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

from utils import bootcampviztools as bt
from utils import toolbox_ML as tl

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error,
mean_absolute_percentage_error, root_mean_squared_error
from sklearn.model_selection import GridSearchCV, cross_val_score,
train_test_split

from catboost import CatBoostRegressor
from xgboost import XGBRegressor
```

Objetivo del proyecto

El objetivo de este proyecto es crear un modelo de ML que permita predecir el precio de venta de una casa teniendo en cuenta una serie de características. Para conseguir el objetivo realizaremos un modelo de regresión supervisado.

Extracción y entendimiento de los datos

Los datos han sido obtenidos de un dataset de **Kaggle**: Link de los datos.

Los datos contienen información sobre la venta de viviendas en el condado de King, Washington desde mayo de 2014 hasta mayo de 2015. Para tener los datos de una manera más accesible a la hora de manipularlos han sido almacenados en la carpeta `data`, así como otras posibles versiones de los datos en las que se hayan eliminado o añadido datos.

Los modelos que serán usados y comparados son: **Regresión Lineal**, **Random Forest Regressor**, **XGBoost** y **LightGBM**.

```
"""Carga de los datos"""
df = pd.read_csv('./data/kc_house_data.csv')
print(df.info())
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
```

```
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
None

           id             date     price  bedrooms  bathrooms
sqft_living  \
0  7129300520  20141013T000000  221900.0         3       1.00
1180
1  6414100192  20141209T000000  538000.0         3       2.25
2570
2  5631500400  20150225T000000  180000.0         2       1.00
770
3  2487200875  20141209T000000  604000.0         4       3.00
1960
4  1954400510  20150218T000000  510000.0         3       2.00
1680

   sqft_lot  floors  waterfront  view  ...  grade  sqft_above
sqft_basement  \
0      5650     1.0           0     0  ...      7        1180
0
1      7242     2.0           0     0  ...      7        2170
400
2     10000     1.0           0     0  ...      6         770
0
3      5000     1.0           0     0  ...      7        1050
910
4      8080     1.0           0     0  ...      8        1680
0

   yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
0      1955             0    98178  47.5112 -122.257           1340
1      1951          1991    98125  47.7210 -122.319           1690
2      1933             0    98028  47.7379 -122.233           2720
```

```
3      1965             0    98136  47.5208 -122.393           1360
4      1987             0    98074  47.6168 -122.045           1800

   sqft_lot15
0        5650
1        7639
2        8062
3        5000
4        7503

[5 rows x 21 columns]
```

```
pd.options.display.float_format = '{:.3f}'.format
df.describe().T
```

```
                 count           mean            std         min  \
id             21613.000 4580301520.865 2876565571.312 1000102.000
price          21613.000     540088.142     367127.196   75000.000
bedrooms       21613.000          3.371          0.930       0.000
bathrooms      21613.000          2.115          0.770       0.000
sqft_living    21613.000       2079.900        918.441     290.000
sqft_lot       21613.000      15106.968      41420.512     520.000
floors         21613.000          1.494          0.540       1.000
waterfront     21613.000          0.008          0.087       0.000
view           21613.000          0.234          0.766       0.000
condition      21613.000          3.409          0.651       1.000
grade          21613.000          7.657          1.175       1.000
sqft_above     21613.000       1788.391        828.091     290.000
sqft_basement  21613.000        291.509        442.575       0.000
yr_built       21613.000       1971.005         29.373    1900.000
yr_renovated   21613.000         84.402        401.679       0.000
zipcode        21613.000      98077.940         53.505   98001.000
lat            21613.000         47.560          0.139      47.156
long           21613.000       -122.214          0.141    -122.519
sqft_living15  21613.000       1986.552        685.391     399.000
sqft_lot15     21613.000      12768.456      27304.180     651.000

                          25%            50%            75%
max
id             2123049194.000 3904930410.000 7308900445.000
9900000190.000
price              321950.000     450000.000     645000.000
7700000.000
bedrooms                3.000          3.000          4.000
33.000
bathrooms               1.750          2.250          2.500
8.000
sqft_living          1427.000       1910.000       2550.000
13540.000
sqft_lot             5040.000       7618.000      10688.000
```

```
1651359.000
floors                 1.000          1.500          2.000
3.500
waterfront             0.000          0.000          0.000
1.000
view                   0.000          0.000          0.000
4.000
condition              3.000          3.000          4.000
5.000
grade                  7.000          7.000          8.000
13.000
sqft_above          1190.000       1560.000       2210.000
9410.000
sqft_basement          0.000          0.000        560.000
4820.000
yr_built            1951.000       1975.000       1997.000
2015.000
yr_renovated           0.000          0.000          0.000
2015.000
zipcode            98033.000      98065.000      98118.000
98199.000
lat                   47.471         47.572         47.678
47.778
long                -122.328       -122.230       -122.125       -
121.315
sqft_living15       1490.000       1840.000       2360.000
6210.000
sqft_lot15          5100.000       7620.000      10083.000
871200.000
```

Se puede ver que el tipo de datos de las columnas del dataset son de tipo `int` y `float`, es decir, de tipo numérico. El dataset cuenta con un total de 21 columnas y un total de 21613 filas (o instancias). No presenta valores nulos.

Encontramos una columna tipo `object`: **date**. Esta columna nos indica la fecha en la que se realizó la venta.

En cuanto a la descripción de los valores de la columna `price` podemos ver que el 75% de los precios se encuentran por debajo de 645.000 dólares, mientras que el valor máximo se encuentra en 7.700.7000 dólares, algo que puede indicar valores outliers en nuestra variable target. También podemos encontrar esto en la columna `bathrooms` donde hay un valor máximo de 33 baños mientras que el 75% están por debajo de 4 baños.

La variable `waterfront` es una variable **binaria**, ya que tanto en sus cuartiles como en su máximo y mínimo los únicos valores que encontramos son 0 y 1.

Vamos a realizar una tabla informativa para poder entender de una manera más sencilla qué representa cada variable.

Variable	Tipo	Descripción	Notas
id	int64	Identificador único de la vivienda	Puede no ser relevante para el análisis
date	object	Fecha de venta de la vivienda	Formato de fecha puede requerir conversión
price	float64	Precio de venta de la vivienda en dólares	Variable objetivo para predicción de precios
bedrooms	int64	Número de habitaciones en la vivienda	Puede incluir dormitorios pequeños
bathrooms	float64	Número de baños en la vivienda	Considerado medios baños como 0.5 (los que no tienen ducha)
sqft_living	int64	Metros cuadrados de la vivienda habitable	Relacionado con el precio
sqft_lot	int64	Metros cuadrados del terreno	Incluye jardín y otros espacios exteriores
floors	float64	Número de pisos de la vivienda	Puede ser decimal si hay entrepisos
waterfront	int64	1 si la vivienda tiene vista al agua, 0 si no	Variable categórica binaria
view	int64	Puntuación de la vista de la vivienda (0-4)	0 indica sin vista, 4 es la mejor vista
condition	int64	Estado general de la vivienda (1-5)	1 es malo, 5 es excelente
grade	int64	Calidad de la construcción y acabados (1-13)	Basado en estándares de construcción
sqft_above	int64	Metros cuadrados de la	Excluye

Variable	Tipo	Descripción	Notas
		parte sobre el suelo	sótano
sqft_basement	int64	Metros cuadrados del sótano	0 si no tiene sótano
yr_built	int64	Año de construcción de la vivienda	Puede influir en el precio y estado
yr_renovated	int64	Año de la última renovación, 0 si nunca ha sido renovada	Puede afectar el valor de la vivienda
zipcode	int64	Código postal de la ubicación	Puede utilizarse para análisis geoespacial
lat	float64	Latitud de la vivienda	Coordenada geográfica
long	float64	Longitud de la vivienda	Coordenada geográfica
sqft_living15	int64	Metros cuadrados promedio de las viviendas cercanas	Indicador del valor del vecindario
sqft_lot15	int64	Metros cuadrados promedio del terreno en la zona	Puede influir en el precio

```
'''Cambiar formato de la fecha de venta'''
df = df.rename({'date':'sale_date'}, axis = 1)
df['sale_date'] = pd.to_datetime(df['sale_date'], yearfirst=True)
```

Cardinalidad, valores missing y tipo de variable.

Aunque ya hemos realizado una tabla informativa de cada variable anteriormente, la siguiente tabla nos aportará información sobre la cardinalidad de las variables, el número de valores únicos, porcentaje de valores missing y el tipo de variable. Esta información es útil para el análisis de las variables y como tratarlas.

