problemasderegresion

August 29, 2024

Paola Félix Torres A00227869

1 ==========

1.1 EJERCICIO 1

2 ==========

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

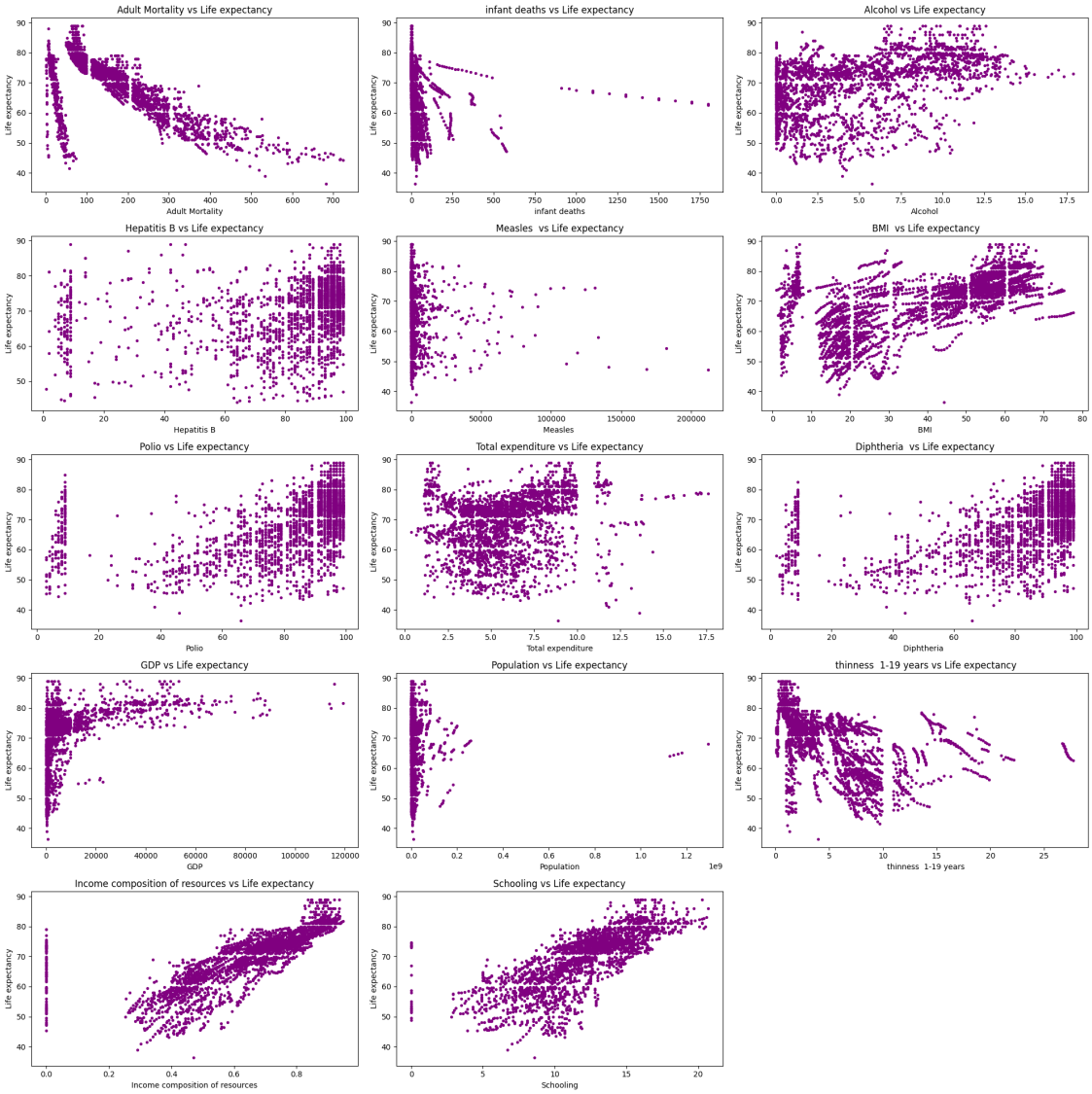
[2]:
```
file_path = file_path = "C:
 ↪\\Users\\pfeli\\Downloads\\ACT1MACHINE\\life_expectancy_data.csv"
data = pd.read_csv(file_path)
```

