Inteligencia de Negocio

Práctica 3: Competición en Kaggle

Autor

Mónica Calzado Granados, Grupo 2

monicacg1111@correo.ugr.es

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

1 Tabla Leaderboard

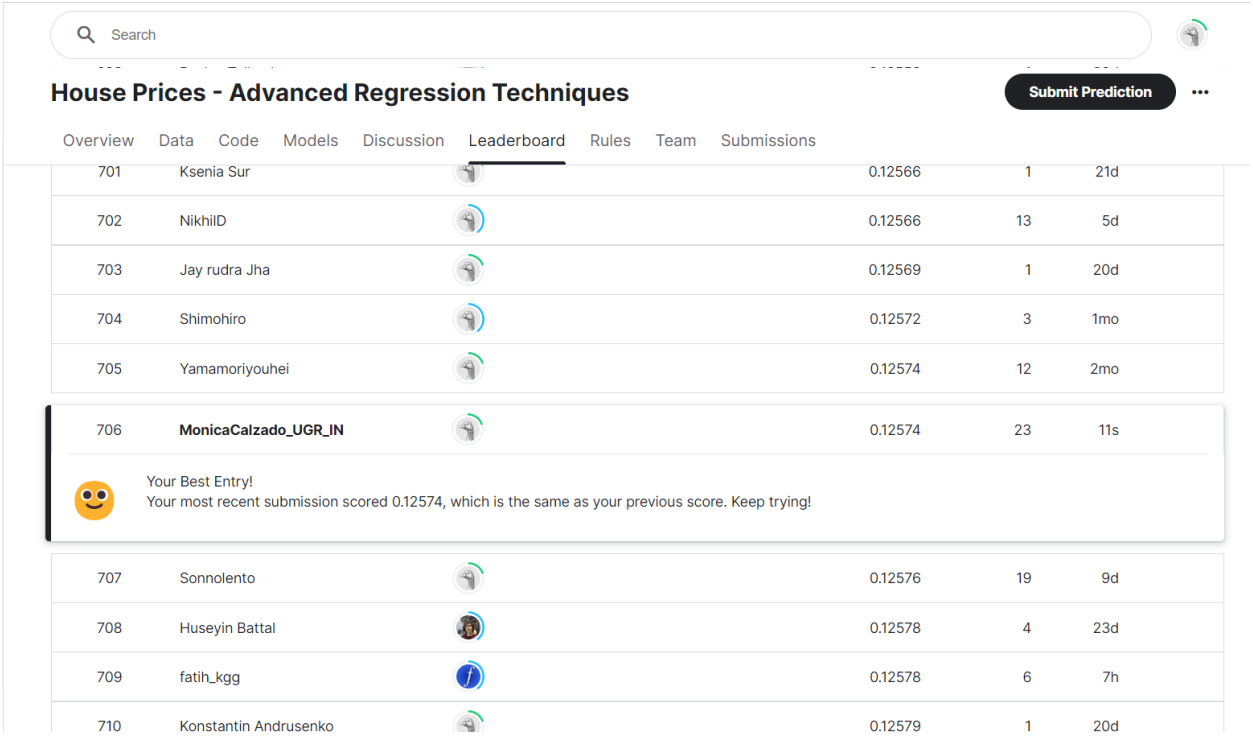


Figure 1: Captura de la tabla Leaderboard

Contents

2 Introducción

En esta práctica, con la finalidad de aplicar métodos avanzados de aprendizaje supervisado, vamos a participar en una competición de Kaggle de regresión, **House Prices - Advanced Regression Techniques**.

El objetivo de esta competición es predecir, dadas las características de una casa, su precio. Para cada casa vamos a disponer de una serie de atributos que describen las características de la casa, y de los que nos podremos servir para aproximar lo máximo posible el precio de venta de la casa. La variable objetivo es *SalePrice*, que es lo que pretendemos predecir.

En total disponemos de 81 atributos distintos, aunque hay que ignorar el *Id* de la transacción (pues sólo sirve para identificar el ejemplo). Estos atributos hacen referencias a características como el tamaño, el número de habitaciones, número de baños, la calefacción, el tipo de calle, la zona, la orientación, fechas de construcción y de reformas, etc. Observamos que entre los atributos podemos encontrar tanto datos numéricos en distintas escalas, como datos categóricos.

El conjunto de entrenamiento consta de 1460 instancias con su etiqueta *SalePrice*, que indica el precio de venta del inmueble.

El rendimiento en esta competición se evaluará mediante la media de la raíz del error cuadrático medio (RMSE) entre los algoritmos del precio predicho y el precio observado.

3 Visualización de datos

3.1 Distribution plot

Representamos en 2 la distribución de los valores de venta en los datos de entrenamiento. Observamos que las casas se concentran en torno a un precio de venta de 150k dólares, por

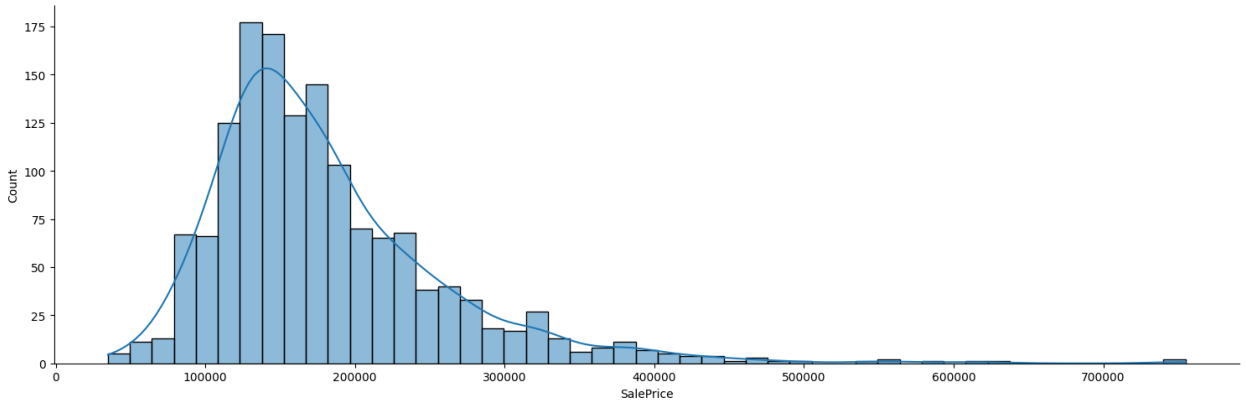


Figure 2: Distribution plot de SalePrice

lo que la distribución es sesgada a la derecha (right skewed).

3.2 Boxplot

Visualizamos varios boxplots para observar cómo cambia la distribución según otros atributos, como el tipo de calle (Street).

