No description has been provided for this image

Actividad Grupal – Análisis e Interpretación de Datos

Profesor: Gustavo Fontana Riera

Fecha de entrega: 05/05/2025

Alumnos:
- Erika Samara Alvares Angelim
- Laila
- Rocío
- Joel
- Pepe

Instalar las bibliotecas necesarias

In [36]:
```python
import sys
import subprocess
from importlib.metadata import version, PackageNotFoundError

# Lista de librerías requeridas
required_libraries = [
    'nbconvert',
    'notebook-as-pdf', 'PyPDF2==2.12.1', 'pyppeteer',
    'adjustText', 'weasyprint', 'dask-expr', 'nbconvert[webpdf] pyppeteer',
    'playwright'

]

# Lista de dependencias del sistema necesarias para Pandoc y exportación a PDF
system_dependencies = [
    'texlive-xetex', 'texlive-fonts-recommended', 'pandoc'
]

# Función para instalar dependencias del sistema en Google Colab
def install_system_dependencies(dependencies):
    try:
        print(("Instalando dependencias del sistema necesarias para "
        "Pandoc y nbconvert..."))
        subprocess.check_call(['apt-get', 'update'], stdout=subprocess.DEVNULL,
                              stderr=subprocess.DEVNULL)
        subprocess.check_call(['apt-get', 'install', '-y'] + dependencies,
                              stdout=subprocess.DEVNULL,
                              stderr=subprocess.DEVNULL)
        print("Instalación de dependencias del sistema completada.")
    except subprocess.CalledProcessError as e:
        print(f"Error instalando dependencias del sistema: {e}")
        sys.exit("No se pudieron instalar las dependencias del sistema. "\
         "Verifique los permisos o la configuración de su entorno.")


# Función para instalar librerías de Python faltantes
def install_libraries(libraries):
    for lib in libraries:
        try:
            print(f"Instalando {lib}...")
            subprocess.check_call([sys.executable, '-m', 'pip', 'install', lib])
        except Exception as e:
            print(f"Error instalando {lib}: {e}")

# Verificar qué librerías están faltando
missing_libraries = []
for lib in required_libraries:
    try:
        # Verificar si ya está instalada
        version(lib.split('==')[0])  # Separar si hay versión específica
        print(f"{lib} ya está instalado, versión: \
        {version(lib.split('==')[0])}")
    except PackageNotFoundError:
        print(f"{lib} no está instalado.")
        missing_libraries.append(lib)

# Resolver conflictos de versiones para urllib3 si pyppeteer está en la lista
if 'pyppeteer' in missing_libraries or 'pyppeteer' in required_libraries:
    try:
        print("Resolviendo conflictos con urllib3 para pyppeteer...")
        subprocess.check_call([sys.executable, '-m', 'pip', 'uninstall',
                               'urllib3', '-y'])
        subprocess.check_call([sys.executable, '-m', 'pip', 'install',
                               'urllib3<2.0.0'])
        print("Versión de urllib3 ajustada para compatibilidad con pyppeteer.")
    except Exception as e:
        print(f"Error ajustando urllib3: {e}")

# Instalar dependencias del sistema en Google Colab
install_system_dependencies(system_dependencies)

# Instalar librerías faltantes
if missing_libraries:
    install_libraries(missing_libraries)
else:
    print("Todas las librerías requeridas ya están instaladas.")

# Imprimir las versiones de las librerías instaladas
print("\nVersiones de las librerías instaladas:")
for lib in required_libraries:
    try:
        print(f"{lib}: {version(lib.split('==')[0])}")
    except PackageNotFoundError:
        print(f"{lib}: No instalado")

# Verificar todas las librerías instaladas
#print("\nListado completo de librerías instaladas:")
#!pip list
```

```
nbconvert ya está instalado, versión:         7.16.6
notebook-as-pdf ya está instalado, versión:         0.5.0
PyPDF2==2.12.1 ya está instalado, versión:         2.12.1
pyppeteer ya está instalado, versión:         2.0.0
adjustText ya está instalado, versión:         1.3.0
weasyprint ya está instalado, versión:         65.1
dask-expr ya está instalado, versión:         1.1.21
nbconvert[webpdf] pyppeteer no está instalado.
playwright ya está instalado, versión:         1.52.0
Resolviendo conflictos con urllib3 para pyppeteer...
Versión de urllib3 ajustada para compatibilidad con pyppeteer.
Instalando dependencias del sistema necesarias para Pandoc y nbconvert...
Instalación de dependencias del sistema completada.
Instalando nbconvert[webpdf] pyppeteer...
Error instalando nbconvert[webpdf] pyppeteer: Command '['/usr/bin/python3', '-m', 'pip', 'install',
'nbconvert[webpdf] pyppeteer']' returned non-zero exit status 1.

Versiones de las librerías instaladas:
nbconvert: 7.16.6
notebook-as-pdf: 0.5.0
PyPDF2==2.12.1: 2.12.1
pyppeteer: 2.0.0
adjustText: 1.3.0
weasyprint: 65.1
dask-expr: 1.1.21
nbconvert[webpdf] pyppeteer: No instalado
playwright: 1.52.0
```

Importar librerías

In [37]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import json
```

Configurar el entorno

In [38]:
```python
import warnings
warnings.filterwarnings("ignore")

# Mostrar todas las columnas sin truncamiento
pd.set_option('display.max_columns', None)       # Muestra todas las columnas
pd.set_option('display.max_rows', None)          # Muestra todas las filas

pd.options.display.float_format = '{:.8f}'.format
#pd.set_option('display.width', 1000)


# Ocultar warnings
warnings.filterwarnings('ignore')

# Ajustar el ancho del contenedor de Jupyter/Colab:
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))


plt.style.use('ggplot')

#plt.style.use('fivethirtyeight')
#plt.rcParams['figure.figsize'] = (10, 6)
```

In [39]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/
drive", force_remount=True).
```

Configurar Drive

In [40]:
```python
%%time
# Montar el drive para obtener el csv
from google.colab import drive

# Monta Google Drive
drive.mount('/content/drive')

# Acceder al archivo CSV proporcionando la ruta correcta desde tu Google Drive
# file_path = '/content/drive/MyDrive/Curso_IA_SkillUp_IBM/creditcard.csv'
file_path = '/content/drive/MyDrive/Cursos/UNIR - BigDataAI/proyectos/bases-de-datos-para-e-BigData
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/
drive", force_remount=True).
CPU times: user 14 ms, sys: 1.9 ms, total: 15.9 ms
Wall time: 962 ms
```