[3]:
```
#view all data titles
print(data.columns)
```

```
Index(['Country', 'Year', 'Status', 'Life expectancy ', 'Adult Mortality',
       'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B',
       'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
       'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population',
       ' thinness  1-19 years', ' thinness 5-9 years',
       'Income composition of resources', 'Schooling'],
      dtype='object')
```

[129]:
```
variables_independientes = [
    'Adult Mortality', 'infant deaths', 'Alcohol', 'Hepatitis B',
    'Measles ', ' BMI ', 'Polio', 'Total expenditure', 'Diphtheria ',
    'GDP', 'Population', ' thinness  1-19 years', 'Income composition of␣
 ↪resources', 'Schooling'
]

data_filtered = data[['Life expectancy '] + variables_independientes]
```

[130]:
```
#Grafica cada variable predictora vs la variable de respuesta asignadas a tu␣
 ↪número de matrícula.
plt.figure(figsize=(20, 20))
for i, column in enumerate(variables_independientes, 1):
    plt.subplot(5, 3, i)
    plt.scatter(data_filtered[column], data_filtered['Life expectancy '], s=8,␣
 ↪color='purple')
    plt.title(f'{column} vs Life expectancy')
    plt.xlabel(column)
    plt.ylabel('Life expectancy')
plt.tight_layout()
plt.show()
```



[148]:
```
#Implementa la fórmula directa para calcular los coeficientes de un modelo de␣
 ↪regresión lineal,
# y obtenga con ella el modelo que corresponde a la variable de respuesta y las␣
 ↪variables predictoras
# asignadas a tu número de matrícula.

data_filtered_clean = data_filtered.dropna()

X = data_filtered_clean[variables_independientes].values
y = data_filtered_clean['Life expectancy '].values

X = np.c_[np.ones(X.shape[0]), X]
X_transpose = X.T
beta = np.linalg.inv(X_transpose @ X) @ X_transpose @ y

print("Coeficientes del modelo: ", beta)
```

```
Coeficientes del modelo:  [ 5.35782141e+01 -2.91735454e-02 -2.06761739e-03
-1.72800489e-01
 -8.61927089e-04  8.61731917e-06  3.88054457e-02  8.99821933e-03
 -2.59249090e-02  2.07080369e-02  6.75333212e-05  2.76190754e-09
 -8.64645239e-02  1.11081418e+01  8.82776248e-01]
```

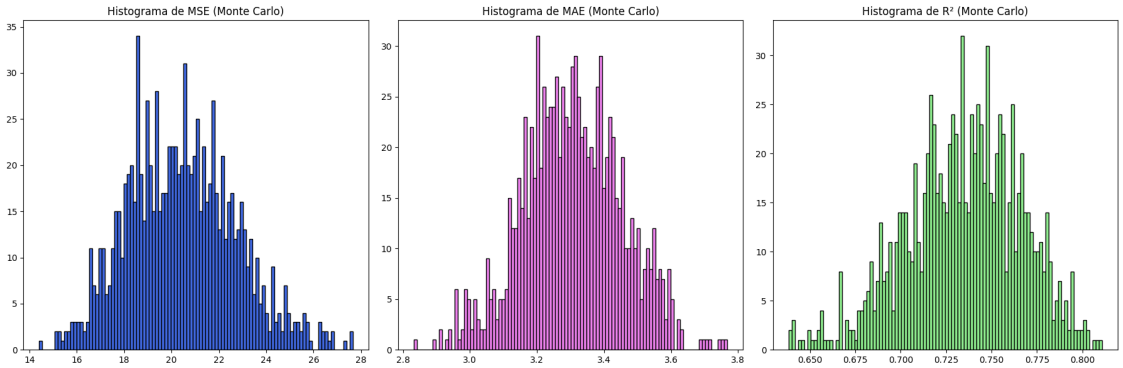
[150]:
```
#Evalúa con validación cruzada de k-pliegues tu modelo, calculando los valores␣
 ↪de R2, MSE y MAE.

from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

n_folds = 5
kf = KFold(n_splits=n_folds, shuffle=True)

mse_cv = []
mae_cv = []
r2_cv = []
for train_index, test_index in kf.split(X):
    # Fase de entrenamiento
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Ajusta el modelo usando tu propia fórmula
    beta_cv = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train

    # Fase de prueba
    y_pred_cv = X_test @ beta_cv

    # Cálculo de las métricas y almacenamiento
```

```
    mse_cv.append(mean_squared_error(y_test, y_pred_cv))
    mae_cv.append(mean_absolute_error(y_test, y_pred_cv))
    r2_cv.append(r2_score(y_test, y_pred_cv))

    print(f'  MSE: {mse_cv[-1]}')
    print(f'  MAE: {mae_cv[-1]}')
    print(f'  R^2: {r2_cv[-1]}')
    print('---')

# Imprime las métricas promedio de la validación cruzada
print('MSE promedio:', np.mean(mse_cv))
print('MAE promedio:', np.mean(mae_cv))
print('R^2 promedio:', np.mean(r2_cv))
```

```
  MSE: 18.2923906069606
  MAE: 3.1020253512477667
  R^2: 0.7289155938641572
---
  MSE: 18.282538603187657
  MAE: 3.2005159259511786
  R^2: 0.7526240267499574
---
  MSE: 21.518383638166927
  MAE: 3.191514746483075
  R^2: 0.7387049029254908
---
  MSE: 17.428163781042322
  MAE: 3.011177440483889
  R^2: 0.7821833664413371
---
  MSE: 17.10055043654043
  MAE: 3.0168706324220422
  R^2: 0.7920550349269769
---
MSE promedio: 18.524405413179586
MAE promedio: 3.1044208193175904
R^2 promedio: 0.7588965849815839
```

