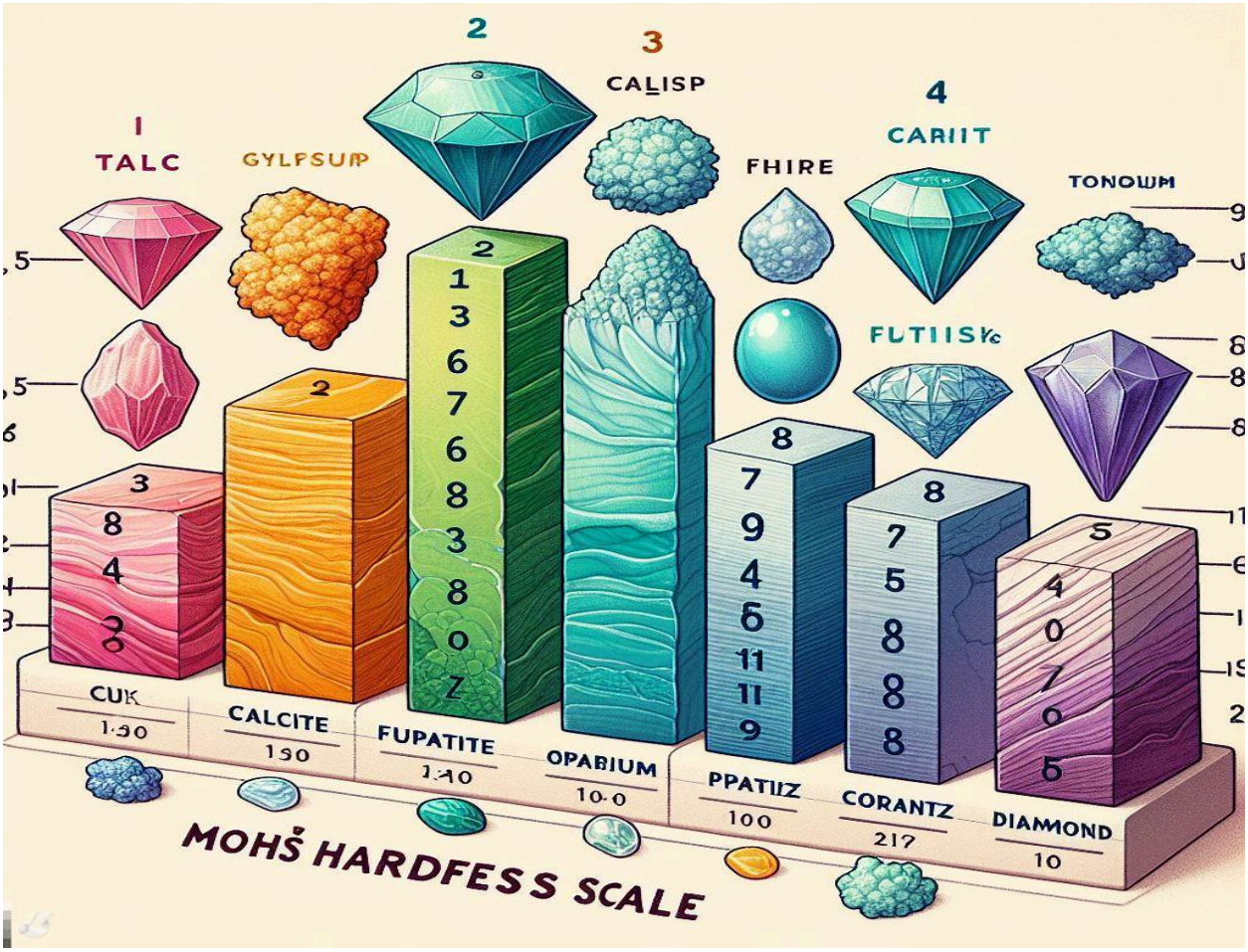
Predicción de la dureza de Moh



Néstor Batista Díaz

18 de enero del 2024

ÍNDICE

DATASET

El dataset elegido para este trabajo está sacado de “**Prediction of Mohs Hardness with Machine Learning”** de kaggle**.** Como en este trabajo el objetivo es predecir la dureza de Moh utilizaremos un modelo de regresión, para este caso he elegido “**LinearRegression”** de la librería **sklearn**.

DESCRIPCIÓN

La dureza, o valor cuantitativo de la resistencia a la deformación permanente o plástica, desempeña un papel muy crucial en el diseño de materiales en muchas aplicaciones, como los revestimientos cerámicos y los abrasivos. El ensayo de dureza es un método especialmente útil por ser no destructivo y sencillo de aplicar para medir las propiedades plásticas de un material. En este estudio, propuse un enfoque de aprendizaje automático o estadístico para predecir la dureza de materiales naturales, que integra características atómicas y electrónicas de la composición directamente a través de una amplia variedad de composiciones minerales y sistemas cristalinos. En primer lugar, se extrajeron de la composición las características atómicas y electrónicas, como los radios de van der Waals y covalentes, así como el número de electrones de valencia.

En este estudio, el autor entrenó un conjunto de clasificadores para comprender si las características de composición pueden utilizarse para predecir la dureza Moh de minerales de diferentes espacios químicos, estructuras cristalinas y clases de cristales. El conjunto de datos para entrenar y probar los modelos de clasificación utilizados en este estudio se originó a partir de datos experimentales de dureza Moh, sus clases cristalinas y composiciones químicas de minerales naturales recogidos en el “Physical and Optical Properties of Minerals CRC Handbook of Chemistry and Physics” y la base de datos de estructuras cristalinas del “American Mineralogist”. La base de datos contiene 369 minerales con nombres exclusivos. Debido a la presencia de múltiples combinaciones de composición para minerales con el mismo nombre, el primer paso consistió en realizar permutaciones de composición en estos minerales. Se obtuvo así una base de datos de 622 minerales con composiciones únicas: 210 monoclínicos, 96 romboédricos, 89 hexagonales, 80 tetragonales, 73 cúbicos, 50 ortorrómbicos, 22 triclínicos, 1 trigonal y 1 estructura amorfa. Se recopiló un conjunto de datos independiente para validar el rendimiento del modelo. El conjunto de datos de validación contiene la composición, la estructura cristalina y los valores de dureza Moh de 51 monocristales sintéticos publicados en la literatura. El conjunto de datos de validación incluye 15 estructuras cristalinas monoclínicas, 7 tetragonales, 7 hexagonales, 6 ortorrómbicas, 4 cúbicas y 3 romboédricas.

En este estudio, el autor construyó una base de datos de descriptores de rasgos composicionales que caracterizan materiales naturales obtenidos directamente del “Physical and Optical Properties of Minerals CRC Handbook45”. Este amplio conjunto de datos basados en la composición nos permite entrenar modelos capaces de predecir la dureza en una amplia variedad de composiciones minerales y clases de cristales. Cada material, tanto en los conjuntos de datos de minerales naturales como en los de monocristales artificiales, estaba representado por 11 descriptores atómicos. Las características elementales son el número de electrones, el número de electrones de valencia, el número atómico, la electronegatividad de Pauling del estado de oxidación más común, los radios atómicos covalentes, los radios de van der Waals y la energía de ionización del neutro.

PROCESAMIENTO DE DATOS

En este proyecto de kaggle hay dos datasets, para hacer este trabajo he decidido escoger el dataset “Mineral_Dataset_Supplementary_Info.csv” dado que solo contiene datos numéricos y será más fácil tratarlo.

```
df = pd.read_csv("https://raw.githubusercontent.com/Nestorbd/Prediction-of-Mohs-Hardness/master/Mineral_Dataset_Supplementary_Info.csv").drop('Unnamed: 0', axis=1)
target = df.pop("Hardness")
df.insert(df.shape[1], "Hardness", target)
df.shape
```

Python

```
(622, 12)
```

Primero almacenamos el dataset en una variable. Como nuestro objetivo es predecir la dureza trasladamos este valor al final de la matriz.

ANÁLISIS DE DATOS

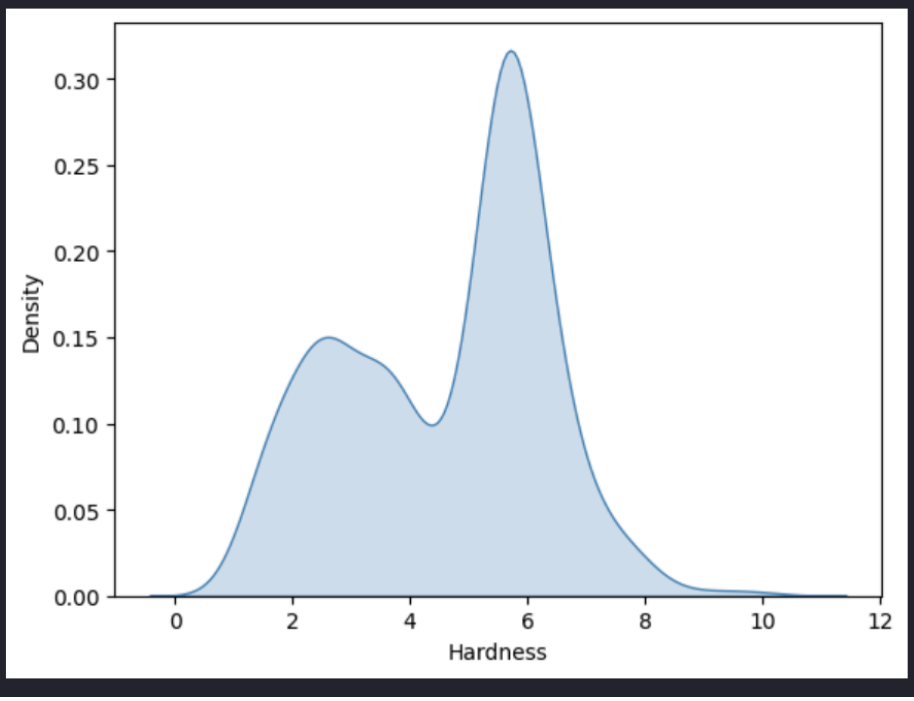
```
df.describe()
```

Python

	allelectrons_Total	density_Total	allelectrons_Average	val_e_Average	atomicweight_Average	ionenergy_Average	el_neg_chi_Average	R_vdw_element_Average	R_cov_element_A
count	622.000000	622.000000	622.000000	622.000000	622.000000	622.000000	622.000000	622.000000	622.
mean	312.895691	27.864836	14.808027	4.419379	32.243577	11.108756	2.626550	1.670637	0.
std	853.331650	39.243940	9.963898	0.807960	24.586056	1.748614	0.402856	0.256487	0.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	98.250000	10.561656	9.371995	4.058333	18.833820	10.961967	2.629196	1.612857	0.
50%	186.500000	20.744196	10.852814	4.636364	22.197615	11.359643	2.708831	1.713333	0.
75%	390.000000	34.824490	17.416667	4.800000	38.852958	11.804828	2.774951	1.763561	0.
max	15300.000000	643.093804	67.000000	6.000000	167.400000	14.163933	3.426381	2.250000	1.

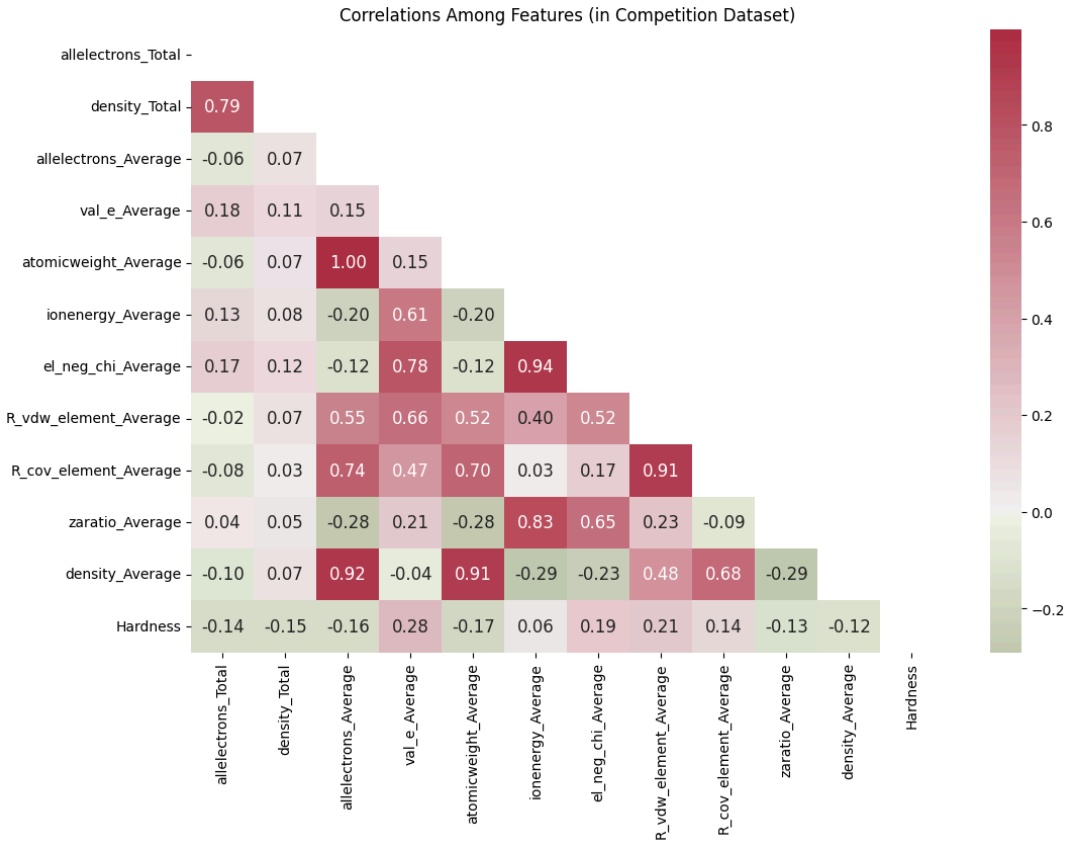
Aquí podemos ver más detalles sobre el dataset elegido.

```
sns.kdeplot(data = df, x = 'Hardness', fill = True, color = 'steelblue');
```



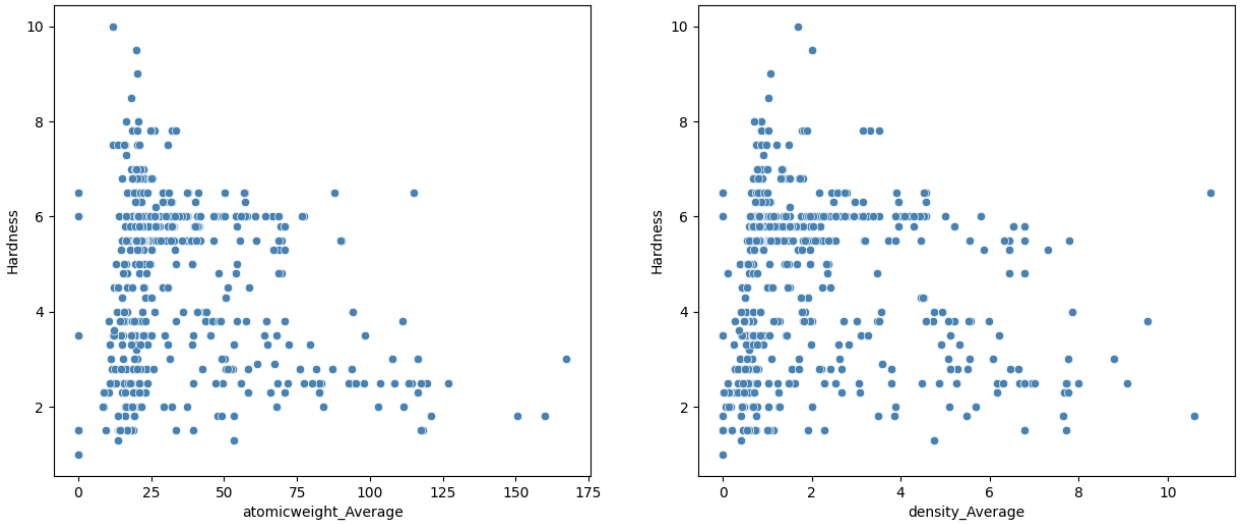
Como podemos ver, este gráfico es multimodal, hay dos picos que representan los valores más comunes.

```
corr_mat_data = df.corr()
data_mask = np.triu(np.ones_like(corr_mat_data, dtype = bool))

cmap = sns.diverging_palette(100, 7, s = 75, l = 40, n = 5, center = 'light', as_cmap = True)

plt.figure(figsize = (12, 8))
sns.heatmap(corr_mat_data, annot = True, cmap = cmap, fmt = '.2f', center = 0,
            annot_kws = {'size': 12}, mask = data_mask).set_title('Correlations Among Features (in Competition Dataset)');
```



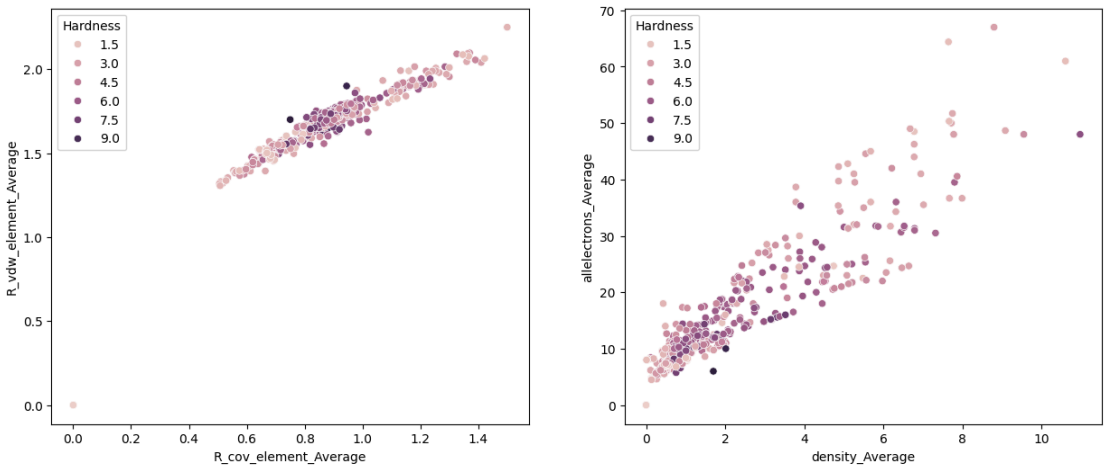
Hacemos una matriz de correlación, con ella podemos ver que hay muchos campos que están relacionados, los más destacados son “allectrons_Average”, “val_e_Average” e “ionenergy_Average”.

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

sns.scatterplot(ax = axes[0], data = df, x = 'atomicweight_Average', y = 'Hardness', color = 'steelblue')
sns.scatterplot(ax = axes[1], data = df, x = 'density_Average', y = 'Hardness', color = 'steelblue');
```



En términos de patrones, no hay mucho que aprender de los gráficos anteriores. Sin embargo, en el panel izquierdo, hay algunos puntos de datos (en la parte inferior derecha) que parecen ser valores atípicos (en “atomicweight_Average”). En el panel derecho, hay algunos puntos de datos (a la derecha) que parecen ser valores atípicos (en “density_Average”).

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

sns.scatterplot(ax = axes[0], data = df, x = 'R_cov_element_Average', y = 'R_vdw_element_Average', hue = 'Hardness')
sns.scatterplot(ax = axes[1], data = df, x = 'density_Average', y = 'allelectrons_Average', hue = 'Hardness');
```



SELECCIÓN DE CARACTERÍSTICAS

Para seleccionar las características hay que tener en cuenta que estamos con un problema de regresión así que en vez de “SelectKBest” vamos a usar “Lasso” para elegir las mejores características, con las cuales vamos a entrenar nuestro modelo.

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split

X = df.drop('Hardness', axis=1)
y = df['Hardness']

# Dividir los datos 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)

# Inicializar el modelo de regresión LASSO
lasso_model = Lasso(alpha=1.0)

# Entrenar el modelo
lasso_model.fit(X_train, y_train)

# Obtener los coeficientes no nulos (características seleccionadas)
selected_features = X.columns[lasso_model.coef_ != 0]

# Mostrar las características seleccionadas
print("Características seleccionadas:", selected_features)
```

```
Características seleccionadas: Index(['allelectrons_Total', 'density_Total', 'atomicweight_Average'], dtype='object')
```

OUTLIERS

Como anteriormente observamos en las gráficas que aparentemente habían valores atípicos vamos a comprobar si es verdad.

```python
df_columns_selected = df[selected_features]
k=1.8
for i,column in enumerate(df_columns_selected,1):
    print(column)
    Q1 = np.quantile(df_columns_selected[column],0.25)
    Q3 = np.quantile(df_columns_selected[column],0.75)
    IQR = Q3 - Q1
    xL=Q1 - k * IQR
    xU=Q3 + k * IQR
    print(f" Banda= [ {xL},{xU}]")
    outlaiers = 0
    for x in  range(len(df_columns_selected)):
            if df_columns_selected[column][x] < xL  or df_columns_selected[column][x]>xU:
                outlaiers +=1
                # print(f" El dato[{x}]={df_columns_selected[column][x]} es un outlier")
    print(f"Hay {outlaiers} outlaiers")
```

```
allelectrons_Total
 Banda= [ -426.9,915.15]
Hay 14 outlaiers
density_Total
 Banda= [ -33.11144520000001,78.49759120000002]
Hay 16 outlaiers
atomicweight_Average
 Banda= [ -17.20062643977206,74.8874043184249]
Hay 44 outlaiers
```

Como podemos observar hay varios valores atípicos en los datos seleccionados. Los eliminaremos para que no interfieran con el entrenamiento de nuestro modelo.

```python
# Calcular el rango intercuartílico (IQR) para cada columna
Q1 = df_columns_selected[selected_features].quantile(0.25)
Q3 = df_columns_selected[selected_features].quantile(0.75)
IQR = Q3 - Q1
xL=Q1 - k * IQR
xU=Q3 + k * IQR

# Definir un filtro para eliminar outliers
filtro_outliers = ~((df_columns_selected[selected_features] < xL) | (df_columns_selected[selected_features] > xU)).any(axis=1)

# Aplicar el filtro para mantener solo los datos sin outliers
df_sin_outliers = df[filtro_outliers]

# Visualizar la diferencia de filas
outlaiers = df.shape[0]- df_sin_outliers.shape[0]

# Verificar la forma del DataFrame después de eliminar outliers
print(f'Forma del DataFrame antes de eliminar outliers: {df_columns_selected.shape}')
print(f'Forma del DataFrame después de eliminar outliers: {df_sin_outliers.shape}')
print(f'Se han eliminado {outlaiers} outliers')
```

```
Forma del DataFrame antes de eliminar outliers: (622, 3)
Forma del DataFrame después de eliminar outliers: (557, 12)
Se han eliminado 65 outliers
```

NORMALIZACIÓN

Por último, antes de empezar a entrenar, normalizamos el dataset, para ello he decidido usar "StandardScaler" de sklearn.

```
from sklearn.preprocessing import StandardScaler

column_names = df_sin_outliers.columns.values
index_to_remove = np.where(column_names == 'Hardness')[0]
column_names = np.delete(column_names, index_to_remove)

scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_sin_outliers.drop('Hardness', axis=1))
scaled_df = pd.DataFrame(scaled_data, columns=column_names)
```

ENTRENAMIENTO

Como mencionamos al principio, se usará el modelo "LinearRegression" para entrenar.

Primero, dividimos los datos de entrenamiento y los datos de test.

```
X = df_sin_outliers[selected_features]
y = df_sin_outliers['Hardness']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Vamos a hacer dos entrenamientos, uno sin validación cruzada y otro con validación cruzada, los compararemos con el error cuadrático medio (MSE).

SIN CROSS VALIDATION

```
from sklearn.metrics import mean_squared_error
model_sin_cross_val = LinearRegression()
model_sin_cross_val.fit(X_train, y_train)
# Predicciones en el conjunto de prueba
predicciones_sin_cross_val = model_sin_cross_val.predict(X_test)

# Calcular el error cuadrático medio sin cross-validation
mse_sin_cv = mean_squared_error(y_test, predicciones_sin_cross_val)
print(f'Error cuadrático medio sin cross-validation: {mse_sin_cv}')
```

```
Error cuadrático medio sin cross-validation: 2.638260434322165
```

Como podemos ver, en este caso el MSE es de 2.64, un error algo elevado.