```
tl.describe_df(df)
```

```
          columna             tipo  %_nulos  valores_unicos
%_cardinalidad
0              id            int64    0.000           21436
99.180
1       sale_date  datetime64[ns]    0.000             372
1.720
2           price          float64    0.000            4028
18.640
```

```
3        bedrooms           int64    0.000              13
0.060
4       bathrooms         float64    0.000              30
0.140
5     sqft_living           int64    0.000            1038
4.800
6        sqft_lot           int64    0.000            9782
45.260
7          floors         float64    0.000               6
0.030
8      waterfront           int64    0.000               2
0.010
9            view           int64    0.000               5
0.020
10      condition           int64    0.000               5
0.020
11          grade           int64    0.000              12
0.060
12     sqft_above           int64    0.000             946
4.380
13  sqft_basement           int64    0.000             306
1.420
14       yr_built           int64    0.000             116
0.540
15   yr_renovated           int64    0.000              70
0.320
16        zipcode           int64    0.000              70
0.320
17            lat         float64    0.000            5034
23.290
18           long         float64    0.000             752
3.480
19  sqft_living15           int64    0.000             777
3.600
20     sqft_lot15           int64    0.000            8689
40.200
```

Podemos ver que hay algunas variables categóricas como `waterfront` siendo esta binaria y ya mencionada anteriormente, `floor` con 6 valores únicos, `view` con 5 valores únicos, `condition` con 5 valores únicos. También se podrían tratar como categóricas en un principio las features `bedrooms` y `grade`, pero lo veremos más adelante en el análisis de features.

La columna id no contiene un 100% de la cardinalidad, pero se queda muy próximo. Esto indica que hay algunos índices que están duplicados en el df, algo que no es común ya que el id debería de ser un identificador único.

```
df['id'].duplicated().value_counts()
```

```
id
False    21436
True       177
Name: count, dtype: int64
```

Hay un total de 177 ids que están duplicados (1%).

```
df.loc[df['id'].duplicated(keep=False)].sort_values(by = 'id',
ascending=False).head()
```

```
               id  sale_date      price  bedrooms  bathrooms
sqft_living  \
1085   9834200885 2014-07-17 360000.000         4      2.500
2080
1086   9834200885 2015-04-20 550000.000         4      2.500
2080
15199  9834200305 2014-07-16 350000.000         3      1.000
1790
15200  9834200305 2015-02-10 615000.000         3      1.000
1790
6345   9828200460 2014-06-27 260000.000         2      1.000
700

       sqft_lot  floors  waterfront  view  ...  grade  sqft_above  \
1085       4080   1.000           0     0  ...      7        1040
1086       4080   1.000           0     0  ...      7        1040
15199      3876   1.500           0     0  ...      7        1090
15200      3876   1.500           0     0  ...      7        1090
6345       4800   1.000           0     0  ...      7         700

       sqft_basement  yr_built  yr_renovated  zipcode    lat     long
\
1085            1040      1962             0    98144 47.572 -122.290

1086            1040      1962             0    98144 47.572 -122.290

15199            700      1904             0    98144 47.575 -122.288

15200            700      1904             0    98144 47.575 -122.288

6345               0      1922             0    98122 47.615 -122.300


       sqft_living15  sqft_lot15
1085            1340        4080
1086            1340        4080
15199           1360        4080
15200           1360        4080
6345            1440        4800
```

```
[5 rows x 21 columns]
```

Viendo los ids repetidos en el df observamos que las únicas columnas que cambian son la fecha de venta y el precio de venta, esto indica que hay viviendas que han sido vendidas más de una vez durante este periodo de tiempo.

Para saber la cantidad de veces que se ha vendido una casa crearemos una nueva columna llamada `ventas` que contabilizará el número de veces en base a cuántas veces se repite el id.

```
df['ventas'] = df.groupby('id')['id'].transform('count')
df[df['ventas'] > 1]
df.ventas.value_counts()

ventas
1    21260
2      350
3        3
Name: count, dtype: int64
```

Hay un total de 170 casas que han sido vendidas 2 veces y hay una casa que ha sido vendida un total de 3 veces.

Como nuestro objetivo principal consiste en predecir el precio de una casa de manera inicial nos quedaremos con el primer registro de la venta de la casa y eliminaremos el resto ya que las futuras ventas han sido condicionadas por el precio anterior de la misma.

```
df = df.drop_duplicates(subset=['id'], keep='first')
df.ventas.value_counts()

ventas
1    21260
2      175
3        1
Name: count, dtype: int64
```

Usaremos la fecha de venta de las viviendas como índice del dataset ya que no nos aportará mayor utilidad para el modelo.

Borraremos el id de las viviendas ya que tampoco nos servirá para la creación del modelo.

```
df = df.set_index('sale_date')
df = df.drop(columns='id', axis = 1)

bedrooms_33 = df[df['bedrooms'] == 33].index
df = df.drop(bedrooms_33)

df_limpio = df.to_csv('./data/df_limpio_modelo.csv')
```

Análisis de los datos

Una vez hemos establecido un objetivo de negocio, extraído, entendido y limpiado los datos, comenzamos con el análisis de las variables, tanto de la variable `target` (precio en nuestro caso) y el resto de variables.

Para comenzar con el análisis recuperaremos el nuevo dataset con todos los cambios que hemos realizado anteriormente, el cual se encuentra el la carpeta `data`.

- Eliminamos la columna `ventas` ya que ya no es necesaria.
- Establecemos de nuevo `sale_date` como nuestro índice.

```
df = pd.read_csv("./data/df_limpio_modelo.csv")
df = df.drop(columns='ventas', axis = 1)
df = df.set_index('sale_date')
df.head()
```

```
                price  bedrooms  bathrooms  sqft_living  sqft_lot
floors  \
sale_date

2014-10-13 221900.000         3      1.000         1180      5650
1.000
2014-12-09 538000.000         3      2.250         2570      7242
2.000
2015-02-25 180000.000         2      1.000          770     10000
1.000
2014-12-09 604000.000         4      3.000         1960      5000
1.000
2015-02-18 510000.000         3      2.000         1680      8080
1.000

            waterfront  view  condition  grade  sqft_above
sqft_basement  \
sale_date

2014-10-13           0     0          3      7        1180
0
2014-12-09           0     0          3      7        2170
400
2015-02-25           0     0          3      6         770
0
2014-12-09           0     0          5      7        1050
910
2015-02-18           0     0          3      8        1680
0

            yr_built  yr_renovated  zipcode    lat     long
sqft_living15  \
sale_date
```

```
2014-10-13      1955             0    98178 47.511 -122.257
1340
2014-12-09      1951          1991    98125 47.721 -122.319
1690
2015-02-25      1933             0    98028 47.738 -122.233
2720
2014-12-09      1965             0    98136 47.521 -122.393
1360
2015-02-18      1987             0    98074 47.617 -122.045
1800