[54]:
```
#Utiliza validación cruzada de Monte Carlo con 1000 iteraciones para encontrar␣
 ↪histogramas de R2, MSE y MAE.

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import ShuffleSplit

n_iterations = 1000
shuffle_split = ShuffleSplit(n_splits=n_iterations, test_size=0.2,␣
 ↪random_state=1)
```

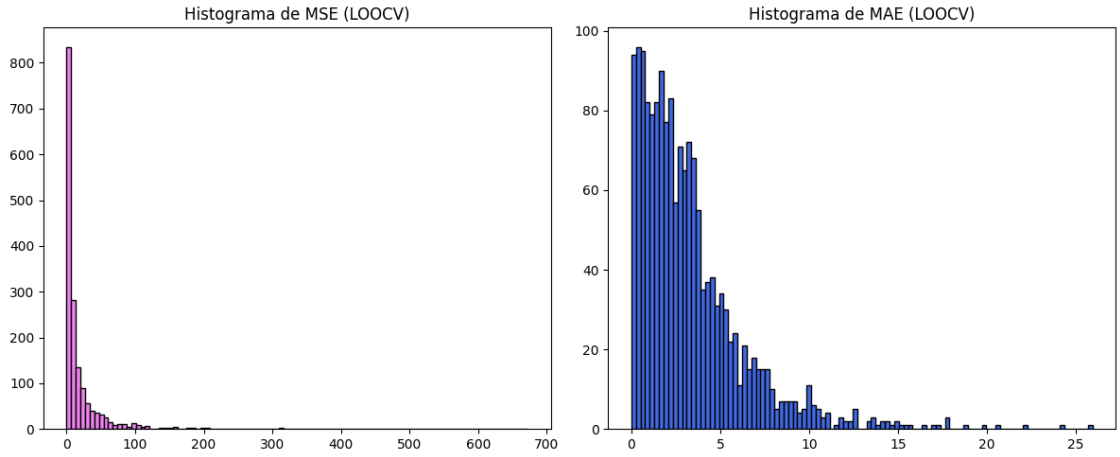
```
mse_mc = []
mae_mc = []
r2_mc = []

for train_index, test_index in shuffle_split.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    beta_mc = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train

    y_pred_mc = X_test @ beta_mc

    mse_mc.append(mean_squared_error(y_test, y_pred_mc))
    mae_mc.append(mean_absolute_error(y_test, y_pred_mc))
    r2_mc.append(r2_score(y_test, y_pred_mc))

plt.figure(figsize=(18, 6))

plt.subplot(1, 3, 1)
plt.hist(mse_mc, bins=100, color='royalblue', edgecolor='black')
plt.title('Histograma de MSE (Monte Carlo)')

plt.subplot(1, 3, 2)
plt.hist(mae_mc, bins=100, color='violet', edgecolor='black')
plt.title('Histograma de MAE (Monte Carlo)')

plt.subplot(1, 3, 3)
plt.hist(r2_mc, bins=100, color='lightgreen', edgecolor='black')
plt.title('Histograma de R² (Monte Carlo)')

plt.tight_layout()
plt.show()

print('MSE promedio:', np.mean(mse_mc))
print('MAE promedio:', np.mean(mae_mc))
print('R^2 promedio:', np.mean(r2_mc))
```




```
MSE promedio: 20.42654258062807
MAE promedio: 3.306392508864157
R^2 promedio: 0.7345693322221355
```

