02 Uso de **MSE** (Mean Squared Error) y **MAE** (Mean Absolute Error)

así como sus implicaciones en el entrenamiento de un modelo. Vamos a construir un modelo de regresión simple y comparar los efectos de ambas funciones de pérdida.

Ejercicio práctico: Comparación de MSE y MAE en un modelo de regresión

Objetivo:

Implementar un modelo de regresión lineal y comparar cómo afectan las funciones de pérdida **MSE** y **MAE** en la optimización del modelo.

Requisitos:

- **Python** (con bibliotecas como NumPy, Matplotlib, y Scikit-learn)
- **Jupyter Notebook** o entorno de desarrollo similar

Pasos:

1. **Generar un conjunto de datos sintético**: Vamos a crear un conjunto de datos con una relación lineal con algo de ruido aleatorio.
2. **Entrenar el modelo de regresión lineal con MSE**: Usaremos la función de pérdida MSE para entrenar el modelo.
3. **Entrenar el modelo de regresión lineal con MAE**: Usaremos la función de pérdida MAE para entrenar el modelo.
4. **Comparar los resultados**: Evaluamos cómo el modelo entrenado con MSE y con MAE se comporta en cuanto a precisión y sensibilidad a los valores atípicos.

Voy a explicar cada parte del código, desglosando las instrucciones:

Importación de librerías

```
import numpy as np
```

- `import`: Palabra clave para cargar módulos
- `numpy`: Biblioteca para operaciones numéricas
- `as np`: Alias para usar la biblioteca

```
import matplotlib.pyplot as plt
```

- `matplotlib.pyplot`: Submódulo para visualización gráfica
- `as plt`: Alias asignado

```
from sklearn.linear_model import LinearRegression, SGDRegressor
```

- `from sklearn.linear_model`: Especifica submódulo
- `import`: Palabra clave para importación
- `LinearRegression, SGDRegressor`: Clases específicas para modelos lineales

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

- `sklearn.metrics`: Submódulo de métricas
- `mean_squared_error, mean_absolute_error`: Funciones para evaluar errores

```
from sklearn.model_selection import train_test_split
```

- `train_test_split`: Función para dividir datos

Generación de datos sintéticos

```
np.random.seed(42)
```

- `np.random.seed()`: Método para fijar semilla aleatoria
- `42`: Valor entero para reproducibilidad

```
X = np.linspace(0, 10, 100)
```

- `X`: Variable para datos de entrada
- `np.linspace()`: Función para generar valores equidistantes
- `0, 10`: Rango de inicio y fin

- `100`: Número de puntos a generar

```
y = 2 * X + 1 + np.random.normal(0, 2, X.shape[0])
```

- `y`: Variable para etiquetas
- `2 * X + 1`: Función lineal (pendiente 2, intercepto 1)
- `np.random.normal()`: Función para generar ruido gaussiano
- `0, 2`: Media y desviación estándar del ruido
- `X.shape[0]`: Número de elementos en X

División en conjuntos de entrenamiento y prueba

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

- `X_train, X_test, y_train, y_test`: Variables asignadas
- `train_test_split()`: Función para dividir datos
- `X, y`: Datos a dividir
- `test_size=0.2`: 20% para prueba, 80% para entrenamiento
- `random_state=42`: Semilla para reproducibilidad

Reformateo de datos

```
X_train = X_train.reshape(-1, 1)
```

- `X_train.reshape()`: Método para cambiar dimensiones
- `-1`: Dimensión automática (número de filas)
- `1`: Una columna
- Resultado: Matriz de formato (n_muestras, 1)

```
X_test = X_test.reshape(-1, 1)
```

- Operación similar para datos de prueba

Entrenamiento con MSE (Error Cuadrático Medio)

```
model_mse = LinearRegression()
```

- `model_mse`: Variable para el modelo MSE
- `LinearRegression()`: Constructor del modelo

```
model_mse.fit(X_train, y_train)
```

