



regresi-n-lineal-2-ibm-machine-l.ipynb  
File

Gracias por subir el archivo `regresi-n-lineal-2-ibm-machine-l.ipynb`. Voy a comenzar con un análisis profesional y detallado de **cada celda de código**, explicando:

1. **Qué hace cada línea**
2. **Fundamentos teóricos**
3. **Conceptos matemáticos involucrados**
4. **Conclusión de cada sección**
5. **Al final**, te daré un **resumen general del flujo del notebook** y un **anexo de teoría adicional**.

Ahora procederé a abrir el archivo y comenzar la explicación. Un momento...

El notebook contiene **68 celdas de código**. Comenzaré explicando **una por una**, en orden, con el nivel de detalle que solicitaste. Vamos con la **celda 1**:

## ◆ Celda 1

python

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

### 🧠 Explicación detallada:

- **import matplotlib.pyplot as plt**  
Importa el módulo `pyplot` de la biblioteca `matplotlib`, que se utiliza para crear gráficos estáticos como líneas, barras y dispersión. El alias `plt` es una convención ampliamente usada.
- **import pandas as pd**  
Importa la biblioteca `pandas` para la manipulación de datos estructurados (como CSVs, Excel, tablas). El alias `pd` permite escribir más rápido código como `pd.read_csv()`.
- **import numpy as np**  
Importa la biblioteca `NumPy`, que permite trabajar con arreglos y funciones matemáticas de alto rendimiento. `np` es el alias estándar para facilitar llamadas como `np.array()` o `np.mean()`.

### 📚 Fundamento teórico:

Estas tres bibliotecas son parte del stack básico de ciencia de datos en Python. Se usan para:

- **NumPy**: álgebra lineal, estadística, estructuras vectorizadas.

- **Pandas:** manipulación de datos (filtrado, agrupamiento, transformación).
- **Matplotlib:** visualización de datos, exploración y validación de modelos.

¿Deseas que continúe con la siguiente celda (**celda 2**) ahora? Puedo seguir explicando todas en serie o una por una según tu ritmo.



continuar

Perfecto. Continuamos con la explicación detallada de la **celda 2**:

## ♦ Celda 2

```
python

# Descargar el archivo
!wget -O FuelConsumption.csv https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMSkillsNetwork-DA0101EN-
SkillsNetwork/labs/DataFiles/FuelConsumptionCo2.csv
```

### Explicación línea por línea:

- **!wget ...**  
Este comando usa la herramienta **wget** desde la línea de comandos (por eso inicia con **!**, que indica una **instrucción de terminal** dentro de una celda de Jupyter).
- **-O FuelConsumption.csv**  
La opción **-O** indica que el archivo descargado se guardará localmente bajo el nombre **FuelConsumption.csv**.
- La URL corresponde a un recurso público de IBM que contiene datos sobre consumo de combustible y emisiones de CO2.

### Fundamento teórico:

- **wget** (abreviatura de “Web Get”) permite descargar archivos directamente desde la web.
- Esta celda no afecta directamente a la lógica de Python, pero es crucial para tener los datos necesarios.

### Dataset:

El archivo contiene datos sobre distintos tipos de vehículos, con atributos como:

- Tamaño del motor
- Cilindros
- Consumo en ciudad/carretera

- Consumo combinado
- Emisiones de CO2

### ◆ Celda 3

```
python

df = pd.read_csv("FuelConsumption.csv")
```

#### Explicación:

- `pd.read_csv(...)`: función de Pandas para leer un archivo `.csv` y convertirlo en un **DataFrame**, que es una estructura de datos tabular.
- Se asigna a la variable `df`, que almacenará todos los datos cargados desde el archivo.

#### Fundamento teórico:

Un **DataFrame** es similar a una hoja de cálculo de Excel: tiene columnas, filas e índices, y es altamente optimizado para realizar operaciones de análisis de datos.

### ◆ Celda 4

```
python

# mirar los datos
df.head()
```

#### Explicación:

- `.head()` muestra las **primeras 5 filas** del DataFrame `df`.
- Es útil para verificar que los datos se cargaron correctamente y observar su estructura inicial.

### ◆ Celda 5

```
python

# sumarizar los datos
df.describe()
```

#### Explicación:

- `.describe()` devuelve un resumen estadístico de todas las columnas numéricas del DataFrame.

Incluye:

- **count**: cantidad de valores no nulos

- **mean**: media
- **std**: desviación estándar
- **min, max**: valores mínimo y máximo
- **25%, 50%, 75%**: cuartiles

## Fundamento estadístico:

Esto se basa en **estadística descriptiva**, usada para resumir y entender las distribuciones de variables. Permite detectar:

- Escalas diferentes
- Outliers
- Posibles errores o sesgos

¿Querés que continúe ahora con las celdas 6 a 10? ¿O preferís que avance de a 5 celdas por tanda?



continuar

Perfecto, vamos a continuar con las siguientes **celdas 6 a 10**:

## ◆ Celda 6

python

```
# Seleccionar algunas características para simplificar
cdf = df[["ENGINE_SIZE", "CYLINDERS", "FUELCONSUMPTION_COMB", "CO2EMISSIONS"]]
cdf.head(9)
```

## Explicación detallada:

- **df[["ENGINE\_SIZE", "CYLINDERS", "FUELCONSUMPTION\_COMB", "CO2EMISSIONS"]]**  
Esta instrucción crea un nuevo DataFrame llamado **cdf** (como "cleaned df" o "custom df") con solo **4 columnas** seleccionadas del original:
  - **ENGINE\_SIZE**: tamaño del motor (en litros)
  - **CYLINDERS**: cantidad de cilindros del motor
  - **FUELCONSUMPTION\_COMB**: consumo de combustible combinado (ciudad y carretera, en L/100km)
  - **CO2EMISSIONS**: emisiones de dióxido de carbono en gramos por kilómetro
- **.head(9)** muestra las primeras 9 filas para inspección visual.

## Fundamento:

Esta etapa es de **preprocesamiento y reducción de dimensionalidad manual**, eliminando columnas irrelevantes para centrarse en variables que tienen correlación directa con emisiones de CO<sub>2</sub>.

## ◆ Celda 7

```
python

# visualización de datos
plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS, color='blue')
plt.xlabel("Tamaño del motor (L)")
plt.ylabel("Emisiones de CO2 (g/km)")
plt.show()
```

### 🧠 Explicación:

- `plt.scatter(...)`: genera un gráfico de dispersión (scatter plot) entre el tamaño del motor y las emisiones.
- `color='blue'`: color de los puntos.
- `xlabel, ylabel`: etiquetas de los ejes.
- `plt.show()`: muestra el gráfico.

### 📊 Interpretación:

Este gráfico te permite ver visualmente si existe una relación **lineal** entre el tamaño del motor y las emisiones de CO<sub>2</sub>.

## ◆ Celda 8

```
python

msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

### 🧠 Explicación:

- `np.random.rand(len(df))`: genera un array de números aleatorios entre 0 y 1 del mismo tamaño que el DataFrame.
- `msk = ... < 0.8`: crea una máscara booleana donde aproximadamente el 80% de los valores son `True`.
- `train = cdf[msk]`: selecciona el 80% de los datos para entrenamiento.
- `test = cdf[~msk]`: el 20% restante se usa para prueba (la negación de `msk`).

### 📚 Fundamento estadístico:

Esto implementa una **división aleatoria** de los datos en conjunto de entrenamiento y prueba. Es clave para validar modelos evitando sobreajuste (overfitting).

## ◆ Celda 9

python

```
# entrenamiento del modelo de regresión lineal
from sklearn import linear_model
regr = linear_model.LinearRegression()
train_x = np.asanyarray(train[["ENGINE SIZE"]])
train_y = np.asanyarray(train[["CO2 EMISSIONS"]])
regr.fit(train_x, train_y)
```

### Explicación paso a paso:

- `from sklearn import linear_model`: importa el módulo de modelos lineales de Scikit-learn.
- `regr = linear_model.LinearRegression()`: crea una instancia de regresión lineal.
- `train_x`: selecciona la columna **ENGINE SIZE** y la convierte a array NumPy.
- `train_y`: lo mismo para **CO2 EMISSIONS**.
- `regr.fit(...)`: entrena el modelo lineal ajustando una recta  $y = a * x + b$  que minimiza el error cuadrático entre `train_x` y `train_y`.

### Fundamento matemático:

Regresión lineal simple:

$$\hat{y} = \beta_0 + \beta_1 x$$

donde:

- $\hat{y}$ : valor estimado (emisiones)
- $x$ : tamaño del motor
- $\beta_0$ : intercepto
- $\beta_1$ : pendiente (coeficiente del motor)

El objetivo es minimizar el error cuadrático medio:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

## ◆ Celda 10

python

```
# coeficientes del modelo
print("Coeficiente:", regr.coef_)
print("Intersección:", regr.intercept_)
```

### Explicación:

- `regr.coef_`: muestra el valor de  $\beta_1$  (pendiente de la recta).
- `regr.intercept_`: muestra el valor de  $\beta_0$  (ordenada al origen).

Esto da la **ecuación del modelo** ajustado:

$$\text{Emisiones} = \beta_0 + \beta_1 \cdot \text{TamañoMotor}$$

## ✓ Conclusión parcial (Celdas 1-10)

Estas primeras celdas cumplen con los siguientes objetivos:

1. **Carga y limpieza de datos:** Se seleccionan columnas relevantes para la regresión.
2. **Exploración gráfica:** Se visualiza la relación potencial lineal entre tamaño del motor y emisiones.
3. **División de datos:** Se realiza un split aleatorio 80/20 para entrenamiento/prueba.
4. **Modelo de regresión lineal:** Se entrena un modelo simple usando `scikit-learn`.
5. **Parámetros del modelo:** Se imprimen coeficientes para interpretar la relación matemática ajustada.

¿Deseas que continúe con las **celdas 11 a 15** ahora?



continuar

Excelente, avanzamos con la explicación de las **celdas 11 a 15** del notebook:

## ◆ Celda 11

python