- **Street:** cómo cambia el precio de venta según el tipo de calle (pavimentada o gravilla). Vemos en 3 que las casas construidas en una calle pavimentada tienen ligeramente mayor precio que las casas rodeadas de gravilla.
- **OverallQual:** La calidad general de la casa podría estar relacionada con el precio de venta. Observamos en 4 que el precio aumenta muy considerablemente conforme a la calidad general de la casa, lo cual tiene sentido.
- **YearBuilt:** puede ser interesante observar cómo se distribuyen las ventas en función del año de construcción de la casa. Lo hemos representado en intervalos de 10 años porque si no era inobservable. Vemos en 5 que la fecha de construcción no es determinante del precio de la casa, pues este se va manteniendo aunque la casa tenga más de un siglo. Suponemos entonces que lo que influye no es esto sino la última reforma. Veamos si estamos en lo cierto.
- **YearRemodAdd:** puede ser interesante observar cómo se distribuyen las ventas en función del año de la última reforma de la casa. De nuevo lo representamos en intervalos de 10 años. Observamos en 6 una mayor correlación del precio con respecto al año de la última reforma, pues el precio va subiendo conforme el año es más reciente. Además, las casas muy caras (por encima de 300k dólares se concentran en torno a las que han tenido reforma en los últimos 30 años)