- `.fit()`: Método para entrenar el modelo
- `X_train, y_train`: Datos de entrenamiento

```
y_pred_mse = model_mse.predict(X_test)
```

- `y_pred_mse`: Variable para predicciones
- `.predict()`: Método para predecir
- `X_test`: Datos de entrada para predicción

Entrenamiento con MAE (Error Absoluto Medio)

```
model_mae = SGDRegressor(loss='epsilon_insensitive', epsilon=0.1)
```

- `model_mae`: Variable para el modelo MAE
- `SGDRegressor()`: Constructor (Descenso de Gradiente Estocástico)
- `loss='epsilon_insensitive'`: Parámetro de función de pérdida L1 (MAE)
- `epsilon=0.1`: Margen de error ignorado

```
model_mae.fit(X_train, y_train)
```

- Entrenamiento con los mismos datos

```
y_pred_mae = model_mae.predict(X_test)
```

- Predicciones con el modelo MAE

Evaluación de modelos

```
mse = mean_squared_error(y_test, y_pred_mse)
```

- `mse`: Variable para almacenar error
- `mean_squared_error()`: Función de métrica
- `y_test, y_pred_mse`: Valores reales y predichos

```
mae = mean_absolute_error(y_test, y_pred_mae)
```

- `mae`: Variable para error absoluto
- `mean_absolute_error()`: Función de métrica

Impresión de resultados

```
print(f"Error Cuadrático Medio (MSE): {mse:.4f}")
```

- `print()`: Función para mostrar texto
- `f"..."`: String formateado (f-string)
- `{mse:.4f}`: Variable con formato (4 decimales)

```
print(f"Error Absoluto Medio (MAE): {mae:.4f}")
```

- Impresión similar para MAE

Visualización de resultados

```
plt.figure(figsize=(12, 6))
```

- `plt.figure()`: Función para crear figura
- `figsize=(12, 6)`: Tamaño en pulgadas (ancho, alto)

```
plt.scatter(X_test, y_test, color='blue', label='Datos reales')
```

- `plt.scatter()`: Función para gráfico de dispersión
- `X_test, y_test`: Coordenadas x,y
- `color='blue'`: Parámetro de color
- `label='Datos reales'`: Etiqueta para leyenda

```
plt.plot(X_test, y_pred_mse, color='red', label='Modelo con MSE')
```

- `plt.plot()`: Función para línea continua
- `X_test, y_pred_mse`: Datos x,y para la línea
- `color='red'`: Color rojo
- `label='Modelo con MSE'`: Etiqueta para leyenda

```
plt.plot(X_test, y_pred_mae, color='green', label='Modelo con MAE')
```

- Gráfico similar para modelo MAE con color verde

```
plt.xlabel('X')
```

- `plt.xlabel()`: Función para etiqueta eje X
- `'X'`: Texto de la etiqueta

```
plt.ylabel('y')
```

- `plt.ylabel()`: Función para etiqueta eje Y

```
plt.legend()
```

- `plt.legend()`: Función para mostrar leyenda

```
plt.title('Comparación de Modelos de Regresión con MSE y MAE')
```

- `plt.title()`: Función para título del gráfico
- `'Comparación...'`: Texto del título

```
plt.show()
```

- `plt.show()`: Función para mostrar el gráfico en pantalla

Este código compara dos enfoques de regresión lineal: uno optimizado para minimizar el error cuadrático medio (MSE) y otro para minimizar el error absoluto medio (MAE), mostrando gráficamente las diferencias entre ambos modelos al predecir datos con ruido.

Conclusion:

El resultado de ambos es similar. Esto es algo esperado y tiene una explicación estadística:

En este caso particular, ambos modelos están intentando encontrar los parámetros óptimos para una regresión lineal (y = mx + b) sobre datos generados con una relación lineal subyacente (2*X + 1) con ruido gaussiano añadido.

