1-modulo-1-eda

April 20, 2024

1 Curso: Análisis Exploratorio de Datos (EDA)

Bienvenido al curso de "*Análisis Esploratorio de Datos (EDA)*". Durante esta experiencia educativa, nos sumergiremos en el uso de técnicas de la estadística descriptiva y la visualización de datos que nos permitirán evaluar la calidad y entender el comportamiento de un conjunto de datos. Desarrollaremos procesos preliminares a la construcicón de conclusiones tales como descripciones generales de los datos, limpieza de datos y diagnóstico del dataset. Posteriormente intentaremos extraer los comportamientos más releventes de los datos mediante la descripción estadística y visual de cada variable tanto numérica como categórica, para posteriormente determinar las tendencias y relaciones entre las variables de nuestros datos.

---

2 Contenidos:

1. **Importación de datos**
2. **Conocimientos Preliminares**

- Estadística descriptiva básica
- Pandas y el manejo de datos

3. **Exploración Inicial**

- Exploración inicial de los datos
- Estadística descriptiva de los datos

4. **Análisis Univariado**

- Variables Numéricas
- Variables Categóricas

5. **Análisis Bivariado**

6.

2.1 Análisis por Agrupación

3 Dependencias

[ ]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
%matplotlib inline
```

4 1. Importación de datos

El diccionario de datos, también conocido como "Data Dictionary" en inglés, es una herramienta fundamental en el campo de la Ciencia de Datos y el Análisis Exploratorio de Datos (EDA). Se trata de un documento que proporciona información detallada y descriptiva sobre los datos utilizados en un proyecto o conjunto de datos. El diccionario de datos es una herramienta esencial que contribuye significativamente a la transparencia, calidad y eficiencia en el manejo y análisis de datos en proyectos.

- **Nombres de variables o columnas:** Enumera todas las variables o campos presentes en el conjunto de datos.
- **Tipo de datos:** Indica el tipo de información que contiene cada variable, como numérico, categórico, fecha, etc.
- **Descripción de variables:** Proporciona una explicación detallada de lo que representa cada variable. Esto puede incluir su significado, unidades de medida y cualquier otra información relevante.
- **Rangos y valores permitidos:** Especifica los rangos de valores que puede tener cada variable y cualquier restricción o regla asociada.
- **Origen de datos:** Indica la fuente de donde provienen los datos, cómo se recopilaron y cualquier proceso de transformación aplicado.
- **Formato de representación:** Define cómo se almacena y muestra la información, como el formato de fecha, el número de decimales, etc.

La importancia de conocer el diccionario de datos radica en varios aspectos:

1. **Comprensión del contexto:** Facilita la comprensión del contexto y el significado de cada variable, permitiendo a los analistas y científicos de datos interpretar adecuadamente los resultados.
2. **Consistencia y calidad de datos:** Ayuda a garantizar la consistencia y calidad de los datos al proporcionar información sobre los formatos y rangos permitidos, lo que puede ser crucial para identificar posibles problemas o anomalías en los datos.
3. **Colaboración:** Facilita la colaboración entre diferentes equipos o personas involucradas en un proyecto al proporcionar una referencia común y comprensible sobre la estructura y contenido de los datos.

4. **Documentación:** Sirve como una valiosa documentación para futuros análisis y proyectos, permitiendo a otros entender rápidamente la naturaleza de los datos sin tener que explorar el conjunto de datos de forma exhaustiva.

Para ilustrar un proceso de EDA (Exploratory Data Analysis), a continuación emplearemos el 'housing-dataset', un conjunto de datos genérico que especifica el valor del precio de inmuebles de acuerdo a diferentes características. El diccionario de este dataset es el siguiente, donde se encuentran 13 variables y en total 545 observaciones.

Variable	Descripción	Tipología
price	Precio (USD)	Cuantitativa continua
area	Área del predio (pies cuadrados)	Cuantitativa continua
bedrooms	Cantidad de habitaciones	Cuantitativa discreta
bathrooms	Cantidad de baños	Cuantitativa discreta
stories	Cantidad de pisos-niveles	Cuantitativa discreta
mainroad	Ubicación sobre una calle principal	Cualitativa binaria
guestroom	Cuarto para invitados	Cualitativa binaria
basement	Sotano	Cualitativa binaria
hotwaterheating	Calentador de agua	Cualitativa binaria
airconditioning	Aire acondicionado	Cualitativa binaria
parking	Cantidad de parqueaderos	Cuantitativa discreta
prefarea	Barrio o zona privilegiada de la ciudad	Cualitativa binaria
furnishing status	Estado de amoblamiento	Cualitativa ordinal

[ ]:
```
# Importación de datos
path = 'https://raw.githubusercontent.com/AsorKy/Datasets/main/Housing.csv'
df = pd.read_csv(path)
```

El EDA puede entenderse a partir de momentos generales de acuerdo al tipo de preguntas que se desea contestar y los procedimientos implicados:

1. Preguntas preliminares:

- ¿Que deseamos encontrar?
- ¿Que deseamos saber de los datos?
- ¿Cuál es la razon del análisis?

2. Exploración inicial:

- Determinar el número de observaciones y variables del conjunto de datos.
- Identificar los tipos de variables.
- Estadísticas descriptivas básicas para obtener una visión general.
- Determinar si todas las observaciones son necesarias.
- Determinar si todas las variables son necesarias.

3. Categorización de variables:

- Identificar las variables categóricas.
- Identificar las variables continuas.
- Determinar como explorar cada variable dependiendo de su categoría.

4. Manejo de datos faltantes:

- Identificación y manejo de valores nulos o faltantes.
- Evaluación del impacto de los datos faltantes en el análisis.

5. Limpieza de datos:

- Tratamiento de datos atípicos o anómalos.
- Corrección de errores en los datos.

6. Transformación de variables:

- Transformación de variables para ajustarse a requisitos de modelos o mejorar la distribución.
- Creación de nuevas variables si es necesario.

7. Análisis Univariado:

- Análisis de una variable a la vez para comprender su distribución y estadísticas descriptivas.
- Uso de gráficos como histogramas, box plots, etc.

8. Análisis Bivariado:

- Exploración de relaciones entre dos variables.
- Uso de gráficos de dispersión, matrices de correlación, etc.

9. Análisis Multivariado:

- Exploración de interacciones entre tres o más variables.
- Uso de técnicas como análisis de componentes principales (PCA), gráficos 3D, etc.

10. Análisis Temporal o Espacial:

- Si es aplicable, análisis de patrones a lo largo del tiempo o en diferentes ubicaciones geográficas.

11. Conclusion y Resumen:

- Síntesis de hallazgos clave.
- Identificación de posibles áreas de interés para un análisis más detallado.

12. Preparación para Modelado:

- Selección de variables relevantes.
- Normalización o estandarización si es necesario.
- División del conjunto de datos en conjuntos de entrenamiento y prueba.

5 2. Conocimientos preliminares

5.0.1 2.1. Estadística descriptiva básica

Los estadísticos de tendencia central son medidas que representan el valor central o típico de un conjunto de datos. Son útiles para resumir la distribución de los datos y proporcionar una idea general de dónde se concentran los datos. Los tres estadísticos de tendencia central más comunes son la media, la mediana y la moda.

1. **Media o promedio**: La media es el valor obtenido al sumar todos los valores de un conjunto de datos y luego dividir esa suma por el número total de valores. Se calcula con la siguiente ecuación:

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n}$$

2. **Mediana**: La mediana es el valor que separa el conjunto de datos ordenados en dos partes iguales: la mitad de los datos están por encima de la mediana y la otra mitad están por debajo. Si el número de datos es impar, la mediana es simplemente el valor en el centro del conjunto de datos. Si el número de datos es par, la mediana es el promedio de los dos valores centrales.

$$\text{Mediana} = \begin{cases} x_{\frac{n+1}{2}} & \text{si } n \text{ es impar} \\ \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2} & \text{si } n \text{ es par} \end{cases}$$

3. **Moda**: La moda es el valor que aparece con mayor frecuencia en un conjunto de datos. Puede haber más de una moda (en cuyo caso los datos son multimodales) o puede no haber ninguna moda (cuando todos los valores tienen la misma frecuencia).

$$\text{Moda} = \arg\max_x f(x)$$

Las medidas de dispersión son estadísticas que proporcionan información sobre la variabilidad o dispersión de un conjunto de datos. Son útiles para comprender la distribución de los datos y la distancia entre los valores individuales y el centro del conjunto de datos. Las principales medidas de dispersión incluyen la varianza, la desviación estándar, el rango y el rango intercuartílico.

1. **Varianza**: La varianza mide la dispersión de los datos alrededor de la media. Es la media de los cuadrados de las desviaciones de cada valor respecto a la media del conjunto de datos.

$$\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n}$$

2. **Desviación estándar**: La desviación estándar es simplemente la raíz cuadrada de la varianza. Proporciona una medida de dispersión en las mismas unidades que los datos originales, lo que la hace más interpretable.

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2}$$

3. **Rango**:El rango es la diferencia entre el valor máximo y el valor mínimo en un conjunto de datos. Es una medida simple de dispersión pero puede ser afectada por valores atípicos.

$$\text{Rango} = x_{\text{máx}} - x_{\text{mín}}$$

4. **Rango Intercuartílico (IQR)**: El rango intercuartílico es la diferencia entre el tercer cuartil (Q3) y el primer cuartil (Q1). Es una medida de dispersión más robusta que el rango, ya que no se ve afectada por valores extremos.

5. **Cuartiles**: Los cuartiles son medidas estadísticas que dividen un conjunto de datos ordenados en cuatro partes iguales. Se utilizan para describir la distribución de los datos y proporcionar información sobre la dispersión y la posición de los valores dentro del conjunto de datos. Los tres cuartiles principales son:

$$\text{RIC} = Q3 - Q1$$

$$Q1 = x_{\frac{n+1}{4}}$$

$$Q2 = \begin{cases} x_{\frac{n+1}{2}} & \text{si } n \text{ es impar} \\ \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2} & \text{si } n \text{ es par} \end{cases}$$

$$Q3 = x_{\frac{3(n+1)}{4}}$$

El coeficiente de variación (CV) es una medida estadística que se utiliza para cuantificar la variabilidad relativa de una variable en relación con su media. Se calcula como la desviación estándar de la variable dividida por su media, y se expresa como un porcentaje. La fórmula para el coeficiente de variación (CV) es la siguiente:

$$CV(X) = \frac{std(X) * 100}{mean(X)}$$

- $CV(X)$ es el coeficiente de variación de la variable X
- $std(X)$ es la desviación estándar de la variable X
- $mean(X)$ es la media de la variable X

El coeficiente de variación es útil cuando se desea comparar la variabilidad de diferentes conjuntos de datos que pueden tener unidades de medida diferentes o escalas distintas. Por ejemplo, si se están comparando dos distribuciones de ingresos en diferentes países, el CV permite evaluar cuál de las distribuciones es más variable en relación con su media, independientemente de las unidades de moneda utilizadas en cada país.

Un coeficiente de variación bajo indica que la variabilidad de la variable es relativamente pequeña en relación con su media, mientras que un coeficiente de variación alto indica una mayor variabilidad relativa en comparación con la media. El CV se expresa como un porcentaje para facilitar su interpretación y comparación entre diferentes variables o conjuntos de datos.

5.0.2 2.2. Pandas y el manejo de datos

[ ]:
```
# Veamos cuántos valores posibles toma la variable furnishingstatus
df_house['furnishingstatus'].nunique()
```

[ ]:
```
3
```

[ ]:
```
# Hagamos un conteo  por categoría de esta variable
df_house['furnishingstatus'].value_counts()
```

[ ]:
```
semi-furnished    227
unfurnished       178
furnished         140
Name: furnishingstatus, dtype: int64
```

[ ]:
```
# Este objeto se puede operar, por ejemplo para obtener un porcentaje
df_house['bathrooms'].value_counts()*100 / df_house.shape[0]
```

[ ]:
```
1    73.577982
2    24.403670
3     1.834862
4     0.183486
Name: bathrooms, dtype: float64
```

[ ]:
```
df_house['bedrooms'].value_counts(normalize = True, ascending = True)*100
```

[ ]:
```
6     0.366972
1     0.366972
5     1.834862
4    17.431193
2    24.954128
3    55.045872
Name: bedrooms, dtype: float64
```

6 3. Exploración inicial

En esta etapa, nos ocupamos de los siguientes procesos: * Determinar el número de observaciones y variables del conjunto de datos. * Identificar los tipos de variables. * Estadísticas descriptivas básicas para obtener una visión general. * Determinar si todas las observaciones son necesarias. * Determinar si todas las variables son necesarias.

6.0.1 3.1. Exploración inicial de los datos

¿Cual es el tamaño de nuestros datos?

[ ]:
```
# Dimensiones de los datos
df.shape
```

[ ]:
```
(545, 13)
```

¿Cómo lucen nuestros datos?

[ ]:
```
# Vista de las primeras 5 filas
df.head()
```

[ ]:
```
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000  7420         4          2        3      yes        no       no
```

```
1  12250000  8960         4          4        4      yes        no       no
2  12250000  9960         3          2        2      yes        no      yes
3  12215000  7500         4          2        2      yes        no      yes
4  11410000  7420         4          1        2      yes       yes      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no             yes        2      yes        furnished
1              no             yes        3       no        furnished
2              no              no        2      yes   semi-furnished
3              no             yes        3      yes        furnished
4              no             yes        2       no        furnished
```

[ ]:
```
# Vista de las últimas 5 filas
df.tail()
```

[ ]:
```
       price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
540  1820000  3000         2          1        1      yes        no      yes
541  1767150  2400         3          1        1       no        no       no
542  1750000  3620         2          1        1      yes        no       no
543  1750000  2910         3          1        1       no        no       no
544  1750000  3850         3          1        2      yes        no       no

    hotwaterheating airconditioning  parking prefarea furnishingstatus
540              no              no        2       no      unfurnished
541              no              no        0       no   semi-furnished
542              no              no        0       no      unfurnished
543              no              no        0       no        furnished
544              no              no        0       no      unfurnished
```

Para conocer los tipos de variables, podemos usar el método `DataFrame.info()` el cual retorna un resumen de los datos que incluye: * Dimensiones del DataFrame * El nombre de las variables * El conteo de valores no nulos * Tipología de cada variable.

¿Qué tipos de variables tiene nuestro conjunto de datos?

[ ]:
```
# Identificar las variables
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
```

```
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

[ ]:
```
# Cambiar el tipo 'object' de las variables categóricas a 'category'
categories = {
    "mainroad":"category",
    "guestroom":"category",
    "basement":"category",
    "hotwaterheating":"category",
    "airconditioning":"category",
    "prefarea":"category",
    "furnishingstatus":"category"
}
path = 'https://raw.githubusercontent.com/AsorKy/Datasets/main/Housing.csv'
df = pd.read_csv(path,dtype = categories)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    category
 6   guestroom         545 non-null    category
 7   basement          545 non-null    category
 8   hotwaterheating   545 non-null    category
 9   airconditioning   545 non-null    category
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    category
 12  furnishingstatus  545 non-null    category
dtypes: category(7), int64(6)
memory usage: 29.8 KB
```

Podemos observar que existen 6 variables de tipo entero y 7 de tipo categórico; por otra parte, se evidencia que no existen valores nulos.