Carga los Datos

Carga los datasets de Airbnb de Madrid

In [41]:
```python
url_listings_madrid = "https://data.insideairbnb.com/spain/comunidad-de-madrid/madrid/2024-12-12/da
url_calendar_madrid = "https://data.insideairbnb.com/spain/comunidad-de-madrid/madrid/2024-12-12/da
url_reviews_madrid = "https://data.insideairbnb.com/spain/comunidad-de-madrid/madrid/2024-12-12/dat
url_neighbournhood_madrid = "https://data.insideairbnb.com/spain/comunidad-de-madrid/madrid/2024-12
url_neighbournhood_geojson_madrid = "https://data.insideairbnb.com/spain/comunidad-de-madrid/madrid

# Low_memory=False: Carga el archivo completo antes de decidir los tipos de datos de cada columna.
listings_madrid = pd.read_csv(url_listings_madrid, compression='gzip', low_memory=False)
calendar_madrid = pd.read_csv(url_calendar_madrid, compression='gzip', low_memory=False)
reviews_madrid = pd.read_csv(url_reviews_madrid, compression='gzip', low_memory=False)
neighbournhood_madrid = pd.read_csv(url_neighbournhood_madrid, low_memory=False)
neighbournhood_geojson_madrid = pd.read_json(url_neighbournhood_geojson_madrid)


# Carga local de los datasets de Madrid
#listings = pd.read_csv("./datos/madrid/listings.csv.gz", compression='gzip', low_memory=False)
#calendar = pd.read_csv("./datos/madrid/calendar.csv.gz", compression='gzip', low_memory=False)
#reviews = pd.read_csv("./datos/madrid/reviews.csv.gz", compression='gzip', low_memory=False)
#neighbournhood = pd.read_csv("./datos/madrid/neighbourhoods.csv", low_memory=False)
```

Carga los datasets de Airbnb de sevilla

In [ ]:
```python
url_listings_sevilla = "https://data.insideairbnb.com/spain/andaluc%C3%ADa/sevilla/2024-12-31/data/
url_calendar_sevilla = "https://data.insideairbnb.com/spain/andaluc%C3%ADa/sevilla/2024-12-31/data/
url_reviews_sevilla = "https://data.insideairbnb.com/spain/andaluc%C3%ADa/sevilla/2024-12-31/data/r
url_neighbournhood_sevilla = "https://data.insideairbnb.com/spain/andaluc%C3%ADa/sevilla/2024-12-31
url_neighbournhood_geojson_sevilla = "https://data.insideairbnb.com/spain/andaluc%C3%ADa/sevilla/20



listings_sevilla = pd.read_csv(url_listings_sevilla, compression='gzip', low_memory=False)
calendar_sevilla = pd.read_csv(url_calendar_sevilla, compression='gzip', low_memory=False)
reviews_sevilla = pd.read_csv(url_reviews_sevilla, compression='gzip', low_memory=False)
neighbournhood_sevilla = pd.read_csv(url_neighbournhood_sevilla, low_memory=False)
neighbournhood_geojson_sevilla = pd.read_json(url_neighbournhood_geojson_sevilla)


# Carga local de los datasets de Sevilla
#listings_sevilla = pd.read_csv("./datos/sevilla/listings.csv.gz", compression='gzip', low_memory=F
#calendar_sevilla = pd.read_csv("./datos/sevilla/calendar.csv.gz", compression='gzip', low_memory=F
#reviews_sevilla = pd.read_csv("./datos/sevilla/reviews.csv.gz", compression='gzip', low_memory=Fal
#neighbournhood_sevilla = pd.read_csv("./datos/sevilla/neighbourhoods.csv", low_memory=False)
```

ETL

Relacionar los Datasets de Madrid y generar un unico Dataset

In [ ]:
```python
# Paso 1: Agrupa los comentarios por listing_id (si hay muchos comentarios por anuncio)
reviews_summary_madrid = reviews_madrid.groupby('listing_id')['comments'] \
    .apply(lambda x: ' || '.join(x.dropna().astype(str))).reset_index()
reviews_summary_madrid.rename(columns={'comments': 'all_comments'}, inplace=True)

# Paso 2: Agrupa el calendario por listing_id para obtener algunas estadísticas agregadas
calendar_summary_madrid = calendar_madrid.groupby('listing_id').agg({
    'price': lambda x: x.mode().iloc[0] if not x.mode().empty else None,  # Precio más común
    'available': lambda x: x.value_counts().idxmax(),                    # Valor más frecuente de d
    'minimum_nights': 'mean',
    'maximum_nights': 'mean'
}).reset_index()

calendar_summary_madrid.rename(columns={
    'price': 'typical_price',
    'available': 'most_common_availability',
    'minimum_nights': 'avg_minimum_nights',
    'maximum_nights': 'avg_maximum_nights'
}, inplace=True)

# Paso 3: Unir todo por id / listing_id
airbnb_madrid_full = listings_madrid.merge(calendar_summary_madrid, how='left', left_on='id', right
airbnb_madrid_full = airbnb_madrid_full.merge(reviews_summary_madrid, how='left', left_on='id', rig

# Paso 4: Elimina columnas duplicadas tras los merges
airbnb_madrid_full.drop(columns=['listing_id_x', 'listing_id_y'], errors='ignore', inplace=True)

# Agrega la columna ciudad y almacena en este caso como "Madrid"
airbnb_madrid_full['city'] = 'Madrid'
```

Relacionar los Datasets de Sevilla y generar un unico Dataset

In [ ]:
```python
# Paso 1: Agrupa los comentarios por listing_id (si hay muchos comentarios por anuncio)
reviews_summary_sevilla = reviews_madrid.groupby('listing_id')['comments'] \
    .apply(lambda x: ' || '.join(x.dropna().astype(str))).reset_index()
reviews_summary_sevilla.rename(columns={'comments': 'all_comments'}, inplace=True)

# Paso 2: Agrupa el calendario por listing_id para obtener algunas estadísticas agregadas
calendar_summary_sevilla = calendar_sevilla.groupby('listing_id').agg({
    'price': lambda x: x.mode().iloc[0] if not x.mode().empty else None,  # Precio más común
    'available': lambda x: x.value_counts().idxmax(),                    # Valor más frecuente de d
    'minimum_nights': 'mean',
    'maximum_nights': 'mean'
}).reset_index()

calendar_summary_sevilla.rename(columns={
    'price': 'typical_price',
    'available': 'most_common_availability',
    'minimum_nights': 'avg_minimum_nights',
    'maximum_nights': 'avg_maximum_nights'
}, inplace=True)

# Paso 3: Unir todo por id / listing_id
airbnb_sevilla_full = listings_sevilla.merge(calendar_summary_sevilla, how='left', left_on='id', ri
airbnb_sevilla_full = airbnb_sevilla_full.merge(reviews_summary_sevilla, how='left', left_on='id',

# Paso 4: Elimina columnas duplicadas tras los merges
airbnb_sevilla_full.drop(columns=['listing_id_x', 'listing_id_y'], errors='ignore', inplace=True)

# Agrega la columna ciudad y almacena en este caso como "Sevilla"
airbnb_sevilla_full['city'] = 'Sevilla'
```

