Building QSAR machine learning models for drug discovery against Lepra

Step 2: Installing the lazypredict package

**Step 3: Importing and visualizing the pre-processed database

```
### 3.1: Importing the dataset.

from google.colab import files
uploaded = files.upload()
```

```
Escolher arquivos eNOS-data…l_LEPRA.csv
```

- **eNOS-dataset_final_LEPRA.csv**(text/csv) - 3394356 bytes, last modified: 21/12/2024 - 100% done

```
Saving eNOS-dataset_final_LEPRA.csv to eNOS-dataset_final_LEPRA (2).csv
```

```
### 3.2: Visualizing the imported dataset.

import pandas as pd
df1 = pd.read_csv("eNOS-dataset_final_LEPRA.csv")
display (df1)
```

	Unnamed: 0	PubchemFP0	PubchemFP1	PubchemFP2	PubchemFP3	PubchemFP4	PubchemFP5	PubchemFP6	PubchemFP7	PubchemFP8	...	Pub
0	0	1	1	1	0	0	0	0	0	0	...	
1	1	1	1	1	0	0	0	0	0	0	...	
2	2	1	1	1	0	0	0	0	0	0	...	
3	3	0	0	0	0	0	0	0	0	0	...	
4	4	1	1	1	0	0	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1892	1892	1	0	0	0	0	0	0	0	0	...	
1893	1893	1	0	0	0	0	0	0	0	0	...	
1894	1894	1	0	0	0	0	0	0	0	0	...	
1895	1895	1	0	0	0	0	0	0	0	0	...	
1896	1896	1	0	0	0	0	0	0	0	0	...	

1897 rows × 883 columns

```
## Removing non-informative variables.

df1 = df1.drop("Unnamed: 0", axis = 1)
```

```
# Removing empty rows (independent variables).

df1 = df1.dropna()
```

```
df1
```

	PubchemFP0	PubchemFP1	PubchemFP2	PubchemFP3	PubchemFP4	PubchemFP5	PubchemFP6	PubchemFP7	PubchemFP8	PubchemFP9	...	Pu
0	1	1	1	0	0	0	0	0	0	1	...	
1	1	1	1	0	0	0	0	0	0	1	...	
2	1	1	1	0	0	0	0	0	0	1	...	
3	0	0	0	0	0	0	0	0	0	1	...	
4	1	1	1	0	0	0	0	0	0	1	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1892	1	0	0	0	0	0	0	0	0	1	...	
1893	1	0	0	0	0	0	0	0	0	1	...	
1894	1	0	0	0	0	0	0	0	0	1	...	
1895	1	0	0	0	0	0	0	0	0	1	...	
1896	1	0	0	0	0	0	0	0	0	1	...	

1897 rows × 882 columns

```
# Removing infinite values.

import pandas as pd

# Replacing infinite values with NaN.

df1.replace([float('inf'), float('-inf')], pd.NA, inplace=True)

# Removing rows that contain NaN values.

df1.dropna(inplace=True)
```

```
df1
```

	PubchemFP0	PubchemFP1	PubchemFP2	PubchemFP3	PubchemFP4	PubchemFP5	PubchemFP6	PubchemFP7	PubchemFP8	PubchemFP9	...	Pu
0	1	1	1	0	0	0	0	0	0	1	...	
1	1	1	1	0	0	0	0	0	0	1	...	
2	1	1	1	0	0	0	0	0	0	1	...	
3	0	0	0	0	0	0	0	0	0	1	...	
4	1	1	1	0	0	0	0	0	0	1	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1892	1	0	0	0	0	0	0	0	0	1	...	
1893	1	0	0	0	0	0	0	0	0	1	...	
1894	1	0	0	0	0	0	0	0	0	1	...	
1895	1	0	0	0	0	0	0	0	0	1	...	
1896	1	0	0	0	0	0	0	0	0	1	...	

1897 rows × 882 columns

Importing and pre-processing external dataset

Step 4: Building ML models

```
x = df1.drop("pIC50", axis = 1)
y = df1["pIC50"]
```

```
x
```

	PubchemFP0	PubchemFP1	PubchemFP2	PubchemFP3	PubchemFP4	PubchemFP5	PubchemFP6	PubchemFP7	PubchemFP8	PubchemFP9	...	Pu
0	1	1	1	0	0	0	0	0	0	1	...	
1	1	1	1	0	0	0	0	0	0	1	...	
2	1	1	1	0	0	0	0	0	0	1	...	
3	0	0	0	0	0	0	0	0	0	1	...	
4	1	1	1	0	0	0	0	0	0	1	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1892	1	0	0	0	0	0	0	0	0	1	...	
1893	1	0	0	0	0	0	0	0	0	1	...	
1894	1	0	0	0	0	0	0	0	0	1	...	
1895	1	0	0	0	0	0	0	0	0	1	...	
1896	1	0	0	0	0	0	0	0	0	1	...	

1897 rows × 881 columns

```
### 3.1. Removing descriptors with low variance
from sklearn.feature_selection import VarianceThreshold

def remove_baixa_variancia(input_data, threshold=0.1):
    selection = VarianceThreshold(threshold)
    selection.fit(input_data)
    return input_data[input_data.columns[selection.get_support(indices=True)]]
x = remove_baixa_variancia(x, threshold=0.1)
x
```

	PubchemFP0	PubchemFP1	PubchemFP2	PubchemFP13	PubchemFP14	PubchemFP15	PubchemFP20	PubchemFP23	PubchemFP24	PubchemFP33
0	1	1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	0	1	0	0	0
2	1	1	1	0	1	0	1	0	0	0
3	0	0	0	0	1	0	1	0	0	0
4	1	1	1	0	1	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...
1892	1	0	0	1	0	0	1	0	0	0
1893	1	0	0	1	0	0	1	0	0	0
1894	1	0	0	1	0	0	1	0	0	0
1895	1	0	0	1	0	0	1	0	0	0
1896	1	0	0	1	0	0	1	0	0	0

1897 rows × 230 columns

```
y
```

	pIC50
0	7.853872
1	7.744727
2	6.892790
3	7.823909
4	8.000000
...	...
1892	6.847712
1893	7.468521
1894	7.096910
1895	7.619789
1896	5.881405

1897 rows × 1 columns

dtype: float64

Splitting the data into training and testing sets.

```
###  Importing packages for visualizing the results of machine learning models.

import seaborn as sns


# Importing packages for splitting the data into training and testing sets.
import sklearn
from sklearn.model_selection import train_test_split

x_treino, x_teste, y_treino, y_teste = train_test_split(x, y, test_size=0.3, random_state=100)
```