Algunos motivos por los que las líneas casi se solapan:

1. **Distribución del ruido**: El ruido utilizado es gaussiano (normal) con `np.random.normal(0, 2, X.shape[0])`. Cuando el ruido tiene una distribución simétrica como la normal, tanto MSE como MAE tienden a encontrar soluciones similares.

2. **Relación verdaderamente lineal**: Los datos fueron generados con una relación perfectamente lineal antes de añadir el ruido. Ambos modelos

pueden capturar fácilmente esta estructura.

3. **Suficientes datos**: Con 100 puntos de datos, ambos modelos tienen suficiente información para converger a soluciones similares.

4. **Ausencia de valores atípicos significativos**: La diferencia entre MSE y MAE es más notable cuando hay valores atípicos (outliers) importantes, ya que:

    - MSE penaliza más los errores grandes debido al cuadrado
    - MAE trata todos los errores proporcionales a su magnitud

Si quisieras ver una diferencia más marcada entre los dos modelos, podrías:

- Añadir algunos valores atípicos extremos
- Usar una distribución de ruido no simétrica
- Generar datos con una relación no lineal

En entornos reales, las diferencias entre MSE y MAE suelen ser más evidentes debido a la presencia de outliers y relaciones más complejas en los datos.

Entendido, voy a desglosar y explicar el código siguiendo ese formato específico:

Importación de librerías

```
import numpy as np
```

- `import`: Palabra clave para importar módulos
- `numpy`: Biblioteca para cálculos numéricos
- `as np`: Alias para referirse a numpy

```
import matplotlib.pyplot as plt
```

- `import`: Palabra clave para importar módulos
- `matplotlib.pyplot`: Módulo para visualización gráfica
- `as plt`: Alias para referirse a matplotlib.pyplot

```
from sklearn.linear_model import LinearRegression, SGDRegressor
```

- `from`: Indica importación específica desde un módulo
- `sklearn.linear_model`: Módulo de modelos lineales de scikit-learn
- `import`: Palabra clave para importar

- `LinearRegression, SGDRegressor`: Clases específicas que se importan

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

- `from`: Indica importación específica
- `sklearn.metrics`: Módulo de métricas de evaluación
- `import`: Palabra clave para importar
- `mean_squared_error, mean_absolute_error`: Funciones para evaluar modelos

```
from sklearn.model_selection import train_test_split
```

- `from`: Indica importación específica
- `sklearn.model_selection`: Módulo para partición de datos
- `import`: Palabra clave para importar
- `train_test_split`: Función para dividir datos

```
from sklearn.preprocessing import PolynomialFeatures
```

- `from`: Indica importación específica
- `sklearn.preprocessing`: Módulo para preprocesamiento
- `import`: Palabra clave para importar
- `PolynomialFeatures`: Clase para transformación polinómica

```
from sklearn.pipeline import Pipeline
```

- `from`: Indica importación específica
- `sklearn.pipeline`: Módulo para flujos de trabajo
- `import`: Palabra clave para importar
- `Pipeline`: Clase para encadenar transformaciones

Generación de datos

```
np.random.seed(42)
```

- `np.random`: Submódulo de numpy para generación aleatoria
- `seed()`: Función para fijar la semilla aleatoria
- `42`: Valor de la semilla para reproducibilidad

```
X = np.linspace(0, 10, 100)
```

- X: Variable para almacenar los datos de entrada
- np.linspace(): Función para generar secuencia equidistante
- 0, 10: Rango de valores (inicio, fin)
- 100: Número de puntos a generar

```
y_true = X**2 - 3*X + 2
```

- y_true: Variable para valores reales sin ruido
- X**2: X elevado al cuadrado
- -3*X: Multiplicación de X por -3
- +2: Suma del valor constante 2
- Ecuación cuadrática: y = x² - 3x + 2

```
ruido = np.random.chisquare(df=2, size=X.shape[0]) - 2
```

