Práctica 2

Luis García Fresno

Contenido

1. Introducción

Para la realización de esta práctica se van a utilizar varios datasets con el fin de entender hasta qué punto puede afectar a un fenómeno como la gentrificación la aparición de servicios como Airbnb. Para ello, se van a utilizar 3 datasets, de los cuales 2 se han obtenido a través de Kaggle y otro a través del servicio de datos públicos del ayuntamiento de Madrid.

En primer lugar, se va a describir de forma breve la gentrificación para poner en contexto todo este trabajo, siendo la misma un fenómeno que consiste en la remodelación/rehabilitación de zonas de una gran ciudad (principalmente céntricas o con un potencial atractivo turístico) que se traducen en un cambio de la población de dichas zonas, produciéndose un cambio de personas con un menor poder adquisitivo por otras que sí pueden hacer frente al nuevo coste de vida en dichas áreas.

La ciudad elegida para realizar este análisis ha sido Madrid debido a la facilidad con la que se han encontrado los datos en internet, teniendo por un lado un dataset que describe hasta 21.000 viviendas extraídas de los principales portales de compra-venta de inmuebles del país, otro conjunto de datos que hace referencia a casi 20.000 pisos de alquiler que se ofertan en la misma ciudad y, por último, un dataset con los códigos postales y los distritos a los que pertenecen. Además, se ha utilizado como material de apoyo para las tareas de visualización un conjunto de distritos representados como polígonos para poder realizar un mapeo de la ciudad con la idea de que los análisis realizados se puedan visualizar de una forma más clara.

Se deben tener en cuenta algunas consideraciones a la hora de realizar el análisis, puesto que los conjuntos de datos, como se explicará en el punto siguiente, no cuentan con un histórico de datos y hacen referencia al año 2020 y 2021 respectivamente según la información que se ha podido extraer a través de Kaggle. Por tanto, la idea no es interpretar el efecto a lo largo de los años de la aparición de Airbnb y plataformas similares, si no tratar de entender cómo afectan este tipo de plataformas a la estimación del precio de una vivienda en una zona donde la densidad de ofertas de este tipo es elevada. Para ello se realizarán los análisis oportunos y, finalmente, se mostrarán diversas gráficas que demuestren si la hipótesis es válida o no.

Además de las gráficas, se van a incluir una serie de mapas a partir de la librería Pydeck, utilizada en conjunto con H3, la librería de Uber para mapear el globo terráqueo. La idea de utilizar estas dos herramientas es la de representar de una forma clara la distribución de apartamentos de alquiler a lo largo y ancho de la superficie de la ciudad, así como otros índices que se construirán en base a los datos que aportan los distintos datasets. Además, tratándose de una librería como H3, es necesario obtener la localización de los pisos en venta, por lo que se realizarán y explicarán algunas implementaciones de geocoding para tratar de obtener dicha información.

2. Selección de datos

A la hora de la integración de los datos, como se ha comentado en el primer apartado, vamos a servirnos de 3 datasets como base para poder realizar un dataset final que sea un conjunto de: el listado de casas a la venta en Madrid en el año 2020, el listado de Airbnb de la ciudad de Madrid para el año 2021 y un dataset de apoyo con un desglose de códigos postales y distritos.

Para realizar esta fusión, se ha tomado al dataset con los datos de venta como la base a la que adicionar el resto de información que se ha considerado importante para el desarrollo de la práctica.

Para ello, se ha seguido un proceso de unión horizontal, añadiendo como columnas la densidad de pisos listados en Airbnb por código postal y el propio código postal de las viviendas listadas en el dataset de ventas. Para la obtención del código postal se emplearon en un primer momento técnicas de codificación geográfica, haciendo uso de la API de Open Street Maps y, posteriormente, de la API de Google Maps. No obstante, la información obtenida en base a este método ofrecía unos resultados pobres ya que la información en las direcciones de las casas, así como la propia descripción de los distritos en el dataset de ventas, no eran todo lo precisas que habría sido necesario.

Como alternativa, se ha recurrido a un dataset con la información de todos los distritos de la ciudad, disponible en el portal de datos abiertos de la Comunidad de Madrid. De este modo, ha sido posible la extracción del código postal de un mayor número de distritos. La extracción del código postal se ha realizado también sobre el dataset de Airbnb, obteniendo de esta forma la densidad de pisos listados por distrito. Finalmente, se ha adicionado la columna al dataset de ventas utilizando como índice el código postal.

Respecto a los datos del dataset de ventas, como la idea es tratar de discernir la importancia que pueden tener los Airbnb listados en una ciudad sobre el precio de venta de los pisos en esa misma ciudad, la idea es quedarnos con todas aquellas columnas que puedan tener relación directa con la estimación del precio, es decir, metros cuadrados, número de habitaciones, etc.

Para ello, se ha realizado una selección de entre las 54 columnas disponibles en dicho dataset para, posteriormente, utilizar la información que ha pasado este primer corte para continuar con el desarrollo de la práctica.

3. Valores nulos y outliers

Si hablamos de valores nulos, se han realizado 3 tratamientos distintos en función del tipo de variable. Se ha seguido el criterio que se muestra en la siguiente foto:

Para valores cuantitativos, variables continuas, se han rellenado los valores nulos con la media, con la idea de que no se altere el conjunto de la muestra sin perder además las filas que no cuenten con valores para dichas columnas. En este caso, se ha añadido la media del valor medio por metro cuadrado y la media de metros cuadrados útiles.

Para variables con valores binarios, 1 y nulo, se ha asumido que nulo significa 0, pues no hay valores 0. Por tanto, se ha sustituido el valor nulo por 0 quedando la columna completa.

Por último, para valores donde sí existen 1, 0 y el valor nulo, se han sustraído las filas cuyos valores para dichas columnas sean nulos. Como se han filtrado previamente las variables, sabemos que el porcentaje de nulos no es elevado y por tanto podemos asumir esa pérdida de información.