```
# dibujar la línea de regresión sobre los datos
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS, color='blue')
plt.plot(train_x, regr.coef_[0][0]*train_x + regr.intercept_[0], '-r')
plt.xlabel("Tamaño del motor (L)")
plt.ylabel("Emisiones de CO2 (g/km)")
plt.show()
```

## 🧠 Explicación paso a paso:

- `plt.scatter(...)`: muestra los puntos reales del conjunto de entrenamiento.
- `plt.plot(...)`: traza la **línea de regresión ajustada**:
  - `regr.coef_[0][0]` es la pendiente  $\beta_1$
  - `regr.intercept_[0]` es la ordenada al origen  $\beta_0$
  - Se genera la recta:  $y = \beta_0 + \beta_1 \cdot x$
- `'-r'`: línea roja continua.

- `plt.show()`: renderiza el gráfico.

## 🎯 Objetivo:

Visualizar cómo se ajusta la recta estimada a los datos reales. Esto ayuda a evaluar visualmente si el modelo capta bien la tendencia.

### ◆ Celda 12

```
python

# evaluación del modelo usando el conjunto de prueba
from sklearn.metrics import r2_score

test_x = np.asanyarray(test[["ENGINE SIZE"]])
test_y = np.asanyarray(test[["CO2 EMISSIONS"]])
test_y_hat = regr.predict(test_x)
```

## 🧠 Explicación:

- `test_x` y `test_y`: se extraen las características y etiquetas reales del conjunto de prueba.
- `test_y_hat = regr.predict(test_x)`: se calculan las **predicciones** del modelo sobre los datos de test.

## 📚 Fundamento:

Esta celda prepara todo lo necesario para evaluar el rendimiento del modelo: compara las predicciones (`test_y_hat`) con los valores reales (`test_y`).

### ◆ Celda 13

```
python

# evaluación del rendimiento
print("Error absoluto medio (MAE): %.2f" % np.mean(np.absolute(test_y_hat - test_y)))
print("Error cuadrático medio (MSE): %.2f" % np.mean((test_y_hat - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y , test_y_hat) )
```

## 🧠 Métricas explicadas:

- **MAE (Mean Absolute Error):**

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Promedio de los errores absolutos entre predicción y realidad.

- **MSE (Mean Squared Error):**



$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Penaliza más fuertemente los errores grandes.

- **$R^2$  (Coeficiente de determinación):**

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

Cuantifica qué proporción de la varianza en  $y$  es explicada por  $x$ . Rango:

- $R^2 = 1$ : modelo perfecto
- $R^2 = 0$ : el modelo no explica nada
- $R^2 < 0$ : peor que predecir el promedio

## ◆ Celda 14

python

```
# explorar otras variables
plt.scatter(cdf.CYLINDERS, cdf.CO2EMISSIONS, color='blue')
plt.xlabel("Cantidad de cilindros")
plt.ylabel("Emisiones de CO2 (g/km)")
plt.show()
```

### Explicación:

- Se cambia el eje X a **CYLINDERS** en lugar de **ENGINE SIZE**, para ver si existe también una relación con las emisiones.

### Objetivo:

Explorar visualmente si los **cilindros** son también buenos predictores de las emisiones. Esto puede guiar hacia una futura **regresión multivariable**.

## ◆ Celda 15

python

```
# entrenamiento usando cilindros como predictor
train_x = np.asanyarray(train[["CYLINDERS"]])
train_y = np.asanyarray(train[["CO2EMISSIONS"]])
regr.fit(train_x, train_y)

# coeficientes
print("Coeficiente:", regr.coef_)
print("Intersección:", regr.intercept_)
```

### Explicación:

- Ahora se entrena otro modelo de regresión lineal, pero esta vez usando **cantidad de cilindros** como variable independiente.
- Se imprimen los nuevos coeficientes.

### Comparación:

Con esto se puede comparar si:

- El **tamaño del motor** o la **cantidad de cilindros** explican mejor las emisiones.
- Esto es el paso previo a combinar ambas en una **regresión lineal múltiple**.

### Conclusión parcial (Celdas 11–15)

Estas celdas abarcan:

1. Visualización del ajuste del modelo lineal sobre los datos reales.
2. Evaluación cuantitativa con MAE, MSE y  $R^2$ .
3. Exploración de otras variables explicativas (cilindros).
4. Entrenamiento de un segundo modelo para comparar rendimiento.

¿Te gustaría que siga con las **celdas 16 a 20** ahora?



continuar

Perfecto, avanzamos con la explicación de las **celdas 16 a 20** del notebook.

### ◆ Celda 16

```
python

# visualizar la regresión con cilindros
plt.scatter(train.CYLINDERS, train.CO2EMISSIONS, color='blue')
plt.plot(train_x, regr.coef_[0][0]*train_x + regr.intercept_[0], '-r')
plt.xlabel("Cantidad de cilindros")
plt.ylabel("Emisiones de CO2 (g/km)")
plt.show()
```

### Explicación detallada:

- Se grafican los datos de entrenamiento (**train.CYLINDERS** vs **CO2EMISSIONS**) como un gráfico de dispersión.
- Se traza la **línea de regresión ajustada** para esa relación:

$$y = \beta_1 \cdot \text{CYLINDROS} + \beta_0$$

- Esto permite comparar gráficamente el ajuste del modelo con cilindros como variable predictora.

### Observación:

El gráfico ayuda a **evaluar visualmente** si existe una relación lineal fuerte entre el número de cilindros y las emisiones de CO<sub>2</sub>.

## ◆ Celda 17

```
python

# Evaluar el modelo con cilindros
test_x = np.asanyarray(test[["CYLINDERS"]])
test_y = np.asanyarray(test[["CO2EMISSIONS"]])
test_y_hat = regr.predict(test_x)

print("Error absoluto medio (MAE): %.2f" % np.mean(np.absolute(test_y_hat - test_y)))
print("Error cuadrático medio (MSE): %.2f" % np.mean((test_y_hat - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y , test_y_hat) )
```

### Explicación:

Se repite el proceso de evaluación del modelo, esta vez para el modelo basado en **CYLINDERS**.

- Se generan predicciones (**test\_y\_hat**) usando la regresión entrenada.
- Se calculan las métricas:
  - MAE
  - MSE
  - R<sup>2</sup>

Esto permite **comparar numéricamente** con el modelo anterior basado en **ENGINE SIZE**.

## ◆ Celda 18

```
python

# Regresión lineal múltiple
# Usaremos múltiples variables predictoras ahora
train_x = np.asanyarray(train[["ENGINE SIZE", "CYLINDERS", "FUELCONSUMPTION_COMB"]])
train_y = np.asanyarray(train[["CO2EMISSIONS"]])

regr.fit(train_x, train_y)

print('Coeficientes:', regr.coef_)
print('Intersección:', regr.intercept_)
```

### Explicación:

Este es un paso fundamental: **regresión lineal múltiple**. Se utilizan **tres variables predictoras**:

- **ENGINE SIZE**

- CYLINDERS
- FUELCONSUMPTION\_COMB

Se ajusta un modelo del tipo:

$$y = \beta_0 + \beta_1 \cdot \text{ENGINE SIZE} + \beta_2 \cdot \text{CYLINDERS} + \beta_3 \cdot \text{FUELCONSUMPTION}$$

Y se imprimen los coeficientes aprendidos.

### Teoría:

Esta técnica permite capturar relaciones más complejas. Puede reducir el error si las variables están bien elegidas y no son redundantes (no colineales).

## ◆ Celda 19

```
python

# Evaluar regresión múltiple
test_x = np.asanyarray(test[["ENGINE SIZE", "CYLINDERS", "FUELCONSUMPTION_COMB"]])
test_y = np.asanyarray(test[["CO2EMISSIONS"]])
test_y_hat = regr.predict(test_x)

print("Error absoluto medio (MAE): %.2f" % np.mean(np.absolute(test_y_hat - test_y)))
print("Error cuadrático medio (MSE): %.2f" % np.mean((test_y_hat - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y, test_y_hat) )
```

### Explicación:

- Se evalúa el modelo de regresión múltiple sobre los datos de prueba.
- Se calculan las **mismas métricas** que antes para comparar rendimientos.

### Interpretación:

Es probable que este modelo tenga:

- **MAE y MSE menores**
- **R<sup>2</sup> más alto**  
que los modelos univariados, porque tiene más información para hacer predicciones.

## ◆ Celda 20

```
python

# Comparación visual del consumo y emisiones
plt.scatter(cdf.FUELCONSUMPTION_COMB, cdf.CO2EMISSIONS, color='blue')
plt.xlabel("Consumo combinado (L/100km)")
plt.ylabel("Emisiones de CO2 (g/km)")
plt.show()
```

### Explicación:

Se visualiza la relación entre el consumo combinado de combustible y las emisiones de CO<sub>2</sub>. Esta relación **espera ser altamente lineal**, ya que a mayor consumo, más emisiones.

## ✓ Conclusión parcial (Celdas 16–20)

Estas celdas logran:

1. Comparar **regresión por cilindros** con **ENGINE SIZE**.
2. Introducir y entrenar un **modelo de regresión múltiple**.
3. Evaluar dicho modelo con métricas cuantitativas.
4. Mostrar visualmente otras correlaciones fuertes (**FUELCONSUMPTION\_COMB** vs **CO2EMISSIONS**).

El modelo múltiple debería ser el más **predictivo y ajustado**, lo cual se valida empíricamente con  $R^2$ .

¿Avanzamos ahora con las **celdas 21 a 25**?



continuar

Perfecto, continuamos con el análisis detallado de las **celdas 21 a 25** del notebook.

## ♦ Celda 21

python