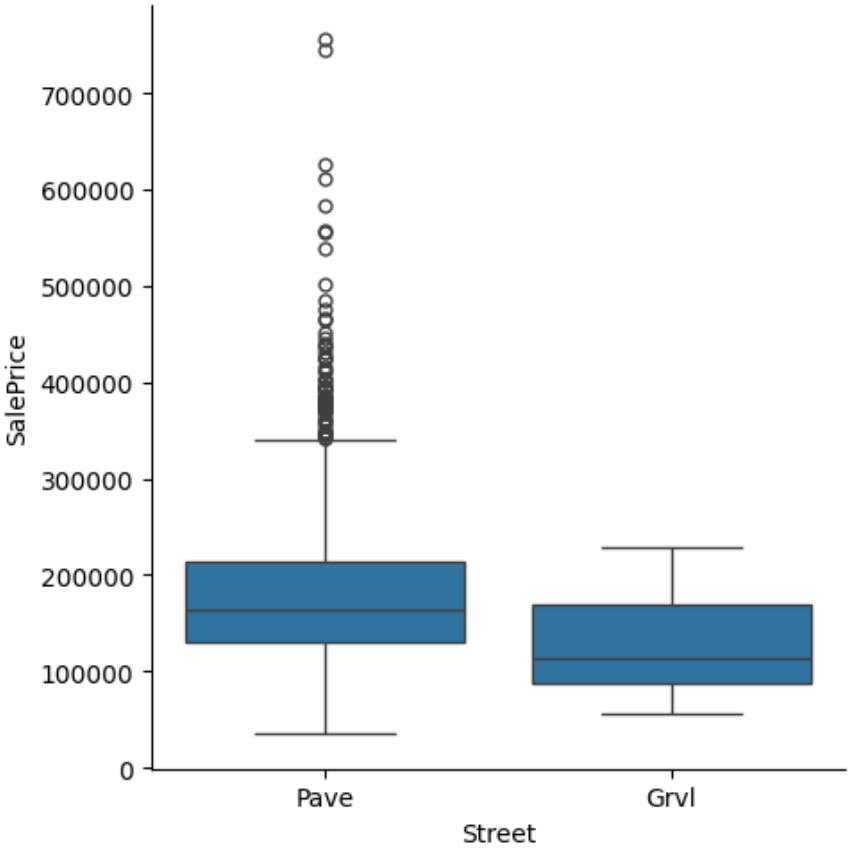


Figure 3: Distribución de Street

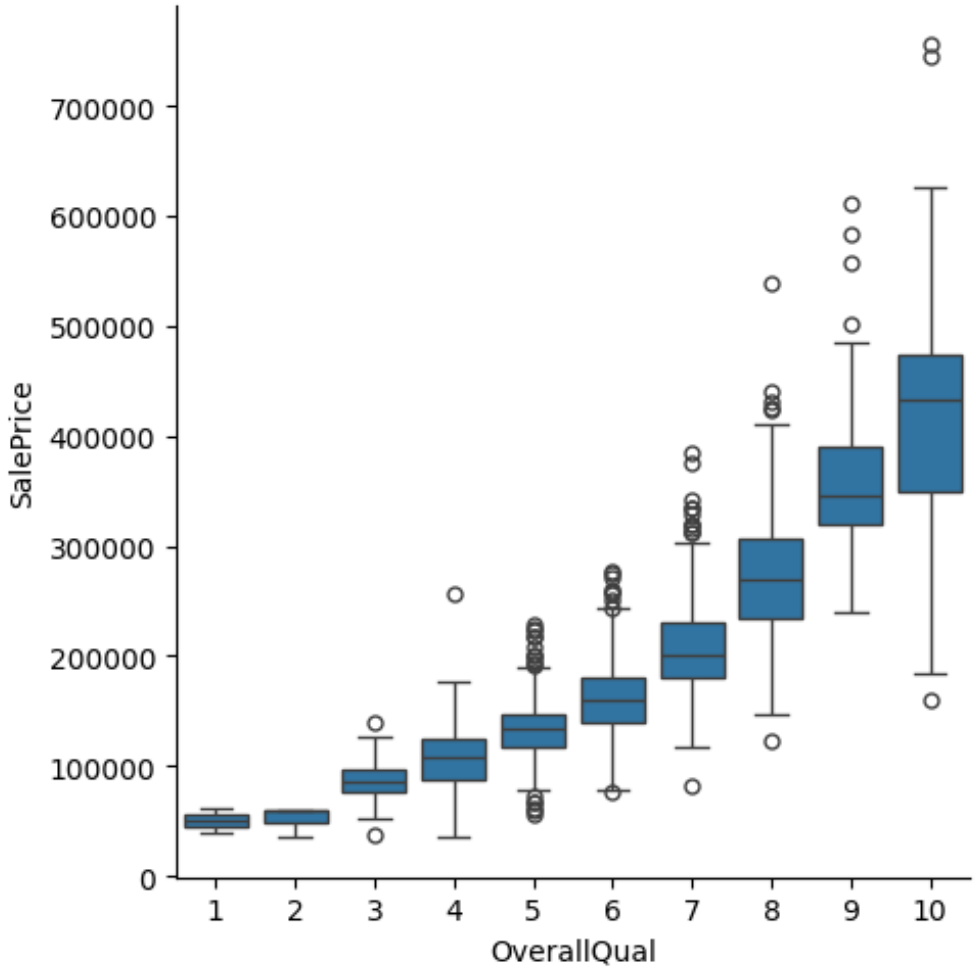


Figure 4: Distribución de OverallQual

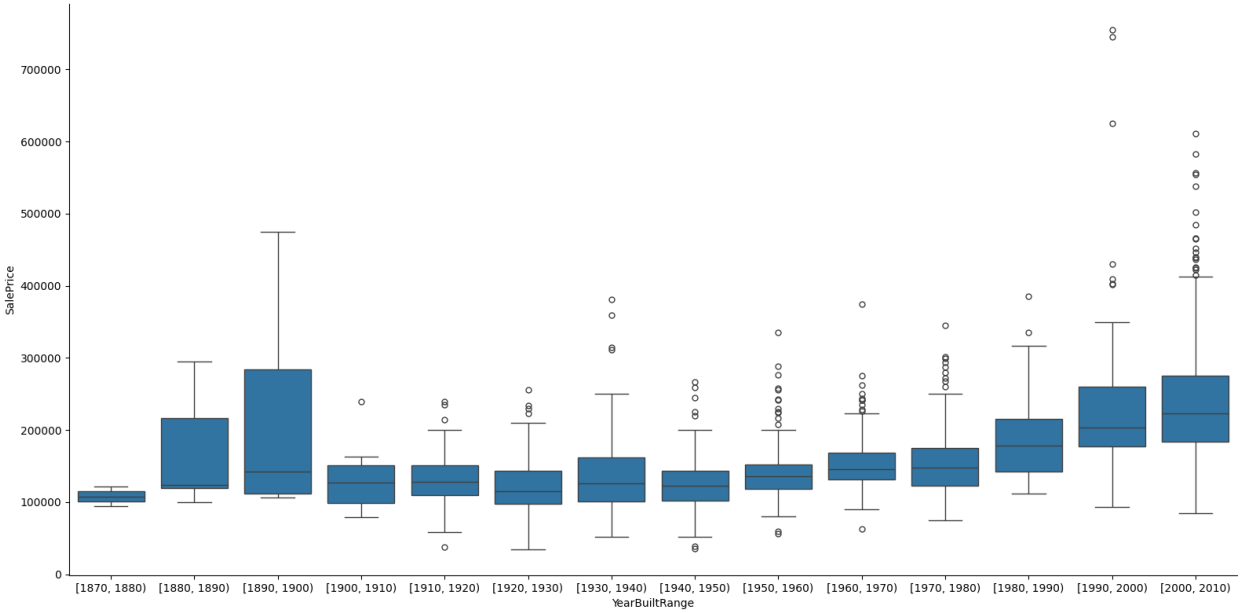


Figure 5: Distribución de YearBuilt

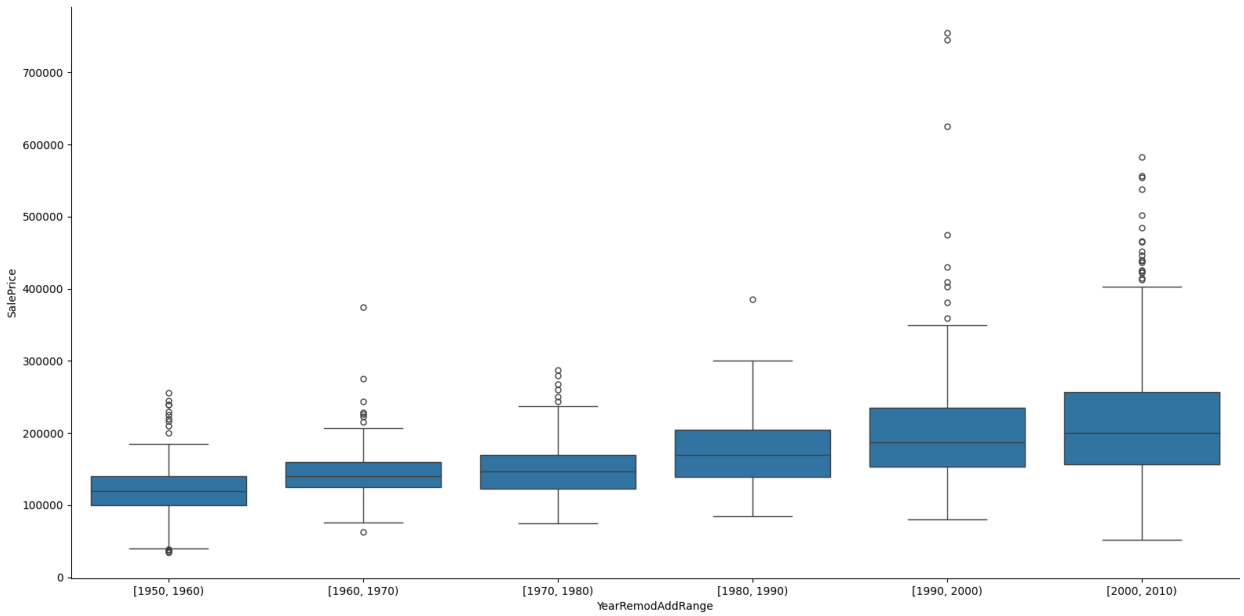


Figure 6: Distribución de YearRemodAdd

3.3 Heatmap

Visualizamos en 7 un heatmap para observar las correlaciones entre las distintas variables y su etiqueta. La finalidad es ver qué variables proporcionan más información sobre el precio de venta; también si hay relación entre las propias variables (para plantearnos hacer agrupamiento de características). Observamos que hay varias agrupaciones con correlaciones