            sqft_lot15
sale_date
2014-10-13        5650
2014-12-09        7639
2015-02-25        8062
2014-12-09        5000
2015-02-18        7503
```

Una vez hemos recuperado nuestro dataset, divideremos los datos en dos grupos:

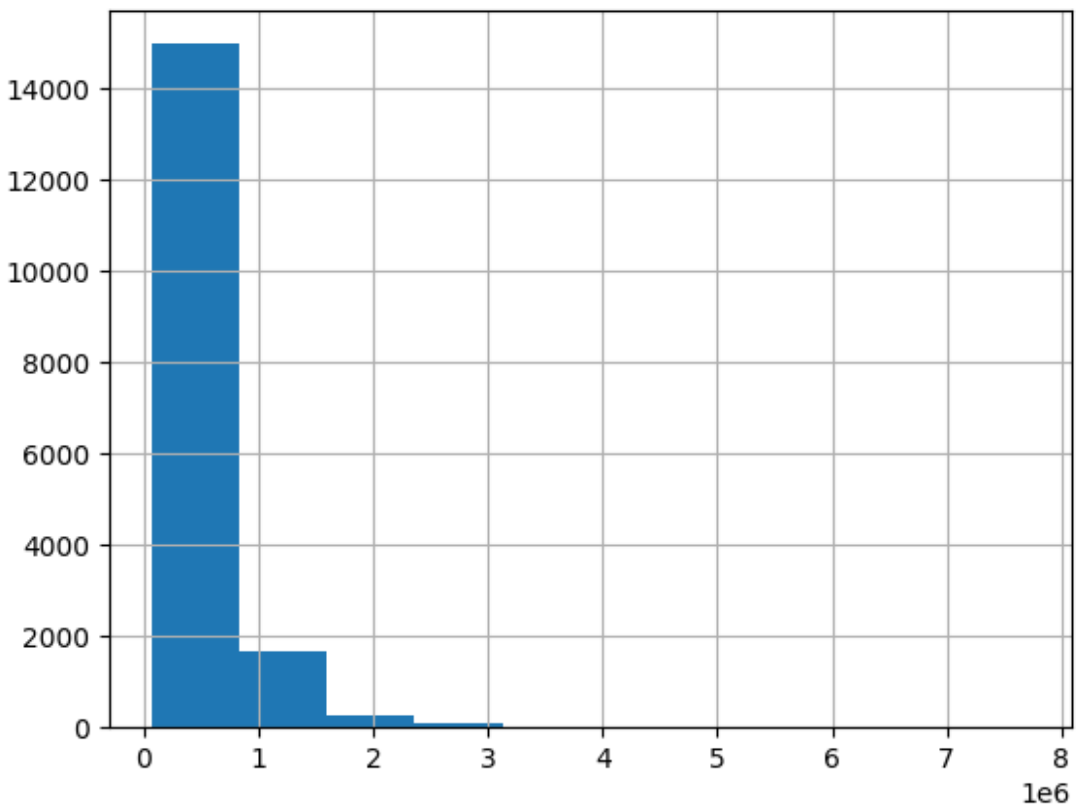
- `Train set`. Este set nos servirá para visualizar los datos y realizar las transformaciones necesarias. También será el set con el que entrenaremos a nuestro modelo / modelos.
- `Test set`. Este set lo guardaremos y lo usaremos contra los resultados obtenidos en nuestro modelo para saber cómo de preciso es.

```
"""Establecer la columna'price' como target"""
target = "price"

train_set, test_set = train_test_split(df, test_size = 0.2,
random_state = 42)
```

Análisis de la variable `target`

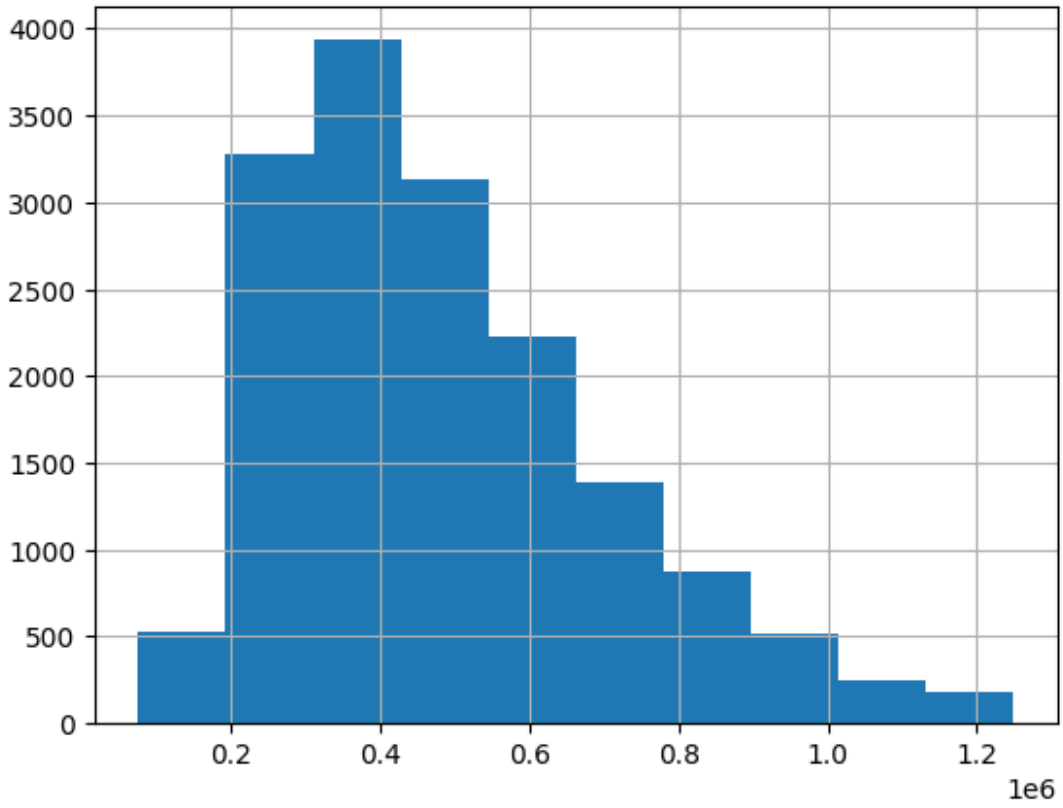
Realizamos un análisis de nuestra variable target para saber el tipo de distribución que tiene.

```
train_set[target].hist();
```



Nuestra variable target no sigue una distribución normal y presenta una cola hacia la derecha.

