

Chat 1 kNN:

<https://chatgpt.com/share/67f2e4c0-8f4c-8007-874a-9be7f6fff09d>

chat 2 kNN

'uci_id': 186, 'name': 'Wine Quality', 'repository_url':

<https://archive.ics.uci.edu/dataset/186/wine+quality>, 'data_url':

<https://archive.ics.uci.edu/static/public/186/data.csv>, 'abstract': 'Two datasets are included, related to red and white vinho verde wine samples, from the north of Portugal. The goal is to model wine quality based on physicochemical tests (see [Cortez et al., 2009],

<http://www3.dsi.uminho.pt/pcortez/wine/>), 'area': 'Business', 'tasks': ['Classification',

'Regression'], 'characteristics': ['Multivariate'], 'num_instances': 4898, 'num_features': 11,

'feature_types': ['Real'], 'demographics': [], 'target_col': ['quality'], 'index_col': None,

'has_missing_values': 'no', 'missing_values_symbol': None, 'year_of_dataset_creation':

2009, 'last_updated': 'Wed Nov 15 2023', 'dataset_doi': '10.24432/C56S3T', 'creators':

['Paulo Cortez', 'A. Cerdeira', 'F. Almeida', 'T. Matos', 'J. Reis'], 'intro_paper': {'ID': 252, 'type':

'NATIVE', 'title': 'Modeling wine preferences by data mining from physicochemical properties',

'authors': 'P. Cortez, A. Cerdeira, Fernando Almeida, Telmo Matos, J. Reis', 'venue':

'Decision Support Systems', 'year': 2009, 'journal': None, 'DOI': None, 'URL':

<https://www.semanticscholar.org/paper/Modeling-wine-preferences-by-data-mining-from-Cortez-Cerdeira/bf15a0ccc14ac1deb5cea570c870389c16be019c>, 'sha': None, 'corpus': None,

'arxiv': None, 'mag': None, 'acl': None, 'pmid': None, 'pmcid': None}, 'additional_info':

{'summary': 'The two datasets are related to red and white variants of the Portuguese "Vinho Verde" wine. For more details, consult: <http://www.vinhoverde.pt/en/> or the reference [Cortez et al., 2009]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory

(the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).\n\nThese datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are many more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.\n', 'purpose': None, 'funded_by': None,

'instances_represent': None, 'recommended_data_splits': None, 'sensitive_data': None,

'preprocessing_description': None, 'variable_info': 'For more information, read [Cortez et al., 2009].\n\nInput variables (based on physicochemical tests):\n\n1 - fixed acidity\n\n2 - volatile acidity\n\n3 - citric acid\n\n4 - residual sugar\n\n5 - chlorides\n\n6 - free sulfur dioxide\n\n7 - total sulfur dioxide\n\n8 - density\n\n9 - pH\n\n10 - sulphates\n\n11 - alcohol\n\nOutput variable (based on sensory data): \n\n12 - quality (score between 0 and 10)', 'citation':

None}} name role type demographic

0 fixed_acidity Feature Continuous None

1 volatile_acidity Feature Continuous None

2 citric_acid Feature Continuous None

3 residual_sugar Feature Continuous None

4 chlorides Feature Continuous None

5 free_sulfur_dioxide Feature Continuous None

6 total_sulfur_dioxide Feature Continuous None

7 density Feature Continuous None

8 pH Feature Continuous None

9 sulphates Feature Continuous None

10 alcohol Feature Continuous None
11 quality Target Integer None
12 color Other Categorical None

Unset

description units missing_values

0 None None no
1 None None no
2 None None no
3 None None no
4 None None no
5 None None no
6 None None no
7 None None no
8 None None no
9 None None no
10 None None no
11 score between 0 and 10 None no
12 red or white None no

Click to add a cell. "esto es lo que he obtenido al ejecutar el siguiente código de python"
:from ucimlrepo import fetch_ucirepo

fetch dataset

```
wine_quality = fetch_ucirepo(id=186)
```

data (as pandas dataframes)

```
X = wine_quality.data.features y = wine_quality.data.targets
```

metadata

```
print(wine_quality.metadata)
```

variable information

```
print(wine_quality.variables) , "como puedo obtener un dataframe con el cual trabajar  
comodamente a partir de lo que ves?"
```

