SPRINT 15

Victor Chang

El servicio de venta de autos usados Rusty Bargain está desarrollando una aplicación para atraer nuevos clientes. Gracias a esa app, puedes averiguar rápidamente el valor de mercado de tu coche. Tienes acceso al historial: especificaciones técnicas, versiones de equipamiento y precios. Tienes que crear un modelo que determine el valor de mercado. A Rusty Bargain le interesa:

- la calidad de la predicción;
- la velocidad de la predicción;
- el tiempo requerido para el entrenamiento

Importación de librerías

In [24]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import mean_squared_error
from time import perf_counter
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.base import clone
```

Importación de datos

In [2]:
```
df = pd.read_csv("C:/Users/Acer/Documents/Tripleten/car_data.csv")
df.head(10)
```

Out[2]:

	DateCrawled	Price	VehicleType	RegistrationYear	Gearbox	Power	Model	Mileag
0	24/03/2016 11:52	480	NaN	1993	manual	0	golf	15000
1	24/03/2016 10:58	18300	coupe	2011	manual	190	NaN	12500
2	14/03/2016 12:52	9800	suv	2004	auto	163	grand	12500
3	17/03/2016 16:54	1500	small	2001	manual	75	golf	15000
4	31/03/2016 17:25	3600	small	2008	manual	69	fabia	9000
5	04/04/2016 17:36	650	sedan	1995	manual	102	3er	15000
6	01/04/2016 20:48	2200	convertible	2004	manual	109	2_reihe	15000
7	21/03/2016 18:54	0	sedan	1980	manual	50	other	4000
8	04/04/2016 23:42	14500	bus	2014	manual	125	c_max	3000
9	17/03/2016 10:53	999	small	1998	manual	101	golf	15000

Preparación de datos

In [3]:
```
# 0) Evitemos trabajar con el df original
df_work = df.copy()
df_work.shape
```

Out[3]:
```
(354369, 16)
```

In [4]:
```
# 1) Convertir fechas a datetime

date_cols = ["DateCrawled", "DateCreated", "LastSeen"]
for c in date_cols:
    df_work[c] = pd.to_datetime(df_work[c], errors="coerce")

df_work[date_cols].dtypes
```

```
C:\Users\Acer\AppData\Local\Temp\ipykernel_8004\3139899552.py:5: UserWarning: Par
sing dates in %d/%m/%Y %H:%M format when dayfirst=False (the default) was specifi
ed. Pass `dayfirst=True` or specify a format to silence this warning.
  df_work[c] = pd.to_datetime(df_work[c], errors="coerce")
C:\Users\Acer\AppData\Local\Temp\ipykernel_8004\3139899552.py:5: UserWarning: Par
sing dates in %d/%m/%Y %H:%M format when dayfirst=False (the default) was specifi
ed. Pass `dayfirst=True` or specify a format to silence this warning.
  df_work[c] = pd.to_datetime(df_work[c], errors="coerce")
```

Out[4]:
```
DateCrawled    datetime64[ns]
DateCreated    datetime64[ns]
LastSeen       datetime64[ns]
dtype: object
```

In [5]:
```
# 2) Limpiar el target (Price): quitar nulos y ceros/negativos
before = df_work.shape[0]
df_work = df_work[df_work["Price"].notna()]
df_work = df_work[df_work["Price"] > 0]
after = df_work.shape[0]

before, after, before - after
```

Out[5]:
```
(354369, 343597, 10772)
```

In [6]:
```
# 3) Filtros conservadores de outliers (años, potencia, kilometraje)
before = df_work.shape[0]

df_work = df_work[df_work["RegistrationYear"].between(1950, 2019, inclusive="bot
df_work = df_work[df_work["Power"].between(1, 600, inclusive="both")]
df_work = df_work[df_work["Mileage"].between(0, 300000, inclusive="both")]

after = df_work.shape[0]
(before, after, before - after), df_work[["RegistrationYear","Power","Mileage"]]
```

Out[6]:
```
((343597, 306802, 36795),
                       count           mean           std     min       25%  \
 RegistrationYear  306802.0    2003.310513      6.897975  1950.0    1999.0
 Power             306802.0     120.215162     53.798342     1.0      75.0
 Mileage           306802.0  128434.071486  36721.361426  5000.0  125000.0