- ruido: Variable para almacenar el ruido
- np.random.chisquare(): Función para distribución chi-cuadrado
- df=2: Parámetro de grados de libertad
- size=X.shape[0]: Tamaño igual al número de elementos en X
- -2: Desplazamiento para centrar cerca de cero

```
y = y_true + ruido
```

- y: Variable para valores con ruido
- y_true: Valores sin ruido
- +: Operador de suma
- ruido: Componente aleatorio

```
num_outliers = 5
```

- num_outliers: Variable para número de valores atípicos
- 5: Cantidad de outliers a generar

```
outlier_indices = np.random.choice(range(len(X)), num_outliers, replace=False)
```

- outlier_indices: Variable para índices de outliers
- np.random.choice(): Función para selección aleatoria
- range(len(X)): Rango de índices disponibles
- num_outliers: Cantidad a seleccionar

- `replace=False`: Sin reemplazo (índices únicos)

```
y[outlier_indices] = y[outlier_indices] + np.random.choice([-15, 15], num_outliers)
```

- `y[outlier_indices]`: Acceso a valores específicos de y
- `=`: Operador de asignación
- `+ np.random.choice([-15, 15], num_outliers)`: Suma valores extremos

División de datos

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

- `X_train, X_test, y_train, y_test`: Variables para conjuntos de datos
- `train_test_split()`: Función para dividir datos
- `X, y`: Datos de entrada y salida
- `test_size=0.2`: Proporción para prueba (20%)
- `random_state=42`: Semilla para reproducibilidad

```
X_train = X_train.reshape(-1, 1)
```

- `X_train`: Variable de datos de entrenamiento
- `reshape()`: Método para cambiar forma del array
- `-1, 1`: Parámetros para forma (filas automáticas, 1 columna)

```
X_test = X_test.reshape(-1, 1)
```

- `X_test`: Variable de datos de prueba
- `reshape()`: Método para cambiar forma
- `-1, 1`: Parámetros para forma (filas automáticas, 1 columna)

Modelo con MSE

```
model_mse = Pipeline([
    ('poly', PolynomialFeatures(degree=2)),
    ('linear', LinearRegression())
])
```

- `model_mse`: Variable para el modelo MSE

- `Pipeline()`: Constructor para encadenar procesos
- `[]`: Lista de pasos del pipeline
- `'poly'`: Nombre del primer paso
- `PolynomialFeatures(degree=2)`: Transformador polinómico
- `'linear'`: Nombre del segundo paso
- `LinearRegression()`: Modelo de regresión lineal

Esta instrucción crea un **Pipeline** (flujo de trabajo) para procesar datos categóricos (como colores, géneros, países, etc.) en dos pasos consecutivos:

1. **Primero**, utiliza un `SimpleImputer` con la estrategia "most_frequent":
    - Esta parte se encarga de manejar los valores faltantes (NaN, NULL, etc.) en las columnas categóricas
    - La estrategia "most_frequent" significa que reemplaza cualquier valor faltante con el valor que aparece con mayor frecuencia en esa columna
    - Por ejemplo, si tienes datos sobre "País" y faltan algunos valores, los reemplazará con el país que más se repite
2. **Segundo**, aplica un `OneHotEncoder` con la opción "drop='first'":
    - Convierte las categorías en columnas numéricas binarias (0 y 1)
    - Por cada categoría, crea una nueva columna donde 1 significa "pertenece a esta categoría" y 0 significa "no pertenece"
    - La opción "drop='first'" elimina la primera categoría para evitar redundancia (esto ayuda a prevenir la "trampa de las variables dummy")
    - Por ejemplo, si "Color" tiene valores "rojo", "verde" y "azul", creará solo columnas para "verde" y "azul" (eliminando "rojo" para evitar información redundante)

En resumen, esta instrucción crea un proceso automatizado que primero rellena cualquier valor faltante con el valor más común, y luego transforma las categorías en un formato numérico que los algoritmos de machine learning pueden entender.

```
model_mse.fit(X_train, y_train)
```

