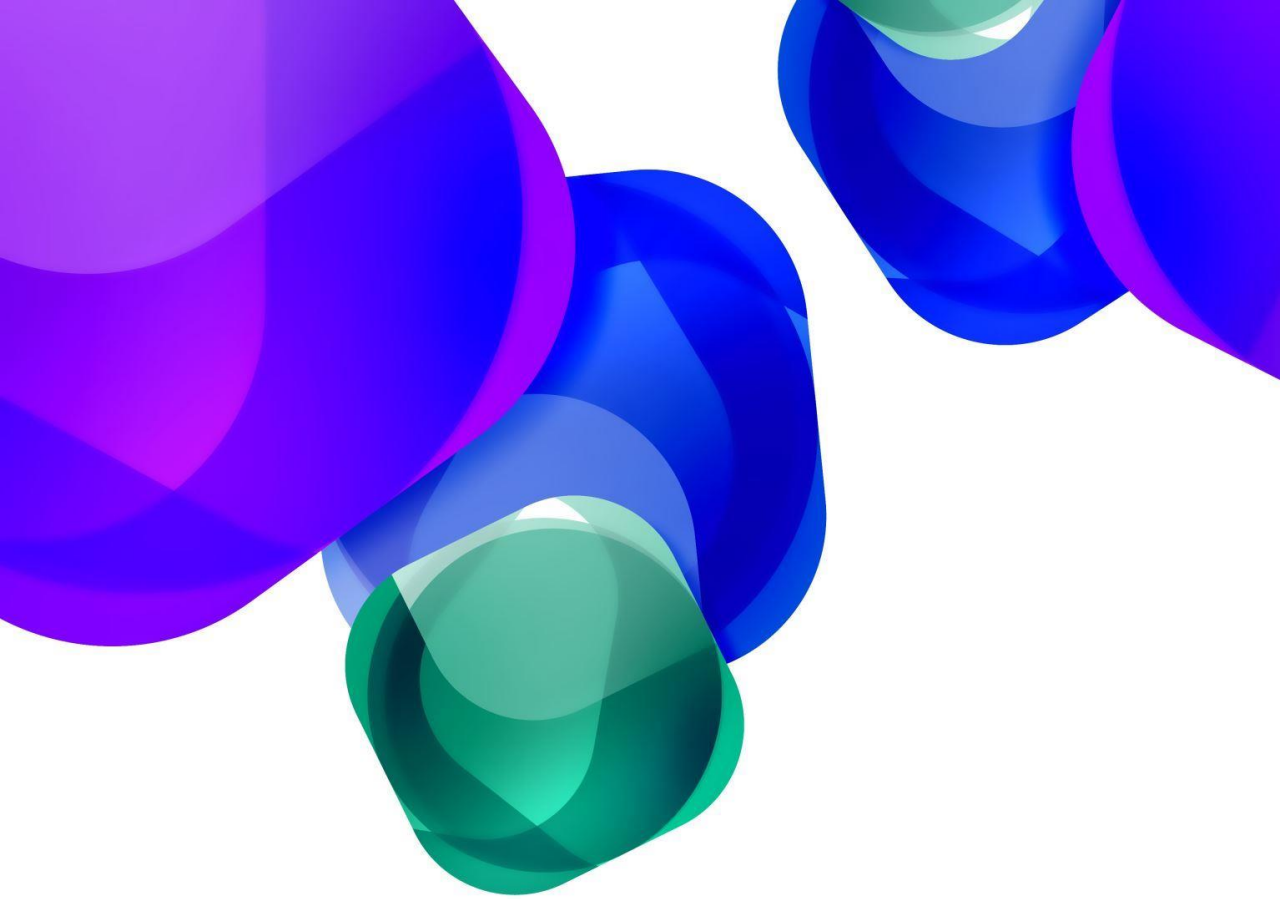


- Caso Práctico VII
- Módulo VII
- MSc Data Science & Business Analytics

@IMF Smart Education

Julián Vázquez Sampedro

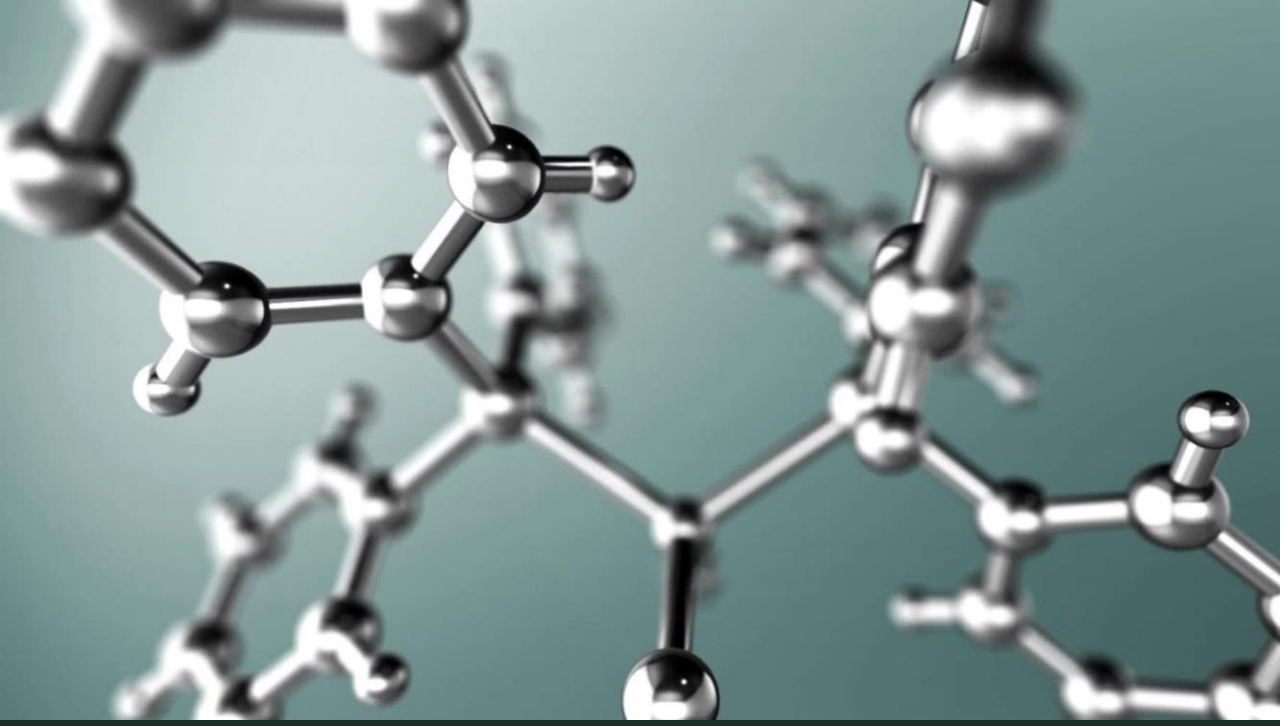
Análisis de Negocio Metodología CRISP-DM