                        50%       75%       max
 RegistrationYear    2003.0    2008.0    2019.0
 Power                110.0     150.0     600.0
 Mileage           150000.0  150000.0  150000.0  )
```

In [7]:
```
# 4) Revisar columnas "débiles" para dropear (PostalCode / NumberOfPictures)
cols_check = [c for c in ["NumberOfPictures", "PostalCode"] if c in df_work.colu
df_work[cols_check].nunique(dropna=False), df_work[cols_check].head()
```

Out[7]:
```
(NumberOfPictures       1
 PostalCode          8120
 dtype: int64,
    NumberOfPictures  PostalCode
 1                 0       66954
 2                 0       90480
 3                 0       91074
 4                 0       60437
 5                 0       33775)
```

In [8]:
```
# 5) Dropear columnas con poca señal (si existen)
drop_cols = [c for c in ["NumberOfPictures", "PostalCode"] if c in df_work.colum
df_work = df_work.drop(columns=drop_cols, errors="ignore")

df_work.shape, drop_cols
```

Out[8]:
```
((306802, 14), ['NumberOfPictures', 'PostalCode'])
```

In [9]:
```
# 6) Ingeniería mínima de fechas: year/month/dayofweek y luego eliminar datetime
for c in date_cols:
    df_work[f"{c}_year"] = df_work[c].dt.year
    df_work[f"{c}_month"] = df_work[c].dt.month
    df_work[f"{c}_dow"] = df_work[c].dt.dayofweek

df_work[[f"{c}_year" for c in date_cols] + [f"{c}_month" for c in date_cols] + [
```

Out[9]:

	DateCrawled_year	DateCreated_year	LastSeen_year	DateCrawled_month	DateCreate
1	2016	2016	2016.0	3	
2	2016	2016	2016.0	3	
3	2016	2016	NaN	3	
4	2016	2016	2016.0	3	
5	2016	2016	2016.0	4	

In [10]:
```
# 7) Eliminar columnas datetime originales
df_work = df_work.drop(columns=date_cols, errors="ignore")
df_work.filter(regex="Date|Seen|Crawled|Created").head()
```

Out[10]:

	DateCrawled_year	DateCrawled_month	DateCrawled_dow	DateCreated_year	DateCre
1	2016	3	3	2016	
2	2016	3	0	2016	
3	2016	3	3	2016	
4	2016	3	3	2016	
5	2016	4	0	2016	

In [11]:
```
# 8) Normalizar categóricas: strip/lower y pasar "unknown"/vacíos a NaN
obj_cols = df_work.select_dtypes(include=["object", "string"]).columns

for c in obj_cols:
    df_work[c] = df_work[c].astype("string").str.strip().str.lower()
    df_work[c] = df_work[c].replace({"": pd.NA, "nan": pd.NA, "none": pd.NA, "un

df_work[obj_cols].isna().mean().sort_values(ascending=False).head(10)
```

Out[11]:
```
NotRepaired    0.153258
VehicleType    0.069455
FuelType       0.064318
Model          0.040502
Gearbox        0.019775
Brand          0.000000
dtype: float64
```

In [12]:
```
# 9) Separar features y target (listo para pipelines/modelos)
target = "Price"
X = df_work.drop(columns=[target])
y = df_work[target].astype(float)
```

```
X.shape, y.shape, y.describe()
```

Out[12]:
```
((306802, 19),
 (306802,),
 count    306802.000000
 mean       4810.867905
 std        4586.537267
 min           1.000000
 25%        1300.000000
 50%        3100.000000
 75%        6950.000000
 max       20000.000000
 Name: Price, dtype: float64)
```

Conclusión del apartado de preparación de datos

La etapa de preparación de datos permitió transformar el conjunto original en una base **consistente, limpia y apta para modelado**, sin introducir transformaciones agresivas que distorsionen la señal económica del precio. Se eliminaron observaciones con valores inválidos en la variable objetivo y outliers claramente implausibles en año de matriculación, potencia y kilometraje, reduciendo el ruido y mejorando la estabilidad de los modelos.

La ingeniería mínima de variables temporales conservó información relevante de las fechas sin generar alta dimensionalidad, mientras que la eliminación de columnas con escasa capacidad explicativa evitó complejidad innecesaria. Asimismo, la estandarización de variables categóricas dejó el dataset preparado para una imputación y codificación coherentes dentro de pipelines de machine learning.

Como resultado, se obtuvo un conjunto final de **306 802 observaciones y 19 características**, suficientemente grande y bien estructurado para comparar de forma justa modelos lineales, basados en árboles y métodos de potenciación del gradiente en las siguientes etapas del proyecto.

Entrenamiento del modelo

In [13]:
```
# 0) Split train/valid/test (si aún no lo hiciste)