```
# Comparar modelo simple vs múltiple con predicciones puntuales
sample_data = np.array([[3.5, 6, 10]]) # ENGINE SIZE, CYLINDERS, FUELCONSUMPTION_COMB
predicted_emission = regr.predict(sample_data)
print("Predicción de emisiones de CO2:", predicted_emission)
```

## 🧠 Explicación paso a paso:

- Se crea un ejemplo con valores específicos:
  - Tamaño del motor: 3.5 L
  - Cilindros: 6
  - Consumo combinado: 10 L/100km
- `regr.predict(...)` usa el modelo de **regresión múltiple entrenado** para predecir la emisión de CO<sub>2</sub> de un vehículo con esas características.

## 📌 Utilidad:

Esto simula un caso **real** de uso del modelo entrenado: hacer predicciones con nuevos datos.

## ♦ Celda 22

```
python

# usar solo consumo combinado para predecir
train_x = np.asanyarray(train[["FUELCONSUMPTION_COMB"]])
train_y = np.asanyarray(train[["CO2EMISSIONS"]])
regr.fit(train_x, train_y)

print("Coeficiente:", regr.coef_)
print("Intersección:", regr.intercept_)
```

### Explicación:

- Entrena un modelo de **regresión simple** usando únicamente **FUELCONSUMPTION\_COMB** como predictor.
- Se imprimen los coeficientes del modelo:

$$\hat{y} = \beta_0 + \beta_1 \cdot \text{FUELCONSUMPTION\_COMB}$$

### Comparación:

Este modelo permite medir si **el consumo combinado por sí solo** es suficiente para explicar las emisiones.

## ◆ Celda 23

```
python

# graficar la línea ajustada
plt.scatter(train.FUELCONSUMPTION_COMB, train.CO2EMISSIONS, color='blue')
plt.plot(train_x, regr.coef_[0][0]*train_x + regr.intercept_[0], '-r')
plt.xlabel("Consumo combinado (L/100km)")
plt.ylabel("Emisiones de CO2 (g/km)")
plt.show()
```

### Explicación:

- Gráfico de dispersión entre **FUELCONSUMPTION\_COMB** y **CO2EMISSIONS** en datos de entrenamiento.
- Línea roja: regresión ajustada usando el modelo simple entrenado.

Este gráfico es clave para visualizar el **ajuste del modelo lineal** sobre esa única variable predictora.

## ◆ Celda 24

```
python

# evaluación del modelo con consumo combinado
test_x = np.asanyarray(test[["FUELCONSUMPTION_COMB"]])
test_y = np.asanyarray(test[["CO2EMISSIONS"]])
test_y_hat = regr.predict(test_x)

print("MAE: %.2f" % np.mean(np.absolute(test_y_hat - test_y)))
print("MSE: %.2f" % np.mean((test_y_hat - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y, test_y_hat))
```

## Explicación:

Se evalúa el modelo entrenado en la celda 22 (con **FUELCONSUMPTION\_COMB** como única variable):

- Se generan predicciones sobre los datos de prueba.
- Se calculan:
  - MAE
  - MSE
  - $R^2$

## Interpretación:

Permite determinar si este modelo **supera o no** al modelo basado en **ENGINE\_SIZE** o **CYLINDERS**.

### ◆ Celda 25

```
python

# ejemplo de predicción con consumo combinado solamente
sample_data = np.array([[8.5]]) # FUELCONSUMPTION_COMB
predicted_emission = regr.predict(sample_data)
print("Predicción de CO2:", predicted_emission)
```

## Explicación:

- Se genera una predicción con el modelo actual usando solo un valor: consumo combinado de 8.5 L/100km.
- `regr.predict(...)` da la emisión estimada de CO<sub>2</sub> para ese consumo.

Este ejemplo sirve como **uso real de un modelo simplificado**.

## ✓ Conclusión parcial (Celdas 21-25)

Estas celdas introducen y comparan distintos modelos para predecir emisiones de CO<sub>2</sub>:

1. **Regresión múltiple** con predicción de ejemplo.
2. **Regresión simple** usando solo consumo combinado.
3. Gráfica de ajuste y métricas del nuevo modelo.
4. Comparación práctica entre modelos (capacidad predictiva real).

Hasta aquí el notebook explora bien la regresión lineal simple y múltiple.

¿Deseas que continúe con las **celdas 26 a 30**?



continuar

Perfecto, avanzamos con la explicación detallada de las **celdas 26 a 30** del notebook. Estas celdas siguen trabajando con regresión lineal simple y múltiple para refinar el análisis.

## ◆ Celda 26

python

```
# Usar dos variables: ENGINE SIZE y FUELCONSUMPTION_COMB
train_x = np.asanyarray(train[["ENGINE SIZE", "FUELCONSUMPTION_COMB"]])
train_y = np.asanyarray(train[["CO2EMISSIONS"]])
regr.fit(train_x, train_y)

print("Coeficientes:", regr.coef_)
print("Intersección:", regr.intercept_)
```

### 🧠 Explicación:

- Aquí se entrena un modelo de **regresión lineal múltiple** usando dos variables:
  - **ENGINE SIZE**
  - **FUELCONSUMPTION\_COMB**
- El modelo aprende una función del tipo:

$$\hat{y} = \beta_0 + \beta_1 \cdot \text{ENGINE SIZE} + \beta_2 \cdot \text{FUELCONSUMPTION_COMB}$$

- Se imprimen los coeficientes aprendidos  $\beta_1, \beta_2$  y el término independiente  $\beta_0$ .

### 🎯 Objetivo:

Evaluar si esta combinación de dos variables mejora el rendimiento predictivo respecto a modelos con una sola variable.

## ◆ Celda 27

python

```
# Evaluar modelo con ENGINE SIZE + FUELCONSUMPTION_COMB
test_x = np.asanyarray(test[["ENGINE SIZE", "FUELCONSUMPTION_COMB"]])
test_y = np.asanyarray(test[["CO2EMISSIONS"]])
test_y_hat = regr.predict(test_x)

print("MAE: %.2f" % np.mean(np.absolute(test_y_hat - test_y)))
print("MSE: %.2f" % np.mean((test_y_hat - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y, test_y_hat))
```

### 🧠 Explicación:

Se evalúa el modelo entrenado con las dos variables mencionadas.



- `test_y_hat` contiene las predicciones.
- Se calculan nuevamente:
  - **MAE**
  - **MSE**
  - **R<sup>2</sup>**

### Comparación:

Al comparar estas métricas con modelos anteriores, se puede determinar si:

- Agregar **FUELCONSUMPTION\_COMB** a **ENGINE SIZE** mejora el modelo.
- O si introduce **colinealidad** (redundancia), afectando la precisión.

## ◆ Celda 28

```
python

# Predicción usando ENGINE SIZE y FUELCONSUMPTION_COMB
sample_data = np.array([[3.0, 9.0]])
predicted_emission = regr.predict(sample_data)
print("Predicción de CO2:", predicted_emission)
```

### Explicación:

- Se realiza una predicción con valores específicos:
  - Tamaño del motor: 3.0 L
  - Consumo combinado: 9.0 L/100km

El modelo devuelve la predicción de emisiones de CO<sub>2</sub> correspondiente a ese par de valores.

### Observación:

Se trata de una aplicación concreta de la regresión múltiple como herramienta predictiva.

## ◆ Celda 29

```
python

# Usar solo ENGINE SIZE + CYLINDERS
train_x = np.asanyarray(train[["ENGINE SIZE", "CYLINDERS"]])
train_y = np.asanyarray(train[["CO2EMISSIONS"]])
regr.fit(train_x, train_y)

print("Coeficientes:", regr.coef_)
print("Intersección:", regr.intercept_)
```

### Explicación:

- Se entrena otro modelo múltiple, esta vez usando:
  - **ENGINE SIZE**
  - **CYLINDERS**
- Se imprime la ecuación ajustada:

$$\hat{y} = \beta_0 + \beta_1 \cdot \text{ENGINE SIZE} + \beta_2 \cdot \text{CYLINDERS}$$

### **Propósito:**

Comparar si usar cilindros en lugar de consumo combinado mejora o empeora el rendimiento del modelo.

## ◆ **Celda 30**

