

Predicción de precios de coches

David Bolea
Marc Domènech

Introducción	2
Análisis de los datos	3
Price	4
Odometer	5
Year	6
Cylinders	7
Condition	8
Columnas restantes	9
Relación de las variables	10
Categorías	10
Numéricas	11
Codificación de las variables categóricas	11
Normalización	12
Features	13
Métodos de Resolución	13
Linear, Ridge y Lasso Regression	14
KNN	14
MLP	14
SVM	15
Decision Tree, Random Forest y Extra Trees	16
Reducción de Dimensionalidad (PCA)	16
Optimización de los modelos	17
Ensamblajes	18
Conclusión	19

Introducción

El objetivo de este trabajo es estudiar cómo se relacionan los precios de los coches con características como pueden ser el año de fabricación, los kilómetros recorridos, su estado, etc...

En primer lugar analizaremos los datos de cada característica y detectaremos aquellos valores anómalos (nulls y outliers). Una vez hayamos solucionado estos problemas, nos encargaremos de estudiar las relaciones que existen entre las distintas variables. De esta forma podremos decidir qué características son más importantes a la hora de predecir el precio de un coche.

Dado el tamaño de nuestro dataset, una vez los datos ya estén preprocesados podremos permitirnos crear una partición de train para entrenar nuestros modelos, una partición de validación para poderlos comparar y finalmente una de test para poder presentar los resultados finales.

Para encontrar el modelo que más se ajusta a nuestro problema usaremos modelos de distintas familias para poder tener una intuición de qué tipo de algoritmos tenemos que usar. En nuestro caso probaremos modelos lineales y no lineales.

Finalmente analizaremos el modelo o los modelos que más se ajusten a nuestro problema y discutiremos el proceso que hemos seguido a la hora de realizar la práctica y de cómo hemos afrontado los distintos problemas que se nos han planteado.

Análisis de los datos

Para este proyecto disponemos de un dataset con variables tanto numéricas como categóricas y es por ello que tendremos que ir con cuidado a la hora del preprocesado de datos.

Solo al ver el dataset ya nos damos cuenta de que hay atributos que no son necesarios para predecir el precio y otros que tienen demasiados valores nulos como para tenerlos en cuenta. Estos atributos son los siguientes:

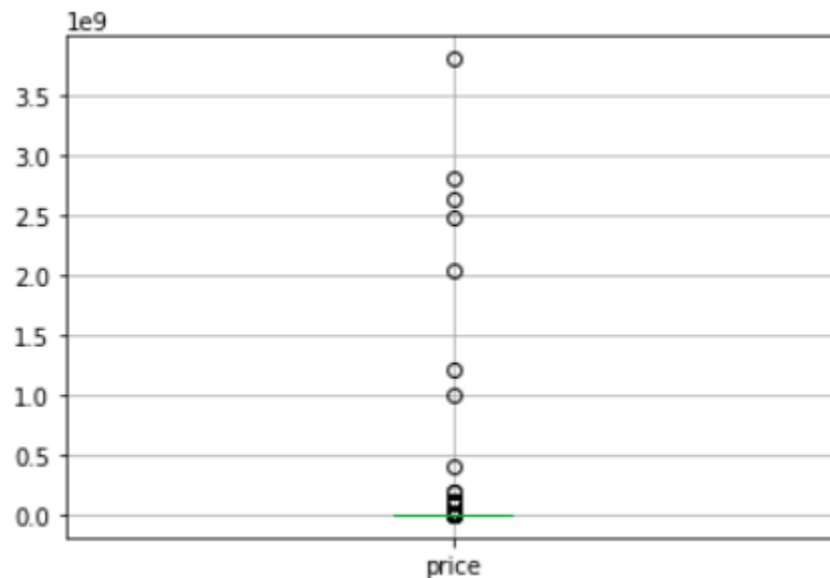
id, url, region_url, title_status, vin, size, paint_color, image_url, description, county, state, lat, long y region.

Una vez descartados los atributos innecesarios, lo primero que hacemos es mostrar cuántos valores nulos hay en cada uno de los atributos del dataset.

	null	percent
condition	24713800	58.307
cylinders	22617800	53.362
drive	19273800	45.472
type	18270000	43.104
odometer	15327200	36.161
manufacturer	11061500	26.097
model	9847300	23.233
fuel	9664300	22.801
transmission	9579200	22.600
year	9511400	22.440
price	0	0.000

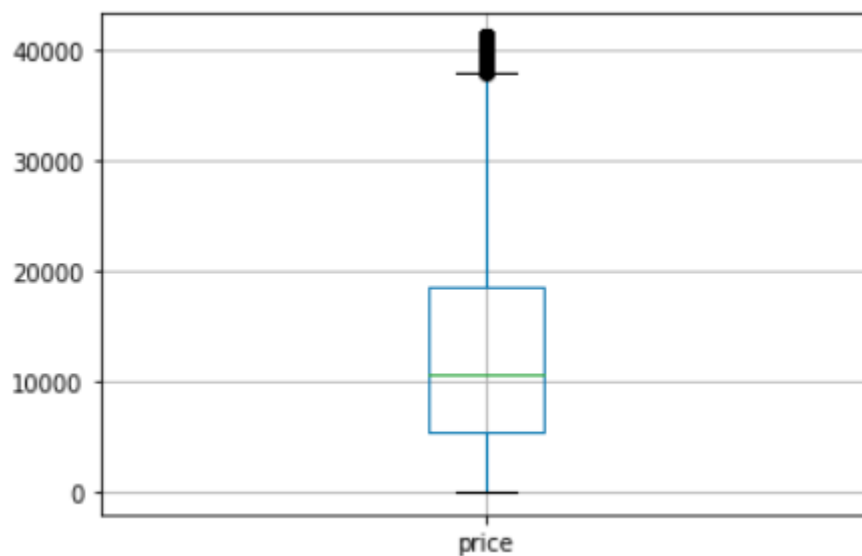
Price

El atributo precio representa el precio que tiene el coche en cuestión. Este atributo será el target que intentaremos predecir. Su distribución antes de aplicar el preprocesado es la siguiente:

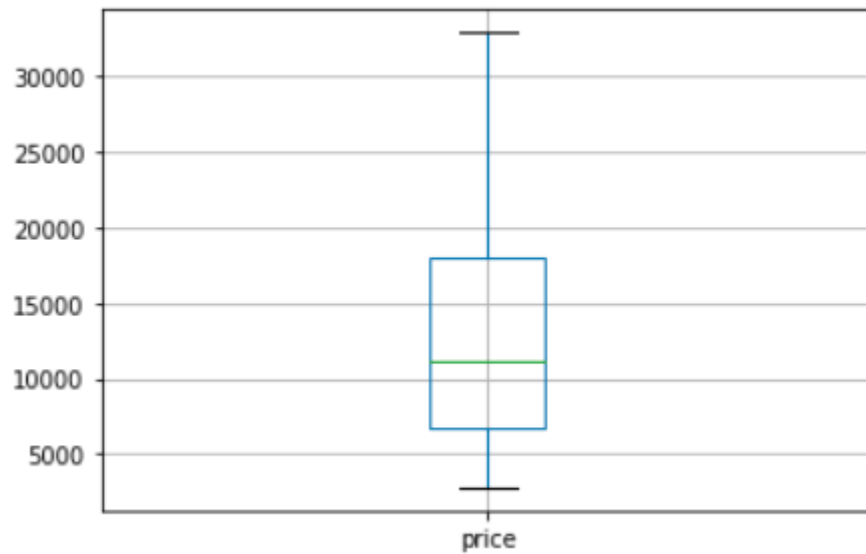


Podemos observar como por un lado tenemos demasiados outliers, por lo tanto, intentaremos aplicar un IQR para ver si podemos solucionar el problema.

Por el otro lado vemos que hay 35.025 coches que tienen un precio de 0€. Estos coches hemos decidido eliminarlos ya que consideramos que son valores sin sentido.

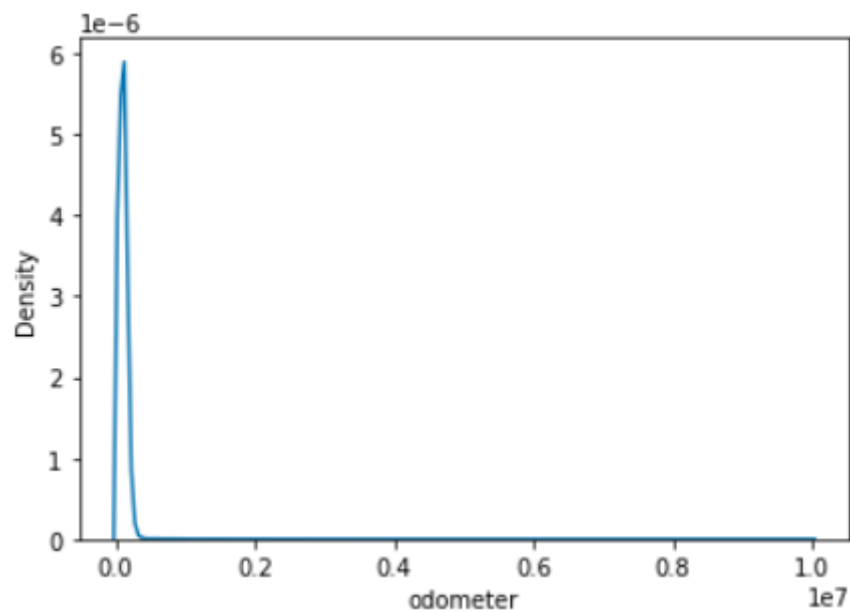


Después de eliminar los coches que tenían como precio 0€, vemos como el IQR no ha acabado de solucionarnos el problema de los outliers. Por esta razón hemos decidido aplicar los percentiles 90-10 para quedarnos con los datos que hemos considerado que son más relevantes.

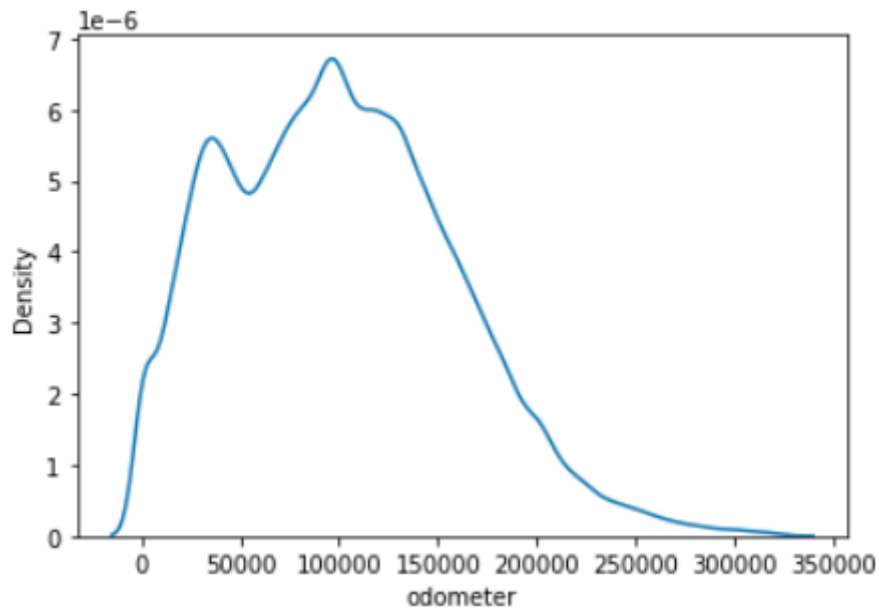


Odometer

Odometer, representa los kilómetros recorridos por el coche. Para esta columna hemos dibujado un kdeplot que nos muestra la distribución de los datos.

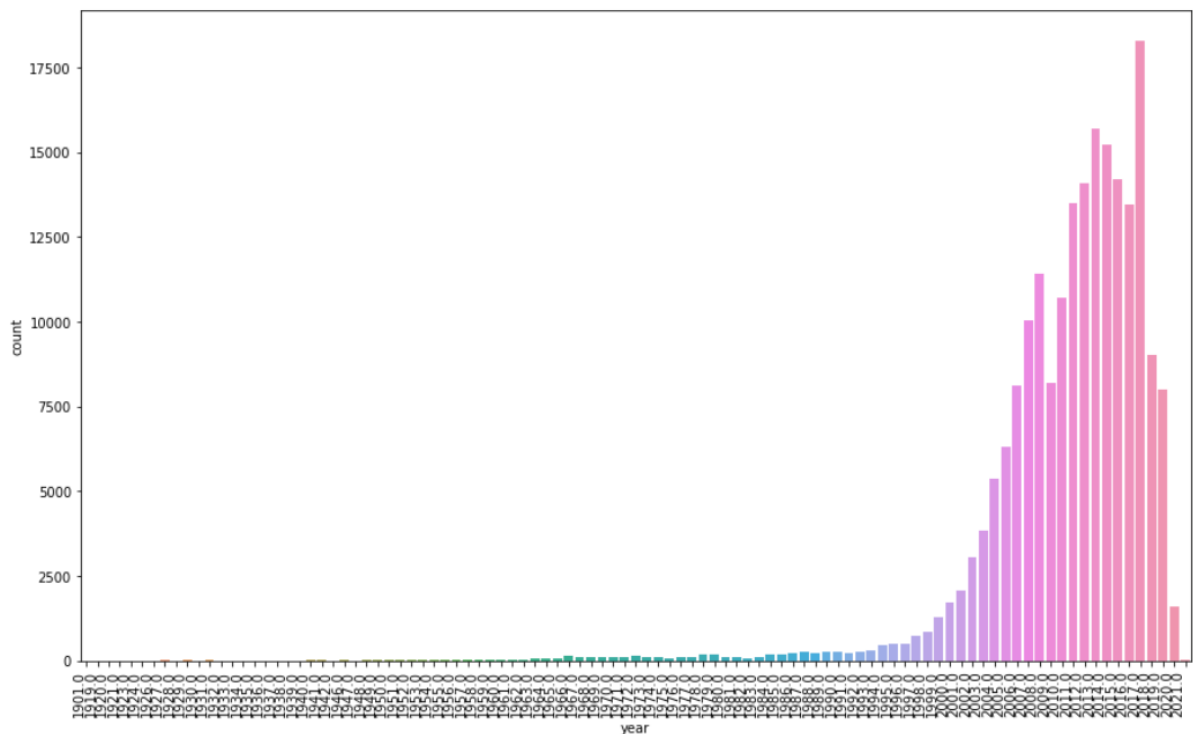


Como podemos observar tenemos bastantes outliers en este atributo. Viendo la gráfica, nos damos cuenta de que la mayor parte de los coches han recorrido menos de unos 325.000km y es por eso que hemos decidido cortar y quedarnos con todos los valores menores a 325.000. Además, valores tan elevados de odometer no son reales.

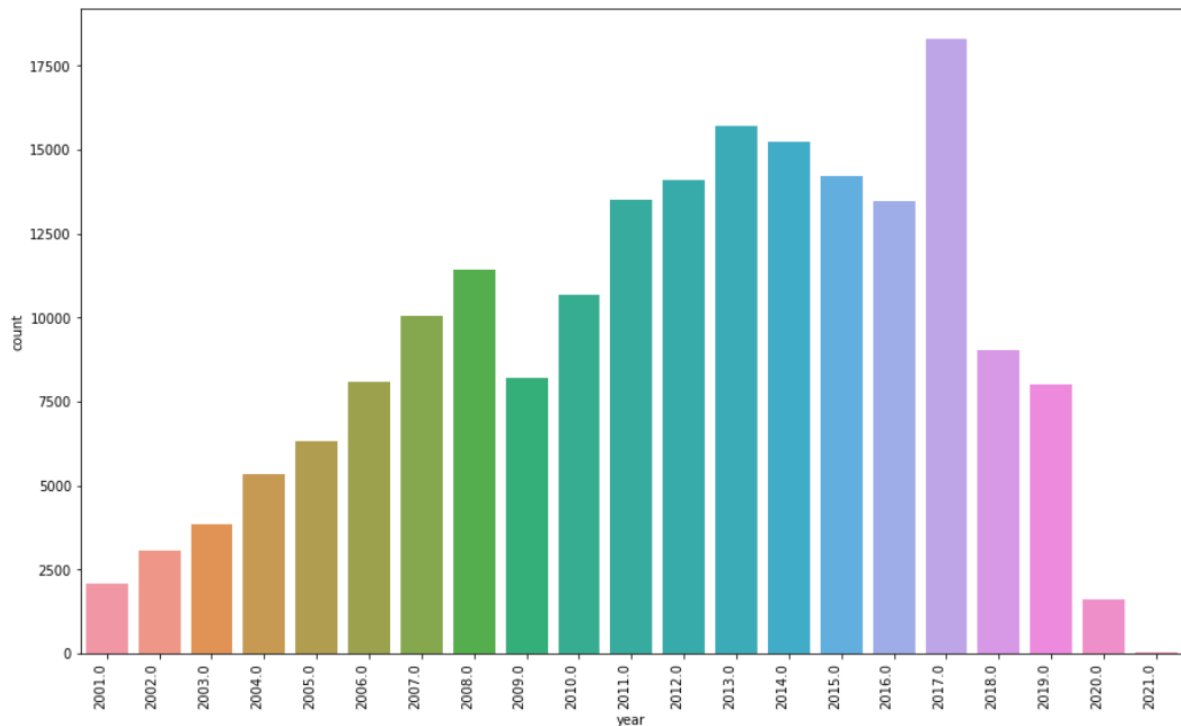


Year

Year, representa el año de fabricación del automóvil. La distribución de este atributo es la siguiente:



Viendo las distribución de los datos, podemos ver como la gran mayoría de los datos los podemos encontrar a partir del año 2000 hacia adelante. Por este motivo hemos decidido cortar, y eliminar los anteriores a esta fecha.



Cylinders

Este atributo representa el número de cilindros que tiene el coche. Es un atributo categórico, por eso vamos a mostrar qué valores puede tomar.

```

4 cylinders    51990
6 cylinders    45457
8 cylinders    26979
5 cylinders     1192
10 cylinders     478
other           315
3 cylinders      299
12 cylinders      33

```

Lo primero que hemos hecho ha sido eliminar la palabra cylinder de nuestros valores, es decir, hemos pasado de tener '6 cylinders' a tener '6'.

Seguidamente nos hemos quedado con las muestras que tienen un valor definido de cilindros y, por lo tanto, desechar todas las que tenían el valor 'other'. Hemos decidido tomar esta decisión debido a que 'other' puede ser cualquier valor e intentar imputarlo sería inventarnos posibles valores.

Finalmente, debido a que intentamos imputar los valores nulos mediante ffill y nos dio unos resultados nefastos, decidimos hacer un drop de estos valores y así conseguimos un valor de predicción bastante mejor.

4	51990
6	45457
8	26979
5	1192
10	478
3	299
12	33

Condition

Este atributo se refiere a en qué condiciones se encuentra el coche. Su valor es categórico.

excellent	46562
good	31170
like new	9683
fair	1221
new	259
salvage	82

En el momento de tratar la variable condition nos encontramos con un problema. Al principio de todo, cuando calculamos el porcentaje de nulos de cada clase, obtuvimos que condition tenía casi un 60% de valores nulos y una de las opciones que podemos tomar es eliminar este atributo. Por otro lado, pensamos que no debemos hacer eso ya que la condición de un coche es una característica importante a la hora de ponerle un precio.

Finalmente como sabemos que la condición de un coche guarda una gran relación con la columna odometer, ya que contra mas kilometros recorre un coche más fácil es que un coche se encuentre en peor estado, utilizamos la variable odometer para imputar los valores nulos de condition.

excellent	46562
good	31170
new	23426
fair	15505
like new	9683
salvage	82

Columnas restantes

A estas alturas ya hemos preprocesado todas aquellas columnas que tenían más missings. Por tanto, dado el tamaño de nuestro dataset, nos podemos permitir eliminar todos los nulos restantes debido a que las columnas que nos quedan por procesar tienen un porcentaje muy pequeño de estos. Es por ello que eliminaremos los nulos de las columnas siguientes:

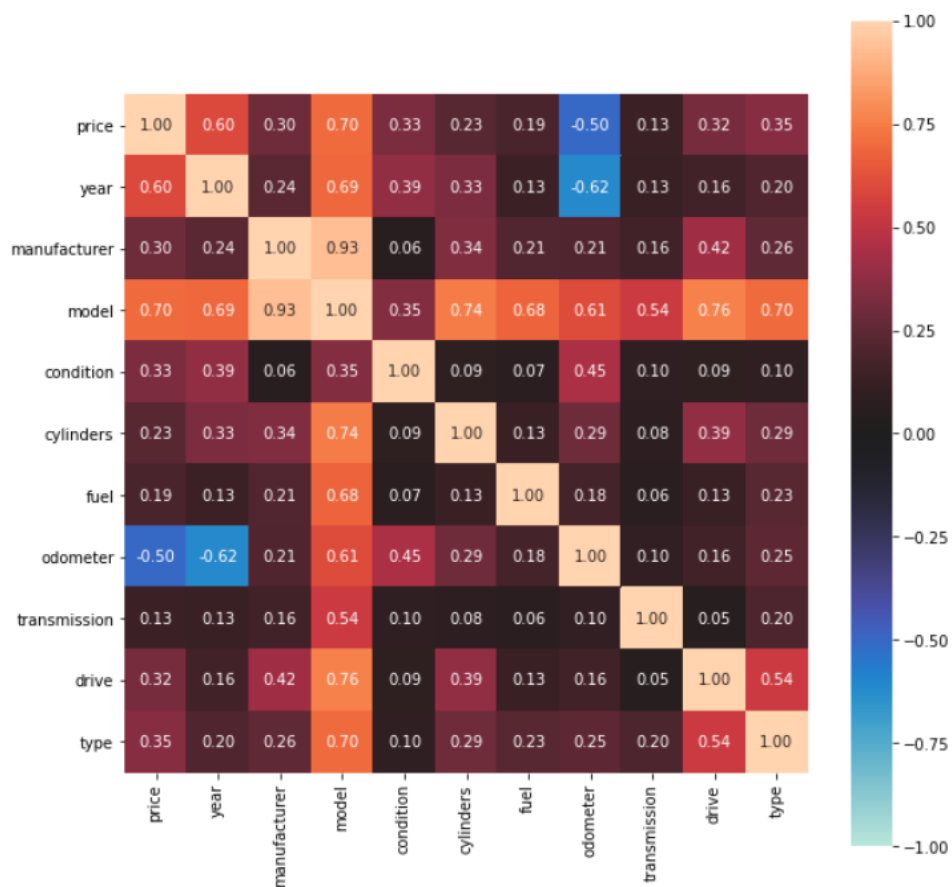
- Fuel
- Transmission
- Model
- Manufacturer
- Type
- Drive

Relación de las variables

En este apartado estudiaremos como de relacionadas están nuestras variables respecto al precio. Lo dividiremos en dos partes: variables categóricas y numéricas.

Categóricas

Para poder ver como de relacionadas están las variables categóricas respecto al precio, utilizaremos una tabla de asociaciones. La manera de interpretar esta tabla es mirando la primera columna, y contra más cercano esté el valor a 1 o a -1, más relacionada está esa variable con el precio del coche.

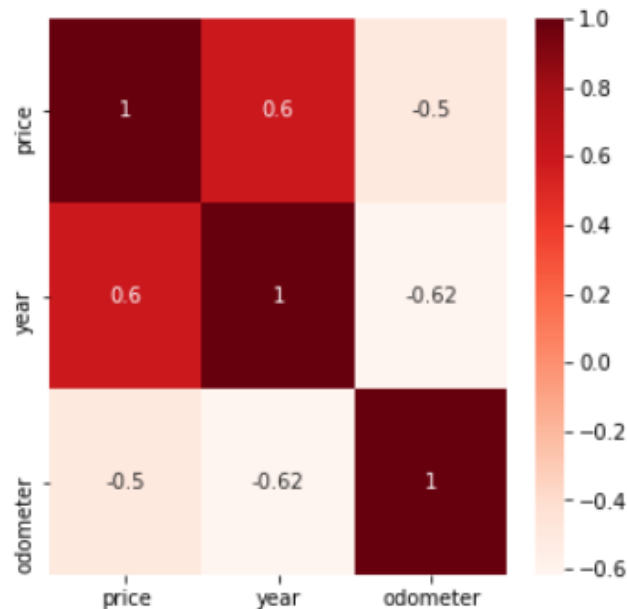


Por lo tanto, una vez analizada la tabla, podemos decir que las variables más relevantes a la hora de predecir el precio de un coche son:

- Model
- Condition
- Drive
- Type
- Manufacturer

Numéricas

En el caso de las variables numéricas podemos observar como de relacionadas están respecto al precio utilizando un heatmap. La forma de interpretar esta tabla es la misma que en el caso de las categóricas, contra más cerca de 1 o -1 esté el valor, más relacionadas están las variables.



Por lo tanto, como podemos observar, en este caso todas las variables numéricas están altamente relacionadas con el precio.

Codificación de las variables categóricas

Para transformar las variables categóricas en numéricas hemos decidido aplicar One Hot Encoding a aquellas variables que tengan pocos valores distintos y para las columnas que tienen muchos más valores posibles hemos aplicado un Label Encoding.

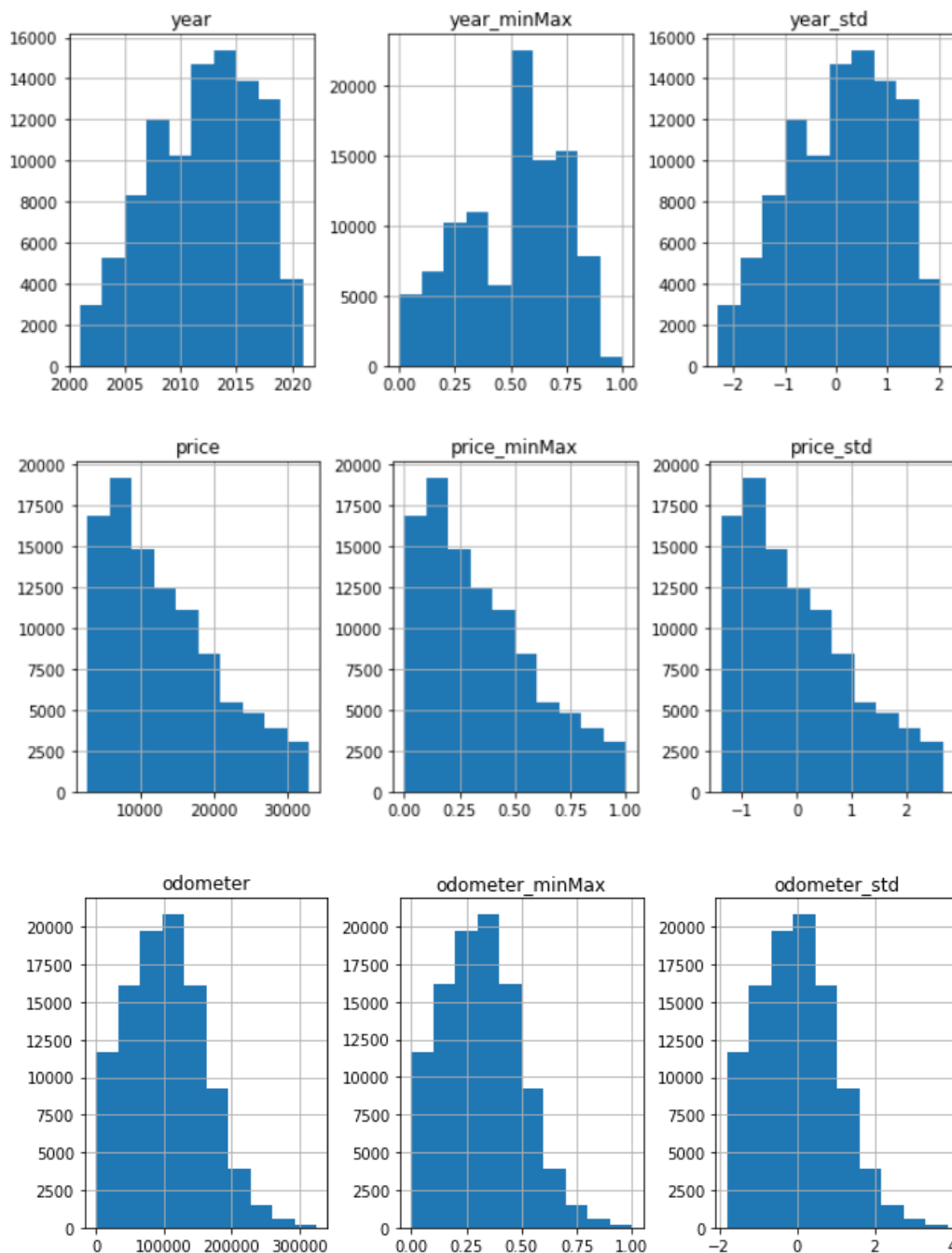
En nuestro caso, debido a que por ejemplo el modelo del coche (model) tiene más de 27 mil valores diferentes, aplicar one hot encoding a esta característica nos generaría más de 27 mil columnas y haría que nuestro dataset tuviera un tamaño demasiado grande.

Es por esto que una vez consultados cuantos valores puede tomar cada variable categórica, hemos decidido aplicar Label Encoding a manufacturer, model y type. Por otro lado, hemos decidido aplicar one hot encoding a condition y drive.

Normalización

Dado que tenemos rangos de datos muy distintos, por ejemplo entre el modelo del coche (model) y el año de fabricación (year), es necesario aplicar algún tipo de normalización/estandarización de los datos si queremos usar modelos que usen distancias euclídeas como por ejemplo el KNN.

Primero de todo aplicamos MinMaxScaler y StandardScaler tanto a price, year y odometer para poder observar cual sería la mejor opción y estos son los resultados:



Como podemos observar, cualquiera de las dos transformaciones mantiene la distribución de los datos y por lo tanto, decidimos usar MinMaxScaler junto a los atributos model y manufacturer.

Features

Finalmente después del preprocesado y de mirar como de relevantes son cada una de las variables, obtenemos las columnas que usaremos para predecir el precio:

- Year
- Manufacturer
- Model
- Odometer
- Type
- Condition
- Drive

Métodos de Resolución

Para resolver el problema que planteamos, hemos decidido plantearlo como un problema de regresión, ya que nos parece interesante intentar predecir el precio exacto de un coche. Hemos tomado esta decisión porque creemos que tiene más sentido considerar el precio como una variable continua y no como una variable discreta.