6.0.2 3.2. Estadística descriptiva de los datos

El método `.describe()` en Pandas es una herramienta útil para obtener estadísticas descriptivas de las columnas numéricas de un DataFrame. Proporciona un resumen conciso que incluye medidas estadísticas clave, lo que facilita la comprensión inicial de la distribución y tendencias en los datos. Dentro de la información proporcionada encontramos: * Conteo * Promedio * Desviación estándar * Valor mínimo y máximo (rango) * Cuartiles - percentiles 25%, 50% (mediana) y 75%

No obstante, los métodos que permiten calcular los estadísticos de tendencia central y dispersión básicos de la estadística descriptiva son:

- `.min()`: Retorna el valor mínimo de la variable.
- `.max()`: Retorna el valor máximo de la variable.
- `.mean()`: Retorna el valor promedio de la variable.
- `.median()`: Retorna el valor de la media de la variable.
- `.mode()`: Retorna la moda de la variable.
- `.quantile()`: Retorna el cuantil especificado o lísta de cuartiles especificados.
- `.std()`: Retorna la desviación estándard de la variable.

[ ]:
```
# Descripción estadística inicial de los datos
df.describe()
```

[ ]:
```
              price          area    bedrooms   bathrooms     stories  \
count  5.450000e+02    545.000000  545.000000  545.000000  545.000000
mean   4.766729e+06   5150.541284    2.965138    1.286239    1.805505
std    1.870440e+06   2170.141023    0.738064    0.502470    0.867492
min    1.750000e+06   1650.000000    1.000000    1.000000    1.000000
25%    3.430000e+06   3600.000000    2.000000    1.000000    1.000000
50%    4.340000e+06   4600.000000    3.000000    1.000000    2.000000
75%    5.740000e+06   6360.000000    3.000000    2.000000    2.000000
max    1.330000e+07  16200.000000    6.000000    4.000000    4.000000

          parking
count  545.000000
mean     0.693578
std      0.861586
min      0.000000
25%      0.000000
50%      0.000000
75%      1.000000
max      3.000000
```

Otra forma de calcular las estadísticas de tendencia central es usar directamente los métodos `.mean()`, `.median()` y `.mode()`.

[ ]:
```
# Medidas de tendencia central del conjunto de datos
mean = df.mean(numeric_only=True)
median = df.median(numeric_only=True)
mode = df.mode(numeric_only=True)
```

```
variables = mean.index

medidas_centrales = pd.DataFrame({
    'Variable': mean.index,
    'Mean': mean.values,
    'Median': median.values,
    'Mode': mode.iloc[0].values  # Tomar solo la primera fila de la moda (puede␣
 ↪haber varias)
})

medidas_centrales
```

[ ]:

	Variable	Mean	Median	Mode
0	price	4.766729e+06	4340000.0	3500000.0
1	area	5.150541e+03	4600.0	6000.0
2	bedrooms	2.965138e+00	3.0	3.0
3	bathrooms	1.286239e+00	1.0	1.0
4	stories	1.805505e+00	2.0	2.0
5	parking	6.935780e-01	0.0	0.0

[ ]:
```
# Pueden existir varias modas
df.mode()
```

[ ]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	3500000	6000.0	3.0	1.0	2.0	yes	no	no	no	no	0.0	no	semi-furnished
1	4200000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

El método `DataFrame.describe()` también permite hacer un resumen para variables categóricas, donde la información provista es: * Conteo * Cantidad de valores únicos * Categoría con mayor frecuencia - moda * Frecuencia de la moda

[ ]:
```
# Descripción estadística inicial de los datos categóricos
# df.describe(include='object')
df.describe(include='category')
```

[ ]:

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea	furnishingstatus
count	545	545	545	545	545	545	
unique	2	2	2	2	2	2	
top	yes	no	no	no	no	no	
freq	468	448	354	520	373	417	

```
count                545
unique                 3
top       semi-furnished
freq                 227
```

Para conocer los nombres de las variables del dataset empleamos `.columns`, en donde si queremos consultar los valores específicos de una columna empleamos la estructura de consulta `df['nombre de la variable´]`.

[ ]:
```
# Consulta de los valores de una variable
columnas = df.columns
print('Las columnas de su dataset son= ', columnas)
print('El tipo de "columnas" es =' , type(columnas) )
print('El tipo de "columnas" es =' , type(columnas.tolist()) )
columnas[0]
```

```
Las columnas de su dataset son=  Index(['price', 'area', 'bedrooms',
'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
El tipo de "columnas" es = <class 'pandas.core.indexes.base.Index'>
El tipo de "columnas" es = <class 'list'>
```

[ ]:
```
'price'
```

Note que cuando usted ejecuta una consulta de las columnas del dataset, el tipo de objeto resultante de la consulta es una serie de pandas.

[ ]:
```
# Consulta simple en pandas
df['price']
```

[ ]:
```
0      13300000
1      12250000
2      12250000
3      12215000
4      11410000
         ...
540     1820000
541     1767150
542     1750000
543     1750000
544     1750000
Name: price, Length: 545, dtype: int64
```

Para extraer los valores y los indices de la serie, debe usar respectivamente `.values` y `.index`.

[ ]:
```
# Valores de una serie
print( df['price'].values[0:5] )

# Indices de una serie
print( list(df['price'].index[0:5]) )
```

```
[13300000 12250000 12250000 12215000 11410000]
[0, 1, 2, 3, 4]
```

Para extraer información a cerca de las columnas con variables categóricas empleamos los métodos del método `.cat()`.

¿Qué categorías componen a una de mis variables categóricas?

[ ]:
```
# Extraer las categorías de una variable categórica
df["furnishingstatus"].cat.categories
```

[ ]:
```
Index(['furnished', 'semi-furnished', 'unfurnished'], dtype='object')
```

A veces, para enriquecer la información que nos proporciona un conjunto de datos o simplemente para interpretarla de una manera más fácil o apropiada, se emplean transformaciones en las variables para mantenerlas, ya sea dentro de una escala o sistema métrico. Por otra parte, es posible extraer información implícita de nuestro objeto de estudio al crear variables nuevas a partir de las variables disponibles. Por ejemplo, a continuación realizaremos un escalamiento de dos variables, calcularemos el precio por área y cambiaremos la escala de las variables 'price' y 'area'.

[ ]:
```
# Creación de una variable dentro del dataset y escalamiento de variables
df_house = df.copy()
df_house["price"] = df_house["price"] / 1e6
df_house["area"] = df_house["area"]*0.092903
df_house["price_per_area"] = df_house["price"]*1e6/df_house["area"]
```

[ ]:
```
df_house.describe()
```

[ ]:
```
              price         area    bedrooms   bathrooms     stories  \
count   545.000000   545.000000  545.000000  545.000000  545.000000
mean      4.766729   478.500737    2.965138    1.286239    1.805505
std       1.870440   201.612611    0.738064    0.502470    0.867492
min       1.750000   153.289950    1.000000    1.000000    1.000000
25%       3.430000   334.450800    2.000000    1.000000    1.000000
50%       4.340000   427.353800    3.000000    1.000000    2.000000
75%       5.740000   590.863080    3.000000    2.000000    2.000000
max      13.300000  1505.028600    6.000000    4.000000    4.000000

           parking  price_per_area
count   545.000000      545.000000
mean      0.693578    10692.087213
std       0.861586     3730.095094
min       0.000000     2910.514731
```

```
25%       0.000000     8023.103348
50%       0.000000    10251.347668
75%       1.000000    12751.099368
max       3.000000    28416.735735
```

[ ]:
```
df_house.describe(include="category")
```

[ ]:
```
       mainroad guestroom basement hotwaterheating airconditioning prefarea  \
count       545       545      545             545             545      545
unique        2         2        2               2               2        2
top         yes        no       no              no              no       no
freq        468       448      354             520             373      417

       furnishingstatus
count               545
unique                3
top      semi-furnished
freq                227
```

Las medidas de dispersión en estadísticas proporcionan información crucial sobre la extensión o variabilidad de un conjunto de datos. En Pandas, estas medidas se pueden calcular fácilmente y desempeñan un papel fundamental en el Análisis Exploratorio de Datos (EDA). Estas métricas son útiles para:

1. Evaluación de variabilidad.
2. Identificación de outliers.

Algunas de las medidas de dispersión comunes incluyen:

- Desviación estándar
- Varianza
- Rango
- Rango intercuartílico (IQR)”

¿Cuál es el rango de nuestras variables?

[ ]:
```
# Rango de los datos
min = df_house.min(numeric_only=True)
max = df_house.max(numeric_only=True)

variables = min.index
rangos = pd.DataFrame({
    'Mínimo': min.values,
    'Máximo': max.values,
    'Rango': max - min
})
rangos
```

[ ]:

	Mínimo	Máximo	Rango
price	1.750000	13.300000	11.550000
area	153.289950	1505.028600	1351.738650
bedrooms	1.000000	6.000000	5.000000
bathrooms	1.000000	4.000000	3.000000
stories	1.000000	4.000000	3.000000
parking	0.000000	3.000000	3.000000
price_per_area	2910.514731	28416.735735	25506.221004

El rango intercuartílico (IQR, por sus siglas en inglés) y los cuartiles son medidas estadísticas que se utilizan para comprender la distribución de un conjunto de datos y para identificar la variabilidad en los datos.

- Los cuartiles son útiles para entender cómo se distribuyen los datos y proporcionan información sobre la dispersión y la forma de la distribución.
- El IQR es una medida de dispersión que se centra en la mitad central de los datos. Es menos sensible a valores atípicos que el rango completo y proporciona una indicación de la variabilidad de los datos en el intervalo intercuartílico. Es indispensable para identificar datos como outliers.

[ ]:
```
# Calculo de los cuantiles
cuantiles = df_house.quantile(q=[0.75, 0.50, 0.25], numeric_only=True)
cuantiles = cuantiles.transpose().rename_axis('Variable').reset_index()
cuantiles
```

[ ]:

	Variable	0.75	0.5	0.25
0	price	5.740000	4.340000	3.430000
1	area	590.863080	427.353800	334.450800
2	bedrooms	3.000000	3.000000	2.000000
3	bathrooms	2.000000	1.000000	1.000000
4	stories	2.000000	2.000000	1.000000
5	parking	1.000000	0.000000	0.000000
6	price_per_area	12751.099368	10251.347668	8023.103348

[ ]:
```
# Cálculo del IQR
cuantiles['IQR'] = cuantiles[0.75] - cuantiles[0.25]

# Cálculo del limite inferior y superior
cuantiles['Limite inferior'] = cuantiles[0.25] - 1.5*cuantiles['IQR']
cuantiles['Limite superior'] = cuantiles[0.75] + 1.5*cuantiles['IQR']
cuantiles
```

[ ]:

	Variable	0.75	0.5	0.25	IQR	\
0	price	5.740000	4.340000	3.430000	2.31000	
1	area	590.863080	427.353800	334.450800	256.41228	
2	bedrooms	3.000000	3.000000	2.000000	1.00000	
3	bathrooms	2.000000	1.000000	1.000000	1.00000	
4	stories	2.000000	2.000000	1.000000	1.00000	

```
5         parking      1.000000      0.000000     0.000000     1.00000
6   price_per_area  12751.099368  10251.347668  8023.103348  4727.99602

   Limite inferior  Limite superior
0       -0.035000         9.205000
1      -50.167620       975.481500
2        0.500000         4.500000
3       -0.500000         3.500000
4       -0.500000         3.500000
5       -1.500000         2.500000
6      931.109318     19843.093398
```

Calculemos ahora, el coeficiente de variación de cada una de nuestras variables.

[ ]:
```
# Finción para calcular la variabilidad de nuestro dataset numérico
def CV_df(df):
    categorical = df.select_dtypes(include='category').columns
    df_numerico = df.drop(columns=categorical)
    non_categorical = df_numerico.columns

    CV = {columna: (df_numerico[columna].std() * 100) / df_numerico[columna].
 ↪mean() for columna in non_categorical}
    CV = pd.DataFrame([CV], columns=non_categorical)
    return CV

CV_df(df_house)
```

[ ]:
```
       price       area   bedrooms  bathrooms    stories     parking  \
0  39.239477  42.134232  24.891386  39.065041  48.047093  124.223342

   price_per_area
0       34.886501
```

7 4. Análisis Univariado

El análisis univariado es una técnica estadística que se centra en el estudio de una única variable a la vez. En otras palabras, examina las propiedades y características de una variable sin considerar la influencia de otras variables. Este tipo de análisis es fundamental en estadísticas descriptivas y es una etapa esencial en el proceso de exploración y comprensión de conjuntos de datos.

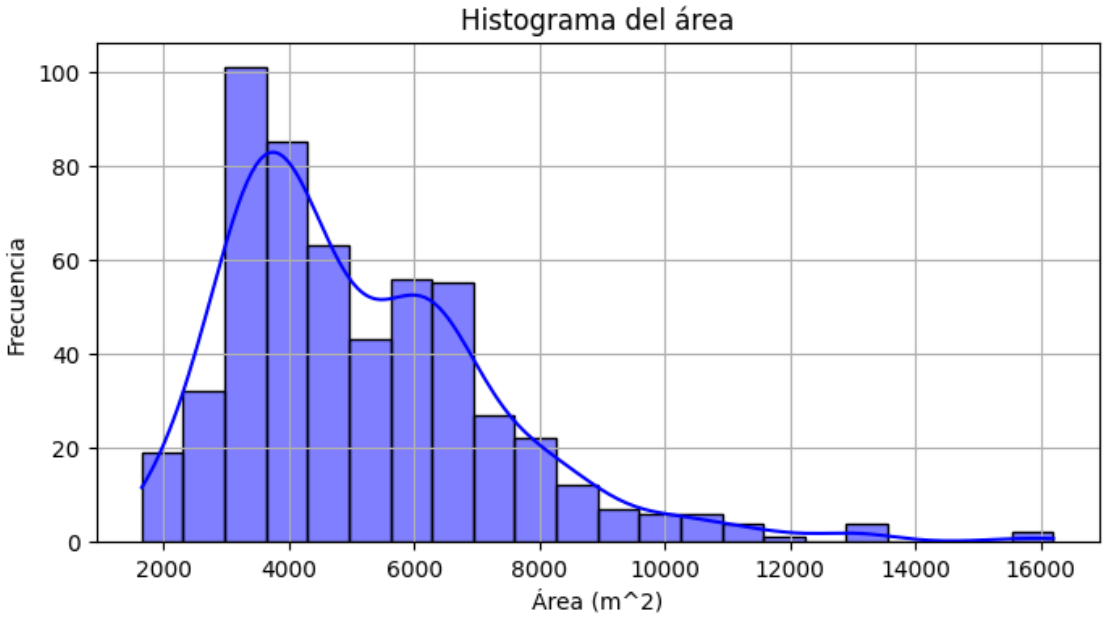
Las principales características y objetivos del análisis univariado incluyen: * Medidas de Tendencia Central * Medidas de Dispersión * Visualización * Descripción de la Distribución * Identificación de Outliers