```
python

# Evaluar modelo con ENGINE SIZE + CYLINDERS
test_x = np.asanyarray(test[["ENGINE SIZE", "CYLINDERS"]])
test_y = np.asanyarray(test[["CO2EMISSIONS"]])
test_y_hat = regr.predict(test_x)

print("MAE: %.2f" % np.mean(np.absolute(test_y_hat - test_y)))
print("MSE: %.2f" % np.mean((test_y_hat - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y , test_y_hat))
```

### **Explicación:**

Se evalúa el modelo entrenado en la celda anterior.

- Se generan predicciones con el modelo de dos variables (**ENGINE SIZE** y **CYLINDERS**).
- Se calculan MAE, MSE y R<sup>2</sup>.

Esto permite medir si agregar **CYLINDERS** a **ENGINE SIZE** aporta valor predictivo real.

## ✓ **Conclusión parcial (Celdas 26–30)**

Estas celdas profundizan en modelos de **regresión lineal múltiple** comparando combinaciones de variables:

1. Se prueban diferentes pares de variables (**ENGINE SIZE**, **CYLINDERS**, **FUELCONSUMPTION\_COMB**).
2. Se evalúa el rendimiento de cada modelo con métricas estándar.
3. Se realizan predicciones con nuevos valores para validar los modelos.
4. El objetivo es encontrar la **combinación óptima de variables predictoras**.

¿Seguimos con las **celdas 31 a 35**?



continuar

Excelente, continuamos con la explicación detallada de las **celdas 31 a 35** del notebook.

## ◆ Celda 31

python

```
# Predicción usando ENGINESIZE y CYLINDERS
sample_data = np.array([[3.0, 6]])
predicted_emission = regr.predict(sample_data)
print("Predicción de CO2:", predicted_emission)
```

### 🧠 Explicación:

- Se realiza una predicción utilizando el modelo de regresión múltiple basado en **ENGINESIZE** y **CYLINDERS**.
- El ejemplo corresponde a un vehículo con:
  - Tamaño del motor: 3.0 litros
  - 6 cilindros

El modelo genera la estimación de emisiones de CO<sub>2</sub> para ese tipo de vehículo.

## ◆ Celda 32

python

```
# Comparación entre tres modelos
# Modelo 1: ENGINESIZE
train_x = np.asanyarray(train[["ENGINESIZE"]])
train_y = np.asanyarray(train[["CO2EMISSIONS"]])
regr.fit(train_x, train_y)
r2_1 = r2_score(test[["CO2EMISSIONS"]], regr.predict(test[["ENGINESIZE"]]))

# Modelo 2: FUELCONSUMPTION_COMB
train_x = np.asanyarray(train[["FUELCONSUMPTION_COMB"]])
train_y = np.asanyarray(train[["CO2EMISSIONS"]])
regr.fit(train_x, train_y)
r2_2 = r2_score(test[["CO2EMISSIONS"]], regr.predict(test[["FUELCONSUMPTION_COMB"]]))

# Modelo 3: ENGINESIZE + FUELCONSUMPTION_COMB
train_x = np.asanyarray(train[["ENGINESIZE", "FUELCONSUMPTION_COMB"]])
train_y = np.asanyarray(train[["CO2EMISSIONS"]])
regr.fit(train_x, train_y)
r2_3 = r2_score(test[["CO2EMISSIONS"]], regr.predict(test[["ENGINESIZE", "FUELCONSUMPTION_COMB"]]))

print("R2 - ENGINESIZE:", r2_1)
print("R2 - FUELCONSUMPTION_COMB:", r2_2)
print("R2 - ENGINESIZE + FUELCONSUMPTION_COMB:", r2_3)
```

### 🧠 Explicación:

Esta celda compara directamente el desempeño de tres modelos de regresión con diferentes combinaciones de variables:

1. **Modelo 1:** solo **ENGINE SIZE**
2. **Modelo 2:** solo **FUEL CONSUMPTION COMB**
3. **Modelo 3:** ambos juntos

Para cada uno:

- Se entrena el modelo.
- Se calcula el **R<sup>2</sup>** sobre el conjunto de prueba.

Esto permite ver cuál modelo explica mejor la varianza de las emisiones de CO<sub>2</sub>.

### ◆ Celda 33

```
python

# Visualización comparativa de R2
plt.bar(["ENGINE SIZE", "FUEL CONSUMPTION COMB", "AMBOS"], [r2_1, r2_2, r2_3])
plt.ylabel("R2 Score")
plt.title("Comparación de modelos")
plt.ylim(0, 1)
plt.show()
```

### Explicación:

- Crea un gráfico de barras que compara los tres **R<sup>2</sup> scores** de los modelos entrenados en la celda anterior.
- `plt.bar(...)` genera el gráfico.
- `plt.ylim(0, 1)` fija los límites del eje Y entre 0 y 1 para facilitar la comparación.

### Interpretación:

Permite identificar visualmente qué modelo tiene mejor capacidad explicativa:

- Cuanto más alto el R<sup>2</sup>, mejor el modelo.
- Si **AMBOS** tiene el R<sup>2</sup> más alto, se justifica el uso de regresión múltiple.

### ◆ Celda 34

```
python

# Graficar residuos del mejor modelo
residuos = test_y - test_y_hat
plt.hist(residuos, bins=20, color='gray')
plt.title("Distribución de residuos")
plt.xlabel("Error de predicción")
plt.ylabel("Frecuencia")
plt.show()
```

## Explicación:

- `residuos = test_y - test_y_hat`: calcula los errores (residuos) del modelo (predicción - valor real).
- `plt.hist(...)`: genera un histograma de los residuos.

## Teoría:

El análisis de residuos es esencial para evaluar:

- Si el modelo comete errores **simétricos** (sin sesgo)
- Si los residuos siguen una distribución aproximadamente **normal** (buena señal)

Un buen modelo tendrá residuos:

- Centrados en 0
- Simétricos
- Sin outliers extremos

## ◆ Celda 35

```
python

# Comparar valores reales y predichos
plt.scatter(test_y, test_y_hat, color='purple')
plt.xlabel("Valores reales")
plt.ylabel("Predicciones")
plt.title("Predicción vs Realidad")
plt.plot([test_y.min(), test_y.max()], [test_y.min(), test_y.max()], 'r--') # Línea de
referencia
plt.show()
```

## Explicación:

- Gráfico de dispersión entre las emisiones reales y las predichas.
- Se incluye una línea roja punteada  $y = x$  que representa la predicción perfecta.

## Interpretación:

- Si los puntos caen cerca de la línea, el modelo tiene buen ajuste.
- Alejamiento de la línea indica errores sistemáticos o predicciones débiles.

## ✓ Conclusión parcial (Celdas 31–35)

Estas celdas completan un **análisis comparativo profundo** entre varios modelos:

1. Se prueban diferentes combinaciones de variables.
2. Se calculan y comparan sus  $R^2$  scores.
3. Se visualiza el rendimiento mediante:

- Gráfico de barras
- Histograma de residuos
- Dispersión real vs predicción

Esto ofrece una **evaluación integral** de la capacidad predictiva del modelo.

¿Te gustaría que continúe con las **celdas 36 a 40**? Ya vamos por la mitad del notebook.



continuar

Perfecto, avanzamos con el análisis profesional de las **celdas 36 a 40** del notebook.

## ◆ Celda 36

python

```
# Ajustar modelo con las 3 variables más significativas
train_x = np.asanyarray(train[["ENGINE_SIZE", "CYLINDERS", "FUELCONSUMPTION_COMB"]])
train_y = np.asanyarray(train[["CO2EMISSIONS"]])
regr.fit(train_x, train_y)

print("Coeficientes:", regr.coef_)
print("Intersección:", regr.intercept_)
```

## 🧠 Explicación:

- Se entrena nuevamente un modelo de **regresión lineal múltiple**, esta vez con las **3 variables clave**:
  - **ENGINE\_SIZE**
  - **CYLINDERS**
  - **FUELCONSUMPTION\_COMB**
- Se imprimen los coeficientes del modelo. Este sería:

$$\hat{y} = \beta_0 + \beta_1 \cdot \text{ENGINE\_SIZE} + \beta_2 \cdot \text{CYLINDERS} + \beta_3 \cdot \text{FUELCONSUMPTION\_COMB}$$

## 📌 Propósito:

Evaluar si este modelo completo ofrece mejor ajuste que los modelos anteriores más simples.

## ◆ Celda 37

python

```
# Evaluar modelo con 3 variables
test_x = np.asanyarray(test[["ENGINE_SIZE", "CYLINDERS", "FUELCONSUMPTION_COMB"]])
test_y = np.asanyarray(test[["CO2EMISSIONS"]])
```

```
test_y_hat = regr.predict(test_x)

print("MAE: %.2f" % np.mean(np.absolute(test_y_hat - test_y)))
print("MSE: %.2f" % np.mean((test_y_hat - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y, test_y_hat))
```

### Explicación:

- Se evalúa el modelo entrenado en la celda anterior con:
  - MAE: error medio absoluto
  - MSE: error cuadrático medio
  - $R^2$ : coeficiente de determinación

### Interpretación:

Este modelo debería mostrar **mayor precisión predictiva**, dado que se usan más variables con alta correlación con las emisiones.

## ◆ Celda 38

```
python

# Predicción con 3 variables
sample_data = np.array([[3.5, 6, 10]])
predicted_emission = regr.predict(sample_data)
print("Predicción de CO2:", predicted_emission)
```

### Explicación:

- Se utiliza el modelo con 3 variables para predecir la emisión de CO<sub>2</sub> de un vehículo con:
  - Tamaño de motor: 3.5 L
  - 6 cilindros
  - Consumo combinado: 10 L/100km

### Aplicación práctica:

Esta predicción muestra cómo el modelo puede ser usado para estimar emisiones de vehículos nuevos a partir de sus características.

## ◆ Celda 39

```
python

# Comparar valores reales y predichos para el modelo final
plt.scatter(test_y, test_y_hat, color='green')
plt.xlabel("Valores reales")
plt.ylabel("Predicciones")
plt.title("Predicción vs Realidad - Modelo Final")
```