UNIÓN FINAL DE MADRID + SEVILLA

In [ ]:
```python
# Unir los dos dataframes completos
airbnb_full = pd.concat([airbnb_madrid_full, airbnb_sevilla_full], ignore_index=True)

# Comprobar dimensiones y columnas comunes
print(airbnb_full.shape)
print(airbnb_full['city'].value_counts())
```

Seleccionar variables objetivos

- city
- price
- room_type
- bedrooms
- review_scores_rating (valoracion)
- neighbourhood (barrio)

In [ ]:
```python
# Creamos un nuevo daframe solamente con las variables objetivo y renombramos las columnas review_s
airbnb_variables_objetivos = airbnb_full[[
    'city',
    'price',
    'room_type',
    'bedrooms',
    'review_scores_rating',
    'neighbourhood_cleansed'
]].rename(columns={
    'review_scores_rating': 'valoracion',
    'neighbourhood_cleansed': 'neighbourhood'
}).copy()  # .copy() evita problemas de referencia

# Resetear el indice
airbnb_variables_objetivos = airbnb_variables_objetivos.reset_index(drop=True)

# Verificar las primeras filas
airbnb_variables_objetivos.head()
```

Verificar tipos de datos

In [ ]:
```python
airbnb_variables_objetivos.info()
```

Tratar los tipos de datos

In [ ]:
```python
# Eliminar símbolos no numéricos y convertir a float
airbnb_variables_objetivos['price'] = (
    airbnb_variables_objetivos['price']
    .astype(str)  # Asegurar que sea string
    .str.replace('[$,]', '', regex=True)  # Eliminar "$" y comas
    .astype(float)  # Convertir a float
)
```

In [ ]:
```python
# Filtra las filas donde el precio es negativo
precios_negativos = airbnb_variables_objetivos[airbnb_variables_objetivos['price'] < 0]

# Muestra las primeras filas para revisar los resultados
print(precios_negativos)
```

Verifica si hay valores nulos (faltantes)

In [ ]:
```python
# Verificar datos faltantes

# Función para tratar valorees nulos
def porcentaje_nulos_totales(df):
    total_celdas = df.size  # Filas * Columnas del DataFrame recibido
    total_nulos = df.isnull().sum().sum()
    return (total_nulos / total_celdas) * 100


# Porcentaje total de valores nulos (usando el DataFrame correcto)
porcentaje_valores_nulos = porcentaje_nulos_totales(airbnb_variables_objetivos)

# Imprimir resultados (corregido)
print(f"""
Airbnb Variables Objetivos:

-----------------------------------------------------------
Suma de valores nulos por columna:
-----------------------------------------------------------
{airbnb_variables_objetivos.isnull().sum()}

-----------------------------------------------------------
Porcentaje de valores nulos por columna:
-----------------------------------------------------------
{((airbnb_variables_objetivos.isnull().sum() / len(airbnb_variables_objetivos)) * 100).round(2).to_

-----------------------------------------------------------
Porcentaje total de valores nulos:
-----------------------------------------------------------
- Porcentaje total de valores nulos: {porcentaje_valores_nulos:.2f}%

-----------------------------------------------------------
Interpretación de Resultados:
-----------------------------------------------------------
Campos críticos (>20% nulos):

Ej: neighborhood_overview (17,875 nulos) → Considerar eliminación o imputación masiva.

Campos con nulos aceptables (<5%):
# Añade aquí columnas de tu dataset con menos del 5% de nulos
Ej: bedrooms ({(2661 / len(airbnb_variables_objetivos)) * 100:.2f}% nulos) → Imputar con la mediana

Columnas 100% nulas:
→ Ej: calendar_updated (34,793 nulos) → Eliminar directamente.

""")
```

Tratar valores nulos

Critérios para imputación de valores nulos:

El nuevo DataFrame airbnb_variables_objetivos_limpio tendrá:

- price: Imputación estocástica (Mediana + Ruído) en grupo (ciudad + barrio + tipo de habitación).
- bedrooms: Imputado con la mediana global (ej: 1 dormitorio).
- valoracion: Imputado con la media global (ej: 4.7) o categoría "Sin valoración".

Imputación Estocástica (Mediana + Ruido) en la variable "price"

Imputa valores nulos en 'price' aplicando una estrategia jerárquica: primero usa la mediana del grupo (ciudad, barrio, tipo de habitación) con ruido estocástico (IQR); si el grupo carece de datos, recurre progresivamente a la mediana ciudad-barrio, ciudad-tipo de habitación o global.