- `model_mse`: Objeto del modelo
- `fit()`: Método para entrenar el modelo
- `X_train, y_train`: Datos de entrenamiento

```
y_pred_mse = model_mse.predict(X_test)
```

- `y_pred_mse`: Variable para predicciones
- `model_mse.predict()`: Método para predecir
- `X_test`: Datos de entrada para predicción

Modelo con MAE

```
model_mae = Pipeline([
    ('poly', PolynomialFeatures(degree=2)),
    ('sgd', SGDRegressor(loss='epsilon_insensitive',
                         epsilon=0,
                         max_iter=10000,
                         tol=1e-5,
                         random_state=42))
])
```

- `model_mae`: Variable para el modelo MAE
- `Pipeline()`: Constructor para pipeline
- `'poly'`: Nombre del primer paso
- `PolynomialFeatures(degree=2)`: Transformador polinómico
- `'sgd'`: Nombre del segundo paso
- `SGDRegressor()`: Modelo de regresión con descenso de gradiente estocástico
- `loss='epsilon_insensitive'`: Parámetro de función de pérdida
- `epsilon=0`: Sin margen de tolerancia
- `max_iter=10000`: Máximo número de iteraciones
- `tol=1e-5`: Tolerancia para convergencia
- `random_state=42`: Semilla para reproducibilidad

Esta instrucción crea un modelo especial para predecir valores numéricos, diseñado para ser menos sensible a valores extremos (outliers). Funciona en dos pasos:

1. **Primero**, con `PolynomialFeatures(degree=2)`:
    - Transforma tus datos originales añadiendo términos cuadráticos
    - Por ejemplo, si tienes una variable X, crea nuevas variables como X^2, permitiendo al modelo capturar relaciones curvas o no lineales
    - Esto es como darle al modelo "superpoderes" para detectar patrones más complejos que una simple línea recta
2. **Segundo**, con `SGDRegressor` configurado de forma especial:

    - Usa un algoritmo de aprendizaje llamado "descenso de gradiente estocástico" que va mejorando poco a poco sus predicciones
    - `loss='epsilon_insensitive', epsilon=0`: Configura el modelo para minimizar el error absoluto (MAE) en lugar del error cuadrático (MSE)
    - Esta configuración hace que el modelo sea más resistente a valores extremos, ya que no penaliza demasiado los errores grandes
    - `max_iter=10000`: Le da hasta 10,000 intentos para encontrar la mejor solución
    - `tol=1e-5`: Establece un nivel de precisión para saber cuándo dejar de mejorar
    - `random_state=42`: Asegura que el modelo dé los mismos resultados cada vez que se ejecute

Este código crea un modelo que puede capturar relaciones curvas en los datos y que es especialmente bueno lidiando con datos que tienen valores atípicos o extremos.

```
model_mae.fit(X_train, y_train)
```

- `model_mae`: Objeto del modelo
- `fit()`: Método para entrenar
- `X_train, y_train`: Datos de entrenamiento

```
y_pred_mae = model_mae.predict(X_test)
```

- `y_pred_mae`: Variable para predicciones
- `model_mae.predict()`: Método para predecir
- `X_test`: Datos de entrada para predicción

Evaluación de modelos

```
mse = mean_squared_error(y_test, y_pred_mse)
```

- `mse`: Variable para almacenar error cuadrático
- `mean_squared_error()`: Función de métrica
- `y_test, y_pred_mse`: Valores reales y predichos

```
mae = mean_absolute_error(y_test, y_pred_mae)
```

- `mae`: Variable para error absoluto
- `mean_absolute_error()`: Función de métrica
- `y_test, y_pred_mae`: Valores reales y predichos

```
print(f"Error Cuadrático Medio (MSE): {mse:.4f}")
```

- `print()`: Función para mostrar texto
- `f"..."`: String con formato
- `{mse:.4f}`: Variable formateada con 4 decimales

```
print(f"Error Absoluto Medio (MAE): {mae:.4f}")
```