```
precios_bajos = train_set.loc[train_set[target] < 1250000]
precios_bajos[target].hist();
```



También se observa que la mayoría de las casas tienen un precio entre 200.000 y 1.000.000 de dólares. Hay algunas casas que superan esos precios con creces, llegando hasta un valor máximo de casi 8.000.000 de dólares. Esta no es la distribución ideal para un modelo de regresión lineal.

Análisis de las features

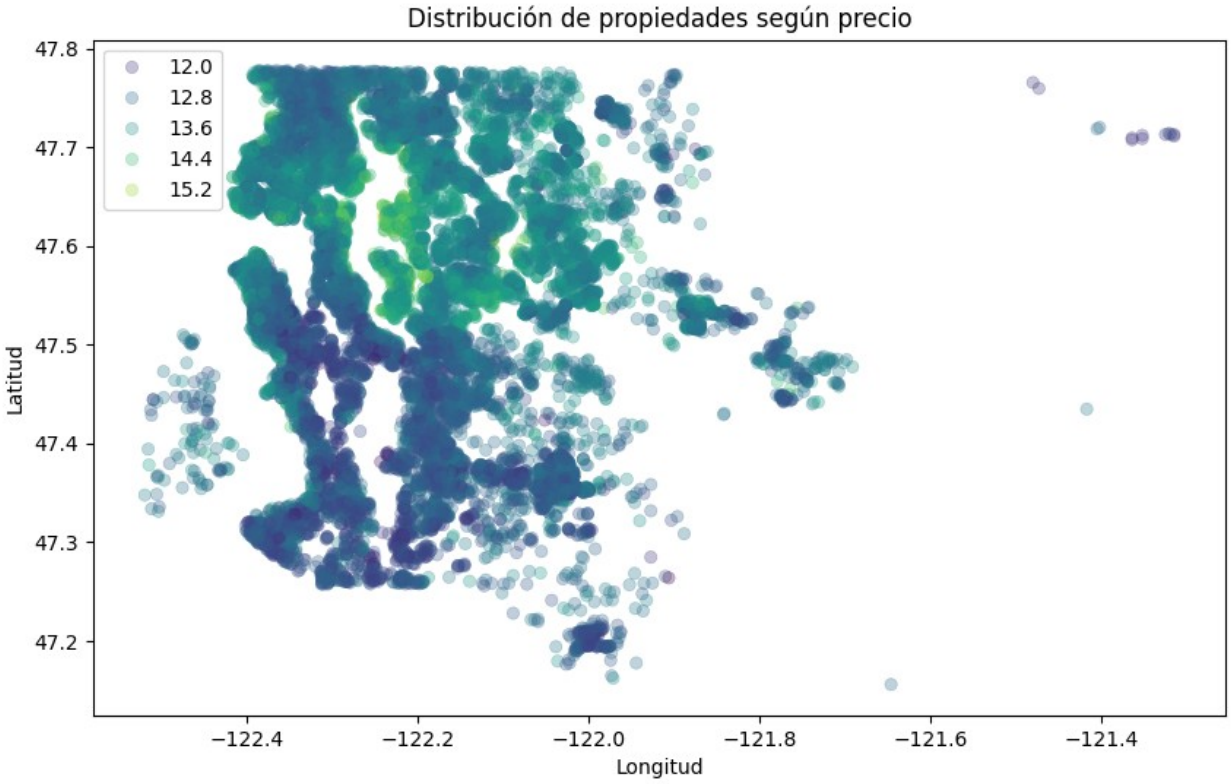
Ahora realizaremos un análisis del resto de features que se encuentran en el dataset. Para realizar el análisis dividiremos las features en dos grupos:

- Numéricas
- Categóricas

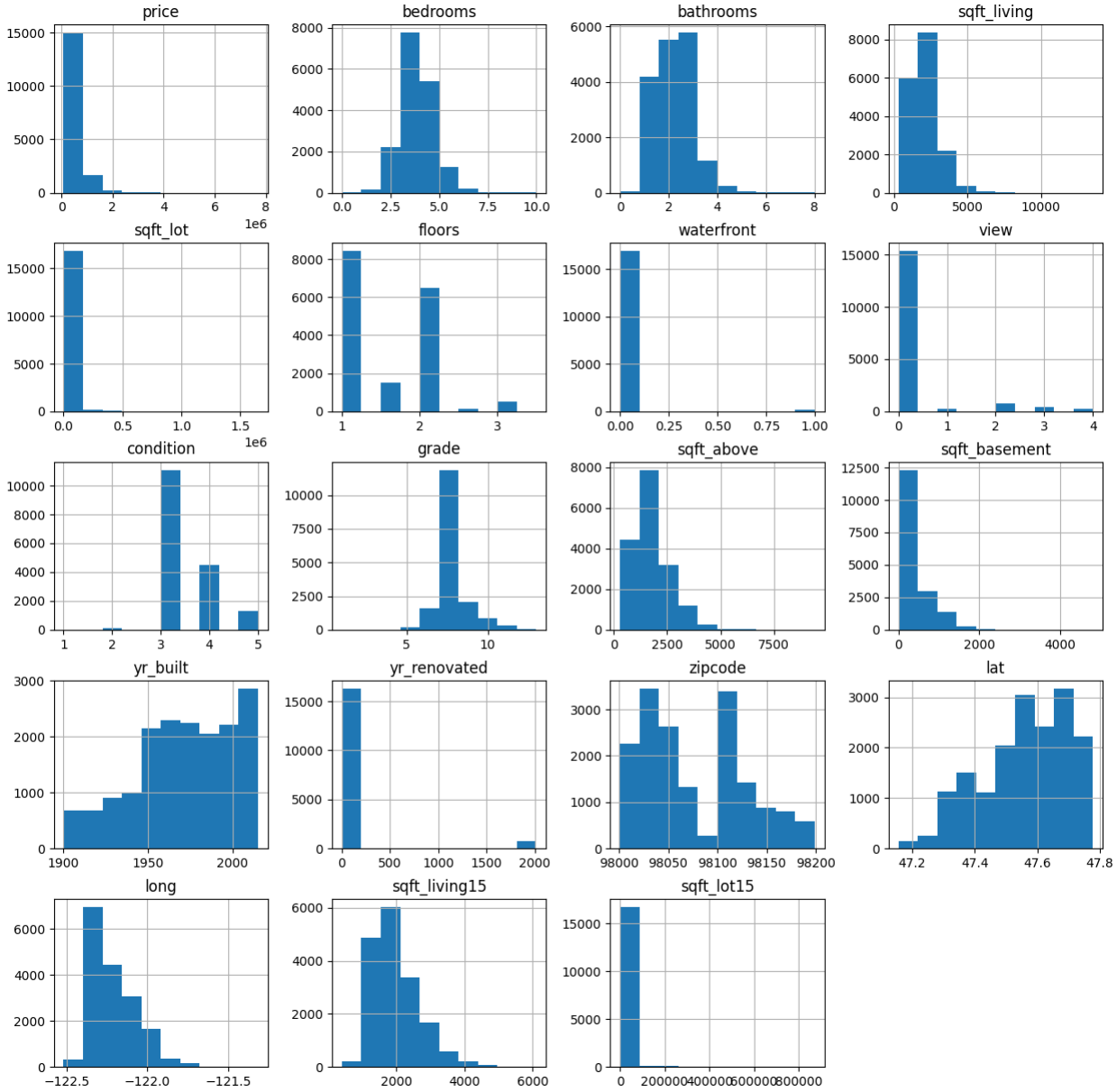
Con los datos de `lat` y `long` podemos representar las cosas de manera geográfica en un gráfico teniendo el cuenta el precio de las viviendas. Para esta representación gráfica los datos han sido transformados logarítmicamente para que sean más fáciles de representar.

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=train_set['long'], y=train_set['lat'], alpha=0.3,
edgecolor=None, legend=True, hue = np.log(train_set[target]),
palette='viridis')

plt.xlabel("Longitud")
plt.ylabel("Latitud")
```

```
plt.title("Distribución de propiedades según precio")
plt.legend()
plt.show()
```




```
train_set.hist(figsize = (15,15));
```



La feature `yr_renovated` la consideraremos como categórica ya que el 95% de las viviendas no han sido reformadas, por lo que podremos convertirla en una categórica binaria.

```
features = train_set.columns.to_list()
features.remove(target)

cat_features =
['bedrooms','bathrooms','floors','waterfront','view','condition','grad
e','yr_renovated']
num_features = [col for col in train_set.columns if col not in
cat_features]

num_features
```

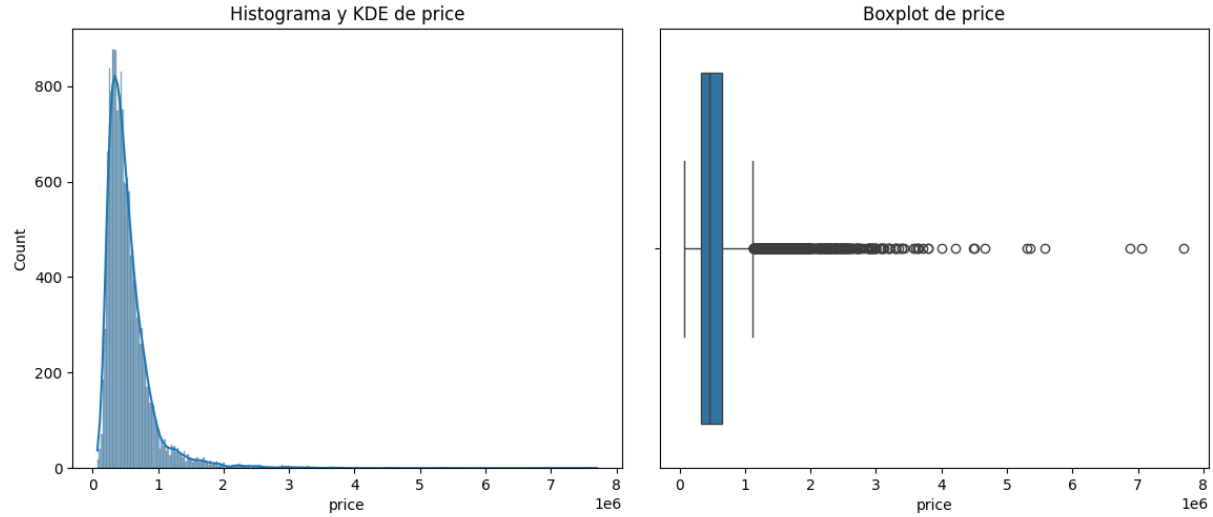
```
['price',
 'sqft_living',
 'sqft_lot',
 'sqft_above',
 'sqft_basement',
 'yr_built',
 'zipcode',
 'lat',
 'long',
 'sqft_living15',
 'sqft_lot15']
```

Análisis de las **features numéricas**

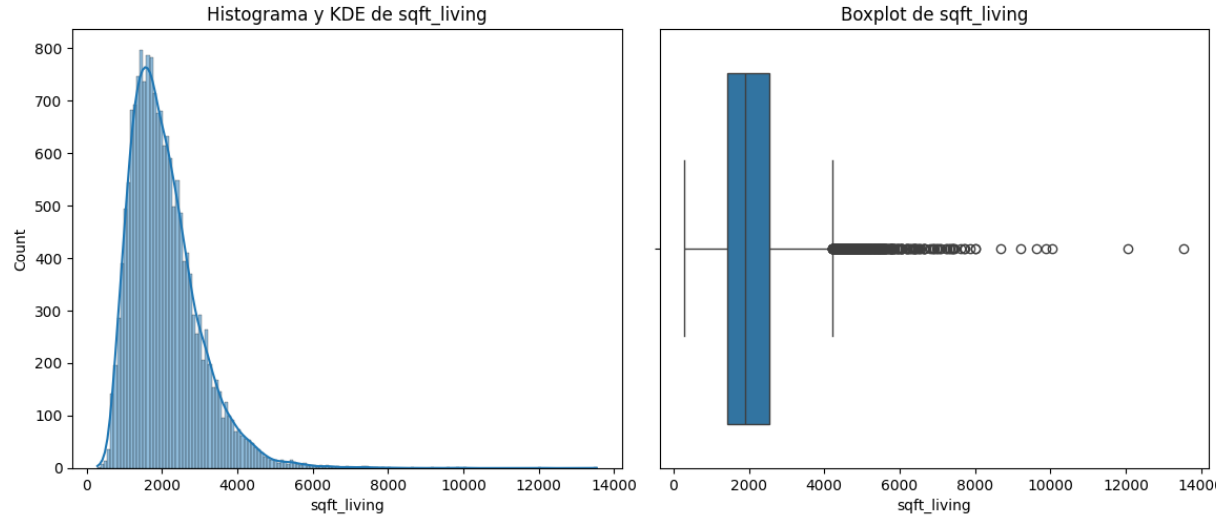
Primero realizaremos un análisis EDA de las features numéricas para saber cómo son sus distribuciones. Para ello representaremos los datos de manera visual a través de boxplots e histogramas de densidad.

```
bt.plot_combined_graphs(df = train_set, columns=num_features,
whisker_width=1.5)
```

