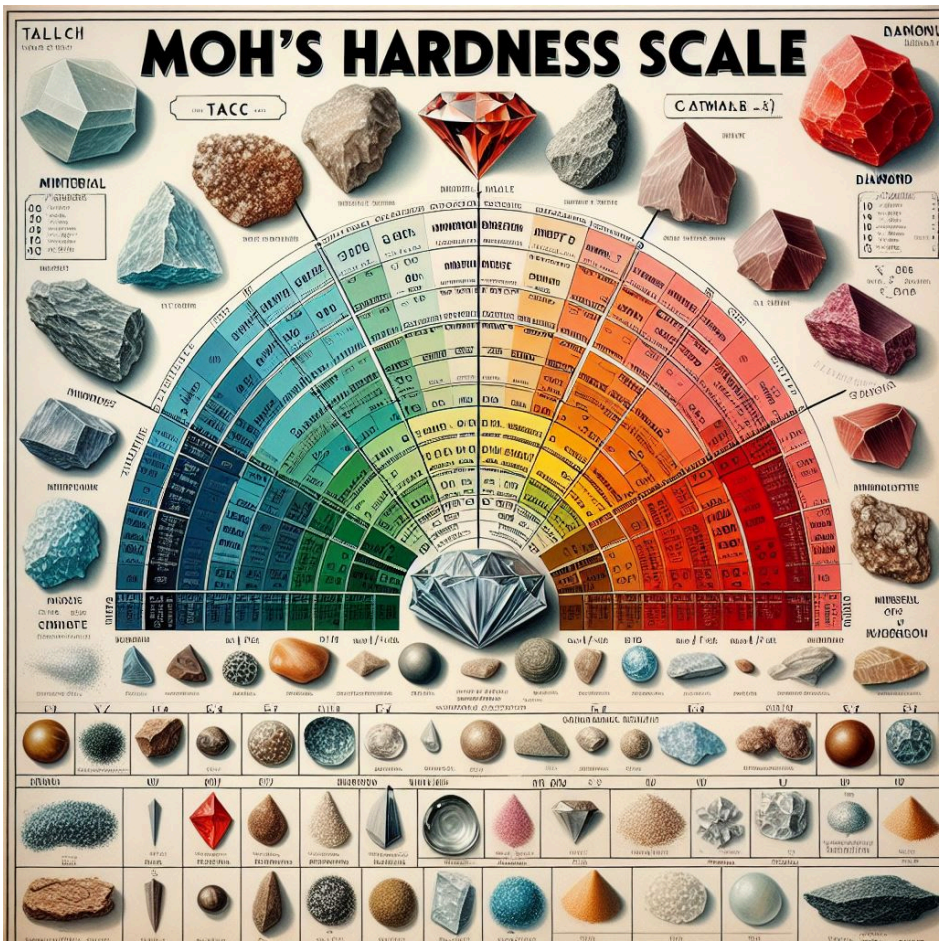
PREDICCIÓN DE LA DUREZA DE MOH



Néstor Batista Díaz

28 de Enero del 2024

ÍNDICE

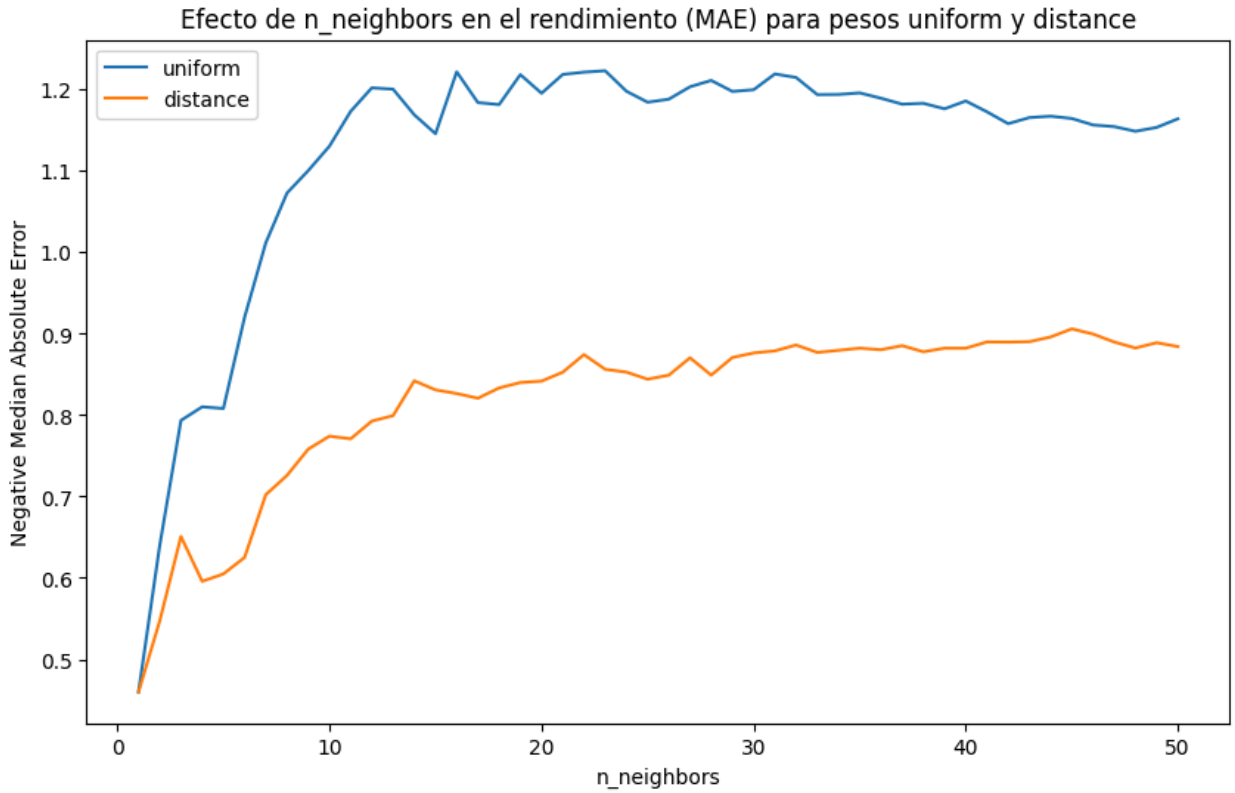
ENTRENANDO CON KNN

```
# Define el espacio de búsqueda de parámetros
param_grid = {'n_neighbors': np.arange(1, 51), 'weights': ['uniform', 'distance']}

# Crea el modelo KNN
knn = KNeighborsRegressor()

# Realiza la búsqueda de hiperparámetros utilizando Grid Search
grid_search_knn = GridSearchCV(knn, param_grid, cv=skf, scoring="neg_median_absolute_error")
grid_search_knn.fit(X_train, y_train)

# Obtiene los resultados
resultados = grid_search_knn.cv_results_
mejores_parametros = grid_search_knn.best_params_

# Extrae los resultados para 'uniform' y 'distance'
scores_uniform = -resultados['mean_test_score'][resultados['param_weights'] == 'uniform']
scores_distance = -resultados['mean_test_score'][resultados['param_weights'] == 'distance']
parametros = resultados['param_n_neighbors'][resultados['param_weights'] == 'uniform']

# Imprime los mejores parámetros encontrados
print("Mejores Parámetros:", mejores_parametros, "MAE:", -grid_search_knn.best_score_)

# Dibuja la gráfica
plt.figure(figsize=(10, 6))
plt.plot(parametros, scores_uniform, label='uniform')
plt.plot(parametros, scores_distance, label='distance')
plt.title('Efecto de n_neighbors en el rendimiento (MAE) para pesos uniform y distance')
plt.xlabel('n_neighbors')
plt.ylabel('Negative Median Absolute Error')
plt.legend()
plt.show()

✓ 1.3s
Mejores Parámetros: {'n_neighbors': 1, 'weights': 'uniform'} MAE: 0.45999999999999996
```

Utilizamos *GridSearchCV* para elegir los mejores hyperparametros para entrenar nuestro modelo.

Como podemos observar los mejores hyperparametros son ‘n_neighbors’:1 y ‘weights’: ‘uniform’.

Esto significa que por cada dato, mirara el dato más cercano y los pesos serán uniformes, es decir que no importa cuan lejos o cerca esté el vecino.