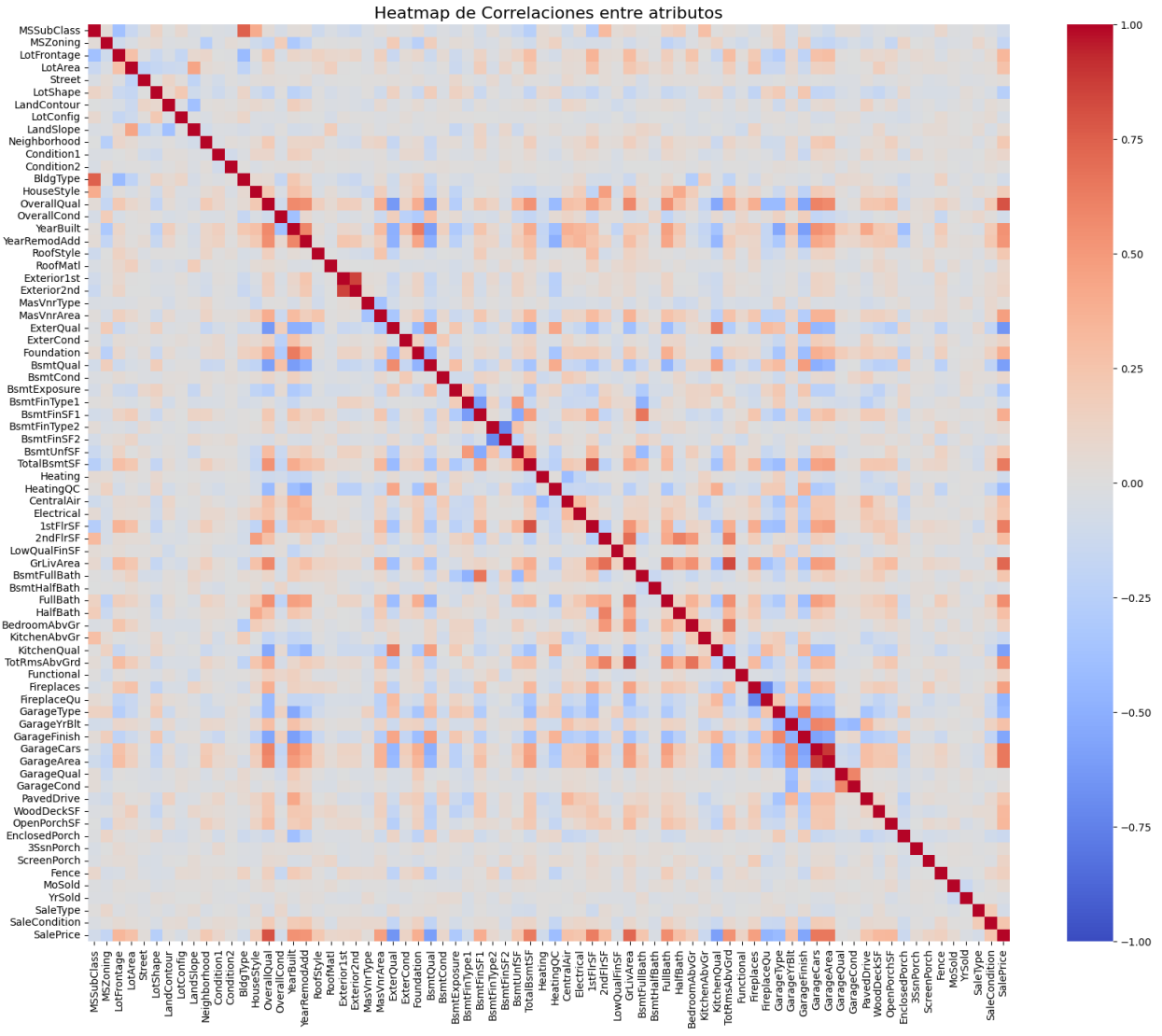


Figure 7: Heatmap de correlaciones

bastante altas, por ejemplo:

- **BldgType y MSSubClass:** la primera variable hace referencia al tipo de vivienda (unifamiliar, bifamiliar, dúplex, casa adosada...) mientras que la segunda variable identifica el tipo de vivienda mediante unos códigos, con más precisión. Tenemos entonces dos variables que están representado lo mismo a efectos generales, de ahí la alta correlación.

- **GarageCars y GarageArea:** tiene sentido que el número de plazas del garaje esté relacionada con el área del mismo.

- **Exterior1st y Exterior2nd:** revestimiento exterior de la casa, tiene sentido que sea del mismo material tanto en la primera planta como en la segunda.
- **TotalBsmtSF y 1stFlrSF:** el área del sótano y de la primera planta suele coincidir.
- **TotRmsAbvGrd y GrLivArea:** hay correlación entre el total de habitaciones sobre el nivel del suelo y área habitable sobre el nivel del suelo.

Llama la atención que además de encontrar correlaciones positivas, también hay fuertes correlaciones negativas:

- **BsmtFinType2 y BsmtFinSF2:** correlación entre la puntuación del área acabada del sótano y el área acabada. Esto podría deberse a que, en algunas casas, parte del sótano puede no estar completamente terminada o puede tener acabados de baja calidad, al tratarse de un sótano tan grande.
- **Fireplaces y FireplaceQu:** mientras más chimeneas hay peor calidad tienen.

Veamos las correlaciones entre algunos atributos y SalePrice:

- **OverallQual:** como hemos visto en el boxplot, hay una fuerte correlación entre la calidad de la casa y su precio.
- **GrLiveArea, 1stFlrSF, TotalBsmtSF:** los atributos de áreas están directamente relacionados con el precio.
- **GarageCars, GarageArea:** un garaje amplio implica un mayor precio de la casa.

4 Procesado de los datos

Antes de aplicar ningún modelo, debemos preparar los datos para trabajar con ellos. Al principio nos proporcionan un Notebook donde se imputan los valores perdidos y se realiza el etiquetado. Posteriormente hemos ido añadiendo otros tipos de procesamiento de los datos.

4.1 Outliers

Para detectar outliers, hemos representado varios Scatter plot de atributos numéricos con respecto a SalePrice. Para ello hemos empleado la siguiente función:

```
def scatter_plot(atributo, datos):
    fig, ax = plt.subplots()
    ax.scatter(x = datos[atributo], y = datos['SalePrice'], edgecolor='black')
    plt.ylabel('SalePrice', fontsize=10)
    plt.xlabel(atributo, fontsize=10)
    plt.show()
```

Los atributos numéricos estudiados han sido GrLivArea, 1stFlrSF, 2ndFlrSF, GarageArea y PoolArea. En algunos casos hemos visualizado outliers y los hemos eliminado, como con *GrLivArea* en 8.

En el caso de los atributos de tipo categórico, hemos representado un boxplot para los siguientes atributos: OverallQual, TotRmsAbvGrd. Podemos ver un ejemplo en 9.

4.2 Valores perdidos

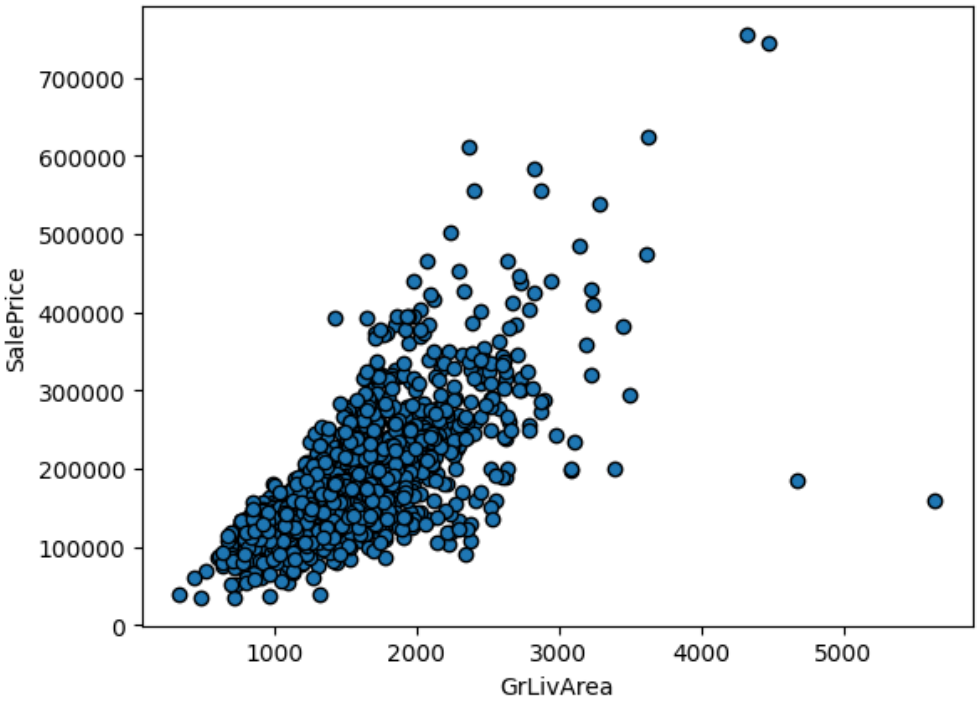
En primer lugar hemos contado el número total de valores perdidos con `train.isnull().sum().sum()` y con `test.isnull().sum().sum()`. Hemos obtenido casi 8000 valores perdidos en cada conjunto de datos.

En un principio lo que se hace es sustituir los datos numéricos perdidos por la mediana, y los datos categóricos por el más frecuente. Sin embargo este tratamiento es muy pobre, pues, entre otras cosas, hay atributos con tantos valores perdidos que estas medidas no tienen sentido.

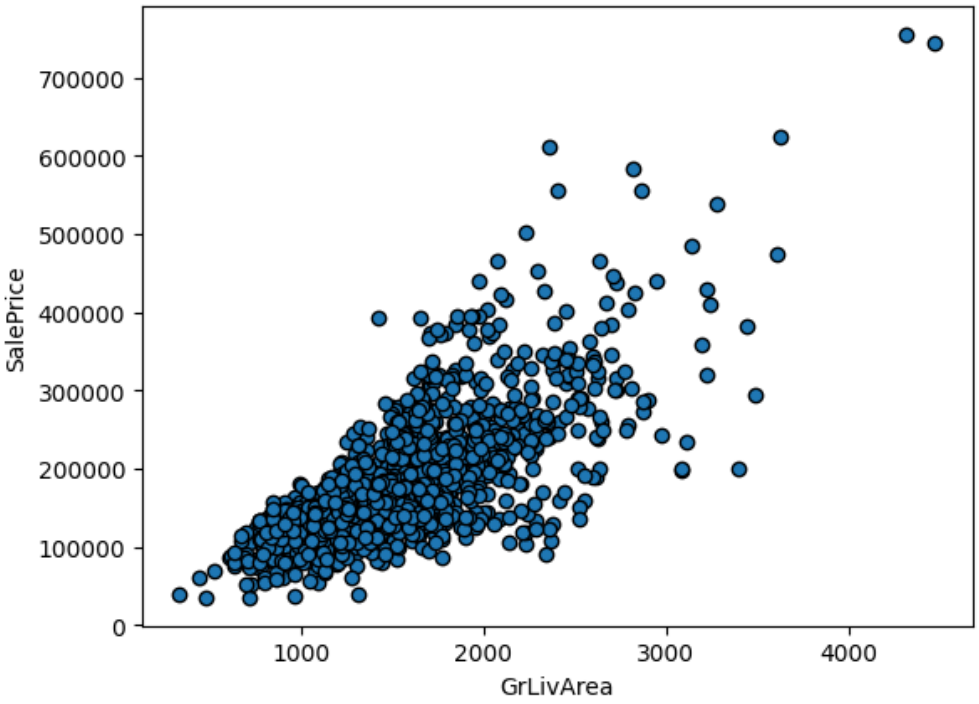
Lo que hemos hecho entonces ha sido observar qué atributos tiene un mayor número de missing values, con esta función:

```
def Valores_perdidos_columnas(datos):
    total = datos.isnull().sum().sort_values(ascending=False)
    valores_unicos = datos.nunique() #n de valores distintos, si es 1 no nos sirve la
    ↪  columna
    missing_data = pd.concat([total, valores_unicos], axis=1,
                             keys=['nº de valores perdidos', 'valores unicos'], sort =
                             ↪  False)

    return missing_data
```

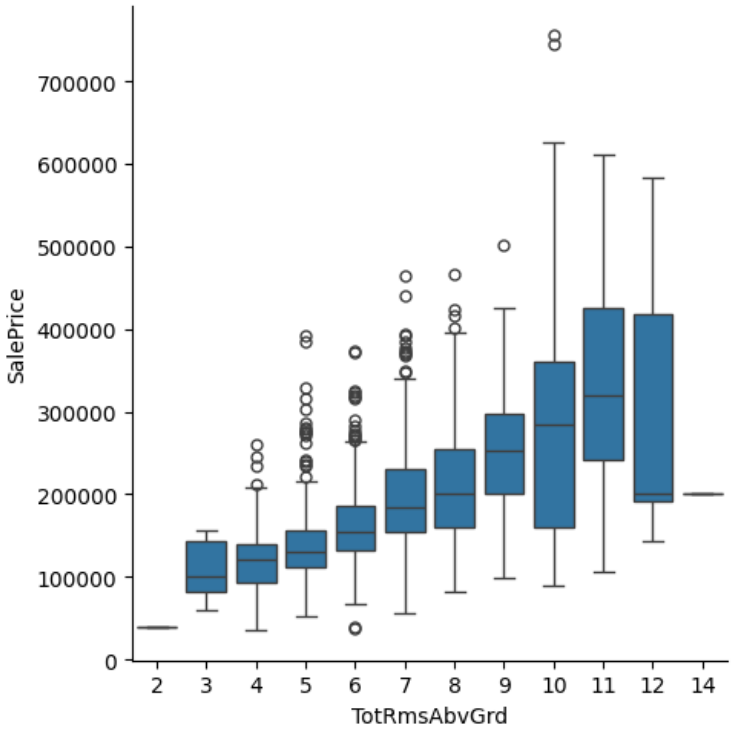


(a) Antes

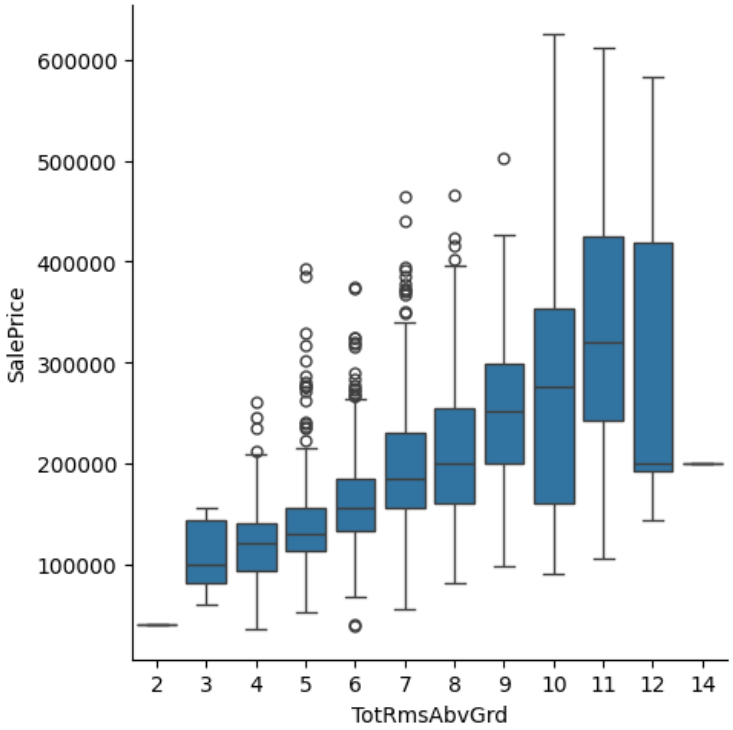


(b) Después

Figure 8: Distribución antes y después de eliminar outliers



(a) Antes



(b) Después

Figure 9: Distribución antes y después de eliminar outliers

Si hay algún atributo con valor único (que sólo toma un valor en todo el dataframe), lo eliminamos, como es el caso de *Utilities*, pues no nos está aportando información ninguna.