- Valencian Telecommunications, SA (VALTEL) es una compañía prestadora de servicios de telecomunicaciones
- En su plan estratégico, se contempla la diversificación de servicios y se plantea el diseño de nuevas áreas de la compañía dedicadas a comercializar a través de internet bienes de interés para el colectivo de ciudadanos no nacionales
- VALTEL desea ofertar dentro de su *app* móvil un servicio que permita a sus clientes buscar vehículos en venta y ofrecerles un precio competitivo. Para conocer dicho precio, el área de I+D ha conseguido un *dataset* (**Coches_Segunda_Mano.csv**) con las ventas de vehículos de segunda mano y encarga al área de Ciencia de Datos la estimación del precio de cada vehículo para poder elaborar una tarifa competitiva.
- Por tanto, el trabajo para realizar será el siguiente:

1 - Utilizar la metodología CRISP-DM para estimar el precio de un vehículo (campo *Precio*). Se recomienda utilizar como base el cuaderno del caso de repaso o bien *BigML*.

2 - Realizar una narrativa que comunique los hallazgos. Por simplicidad, se recomienda utilizar PowerPoint o equivalente para realizar la narrativa.

1. Objetivo



2. Metodología CRISP-DM

La Metodología CRISP-DM (CRoss-Industry Standard Process for Data Mining) es un proceso estándar y robusto utilizado en el campo del análisis de datos y la minería de datos. Proporciona una estructura sólida que guía a los profesionales a través de las fases de un proyecto de minería de datos, desde la comprensión del negocio y los objetivos del proyecto hasta la implementación y el seguimiento de los resultados.

Las fases principales de la Metodología CRISP-DM son las siguientes:

- Comprensión del negocio: En esta fase, se define el problema del negocio, los objetivos del proyecto y los requisitos desde una perspectiva de negocio.
- Comprensión de los datos: Se recopilan los datos necesarios para el proyecto, se exploran y se comprenden en profundidad.
- Preparación de los datos: Los datos se limpian, se transforman y se preparan para el modelado.
- Modelado: Se selecciona y se aplica un modelo de minería de datos apropiado para el problema en cuestión.
- Evaluación: Se evalúa la calidad del modelo y se determina si cumple con los objetivos del negocio.
- Despliegue: Se implementa el modelo en el entorno de producción.
- Seguimiento: Se realiza un seguimiento continuo del rendimiento del modelo y se realizan ajustes según sea necesario.

3. Exploración inicial

Emplearemos Python para realizar una primera exploración inicial. Se importarán las siguientes librerías necesarias:

```
import pandas as pd

from sklearn.linear_model import LinearRegression, Ridge, Lasso

from sklearn.preprocessing import MinMaxScaler,
OneHotEncoder

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error, r2_score,
mean_squared_error

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import linear_model
```

3.1 Tamaño del Dataset

```python
# Cargamos los datos del archivo csv
df = pd.read_csv("Coches_Segunda_Mano.csv")
# Observamos el tamaño del conjunto de datos
print("Tamaño del conjunto de datos: ", df.shape)
# Mostramos los primeros 10 registros
print("Primeros 10 registros: ", df.head(10))
# Mostramos los últimos 10 registros
print("Ultimos 10 registros: ", df.tail(10))
# Mostramos el tipo de dato de los primeros 10 registros
print("Tipo de dato: ", df.iloc[:10].dtypes)
```

```
Tamaño del conjunto de datos:  (11914, 16)
Primeros 10 registros:    Marca      Modelo  year                     Combustible
0   BMW  1 Series M  2011  premium unleaded (required)  335.0         6.0
1   BMW    1 Series  2011  premium unleaded (required)  300.0         6.0
2   BMW    1 Series  2011  premium unleaded (required)  300.0         6.0
3   BMW    1 Series  2011  premium unleaded (required)  230.0         6.0
4   BMW    1 Series  2011  premium unleaded (required)  230.0         6.0
5   BMW    1 Series  2012  premium unleaded (required)  230.0         6.0
6   BMW    1 Series  2012  premium unleaded (required)  300.0         6.0
7   BMW    1 Series  2012  premium unleaded (required)  300.0         6.0
8   BMW    1 Series  2012  premium unleaded (required)  230.0         6.0
9   BMW    1 Series  2013  premium unleaded (required)  230.0         6.0

  Transmision          Traccion  Puertas  \
0      MANUAL  rear wheel drive      2.0
1      MANUAL  rear wheel drive      2.0
2      MANUAL  rear wheel drive      2.0
3      MANUAL  rear wheel drive      2.0
4      MANUAL  rear wheel drive      2.0
5      MANUAL  rear wheel drive      2.0
6      MANUAL  rear wheel drive      2.0
7      MANUAL  rear wheel drive      2.0
8      MANUAL  rear wheel drive      2.0
9      MANUAL  rear wheel drive      2.0
...
Consumo Ciudad       float64
Popularidad            int64
Precio               float64
dtype: object
```

3.2 Tamaño del Dataset (Resultados)

```
Tamaño del conjunto de datos:  (11914, 16)
Primeros 10 registros:    Marca      Modelo  year                 Combustible
0   BMW  1 Series M  2011  premium unleaded (required)  335.0          6.0
1   BMW    1 Series  2011  premium unleaded (required)  300.0          6.0
2   BMW    1 Series  2011  premium unleaded (required)  300.0          6.0
3   BMW    1 Series  2011  premium unleaded (required)  230.0          6.0
4   BMW    1 Series  2011  premium unleaded (required)  230.0          6.0
5   BMW    1 Series  2012  premium unleaded (required)  230.0          6.0
6   BMW    1 Series  2012  premium unleaded (required)  300.0          6.0
7   BMW    1 Series  2012  premium unleaded (required)  300.0          6.0
8   BMW    1 Series  2012  premium unleaded (required)  230.0          6.0
9   BMW    1 Series  2013  premium unleaded (required)  230.0          6.0

  Transmision          Traccion  Puertas  \
0      MANUAL  rear wheel drive      2.0
1      MANUAL  rear wheel drive      2.0
2      MANUAL  rear wheel drive      2.0
3      MANUAL  rear wheel drive      2.0
4      MANUAL  rear wheel drive      2.0
5      MANUAL  rear wheel drive      2.0
6      MANUAL  rear wheel drive      2.0
7      MANUAL  rear wheel drive      2.0
8      MANUAL  rear wheel drive      2.0
9      MANUAL  rear wheel drive      2.0
...
Consumo Ciudad       float64
Popularidad            int64
Precio               float64
dtype: object
```

Estos resultados proporcionan información relevante sobre el conjunto de datos que se está analizando. Aquí está un comentario sobre cada parte:

- "Tamaño del conjunto de datos: (11914, 16)": Indica que el conjunto de datos contiene 11914 filas y 16 columnas, lo que significa que hay 11914 registros y 16 variables o características en total.