X_train, X_tmp, y_train, y_tmp = train_test_split(X, y, test_size=0.40, random_s
X_valid, X_test, y_valid, y_test = train_test_split(X_tmp, y_tmp, test_size=0.50

X_train.shape, X_valid.shape, X_test.shape
```

Out[13]:
```
((184081, 19), (61360, 19), (61361, 19))
```

In [14]:
```
def sklearn_ready(X):
    X2 = X.copy()
    # fuerza a object para evitar pandas StringDtype/NAType
    X2 = X2.astype(object)
    # convierte cualquier faltante (incluye pd.NA) a np.nan
    X2 = X2.where(pd.notna(X2), np.nan)
```

```
    return X2

X_train = sklearn_ready(X_train)
X_valid = sklearn_ready(X_valid)
X_test  = sklearn_ready(X_test)

# recomputar columnas por dtype (después del fix)
num_cols = X_train.select_dtypes(include=["number"]).columns.tolist()
cat_cols = X_train.columns.difference(num_cols).tolist()

# reconstruir preprocess (clave)
numeric_preprocess = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median", missing_values=np.nan))
])

categorical_preprocess = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent", missing_values=np.nan)),
    ("ohe", OneHotEncoder(handle_unknown="ignore"))
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_preprocess, num_cols),
        ("cat", categorical_preprocess, cat_cols),
    ],
    remainder="drop"
)

# comprobación rápida: si esto corre, ya no hay pd.NA molestando
X_train.isna().sum().sum(), len(num_cols), len(cat_cols)
```

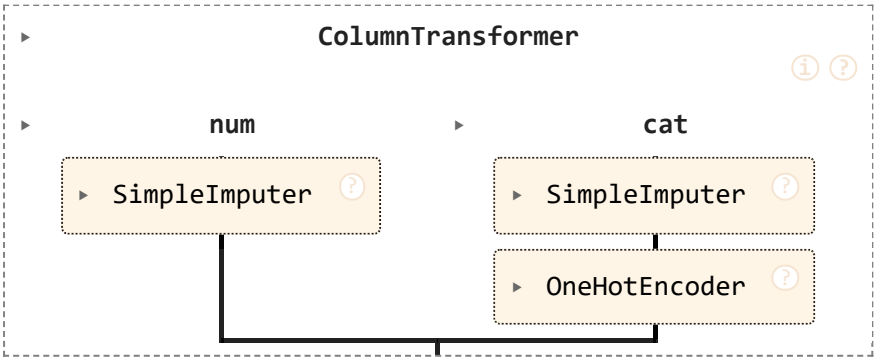
Out[14]: (np.int64(256527), 0, 19)

In [15]:
```
# 1) Identificar columnas numéricas y categóricas (comprobación)
num_cols = X_train.select_dtypes(include=["number"]).columns.tolist()
cat_cols = X_train.columns.difference(num_cols).tolist()

len(num_cols), len(cat_cols), num_cols, cat_cols
```

Out[15]:
```
(0,
 19,
 [],
 ['VehicleType',
  'RegistrationYear',
  'Gearbox',
  'Power',
  'Model',
  'Mileage',
  'RegistrationMonth',
  'FuelType',
  'Brand',
  'NotRepaired',
  'DateCrawled_year',
  'DateCrawled_month',
  'DateCrawled_dow',
  'DateCreated_year',
  'DateCreated_month',
  'DateCreated_dow',
  'LastSeen_year',
  'LastSeen_month',
  'LastSeen_dow'])
```

In [16]:
```
# 2) Preprocesamiento reutilizable (imputación + OHE) para modelos de sklearn


numeric_preprocess = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median"))
])

categorical_preprocess = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("ohe", OneHotEncoder(handle_unknown="ignore"))
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_preprocess, num_cols),
        ("cat", categorical_preprocess, cat_cols),
    ],
    remainder="drop"
)

preprocess
```

Out[16]:



In [17]:
```
# 3) Funciones de evaluación (RMSE + tiempos de fit y predict)
```