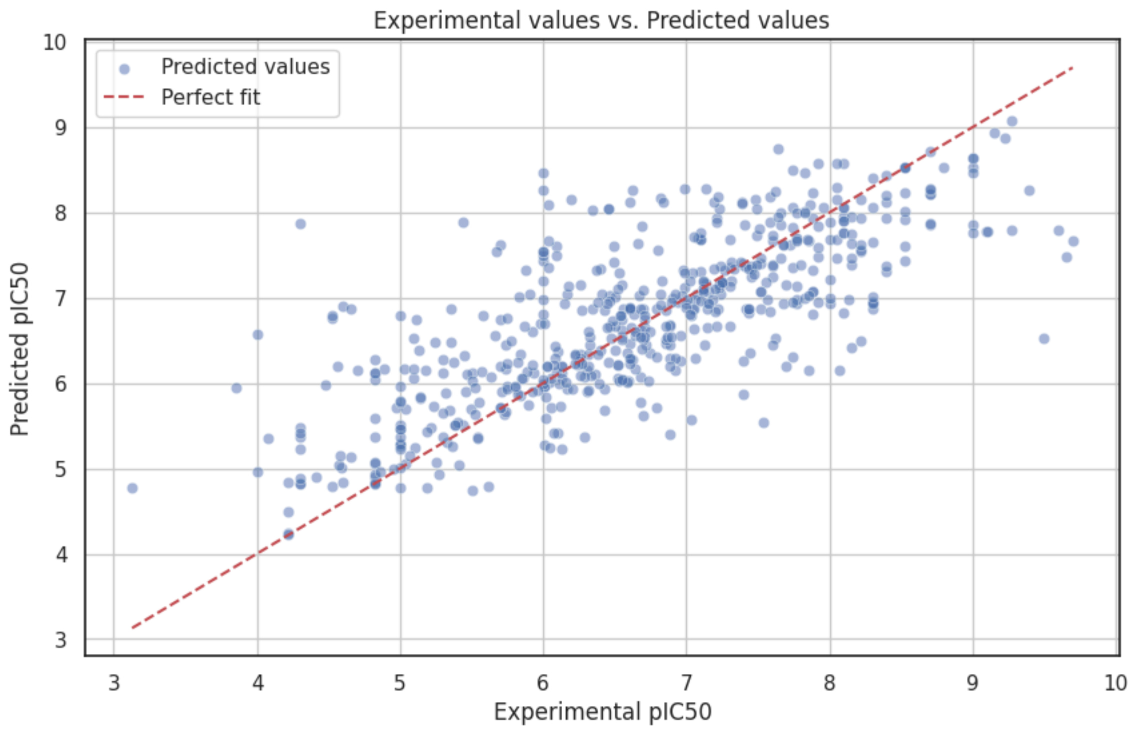
Top 1: Machine learning

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np

from sklearn.ensemble import RandomForestRegressor

modelo1 = RandomForestRegressor()       # Insert a machine learning model here.
modelo1.fit(x_treino, y_treino)


y_predito1 = modelo1.predict(x_teste)


# Calculating the R²
r2_treino = modelo1.score(x_treino, y_treino)
print("R2 treino : " + str(r2_treino))

# Test results

r2_teste = modelo1.score(x_teste, y_teste)
print("R2 teste : " + str(r2_teste))

# Calculating MSE (Mean Squared Error)
mse_treino = mean_squared_error(y_treino, modelo1.predict(x_treino))
print("MSE treino : " + str(mse_treino))

mse_teste = mean_squared_error(y_teste, y_predito1)
print("MSE teste : " + str(mse_teste))

# Calculating RMSE (Root Mean Squared Error)
rmse_treino = np.sqrt(mse_treino)
print("RMSE treino : " + str(rmse_treino))

rmse_teste = np.sqrt(mse_teste)
print("RMSE teste : " + str(rmse_teste))

# Calculating MAE (Mean Absolute Error)
```

```
mae_treino = mean_absolute_error(y_treino, modelo1.predict(x_treino))
print("MAE treino : " + str(mae_treino))

mae_teste = mean_absolute_error(y_teste, y_predito1)
print("MAE teste : " + str(mae_teste))
```

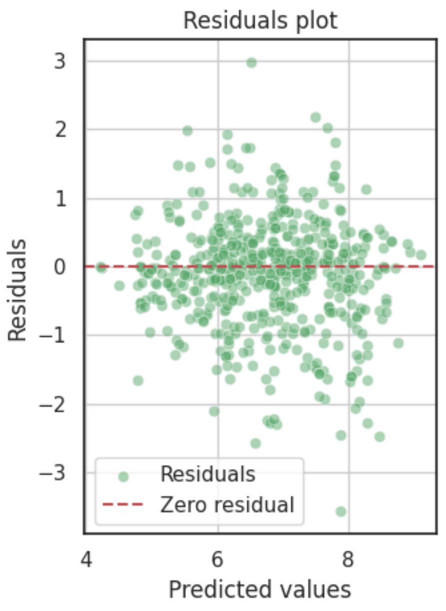
```
R2 treino : 0.8588652459259454
R2 teste : 0.5712796732460363
MSE treino : 0.21066239620578034
MSE teste : 0.6024509233588043
RMSE treino : 0.45897973398155645
RMSE teste : 0.7761771211255871
MAE treino : 0.3098427729922266
MAE teste : 0.5560269618097684
```

```
# Building the graph of experimental values versus predicted values
import matplotlib.pyplot as plt  # Add this line to import matplotlib.pyplot:

import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_teste, y=y_predito1, alpha=0.5, label='Predicted values', color='b')
sns.lineplot(x=[y_teste.min(), y_teste.max()], y=[y_teste.min(), y_teste.max()], color='r', linestyle='--', label='Perfect fit')
plt.xlabel('Experimental pIC50')
plt.ylabel('Predicted pIC50')
plt.title('Experimental values vs. Predicted values')
plt.legend()
plt.grid(True)
plt.show()
```



Residual plot for the training set

```
# Building the residual plot
residuos = y_teste - y_predito1

plt.subplot(1, 2, 2)
sns.scatterplot(x=y_predito1, y=residuos, alpha=0.5, label='Residuals', color='g')
plt.axhline(y=0, color='r', linestyle='--', label='Zero residual')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.title('Residuals plot')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



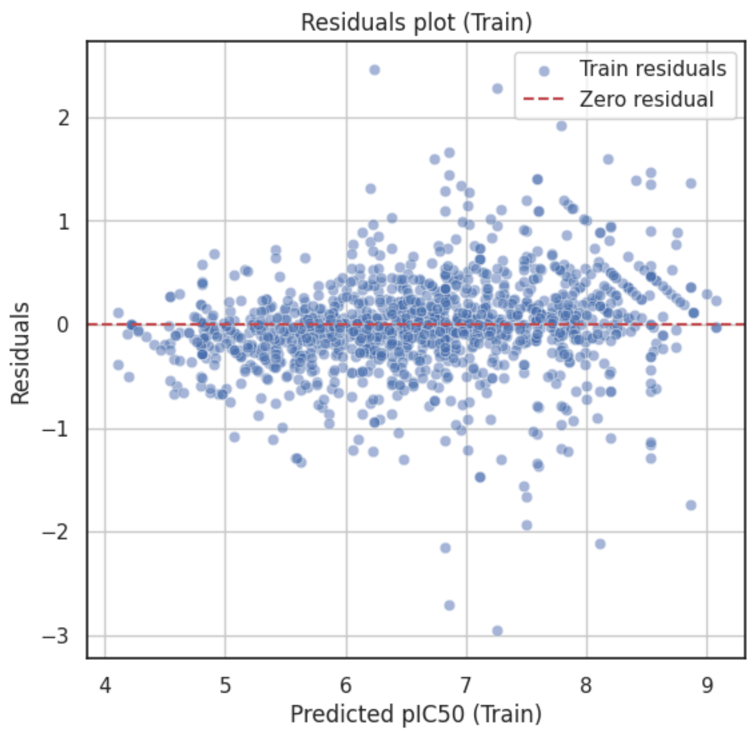
Removing outliers

```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Step 1: Calculate residuals
residuos = y_teste - y_predito1

# Step 2: Calculate the standard deviation of residuals
desvio_padrao_residuos = np.std(residuos)

# Step 3: Define a threshold for outlier detection
limite = 1.5 * desvio_padrao_residuos

# Step 4: Identify outliers
outlier_mask = np.abs(residuos) <= limite

# Step 5: Remove outliers while preserving feature names
x_teste_sem_outliers = x_teste[outlier_mask]
y_teste_sem_outliers = y_teste[outlier_mask]

# Step 6: Train the model on the training data
modelo_sem_outliers = RandomForestRegressor()
modelo_sem_outliers.fit(x_treino, y_treino)

# Step 7: Make predictions on the cleaned test data
y_predito_treino_sem_outliers = modelo_sem_outliers.predict(x_treino)
y_predito_teste_sem_outliers = modelo_sem_outliers.predict(x_teste_sem_outliers)

# Step 8: Calculate performance metrics for the training data without outliers
r2_sem_outliers_treino = modelo_sem_outliers.score(x_treino, y_treino)
r2_sem_outliers_teste = modelo_sem_outliers.score(x_teste_sem_outliers, y_teste_sem_outliers)

rmse_sem_outliers_treino = np.sqrt(mean_squared_error(y_treino, y_predito_treino_sem_outliers))
rmse_sem_outliers_teste = np.sqrt(mean_squared_error(y_teste_sem_outliers, y_predito_teste_sem_outliers))

mae_sem_outliers_treino = mean_absolute_error(y_treino, y_predito_treino_sem_outliers)
mae_sem_outliers_teste = mean_absolute_error(y_teste_sem_outliers, y_predito_teste_sem_outliers)

mse_sem_outliers_treino = mean_squared_error(y_treino, y_predito_treino_sem_outliers)
mse_sem_outliers_teste = mean_squared_error(y_teste_sem_outliers, y_predito_teste_sem_outliers)

# Step 9: Print the results
print("R2 treino (sem outliers): " + str(r2_sem_outliers_treino))
print("R2 teste (sem outliers): " + str(r2_sem_outliers_teste))
print("RMSE treino (sem outliers): " + str(rmse_sem_outliers_treino))
print("RMSE teste (sem outliers): " + str(rmse_sem_outliers_teste))
print("MAE treino (sem outliers): " + str(mae_sem_outliers_treino))
print("MAE teste (sem outliers): " + str(mae_sem_outliers_teste))
print("MSE treino (sem outliers): " + str(mse_sem_outliers_treino))
print("MSE teste (sem outliers): " + str(mse_sem_outliers_teste))
```

