USED CARS DATASET

Análisis del dataset Used Cars.

Emilio Luna Mejías

28/01/2021

Master Big Data

INTRODUCCIÓN

El dataset utilizado es Used Cars Dataset, de Kaggle. Este dataset contiene información de gran cantidad de vehículos usados a la venta en Estados Unidos, incluyendo el precio, condición, marca, latitud/longitud y otros 18 atributos.

PREPARACIÓN DEL DATASET

Se ha considerado como ruido aquellos vehículos con un precio inferior a 1 millón de dólares, ya que en el dataset aparecen algunos vehículos con precios desorbitados que pueden deberse a algún tipo de error (por ejemplo una camioneta Chevrolet por 3 billones de dólares).

También se han eliminado los vehículos con precios demasiado bajos (hay un gran número de vehículos por 0 dólares con los que no tendría sentido trabajar).

```
cars = cars[cars$price < 1000000, ]
cars = cars[cars$price > 1, ]
```

Hay una gran cantidad de atributos que no resultan de interés (identificadores, URLs, etc), directamente suprimimos estas columnas.

```
cars$url = NULL
cars$id = NULL
cars$region_url = NULL
cars$VIN = NULL
cars$image_url = NULL
cars$description = NULL
cars$posting_date = NULL
```

ELIMINACIÓN DE VALORES NA

A continuación, eliminaremos los valores perdidos. En este paso hay que tener en cuenta

que algunas columnas tienen tantos valores perdidos que si los eliminamos podemos perder demasiados datos. Por ello hay que valorar en cada caso si es preferible eliminar las filas con valores NA o si es mejor no utilizar ese atributo. En este caso, el dataset cuenta con una cantidad de filas tan grande que la eliminación de estos valores no supone un problema, ya que igualmente íbamos a tener que reducirlo a una muestra más pequeña para poder trabajar.

```
# Numéricos
cars = cars[is.na(cars$year) == FALSE, ]
cars = cars[is.na(cars$odometer) == FALSE, ]
cars = cars[is.na(cars$lat) == FALSE, ]
cars = cars[is.na(cars$long) == FALSE, ]
# Categóricos
cars = cars[cars$manufacturer != "", ]
cars = cars[cars$model != "", ]
cars = cars[cars$fuel != "", ]
cars = cars[cars$title_status != "", ]
cars = cars[cars$transmission != "", ]
cars = cars[cars$condition != "", ] # 42% de missing values
cars = cars[cars$cylinders != "", ] # 37% de missing values
cars = cars[cars$size != "", ] # 70% de missing values
cars = cars[cars$drive != "", ] # 29% de missing values
cars = cars[cars$type != "", ] # 25% de missing values
cars = cars[cars$paint_color != "", ] # 31% de missing values
```

FACTORIZACIÓN DE ATRIBUTOS CATEGÓRICOS

```
cars$state = as.factor(cars$state)
cars$region = as.factor(cars$region)
cars$manufacturer = as.factor(cars$manufacturer)
cars$model = as.factor(cars$model)
cars$fuel = as.factor(cars$fuel)
cars$title_status = as.factor(cars$title_status)
cars$transmission = as.factor(cars$transmission)
cars$condition = as.factor(cars$condition)
cars$cylinders = as.factor(cars$cylinders)
cars$size = as.factor(cars$size)
cars$drive = as.factor(cars$drive)
```

```
cars$type = as.factor(cars$type)
cars$paint_color = as.factor(cars$paint_color)
```

Tras realizar todo el preprocesamiento hemos pasado de tener un dataset con 26 columnas y 458213 filas a tener 19 columnas y 83012 filas (aproximadamente un 18% de las filas del dataset original).

VISUALIZACIÓN Y ANÁLISIS EXPLORATORIO

MAPAS Y COORDENADAS GEOGRÁFICAS

Como el dataset incluye las coordenadas geográficas de los distintos coches, se ha considerado interesante aprovechar para realizar la visualización mediante mapas.

Lo primero es generar una variable “mapa” con las coordenadas del mapa.

```
library(ggmap)
library(tmaptools)
map = ggmap(get_stamenmap(c(-126, 24, -65, 49), zoom = 4))
```

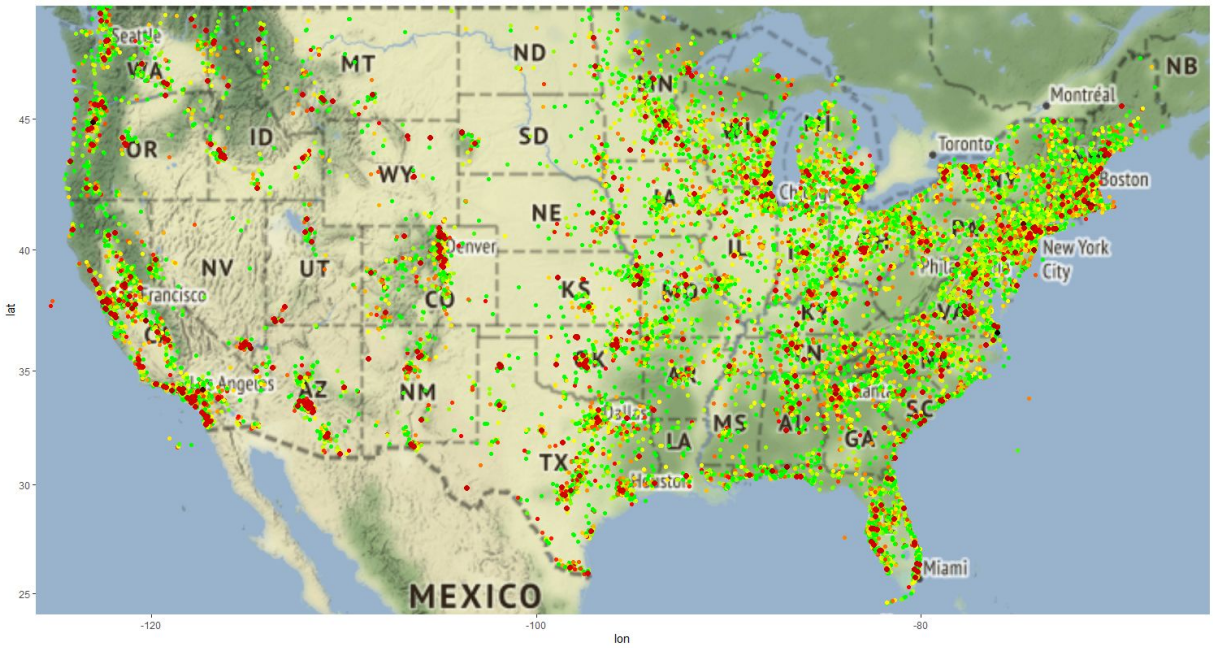
Lo siguiente es crear una paleta de colores para representar cada punto de un color en función de su precio de venta. Como existe un pequeño porcentaje de coches cuyo precio es demasiado elevado respecto del resto, pueden hacer que la escala se distorsione, por ello se ha creado una escala independiente para aquellos coches que cuestan más de 50000$, diferenciando en dos subconjuntos: “cheap_cars” y “expensive_cars”. Los primeros se representan con colores verde, amarillo y rojo. Los segundos van de rojo a negro, además para diferenciarlos estos puntos se representan con un mayor tamaño.

```
map +
  geom_point(data=cheap_cars,
             aes(x=long, y=lat),
             color=cheap_color)  +
  geom_point(data=expensive_cars,
             aes(x=long, y=lat),
             color=expensive_color,
             size=2)

map +
```