```
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

def eval_model(model, X_train, y_train, X_valid, y_valid, name="model"):
    pipe = Pipeline(steps=[("preprocess", preprocess), ("model", model)])

    t0 = perf_counter()
    pipe.fit(X_train, y_train)
    fit_time = perf_counter() - t0

    t1 = perf_counter()
    pred_valid = pipe.predict(X_valid)
    pred_time = perf_counter() - t1

    score = rmse(y_valid, pred_valid)
    return {"model": name, "rmse_valid": score, "fit_s": fit_time, "pred_s": pre

# comprobación rápida de que la función corre:
"ok"
```

Out[17]: 'ok'

In [18]:
```
# 4) Baseline: Regresión lineal (sanity check)

res_lr = eval_model(LinearRegression(), X_train, y_train, X_valid, y_valid, name
res_lr
```

Out[18]:
```
{'model': 'LinearRegression',
 'rmse_valid': np.float64(2007.592102635502),
 'fit_s': 15.31965799999989,
 'pred_s': 0.8460234000003766,
 'pipeline': Pipeline(steps=[('preprocess',
                  ColumnTransformer(transformers=[('num',
                                                   Pipeline(steps=[('imputer',
                                                                    SimpleImput
er(strategy='median'))]),
                                                   []),
                                                  ('cat',
                                                   Pipeline(steps=[('imputer',
                                                                    SimpleImput
er(strategy='most_frequent')),
                                                                   ('ohe',
                                                                    OneHotEncod
er(handle_unknown='ignore'))]),
                                                   ['VehicleType',
                                                    'RegistrationYear',
                                                    'Gearbox', 'Power', 'Mode
l',
                                                    'Mileage',
                                                    'RegistrationMonth',
                                                    'FuelType', 'Brand',
                                                    'NotRepaired',
                                                    'DateCrawled_year',
                                                    'DateCrawled_month',
                                                    'DateCrawled_dow',
                                                    'DateCreated_year',
                                                    'DateCreated_month',
                                                    'DateCreated_dow',
                                                    'LastSeen_year',
                                                    'LastSeen_month',
                                                    'LastSeen_dow'])])),
                 ('model', LinearRegression())])}
```

In [19]:
```
res_dt = eval_model(
    DecisionTreeRegressor(random_state=42, max_depth=15, min_samples_leaf=5),
    X_train, y_train, X_valid, y_valid,
    name="DecisionTree"
)
res_dt
```

Out[19]:
```
{'model': 'DecisionTree',
 'rmse_valid': np.float64(2432.2168041602363),
 'fit_s': 21.055801399999837,
 'pred_s': 0.8699842999999419,
 'pipeline': Pipeline(steps=[('preprocess',
                  ColumnTransformer(transformers=[('num',
                                                   Pipeline(steps=[('imputer',
                                                                    SimpleImput
er(strategy='median'))]),
                                                   []),
                                                  ('cat',
                                                   Pipeline(steps=[('imputer',
                                                                    SimpleImput
er(strategy='most_frequent')),
                                                                   ('ohe',
                                                                    OneHotEncod
er(handle_unknown='ignore'))]),
                                                   ['VehicleType',
                                                    'RegistrationYear',
                                                    'Gearbox', 'Power', 'Mode
l',
                                                    'Mileage',
                                                    'RegistrationMonth',
                                                    'FuelType', 'Brand',
                                                    'NotRepaired',
                                                    'DateCrawled_year',
                                                    'DateCrawled_month',
                                                    'DateCrawled_dow',
                                                    'DateCreated_year',
                                                    'DateCreated_month',
                                                    'DateCreated_dow',
                                                    'LastSeen_year',
                                                    'LastSeen_month',
                                                    'LastSeen_dow'])])),
                 ('model',
                  DecisionTreeRegressor(max_depth=15, min_samples_leaf=5,
                                        random_state=42))])}
```

In [20]:
```
sample_size = 60000  # 40k-80k suele ir bien
rng = np.random.RandomState(42)
idx = rng.choice(len(X_train), size=min(sample_size, len(X_train)), replace=Fals