In [ ]:
```python
# Función de imputación de precios
def impute_price(group):
    """
    Imputa valores nulos en 'price' usando:
    - Mediana + ruido del grupo (city, neighbourhood, room_type)
    - Fallback a medianas jerárquicas si el grupo está vacío
    """
    try:
        # Caso 1: Grupo tiene datos -> Imputación estocástica (IQR)
        if group['price'].notna().sum() > 0:
            median = group['price'].median()
            q1 = group['price'].quantile(0.25)
            q3 = group['price'].quantile(0.75)
            iqr = q3 - q1

            # Generar ruido controlado
            if iqr == 0:
                noise = 0
            else:
                noise = np.random.uniform(-0.5 * iqr, 0.5 * iqr)

            # Imputar precio con ruido, asegurando que no sea negativo ni cero
            imputed_price = median + noise
            if imputed_price <= 0:
                imputed_price = median  # Si el precio imputado es 0 o negativo, usar la mediana

            group['price'] = group['price'].fillna(imputed_price)
            return group

        # Caso 2: Grupo vacío -> Fallback jerárquico
        city = group['city'].iloc[0] if group['city'].notna().any() else None
        neighbourhood = group['neighbourhood'].iloc[0] if group['neighbourhood'].notna().any() else
        room_type = group['room_type'].iloc[0] if group['room_type'].notna().any() else None

        # Jerarquía de fallbacks
        fallback_median = None

        # 1. city + neighbourhood
        if city and neighbourhood:
            mask = (airbnb_variables_objetivos['city'] == city) & \
                   (airbnb_variables_objetivos['neighbourhood'] == neighbourhood)
            fallback_median = airbnb_variables_objetivos.loc[mask, 'price'].median()

        # 2. city + room_type
        if pd.isna(fallback_median) and city and room_type:
            mask = (airbnb_variables_objetivos['city'] == city) & \
                   (airbnb_variables_objetivos['room_type'] == room_type)
            fallback_median = airbnb_variables_objetivos.loc[mask, 'price'].median()

        # 3. Mediana global
        if pd.isna(fallback_median):
            fallback_median = airbnb_variables_objetivos['price'].median()

        group['price'] = fallback_median
        return group

    except Exception as e:
        print(f"Error en grupo {group.name}: {str(e)}")
        return group


########################################
# EJECUCIÓN Y VALIDACIÓN
########################################

# 1. Resetear índice del DataFrame original (crítico para groupby)
airbnb_variables_objetivos = airbnb_variables_objetivos.reset_index(drop=True)

# 2. Aplicar imputación y guardar en un nuevo dataframe
airbnb_variables_objetivos_limpio = (
    airbnb_variables_objetivos
    .groupby(['city', 'neighbourhood', 'room_type'], group_keys=False)  # Agrupar por estas columna
    .apply(impute_price)  # Aplicar la función de imputación
    .reset_index(drop=True)  # Resetear los índices para tener un dataframe limpio
)

# Ahora, el dataframe limpio con los valores imputados está en airbnb_variables_objetivos_limpio
```

In [ ]:
```python
# Filtra las filas donde el precio es negativo
precios_negativos = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos['price'] < 0]

# Muestra las primeras filas para revisar los resultados
print(precios_negativos)
```

Imputación de valores en las variables "bedrooms" y "valoracion"

In [ ]:
```python
# 1. Imputar 'bedrooms' (mediana global)
# --------------------------------------
median_bedrooms = airbnb_variables_objetivos_limpio['bedrooms'].median()
airbnb_variables_objetivos_limpio['bedrooms'] = airbnb_variables_objetivos_limpio['bedrooms'].filln

# 2. Imputar 'valoracion' (media global o categoría)
# -------------------------------------------------
mean_valoracion = airbnb_variables_objetivos_limpio['valoracion'].mean()
airbnb_variables_objetivos_limpio['valoracion'] = airbnb_variables_objetivos_limpio['valoracion'].f

# Opcional: Si prefieres categoría "Sin valoración"
#airbnb_variables_objetivos_limpio['valoracion'] = airbnb_variables_objetivos_limpio['valoracion'].

# 3. Verificar que no queden nulos
# --------------------------------
print("Nulos restantes en el DataFrame limpio:")
print(airbnb_variables_objetivos_limpio.isnull().sum())
```

Verifica la distribución de las variables numéricas después de la imputación

In [ ]:
```python
# Para 'price'
print("\nDistribución de 'price' antes vs después:")
print("Original:", airbnb_variables_objetivos['price'].describe().round(2))
print("Limpio:", airbnb_variables_objetivos_limpio['price'].describe().round(2))

# Para 'valoracion'
print("\nDistribución de 'valoracion' antes vs después:")
print("Original:", airbnb_variables_objetivos['valoracion'].describe().round(2))
print("Limpio:", airbnb_variables_objetivos_limpio['valoracion'].describe().round(2))
```

In [ ]:
```python
# Verificar los valores únicos de la columna 'city'
print(airbnb_variables_objetivos_limpio['city'].unique())
```

Seleccionar solamente alquilleres menores que 1000 usd

Evaluar si realmente tiene sentido seleccionar solamente los airbnb que tengan precio menor que 1000 USD una vez que ya apliqué el Rango Intercuartílico (IRQ) en la fase de limpieza y tratamiento de los datos en las celdas anteriores. Caso no sea necesario bastaba comentar la linea.

In [ ]:
```python
#airbnb_variables_objetivos_limpio = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos_l
airbnb_variables_objetivos_limpio.shape
```

Mini EDA

Estadistíca Descriptiva General

In [ ]:
```python
airbnb_variables_objetivos_limpio.describe().round(2)
```

Estadisticas Descriptiva por Ciudad y Valoración General

In [ ]:
```python
#Estadisticas Descriptiva por Ciudad
airbnb_variables_objetivos_limpio.groupby('city')['price'].describe().round(2)
```

Estadistica Descriptiva por Ciudad y Barrio

In [ ]:
```python
airbnb_variables_objetivos_limpio.groupby(['city', 'neighbourhood'])['price'].describe().round(2)
```

Estadística descriptiva del precio

Obtener resumen estadístico de la columna "price"

In [ ]:
```python
#print("Estadística descriptiva de precios:")
airbnb_variables_objetivos_limpio['price'].describe().round(2)
```

Análisis de Precios por Tipo de Alojamiento en Sevilla y Madrid

In [ ]:
```python
# Calcular estadísticas por tipo de alojamiento para Sevilla y Madrid
grouped_sevilla = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos_limpio['city'] == 'S
grouped_madrid = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos_limpio['city'] == 'Ma

# Redondear los valores a dos decimales
grouped_sevilla = grouped_sevilla.round({'mean': 2, 'median': 2, 'std': 2})
grouped_madrid = grouped_madrid.round({'mean': 2, 'median': 2, 'std': 2})

# Mostrar resultados
print("\nEstadísticas de precios por tipo de alojamiento en Sevilla:")
print(grouped_sevilla)

print("\nEstadísticas de precios por tipo de alojamiento en Madrid:")
print(grouped_madrid)
```

In [ ]:
```python
# Filtra las filas donde el precio es negativo
precios_negativos = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos_limpio['price'] <

# Muestra las primeras filas para revisar los resultados
print(precios_negativos)
```

Cálculo de la correlación entre el número de habitaciones y el precio en Sevilla y Madrid

In [ ]:
```python
# Matriz de correlación entre precio y número de habitaciones para Sevilla y Madrid
# Filtramos los datos por ciudad y calculamos la correlación entre 'price' y 'bedrooms' para cada c

# Filtrar los datos para Sevilla y Madrid
df_sevilla = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos_limpio['city'] == 'Sevill
df_madrid = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos_limpio['city'] == 'Madrid'

# Correlación para Sevilla
correlation_sevilla = df_sevilla[['price', 'bedrooms']].corr().loc['bedrooms', 'price']
print(f"\nCorrelación entre número de habitaciones y precio en Sevilla: {correlation_sevilla:.2f}")

# Correlación para Madrid
correlation_madrid = df_madrid[['price', 'bedrooms']].corr().loc['bedrooms', 'price']
print(f"Correlación entre número de habitaciones y precio en Madrid: {correlation_madrid:.2f}")
```

Tabla de correlación

¿Qué revelan estas tablas?

