LSTM

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import r2_score, mean_squared_error
from keras.models import Sequential
from keras.layers import LSTM, Dense
import matplotlib.pyplot as plt
import time
from scipy.stats import linregress


start_time = time.time()

file_path = ‘’
df = pd.read_excel(file_path)

df = df[df['PCE (best) %'] != 'na']
print(df.head())

columnas_categoricas = df.select_dtypes(include=['object']).columns.tolist()

label_encoder = LabelEncoder()
categoricas_transformadas = {}
for columna in columnas_categoricas:
    df[columna] = label_encoder.fit_transform(df[columna])
    categoricas_transformadas[columna] = dict(zip(label_encoder.classes_,
label_encoder.transform(label_encoder.classes_)))

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values.reshape(-1, 1)

scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled,
test_size=0.2, random_state=42)

X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

```
model = Sequential()
model.add(LSTM(units=200, input_shape=(X_train.shape[1], X_train.shape[2]),
return_sequences=True))
model.add(LSTM(units=200))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')

model.fit(X_train, y_train, epochs=1000, batch_size=32,
validation_data=(X_test, y_test))

y_pred_scaled = model.predict(X_test)


y_pred = scaler_y.inverse_transform(y_pred_scaled)

y_test_actual = scaler_y.inverse_transform(y_test)

top_indices = np.argsort(y_test_actual[:, 0])[::-1][:4]
top_combinations_original = df.iloc[top_indices, :-1]

mejores_combinaciones_con_nombres = []
for i, indices_combinacion in enumerate(top_indices):
    combinacion_transformada = top_combinations_original.iloc[i].to_dict()
    combinacion_actual = {}
    for columna, valor in combinacion_transformada.items():
        if columna in columnas_categoricas:
            valor_original =
list(categoricas_transformadas[columna].keys())[list(categoricas_transformad
as[columna].values()).index(valor)]
        else:
            valor_original = valor
        combinacion_actual[columna] = valor_original
    mejores_combinaciones_con_nombres.append(combinacion_actual)
    print(f"Mejor combinación {i + 1}: {combinacion_actual}")

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual, label='Actual')
plt.plot(y_pred, label='Predicted')
plt.scatter(top_indices, y_test_actual[top_indices], c='red', label='Top
Combinations')
plt.legend()
plt.title('Comparación de Eficiencias')
plt.show()

plt.figure(figsize=(10, 6))
```

```
bar_width = 0.35
bar_positions = np.arange(len(top_indices))
plt.bar(bar_positions, y_test_actual[top_indices][:, 0], bar_width,
label='Actual')
plt.bar(bar_positions + bar_width, y_pred[top_indices][:, 0], bar_width,
label='Predicted')
plt.xticks(bar_positions + bar_width / 2, [f'Mejor {i + 1}' for i in
range(len(top_indices))])
plt.legend()
plt.title('Comparación de Eficiencias')
plt.show()

end_time = time.time()
execution_time = end_time - start_time

r2 = r2_score(y_test_actual, y_pred)
mse = mean_squared_error(y_test_actual, y_pred)

tabla_resultados = pd.DataFrame({'R^2': [r2], 'MSE': [mse]})
print("\nTabla de Resultados:")
print(tabla_resultados)

slope, intercept, r_value, p_value, std_err =
linregress(y_test_actual.flatten(), y_pred.flatten())

tabla_metricas = pd.DataFrame({'R²': [r2], 'MSE': [mse], 'Pendiente':
[slope],
                               'Intersección': [intercept], 'Coeficiente de
Correlación (r)': [r_value],
                               'Valor p': [p_value], 'Tiempo de ejecución
(segundos)': [execution_time]})
print("\nMétricas de Validación:")
print(tabla_metricas)

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual / 3.5, label='Reales (Conjunto de entrenamiento)')
plt.plot(y_pred / 3.5, label='Predicciones (Conjunto de entrenamiento)')
plt.legend()
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()

plt.figure(figsize=(10, 6))
y_pred_full = model.predict(X_scaled.reshape((X_scaled.shape[0], 1,
X_scaled.shape[1])))
```

```
y_pred_full = scaler_y.inverse_transform(y_pred_full.reshape(-1, 1))
y_actual_full = scaler_y.inverse_transform(y_scaled)
plt.plot(y_actual_full / 3.5, label='Reales (Conjunto completo)')
plt.plot(y_pred_full / 3.5, label='Predicciones (Conjunto completo)')
plt.legend()
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()

top_indices_pred_full = np.argsort(y_pred_full[:, 0])[::-1][:4]
top_combinations_original_pred_full = df.iloc[top_indices_pred_full, :-1]

mejores_combinaciones_con_nombres_pred_full = []
for i, indices_combinacion in enumerate(top_indices_pred_full):
    combinacion_transformada_pred_full =
top_combinations_original_pred_full.iloc[i].to_dict()
    combinacion_actual_pred_full = {}
    for columna, valor in combinacion_transformada_pred_full.items():
        if columna in columnas_categoricas:
            valor_original_pred_full =
list(categoricas_transformadas[columna].keys())[list(categoricas_transformad
as[columna].values()).index(valor)]
        else:
            valor_original_pred_full = valor
        combinacion_actual_pred_full[columna] = valor_original_pred_full
    mejores_combinaciones_con_nombres_pred_full.append(combinacion_actual_pr
ed_full)
    print(f"Mejor combinación según predicciones en el conjunto completo {i
+ 1}: {combinacion_actual_pred_full}")

hiperparametros_lstm = {
    'Unidades LSTM 1': model.layers[0].units if hasattr(model.layers[0],
'units') else None,
    'Unidades LSTM 2': model.layers[1].units if hasattr(model.layers[1],
'units') else None,
    'Unidades Capa de Salida': model.layers[2].units if
hasattr(model.layers[2], 'units') else None,
    'Optimizador': model.optimizer.__class__.__name__,
    'Función de Pérdida': model.loss,
}

hiperparametros_entrenamiento = {
    'Épocas': model.history.params['epochs'],
    'Tamaño de Batch': 32,
}
```

```
hiperparametros = {**hiperparametros_lstm, **hiperparametros_entrenamiento}

tabla_hiperparametros = pd.DataFrame([hiperparametros])
print("\nTabla de Hiperparámetros del Modelo LSTM:")
print(tabla_hiperparametros)

y_train_pred_scaled = model.predict(X_train)

y_train_pred = scaler_y.inverse_transform(y_train_pred_scaled)

y_train_actual = scaler_y.inverse_transform(y_train)

plt.figure(figsize=(10, 6))
plt.plot(y_train_actual / 3.5, label='Actual (Conjunto de entrenamiento)')
plt.plot(y_train_pred / 3.5, label='Predicho (Conjunto de entrenamiento)')
plt.legend()
plt.title('Comparación de Eficiencias (Conjunto de entrenamiento)')
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()


y_test_pred_scaled = model.predict(X_test)

y_test_pred = scaler_y.inverse_transform(y_test_pred_scaled)

y_test_actual = scaler_y.inverse_transform(y_test)

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual / 3.5, label='Actual (Conjunto de pruebas)')
plt.plot(y_test_pred / 3.5, label='Predicho (Conjunto de pruebas)')
plt.legend()
plt.title('Comparación de Eficiencias (Conjunto de pruebas)')
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()

sorted_indices_train = np.argsort(y_train_actual.flatten())

plt.figure(figsize=(10, 6))
plt.plot(y_train_actual[sorted_indices_train] / 3.5, label='Real (Training
set)')
plt.plot(y_train_pred[sorted_indices_train] / 3.5, label='Predictions
(Training set)')
```

```
plt.legend()
plt.xlabel('Perovskite cell')
plt.ylabel('Efficiency (%)')
plt.show()

sorted_indices_test = np.argsort(y_test_actual.flatten())

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual[sorted_indices_test] / 3.5, label='Real (test set)')
plt.plot(y_test_pred[sorted_indices_test] / 3.5, label='Predictions (test
set )')
plt.legend()
plt.xlabel('Perovskite cell')
plt.ylabel('Efficiency (%)')
plt.show()

import scipy.stats as stats

residuos = y_test_actual - y_pred

plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuos)
plt.xlabel('Predicciones')
plt.ylabel('Residuos')
plt.title('Residuos vs Predicciones')
plt.axhline(y=0, color='r', linestyle='-')
plt.grid(True)
plt.show()

shapiro_test = stats.shapiro(residuos)
print("Prueba de Normalidad (Shapiro-Wilk):")
print("Estadística de prueba:", shapiro_test[0])
print("Valor p:", shapiro_test[1])

plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuos)
plt.xlabel('Predicciones')
plt.ylabel('Residuos')
plt.title('Residuos vs Predicciones')
plt.axhline(y=0, color='r', linestyle='-')
plt.grid(True)
plt.show()
```