Puedes obtener un DataFrame de Pandas con todas las características y la variable objetivo de la siguiente manera:

Código para obtener un DataFrame completo

```
Python
import pandas as pd
from ucimlrepo import fetch_ucirepo

# Cargar el dataset
wine_quality = fetch_ucirepo(id=186)

# Crear un DataFrame con las características y la variable
objetivo
df = pd.concat([wine_quality.data.features,
wine_quality.data.targets], axis=1)

# Mostrar las primeras filas del DataFrame
print(df.head())
```

Explicación:

- `wine_quality.data.features`: Contiene las características del vino.
- `wine_quality.data.targets`: Contiene la columna objetivo (`quality`).
- `pd.concat(..., axis=1)`: Une ambas partes en un solo DataFrame.

Ahora puedes trabajar con `df` como cualquier otro DataFrame de Pandas. 🚀

como se ejecuta una celda en jupyter labs?

En **JupyterLab**, puedes ejecutar una celda de código de varias maneras:

1) Atajo de teclado (el más rápido)

- **Shift + Enter** → Ejecuta la celda y pasa a la siguiente.
- **Ctrl + Enter** → Ejecuta la celda y se queda en la misma.
- **Alt + Enter** → Ejecuta la celda y crea una nueva debajo.

2 Botón en la barra de herramientas

- Haz clic en la celda que quieres ejecutar.
- Pulsa el botón "► Run" (en la parte superior).

3 Menú

- Ve a "Run" en la barra superior y elige "Run Selected Cells".

Si tienes más preguntas sobre JupyterLab, dime. ¡A programar! 🚀

comprobar si en el dataframe hay NaN

Para comprobar si hay valores **NaN (faltantes)** en tu DataFrame en **JupyterLab**, puedes usar los siguientes métodos:

1 Ver si hay algún NaN en todo el DataFrame

```
Python  
df.isnull().sum()
```

- ♦ Esto mostrará cuántos valores NaN hay en cada columna.

2 Ver si hay algún NaN en todo el DataFrame

```
Python  
df.isnull().values.any()
```

- ♦ Devuelve **True** si hay al menos un NaN, **False** si no hay ninguno.

3 Mostrar solo las filas con NaN

```
Python  
df[df.isnull().any(axis=1)]
```

- ♦ Muestra solo las filas que contienen valores faltantes.

Según la descripción del dataset que me pasaste, **no debería haber valores NaN**, pero siempre es bueno verificar. 🚀

Aquí tienes un script en **Python** que aplica **PCA (Análisis de Componentes Principales)** a un DataFrame para reducir la dimensionalidad antes de realizar una **regresión**:

genera un script que genere un PCA a partir de un dataframe. Dicho PCA es para hacer una regresion

Qué hace el script:

1. **Carga** un DataFrame (**df**) con características y una variable objetivo.
2. **Escala** las características con **StandardScaler** (PCA es sensible a la escala).
3. **Aplica PCA** para reducir la dimensionalidad.
4. **Divide los datos** en entrenamiento y prueba.
5. **Entrena** un modelo de regresión (Regresión Lineal).
6. **Evalúa** el modelo con **R²** y el error cuadrático medio (MSE).

Código Completo:

```
Python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# ① Simulación de un DataFrame (sustituir con tus datos)
np.random.seed(42)
df = pd.DataFrame(np.random.rand(100, 6), columns=['X1', 'X2', 'X3', 'X4', 'X5', 'Target'])

# ② Separar características (X) y variable objetivo (y)
X = df.drop(columns=['Target'])
y = df['Target']

# ③ Escalar las características (PCA es sensible a la escala)
scaler = StandardScaler()
```

```

X_scaled = scaler.fit_transform(X)

# ④ Aplicar PCA (reducir a 2 componentes como ejemplo)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# ⑤ Dividir en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_pca, y,
test_size=0.2, random_state=42)

# ⑥ Entrenar un modelo de regresión
model = LinearRegression()
model.fit(X_train, y_train)

# ⑦ Hacer predicciones
y_pred = model.predict(X_test)

# ⑧ Evaluar el modelo
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f'R² Score: {r2:.4f}')
print(f'Mean Squared Error: {mse:.4f}')

# ⑨ Visualizar los datos transformados con PCA
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='coolwarm',
edgecolors='k')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Datos transformados con PCA')
plt.colorbar(label='Valor objetivo')
plt.show()

```

♦ Explicación de los pasos:

- ✓ **Escalado:** PCA requiere que los datos estén normalizados.
- ✓ **Reducción de dimensionalidad:** Se usa `PCA(n_components=2)`, pero puedes cambiarlo.
- ✓ **Regresión:** Se usa `LinearRegression`, pero puedes probar con otros modelos.