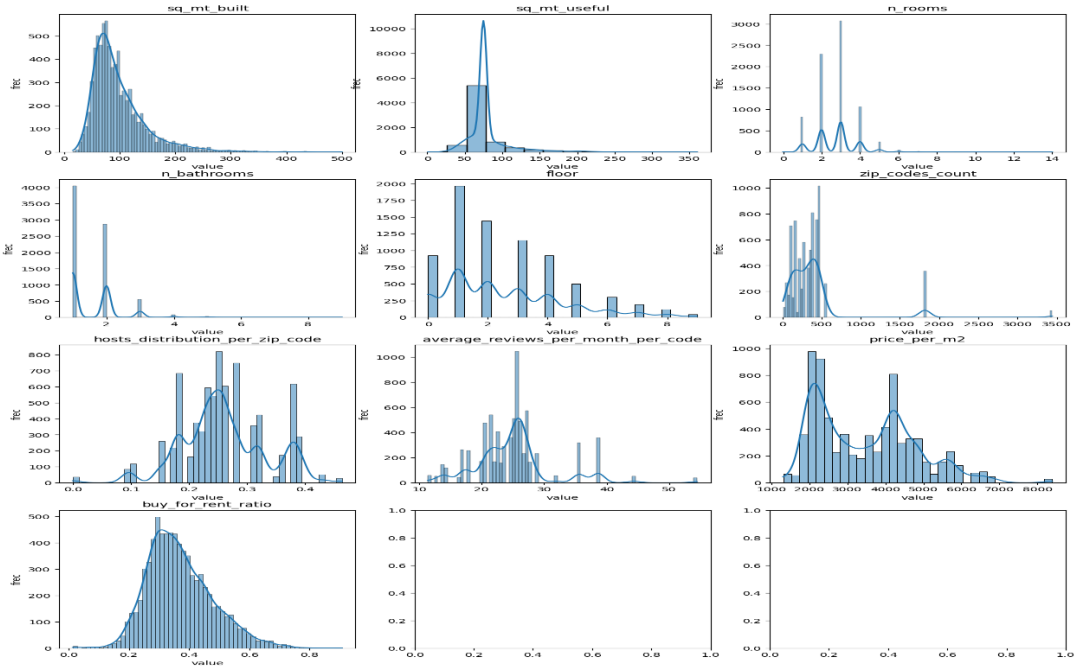
También se han quitado las columnas relacionadas con la calefacción, pues el porcentaje de valores nulos era elevado y supondría reducir la muestra de manera considerable.

Además de tratar los valores nulos, se ha realizado un tratamiento de los valores relacionados con la columna que indica la planta del piso listado. Al tratarse de una variable categórica y ordinal, se ha planteado que el bajo sea representado como la planta 0 (con la idea de realizar, posteriormente, una transformación Box-Cox si fuera necesaria). Para ello, se han extraído los valores que se encuentran almacenados en la columna 'floor', creado un diccionario que haga referencia a los valores a sustituir. Se ha considerado la entreplanta como 1 debido a que páginas como Idealista considera que es una planta intermedia.

A continuación, se muestra una imagen en la que se encuentra la función desarrollada para mostrar las distribuciones de las variables no binarias. De esta forma, se espera entender el tipo de distribución que tenemos entre manos.

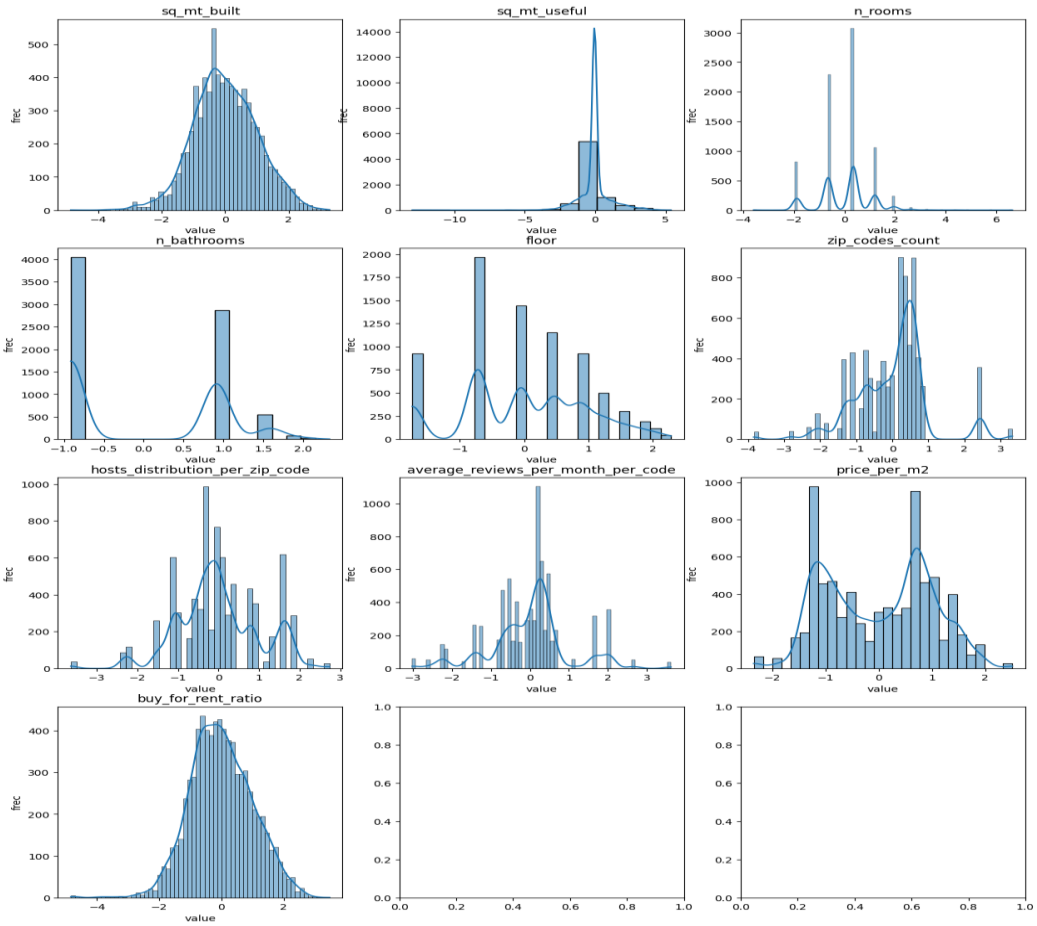
```python
def plot_dist(df: pd.DataFrame) -> None:

    col = df.columns
    row_number = (len(col) // 3) + (1 if len(col) % 3 != 0 else 0)
    col_number = 3

    # Subplots
    fig, axes = plt.subplots(row_number, 3, figsize=(15, 5*row_number))
    # fig.subplots_adjust(hspace=0.5)

    # Iter through cols
    for i, columna in enumerate(col):
        fila = i // col_number
        columna_subplot = i % col_number

        if row_number > 1:
            ax = axes[fila, columna_subplot]
        else:
            ax = axes[columna_subplot]

        # Histogram
        sns.histplot(data=df, x=columna, kde=True, ax=ax)
        ax.set_xlabel('value')
        ax.set_ylabel('frec')
        ax.set_title(f'{columna}')

    plt.show()
```