- "Primeros 10 registros": Muestra los primeros 10 registros del conjunto de datos, que incluyen detalles como la marca, el modelo, el año, el tipo de combustible, los caballos de fuerza, los cilindros, la transmisión, la tracción, el número de puertas y otros detalles de los automóviles.

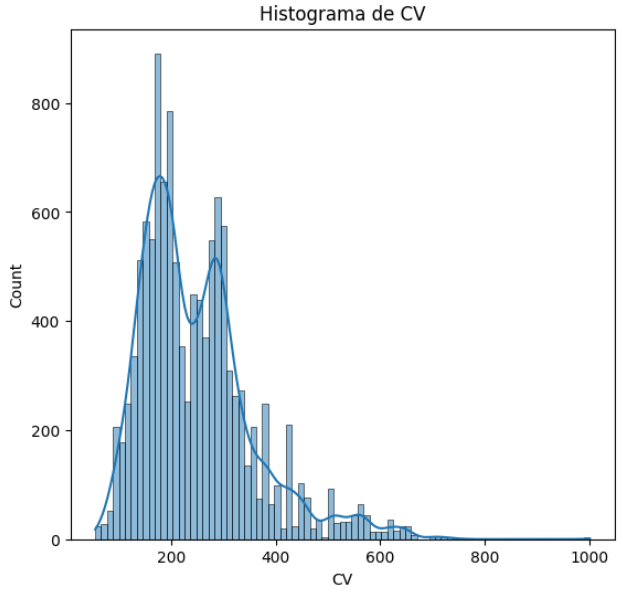
- "Descripción del conjunto de datos": Proporciona información sobre los tipos de datos de las diferentes columnas en el conjunto de datos. Por ejemplo, se mencionan los tipos de datos como object (para variables de texto), int64 (para variables numéricas enteras) y float64 (para variables numéricas con decimales) junto con los nombres de las columnas correspondientes.

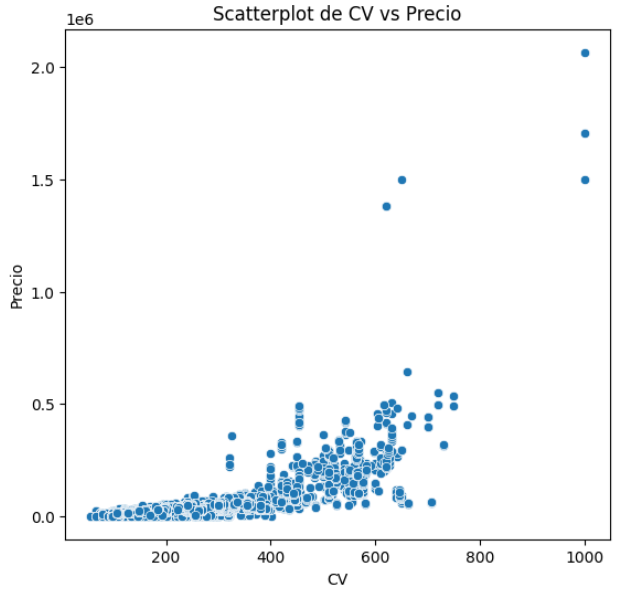
4. Análisis Univariante

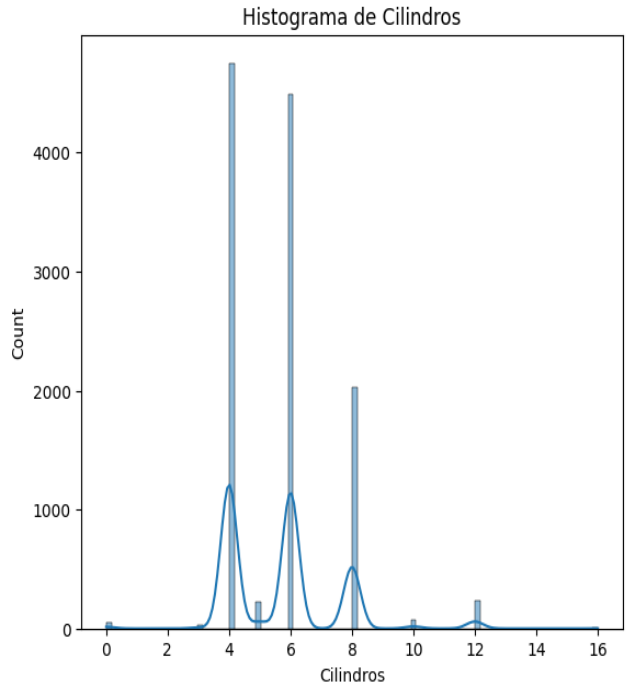
```python
# Dividimos las variables en cualitativas y cuantitativas
variables_cualitativas = df.select_dtypes(include=['object'])
variables_cuantitativas = df.select_dtypes(include=['int64', 'float64'])

# Análisis de variables cualitativas
for column in variables_cualitativas:
    num_classes = len(df[column].unique())
    print(f"Variable: {column}, Número de clases: {num_classes}")
    plt.figure(figsize=(8, 6))
    sns.countplot(x=column, data=df)
    plt.title(f"Distribucion de valores para {column}")
    plt.show()

# Análisis de variables cuantitativas
for column in variables_cuantitativas:
    mean_val = df[column].mean()
    min_val = df[column].min()
    max_val = df[column].max()
    median_val = df[column].median()
    print(f"Variable: {column}")
    print(f"Media: {mean_val}, Mínimo: {min_val}, Máximo: {max_val}, Mediana: {median_val}")

    plt.figure(figsize=(14, 6))
    plt.subplot(1, 2, 1)
    sns.histplot(df[column], kde=True)
    plt.title(f"Histograma de {column}")

    plt.subplot(1, 2, 2)
    sns.scatterplot(x=column, y='Precio', data=df)
    plt.title(f"Scatterplot de {column} vs Precio")
    plt.show()
```

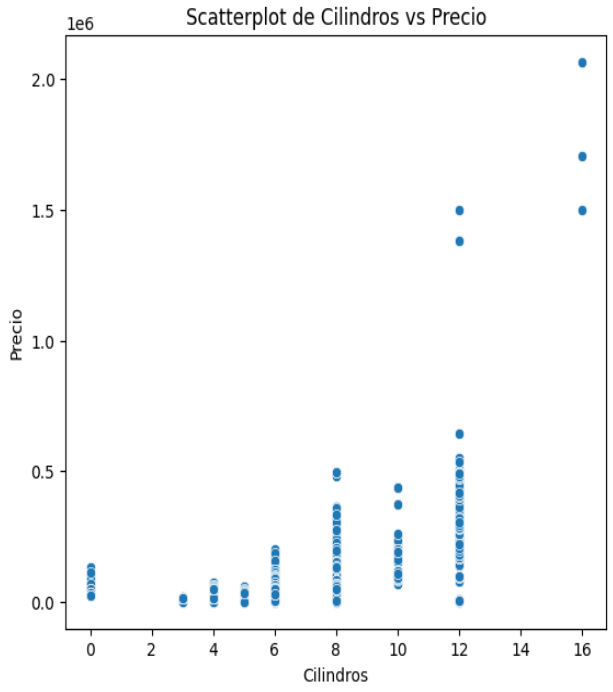
4.1 Resultados gráficos del análisis

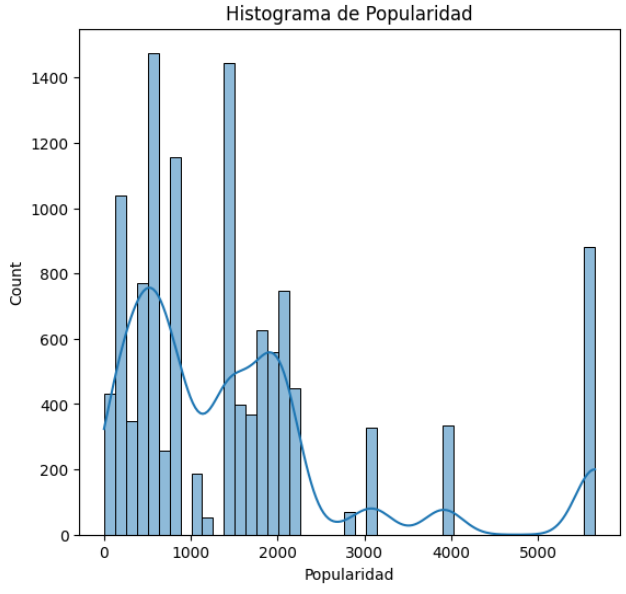
Se presentan algunos de los resultados del análisis univariante

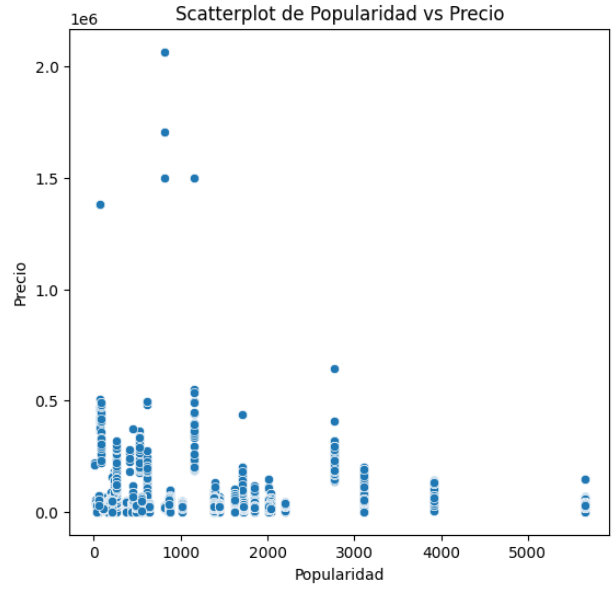












4.1 Explicación del análisis univariante

```
Variable: Marca, Número de clases: 48
Variable: Modelo, Número de clases: 915
Variable: Combustible, Número de clases: 11
Variable: Transmision, Número de clases: 5
Variable: Traccion, Número de clases: 4
Variable: Mercado, Número de clases: 72
Variable: tipo, Número de clases: 3
Variable: Estilo, Número de clases: 16
Variable: year
Media: 2010.384337753903, Mínimo: 1990, Máximo: 2017, Mediana: 2015.0
Variable: Puertas
Media: 3.4360933825999327, Mínimo: 2.0, Máximo: 4.0, Mediana: 4.0
Variable: Consumo Carretera
Media: 26.637485311398354, Mínimo: 12.0, Máximo: 354.0, Mediana: 26.0
Variable: Consumo Ciudad
Media: 19.73325499412456, Mínimo: 7.0, Máximo: 137.0, Mediana: 18.0
Variable: Popularidad
Media: 1554.9111969111968, Mínimo: 2, Máximo: 5657, Mediana: 1385.0
Variable: Precio
Media: 40594.737032063116, Mínimo: 2000.0, Máximo: 2065902.0, Mediana: 29995.0
```

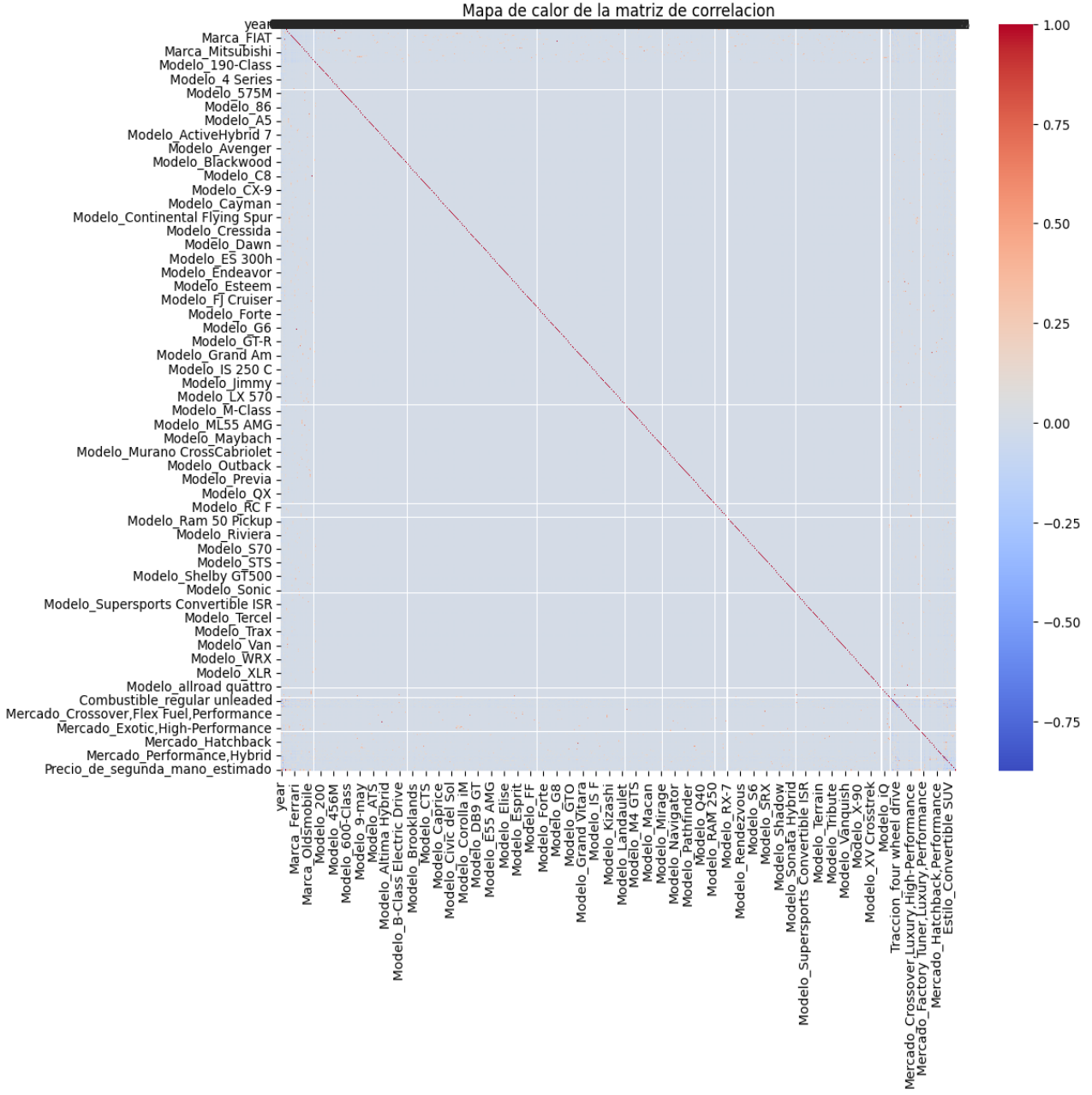
Estos resultados del análisis univariante proporcionan información importante sobre varias variables en el conjunto de datos.

- **Variable: Marca, Número de clases: 48**: Indica que hay 48 marcas diferentes en el conjunto de datos, lo que sugiere una diversidad considerable en términos de fabricantes de automóviles.
- **Variable: Modelo, Número de clases: 915**: Muestra que hay 915 modelos únicos en el conjunto de datos, lo que indica una amplia variedad de modelos de automóviles.
- **Variable: Combustible, Número de clases: 11**: Indica que hay 11 tipos diferentes de combustible en el conjunto de datos, lo que refleja la variedad de opciones de combustible utilizadas por los vehículos.
- **Variable: Transmisión, Número de clases: 5**: Muestra que hay 5 tipos diferentes de transmisiones en el conjunto de datos, lo que sugiere una variedad de tipos de transmisión utilizados en los vehículos.
- **Variable: Mercado, Número de clases: 72**: Muestra que hay 72 tipos diferentes de mercados representados en el conjunto de datos, lo que sugiere una diversidad geográfica en términos de regiones de mercado.
- **Variable: Tipo, Número de clases: 3**: Indica que hay 3 tipos diferentes de vehículos en el conjunto de datos, lo que refleja una categorización general de los automóviles en el conjunto de datos.
- **Variable: Estilo, Número de clases: 16**: Muestra que hay 16 estilos diferentes de vehículos en el conjunto de datos, lo que sugiere una diversidad en términos de estilos de carrocería de los automóviles.
- **Variable: year**: La media, el mínimo, el máximo y la mediana muestran la distribución de los años de fabricación de los automóviles en el conjunto de datos.
- **Variable: Puertas**: La media, el mínimo, el máximo y la mediana muestran la distribución del número de puertas de los automóviles en el conjunto de datos.
- **Variable: Popularidad**: La media, el mínimo, el máximo y la mediana indican la distribución de la popularidad de los automóviles en el conjunto de datos.
- **Variable: Precio**: La media, el mínimo, el máximo y la mediana muestran la distribución de los precios de los automóviles en el conjunto de datos.

5. Análisis Multivariante

```
"""
Para realizar un análisis multivariante se usará una matriz de correlación y un HeatMap
- Matriz de correlación: Tabla que muestra las correlaciones entre todas las variables en un dataset
Las correlaciones comprenden (-1, 1) donde -1 indica correlación negativa perfecta, 1 indica correlación positiva perfecta y 0 ausencia de correlación. Gracias a esto se odentifican patrones y relaciones en el conjunto de datos
- HeatMap: Mapa de calor o representación gráfica de datos en la que los valores de una matriz se representan como colores.
Colores más oscuros implican correlaciones más fuertes, y colores tenues indican ausencia de correlación o correlación débil.
Los HeatMaps proporcionan una representación visual efectiva en el análisis multivariante.
"""
# Calculamos la matriz de correlacion
df = pd.get_dummies(df)
corr_matrix = df.corr()
# Creamos un HeatMap de la matriz de correlación
plt.figure(figsize = (12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
# Annot en True para mostrar valores de matriz en celdas correspondientes
# fmt = '.2f' indica valores con dos decimales flotantes
plt.title('Mapa de calor de la matriz de correlacion')
plt.show()
```

Se presentan algunos de los resultados del análisis multivariante (HeatMap):



5.1 Resultados Análisis Multivariante

	year	CV	Cilindros	Puertas	Consumo Carretera	Consumo Ciudad
year	1.000000	0.337445	−0.026711	0.245974	0.265607	0.220111
CV	0.337445	1.000000	0.788187	−0.128624	−0.423646	−0.474667
Cilindros	−0.026711	0.788187	1.000000	−0.147553	−0.614706	−0.634463
Puertas	0.245974	−0.128624	−0.147553	1.000000	0.116884	0.137317
Consumo Carretera	0.265607	−0.423646	−0.614706	0.116884	1.000000	0.842833
...	...	...	...	...	...	...
Estilo_Regular Cab Pickup	−0.125201	−0.050514	0.031919	−0.297078	−0.129748	−0.092494
Estilo_Sedan	0.020542	−0.052468	−0.111749	0.365808	0.270168	0.165327
Estilo_Wagon	−0.057618	−0.107133	−0.105470	0.144384	0.047887	0.052363
Precio_de_segunda_mano_estimado	0.216931	0.680837	0.571402	−0.147982	−0.215021	−0.240675
Precio_de_segunda_mano_ajustado	0.216931	0.680837	0.571402	−0.147982	−0.215021	−0.240675

	Precio_de_segunda_mano_estimado	Precio_de_segunda_mano_ajustado
...		
Precio_de_segunda_mano_estimado	1.000000	1.000000
Precio_de_segunda_mano_ajustado	1.000000	1.000000

- En la diagonal principal, los valores son 1, lo que indica una correlación perfecta de las variables consigo mismas, lo cual es obvio ya que se están comparando las variables consigo mismas.

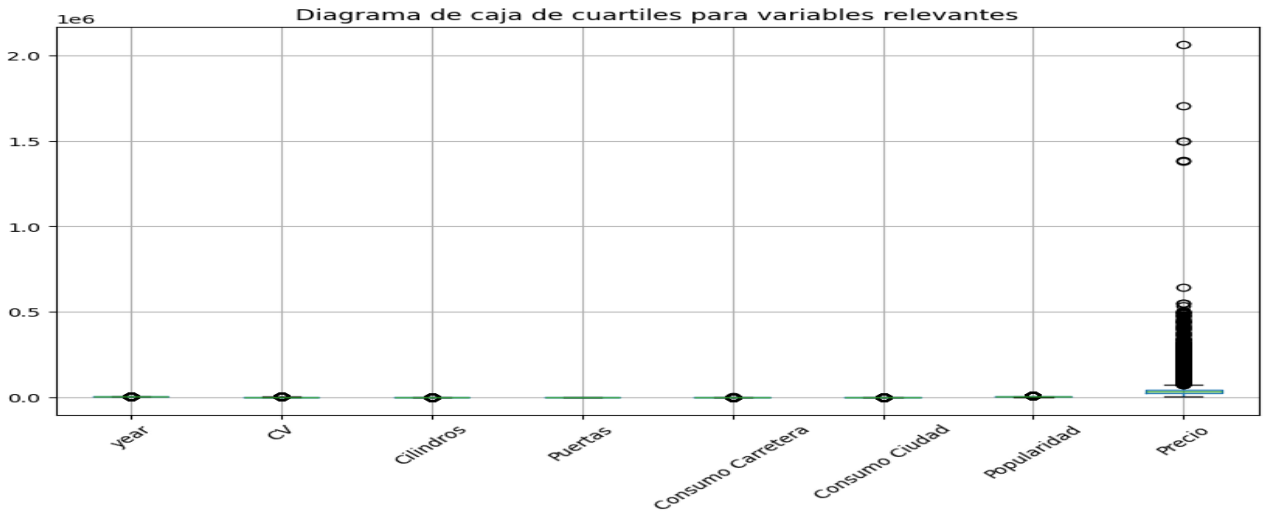
- Las variables "CV" (caballos de fuerza) y "Cilindros" muestran una correlación positiva significativa (0.788187), lo que sugiere que los vehículos con más caballos de fuerza tienden a tener más cilindros.

- "Consumo Carretera" y "Consumo Ciudad" muestran una correlación negativa con "CV" y "Cilindros", lo que sugiere que los vehículos con más caballos de fuerza y más cilindros tienden a tener un menor consumo de combustible.

- "Precio_de_segunda_mano_estimado" y "Precio_de_segunda_mano_ajustado" tienen una correlación perfecta de 1, lo que sugiere que estas dos variables están altamente relacionadas entre sí, lo cual es esperable dado que generalmente se utilizan para medir lo mismo.

6. Análisis de calidad del dato

```python
# Verifica la presencia de valores nulos
print(df.isnull().sum())
# Identificamos valores atípicos
Q1 = variables_cuantitativas.quantile(0.25)
Q3 = variables_cuantitativas.quantile(0.75)
IQR = (Q3 - Q1)
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
# Alineamos el DataFrame con los límites
df_aligned, lower_bound_aligned = df.align(lower_bound, axis=1, join='inner')
df_aligned, upper_bound_aligned = df.align(upper_bound, axis=1, join='inner')

# Identificamos valores atípicos
outliers = ((df_aligned < lower_bound_aligned) | (df_aligned > upper_bound_aligned)).sum()
print(outliers)
# Seleccionamos columnas relevantes
columnas_relevantes = ['year', 'CV', 'Cilindros', 'Puertas',
                       'Consumo Carretera', 'Consumo Ciudad', 'Popularidad', 'Precio']

# Creamos gráficos de caja para los cuartiles
plt.figure(figsize=(10, 6))
df[columnas_relevantes].boxplot()
plt.title('Diagrama de caja de cuartiles para variables relevantes')
plt.xticks(rotation=45)
plt.show()
```



- **Year: 0**: Indica que hay 0 valores faltantes en la variable 'year', lo que sugiere que no faltan datos en esta columna en el conjunto de datos.
- **CV: 69, Cilindros: 30, Puertas: 6, Consumo Carretera: 0**: Indica el número de valores faltantes en cada una de estas variables, lo que sugiere que estas variables tienen valores faltantes en diferentes cantidades. Por ejemplo, 'CV' tiene 69 valores faltantes, 'Cilindros' tiene 30 valores faltantes, 'Puertas' tiene 6 valores faltantes y 'Consumo Carretera' no tiene valores faltantes.
- **Year: 661, CV: 509, Cilindros: 357, Puertas: 0**: Estos números representan valores atípicos o valores inesperados en las variables mencionadas. Por ejemplo, 'year' tiene 661 valores únicos, 'CV' tiene 509 valores únicos y 'Cilindros' tiene 357 valores únicos, lo que sugiere una amplia variación en estos atributos.

7. Preparación del dataset

```
# Eliminación de valores nulos
df = df.dropna()

# Eliminación de valores duplicados
df = df.drop_duplicates()

# Normalización de la variable "Popularidad" utilizando MinMaxScaler
scaler = MinMaxScaler()
df['Popularidad'] = scaler.fit_transform(df['Popularidad'].values.reshape(-1, 1))
print(df)
```

La preparación del dataset se refiere al proceso de manipulación y transformación de los datos en bruto para que estén en un formato adecuado y limpio para su posterior análisis o modelado. Este proceso es crucial para garantizar que los datos sean precisos, completos y relevantes para los objetivos específicos del análisis o del proyecto.

Aquí hay una explicación de cada línea:

- **Eliminación de valores nulos**:
  - **df = df.dropna()**: Esta línea elimina todas las filas que contienen valores nulos o NaN en el DataFrame. Ayuda a limpiar el conjunto de datos de cualquier fila que contenga datos faltantes, lo que facilita el análisis y el modelado posteriores.
- **Eliminación de valores duplicados**:
  - **df = df.drop_duplicates()**: Esta línea elimina las filas duplicadas en el DataFrame, si las hay. Al eliminar duplicados, se asegura de que cada fila en el conjunto de datos sea única, lo que puede ser crucial para evitar problemas en el análisis posterior.
- **Normalización de la variable "Popularidad" utilizando MinMaxScaler**:
  - **scaler = MinMaxScaler()**: Crea un objeto de escalador MinMaxScaler, que se utiliza para normalizar los valores de la variable "Popularidad" en un rango específico.
  - **df['Popularidad'] = scaler.fit_transform(df['Popularidad'].values.reshape(-1, 1))**: Aplica el escalado MinMax a la columna 'Popularidad' del DataFrame, lo que garantiza que los valores estén en el rango específico definido por el escalador. La función **reshape** se utiliza para darle al array de valores la forma requerida para el escalado.

7.1 Preparación del dataset (Resultados)

```
# Eliminación de valores nulos
df = df.dropna()

# Eliminación de valores duplicados
df = df.drop_duplicates()

# Normalización de la variable "Popularidad" utilizando MinMaxScaler
scaler = MinMaxScaler()
df['Popularidad'] = scaler.fit_transform(df['Popularidad'].values.reshape(-1, 1))
print(df)
```

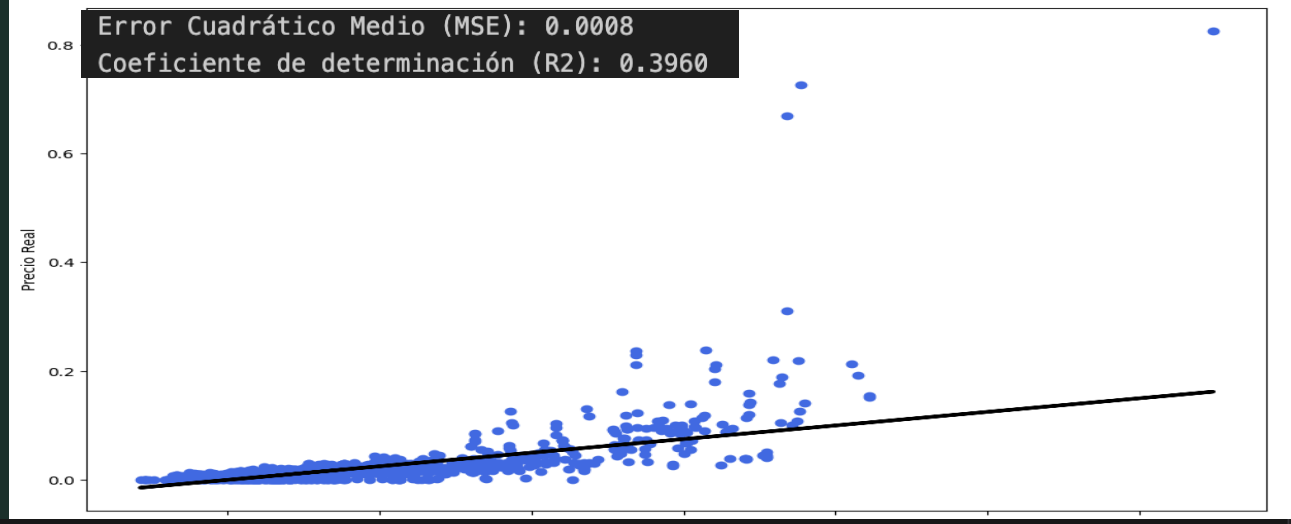
```
       year     CV  Cilindros  Puertas  Consumo Carretera  Consumo Ciudad  \
0      2011  335.0        6.0      2.0               26.0            19.0
1      2011  300.0        6.0      2.0               28.0            19.0
2      2011  300.0        6.0      2.0               28.0            20.0
3      2011  230.0        6.0      2.0               28.0            18.0
4      2011  230.0        6.0      2.0               28.0            18.0
...     ...    ...        ...      ...                ...             ...
11909  2012  300.0        6.0      4.0               23.0            16.0
11910  2012  300.0        6.0      4.0               23.0            16.0
11911  2012  300.0        6.0      4.0               23.0            16.0
11912  2013  300.0        6.0      4.0               23.0            16.0
11913  2006  221.0        6.0      4.0               26.0            17.0

       Popularidad   Precio  Marca_Acura  Marca_Alfa Romeo  ...  \
0         0.692131  46135.0        False             False  ...
1         0.692131  40650.0        False             False  ...
2         0.692131  36350.0        False             False  ...
3         0.692131  29450.0        False             False  ...
4         0.692131  34500.0        False             False  ...
...            ...      ...          ...               ...  ...
11909     0.035721  46120.0         True             False  ...
11910     0.035721  56670.0         True             False  ...
11911     0.035721  50620.0         True             False  ...
11912     0.035721  50920.0         True             False  ...
11913     0.010433  28995.0        False             False  ...
...
11912                    False         False         False
11913                    False          True         False
```

Estos resultados son una muestra de los datos preparados después del proceso de preprocesamiento y transformación:

- **Datos numéricos**: Se muestran valores numéricos para variables como 'year', 'CV', 'Cilindros', 'Puertas', 'Consumo Carretera', 'Consumo Ciudad', 'Popularidad' y 'Precio'. Estos valores representan diferentes características de los automóviles en el conjunto de datos.
- **Datos categóricos**: Se presentan variables booleanas como 'Marca_Acura', 'Marca_Alfa Romeo' y otras, que representan diferentes marcas de automóviles y características asociadas. Estas variables suelen ser el resultado de la codificación de variables categóricas para su uso en modelos de aprendizaje automático.

Los datos proporcionados han sido procesados y transformados para su uso en un análisis o modelo específico, lo que incluye técnicas como la codificación one-hot para variables categóricas, normalización de datos numéricos y otros pasos de preprocesamiento.

8. Desarrollo y evaluación del modelo: Regresión Lineal Múltiple




```python
# Definimos las columnas predictoras y la columna objetivo
x_cols = ['year', 'CV', 'Cilindros', 'Consumo Carretera', 'Consumo Ciudad', 'Popularidad']
y_col = 'Precio'

# Normalizamos las variables para una mejor interpretación
scaler = MinMaxScaler()
x = scaler.fit_transform(df[x_cols])
y = scaler.fit_transform(df[y_col].values.reshape(-1, 1))
# Se convierte a un array 2D antes de normalizar

# Generamos el conjunto de entrenamiento y prueba
# (test size 0.2 implica que 20% de datos para pruebas y 80% para entrenamiento)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Entrenamos el modelo de regresión lineal
reg = linear_model.LinearRegression()
reg.fit(x_train, y_train)

# Realizamos predicciones en el conjunto de prueba
y_pred = reg.predict(x_test)

# Calculamos métricas de evaluación
print(f"Error Cuadrático Medio (MSE): {mean_squared_error(y_test, y_pred):.4f}")
print(f"Coeficiente de determinación (R2): {r2_score(y_test, y_pred):.4f}")

# Visualizamos los resultados
fig, ax = plt.subplots(figsize=(12, 8))
plt.scatter(y_pred, y_test, color='royalblue')
plt.plot(y_pred, y_pred, color='black', linewidth=3)

plt.xlabel('Precio Estimado')
plt.ylabel('Precio Real')
plt.show()
```

```python
# Ridge
ridge = Ridge(alpha=0.1)
ridge.fit(x_train, y_train)
y_pred_ridge = ridge.predict(x_test)

print(f"Error cuadrático medio (MSE) para Ridge: {mean_squared_error(y_test, y_pred_ridge):.4f}")
print(f"Coeficiente de determinación (R2) para Ridge: {r2_score(y_test, y_pred_ridge):.4f}")

# Lasso
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.1)  # Selecciona un valor apropiado para alpha
lasso.fit(x_train, y_train)
y_pred_lasso = lasso.predict(x_test)

print(f"Error cuadrático medio (MSE) para Lasso: {mean_squared_error(y_test, y_pred_lasso):.4f}")
print(f"Coeficiente de determinación (R2) para Lasso: {r2_score(y_test, y_pred_lasso):.4f}")
```