7.0.1 4.1. Variables numéricas

Visualización de una variable objetivo

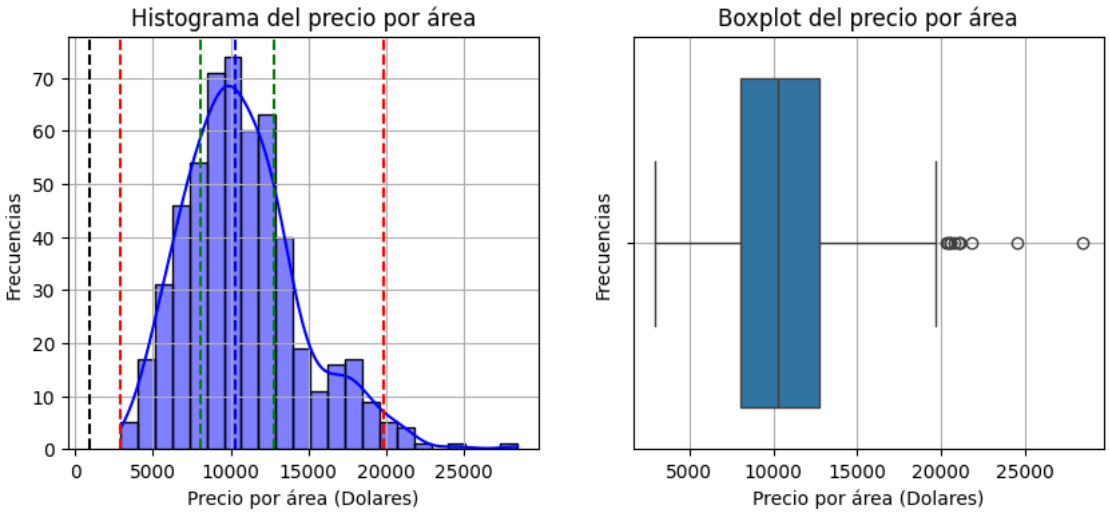
[ ]:
```
# Histograma de la variable precio
plt.figure(figsize=(8,4))
sns.histplot(data = df_house, x="price",kde=True, color='blue')

plt.axvline(
    x = df_house['price'].mean(),
    color = 'red',
    linestyle = 'dashed'
)

plt.axvline(
    x = df_house['price'].median(),
    color = 'green',
    linestyle = 'dashed'
)

plt.axvline(
    x = df_house['price'].mode()[0],
    color = 'black',
    linestyle = 'dashed'
)

plt.title("Histograma del Precio")
plt.xlabel("Precio (Millones de dolares)")
plt.ylabel("Frecuencia")
plt.grid(True)
plt.show()
```

[ ]:
```
# Histograma de la variable area
plt.figure(figsize=(8,4))
sns.histplot(data=df, x="area", kde=True, color='blue')
plt.title("Histograma del área")
plt.xlabel("Área (m^2)")
plt.ylabel("Frecuencia")
plt.grid(True)
plt.show()
```



[ ]:
```
# Histograma y boxplot del precio por área
fig, ax = plt.subplots(1,2,figsize=(10,4))

sns.histplot(data = df_house, x="price_per_area", kde=True, color='blue', ax =␣
↪ax[0])
sns.boxplot(data  = df_house, x="price_per_area", ax = ax[1])

q1 = df_house['price_per_area'].quantile(0.25)
q2 = df_house['price_per_area'].quantile(0.5)
q3 = df_house['price_per_area'].quantile(0.75)
low_lim = q1 - 1.5*(q3-q1)
up_lim  = q3 + 1.5*(q3-q1)

ax[0].axvline(x = low_lim, color='black', linestyle='dashed')
ax[0].axvline(x = up_lim, color='red', linestyle='dashed')
```

```
ax[0].axvline(x = df_house['price_per_area'].min(), color='red',␣
 ↪linestyle='dashed')

ax[0].axvline(x = q1, color='green', linestyle='dashed')
ax[0].axvline(x = q2, color='blue', linestyle='dashed')
ax[0].axvline(x = q3, color='green', linestyle='dashed')

ax[0].set_title("Histograma del precio por área")
ax[0].set_xlabel("Precio por área (Dolares)")
ax[0].set_ylabel("Frecuencias")
ax[1].set_title("Boxplot del precio por área")
ax[1].set_xlabel("Precio por área (Dolares)")
ax[1].set_ylabel("Frecuencias")
ax[0].grid(True)
ax[1].grid(True)
plt.show()
```



Según el boxplot, exísten valores atípicos a partir del límite superior, para poder saber qué registros son éstos, podemos emplear una consulta con filtro.

[ ]:
```
# Detección de outliers según el precio por área
outliers_up = df_house[df_house["price_per_area"] > up_lim]
print(outliers_up.shape)
outliers_up.describe()
```

```
(11, 14)
```

[ ]:
```
            price        area   bedrooms  bathrooms    stories    parking  \
count   11.000000   11.000000  11.000000  11.000000  11.000000  11.000000
mean     6.085227  278.092462   3.454545   1.363636   2.181818   0.909091
```

```
std     2.352199  103.785602   0.687552   0.504525   0.404520   0.831209
min     3.150000  153.289950   3.000000   1.000000   2.000000   0.000000
25%     4.235000  193.238240   3.000000   1.000000   2.000000   0.000000
50%     5.495000  261.707751   3.000000   1.000000   2.000000   1.000000
75%     8.697500  363.250730   4.000000   2.000000   2.000000   1.500000
max     9.240000  427.353800   5.000000   2.000000   3.000000   2.000000

       price_per_area
count       11.000000
mean     21869.172479
std       2470.999008
min      20290.312091
25%      20557.790837
50%      20996.703304
75%      21466.913629
max      28416.735735
```

[ ]:
```
outliers_up
```

[ ]:
```
      price        area  bedrooms  bathrooms  stories mainroad guestroom  \
13   9.2400  325.160500         4          2        2      yes        no
18   8.8900  427.353800         3          2        2      yes       yes
20   8.7500  401.340960         3          1        2      yes        no
23   8.6450  423.637680         3          2        2      yes       yes
108  6.1075  301.005720         4          1        3      yes        no
157  5.4950  261.707751         4          2        2       no       yes
271  4.3400  176.980215         5          1        2       no        no
282  4.2700  202.064025         3          1        2       no       yes
302  4.2000  199.276935         3          1        3      yes        no
345  3.8500  187.199545         3          1        2      yes        no
449  3.1500  153.289950         3          1        2       no        no

    basement hotwaterheating airconditioning  parking prefarea  \
13        no             yes              no        2       no
18        no              no             yes        2       no
20       yes             yes              no        2       no
23       yes              no             yes        1       no
108       no              no              no        1       no
157      yes              no              no        1       no
271      yes              no              no        0       no
282      yes              no             yes        0       no
302       no              no              no        1      yes
345      yes              no              no        0      yes
449      yes              no              no        0       no

    furnishingstatus  price_per_area
13         furnished    28416.735735
```

```
18          furnished    20802.435827
20     semi-furnished    21801.911273
23          furnished    20406.588951
108    semi-furnished    20290.312091
157         furnished    20996.703304
271    semi-furnished    24522.515130
282       unfurnished    21131.915986
302       unfurnished    21076.197303
345    semi-furnished    20566.289304
449       unfurnished    20549.292370
```

[ ]:
```
outliers_up.count()
```

[ ]:
```
price               11
area                11
bedrooms            11
bathrooms           11
stories             11
mainroad            11
guestroom           11
basement            11
hotwaterheating     11
airconditioning     11
parking             11
prefarea            11
furnishingstatus    11
price_per_area      11
dtype: int64
```

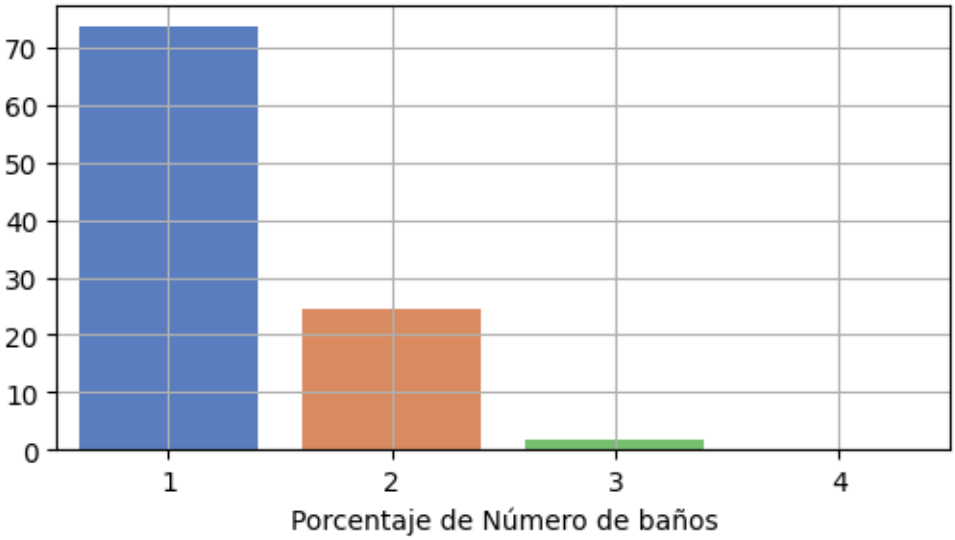
[ ]:
```
# Cantidad de datos de una variable numérica discreta
df_house['bathrooms'].value_counts()
```

[ ]:
```
1    401
2    133
3     10
4      1
Name: bathrooms, dtype: int64
```

[ ]:
```
# Porcentaje de datos para una variable numérica discreta
porcentaje_baños = df_house['bathrooms'].value_counts(normalize=True)*100

plt.figure(figsize=(6,3))
sns.barplot(
    x = porcentaje_baños.index,
    y = porcentaje_baños.values,
    palette = 'muted'
    )
```

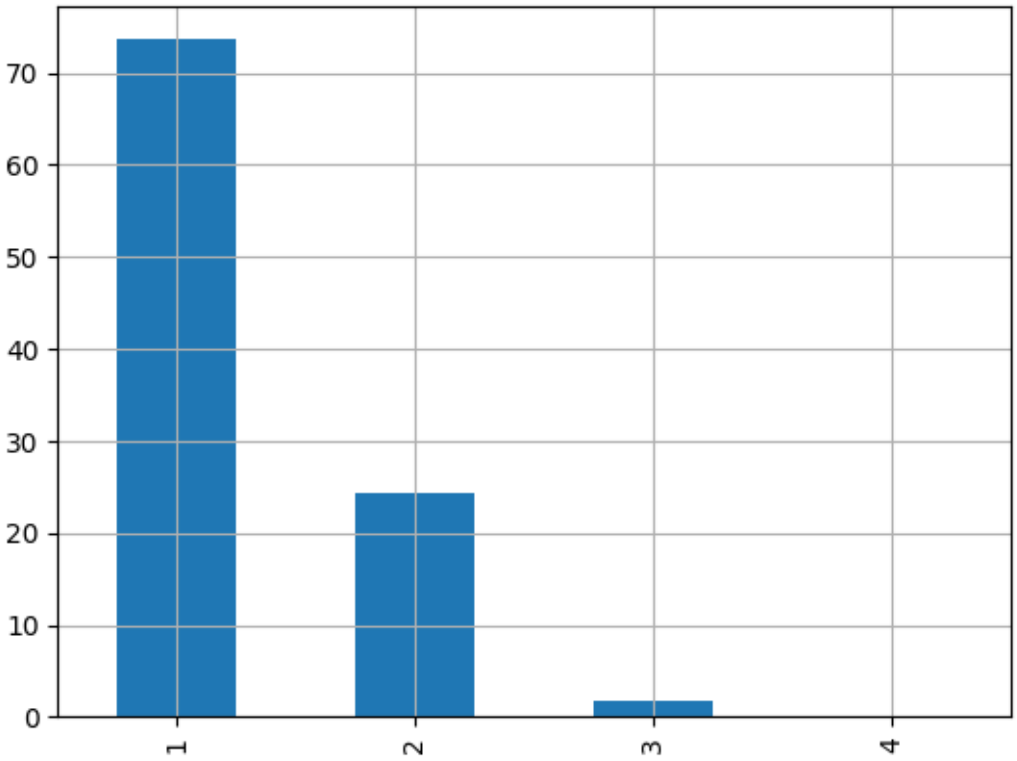
```
plt.grid(True)
plt.xlabel('Porcentaje de Número de baños')
plt.show()
```

```
<ipython-input-29-83a3b4e781ec>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(
```



Pandas tiene funciones integradas para la visualización de datos, las cuales son construidas sobre la biblioteca Matplotlib. Estas funciones permiten generar gráficos directamente desde un DataFrame de Pandas.

[ ]:
```
porcentaje_baños.plot(kind='bar')
plt.grid(True)
```



7.0.2 4.2. Variables categóricas

Ejecutamos un plot de barras para ver el conteo de las cariables categoricas de acuerdo a sus clases.

[ ]:
```
# Variables categóricas
categoricas = df_house.describe(include='category').columns.tolist()
categoricas
```

[ ]:
```
['mainroad',
 'guestroom',
 'basement',
 'hotwaterheating',
 'airconditioning',
 'prefarea',
 'furnishingstatus']
```

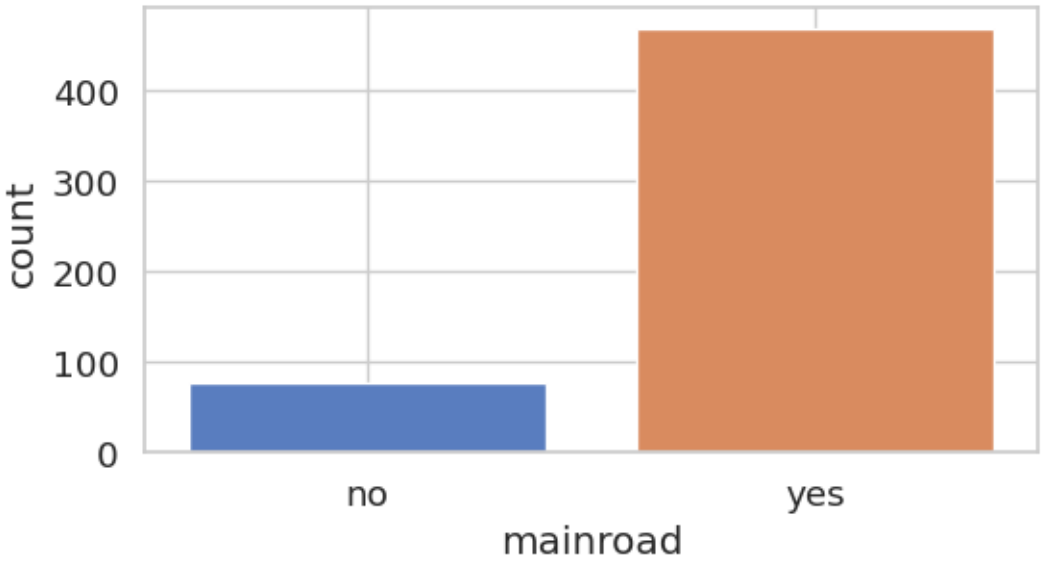
[ ]:
```
# Conteo de clases
df_house['prefarea'].value_counts()
```

[ ]:
```
no     417
yes    128
Name: prefarea, dtype: int64
```

¿Cuántos inmuebles están cerca a una vía principal?

