Practica 2- Datos númericos

Nombre: Paulina Aldape

In [1]:
```
#importar librerias
import pandas as pd
import numpy as np
```

In [37]:
```
#leer la base de datos
bd_casas=pd.read_csv('/content/SaratogaHouses.csv')
```

In [38]:
```
#mostrar la base de datos
bd_casas.head(10)
```

Out[38]:

	price	lotSize	age	landValue	livingArea	pctCollege	bedrooms	fireplaces	bathrooms	roo
0	132500	0.09	42	50000	906	35	2	1	1.0	
1	181115	0.92	0	22300	1953	51	3	0	2.5	
2	109000	0.19	133	7300	1944	51	4	1	1.0	
3	155000	0.41	13	18700	1944	51	3	1	1.5	
4	86060	0.11	0	15000	840	51	2	0	1.0	
5	120000	0.68	31	14000	1152	22	4	1	1.0	
6	153000	0.40	33	23300	2752	51	4	1	1.5	
7	170000	1.21	23	14600	1662	35	4	1	1.5	
8	90000	0.83	36	22200	1632	51	3	0	1.5	
9	122900	1.94	4	21200	1416	44	3	0	1.5	

Tipo de cada columna

In [39]:
```
# Tipo de cada columna
# ==============================================================================
# En pandas, el tipo "object" hace referencia a strings
# datos.dtypes
bd_casas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   price           1728 non-null   int64  
 1   lotSize         1728 non-null   float64
 2   age             1728 non-null   int64  
 3   landValue       1728 non-null   int64  
 4   livingArea      1728 non-null   int64  
 5   pctCollege      1728 non-null   int64  
 6   bedrooms        1728 non-null   int64  
 7   fireplaces      1728 non-null   int64  
 8   bathrooms       1728 non-null   float64
 9   rooms           1728 non-null   int64  
 10  heating         1728 non-null   object 
 11  fuel            1728 non-null   object 
 12  sewer           1728 non-null   object 
 13  waterfront      1728 non-null   object 
 14  newConstruction 1728 non-null   object 
 15  centralAir      1728 non-null   object 
dtypes: float64(2), int64(8), object(6)
memory usage: 216.1+ KB
```

Todas las variables tienen el tipo adecuado

Numero de observaciones y valores ausentes

In [40]:
```
# Dimensiones del dataset
# ==============================================================================
bd_casas.shape
```

Out[40]:
```
(1728, 16)
```

In [41]:
```
# Número de datos ausentes por variable
# ==============================================================================
==
bd_casas.isna().sum().sort_values()
```

Out[41]:
```
price              0
lotSize            0
age                0
landValue          0
livingArea         0
pctCollege         0
bedrooms           0
fireplaces         0
bathrooms          0
rooms              0
heating            0
fuel               0
sewer              0
waterfront         0
newConstruction    0
centralAir         0
dtype: int64
```

Ninguna variable contiene valores ausentes.

Variable respuesta

Para esta base de datos, la variable de respuesta corresponde a la variable de **Precio**