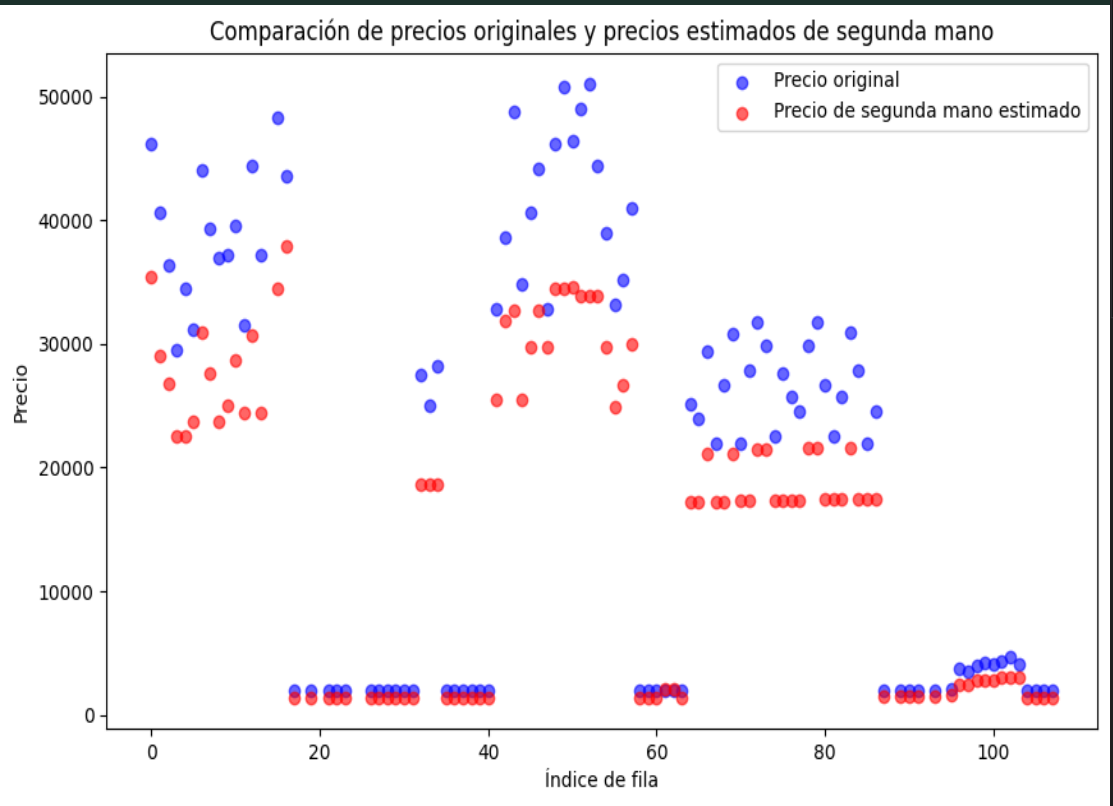
✓ 0.1s Python

```
Error cuadrático medio (MSE) para Ridge: 0.0008
Coeficiente de determinación (R2) para Ridge: 0.3958
Error cuadrático medio (MSE) para Lasso: 0.0013
Coeficiente de determinación (R2) para Lasso: -0.0010
```

8.1 Desarrollo y evaluación del modelo: Random Forest

```python
from sklearn.ensemble import RandomForestRegressor

# Definir las columnas predictoras y la columna objetivo
x_cols = ['year', 'CV', 'Cilindros', 'Consumo Carretera', 'Consumo Ciudad', 'Popularidad']
y_col = 'Precio'

# Crear y entrenar el modelo de RandomForestRegressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(df[x_cols], df[y_col])

# Ajustar el precio estimado de segunda mano considerando el descuento
porcentaje_descuento = 0.7  # 30% de descuento
df['Precio_de_segunda_mano_estimado'] = rf_model.predict(df[x_cols])
df['Precio_de_segunda_mano_ajustado'] = df['Precio_de_segunda_mano_estimado'] * porcentaje_descuento

# Visualizar los resultados
print(df[['Precio', 'Precio_de_segunda_mano_estimado', 'Precio_de_segunda_mano_ajustado']].head(100))

# Crear gráfico de dispersión para comparar los precios originales y los precios estimados de segunda mano
fig, ax = plt.subplots(figsize=(10, 6))
ax.scatter(df.index[:100], df['Precio'].head(100), label='Precio original', color='b', alpha=0.6)
ax.scatter(df.index[:100], df['Precio_de_segunda_mano_ajustado'].head(100), label='Precio de segunda mano estimado', color='r', alpha=0.6)
ax.set_xlabel('Índice de fila')
ax.set_ylabel('Precio')
ax.set_title('Comparación de precios originales y precios estimados de segunda mano')
ax.legend()
plt.show()
```

✓ 1.6s



	Precio	Precio_de_segunda_mano_estimado	Precio_de_segunda_mano_ajustado
0	46135.0	50654.916667	35458.441667
1	40650.0	41432.300000	29002.610000
2	36350.0	38295.300000	26806.710000
3	29450.0	32112.658333	22478.860833
4	34500.0	32112.658333	22478.860833
..	...	...	...
103	4107.0	4306.797000	3014.757900
104	2000.0	2000.000000	1400.000000
105	2000.0	2017.941667	1412.559167
106	2000.0	2000.000000	1400.000000
107	2000.0	2000.000000	1400.000000

[100 rows x 3 columns]

9. Resultados: Evaluación

	Precio	Precio_de_segunda_mano_estimado	Precio_de_segunda_mano_ajustado
0	46135.0	50654.916667	35458.441667
1	40650.0	41432.300000	29002.610000
2	36350.0	38295.300000	26806.710000
3	29450.0	32112.658333	22478.860833
4	34500.0	32112.658333	22478.860833
..	...	...	...
103	4107.0	4306.797000	3014.757900
104	2000.0	2000.000000	1400.000000
105	2000.0	2017.941667	1412.559167
106	2000.0	2000.000000	1400.000000
107	2000.0	2000.000000	1400.000000

[100 rows x 3 columns]

Justificación para usar RandomForest() en lugar de regresión lineal:

- RandomForest es un método de aprendizaje automático que se basa en un conjunto de árboles de decisión y es conocido por su capacidad para manejar conjuntos de datos complejos y no lineales. Puede capturar relaciones no lineales entre variables y es menos propenso al sobreajuste que la regresión lineal, lo que lo hace más robusto en diferentes conjuntos de datos.
- Dado que el R2 de RandomForest es más alto y el MSE es más bajo en comparación con la regresión lineal, es probable que RandomForest produzca predicciones más precisas y consistentes para el precio de segunda mano de los automóviles en este caso específico.

Esto sugiere que RandomForest() se ajusta mejor a los datos y es más efectivo para predecir el precio de segunda mano en comparación con los modelos de regresión lineal.

Los resultados individuales obtenidos para el precio de segunda mano para cada vehículo se pueden observar en el archivo .ipynb entregado en conjunto con este PPT.