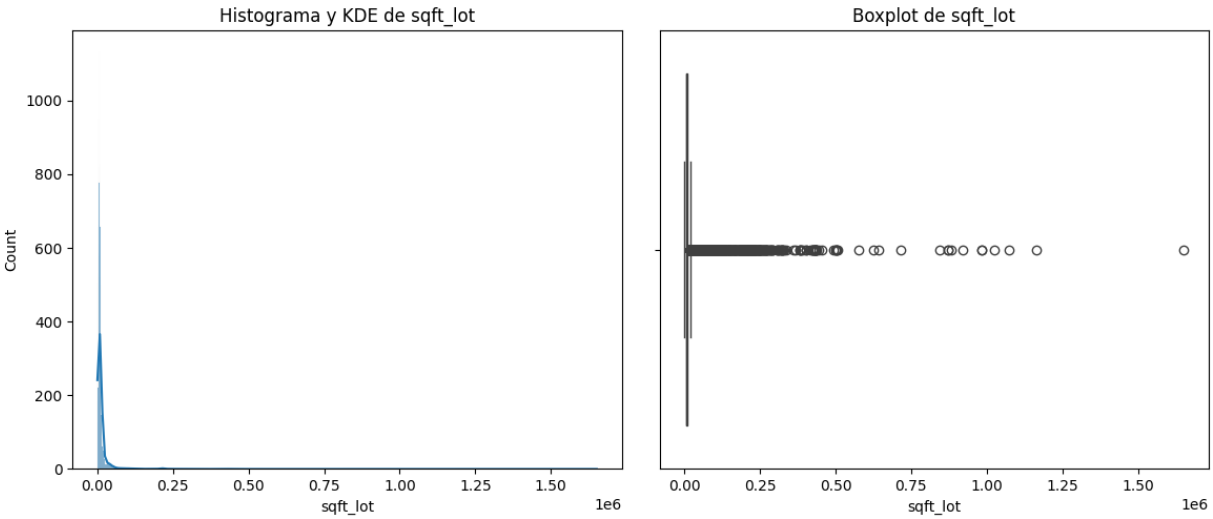
```
(11, 2)
```


Histograma y KDE de price
Count
price
1e6

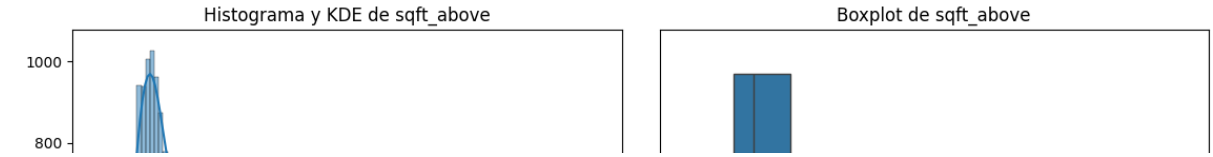
Boxplot de price
price
1e6


Histograma y KDE de sqft_living
Count
sqft_living

Boxplot de sqft_living
sqft_living


Histograma y KDE de sqft_lot
Count
sqft_lot
1e6

Boxplot de sqft_lot
sqft_lot
1e6


Histograma y KDE de sqft_above

Boxplot de sqft_above

En los gráficos anteriores se puede observar que todas las features salvo `yr_built`,`zipcode` y `lat` presentan una gran cantidad de valores **outliers**. No será necesario aplicar transformaciones a estas variables con outliers ya que los algoritmos basados en árboles saben como gestionarlos. Tampoco aplicaremos una transformación a la feature `long` ya que es una medida geográfica, al igual que lat.

Correlación features numéricas

Realizaremos esta correlación a través de la `correlación de Pearson`

```
corr = train_set[num_features].corr()
corr[target].sort_values(ascending = False)
```

```
price            1.000
sqft_living      0.705
sqft_above       0.609
sqft_living15    0.586
sqft_basement    0.333
lat              0.303
sqft_lot         0.090
sqft_lot15       0.082
yr_built         0.054
long             0.018
zipcode         -0.057
Name: price, dtype: float64
```

La mayoría de las features presentan una buena correlación con el precio. Para quedarnos con las mejores estableceremos un criterio que consistirá en tener un mínimo de correlación para ser considerada como relevante para el modelo.

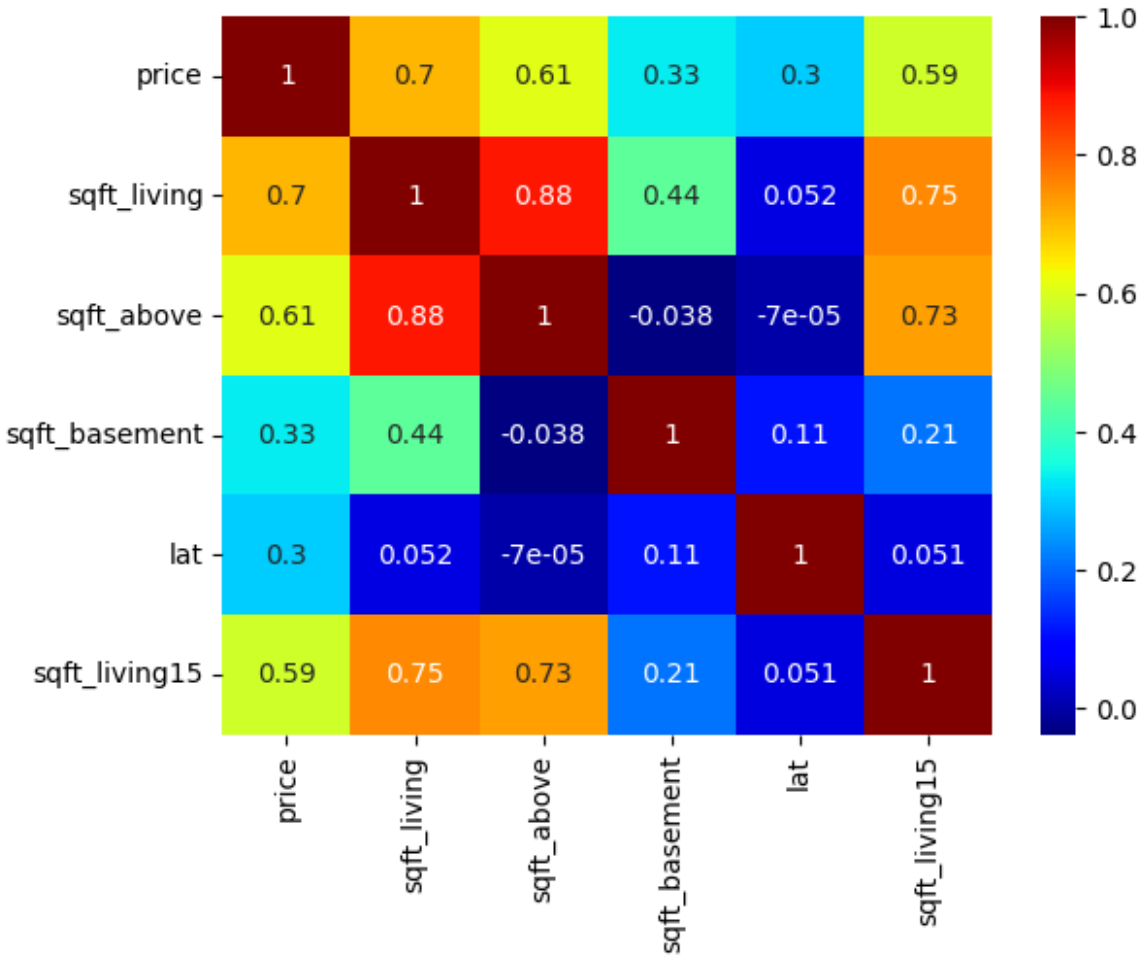
```
pearson_num_features = []
criterio = 0.15
for col in num_features:
    if corr[col][target] > criterio:
        pearson_num_features.append(col)

pearson_num_features
```

```
['price', 'sqft_living', 'sqft_above', 'sqft_basement', 'lat',
'sqft_living15']
```