[ ]:
```
plt.figure(figsize=(6,3))
sns.countplot(data=df_house,x=categoricas[0], palette = 'muted')
plt.grid(True)
```

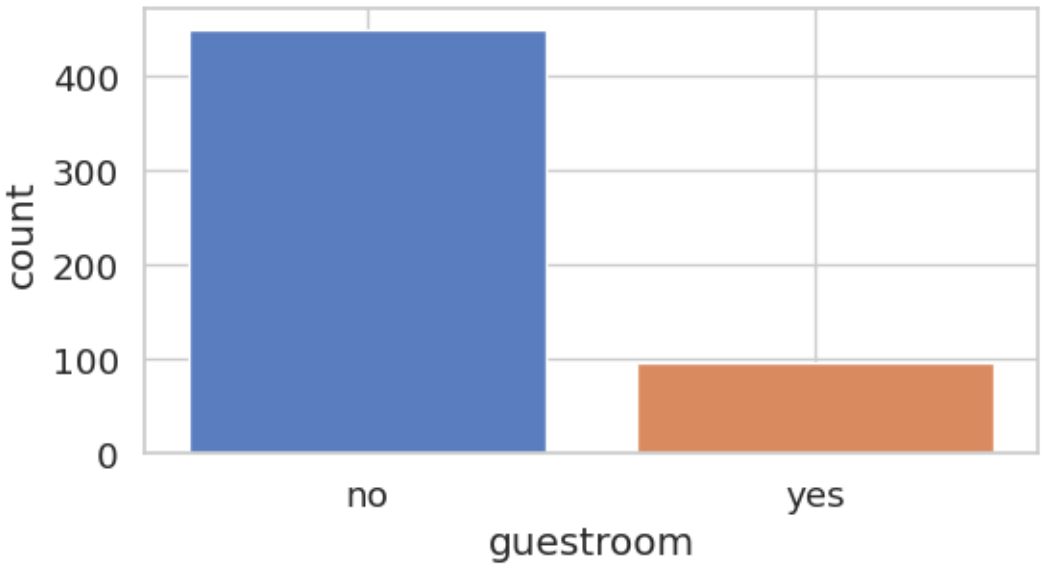
```
<ipython-input-60-47dcfe122575>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=df_house,x=categoricas[0], palette = 'muted')
```



¿Cuántos inmuebles poseen habitación para huéspedes?

[ ]:
```
plt.figure(figsize=(6,3))
sns.countplot(data=df_house,x=categoricas[1], palette = 'muted')
plt.grid(True)
```

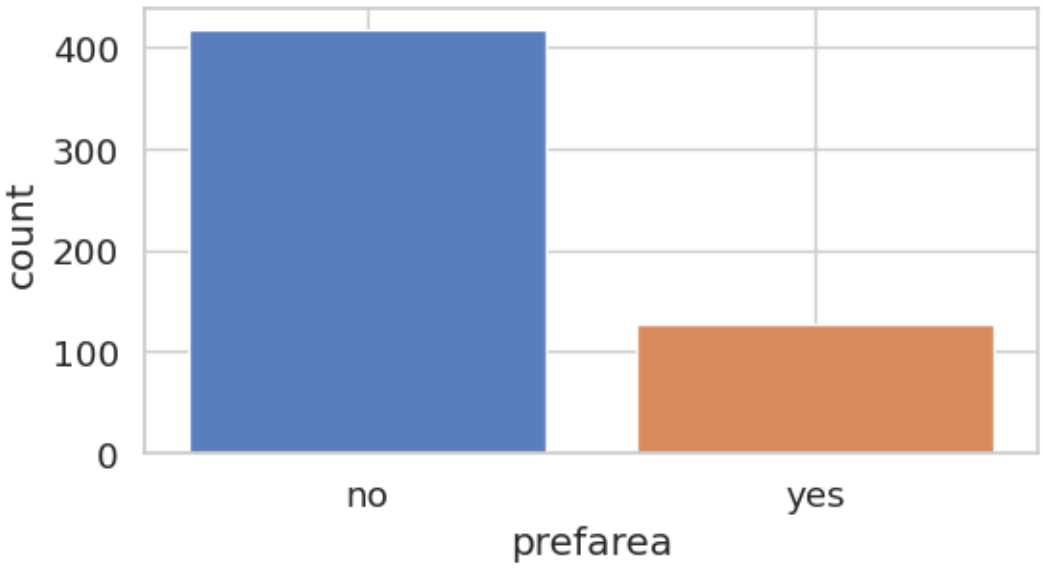
```
<ipython-input-59-3fc43b4ecd5c>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=df_house,x=categoricas[1], palette = 'muted')
```



¿Cuántos inmuebles están en una zona preferencial de la ciudad?

[ ]:
```
plt.figure(figsize=(6,3))
sns.countplot(data=df_house,x=categoricas[-2], palette = 'muted')
plt.grid(True)
```

```
<ipython-input-58-f1332b40ecd8>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=df_house,x=categoricas[-2], palette = 'muted')
```

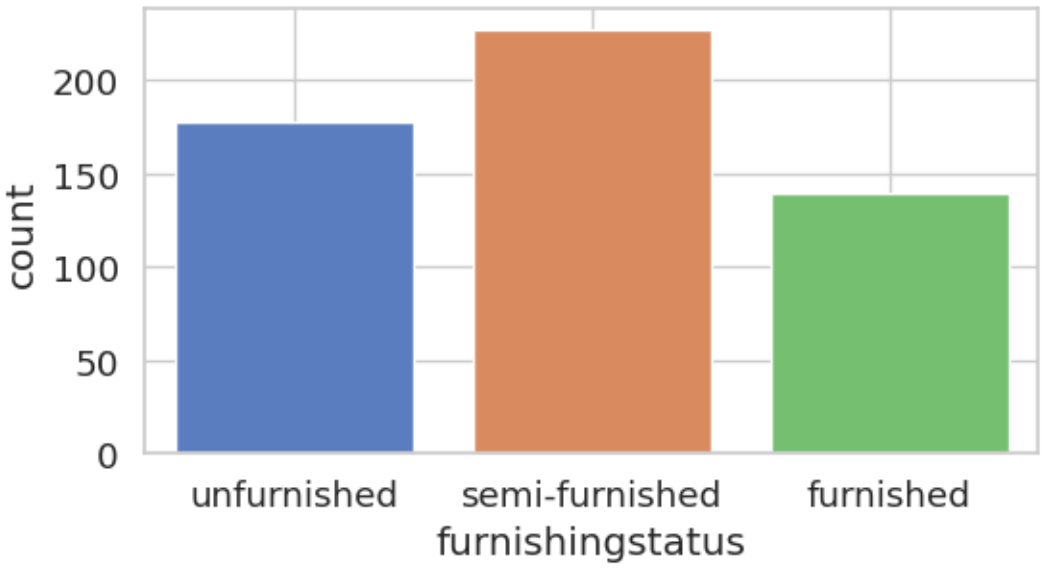


[ ]:
```
order = df_house['furnishingstatus'].cat.categories.tolist()[::-1]
order
```

[ ]:
```
['unfurnished', 'semi-furnished', 'furnished']
```

[ ]:
```
plt.figure(figsize=(6,3))
sns.countplot(data = df_house,
              x = 'furnishingstatus',
              palette = 'muted',
              order = order
              )
plt.grid(True)
```

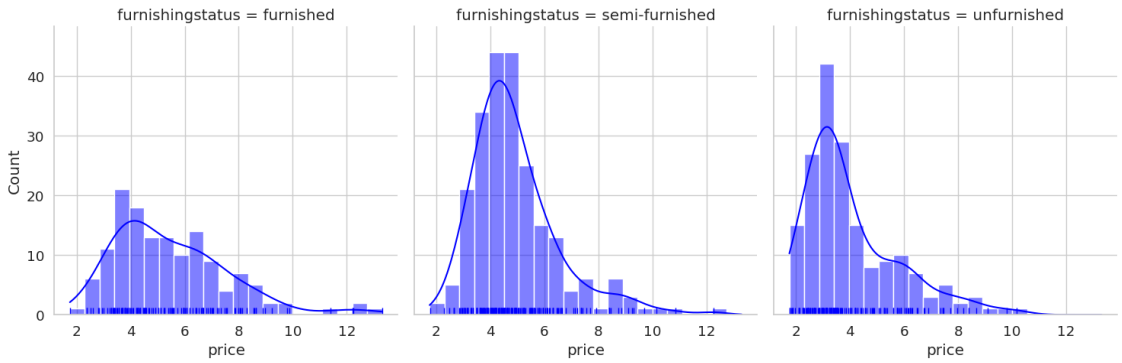
```
<ipython-input-61-665ce059c9b1>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data = df_house,
```



8 5. Análisis Bivariado

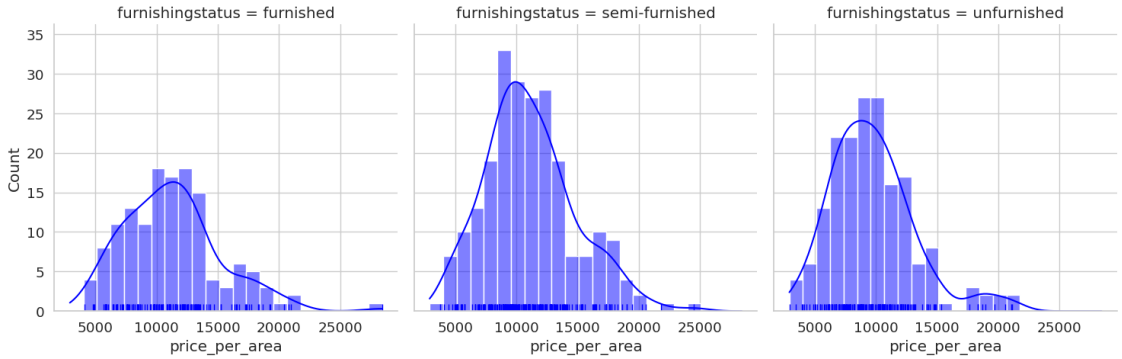
¿Cuál es la distribución de precios de acuerdo al estado de amoblamiento?

[ ]:
```
# Crear un displot con facetas por la variable 'furnishing status'
sns.displot(data=df_house, x='price',
            kde=True,
            color='blue', rug=True,
            col='furnishingstatus')
plt.show()
```

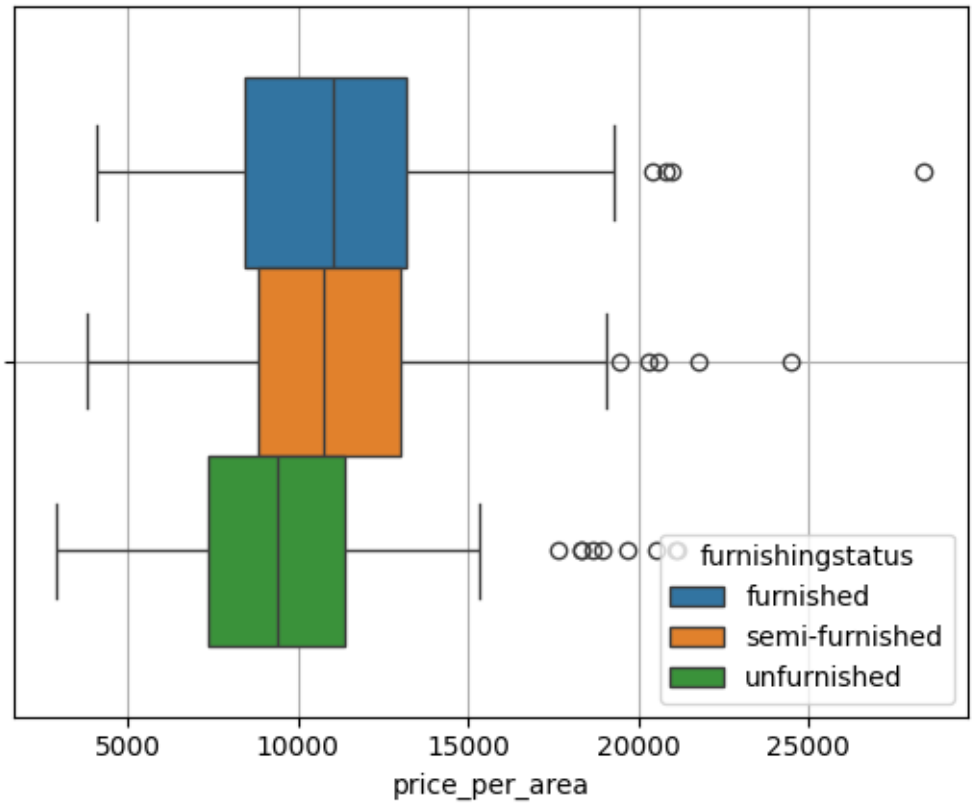


[ ]:
```
# Crear un displot con facetas por la variable 'furnishing status'
sns.displot(data=df_house, x='price_per_area',
            kde=True,
```

```
            color='blue', rug=True,
            col='furnishingstatus')
plt.show()
```



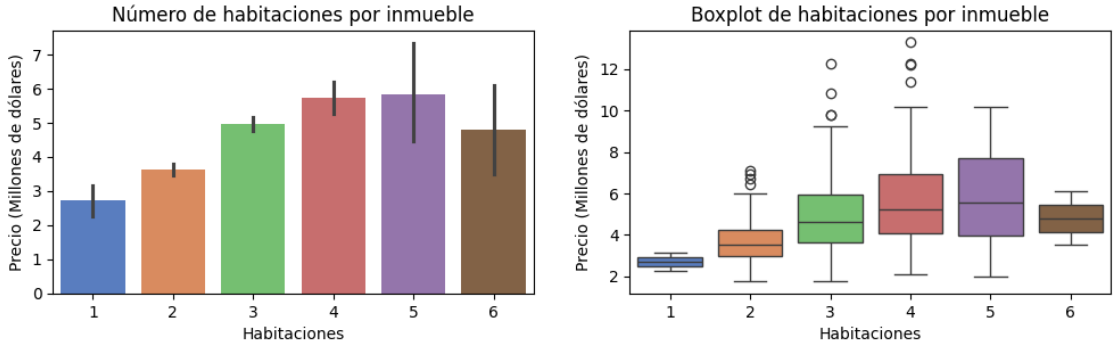
[ ]:
```
# Crear un displot con facetas por la variable 'furnishing status'
sns.boxplot(data=df_house, x='price_per_area',
            hue='furnishingstatus')
plt.grid(True)
plt.show()
```



- ¿Cuál es el precio promedio por número de habitaciones?
- ¿Cuál es el número de habitaciones que posee la mayor variabilidad de precio?
- ¿Cuál es el número de habitaciones que posee más valores atípicos?
- ¿Cuál es el número de habitaciones con mayor precio promedio?