X_train_rf = X_train.iloc[idx]
y_train_rf = y_train.iloc[idx]

rf = RandomForestRegressor(
    random_state=42,
    n_estimators=150,      # baja de 200
    max_depth=20,          # limita profundidad
    min_samples_leaf=5,    # hojas más grandes => menos splits
    max_features="sqrt",   # menos features por split => más rápido
    n_jobs=-1
)

res_rf = eval_model(rf, X_train_rf, y_train_rf, X_valid, y_valid, name="RandomFo
res_rf
```

Out[20]:
```
{'model': 'RandomForest(sampled)',
 'rmse_valid': np.float64(2809.967002206178),
 'fit_s': 11.491241199999877,
 'pred_s': 1.5249574999998003,
 'pipeline': Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
                                                                   SimpleImput
er(strategy='median'))]),
                                                  []),
                                                 ('cat',
                                                  Pipeline(steps=[('imputer',
                                                                   SimpleImput
er(strategy='most_frequent')),
                                                                  ('ohe',
                                                                   OneHotEncod
er(handle_unknown='ignore'))]),
                                                  ['VehicleType',
                                                   'RegistrationYear',
                                                   'Gearbox', 'Power', 'Mode
l',
                                                   'Mileage',
                                                   'RegistrationMonth',
                                                   'FuelType', 'Brand',
                                                   'NotRepaired',
                                                   'DateCrawled_year',
                                                   'DateCrawled_month',
                                                   'DateCrawled_dow',
                                                   'DateCreated_year',
                                                   'DateCreated_month',
                                                   'DateCreated_dow',
                                                   'LastSeen_year',
                                                   'LastSeen_month',
                                                   'LastSeen_dow'])])),
                ('model',
                 RandomForestRegressor(max_depth=20, max_features='sqrt',
                                       min_samples_leaf=5, n_estimators=150,
                                       n_jobs=-1, random_state=42))])}
```

In [21]:
```
results = pd.DataFrame([res_lr, res_dt, res_rf]).drop(columns=["pipeline"])
results.sort_values("rmse_valid")
```

Out[21]:

	model	rmse_valid	fit_s	pred_s
0	LinearRegression	2007.592103	15.319658	0.846023
1	DecisionTree	2432.216804	21.055801	0.869984
2	RandomForest(sampled)	2809.967002	11.491241	1.524957

In [22]:
```
all_res = [res_lr, res_dt, res_rf]          # agrega aquí otros res_* si entrena
best_res = min(all_res, key=lambda r: r["rmse_valid"])
best_name = best_res["model"]
best_pipe = best_res["pipeline"]            # pipeline YA entrenado (fit), con s

best_name, float(best_res["rmse_valid"])
```

Out[22]:
```
('LinearRegression', 2007.592102635502)
```

Análisis y conclusión del entrenamiento

En esta primera ronda de modelos, la **regresión lineal** funciona como una prueba de cordura y obtiene el mejor desempeño en validación (**RMSE ≈ 2007.6**), con un tiempo de entrenamiento moderado (**~15.9 s**) y predicción rápida (**~0.82 s**). El **árbol de decisión** resulta claramente inferior (**RMSE ≈ 2432.2**) pese a entrenar en un tiempo similar, lo que es consistente con su tendencia a sobreajustar y generalizar peor sin un ajuste más fino.

El **Random Forest** se entrenó con una muestra para evitar tiempos excesivos, pero aun así rinde peor (**RMSE ≈ 2810.0**) y además predice más lento (**~1.47 s**). Este resultado sugiere que, en esta configuración, el bosque no está capturando mejor la estructura del precio y que la combinación de **muestreo + alta dimensionalidad por OHE** puede estar limitando su capacidad predictiva.

En síntesis, el baseline es coherente y deja una señal importante: para superar a la regresión lineal será clave usar un método de **potenciación del gradiente** (p. ej., LightGBM/CatBoost/boosting de sklearn) y ajustar hiperparámetros con cuidado, buscando mejorar RMSE sin sacrificar excesivamente el tiempo de entrenamiento y la velocidad de inferencia.

Análisis del modelo

In [ ]:

Lista de control

Escribe 'x' para verificar. Luego presiona Shift+Enter

- [x] Jupyter Notebook está abierto
- [ ] El código no tiene errores- [ ] Las celdas con el código han sido colocadas en orden de ejecución- [ ] Los datos han sido descargados y preparados- [ ] Los modelos han sido entrenados
- [ ] Se realizó el análisis de velocidad y calidad de los modelos

In [ ]: