Informe del script de modelizado de datos sobre dataset de viviendas

Marc Roca Salvans

Matemáticas y estadística para la IA

Universidad Alfonso X el Sabio

Índice:

Anexo de imagenes:

Ilustración 1: Ejemplo de Boxplot con la variable SalePrice

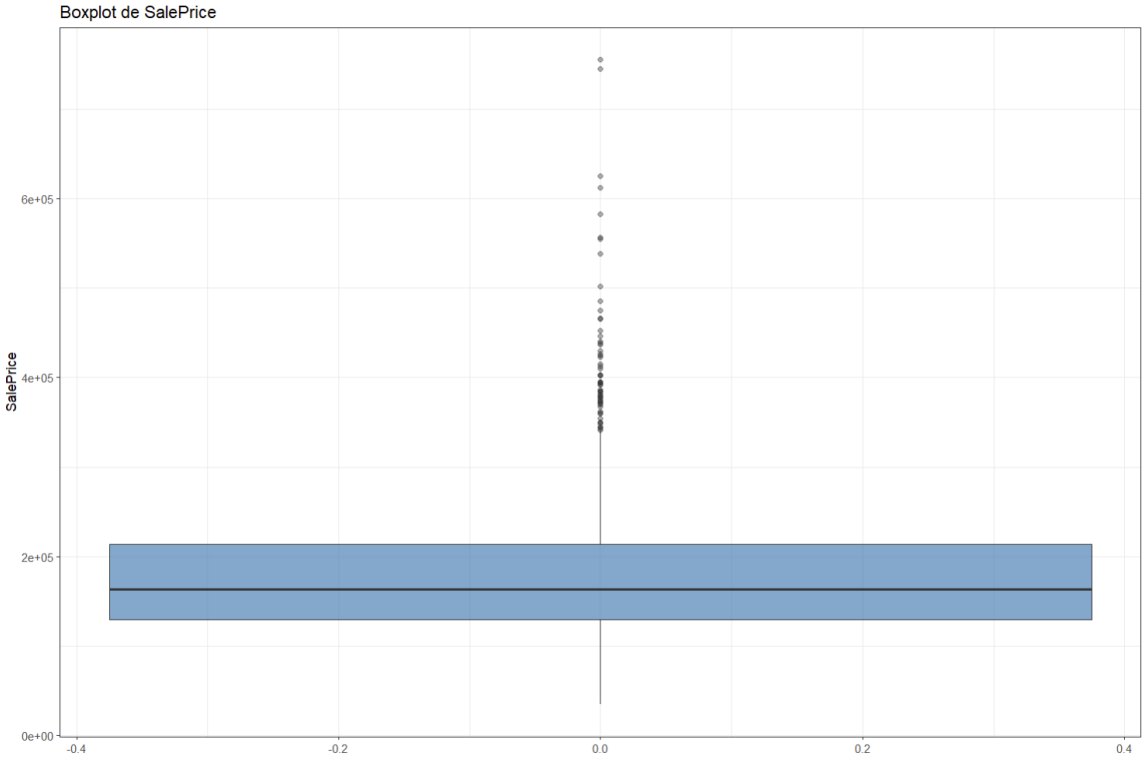

Ilustración 2: Ejemplo de Boxplot con la variable SaleCondition VS SalePrice

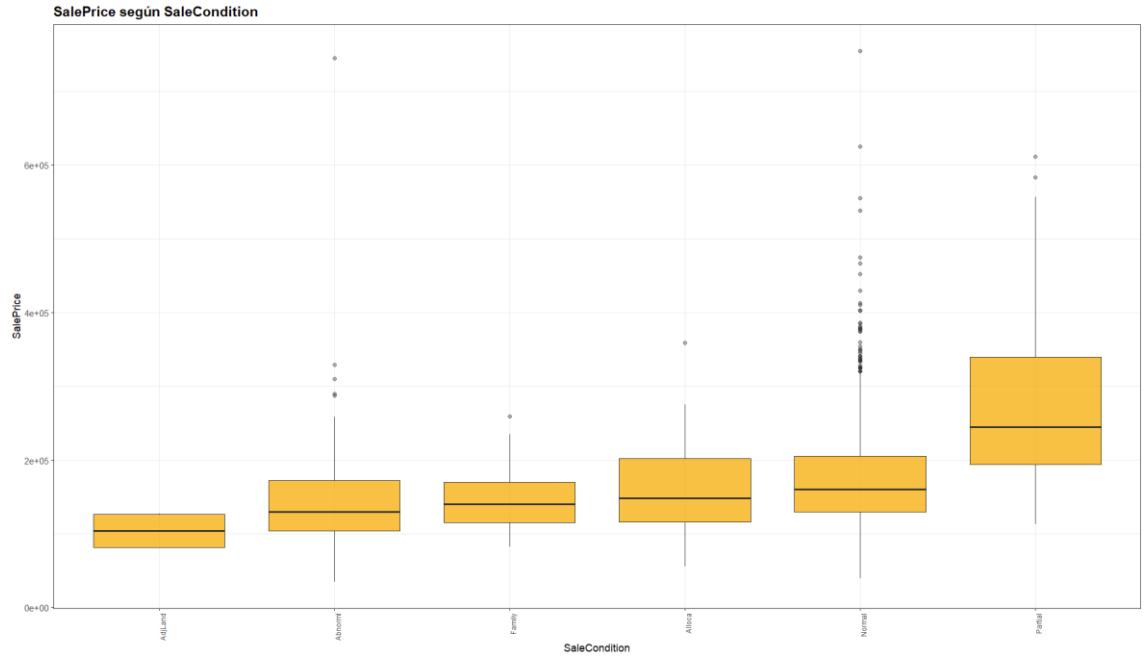