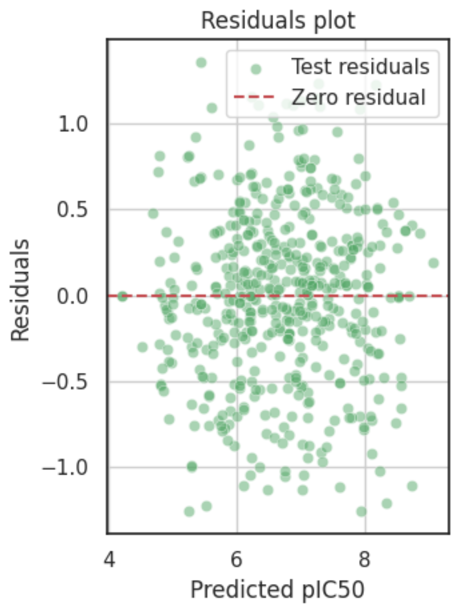
```
R2 treino (sem outliers): 0.8598877305682961
R2 teste (sem outliers): 0.7933290128506474
```

```
RMSE treino (sem outliers): 0.45731411726693616
RMSE teste (sem outliers): 0.4990911961445513
MAE treino (sem outliers): 0.30717378032416925
MAE teste (sem outliers): 0.39077895897152776
MSE treino (sem outliers): 0.20913620185163703
MSE teste (sem outliers): 0.24909202206899897
```

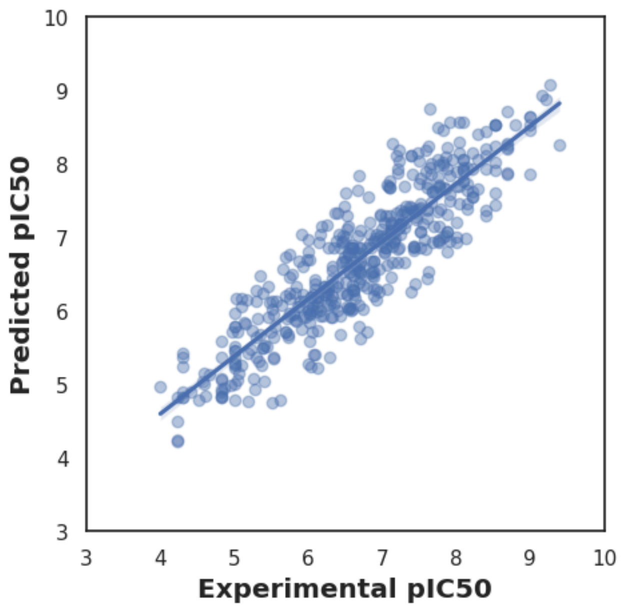
```
# Calculating the residuals for training and testing sets
residuos_treino = y_treino - y_predito_treino_sem_outliers
residuos_teste = y_teste_sem_outliers - y_predito_teste_sem_outliers

#Plotting the residual plots for training and testing sets
plt.figure(figsize=(14, 6))

# Residual plot for the training data
plt.subplot(1, 2, 1)
sns.scatterplot(x=y_predito_treino_sem_outliers, y=residuos_treino, alpha=0.5, color='b', label='Train residuals')
plt.axhline(y=0, color='r', linestyle='--', label='Zero residual')
plt.xlabel('Predicted pIC50 (Train)')
plt.ylabel('Residuals')
plt.title('Residuals plot (Train)')
plt.legend()
plt.grid(True)
```



```
# Residual plot for the testing data
plt.subplot(1, 2, 2)
sns.scatterplot(x=y_predito_teste_sem_outliers, y=residuos_teste, alpha=0.5, color='g', label='Test residuals')
plt.axhline(y=0, color='r', linestyle='--', label='Zero residual')
plt.xlabel('Predicted pIC50')
plt.ylabel('Residuals')
plt.title('Residuals plot')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(color_codes=True)
sns.set_style("white")

# Calculating the residuals

residuos = y_teste - y_predito1

# Calculating the square root of the mean squared error (RMSE)
rmse = np.sqrt(mean_squared_error(y_teste, y_predito1))

# Calculating the standard deviation of the residuals
desvio_padrao_residuos = np.std(residuos)

# Defining a threshold to identify outliers
limite = 1.5 * desvio_padrao_residuos

# Finding the indices of points that are beyond the threshold
indices_outliers = np.where(np.abs(residuos) > limite)

# Removing the outliers from the data
y_teste_sem_outliers = np.delete(y_teste, indices_outliers)
y_predito_sem_outliers = np.delete(y_predito1, indices_outliers)


# Make sure that the arrays y_teste_sem_outliers and y_predito_sem_outliers are of type float64
y_teste_sem_outliers = y_teste_sem_outliers.astype(np.float64)
y_predito_sem_outliers = y_predito_sem_outliers.astype(np.float64)

# Building the scatter plot without outliers
ax = sns.regplot(x=y_teste_sem_outliers, y=y_predito_sem_outliers, scatter_kws={'alpha':0.4})
ax.set_xlabel('Experimental pIC50', fontsize='large', fontweight='bold')
ax.set_ylabel('Predicted pIC50', fontsize='large', fontweight='bold')
ax.set_xlim(3, 10)
ax.set_ylim(3, 10)
ax.figure.set_size_inches(5, 5)
plt.show()
```



```
y_teste_sem_outliers
```

```
array([6.5543958 , 4.30103   , 8.09691001, 6.05749589, 6.04575749,
       7.21467016, 5.        , 7.30103   , 7.4202164 , 6.49349497,
       5.69897   , 7.95860731, 6.26121944, 6.00877392, 6.31605287,
       7.74472749, 5.21467016, 6.30980392, 5.74641971, 4.        ,
       5.9788107 , 6.66756154, 7.09691001, 4.22184875, 7.04575749,
       7.52287875, 7.23657201, 7.04095861, 6.16749109, 8.69897   ,
       9.        , 6.74472749, 4.82390874, 7.39794001, 4.97506303,
       6.67778071, 4.58838029, 4.52287875, 7.24412514, 7.22914799,
       7.13667714, 6.95860731, 8.69897   , 7.52287875, 5.        ,
       7.43179828, 6.55284197, 5.        , 7.48148606, 6.52287875,
       5.84193921, 6.1266794 , 5.4424928 , 7.45099674, 5.50723961,
       6.88605665, 6.93930216, 6.61978876, 4.82390874, 6.36251027,
       5.18708664, 7.88605665, 6.39794001, 6.12493874, 8.39794001,
       7.25181197, 7.60205999, 6.99139983, 8.52287875, 4.82390874,
       6.22184875, 6.43179828, 5.09151498, 7.92081875, 7.08092191,
       7.88605665, 5.50584541, 5.10237291, 7.74472749, 5.32790214,
       6.30891851, 6.01322827, 9.15490196, 6.89962945, 5.1426675 ,
       7.40893539, 7.76955108, 7.38721614, 8.        , 6.2644011 ,
       8.04575749, 4.82390874, 6.46852108, 6.70996539, 8.69897   ,
       7.45593196, 8.21467016, 5.36653154, 8.        , 5.79860288,
       9.26760624, 6.88605665, 5.04000516, 7.19382003, 5.74472749,
       6.69897   , 5.0619809 , 6.12493874, 7.07058107, 7.06048075,
       5.49485002, 7.60205999, 6.33724217, 5.73992861, 8.52287875,
       6.46852108, 5.13924288, 7.30980392, 7.06048075, 5.40893539,
       6.08777794, 5.83803338, 4.82390874, 6.88605665, 5.23210238,
       8.52287875, 6.67778071, 7.67778071, 6.25181197, 6.31875876,
       8.52287875, 6.69897   , 7.49485002, 6.7878124 , 5.93554201,
       8.09691001, 7.86327943, 6.09691001, 4.60205999, 8.09691001,
       8.        , 6.60730305, 4.56863624, 5.        , 7.7212464 ,
       7.60205999, 7.14874165, 5.86327943, 6.79588002, 5.19111413,
       8.09691001, 7.30980392, 5.52287875, 5.74136272, 6.52287875,
       5.66214157, 7.09691001, 7.22184875, 6.50445566, 5.69897   ,
       4.22184875, 7.30103   , 7.61978876, 6.        , 8.09691001,
       6.85387196, 7.58502665, 7.58502665, 7.12493874, 8.69897   ,
       6.55284197, 7.        , 8.09691001, 7.21467016, 6.32330639,
       7.82390874, 6.50031292, 7.45593196, 6.00436481, 7.22184875,
       6.54515514, 4.82390874, 6.60205999, 7.60205999, 6.32697909,
       7.88605665, 6.36855623, 6.00833099, 5.31875876, 7.65757732,
       6.08618615, 7.32790214, 8.22184875, 6.12493874, 5.61978876,
       5.        , 8.        , 7.26760624, 9.39794001, 7.67778071,
       5.        , 7.1426675 , 6.49757288, 7.43179828, 8.69897   ,
       6.82390874, 6.12493874, 7.74472749, 7.56863624, 7.22914799,
       7.43179828, 7.09691001, 6.61083392, 6.8569852 , 7.05354773,
       5.73259358, 6.60205999, 6.56224944, 9.        , 7.15490196,
       7.4436975 , 5.        , 8.52287875, 7.05551733, 6.92081875,
       6.97061622, 6.74472749, 7.92081875, 6.52578374, 6.20065945,
       6.28399666, 6.70996539, 7.23657201, 6.        , 7.76955108,
       9.        , 6.16241156, 8.04575749, 9.22184875, 7.11918641,
       6.71219827, 6.31875876, 6.39040559, 6.97469413, 5.45593196,
       7.65757732, 6.88605665, 6.43179828, 6.07058107, 6.23657201,
       7.76955108, 8.52287875, 4.58502665, 6.63264408, 6.41116827,
       7.76955108, 6.08092191, 6.26760624, 5.02918839, 6.22184875,
       7.02687215, 6.69897   , 6.1739252 , 5.46852108, 6.26760624,
       5.34141628, 5.04095861, 6.09691001, 6.46852108, 6.85387196,
       7.11350927, 5.        , 5.        , 5.30103   , 8.22184875,
       7.40893539, 8.15490196, 4.82390874, 7.79588002, 7.30103   ,
       5.92081875, 5.89997427, 6.53313238, 7.88605665, 6.47495519,
       9.        , 5.1739252 , 6.52287875, 7.38721614, 8.09691001,
```