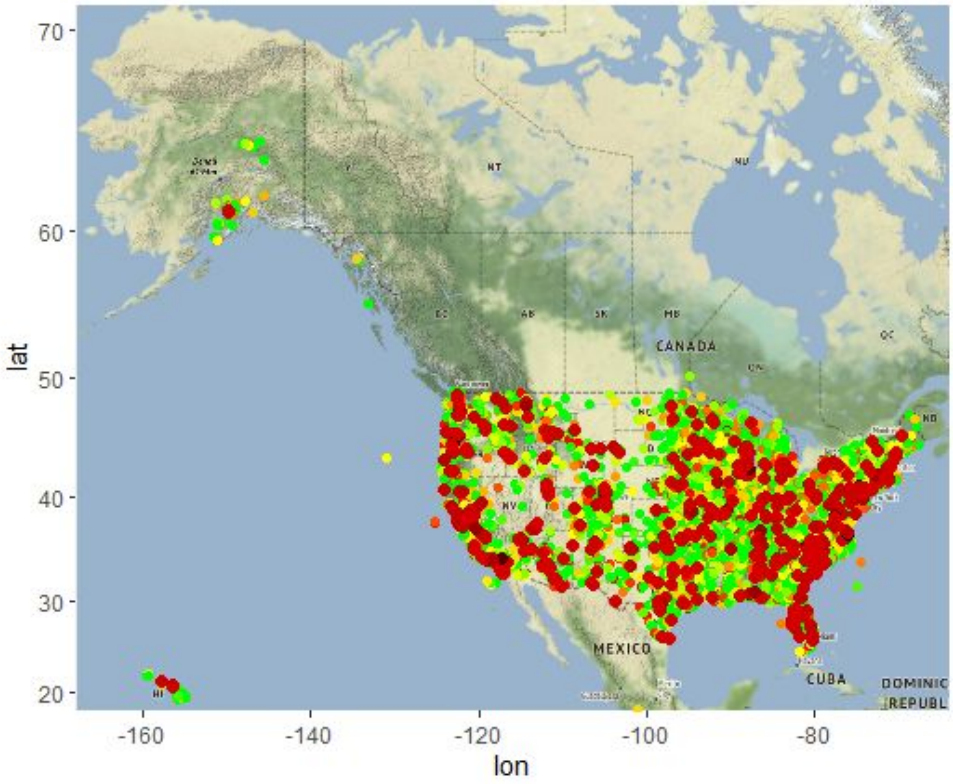
```
stat_density_2d(data=cars,
                aes(x=long, y=lat, fill=..level..),
                geom = "polygon",
                alpha = .3,
                color = "red") +
scale_fill_gradient2("Vehículos en venta",
                     low = "white",
                     mid = "yellow",
                     high = "red")
```

Los puntos se distribuyen sobre todo en la mitad este del país y a lo largo de la costa oeste, disminuyendo su densidad sobre todo en el centro, zonas más rurales y desierto de Nevada.

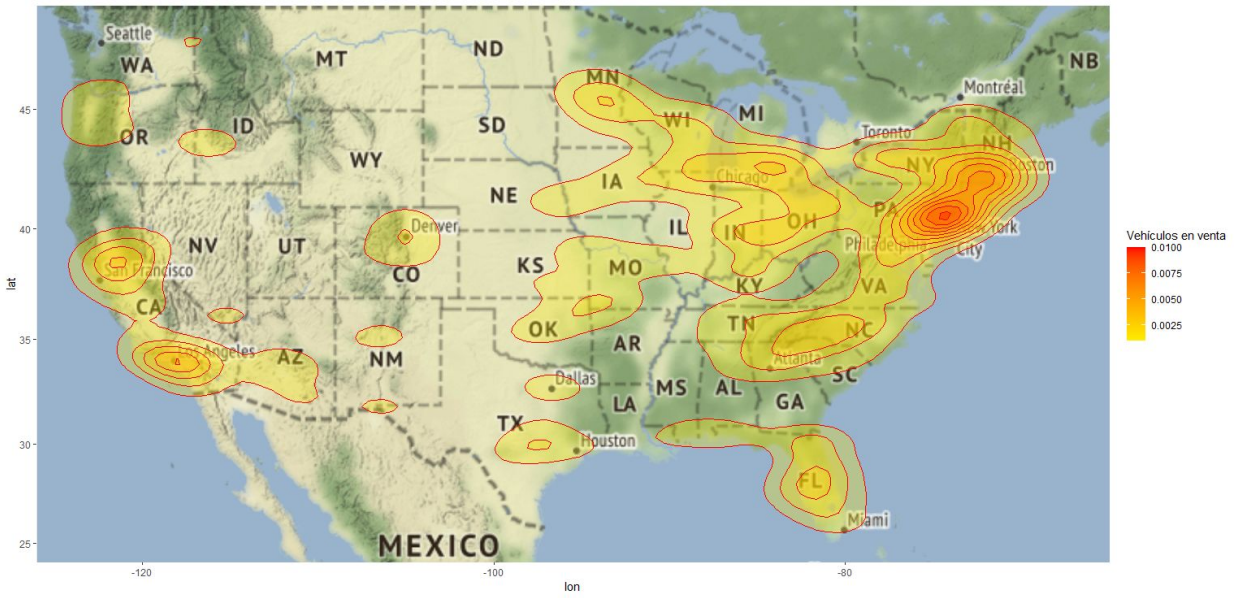
Donde se acumulan los puntos más oscuros y de mayor tamaño (coches más lujosos), coincide por lo general con la ubicación de grandes ciudades.



También se ha probado a incluir Alaska y Hawaii en el mapa, pero al hacer esto tenemos que reducir tanto el zoom que la visualización empeora considerable. Aquí podemos destacar como en la mayor parte de Alaska no hay ni un punto y todos se concentran en una misma zona.



En el mapa de densidad se confirma lo que hemos visto en el mapa anterior. La venta de coches se acumula en la mitad este (con el mayor foco en Nueva York) y en la costa oeste (principalmente en California).



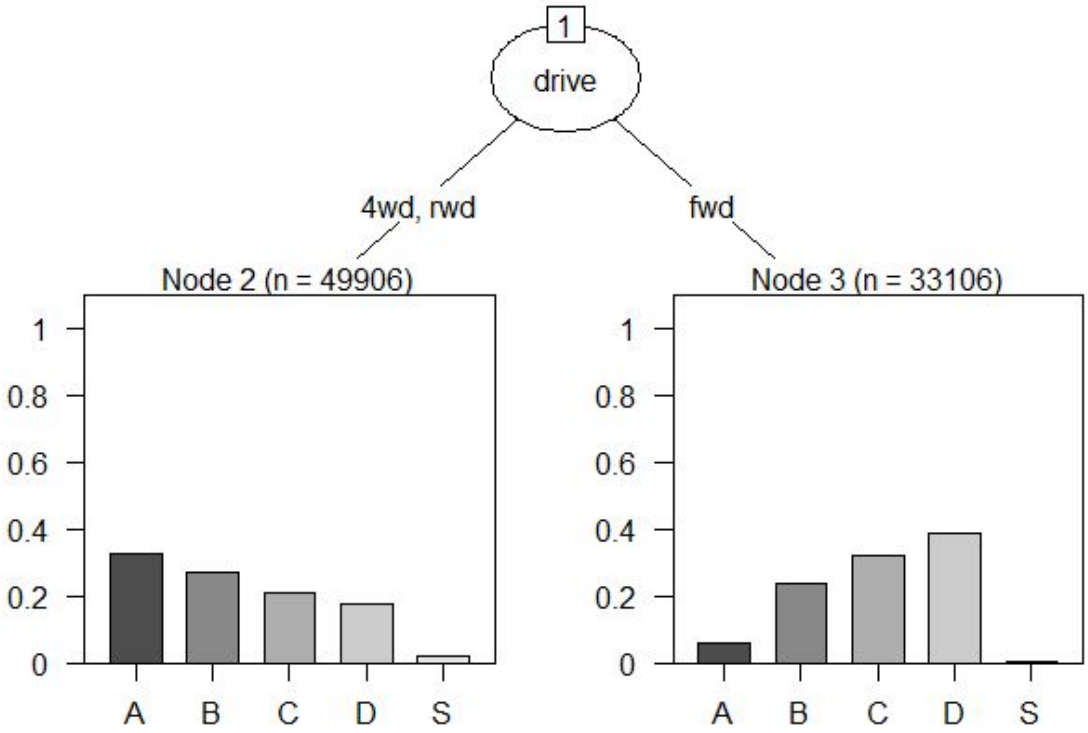
HISTOGRAMAS

A continuación vamos a estudiar la distribución de algunos atributos. Para ello usaremos pequeños árboles de decisión (utilizando únicamente una variable) y gráficos de barras.