Ilustración 3: Ejemplo de Boxplot con la variable GrLivArea VS SalePrice

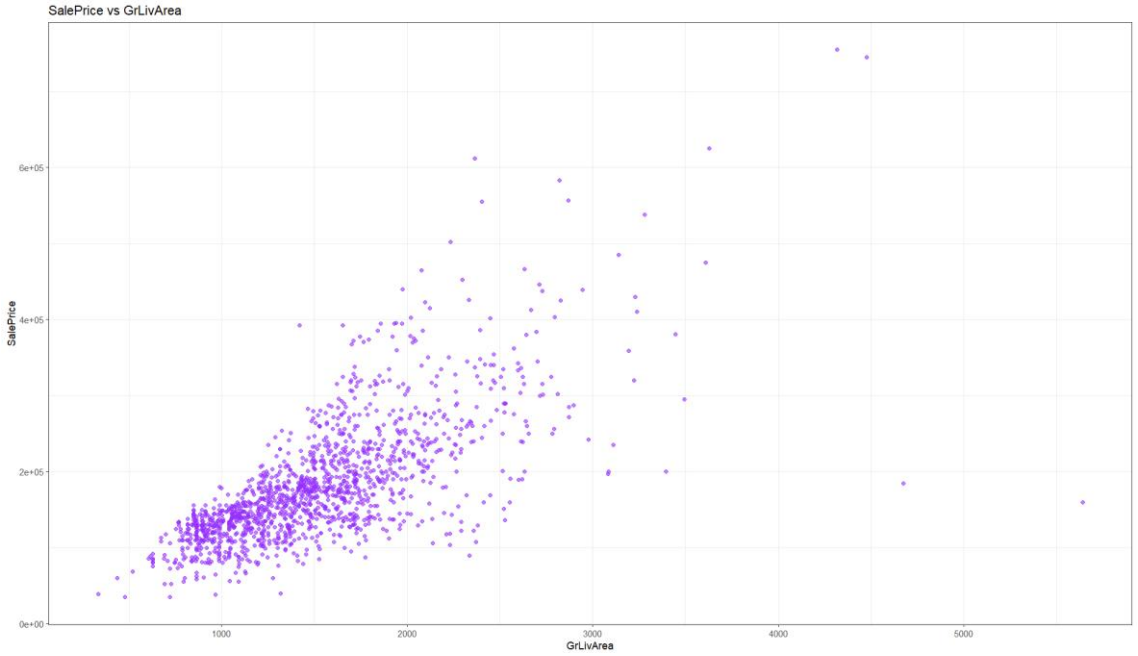

Ilustración 4: Matriz de correlación

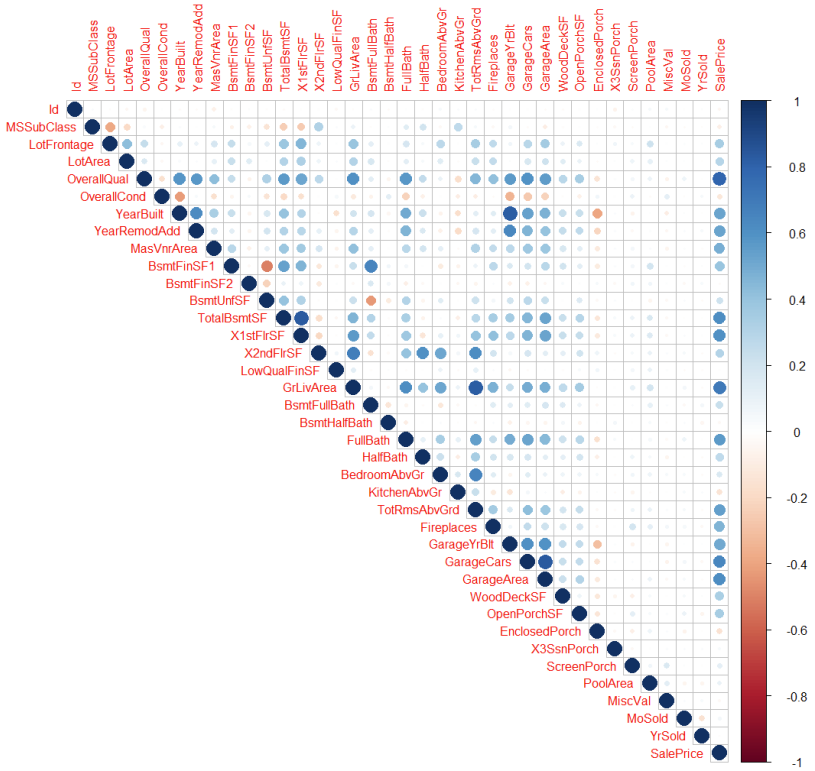

Ilustración 5: PCA Plot

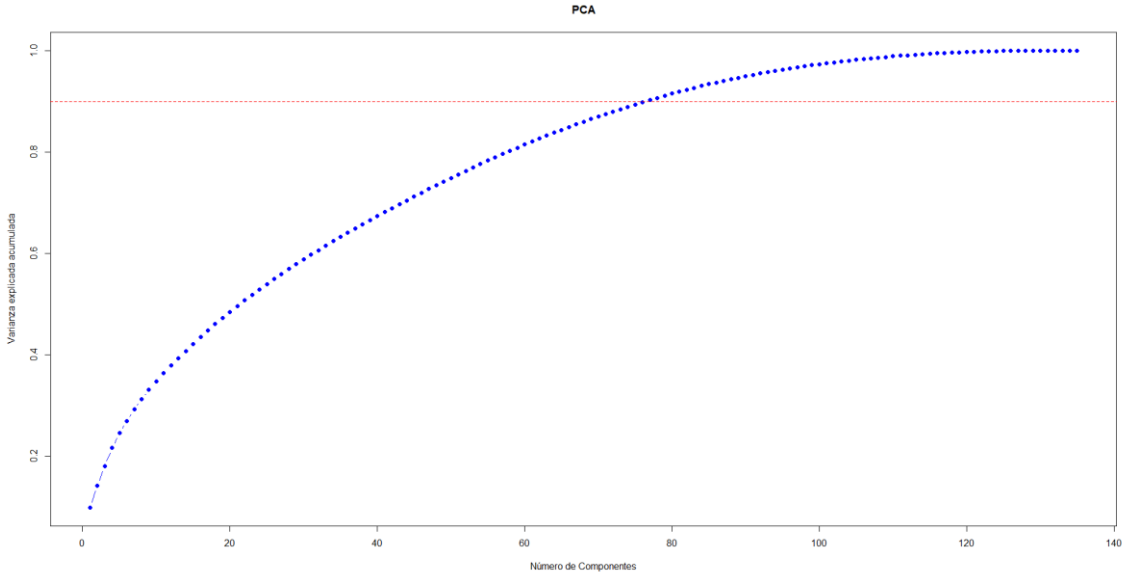

Ilustración 6: Mejor K para el PCA

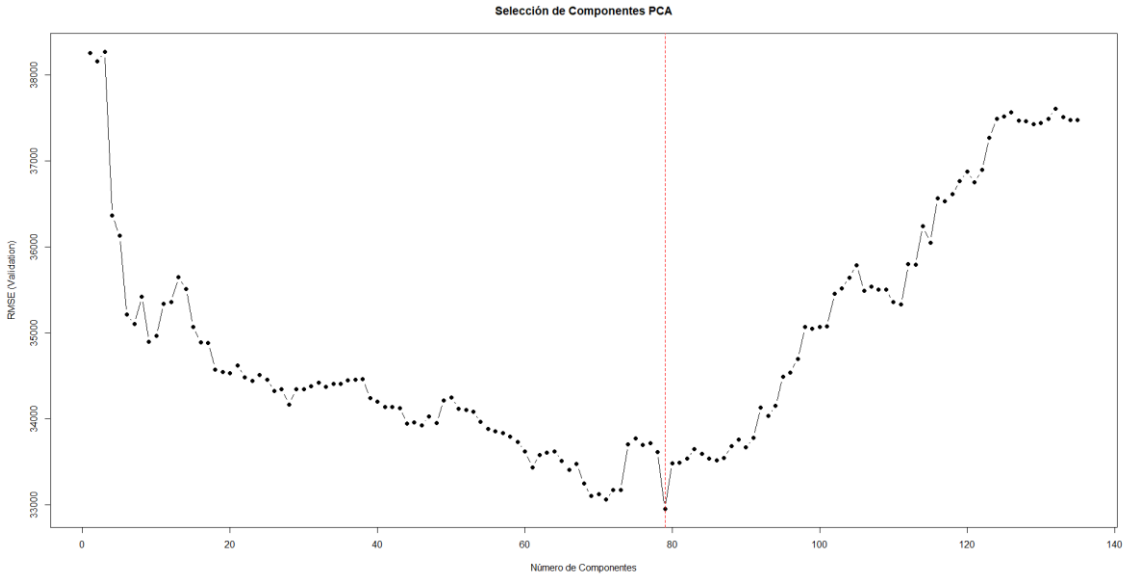

Ilustración 7: Mejor Lambda para Lasso

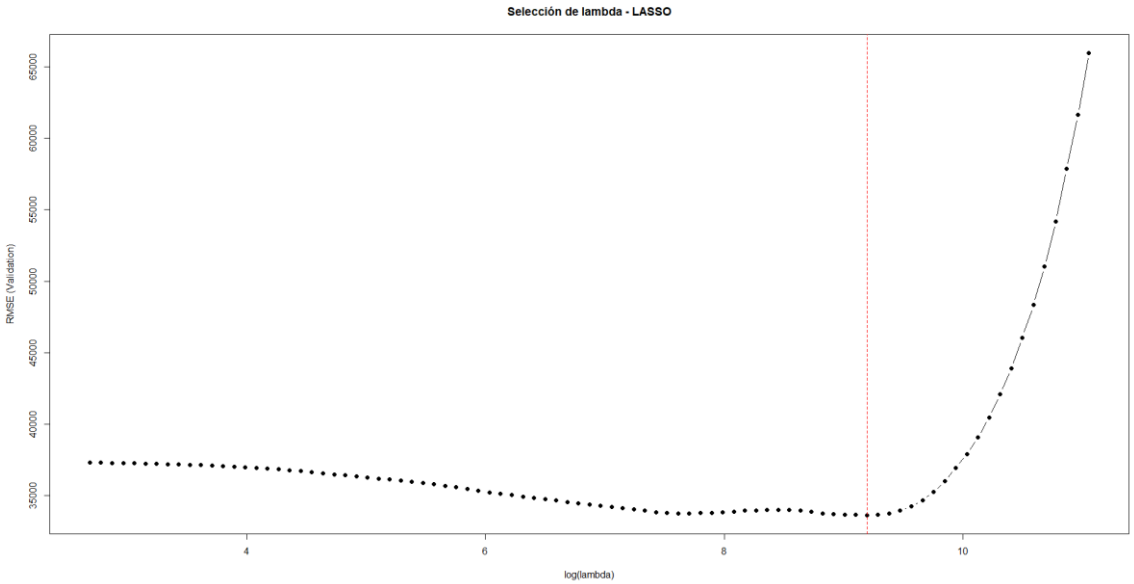

Ilustración 8: Mejor Lambda para Ridge