- Diferencias entre Ciudades:
- ¿La relación precio-valoración es igual en ambas ciudades?
- ¿Cómo afecta el número de dormitorios al precio en cada ubicación?

Patrones Ocultos:

- ¿Los tipos de habitación impactan diferente en cada ciudad?
- ¿Existen correlaciones contraintuitivas?

In [ ]:
```python
# Filtrar datos por ciudad
df_madrid = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos_limpio['city'] == 'Madrid'
df_sevilla = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos_limpio['city'] == 'Sevill

# Función para procesar cada ciudad
def calcular_correlaciones_ciudad(df, nombre_ciudad):
    # Preparar datos y codificar room_type
    df_encoded = pd.get_dummies(df[['price', 'bedrooms', 'room_type', 'valoracion']],
                               columns=['room_type'])

    # Calcular matriz de correlación
    corr_matrix = df_encoded.corr(numeric_only=True).round(2)

    # Correlación específica valoracion-price
    corr_val_precio = corr_matrix.loc['valoracion', 'price']

    # Imprimir resultados
    print(f"\n□ {nombre_ciudad} □")
    print(f"Correlación Valoración-Precio: {corr_val_precio:.2f}")
    print("\nMatriz de Correlación Completa:")

    return corr_matrix.style.background_gradient(cmap='coolwarm', vmin=-1, vmax=1)\
                     .set_caption(f"Tabla: {nombre_ciudad}")\
                     .format("{:.2f}", na_rep="-")

# Calcular y mostrar para Madrid
tabla_madrid = calcular_correlaciones_ciudad(df_madrid, "Madrid")
display(tabla_madrid)

# Calcular y mostrar para Sevilla
tabla_sevilla = calcular_correlaciones_ciudad(df_sevilla, "Sevilla")
display(tabla_sevilla)
```

Análisis Tabla de Correlación Madrid

Interpretación Técnica de la Matriz de Correlación (Madrid):

La matriz de correlación muestra las relaciones lineales entre las variables analizadas en los alojamientos de Airbnb en Madrid. A continuación, se destacan los hallazgos clave:

---

1. Relación Precio-Valoración

- **Correlación: 0.01**
  - **Interpretación:** No existe una relación lineal significativa entre el precio y la valoración.
  - **Implicación:** Los precios más altos *no garantizan* mejores valoraciones, y viceversa.
  - **Posible causa:** Factores como la ubicación, servicios adicionales o atención del anfitrión podrían influir más en las valoraciones que el precio.

---

2. Precio vs Tipo de Habitación

- **Entire home/apt (Correlación: 0.11)**
  - Las propiedades completas tienen una ligera tendencia a ser más caras.
- **Private room (Correlación: -0.11)**
  - Las habitaciones privadas muestran una correlación negativa débil con el precio.
- **Relación inversa (Entire home/apt vs Private room: -0.96)**
  - **Interpretación:** Existe una correlación negativa casi perfecta entre estos tipos de habitación, lo que confirma que son categorías mutuamente excluyentes (un alojamiento no puede ser ambos).

---

3. Dormitorios vs Precio

- **Correlación: 0.10**
  - **Hallazgo:** Aunque positiva, la relación entre el número de dormitorios y el precio es muy débil.
  - **Insight:** En Madrid, el tamaño de la propiedad (medido en dormitorios) *no es un factor determinante* en el precio. Podrían influir más características como la ubicación o el diseño.

---

4. Valoración vs Otras Variables

- **Todas las correlaciones cercanas a 0** (entre -0.04 y 0.03).
  - **Conclusión:** La valoración no está relacionada linealmente con:
    - Precio.
    - Número de dormitorios.
    - Tipo de habitación.
  - **Recomendación:** Investigar variables no analizadas (ej: limpieza, comunicación del anfitrión, amenities) para entender los drivers de las valoraciones.

---

5. Relaciones entre Tipos de Habitación

- **Entire home/apt vs Private room: -0.96**
  - **Validación técnica:** Confirma que las variables dummy están bien construidas (un alojamiento solo pertenece a una categoría).

Conclusiones para el Informe:

1. **Estrategia de precios:** En Madrid, el tipo de habitación (Entire home/apt) justifica precios ligeramente más altos, pero no existen diferencias significativas basadas en dormitorios.
2. **Valoraciones:** Se requieren análisis cualitativos (ej: texto de reseñas) para identificar factores no cuantificados en esta matriz.
3. **Segmentación:** La alta correlación inversa entre tipos de habitación sugiere que los huéspedes eligen entre "propiedades completas" o "habitaciones privadas", no ambas.

Nota: Correlación ≠ Causalidad. Se recomienda complementar con análisis de regresión para identificar predictores significativos del precio.

Este análisis proporciona una base cuantitativa para decisiones estratégicas en pricing y mejora de la experiencia del huésped en Madrid.

Análisis Tabla de Correlación Sevilla

Interpretación Técnica de la Matriz de Correlación (Sevilla):

La matriz de correlación revela las relaciones lineales entre variables clave en los alojamientos de Airbnb en Sevilla. A continuación, se desglosan los hallazgos principales:

---

1. Relación Precio-Valoración

- **Correlación: -0.01**
  - **Interpretación:** Al igual que en Madrid, no existe una relación lineal significativa entre precio y valoración.
  - **Implicación:** Los precios altos no garantizan mejores valoraciones, y las valoraciones altas no están vinculadas a precios elevados.
  - **Posible causa:** Factores no analizados (ej: calidad del servicio, ubicación exacta, limpieza) podrían influir más en las valoraciones.

---

2. Precio vs Tipo de Habitación

- **Entire home/apt (Correlación: 0.06)**
  - Las propiedades completas tienen una correlación positiva muy débil con el precio, similar a Madrid pero menos pronunciada.
- **Private room (Correlación: -0.06)**
  - Ligera tendencia a precios más bajos en habitaciones privadas.
- **Relación inversa (Entire home/apt vs Private room: -0.98)**
  - **Interpretación:** Correlación negativa casi perfecta, confirmando la exclusividad mutua de las categorías (un alojamiento no puede ser ambas cosas).