Antes de nada, como el precio de venta es una variable numérica, se han agrupado los vehículos en diferentes categorías A, B, C y D (de más caro a más barato, y una categoría S, por encima de todas las demás.

```
cars$category = "S"
cars[cars$price < 50000,]$category = "A"
cars[cars$price < 17000,]$category = "B"
cars[cars$price < 9000,]$category = "C"
cars[cars$price < 5000,]$category = "D"
```

Vamos a comenzar con el atributo “drive”, que representa el tipo de tracción del vehículo, con tres posibles valores: delantera (fwd), trasera (rwd) o en las cuatro ruedas (4wd).

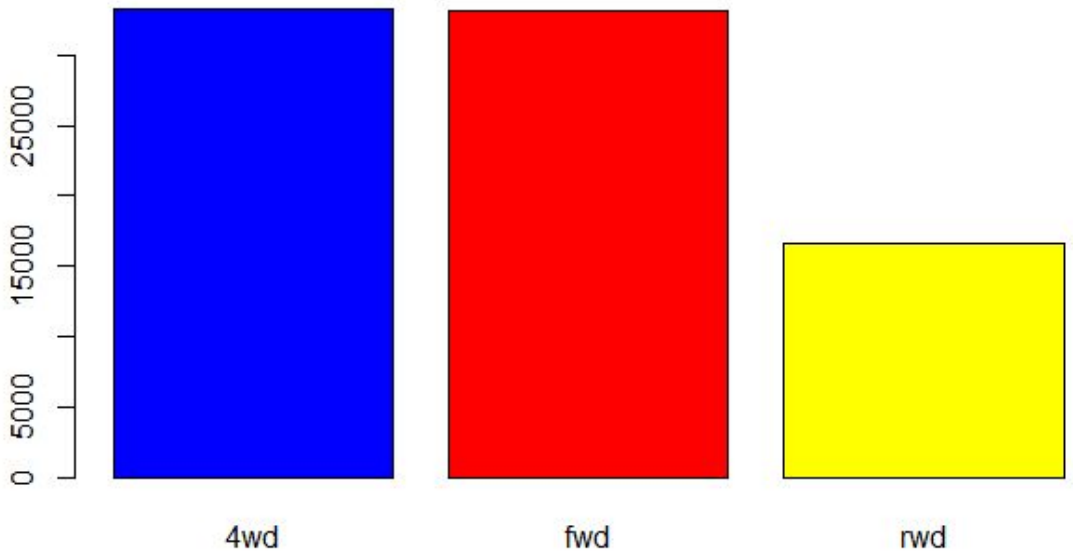


Se observa como los coches más baratos son los que tienen tracción delantera, mientras que los 4x4 y los rwd pertenecen a categorías más altas.

Generamos gráficos de barras:

```
plot(cars$drive, col=c('blue','red','yellow'))
barplot(table(cars$drive, cars$category),
        col=c('blue','red','yellow'),
        beside = T,
        legend.text = T)
```

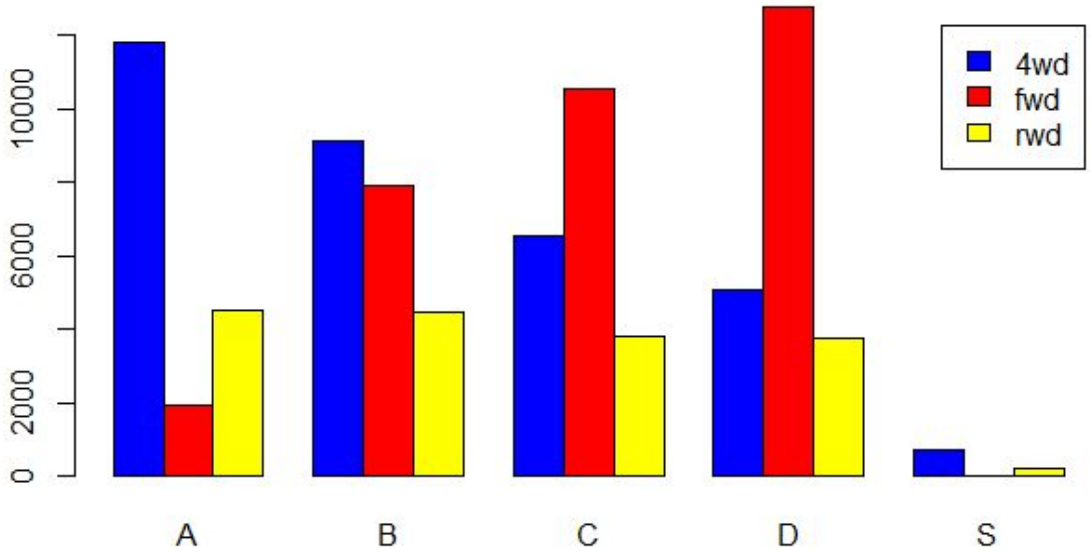
En el primer gráfico vemos los tipos de tracción sin agruparlos por categorías. En total, el número de coches con tracción trasera es muy inferior a los otros dos tipos, que se mantienen igualados.



En el segundo gráfico, se muestran agrupados según su categoría. Vemos cómo a pesar de que hay aproximadamente la misma cantidad de 4wd y de fwd, estos se acumulan en categorías distintas, abundando los 4x4 en categorías más caras y los fwd en las más baratas.

Hay que destacar también como en la mayoría de las categorías la proporción de coches con tracción trasera es inferior al resto, excepto en las categorías más caras, donde llega a superar a la tracción delantera. Esto es debido a que los deportivos caros suelen tener tracción trasera. En la categoría S directamente se podría decir que no hay coches con tracción delantera.

Como conclusión de estos gráficos podemos decir que entre la población de Estados Unidos los vehículos 4x4 son muy populares y que sólo una pequeña parte de la población puede tener coches con tracción trasera ya que suelen ser deportivos muy caros.