Aquí podemos observar que en general es mejor ajustar los pesos en ‘distance’ que en ‘uniform’ para darle prioridad a la distancia entre vecinos.

```
results = pd.DataFrame(resultados)
results = results.sort_values(by='rank_test_score')
results.head()
```

✓ ✓ 0.4s Python

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	param_weights	params	split0_test_score	split1_test_score	split2_test_score	split3_test_sc
0	0.001103	2.049457e-04	0.001200	0.000401	1	uniform	{'n_neighbors': 1, 'weights': 'uniform'}	-0.300000	-0.500000	-0.500000	-0.500
1	0.001002	6.342206e-04	0.001101	0.000202	1	distance	{'n_neighbors': 1, 'weights': 'distance'}	-0.300000	-0.500000	-0.500000	-0.500
3	0.000802	4.009781e-04	0.001201	0.000399	2	distance	{'n_neighbors': 2, 'weights': 'distance'}	-0.506774	-0.605742	-0.653143	-0.470
7	0.001201	4.003808e-04	0.001070	0.000142	4	distance	{'n_neighbors': 4, 'weights': 'distance'}	-0.612532	-0.700000	-0.612479	-0.500
9	0.001000	7.478899e-07	0.001228	0.000456	5	distance	{'n_neighbors': 5, 'weights': 'distance'}	-0.627254	-0.691354	-0.663203	-0.500

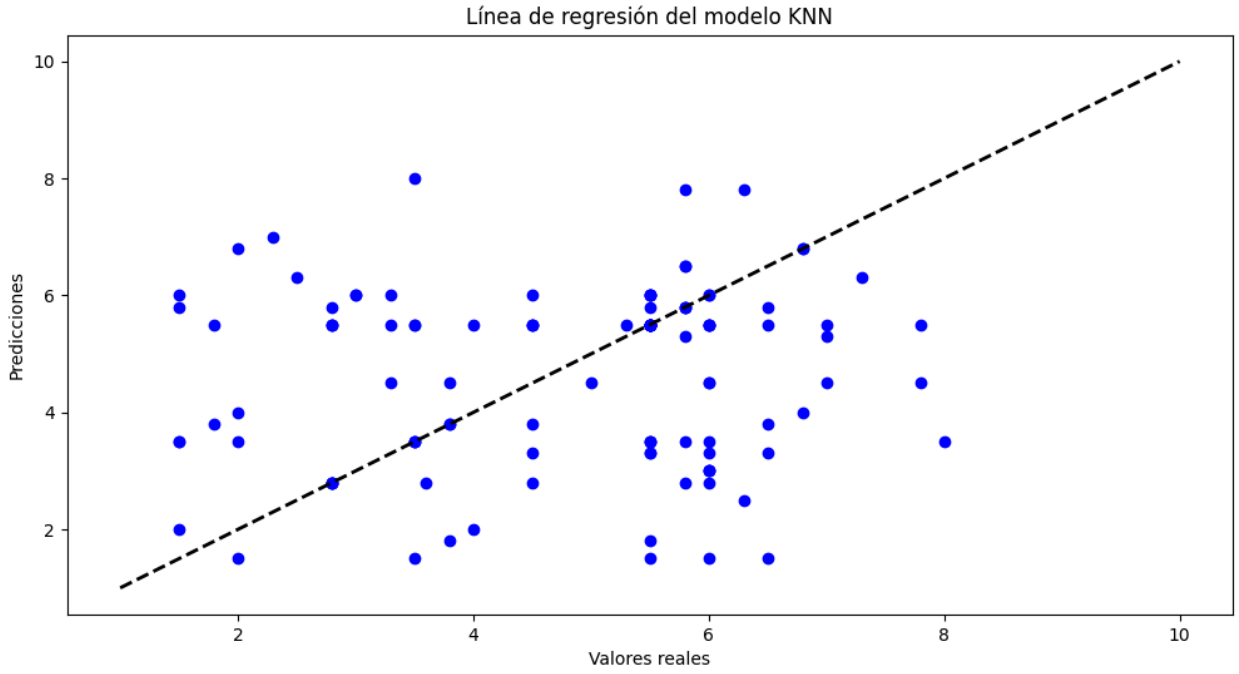
Por otro lado, en la tabla generada por *GridSearchCV* del entrenamiento del modelo para ‘n_neighbors’ = 1, da igual que peso uses.

```python
model_knn = KNeighborsRegressor(**mejores_parametros)

# Use cross_val_score with the pipeline
kn_cv_scores = cross_val_score(model_knn, X_train, y_train, cv=skf, n_jobs=-1, scoring="neg_median_absolute_error")


print("Scores: ",-kn_cv_scores)
print(f'Mediana del error absoluto (MAE) con el modelo KNN: {-kn_cv_scores.mean()}')
```