---

3. Dormitorios vs Precio

- **Correlación: 0.10**
  - **Hallazgo:** Igual que en Madrid, el número de dormitorios tiene un impacto mínimo en el precio.
  - **Insight:** En Sevilla, el tamaño de la propiedad no es un driver principal de precio. La ubicación o amenities podrían ser más relevantes.

---

4. Valoración vs Otras Variables

- **Todas las correlaciones cercanas a 0** (entre -0.04 y 0.07).
  - **Conclusión:** La valoración no está ligada linealmente a:
    - Precio.
    - Dormitorios.
    - Tipo de habitación.
  - **Recomendación:** Analizar variables cualitativas (ej: comentarios de reseñas) para identificar drivers ocultos.

---

5. Relaciones entre Tipos de Habitación

- **Entire home/apt vs Private room: -0.98**
  - **Validación técnica:** Coherencia con la codificación dummy, donde un alojamiento solo puede pertenecer a una categoría.

Diferencias Clave vs Madrid

1. **Dormitorios y Entire home/apt (Correlación: 0.20 en Sevilla vs 0.18 en Madrid):**
   - En Sevilla, las propiedades completas tienden ligeramente a tener más dormitorios, pero la relación sigue siendo débil.
2. **Valoración y Dormitorios (Correlación: 0.07 en Sevilla vs -0.01 en Madrid):**
   - Aunque insignificante, sugiere que en Sevilla *podría* haber una mínima tendencia a valorar mejor propiedades con más dormitorios.

Conclusiones para el Informe:

1. **Estrategia de precios:** En Sevilla, el tipo de habitación influye menos en el precio que en Madrid. Las propiedades completas no justifican precios significativamente más altos.
2. **Valoraciones:** Al igual que en Madrid, se requieren análisis no lineales o cualitativos para entender los motivos detrás de las puntuaciones.
3. **Segmentación de mercado:** La elección entre "propiedad completa" y "habitación privada" es igualmente excluyente en ambas ciudades, pero en Sevilla la correlación precio-tipo es aún más tenue.

Recomendación final: Realizar un estudio comparativo entre ciudades para identificar si las diferencias en correlaciones reflejan variaciones culturales o de mercado.

Este análisis proporciona una base cuantitativa para optimizar estrategias comerciales en Sevilla, destacando la necesidad de enfoques complementarios para entender las valoraciones.

Visualizaciones

Grafico Comparativo

Distribución del precio por tipo de alojamiento y ciudad

In [ ]:
```python
# Filtra los precios que son menores a 1000 euros
airbnb_variables_objetivos_limpio_filtrado = airbnb_variables_objetivos_limpio[airbnb_variables_obj

# Grafico comparativo de distribución del precio por alojamiento y ciudad
sns.boxplot(data=airbnb_variables_objetivos_limpio_filtrado, x='room_type', y='price', hue='city')
plt.title = ('Distribución del precio por tipo de alojamiento y ciudad')
plt.xticks(rotation=15)
plt.savefig('boxplot_filtrado.png')
plt.show()
```

Estadistica Robusta (Pepe)

Aunque ya se aplicó una limpieza de precios mediante el rango intercuartílico (IQR), se ha decidido conservar esta gráfica como herramienta **visual de validación**. Su objetivo es comparar medidas robustas (mediana e IQR) frente a medidas sensibles a valores extremos (media y desviación estándar) para verificar si los precios altos que persisten son esperables, especialmente en ciudades como Madrid o Sevilla.

No se utilizará esta gráfica como criterio adicional de filtrado, sino como **respaldo visual y exploratorio** del comportamiento de los precios por ciudad.

In [ ]:
```python
# 1. Filtrar los precios menores a 1000
airbnb_variables_objetivos_limpio_filtrado = airbnb_variables_objetivos_limpio[airbnb_variables_obj

# 2. Agrupa por ciudad y calcula estadísticas básicas
stats = airbnb_variables_objetivos_limpio_filtrado.groupby('city')['price'].agg(['mean', 'std', 'me

# 3. Calcula los cuartiles y el rango intercuartílico (IQR)
q1 = airbnb_variables_objetivos_limpio_filtrado.groupby('city')['price'].quantile(0.25)
q3 = airbnb_variables_objetivos_limpio_filtrado.groupby('city')['price'].quantile(0.75)
stats['iqr'] = q3 - q1

# 4. Prepara las posiciones para las barras
cities = stats.index.tolist()
x = range(len(cities))
width = 0.35

# 5. Dibuja barras con barras de error
fig, ax = plt.subplots(figsize=(8, 5))
ax.bar([i - width/2 for i in x],
       stats['mean'],
       width,
       yerr=stats['std'],
       capsize=5,
       label='Media ± Desviación estándar',
       color='skyblue')
ax.bar([i + width/2 for i in x],
       stats['median'],
       width,
       yerr=stats['iqr'],
       capsize=5,
       label='Mediana ± IQR',
       color='lightgreen')

# 6. Ajustes finales
ax.set_xticks(x)
ax.set_xticklabels(cities, rotation=45)
ax.set_title('Estadísticas Robusta vs No Robusta (Precio por ciudad)')
ax.set_ylabel('Precio (€)')
ax.legend()
plt.tight_layout()
plt.savefig('EstadisticaRVSnR.png')
plt.show()
```

Análisis de Correlación entre Variables de Alojamiento en Sevilla y Madrid

In [ ]:
```python
# Filtrar los datos para Sevilla y Madrid
sevilla_df = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos_limpio['city'] == 'Sevill
madrid_df = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos_limpio['city'] == 'Madrid'

# Crear matriz de correlación para Sevilla
corr_sevilla = sevilla_df[['price', 'bedrooms', 'valoracion']].corr()

# Crear matriz de correlación para Madrid
corr_madrid = madrid_df[['price', 'bedrooms', 'valoracion']].corr()



# Crear una figura con dos subgráficas (2 filas, 1 columna)
plt.figure(figsize=(16, 6))

# Primer gráfico: Correlación Sevilla
plt.subplot(1, 2, 1)  # 1 fila, 2 columnas, 1er gráfico
sns.heatmap(corr_sevilla,
            annot=True,
            cmap='coolwarm',
            vmin=-1,
            vmax=1)
plt.gca().set_title("Correlación entre variables - Sevilla")

# Segundo gráfico: Correlación Madrid
plt.subplot(1, 2, 2)  # 1 fila, 2 columnas, 2do gráfico
sns.heatmap(corr_madrid,
            annot=True,
            cmap='coolwarm',
            vmin=-1,
            vmax=1)
plt.gca().set_title("Correlación entre variables - Madrid")

# Ajustar el espacio entre los gráficos
plt.tight_layout()

# Mostrar las gráficas
plt.show()

#### Identificar pares de variables con alta correlación (umbral > 0.7 o < -0.7)

# Filtrar pares con alta correlación (umbral > 0.7 o < -0.7)
# Madrid
correlated_pairs_madrid = corr_madrid.abs().unstack().sort_values(ascending=False)
high_correlation_pairs_madrid = correlated_pairs_madrid[(correlated_pairs_madrid > 0.7) & (correlat
print(high_correlation_pairs_madrid)

#Sevilla
correlated_pairs_sevilla = corr_sevilla.abs().unstack().sort_values(ascending=False)
high_correlation_pairs_sevilla = correlated_pairs_sevilla[(correlated_pairs_sevilla > 0.7) & (corre
print(high_correlation_pairs_sevilla)
```

