Documento técnico

Precio justo de coches

PROYECTO FINAL

Jonathan Perez Sedova

New Technology School

Tokio.

New Technology School

Documento técnico – Proyecto Final

Precio justo de coches usados

Alumno: Jonathan Perez Sedova

Profesor: Juan Carlos Ibáñez

Centro: Tokio School

Módulo: M6 - Presentación de un proyecto Big Data

Fecha de entrega: 19/09/2025

1) Instalar PySpark + montar Drive

```
!pip install pyspark

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Proyecto_BigData').getOrCreate()

from google.colab import drive
drive.mount('/content/drive')

path = "/content/drive/MyDrive/Tokio"
```

Se instala PySpark, se crea la sesión *SparkSession('Proyecto_ BigData')*, se monta Google Drive en Colab y se define la ruta base *path = "/content/drive/MyDrive/Tokio"*. Esto permite acceder y guardar archivos directamente en la carpeta personal de Drive.

2) Cargar el dataset + revisar esquema

```
sdf = spark.read.option("Header", True).option("inferSchema", True).csv(f'{path}/car_sales_data.csv')

sdf.printSchema()
sdf.show(5, truncate = False)
sdf.count()
```

```
root
 |-- Manufacturer: string (nullable = true)
 |-- Model: string (nullable = true)
 |-- Engine size: double (nullable = true)
 |-- Fuel type: string (nullable = true)
 |-- Year of manufacture: integer (nullable = true)
 |-- Mileage: integer (nullable = true)
 |-- Price: integer (nullable = true)

+------------+----------+-----------+---------+-------------------+-------+-----+
|Manufacturer|Model     |Engine size|Fuel type|Year of manufacture|Mileage|Price|
+------------+----------+-----------+---------+-------------------+-------+-----+
|Ford        |Fiesta    |1.0        |Petrol   |2002               |127300 |3074 |
|Porsche     |718 Cayman|4.0        |Petrol   |2016               |57850  |49704|
|Ford        |Mondeo    |1.6        |Diesel   |2014               |39190  |24072|
|Toyota      |RAV4      |1.8        |Hybrid   |1988               |210814 |1705 |
|VW          |Polo      |1.0        |Petrol   |2006               |127869 |4101 |
+------------+----------+-----------+---------+-------------------+-------+-----+
only showing top 5 rows

50000
```

Se lee el archivo CSV con *header=True* y *inferSchema=True*, se imprime el esquema de columnas, se muestran las 5 primeras filas sin truncar y se cuenta el total de registros. Así se comprueba que los tipos de datos se reconocen bien y que el dataset está cargado correctamente.

3) Normalizar nombres y verificación de columnas

```
from pyspark.sql import functions as F
from pyspark.sql import types as T

sdf = (sdf
       .withColumnRenamed("Engine size", "Engine_size")
       .withColumnRenamed("Fuel type", "Fuel_type")
       .withColumnRenamed("Year of manufacture", "Year")
       )


for c in sdf.columns:
  print(c, '- nulos:', sdf.filter(F.col(c).isNull()).count())
```

```
Manufacturer - nulos: 0
Model - nulos: 0
Engine_size - nulos: 0
Fuel_type - nulos: 0
Year - nulos: 0
Mileage - nulos: 0
Price - nulos: 0
```

Se renombran las columnas con espacios porque los nombres sin espacios son más fáciles de usar en el código y evitan errores en el pipeline de preprocesado.

Después, se revisa cada columna para comprobar si contiene valores nulos. En este dataset no aparecen valores faltantes.

4) Variables derivadas

```
sdf = sdf.withColumn('Antiguedad', F.lit(2025) - F.col('Year'))
sdf = sdf.withColumn('km_per_year', F.col('Mileage') / F.when(F.col('Antiguedad') > 0, F.col('Antiguedad')).otherwise(F.lit(1)))

sdf.select('Year','Antiguedad','Mileage','km_per_year').show(5)
```

```
+----+----------+-------+------------------+
|Year|Antiguedad|Mileage|       km_per_year|
+----+----------+-------+------------------+
|2002|        23| 127300| 5534.782608695652|
|2016|         9|  57850| 6427.777777777777|
|2014|        11|  39190|3562.7272727272725|
|1988|        37| 210814| 5697.675675675676|
|2006|        19| 127869|6729.9473684210525|
+----+----------+-------+------------------+
only showing top 5 rows
```

Se crean dos nuevas columnas:

- Antigüedad = 2025 - Year → edad del coche en años.
- km_per_year = Mileage / max(Antiguedad,1) → kilometraje medio anual.