Hyperparameters optimization

```
# 1. Random Search
```

```
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error
from scipy.stats import randint, uniform

# Define the hyperparameter space for Random Forest
param_dist = {
    'n_estimators': randint(100, 500),
    'max_depth': randint(3, 20),
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 20),
    'bootstrap': [True, False]
}

# Setup Random Search with Random Forest
random_search = RandomizedSearchCV(
    RandomForestRegressor(),
    param_distributions=param_dist,
    n_iter=100,  # Number of iterations for Random Search
    cv=10,
    verbose=1,
    random_state=42,
    n_jobs=1  # Use only one core
)

# Fit Random Search
random_search.fit(x_treino, y_treino)

# Get the best model
modelo_sem_outliers = random_search.best_estimator_

# Calculate predictions
y_predito_treino_sem_outliers = modelo_sem_outliers.predict(x_treino)
y_predito_teste_sem_outliers = modelo_sem_outliers.predict(x_teste_sem_outliers)

# Calculate metrics
r2_sem_outliers_treino = modelo_sem_outliers.score(x_treino, y_treino)
r2_sem_outliers_teste = modelo_sem_outliers.score(x_teste_sem_outliers, y_teste_sem_outliers)
rmse_sem_outliers_treino = np.sqrt(mean_squared_error(y_treino, y_predito_treino_sem_outliers))
rmse_sem_outliers_teste = np.sqrt(mean_squared_error(y_teste_sem_outliers, y_predito_teste_sem_outliers))
mae_sem_outliers_treino = np.mean(np.abs(y_treino - y_predito_treino_sem_outliers))
mae_sem_outliers_teste = np.mean(np.abs(y_teste_sem_outliers - y_predito_teste_sem_outliers))
mse_sem_outliers_treino = mean_squared_error(y_treino, y_predito_treino_sem_outliers)
mse_sem_outliers_teste = mean_squared_error(y_teste_sem_outliers, y_predito_teste_sem_outliers)

# Print the results
print("Best Parameters:", random_search.best_params_)
print("R2 treino (sem outliers):", r2_sem_outliers_treino)
print("R2 teste (sem outliers):", r2_sem_outliers_teste)
print("RMSE treino (sem outliers):", rmse_sem_outliers_treino)
print("RMSE teste (sem outliers):", rmse_sem_outliers_teste)
print("MAE treino (sem outliers):", mae_sem_outliers_treino)
print("MAE teste (sem outliers):", mae_sem_outliers_teste)
print("MSE treino (sem outliers):", mse_sem_outliers_treino)
print("MSE teste (sem outliers):", mse_sem_outliers_teste)
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
Best Parameters: {'bootstrap': True, 'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 8, 'n_estimators': 108}
R2 treino (sem outliers): 0.8106597727170946
R2 teste (sem outliers): 0.7755520028351189
RMSE treino (sem outliers): 0.5316159120520643
RMSE teste (sem outliers): 0.520113371768752
MAE treino (sem outliers): 0.38739321876276406
MAE teste (sem outliers): 0.4097625897331498
MSE treino (sem outliers): 0.2826154779469481
MSE teste (sem outliers): 0.27051791949266013
```

```
# Bayesian aproach
```

```
!pip install scikit-optimize
```

```
Collecting scikit-optimize
  Downloading scikit_optimize-0.10.2-py2.py3-none-any.whl.metadata (9.7 kB)
```

```
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.4.2)
Collecting pyaml>=16.9 (from scikit-optimize)
  Downloading pyaml-24.12.1-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: numpy>=1.20.3 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.26.4)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.6.0)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from scikit-optimize) (24.2)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from pyaml>=16.9->scikit-optimize) (6.0.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikit-opt
Downloading scikit_optimize-0.10.2-py2.py3-none-any.whl (107 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 107.8/107.8 kB 3.7 MB/s eta 0:00:00
Downloading pyaml-24.12.1-py3-none-any.whl (25 kB)
Installing collected packages: pyaml, scikit-optimize
Successfully installed pyaml-24.12.1 scikit-optimize-0.10.2
```

```python
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from skopt import BayesSearchCV  # Import BayesSearchCV for Bayesian optimization
from skopt.space import Integer, Real, Categorical

# Define the hyperparameter space for Bayesian Search with Random Forest
param_space = {
    'n_estimators': Integer(100, 500),
    'max_depth': Integer(3, 20),
    'min_samples_split': Integer(2, 20),
    'min_samples_leaf': Integer(1, 20),
    'bootstrap': Categorical([True, False])
}

# Setup Bayesian Search with Random Forest
bayes_search = BayesSearchCV(
    RandomForestRegressor(),
    search_spaces=param_space,
    n_iter=50,  # Number of iterations for the Bayesian search
    cv=10,
    verbose=1,
    random_state=42,
    n_jobs=1  # Run in single thread
)

try:
    # Fit Bayesian Search
    bayes_search.fit(x_treino, y_treino)
except Exception as e:
    print("Error during fitting:", str(e))

# Get the best model from Bayesian optimization
modelo_bayesiano_sem_outliers = bayes_search.best_estimator_

# Calculate predictions
y_predito_treino_sem_outliers = modelo_bayesiano_sem_outliers.predict(x_treino)
y_predito_teste_sem_outliers = modelo_bayesiano_sem_outliers.predict(x_teste_sem_outliers)

# Calculate metrics
r2_sem_outliers_treino = modelo_bayesiano_sem_outliers.score(x_treino, y_treino)
r2_sem_outliers_teste = modelo_bayesiano_sem_outliers.score(x_teste_sem_outliers, y_teste_sem_outliers)
rmse_sem_outliers_treino = np.sqrt(mean_squared_error(y_treino, y_predito_treino_sem_outliers))
rmse_sem_outliers_teste = np.sqrt(mean_squared_error(y_teste_sem_outliers, y_predito_teste_sem_outliers))
mae_sem_outliers_treino = np.mean(np.abs(y_treino - y_predito_treino_sem_outliers))
mae_sem_outliers_teste = np.mean(np.abs(y_teste_sem_outliers - y_predito_teste_sem_outliers))
mse_sem_outliers_treino = mean_squared_error(y_treino, y_predito_treino_sem_outliers)
mse_sem_outliers_teste = mean_squared_error(y_teste_sem_outliers, y_predito_teste_sem_outliers)

# Print the results
print("Best Parameters:", bayes_search.best_params_)
print("R2 treino (sem outliers):", r2_sem_outliers_treino)
print("R2 teste (sem outliers):", r2_sem_outliers_teste)
print("RMSE treino (sem outliers):", rmse_sem_outliers_treino)
print("RMSE teste (sem outliers):", rmse_sem_outliers_teste)
print("MAE treino (sem outliers):", mae_sem_outliers_treino)
print("MAE teste (sem outliers):", mae_sem_outliers_teste)
print("MSE treino (sem outliers):", mse_sem_outliers_treino)
print("MSE teste (sem outliers):", mse_sem_outliers_teste)
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Fitting 10 folds for each of 1 candidates, totalling 10 fits
Best Parameters: OrderedDict([('bootstrap', True), ('max_depth', 19), ('min_samples_leaf', 1), ('min_samples_split', 12), ('n_est
R2 treino (sem outliers): 0.7805705106626772
R2 teste (sem outliers): 0.7557090932363606
RMSE treino (sem outliers): 0.5723003621039547
RMSE teste (sem outliers): 0.54261750849928
MAE treino (sem outliers): 0.425600464298761
MAE teste (sem outliers): 0.4299008119425324
MSE treino (sem outliers): 0.32752770446431767
```

```
# 3.Grid Search
```

```
!pip install --upgrade scikit-learn joblib
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.6.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
```