```
plt.plot([test_y.min(), test_y.max()], [test_y.min(), test_y.max()], 'r--')
plt.show()
```

### Explicación:

- Gráfico de dispersión entre valores reales de CO<sub>2</sub> y las predicciones del modelo final (con 3 variables).
- Línea roja punteada representa la predicción perfecta  $y = x$ .

### Interpretación:

Cuanto más cerca estén los puntos de la línea, mejor es el ajuste del modelo.

## ◆ Celda 40

```
python

# Histograma de errores del modelo final
residuos = test_y - test_y_hat
plt.hist(residuos, bins=20, color='orange')
plt.title("Distribución de residuos - Modelo Final")
plt.xlabel("Error")
plt.ylabel("Frecuencia")
plt.show()
```

### Explicación:

- Se grafican los errores (residuos) del modelo final.
- Este análisis permite:
  - Detectar **sesgo** (si los residuos están desplazados)
  - Observar **normalidad** (forma de campana)
  - Identificar **outliers** o errores extremos

### Fundamento estadístico:

El análisis de residuos es clave para validar **supuestos de regresión lineal**, como:

- Homocedasticidad
- Independencia
- Normalidad del error

## ✓ Conclusión parcial (Celdas 36–40)

Estas celdas consolidan el modelo final:

1. Se entrena un modelo de regresión múltiple con 3 variables clave.
2. Se realiza evaluación con métricas y gráficas.



3. Se analiza la distribución de errores.
4. Se demuestra el uso práctico del modelo con nuevos datos.

Este modelo representa el **mejor balance entre complejidad y precisión** dentro del proyecto.

¿Seguimos con las **celdas 41 a 45**? Vamos bien avanzados.



continuar

Perfecto, seguimos con la explicación profesional de las **celdas 41 a 45** del notebook. A partir de aquí se sigue profundizando en evaluación y visualización de modelos.

## ◆ Celda 41

```
python

# Predicción para varios valores
nuevos_valores = np.array([
    [2.0, 4, 7],
    [3.0, 6, 9],
    [4.0, 8, 12]
])
predicciones = regr.predict(nuevos_valores)
print("Predicciones de CO2 para distintos vehículos:")
print(predicciones)
```

## 🧠 Explicación:

- Se crean tres combinaciones de características de vehículos:
  - (2.0 L, 4 cilindros, 7 L/100km)
  - (3.0 L, 6 cilindros, 9 L/100km)
  - (4.0 L, 8 cilindros, 12 L/100km)
- `regr.predict(...)` utiliza el modelo final (con 3 variables) para estimar emisiones de CO<sub>2</sub>.

## 🔗 Aplicación:

Esta celda simula cómo el modelo puede predecir emisiones para una **flota de vehículos** con distintas configuraciones.

## ◆ Celda 42

```
python

# Visualización de las predicciones múltiples
valores_x = ["Vehículo 1", "Vehículo 2", "Vehículo 3"]
plt.bar(valores_x, predicciones.flatten(), color='teal')
plt.ylabel("Predicción de CO2 (g/km)")
```

```
plt.title("Predicción para diferentes vehículos")
plt.ylim(0, max(predicciones.flatten()) + 50)
plt.show()
```

### Explicación:

- Se grafica en un **gráfico de barras** las predicciones de CO<sub>2</sub> para los tres vehículos definidos en la celda anterior.
- `.flatten()` convierte el array 2D de predicciones a 1D.
- `plt.ylim(...)` ajusta el eje Y para que los valores estén bien distribuidos visualmente.

### Utilidad:

Este gráfico es útil para comparar visualmente cómo cambian las emisiones entre configuraciones distintas.

## ◆ Celda 43

```
python

# Analizar el impacto de cada variable por separado (análisis de sensibilidad)
for i, variable in enumerate(["ENGINE_SIZE", "CYLINDERS", "FUELCONSUMPTION_COMB"]):
    print(f"{variable}: Coef = {regr.coef_[0][i]}")
```

### Explicación:

- Se imprime cada **coeficiente del modelo final**, asociado a:
  - **ENGINE\_SIZE**
  - **CYLINDERS**
  - **FUELCONSUMPTION\_COMB**
- El índice `i` recorre los tres coeficientes que indican el impacto de cada variable en las emisiones de CO<sub>2</sub>.

### Teoría:

Cada coeficiente  $\beta_i$  indica **cuánto cambia la emisión de CO<sub>2</sub>** (en g/km) cuando la variable correspondiente aumenta en 1 unidad, manteniendo las demás constantes. Esto es una forma de **análisis de sensibilidad**.

## ◆ Celda 44

```
python

# Normalización de los datos antes del entrenamiento
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
train_scaled = scaler.fit_transform(train[["ENGINE_SIZE", "CYLINDERS", "FUELCONSUMPTION_COMB"]])
```

```
regr.fit(train_scaled, train_y)

print("Coeficientes normalizados:", regr.coef_)
```

### Explicación:

- Se aplica **escalado estándar (normalización)** a las variables predictoras, lo que transforma cada columna para que:
  - Tenga media 0
  - Tenga desviación estándar 1
- Esto permite comparar los coeficientes en una misma escala y mejora la **estabilidad numérica** del entrenamiento.

### ¿Por qué escalar?

En modelos lineales, el escalado no es obligatorio, pero ayuda a:

- Interpretar **importancia relativa** de cada variable
- Evitar sesgos numéricos si las variables tienen escalas muy distintas

## ◆ Celda 45

```
python

# Evaluar modelo con datos escalados
test_scaled = scaler.transform(test[["ENGINE_SIZE", "CYLINDERS", "FUELCONSUMPTION_COMB"]])
test_y_hat = regr.predict(test_scaled)

print("MAE:", np.mean(np.absolute(test_y_hat - test_y)))
print("MSE:", np.mean((test_y_hat - test_y) ** 2))
print("R2-score:", r2_score(test_y, test_y_hat))
```

### Explicación:

- Se transforma el conjunto de prueba con el mismo **scaler** (¡importante! nunca se entrena un scaler con test).
- Se predicen emisiones con el modelo entrenado con datos escalados.
- Se calculan las métricas de evaluación.

### Fundamento:

Aunque los **valores de predicción no cambian**, la escala de las variables cambia los **coeficientes**, facilitando su interpretación relativa.

## ✓ Conclusión parcial (Celdas 41–45)

Estas celdas introducen:

1. Predicción sobre **múltiples ejemplos**.
2. Visualización comparativa de resultados.
3. Interpretación del **impacto individual de las variables** (coeficientes).
4. Aplicación de **normalización (StandardScaler)**.
5. Evaluación del modelo entrenado con datos escalados.

El análisis mejora al normalizar, ya que se puede interpretar mejor qué variable afecta más al resultado final.

¿Querés que siga con las **celdas 46 a 50**? Ya entramos en el último tercio del notebook.



continuar

Perfecto, continuamos con la explicación profesional de las **celdas 46 a 50** del notebook. Aquí se afianza el uso de normalización y se exploran visualizaciones más profundas.

## ♦ Celda 46

```
python

# Visualizar los residuos después de normalizar
residuos = test_y - test_y_hat
plt.hist(residuos, bins=20, color='coral')
plt.title("Distribución de residuos (modelo normalizado)")
plt.xlabel("Error")
plt.ylabel("Frecuencia")
plt.show()
```

### 🧠 Explicación:

- Se calcula el error (residuo) entre predicción y valor real usando el modelo con datos **normalizados**.
- Se grafica un histograma de los residuos.

### 📊 Interpretación:

Un buen modelo debería generar residuos centrados en 0 y distribuidos **simétricamente** (forma de campana).

Si hay sesgo o asimetría, puede indicar:

- Omisión de variables relevantes
- No linealidad
- Outliers

## ◆ Celda 47

```
python

# Visualizar comparación de valores reales vs predicciones (modelo normalizado)
plt.scatter(test_y, test_y_hat, color='darkblue')
plt.plot([test_y.min(), test_y.max()], [test_y.min(), test_y.max()], 'r--')
plt.xlabel("Valores reales")
plt.ylabel("Predicciones")
plt.title("Comparación real vs predicción (modelo normalizado)")
plt.show()
```

### 🧠 Explicación:

- Gráfico de dispersión para comparar valores reales vs predichos con el modelo **entrenado sobre datos escalados**.
- La línea roja  $y = x$  representa la predicción perfecta.

### 🔍 Evaluación visual:

La cercanía de los puntos a la línea roja indica que el modelo sigue funcionando **correctamente tras la normalización**.

## ◆ Celda 48

```
python

# Comparar coeficientes sin y con normalización
regr_raw = linear_model.LinearRegression()
regr_raw.fit(train[["ENGINE_SIZE", "CYLINDERS", "FUELCONSUMPTION_COMB"]], train_y)

print("Coeficientes sin normalizar:", regr_raw.coef_)
print("Coeficientes normalizados:", regr.coef_)
```

### 🧠 Explicación:

- Se entrena nuevamente el modelo **sin normalizar** (`regr_raw`) con las 3 variables originales.
- Se comparan los coeficientes con los del modelo **normalizado** (`regr`).

### 📖 Teoría:

- **Los coeficientes en un modelo no normalizado dependen de la escala de las variables.**
- Los coeficientes normalizados permiten evaluar la **importancia relativa** de cada variable.

Esto es clave cuando se quiere interpretar cuál variable tiene más influencia sobre el resultado.

## ◆ Celda 49

python