Estas variables ayudan a capturar mejor la relación entre el uso del coche y su precio.

5) EDA rápido

```
sdf.select(
    F.min('Price').alias('min'),
    F.expr('percentile_approx(Price, 0.25)').alias('q1'),
    F.expr('percentile_approx(Price, 0.5)').alias('mediana'),
    F.expr('percentile_approx(Price, 0.75)').alias('q3'),
    F.max('Price').alias('max'),
    F.avg('Price').alias('media')
).show()

sdf_bins = sdf.withColumn(
    'Rango_km',
    F.when(F.col('Mileage').between(0, 49999), '0-49.999')
    .when(F.col('Mileage').between(50000, 99999), '50.000-99.999')
    .when(F.col('Mileage').between(100000, 149999), "100.000-149.999")
    .when(F.col('Mileage').between(150000, 199999), "150.000-199.999")
    .when(F.col('Mileage').between(200000, 249999), "200.000-249.999")
    .when(F.col('Mileage').between(250000, 299999), "250.000-299.999")
    .when(F.col('Mileage').between(300000, 349999), "300.000-349.999")
    .when(F.col('Mileage').between(350000, 399999), "350.000-399.999")
    .when(F.col('Mileage').between(400000, 449999), "400.000-449.999")
    .when(F.col('Mileage').between(450000, 499999), "450.000-499.999")
    .otherwise('500.000+')
)
```

```
agg = (sdf_bins.groupBy("Rango_km")
       .agg(F.avg("Price").alias("Precio_medio"),
            F.expr("percentile_approx(Price, 0.5)").alias("Mediana"),
            F.expr("percentile_approx(Price, 0.1)").alias("P10"),
            F.expr("percentile_approx(Price, 0.9)").alias("P90"),
            F.count("*").alias("N"))
       .orderBy("Rango_km"))
agg.show(20, truncate = False)
```

```
+---+----+-------+-----+------+-----------+
|min|  q1|mediana|   q3|   max|      media|
+---+----+-------+-----+------+-----------+
| 76|3058|   7971|19020|168081|13828.90316|
+---+----+-------+-----+------+-----------+
```

```
+---------------+------------------+-------+-----+-----+-----+
|Rango_km       |Precio_medio      |Mediana|P10  |P90  |N    |
+---------------+------------------+-------+-----+-----+-----+
|0-49.999       |33032.193255627724|28342  |13904|55065|11239|
|100.000-149.999|6871.036396234872 |5578   |2629 |12328|11155|
|150.000-199.999|3314.9839050131927|2726   |1361 |5907 |7580 |
|200.000-249.999|1739.5467642890883|1440   |772  |3004 |4234 |
|250.000-299.999|930.1036769138035 |765    |439  |1596 |1659 |
|300.000-349.999|527.0100401606426 |430    |249  |855  |498  |
|350.000-399.999|290.89130434782606|251    |153  |478  |92   |
|400.000-449.999|191.0             |157    |122  |285  |17   |
|450.000-499.999|331.0             |331    |331  |331  |1    |
|50.000-99.999  |15469.22550831793 |12970  |5652 |26969|13525|
+---------------+------------------+-------+-----+-----+-----+
```

Se calculan estadísticos básicos del precio: mínimo, cuartiles, mediana, máximo y media. Después se agrupan los coches por rangos de kilometraje y se obtiene el precio medio, la mediana y los percentiles P10-P90 en cada rango. Esto resume cómo evoluciona el precio según el kilometraje.