In [10]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

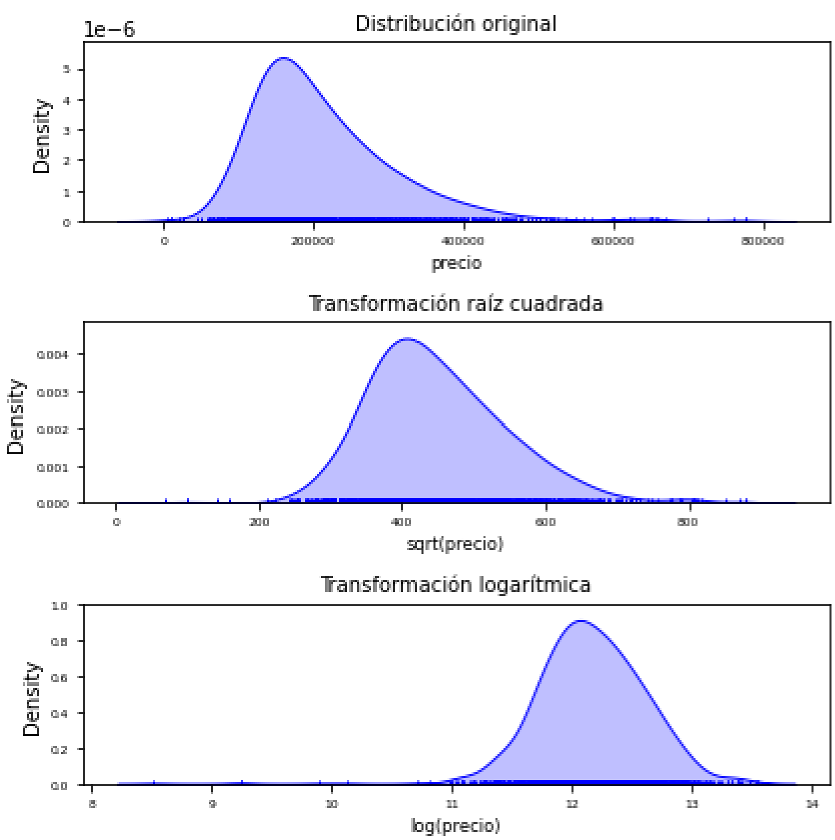
In [43]:
```
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(6, 6))
sns.distplot(
    bd_casas.price,
    hist    = False,
    rug     = True,
    color   = "blue",
    kde_kws = {'shade': True, 'linewidth': 1},
    ax      = axes[0]
)
axes[0].set_title("Distribución original", fontsize = 'medium')
axes[0].set_xlabel('precio', fontsize='small')
axes[0].tick_params(labelsize = 6)

sns.distplot(
    np.sqrt(bd_casas.price),
    hist    = False,
    rug     = True,
    color   = "blue",
    kde_kws = {'shade': True, 'linewidth': 1},
    ax      = axes[1]
)
axes[1].set_title("Transformación raíz cuadrada", fontsize = 'medium')
axes[1].set_xlabel('sqrt(precio)', fontsize='small')
axes[1].tick_params(labelsize = 6)

sns.distplot(
    np.log(bd_casas.price),
    hist    = False,
    rug     = True,
    color   = "blue",
    kde_kws = {'shade': True, 'linewidth': 1},
    ax      = axes[2]
)
axes[2].set_title("Transformación logarítmica", fontsize = 'medium')
axes[2].set_xlabel('log(precio)', fontsize='small')
axes[2].tick_params(labelsize = 6)

fig.tight_layout()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2103: FutureW
arning: The `axis` variable is no longer used and will be removed. Instead, a
ssign variables directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2103: FutureW
arning: The `axis` variable is no longer used and will be removed. Instead, a
ssign variables directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2103: FutureW
arning: The `axis` variable is no longer used and will be removed. Instead, a
ssign variables directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
```



La variable de precio tiene una distribución asimétrica con una cola positiva debido a que, unas pocas viviendas, tienen un nivel de precio superior a la media.

Existen varias librerías en python que permiten identificar a qué distribución se ajustan mejor los datos, una de ellas es fitter. Esta librería permite ajustar cualquiera de las 80 distribuciones implementadas en scipy.

In [17]:
```
!pip install Fitter
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: Fitter in /usr/local/lib/python3.8/dist-packages (1.5.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.8/dist-packages (from Fitter) (1.3.5)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from Fitter) (4.64.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.8/dist-packages (from Fitter) (1.2.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-packages (from Fitter) (3.2.2)
Requirement already satisfied: click in /usr/local/lib/python3.8/dist-packages (from Fitter) (7.1.2)
Requirement already satisfied: scipy>=0.18 in /usr/local/lib/python3.8/dist-packages (from Fitter) (1.7.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from Fitter) (1.21.6)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->Fitter) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->Fitter) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotlib->Fitter) (0.11.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->Fitter) (3.0.9)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas->Fitter) (2022.7)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.1->matplotlib->Fitter) (1.15.0)
```

In [20]:
```
import fitter
from fitter import Fitter, get_common_distributions
```