Colinealidad

Una vez escogidas estas features, analizaremos si estas tienen algún tipo de colinealidad entre ellas. Es importante saber si hay una fuerte correlación entre features independientes ya que pueden perjudicar al modelo. Para saber la correlación entre ellas de manera visual usaremos un **heatmap**.

```
sns.heatmap(data = corr.loc[pearson_num_features,
pearson_num_features], annot=True, cmap = 'jet');
```

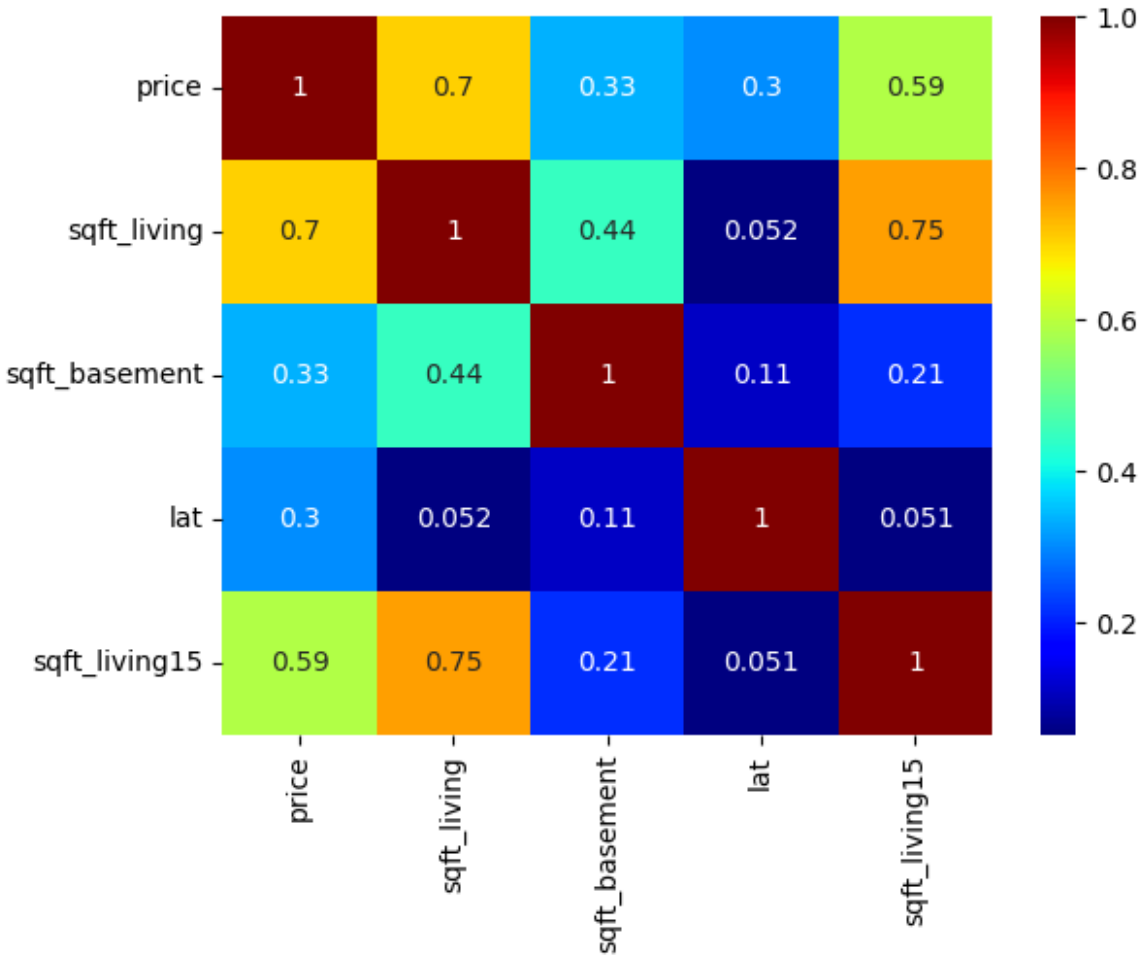


Estas son las colinealidades que encontramos:

- `sqft_above` tiene colinealidad con: sqft_living (88%), sqft_living15 (73%).
- `sqft_living15` tiene colinealidad con: sqft_living (75%).

Eliminaremos de la lista `sqft_above` ya que es la que mayor colinealidad presenta.

```
pearson_num_features.remove('sqft_above')

sns.heatmap(data = corr.loc[pearson_num_features,
pearson_num_features], annot=True, cmap = 'jet');
```



Estas son las features numéricas que usaremos para nuestro modelo.

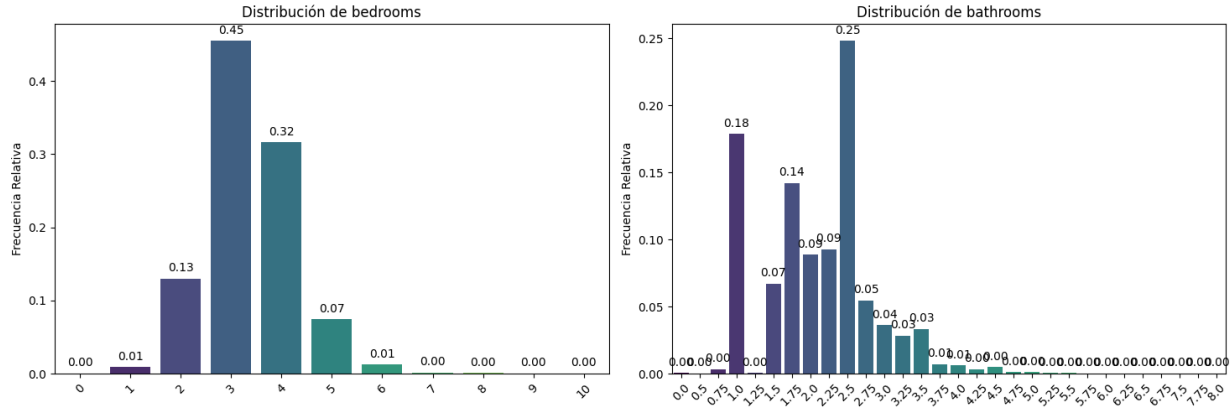
```
pearson_num_features.remove(target)

pearson_num_features
```

```
['sqft_living', 'sqft_basement', 'lat', 'sqft_living15']
```

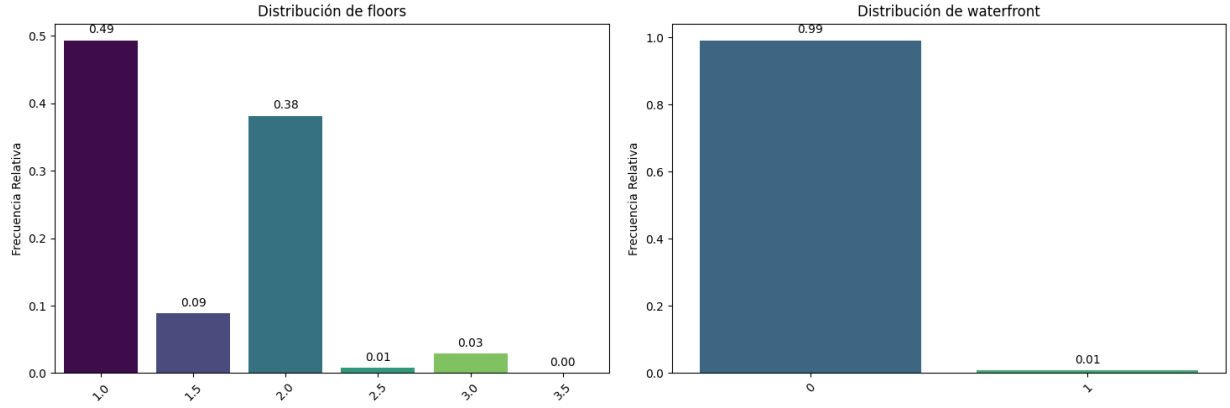
Análisis de las `features categóricas`

Una vez hemos analizado y seleccionado las features numéricas, pasaremos a realizar el proceso de análisis para las features categóricas del dataset. Para ello usaremos visualizaciones distintas y correlaciones diferentes a las usadas con las numéricas.