[ ]:
```
# Descripción del precio con respecto al número de habitaciones
fig, ax = plt.subplots(1,2,figsize=(12,3))
sns.barplot(data=df_house, x="bedrooms", y="price", ax=ax[0], palette = 'muted')
sns.boxplot(data=df_house, x="bedrooms", y="price", ax=ax[1], palette = 'muted')

ax[0].set_title("Número de habitaciones por inmueble")
ax[0].set_xlabel("Habitaciones")
ax[0].set_ylabel("Precio (Millones de dólares)")
ax[1].set_title("Boxplot de habitaciones por inmueble")
ax[1].set_xlabel("Habitaciones")
ax[1].set_ylabel("Precio (Millones de dólares)")

plt.show()
```

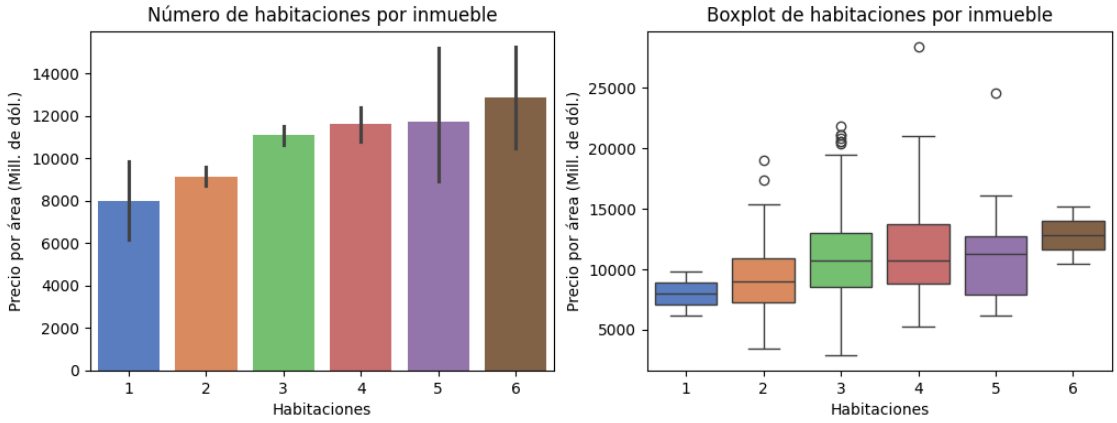
```
<ipython-input-42-3dd5bf2bb1f9>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=df_house, x="bedrooms", y="price", ax=ax[0], palette =
'muted')
<ipython-input-42-3dd5bf2bb1f9>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df_house, x="bedrooms", y="price", ax=ax[1], palette =
'muted')
```



- ¿Cómo depende el precio por área del número de habitaciones?

[ ]:
```
# Descripción del precio por área con respecto al número de habitaciones
fig, ax = plt.subplots(1,2,figsize=(12,4))
sns.barplot(data=df_house, x="bedrooms", y="price_per_area", ax=ax[0], palette␣
 ↪= 'muted')
sns.boxplot(data=df_house, x="bedrooms", y="price_per_area", ax=ax[1], palette␣
 ↪= 'muted')

ax[0].set_title("Número de habitaciones por inmueble")
ax[0].set_xlabel("Habitaciones")
ax[0].set_ylabel("Precio por área (Mill. de dól.)")
ax[1].set_title("Boxplot de habitaciones por inmueble")
ax[1].set_xlabel("Habitaciones")
ax[1].set_ylabel("Precio por área (Mill. de dól.)")

plt.show()
```

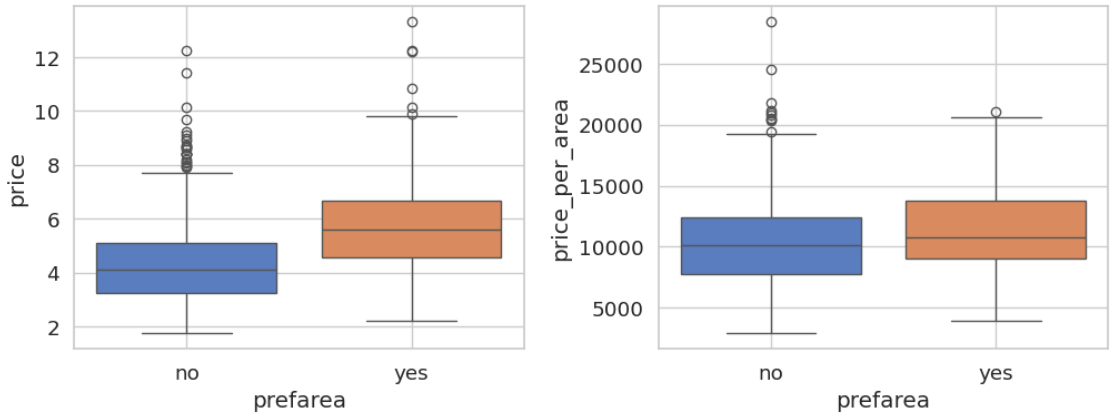
```
<ipython-input-43-91758d21caff>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=df_house, x="bedrooms", y="price_per_area", ax=ax[0], palette
= 'muted')
<ipython-input-43-91758d21caff>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df_house, x="bedrooms", y="price_per_area", ax=ax[1], palette
= 'muted')
```



- ¿Cómo afecta la zona preferencial el precio por área?

[ ]:
```
# Primero, observamos si existe un patrón entre la proporción entre el precio y el área preferencial
fig, ax = plt.subplots(1,2,figsize=(12,4))
sns.boxplot(data=df_house, x="prefarea",y="price", palette = 'muted', ax = ax[0])
sns.boxplot(data=df_house, x="prefarea",y="price_per_area", palette = 'muted', ax = ax[1])
plt.subplots_adjust(wspace=0.3)
ax[0].grid(True)
ax[1].grid(True)
plt.show()
```

```
<ipython-input-62-223af6681677>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(data=df_house, x="prefarea",y="price", palette = 'muted', ax = ax[0])
<ipython-input-62-223af6681677>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(data=df_house, x="prefarea",y="price_per_area", palette = 'muted', ax = ax[1])
```



- Exísten ciertos inmuebles que no están en un área preferencial pero aún así, superan el precio y precio por área de los inmuebles en áreas preferenciales. ¿Qué características poseen éstos inmuebles?

[ ]:
```
# Selección de valores atípicos para inmuebles no ubicados en áreas␣
↪preferenciales
condicion = (df_house['prefarea']=='no') & (df_house['price_per_area'] >= 18000)
outliers_no_preferenciales = df_house.loc[condicion]
```

[ ]:
```
outliers_no_preferenciales.describe()
```

[ ]:
```
            price        area   bedrooms  bathrooms    stories    parking  \
count   18.000000   18.000000  18.000000  18.000000  18.000000  18.000000
mean     5.987139  293.945092   3.277778   1.555556   2.000000   0.722222
std      1.854233   82.399610   0.574513   0.615699   0.342997   0.826442
min      3.150000  153.289950   3.000000   1.000000   1.000000   0.000000
25%      4.651500  242.709088   3.000000   1.000000   2.000000   0.000000
50%      5.477500  285.676725   3.000000   1.500000   2.000000   0.500000
75%      7.218750  343.276585   3.000000   2.000000   2.000000   1.000000
max      9.240000  427.353800   5.000000   3.000000   3.000000   2.000000

       price_per_area
count       18.000000
mean     20404.641836
std       2563.916088
min      18037.712192
25%      18658.495770
50%      19867.401898
75%      20948.136434
max      28416.735735
```

[ ]:
```
outliers_no_preferenciales.describe(include='category')
```

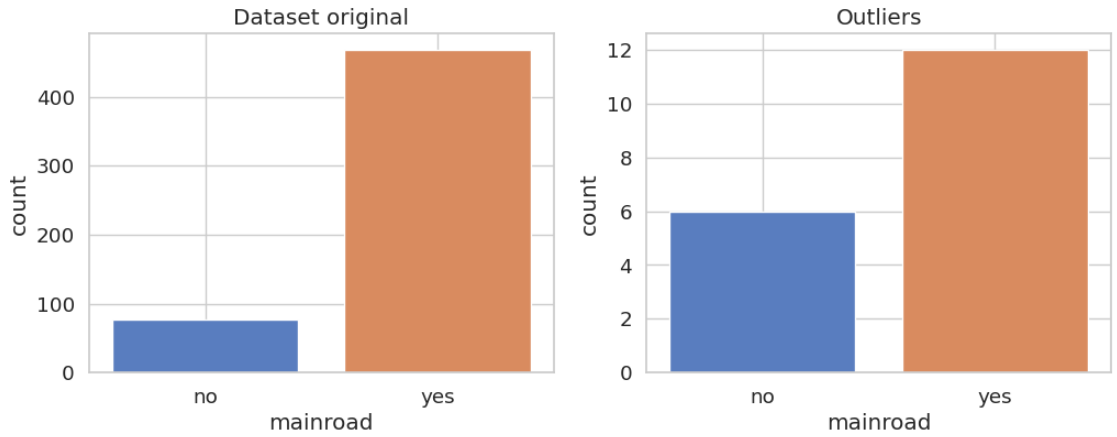
[ ]:

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea	furnishingstatus
count	18	18	18	18	18	18	18
unique	2	2	2	2	2	1	3
top	yes	no	yes	no	no	no	furnished
freq	12	12	13	15	13	18	7

- ¿Es la cercanía a una vía principal, un factor importante en el precio de éstos outliers?

[ ]:
```
# Mainroad como un factor importante en el precio de inmuebles no preferenciales
fig,ax = plt.subplots(1,2,figsize=(12,4))
sns.countplot(data=df_house,x="mainroad", palette = 'muted', ax=ax[0])
sns.countplot(data=outliers_no_preferenciales,x="mainroad", palette = 'muted',␣
 ↪ax=ax[1])

ax[0].set_title("Dataset original")
ax[1].set_title("Outliers")
ax[0].grid(True)
ax[1].grid(True)
plt.show()
```

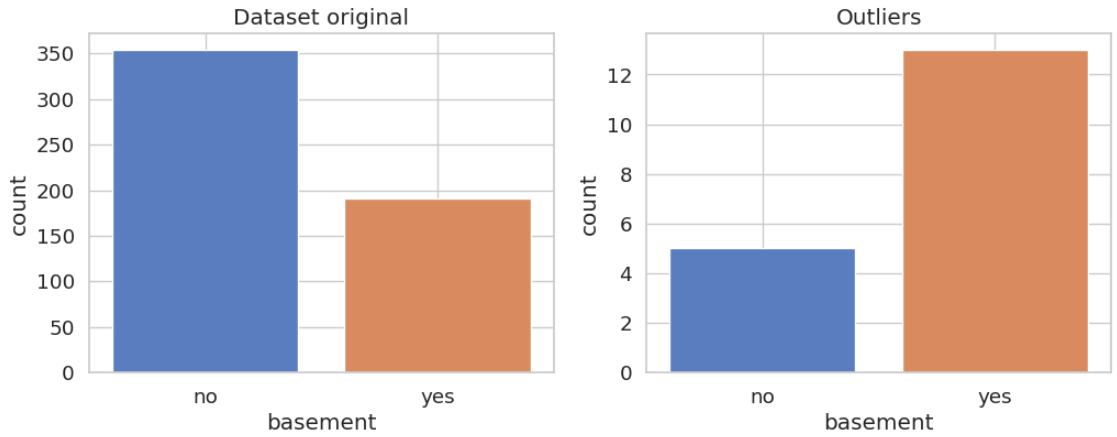
```
<ipython-input-63-5a639a51861f>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=df_house,x="mainroad", palette = 'muted', ax=ax[0])
<ipython-input-63-5a639a51861f>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=outliers_no_preferenciales,x="mainroad", palette = 'muted',
ax=ax[1])
```



- ¿Tener sótano es importante para el precio?

[ ]:
```
# Basement como un factor importante en el precio de inmuebles no preferenciales
fig,ax = plt.subplots(1,2,figsize=(12,4))
sns.countplot(data=df_house,x="basement", palette = 'muted', ax=ax[0])
sns.countplot(data=outliers_no_preferenciales, x="basement", palette = 'muted',␣
 ↪ax=ax[1])

ax[0].set_title("Dataset original")
ax[1].set_title("Outliers")
ax[0].grid(True)
ax[1].grid(True)
plt.show()
```

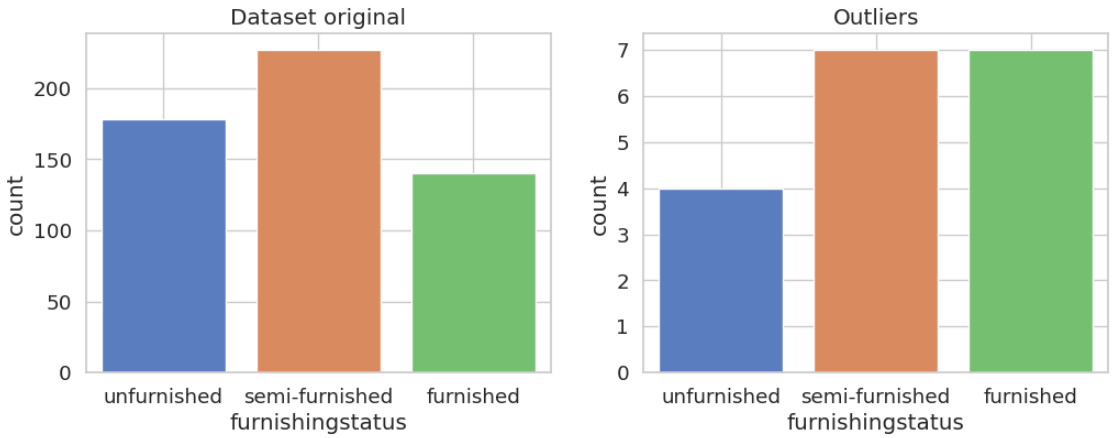
```
<ipython-input-64-acca65b07581>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=df_house,x="basement", palette = 'muted', ax=ax[0])
<ipython-input-64-acca65b07581>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=outliers_no_preferenciales, x="basement", palette =
'muted', ax=ax[1])
```



- ¿Es el amoblamiento importante para el precio del inmueble?