Hemos eliminado los atributos con mayor cantidad de missing values: *PoolQC*, *MiscFeature* y *Alley*.

- **PoolQC:** indica la calidad de la piscina, la gran mayoría de casas no tienen así que contestan NA. No lo consideramos por tanto un atributo determinante y podemos prescindir de él.
- **MiscFeature:** indica si la casa posee otras instalaciones aparte de las ya mencionadas, como ascensor propio, segundo garaje, pista de tenis... Muy pocas casas tienen algo de esto así que también lo vamos a descartar.
- **Alley:** si hay acceso a la casa desde un callejón, de qué tipo es: gravilla o pavimentado. De nuevo parece ser que las casas no tienen acceso por callejón, así que lo eliminamos.

Además, hemos observado que hay atributos con valor NaN (valor perdido), cuando realmente hacen referencia a la ausencia de dicha cualidad. Por ejemplo, en el atributo *Fence*, el valor Nan significa que no hay valla. Hemos arreglado esta malinterpretación sustituyendo *NaN* por *nofence*. Hemos realizado el mismo procedimiento para varios atributos más.

Después de realizar estos cambios, imputamos los missing values restantes con la mediana y el valor más frecuente, como se hacía en un principio.

Realizamos todo el procesamiento de missing values con la siguiente función:

```
def valores_perdidos_eliminar(datos):
    #PUNTO 1
    #Eliminar columnas con muchos valores perdidos. Eliminar columnas con unique_values =1
    #Como no hay piscinas, elimino todos los atributos relacionados con las piscinas. Lo mismo con la miscelánea.
    datos=datos.drop(['PoolQC', 'PoolArea', 'MiscFeature', 'MiscVal', 'Alley', 'Utilities'], axis=1)

    #PUNTO 2
    # Rellenar valores faltantes en la columna 'Fence' con 'nofence'
    datos['Fence'] = datos['Fence'].fillna('nofence')
    datos['FireplaceQu'] = datos['FireplaceQu'].fillna('nofireplace')
    datos['MasVnrType'] = datos['MasVnrType'].fillna('None')

    #PUNTO 3
    #Las variables de garaje con valores perdidos normalmente es porque no hay garaje: 'nogaraje'
    # Variables categóricas, las rellenamos con 'nogarage'
    for col in ('GarageCond', 'GarageType', 'GarageFinish', 'GarageQual'):
        datos[col] = datos[col].fillna('nogarage')

    # Variable numérica, la rellenamos con 0
```

```
    datos['GarageYrBlt'] = datos['GarageYrBlt'].fillna(0)

    #Hago lo mismo con las variables relacionadas con el basement
    # Variables categóricas, las rellenamos con 'nobasement'
    for col in ('BsmtCond', 'BsmtExposure', 'BsmtQual', 'BsmtFinType1', 'BsmtFinType2'):
        datos[col] = datos[col].fillna('None')


    #SI AÚN QUEDA ALGÚN MISSING VALUE:

    #PUNTO 4
    #Reemplazo los valores numéricos restantes por la mediana
    col_num = list(datos.select_dtypes(include=np.number).columns)
    if ('SalePrice' in col_num):
        col_num.remove('SalePrice')
    imputer_num = SimpleImputer(strategy="median")
    imputer_num.fit(input_all[col_num])
    datos[col_num] = imputer_num.transform(datos[col_num])

    #PUNTO 5
    #Reemplazo los datos categóricos restantes por el más frecuente
    imputer_cat = SimpleImputer(strategy="most_frequent")
    col_cat = list(datos.select_dtypes(exclude=np.number).columns) #Cojo los atributos
    ↪  categoricos
    imputer_cat.fit(datos[col_cat])
    datos[col_cat] = imputer_cat.transform(datos[col_cat])

    return datos
```

4.3 Etiquetado

Hemos convertido los atributos categóricos en numéricos. **LabelEncoder** es una herramienta de la biblioteca scikit-learn que se utiliza para convertir variables categóricas en numéricas. El objetivo es permitir que los algoritmos de aprendizaje automático trabajen con estos atributos de manera más efectiva.

5 Progreso

5.1 Tabla de soluciones subidas

Nombre	Fecha y hora	Posición	Score (Train)	Score (Kaggle)	Preprocesado	Algoritmo y parámetros
prueba1	31/12/2023 11:44	3716	0.04652	0.20145	Primer ejemplo de Prado: Imputación mediante mediana y valor más frecuente + etiquetado	Árbol de decisión (por defecto)
prueba2	31/12/2023 17:23	3716	0.04327	0.2274	Elimino ejemplos con más de 5 valores perdidos + imputación simple+ etiquetado	Árbol de decisión (por defecto)
prueba3	31/12/2023 17:25	3716	0.04909	0.26719	Elimino ejemplos con más de 4 valores perdidos + imputación simple + etiquetado	Árbol de decisión (por defecto)
prueba4	31/12/2023 17:28	3716	0.10801	0.35313	Elimino ejemplos con más de 2 valores perdidos + imputación simple +etiquetado	Árbol de decisión (por defecto)
prueba5	01/01/2024 10:26	3716	0.04684	0.2035	Elimino atributos con muchos valores perdidos. Elimino atributos con 1 única respuesta. Etiquetado	Árbol de decisión (por defecto)
prueba6	01/01/2024 11:24	3716	0.047086	0.20628	Gestiono los missing values distinguiendo casos (algunos los elimino, otros los imputo con 'most_frequent' o 'median')	Árbol de decisión (por defecto)
prueba7	01/01/2024 18:33	3713	0.04208	0.20103	Elimino algunos outliers	Árbol de decisión (por defecto)
prueba8	01/01/2024 18:55	2842	0.020186	0.15204	Aplicando Random Forest	Random Forest (criterion='squared_error', max_depth=10)
prueba9	01/01/2024 19:02	2541	0.021628	0.14803	Pruebo a ejecutar lo que hay en el segundo notebook de Prado	Random Forest (criterion='squared_error', max_depth=15)
prueba10	04/01/2024 10:08	2541	0.02259	0.15968	Tomo sólo los 10 atributos más importantes (visualización de la importancia)	Random Forest (criterion='squared_error', max_depth=10)
prueba11	04/01/2024 10:27	2541	0.01991	0.14871	Outliers+selección de características+imputación	Tuning automático con Random Forest (criterion='squared_error', max_depth=15)
prueba12	04/01/2024 10:33	2541	0.02272	0.15955	Tomo los 10 atributos con mayor importancia	Tuning automático con Random Forest (criterion='squared_error', max_depth=15)
prueba13	04/01/2024 12:41	2541	0.01486	0.1559	Outliers+selección de características+imputación + etiquetado	Catboost (NO QUERÍA SUBIR ESTE)