Como podemos ver en la imagen que se mostrará a continuación, las distintas variables que se van a analizar no siguen una distribución normal. Se les aplicará a todas ellas una transformación *box-cox*, pues tras realizar un análisis previo mediante Shapiro-Wilk, vemos que el p-valor de todas las variables se encuentra por debajo del umbral y por tanto no podemos decir que las variables sigan una distribución normal.



Teniendo en cuenta todo lo anterior, vamos se va a realizar un tratamiento de valores atípicos a partir de dos conjuntos de datos, uno transformado mediante box-cox y otro sin transformar. Posteriormente se utilizarán ambos conjuntos para realizar un análisis de todas las variables del conjunto: análisis de correlación, correlación con la variable objetivo, regresión con la variable objetivo, regresor xgboost y regresor mediante una pequeña red neuronal.

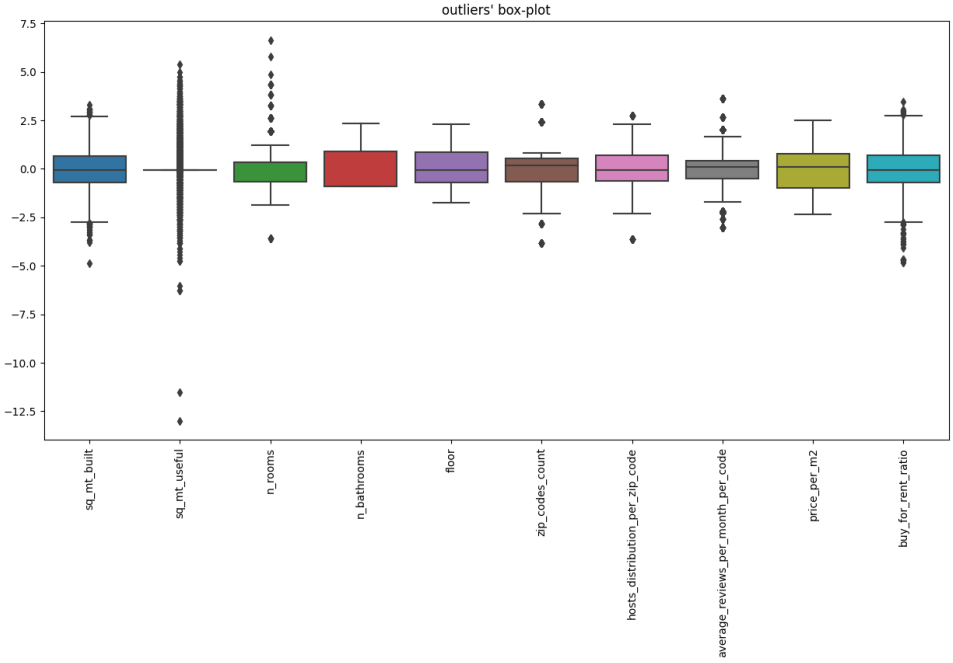
Tras realizar la transformación, las variables tienen la siguiente distribución:

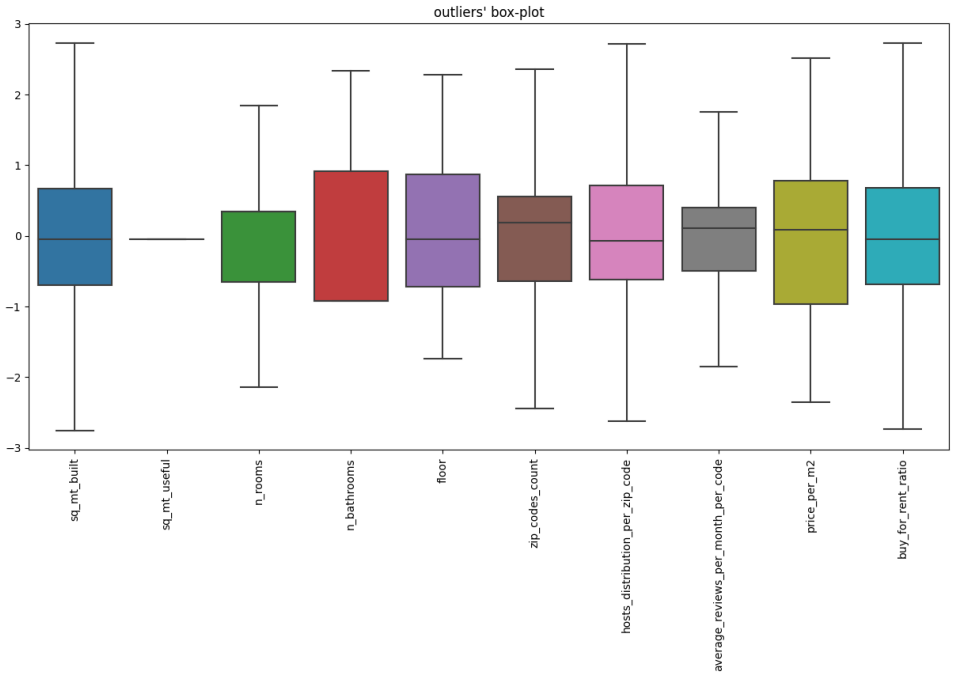


Como vemos, la normalidad de la distribución mejora ligeramente, pero al tener también variables categóricas en general la normalidad no mejora en exceso.

Si nos fijamos en las siguientes imágenes, veremos el análisis de valores extremos, haciendo un análisis basado en cuartiles y en el rango intercuartílico para eliminar los valores extremos. En el notebook de jupyter de valores nulos y valores extremos, al final, se encuentran las siguientes gráficas y el código asociado. La idea es eliminar todos aquellos valores menores al primer cuartil menos 1.5 veces el rango intercuartílico y todos aquellos valores mayores que el tercer cuartil más 1.5 veces el rango intercuartílico.