In [44]:
```
distribuciones = ['cauchy', 'chi2', 'expon',  'exponpow', 'gamma',
                  'norm', 'powerlaw', 'beta', 'logistic']

fitter = Fitter(bd_casas.price, distributions=distribuciones)
fitter.fit()
fitter.summary(Nbest=10, plot=False)
```

```
Fitting 9 distributions: 100%|██████████| 9/9 [00:01<00:00,  6.96it/s]
```

Out[44]:

	sumsquare_error	aic	bic	kl_div	ks_statistic	ks_pvalue
beta	2.497420e-11	3068.852573	-55037.908642	inf	0.051274	2.180145e-04
logistic	4.913880e-11	3147.967042	-53883.297844	inf	0.071310	4.368626e-08
cauchy	5.221450e-11	2956.669693	-53778.388707	inf	0.121728	9.001774e-23
chi2	5.776892e-11	3321.818880	-53596.249282	inf	0.093520	1.326966e-13
norm	6.947514e-11	3324.534158	-53284.856663	inf	0.104149	8.972690e-17
expon	2.915346e-10	2824.103160	-50806.577128	inf	0.316530	2.028906e-154
powerlaw	3.078034e-10	2741.669837	-50705.287086	inf	0.384832	9.418212e-231
exponpow	4.841645e-10	inf	-49922.566370	NaN	1.000000	0.000000e+00
gamma	4.841645e-10	inf	-49922.566370	3.958212	0.947917	0.000000e+00

Variables númericas

In [45]:
```
# Variables numéricas
# ===========================================================================
==
bd_casas.select_dtypes(include=['float64', 'int']).describe()
```

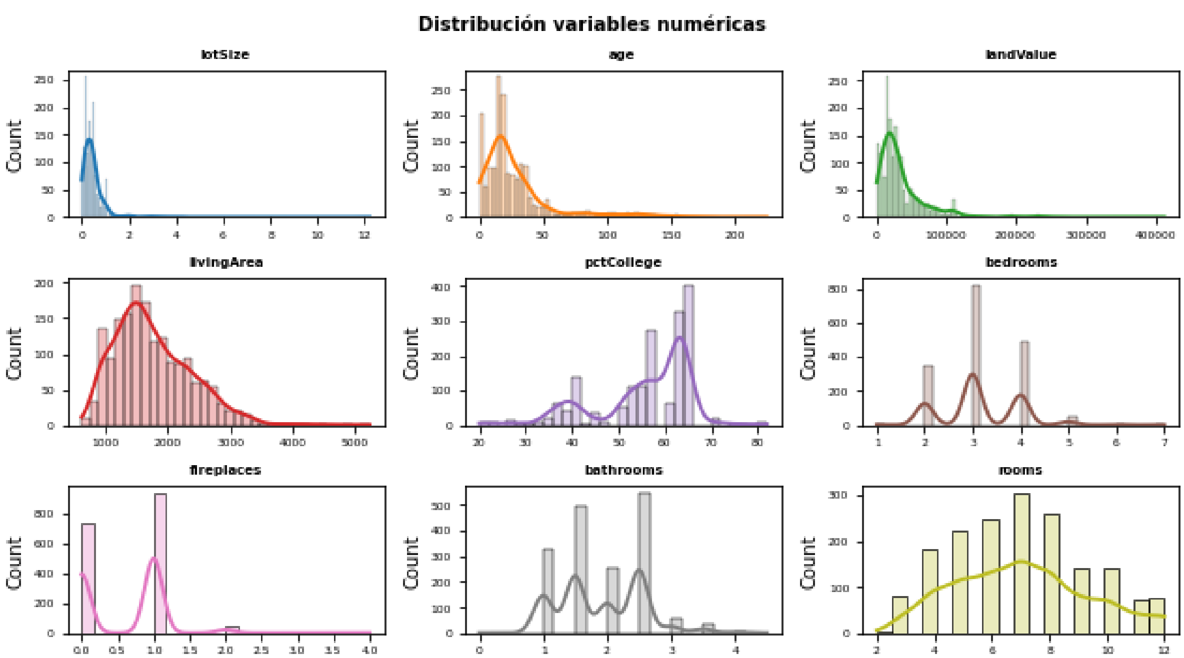
Out[45]:

	price	lotSize	age	landValue	livingArea	pctCollege	bed
count	1728.000000	1728.000000	1728.000000	1728.000000	1728.000000	1728.000000	1728.0
mean	211966.705440	0.500214	27.916088	34557.187500	1754.975694	55.567708	3.1
std	98441.391015	0.698680	29.209988	35021.168056	619.935553	10.333581	0.8
min	5000.000000	0.000000	0.000000	200.000000	616.000000	20.000000	1.0
25%	145000.000000	0.170000	13.000000	15100.000000	1300.000000	52.000000	3.0
50%	189900.000000	0.370000	19.000000	25000.000000	1634.500000	57.000000	3.0
75%	259000.000000	0.540000	34.000000	40200.000000	2137.750000	64.000000	4.0
max	775000.000000	12.200000	225.000000	412600.000000	5228.000000	82.000000	7.0

In [47]:
```
# Gráfico de distribución para cada variable numérica
# ==============================================================================
==
# Ajustar número de subplots en función del número de columnas
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(9, 5))
axes = axes.flat
columnas_numeric = bd_casas.select_dtypes(include=['float64', 'int']).columns
columnas_numeric = columnas_numeric.drop('price')

for i, colum in enumerate(columnas_numeric):
    sns.histplot(
        data    = bd_casas,
        x       = colum,
        stat    = "count",
        kde     = True,
        color   = (list(plt.rcParams['axes.prop_cycle'])*2)[i]["color"],
        line_kws= {'linewidth': 2},
        alpha   = 0.3,
        ax      = axes[i]
    )
    axes[i].set_title(colum, fontsize = 7, fontweight = "bold")
    axes[i].tick_params(labelsize = 6)
    axes[i].set_xlabel("")


fig.tight_layout()
plt.subplots_adjust(top = 0.9)
fig.suptitle('Distribución variables numéricas', fontsize = 10, fontweight =
"bold");
```



Se puede observar que la variable de outcome es una variable chimeneas por tener pocos valores se puede considerar como tipo string

In [48]:
```
# Valores observados de outcome
# ===========================================================================
==
bd_casas.fireplaces.value_counts()
```

Out[48]:
```
1    942
0    740
2     42
4      2
3      2
Name: fireplaces, dtype: int64
```

In [49]:
```
# Se convierte la variable outcome tipo string
# ===========================================================================
==
bd_casas.fireplaces = bd_casas.fireplaces.astype("str")
```

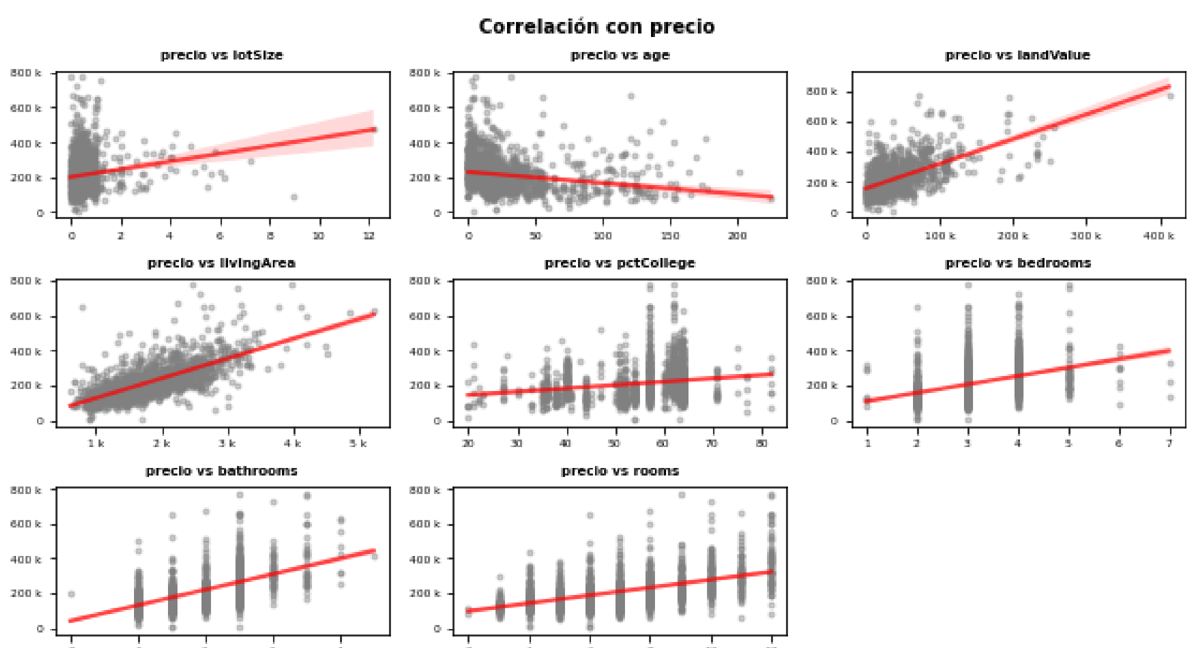
In [28]:
```
import matplotlib.ticker as ticker
```