- ✓ **Evaluación:** Se mide el rendimiento con R^2 y **MSE**.
- ✓ **Visualización:** Se grafican los datos después del PCA.

♦ Puedes ajustar **n_components** en `PCA(n_components=X)` según lo que necesites.



la regresión la quiero hacer con kNN

Aquí tienes un script en **Python** que aplica **PCA** y luego usa **K-Nearest Neighbors Regression (kNN)** para hacer la regresión.

Pasos del Script:

- 1 **Carga los datos** en un DataFrame.
- 2 **Separa las características (X) y la variable objetivo (y).**
- 3 **Escala las características** con `StandardScaler` (PCA es sensible a la escala).
- 4 **Aplica PCA** para reducir la dimensionalidad.
- 5 **Divide los datos** en entrenamiento y prueba.
- 6 **Entrena un modelo de regresión con `KNeighborsRegressor`.**
- 7 **Evalúa el modelo** con R^2 y el error cuadrático medio (**MSE**).
- 8 **Grafica** los datos transformados con PCA.

Código Completo:

```
Python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# 1 Simulación de un DataFrame (sustituir con tus datos)
np.random.seed(42)
df = pd.DataFrame(np.random.rand(100, 6), columns=['X1', 'X2',
'X3', 'X4', 'X5', 'Target'])
```

```

# ② Separar características (X) y variable objetivo (y)
X = df.drop(columns=['Target'])
y = df['Target']

# ③ Escalar las características (PCA es sensible a la escala)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ④ Aplicar PCA (reducir a 2 componentes como ejemplo)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# ⑤ Dividir en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_pca, y,
test_size=0.2, random_state=42)

# ⑥ Entrenar un modelo de kNN regresión
k = 5 # Número de vecinos
knn = KNeighborsRegressor(n_neighbors=k)
knn.fit(X_train, y_train)

# ⑦ Hacer predicciones
y_pred = knn.predict(X_test)

# ⑧ Evaluar el modelo
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f'R² Score: {r2:.4f}')
print(f'Mean Squared Error: {mse:.4f}')

# ⑨ Visualizar los datos transformados con PCA
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='coolwarm',
edgecolors='k')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Datos transformados con PCA')
plt.colorbar(label='Valor objetivo')

```



```
plt.show()
```

♦ Explicación de los pasos:

- ✓ **PCA** reduce la dimensionalidad para mejorar la eficiencia de kNN.
- ✓ Se usa **KNeighborsRegressor** con **n_neighbors=5**, pero puedes cambiarlo.
- ✓ Se mide la calidad del modelo con **R²** y **MSE**.
- ✓ **Visualización**: Gráfica de los datos en el espacio reducido con PCA.

♦ Puedes ajustar **n_neighbors** en **KNeighborsRegressor(n_neighbors=k)** para mejorar el modelo. 🚀

ValueError Traceback (most recent call last) File

~\anaconda3\Lib\site-packages\matplotlib\axes_axes.py:4618, in
Axes._parse_scatter_color_args(c, edgecolors, kwargs, xsize, get_next_color_func) 4617
try: # Is 'c' acceptable as PathCollection facecolors? -> 4618 colors =
mcolors.to_rgba_array(c) 4619 except (TypeError, ValueError) as err:

File ~\anaconda3\Lib\site-packages\matplotlib\colors.py:512, in to_rgba_array(c, alpha) 511
else: --> 512 rgba = np.array([to_rgba(cc) for cc in c]) 514 if alpha is not None:

File ~\anaconda3\Lib\site-packages\matplotlib\colors.py:314, in to_rgba(c, alpha) 313 if rgba
is None: # Suppress exception chaining of cache lookup failure. --> 314 rgba =
_to_rgba_no_colorcycle(c, alpha) 315 try:

El error se debe a que **c=y** en **plt.scatter()** espera un array numérico, pero si **y** es una serie de pandas, a veces puede tener problemas de formato.