prueba14	04/01/2024 12:49	1177	0.012868	0.13083	Outliers+selección de características+imputación+ etiquetado	Catboost (iterations=5000, learning_rate=0.02, depth=4, eval_metric='RMSE', early_stopping_rounds=20, verbose=0)
prueba15	04/01/2024 16:17	1177	0.01486	0.13125	Tomo los 10 atributos con mayor importancia+ etiquetado	Catboost (iterations=5000, learning_rate=0.02, depth=4, eval_metric='RMSE', early_stopping_rounds=20, verbose=0)
prueba16	04/01/2024 16:31	1177	0.01649	0.14368	Tomo los 20 atributos con mayor importancia+ etiquetado	Catboost (iterations=5000, learning_rate=0.02, depth=4, eval_metric='RMSE', early_stopping_rounds=20, verbose=0)
prueba17	04/01/2024 16:38	1177	0.01381	0.13631	Tomo los 40 atributos con mayor importancia+ etiquetado	Catboost (iterations=5000, learning_rate=0.02, depth=4, eval_metric='RMSE', early_stopping_rounds=20, verbose=0)
prueba18	04/01/2024 17:47	1177	0.018225	0.13154	Outliers+selección de características+imputación	XGBoost (n_estimators=5000, learning_rate=0.02, colsample_bytree=0.5, subsample=0.5, min_child_weight=2, verbose=0)
prueba19	04/01/2024 17:59	1177	0.018225	0.13231	Outliers+selección de características+imputación+ etiquetado	Bagging ensemble (catboost y xgboost con los parámetros anteriores)
prueba20	05/01/2024 09:09	1139	0.012864	0.13037	Outliers+selección de características+imputación+ etiquetado	Catboost con tuning automático (iterations=5000, learning_rate=0.01, depth=6, eval_metric='RMSE', early_stopping_rounds=20, verbose=0)
prueba21	07/01/2024 10:28	708	0.012953	0.12574	Outliers+selección de características+imputación	Catboost (iterations=5000, learning_rate=0.01, depth=6, eval_metric='RMSE', early_stopping_rounds=20, verbose=0)
prueba22	07/01/2024 20:37	706	0.013244	0.13430	Outliers+selección de características+imputación+ etiquetado	XGBoost con tuning (n_estimators=2000, learning_rate=0.05, colsample_bytree=0.45, subsample=0.8, min_child_weight=2, verbose=0)
prueba23	07/01/2024 20:46	706	0.012953	0.12574	Lo mismo que la prueba21	Lo mismo que la prueba21

5.2 Algoritmos empleados

Los algoritmos que hemos probado a emplear para construir el modelo han sido los siguientes:

- **Árbol de decisión.** Se trata de un modelo de aprendizaje supervisado que representa decisiones en forma de árbol. Cada nodo del árbol representa un atributo. Los árboles de decisión pueden capturar patrones complejos en los datos, lo cual puede resultar útil para predecir precios de casas en función de muchas características.

- **Random Forest.** Es un conjunto de árboles de decisión, donde cada árbol se entrena en una muestra diferente del conjunto de datos, con selección aleatoria de características. La predicción final se obtiene promediando las predicciones de todos

los árboles. La ventaja de Random Forest es que reduce el sobreajuste, al combinar múltiples modelos. Es robusto y maneja bien conjuntos de datos grandes.

- **CatBoost.** Es un algoritmo de gradient boosting diseñado para manejar automáticamente variables tanto categóricas como numéricas sin necesidad de preprocesamiento adicional. Su capacidad para manejar variables categóricas automáticamente puede simplificar el proceso de modelado.

- **XGBoost.** Es un algoritmo de gradient boosting que utiliza árboles de decisión como modelos base. Destaca por su eficiencia y escalabilidad, así como por la incorporación de técnicas como la regularización y la poda. XGBoost es muy utilizado en competiciones de ciencia de datos debido a su rendimiento sólido. Puede manejar conjuntos de datos grandes y complejos.

- **Bagging Ensemble.** Es una técnica que combina múltiples modelos independientes entrenados en subconjuntos aleatorios del conjunto de datos. La predicción final se obtiene mediante regresión o mediante votación las predicciones individuales. Hemos elegido aplicarlo en nuestro problema pues Bagging puede mejorar la estabilidad y la precisión del modelo, reduciendo el sobreajuste. Lo aplicaremos con CatBoost y XGBoost.

Cabe destacar que CatBoost puede manejar atributos categóricos directamente sin preprocesamiento adicional, mientras que los demás algoritmos pueden manejar variables categóricas pero generalmente requieren de algún tipo de preprocesamiento, como el uso de etiquetas (Label Encoding), para convertir las variables categóricas en formato numérico antes de pasar los datos al modelo.

5.3 Explicación de los avances realizados

La primera entrega fue el fichero proporcionado por los profesores en Prado, el cual realiza **imputación simple**, sustituyendo los valores perdidos de los atributos numéricos por la mediana, y los de los categóricos por el valor más frecuente. Además se aplica como modelo un **Decision Tree**. Esta subida se realizó con la intención de marcar un límite a superar. Cada entrega tendrá un nombre del tipo pruebaX.ipynb con su correspondiente pruebaX.csv, donde X es un número.

En primer lugar, al observar que había tantos atributos, traté de deshacerme de algunos ejemplos con muchos valores perdidos, eliminando en primer lugar los ejemplos con más de 5 valores perdidos y posteriormente con más de 4 y de 2. Ningún caso dio mejor Score, así que descartamos este procesamiento.

Posteriormente empecé a estudiar los **missing values** y construí una función para eliminar atributos con muchos valores perdidos. Además eliminé atributos con un unique value (con un único valor en todas sus respuestas). A eso le añadí el gestionar los missing values distinguiendo casos (estudiando atributos uno a uno o por grupos en caso de estar relacionados), pues había muchos valores perdidos que se debían a una mala interpretación de las respuestas. Por ejemplo, en el atributo Fence encontrábamos el valor *NaN* para referirse a que no había valla, por lo que no se trataba de un valor perdido; esto lo arreglamos sustituyendo por *nofence*.

Después de tratar los missing values, estudié los **outliers** para algunos de los atributos más importantes. Para entonces seguía sin obtener mejores resultados, así que comencé a probar con otros algoritmos.

Al probar a aplicar **Random Forest**, la puntuación mejoró mucho, subiendo más de 1000 puestos en el ranking.