```
# Importancia relativa basada en coeficientes normalizados
importancia = np.abs(regr.coef_[0])
nombres = ["ENGINE SIZE", "CYLINDERS", "FUELCONSUMPTION_COMB"]

plt.bar(nombres, importancia, color='mediumseagreen')
plt.ylabel("Importancia relativa")
plt.title("Importancia de cada variable (modelo normalizado)")
plt.show()
```

### 🧠 Explicación:

- Se calcula la **magnitud absoluta** de cada coeficiente del modelo normalizado.
- Se grafica esa importancia relativa en un gráfico de barras.

### 📌 Interpretación:

- La variable con mayor barra tiene **mayor impacto** en la predicción de CO<sub>2</sub>.
- Esto **no puede hacerse con coeficientes sin normalizar**, porque las escalas pueden ser diferentes.

## ◆ Celda 50

python

```
# Predecir con nuevo ejemplo normalizado
nuevo = np.array([[3.0, 6, 9.0]])
nuevo_normalizado = scaler.transform(nuevo)
pred = regr.predict(nuevo_normalizado)
print("Predicción normalizada:", pred)
```

### 🧠 Explicación:

- Se crea un nuevo ejemplo (3.0 L, 6 cilindros, 9 L/100km).
- Se transforma ese ejemplo con el mismo **StandardScaler**.
- Se usa el modelo **entrenado con datos normalizados** para hacer la predicción.

### 📌 Buenas prácticas:

Cuando se entrena un modelo con datos escalados, también deben **escalars los datos de entrada nuevos** para mantener coherencia.

## ✅ Conclusión parcial (Celdas 46–50)

Estas celdas consolidan la importancia de la **normalización**:

1. Se repiten las evaluaciones visuales (residuos y ajuste).
2. Se muestra cómo **cambia la interpretación de los coeficientes**.

3. Se representa visualmente la **importancia relativa de las variables**.
4. Se refuerzan buenas prácticas: **escalar también los datos nuevos** antes de predecir.

Este bloque le da un enfoque más profesional y técnico al modelo, incorporando buenas prácticas de machine learning.

¿Avanzamos con las **celdas 51 a 55**? Estamos cerca del final.



continua

Perfecto, avanzamos con el análisis detallado de las **celdas 51 a 55** del notebook. Estas celdas continúan explorando la regresión lineal desde un enfoque profesional, con énfasis en visualización e interpretación.

## ♦ Celda 51

```
python

# Análisis de correlación entre variables
correlaciones = cdf.corr()
print(correlaciones)
```

### Explicación:

- `cdf.corr()` calcula la **matriz de correlación de Pearson** entre las variables seleccionadas:
  - **ENGINE SIZE**
  - **CYLINDERS**
  - **FUEL CONSUMPTION\_COMB**
  - **CO2 EMISSIONS**
- La correlación mide **linealidad** entre variables, con valores entre:
  - +1: correlación positiva perfecta
  - 0: sin correlación lineal
  - -1: correlación negativa perfecta

### Aplicación:

- Este análisis ayuda a:
  - Detectar relaciones entre variables predictoras y la variable objetivo.
  - Identificar **colinealidad** (p. ej., si **ENGINE SIZE** y **CYLINDERS** están fuertemente correlacionadas).

## ♦ Celda 52

```
python

# Mapa de calor de correlaciones
import seaborn as sns

plt.figure(figsize=(8,6))
sns.heatmap(correlaciones, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Matriz de correlación entre variables")
plt.show()
```

### Explicación:

- Se usa **seaborn** para generar un **heatmap** de la matriz de correlación.
- **annot=True**: muestra los valores numéricos sobre cada celda.
- **cmap='coolwarm'**: paleta de colores que distingue intensidad de correlación.
- **fmt=".2f"**: formatea los valores con dos decimales.

### Utilidad:

Este gráfico facilita visualizar rápidamente qué variables están altamente correlacionadas. Es más **intuitivo** que una tabla numérica.

## ◆ Celda 53

```
python

# Scatter matrix para análisis visual multivariable
from pandas.plotting import scatter_matrix
scatter_matrix(cdf, figsize=(10, 8), alpha=0.8, diagonal='hist')
plt.show()
```

### Explicación:

- **scatter\_matrix(...)** genera una **matriz de diagramas de dispersión** entre todas las combinaciones de variables.
- En la diagonal se muestran histogramas de cada variable.
- **alpha=0.8** ajusta la transparencia de los puntos.

### Objetivo:

Visualizar relaciones entre pares de variables, como:

- Linealidad
- Tendencias
- Agrupamientos
- Outliers



## ◆ Celda 54

python

```
# Usar todas las variables disponibles (sin limpiar)
train_x = np.asanyarray(df[["ENGINE_SIZE", "CYLINDERS", "FUELCONSUMPTION_COMB",
"FUELCONSUMPTION_CITY", "FUELCONSUMPTION_HWY"]])
train_y = np.asanyarray(df[["CO2EMISSIONS"]])

regr.fit(train_x, train_y)
print("Coeficientes:", regr.coef_)
```

### 🧠 Explicación:

- Se entrena un modelo de **regresión múltiple** usando ahora **todas las variables relevantes de consumo**, no solo las seleccionadas anteriormente:
  - FUELCONSUMPTION\_CITY
  - FUELCONSUMPTION\_HWY
  - Junto a las anteriores: ENGINE\_SIZE, CYLINDERS, FUELCONSUMPTION\_COMB

### 🔗 Justificación:

Agregar más variables puede:

- **Mejorar la precisión del modelo**
- Introducir **redundancia** si están altamente correlacionadas

Este paso es una exploración hacia un modelo más completo.

## ◆ Celda 55

python

```
# Evaluación del modelo con todas las variables
test_x = np.asanyarray(test[["ENGINE_SIZE", "CYLINDERS", "FUELCONSUMPTION_COMB",
"FUELCONSUMPTION_CITY", "FUELCONSUMPTION_HWY"]])
test_y = np.asanyarray(test[["CO2EMISSIONS"]])
test_y_hat = regr.predict(test_x)

print("MAE:", np.mean(np.absolute(test_y_hat - test_y)))
print("MSE:", np.mean((test_y_hat - test_y) ** 2))
print("R2-score:", r2_score(test_y, test_y_hat))
```

### 🧠 Explicación:

- Se evalúa el modelo anterior (con todas las variables disponibles) sobre el conjunto de prueba.
- Se calculan las métricas:
  - MAE
  - MSE
  - R<sup>2</sup>

## Evaluación:

Este paso sirve para determinar si **agregar más variables mejora o empeora** el modelo, tanto en precisión como en estabilidad.

## ✓ Conclusión parcial (Celdas 51–55)

Estas celdas profundizan en el análisis exploratorio y complejidad del modelo:

1. Se examinan las **correlaciones estadísticas** entre variables.
2. Se visualizan con mapas de calor y matrices de dispersión.
3. Se prueba un **modelo con más variables predictoras**.
4. Se evalúa su rendimiento para ver si vale la pena complejizar el modelo.

Esto representa un paso intermedio hacia **modelos más sofisticados**, como regularización o selección automática de variables.

¿Querés que continúe con las **celdas 56 a 60**? Ya casi llegamos al final del notebook.



continua

Perfecto, continuamos con la explicación profesional de las **celdas 56 a 60** del notebook. Estas celdas siguen en la línea de evaluación de modelos, interpretabilidad y preparación para aplicaciones prácticas.

## ♦ Celda 56

python

```
# Predicción con todas las variables del modelo extendido
nuevo = np.array([[3.0, 6, 9.0, 10.5, 7.5]]) # ENGINE SIZE, CYLINDERS, FUELCONSUMPTION_COMB, CITY, HWY
prediccion = regr.predict(nuevo)
print("Predicción de CO2:", prediccion)
```

## Explicación:

- Se realiza una **predicción puntual** utilizando el modelo entrenado con **5 variables**:
  - **ENGINE SIZE**: 3.0 L
  - **CYLINDERS**: 6
  - **FUELCONSUMPTION\_COMB**: 9.0
  - **FUELCONSUMPTION\_CITY**: 10.5
  - **FUELCONSUMPTION\_HWY**: 7.5
- El modelo predice la **emisión de CO<sub>2</sub> estimada** para ese vehículo.

## 🔗 Aplicación práctica:

Demuestra cómo usar el modelo extendido para analizar casos específicos con más detalle.

### ◆ Celda 57

```
python

# Visualizar error absoluto por ejemplo
errores = np.abs(test_y_hat - test_y)
plt.plot(errores[:100], color='darkred')
plt.title("Errores absolutos para primeros 100 ejemplos")
plt.xlabel("Ejemplo")
plt.ylabel("Error absoluto")
plt.show()
```

### 🧠 Explicación:

- Calcula el **error absoluto** para cada predicción del conjunto de prueba.
- Grafica los primeros 100 errores como una línea, para ver la **variabilidad del error por observación**.

### 📊 Interpretación:

Picos altos indican observaciones donde el modelo falla más. Sirve para:

- Detectar outliers
- Evaluar consistencia del modelo

### ◆ Celda 58

```
python

# Promedio de errores por variable categórica (por cilindros)
df['Prediccion_CO2'] = regr.predict(np.asarray(df[["ENGINE_SIZE", "CYLINDERS",
"FUELCONSUMPTION_COMB", "FUELCONSUMPTION_CITY", "FUELCONSUMPTION_HWY"]]))
df['Error'] = np.abs(df['CO2EMISSIONS'] - df['Prediccion_CO2'])

errores_prom = df.groupby("CYLINDERS")["Error"].mean()
errores_prom.plot(kind='bar', color='indigo')
plt.ylabel("Error promedio (g/km)")
plt.title("Error medio por cantidad de cilindros")
plt.show()
```

### 🧠 Explicación:

- Se agregan dos nuevas columnas al DataFrame:
  - **Prediccion\_CO2**: resultado del modelo extendido
  - **Error**: diferencia absoluta con el valor real
- Luego, se **agrupa por cantidad de cilindros** y se calcula el error promedio por grupo.

- Se grafica un **barchart** para visualizar si el modelo tiene **mejor o peor rendimiento según el tipo de motor**.

### **Insight clave:**

Este tipo de análisis ayuda a **identificar segmentos** del dataset donde el modelo es más o menos preciso.

## ◆ **Celda 59**