✓ 0.5s

```
Scores:  [0.3 0.5 0.5 0.5 0.5]
Mediana del error absoluto (MAE) con el modelo KNN: 0.45999999999999996
```

Aquí compruebo mediante *cross_val_score* que el entrenamiento anterior es correcto y la mediana del error absoluto (MAE) es 0.46.



INVESTIGACIÓN SOBRE TÉCNICAS DE ENTRENAMIENTO BASADAS EN ÁRBOLES

He encontrado varios modelos de entrenamiento basados en árboles de regresión. Para este proyecto he seleccionado 3: `RandomForestRegressor`, `GradientBoostingRegressor` y `XGBoost`.

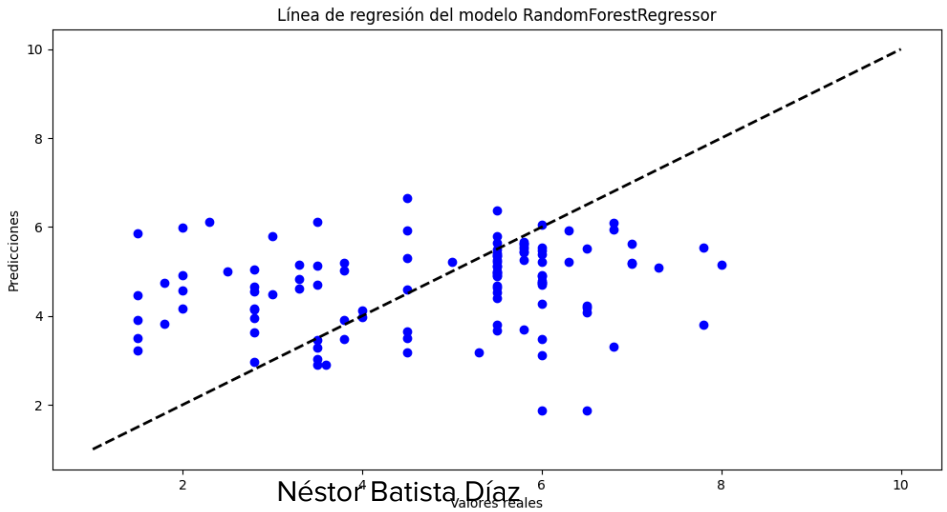
RandomForestRegressor

El Random Forest es un algoritmo de aprendizaje supervisado que se basa en la construcción de múltiples árboles de decisión durante el entrenamiento y la combinación de sus predicciones para obtener un resultado final.

```
# Definir los parámetros a buscar
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Crear el modelo de RandomForestRegressor
rf = RandomForestRegressor(random_state=42)

# Realiza la búsqueda de hiperparámetros utilizando Grid Search
grid_search_rfg = GridSearchCV(rf, param_grid, cv=skf, scoring="neg_median_absolute_error")
grid_search_rfg.fit(X_train, y_train)

# Obtiene los resultados
resultados = grid_search_rfg.cv_results_
mejores_parametros = grid_search_rfg.best_params_

# Imprime los mejores parámetros encontrados
print("Mejores Parámetros:", mejores_parametros, "MAE:", -grid_search_rfg.best_score_)
```

48] ✓ 1m 43.2s

```
Mejores Parámetros: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100} MAE: 0.48858523809523985
```