Obtenemos de esta forma unos box-plot como los siguientes, siendo el primero una representación de los conjuntos de datos a tratar junto con sus correspondientes valores atípicos, mientras que en la segunda gráfica obtenemos los datos limpios, sin dichos valores atípicos. En las imágenes que se muestran, se representan los datos tras la transformación box-cox, pues estando transformados (y algo más normalizados) su representación resulta mucho más clara que para los datos sin transformar.


outliers' box-plot
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
sq_mt_built
sq_mt_useful
n_rooms
n_bathrooms
floor
zip_codes_count
hosts_distribution_per_zip_code
average_reviews_per_month_per_code
price_per_m2
buy_for_rent_ratio


outliers' box-plot
3
2
1
0
−1
−2
−3
sq_mt_built
sq_mt_useful
n_rooms
n_bathrooms
floor
zip_codes_count
hosts_distribution_per_zip_code
average_reviews_per_month_per_code
price_per_m2
buy_for_rent_ratio

4. Análisis

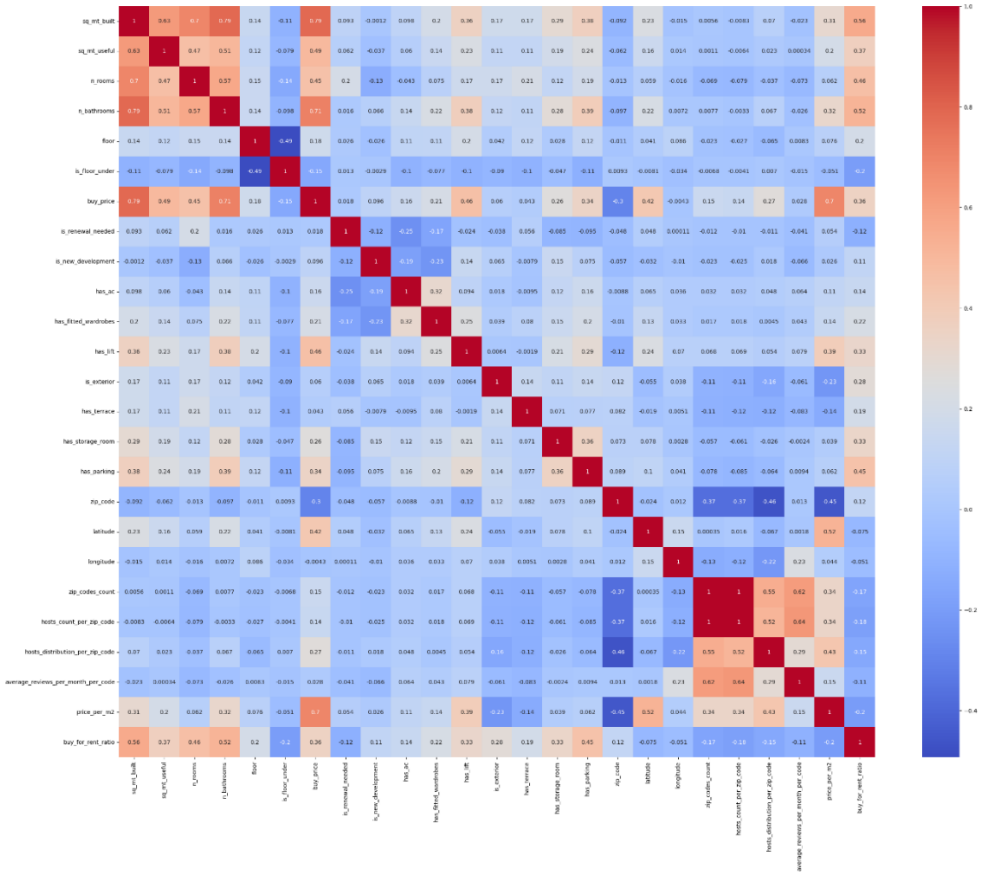
Para realizar el análisis se van a tener en cuenta todos los datos seleccionados desde el apartado 2 de esta práctica, siendo una selección final de varios valores del dataset principal (real estate) así como del dataset complementario (Airbnb). La idea del análisis será la de entender cómo afectarían los pisos de Airbnb al precio de compra de la vivienda en las distintas áreas de Madrid, si es que estos tuviesen algún efecto.

Para ello, se comentará brevemente los resultados del test de Shapiro-Wilk, que como se puede observar en la imagen que se muestra a continuación nos dice que la muestra no está normalmente distribuida a pesar de haber realizado una transformación Box-cox.

	Test Statistic	p-value
sq_mt_built	0.997767	4.361937e-09
sq_mt_useful	0.783560	0.000000e+00
n_rooms	0.902998	0.000000e+00
n_bathrooms	0.715638	0.000000e+00
floor	0.948434	1.401298e-45
zip_codes_count	0.926994	0.000000e+00
hosts_distribution_per_zip_code	0.970963	1.588867e-36
average_reviews_per_month_per_code	0.939428	0.000000e+00
price_per_m2	0.955541	5.114739e-43
buy_for_rent_ratio	0.994196	5.631184e-17

Por otra parte, si nos fijamos en los resultados de la regresión, vemos que los resultados de la muestra para las variables a analizar están altamente dispersos en la mayoría de los casos.

Haciendo ahora un análisis de correlación entre las distintas variables vemos lo siguiente:



Las variables que más correlación muestran entre sí son las relacionadas con el conteo de viviendas por código postal y el conteo de dueños de esos pisos por código postal. Se procede a eliminar una, tratando de evitar así la multicolinearidad.