Para resolver el problema primero probaremos modelos de distintas familias para ver si hay alguno de ellos que parezca ajustarse más a nuestro problema de entrada. Una vez tengamos unos resultados base, cogeremos los mejores modelos e intentaremos tunearlos para ver si podemos conseguir mejores resultados. En nuestro caso probaremos modelos lineales y no lineales.

Para poder llevar a cabo la práctica, dividiremos nuestro dataset en una partición de Train (60%) para poder entrenar nuestros modelos, una partición de Validación (20%) para poder comparar los resultados entre modelos y una partición de Test (20%) para poder mostrar el resultado final con el mejor modelo. Para comparar los modelos, principalmente nos fijaremos en la métrica del R^2 . De todos modos calcularemos el Mean Squared Error, el Mean Absolute Error y el Median Absolute Error para poder tener más información.

Linear, Ridge y Lasso Regression

Primero de todo probamos con modelos muy sencillos para ver si pueden ser una posible solución. Instanciamos los 3 modelos con los parámetros por defecto. En el caso del Ridge le hemos pasado un conjunto de alphas.

	R2	Mean_Square_Error	Median_Absolute_Error	Mean_Absolute_Error	Time
Linear Regression	0.27	0.0263	0.0999	0.124	0.0462
Ridge Regression	0.27	0.0263	0.0999	0.124	11.2
Lasso Regression	0.268	0.0263	0.0997	0.124	1.71

Podemos ver como las regresiones lineales no parecen ofrecernos grandes resultados así que seguiremos buscando

KNN

El segundo modelo a probar será un KNN. Lo instanciamos con una k de 5 vecinos para empezar.

	R2	Mean_Square_Error	Median_Absolute_Error	Mean_Absolute_Error	Time
Linear Regression	0.27	0.0263	0.0999	0.124	0.0462
Ridge Regression	0.27	0.0263	0.0999	0.124	11.2
Lasso Regression	0.268	0.0263	0.0997	0.124	1.71
KNN-5	0.771	0.012	0.0446	0.0717	1.44

Podemos ver como este modelo ya nos da resultados bastante buenos, así nos lo guardaremos como posible candidato a resolver el problema.

MLP

En este caso probaremos un par de MLP con distintas arquitecturas para ver si nos puede ofrecer resultados decentes. En primer lugar probaremos una arquitectura de [1] y posteriormente probaremos una arquitectura de [8,8,8], sin regularización y con una función de activación logística.

	R2	Mean_Square_Error	Median_Absolute_Error	Mean_Absolute_Error	Time
Linear Regression	0.27	0.0263	0.0999	0.124	0.0462
Ridge Regression	0.27	0.0263	0.0999	0.124	11.2
Lasso Regression	0.268	0.0263	0.0997	0.124	1.71
KNN-5	0.771	0.012	0.0446	0.0717	1.44
MLP-1	-4.13e+03	0.0615	0.194	0.206	0.0686
MLP-8-8-8	-3.82e+04	0.0615	0.195	0.206	0.311

Podemos ver como el resultado de las MLP es muy malo. El modelo KNN es un modelo muy sencillo de calcular y nos da resultados mucho mejores, es por eso que no perderemos mucho el tiempo intentando optimizar una MLP. Así que seguiremos probando modelos nuevos que puedan competir con el KNN.

SVM

Ahora probaremos con las SVM, unos modelos muy potentes que podrían ofrecernos resultados muy buenos. Instanciamos un modelos SVM por defecto para ver que resultados puede ofrecernos.

	R2	Mean_Square_Error	Median_Absolute_Error	Mean_Absolute_Error	Time
Linear Regression	0.27	0.0263	0.0999	0.124	0.0462
Ridge Regression	0.27	0.0263	0.0999	0.124	11.2
Lasso Regression	0.268	0.0263	0.0997	0.124	1.71
KNN-5	0.771	0.012	0.0446	0.0717	1.44
MLP-1	-4.13e+03	0.0615	0.194	0.206	0.0686
MLP-8-8-8	-3.82e+04	0.0615	0.195	0.206	0.311
SVM-default	0.47	0.0209	0.0784	0.106	157

Como podemos ver el resultado no puede competir contra el KNN. Además podemos ver como ha tardado 157 segundos en entrenar la SVM por defecto frente a los 1,44 segundos del KNN. Por este motivo no nos entretendremos demasiado en intentar tunear una SVM ya que muy rápidamente se nos disparará el tiempo de entrenamiento.

Decision Tree, Random Forest y Extra Trees

Finalmente probaremos de instanciar estos 3 modelos con los parámetros por defecto para ver si podemos por lo menos plantarle cara al KNN. En el caso del Decision Tree, lo configuraremos con un max depth de 2, ya que vemos que nos ofrece buenos resultados.


	R2	Mean_Square_Error	Median_Absolute_Error	Mean_Absolute_Error	Time
Linear Regression	0.27	0.0263	0.0999	0.124	0.0462
Ridge Regression	0.27	0.0263	0.0999	0.124	11.2
Lasso Regression	0.268	0.0263	0.0997	0.124	1.71
KNN-5	0.771	0.012	0.0446	0.0717	1.44
MLP-1	-4.13e+03	0.0615	0.194	0.206	0.0686
MLP-8-8-8	-3.82e+04	0.0615	0.195	0.206	0.311
SVM-default	0.47	0.0209	0.0784	0.106	157
DecisionTree-default	-0.251	0.0342	0.115	0.143	0.0636
RandomForest-default	0.875	0.00669	0.0274	0.0498	27.1
Extra Trees-default	0.861	0.00759	0.0218	0.0493	17.5

Como podemos ver el Decision Tree nos ha dado resultados nefastos pero por el otro lado, el Random Forest y el Extra Trees han superado ampliamente la puntuación del KNN. Por este motivo estos dos modelos se convertirán en los principales candidatos a resolver nuestro problema.

