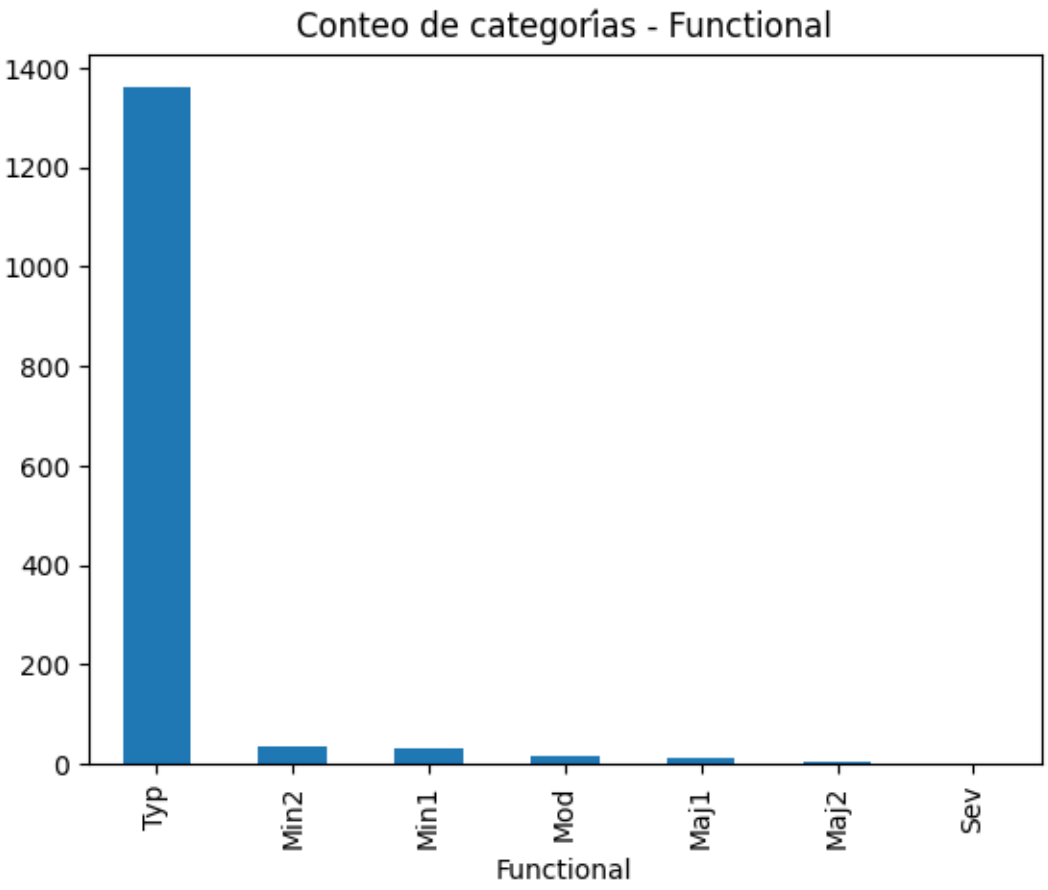

```
CentralAir
Y    1365
N      95
Name: count, dtype: int64
```



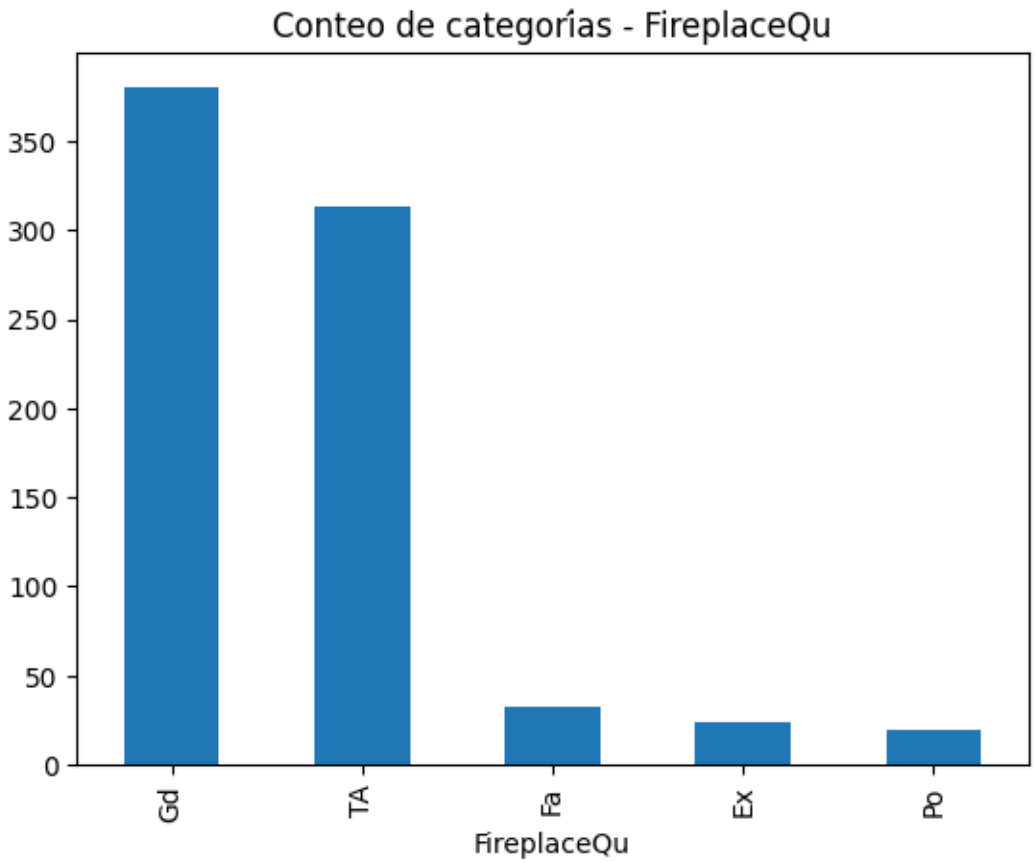

```
Electrical
SBrkr    1334
FuseA      94
FuseF      27
FuseP       3
Mix         1
Name: count, dtype: int64
```



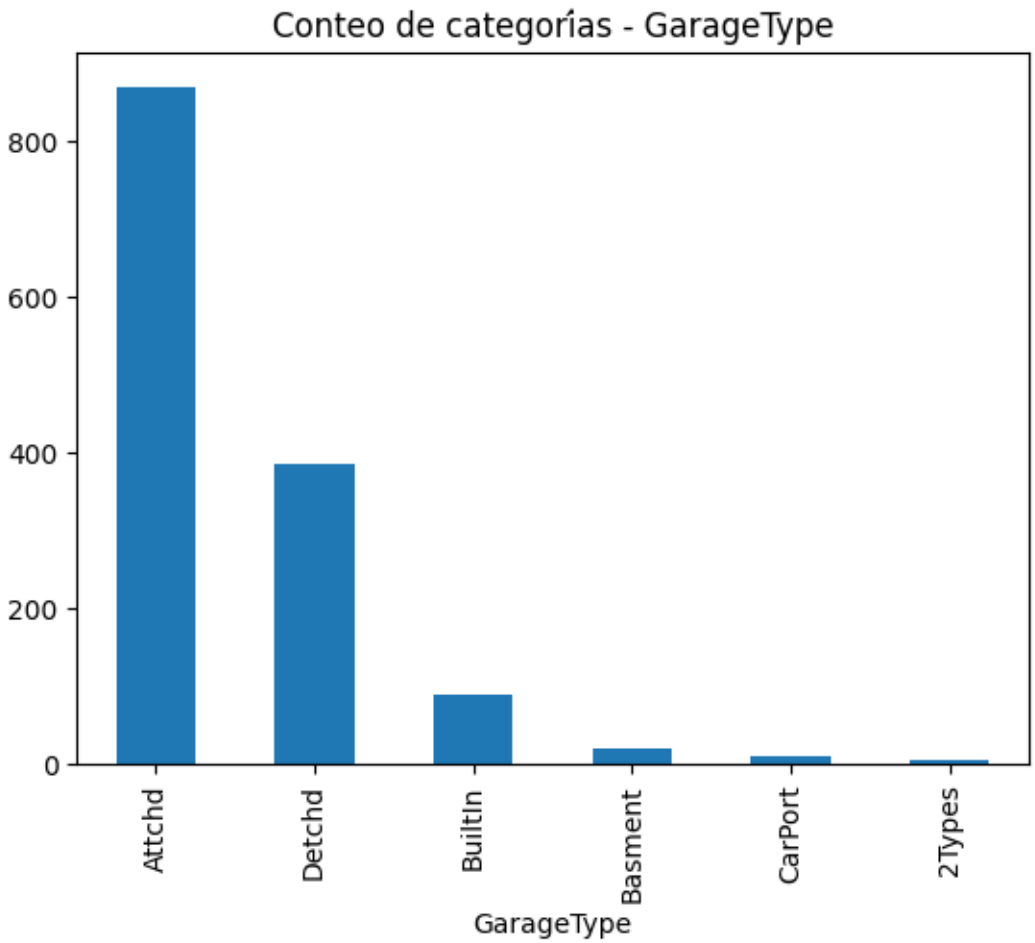

```
KitchenQual
TA    735
Gd    586
Ex    100
Fa     39
Name: count, dtype: int64
```



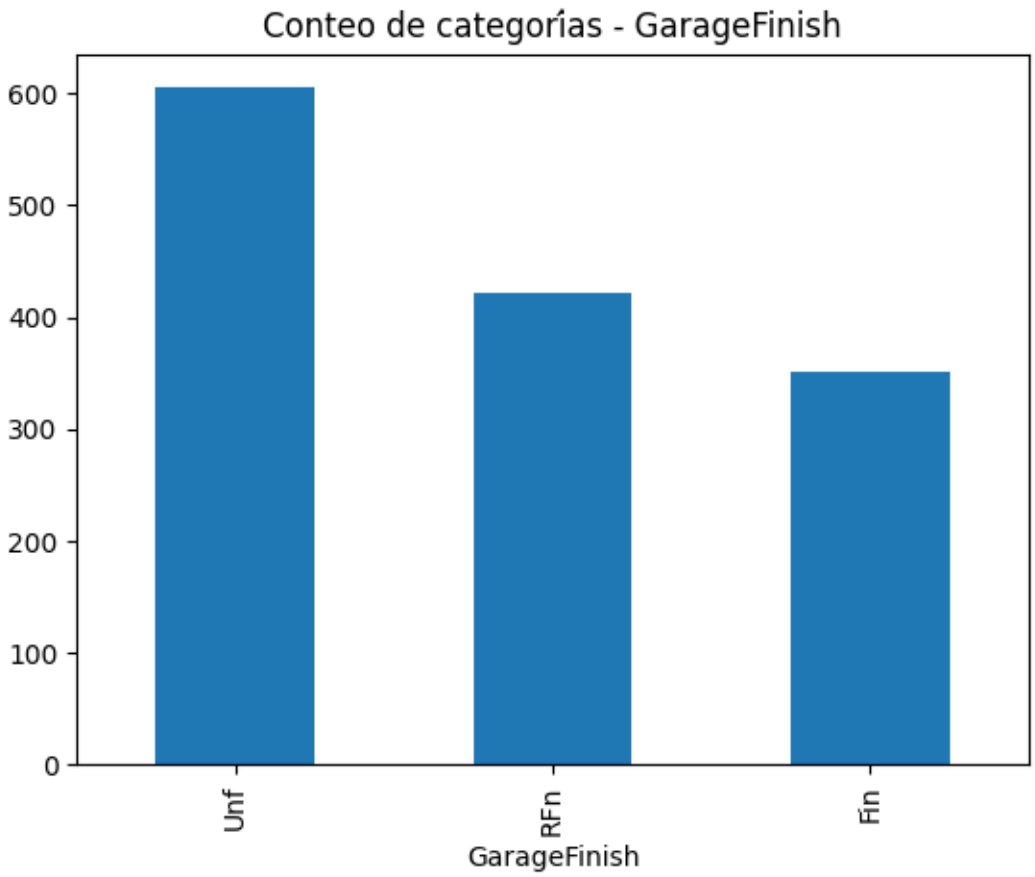

```
Functional
Typ     1360
Min2      34
Min1      31
Mod       15
Maj1      14
Maj2       5
Sev        1
Name: count, dtype: int64
```



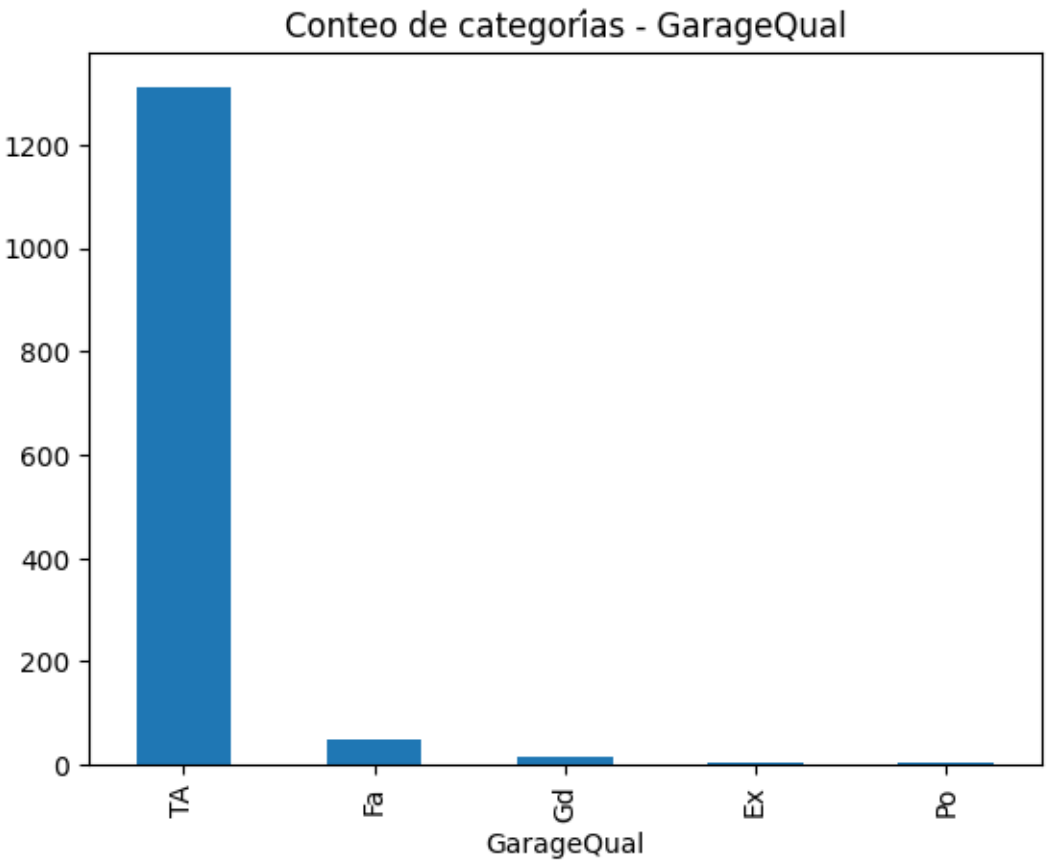

```
FireplaceQu
Gd    380
TA    313
Fa     33
Ex     24
Po     20
Name: count, dtype: int64
```



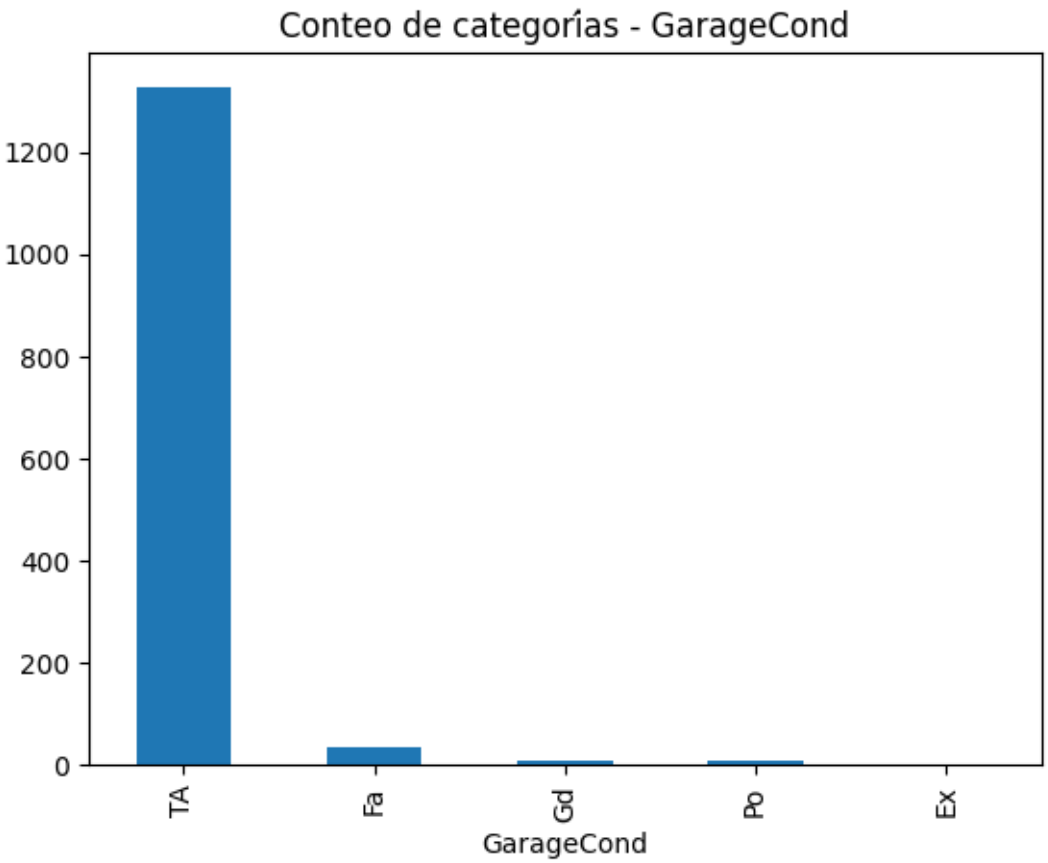

```
GarageType
Attchd     870
Detchd     387
BuiltIn     88
Basment     19
CarPort      9
2Types       6
Name: count, dtype: int64
```