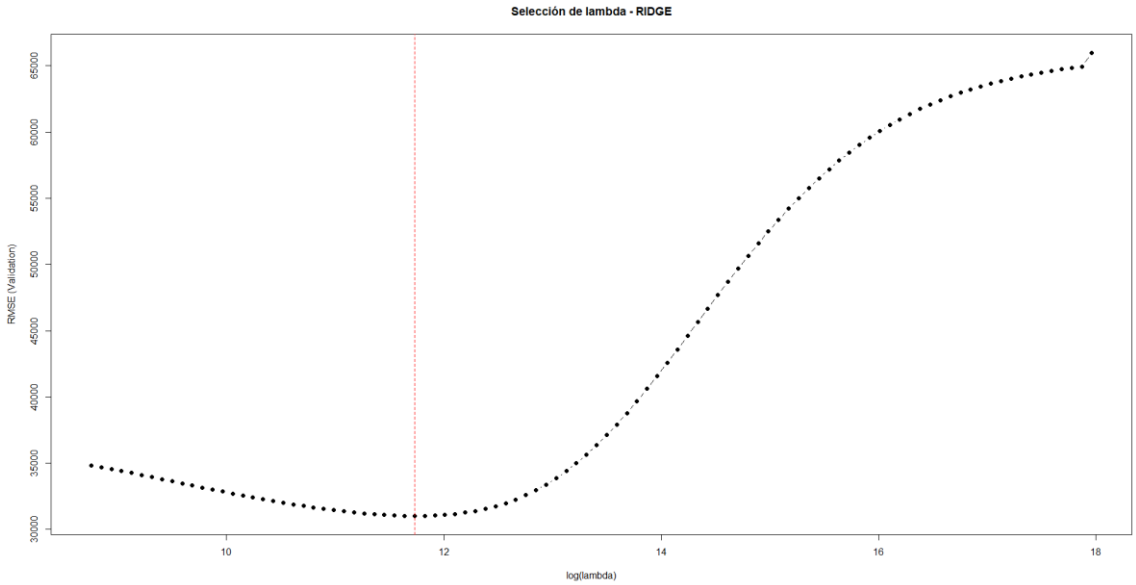

Ilustración 9: Resultados finales de las métricas sobre los modelos

Comparación de Modelos: RMSE, MAE y R^2

	Modelo	RMSE_Validation	RMSE_Test	MAE_Validation	MAE_Test	R2_Validation	R2_Test
3	RIDGE	31023.42463842596	42394.20760543328	19292.57379293946	22029.82808679718	0.7771066895849129	0.7684539069819953
5	RIDGE + PCA	31023.78004142413	42414.59260660263	19358.12844409173	22138.10106325563	0.7771015826435955	0.7682311783380751
4	LASSO + PCA	33491.99983800721	43173.45116804655	20773.09580384836	22206.79625132624	0.7402235841325454	0.7598636276625578
1	PCA + Linear	32951.1187366613	43214.38837561129	20870.82884785536	22416.92831859146	0.7485463825367177	0.7594080155715996
2	LASSO	33626.87083391113	46629.41060372368	21298.9564120006	23624.25712480826	0.7381271522076238	0.7198799050519655