In [51]:
```
# Gráfico de distribución para cada variable numérica
# ============================================================================
==
# Ajustar número de subplots en función del número de columnas
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(9, 5))
axes = axes.flat
columnas_numeric = bd_casas.select_dtypes(include=['float64', 'int']).columns
columnas_numeric = columnas_numeric.drop('price')

for i, colum in enumerate(columnas_numeric):
    sns.regplot(
        x           = bd_casas[colum],
        y           = bd_casas['price'],
        color       = "gray",
        marker      = '.',
        scatter_kws = {"alpha":0.4},
        line_kws    = {"color":"r","alpha":0.7},
        ax          = axes[i]
    )
    axes[i].set_title(f"precio vs {colum}", fontsize = 7, fontweight = "bold")
    #axes[i].ticklabel_format(style='sci', scilimits=(-4,4), axis='both')
    axes[i].yaxis.set_major_formatter(ticker.EngFormatter())
    axes[i].xaxis.set_major_formatter(ticker.EngFormatter())
    axes[i].tick_params(labelsize = 6)
    axes[i].set_xlabel("")
    axes[i].set_ylabel("")

# Se eliminan los axes vacíos
for i in [8]:
    fig.delaxes(axes[i])

fig.tight_layout()
plt.subplots_adjust(top=0.9)
fig.suptitle('Correlación con precio', fontsize = 10, fontweight = "bold");
```



Correlacion variables numericas

In [54]:
```
# Correlación entre columnas numéricas
# ==============================================================================
==

def tidy_corr_matrix(corr_mat):
    '''
    Función para convertir una matrix de correlación de pandas en formato tidy
    '''
    corr_mat = corr_mat.stack().reset_index()
    corr_mat.columns = ['variable_1','variable_2','r']
    corr_mat = corr_mat.loc[corr_mat['variable_1'] != corr_mat['variable_2'],
:]
    corr_mat['abs_r'] = np.abs(corr_mat['r'])
    corr_mat = corr_mat.sort_values('abs_r', ascending=False)

    return(corr_mat)



corr_matrix = bd_casas.select_dtypes(include=['float64', 'int']).corr(method
='pearson')
tidy_corr_matrix(corr_matrix).head(30)
```

Out[54]:

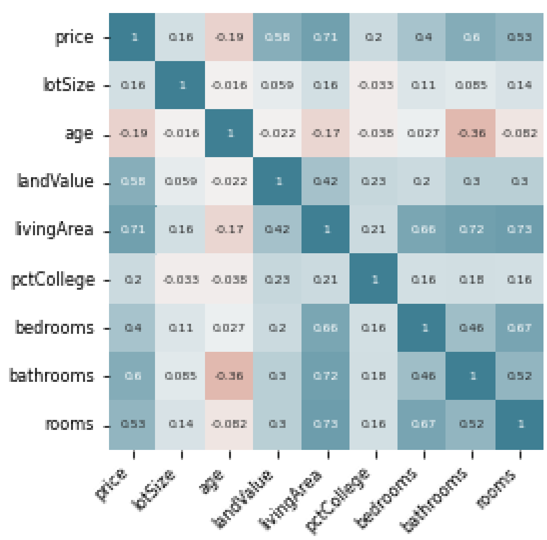
	variable_1	variable_2	r	abs_r
44	livingArea	rooms	0.733666	0.733666
76	rooms	livingArea	0.733666	0.733666
67	bathrooms	livingArea	0.718564	0.718564
43	livingArea	bathrooms	0.718564	0.718564
36	livingArea	price	0.712390	0.712390
4	price	livingArea	0.712390	0.712390
78	rooms	bedrooms	0.671863	0.671863
62	bedrooms	rooms	0.671863	0.671863
42	livingArea	bedrooms	0.656196	0.656196
58	bedrooms	livingArea	0.656196	0.656196
63	bathrooms	price	0.597250	0.597250
7	price	bathrooms	0.597250	0.597250
3	price	landValue	0.581266	0.581266
27	landValue	price	0.581266	0.581266
8	price	rooms	0.531170	0.531170
72	rooms	price	0.531170	0.531170
71	bathrooms	rooms	0.517585	0.517585
79	rooms	bathrooms	0.517585	0.517585
61	bedrooms	bathrooms	0.458033	0.458033
69	bathrooms	bedrooms	0.458033	0.458033
39	livingArea	landValue	0.423441	0.423441
31	landValue	livingArea	0.423441	0.423441
54	bedrooms	price	0.400349	0.400349
6	price	bedrooms	0.400349	0.400349
65	bathrooms	age	-0.361897	0.361897
25	age	bathrooms	-0.361897	0.361897
75	rooms	landValue	0.298865	0.298865
35	landValue	rooms	0.298865	0.298865
66	bathrooms	landValue	0.297498	0.297498
34	landValue	bathrooms	0.297498	0.297498

In [55]:

```
# Heatmap matriz de correlaciones
# ==============================================================================
==
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(4, 4))

sns.heatmap(
    corr_matrix,
    annot     = True,
    cbar      = False,
    annot_kws = {"size": 6},
    vmin      = -1,
    vmax      = 1,
    center    = 0,
    cmap      = sns.diverging_palette(20, 220, n=200),
    square    = True,
    ax        = ax
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation = 45,
    horizontalalignment = 'right',
)

ax.tick_params(labelsize = 8)
```



De las variables con una correlacion más alta entre ellas son la variable de livingArea con la variable de precio, baños, recamaras y cuartos. La correlación negativa más alta que se tiene es entre la variable de Años de la propiedad y el numero de baños.

Variables cualitativas

In [56]:
```
# Variables cualitativas (tipo object)
# ===========================================================================
==
bd_casas.select_dtypes(include=['object']).describe()
```

Out[56]:

	fireplaces	heating	fuel	sewer	waterfront	newConstruction	centralAir
count	1728	1728	1728	1728	1728	1728	1728
unique	5	3	3	3	2	2	2
top	1	hot air	gas	public/commercial	No	No	No
freq	942	1121	1197	1213	1713	1647	1093

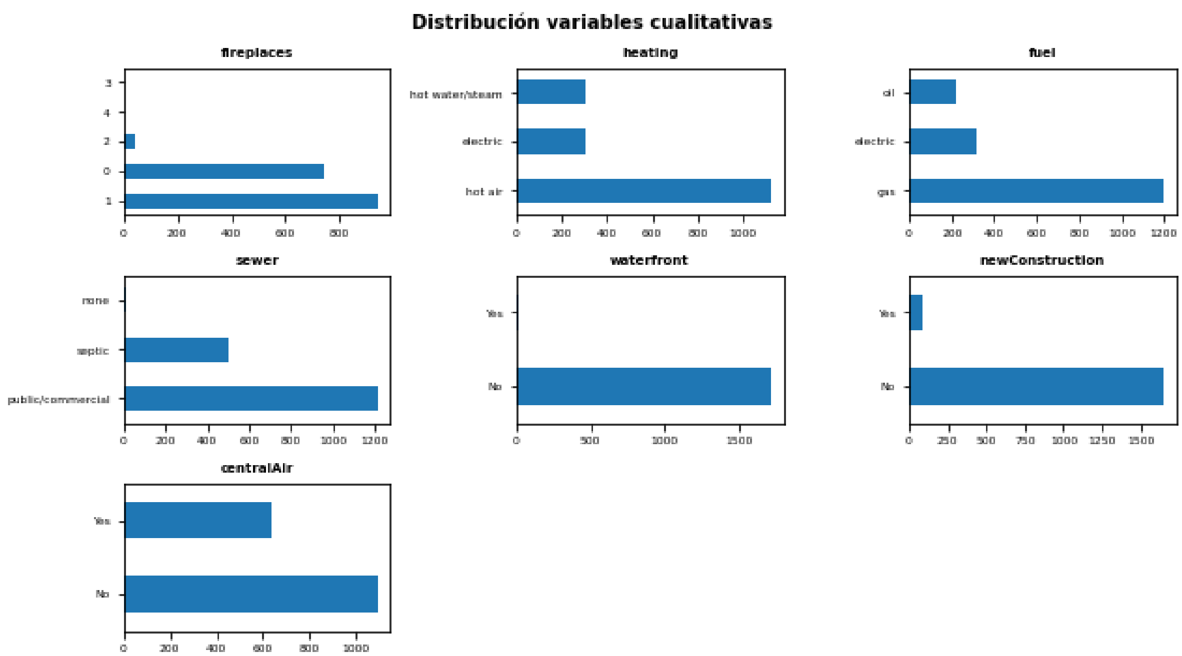
In [58]:

```python
# Gráfico para cada variable cualitativa
# ============================================================================
==
# Ajustar número de subplots en función del número de columnas
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(9, 5))
axes = axes.flat
columnas_object = bd_casas.select_dtypes(include=['object']).columns

for i, colum in enumerate(columnas_object):
    bd_casas[colum].value_counts().plot.barh(ax = axes[i])
    axes[i].set_title(colum, fontsize = 7, fontweight = "bold")
    axes[i].tick_params(labelsize = 6)
    axes[i].set_xlabel("")

# Se eliminan los axes vacíos
for i in [7, 8]:
    fig.delaxes(axes[i])

fig.tight_layout()
plt.subplots_adjust(top=0.9)
fig.suptitle('Distribución variables cualitativas',
             fontsize = 10, fontweight = "bold");
```



Si alguno de los niveles de una variable cualitativa tiene muy pocas observaciones en comparación a los otros niveles, puede ocurrir que, durante la validación cruzada o bootstrapping, algunas particiones no contengan ninguna observación de dicha clase (varianza cero), lo que puede dar lugar a errores. En estos casos, suele ser conveniente:

Eliminar las observaciones del grupo minoritario si es una variable multiclase.

Eliminar la variable si solo tiene dos niveles.

Agrupar los niveles minoritarios en un único grupo.

Asegurar que, en la creación de las particiones, todos los grupos estén representados en cada una de ellas.

Para este caso, hay que tener precaución con la variable chimenea. Se unifican los niveles de 2, 3 y 4 en un nuevo nivel llamado "2_mas".

In [60]:
```
bd_casas.fireplaces.value_counts().sort_index()
```

Out[60]:
```
0    740
1    942
2     42
3      2
4      2
Name: fireplaces, dtype: int64
```

In [61]:
```
dic_replace = {'2': "2_mas",
               '3': "2_mas",
               '4': "2_mas"}

bd_casas['fireplaces'] = bd_casas['fireplaces'] \
                     .map(dic_replace) \
                     .fillna(bd_casas['fireplaces'])
```