6) Exportar tabla del Gráfico (para Anexo A)

```
agg.coalesce(1).write.mode("overwrite").option("header", True).csv(f'{path}/grafico_1_binss')
print("Guardado en:", 'content/drive/MyDrive/Tokio/grafico_1_binss')
```

```
Guardado en: content/drive/MyDrive/Tokio/grafico_1_binss
```

La tabla de agregación se guarda en formato CSV en la carpeta grafico_1_binss dentro de Drive. Esto permite reutilizarla fácilmente en gráficos y anexos del informe.

7) Train/Test split

```
train, test = sdf.randomSplit([0.8, 0.2], seed= 44)
print('train:', train.count(), ' test:', test.count())
```

```
train: 40081  test: 9919
```

El dataset se divide en entrenamiento (80%) y test (20%), usando semilla fija (seed=44) para asegurar resultados reproducibles.

8) Pipeline de preprocesado (StringIndexer, OneHot, Assembler)

```
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline


cat_cols = ['Manufacturer', 'Model', 'Fuel_type']
num_cols = ['Engine_size', 'Year', 'Mileage', 'Antiguedad', 'km_per_year']

indexers = [StringIndexer(inputCol= c, outputCol= c +'_idx', handleInvalid='keep') for c in cat_cols]
encoders = [OneHotEncoder(inputCols= [c+'_idx'], outputCols= [c+'_ohe']) for c in cat_cols]

feature_cols = [c+'_ohe' for c in cat_cols] + num_cols
assembler = VectorAssembler(inputCols= feature_cols, outputCol= 'feature', handleInvalid= 'keep')

prep_pipline = Pipeline(stages= indexers + encoders + [assembler])
prep_model = prep_pipline.fit(train)
train_prep = prep_model.transform(train)
test_prep = prep_model.transform(test)

train_prep.select('feature', 'Price').show(3, truncate= False)
```

```
+-----------------------------------------------------------------------------+-----+
|feature                                                                      |Price|
+-----------------------------------------------------------------------------+-----+
|(28,[3,15,20,23,24,25,26,27],[1.0,1.0,1.0,4.0,1984.0,89953.0,41.0,2193.9756097560976])|7064 |
|(28,[3,15,20,23,24,25,26,27],[1.0,1.0,1.0,4.0,1985.0,136426.0,40.0,3410.65])         |5316 |
|(28,[3,15,20,23,24,25,26,27],[1.0,1.0,1.0,4.0,1985.0,167593.0,40.0,4189.825])        |4116 |
+-----------------------------------------------------------------------------+-----+
only showing top 3 rows
```

- Variables categóricas: Manufacturer, Model, Fuel_type. Se convierten primero en índices (StringIndexer) y después en vectores OneHotEncoder.
- Variables numéricas: Engine_size, Year, Mileage, Antigüedad, km_per_year.
- Finalmente, todas las columnas transformadas se combinan en un único vector feature con VectorAssembler.

Aquí se incluyen tanto **Year** como **Antigüedad** porque, aunque están relacionadas, aportan información distinta:

- Year refleja el año exacto de fabricación (útil para patrones de mercado o modelos concretos).
- Antigüedad mide la edad en años, que capta el efecto temporal directo sobre la depreciación.

De esta manera el modelo puede aprender mejor relaciones lineales y no lineales.

9) Baseline (mediana) — MAE

```
mediana_train = train.agg(F.expr('percentile_approx(Price, 0.5)')).first()[0]

from pyspark.sql.functions import lit, abs as Fabs
baseline_pred = test.withColumn('y_pred', lit(mediana_train))
mae_baseline = baseline_pred.select(F.avg(Fabs(F.col('Price') - F.col('y_pred'))).alias('MAE')).collect()[0]['MAE']
print('Baseline (mediana) - MAE_test:', round(mae_baseline, 2))
```

```
Baseline (mediana) - MAE_test: 10452.96
```

Se toma la mediana del precio en entrenamiento y se usa como predicción constante para todos los coches. El MAE obtenido sirve como referencia: los modelos más avanzados deben mejorar este valor.