1. Introducción y contextualización del problema

En este proyecto, tenemos a nuestra disposición un conjunto de datos que describe diversas características de viviendas (tamaño, ubicación, condiciones, etc.) junto con sus precios de venta. El objetivo principal es construir un modelo de machine learning en R que pueda predecir el precio de una casa en función de sus características. Desde un punto de vista estadístico, el problema presenta varias dificultades relevantes:

- Alta dimensionalidad tras la codificación de variables categóricas.
- Presencia de valores perdidos (missing values).
- Variables con distribuciones asimétricas y posibles outliers.
- Multicolinealidad entre predictores numéricos.

2. Análisis exploratorio de datos (EDA)

Vamos a necesitar diferentes librerías para el funcionamiento de nuestro script:
tidyverse: Utilizada para diferentes funcionalidades (ggplot2, dplyr, readr, etc).
skimr: Resumen rápido y compacto del dataset.
janitor: Limpieza y estandarización de nombres de columnas.
naniar: Analizar y visualizar valores faltantes (NA).
corrplot: Crear mapas de calor de correlaciones.
GGally: Matrices de gráficos tipo "ggpairs" para EDA.
caret: Particionado de datos (train/val/test) y utilidades de modelado.
recipes: Preprocesamiento estructurado (centrado, escalado, etc.).
glmnet: Modelos Lasso y Ridge (regresiones penalizadas).
yardstick: Cálculo de métricas de rendimiento como RMSE/MAE (aunque finalmente usas funciones propias).
Hmisc: Utilidades estadísticas adicionales (imputación, descriptivos).
embed: Métodos modernos de codificación categórica (target/frequency encoding).
knitr: Formatear la tabla final en salida de código.
DT: Mostrar la tabla final interactiva y visual en forma de datatable.

Carga del dataset

Cargamos los datos simplemente con un read.csv para leer nuestro csv llamado "train.csv"

Visualización de datos

En este apartado vamos a ver de manera sencilla y resumida los datos de nuestro dataset:

- Miraremos las dimensiones utilizando la función dim()
- Sacaremos un mini resumen del dataset con la función str() para poder visualizar el tipo de datos de cada columna/variable y los primeros valores
- Obtendremos con la función summary() vamos a ver por encima datos mas específicos de cada variable:
    - **Min**: valor mínimo
    - **1st Qu.**: valor en el que cae el 25% de valores
    - **Median**: la mediana
    - **Mean**: la media
    - **3rd Qu.**: valor en el que cae el 75% de valores
    - **Max**: valor máximo
- Haremos un listado separando las variables numéricas y las categóricas, ya que las tendremos que tratar diferente en el "train".
- Miraremos si hay "NA" en nuestro dataset y juntando las funciones anyNA() y colSums() podemos ver cuantos "NA" hay por variable
- Finalmente, con la función skim() podremos ver un resumen un poco mas amplio de nuestros datos, esta función nos dará nuevos valores interesantes al igual que valores que ya habíamos obtenido. Los nuevos valores por variable serán:
    - **n_missing**: número de valores que son NA
    - **complete_rate**: expresado en porcentaje y comparado con el total de valores, indica el total de valores con algún valor (todos los que no sean NA)
    - **min** (en variables categóricas): número mínimo de caracteres que tiene un valor
    - **max** (en variables categóricas): número máximo de caracteres que tiene un valor
    - **empty** (categóricas): número de valores vacíos ("") en la columna
    - **n_unique** (categóricas): número de valores distintos en la columna
    - **whitespace** (categóricas): número de valores que contienen espacios en blanco al inicio o final
    - **mean** (numéricas): promedio de los valores de la columna
    - **sd** (numéricas): desviación estándar de los valores
    - **p0** (numéricas): percentil 0 (mínimo) de la columna
    - **p25** (numéricas): percentil 25 de la columna (primer cuartil)
    - **p50** (numéricas): percentil 50 de la columna (mediana)

- **p75** (numéricas): percentil 75 de la columna (tercer cuartil)
- **p100** (numéricas): percentil 100 (máximo) de la columna
- **hist** (numéricas): histograma simplificado en texto que indica la distribución