[52]:
```python
#Utiliza el método de validación cruzada asignado a tu matrícula para mostrar los histogramas de MSE y MAE.
#LOOCV

from sklearn.model_selection import LeaveOneOut

loo = LeaveOneOut()
mse_loo = []
mae_loo = []

for train_index, test_index in loo.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Calcular coeficientes usando el conjunto de entrenamiento
    beta = calculate_coefficients(X_train, y_train)

    # Predecir en el conjunto de prueba
    y_pred = predict(X_test, beta)

    # Calcular métricas
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)

    mse_loo.append(mse)
    mae_loo.append(mae)

# Crear histogramas
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
```

```
plt.hist(mse_loo, bins=100, color='violet', edgecolor='black')
plt.title('Histograma de MSE (LOOCV)')

plt.subplot(1, 2, 2)
plt.hist(mae_loo, bins=100, color='royalblue', edgecolor='black')
plt.title('Histograma de MAE (LOOCV)')

plt.tight_layout()
plt.show()

print('MSE promedio:', np.mean(mse_loo))
print('MAE promedio:', np.mean(mae_loo))
```




```
MSE promedio: 20.247828756528826
MAE promedio: 3.29600090071052
```

¿Los histogramas son distintos a los obtenidos con el método de Monte Carlo? Sí, ya que, a diferencia de Monte Carlo, LOOCV tiende a ser más sensible a valores atípicos, lo que puede dar lugar a histogramas sesgados y menos uniformes. En cambio, Monte Carlo tiende a suavizar la variabilidad de los datos debido a la aleatoriedad en las particiones, produciendo distribuciones más normales.

[56]:
```
#Agrega al conjunto de datos columnas que representen los cuadrados de las␣
 ↪variables predictoras
#así como los productos entre pares de variables. Repita los pasos 1, 2 y 3␣
 ↪pero con este nuevo conjunto de datos.

def create_extended_dataframe(df, columns_of_interest):
    squared_features = {f'{col}^2': df[col] ** 2 for col in columns_of_interest}

    interaction_features = {}
    num_vars = len(columns_of_interest)
```