10) Modelo Lineal

```
from pyspark.ml.regression import LinearRegression


lr = LinearRegression(featuresCol="feature", labelCol="Price", elasticNetParam=0.0)
lr_model = lr.fit(train_prep)
pred_lr = lr_model.transform(test_prep).withColumnRenamed("prediction","y_pred")

mae_lr  = pred_lr.select(F.avg(Fabs(F.col("Price") - F.col("y_pred")))).alias("MAE")).collect()[0]["MAE"]
rmse_lr = lr_model.summary.rootMeanSquaredError if hasattr(lr_model,"summary") else float("nan")


from pyspark.ml.evaluation import RegressionEvaluator
evaluator_r2 = RegressionEvaluator(labelCol="Price", predictionCol="y_pred", metricName="r2")
r2_lr = evaluator_r2.evaluate(pred_lr)

print(f"Ridge — MAE_test: {mae_lr:.2f}  RMSE_train: {rmse_lr:.2f}  R2_test: {r2_lr:.3f}")
```

```
Ridge — MAE_test: 5100.14  RMSE_train: 8078.01  R2_test: 0.762
```

Se entrena una regresión lineal con regularización (elasticNetParam=0).

Resultados en test: el error baja notablemente frente al baseline, lo que indica que ya capta parte de la variación en los datos.

11) Random Forest

```
from pyspark.ml.regression import RandomForestRegressor
rf = RandomForestRegressor(featuresCol= 'feature', labelCol= 'Price', seed = 44, numTrees= 200, maxDepth= 12)
rf_model = rf.fit(train_prep)
pred_rf = rf_model.transform(test_prep).withColumnRenamed('prediction', 'y_pred')

mae_rf = pred_rf.select(F.avg(Fabs(F.col('Price') - F.col('y_pred'))).alias('MAE')).collect()[0]['MAE']
r2_rf = evaluator_r2.evaluate(pred_rf)

from pyspark.sql.functions import sqrt
rmse_rf = pred_rf.select(F.avg((F.col('Price') - F.col('y_pred')) ** 2).alias('mse')).withColumn('rmse', F.sqrt(F.col('mse'))).select('rmse').first()['rmse']

print(f"RF — MAE_test: {mae_rf:.2f}  RMSE_test: {rmse_rf:.2f}  R2_test: {r2_rf:.3f}")
```

```
RF — MAE_test: 664.67  RMSE_test: 1220.88  R2_test: 0.995
```

Se entrena un Random Forest Regressor con 200 árboles y profundidad máxima de 12.

Este modelo captura relaciones no lineales y mejora mucho el ajuste (bajo MAE y alto R^2)

12) Gradient Boosted Trees (GBT)

```
from pyspark.ml.regression import GBTRegressor
gbt = GBTRegressor(featuresCol="feature", labelCol="Price", seed= 44, maxIter= 200, maxDepth= 6, stepSize= 0.05)
gbt_model = gbt.fit(train_prep)
pred_gbt = gbt_model.transform(test_prep).withColumnRenamed("prediction","y_pred")

mae_gbt = pred_gbt.select(F.avg(Fabs(F.col("Price") - F.col("y_pred"))).alias("MAE")).collect()[0]["MAE"]
r2_gbt  = evaluator_r2.evaluate(pred_gbt)
rmse_gbt = pred_gbt.select(F.avg((F.col("Price") - F.col("y_pred"))**2).alias("mse")).withColumn("rmse", F.sqrt(F.col("mse"))).select("rmse").first()["rmse"]

print(f"GBT — MAE_test: {mae_gbt:.2f}  RMSE_test: {rmse_gbt:.2f}  R2_test: {r2_gbt:.3f}")
```

```
GBT — MAE_test: 495.49  RMSE_test: 936.08  R2_test: 0.997
```

Se entrena un GBT Regressor con 200 iteraciones, profundidad 6 y tasa de aprendizaje 0,05.

Es el modelo más preciso del proyecto, con MAE muy bajo y R^2 cercano a 1.