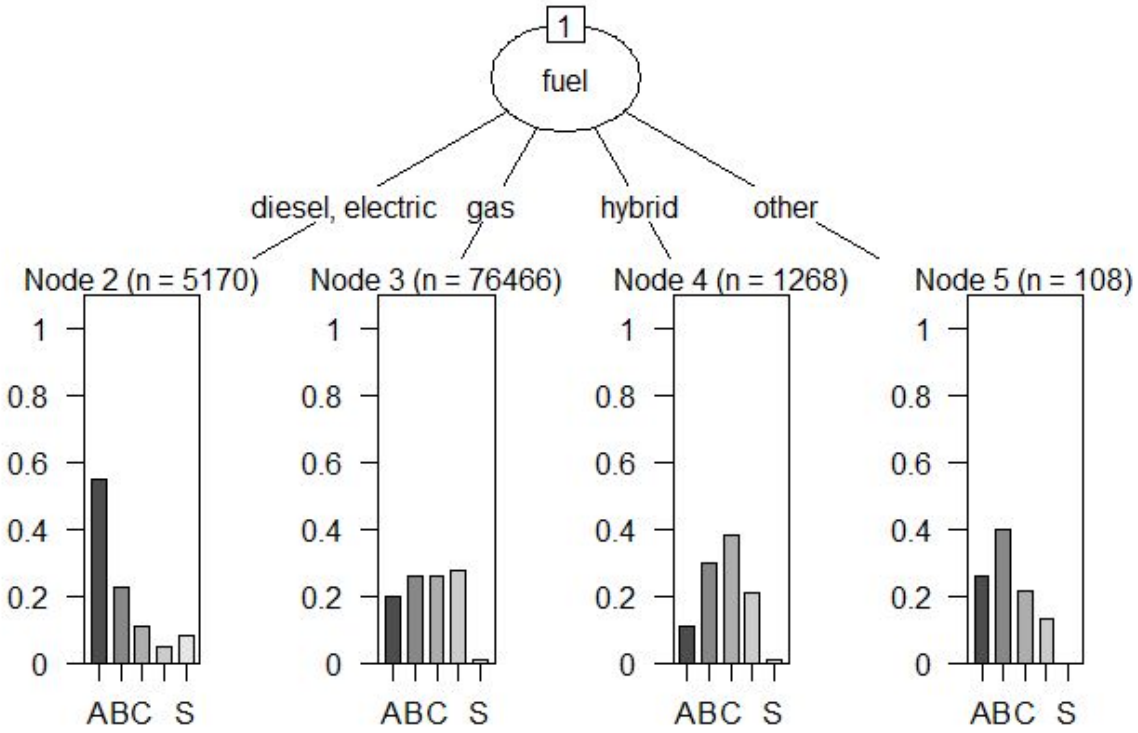


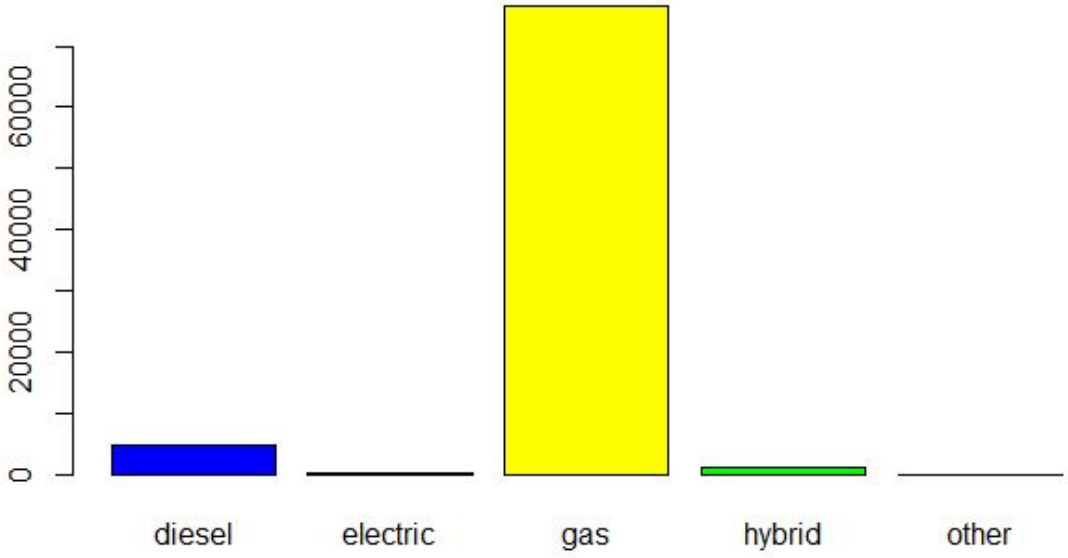
A continuación vamos a estudiar los vehículos según el tipo de combustible.

```
model_C50 = C5.0(category ~ fuel, data=cars)
plot(model_C50)

plot(cars$fuel, col=c('blue','red','yellow','green','grey'
```

Según el árbol de decisión, podríamos decir que los vehículos diésel y los eléctricos son por lo general más caros que los vehículos de gasolina y los híbridos, ya que estos primeros tienen mayor presencia en las clases S y A mientras que los segundos están presentes en las gamas media y baja. En la clase S apenas existe presencia de vehículos de gasolina o híbridos.

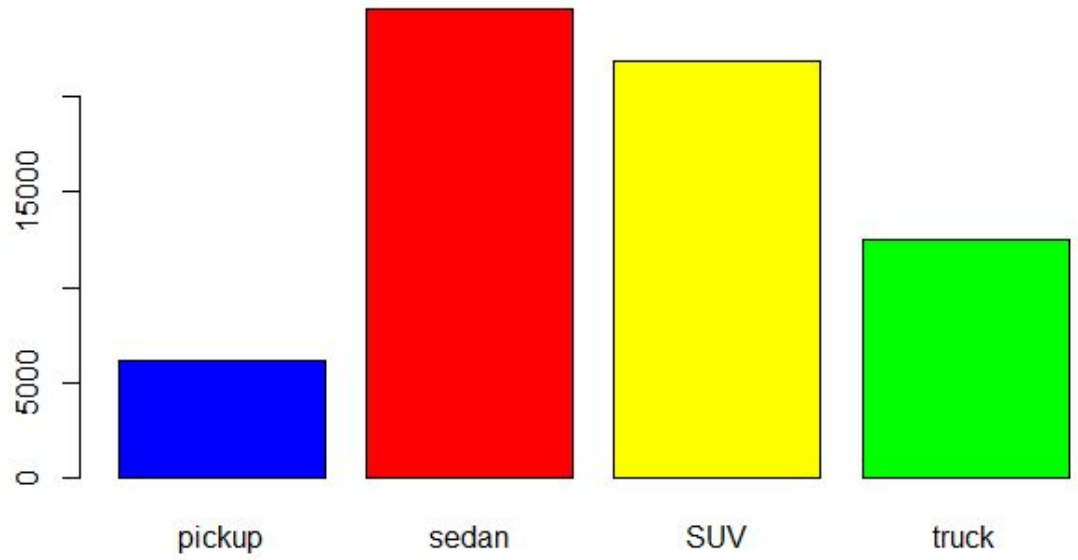
Además, en la gráfica vemos como el combustible más común es con diferencia la gasolina.


1
fuel
diesel, electric
gas
hybrid
other
Node 2 (n = 5170)
Node 3 (n = 76466)
Node 4 (n = 1268)
Node 5 (n = 108)
1
0.8
0.6
0.4
0.2
0
ABC
S


0
20000
40000
60000
diesel
electric
gas
hybrid
other

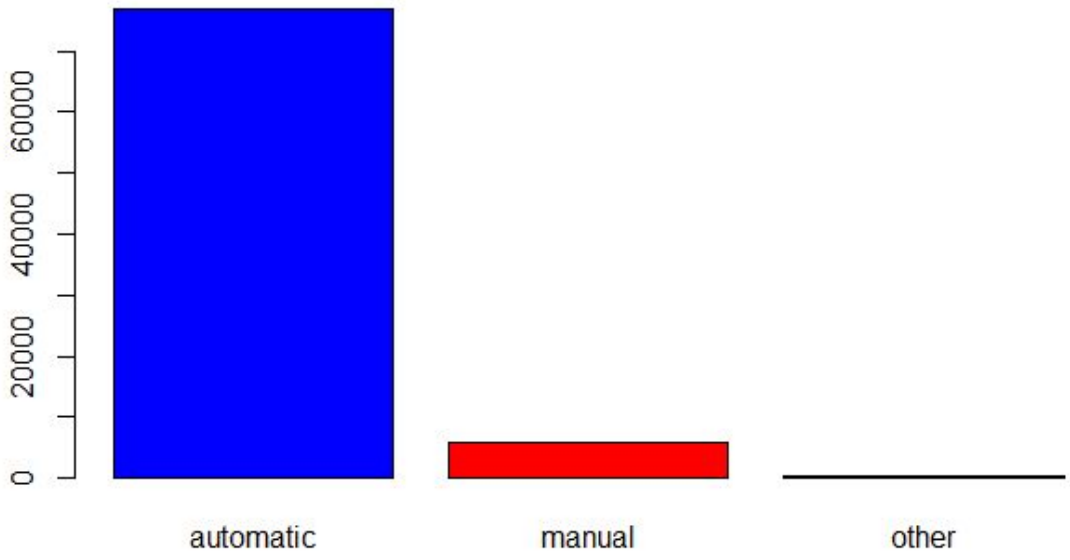
Por otra parte, si estudiamos cuáles son los tipos de vehículos que más abundan, obtenemos que (ignorando los sedanes, que son el tipo de vehículo más común), en Estados Unidos son muy populares los vehículos tipo SUV, camionetas y camiones.

```
type_freq = table(cars$type)
type_freq
barplot(type_freq[type_freq > 5000], col=c('blue','red','yellow','green'))
```