Visualización gráfica de datos

En este apartado vamos a sacar múltiples plots y visualizaciones para representar nuestros datos de manera gráfica y fácil de interpretar.
Vamos a hacer cuatro tipos de gráficos:

1. **Boxplots de valores de las variables numéricas:**

Vamos a sacar boxplots de manera indiscriminada de todos los valores de las variables numéricas, esto nos permitirá ver de manera individual como los valores se reparten en los cuartiles y los outliers.
(Consultar en el Anexo de Imágenes, Ilustración 1: Ejemplo de Boxplot con la variable SalePrice)

2. **Boxplots de las variables categóricas VS la variable objetivo SalePrice:**

A continuación, vamos a sacar los boxplots comparándolos con nuestra variable objetivo que queremos predecir (SalePrice) para ver cómo se distribuyen los valores por cada valor de las variables categóricas.
(Consultar en el Anexo de Imágenes, Ilustración 2: Ejemplo de Boxplot con la variable SaleCondition VS SalePrice)

3. **Scatterplots de las variables numéricas VS la variable objetivo SalePrice:**

Siguiendo con los plots, vamos a sacar los scatterplots comparándolos con nuestra variable objetivo que queremos predecir (SalePrice) para ver cómo se distribuyen los valores por cada valor de las variables numéricas.
(Consultar en el Anexo de Imágenes, Ilustración 3: Ejemplo de Boxplot con la variable GrLivArea VS SalePrice)

4. **Matriz de correlación ordenada por pesos:**

Finalmente, para terminar con los gráficos, vamos a hacer una matriz de correlación de las variables numéricas a partir de su peso. Esta matriz nos permite ver las variables que más influyen en nuestra variable objetivo.
(Consultar en el Anexo de Imágenes, Ilustración 4: Matriz de correlación)

3. Preprocesamiento de datos

Lo primero que vamos a hacer es dividir en X (todas las variables que no son las objetivo e Id) e Y (variable objetivo SalePrice). A continuación, determinaremos la "seed" para que de igual las veces que se ejecute el programa siempre de los mismos resultados. Entonces ya podemos dividir nuestro dataset entre train (60%), validation (20%) y test (20%). Hemos escogido estos valores porque, por ejemplo, si dividíamos 70/15/15 no había suficientes variables únicas en las partes de validation y test para hacer cosas como el One-Hot encoding, es decir, si una variable tiene 3 posibles valores, con esa partición en validation o test a veces no llegaban esos 3 valores.

Modificación de datos en train

Para el escalado de los pasos aplicados y calculados en train no he hecho recipes, ya que lo quería hacer de forma más manual para que se viese más claro los pasos que estaba siguiendo y la jerarquía aplicada que había escogido.

Variables inútiles

Vamos a empezar por sacar en formato de porcentaje los NA que tiene cada variable y, a partir de los resultados, nos vamos a quedar con los nombres de las variables que tengan un porcentaje mayor a 93, ya que para una variable que solo aporta el 7%, como máximo, de datos, según mi criterio, la podemos eliminar. Estas son las variables escogidas y sus porcentajes:

- **Alley**: 93.84966
- **PoolQC**: 99.54442
- **MiscFeature**: 96.58314

$$\text{Porcentaje de NA} = \frac{\sum_{i=1}^{n} \mathbb{1}(y_i \text{ es NA})}{n} \times 100$$

El otro filtraje agresivo que se ha hecho ha sido mirando variables que son o están muy cerca de ser constantes, es decir, que la variable siempre tenga el mismo valor. En este caso, el umbral de porcentaje que he decidido ha sido mayor (98%). Estas son las variables escogidas con sus valores (tener en cuenta que X_train tiene 878 valores):

- **Street**: Grvl (4), Pave (874)
- **Utilities**: AllPub (878)
- **Condition2**: Artery (2), Feedr (5), Norm (868), PosA (1), RRAn (1), RRNn (1)

- **LowQualFinSF**: 0 (862), 53 (1), 80 (2), 120 (1), 156 (1), 205 (1), 232 (1), 371 (1), 384 (1), 392 (1), 397 (1), 473 (1), 479 (1), 513 (1), 528 (1), 572 (1)
- **X3SsnPorch**: 0 (861), 96 (1), 130 (1), 144 (2), 168 (2), 180 (2), 182 (1), 196 (1), 216 (2), 238 (1), 290 (1), 304 (1), 407 (1), 508 (1)
- **PoolArea**: 0 (874), 512 (1), 555 (1), 576 (1), 738 (1)

Ahora ya eliminamos las variables de nuestro "train", ahora si utilizamos la función dim() en X_train obtendremos 878 datos y 70 variables.

Codificación de variables categóricas

Vamos a separar las variables categóricas en tres grupos, según la codificación que le vayamos a aplicar:

- Ordinales: Si las variables tienen cierto orden, por ejemplo: Po, Fa, TA, Gd y EX, en este caso representan un orden de calidad, de pobre (Po) a excelente (EX).
- Nominales: Si no tienen un tipo de orden claro, o no tienen y no son mayores de 8 valores por variable: por ejemplo: IR1, IR2, IR3 y Reg
- Extra: Son las restantes que no tienen ningún orden, pero tienen más de 8 valores.