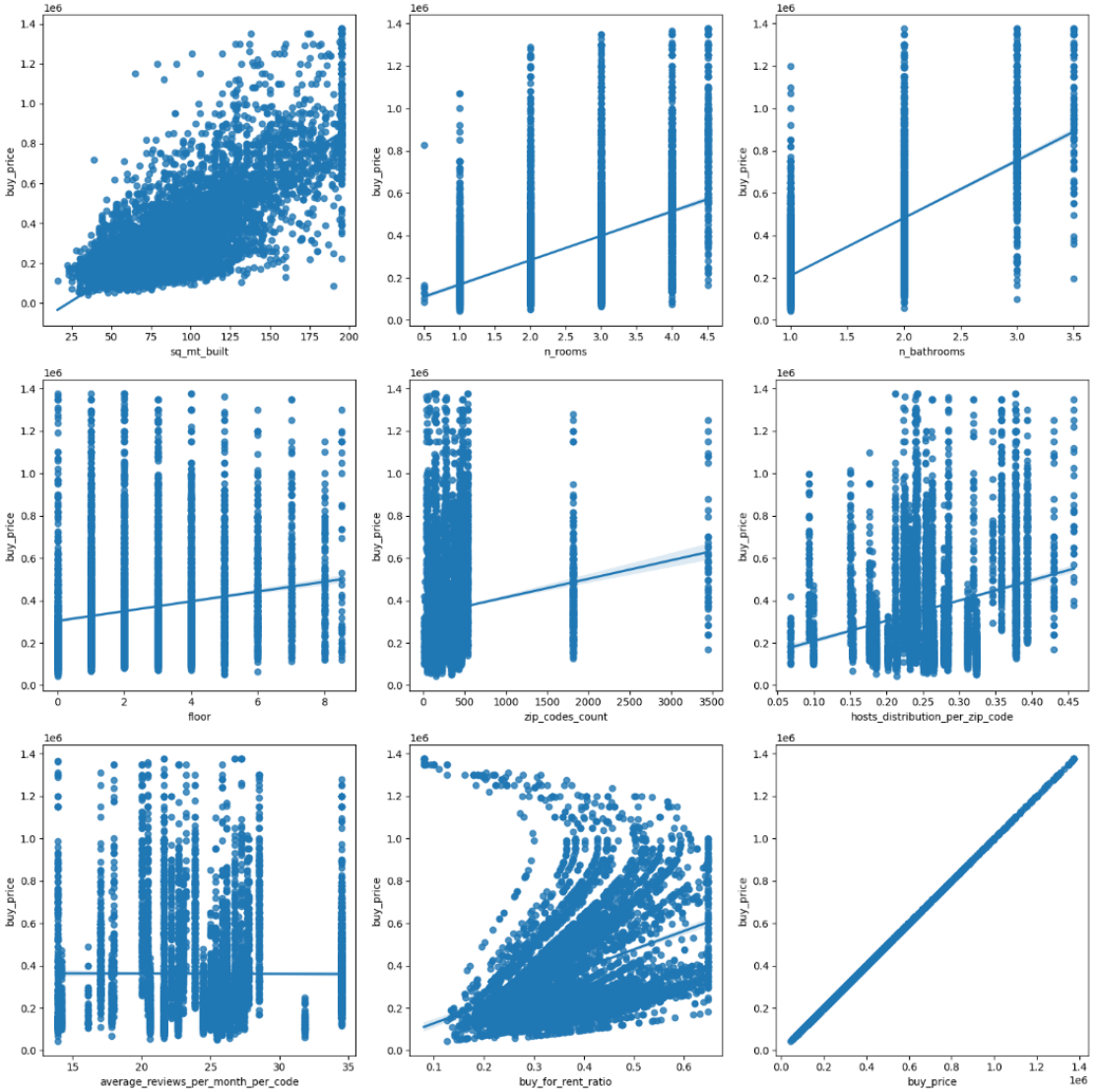
Realizamos ahora un análisis de la correlación entre las variables independientes y nuestra variable objetivo, que a partir de ahora será el precio de la vivienda. Vemos en un primer acercamiento que las variables que más colinearidad tienen con el precio de la vivienda son: precio por metro cuadrado, número de baños por vivienda, precio por metro cuadrado, etc. La primera variable generada a partir de los datos de Airbnb es la distribución de tenedores por código postal, siendo este un índice que se ha desarrollado para entender cómo de distribuidas están las viviendas en una zona determinada, tratando de representar de manera aproximada la posible especulación que puede aparecer en una zona concreta, siendo 0 un área perfectamente distribuida y 1 un área altamente especulativa (todas las viviendas corresponderían a 1 único tenedor).

Nota: estas variables se han reducido, eliminando los metros cuadrados útiles al tratarse de una métrica similar a los metros construidos que no aportaba en exceso, el precio por metro cuadrado debido a que es fácil estimar el precio de la vivienda contando con el precio por metro cuadrado, latitud, longitud y código postal, pues se interpreta que los valores como la distribución por área es información suficiente como para poder estimar de manera razonable el precio de la vivienda.

```
buy_price                              1.000000
sq_mt_built                            0.787568
n_bathrooms                            0.708707
price_per_m2                           0.695857
sq_mt_useful                           0.493094
has_lift                               0.460857
n_rooms                                0.453064
latitude                               0.417419
buy_for_rent_ratio                     0.358460
has_parking                            0.343892
hosts_distribution_per_zip_code        0.269591
has_storage_room                       0.260814
has_fitted_wardrobes                   0.209220
floor                                  0.177487
has_ac                                 0.158938
zip_codes_count                        0.148689
hosts_count_per_zip_code               0.137143
is_new_development                     0.095719
is_exterior                            0.059550
has_terrace                            0.043421
average_reviews_per_month_per_code     0.028381
is_renewal_needed                      0.017620
longitude                             -0.004263
is_floor_under                        -0.151789
zip_code                              -0.299380
Name: buy_price, dtype: float64
```

A continuación, se ha realizado un gráfico para entender la relación lineal que existe entre la variable objetivo y las distintas variables dependientes mediante una simple gráfica de regresiones lineales, como se muestra en el código de la siguiente imagen y la imagen que se muestra a continuación, donde podemos ver que la mayoría de las variables no tienen demasiada relación con el precio a efectos prácticos. Se considera que este análisis es pobre y no es capaz de explicar los datos de manera adecuada, por lo que se ha continuado analizando el conjunto de datos mediante otras técnicas.

```python
def plot_linear_regression(df: pd.DataFrame, target:str)-> None:
    col = df.columns
    row_number = (len(col) // 3) + (1 if len(col) % 3 != 0 else 0)
    col_number = 3

    # Subplots
    fig, axes = plt.subplots(row_number, 3, figsize=(15, 5*row_number))
    # fig.subplots_adjust(hspace=0.5)

    # Iter through cols
    for i, column in enumerate(col):
        fila = i // col_number
        col_subplot = i % col_number

        if row_number > 1:
            ax = axes[fila, col_subplot]
        else:
            ax = axes[col_subplot]

        sns.regplot(x=column, y=target, data=df, ax=ax)
        ax.set_xlabel(column)
        ax.set_ylabel(target)

    # Adjust subplot spacing
    fig.tight_layout()

    plt.show()
```