Finalmente , una vez probados todos los modelos por defecto, podemos decir que los principales modelos a estudiar serán el Random Forest, Extra Trees y el KNN.

Reducción de Dimensionalidad (PCA)

Dado que nuestro dataset es muy grande, probaremos si podemos reducir su dimensionalidad mediante un PCA.

```
 pca = PCA().fit(x_train)

n_components = (pca.explained_variance_ratio_.cumsum() < 0.9).sum()

print(f'Original Columns: {len(data.columns)}')
print(f'After PCA Transformation: {n_components}')
```

```
Original Columns: 15
After PCA Transformation: 0
```

Lamentablemente, después de probarlo vemos que después de aplicar PCA nuestro número de componentes es 0 y es por ello que nos hemos dado cuenta de que no es posible aplicar PCA a nuestros datos.

Optimización de los modelos

Una vez encontrados los modelos que aparentemente parecen funcionar mejor para nuestros datos, probaremos de tunear sus hyper parámetros para ver si podemos mejorar su puntuación.

Después de tunear los 3 modelos, obtenemos los siguientes resultados:

	R2	Mean_Square_Error	Median_Absolute_Error	Mean_Absolute_Error	Time
Linear Regression	0.27	0.0263	0.0999	0.124	0.0462
Ridge Regression	0.27	0.0263	0.0999	0.124	11.2
Lasso Regression	0.268	0.0263	0.0997	0.124	1.71
KNN-5	0.771	0.012	0.0446	0.0717	1.44
MLP-1	-4.13e+03	0.0615	0.194	0.206	0.0686
MLP-8-8-8	-3.82e+04	0.0615	0.195	0.206	0.311
SVM-default	0.47	0.0209	0.0784	0.106	157
DecisionTree-default	-0.251	0.0342	0.115	0.143	0.0636
RandomForest-default	0.875	0.00669	0.0274	0.0498	27.1
Extra Trees-default	0.861	0.00759	0.0218	0.0493	17.5
KNN-Opt-5	0.771	0.012	0.0446	0.0717	1.44
RandomForest-Opt	0.876	0.00666	0.0273	0.0497	54.6
Extra Trees Opt	0.862	0.00755	0.0219	0.0491	50.6

Podemos ver como apenas hemos conseguido mejorar sus puntuaciones. Por este motivo consideramos que el Random Forest se convierte en el favorito para solucionar nuestro problema.

Ensamblajes

Aun habiendo obtenido un muy buen resultado con el Random Forest, no nos quedaremos ahí, e intentaremos exprimir al máximo los modelos que hemos obtenido.

En este apartado ensamblamos los 3 modelos que mejor resultado nos han dado (RF, ET, KNN) para ver si podemos conseguir mejorar su puntuación.

En nuestro caso hemos probado un Voting Regressor, y un Stacking Regressor. En el caso del Stacking, lo hemos combinado con un Gradient Boosting Regressor. Los resultados conseguidos son los siguientes:

	R2	Mean_Square_Error	Median_Absolute_Error	Mean_Absolute_Error	Time
Linear Regression	0.27	0.0263	0.0999	0.124	0.0462
Ridge Regression	0.27	0.0263	0.0999	0.124	11.2
Lasso Regression	0.268	0.0263	0.0997	0.124	1.71
KNN-5	0.771	0.012	0.0446	0.0717	1.44
MLP-1	-4.13e+03	0.0615	0.194	0.206	0.0686
MLP-8-8-8	-3.82e+04	0.0615	0.195	0.206	0.311
SVM-default	0.47	0.0209	0.0784	0.106	157
DecisionTree-default	-0.251	0.0342	0.115	0.143	0.0636
RandomForest-default	0.875	0.00669	0.0274	0.0498	27.1
Extra Trees-default	0.861	0.00759	0.0218	0.0493	17.5
KNN-Opt-5	0.771	0.012	0.0446	0.0717	1.44
RandomForest-Opt	0.876	0.00666	0.0273	0.0497	54.6
Extra Trees Opt	0.862	0.00755	0.0219	0.0491	50.6
Voting Hard	0.86	0.00727	0.0313	0.0537	109
Stackyn	0.88	0.00655	0.0262	0.0492	228

Como podemos ver, apenas hemos podido mejorar el resultado con el Staking Regressor.

Conclusión

Finalmente, el modelo que hemos escogido para resolver nuestro problema ha sido el Random Forest con una precisión del 87,6%. Si bien es cierto que el Stacking Regressor ofrece un resultado ligeramente mejor, el Random Forest nos ofrece prácticamente el mismo resultado pero con mucho menos tiempo de entrenamiento. Por este motivo creemos que si este problema tuviera que resolverse en un escenario real lo mejor sería escoger un modelo con el que pudiéramos conseguir un equilibrio entre precisión y velocidad.