[ ]:
```
# Basement como un factor importante en el precio de inmuebles no preferenciales
orden = ["unfurnished","semi-furnished","furnished"]
fig,ax = plt.subplots(1,2,figsize=(12,4))
sns.countplot(data=df_house, x="furnishingstatus", palette = 'muted',␣
 ↪ax=ax[0],order=orden)
sns.countplot(data=outliers_no_preferenciales, x="furnishingstatus", palette =␣
 ↪'muted', ax=ax[1],order=orden)

ax[0].set_title("Dataset original")
ax[1].set_title("Outliers")
ax[0].grid(True)
ax[1].grid(True)
plt.show()
```

```
<ipython-input-65-a3ae852356c9>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=df_house, x="furnishingstatus", palette = 'muted',
ax=ax[0],order=orden)
<ipython-input-65-a3ae852356c9>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=outliers_no_preferenciales, x="furnishingstatus", palette =
'muted', ax=ax[1],order=orden)
```

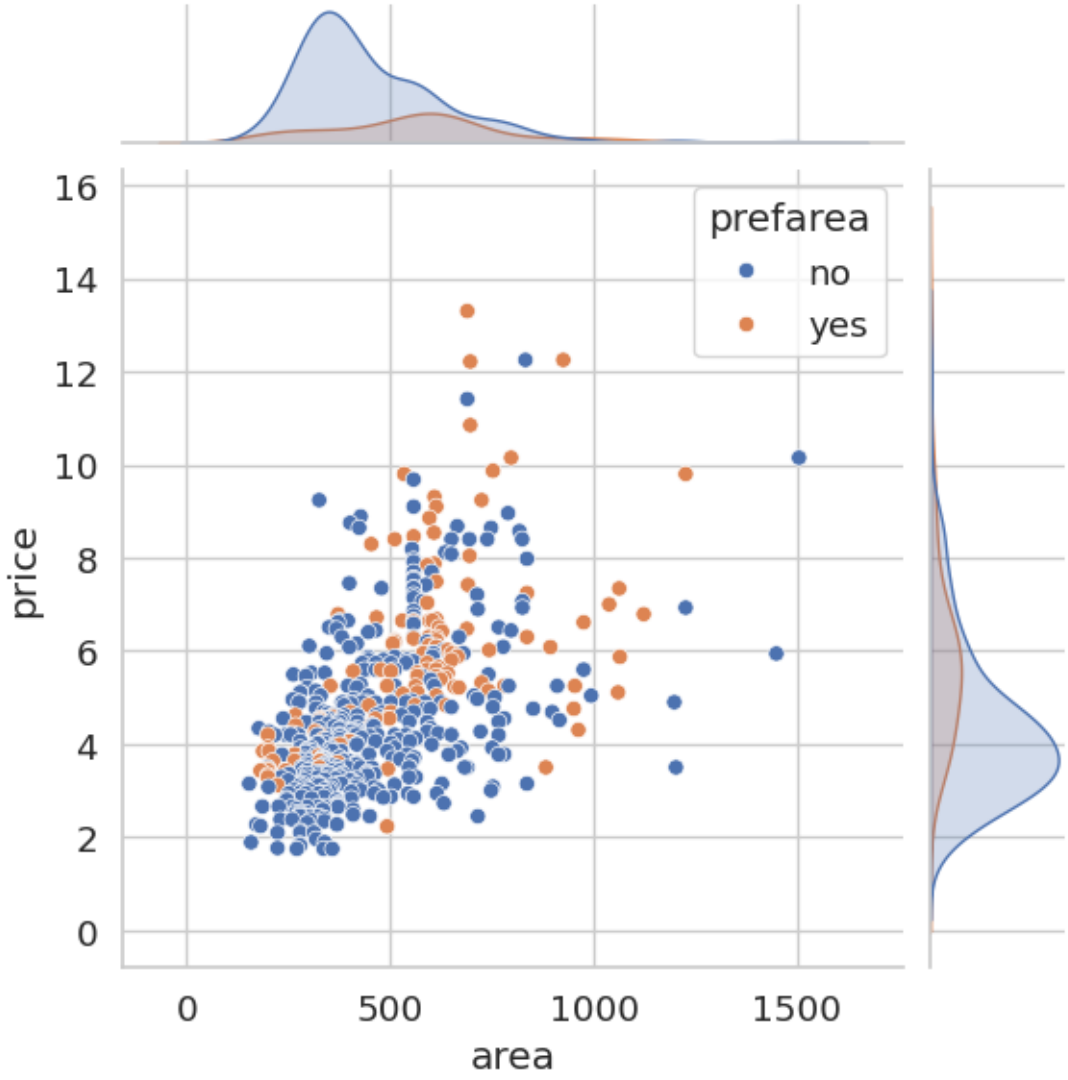


- ¿Qué relación existe entre el precio y el área del inmueble?
- ¿Existe algún patrón entre el precio y el área del inmueble?
- ¿Existe alguna correlación entre ambas variables?

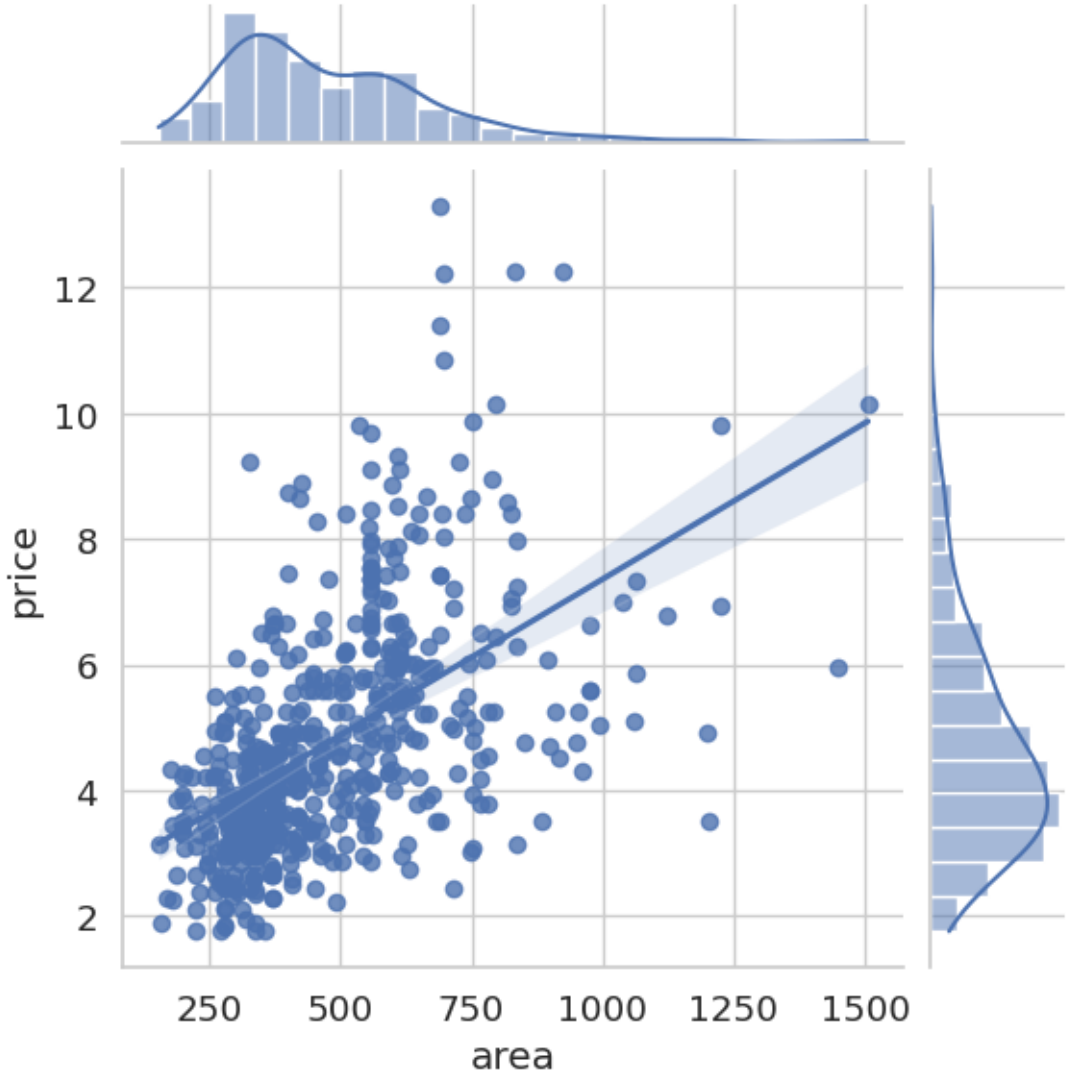
[ ]:
```
sns.set(style="whitegrid", font_scale=1.2)
```

[ ]:
```
# Jointplot y grupos preferenciales
sns.jointplot(data=df_house, x="area", y="price",hue='prefarea')
```

[ ]:
```
<seaborn.axisgrid.JointGrid at 0x7865dbdcf3a0>
```

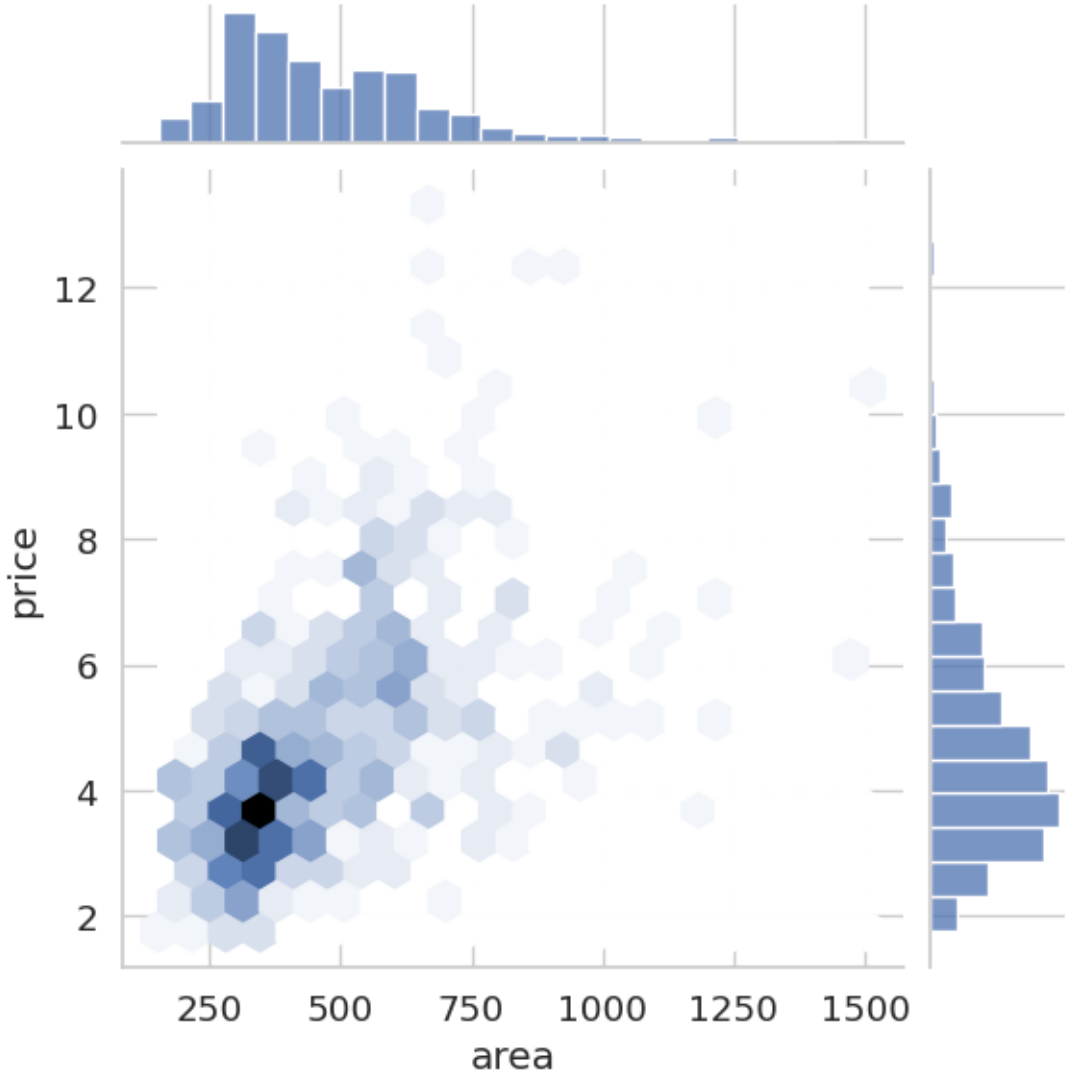


[ ]:
```
# Jointplot y análisis de regresión
sns.jointplot(data=df_house, x="area", y="price", kind='reg')
plt.show()
```



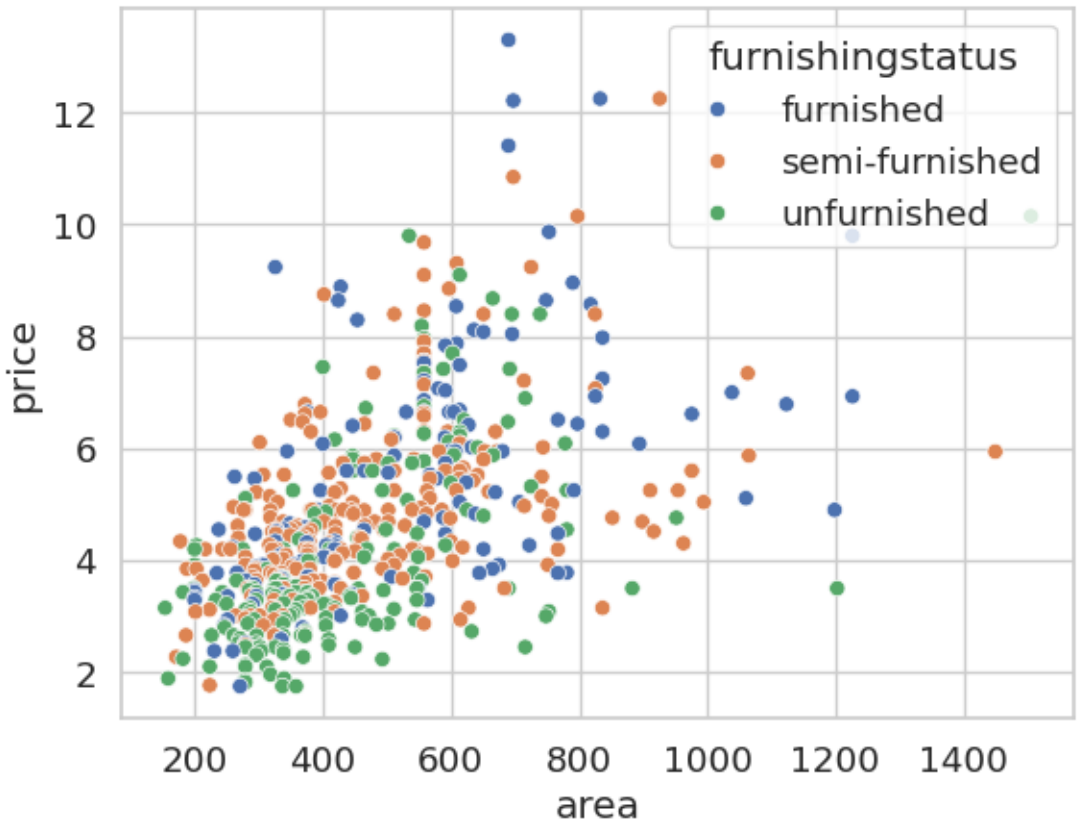
- ¿En que regiones de la relación área-precio, se concentran la mayoría de los regístros?

[ ]:
```
# Concentración de regístros precio-área
sns.jointplot(data=df_house, x="area", y="price", kind="hex")
plt.show()
```



- ¿Exíste alguna relación visible entre el precio, el área y el estado de amoblamiento?