Para continuar con el análisis, se va a realizar una estandarización de los datos mediante la clase StandardScaler de scikit-learn.

```python
target = prev_df_nobc['buy_price']
prev_df_nobc.drop('buy_price', axis=1, inplace=True)
target_bc = prev_df_bc['buy_price']
prev_df_bc.drop('buy_price', axis=1, inplace=True)
```

Python

```python
# Data division
from sklearn.preprocessing import StandardScaler
train_std_x, test_std_x, train_std_y, test_std_y = train_test_split(prev_df_nobc, target, test_size=0.2, random
scaler = StandardScaler()
cols = prev_df_nobc.columns.to_list()
for col in cols:
    train_std_x[col]=scaler.fit_transform(train_std_x[[col]])
    test_std_x[col]=scaler.transform(test_std_x[[col]])
```

Python

Tras la estandarización, se añaden unos cuantos scores para saber qué tal ha ido el entrenamiento del regresor y el resto de modelos. Estos son el R2, que nos permitirá entender si el modelo es capaz de explicar la varianza que aparece en el conjunto de datos, y el resto son los valores de error absoluto medio y error cuadrático medio, ambos negados (en scikit-learn se muestran así).

Además, usamos la validación cruzada para entrenar nuestro regresor y añadimos los datos estandarizados y los scores seleccionados previamente. Tal y como se muestra en la imagen que encontramos a continuación el modelo se entrena, obteniendo un R2 de en torno al 0.7, es decir, explica la varianza razonablemente bien.

```python
# Regresión Lineal para datos sin transformar
scores = ['r2','neg_mean_absolute_error','neg_mean_squared_error']
kfold = sklearn.model_selection.KFold(n_splits=10)
lin_reg_model = LinearRegression()
lin_reg = sklearn.model_selection.cross_validate(
    lin_reg_model,
    train_std_x,
    train_std_y,
    scoring=scores,
    cv=kfold
)
print(
    f"{lin_reg['fit_time'].mean():.3f}s  R2: "
    f"{lin_reg['test_r2'].mean():.3f}  MAE: "
    f"{lin_reg['test_neg_mean_absolute_error'].mean()*-1:.3f}  RMSE: "
    f"{math.sqrt(lin_reg['test_neg_mean_squared_error'].mean()*-1):.3f}"
)
```

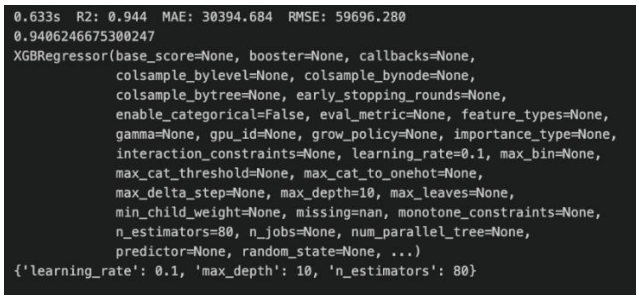
Además del conjunto de datos transformados mediante box-cox, se ha añadido un segundo conjunto, idéntico en sus variables, pero sin transformar, con la idea de ver si los resultados mejoran o empeoran. Se ha probado tanto con el regresor como con el xgboost, pero la diferencia ha sido de apenas una décima para el R2 en ambos casos. No obstante, se puede revisar en el código de la práctica aunque el proceso es exactamente igual salvo que se tienen distintos datos de entrada.

No obstante, la idea es continuar profundizando en el análisis de los datos, por lo que se va a entrenar un xgboostregressor con la idea de ver cuánto podría mejorar nuestro modelo y por tanto cuánto podría mejorar la comprensión de los datos por parte del mismo de manera que podamos sacar unas conclusiones más precisas.

Si analizamos ahora el regresor de xgboost, vemos que se trata de una estructura mucho más compleja, que se basa en un árbol de decisiones. La idea de este entrenamiento es la misma, con la salvedad de que en esta ocasión vamos a utilizar un algoritmo para optimizar los hiper-parámetros del modelo. Estos hiper-parámetros serán la profundidad, los estimadores y la ratio de aprendizaje.

Una vez que se ha entrenado el modelo mediante la optimización de sus hiper-parámetros, se entrena con el conjunto de datos y se realiza un análisis de la importancia de sus parámetros. Para ello se buscó en Google cómo estimar esta importancia de manera sencilla, llegando a varios artículos donde se habla de yellowbrick como una herramienta útil para esta actividad.

```
# xgboost
xgb_model = xgb.XGBRegressor()
xgb_model_ = sklearn.model_selection.cross_validate(
    xgb_model,
    train_std_x,
    train_std_y,
    scoring=scores,
    cv=kfold
)
print(
    f"{xgb_model_['fit_time'].mean():.3f}s  R2: "
    f"{xgb_model_['test_r2'].mean():.3f}  MAE: "
    f"{xgb_model_['test_neg_mean_absolute_error'].mean()*-1:.3f}  RMSE: "
    f"{math.sqrt(xgb_model_['test_neg_mean_squared_error'].mean()*-1):.3f}"
)

# hyperparameters optimization
params = {
    'max_depth': [5,10,20,40],
    'n_estimators': [10,20,40,80],
    'learning_rate': [0.1,0.01]
}
xgrid = GridSearchCV(estimator=xgb.XGBRegressor(),
                     param_grid=params,
                     cv=5,
                     verbose=0)
xgrid.fit(train_std_x, train_std_y)
print(xgrid.best_score_)
print(xgrid.best_estimator_)
print(xgrid.best_params_)
```