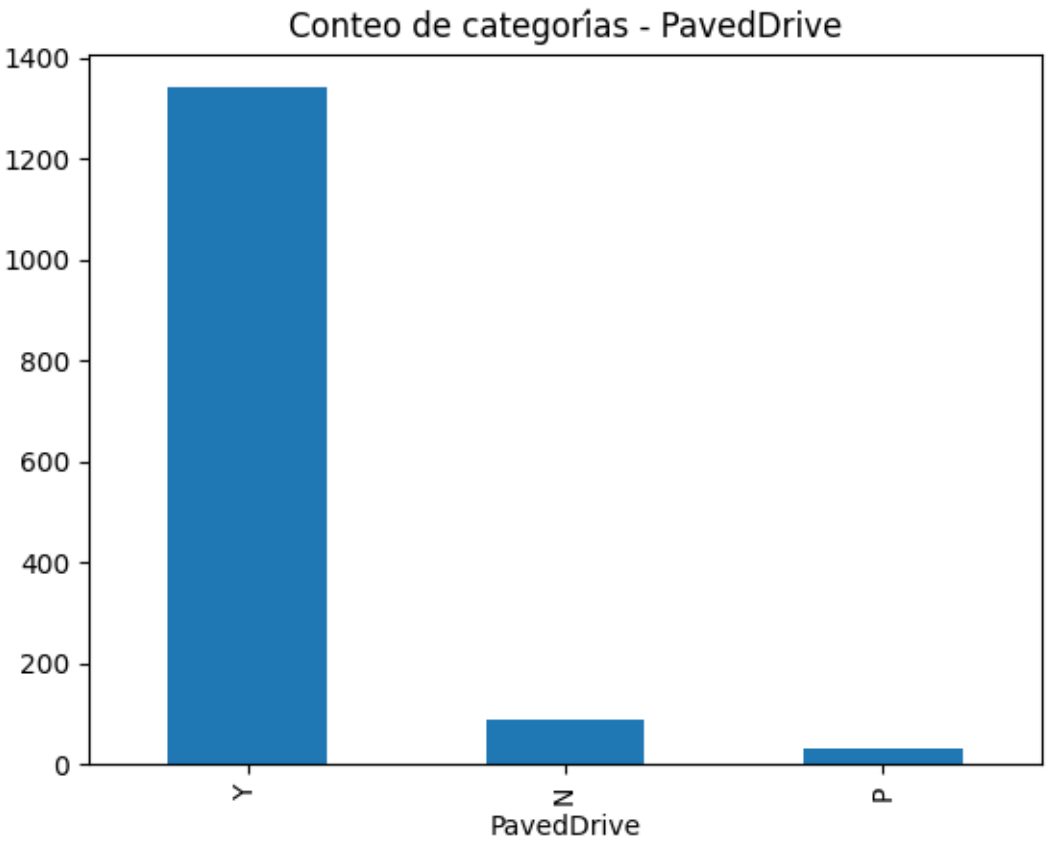

```
GarageFinish
Unf    605
RFn    422
Fin    352
Name: count, dtype: int64
```



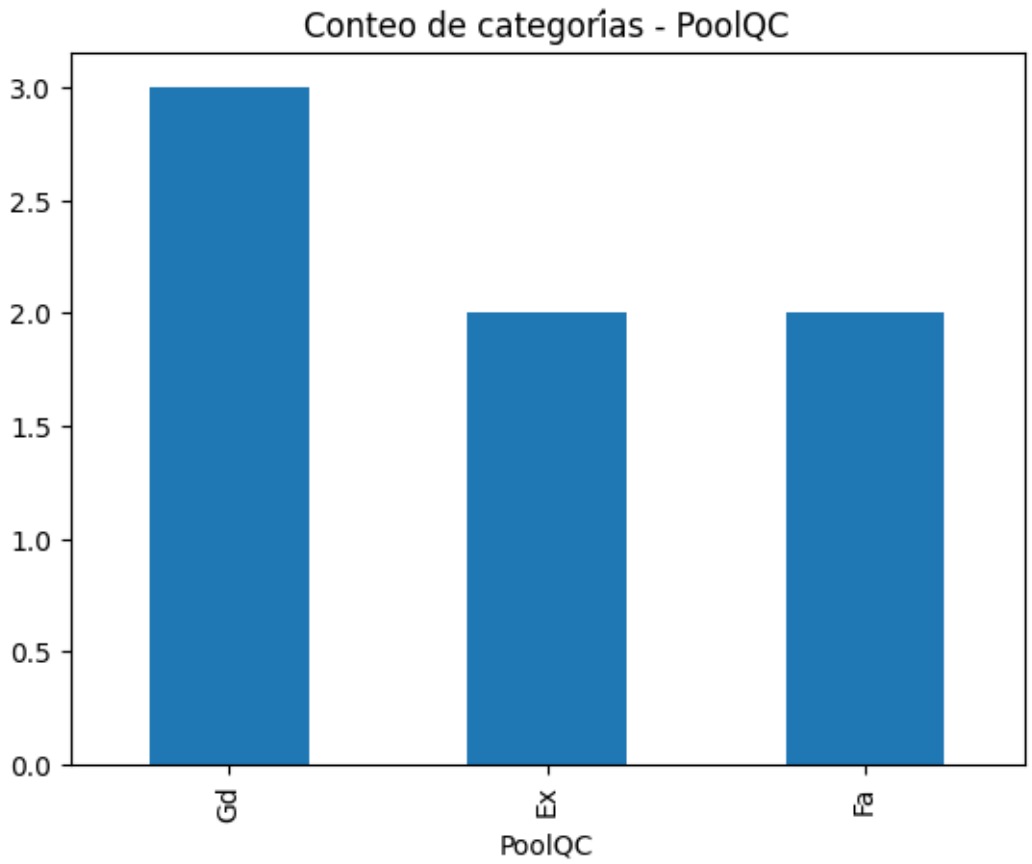

```
GarageQual
TA    1311
Fa      48
Gd      14
Ex       3
Po       3
Name: count, dtype: int64
```



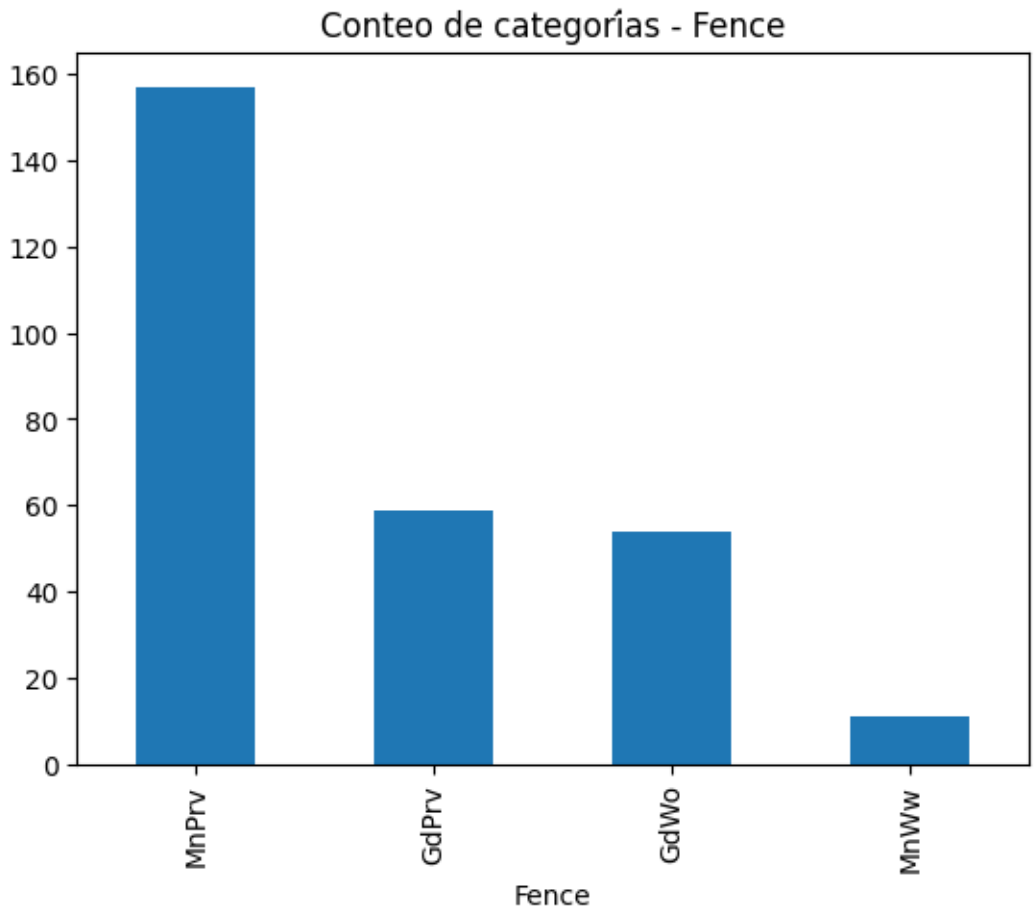

```
GarageCond
TA    1326
Fa      35
Gd       9
Po       7
Ex       2
Name: count, dtype: int64
```



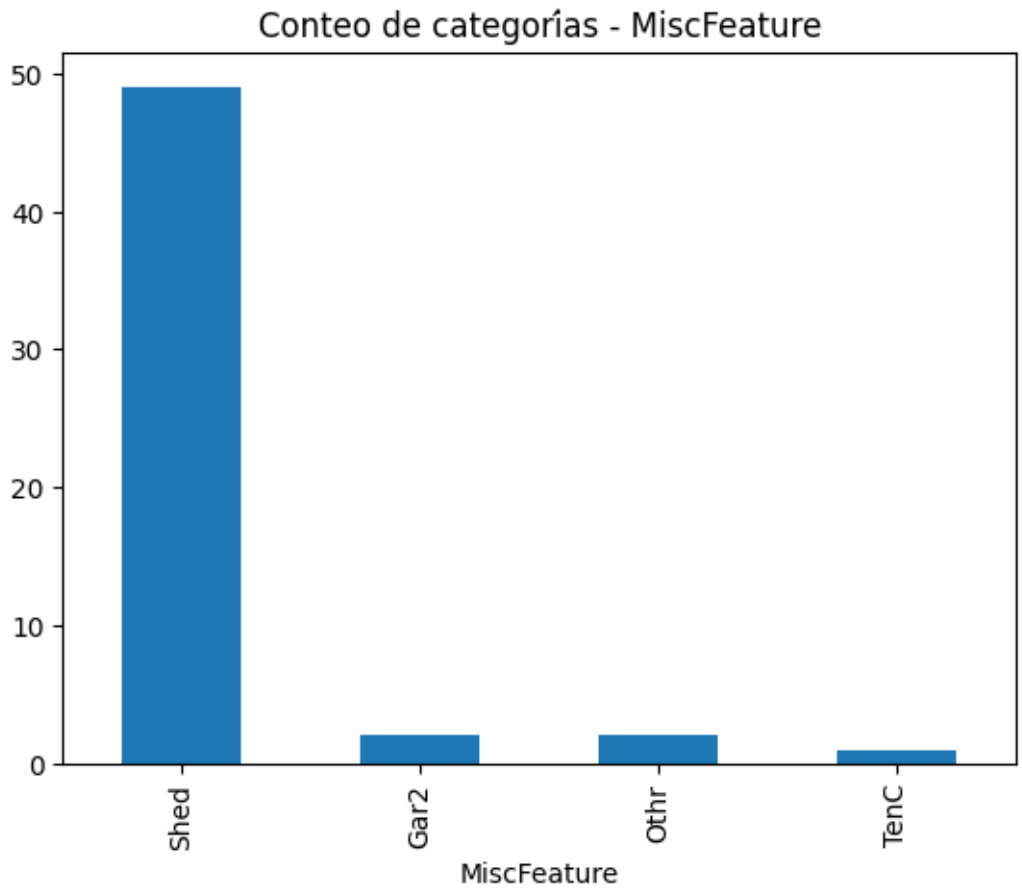

```
PavedDrive
Y    1340
N      90
P      30
Name: count, dtype: int64
```



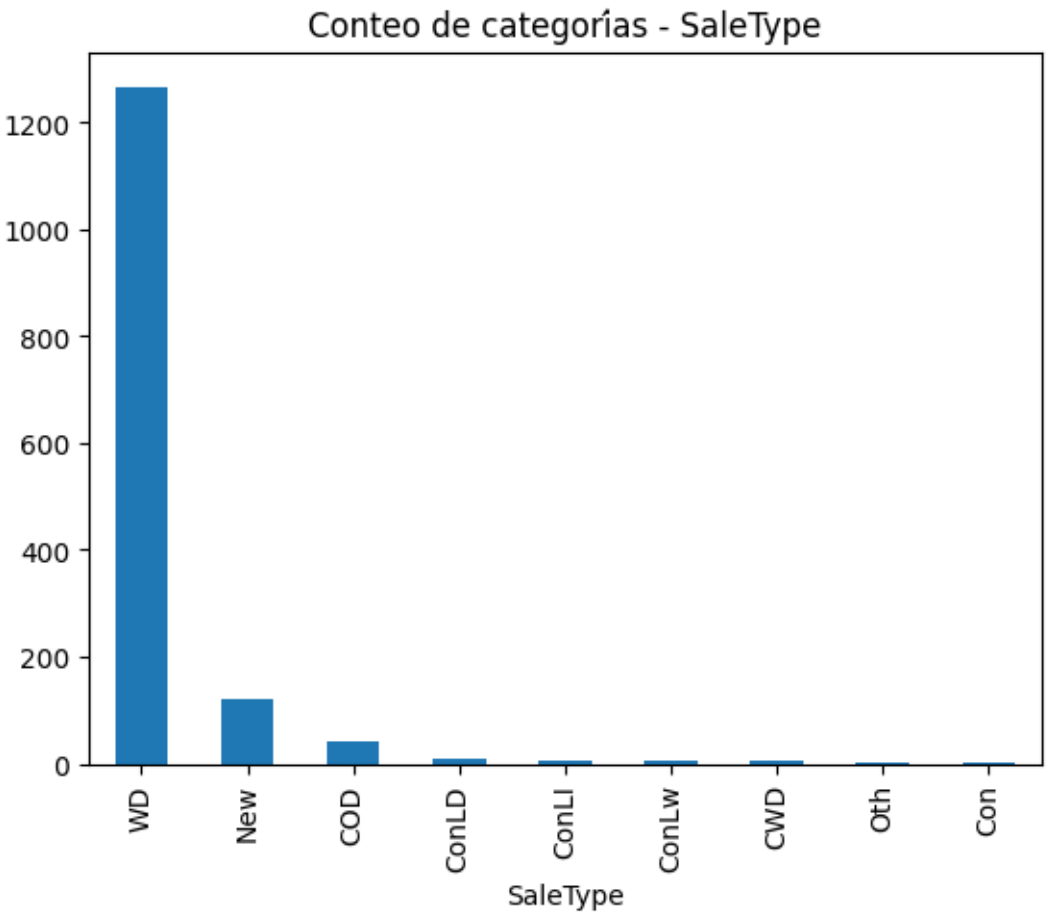

```
PoolQC
Gd    3
Ex    2
Fa    2
Name: count, dtype: int64
```



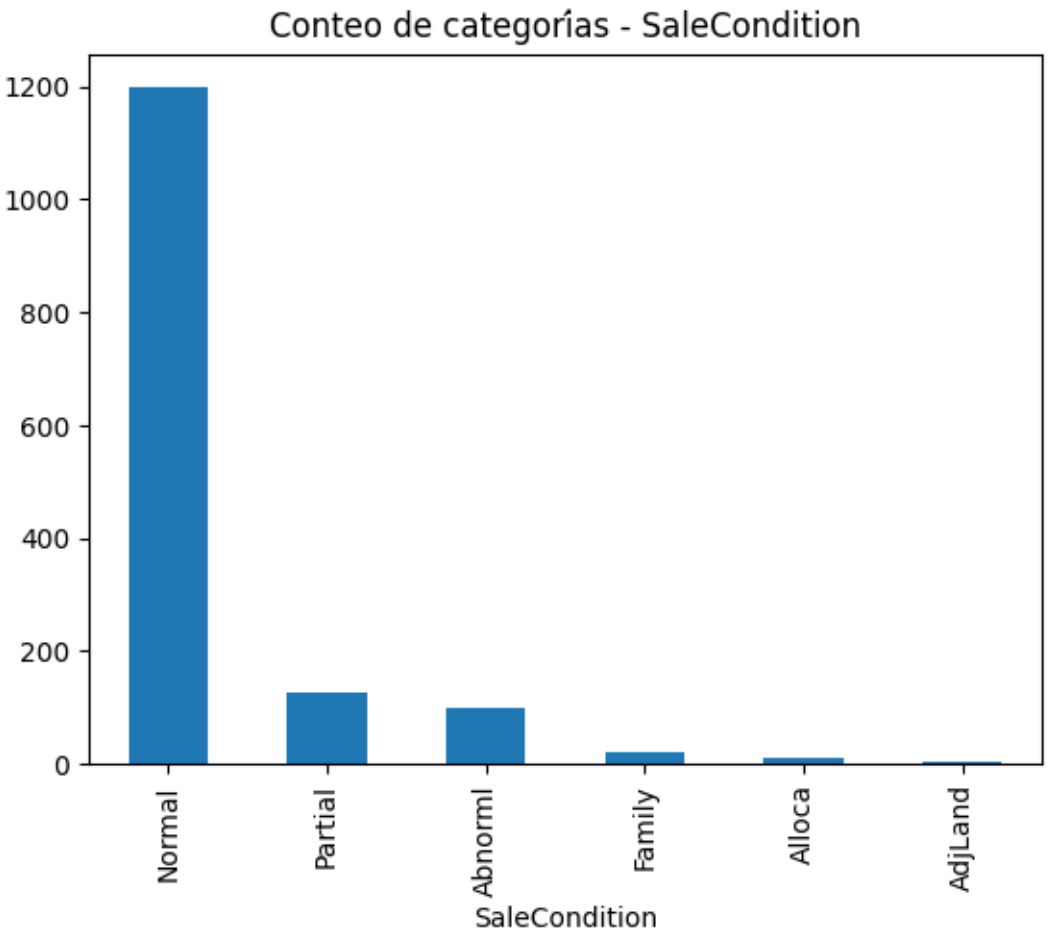

```
Fence
MnPrv    157
GdPrv     59
GdWo      54
MnWw      11
Name: count, dtype: int64
```




```
MiscFeature
Shed    49
Gar2     2
Othr     2
TenC     1
Name: count, dtype: int64
```




```
SaleType
WD        1267
New        122
COD         43
ConLD        9
ConLI        5
ConLw        5
CWD          4
Oth          3
Con          2
Name: count, dtype: int64
```




```
SaleCondition
Normal     1198
Partial     125
Abnorml     101
Family       20
Alloca       12
AdjLand       4
Name: count, dtype: int64
```