SVM

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.svm import SVR
from scipy.stats import linregress
import matplotlib.pyplot as plt
import time
from sklearn.ensemble import RandomForestRegressor

start_time = time.time()
file_path = 'D:\Visual\Perovskita\inverted cop.xlsx'
df = pd.read_excel(file_path)
df = df[df['PCE (best) %'] != 'na']
print(df.head())


columnas_categoricas = df.select_dtypes(include=['object']).columns.tolist()

label_encoder = LabelEncoder()
for columna in columnas_categoricas:
    df[columna] = label_encoder.fit_transform(df[columna])

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values.reshape(-1, 1

scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y)


X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled,
test_size=0.2, random_state=42)

svm_model = SVR(kernel='poly', C=1.0, degree=3 , epsilon=0.2)
svm_model.fit(X_train, y_train.ravel())

y_pred_scaled = svm_model.predict(X_test).reshape(-1, 1)
```

```
y_pred = scaler_y.inverse_transform(y_pred_scaled)

y_test_actual = scaler_y.inverse_transform(y_test)

top_indices = np.argsort(y_test_actual[:, 0])[::-1][:4]
top_combinations = X_test[top_indices, :]
mejores_combinaciones_con_nombres = []
for i, indices_combinacion in enumerate(top_indices):
    nombres_variables = df.columns[np.nonzero(top_combinations[i])]
    combinacion_actual = dict(zip(nombres_variables, top_combinations[i]))
    mejores_combinaciones_con_nombres.append(combinacion_actual)
    print(f"Mejor combinación {i + 1}: {combinacion_actual}")

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual, label='Actual')
plt.plot(y_pred, label='Predicted')
plt.scatter(top_indices, y_test_actual[top_indices], c='red', label='Top
Combinations')
plt.legend()
plt.title('Comparación de Eficiencias')
plt.show()

plt.figure(figsize=(10, 6))
bar_width = 0.35
bar_positions = np.arange(len(top_indices))
plt.bar(bar_positions, y_test_actual[top_indices][:, 0], bar_width,
label='Actual')
plt.bar(bar_positions + bar_width, y_pred[top_indices][:, 0], bar_width,
label='Predicted')
plt.xticks(bar_positions + bar_width / 2, [f'Mejor {i + 1}' for i in
range(len(top_indices))])
plt.legend()
plt.title('Comparación de Eficiencias')
plt.show()
r2 = r2_score(y_test_actual, y_pred)
mse = mean_squared_error(y_test_actual, y_pred)

mae = np.mean(np.abs(y_test_actual - y_pred))
rmse = np.sqrt(mse)

tabla_metricas = pd.DataFrame({'R²': [r2], 'MSE': [mse], 'MAE': [mae],
'RMSE': [rmse]})
print("\nMétricas de Validación:")
print(tabla_metricas)
```

```
plt.figure(figsize=(10, 6))
plt.plot(y_test_actual, label='Actual (Test Set)')
plt.plot(y_pred, label='Predicted (Test Set)')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Pruebas')
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(scaler_y.inverse_transform(y_scaled), label='Actual (Full
Dataset)')
plt.plot(scaler_y.inverse_transform(svm_model.predict(X_scaled).reshape(-1,
1)), label='Predicted (Full Dataset)')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto Completo')
plt.show()

r2 = r2_score(y_test_actual, y_pred)
mse = mean_squared_error(y_test_actual, y_pred)

mae = np.mean(np.abs(y_test_actual - y_pred))
rmse = np.sqrt(mse)

slope, intercept, r_value, p_value, std_err =
linregress(y_test_actual.flatten(), y_pred.flatten())
end_time = time.time()

execution_time = end_time - start_time


tabla_metricas = pd.DataFrame({'R²': [r2], 'MSE': [mse], 'MAE': [mae],
'RMSE': [rmse], 'Pendiente': [slope],
                               'Intersección': [intercept], 'Coeficiente de
Correlación (r)': [r_value],
                               'Valor p': [p_value], 'tiempo':
[execution_time]})
print("\nMétricas de Validación:")
print(tabla_metricas)

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual, label='Actual')
plt.plot(y_pred, label='Predicted')
plt.scatter(top_indices, y_test_actual[top_indices], c='red', label='Top
Combinations')
plt.legend()
plt.title('Comparación de Eficiencias')
```

```
plt.show()

hiperparametros_svm = {
    'C': svm_model.C,
    'kernel': svm_model.kernel,
    'degree': svm_model.degree,
    'epsilon': svm_model.epsilon,
    'tol': svm_model.tol,
    'max_iter': svm_model.max_iter,
}

print(hiperparametros_svm)

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual / 3.5, label='Reales (Conjunto de entrenamiento)')
plt.plot(y_pred / 3.5, label='Predicciones (Conjunto de entrenamiento)')
plt.legend()
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(scaler_y.inverse_transform(y_scaled) / 3.5, label='Reales (Conjunto
completo)')
plt.plot(scaler_y.inverse_transform(svm_model.predict(X_scaled).reshape(-1,
1)) / 3.5,
         label='Predicciones (Conjunto completo)')
plt.legend()
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()

y_train_pred_scaled = svm_model.predict(X_train).reshape(-1, 1)

y_train_pred = scaler_y.inverse_transform(y_train_pred_scaled)

y_train_actual = scaler_y.inverse_transform(y_train)

top_indices_train = np.argsort(y_train_actual[:, 0])[::-1][:4]

plt.figure(figsize=(10, 6))
plt.plot(y_train_actual / 3.5, label='Actual (Training Set)')
plt.plot(y_train_pred / 3.5, label='Predicted (Training Set)')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Entrenamiento')
```

```
plt.show()

y_test_pred_scaled = svm_model.predict(X_test).reshape(-1, 1)

y_test_pred = scaler_y.inverse_transform(y_test_pred_scaled)

y_test_actual = scaler_y.inverse_transform(y_test)

top_indices_test = np.argsort(y_test_actual[:, 0])[::-1][:4]

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual / 3.5, label='Actual (Test Set)')
plt.plot(y_test_pred / 3.5, label='Predicted (Test Set)')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Pruebas')
plt.show()

y_test_pred_scaled = svm_model.predict(X_test).reshape(-1, 1)

y_test_pred = scaler_y.inverse_transform(y_test_pred_scaled)

y_test_actual = scaler_y.inverse_transform(y_test)

top_indices_test = np.argsort(y_test_actual[:, 0])[::-1][:4]

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual / 3.5, label='Actual (Test Set)')
plt.plot(y_test_pred / 3.5, label='Predicted (Test Set)')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Pruebas')
plt.show()


sorted_indices_train = np.argsort(y_train_actual[:, 0])

plt.figure(figsize=(10, 6))
plt.plot(y_train_actual[sorted_indices_train] / 3.5, label='Reales (Conjunto
de entrenamiento)')
plt.plot(y_train_pred[sorted_indices_train] / 3.5, label='Predicciones
(Conjunto de entrenamiento)')
plt.legend()
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()
```

```
sorted_indices_test = np.argsort(y_test_actual[:, 0])

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual[sorted_indices_test] / 3.5, label='Reales (Conjunto
de prueba)')
plt.plot(y_test_pred[sorted_indices_test] / 3.5, label='Predicciones
(Conjunto de prueba)')
plt.legend()
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()
```

RANDOM FOREST

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error
from scipy.stats import linregress
import time
from sklearn.ensemble import RandomForestRegressor
import scipy.stats as stats

start_time = time.time()

file_path = ‘’
df = pd.read_excel(file_path)

df = df[df['PCE (best) %'] != 'na']
print(df.head())
```

```
columnas_categoricas = df.select_dtypes(include=['object']).columns.tolist()

label_encoder = LabelEncoder()
categoricas_transformadas = {}
for columna in columnas_categoricas:
    df[columna] = label_encoder.fit_transform(df[columna])
    categoricas_transformadas[columna] = dict(zip(label_encoder.classes_,
label_encoder.transform(label_encoder.classes_)))

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values.reshape(-1, 1)
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled,
test_size=0.2, random_state=42)

random_forest_model = RandomForestRegressor(n_estimators=31,
random_state=42)
random_forest_model.fit(X_train, y_train.ravel())

y_pred = scaler_y.inverse_transform(y_pred_scaled.reshape(-1, 1))

y_test_actual = scaler_y.inverse_transform(y_test)

df_pred = pd.DataFrame(np.hstack((X_test, y_pred)), columns=df.columns)
df_actual = pd.DataFrame(np.hstack((X_test, y_test_actual)),
columns=df.columns)

r2 = r2_score(y_test_actual, y_pred)
mse = mean_squared_error(y_test_actual, y_pred)

tabla_resultados = pd.DataFrame({'R^2': [r2], 'MSE': [mse]})
print("\nTabla de Resultados:")
print(tabla_resultados)

r2 = r2_score(y_test_actual, y_pred)
mse = mean_squared_error(y_test_actual, y_pred)
mae = np.mean(np.abs(y_test_actual - y_pred))
rmse = np.sqrt(mse)
```

```
slope, intercept, r_value, p_value, std_err =
linregress(y_test_actual.flatten(), y_pred.flatten())

tabla_metricas = pd.DataFrame({'R²': [r2], 'MSE': [mse], 'MAE': [mae],
'RMSE': [rmse], 'Pendiente': [slope],
                               'Intersección': [intercept], 'Coeficiente de
Correlación (r)': [r_value],
                               'Valor p': [p_value]})
print("\nMétricas de Validación:")
print(tabla_metricas)
top_indices = np.argsort(y_test_actual[:, 0])[::-1][:4]

mejores_combinaciones_con_nombres = []
for i, indices_combinacion in enumerate(top_indices):
    combinacion_transformada = top_combinations_original.iloc[i].to_dict()
    combinacion_actual = {}
    for columna, valor in combinacion_transformada.items():
        if columna in columnas_categoricas:
            valor_original =
list(categoricas_transformadas[columna].keys())[list(categoricas_transformad
as[columna].values()).index(valor)]
        else:
            valor_original = valor
        combinacion_actual[columna] = valor_original
    mejores_combinaciones_con_nombres.append(combinacion_actual)
    print(f"Mejor combinación {i + 1}: {combinacion_actual}")

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual, label='Actual')
plt.plot(y_pred, label='Predicted')
plt.scatter(top_indices, y_test_actual[top_indices], c='red', label='Top
Combinations')
plt.legend()
plt.title('Comparación de Eficiencias')
plt.show()

y_pred_full_scaled = random_forest_model.predict(X_scaled)

y_pred_full = scaler_y.inverse_transform(y_pred_full_scaled.reshape(-1, 1))

top_indices_full = np.argsort(y_pred_full[:, 0])[::-1][:4]
top_combinations_full_original = df.iloc[top_indices_full, :-1]

mejores_combinaciones_full_con_nombres = []
for i, indices_combinacion in enumerate(top_indices_full):
```

```
    combinacion_transformada =
top_combinations_full_original.iloc[i].to_dict()
    combinacion_actual = {}
    for columna, valor in combinacion_transformada.items():
        if columna in columnas_categoricas:
            valor_original =
list(categoricas_transformadas[columna].keys())[list(categoricas_transformad
as[columna].values()).index(valor)]
        else:
            valor_original = valor
        combinacion_actual[columna] = valor_original
    mejores_combinaciones_full_con_nombres.append(combinacion_actual)
    print(f"Mejor combinación {i + 1} en el Conjunto de Datos Completo:
{combinacion_actual}")

plt.figure(figsize=(10, 6))
plt.plot(y, label='Actual')
plt.plot(y_pred_full, label='Predicted')
plt.scatter(top_indices_full, y[top_indices_full], c='red', label='Top
Combinations')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Datos Completo')
plt.show()

end_time = time.time()

execution_time = end_time - start_time

mse = mean_squared_error(y_test_actual, y_pred)

slope, intercept, r_value, p_value, std_err =
linregress(y_test_actual.flatten(), y_pred.flatten())
tabla_metricas = pd.DataFrame({
    'R²': [r2],
    'MSE': [mse],
    'Valor p': [p_value],
    'Tiempo de Ejecución (s)': [execution_time]
})

print("\nMétricas de Validación:")
print(tabla_metricas)
plt.figure(figsize=(10, 6))
plt.plot(y_test_actual / 3.5, label='Reales (Conjunto de entrenamiento)')
plt.plot(y_pred / 3.5, label='Predicciones (Conjunto de entrenamiento)')
plt.legend()
```

```
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(scaler_y.inverse_transform(y_scaled) / 3.5, label='Reales (Conjunto
completo)')
plt.plot(scaler_y.inverse_transform(random_forest_model.predict(X_scaled).re
shape(-1, 1)) / 3.5,
         label='Predicciones (Conjunto completo)')
plt.legend()
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()

hiperparametros_random_forest = {
    'n_estimators': random_forest_model.n_estimators,
    'criterion': random_forest_model.criterion,
    'random_state': random_forest_model.random_state,
    'max_depth': random_forest_model.max_depth,
    'min_samples_split': random_forest_model.min_samples_split,
    'min_samples_leaf': random_forest_model.min_samples_leaf,
}

print(hiperparametros_random_forest)

y_train_pred_scaled = random_forest_model.predict(X_train)

y_train_pred = scaler_y.inverse_transform(y_train_pred_scaled.reshape(-1,
1))

y_train_actual = scaler_y.inverse_transform(y_train)

plt.figure(figsize=(10, 6))
plt.plot(y_train_actual / 3.5, label='Actual')
plt.plot(y_train_pred / 3.5, label='Predicted')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Entrenamiento')
plt.show()
y_test_pred_scaled = random_forest_model.predict(X_test)

y_test_pred = scaler_y.inverse_transform(y_test_pred_scaled.reshape(-1, 1))
y_test_actual = scaler_y.inverse_transform(y_test)

plt.figure(figsize=(10, 6))
```

```
plt.plot(y_test_actual / 3.5, label='Actual')
plt.plot(y_test_pred / 3.5, label='Predicted')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Pruebas')
plt.show()

sorted_indices_train_pred = np.argsort(y_train_pred[:, 0])
sorted_indices_train_actual = np.argsort(y_train_actual[:, 0])

y_train_pred_sorted = y_train_pred[sorted_indices_train_pred]
y_train_actual_sorted = y_train_actual[sorted_indices_train_actual]

plt.figure(figsize=(10, 6))
plt.plot(y_train_actual_sorted / 3.5, label='Reales (Conjunto de
entrenamiento)')
plt.plot(y_train_pred_sorted / 3.5, label='Predicciones (Conjunto de
entrenamiento)')
plt.legend()
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()


sorted_indices_pred = np.argsort(y_test_pred[:, 0])
sorted_indices_actual = np.argsort(y_test_actual[:, 0])

y_test_pred_sorted = y_test_pred[sorted_indices_pred]
y_test_actual_sorted = y_test_actual[sorted_indices_actual]

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual_sorted / 3.5, label='Reales (Conjunto de prueba)')
plt.plot(y_test_pred_sorted / 3.5, label='Predicciones (Conjunto de
prueba)')
plt.legend()
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()

y_train_pred_scaled = random_forest_model.predict(X_train)

y_train_pred = scaler_y.inverse_transform(y_train_pred_scaled.reshape(-1,
1))

r2_train = r2_score(y_train_actual, y_train_pred)
r2_pred = r2_score(y_test_actual, y_test_pred)
```

```
print("Coeficiente de determinación (R^2) en el conjunto de entrenamiento:",
r2_train)
print(r2_pred)

residuos = y_test_actual - y_pred


plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuos)
plt.xlabel('Predicciones')
plt.ylabel('Residuos')
plt.title('Residuos vs Predicciones')
plt.axhline(y=0, color='r', linestyle='-')
plt.grid(True)
plt.show()

shapiro_test = stats.shapiro(residuos)
print("Prueba de Normalidad (Shapiro-Wilk):")
print("Estadística de prueba:", shapiro_test[0])
print("Valor p:", shapiro_test[1])

plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuos)
plt.xlabel('Predicciones')
plt.ylabel('Residuos')
plt.title('Residuos vs Predicciones')
plt.axhline(y=0, color='r', linestyle='-')
plt.grid(True)
plt.show()
```

GRADIENT BOOSTING

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.ensemble import GradientBoostingRegressor
from scipy.stats import linregress
import matplotlib.pyplot as plt

file_path = ‘’
df = pd.read_excel(file_path)
```

```
df = df[df['PCE (best) %'] != 'na']
print(df.head())
print(df.describe())

columnas_categoricas = df.select_dtypes(include=['object']).columns.tolist()

label_encoder = LabelEncoder()
for columna in columnas_categoricas:
    df[columna] = label_encoder.fit_transform(df[columna])

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values.reshape(-1, 1)

scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled,
test_size=0.2, random_state=42)

gradient_boosting_model = GradientBoostingRegressor(n_estimators=17500,
learning_rate=0.01, random_state=42)
gradient_boosting_model.fit(X_train, y_train.ravel())

y_pred_scaled = gradient_boosting_model.predict(X_test).reshape(-1, 1)

y_pred = scaler_y.inverse_transform(y_pred_scaled)

y_test_actual = scaler_y.inverse_transform(y_test)

top_indices = np.argsort(y_test_actual[:, 0])[::-1][:4]
top_combinations = X_test[top_indices, :]

mejores_combinaciones_con_nombres = []
for i, indices_combinacion in enumerate(top_indices):
    nombres_variables = df.columns[np.nonzero(top_combinations[i])]
    combinacion_actual = dict(zip(nombres_variables, top_combinations[i]))
    mejores_combinaciones_con_nombres.append(combinacion_actual)
    print(f"Mejor combinación {i + 1}: {combinacion_actual}")

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual, label='Actual')
plt.plot(y_pred, label='Predicted')
```

```
plt.scatter(top_indices, y_test_actual[top_indices], c='red', label='Top
Combinations')
plt.legend()
plt.title('Comparación de Eficiencias')
plt.show()

plt.figure(figsize=(10, 6))
bar_width = 0.35
bar_positions = np.arange(len(top_indices))
plt.bar(bar_positions, y_test_actual[top_indices][:, 0], bar_width,
label='Actual')
plt.bar(bar_positions + bar_width, y_pred[top_indices][:, 0], bar_width,
label='Predicted')
plt.xticks(bar_positions + bar_width / 2, [f'Mejor {i + 1}' for i in
range(len(top_indices))])
plt.legend()
plt.title('Comparación de Eficiencias')
plt.show()

r2 = r2_score(y_test_actual, y_pred)
mse = mean_squared_error(y_test_actual, y_pred)

mae = np.mean(np.abs(y_test_actual - y_pred))
rmse = np.sqrt(mse)

tabla_metricas = pd.DataFrame({'R²': [r2], 'MSE': [mse], 'MAE': [mae],
'RMSE': [rmse]})
print("\nMétricas de Validación:")
print(tabla_metricas)

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual, label='Actual (Test Set)')
plt.plot(y_pred, label='Predicted (Test Set)')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Pruebas')
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(scaler_y.inverse_transform(y_scaled), label='Actual (Full
Dataset)')
plt.plot(scaler_y.inverse_transform(gradient_boosting_model.predict(X_scaled
).reshape(-1, 1)), label='Predicted (Full Dataset)')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto Completo')
plt.show()
```

```
r2 = r2_score(y_test_actual, y_pred)
mse = mean_squared_error(y_test_actual, y_pred)

mae = np.mean(np.abs(y_test_actual - y_pred))
rmse = np.sqrt(mse)

slope, intercept, r_value, p_value, std_err =
linregress(y_test_actual.flatten(), y_pred.flatten())

tabla_metricas = pd.DataFrame({'R²': [r2], 'MSE': [mse], 'MAE': [mae],
'RMSE': [rmse], 'Pendiente': [slope],
                               'Intersección': [intercept], 'Coeficiente de
Correlación (r)': [r_value],
                               'Valor p': [p_value]})
print("\nMétricas de Validación:")
print(tabla_metricas)

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual / 3.5, label='Actual')
plt.plot(y_pred / 3.5, label='Predicted')
plt.scatter(top_indices, y_test_actual[top_indices], c='red', label='Top
Combinations')
plt.legend()
plt.title('Comparación de Eficiencias')
plt.show()

plt.figure(figsize=(10, 6))
bar_width = 0.35
bar_positions = np.arange(len(top_indices))
plt.bar(bar_positions, y_test_actual[top_indices][:, 0] / 3.5, bar_width,
label='Actual')
plt.bar(bar_positions + bar_width, y_pred[top_indices][:, 0] / 3.5,
bar_width, label='Predicted')
plt.xticks(bar_positions + bar_width / 2, [f'Mejor {i + 1}' for i in
range(len(top_indices))])
plt.legend()
plt.title('Comparación de Eficiencias')
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual / 3.5, label='Reales (Conjunto de entrenamiento)')
plt.plot(y_pred / 3.5, label='Predicciones (Conjunto de entrenamiento)')
plt.legend()
plt.xlabel('Celda de perovskita')
```

```
plt.ylabel('Eficiencia (%)')
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(scaler_y.inverse_transform(y_scaled) / 3.5, label='Reales (Conjunto
completo)')
plt.plot(scaler_y.inverse_transform(gradient_boosting_model.predict(X_scaled
).reshape(-1, 1)) / 3.5,
         label='Predicciones (Conjunto completo)')
plt.legend()
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()

y_pred_train_scaled = gradient_boosting_model.predict(X_train).reshape(-1,
1)

y_pred_train = scaler_y.inverse_transform(y_pred_train_scaled)

y_train_actual = scaler_y.inverse_transform(y_train)

plt.figure(figsize=(10, 6))
plt.plot(y_train_actual / 3.5, label='Actual (Conjunto de entrenamiento)')
plt.plot(y_pred_train / 3.5, label='Predicted (Conjunto de entrenamiento)')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Entrenamiento')
plt.show()

sorted_indices_test = np.argsort(y_test_actual[:, 0])
sorted_indices_train = np.argsort(y_train_actual[:, 0])

plt.figure(figsize=(10, 6))
plt.plot(y_train_actual[sorted_indices_train] / 3.5, label='Reales (Conjunto
de entrenamiento)')
plt.plot(y_pred_train[sorted_indices_train] / 3.5, label='Predicciones
(Conjunto de entrenamiento)')
plt.legend()
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual[sorted_indices_test] / 3.5, label='Reales (Conjunto
de prueba)')
```

```
plt.plot(y_pred[sorted_indices_test] / 3.5, label='Predicciones (Conjunto de prueba)')
plt.legend()
plt.xlabel('Celda de perovskita')
plt.ylabel('Eficiencia (%)')
plt.show()


y_pred_full_scaled = gradient_boosting_model.predict(X_scaled).reshape(-1, 1)

y_pred_full = scaler_y.inverse_transform(y_pred_full_scaled)

y_full_actual = scaler_y.inverse_transform(y_scaled)

sorted_indices_full = np.argsort(y_full_actual[:, 0])

plt.figure(figsize=(10, 6))
plt.plot(y_full_actual[sorted_indices_full]/3.5, label='Actual (Conjunto completo)')
plt.plot(y_pred_full[sorted_indices_full]/3.5, label='Predicted (Conjunto completo)')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto Completo (Ordenado)')
plt.show()

r2_full = r2_score(y_full_actual, y_pred_full)
mse_full = mean_squared_error(y_full_actual, y_pred_full)

mae_full = np.mean(np.abs(y_full_actual - y_pred_full))
rmse_full = np.sqrt(mse_full)

tabla_metricas_full = pd.DataFrame({'R² (Full)': [r2_full], 'MSE (Full)': [mse_full], 'MAE (Full)': [mae_full], 'RMSE (Full)': [rmse_full]})
print("\nMétricas de Validación para el Conjunto Completo:")
print(tabla_metricas_full)

import scipy.stats as stats

residuos = y_test_actual - y_pred

plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuos)
plt.xlabel('Predicciones')
plt.ylabel('Residuos')
```

```
plt.title('Residuos vs Predicciones')
plt.axhline(y=0, color='r', linestyle='-')
plt.grid(True)
plt.show()

shapiro_test = stats.shapiro(residuos)
print("Prueba de Normalidad (Shapiro-Wilk):")
print("Estadística de prueba:", shapiro_test[0])
print("Valor p:", shapiro_test[1])

plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuos)
plt.xlabel('Predicciones')
plt.ylabel('Residuos')
plt.title('Residuos vs Predicciones')
plt.axhline(y=0, color='r', linestyle='-')
plt.grid(True)
plt.show()
```

XGBOOST

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import r2_score, mean_squared_error
from xgboost import XGBRegressor
import matplotlib.pyplot as plt
from scipy.stats import linregress

file_path = ‘’
df = pd.read_excel(file_path)
df = df[df['PCE (best) %'] != 'na']
print(df.head())

columnas_categoricas = df.select_dtypes(include=['object']).columns.tolist()

label_encoder = LabelEncoder()
for columna in columnas_categoricas:
    df[columna] = label_encoder.fit_transform(df[columna])
```

```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values.reshape(-1, 1)

scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled,
test_size=0.2, random_state=42)
xgb_model = XGBRegressor(n_estimators=30000, learning_rate=0.001,
colsample_bytree=0.9, max_depth=7, subsample=0.9, random_state=42)
xgb_model.fit(X_train, y_train.ravel())

y_pred_scaled_xgb = xgb_model.predict(X_test)

y_pred_xgb = scaler_y.inverse_transform(y_pred_scaled_xgb.reshape(-1, 1))

y_test_actual_xgb = scaler_y.inverse_transform(y_test)


top_indices_xgb = np.argsort(y_test_actual_xgb[:, 0])[::-1][:4]
top_combinations_xgb = X_test[top_indices_xgb, :]

mejores_combinaciones_con_nombres_xgb = []
for i, indices_combinacion in enumerate(top_indices_xgb):
    nombres_variables = df.columns[np.nonzero(top_combinations_xgb[i])]
    combinacion_actual = dict(zip(nombres_variables,
top_combinations_xgb[i]))
    mejores_combinaciones_con_nombres_xgb.append(combinacion_actual)
    print(f"Mejor combinación {i + 1} (XGBoost): {combinacion_actual}")

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual_xgb, label='Actual')
plt.plot(y_pred_xgb, label='XGBoost Predicted')
plt.scatter(top_indices_xgb, y_test_actual_xgb[top_indices_xgb], c='red',
label='Top Combinations (XGBoost)')
plt.legend()
plt.title('Comparación de Eficiencias (XGBoost)')
plt.show()

r2 = r2_score(y_test_actual_xgb, y_pred_xgb)
mse = mean_squared_error(y_test_actual_xgb, y_pred_xgb)
```

```
mae = np.mean(np.abs(y_test_actual_xgb - y_pred_xgb))
rmse = np.sqrt(mse)

tabla_metricas = pd.DataFrame({'R²': [r2], 'MSE': [mse], 'MAE': [mae],
'RMSE': [rmse]})
print("\nMétricas de Validación:")
print(tabla_metricas)



mae = np.mean(np.abs(y_test_actual_xgb - y_pred_xgb))
rmse = np.sqrt(mse)

slope, intercept, r_value, p_value, std_err =
linregress(y_test_actual_xgb.flatten(), y_pred_xgb.flatten())

tabla_metricas = pd.DataFrame({'R²': [r2], 'MSE': [mse], 'MAE': [mae],
'RMSE': [rmse], 'Pendiente': [slope],
                               'Intersección': [intercept], 'Coeficiente de
Correlación (r)': [r_value],
                               'Valor p': [p_value]})
print("\nMétricas de Validación:")
print(tabla_metricas)


y_train_pred_scaled_xgb = xgb_model.predict(X_train)


y_train_pred_xgb =
scaler_y.inverse_transform(y_train_pred_scaled_xgb.reshape(-1, 1))

y_train_actual_xgb = scaler_y.inverse_transform(y_train)

plt.figure(figsize=(10, 6))
plt.plot(y_train_actual_xgb/3.5, label='Actual (Entrenamiento)')
plt.plot(y_train_pred_xgb/3.5, label='XGBoost Predicted (Entrenamiento)')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Entrenamiento
(XGBoost)')
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual_xgb/3.5, label='Actual (Prueba)')
plt.plot(y_pred_xgb/3.5, label='XGBoost Predicted (Prueba)')
```

```
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Prueba (XGBoost)')
plt.show()

sorted_indices_train_xgb = np.argsort(y_train_actual_xgb[:, 0])

plt.figure(figsize=(10, 6))
plt.plot(y_train_actual_xgb[sorted_indices_train_xgb]/3.5, label='Actual
(Conjunto de Entrenamiento)')
plt.plot(y_train_pred_xgb[sorted_indices_train_xgb]/3.5, label='XGBoost
Predicted (Conjunto de Entrenamiento)')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Entrenamiento
(Ordenado)')
plt.show()



sorted_indices_xgb = np.argsort(y_test_actual_xgb[:, 0])

plt.figure(figsize=(10, 6))
plt.plot(y_test_actual_xgb[sorted_indices_xgb]/3.5, label='Actual (Conjunto
de Prueba)')
plt.plot(y_pred_xgb[sorted_indices_xgb]/3.5, label='XGBoost Predicted
(Conjunto de Prueba)')
plt.legend()
plt.title('Comparación de Eficiencias en el Conjunto de Prueba (Ordenado)')
plt.show()
```