```
python

# Análisis de errores por tipo de combustible (si existiera esa variable)
if "FUELTYPE" in df.columns:
    errores_prom = df.groupby("FUELTYPE")["Error"].mean()
    errores_prom.plot(kind='bar', color='darkcyan')
    plt.ylabel("Error promedio")
    plt.title("Error medio por tipo de combustible")
    plt.show()
else:
    print("No hay columna FUELTYPE en el dataset.")
```

### **Explicación:**

- Se intenta analizar si el **tipo de combustible** influye en la precisión del modelo.
- Si existe la columna **FUELTYPE**, se agrupan los errores por ese campo y se grafican.
- Si no existe, se muestra un mensaje.

### **Importancia:**

Este paso intenta usar **variables categóricas** para validar el rendimiento del modelo en distintos subgrupos, lo cual es parte de la **evaluación ética y técnica** de un modelo.

## ◆ **Celda 60**

```
python

# Guardar modelo entrenado
import joblib
joblib.dump(regr, "modelo_regresion_final.pkl")
```

### **Explicación:**

- Se usa la biblioteca **joblib** para guardar el modelo entrenado como archivo **.pkl** (pickle).
- Esto permite **usar el modelo en el futuro** sin tener que volver a entrenarlo.

### **Aplicación profesional:**

Guardar el modelo es un paso esencial en cualquier flujo de trabajo de machine learning que va a ser **desplegado o reutilizado**.

## ✓ Conclusión parcial (Celdas 56–60)

Este bloque cierra el notebook con funcionalidades avanzadas:

1. Predicciones con el modelo extendido.
2. Visualización de errores por observación y por grupo (cilindros, tipo de combustible).
3. Análisis de **subgrupos** donde el modelo tiene mejor o peor precisión.
4. Guarda del modelo entrenado para **reutilización o deployment**.

Estas prácticas reflejan un enfoque **completo y profesional de modelado predictivo**.

¿Querés que continúe con las celdas 61 a 68 (las últimas)? Así completamos el análisis total del notebook.



continua

Perfecto, vamos a finalizar el análisis explicando las **últimas celdas: 61 a 68** del notebook. Aquí se consolidan prácticas profesionales como la carga del modelo guardado y su aplicación a nuevos datos.

### ♦ Celda 61

```
python

# Cargar el modelo entrenado desde archivo
modelo_cargado = joblib.load("modelo_regresion_final.pkl")
```

#### Explicación:

- Usa `joblib.load()` para **leer el modelo previamente guardado** (`modelo_regresion_final.pkl`).
- Se almacena en la variable `modelo_cargado`.

#### Aplicación:

Este paso permite usar el modelo sin volver a entrenarlo, lo cual es **eficiente** y esencial para desplegarlo en producción.

### ♦ Celda 62

```
python

# Predicción con el modelo cargado
nuevo = np.array([[3.5, 6, 9.0, 10.5, 7.5]])
pred = modelo_cargado.predict(nuevo)
print("Predicción con modelo cargado:", pred)
```

### Explicación:

- Se realiza una predicción con el modelo recuperado desde disco.
- Se usan las mismas 5 variables que en el entrenamiento:
  - Tamaño motor, cilindros, consumo combinado, ciudad y autopista.

### Validación:

Esto comprueba que el modelo cargado **funciona exactamente igual** que el original.

## ◆ Celda 63

```
python

# Crear función para predecir emisiones
def predecir_emision(eng_size, cyl, comb, city, hwy):
    entrada = np.array([[eng_size, cyl, comb, city, hwy]])
    return modelo_cargado.predict(entrada)[0][0]
```

### Explicación:

- Se define una función `predecir_emision(...)` que encapsula el proceso de predicción.
- Recibe las cinco variables como parámetros.
- Devuelve el valor predicho (como número escalar, no array).

### Ventaja:

Esta función facilita la **reutilización del modelo** por parte de otros scripts o interfaces (por ejemplo, una app o API).

## ◆ Celda 64

```
python

# Ejemplo de uso de la función
print("Emisión estimada:", predecir_emision(2.5, 4, 7.5, 8.5, 6.0), "g/km")
```

### Explicación:

- Se invoca la función creada con valores específicos:
  - 2.5 L, 4 cilindros, 7.5 combinado, 8.5 ciudad, 6.0 autopista
- Se imprime la predicción.

Esto simula un caso real donde el usuario ingresa los datos y recibe la estimación de emisiones.

## ◆ Celda 65

python

```
# Interface básica usando input()
try:
    eng = float(input("Tamaño del motor (L): "))
    cyl = int(input("Cantidad de cilindros: "))
    comb = float(input("Consumo combinado: "))
    city = float(input("Consumo ciudad: "))
    hwy = float(input("Consumo autopista: "))
    resultado = predecir_emision(eng, cyl, comb, city, hwy)
    print(f"Emisión estimada: {resultado:.2f} g/km")
except:
    print("Error en los valores ingresados.")
```

### Explicación:

- Se crea una **interfaz de consola** simple para ingresar valores manualmente.
- El código:
  - Solicita valores al usuario
  - Ejecuta la predicción
  - Muestra el resultado

### Utilidad:

Esta celda transforma el modelo en una **herramienta interactiva básica**, lista para su uso inmediato.

## ◆ Celda 66

python

```
# Crear archivo CSV con predicciones para todo el dataset
df["Pred_CO2"] = modelo_cargado.predict(np.asanyarray(df[["ENGINE_SIZE", "CYLINDERS",
"FUELCONSUMPTION_COMB", "FUELCONSUMPTION_CITY", "FUELCONSUMPTION_Hwy"]]))
df.to_csv("predicciones_completas.csv", index=False)
```

### Explicación:

- Se calculan predicciones para **todos los registros del dataset original**.
- Se agrega una nueva columna **Pred\_CO2**.
- Se guarda todo en un archivo CSV llamado **predicciones\_completas.csv**.

### Aplicación profesional:

Esto permite exportar las predicciones a una hoja de cálculo o un sistema externo.

## ◆ Celda 67

python

```
# Ver primeras filas del archivo generado
df[["ENGINE_SIZE", "CYLINDERS", "FUEL_CONSUMPTION_COMB", "CO2EMISSIONS", "Pred_CO2"]].head()
```

### Explicación:

- Muestra las primeras filas del DataFrame con las columnas originales y la predicción.
- Esto permite comparar **emisiones reales vs estimadas** directamente.

## ◆ Celda 68

python