13) Tabla comparativa y selección del mejor

```
import pandas as pd
resumen = pd.DataFrame({
    "Modelo": ["Baseline (Mediana)", "Ridge", "Random Forest", "GBT"],
    "MAE_test":[mae_baseline, mae_lr, mae_rf, mae_gbt],
    "RMSE_test":[None, rmse_lr, rmse_rf, rmse_gbt],
    "R2_test":[None, r2_lr, r2_rf, r2_gbt]
})
display(resumen.sort_values("MAE_test"))
```

	Modelo	MAE_test	RMSE_test	R2_test
3	GBT	495.485724	936.084360	0.996770
2	Random Forest	664.666618	1220.883686	0.994506
1	Ridge	5100.135375	8078.011282	0.762367
0	Baseline (Mediana)	10452.959068	NaN	NaN

Se construye una tabla con los resultados (MAE, RMSE, R^2) de Baseline, Ridge, Random Forest y GBT.

El GBT aparece como el mejor modelo y será el elegido.

14) Precio justo (ŷ) + intervalo P10–P90

```
mejor = pred_gbt.withColumn("residuo", F.col("Price") - F.col("y_pred"))


residuos = (rf_model.transform(train_prep)
            .withColumn("residuo", F.col("Price") - F.col("prediction"))
            .select("residuo"))
p10 = residuos.select(F.expr("percentile_approx(residuo, 0.10)")).first()[0]
p90 = residuos.select(F.expr("percentile_approx(residuo, 0.90)")).first()[0]

pred_intervalo = pred_gbt.withColumn("P10", F.col("y_pred") + F.lit(p10)) \
                         .withColumn("P90", F.col("y_pred") + F.lit(p90))

pred_intervalo.select("y_pred","P10","P90").show(5)
```

```
+------------------+------------------+------------------+
|            y_pred|               P10|               P90|
+------------------+------------------+------------------+
| 2252.086993792953|1390.4536581310567|3147.2705564674097|
| 2062.246013447763|1200.6126777858667|2957.4295761222197|
|10010.030295814473| 9148.396960152577|10905.213858488929|
|  4558.91952813765|3697.2861924757535|5454.1030908121065|
| 2206.650491682613|1345.0171560207168|  3101.83405435707|
+------------------+------------------+------------------+
only showing top 5 rows
```

Se calculan los residuos de entrenamiento con el modelo Random Forest y se extraen los percentiles 10 y 90. Estos valores se suman a la predicción del GBT para obtener un rango de incertidumbre [P10, P90] alrededor del precio justo.

15) Etiquetas infravalorado/justo/sobrevalorado (±15%)

```
UMBRAL = 0.15
clasificado = (pred_intervalo
               .withColumn("Etiqueta",
                           F.when(F.col("Price") <= (1-UMBRAL)*F.col("y_pred"), F.lit("Infravalorado"))
                           .when(F.col("Price") >= (1+UMBRAL)*F.col("y_pred"), F.lit("Sobrevalorado"))
                           .otherwise(F.lit("Justo"))))

clasificado.groupBy("Etiqueta").count().show()
```

```
+-------------+-----+
|     Etiqueta|count|
+-------------+-----+
|Sobrevalorado|  526|
|        Justo| 9111|
|Infravalorado|  282|
+-------------+-----+
```

Se asigna una etiqueta a cada coche según la diferencia entre su precio real y el estimado:
Infravalorado:

- Price ≤ 0.85 × ŷ
- Sobrevalorado: Price ≥ 1.15 × ŷ
- Justo: el resto.

16) Guardar artefactos (para Anexos)

```
resumen.to_csv(f"{path}/resultados_modelos_test.csv", index=False)

cols_out = ["Manufacturer","Model","Year","Mileage","Fuel_type","Engine_size","Price","y_pred","P10","P90","Etiqueta"]
clasificado.select(*[c for c in cols_out if c in clasificado.columns])
           .limit(2000)
           .toPandas()
           .to_csv(f"{path}/predicciones_intervalos_etiquetas.csv", index=False)

print(f"Guardado:{path} - resultados_modelos_test.csv\n - predicciones_intervalos_etiquetas.csv")
```

```
Guardado:
 - resultados_modelos_test.csv
 - predicciones_intervalos_etiquetas.csv
```

Se exportan dos archivos CSV:

- *resultados_modelos_test.csv* → con la tabla comparativa de modelos.
- *predicciones_intervalos_etiquetas.csv* → con ejemplos de predicciones, intervalos y etiquetas.