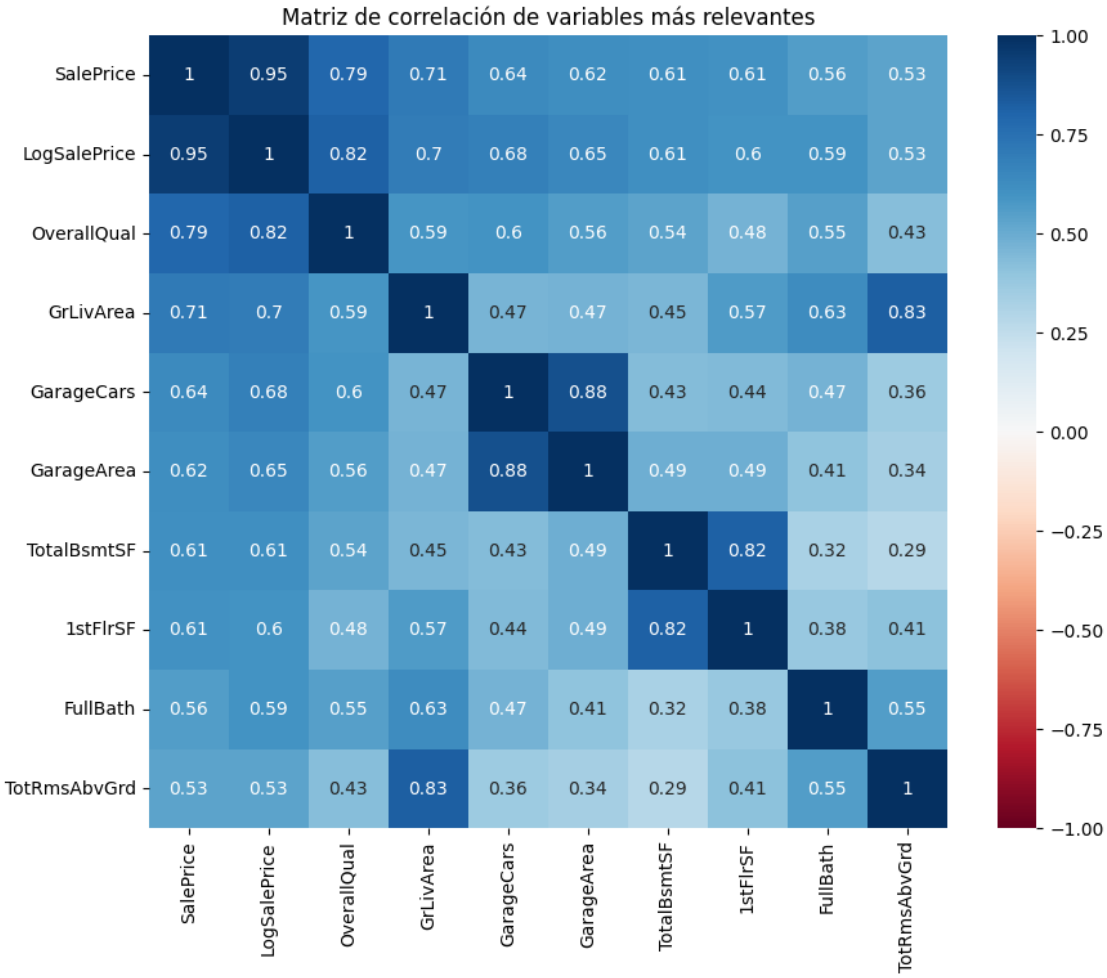
1.7 7. Análisis bivariante: correlación con la variable objetivo (SalePrice)

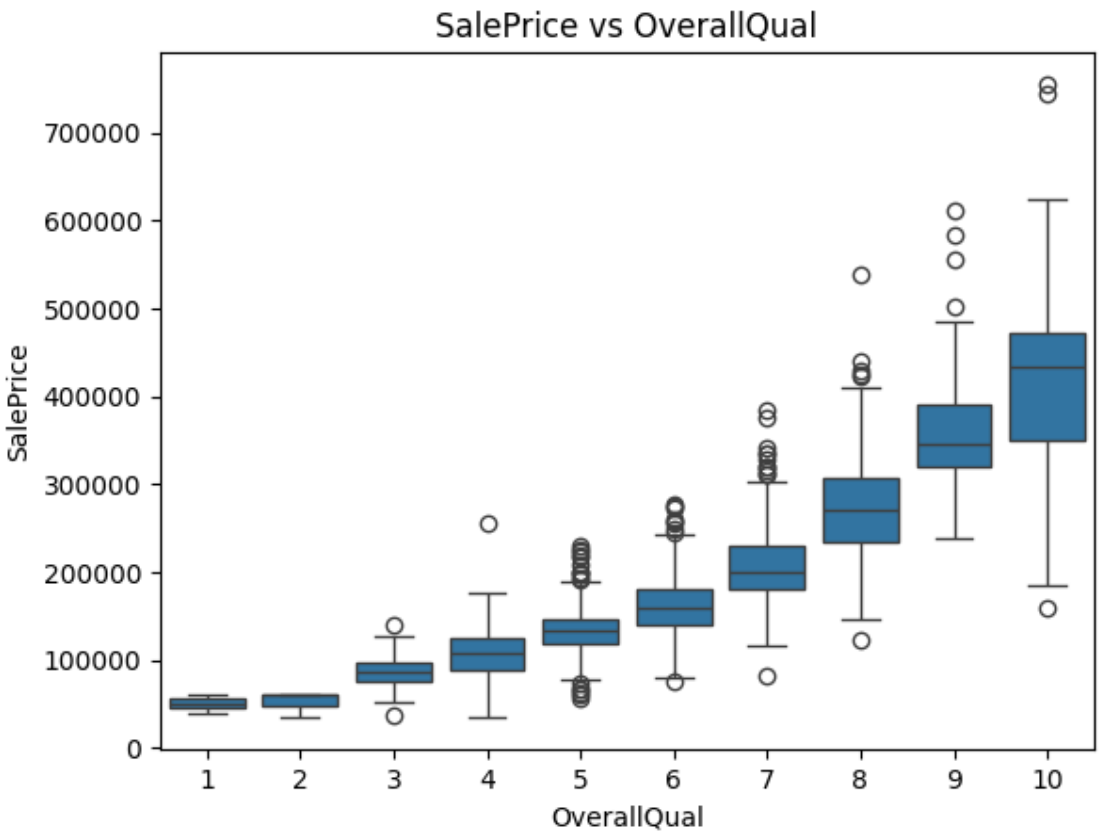
1. Calculamos la correlación (Pearson) para variables numéricas.
2. Graficamos un heatmap de las más correlacionadas con SalePrice.
3. Vemos ejemplos de boxplots o scatterplots con variables que más destacan.

[9]:
```
# Filtrar únicamente las columnas numéricas (asegúrate de que 'SalePrice' sea␣
↪numérica).
numeric_df = train.select_dtypes(include=['int64','float64'])
```

```
# Matriz de correlación para ver relación con SalePrice (solo columnas␣
 ↪numéricas)
corr_matrix = numeric_df.corr()

# Seleccionamos las 10 variables con mayor correlación (en valor absoluto) con␣
 ↪SalePrice
top_corr = corr_matrix['SalePrice'].abs().sort_values(ascending=False).head(10)
print("Variables con mayor correlación con SalePrice:\n", top_corr)

# Heatmap con las variables más correlacionadas
top_vars = top_corr.index
plt.figure(figsize=(10, 8))

sns.heatmap(numeric_df[top_vars].corr(),
            annot=True, cmap='RdBu', vmin=-1, vmax=1)
plt.title('Matriz de correlación de variables más relevantes')
plt.show()

# Ejemplo de análisis con OverallQual o GrLivArea:
# Estas variables deben existir en train.
# Si no las filtras, asegúrate de que la columna sea numérica.
sns.boxplot(x='OverallQual', y='SalePrice', data=train)
plt.title('SalePrice vs OverallQual')
plt.show()
```

```
Variables con mayor correlación con SalePrice:
 SalePrice       1.000000
LogSalePrice    0.948374
OverallQual     0.790982
GrLivArea       0.708624
GarageCars      0.640409
GarageArea      0.623431
TotalBsmtSF     0.613581
1stFlrSF        0.605852
FullBath        0.560664
TotRmsAbvGrd    0.533723
Name: SalePrice, dtype: float64
```


Matriz de correlación de variables más relevantes
SalePrice 1 0.95 0.79 0.71 0.64 0.62 0.61 0.61 0.56 0.53
LogSalePrice 0.95 1 0.82 0.7 0.68 0.65 0.61 0.6 0.59 0.53
OverallQual 0.79 0.82 1 0.59 0.6 0.56 0.54 0.48 0.55 0.43
GrLivArea 0.71 0.7 0.59 1 0.47 0.47 0.45 0.57 0.63 0.83
GarageCars 0.64 0.68 0.6 0.47 1 0.88 0.43 0.44 0.47 0.36
GarageArea 0.62 0.65 0.56 0.47 0.88 1 0.49 0.49 0.41 0.34
TotalBsmtSF 0.61 0.61 0.54 0.45 0.43 0.49 1 0.82 0.32 0.29
1stFlrSF 0.61 0.6 0.48 0.57 0.44 0.49 0.82 1 0.38 0.41
FullBath 0.56 0.59 0.55 0.63 0.47 0.41 0.32 0.38 1 0.55
TotRmsAbvGrd 0.53 0.53 0.43 0.83 0.36 0.34 0.29 0.41 0.55 1
SalePrice
LogSalePrice
OverallQual
GrLivArea
GarageCars
GarageArea
TotalBsmtSF
1stFlrSF
FullBath
TotRmsAbvGrd
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



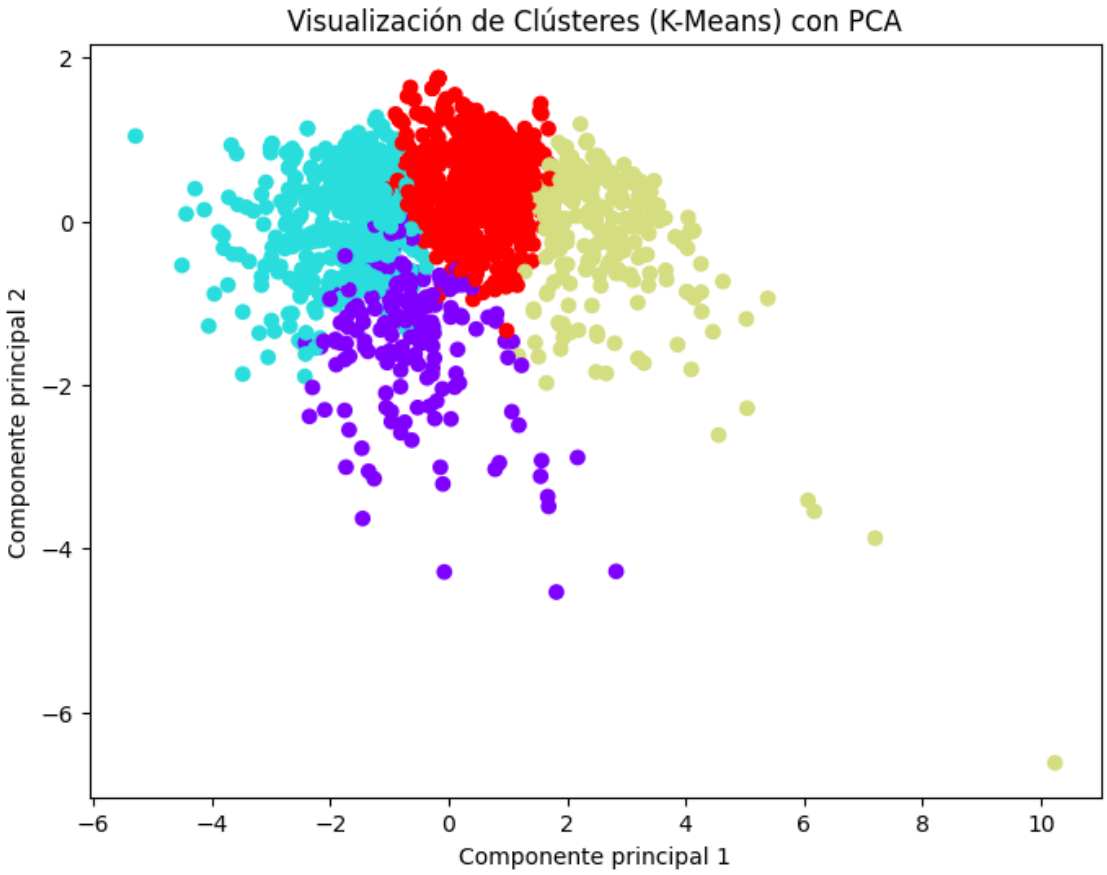
1.8 8. Análisis de agrupamiento (Clustering)

Usamos K-Means para crear grupos de casas similares y describirlos.

[10]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# Selecciona algunas variables numéricas representativas
cluster_features = ['OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF',␣
 ↪'YearBuilt']
df_cluster = train[cluster_features].dropna()

# Escalado
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_cluster)

# K-Means (k=4 como ejemplo)
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X_scaled)
```

```
labels = kmeans.labels_

df_cluster['Cluster'] = labels
cluster_summary = df_cluster.groupby('Cluster')[cluster_features].mean()
print(cluster_summary)

# PCA para visualización en 2D
pca = PCA(n_components=2)
pca_components = pca.fit_transform(X_scaled)
plt.figure(figsize=(8, 6))
plt.scatter(pca_components[:, 0], pca_components[:, 1], c=labels,␣
 ↪cmap='rainbow')
plt.xlabel('Componente principal 1')
plt.ylabel('Componente principal 2')
plt.title('Visualización de Clústeres (K-Means) con PCA')
plt.show()
```

```
         OverallQual    GrLivArea  GarageCars  TotalBsmtSF    YearBuilt
Cluster
0           5.898305  1893.214689    1.638418   889.451977  1927.355932
1           4.852814  1085.935065    1.021645   813.045455  1951.588745
2           8.040161  2098.714859    2.650602  1620.738956  1999.867470
3           6.323427  1491.601399    2.024476  1061.578671  1988.300699
```



1.9 9. División del Dataset en conjuntos de Entrenamiento y Prueba

Separamos aleatoriamente el conjunto de datos preprocesados ya que se trata de una regresión y no de una clasificación o balanceo.

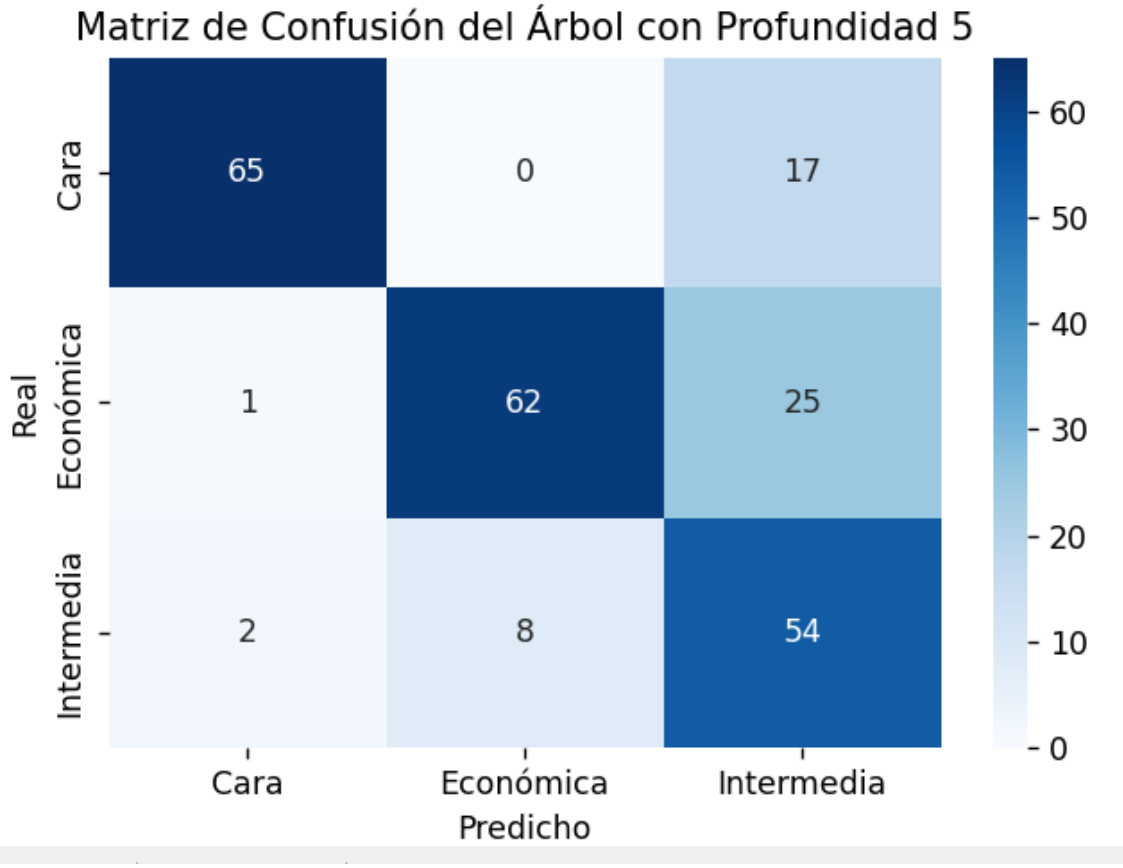
[28]:
```
# 1. Eliminar columnas con más del 50% de valores nulos
threshold = 0.5
missing_fraction = train.isnull().sum() / len(train)
columns_to_drop = missing_fraction[missing_fraction > threshold].index
df_cleaned = train.drop(columns=columns_to_drop)

# 2. Imputar valores faltantes
# Para variables numéricas, rellenamos con la mediana
num_cols = df_cleaned.select_dtypes(include=['int64', 'float64']).columns
df_cleaned[num_cols] = df_cleaned[num_cols].fillna(df_cleaned[num_cols].
 ↪median())

# Para variables categóricas, rellenamos con el valor más frecuente (moda)
cat_cols = df_cleaned.select_dtypes(include=['object']).columns
```

```
df_cleaned[cat_cols] = df_cleaned[cat_cols].apply(lambda x: x.fillna(x.
 ↪mode()[0]))

# 3. Codificar variables categóricas
label_encoders = {}
for col in cat_cols:
    le = LabelEncoder()
    df_cleaned[col] = le.fit_transform(df_cleaned[col])
    label_encoders[col] = le  # Guardamos los codificadores por si se necesitan␣
 ↪luego

# 4. Escalar variables numéricas
scaler = StandardScaler()
df_cleaned[num_cols] = scaler.fit_transform(df_cleaned[num_cols])

# 5. Dividir en entrenamiento y prueba (80% - 20%)
train_df, test_df = train_test_split(df_cleaned, test_size=0.2, train_size=0.8)

# Guardar los conjuntos preprocesados, comentamos esto porque se crean los␣
 ↪archivos csv cada que se ejecuta
#train_path = "train_preprocessed.csv"
#test_path = "test_preprocessed.csv"

#train_df.to_csv(train_path, index=False)
#test_df.to_csv(test_path, index=False)

# Mostrar el número de filas en cada conjunto
print(f"Tamaño del conjunto de entrenamiento: {len(train_df)} filas")
print(f"Tamaño del conjunto de prueba: {len(test_df)} filas")
```

```
Tamaño del conjunto de entrenamiento: 1168 filas
Tamaño del conjunto de prueba: 292 filas
```

1.10 10. Ingeniería de características

Para determinar qué variables pueden ser los mejores predictores para el precio de las casas, nos basamos en la matriz de correlación de las variables más relevantes con respecto a SalePrice



Según los resultados de la matriz de correlación, las variables que pueden ser mejores predictores para el precio de las casas son OverallQual que es la calidad general, GrLivArea que representa el área habitable, GarageCars que es la capacidad del garage de la casa, GarageArea que se refiere al tamaño del garaje, TotalBsmtSF que es el área total del sotano y el área del primer piso 1stFlrSF. Todas las variables anteriores tienen una correlación alta o mayor a 0.5 lo que indica que son las mejores o son las que más se relacionan con el precio de una casa.

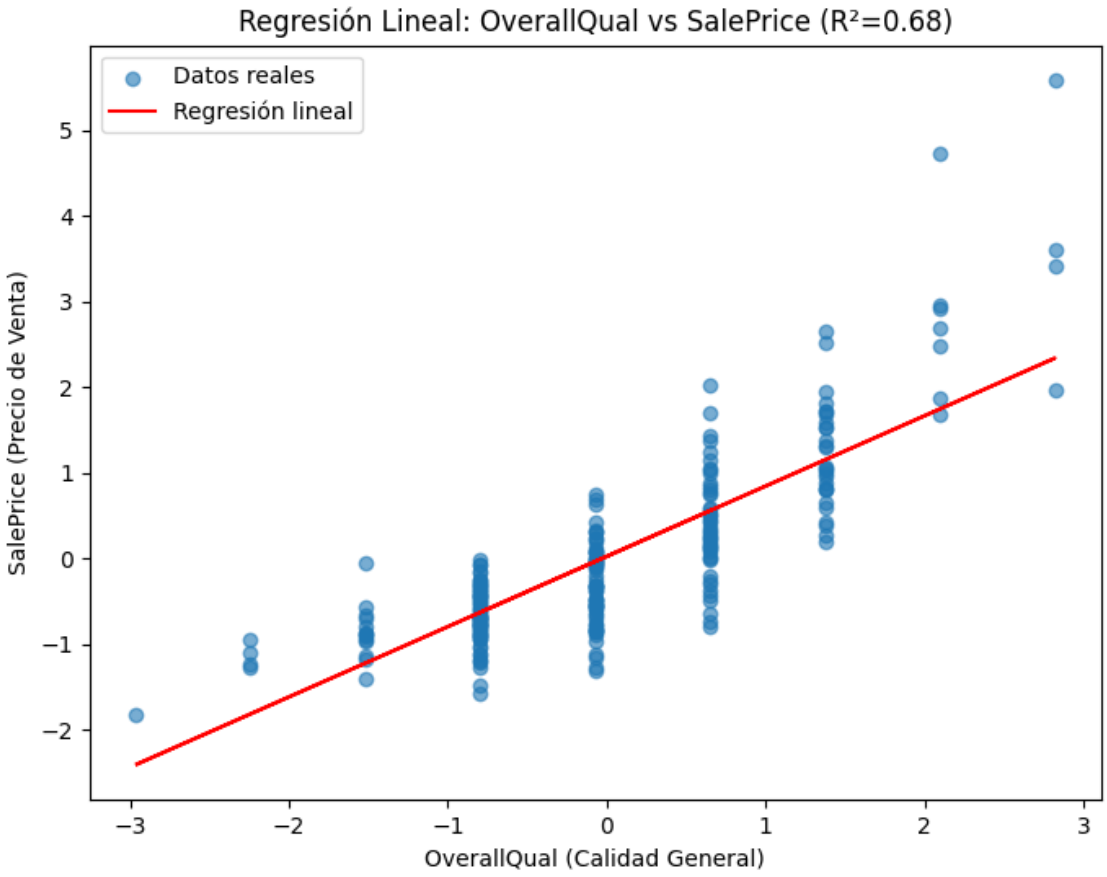
1.11 11. Modelo univariado de regresión lineal para predecir el precio de las casas

Para este modelo se seleccionó a la variable OverallQual que representa la calidad general de las casas

[29]:
```
# Cargar el dataset preprocesado
train_df = pd.read_csv("train_preprocessed.csv")

# Seleccionar la variable independiente (OverallQual) y la dependiente␣
↪(SalePrice)
X = train_df[["OverallQual"]]  # Variable predictora
y = train_df["SalePrice"]       # Variable objetivo

# Dividir los datos en entrenamiento (80%) y prueba (20%)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# EVALUANDO TRAIN Y TEST - Solo Selecciono una columna
ytrain = y_train.values.reshape(-1,1)
ytest = y_test.values.reshape(-1,1)
Xtrain = X_train['OverallQual'].values.reshape(-1,1)
xtest = X_test['OverallQual'].values.reshape(-1,1)

# Crear el modelo de regresión lineal
lm = LinearRegression()
lm.fit(Xtrain, ytrain)

# Hacer predicciones
y_pred = lm.predict(xtest)

# Calcular métricas del modelo
r2 = r2_score(ytest, y_pred)
mse = mean_squared_error(ytest, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R squared: {r2:.2f}")


# Crear gráfico de regresión
plt.figure(figsize=(8, 6))
plt.scatter(xtest, ytest, label="Datos reales", alpha=0.6)
plt.plot(xtest, y_pred, color="red", label="Regresión lineal")
plt.xlabel("OverallQual (Calidad General)")
plt.ylabel("SalePrice (Precio de Venta)")
plt.title(f"Regresión Lineal: OverallQual vs SalePrice (R²={r2:.2f})")
plt.legend()
plt.show()

# Resumen del modelo con statsmodels
X_train_sm = sm.add_constant(Xtrain)  # Agregar constante para el modelo
model_sm = sm.OLS(ytrain, X_train_sm).fit()
model_summary = model_sm.summary()

# Mostrar métricas y resumen del modelo
print(model_summary)
```

```
Mean Squared Error: 0.38
R squared: 0.68
```



OLS Regression Results

Dep. Variable:	y	R-squared:	0.632
Model:	OLS	Adj. R-squared:	0.631
Method:	Least Squares	F-statistic:	1598.
Date:	Sun, 02 Mar 2025	Prob (F-statistic):	2.45e-204
Time:	21:22:27	Log-Likelihood:	-881.31
No. Observations:	934	AIC:	1767.
Df Residuals:	932	BIC:	1776.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0220	0.020	1.079	0.281	-0.018	0.062
x1	0.8212	0.021	39.978	0.000	0.781	0.861

Omnibus:	445.668	Durbin-Watson:	1.976
Prob(Omnibus):	0.000	Jarque-Bera (JB):	4934.059
Skew:	1.893	Prob(JB):	0.00

```
Kurtosis:                       13.604   Cond. No.                         1.02
==============================================================================

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

Según la regresión lineal que se obtuvo, tenemos un coeficiente de determinación R cuadrado de 0.68 que indica que el 68% de la variabilidad en el precio de las casas se explica por OverallQual. Podemos observar que la regresión lineal es positiva por lo que los datos siguen esta tendencia, lo cual es lógico porque entre mejor sea la calidad de las casas mayor será el precio de estas. Sin embargo, se puede notar dispersión, lo que indica que otras variables también influyen en el precio de las casas, justo como se pudo observar en la matriz de correlación.

1.11.1 11.1 Residuos

[30]:
```
#Analizando residuales
residuos = ytest - y_pred
print("Los residuos son: ", len(residuos))

plt.plot(xtest, residuos, 'o', color='darkblue')
plt.axhline(0,color='blue')
plt.title("Gráfico de Residuales")
plt.xlabel("Variable independiente")
plt.ylabel("Residuales")
plt.show()

sns.histplot(residuos, kde=True)  # kde=True para incluir la curva de densidad
plt.xlabel("Residuales")
plt.title("Distribución de los Residuales")
plt.show()
```

```
Los residuos son:  234
```

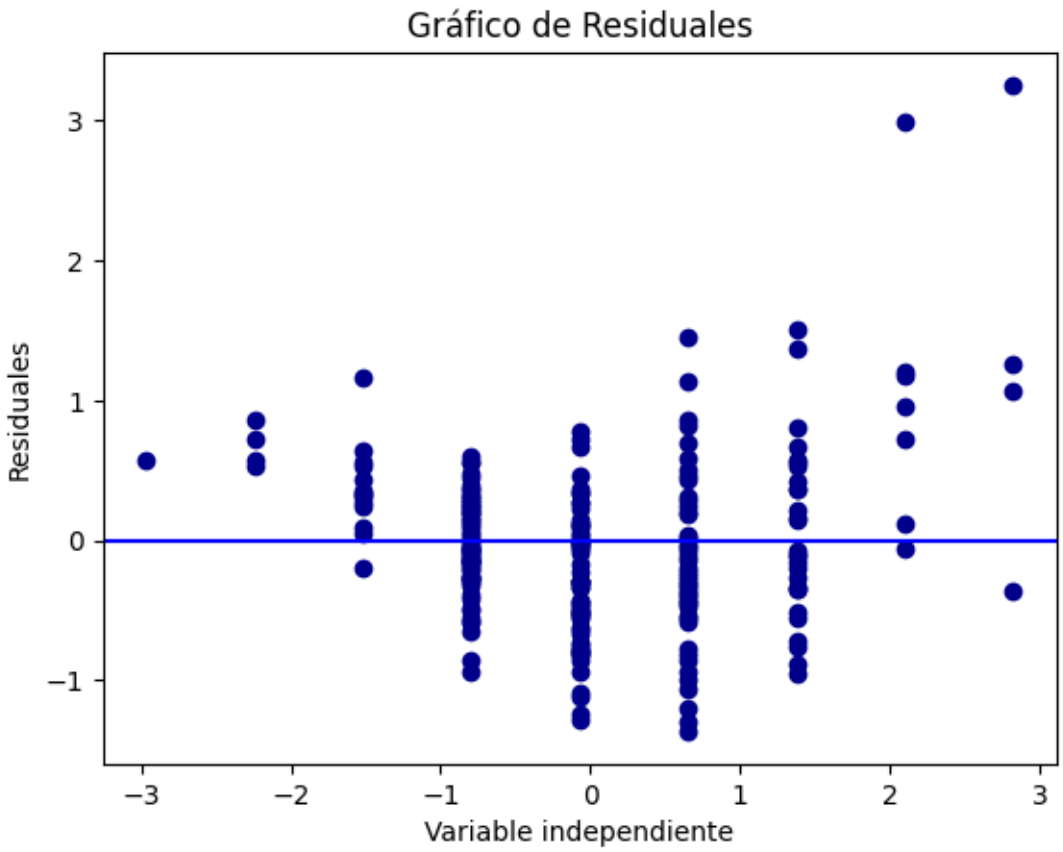
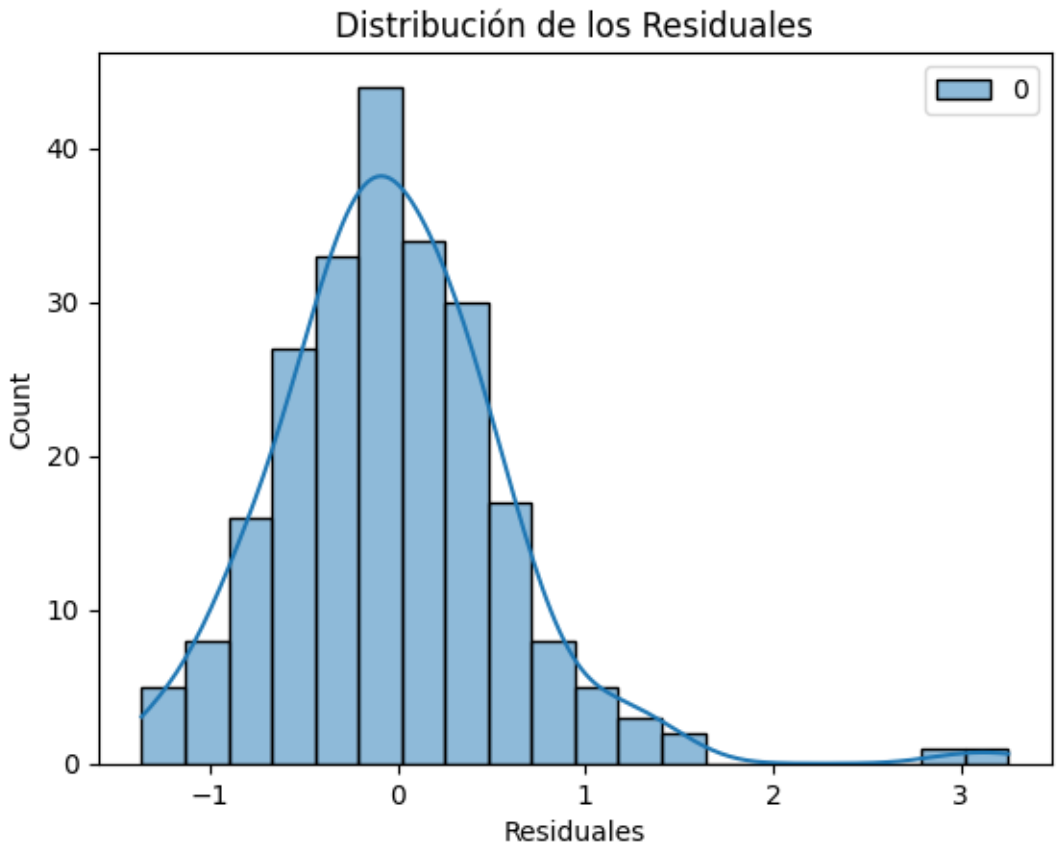


Gráfico de Residuales
Residuales
Variable independiente
3
2
1
0
−1
−3
−2
−1
0
1
2
3



Según los resultados de los residuos, su distribución parece tener una forma normal de campa simetrica, lo que indica que los errores están distribuidos de manera normal. la mayoría de los residuos se agrupan alrededor de cero, que quiere decir que el modelo no tiene un sesgo sistemático en las predicciones.

1.12 12. Modelo multivariado de regresión lineal para predecir el precio de las casas

Para este modelo utilizamos todas las variables numericas para predecir el precio de las casas

[31]:
```
# Seleccionar solo las variables numéricas
numerical_features = train_df.select_dtypes(include=['number']).columns
numerical_features = numerical_features.drop("SalePrice")  # Excluir la␣
 ↪variable objetivo
print("Variables utilizadas en el modelo:", numerical_features.tolist())


# Definir variables predictoras (X) y variable objetivo (y)
X = train_df[numerical_features]
y = train_df["SalePrice"]
```

```
# Dividir los datos en conjunto de entrenamiento (80%) y prueba (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Crear el modelo de regresión lineal
lm = LinearRegression()
lm.fit(X_train, y_train)

# Hacer predicciones en el conjunto de prueba
y_pred = lm.predict(X_test)

# Calcular métricas del modelo
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R squared: {r2:.2f}")

# Análisis del modelo con statsmodels
X_train_sm = sm.add_constant(X_train)  # Agregar constante para la regresión
model_sm = sm.OLS(y_train, X_train_sm).fit()
print(model_sm.summary())

# Gráfico de predicciones vs valores reales
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.6)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color="red",␣
 ↪linestyle="--")  # Línea de referencia
plt.xlabel("Valores Reales")
plt.ylabel("Predicciones")
plt.title(f"Predicciones vs Valores Reales (R²={r2:.2f})")
plt.show()
```

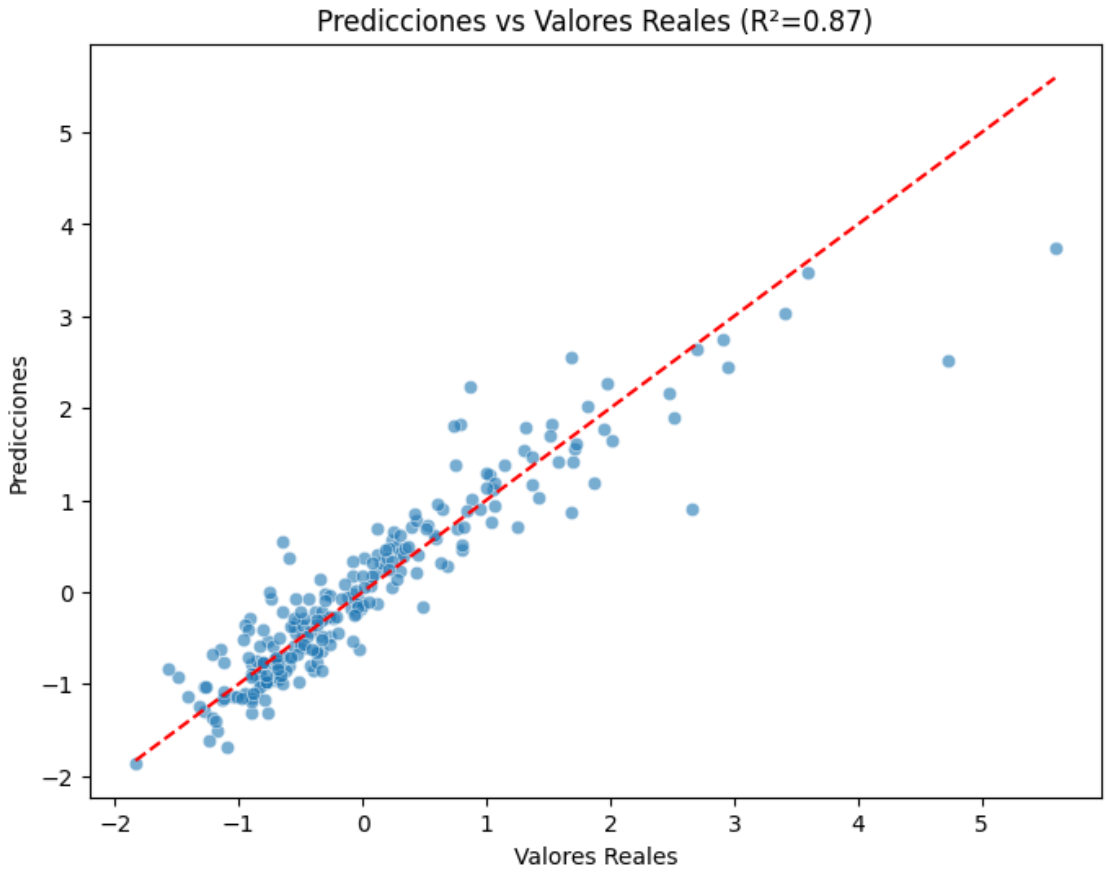
```
Variables utilizadas en el modelo: ['Id', 'MSSubClass', 'MSZoning',
'LotFrontage', 'LotArea', 'Street', 'LotShape', 'LandContour', 'Utilities',
'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
'BldgType', 'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt',
'YearRemodAdd', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd',
'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond',
'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2',
'BsmtUnfSF', 'TotalBsmtSF', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical',
'1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
'KitchenQual', 'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu',
'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea',
'GarageQual', 'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
```

```
'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold',
'YrSold', 'SaleType', 'SaleCondition']
Mean Squared Error: 0.16
R squared: 0.87
                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.911
Model:                            OLS   Adj. R-squared:                  0.904
Method:                 Least Squares   F-statistic:                     123.1
Date:                Sun, 02 Mar 2025   Prob (F-statistic):               0.00
Time:                        21:22:39   Log-Likelihood:                -215.59
No. Observations:                 934   AIC:                             577.2
Df Residuals:                     861   BIC:                             930.5
Df Model:                          72
Covariance Type:            nonrobust
==============================================================================
=
                    coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-
const            -0.0425      0.287     -0.148      0.882      -0.606
0.521
Id               -0.0051      0.011     -0.467      0.641      -0.026
0.016
MSSubClass       -0.0120      0.025     -0.488      0.626      -0.060
0.036
MSZoning         -0.0231      0.021     -1.118      0.264      -0.064
0.017
LotFrontage       0.0356      0.015      2.314      0.021       0.005
0.066
LotArea           0.0662      0.012      5.358      0.000       0.042
0.090
Street            0.3963      0.174      2.283      0.023       0.056
0.737
LotShape         -0.0086      0.008     -1.036      0.300      -0.025
0.008
LandContour      -0.0047      0.017     -0.274      0.784      -0.038
0.029
Utilities     -4.172e-16   2.01e-16     -2.071      0.039   -8.13e-16
-2.17e-17
LotConfig        -0.0085      0.007     -1.220      0.223      -0.022
0.005
LandSlope        -0.0506      0.046     -1.104      0.270      -0.141
0.039
Neighborhood      0.0007      0.002      0.358      0.721      -0.003
0.005
Condition1       -0.0107      0.013     -0.812      0.417      -0.037
```

```
0.015
Condition2       -0.0103      0.037     -0.282      0.778      -0.082
0.062
BldgType         -0.0334      0.019     -1.781      0.075      -0.070
0.003
HouseStyle       -0.0016      0.008     -0.199      0.842      -0.018
0.015
OverallQual       0.1354      0.021      6.590      0.000       0.095
0.176
OverallCond       0.0540      0.014      3.772      0.000       0.026
0.082
YearBuilt         0.1034      0.030      3.428      0.001       0.044
0.163
YearRemodAdd      0.0323      0.017      1.849      0.065      -0.002
0.067
RoofStyle         0.0139      0.015      0.947      0.344      -0.015
0.043
RoofMatl          0.0173      0.019      0.891      0.373      -0.021
0.056
Exterior1st      -0.0169      0.006     -2.608      0.009      -0.030
-0.004
Exterior2nd       0.0104      0.006      1.806      0.071      -0.001
0.022
MasVnrArea        0.0440      0.013      3.402      0.001       0.019
0.069
ExterQual        -0.1923      0.025     -7.830      0.000      -0.240
-0.144
ExterCond         0.0101      0.016      0.642      0.521      -0.021
0.041
Foundation       -0.0027      0.020     -0.133      0.894      -0.042
0.037
BsmtQual         -0.0973      0.018     -5.450      0.000      -0.132
-0.062
BsmtCond          0.0333      0.019      1.791      0.074      -0.003
0.070
BsmtExposure     -0.0254      0.011     -2.287      0.022      -0.047
-0.004
BsmtFinType1      0.0205      0.008      2.535      0.011       0.005
0.036
BsmtFinSF1        0.1433      0.014     10.108      0.000       0.115
0.171
BsmtFinType2      0.0129      0.017      0.772      0.440      -0.020
0.046
BsmtFinSF2        0.0253      0.016      1.620      0.106      -0.005
0.056
BsmtUnfSF        -0.0306      0.011     -2.728      0.006      -0.053
-0.009
TotalBsmtSF       0.1274      0.016      7.756      0.000       0.095
```

```
0.160
Heating          0.0012      0.035       0.034      0.973      -0.067
0.070
HeatingQC       -0.0093      0.008      -1.192      0.234      -0.025
0.006
CentralAir       0.0386      0.057       0.680      0.496      -0.073
0.150
Electrical      -0.0144      0.012      -1.205      0.228      -0.038
0.009
1stFlrSF         0.1421      0.019       7.336      0.000       0.104
0.180
2ndFlrSF         0.1748      0.016      11.068      0.000       0.144
0.206
LowQualFinSF     0.0029      0.012       0.243      0.808      -0.021
0.026
GrLivArea        0.2500      0.017      14.640      0.000       0.216
0.283
BsmtFullBath    -0.0112      0.017      -0.656      0.512      -0.045
0.022
BsmtHalfBath     0.0029      0.012       0.247      0.805      -0.020
0.026
FullBath        -0.0271      0.019      -1.403      0.161      -0.065
0.011
HalfBath        -0.0222      0.016      -1.369      0.171      -0.054
0.010
BedroomAbvGr    -0.0833      0.017      -4.776      0.000      -0.118
-0.049
KitchenAbvGr    -0.0386      0.014      -2.842      0.005      -0.065
-0.012
KitchenQual     -0.0813      0.018      -4.480      0.000      -0.117
-0.046
TotRmsAbvGrd     0.0355      0.024       1.494      0.136      -0.011
0.082
Functional       0.0674      0.012       5.471      0.000       0.043
0.092
Fireplaces       0.0297      0.015       2.040      0.042       0.001
0.058
FireplaceQu     -0.0375      0.014      -2.722      0.007      -0.065
-0.010
GarageType       0.0136      0.008       1.717      0.086      -0.002
0.029
GarageYrBlt      0.0164      0.020       0.807      0.420      -0.024
0.056
GarageFinish     0.0271      0.018       1.464      0.143      -0.009
0.063
GarageCars       0.0348      0.027       1.302      0.193      -0.018
0.087
GarageArea       0.0117      0.026       0.445      0.657      -0.040
```

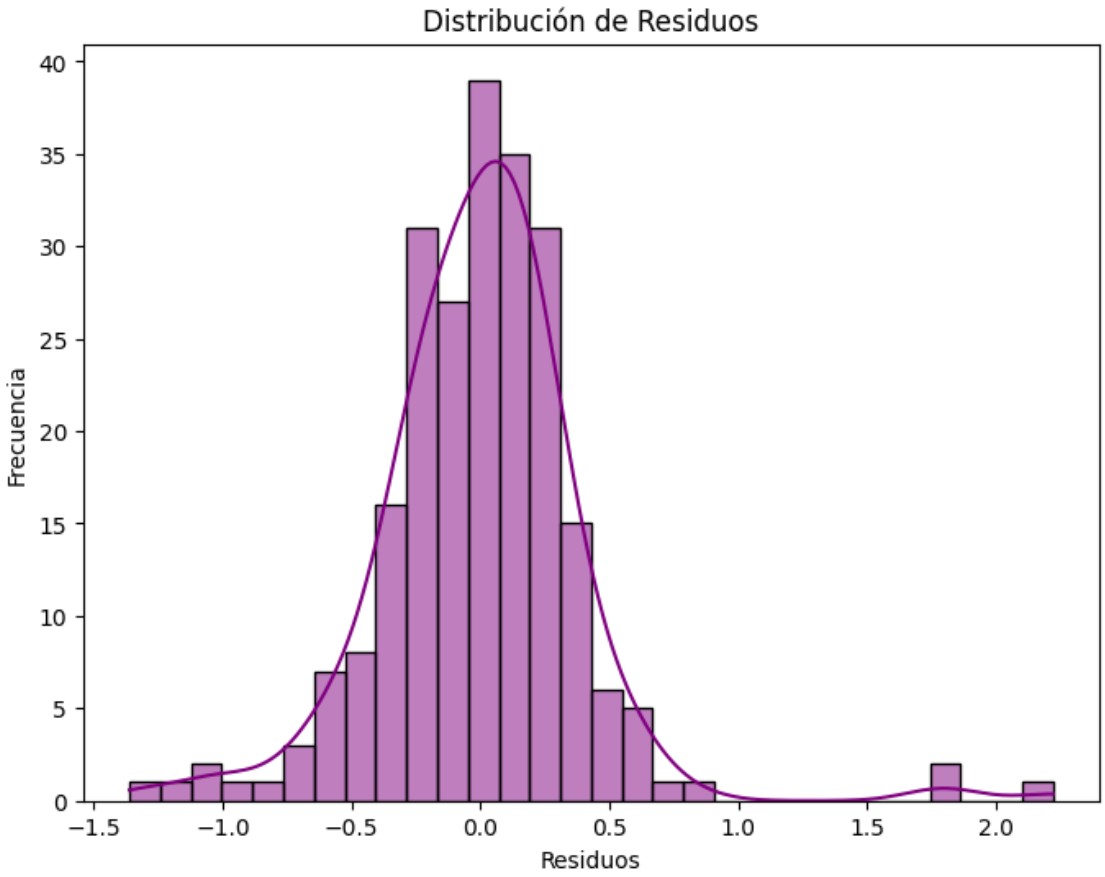
```
0.064
GarageQual       -0.0045      0.023     -0.194      0.846      -0.050
0.041
GarageCond        0.0496      0.027      1.864      0.063      -0.003
0.102
PavedDrive       -0.0162      0.026     -0.617      0.538      -0.068
0.035
WoodDeckSF        0.0066      0.012      0.555      0.579      -0.017
0.030
OpenPorchSF       0.0071      0.012      0.592      0.554      -0.016
0.031
EnclosedPorch     0.0097      0.012      0.800      0.424      -0.014
0.033
3SsnPorch        -0.0014      0.011     -0.131      0.896      -0.023
0.020
ScreenPorch       0.0323      0.011      3.041      0.002       0.011
0.053
PoolArea          0.0559      0.011      5.267      0.000       0.035
0.077
MiscVal           0.0027      0.010      0.282      0.778      -0.016
0.022
MoSold           -0.0164      0.011     -1.483      0.138      -0.038
0.005
YrSold           -0.0109      0.011     -0.992      0.322      -0.032
0.011
SaleType         -0.0025      0.007     -0.348      0.728      -0.017
0.012
SaleCondition     0.0322      0.011      2.997      0.003       0.011
0.053
==============================================================================
Omnibus:                      273.746   Durbin-Watson:                   2.056
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2450.498
Skew:                           1.071   Prob(JB):                         0.00
Kurtosis:                      10.640   Cond. No.                     1.01e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 5.97e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```



En la regresión lineal que se obtuvo con todas las variables numéricas, presenta un coeficiente de determinación R cuadrado de 0.87 que indica que el 87% de la variabilidad en el precio de las casas se explica por las variables numéricas. Se puede observar que los puntos están cerca de la línea roja, lo que indica que el modelo está funcionando bien.

1.13 12.1 Residuos

[32]:
```
# Analizando los residuos
residuos = y_test - y_pred

# Gráfico de distribución de residuos
plt.figure(figsize=(8, 6))
sns.histplot(residuos, bins=30, kde=True, color="purple")
plt.xlabel("Residuos")
plt.ylabel("Frecuencia")
plt.title("Distribución de Residuos")
plt.show()
```



1.13.1 13. Análisis de multicolinealidad, correlación y detección de sobreajuste

En este paso, se analizan las posibles correlaciones entre las variables predictoras y se determina si existe multicolinealidad. También se revisa el ajuste del modelo para identificar si está ocurriendo un sobreajuste (overfitting).

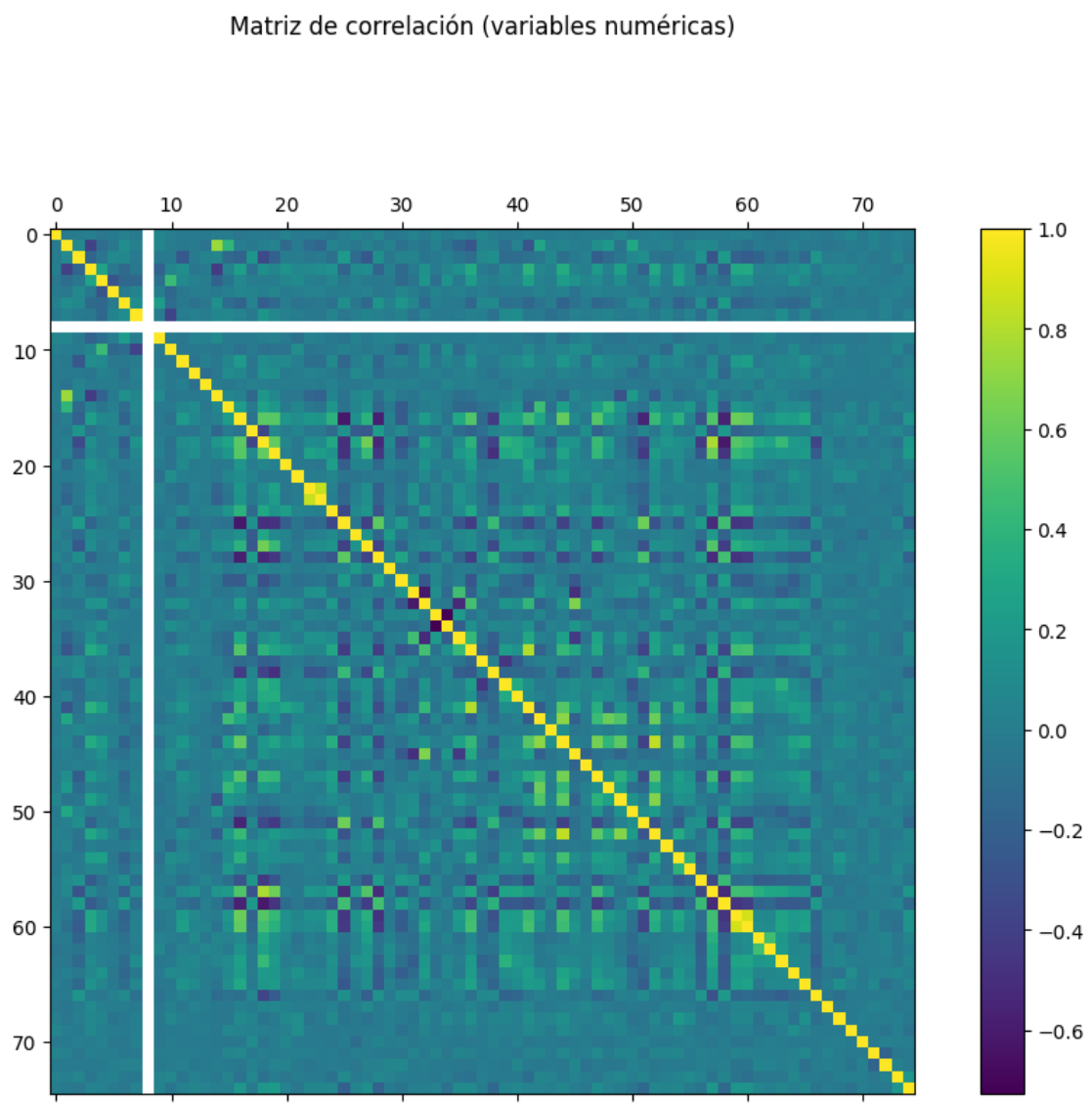
Pasos principales: 1. Calcular la matriz de correlación para las variables numéricas y observar las variables más correlacionadas entre sí y con la variable respuesta (SalePrice). 2. Calcular el Factor de Inflación de Varianza (VIF) para detectar multicolinealidad. 3. Entrenar un modelo de Regresión Lineal Múltiple con todas las variables numéricas seleccionadas (en pasos previos). 4. Analizar la puntuación R^2 en entrenamiento y en validación (o prueba) para evidenciar posible sobreajuste.

Según los resultados de los residuos, su distribución parece tener una forma normal de campa simétrica, lo que indica que los errores están distribuidos de manera normal. la mayoría de los residuos se agrupan alrededor de cero, que quiere decir que el modelo no tiene un sesgo sistemático en las predicciones.

[35]:
```
# 1. Matriz de correlación (variables numéricas)
# Asegurar de seleccionar solo columnas numéricas
```

```
numeric_features = X_train.select_dtypes(include=[np.number])
corr_matrix = numeric_features.corr()

plt.figure(figsize=(12, 8))
plt.matshow(corr_matrix, fignum=0)
plt.title("Matriz de correlación (variables numéricas)", pad=100)
plt.colorbar()
plt.show()

# 2. Función para calcular el VIF de cada variable
def calcular_vif(df):
    """
    Calcula el VIF de cada columna en un DataFrame.
    VIF = 1 / (1 - R^2)
    """
    vif_data = []
    # Se añade la constante para usar en el modelo
    df_const = sm.add_constant(df)

    for i, col in enumerate(df.columns):
        # Se elimina la columna 'col' para ver su VIF
        X_temp = df_const.drop(columns=[col], errors='ignore')
        y_temp = df_const[col]

        model = sm.OLS(y_temp, X_temp)
        results = model.fit()

        r2 = results.rsquared
        vif = np.inf if r2 == 1 else 1/(1 - r2)

        vif_data.append((col, vif))

    # Ordenamos de mayor a menor VIF
    vif_data.sort(key=lambda x: x[1], reverse=True)
    return vif_data

# Calculamos el VIF ignorando la variable 'SalePrice' si estuviera en␣
 ↪numeric_features
vif_resultados = calcular_vif(numeric_features.drop(columns=['SalePrice'],␣
 ↪errors='ignore'))
print("VIF de las variables (orden descendente):")
for var, val in vif_resultados:
    print(f"{var}: {val:.2f}")

# 3. Entrenar el modelo de Regresión Lineal con TODAS las variables numéricas
linreg_all = LinearRegression()
linreg_all.fit(X_train, y_train)
```

```python
# 4. Métricas en el conjunto de entrenamiento para ver si hay sobreajuste
y_pred_train = linreg_all.predict(X_train)
r2_train = r2_score(y_train, y_pred_train)

#  mean_squared_error sin el parámetro 'squared'
mse_train = mean_squared_error(y_train, y_pred_train)
rmse_train = np.sqrt(mse_train)

print("R2 (Train) con todas las variables:", r2_train)
print("RMSE (Train) con todas las variables:", rmse_train)

# Métricas en el conjunto de prueba
y_pred_test = linreg_all.predict(X_test)
r2_test = r2_score(y_test, y_pred_test)

mse_test = mean_squared_error(y_test, y_pred_test)
rmse_test = np.sqrt(mse_test)

print("R2 (Test) con todas las variables:", r2_test)
print("RMSE (Test) con todas las variables:", rmse_test)
```




```
/Users/hansellopez/Documents/Projects/DataMining/Modelos-Regresion-Lineal-Data-
Mining/.venv/lib/python3.12/site-
packages/statsmodels/regression/linear_model.py:1782: RuntimeWarning: invalid
value encountered in scalar divide
  return 1 - self.ssr/self.centered_tss

VIF de las variables (orden descendente):
Utilities: nan
BsmtFinSF1: inf
BsmtFinSF2: inf
BsmtUnfSF: inf
TotalBsmtSF: inf
1stFlrSF: inf
```

```
2ndFlrSF: inf
LowQualFinSF: inf
GrLivArea: inf
YearBuilt: 8.22
GarageCars: 6.77
GarageArea: 6.53
MSSubClass: 5.37
TotRmsAbvGrd: 5.17
BldgType: 4.40
Exterior1st: 4.07
Exterior2nd: 4.00
GarageYrBlt: 3.91
OverallQual: 3.84
FullBath: 3.48
YearRemodAdd: 2.81
BedroomAbvGr: 2.77
BsmtFullBath: 2.67
ExterQual: 2.62
HalfBath: 2.47
BsmtFinType2: 2.38
HouseStyle: 2.34
BsmtQual: 2.24
GarageFinish: 2.16
Foundation: 2.09
KitchenQual: 2.08
BsmtFinType1: 2.05
Fireplaces: 1.94
OverallCond: 1.93
KitchenAbvGr: 1.83
LotFrontage: 1.82
CentralAir: 1.82
GarageType: 1.79
LotArea: 1.72
LandSlope: 1.70
GarageQual: 1.69
HeatingQC: 1.67
BsmtExposure: 1.53
PavedDrive: 1.51
GarageCond: 1.50
FireplaceQu: 1.47
MasVnrArea: 1.44
Heating: 1.40
MSZoning: 1.38
Functional: 1.37
Electrical: 1.35
LandContour: 1.34
EnclosedPorch: 1.34
OpenPorchSF: 1.32
```

```
BsmtHalfBath: 1.32
WoodDeckSF: 1.32
ExterCond: 1.31
RoofStyle: 1.31
Neighborhood: 1.27
RoofMatl: 1.26
LotShape: 1.25
ScreenPorch: 1.22
Street: 1.19
LotConfig: 1.18
SaleCondition: 1.18
SaleType: 1.17
BsmtCond: 1.16
Condition1: 1.14
YrSold: 1.13
MoSold: 1.11
PoolArea: 1.11
Condition2: 1.10
Id: 1.08
3SsnPorch: 1.07
MiscVal: 1.06
R2 (Train) con todas las variables: 0.9114577089724687
RMSE (Train) con todas las variables: 0.3047950156392137
R2 (Test) con todas las variables: 0.8656974161878999
RMSE (Test) con todas las variables: 0.3959902743662094
```

Lo anterior se basa en que:

- **TotalBsmtSF** ya resume la información de `BsmtFinSF1`, `BsmtFinSF2` y `BsmtUnfSF`.
- **GrLivArea** agrupa el área habitable de 1er y 2do piso, además de `LowQualFinSF`.
- **GarageCars** y **GarageArea** miden casi lo mismo; conviene elegir la que mejor se correlacione con **SalePrice**.
- **Utilities** suele no variar o aportar mucho, por lo que suele eliminarse.
- **YearBuilt** puede resultar prescindible si **YearRemodAdd** está muy correlacionado y refleja mejor la actualización de la casa.

1.13.2 14. Nuevo modelo (reducción de variables o regularización) y análisis de residuos

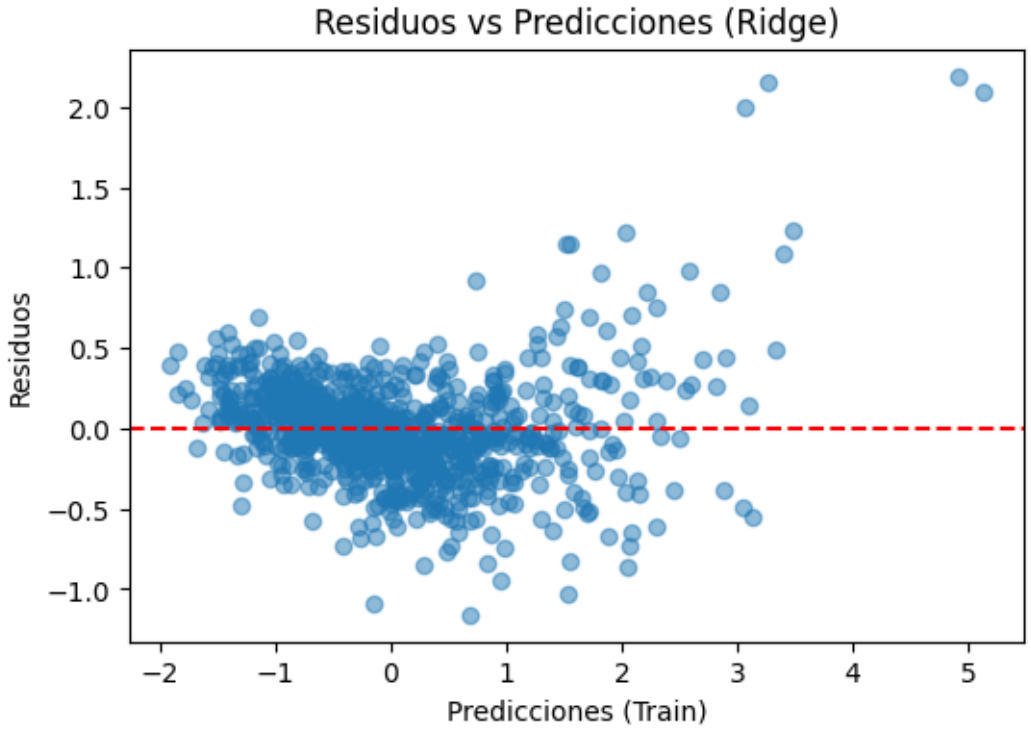
Si se detecta multicolinealidad (VIF alto en algunas variables) o signos de sobreajuste (alta diferencia de desempeño entre entrenamiento y prueba), podemos: - Eliminar variables con VIF muy alto para reducir multicolinealidad. - Emplear modelos de regularización (por ejemplo, Lasso o Ridge). - Seleccionar solo las variables más predictivas según análisis de correlación o importancia del modelo.

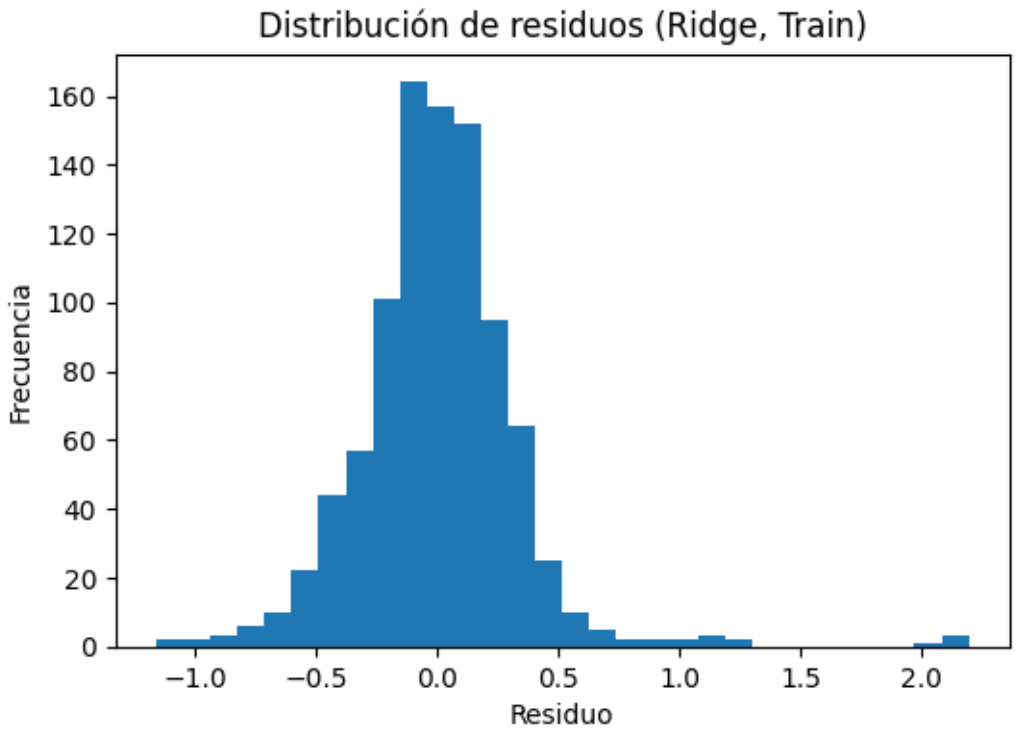
Luego de ajustar el nuevo modelo, se analizan los residuos para verificar la calidad del ajuste (normalidad de residuos, homocedasticidad, etc.). Se suelen graficar: - Gráfico de valores predichos vs. residuos. - Histograma o Q-Q plot de residuos para ver su distribución.

[39]:
```
# ================================
# Regularización con Ridge
# ================================
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score

# Variables a eliminar (según análisis de VIF y correlaciones)
vars_a_eliminar = [
    "BsmtFinSF1", "BsmtFinSF2", "BsmtUnfSF",  # ya se resume en TotalBsmtSF
    "1stFlrSF", "2ndFlrSF", "LowQualFinSF",   # ya se incluye en GrLivArea
    "GarageArea",                             # muy correlacionada con␣
 ↪GarageCars
    "Utilities",                              # poca variación / utilidad
    "YearBuilt"                               # fuertemente correlacionada con␣
 ↪YearRemodAdd
]

# Reducción de columnas en train y test
X_train_reduced = X_train.drop(columns=vars_a_eliminar, errors='ignore')
X_test_reduced = X_test.drop(columns=vars_a_eliminar, errors='ignore')

# Entrenar el modelo con Ridge (penalización alpha=10.0)
ridge_model = Ridge(alpha=10.0)
ridge_model.fit(X_train_reduced, y_train)

# Predicciones en el set de entrenamiento
y_train_pred_ridge = ridge_model.predict(X_train_reduced)
residuos_train = y_train - y_train_pred_ridge

# Cálculo de métricas en entrenamiento
r2_train_ridge = r2_score(y_train, y_train_pred_ridge)
mse_train_ridge = mean_squared_error(y_train, y_train_pred_ridge)
rmse_train_ridge = np.sqrt(mse_train_ridge)

print("R2 (Train) [Ridge]:", r2_train_ridge)
print("MSE (Train) [Ridge]:", mse_train_ridge)
print("RMSE (Train) [Ridge]:", rmse_train_ridge)

# ==============
# Análisis de residuos
# ==============
# Gráfica de residuos vs. predicciones
plt.figure(figsize=(6,4))
plt.scatter(y_train_pred_ridge, residuos_train, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
```

```
plt.title("Residuos vs Predicciones (Ridge)")
plt.xlabel("Predicciones (Train)")
plt.ylabel("Residuos")
plt.show()

# Histograma de residuos para ver su distribución
plt.figure(figsize=(6,4))
plt.hist(residuos_train, bins=30)
plt.title("Distribución de residuos (Ridge, Train)")
plt.xlabel("Residuo")
plt.ylabel("Frecuencia")
plt.show()
```

```
R2 (Train) [Ridge]: 0.9018920049655823
MSE (Train) [Ridge]: 0.1029364926729259
RMSE (Train) [Ridge]: 0.32083717470537276
```





1.13.3 15. Uso de cada modelo con el conjunto de prueba y eficiencia en la predicción

En este punto, utilizamos los modelos entrenados (por ejemplo, el modelo de Regresión Lineal con todas las variables y el modelo de Ridge regularizado con variables reducidas) para predecir en el conjunto de prueba. Luego comparamos las métricas (R^2, RMSE, etc.) para determinar la calidad de las predicciones de cada modelo. Haciendo uso de métricas como: - R^2 (Coeficiente de Determinación) - RMSE (Raíz del Error Cuadrático Medio) - MAE (Error Absoluto Medio), etc.

Se comparan los resultados para determinar cuál modelo realiza mejores predicciones.

[40]:
```
# Evaluación del modelo original (linreg_all) en Test
y_pred_test_all = linreg_all.predict(X_test)
r2_test_all = r2_score(y_test, y_pred_test_all)
mse_test_all = mean_squared_error(y_test, y_pred_test_all)
rmse_test_all = np.sqrt(mse_test_all)

print("=== Modelo Original (todas las variables) ===")
print("R2 (Test):", r2_test_all)
print("MSE (Test):", mse_test_all)
print("RMSE (Test):", rmse_test_all)
print()

# Evaluación del modelo Ridge (vars reducidas) en Test
y_pred_test_ridge = ridge_model.predict(X_test_reduced)
```

```
r2_test_ridge = r2_score(y_test, y_pred_test_ridge)
mse_test_ridge = mean_squared_error(y_test, y_pred_test_ridge)
rmse_test_ridge = np.sqrt(mse_test_ridge)

print("=== Modelo Ridge (variables reducidas) ===")
print("R2 (Test):", r2_test_ridge)
print("MSE (Test):", mse_test_ridge)
print("RMSE (Test):", rmse_test_ridge)
```

```
=== Modelo Original (todas las variables) ===
R2 (Test): 0.8656974161878999
MSE (Test): 0.15680829739262583
RMSE (Test): 0.3959902743662094

=== Modelo Ridge (variables reducidas) ===
R2 (Test): 0.8667792695374293
MSE (Test): 0.15554515280558212
RMSE (Test): 0.3943921307602145
```

2 Proyecto 2 "House Prices: Advanced Regression Techniques" - Parte 2 Arbole de decisión

2.1 1. Uso de los mismos conjuntos de entrenamiento y prueb

[ ]:
```
# ========================================
# 1. Carga de datos y separación en train/test
# ========================================
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score

# Cargamos los datos
train_df = pd.read_csv("train_preprocessed.csv")
# test_df = pd.read_csv("test_preprocessed.csv")

# Con variable objetivo es 'SalePrice'
X = train_df.drop(["SalePrice"], axis=1)
y = train_df["SalePrice"]

# mantener la misma proporción y la misma semilla (random_state) que en la␣
 ↪entrega anterior
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
```

```
                                                    random_state=42)

print("Tamaño de X_train:", X_train.shape)
print("Tamaño de X_test: ", X_test.shape)
```

```
Tamaño de X_train: (934, 75)
Tamaño de X_test:  (234, 75)
```

3 2. Elaborar un árbol de regresión con todas las variables

Entrenamos un modelo de árbol de decisión para **regresión** utilizando todas las variables predictoras. Luego veremos qué tan bien se ajusta a los datos.

[2]:
```
# ==========================================
# 2. Árbol de Regresión (modelo base)
# ==========================================
from sklearn.tree import DecisionTreeRegressor

# Creamos el modelo con una configuración básica (sin limitar la profundidad)
reg_tree = DecisionTreeRegressor(random_state=42)
reg_tree.fit(X_train, y_train)

print("Árbol de regresión entrenado con éxito.")
```

```
Árbol de regresión entrenado con éxito.
```

4 3. Usar el modelo para predecir y analizar el resultado

Se evalúa el modelo realizando predicciones sobre el conjunto de prueba y calculando métricas de error como MSE y RMSE, así como el R^2 (coeficiente de determinación).

[3]:
```
# ==========================================
# 3. Predicción y evaluación del árbol de regresión
# ==========================================
from sklearn.metrics import mean_squared_error, r2_score

# Realizamos la predicción sobre el set de prueba
y_pred = reg_tree.predict(X_test)

# Cálculo de métricas
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred)

print("MSE:", mse)
print("RMSE:", rmse)
print("R2:", r2)
```

```
MSE: 0.269173752305001
RMSE: 0.5188195758691079
R2: 0.7694590718089135
```

5 4. Crear al menos 3 modelos más variando la profundidad del árbol

Probaremos diferentes valores para el parámetro de profundidad (`max_depth`) y compararemos los resultados en términos de MSE, RMSE y R^2.

[4]:
```python
# ========================================
# 4. Probando diferentes profundidades del árbol
# ========================================
import numpy as np

depths = [2, 5, 10]  # Diferentes profundidades de ejemplo

for d in depths:
    reg_tree_temp = DecisionTreeRegressor(max_depth=d, random_state=42)
    reg_tree_temp.fit(X_train, y_train)
    y_pred_temp = reg_tree_temp.predict(X_test)

    mse_temp = mean_squared_error(y_test, y_pred_temp)
    rmse_temp = mse_temp ** 0.5
    r2_temp = r2_score(y_test, y_pred_temp)

    print(f"Profundidad: {d}")
    print(f" -> MSE: {mse_temp:.2f}")
    print(f" -> RMSE: {rmse_temp:.2f}")
    print(f" -> R2: {r2_temp:.4f}")
    print("-"*30)
```

```
Profundidad: 2
 -> MSE: 0.42
 -> RMSE: 0.65
 -> R2: 0.6387
------------------------------
Profundidad: 5
 -> MSE: 0.22
 -> RMSE: 0.47
 -> R2: 0.8090
------------------------------
Profundidad: 10
 -> MSE: 0.29
 -> RMSE: 0.54
 -> R2: 0.7494
------------------------------
```

6 5. Comparación con el modelo de regresión lineal de la entrega anterior

Error cuadrático medio (MSE):

- Árbol de regresión: 0.2692
- Regresión lineal (original): 0.1568
- Regresión con Ridge: 0.1555

Raíz del error cuadrático medio (RMSE):

- Árbol de regresión: 0.5188
- Regresión lineal (original): 0.3960
- Regresión con Ridge: 0.3944

Coeficiente de determinación (R^2):

- Árbol de regresión: 0.7695
- Regresión lineal (original): 0.8657
- Regresión con Ridge: 0.8668

Según los resultados, se puede observar que en ambas versiones de la regresión lineal existe un MSE menor, lo que indica que sus predicciones son más precisas en promedio. Asimismo, el RMSE también es menor en la regresión lineal lo que confirma que los errores individuales tienden a ser menores en comparación con el árbol de regresión. También la regresión lineal explica mejor la variabilidad de los datos ya que vemos un coeficiente de determinación más cercano a uno en la regresión lineal. Por lo tanto, se puede concluir que, para este caso, la regresión lineal es un mejor modelo.

7 6. Creación de la variable de clasificación (Económicas, Intermedias, Caras)

Definimos una nueva variable categórica (por ejemplo, usando cuantiles) a partir del precio de la vivienda (SalePrice). Luego, esa variable será la nueva y para un árbol de clasificación.

[5]:
```
# ========================================
# 6. Creando variable categórica de 3 clases
# ========================================
import numpy as np

# Calculamos dos cuantiles para dividir en 3 grupos (Económicas, Intermedias,␣
 ↪Caras)
q1 = train_df["SalePrice"].quantile(0.33)
q2 = train_df["SalePrice"].quantile(0.66)

def categorizar_precio(precio):
    if precio <= q1:
```

```
        return "Económica"
    elif precio <= q2:
        return "Intermedia"
    else:
        return "Cara"

# Creamos la nueva columna en el DataFrame
train_df["PrecioCat"] = train_df["SalePrice"].apply(categorizar_precio)

# Observamos la distribución de las categorías
print(train_df["PrecioCat"].value_counts())
```

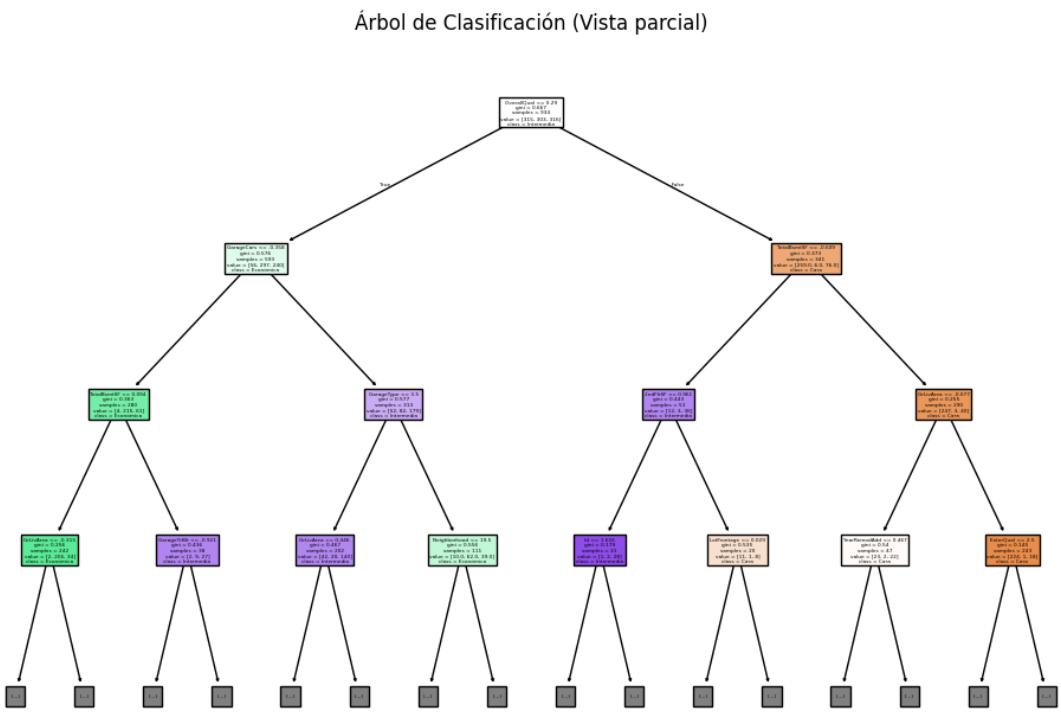
```
PrecioCat
Cara          397
Económica     391
Intermedia    380
Name: count, dtype: int64
```

8 7. Árbol de clasificación utilizando la nueva variable de 3 clases

Ahora creamos un árbol de decisión para clasificación, usando `PrecioCat` como variable objetivo y excluyendo `SalePrice` del conjunto de predictores. Mostramos gráficamente el árbol y su profundidad.

[10]:
```
# ========================================
# 7. Entrenamiento del árbol de clasificación
# ========================================
import os
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split

# Verificar si la carpeta 'img' existe; si no, crearla
if not os.path.exists("img"):
    os.makedirs("img")

# X_clf: todas las columnas excepto SalePrice y la nueva variable PrecioCat
X_clf = train_df.drop(["SalePrice", "PrecioCat"], axis=1)

# y_clf: la variable categórica
y_clf = train_df["PrecioCat"]

# Dividimos en train y test
X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(
    X_clf, y_clf, test_size=0.2, random_state=42
)
```

```python
# Creamos el árbol de clasificación
clf_tree = DecisionTreeClassifier(random_state=42)
clf_tree.fit(X_train_clf, y_train_clf)

# Visualización del árbol (versión resumida, máximo 3 niveles)
plt.figure(figsize=(12, 8))
plot_tree(
    clf_tree,
    feature_names=X_clf.columns,
    class_names=clf_tree.classes_,
    filled=True,
    max_depth=3
)
plt.title("Árbol de Clasificación (Vista parcial)")

# Guardar la figura como PNG en la carpeta 'img'
plt.savefig("img/arbol_clasificacion.png", dpi=300, bbox_inches='tight')
plt.show()

# Información sobre la complejidad del árbol
print("Profundidad total del árbol de clasificación:", clf_tree.get_depth())
print("Nodos (hojas) en el árbol de clasificación:", clf_tree.get_n_leaves())
print("¡La imagen se ha guardado en 'img/arbol_clasificacion.png'!")
```



```
Profundidad total del árbol de clasificación: 14
Nodos (hojas) en el árbol de clasificación: 120
¡La imagen se ha guardado en 'img/arbol_clasificacion.png'!
```

9 8. Eficiencia del Árbol de clasificación

Utilizamos el modelo con el conjunto de prueba y determinamos la eficiencia del algoritmo para clasificar.

[ ]:
```
# ================================
# Evaluación del modelo
# ================================

# Predicciones en el conjunto de prueba
y_pred_clf = clf_tree.predict(X_test_clf)

# Calcular precisión
accuracy = accuracy_score(y_test_clf, y_pred_clf)
print(f"Precisión del modelo: {accuracy:.4f}")
```

Precisión del modelo: 0.7265

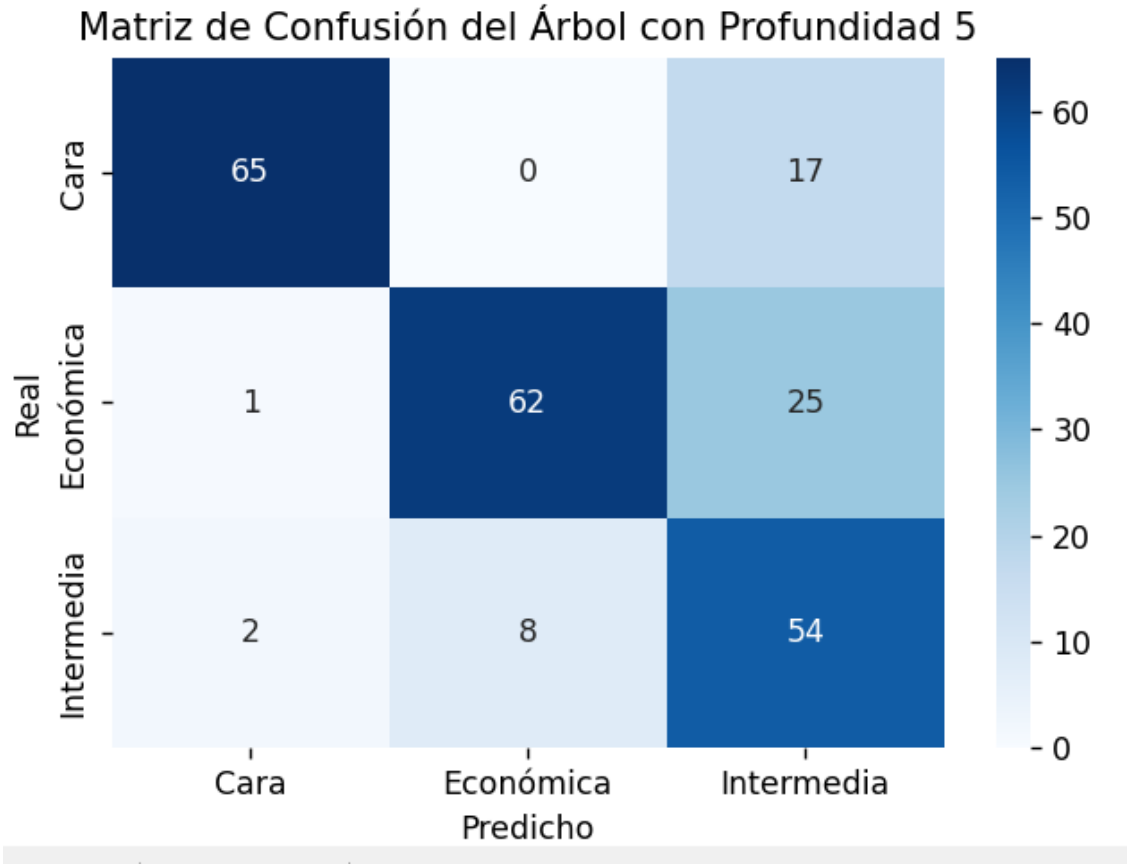
Los resultados indican que el modelo tiene una precisión de 0.7265, lo que significa que el modelo clasifica correctamente el 72.65% de las instancias en el conjunto de prueba. Tiene un 72.65% de precisión.

10 9. Matriz de confusión

Analizamos la eficiencia del algoritmo usando una matriz de confusión para el árbol de clasificación.

[ ]:
```
# Mostrar matriz de confusión
conf_matrix = confusion_matrix(y_test_clf, y_pred_clf)
print("Matriz de confusión:")
print(conf_matrix)

# Mostrar reporte de clasificación
class_report = classification_report(y_test_clf, y_pred_clf)
print("Reporte de clasificación:")
print(class_report)

# Visualización de la matriz de confusión
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=clf_tree.classes_, yticklabels=clf_tree.classes_)
plt.xlabel("Predicho")
plt.ylabel("Real")
```

```
plt.title("Matriz de Confusión")
plt.show()
```




```
Modelo con profundidad 3:
  - Precisión promedio de validación cruzada: 0.7414
  - Desviación estándar: 0.0143
-------------------------------------------------
Modelo con profundidad 5:
  - Precisión promedio de validación cruzada: 0.7560
  - Desviación estándar: 0.0241
-------------------------------------------------
Modelo con profundidad 7:
  - Precisión promedio de validación cruzada: 0.7543
  - Desviación estándar: 0.0134
-------------------------------------------------
```

Según los datos que muestra la matriz de confusión y su reporte de clasificación, clasificó correctamente la categoría de precio para casas caras con un valor de 69 aciertos, asimismo para la categoría

económica con 60 aciertos y la intermedia con 41, teniendo un total de 170 aciertos. Por otro lado, presenta 64 casos mal clasificados.

El modelo cometió más errores en la clase intermedia, con 23 casos mal clasificados como económica y 8 como cara.

Errores más comunes: - Clasificar "Económica" → "Intermedia": 23 veces - Clasificar "Intermedia" → "Económica": 15 veces - Clasificar "Intermedia" → "Cara": 8 veces - Clasificar "Cara" → "Intermedia": 12 veces

Existe un tipo de confusión entre las variables económica e intermedia lo que puede significar que la diferencia entre estas dos no está siendo bien capturada por el modelo. La clase cara es la mejor identificada pero se confunde con intermedia en 12 casos lo que indica que algunas de las casas caras tienen caracteristicas similares a las de clase intermedia.

El modelo es mejor para clasificar casas de clase cara y económica, no tanto para intermedias.

Importancia de los errores: Radica en que, si una casa cara es clasificada como intermedia, significa una pérdida para el vendedor porque puede estarse vendiendo por debajo de su valor verdadero, este es el error más grave y debe minimizarse. También si una casa económica es clasificada como intermedia o cara puede generar problemas entre el vendedor y el comprador, ya que el comprador estaría pagando más de lo que vale u ofrece la casa, lo que llevaría a reclamos y en el peor de los casos, pérdida para el vendedor.

11 10. Modelo con validación cruzada

Entrenamos un modelo con validación cruzada y comparamos con el modelo anterior.

[ ]:
```
# Dividimos en train y test
X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(
    X_clf, y_clf, test_size=0.2, random_state=42
)

# Creamos el árbol de clasificación
clf_tree = DecisionTreeClassifier(random_state=42)
clf_tree.fit(X_train_clf, y_train_clf)

# Validación cruzada
cv_scores = cross_val_score(clf_tree, X_clf, y_clf, cv=5, scoring='accuracy')

# Mostrar resultados de la validación cruzada
print(f"Precisión promedio de validación cruzada: {cv_scores.mean():.4f}")
print(f"Desviación estándar de la validación cruzada: {cv_scores.std():.4f}")
```

- Precisión promedio de validación cruzada: 0.7414
- Desviación estándar de la validación cruzada: 0.0119

Como resultado del modelo con validación cruzada, se tiene una precisión de 0.7414 lo que indica que el modelo clasifica correctamente el 74.14% de las instancias. Esta precisión es mayor comparada con la precisión del modelo sin validación cruzada, por lo que podemos decir que la validación cruzada mejoró el rendimiento.

12 11. Tres modelos más cambiando la profundidad

Realizamos 3 modelos más, cambiando la profundidad del árbol y determinamos cuál funcionó mejor

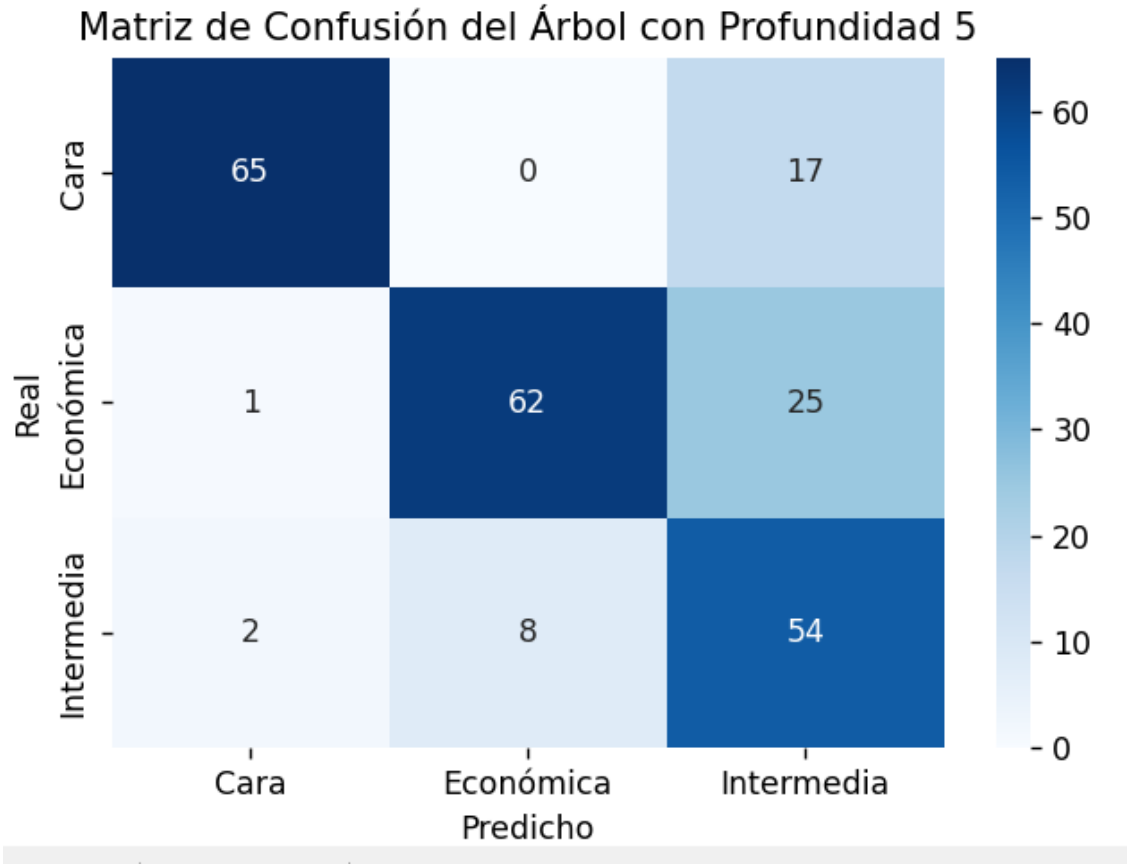
[ ]:
```
# Definir diferentes profundidades para los árboles
depths = [3, 5, 7]

# Crear y evaluar modelos con diferentes profundidades
for depth in depths:
    # Crear el árbol con la profundidad especificada
    clf_tree = DecisionTreeClassifier(max_depth=depth, random_state=42)

    # Realizar validación cruzada y obtener las precisiones
    cv_scores = cross_val_score(clf_tree, X_clf, y_clf, cv=5,␣
 ↪scoring='accuracy')

    # Mostrar resultados
    print(f"Modelo con profundidad {depth}:")
    print(f"  - Precisión promedio de validación cruzada: {cv_scores.mean():.
 ↪4f}")
    print(f"  - Desviación estándar: {cv_scores.std():.4f}")
    print("-" * 50)
```

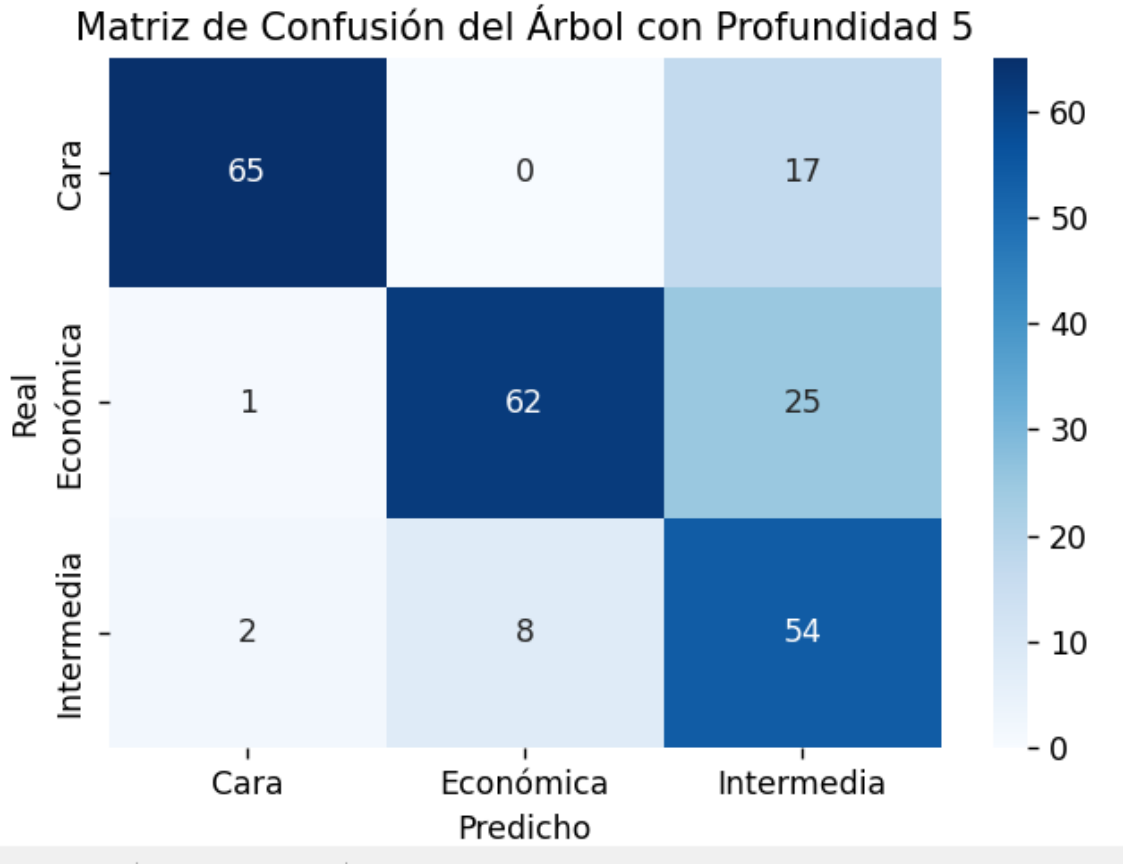
```
Modelo con profundidad 3:
  - Precisión promedio de validación cruzada: 0.7414
  - Desviación estándar: 0.0143
--------------------------------------------------
Modelo con profundidad 5:
  - Precisión promedio de validación cruzada: 0.7560
  - Desviación estándar: 0.0241
--------------------------------------------------
Modelo con profundidad 7:
  - Precisión promedio de validación cruzada: 0.7543
  - Desviación estándar: 0.0134
--------------------------------------------------
```

Se puede notar que el modelo con profundidad 5 es el que tiene mejor precisión con un valor de 0.7560 o 75.60%. A continuación, con el conjunto de prueba realizaremos una matriz de confusión para visualizar mejor los resultados del modelo final con profundidad que tiene mayor precisión:

[ ]:
```
# ================================
# Evaluación final del modelo (con validación cruzada)
# ================================
# El modelo final con la mejor profundidad se entrenará y evaluará aquí
```

```
best_depth = 5
clf_tree_best = DecisionTreeClassifier(max_depth=best_depth, random_state=42)
clf_tree_best.fit(X_train_clf, y_train_clf)

# Predicciones y evaluación
y_pred_clf_best = clf_tree_best.predict(X_test_clf)
accuracy_best = accuracy_score(y_test_clf, y_pred_clf_best)
print(f"Precisión del modelo con profundidad {best_depth}: {accuracy_best:.4f}")

# Mostrar matriz de confusión
conf_matrix_best = confusion_matrix(y_test_clf, y_pred_clf_best)
print("Matriz de confusión del mejor modelo:")
print(conf_matrix_best)

# Mostrar reporte de clasificación
class_report_best = classification_report(y_test_clf, y_pred_clf_best)
print("Reporte de clasificación del mejor modelo:")
print(class_report_best)

# Visualización de la matriz de confusión
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_best, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=clf_tree_best.classes_, yticklabels=clf_tree_best.classes_)
plt.xlabel("Predicho")
plt.ylabel("Real")
plt.title(f"Matriz de Confusión del Árbol con Profundidad {best_depth}")
plt.show()
```


Matriz de Confusión del Árbol con Profundidad 5
Cara
Económica
Intermedia
Real
65
0
17
1
62
25
2
8
54
Cara
Económica
Intermedia
Predicho
60
50
40
30
20
10
0



Según los resultados, el modelo con profundidad 5 es el que tiene una mejor precisión al momento de clasificar una casa entre las categorías "Cara", "intermedia" y "económica". En la matriz de confusión se puede observar una mejoría en la clasificación de las casas, ya que ahora hay 65 aciertos en la clase cara, 62 en económica y 54 en intermedia, para un total de 181 aciertos. Mejorar la profundidad contribuyó a obtener un mejor rendimiento del modelo.

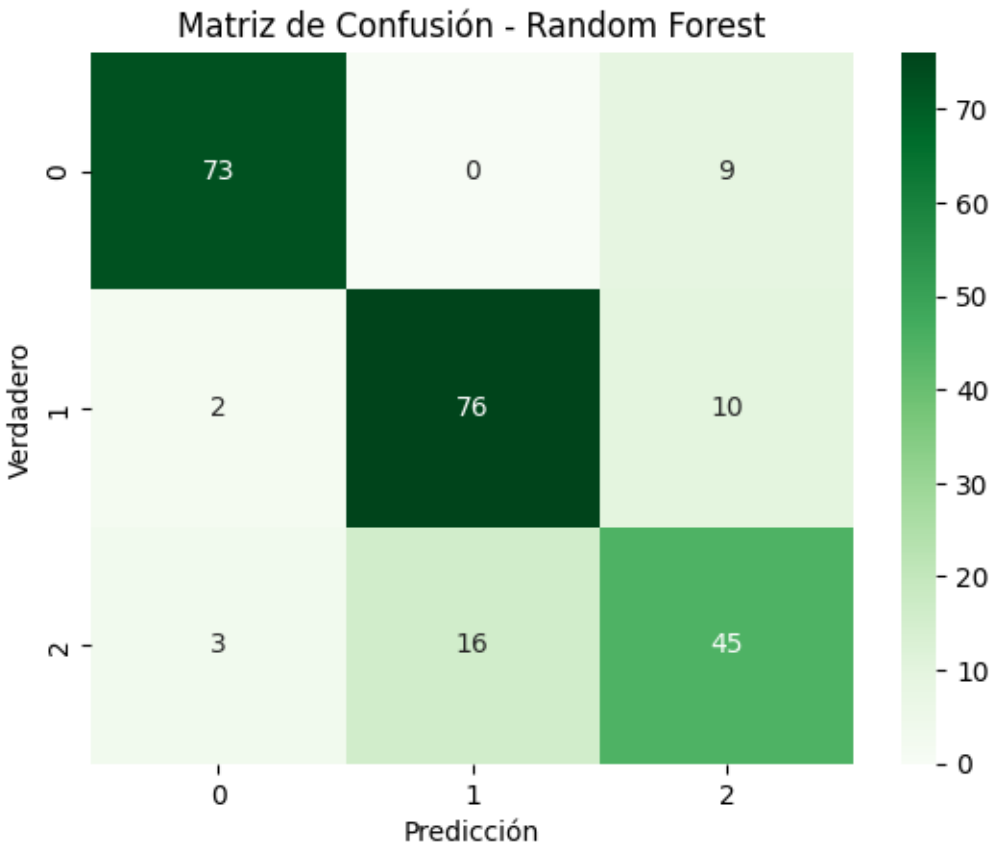
13 12. Análisis utilizando Random Forest

Para comparar con el árbol de decisión, entrenamos ahora un modelo de **Random Forest** (ensamble de múltiples árboles). Revisamos métricas de exactitud, matriz de confusión y observamos si mejora con respecto al árbol de decisión simple

[12]:
```
# ========================================
# 12. Random Forest para clasificación (corregido)
# ========================================
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Entrenamos un RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
rf_clf.fit(X_train_clf, y_train_clf)

# Predicción sobre el test
y_pred_rf = rf_clf.predict(X_test_clf)

# Exactitud del Random Forest
accuracy_rf = accuracy_score(y_test_clf, y_pred_rf)
print("Exactitud (accuracy) de Random Forest en test:", accuracy_rf)

# Matriz de confusión para Random Forest
cm_rf = confusion_matrix(y_test_clf, y_pred_rf)
print("Matriz de Confusión - Random Forest:")
print(cm_rf)

# Visualización de la matriz
import seaborn as sns
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Greens')
plt.title("Matriz de Confusión - Random Forest")
plt.xlabel("Predicción")
plt.ylabel("Verdadero")
plt.show()

# Reporte de clasificación
report_rf = classification_report(y_test_clf, y_pred_rf)
print("Reporte de Clasificación - Random Forest:")
print(report_rf)
```

```
Exactitud (accuracy) de Random Forest en test: 0.8290598290598291
Matriz de Confusión - Random Forest:
[[73  0  9]
 [ 2 76 10]
 [ 3 16 45]]
```




```
Reporte de Clasificación - Random Forest:
              precision    recall  f1-score   support

       Cara       0.94      0.89      0.91        82
   Económica      0.83      0.86      0.84        88
  Intermedia      0.70      0.70      0.70        64

    accuracy                           0.83       234
   macro avg      0.82      0.82      0.82       234
weighted avg      0.83      0.83      0.83       234
```

El modelo alcanzó una exactitud (accuracy) de aproximadamente 0.83 (82.9%) en el conjunto de prueba, lo que refleja un incremento respecto al árbol de decisión simple. De acuerdo con la matriz de confusión, la clase Cara obtuvo 73 aciertos y se confundió con Intermedia en 9 ocasiones; por otro lado, la clase Económica tuvo 76 aciertos y 10 confusiones con Intermedia. La clase Intermedia fue la más propensa a errores, registrando 16 casos mal clasificados como Económica o Cara, lo que coincide con la tendencia habitual de que las clases intermedias resulten más difíciles de distinguir.

En el reporte de clasificación, destaca la precisión (precision) de la clase Cara (0.94) y la exhaustividad (recall) de la clase Económica (0.86). La categoría Intermedia, en cambio, presenta valores

más bajos (0.70 de precisión y 0.70 de recall), confirmando que el modelo tiende a confundir las casas Intermedias con las de mayor o menor precio. Aun así, se logra un f1-score global de 0.83 y un promedio macro de 0.82, indicando un buen desempeño en las tres clases.

En conjunto, estos resultados evidencian que el Random Forest mejora la clasificación respecto a un solo árbol de decisión, pues combina múltiples árboles y, por ende, reduce la varianza y mejora la robustez del modelo. Sin embargo, aún es posible observar ciertos errores en la diferenciación de las categorías Intermedias, un patrón que podría abordarse mediante un mejor ajuste de hiperparámetros (como n_estimators o la profundidad máxima de los árboles en el bosque) o incluyendo variables adicionales que ayuden al modelo a discernir mejor entre las clases.