Visualizar la distribución de variables clave en el dataset (airbnb_variables_objetivos_limpio)

In [ ]:
```python
# Eliminar variable title si existe
if 'title' in globals():
    del title

# ===========================================================================
# ANÁLISIS PARA MADRID
# ===========================================================================
data_madrid = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos_limpio['city'] == 'Madri

# Configuración de estilo
sns.set_theme(style="whitegrid", palette="pastel")
fig, axs = plt.subplots(2, 2, figsize=(16, 12))
fig.suptitle('Análisis de Propiedades en Madrid', fontsize=18, fontweight='bold', color='#2C3E50')

# 1. Distribución de Precios por Tipo de Habitación
sns.histplot(data=data_madrid,
             x='price',
             hue='room_type',
             bins=30,
             ax=axs[0, 0])
axs[0, 0].set_title('¿Cómo se distribuyen los precios entre tipos de habitación?\n¿Existen diferenc
                   fontsize=12, pad=10)
axs[0, 0].set_xlabel('Precio (€)', fontsize=10)
axs[0, 0].set_ylabel('Frecuencia', fontsize=10)
axs[0, 0].set_xlim(0, 300)

# 2. Relación Precio-Valoración
sns.scatterplot(data=data_madrid,
                x='valoracion',
                y='price',
                hue='room_type',
                alpha=0.7,
                ax=axs[0, 1])
axs[0, 1].set_title('¿Las propiedades más caras tienen mejor valoración?\n¿Varía esta relación por
                   fontsize=12, pad=10)
axs[0, 1].set_xlabel('Valoración (1-5)', fontsize=10)
axs[0, 1].set_ylabel('Precio (€)', fontsize=10)

# 3. Distribución de Dormitorios
sns.histplot(data=data_madrid,
             x='bedrooms',
             bins=15,
             kde=True,
             ax=axs[1, 0])
axs[1, 0].set_title('¿Cuál es el tamaño más común de las propiedades?\n¿Cómo se distribuyen los dor
                   fontsize=12, pad=10)
axs[1, 0].set_xlabel('Dormitorios', fontsize=10)
axs[1, 0].set_ylabel('Frecuencia', fontsize=10)
axs[1, 0].set_xticks(range(0, 11))

# 4. Precios en Barrios Top
top_barrios_madrid = data_madrid['neighbourhood'].value_counts().index[:5]
sns.boxplot(data=data_madrid[data_madrid['neighbourhood'].isin(top_barrios_madrid)],
            x='neighbourhood',
            y='price',
            ax=axs[1, 1])
axs[1, 1].set_title('¿Qué barrios tienen precios más estables?\n¿Dónde encontramos outliers signifi
                   fontsize=12, pad=10)
axs[1, 1].set_xlabel('')
axs[1, 1].set_ylabel('Precio (€)', fontsize=10)
axs[1, 1].tick_params(axis='x', rotation=45)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

# ===========================================================================
# ANÁLISIS PARA SEVILLA
# ===========================================================================
data_sevilla = airbnb_variables_objetivos_limpio[airbnb_variables_objetivos_limpio['city'] == 'Sevi

fig, axs = plt.subplots(2, 2, figsize=(16, 12))
fig.suptitle('Análisis de Propiedades en Sevilla', fontsize=18, fontweight='bold', color='#2C3E50')

# 1. Distribución de Precios por Tipo de Habitación (Sevilla)
sns.histplot(data=data_sevilla,
             x='price',
             hue='room_type',
             bins=30,
             ax=axs[0, 0])
axs[0, 0].set_title('¿Cómo comparan los precios con Madrid?\n¿Qué tipo de habitación domina el merc
                   fontsize=12, pad=10)
axs[0, 0].set_xlabel('Precio (€)', fontsize=10)
axs[0, 0].set_ylabel('Frecuencia', fontsize=10)
axs[0, 0].set_xlim(0, 300)

# 2. Relación Precio-Valoración (Sevilla)
sns.scatterplot(data=data_sevilla,
                x='valoracion',
                y='price',
                hue='room_type',
                alpha=0.7,
                ax=axs[0, 1])
axs[0, 1].set_title('¿Se mantiene la relación precio-valoración?\n¿Hay diferencias con Madrid?',
                   fontsize=12, pad=10)
axs[0, 1].set_xlabel('Valoración (1-5)', fontsize=10)
axs[0, 1].set_ylabel('Precio (€)', fontsize=10)

# 3. Distribución de Dormitorios (Sevilla)
sns.histplot(data=data_sevilla,
             x='bedrooms',
             bins=15,
             kde=True,
             ax=axs[1, 0])
axs[1, 0].set_title('¿El tamaño de las propiedades es diferente?\n¿Qué distribución de dormitorios
                   fontsize=12, pad=10)
axs[1, 0].set_xlabel('Dormitorios', fontsize=10)
axs[1, 0].set_ylabel('Frecuencia', fontsize=10)
axs[1, 0].set_xticks(range(0, 11))

# 4. Precios en Barrios Top (Sevilla)
top_barrios_sevilla = data_sevilla['neighbourhood'].value_counts().index[:5]
sns.boxplot(data=data_sevilla[data_sevilla['neighbourhood'].isin(top_barrios_sevilla)],
            x='neighbourhood',
            y='price',
            ax=axs[1, 1])
axs[1, 1].set_title('¿Cómo se comparan los barrios populares con Madrid?\n¿Patrones de precios dife
                   fontsize=12, pad=10)
axs[1, 1].set_xlabel('')
axs[1, 1].set_ylabel('Precio (€)', fontsize=10)
axs[1, 1].tick_params(axis='x', rotation=45)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

precios medios por tipo de habitación y ciudad

Interpretación potencial:

- ¿Los apartamentos completos son significativamente más caros en Madrid?
- ¿Qué ciudad ofrece mejores precios en habitaciones compartidas?
- ¿Existe mayor variación de precios en algún tipo específico de alojamiento?

In [ ]:
```python
# ---------------------------------------------------------------
# PASO 1: LIMPIEZA DE VARIABLES (EJECUTAR ESTO PRIMERO)
# ---------------------------------------------------------------
import matplotlib.pyplot as plt