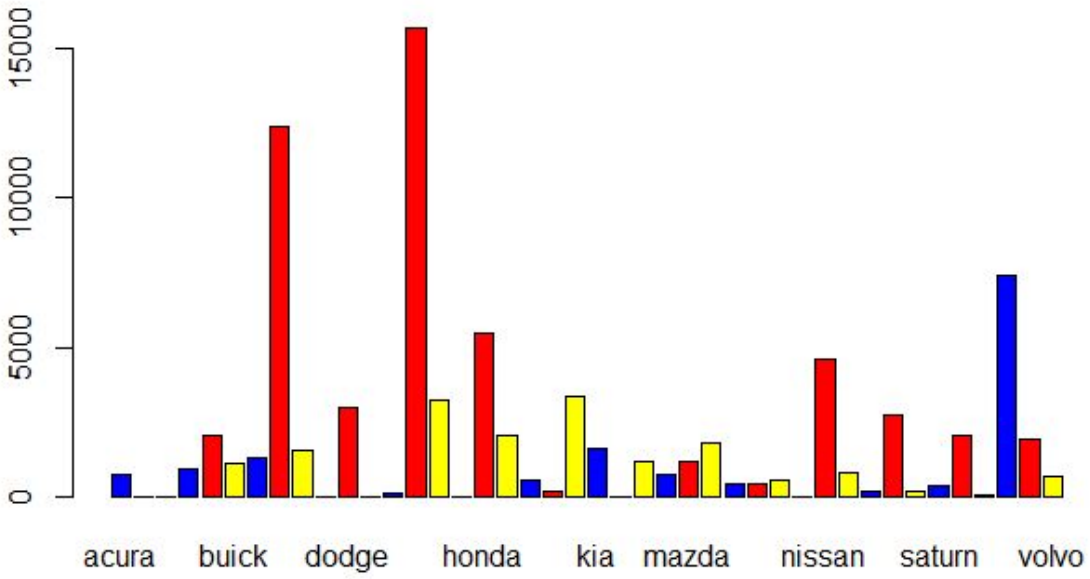
Podemos decir que en Estados Unidos prefieren los vehículos automáticos, ya que apenas hay vehículos con transmisión manual.

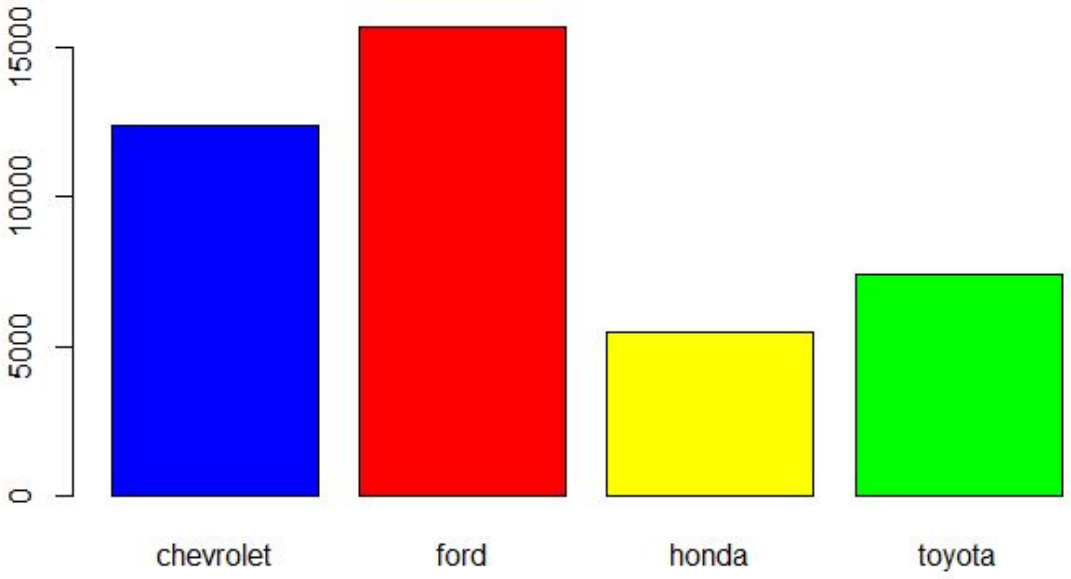
```
plot(cars$transmission, col=c('blue','red','yellow'))
```



También podemos ver las marcas más populares en Estados Unidos, que son con mucha diferencia Ford y Chevrolet.

```
plot(cars$manufacturer, col=c('blue','red','yellow'))

manuf_freq = table(cars$manufacturer)
manuf_freq
barplot(manuf_freq[manuf_freq > 5000],
col=c('blue','red','yellow','green'))
```


0
5000
10000
15000
acura
buick
dodge
honda
kia
mazda
nissan
saturn
volvo


0
5000
10000
15000
chevrolet
ford
honda
toyota

REGRESIÓN LINEAL

Antes de nada, dividiremos el dataset en dos conjuntos de train (75%) y test (25%), esta misma división será utilizada para el resto de modelos que se harán más adelante.

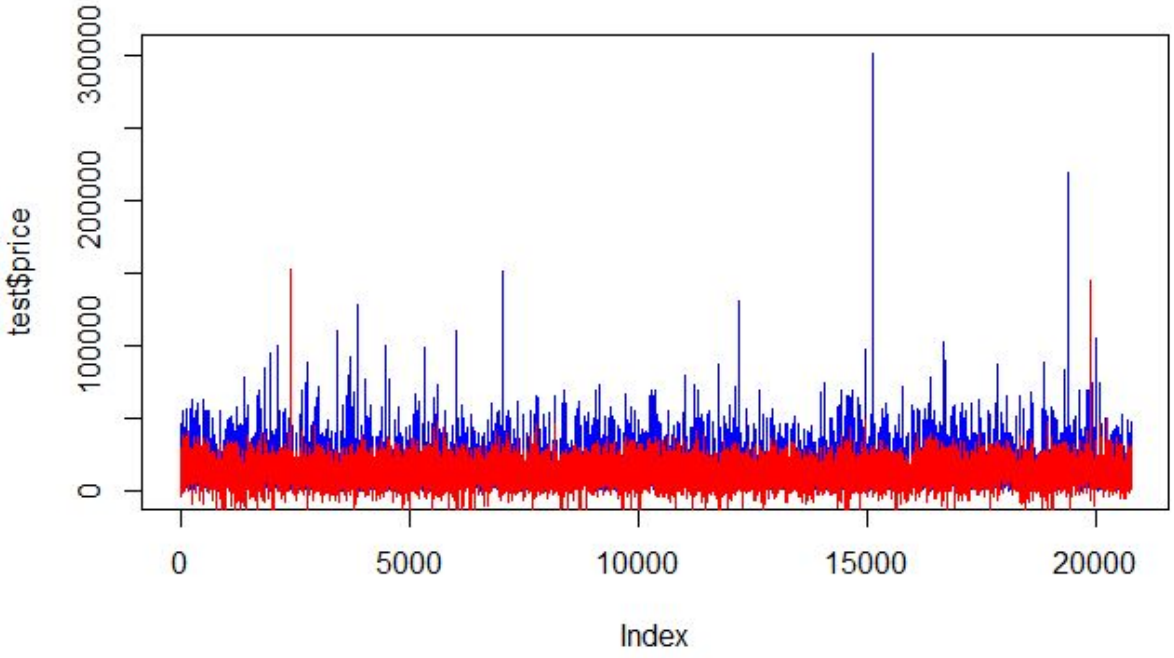
```
# Tamaño de la muestra de entrenamiento (75% del total)
smp_size = floor(0.75 * nrow(cars))

# Obtenemos smp_size filas aleatorias sin repetición
train_ind = sample(seq_len(nrow(cars)), size = smp_size)

# Asignamos las filas obtenidas al conjunto de entrenamiento y el resto al
# de test
train = cars[train_ind,]
test = cars[-train_ind,]
```