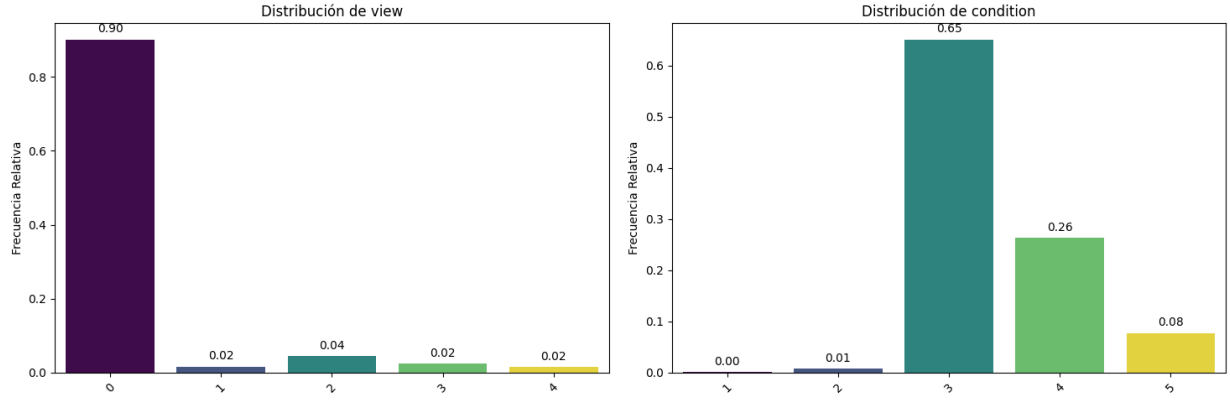
```
bt.pinta_distribucion_categoricas(df = train_set,
columnas_categoricas=cat_features, relativa=True,
mostrar_valores=True)
```


Distribución de bedrooms
Frecuencia Relativa
0.00 0.01 0.13 0.45 0.32 0.07 0.01 0.00 0.00 0.00 0.00
0 1 2 3 4 5 6 7 8 9 10

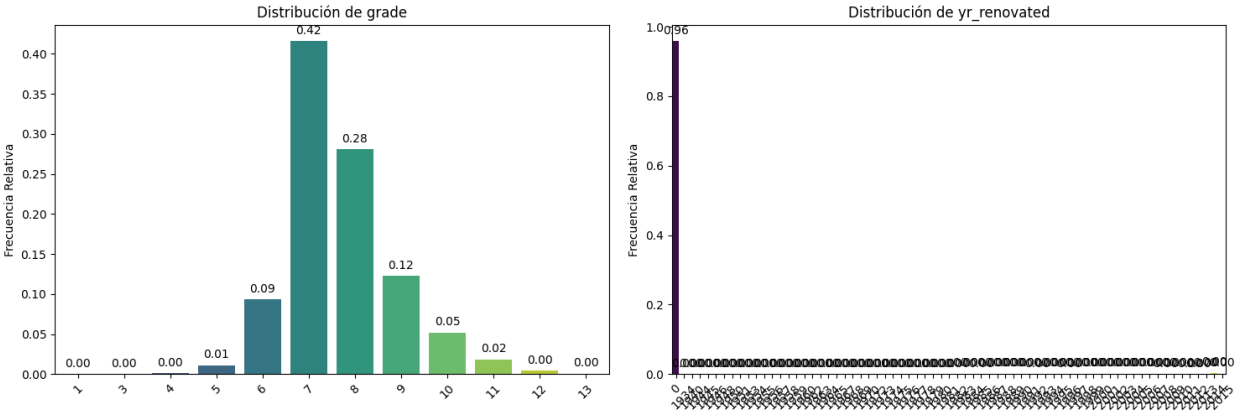
Distribución de bathrooms
Frecuencia Relativa
0.00 0.00 0.00 0.18 0.00 0.07 0.14 0.09 0.09 0.25 0.05 0.04 0.03 0.03
0.0 0.5 0.75 1.0 1.25 1.5 1.75 2.0 2.25 2.5 2.75 3.0 3.25 3.5 3.75 4.0 4.25 4.5 4.75 5.0 5.25 5.5 5.75 6.0 6.25 6.5 6.75 7.5 7.75 8.0


Distribución de floors
Frecuencia Relativa
0.49 0.09 0.38 0.01 0.03 0.00
1.0 1.5 2.0 2.5 3.0 3.5

Distribución de waterfront
Frecuencia Relativa
0.99 0.01
0 1


Distribución de view
Frecuencia Relativa
0.90 0.02 0.04 0.02 0.02
0 1 2 3 4

Distribución de condition
Frecuencia Relativa
0.00 0.01 0.65 0.26 0.08
1 2 3 4 5


Distribución de grade
Frecuencia Relativa
0.00 0.00 0.00 0.01 0.09 0.42 0.28 0.12 0.05 0.02 0.00 0.00
1 3 4 5 6 7 8 9 10 11 12 13

Distribución de yr_renovated
Frecuencia Relativa
0.96

En cada variable categórica se puede observar que hay un grupo dominante que concentra la mayoría de los valores.

- `waterfront` = 0 (no tiene vistas al mar) concentra el 99% de los datos.
- `yr_renovated` = 0 (no ha sido renovada) concentra el 96% de los datos. Esta variable podría ser convertida a binaria: Ha sido reformada (1) o no (0).
- `view` = 0 (no son buenas vistas) concentra el 90% de los datos.
- `condition` = 3 concentra el 65% de los datos.
- `grade` = 7 concentra el 42% de los datos.

Correlación de las features

Una vez visualizada y analizada la distribución de las features, veamos la correlación que tienen con la variable target (price). Para saber la correlación de las variables categóricas con la target, usaremos la correlación `ANOVA`

```
from scipy.stats import f_oneway

anova_results = {}
for col in cat_features:
    grupos = [train_set[target][train_set[col] == valor] for valor in
train_set[col].unique()]
    stats, p_value = f_oneway(*grupos)
    anova_results[col] = p_value

#Para verlo en formato Dataframe
# anova_df = pd.DataFrame(list(anova_results.items()), columns =
["Variable", "p_value"])
anova_results
```

```
{'bedrooms': 0.0,
 'bathrooms': 0.0,
 'floors': 7.6218201057e-314,
 'waterfront': 6.27440011047515e-262,
 'view': 0.0,
 'condition': 1.8764452127159177e-18,
 'grade': 0.0,
 'yr_renovated': 1.1694948220099811e-73}
```

Todas las features categóricas tienen un p_value inferior a 0.05, por lo que podemos rechazar la hipótesis nula y aceptar que estas features tiene una relación estadística significativa con la target. Las usaremos en el modelo.

En cuanto a la feature `yr_renovated` crearemos una nueva feature llamada `reformada` tanto para train set como test set que guardará los datos de manera binaria: Si la vivienda ha sido reformda (1) o no (0).

```
"""Cambiar la columna 'yr_renovated' a una binaria: Ha sido reformada
(1) o no (0)"""
```

```
"""Para train set"""
train_set['reformada'] = (train_set['yr_renovated'] != 0).astype(int)

"""Para test set"""
test_set['reformada'] = (test_set['yr_renovated'] != 0).astype(int)

cat_features.append('reformada')
cat_features.remove('yr_renovated')
```