# Verificar y eliminar variables conflictivas
conflict_vars = ['title', 'Title', 'TITLE']
for var in conflict_vars:
    if var in globals():
        del globals()[var]
        print(f"Variable conflictiva '{var}' eliminada")

# Reiniciar configuración de matplotlib
plt.close('all')
plt.rcParams.update(plt.rcParamsDefault)

# ---------------------------------------------------------------
# PASO 2: CÓDIGO GRÁFICO CORREGIDO (EJECUTAR DESPUÉS DEL PASO 1)
# ---------------------------------------------------------------
import seaborn as sns

# Calcular precios medios
grouped = airbnb_variables_objetivos_limpio.groupby(['room_type', 'city'])['price'].mean().reset_in

# Configurar figura
fig, ax = plt.subplots(figsize=(12, 6))
sns.set_theme(style="whitegrid", palette="pastel")

# Crear gráfico
ax = sns.barplot(
    x='room_type',
    y='price',
    hue='city',
    data=grouped,
    errorbar=None,
    palette={"Madrid": "#e74c3c", "Sevilla": "#2980b9"},
    saturation=0.85  # Intensidad de color aumentada
)

# Personalización avanzada
ax.set_title(
    'Comparativa de precios medios\nMadrid vs Sevilla',
    fontsize=16,
    pad=20,
    fontweight='bold',
    color='#2c3e50',
    loc='left'  # Alineación izquierda
)

ax.set_xlabel(
    'Tipo de alojamiento',
    fontsize=12,
    labelpad=15,
    fontstyle='italic'
)

ax.set_ylabel(
    'Precio medio ($)',
    fontsize=12,
    labelpad=15,
    fontstyle='italic'
)

# Rotación y formato de ejes
plt.xticks(
    rotation=20,
    fontsize=11,
    fontfamily='DejaVu Sans'  # Fuente más legible
)

# Etiquetas de valor mejoradas
for p in ax.patches:
    ax.text(
        p.get_x() + p.get_width() / 2.,
        p.get_height() + 5,  # Posición vertical ajustada
        f'${p.get_height():.0f}',
        ha='center',
        va='center',
        fontsize=10,
        color='#34495e'
    )

# Leyenda personalizada
leg = ax.legend(
    title='Ciudad',
    title_fontsize='13',
    fontsize=11,
    frameon=True,
    shadow=True,
    facecolor='#f8f9fa'  # Fondo claro
)

# Ajustes finales
plt.tight_layout()
plt.show()
```

Conversión del Dataset a Formato JSON

¿Cómo validar que los archivos .json tienen estructura válida? Para asegurarte de que los archivos .json estén bien estructurados y sean compatibles con MongoDB, debes cumplir con lo siguiente:

Estructura válida JSON Puedes validarlos en línea en herramientas como:

https://jsonlint.com/

https://jsonformatter.org/

Compatible con MongoDB MongoDB puede importar:

- Un JSON por línea (ndjson o JSONL)
- O un archivo con un array de objetos JSON, así:

```
[
  { "nombre": "Juan", "edad": 30 },
  { "nombre": "Ana", "edad": 25 }
]
```

Recomendación: Usa el formato de array de objetos si vas a cargar con mongoimport.

In [ ]:
```python
# Información del dataset limpio
airbnb_variables_objetivos_limpio.info()
```

In [ ]:
```python
# Limitar la cantidad de filas a 10 filas para evitar que los archivos .json sean demasiado grandes
airbnb_madrid_y_servilla_limited = airbnb_variables_objetivos_limpio.head(10)

# Dataframe airbnb_variables_objetivos_limpio completo
airbnb_madrid_y_servilla_completo = airbnb_variables_objetivos_limpio.copy()
```

In [ ]:
```python
# Enviar todo el contenido del Dataframe a un archivo .json (No se limita la cantidad de filas)
#alojamientos_turisticos.to_json('airbnb_variables_objetivos_limpio.json', orient='records', force_

airbnb_madrid_y_servilla_limited.to_json('airbnb_madrid_y_sevilla_limited.json', orient='records',
airbnb_madrid_y_servilla_completo.to_json('airbnb_madrid_y_sevilla_completo.json', orient='records'
```

Exportar cuaderno a formato HTML y a formato PDF

Descarga el archivo generado directamente desde Colab

In [ ]:
```python
import shutil
from google.colab import files

# Función para limpiar y validar el notebook
def fix_notebook_metadata(filepath):
    with open(filepath, 'r', encoding='utf-8') as file:
        notebook = json.load(file)

    for cell in notebook.get('cells', []):
        if 'outputs' in cell:
            for output in cell['outputs']:
                # Agregar 'metadata' si no existe
                if 'metadata' not in output:
                    output['metadata'] = {}

    # Guardar el notebook limpio
    with open(filepath, 'w', encoding='utf-8') as file:
        json.dump(notebook, file, indent=2, ensure_ascii=False)

# Rutas de archivos
notebook_path = "/content/drive/MyDrive/Cursos/UNIR - BigDataAI/proyectos/bases-de-datos-para-e-Big
html_output_path = "/content/drive/MyDrive/Cursos/UNIR - BigDataAI/proyectos/bases-de-datos-para-e-
pdf_output_path_viahtml = "/content/drive/MyDrive/Cursos/UNIR - BigDataAI/proyectos/bases-de-datos-

# Corregir el notebook
print("\nCorrigiendo y validando el notebook...")
fix_notebook_metadata(notebook_path)

# Exportar a HTML
print("\nExportando a formato HTML...")
!jupyter nbconvert --to html --output "$html_output_path" "$notebook_path"

# Exportar a PDF usando PDFviaHTML
print("\nExportando a formato PDF usando nbconvert con PDFviaHTML...")
!jupyter nbconvert --to pdfviahtml --output "$pdf_output_path_viahtml" "$notebook_path"

# Descargar archivos generados
print("\nDescargando los archivos generados...")
try:
    files.download(html_output_path)  # Descargar HTML
    files.download(pdf_output_path_viahtml)  # Descargar PDF generado con PDFviaHTML
except Exception as e:
    print(f"Error descargando los archivos: {e}")

print("\nExportación y descarga completadas.")
```