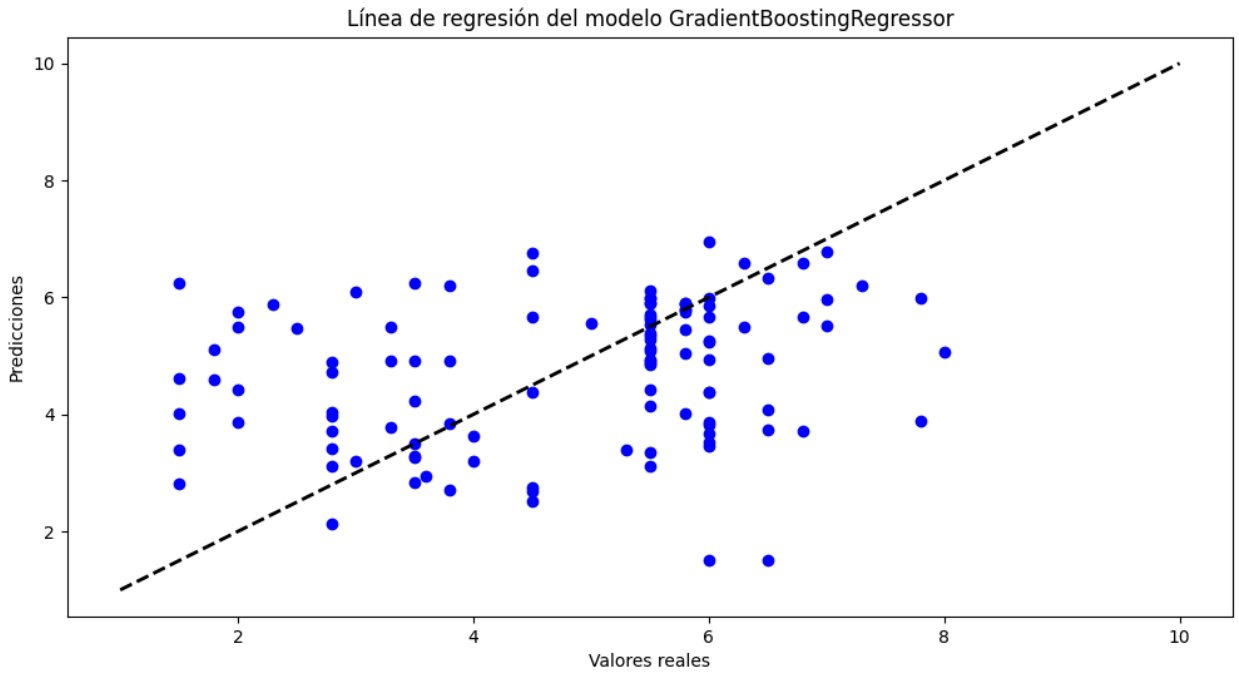


GradientBoostingRegressor

Este es un algoritmo que construye árboles de decisión de forma secuencial, tratando de corregir los errores de predicción de los árboles anteriores.

```python
# Definir los parámetros a buscar
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3]
}

# Crear el modelo de GradientBoostingRegressor
gbm = GradientBoostingRegressor(random_state=42)

# Realizar la búsqueda en cuadrícula
grid_search_gbr = GridSearchCV(gbm, param_grid, cv=skf, scoring="neg_median_absolute_error")
grid_search_gbr.fit(X_train, y_train)

# Obtener los mejores parámetros
best_params = grid_search_gbr.best_params_
print("Mejores parámetros:", best_params, "MAE:", -grid_search_gbr.best_score_)
```

```
✓ 2m 17.8s
Mejores parámetros: {'learning_rate': 0.1, 'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 200} MAE: 0.46819381764699913
```



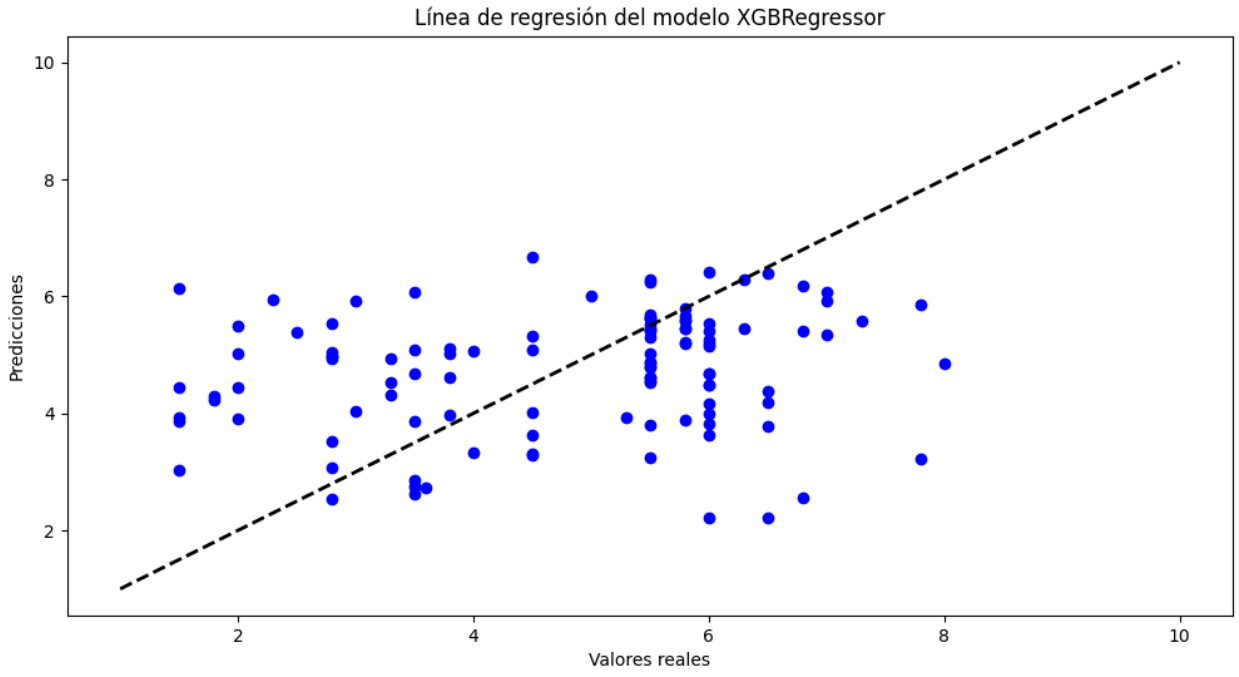
XGBoost

Extreme Gradient Boosting (XGBoost) es una implementación optimizada de algoritmos de boosting que ha demostrado ser muy efectiva en competiciones de ciencia de datos.

```python
import xgboost as xgb

# Definir los parámetros a buscar
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_child_weight': [1, 2, 3],
    'gamma': [0.1, 0.2, 0.3]
}

# Crear el modelo de XGBoost
xg_reg = xgb.XGBRegressor(objective ='reg:squarederror', random_state=42)

# Realizar la búsqueda en cuadrícula
grid_search_xgb = GridSearchCV(estimator=xg_reg, param_grid=param_grid, cv=skf, scoring="neg_median_absolute_error")
grid_search_xgb.fit(X_train, y_train)

# Obtener los mejores parámetros
mejores_parametros = grid_search_xgb.best_params_
print("Mejores parámetros:", best_params, "MAE:", -grid_search_xgb.best_score_)
```

✓ 52.9s

```
Mejores parámetros: {'learning_rate': 0.1, 'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 200} MAE: 0.5102458477020264
```



SELECCIONAR TÉCNICA

```
    skf = KFold(n_splits = 5)

    model_LinearRegression_con_cross_val = LinearRegression()

    predicciones_con_cross_val = cross_val_score(model_LinearRegression_con_cross_val, X_train, y_train, scoring=scorer, cv=skf, n_jobs=-1)
    cv_mae_linearRegression = -np.mean(predicciones_con_cross_val)

    print(f'Error cuadrático medio con LinearRegression con cross-validation: {cv_mae_linearRegression}')
[64] ✓ 1.6s
...  Error cuadrático medio con LinearRegression con cross-validation: 1.1066564581346658
```

```
    model_knn = KNeighborsRegressor(**mejores_parametros)

    Use cross_val_score with the pipeline
    kn_cv_scores = cross_val_score(model_knn, X_train, y_train, cv=skf, n_jobs=-1, scoring="neg_median_absolute_error")


    print("Scores: ",-kn_cv_scores)
    print(f'Mediana del error absoluto (MAE) con el modelo KNN: {-kn_cv_scores.mean()}')

6] ✓ 0.5s
  Scores:  [0.3 0.5 0.5 0.5 0.5]
  Mediana del error absoluto (MAE) con el modelo KNN: 0.45999999999999996
```

```
    # Crear el modelo de Random Forest
    model_randomForestRegressor = RandomForestRegressor(**mejores_parametros, random_state=42)

    # Realizar la validación cruzada con 5 particiones (k=5)
    mae_scores = cross_val_score(model_randomForestRegressor, X_train, y_train, cv=skf, scoring="neg_median_absolute_error")

    print("Scores: ",-mae_scores)
    print('Mediana del error absoluto (MAE) con el modelo RandomForestRegressor:', -mae_scores.mean())

50] ✓ 0.6s
  Scores:  [0.47859286 0.39466667 0.614      0.525      0.43066667]
  Mediana del error absoluto (MAE) con el modelo RandomForestRegressor: 0.48858523809523985
```

```
    # Entrenar el modelo con los mejores parámetros
    model_gradientBoostingRegressor = GradientBoostingRegressor(**best_params, random_state=42)

    mae_scores = cross_val_score(model_gradientBoostingRegressor, X_train, y_train, cv=skf, scoring="neg_median_absolute_error")

    print("Scores: ",-mae_scores)
    print('Mediana del error absoluto (MAE) con el modelo GradientBoostingRegressor:', -mae_scores.mean())

44] ✓ 0.7s
  Scores:  [0.42668344 0.43222764 0.51690732 0.41127252 0.55387817]
  Mediana del error absoluto (MAE) con el modelo GradientBoostingRegressor: 0.46819381764699913
                                                    + Código   + Markdown
```

```
# Entrenar el modelo con los mejores parámetros
model_xgb = xgb.XGBRegressor(objective ='reg:squarederror', **mejores_parametros, random_state=42)

mae_scores = cross_val_score(model_xgb, X_train, y_train, cv=skf, scoring="neg_median_absolute_error")

# Evaluar el modelo
print("Scores: ",-mae_scores)
print('Mediana del error absoluto (MAE) con el modelo XGBRegressor:', -mae_scores.mean())
```

✓ 0.1s

```
Scores:  [0.49229193 0.46146994 0.61470389 0.49304295 0.48972054]
Mediana del error absoluto (MAE) con el modelo XGBRegressor: 0.5102458477020264
```

En primer lugar, el modelo LinearRegression queda descartado porque tiene un error muy grande en comparación al resto. Los otros 4, tienen un MAE bastante parecido, pero he decidido escoger KNN, no solo porque tiene la MAE más pequeña sino porque también es uno de los modelos más utilizados.

EXPORTAR FICHERO .pkl

Exportar modelo en .pkl

```
# Exportar el modelo a un fichero
joblib.dump(grid_search_knn,'mohs_hardness_knn_modelo_entrenado.pkl')
```

✓ 0.4s

```
['mohs_hardness_knn_modelo_entrenado.pkl']
```

IMPORTAR FICHERO .pkl

```
modelo_knn_entrenado = KNeighborsRegressor()
modelo_knn_entrenado = joblib.load('mohs_hardness_knn_modelo_entrenado.pkl')
-modelo_knn_entrenado.score(X_test, y_test)
```

✓ 0.8s

```
0.3999999999999999
```

PREDICCIÓN

```
test = pd.read_csv("test.csv")
test
```

✓ 0.5s

	id	allelectrons_Total	density_Total	atomicweight_Average	Hardness
0	1	40.0	3.18	18.99	4.0
1	2	164.0	19.92	32.00	6.5
2	3	605.2	61.36	27.92	4.5
3	4	30.0	2.75	60.08	7.0
4	5	6.0	3.50	14.00	10.0
5	6	70.0	5.82	20.10	7.0

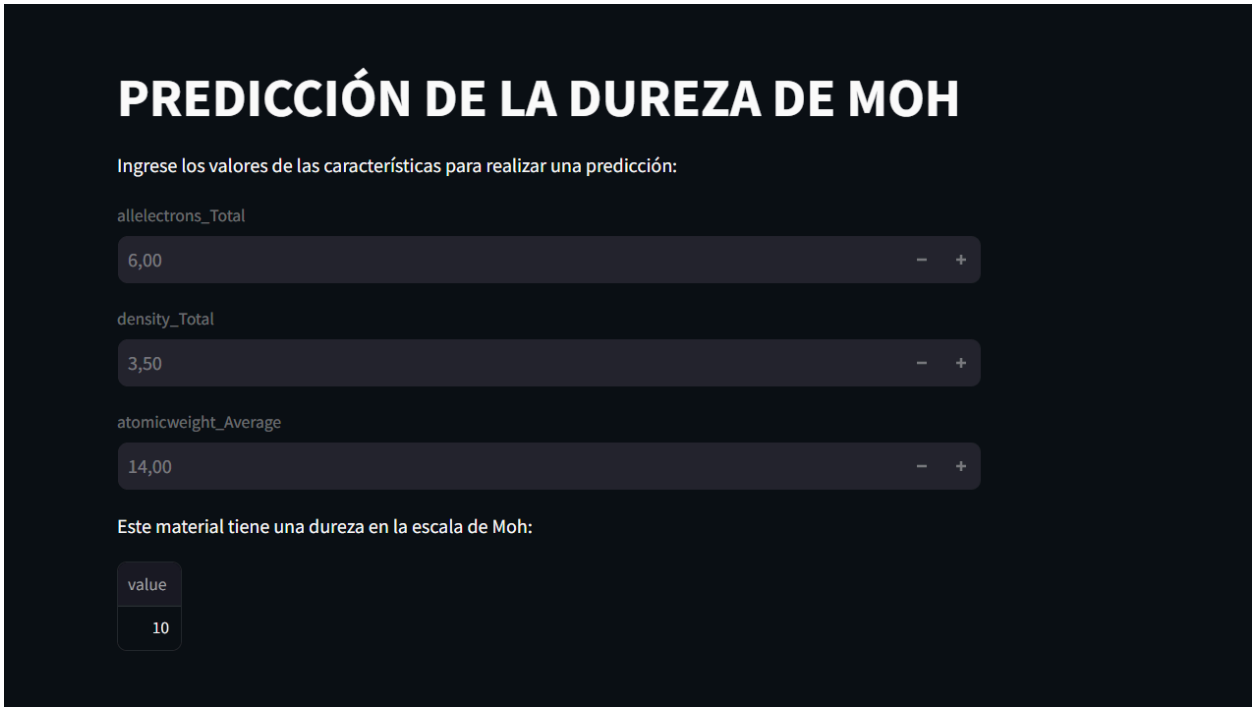
```
pred = modelo_knn_entrenado.predict(test[selected_features])

output = pd.DataFrame({ 'id': test['id'], 'hardness_pred': pred, 'real_hardness': test["Hardness"]})
output
```

✓ 0.5s

	id	hardness_pred	real_hardness
0	1	4.0	4.0
1	2	5.3	6.5
2	3	5.5	4.5
3	4	4.0	7.0
4	5	10.0	10.0
5	6	7.8	7.0

STREAMLIT

```python
import streamlit as st
import joblib
import numpy as np

# Cargar el modelo entrenado
modelo_knn_entrenado = joblib.load('mohs_hardness_knn_modelo_entrenado.pkl')

# Crear la interfaz de usuario
st.title("PREDICCIÓN DE LA DUREZA DE MOH")
st.write('Ingrese los valores de las características para realizar una predicción:')

allElectronsTotal = st.number_input('allelectrons_Total')
densityTotal = st.number_input('density_Total')
atomicweightAverage = st.number_input('atomicweight_Average')

# Realizar la predicción con el modelo
input_data = np.array([[allElectronsTotal, densityTotal, atomicweightAverage]])
prediction = modelo_knn_entrenado.predict(input_data)

st.write('Este material tiene una dureza en la escala de Moh: ', prediction)
```



CONCLUSIONES

El modelo final presenta algunos fallos en su predicción, pero se acerca bastante al valor real.

Habría que seguir investigando diferentes modelos y probar con otros parámetros para ver cómo afecta esto al modelo.

REFERENCIAS

- Dataset
- Cuaderno de ayuda
- GitHub
- GridSearchCV
- GradientBoostingRegressor
- XGBoost
- Streamlit