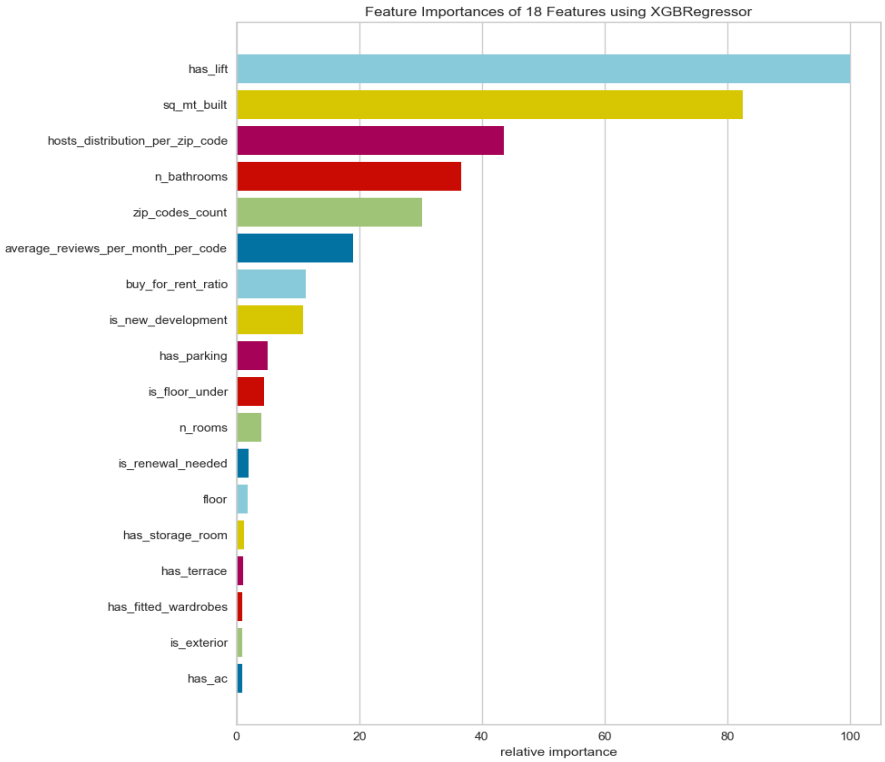
```
0.633s  R2: 0.944  MAE: 30394.684  RMSE: 59696.280
0.9406246675300247
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=10, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=80, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
{'learning_rate': 0.1, 'max_depth': 10, 'n_estimators': 80}
```

```
final_xgb = xgb.XGBRegressor(
    max_depth=xgrid.best_params_['max_depth'],
    n_estimators=xgrid.best_params_['n_estimators']
)
final_xgb.fit(
    train_std_x,
    train_std_y
)
```

Como vemos, el modelo obtiene un R2 mucho mayor, superando el 0.9, lo cual supone una puntuación considerablemente positiva. Por tanto, ahora sólo queda atender a la importancia de los parámetros para tratar de validar o descartar nuestra hipótesis.

Haciendo uso de la librería anteriormente mencionada vemos que el tercer, quinto y sexto valor de nuestro dataset corresponden con información directamente relacionada con los

Airbnb, por lo que en las conclusiones podremos debatir acerca de la importancia relativa de estos pisos de alquiler y este modelo de negocio sobre el precio de compra de los pisos en una gran ciudad (en este caso Madrid).



Por último, sólo quedaba hacer una prueba con una pequeña red como la que se muestra a continuación, donde el número de unidades ocultas será el total de parámetros de entrada de la red, una medida habitual para este tipo de modelos, que además cuenta con 1 capa de entrada, 3 capas ocultas, un dropout y una capa de salida. La idea del dropout es la de reducir el posible overfitting.

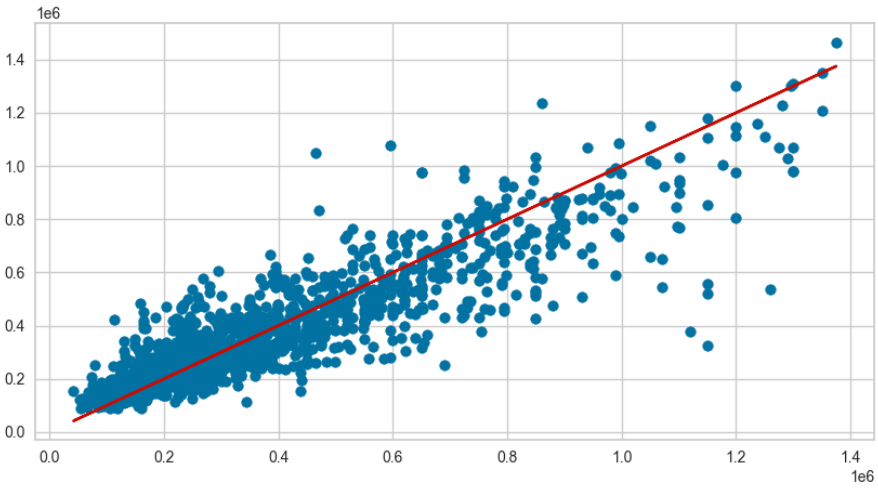
```
# keras model -> lo usaremos en nuestro grid search
def build_model(input_shape=1, hidden_units=30, dropout_rate=0.2):
    model = Sequential()
    model.add(Dense(hidden_units, input_shape=(input_shape,), activation='relu'))
    model.add(Dense(hidden_units, activation='relu'))
    model.add(Dense(hidden_units, activation='relu'))
    model.add(Dense(hidden_units, activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1))
    model.compile(
        loss=tf.keras.losses.MeanSquaredError(reduction="auto", name="mean_squared_error"),
        optimizer=SGD(clipnorm=1.0),

    )
    return model
```

El modelo se ha optimizado mediante un gridsearch, al igual que el xgboost, aunque en este caso se ha buscado una optimización del ratio de dropout y del tamaño del batch. No obstante, se podrían realizar pruebas más complejas, buscando número de capas óptimo, número de neuronas, etc.