```
# Diferencia promedio entre valor real y predicho
error_promedio = np.mean(np.abs(df["CO2EMISSIONS"] - df["Pred_CO2"]))
print(f"Error promedio final del modelo: {error_promedio:.2f} g/km")
```

### Explicación:

- Se calcula la diferencia promedio (error absoluto medio) entre la columna real y la predicción en todo el dataset.

### Indicador final:

Sirve como **métrica global de rendimiento final** para el modelo desplegado.

## CONCLUSIÓN GENERAL DEL NOTEBOOK

El notebook implementa un flujo completo y profesional de **regresión lineal**:

### ◆ 1. Carga y preparación de datos

- Se importan datos desde CSV.
- Se exploran gráficamente relaciones lineales.
- Se hace limpieza y selección de variables.

### ◆ 2. Modelado

- Se entrenan distintos modelos de regresión:
  - Lineal simple
  - Múltiple
  - Con y sin normalización
- Se comparan métricas: MAE, MSE,  $R^2$ .



### ◆ 3. Visualización

- Gráficos de dispersión, histogramas de residuos, correlación, matrices y mapas de calor.
- Análisis de sensibilidad por variables.

### ◆ 4. Interpretación y aplicación

- Se interpretan coeficientes normalizados.
- Se realiza análisis por grupos (**CYLINDERS**, **FUELTYPE**).
- Se crean funciones reutilizables.

### ◆ 5. Producción

- Se guarda y carga el modelo (**joblib**).
- Se aplican predicciones a todo el dataset.
- Se genera un archivo con resultados (**predicciones\_completas.csv**).

## ANEXO: TEORÍA Y CONCEPTOS CLAVE

### 1. Regresión Lineal Simple:

$$y = \beta_0 + \beta_1 x$$

### 2. Regresión Lineal Múltiple:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

### 3. Métricas de evaluación:

- MAE: Error medio absoluto
- MSE: Error cuadrático medio
- $R^2$ : Proporción de varianza explicada

### 4. Normalización (StandardScaler):

Convierte los datos a media 0 y desviación 1. Ayuda en interpretabilidad y estabilidad.

### 5. Overfitting/Underfitting:

- Se controla dividiendo datos en train/test.
- Más variables no siempre mejoran el modelo.

### 6. Correlación:

- Mide la relación lineal entre variables.
- Importante para selección y evaluación de predictores.

### 7. Despliegue de modelos:

- Guardar con **joblib**
- Usar en APIs, interfaces gráficas o notebooks

✅ **AVISO:** Ya he explicado **las 68 celdas** del notebook.

¿Te gustaría que genere un resumen ejecutable del flujo o lo transforme en un pipeline automatizado?



necesito que realices el resumen

¡Perfecto! A continuación te presento un **resumen profesional y ejecutable del flujo completo del proyecto de regresión lineal** basado en el notebook que analizamos:



## RESUMEN DEL PROYECTO: Predicción de emisiones de CO<sub>2</sub> con regresión lineal

### ◆ 1. Carga y exploración de datos

- Se descargó un dataset con características técnicas de vehículos y emisiones de CO<sub>2</sub>.
- Se seleccionaron las variables más relevantes:
  - **ENGINE SIZE** (Tamaño del motor)
  - **CYLINDERS** (Número de cilindros)
  - **FUEL CONSUMPTION\_COMB, CITY, HWY** (Consumos)
  - **CO2 EMISSIONS** (Objetivo)

### ◆ 2. Visualización y análisis exploratorio

- Se utilizaron gráficos de dispersión para explorar correlaciones.
- Se generó una matriz de correlación (`.corr()` y `seaborn.heatmap`).
- Se identificaron relaciones fuertes entre consumo de combustible y emisiones.

### ◆ 3. Entrenamiento de modelos

Se entrenaron **varios modelos**:

- **Regresión lineal simple:**
  - **ENGINE SIZE vs CO2 EMISSIONS**
  - **FUEL CONSUMPTION\_COMB vs CO2 EMISSIONS**
- **Regresión lineal múltiple:**
  - Combinaciones de 2 y 3 variables
  - Modelo extendido con 5 variables

Cada modelo fue evaluado usando:

- MAE (Error medio absoluto)
- MSE (Error cuadrático medio)
- $R^2$  (Coeficiente de determinación)

#### ◆ 4. Normalización de variables

- Se aplicó `StandardScaler` para convertir los datos a media 0 y varianza 1.
- Se reentrenó el modelo con datos escalados.
- Se interpretaron los **coeficientes normalizados** para analizar la **importancia relativa** de cada predictor.

#### ◆ 5. Evaluación del modelo final

- Se analizaron errores:
  - Visualmente (residuos, dispersión)
  - Por subgrupos (cilindros, tipo de combustible)
- Se graficaron:
  - Predicciones vs valores reales
  - Error medio por grupo

#### ◆ 6. Despliegue del modelo

- Se guardó el modelo final con `joblib`.
- Se creó una función `predecir_emision()` para hacer predicciones programáticas.
- Se generó un archivo CSV con las predicciones para todo el dataset (`predicciones_completas.csv`).

#### ◆ 7. Predicciones interactivas

- Se implementó una pequeña interfaz por consola (`input()`) para que el usuario ingrese valores y obtenga su predicción.

### ✓ RESULTADO FINAL

El proyecto logra construir un modelo de regresión lineal robusto y comprensible que:

- Se entrena, valida y guarda correctamente
- Ofrece explicaciones gráficas e interpretables
- Puede usarse tanto en notebooks como en entornos de producción
- Tiene un error promedio final competitivo (reportado en la celda 68)

## HERRAMIENTAS Y LIBRERÍAS USADAS

- `pandas`, `numpy`: manipulación de datos
- `matplotlib`, `seaborn`: visualización
- `sklearn.linear_model.LinearRegression`: modelo de regresión
- `sklearn.metrics`: evaluación
- `sklearn.preprocessing.StandardScaler`: normalización
- `joblib`: serialización del modelo

## RECOMENDACIONES FINALES

### 1. Podrías mejorar el modelo:

- Probando regresión regularizada (Ridge, Lasso)
- Usando técnicas de selección de características automáticas

### 2. Podrías transformar este notebook en una API:

- Usando Flask o FastAPI

### 3. Podrías aplicar validación cruzada (`cross_val_score`) para asegurar que el rendimiento no depende del split inicial.

¿Querés que este resumen lo convierta en un archivo PDF, o que lo formatee como README para un repositorio de GitHub o Kaggle?



Realiza un resumen con toda la teoría de machine learning

¡Excelente decisión! Aquí tienes un **resumen completo y bien estructurado de toda la teoría de Machine Learning (ML)**. Este resumen cubre desde los fundamentos hasta los enfoques modernos, y está diseñado para servir como referencia profesional y educativa.

## RESUMEN GENERAL DE TEORÍA DE MACHINE LEARNING

### 1. ¿Qué es Machine Learning?

**Definición:** ML es una subdisciplina de la inteligencia artificial (IA) que se centra en el desarrollo de algoritmos que **aprenden patrones a partir de datos**, sin ser programados explícitamente para cada tarea.

### 2. Tipos de Aprendizaje

## ■ Aprendizaje Supervisado

- Entrenamiento con **datos etiquetados**.
- Objetivo: predecir una **salida conocida**.

Tarea	Algoritmos típicos	Ejemplo
Clasificación	KNN, SVM, Árboles, Regresión logística	Diagnóstico médico (sí/no)
Regresión	Regresión lineal, Random Forest, XGBoost	Predicción de precios, clima

## ■ Aprendizaje No Supervisado

- Entrenamiento con **datos no etiquetados**.
- Objetivo: **descubrir estructura o patrones** ocultos.

Tarea	Algoritmos típicos	Ejemplo
Agrupamiento	K-means, DBSCAN, HDBSCAN	Segmentación de clientes
Reducción Dim.	PCA, t-SNE, UMAP	Visualización de datos

## ■ Aprendizaje por Refuerzo (Reinforcement Learning)

- Un **agente** toma decisiones en un entorno y recibe **recompensas**.
- Objetivo: **maximizar la recompensa acumulada**.

Componentes	Ejemplo
Agente, Entorno, Política, Recompensa	Juegos, robótica, trading automático

## ◆ 3. Flujo de Trabajo de un Proyecto ML

1. Definición del problema
2. Recolección de datos
3. Exploración y limpieza de datos (EDA)
4. División del dataset (train/test/validation)
5. Selección del modelo
6. Entrenamiento
7. Evaluación (Métricas)
8. Ajuste de hiperparámetros
9. Validación cruzada
10. Despliegue del modelo

## ◆ 4. Modelos y Algoritmos Comunes

## ► Regresión

- **Lineal:**  $y = \beta_0 + \beta_1 x$
- **Ridge/Lasso:** regularización para evitar overfitting

## ► Clasificación

- **Regresión logística**
- **K-Nearest Neighbors (KNN)**
- **Support Vector Machines (SVM)**
- **Árboles de decisión, Random Forest**
- **Gradient Boosting (XGBoost, LightGBM)**

## ► Agrupamiento

- **K-means, DBSCAN, HDBSCAN**

## ► Reducción de Dimensiones

- **PCA** (análisis de componentes principales)
- **t-SNE, UMAP** (técnicas no lineales)

# ◆ 5. Métricas de Evaluación

## Clasificación:

- **Accuracy:**  $\frac{\text{aciertos}}{\text{total}}$
- **Precision, Recall, F1-score**
- **Matriz de confusión**
- **ROC-AUC**

## Regresión:

- **MAE:** Mean Absolute Error
- **MSE:** Mean Squared Error
- **RMSE:** Root MSE
- $R^2$ : coeficiente de determinación

# ◆ 6. Problemas Comunes

## ◆ Overfitting

- Modelo se ajusta demasiado al entrenamiento.

- Soluciones: regularización, más datos, cross-validation.

### ◆ Underfitting

- Modelo es demasiado simple para aprender patrones.
- Soluciones: modelos más complejos, más variables.

### ◆ Data Leakage

- Información del test se filtra en el entrenamiento.
- Solución: validar cuidadosamente el pipeline.

## ◆ 7. Preprocesamiento de Datos

- **Normalización / Estandarización**
- **Codificación de variables categóricas** (One-Hot, Label Encoding)
- **Imputación de valores faltantes**
- **Escalado**: StandardScaler, MinMaxScaler
- **Feature Engineering**: creación de variables

## ◆ 8. Validación y Selección de Modelos

### ➤ Cross-validation:

- K-Fold, Stratified K-Fold
- Evita que el rendimiento dependa de un solo split

### ➤ Búsqueda de hiperparámetros:

- **Grid Search**
- **Random Search**
- **Bayesian Optimization** (Optuna, Hyperopt)

## ◆ 9. Técnicas Avanzadas

- **Stacking / Blending / Voting**
- **AutoML** (TPOT, H2O, Google AutoML)
- **Pipelines de Scikit-learn**
- **Interpretabilidad**: SHAP, LIME

## ◆ 10. Despliegue de Modelos

- **Serialización:** joblib, pickle
- **APIs:** Flask, FastAPI
- **Plataformas:** AWS SageMaker, GCP Vertex AI, Azure ML

## ◆ 11. Ética y ML Responsable

- **Bias** (sesgo)
- **Fairness** (equidad)
- **Explainability**
- **Privacy / Seguridad**
- Evaluación por subgrupos

## ◆ 12. Machine Learning vs Deep Learning

Aspecto	Machine Learning	Deep Learning
Datos necesarios	Medio / bajo	Alto volumen
Ingeniería de features	Importante	Menos necesaria (usa redes profundas)
Interpretabilidad	Alta	Baja
Algoritmos	Árboles, SVM, regresión	Redes neuronales (CNN, RNN, Transformers)

## ✓ ¿Qué sigue?

Puedes usar esta base para profundizar en temas específicos como:

- ML para series temporales
- Modelado causal
- Sistemas de recomendación
- ML interpretativo y justo
- ML en producción (MLOps)

¿Querés que lo formatee como archivo PDF o README para tu portafolio de analista? ¿O preferís una versión imprimible en LaTeX o Word?