También transformaremos la feature `bathroom` en una menor cantidad de grupos para que sea más fácil de entender para el modelo

```
"""Convertir la feature 'bathroom' en una categórica de 4 grupos"""
"""Para train set"""
train_set.loc[train_set['bathrooms'] <= 1.25, 'bathrooms_cat'] = 0  # Pocos baños
train_set.loc[(train_set['bathrooms'] > 1.25) & (train_set['bathrooms'] <= 2.5), 'bathrooms_cat'] = 1  # Estándar
train_set.loc[(train_set['bathrooms'] > 2.5) & (train_set['bathrooms'] <= 3.75), 'bathrooms_cat'] = 2  # Grandes
train_set.loc[train_set['bathrooms'] > 3.75, 'bathrooms_cat'] = 3  # Lujo

"""Para test set"""
test_set.loc[test_set['bathrooms'] <= 1.25, 'bathrooms_cat'] = 0  # Pocos baños
test_set.loc[(test_set['bathrooms'] > 1.25) & (test_set['bathrooms'] <= 2.5), 'bathrooms_cat'] = 1  # Estándar
test_set.loc[(test_set['bathrooms'] > 2.5) & (test_set['bathrooms'] <= 3.75), 'bathrooms_cat'] = 2  # Grandes
test_set.loc[test_set['bathrooms'] > 3.75, 'bathrooms_cat'] = 3  # Lujo

cat_features.append('bathrooms_cat')

cat_features.remove('bathrooms')
```

Unimos las features seleccionadas en la variable `selected_features` como las mejores en base a la correlación que tienen con el precio tanto numéricas como categóricas

```
selected_features = pearson_num_features + cat_features
```

Creación del modelo

Una vez hemos analizado y seleccionado todas las features para nuestro modelo, pasaremos con la creación del modelo predictor de precios. Para este proceso realizaremos una **cross-validation** entre 3 modelos diferentes y nos quedaremos con el que mejor métricas obtenga. Nuestra métrica principal será el **RMSE**

```
"""División del train set"""
x_train = train_set.drop(columns = target, axis = 1)
y_train = train_set[target]

"""División del test set"""
x_test = test_set.drop(columns = target, axis = 1)
y_test = test_set[target]
```

Comparación de modelos

Para obtener el mejor modelo posible usaremos 4 modelos distintos:

- Random Forest Regressor
- XGBoost regressor
- CatBoost regressor

```
rf_reg = RandomForestRegressor(random_state=42, max_depth = 10)
xgb_reg = XGBRegressor(random_state = 42, max_depth = 10)
cat_reg = CatBoostRegressor(random_state=42, max_depth=10, verbose=0)
```

Modelos con todas las features

Primero realizaremos el cross-validation con todas las features del dataset.

```
original_features = train_set.columns.to_list()

original_features.remove(target)
original_features.remove('reformada')

original_features.remove('bathrooms_cat')

original_features
```

```
['bedrooms',
 'bathrooms',
 'sqft_living',
 'sqft_lot',
 'floors',
 'waterfront',
 'view',
 'condition',
 'grade',
 'sqft_above',
 'sqft_basement',
 'yr_built',
 'yr_renovated',
 'zipcode',
 'lat',
 'long',
```

```
 'sqft_living15',
 'sqft_lot15']

models = [('RF_reg', rf_reg),
          ('XGB_reg', xgb_reg),
          ('Catboost_reg', cat_reg)]

results = []

for name, model in models:
    print(f"Evaluando el modelo: {name}")

    x_train_model = x_train[original_features]
    x_test_model = x_test[original_features]

    cv_rmse = np.sqrt(-cross_val_score(model, x_train_model, y_train,
cv=5, scoring='neg_mean_squared_error'))

    model.fit(x_train_model, y_train)

    y_pred = model.predict(x_test_model)

    # Cálculo de MAE
    mae = mean_absolute_error(y_test, y_pred)

    # Cálculo de MAPE
    mape = np.mean(np.abs((y_test - y_pred) / y_test))  # Porcentaje
de error medio

    results.append({
        'Model': name,
        'RMSE (Cross-Val)': np.mean(cv_rmse),
        'MAPE': mape,
        'MAE': mae
    })

results_df = pd.DataFrame(results)
print(results_df)
```

```
Evaluando el modelo: RF_reg
Evaluando el modelo: XGB_reg
Evaluando el modelo: Catboost_reg
          Model  RMSE (Cross-Val)  MAPE       MAE
0        RF_reg        138332.690 0.140 72720.786
1       XGB_reg        139517.530 0.127 67816.722
2  Catboost_reg        125092.499 0.116 60225.260
```

Modelos con las features seleccionadas

Ahora realizaremos el mismo proceso con las features seleccionadas.

```
selected_features
```

```
['sqft_living',
 'sqft_basement',
 'lat',
 'sqft_living15',
 'bedrooms',
 'floors',
 'waterfront',
 'view',
 'condition',
 'grade',
 'reformada',
 'bathrooms_cat']
```

```
models = [('RF_reg', rf_reg),
          ('XGB_reg', xgb_reg),
          ('Catboost_reg', cat_reg)]

results = []

for name, model in models:
    print(f"Evaluando el modelo: {name}")

    x_train_model = x_train[selected_features]
    x_test_model = x_test[selected_features]

    cv_rmse = np.sqrt(-cross_val_score(model, x_train_model, y_train,
cv=5, scoring='neg_mean_squared_error'))

    model.fit(x_train_model, y_train)

    y_pred = model.predict(x_test_model)

    # Cálculo de MAE
    mae = mean_absolute_error(y_test, y_pred)

    # Cálculo de MAPE
    mape = np.mean(np.abs((y_test - y_pred) / y_test))  # Porcentaje
de error medio

    results.append({
        'Model': name,
        'RMSE (Cross-Val)': np.mean(cv_rmse),
        'MAPE': mape,
        'MAE': mae
    })

results_df = pd.DataFrame(results)
print(results_df)
```

```
Evaluando el modelo: RF_reg
Evaluando el modelo: XGB_reg
Evaluando el modelo: Catboost_reg
          Model  RMSE (Cross-Val)  MAPE       MAE
0        RF_reg        162706.946 0.162 86516.556
1       XGB_reg        166916.772 0.159 85457.924
2  Catboost_reg        156864.142 0.150 80040.648
```

Obtenemos mejores resultados en todos los modelos usando todas las features del dataset. El mejor modelo es el `CatBoost regressor`, con un RMSE de 125,092.5 dólares.

Mejora del mejor modelo con **GridSearch**

Una vez tenemos el mejor modelo, aplicaremos un `GridSearch` en el que probaremos distintos parámetros para obtener el mejor resultado posible y así ver si de esta manera mejora.

Grid search con todas las features

```
catboost_model = CatBoostRegressor(random_state=42, verbose=0)

param_grid = {
    'iterations': [200, 500, 1000],  # Número de árboles en el modelo
    'learning_rate': [0.01, 0.05, 0.1],  # Tasa de aprendizaje
    'depth': [4, 6, 8, 10],  # Profundidad de los árboles
}

grid_model = GridSearchCV(estimator=catboost_model,
                          param_grid=param_grid,
                          cv = 5,
                          scoring = 'neg_mean_squared_error')

grid_model.fit(x_train[original_features], y_train)
```

```
GridSearchCV(cv=5,
             estimator=<catboost.core.CatBoostRegressor object at
0x000001EC3B72F6E0>,
             param_grid={'depth': [4, 6, 8, 10], 'iterations': [200,
500, 1000],
                         'learning_rate': [0.01, 0.05, 0.1]},
             scoring='neg_mean_squared_error')
```

```
print("Best parameters:", grid_model.best_params_)
print("Best RMSE:", (-grid_model.best_score_)**0.5)
```

```
Best parameters: {'depth': 6, 'iterations': 1000, 'learning_rate':
0.1}
Best RMSE: 119045.6259385073
```

Predicciones contra el set de test

```
y_pred = grid_model.predict(x_test[original_features])

rmse = root_mean_squared_error(y_test, y_pred)

mae = mean_absolute_error(y_test, y_pred)

mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
print(f"MAPE: {mape}%")
```

```
RMSE: 102675.58008651157
MAE: 60623.42551526358
MAPE: 11.82723183708813%
```

La única métrica que ha mejorado a sido el RMSE, siendo este con el gridsearch de 102,675.58 dólares.

Grabación del modelo

Una vez hemos conseguido el mejor modelo, lo grabaremos en la carpeta `models` para poder usarlo de nuevo si es necesario. Para guardar el modelo usaremos `joblib`.

```
import joblib

# Guardar el modelo
joblib.dump(grid_model.best_estimator_, 'models/catboost_model.pkl')

# Cargar el modelo más tarde
loaded_model = joblib.load('models/catboost_model.pkl')

# Verificar que el modelo cargado funciona correctamente
y_pred_loaded_model = loaded_model.predict(x_test[original_features])
print(f"Predicciones con el modelo cargado:
{y_pred_loaded_model[:5]}")  # Muestra las primeras 5 predicciones
```

```
Predicciones con el modelo cargado: [592986.60890492 285861.78524006
333754.99426148 401826.3219303
 558707.12034682]
```