- `print()`: Función para mostrar texto
- `f"..."`: String con formato
- `{mae:.4f}`: Variable formateada con 4 decimales

Visualización de resultados

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))
```

- `fig`: Variable para figura
- `ax1, ax2`: Variables para subplots
- `plt.subplots()`: Función para crear subplots
- `1, 2`: Parámetros de disposición (1 fila, 2 columnas)
- `figsize=(15, 6)`: Tamaño de la figura en pulgadas

```
ax1.scatter(X_test, y_test, color='blue', label='Datos reales', alpha=0.7)
```

- `ax1`: Primer subplot
- `scatter()`: Método para gráfico de dispersión
- `X_test, y_test`: Datos a graficar
- `color='blue'`: Color de los puntos
- `label='Datos reales'`: Etiqueta para leyenda
- `alpha=0.7`: Transparencia (70% opaco)

```
X_plot = np.linspace(0, 10, 100).reshape(-1, 1)
```

- `X_plot`: Variable para valores X

- `np.linspace(0, 10, 100)`: Genera secuencia
- `reshape(-1, 1)`: Reformatea a columna

```
ax1.plot(X_plot, model_mse.predict(X_plot), color='red', label='Modelo con MSE', linewidth=2)
```

- `ax1.plot()`: Método para gráfico de línea
- `X_plot`: Valores X para la curva
- `model_mse.predict(X_plot)`: Predicciones para esos valores
- `color='red'`: Color de la línea
- `label='Modelo con MSE'`: Etiqueta
- `linewidth=2`: Grosor de línea

```
ax1.plot(X_plot, model_mae.predict(X_plot), color='green', label='Modelo con MAE', linewidth=2)
```

- `ax1.plot()`: Método para gráfico de línea
- `X_plot`: Valores X para la curva
- `model_mae.predict(X_plot)`: Predicciones MAE
- `color='green'`: Color de la línea
- `label='Modelo con MAE'`: Etiqueta
- `linewidth=2`: Grosor de línea

```
ax1.set_xlabel('X')
```

- `ax1.set_xlabel()`: Método para etiqueta X
- `'X'`: Texto de la etiqueta

```
ax1.set_ylabel('y')
```

- `ax1.set_ylabel()`: Método para etiqueta Y
- `'y'`: Texto de la etiqueta

```
ax1.set_title('Comparación de Modelos de Regresión con MSE y MAE')
```

- `ax1.set_title()`: Método para título
- Texto del título entre comillas

```
ax1.legend()
```

- `ax1.legend()`: Método para mostrar leyenda

```
for idx in outlier_indices:
    if X[idx] in X_test.flatten():
        i = np.where(X_test.flatten() == X[idx])[0][0]
        ax2.scatter([X_test[i]], [y_test[i]], color='red', s=100,
edgecolor='black', zorder=5)
```

- for idx in outlier_indices:: Bucle para cada índice
- if X[idx] in X_test.flatten():: Condición si existe en test
- i = np.where(X_test.flatten() == X[idx])[0][0]: Obtiene índice
- ax2.scatter(): Dibuja puntos
- [X_test[i]], [y_test[i]]: Coordenadas del punto
- color='red': Color de relleno
- s=100: Tamaño del punto
- edgecolor='black': Color de borde
- zorder=5: Orden de capa (encima)

```
residuos_mse = y_test - y_pred_mse
```

- residuos_mse: Variable para residuos MSE
- y_test - y_pred_mse: Diferencia entre real y predicho

```
residuos_mae = y_test - y_pred_mae
```

- residuos_mae: Variable para residuos MAE
- y_test - y_pred_mae: Diferencia entre real y predicho

```
ax2.scatter(X_test, residuos_mse, color='red', label='Residuos MSE', alpha=0.7)
```

- ax2.scatter(): Método para dispersión
- X_test, residuos_mse: Datos a graficar
- color='red': Color de puntos
- label='Residuos MSE': Etiqueta
- alpha=0.7: Transparencia