Tras esto probé a ejecutar el segundo fichero que hay en Prado, donde se aplica también Random Forest y se hace un tuning automático de parámetros.

Viendo el contenido de este nuevo fichero, probé a hacer **selección de características**, tomando los 10 atributos con mayor importancia, que los podemos ver en 10, pero no mejoró el rendimiento del algoritmo.

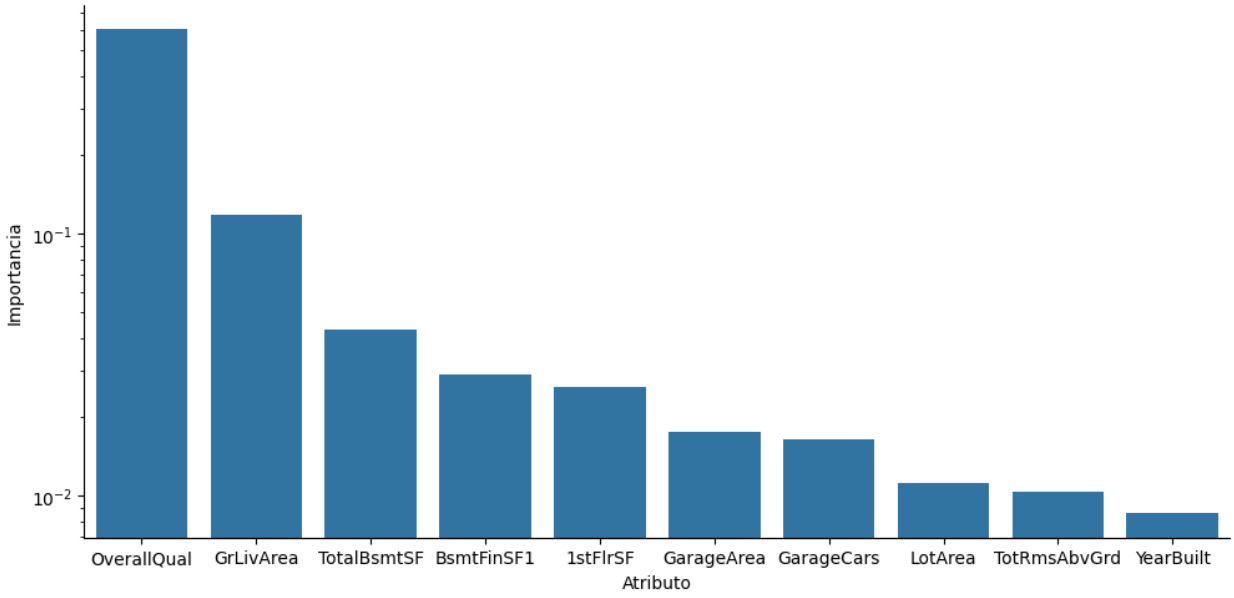


Figure 10: Importancia de los atributos

Posteriormente probé a hacer **tuning automático** en mi Notebook, junto con el preprocesado que anteriormente hacía, y obtuve resultados similares al Notebook de Prado. Entonces junté importancia y tuning, pero no obtuve buenos resultados así que lo descarté de nuevo.

Pasé a probar otros modelos y elegí **Catboost**. Con los parámetros elegidos conseguí la mejor puntuación que había logrado hasta entonces. Además de preprocesamiento realicé eliminación de outliers, selección de características e imputación, además de etiquetado. En las 3 siguientes entregas seguí aplicando Catboost pero seleccionando los 10, 20 y 40 atributos con mayor importancia, sin obtener buenos resultados.

También intenté hacer Bagging ensemble combinando los resultados de **Catboost** y **XGBoost**, pero no mejoré.

En la entrega 20 he hecho Catboost con tuning automático, en una ejecución que ha durado aproximadamente una hora, y obtuve el mejor resultado hasta el momento. El código para ejecutarlo es este:

```
1 from catboost import CatBoostRegressor
2 from sklearn.model_selection import GridSearchCV, KFold, cross_val_score
```

```
3
4  # Definir el espacio de búsqueda de hiperparámetros
5  param_grid = {
6      'iterations': [500, 1000, 2000, 5000],
7      'learning_rate': [0.01, 0.02, 0.03, 0.05],
8      'depth': [2, 4, 6, 8],
9      'verbose': [0]
10 }
11
12 # Inicializar el modelo CatBoost
13 model_catboost = CatBoostRegressor(eval_metric='RMSE', early_stopping_rounds=20)
14 print("inicializado modelo")
15
16 # Configurar la búsqueda de hiperparámetros
17 grid_search = GridSearchCV(estimator=model_catboost, param_grid=param_grid,
18                            scoring='neg_mean_squared_log_error', cv=KFold(n_splits=10),
                               ↪  verbose=1)
19 print("configurada la búsqueda de parámetros")
20
21 # Realizar la búsqueda en la cuadrícula
22 grid_search.fit(X_train, y_train)
23 print("realizada la búsqueda")
24
25
26 # Obtener el mejor modelo y sus hiperparámetros
27 best_model = grid_search.best_estimator_
28 best_params = grid_search.best_params_
29
30 # Imprimir los mejores hiperparámetros
31 print("Mejores hiperparámetros:", best_params)
32
33 # Realizar la validación cruzada con el mejor modelo
34 cv = KFold(n_splits=10)
35 values = cross_val_score(best_model, X_train, y_train,
   ↪  scoring='neg_mean_squared_log_error', cv=cv)
36
37 # Imprimir los resultados de la validación cruzada
38 print("Resultados de la validación cruzada:", values)
39 print("Media de los resultados:", values.mean())
40
41 # Hacer predicciones en el conjunto de prueba
42 pred_cat = best_model.predict(X_test)
43
44 # Crear el DataFrame de salida
45 salida = pd.DataFrame({'Id': test_ids, 'SalePrice': pred_cat})
46
```

```
47 # Guardar el DataFrame en un archivo CSV
48 salida.to_csv("resultados/PruebaX_Catboost_tuning.csv", index=False)
```

En la entrega 21 me percaté de que hasta ahora estaba usando Catboost habiendo realizado antes el etiquetado de los datos categóricos, lo cual no hacía falta. Entonces ejecuté Catboost pasándole los datos sin aplicarles *LabelEncoder*, y con los mismos parámetros de la prueba20, y conseguí una puntuación mucho más óptima.

La entrega 22 ha sido una aplicación de XGBoost con unos parámetros escogidos sin éxito, sin más. He vuelto a subir la entrega 21 para que se guarde esa como la última.

6 Bibliografía

Referencias y material consultado para la realización de la práctica:

- Diapositivas de la asignatura en Prado.
- `https://matplotlib.org/`
- `https://scikit-learn.org/stable/`
- `https://catboost.ai/en/docs/installation/python-installation-test-catboost`
- `https://xgboost.readthedocs.io/en/stable/python/index.html`