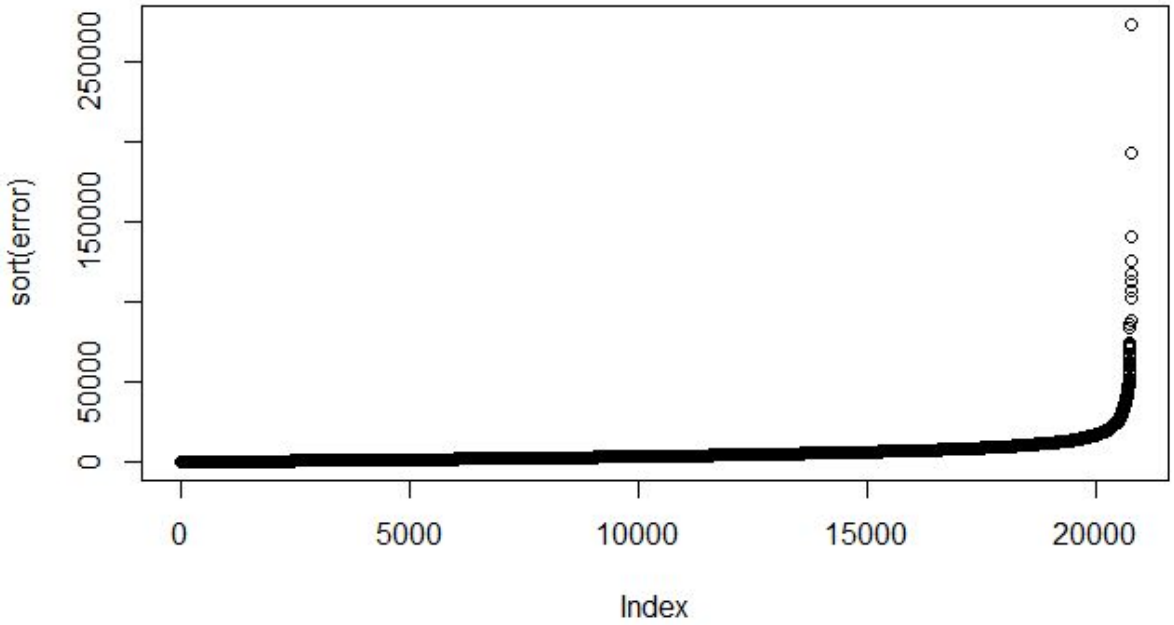
Realizaremos la regresión lineal para clasificar el atributo precio (numérico) en función del resto, exceptuando el atributo X que únicamente es un identificador que no aporta información y quitando el atributo “model” (modelo del vehículo) y “region”, ya que aunque estos atributos sí que podrían aportar información, debido a la gran cantidad valores categóricos distintos que pueden tomar, son muy complicados de procesar.

```
model_LM = lm(price ~
                . - X - model - region,
              data = train)
```

Obtenemos la predicción del modelo sobre el conjunto de test y la representamos en una gráfica: en azul los datos originales y en rojo la predicción. Además obtenemos el error cometido y lo representamos en otra gráfica.

```
p = predict(model_LM, newdata=test)

plot(test$price, type="l", col="blue")
lines(p, col="red")

error = abs(test$price - p)
summary(error)
plot(sort(error))
```


test$price
0
100000
200000
300000
0
5000
10000
15000
20000
Index


sort(error)
0
50000
150000
250000
0
5000
10000
15000
20000
Index

```
> summary(error)
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
     0.02   1638.86   3660.79   5299.44   6844.02 273943.59
```

En la primera gráfica vemos como la predicción no se ajusta demasiado bien al precio real. Si además observamos el error, vemos que de media se comete un error de unos 5299.44$. Es una suma de dinero bastante significativa, sobre todo para los vehículos de menor precio.

ÁRBOLES DE DECISIÓN

Los árboles de decisión sólo pueden predecir valores categóricos, por tanto, al igual que hicimos en el apartado de visualización, dividimos los coches en categorías según su precio. Además utilizamos la separación en train y test del apartado anterior.

Para seleccionar los atributos más adecuados, partimos de un modelo con todos los atributos y mediante la función summary() vemos cuáles son los atributos menos relevantes para el modelo y los eliminamos hasta conseguir maximizar la precisión del modelo.

```
model_C50 = C5.0(category ~
                   year +
                   manufacturer +
                   condition +
                   cylinders +
                   fuel +
                   odometer +
                   title_status +
                   transmission +
                   drive +
                   size +
                   type +
                   lat +
                   long,
                 data = train)
```

Al utilizar tantas variables, el árbol generado no es interpretable, a diferencia de los árboles utilizados en el apartado de visualización. Por tanto en este apartado nos centraremos más en que el modelo sea bueno prediciendo.

Generamos la tabla de confusión comparando los valores reales con la predicción, y con la tabla obtenemos la precisión del modelo, sumando los valores de la diagonal (patrones correctamente clasificados) y dividiendo entre el número total de patrones de test.

```
# Predicción y matriz de confusión
p = predict(model_C50, newdata = test)
t = table(p, test$category)
# Precisión del modelo
(t[1,1] + t[2,2] + t[3,3] + t[4,4]) / nrow(test)
```

```
> t

p      A    B    C    D
  A 4088  481   83   63
  B  566 3936  796  180
  C   48  762 3411  919
  D   27  133  970 4290
> # Precision del modelo
> (t[1,1] + t[2,2] + t[3,3] + t[4,4]) / nrow(test)
[1] 0.7577218
```

Con un 75% de acierto, sigue sin ser perfecto, pero funciona bastante mejor que la regresión lineal. Como cada categoría comprende un rango de precios más o menos grande, el error cometido no es tan notable como en el caso anterior.