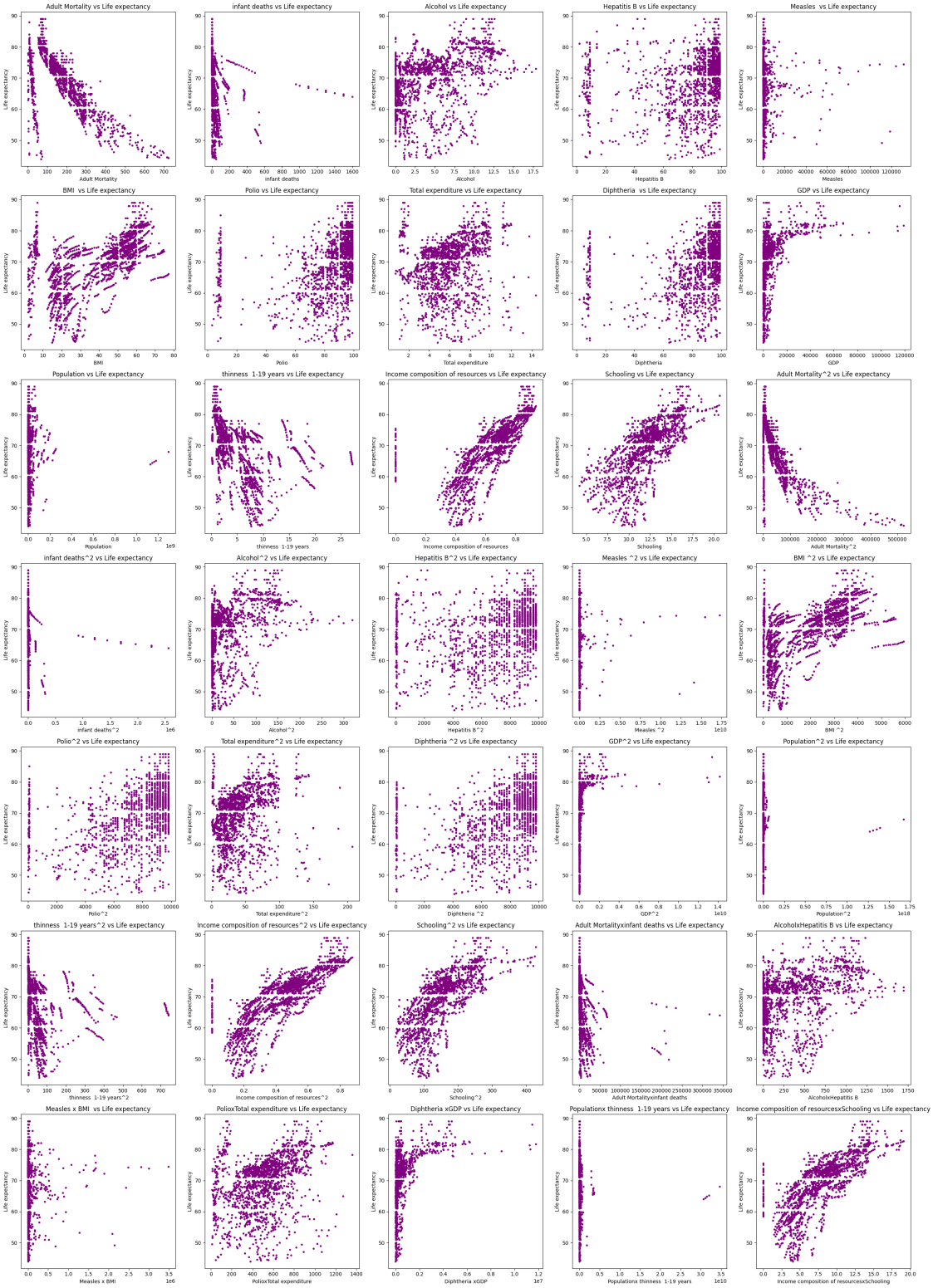
```
    for i in range(0, num_vars, 2):
        if i + 1 < num_vars:
            var1 = columns_of_interest[i]
            var2 = columns_of_interest[i + 1]
            interaction_features[f'{var1}x{var2}'] = df[var1] * df[var2]

    squared_df = pd.DataFrame(squared_features, index=df.index)
    interaction_df = pd.DataFrame(interaction_features, index=df.index)

    df_extended = pd.concat([df, squared_df, interaction_df], axis=1)

    return df_extended

df_original = pd.DataFrame(data_filtered_clean, columns=columns_of_interest +␣
 ↪['Life expectancy '])

df_extended = create_extended_dataframe(df_original, columns_of_interest)

X = df_extended.drop('Life expectancy ', axis=1).values
y = df_extended['Life expectancy '].values
```

[57]:
```
#view number of columns
print(len(df_extended.columns))
print(len(df_original.columns))
#view columns
print(df_extended.columns)
```

```
36
15
Index(['Adult Mortality', 'infant deaths', 'Alcohol', 'Hepatitis B',
       'Measles ', ' BMI ', 'Polio', 'Total expenditure', 'Diphtheria ', 'GDP',
       'Population', ' thinness  1-19 years',
       'Income composition of resources', 'Schooling', 'Life expectancy ',
       'Adult Mortality^2', 'infant deaths^2', 'Alcohol^2', 'Hepatitis B^2',
       'Measles ^2', ' BMI ^2', 'Polio^2', 'Total expenditure^2',
       'Diphtheria ^2', 'GDP^2', 'Population^2', ' thinness  1-19 years^2',
       'Income composition of resources^2', 'Schooling^2',
       'Adult Mortalityxinfant deaths', 'AlcoholxHepatitis B',
       'Measles x BMI ', 'PolioxTotal expenditure', 'Diphtheria xGDP',
       'Populationx thinness  1-19 years',
       'Income composition of resourcesxSchooling'],
      dtype='object')
```