🔧 Solución:

Asegúrate de convertir **y** en un array NumPy antes de pasarlo a **plt.scatter()**:

Python

```
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=np.array(y),  
cmap='coolwarm', edgecolors='k')
```

📌 Explicación:

- **np.array(y)**: Convierte **y** en un array NumPy, evitando problemas con estructuras de Pandas.

- `cmap='coolwarm'`: Especifica un mapa de colores adecuado.
- `edgecolors='k'`: Evita advertencias en `matplotlib`.

Prueba esta solución y dime si sigue el error. 🚀

porque usas `X_pca` para entrenar la regresión con kNN?

Uso `X_pca` para entrenar la regresión con kNN porque estamos aplicando **PCA (Análisis de Componentes Principales)** como un paso previo para reducir la dimensionalidad del dataset antes de entrenar el modelo.

📌 ¿Por qué aplicar PCA antes de kNN?

- 1 **Reducción de dimensionalidad**: Si el dataset tiene muchas variables, **PCA ayuda a reducir la cantidad de características**, manteniendo la información más relevante.
 - 2 **Evita el "curse of dimensionality"**: kNN es muy sensible a la cantidad de dimensiones. Demasiadas características pueden hacer que las distancias entre puntos sean menos significativas.
 - 3 **Mejora la eficiencia**: kNN busca vecinos más rápido en un espacio con menos dimensiones.
 - 4 **Reduce el ruido**: Si hay características poco informativas o redundantes, PCA las elimina automáticamente.
-

📌 ¿Se debe usar PCA siempre antes de kNN?

No siempre. PCA solo es útil si el dataset tiene muchas características o si hay colinealidad entre variables. Si el dataset es pequeño y ya está optimizado, **puede ser mejor entrenar kNN sin PCA**.

- ♦ Si quieres probar kNN sin PCA, cambia esta línea:

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

Así estarías usando **los datos originales escalados en lugar de la versión reducida con PCA**.

¿Quieres comparar los resultados con y sin PCA? 🚀

para una regresión de ciertas características de vino en la matriz X y su calidad en Y, lo ves necesario?

Para el dataset de **Wine Quality**, no veo estrictamente necesario aplicar PCA antes de entrenar un modelo de **kNN Regression**.

📌 ¿Por qué PCA no es necesario en este caso?

- 1️⃣ **Pocas características (11 en total)** → El dataset no es demasiado grande en dimensiones, por lo que **kNN no sufre del "curse of dimensionality"**.
 - 2️⃣ **No hay características altamente redundantes** → Si bien algunas variables pueden estar correlacionadas (por ejemplo, **total sulfur dioxide** y **free sulfur dioxide**), la regresión kNN puede manejarlas sin problemas.
 - 3️⃣ **PCA puede hacer que pierdas interpretabilidad** → Las componentes principales **no tienen un significado físico claro** (como **acidity** o **alcohol**), lo que dificulta la interpretación de los resultados.
 - 4️⃣ **kNN ya es un método basado en distancia** → PCA transforma los datos en una nueva base, lo que puede cambiar la manera en que kNN mide las distancias entre puntos.
-

📌 Cuándo podría ser útil PCA para Wine Quality

- ✅ **Si el dataset tuviera muchas más variables** → Si en vez de 11 variables hubiera 50 o 100, PCA podría ayudar a reducir la dimensionalidad.
 - ✅ **Si hubiera alta correlación entre muchas características** → En ese caso, PCA ayudaría a evitar que el modelo se "distriga" con información redundante.
 - ✅ **Si se quiere hacer una comparación de modelos** → Puedes probar con y sin PCA y comparar el rendimiento.
-

📌 Conclusión: ¿Usar PCA o no?

- ♦ **Lo más recomendable es usar directamente kNN sin PCA en este dataset.**
- ♦ **Prueba con y sin PCA y compara los resultados** para verificar si realmente mejora el modelo.