Para continuar, el entrenamiento se realiza una vez que se han obtenido los hiper-parámetros óptimos durante un máximo de 500 épocas con un early-stopping que monitoriza la pérdida de validación y, en caso de que no mejore durante un máximo de 10 épocas el entrenamiento se detiene de manera automática (contribuyendo también a evitar el sobreajuste).

```python
# Train data previously generated
x_train = train_std_x_bc
y_train = train_std_y_bc
print(x_train.shape)
print(y_train.shape)

# Earlystopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10)

# Create a KerasRegressor based on the build_model function
keras_regressor = KerasRegressor(
    build_fn=build_model,
    input_shape=x_train.shape[1],
    hidden_units=x_train.shape[1],
    dropout_rate=0.2,
    epochs=10,
    batch_size=32
)

# Define the hyperparameters to search
param_grid = {
    # 'hidden_units': [32, 64, 128],
    'dropout_rate': [0.10, 0.20],
    'batch_size': [32, 64]
    }

# Perform grid search with cross-validation
grid_search = GridSearchCV(
    estimator=keras_regressor,
    param_grid=param_grid,
    cv=3
)
grid_search.fit(x_train, y_train)
```

Por último, se adjunta una gráfica del desempeño del modelo, que tiene en torno a un 0.8 de R2, superior al regresor simple, pero lejos de los resultados obtenidos con el xgboost, algo que tiene sentido al tratarse de un algoritmo mucho más complejo y optimizado. No obstante, vemos que la red tiene un desempeño bastante bueno.



5. Visualización

Para realizar la visualización de los datos, se han añadido diversas gráficas para explicar conceptos como la correlación, las regresiones, la normalidad de las variables no binarias... Sin embargo, pensando en aplicar técnicas de visualización algo más complejas basadas en los datos se planteó el uso de h3-python para mapear la ciudad de Madrid y generar mapas de calor en función de las variables a analizar teniendo en cuenta la localización de las viviendas, distribuidas en pares de latitud-longitud.

La librería h3-python es capaz de describir el globo terráqueo como un conjunto de hexágonos de distintos tamaños, seleccionados por el desarrollador, llegando a una precisión sorprendentemente buena. La librería es código abierto, pero está desarrollada por Uber, y por tanto se enfoca en el mapeo de áreas y el conteo de usuarios en dichas áreas para establecer el precio de sus servicios. No obstante, se puede modificar para utilizar estos hexágonos para mapear una ciudad en función de, por ejemplo, la distribución de propietarios de pisos frente al total de los pisos en esa misma zona o un mapa de calor que cubra toda la ciudad y describa el precio medio por área.

Para utilizar esta librería se han realizado labores de geocodificación, pensando en obtener a partir de las direcciones de las casas del dataset principal (real estate) la latitud y la longitud, información utilizada por h3-python para poder generar un hashcode en el que se hace referencia al área cubierta por el hexágono (resolución) y a sus coordenadas en el mapa de la ciudad. De esta forma, se puede establecer un análisis por niveles de los distintos datos que se deseen visualizar, pensando por ejemplo en una precisión del tamaño de una manzana o buscando visualizar datos a partir de superficies mayores como pueden ser uno o varios distritos.

Para la geocodificación se ha utilizado la API de Google maps, pero como se trata de un servicio de pago y el dataset original contaba con 21.000 direcciones se ha utilizado finalmente la API de OpenStreetMaps, que en detrimento es ostensiblemente más lenta. No obstante se han obtenido un total de 13.000 direcciones con sus respectivas latitudes y longitudes. Se ha decidido utilizar ese dataset de menor dimensión para buscar una mayor coherencia entre el análisis estadístico y este análisis visual.

También se ha utilizado la API de Google Maps y OpenStreetMaps para obtener los códigos postales de los distintos distritos que aparecían en el dataset, buscando de esta manera unir la información extraída del dataset de Airbnb con el dataset principal. No obstante y como se explica en los primeros apartados, finalmente se optó por utilizar un dataset que contenía esa información, ya que había bastante información que se perdía a causa de fallos en la API, que no arrojaba resultados satisfactorios para direcciones que aparentemente eran correctas.

En el código se pueden ver 2 mapas interactivos, uno de ellos muestra la distribución del precio medio por área y el otro la distribución de propietarios frente al número de pisos por área.

6. Conclusiones

La idea de esta práctica era la de analizar el impacto de pisos ofertados para alquiler vacacional o de corta-media estancia en un problema como es la gentrificación en una gran ciudad (Madrid, para este caso de estudio).

Para ello, se han utilizado distintos valores extraídos del dataset de Airbnb para tratar de explicar la condensación de estos pisos en zonas de elevado interés (turístico o de cualquier otro tipo), contribuyendo de esta forma a que haya un menor número de pisos en venta y por tanto el precio de los que sí lo están se vea afectado en forma de incrementos.

La complejidad del problema a analizar supone una gran barrera, pues los Airbnb no son el único factor de influencia en algo tan complejo como el mercado de la vivienda de cualquier país. Sin embargo, y tras los análisis de las variables en el xgboost, vemos que sí que hay varios factores relacionados que tienen un peso a la hora de estimar el precio de una vivienda. En concreto, resulta interesante que el índice que explica la distribución de tenedores por área frente al número de pisos tenga un peso relativamente significativo, pues podemos intuir fácilmente que se trata de una zona marcada por la especulación. No obstante, tratándose de un modelo de regresión, factores intrínsecamente relacionados con la vivienda son los que tienen un peso mayor a la hora de estimar el precio de la misma.

Quedan por tanto en un segundo plano aquellos parámetros introducidos a partir de la información extraída del dataset de Airbnb, siendo un segundo plano relativamente significativo y relativamente esclarecedor en cuanto a esta cuestión, concluyendo finalmente que los pisos de alquiler de corta y media estancia sí que pueden suponer un factor que contribuye a la aparición de la gentrificación en las ciudades.

Fuera del análisis estadístico y basado en los modelos, podemos suponer que la gentrificación es el motor para la aparición de este tipo de negocios, pues los grandes tenedores, respaldados por un gran capital, son los principales beneficiados al convertir el centro de las ciudades en espacios turísticos en todos los niveles posibles. Si se tiene en cuenta, además, la aparición del teletrabajo y la disparidad de salarios entre los distintos países de la Unión Europea, no es de extrañar que el centro de Madrid tenga ahora un gran número de inquilinos que pasan unos pocos meses en la ciudad mientras trabajan, teniendo un sueldo acorde a sus países de origen, que generalmente suelen tener un mayor nivel de vida. Se considera por tanto que esta vía podría ser una buena continuación de esta investigación en todo lo relativo a la gentrificación y los factores que fomentan su expansión.