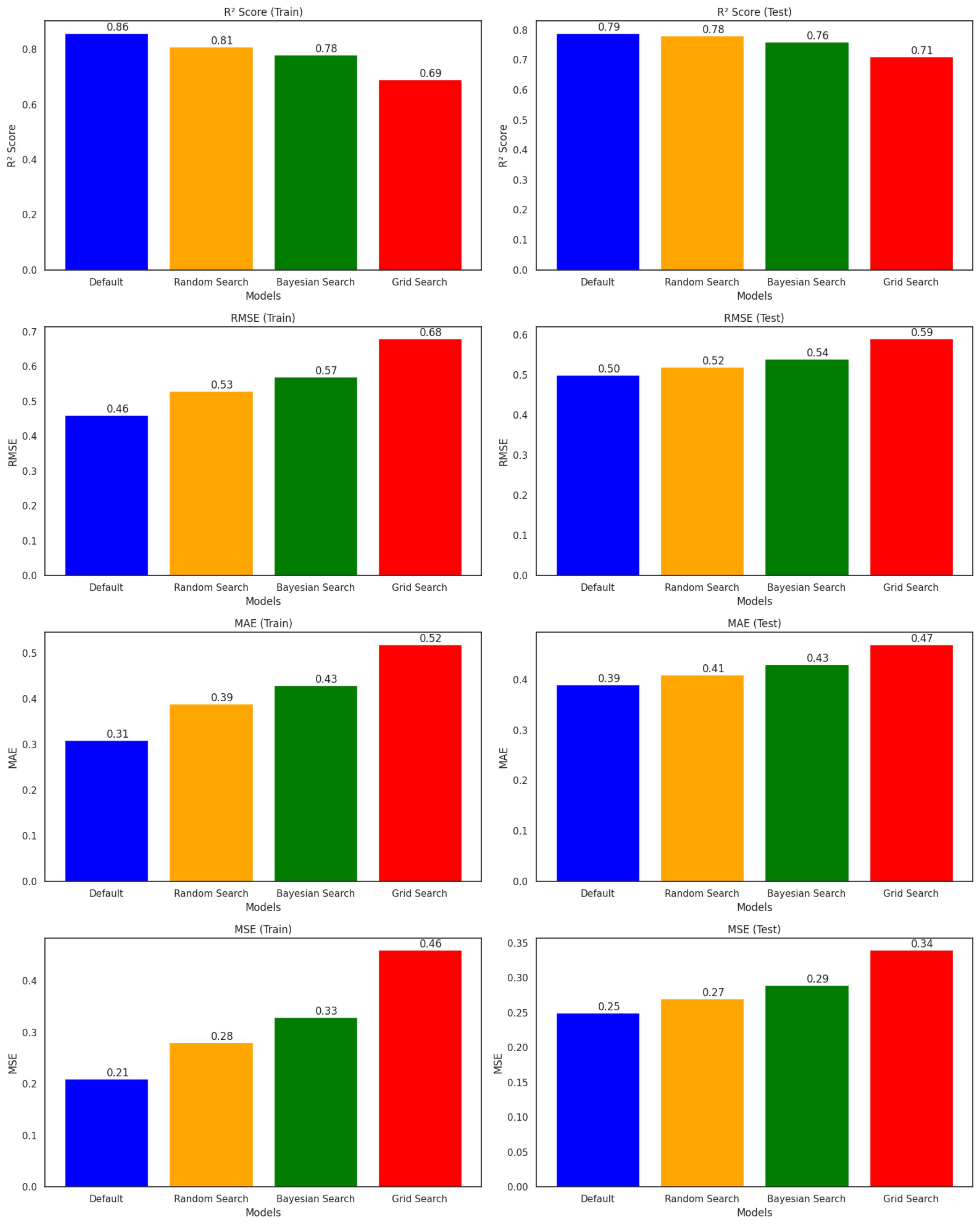
```python
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error

# Define a smaller hyperparameter grid for Random Forest
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 7],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 4],
    'bootstrap': [True, False]
}

# Initialize GridSearchCV with n_jobs set to 1
grid_search = GridSearchCV(
    RandomForestRegressor(),
    param_grid=param_grid,
    cv=10,
    verbose=1,
    n_jobs=1  # Use a single core
)
```

```
# Fit Grid Search
grid_search.fit(x_treino, y_treino)

# Get the best model
modelo_grid_search = grid_search.best_estimator_

# Calculate predictions
y_predito_treino_grid_search = modelo_grid_search.predict(x_treino)
y_predito_teste_grid_search = modelo_grid_search.predict(x_teste_sem_outliers)

# Calculate metrics
r2_grid_search_treino = modelo_grid_search.score(x_treino, y_treino)
r2_grid_search_teste = modelo_grid_search.score(x_teste_sem_outliers, y_teste_sem_outliers)
rmse_grid_search_treino = np.sqrt(mean_squared_error(y_treino, y_predito_treino_grid_search))
rmse_grid_search_teste = np.sqrt(mean_squared_error(y_teste_sem_outliers, y_predito_teste_grid_search))
mae_grid_search_treino = np.mean(np.abs(y_treino - y_predito_treino_grid_search))
mae_grid_search_teste = np.mean(np.abs(y_teste_sem_outliers - y_predito_teste_grid_search))
mse_grid_search_treino = mean_squared_error(y_treino, y_predito_treino_grid_search)
mse_grid_search_teste = mean_squared_error(y_teste_sem_outliers, y_predito_teste_grid_search)

# Print the results
print("Best Parameters:", grid_search.best_params_)
print("R2 treino (Grid Search):", r2_grid_search_treino)
print("R2 teste (Grid Search):", r2_grid_search_teste)
print("RMSE treino (Grid Search):", rmse_grid_search_treino)
print("RMSE teste (Grid Search):", rmse_grid_search_teste)
print("MAE treino (Grid Search):", mae_grid_search_treino)
print("MAE teste (Grid Search):", mae_grid_search_teste)
print("MSE treino (Grid Search):", mse_grid_search_treino)
print("MSE teste (Grid Search):", mse_grid_search_teste)
```

```
Fitting 10 folds for each of 32 candidates, totalling 320 fits
Best Parameters: {'bootstrap': True, 'max_depth': 7, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}
R2 treino (Grid Search): 0.6916275561199599
R2 teste (Grid Search): 0.714977776911293
RMSE treino (Grid Search): 0.6784444683058839
RMSE teste (Grid Search): 0.5861105161982831
MAE treino (Grid Search): 0.5199303969130796
MAE teste (Grid Search): 0.46708262241586734
MSE treino (Grid Search): 0.4602868965748535
MSE teste (Grid Search): 0.3435255371982179
```