Si quieres, te puedo ayudar a comparar ambos enfoques. 🚀

dame un ejemplo de características que puedan tener correlación

Un ejemplo de características que pueden tener **alta correlación** en el dataset de **Wine Quality** son:

♦ **Diòxid de sofre total (`total_sulfur_dioxide`) i Diòxid de sofre lliure (`free_sulfur_dioxide`)** → Com que el **diòxid de sofre lliure** forma part del **diòxid de sofre total**, és molt probable que estiguin fortament correlacionats.

♦ **Densitat (`density`) i Sucre residual (`residual_sugar`)** → A mesura que augmenta la quantitat de sucre residual en el vi, també pot augmentar la seva densitat.

♦ **Acidesa fixa (`fixed_acidity`) i Acidesa cítrica (`citric_acid`)** → Alguns tipus de vins amb més acidesa fixa també poden tenir nivells més alts d'àcid cítric, cosa que pot generar una correlació entre aquestes dues variables.

♦ **Sulfats (`sulphates`) i Diòxid de sofre total (`total_sulfur_dioxide`)** → Ambdós són utilitzats com a conservants en el vi i poden estar relacionats.

Si vols, podem calcular la **matriu de correlació** del dataset i veure exactament quines variables estan més relacionades! 🚀

```
Aixo ja ho hem hagut de arribar a fer en altres projectes, pero per 11 dimensions que te el dataset no ho veiem necessari

[118]: #Dividir en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

[116]: def regression(clf, tuning=False):
    clf.fit(X_train, y_train)
    if tuning:
        print('best pars:', clf.best_params_)
    preds = pd.Series(clf.predict(X_test), name='preds')
    reals = pd.Series(y_test.to_list(), name='target')
    print('mse:', mean_absolute_error(reals, preds))
    print('R2:', r2_score(reals, preds))

[118]: %%time
pars = {'n_neighbors': [1, 3, 5, 7, 11], 'weights': ['uniform', 'distance']}
regression(GridSearchCV(KNeighborsRegressor(), pars), True)

best pars: {'n_neighbors': 11, 'weights': 'distance'}

-----
ValueError                                Traceback (most recent call last)
File <timed exec>:2

Cell In[116], line 5, in regression(clf, tuning)
      3 if tuning:
      4     print('best pars:', clf.best_params_)
----> 5     preds = pd.Series(clf.predict(X_test), name='preds')
      6     reals = pd.Series(y_test.to_list(), name='target')
      7     print('mse:', mean_absolute_error(reals, preds))

File ~\anaconda3\lib\site-packages\pandas\core\series.py:584, in Series.__init__(self, data, index, dtype, name, copy, fastpath)
    582     data = data.copy()
    583 else:
--> 584     data = sanitize_array(data, index, dtype, copy)
    586     manager = _get_option("mode.data_manager", silent=True)
    587     if manager == "block":

File ~\anaconda3\lib\site-packages\pandas\core\construction.py:659, in sanitize_array(data, index, dtype, copy, allow_2d)
    656     subarr = cast(np.ndarray, subarr)
    657     subarr = maybe_infer_to_datetimelike(subarr)
--> 659     subarr = _sanitize_ndim(subarr, data, dtype, index, allow_2d=allow_2d)
    661     if isinstance(subarr, np.ndarray):
    662         # at this point we should have dtype be None or subarr.dtype == dtype
    663         dtype = cast(np.dtype, dtype)

File ~\anaconda3\lib\site-packages\pandas\core\construction.py:718, in _sanitize_ndim(result, data, dtype, index, allow_2d)
    716     if allow_2d:
    717         return result
--> 718     raise ValueError(
    719         f"Data must be 1-dimensional, got ndarray of shape {data.shape} instead"
    720     )
    721 if is_object_dtype(dtype) and isinstance(dtype, ExtensionDtype):
    722     # i.e. NumPyEADtype("O")
    724     result = com.asarray_tuplesafe(data, dtype=np.dtype("object"))

ValueError: Data must be 1-dimensional, got ndarray of shape (1300, 1) instead
```

El error que aparece en tu código es:

"ValueError: Data must be 1-dimensional, got ndarray of shape (1300, 1) instead"

🔍 ¿Qué está pasando?

El problema ocurre en esta línea:

```
Python
preds = pd.Series(clf.predict(X_test), name='preds')
```

o

```
Python
reals = pd.Series(y_test.to_list(), name='reals')
```

🔑 **La razón** es que `clf.predict(X_test)` probablemente devuelve un array **bidimensional** con forma `(1300, 1)`, pero `pd.Series()` espera un array unidimensional `(1300,)`.

🔧 Solución

Convierte `preds` y `reals` en vectores 1D antes de crear las series de Pandas:

```
Python
preds = pd.Series(clf.predict(X_test).flatten(), name='preds')
reals = pd.Series(y_test.values.flatten(), name='reals')
```

O simplemente usa `.ravel()` en vez de `.flatten()`:

```
Python
preds = pd.Series(clf.predict(X_test).ravel(), name='preds')
reals = pd.Series(y_test.values.ravel(), name='reals')
```

♦ **Explicación:** `.flatten()` y `.ravel()` transforman un array 2D en un array 1D, lo que evita el error.

Prueba esto y dime si sigue el problema 🚀

NameError Traceback (most recent call last) File :2

```
Cell In[124], line 5, in regression(clf, tuning)
3 if tuning:
4     print('best pars:', clf.best_params_)
----> 5 preds = pd.Series(clf.predict(X_test),flatten(),name='preds')
6     reals = pd.Series(y_test.to_list(), name='target')
7     print('mse:', mean_absolute_error(reals,preds))
```

NameError: name 'flatten' is not defined

Tienes un error de sintaxis en esta línea:

```
Python
preds = pd.Series(clf.predict(X_test),flatten(),name='preds')
```

● **Error:** `flatten()` no es una función separada, sino un **método de numpy arrays**, y debe aplicarse al resultado de `clf.predict(X_test)`.

✓ Solución correcta

Debes escribirlo así:

```
Python
preds = pd.Series(clf.predict(X_test).flatten(), name='preds')
```

o usando `.ravel()`, que es más eficiente:

```
Python
preds = pd.Series(clf.predict(X_test).ravel(), name='preds')
```

Haz lo mismo con `y_test`:

```
Python
reals = pd.Series(y_test.values.ravel(), name='target')
```

♦ Explicación:

- `clf.predict(X_test)` devuelve un array con forma `(1300, 1)`, pero `pd.Series()` espera una estructura **1D** `(1300,)`.
- `.flatten()` o `.ravel()` convierten un array 2D en 1D.

Prueba esto y dime si te funciona 🚀

AttributeError Traceback (most recent call last) in ?() 1 'Could not get source, probably due dynamically evaluated source code.'

```
~\AppData\Local\Temp\ipykernel_10032\3439217702.py in ?(clf, tuning) 2 clf.fit(X_train, y_train) 3 if tuning: 4 print('best pars:', clf.best_params_) 5 preds = pd.Series(clf.predict(X_test).flatten(),name='preds') ----> 6 reals = pd.Series(y_test.to_list(),
```

```
name='target') 7 print('mse:', mean_absolute_error(reals,preds)) 8 print('R2:',  
r2_score(reals,preds))
```

~\anaconda3\Lib\site-packages\pandas\core\generic.py in ?(self, name) 6295 and name not
in self._accessors 6296 and self._info_axis._can_hold_identifiers_and_holds_name(name)
6297): 6298 return self[name] -> 6299 return object.**getattr**(self, name)

AttributeError: 'DataFrame' object has no attribute 'to_list'

El error se debe a que `y_test` es un **DataFrame** en lugar de una **Serie de Pandas**.

Error:

Python

```
reals = pd.Series(y_test.to_list(), name='target')
```

 **Problema:** `y_test.to_list()` no es válido para un **DataFrame** (es para Series).

Solución correcta:

Si `y_test` es un **DataFrame** con una sola columna, conviértelo en una **Serie** antes de usar `.ravel()`:

Python

```
reals = pd.Series(y_test.squeeze().ravel(), name='target')
```

◆ Explicación:

- `.squeeze()` convierte un DataFrame de una sola columna en una Serie.
 - `.ravel()` transforma el array en 1D (por si acaso).
-

Solución alternativa (mejor enfoque)

Si `y_test` ya es una **Serie** y no un DataFrame, puedes simplemente hacer:

Python

```
reals = pd.Series(y_test.values.ravel(), name='target')
```

 **Prueba esto y dime si funciona!** 

SE ACABO CHATGPT GRATIS