A las variables ordinales les vamos a aplicar una codificación ordinal, es decir, vamos a sustituir los valores que tiene por simples números en orden, cogiendo el caso anterior sería: Po -> 1, Fa -> 2, TA -> 3, Gd -> 4 y EX -> 5. Para que estos valores se apliquen con el orden que nosotros queremos, vamos a hacer una tabla auxiliar donde le daremos los valores por orden a la variable que corresponda.

$$\text{Codificación Ordinal}(C_i) = i, \quad i = 1,2,\dots,k$$

A las variables nominales les haremos una codificación One-Hot, lo que hará que variables que tengan, por ejemplo, dos valores posibles se creen dos tablas donde solo se puede obtener el estado 0 o 1 y variar estos estados para determinar que valor obtiene cada fila, por ejemplo, con la variable CentralAir, la cual puede obtener dos valores Y y N:

Y	N
1	0
0	1

$$\text{OneHot}(x) = [\delta(x = v_1) \quad \delta(x = v_2) \quad \dots \quad \delta(x = v_n)]$$

Finalmente, para las variables restantes aplicaremos una codificación por frecuencia, es decir, calculamos que frecuencia tiene cada valor dentro de la variable y sustituimos ese valor por el número de su frecuencia: Si en una variable hay un 60% de Y, 25% de X y 15% de Z se le daría a cada uno el valor en decimales (X -> 0.25, Y -> 0.6, Z -> 0.15%).

$$\text{Frequent Encoding}(C_i) = \frac{\text{count}(C_i)}{n}$$

Codificación de variables numéricas

En el caso de las variables numéricas hay muy pocas que tengan valores “NA”, por lo que podemos mirarlas una a una para determinar cómo es la mejor manera de codificarlas. Solo tenemos tres variables numéricas que tengan “NA” (LotFrontage, MasVnrArea y GarageYrBlt). A MasVnrArea y GarageYrBlt vamos a codificarlas de manera que si el valor es “NA” se transforme a 0, ya que en sí si no tiene valor significa, respectivamente, que no está hecha de ladrillos y que no tiene garaje. LotFrontage la vamos a codificar en base a la media de la variable Neighborhood, ya que el espacio que tienen en frente de la casa va relacionado directamente con el barrio al que pertenece.

Codificación de variables numéricas

Hay unos casos en especifico que vamos a eliminar para que nuestro modelo entienda bien la relación mas importante entre SalePrice y GrLivArea, si no hacemos este cambio en el train nuestro modelo va a empeorar drasticamente. Vamos a eliminar todas las filas que entren en la condición de SalePrice < 300000 y GrLivArea > 4500. Justo en nuestra partición, como se puede ver en la ejecución del código, no han caído ninguno de estos valores en el train, por eso no eliminamos nada. Este cambio solo se hace en el train.

Escalado a test y validation

Finalmente aplicamos todos estos cambios a las particiones de test y validation utilizando los datos investigados en la partición de train. No voy a utilizar recipes, ya que quiero ver todos los cambios que hago y como se van aplicando a test y validation. Este paso es bastante simple una vez se entiende lo que se ha hecho en el train, ya que es una simple replica a las dos otras particiones.

Normalización

Para normalizar sí que haremos una simple recipe, que normalice y centre, entrenada con la partición de train y se la aplicaremos con la función bake() a train, validation y test.

4. Princiapl Component Analysis (PCA)

Para el PCA sobre el train vamos a utilizar la función prcomp() con el "center" a false y el "scale" a false también, ya que ya lo hemos centrado y escalado en los apartados anteriores. A continuación, calcularemos la varianza explicada acumulada para observar cuantas variables mínimas necesitamos para "explicar" nuestro dataset, en la gráfica vamos a añadir una línea para ver cuantas necesitamos para el 90%.

(Consultar en el Anexo de Imágenes, Ilustración 5: PCA Plot)

$$\text{Varianza explicada acumulada} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i}$$

Para seleccionar la mejor cantidad de variables (k) que utilizar para el entreno de nuestra regresión lineal, vamos a hacer un for() con todas las posibles variables para ver el resultado que da nuestra métrica elegida (RMSE) con cada valor. Vamos a representar estos resultados en formato de gráfica para poder observar el output. Al igual que en la gráfica anterior vamos a marcar con una línea el mejor resultado.

(Consultar en el Anexo de Imágenes, Ilustración 6: Mejor K para el PCA)

Una vez ya tenemos el mejor valor de K para nuestra regresión lineal vamos a entrenar el modelo. (Los resultados de cada modelo están al final del documento)

5. Least Absolute Shrinkage and Selection Operator (LASSO)(L1)

Para Lasso simplemente cogemos las matrices ya creadas y modificadas y se las pasamos al modelo. Vamos a definir el alpha en 1 para decirle a la función glmnet() que estamos utilizando Lasso. Antes de hacer el train hay que mirar cual es la mejor lambda para Lasso, para eso creamos un for(), al igual que para el PCA, que devuelva los valores de la métrica RMSE por cada lambda. Vamos a sacar un gráfico para ver los resultados de la métrica y el logaritmo de lambda.