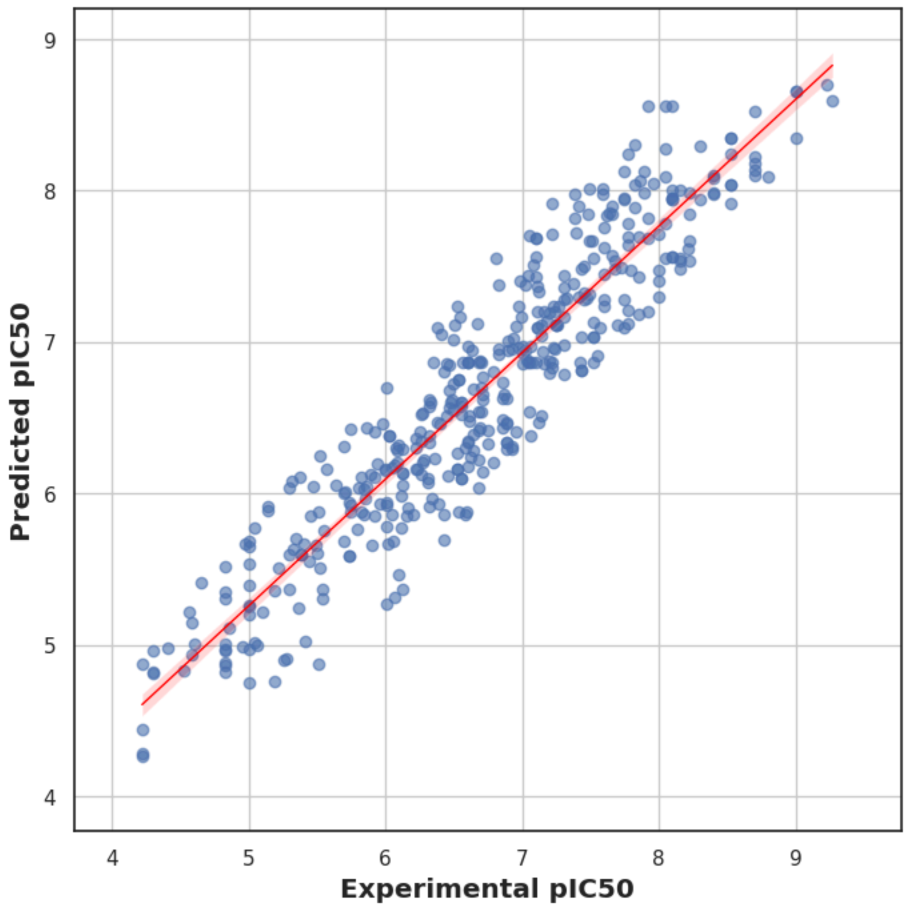
Comparing the Normal, Random Search, Bayesian and Grid Search tuning models

```
import matplotlib.pyplot as plt
import numpy as np

# Replace these placeholder values with actual results
results = {
    'Default': {
        'Train': {
            'R² Score': [0.86],
            'RMSE': [0.46],
            'MAE': [0.31],
            'MSE': [0.21]
        },
        'Test': {
            'R² Score': [0.79],
            'RMSE': [0.50],
            'MAE': [0.39],
            'MSE': [0.25]
        }
    },
    'Random Search': {
        'Train': {
            'R² Score': [0.81],
            'RMSE': [0.53],
            'MAE': [0.39],
            'MSE': [0.28]
        },
        'Test': {
            'R² Score': [0.78],
            'RMSE': [0.52],
            'MAE': [0.41],
            'MSE': [0.27]
        }
    },
    'Bayesian Search': {
        'Train': {
            'R² Score': [0.78],
            'RMSE': [0.57],
            'MAE': [0.43],
            'MSE': [0.33]
        },
        'Test': {
            'R² Score': [0.76],
            'RMSE': [0.54],
            'MAE': [0.43],
            'MSE': [0.29]
        }
    },
    'Grid Search': {
        'Train': {
            'R² Score': [0.69],
            'RMSE': [0.68],
            'MAE': [0.52],
            'MSE': [0.46]
        },
        'Test': {
            'R² Score': [0.71],
            'RMSE': [0.59],
            'MAE': [0.47],
            'MSE': [0.34]
        }
    }
}


# Create figure and axis objects for training and testing comparisons
fig, axs = plt.subplots(4, 2, figsize=(16, 20))

# Define the metrics
metrics = ['R² Score', 'RMSE', 'MAE', 'MSE']
colors = ['blue', 'orange', 'green', 'red']
datasets = ['Train', 'Test']

# Plot each metric for training and testing datasets
for i, metric in enumerate(metrics):
    for j, dataset in enumerate(datasets):
        ax = axs[i, j]
        model_names = list(results.keys())
        metric_values = [results[model][dataset][metric][0] for model in model_names]

        bars = ax.bar(model_names, metric_values, color=colors)
        ax.set_title(f'{metric} ({dataset})')
        ax.set_ylabel(metric)
        ax.set_xlabel('Models')

        # Adding value labels on the bars
        for bar in bars:
            yval = bar.get_height()
            ax.text(bar.get_x() + bar.get_width()/2.0, yval, f'{yval:.2f}', va='bottom')  # va: vertical alignment

plt.tight_layout()
plt.show()
```


R² Score (Train)
R² Score
0.86
0.81
0.78
0.69
R² Score (Test)
0.79
0.78
0.76
0.71
RMSE (Train)
RMSE
0.46
0.53
0.57
0.68
RMSE (Test)
0.50
0.52
0.54
0.59
MAE (Train)
MAE
0.31
0.39
0.43
0.52
MAE (Test)
0.39
0.41
0.43
0.47
MSE (Train)
MSE
0.21
0.28
0.33
0.46
MSE (Test)
0.25
0.27
0.29
0.34
Default
Random Search
Bayesian Search
Grid Search
Models

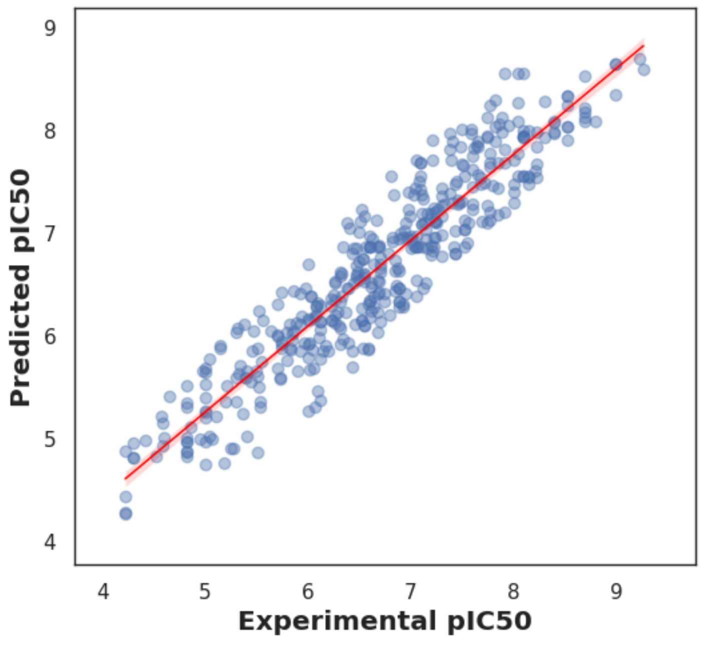
Random Search was selected

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

# Ensure seaborn and matplotlib styles are set
sns.set(color_codes=True)
sns.set_style("white")

# Get predictions from the Random Search optimized model
y_predito_teste_random = modelo_sem_outliers.predict(x_teste_sem_outliers)

# Calculate residuals
residuos = y_teste_sem_outliers - y_predito_teste_random

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mean_squared_error(y_teste_sem_outliers, y_predito_teste_random))

# Calculate standard deviation of residuals
desvio_padrao_residuos = np.std(residuos)

# Define a threshold to identify outliers (2.5 * standard deviation)
limite = 1.5 * desvio_padrao_residuos

# Find indices of points beyond the threshold
indices_outliers = np.where(np.abs(residuos) > limite)

# Remove outliers from data
y_teste_clean = np.delete(y_teste_sem_outliers, indices_outliers)
y_predito_clean = np.delete(y_predito_teste_random, indices_outliers)

# Ensure that arrays are of type float64
y_teste_clean = y_teste_clean.astype(np.float64)
y_predito_clean = y_predito_clean.astype(np.float64)

# Build the scatter plot without outliers
plt.figure(figsize=(8, 8))
ax = sns.regplot(x=y_teste_clean, y=y_predito_clean, scatter_kws={'alpha': 0.6}, line_kws={'color': 'red', 'linewidth': 1})

# Set labels and limits
ax.set_xlabel('Experimental pIC50', fontsize='large', fontweight='bold')
ax.set_ylabel('Predicted pIC50', fontsize='large', fontweight='bold')
ax.set_xlim(min(y_teste_clean) - 0.5, max(y_teste_clean) + 0.5)  # Adjust limits dynamically
ax.set_ylim(min(y_predito_clean) - 0.5, max(y_predito_clean) + 0.5)  # Adjust limits dynamically

# Add grid for better visualization
plt.grid(True)

# Show the plot
plt.show()
```



```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

# Ensure seaborn and matplotlib styles are set
sns.set(color_codes=True)
sns.set_style("white")

# Get predictions from the Random Search optimized model
y_predito_teste_random = modelo_sem_outliers.predict(x_teste_sem_outliers)

# Calculate residuals
residuos = y_teste_sem_outliers - y_predito_teste_random

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mean_squared_error(y_teste_sem_outliers, y_predito_teste_random))

# Calculate standard deviation of residuals
desvio_padrao_residuos = np.std(residuos)

# Define a threshold to identify outliers (2.0 * standard deviation)
limite = 1.5 * desvio_padrao_residuos

# Find indices of points beyond the threshold
indices_outliers = np.where(np.abs(residuos) > limite)

# Remove outliers from data
y_teste_clean = np.delete(y_teste_sem_outliers, indices_outliers)
y_predito_clean = np.delete(y_predito_teste_random, indices_outliers)

# Ensure that arrays are of type float64
y_teste_clean = y_teste_clean.astype(np.float64)
y_predito_clean = y_predito_clean.astype(np.float64)

# Build the scatter plot with trend line
plt.figure(figsize=(6, 6))
ax = sns.regplot(
    x=y_teste_clean,
    y=y_predito_clean,
    scatter_kws={'alpha': 0.4},  # Adjust transparency of scatter points
    line_kws={'color': 'red', 'linewidth': 1}  # Trend line color and thickness
)
```

```
# Set axis labels
ax.set_xlabel('Experimental pIC50', fontsize='large', fontweight='bold')
ax.set_ylabel('Predicted pIC50', fontsize='large', fontweight='bold')

# Set axis limits to keep the plot square
xlim = (min(y_teste_clean) - 0.5, max(y_teste_clean) + 0.5)
ylim = (min(y_predito_clean) - 0.5, max(y_predito_clean) + 0.5)
ax.set_xlim(xlim)
ax.set_ylim(ylim)

# Adjust aspect ratio to be square
ax.set_aspect('equal', adjustable='box')

# Customize spines to keep only vertical and horizontal lines
ax.spines['top'].set_visible(True)
ax.spines['right'].set_visible(True)
ax.spines['left'].set_visible(True)
ax.spines['bottom'].set_visible(True)

# Optionally, adjust the appearance of the spines
ax.spines['top'].set_linewidth(1)    # Top spine width
ax.spines['right'].set_linewidth(1)  # Right spine width
ax.spines['left'].set_linewidth(1)   # Left spine width
ax.spines['bottom'].set_linewidth(1) # Bottom spine width

# Show the plot
plt.show()
```



Start coding or generate with AI.

SHAP VALUES ANALYSIS

```
## Step 1: Installing SHAP VALUES
## Step 3: Import and view the pre-treated database
## step 3: Defining the predictor and response variables
## step 4: Selection of important variables
## Step 5: splitting training and testing data
## Step 6: Training the Top 5 machine learning models
## Step 7: Shap Values
## Step 8: Comparison of important features
```

Screening of FDA-Approved drugs with optimized model

```
### 3.1: Importing the dataset

from google.colab import files
uploaded = files.upload()
```

```
Escolher arquivos DESCRITO…BCHEM.csv
• DESCRITORES PUBCHEM.csv(text/csv) - 2821823 bytes, last modified: 22/07/2024 - 100% done
Saving DESCRITORES PUBCHEM.csv to DESCRITORES PUBCHEM (1).csv
```

```
### 3.2: Visualizing the imported dataset

import pandas as pd
df2 = pd.read_csv("DESCRITORES PUBCHEM.csv")
display (df2)
```

	Unnamed: 0	Name	PubchemFP0	PubchemFP1	PubchemFP2	PubchemFP3	PubchemFP4	PubchemFP5	PubchemFP6	PubchemFP7	..
0	0	ZINC000001530427	1	0	0	0	0	0	0	0	.
1	1	ZINC000003807804	1	1	1	0	0	0	0	0	.
2	2	ZINC000000120286	1	1	0	0	0	0	0	0	.
3	3	ZINC000242548690	1	1	1	1	0	0	0	0	.
4	4	ZINC000000008492	1	0	0	0	0	0	0	0	.
...	...	...	...	...	...	...	...	...	...	...	.
1571	1571	ZINC000022010387	1	1	1	0	0	0	0	0	.
1572	1572	ZINC000022448097	0	0	0	0	0	0	0	0	.
1573	1573	ZINC000100370145	0	0	0	0	0	0	0	0	.
1574	1574	ZINC000059111167	1	1	0	0	0	0	0	0	.
1575	1575	ZINC000169621219	1	1	1	0	0	0	0	0	.

1576 rows × 883 columns