CON CROSS VALIDATION

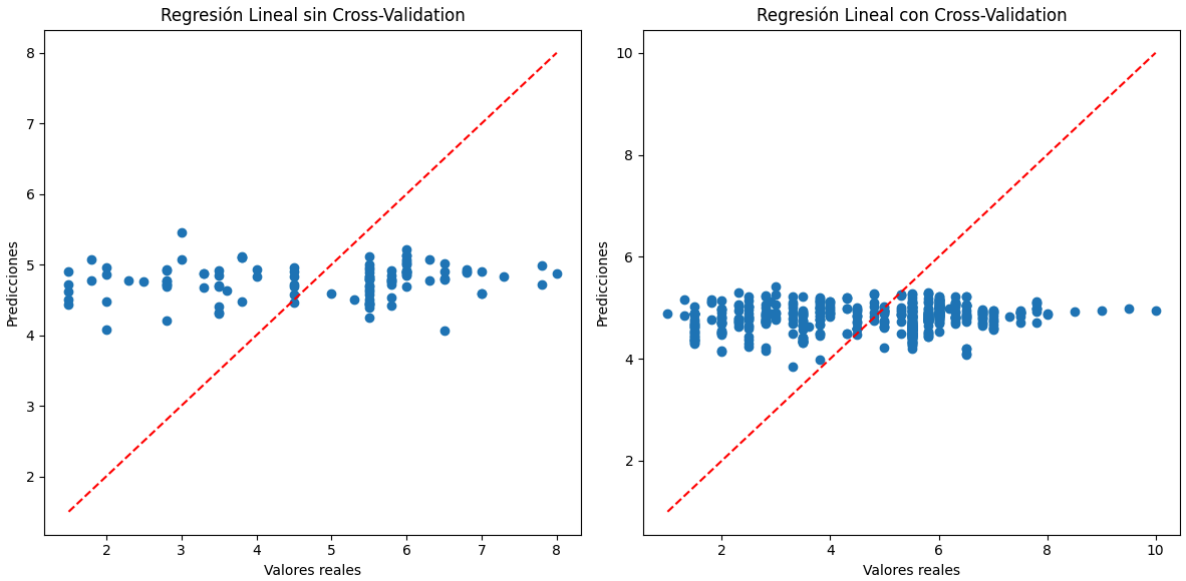
```
# Custom scoring function (MAE in this example)
def custom_scorer(y_true, y_pred):
    mae = median_absolute_error(y_true, y_pred)
    return mae

# Convert the custom scorer to a scorer compatible with cross_val_score
scorer = make_scorer(custom_scorer, greater_is_better=False)

skf = KFold(n_splits = 5)

model_con_cross_val = LinearRegression()
model_con_cross_val.fit(X, y)  # Ajustar el modelo con todos los datos
predicciones_con_cross_val = cross_val_score(model_con_cross_val, X, y, scoring=scorer, cv=skf, n_jobs=-1)
cv_mse = -np.mean(predicciones_con_cross_val)

print(f'Error cuadrático medio con cross-validation: {cv_mse}')
```

```
Error cuadrático medio con cross-validation: 1.194292125041649
```

En este caso, el MSE es de 1.19, algo más normal.

CONCLUSIÓN

En el caso de este modelo, si usamos validación cruzada reducimos en un 53% el error cuadrático medio. Esto es suficiente diferencia para que se tenga en cuenta en el uso del modelo.



Como podemos observar, el modelo entrenado con validación cruzada tiene valores más cercanos a la línea de regresión, lo que confirma que en este caso es mejor usar validación cruzada.

CONCLUSIÓN

El MSE más bajo que se ha podido obtener es de 1.19, se debería comprobar si usando otros modelos de regresión se podría obtener un error más bajo, dado que con este error, aunque no es muy elevado, si se notarán fallos a la hora de predecir la dureza de Moh.

REFERENCIAS

- Dataset
- Cuaderno de ayuda
- GitHub