[ ]:
```
# Relación precio-área-amoblamiento
sns.scatterplot(data=df_house,x="area",y="price",hue='furnishingstatus')
plt.show()
```



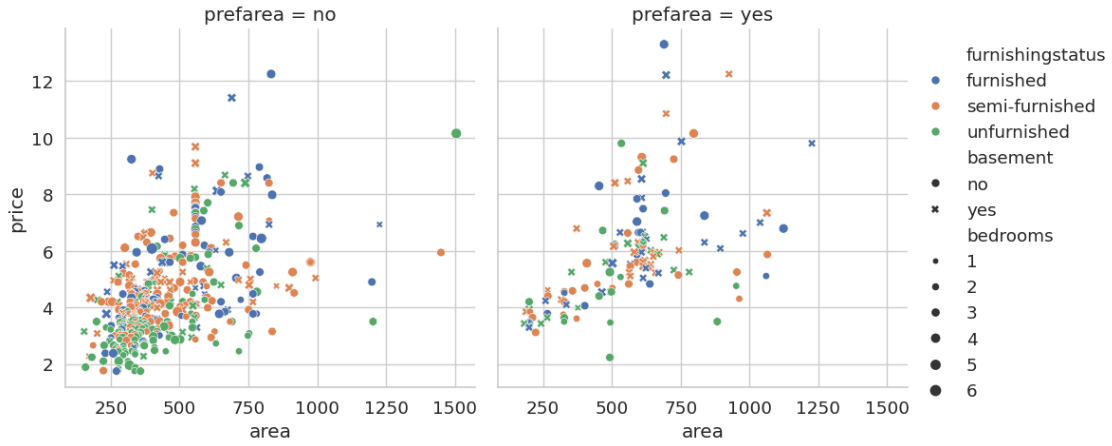
[ ]:
```
df_house.columns
```

[ ]:
```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus', 'price_per_area'],
      dtype='object')
```

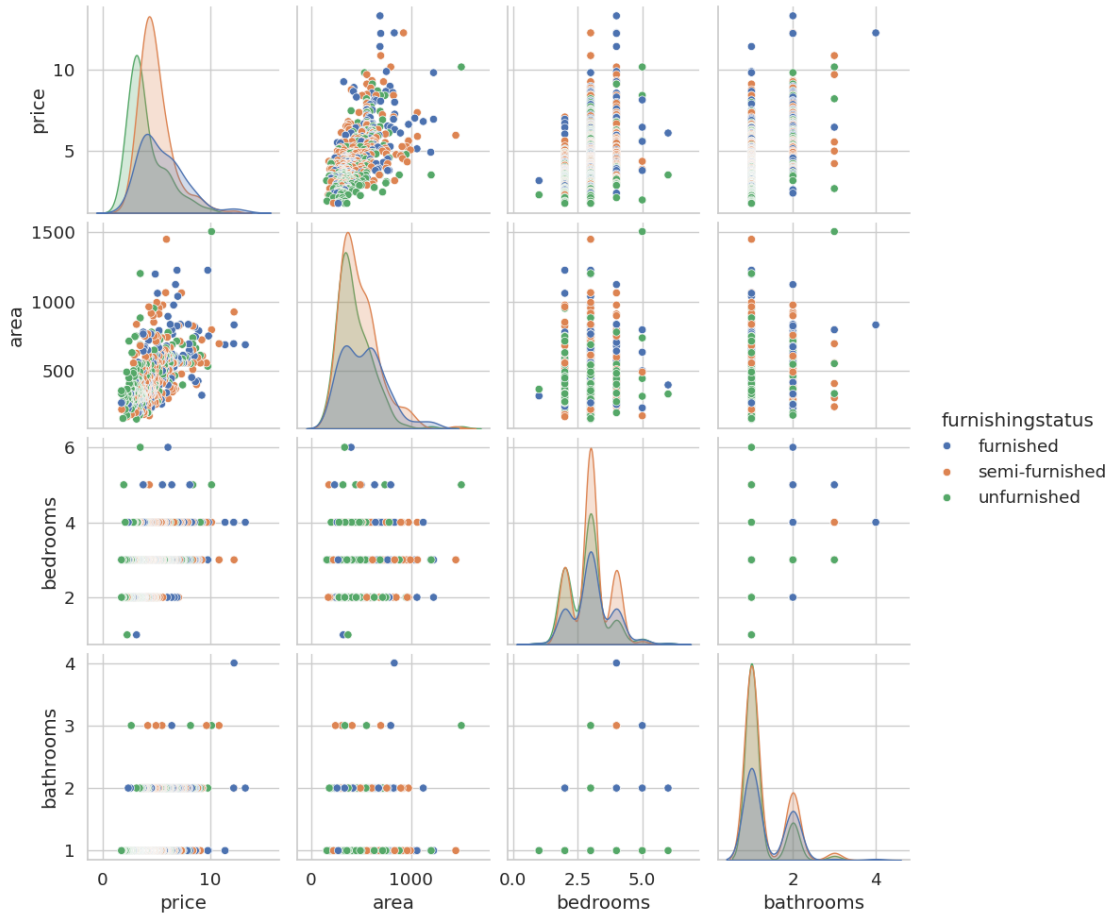
- ¿Es posible encontrar un patrón entre la relación entre el área preferencial, el estado de amoblamiento, la disponibilidad de sótano, el precio y el área del inmueble?

[ ]:
```
# Relplot para mostrar múltiples divisiones de los datos
plt.figure(figsize=(8,8))
sns.relplot(data=df_house,x='area',y='price',
            hue='furnishingstatus',style='basement',size='bedrooms',
            kind='scatter',col='prefarea')

plt.show()
```

```
<Figure size 800x800 with 0 Axes>
```



[ ]:
```
# Pairplot con hue y reg
df_pivot = df_house.drop(columns =␣
  ↪['stories','hotwaterheating','airconditioning','parking','price_per_area'])
sns.pairplot(data=df_pivot,hue='furnishingstatus')
plt.show()
```



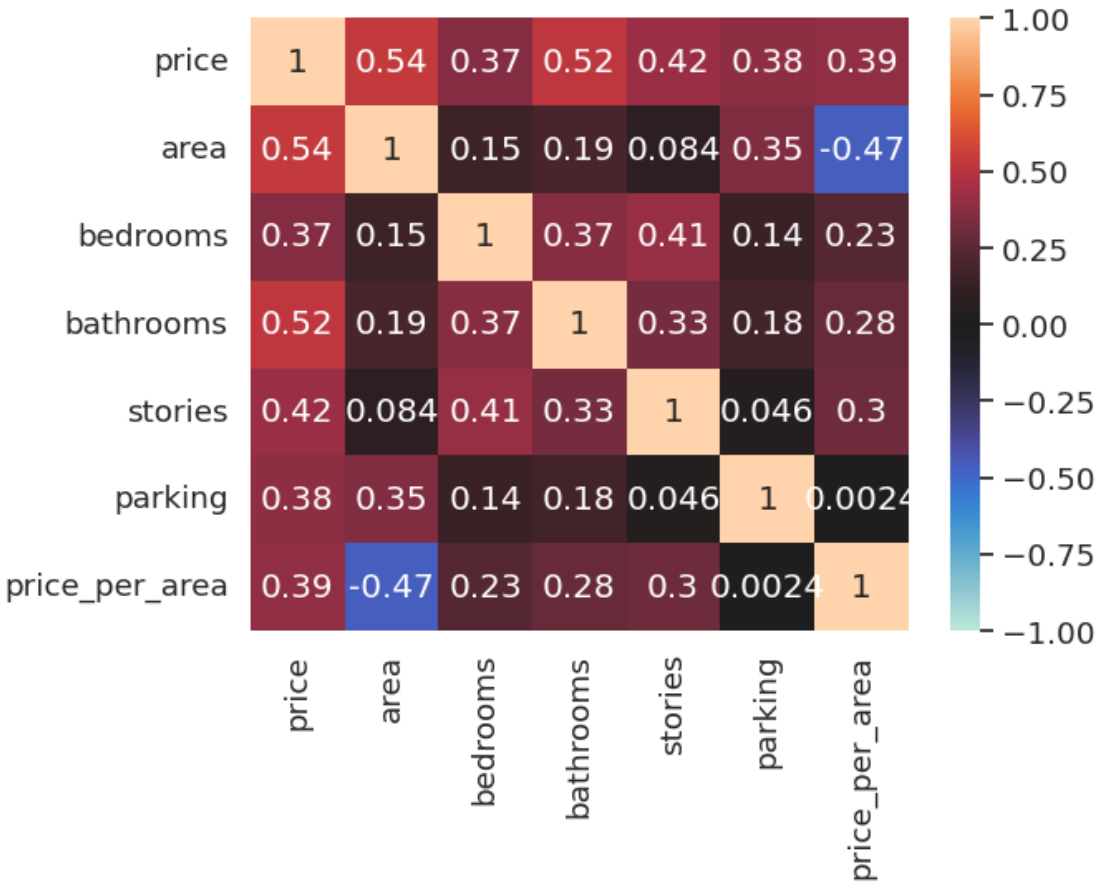
- ¿Exíste correlación (colinealidad) entre las variables?

[ ]:
```
# Correlación entre variables individuales
coef = df['variable_1'].cor(df['variable_2'])
```

[ ]:
```
# Extracción de variables categóricas
variables_categoricas = df_house.select_dtypes(include = ['category'])
df_numerico = df_house.drop(columns = variables_categoricas)
```

[ ]:
```
# Correlación de variables numéricas
correlacion = df_numerico.corr()
sns.heatmap(data= correlacion,
            vmin = -1,
            vmax = 1,
            center = 0,
            annot = True)
```

[ ]: <Axes: >



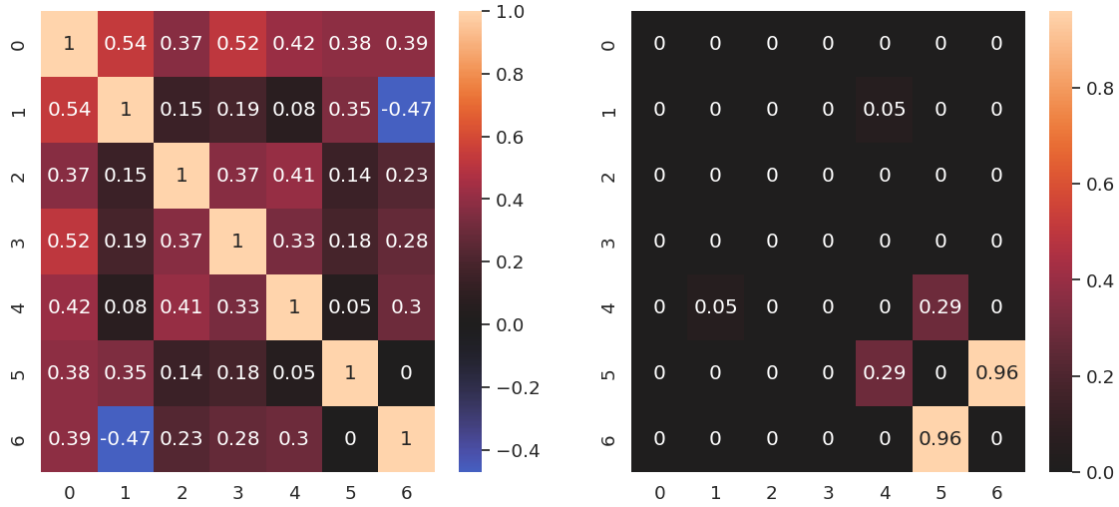
SciPy es una biblioteca de Python utilizada para realizar cálculos científicos y matemáticos. Ofrece una amplia gama de funciones y herramientas para el análisis de datos, la optimización, la interpolación, entre otros. En el contexto del análisis de datos, SciPy proporciona métodos estadísticos clave que permiten evaluar la relación entre variables y determinar su significancia.

Cuando se analizan datos y se buscan relaciones entre variables, es crucial respaldar las conclusiones con estadística inferencial. Una medida común para evaluar la relación entre variables es el coeficiente de correlación, que puede calcularse utilizando métodos disponibles en SciPy como: * `stats.pearsonr`: Coeficiente de correlación r de Pearson. * `stats.spearmanr`: Coeficiente de correlación ρ de Spearman. * `stats.kendalltau`: Coeficiente de correlación τ de Kendall.

Estas funciones devuelven no solo el coeficiente de correlación, sino también el valor p asociado. Si el **p-valor** $< \alpha$ (nivel de significancia), esto implica que la correlación entre variables es estadísticamente significativa y que el valor de los estadísticos de correlación dan cuenta de la colinealidad de las variables. De lo contrario, la correlación puede deberse a la aleatoriedad de los datos independientemente de su valor.

[ ]:
```
from scipy import stats

columns = df_numerico.columns
l = len(columns)
corr_matrix = [ [0 for _ in range(l)] for _ in range(l) ]
p_matrix    = [ [0 for _ in range(l)] for _ in range(l) ]

for i, columna_1 in enumerate(columns):
  for j, columna_2 in enumerate(columns):
    corr_matrix[i][j], p_matrix[i][j] = stats.
  ↪pearsonr(df_numerico[columna_1],df_numerico[columna_2])
    corr_matrix[i][j] = round(corr_matrix[i][j],2)
    p_matrix[i][j]    = round(p_matrix[i][j],2)

print(columns)

fig, ax = plt.subplots(1,2,figsize = (14,6))
sns.heatmap(data = corr_matrix,center=0, annot = True, ax=ax[0])
sns.heatmap(data = p_matrix,   center=0, annot = True, ax=ax[1])
plt.show()
```

```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking',
       'price_per_area'],
      dtype='object')
```



9 6. Análisis por agrupación

Hacer agrupaciones de datos en un Análisis Exploratorio de Datos (EDA, por sus siglas en inglés) con Pandas es una técnica fundamental para comprender la estructura y las características de un conjunto de datos. Las agrupaciones permiten analizar y resumir la información de manera más significativa al agrupar los datos según una o más variables y luego aplicar funciones de resumen o análisis a cada grupo. Dentro de las utilidades de generar agrupaciones en los datos encontramos:

- Resumen de datos por categorías.
- Análisis comparativo.
- Identificación de patrones y tendencias.
- Segmentación de datos.
- Preparación de datos para visualización o modelado.

El producto de una agrupación en pandas en un objeto groupby, el cual es un iterable que contiene el nombre de cada grupo y además un subconjunto o dataframe para cada grupo.

[ ]:
```
grupos = df_house.groupby('bedrooms')
```

[ ]:
```
for nombre, grupo in grupos:
  print('Nombre del grupo:', nombre)
  print(grupo)
```

Para obtener un grupo en específico, empleamos el método `get_group()`.

[ ]:
```
grupos_ = []
for i in range(len(grupos)):
  grupo = grupos.get_group(i+1)
  grupos_.append(grupo)
```

[ ]:
```
grupos_[0]
```

[ ]:
```
     price       area  bedrooms  bathrooms  stories mainroad guestroom  \
445  3.150  320.51535         1          1        1      yes        no
528  2.275  368.82491         1          1        1       no        no

    basement hotwaterheating airconditioning  parking prefarea  \
445       no              no              no        0       no
528       no              no              no        0       no

    furnishingstatus  price_per_area
445        furnished     9827.922438
528      unfurnished     6168.238474
```

Ya que cada agrupación corresponde a un dataframe, nos es posible ejecutar sobre este todas las operaciónes que deseemos. Por ejemplo, veamos el promedio de una agrupación.