```
ax2.scatter(X_test, residuos_mae, color='green', label='Residuos MAE', alpha=0.7)
```

- ax2.scatter(): Método para dispersión
- X_test, residuos_mae: Datos a graficar
- color='green': Color de puntos
- label='Residuos MAE': Etiqueta

- `alpha=0.7`: Transparencia

```
ax2.axhline(y=0, color='black', linestyle='-', alpha=0.3)
```

- `ax2.axhline()`: Método para línea horizontal
- `y=0`: Posición en eje Y
- `color='black'`: Color de línea
- `linestyle='-'`: Estilo continuo
- `alpha=0.3`: Transparencia (30% opaco)

```
ax2.set_xlabel('X')
```

- `ax2.set_xlabel()`: Método para etiqueta X
- `'X'`: Texto de la etiqueta

```
ax2.set_ylabel('Residuos (Valor real - Predicción)')
```

- `ax2.set_ylabel()`: Método para etiqueta Y
- Texto descriptivo entre comillas

```
ax2.set_title('Comparación de Residuos: MSE vs MAE')
```

- `ax2.set_title()`: Método para título
- Texto del título entre comillas

```
ax2.legend()
```

- `ax2.legend()`: Método para mostrar leyenda

```
plt.suptitle('Comparación detallada de modelos con MSE y MAE', fontsize=16)
```

- `plt.suptitle()`: Método para título general
- Texto del título entre comillas
- `fontsize=16`: Tamaño de fuente

```
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```

- `plt.tight_layout()`: Ajusta espaciado
- `rect=[0, 0.03, 1, 0.95]`: Rectángulo para ajuste

```
plt.show()
```

- `plt.show()`: Método para mostrar gráficos

Análisis adicional

```
print("\nAnálisis de diferencias entre modelos:")
```

- `print()`: Función para mostrar texto
- `\n`: Carácter de nueva línea

```
diferencia_predicciones = np.abs(y_pred_mse - y_pred_mae)
```

- `diferencia_predicciones`: Variable para diferencias
- `np.abs()`: Función valor absoluto
- `y_pred_mse - y_pred_mae`: Diferencia entre predicciones

```
print(f"Diferencia media entre predicciones:
{np.mean(diferencia_predicciones):.4f}")
```

- `print()`: Función para mostrar texto
- `np.mean(diferencia_predicciones)`: Promedio de diferencias
- `:.4f`: Formato con 4 decimales

```
print(f"Diferencia máxima entre predicciones:
{np.max(diferencia_predicciones):.4f}")
```

- `print()`: Función para mostrar texto
- `np.max(diferencia_predicciones)`: Máxima diferencia
- `:.4f`: Formato con 4 decimales

```
indices_outliers_test = [i for i, x in enumerate(X_test) if x.item() in
X[outlier_indices]]
```

- `indices_outliers_test`: Variable para índices
- `[i for i, x in enumerate(X_test)]`: Comprensión de lista
- `enumerate(X_test)`: Genera pares (índice, valor)
- `if x.item() in X[outlier_indices]`: Condición si es outlier

```
if indices_outliers_test:
    error_mse_outliers = np.mean(np.abs(y_test[indices_outliers_test] -
y_pred_mse[indices_outliers_test]))
    error_mae_outliers = np.mean(np.abs(y_test[indices_outliers_test] -
y_pred_mae[indices_outliers_test]))
    print(f"\nError medio en outliers:")
    print(f"- MSE: {error_mse_outliers:.4f}")
    print(f"- MAE: {error_mae_outliers:.4f}")
```

- `if indices_outliers_test:`: Condición si hay outliers
- `error_mse_outliers`: Variable para error MSE
- `np.mean(np.abs())`: Promedio de valor absoluto
- `y_test[indices_outliers_test] - y_pred_mse[indices_outliers_test]`: Error en outliers
- `error_mae_outliers`: Similar para MAE
- `print()`: Muestra resultados
- `:.4f`: Formato con 4 decimales