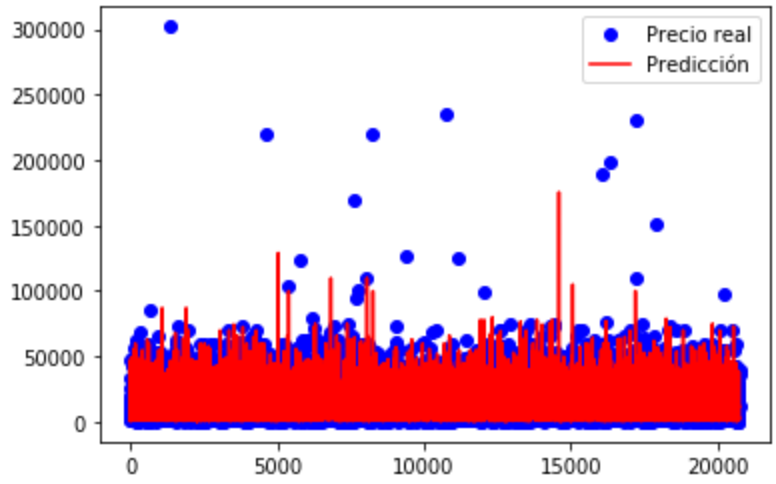
Por otra parte, se ha utilizado la librería Sklearn de Python ya que nos permite crear modelos de árboles de regresión para variables numéricas.

```
dtr = tree.DecisionTreeRegressor()
dtr_model = dtr.fit(train_x, train_y['price'])
train_score = dtr.score(train_x, train_y['price'])
test_score = dtr.score(test_x, test_y['price'])
print('Train Score: ', train_score)
print('Test Score: ', test_score)
```

```
Train Score:  0.9998801927831847
Test Score:  0.6803299111753707
```

La puntuación para el conjunto de entrenamiento es casi perfecta, sin embargo para test baja bastante (sobreajuste). Vamos a ver de forma gráfica la predicción realizada para el test.

```
dtr_predict = dtr.predict(test_x)
plt.plot(np.array(test_y['price']), 'bo', label="Precio real")
plt.plot(dtr_predict, 'r', label="Predicción")
plt.legend()
```



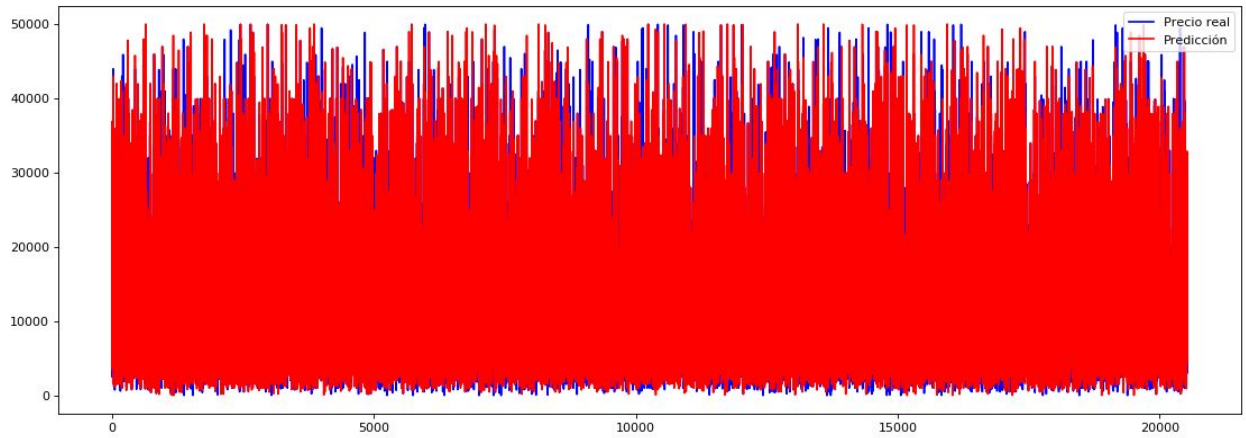
Para los valores más bajos (la mayoría de los puntos), la función de predicción se ajusta bien a los puntos. El problema está en los vehículos más caros, ya que estos puntos difieren mucho del resto de la muestra y actúan como ruido, haciendo que el modelo generado no generalice correctamente al utilizar unos datos distintos a los de entrenamiento.

A continuación se prueba a generar el modelo utilizando únicamente aquellos vehículos que cuestan menos de 50000$.

```
cars2 = cars[cars['price'] < 50000]
```

```
Train Score:  0.9999035209969653
Test Score:  0.8104814097820459
```

La puntuación del test mejora bastante.



REDES NEURONALES

A continuación vemos los resultados obtenidos para los modelos de redes neuronales realizados.

Para que funcionen mejor las redes neuronales, se normalizan las variables numéricas, de manera que tomen valores entre 0 y 1.

```
cars_norm['price'] = (cars['price'] - min(cars['price'])) /
(max(cars['price']) - min(cars['price']))
cars_norm['year'] = (cars['year'] - min(cars['year'])) / (max(cars['year'])
- min(cars['year']))
cars_norm['odometer'] = (cars['odometer'] - min(cars['odometer'])) /
(max(cars['odometer']) - min(cars['odometer']))
cars_norm['lat'] = (cars['lat'] - min(cars['lat'])) / (max(cars['lat']) -
min(cars['lat']))
cars_norm['long'] = (cars['long'] - min(cars['long'])) / (max(cars['long'])
- min(cars['long']))
```

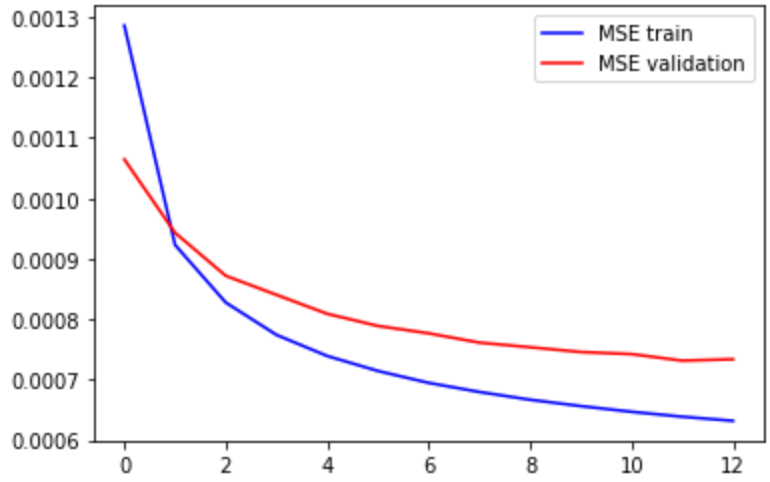
La primera capa tiene 10803 entradas, una por cada atributo. Realmente no hay tantos atributos, pero como las variables categóricas han sido transformadas en variables “dummies” cada una de ellas se traduce en tantas variables distintas como posibles valores tenga el atributo. El modelo tiene dos capas de 100 neuronas y una capa de salida con una única salida (precio).

Se ha utilizado “early stopping”, para que el modelo deje de entrenar cuando ya no se produzcan nuevas mejoras para el conjunto de validación.

```
es = EarlyStopping(monitor='val_loss',
                   mode='min',
                   verbose=1)

model = Sequential()

model.add(Dense(100,
                input_dim=10803,
                activation="tanh"))

model.add(Dense(100,
                activation="tanh"))

model.add(Dense(1,
                activation="tanh"))

model.compile(loss='mean_squared_error',
              optimizer='sgd')

resultado = model.fit(train_x,
                      train_y['price'],
                      validation_data =(test_x, test_y['price']),
                      epochs=1000,
                      callbacks=[es])
```