(Consultar en el Anexo de Imágenes, Ilustración 7: Mejor Lambda para Lasso)

Para Lasso también se ha hecho la versión de Lasso + PCA, donde simplemente se hacen los dos pasos por partes: Primero se hace el PCA sobre el dataset y a continuación se aplica la penalización de Lasso.

$$\text{Lasso}(\beta) = \arg\min_{\beta}\left(\frac{1}{2n}\sum_{i=1}^{n}(y_i - X_i\beta)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\right)$$

6. Ridge Regression (Ridge)(L2)

Para Ridge hacemos exactamente lo mismo que hemos hecho en Lasso, excepto que en la función de glmnet el alpha es 0:

Se han cogido los valores obtenidos de la métrica RMSE y se han comparado con cada lambda que se le puede aplicar a Ridge. Se ha hecho una gráfica para ver los resultados.

(Consultar en el Anexo de Imágenes, Ilustración 8: Mejor Lambda para Ridge)

Para Ridge también se ha hecho la versión de Ridge + PCA, donde simplemente se hacen los dos pasos por partes: Primero se hace el PCA sobre el dataset y a continuación se aplica la penalización de Ridge.

$$\text{Ridge}(\beta) = \arg\min_{\beta}\left(\frac{1}{2n}\sum_{i=1}^{n}(y_i - X_i\beta)^2 + \lambda\sum_{j=1}^{p}\beta_j^2\right)$$

7. Resultados finales

A continuación, se muestra una tabla con los resultados finales sobre validation y test con diferentes modelos (PCA, Lasso, Ridge, Lasso + PCA y Ridge + PCA) utilizando diferentes métricas (RMSE, RAE y R^2). Los resultados están ordenados por el resultado de R^2 de test.

$$\boldsymbol{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$\boldsymbol{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

(Consultar en el Anexo de Imágenes, Ilustración 5: Resultados finales de las métricas sobre los modelos)

También se obtiene una tabla mediante el código en R:

Modelo	RMSE_Validation	RMSE_Test	MAE_Validation	MAE_Test	R2_Validation	R2_Test
PCA + Linear	32951.12	43214.39	20870.83	22416.93	0.75	0.76
LASSO	33626.87	46629.41	21298.96	23624.26	0.74	0.72
RIDGE	31023.42	42394.21	19292.57	22029.83	0.78	0.77
LASSO + PCA	33492.00	43173.45	20773.10	22206.80	0.74	0.76
RIDGE + PCA	31023.78	42414.59	19358.13	22138.10	0.78	0.77

8. Conclusiones

De forma general, los modelos Ridge y Ridge + PCA presentan el mejor rendimiento global, tanto en validación como en test. Ambos alcanzan:

- Menor RMSE (≈ 42 400 en test),
- Menor MAE (≈ 22 000 en test),
- Mayor coeficiente de determinación ($R^2 \approx 0.77$).

Esto indica una mejor capacidad de generalización y una mayor estabilidad frente al sobreajuste en comparación con el resto de los modelos evaluados.

La comparación entre modelos sin y con regularización permite extraer varias conclusiones:

- LASSO presenta un rendimiento inferior a Ridge, tanto en RMSE como en R^2.
- Lo que sugiere que, en este problema, la selección agresiva de variables propia de la penalización L1 elimina información algo relevante, ya que no varía tanto el valor final, afectando negativamente a la capacidad predictiva.
- En cambio, Ridge (L2), al penalizar los coeficientes sin forzar su anulación, maneja mejor la multicolinealidad existente entre predictores numéricos y categóricos codificados.

El uso de Análisis de Componentes Principales (PCA) muestra efectos distintos según el modelo base:

- En el caso de regresión lineal simple, el PCA mejora ligeramente la estabilidad respecto a una regresión directa en alta dimensión, obteniendo valores razonables de R^2 (≈ 0.76 en test).
- Sin embargo, al combinar PCA con Ridge, no se observa una mejora significativa respecto a Ridge sin PCA.
- Los valores prácticamente idénticos entre Ridge y Ridge + PCA indican que la regularización L2 ya es suficiente para controlar la multicolinealidad, haciendo redundante la reducción de dimensionalidad previa.

Por tanto, el PCA aporta más valor cuando el modelo no incorpora regularización explícita.

9. Mejoras

Se podría haber utilizado recipes en todos lados para un traspaso más fácil de los cambios realizados en la partición del train hacia las otras dos. Con mas inspección se podrían haber observado mas casos únicos que hagan que mejoren los resultados finales de las métricas.