```
## Removing non-informative variable
df2 = df2.drop("Unnamed: 0", axis = 1)
df2 = df2.drop("Name", axis = 1)
```

```
df2.head()
```

	PubchemFP0	PubchemFP1	PubchemFP2	PubchemFP3	PubchemFP4	PubchemFP5	PubchemFP6	PubchemFP7	PubchemFP8	PubchemFP9	...	Pubcl
0	1	0	0	0	0	0	0	0	0	1	...	
1	1	1	1	0	0	0	0	0	0	1	...	
2	1	1	0	0	0	0	0	0	0	1	...	
3	1	1	1	1	0	0	0	0	0	1	...	
4	1	0	0	0	0	0	0	0	0	1	...	

5 rows × 881 columns

```
import pandas as pd

# Assuming 'x_treino' is your training DataFrame and 'df2' is the external DataFrame

# Getting the common columns in the same order as x_treino

common_columns = list(x_treino.columns.intersection(df2.columns))  # Common columns in the same order as x_treino

# Check if we have all the necessary columns
if set(common_columns) != set(x_treino.columns):
    raise ValueError("As colunas em df2 não correspondem completamente às colunas em x_treino.")

# Reorganizing the columns in the external DataFrame to match the order of the columns in the training set
external_data_aligned = df2[common_columns]

# Making predictions using the aligned columns in the external data
predictions = modelo_sem_outliers.predict(external_data_aligned)
```

```
# Saving the predicted dataset (predictions)

import pandas as pd

# Convert predictions to a DataFrame (if it's not already)
predicted_df = pd.DataFrame(predictions, columns=['Predicted_Column_Name'])  # Replace 'Predicted_Column_Name' with the appropriate name
```

```
# Save the predicted data as a CSV file
predicted_df.to_csv('FDA RF LEPRA.csv', index=False)

# Trigger the download of the file in Google Colab
from google.colab import files
files.download('FDA RF LEPRA.csv')
```

Saving the model

```
import joblib

# Save the model
joblib.dump(modelo_sem_outliers, 'FLEPRA modelo_sem_outliers-RF F.pkl')
```

```
['FLEPRA modelo_sem_outliers-RF F.pkl']
```

```
# Installing the shape value library
!pip install shap
```

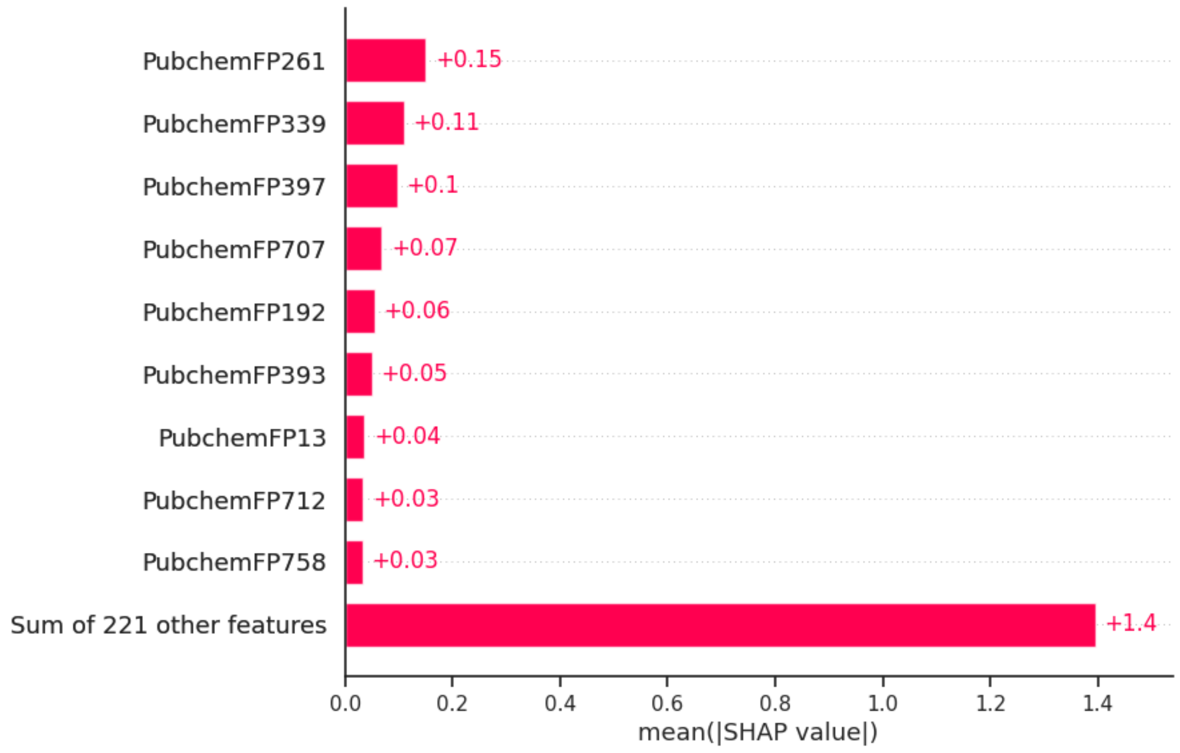
```
Requirement already satisfied: shap in /usr/local/lib/python3.10/dist-packages (0.46.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from shap) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from shap) (1.13.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from shap) (1.6.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-packages (from shap) (4.67.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packages (from shap) (24.2)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.10/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from shap) (3.1.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->shap) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2024.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->shap) (1.17
```

```
# To calculate SHAP values for the model, we need to create an "Explainer" object
# the Explainer object will be used to evaluate a sample or dataset with

# Importing the SHAP library
import shap
```

```
# Adjusting the explainer
explainer = shap.Explainer(modelo_sem_outliers.predict, x_teste_sem_outliers)
# Calculating the SHAP values - it takes a while
shap_values = explainer(x_teste_sem_outliers, max_evals=600)
```

```
PermutationExplainer explainer: 496it [05:03,  1.63it/s]
```

```
# If we simply want the feature importances as determined by the SHAP algorithm,
# we need to get the average mean value of each feature
import numpy as np
from scipy.special import softmax

def print_feature_importances_shap_values(shap_values, features):
    '''
    Prints the feature importances based on SHAP values in an ordered way
    shap_values -> The SHAP values calculated from a shap.Explainer object
    features -> The name of the features, on the order presented to the explainer
    '''
    # Calculates the feature importance (mean absolute shap value) for each feature
    importances = []
    for i in range(shap_values.values.shape[1]):
        importances.append(np.mean(np.abs(shap_values.values[:, i])))
    # Calculates the normalized version
    importances_norm = softmax(importances)
    # Organize the importances and columns in a dictionary
    feature_importances = {fea: imp for imp, fea in zip(importances, features)}
    feature_importances_norm = {fea: imp for imp, fea in zip(importances_norm, features)}
    # Sorts the dictionary
    feature_importances = {k: v for k, v in sorted(feature_importances.items(), key=lambda item: item[1], reverse = True)}
    feature_importances_norm= {k: v for k, v in sorted(feature_importances_norm.items(), key=lambda item: item[1], reverse = True)}
    # Prints the feature importances
    for k, v in feature_importances.items():
        print(f"{k} -> {v:.4f} (softmax = {feature_importances_norm[k]:.4f})")


# Analyzing the global effect of the features:
shap.plots.bar(shap_values)
```



```
# Feature importance summary chart: OPTION 1
shap.summary_plot(shap_values)
```