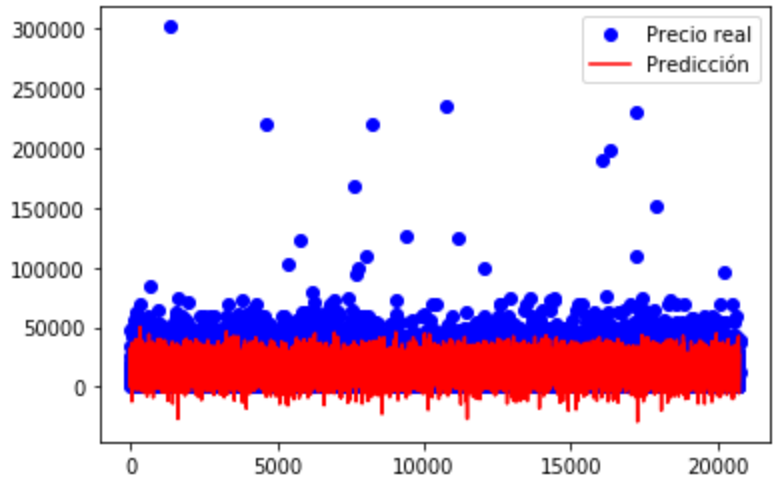
```
Epoch 1/1000
1946/1946 [==============================] - 14s 7ms/step - loss: 0.0013 - val_loss: 0.0011
Epoch 2/1000
1946/1946 [==============================] - 15s 8ms/step - loss: 9.2306e-04 - val_loss: 9.4310e-04
Epoch 3/1000
1946/1946 [==============================] - 16s 8ms/step - loss: 8.2774e-04 - val_loss: 8.7196e-04
Epoch 4/1000
1946/1946 [==============================] - 17s 9ms/step - loss: 7.7423e-04 - val_loss: 8.4047e-04
Epoch 5/1000
1946/1946 [==============================] - 18s 9ms/step - loss: 7.3939e-04 - val_loss: 8.0907e-04
Epoch 6/1000
1946/1946 [==============================] - 22s 11ms/step - loss: 7.1457e-04 - val_loss: 7.8901e-04
Epoch 7/1000
1946/1946 [==============================] - 22s 11ms/step - loss: 6.9490e-04 - val_loss: 7.7683e-04
Epoch 8/1000
1946/1946 [==============================] - 19s 10ms/step - loss: 6.7986e-04 - val_loss: 7.6144e-04
Epoch 9/1000
1946/1946 [==============================] - 20s 10ms/step - loss: 6.6701e-04 - val_loss: 7.5382e-04
Epoch 10/1000
1946/1946 [==============================] - 15s 8ms/step - loss: 6.5649e-04 - val_loss: 7.4590e-04
Epoch 11/1000
1946/1946 [==============================] - 14s 7ms/step - loss: 6.4708e-04 - val_loss: 7.4247e-04
Epoch 12/1000
1946/1946 [==============================] - 15s 7ms/step - loss: 6.3890e-04 - val_loss: 7.3164e-04
Epoch 13/1000
1946/1946 [==============================] - 15s 8ms/step - loss: 6.3213e-04 - val_loss: 7.3397e-04
Epoch 00013: early stopping
```

De las 1000 épocas que se habían fijado como máximo, realmente solo se han ejecutado 13. Visualizando el valor de la función de pérdidas en cada época (error cuadrático medio), vemos cómo efectivamente el MSE de test deja de bajar a partir de las 11 épocas, por lo que no tiene sentido seguir ejecutando nada.



A la hora de realizar una predicción, hay que recordar que los datos que estamos utilizando están normalizados, por lo que si queremos obtener unos valores de precios que podamos interpretar tenemos que volver a la escala inicial. Una vez hecho, representamos la predicción y los valores esperados.

```
# Para deshacer la normalización
precio_real = test_y['price'] * (max(cars['price']) - min(cars['price'])) +
min(cars['price'])
precio_prediccion = prediccion * (max(cars['price']) - min(cars['price']))
+ min(cars['price'])

plt.plot(np.array(precio_real), 'ob', label='Precio real')
plt.plot(precio_prediccion, 'r', label='Predicción')
plt.legend()
```



Al igual que pasaba con los árboles, los valores más altos afectan a la capacidad de generalización del modelo.

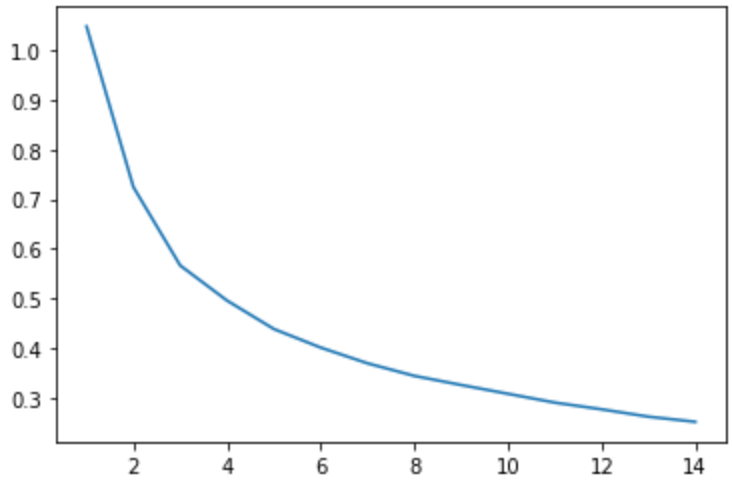
K MEANS

Vamos a agrupar los coches en clusters con el algoritmo k-means.

Lo primero es normalizar las variables numéricas igual que hicimos en el apartado anterior. Una vez normalizado, realizamos un análisis de componentes principales PCA, con el objetivo de reducir la dimensionalidad a 2 variables. Esto es debido a que nuestro dataset tiene demasiadas variables y a la hora de representar gráficamente los clusters utilizamos gráficos bidimensionales.

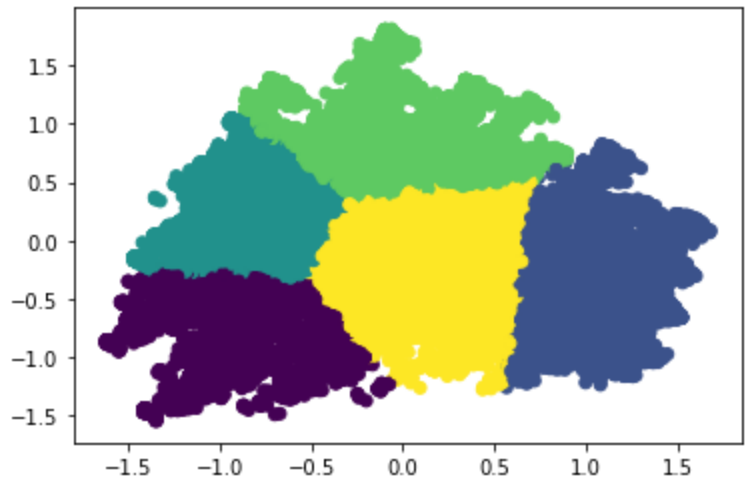
```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(cars)
```

A continuación, se utiliza el método del codo para seleccionar el número de clusters.



Ejecutamos k-means con 5 clusters y lo representamos.

```
km = KMeans(n_clusters = 5,
            random_state = 20)
km.fit(principalComponents)

# En cada eje una componente y el color en función de las etiquetas
obtenidas mediante kmeans
plt.scatter(principalComponents[:,0], principalComponents[:,1], c =
km.labels_)
```



El problema que se presenta aquí, es que a pesar de que el algoritmo es capaz de obtener los clusters, el resultado que obtenemos no tiene ningún tipo de interpretación, las variables representadas en los ejes son las transformaciones realizadas mediante el PCA y no podemos extraer ninguna conclusión a partir de ellas.

CONCLUSIÓN

Como conclusión podemos decir que mediante representaciones gráficas no muy complejas podemos obtener bastante información sobre el mercado de vehículos en Estados Unidos. Entre otras cosas, podemos ver en qué zonas se venden más coches o qué tipos de vehículos son más populares en Estados Unidos por su marca, tipo de tracción, combustible, transmisión, etc. Y podemos saber qué valor tiene cada uno de ellos.

Mediante los modelos generados, hemos visto que para según qué vehículos es difícil predecir el precio ya que se dispone de pocos datos. Esto ocurre con vehículos con precios demasiado elevados y que se salen de la normalidad. Sin embargo, si obviamos estos vehículos y nos centramos en un rango de precios más normal, podemos estimar el precio de venta en función de las características del vehículo.