In [62]:
```
bd_casas.fireplaces.value_counts().sort_index()
```

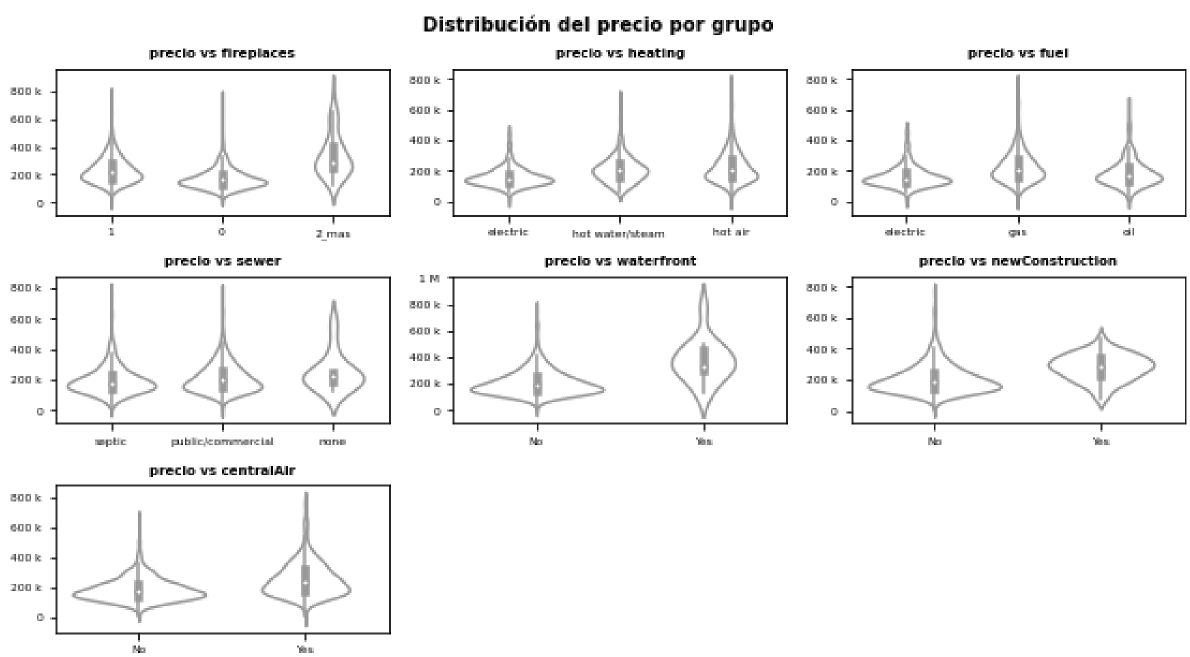
Out[62]:
```
0        740
1        942
2_mas     46
Name: fireplaces, dtype: int64
```

In [64]:
```
# Gráfico relación entre el precio y cada cada variables cualitativas
# ==============================================================================
# Ajustar número de subplots en función del número de columnas
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(9, 5))
axes = axes.flat
columnas_object = bd_casas.select_dtypes(include=['object']).columns

for i, colum in enumerate(columnas_object):
    sns.violinplot(
        x     = colum,
        y     = 'price',
        data  = bd_casas,
        color = "white",
        ax    = axes[i]
    )
    axes[i].set_title(f"precio vs {colum}", fontsize = 7, fontweight = "bold")
    axes[i].yaxis.set_major_formatter(ticker.EngFormatter())
    axes[i].tick_params(labelsize = 6)
    axes[i].set_xlabel("")
    axes[i].set_ylabel("")

# Se eliminan los axes vacíos
for i in [7, 8]:
    fig.delaxes(axes[i])

fig.tight_layout()
plt.subplots_adjust(top=0.9)
fig.suptitle('Distribución del precio por grupo', fontsize = 10, fontweight = "bold");
```



Division de train y test

In [65]:
```
# Reparto de datos en train y test
# ==============================================================================
==
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
                                        bd_casas.drop('price', axis = 'column
s'),
                                        bd_casas['price'],
                                        train_size   = 0.8,
                                        random_state = 1234,
                                        shuffle      = True
                                    )
```

In [66]:
```
print("Partición de entrenamiento")
print("-----------------------")
print(y_train.describe())
```

```
Partición de entrenamiento
-----------------------
count      1382.000000
mean     211436.516643
std       96846.639129
min       10300.000000
25%      145625.000000
50%      190000.000000
75%      255000.000000
max      775000.000000
Name: price, dtype: float64
```

In [67]:
```
print("Partición de test")
print("-----------------------")
print(y_test.describe())
```

```
Partición de test
-----------------------
count       346.000000
mean     214084.395954
std      104689.155889
min        5000.000000
25%      139000.000000
50%      180750.000000
75%      271750.000000
max      670000.000000
Name: price, dtype: float64
```