[ ]:
```
# Estadísticos clásicos para una agrupación completa
grupos.mean()
```

```
<ipython-input-130-d4d5db27aa43>:2: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.mean is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  grupos.mean()
```

[ ]:
```
             price        area  bathrooms   stories   parking  price_per_area
bedrooms
1         2.712500  344.670130   1.000000  1.000000  0.000000     7998.080456
2         3.632022  430.720168   1.058824  1.169118  0.492647     9126.200403
3         4.954598  485.568678   1.266667  1.933333  0.723333    11082.909427
4         5.729758  518.590414   1.621053  2.305263  0.915789    11602.225737
5         5.819800  584.499225   1.800000  2.000000  0.600000    11725.174014
6         4.791500  366.966850   1.500000  2.000000  0.500000    12846.051177
```

[ ]:
```
# Estadística descriptiva básica para un grupo en concreto
grupos_[0].describe()
```

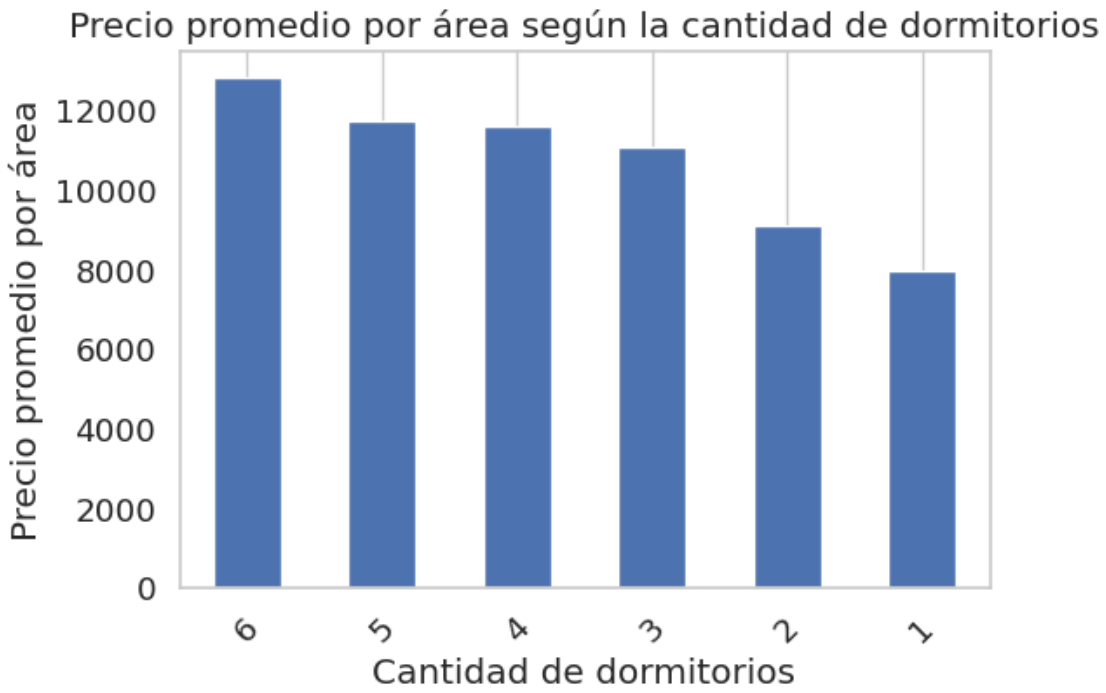
[ ]:
```
          price        area  bedrooms  bathrooms  stories  parking  \
count  2.000000    2.000000       2.0        2.0      2.0      2.0
mean   2.712500  344.670130       1.0        1.0      1.0      0.0
std    0.618718   34.160017       0.0        0.0      0.0      0.0
min    2.275000  320.515350       1.0        1.0      1.0      0.0
25%    2.493750  332.592740       1.0        1.0      1.0      0.0
50%    2.712500  344.670130       1.0        1.0      1.0      0.0
75%    2.931250  356.747520       1.0        1.0      1.0      0.0
max    3.150000  368.824910       1.0        1.0      1.0      0.0
```

```
       price_per_area
count        2.000000
mean      7998.080456
std       2587.787348
min       6168.238474
25%       7083.159465
50%       7998.080456
75%       8913.001447
max       9827.922438
```

- En promedio, ¿qué número de habitaciones posee el precio por área mayor?

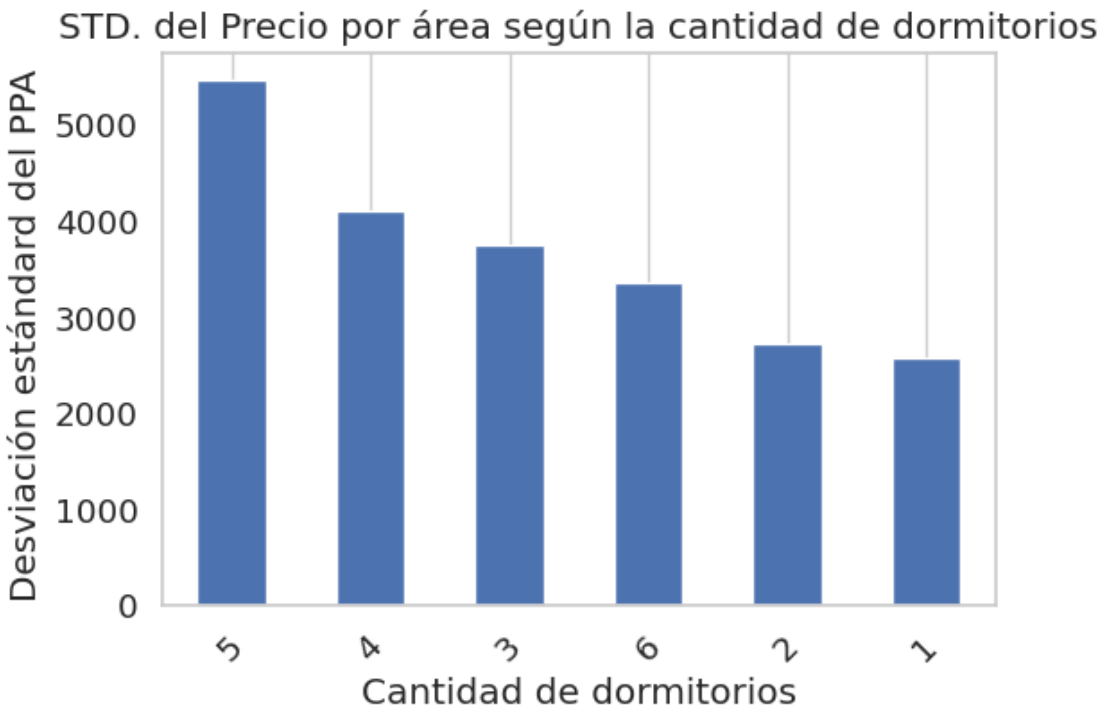
[ ]:
```
bed_promedios = df_house.groupby('bedrooms').mean()['price_per_area']
jerarquia = bed_promedios.sort_values(ascending=False)

# Trazar el gráfico de barras
plt.figure(figsize=(6, 4))
jerarquia.plot(kind='bar')
plt.title('Precio promedio por área según la cantidad de dormitorios')
plt.xlabel('Cantidad de dormitorios')
plt.ylabel('Precio promedio por área')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()
```

```
<ipython-input-137-83a331b50b8a>:1: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.mean is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  bed_promedios = df_house.groupby('bedrooms').mean()['price_per_area']
```



[ ]:
```
bed_promedios = df_house.groupby('bedrooms').std()['price_per_area']
jerarquia = bed_promedios.sort_values(ascending=False)

# Trazar el gráfico de barras
plt.figure(figsize=(6, 4))
jerarquia.plot(kind='bar')
plt.title('STD. del Precio por área según la cantidad de dormitorios')
plt.xlabel('Cantidad de dormitorios')
plt.ylabel('Desviación estándard del PPA ')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()
```

```
<ipython-input-140-cdbf3d59e57e>:1: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.std is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  bed_promedios = df_house.groupby('bedrooms').std()['price_per_area']
```



Para complementar los análisis, es útil realizar agrupaciones dentro de las agrupaciones. Para ello, es posible emplear el método .agg(). Por ejemplo, realicemos una consulta que responda la siguiente pregunta:

- ¿Cuál es el menor precio y la menor área para cada número de habitaciones?

[ ]:
```
# Agregación múltiple
grupos = df_house.groupby('bedrooms')
variables_de_interes = {'price': 'mean', 'area': 'min'}
sub_grupos = grupos.agg(variables_de_interes)

sub_grupos
```

[ ]:

bedrooms	price	area
1	2.712500	320.515350
2	3.632022	170.569908
3	4.954598	153.289950
4	5.729758	199.276935
5	5.819800	176.980215
6	4.791500	334.450800

En Pandas, también es posible realizar consultas anidadas. Intentemos contestar la siguiente pregunta: * ¿Cuál es el mínimo y máximo del área en conjunto con el promedio y la mediana por número de habitaciones?

[ ]:
```
# Agregación anidada
consulta = {'area' : ['min' , 'max'], 'price': ['mean','median']}
consulta_anidada = grupos.agg(consulta)

consulta_anidada
```

[ ]:

	area		price	
	min	max	mean	median
bedrooms				
1	320.515350	368.82491	2.712500	2.7125
2	170.569908	1226.31960	3.632022	3.5350
3	153.289950	1449.28680	4.954598	4.6200
4	199.276935	1123.19727	5.729758	5.2500
5	176.980215	1505.02860	5.819800	5.5825
6	334.450800	399.48290	4.791500	4.7915

Finalmente, realicemos consultas a nuestros datos mediante agrupaciones con respecto a más de una variable.

[ ]:
```
# Agrupaciones con base a múltiples variables
variables_a_agrupar = ['prefarea', 'basement', 'furnishingstatus']
grupos = df_house.groupby(variables_a_agrupar)['price_per_area'].mean()

grupos
```

[ ]:
```
prefarea  basement  furnishingstatus
no        no        furnished           10728.360467
                    semi-furnished      10555.088069
                    unfurnished          8858.815155
          yes       furnished           12085.437995
                    semi-furnished      11903.502608
                    unfurnished         11166.249744
yes       no        furnished           12269.997326
                    semi-furnished      11629.476970
                    unfurnished         10527.165706
          yes       furnished           10998.387379
                    semi-furnished      11317.063656
                    unfurnished         11907.723658
Name: price_per_area, dtype: float64
```

[ ]:
```
# Agrupaciones con base a múltiples variables
variables_a_agrupar = ['prefarea', 'basement', 'furnishingstatus']
grupos_info = df_house.groupby(variables_a_agrupar)['price_per_area'].describe()

grupos_info
```

[ ]:

prefarea	basement	furnishingstatus	count	mean	std
no	no	furnished	66.0	10728.360467	3891.292558
		semi-furnished	116.0	10555.088069	3418.241496
		unfurnished	114.0	8858.815155	2610.258630
	yes	furnished	31.0	12085.437995	4480.732996
		semi-furnished	59.0	11903.502608	4019.072977
		unfurnished	31.0	11166.249744	4247.967768
yes	no	furnished	17.0	12269.997326	4190.025878
		semi-furnished	25.0	11629.476970	3703.266007
		unfurnished	16.0	10527.165706	4590.907998
	yes	furnished	26.0	10998.387379	3244.171038
		semi-furnished	27.0	11317.063656	3639.200077
		unfurnished	17.0	11907.723658	3665.559163

prefarea	basement	furnishingstatus	min	25%	50%
no	no	furnished	4088.618895	8182.833603	10521.448981
		semi-furnished	3767.370268	8266.848551	10131.155799
		unfurnished	2910.514731	7161.054131	8677.704690
	yes	furnished	5651.055402	8407.005544	11530.096504
		semi-furnished	5070.105781	9636.693947	11463.899711
		unfurnished	5409.557308	7811.241126	10729.080375
yes	no	furnished	4820.649072	10116.086831	11906.311696
		semi-furnished	4472.843079	9477.661051	11892.332146
		unfurnished	3965.652914	8605.635933	10462.799297
	yes	furnished	6742.497124	8326.393603	10105.993564
		semi-furnished	6909.040928	8913.989469	9328.133325
		unfurnished	6727.446907	9405.715244	10604.449643

prefarea	basement	furnishingstatus	75%	max
no	no	furnished	12762.777893	28416.735735
		semi-furnished	12391.892955	20290.312091
		unfurnished	10438.755117	17645.762379
	yes	furnished	14747.709312	20996.703304
		semi-furnished	13186.968477	24522.515130
		unfurnished	12808.005837	21131.915986
yes	no	furnished	14246.358156	19293.809997
		semi-furnished	13913.583376	19044.610729
		unfurnished	11380.240926	21076.197303
	yes	furnished	12845.486410	17530.829647
		semi-furnished	13389.561452	20566.289304
		unfurnished	14145.681466	19671.117483

Para finalizar, recordemos los siguientes métodos:

- `df[ [lista de variables] ]` : Filtra el dataframe de acuerdo a un grupo de variables.

- `df.sort_values(by = variable, ascending = True/False)` : Organiza el dataframe deacuerdo a los valores de una variable en forma ascendente o descendente. Por defecto, los ordena de manera ascendente.
- `df[inicio, final, paso]` : Selecciona aquellos regístros del dataframe a partir del regístro `inicio` hasta `final` con condición de paso `paso`.
- `df.loc[condiciones]` : Permite seleccionar regístros del dataframe a partir de un conjunto de condiciones tales como nombres de variables, índices, condiciones de fíltro o combinaciones de estas.

Teniendo en cuenta estos métodos, busquemos el inmueble que cumple las siguientes características: * Es el inmueble más económico dentro de un área preferencial de la ciudad, que posee sótano, amoblamiento completo y su valor está entre el 25% y 50% del valor de aquellos inmuebles que cumplen las anteriores características nombradas.

[ ]:
```
# Encontremos la mejor casa al mejor precio posible
cond1 = df_house['prefarea'] == 'yes'
cond2 = df_house['basement'] == 'yes'
cond3 = df_house['furnishingstatus'] == 'furnished'
cond4 = df_house['price_per_area'] > 8326
cond5 = df_house['price_per_area'] < 10105

(
    df_house
    .loc[cond1 & cond2 & cond3 & cond4 & cond5,
         ['price', 'area','bedrooms', 'bathrooms']]
    .sort_values(by = 'price',
                 ascending = True)
    .head(10)
)
```

[ ]:

	price	area	bedrooms	bathrooms
249	4.543	463.58597	4	2
164	5.390	623.37913	3	2
155	5.530	566.70830	3	2
139	5.740	590.86308	3	1
118	5.950	596.43726	3	1
120	5.950	607.58562	3	1

9.1 # Créditos

Docente: Nicolás Castillo Ojeda

Universidad Pedagógica y Tecnológica de Colombia - *Diplomado en Data Science - Cohorte I - 2024*