[58]:
```
#PASO 1 - GRAFICAR

import matplotlib.pyplot as plt
```

```
import numpy as np

predictor_columns = [col for col in df_extended.columns if col != 'Life␣
 ↪expectancy ']
num_variables = len(predictor_columns)

num_cols = 5
num_rows = int(np.ceil(num_variables / num_cols))

plt.figure(figsize=(num_cols * 5, num_rows * 5))

for i, column in enumerate(predictor_columns, 1):
    plt.subplot(num_rows, num_cols, i)
    plt.scatter(df_extended[column], df_extended['Life expectancy '], s=8,␣
 ↪color='purple')
    plt.title(f'{column} vs Life expectancy')
    plt.xlabel(column)
    plt.ylabel('Life expectancy')

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


Adult Mortality vs Life expectancy
infant deaths vs Life expectancy
Alcohol vs Life expectancy
Hepatitis B vs Life expectancy
Measles vs Life expectancy
BMI vs Life expectancy
Polio vs Life expectancy
Total expenditure vs Life expectancy
Diphtheria vs Life expectancy
GDP vs Life expectancy
Population vs Life expectancy
thinness 1-19 years vs Life expectancy
Income composition of resources vs Life expectancy
Schooling vs Life expectancy
Adult Mortality^2 vs Life expectancy
infant deaths^2 vs Life expectancy
Alcohol^2 vs Life expectancy
Hepatitis B^2 vs Life expectancy
Measles ^2 vs Life expectancy
BMI ^2 vs Life expectancy
Polio^2 vs Life expectancy
Total expenditure^2 vs Life expectancy
Diphtheria ^2 vs Life expectancy
GDP^2 vs Life expectancy
Population^2 vs Life expectancy
thinness 1-19 years^2 vs Life expectancy
Income composition of resources^2 vs Life expectancy
Schooling^2 vs Life expectancy
Adult Mortalityxinfant deaths vs Life expectancy
AlcoholxHepatitis B vs Life expectancy
Measles x BMI vs Life expectancy
PolioxTotal expenditure vs Life expectancy
Diphtheria xGDP vs Life expectancy
Populationx thinness 1-19 years vs Life expectancy
Income composition of resourcesxSchooling vs Life expectancy
Life expectancy

[59]:
```python
#PASO 2 -- CALCULAR COEFICIENTES

import numpy as np

def calculate_coefficients(X, y):
    X_transpose = X.T
    beta = np.linalg.inv(X_transpose @ X) @ X_transpose @ y
    return beta

def predict(X, beta):
    return X @ beta

X = df_extended.drop('Life expectancy ', axis=1).values
y = df_extended['Life expectancy '].values

X = np.c_[np.ones(X.shape[0]), X]

beta = calculate_coefficients(X, y)
print("Coeficientes del modelo: ", beta)
```

```
Coeficientes del modelo:  [ 5.82657398e+01  5.70835292e-03 -1.19204879e-02
-1.97330803e-01
  2.44551876e-02  4.73703298e-05  2.63551919e-02 -3.32375308e-02
  1.44897751e-01 -7.60202934e-02  1.05854787e-04  2.74711918e-09
 -6.80606466e-01  1.86799315e+00  5.29184933e-02 -5.26723764e-05
  4.49875990e-07 -1.00086687e-02 -2.89497686e-04  2.58689599e-11
 -2.94734835e-04  2.07396196e-04 -1.12651253e-02  9.20892983e-04
 -2.24941857e-10 -1.47611420e-17  3.69422675e-02  7.30575725e+01
  8.30968234e-02  1.10044593e-05  2.85831407e-04 -2.46715706e-07
  1.38062551e-03 -9.41570485e-07  5.70864324e-10 -3.94681354e+00]
```

[64]:
```python
#PASO 3 -- VALIDACIÓN CRUZADA

from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

n_folds = 5
kf = KFold(n_splits=n_folds, shuffle=True)

mse_cv = []
mae_cv = []
r2_cv = []
for train_index, test_index in kf.split(X):
    # Fase de entrenamiento
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
    # Ajusta el modelo usando tu propia fórmula
    beta_cv = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train

    # Fase de prueba
    y_pred_cv = X_test @ beta_cv

    # Cálculo de las métricas y almacenamiento
    mse_cv.append(mean_squared_error(y_test, y_pred_cv))
    mae_cv.append(mean_absolute_error(y_test, y_pred_cv))
    r2_cv.append(r2_score(y_test, y_pred_cv))

# Imprime las métricas promedio de la validación cruzada
print('MSE promedio:', np.mean(mse_cv))
print('MAE promedio:', np.mean(mae_cv))
print('R^2 promedio:', np.mean(r2_cv))
```

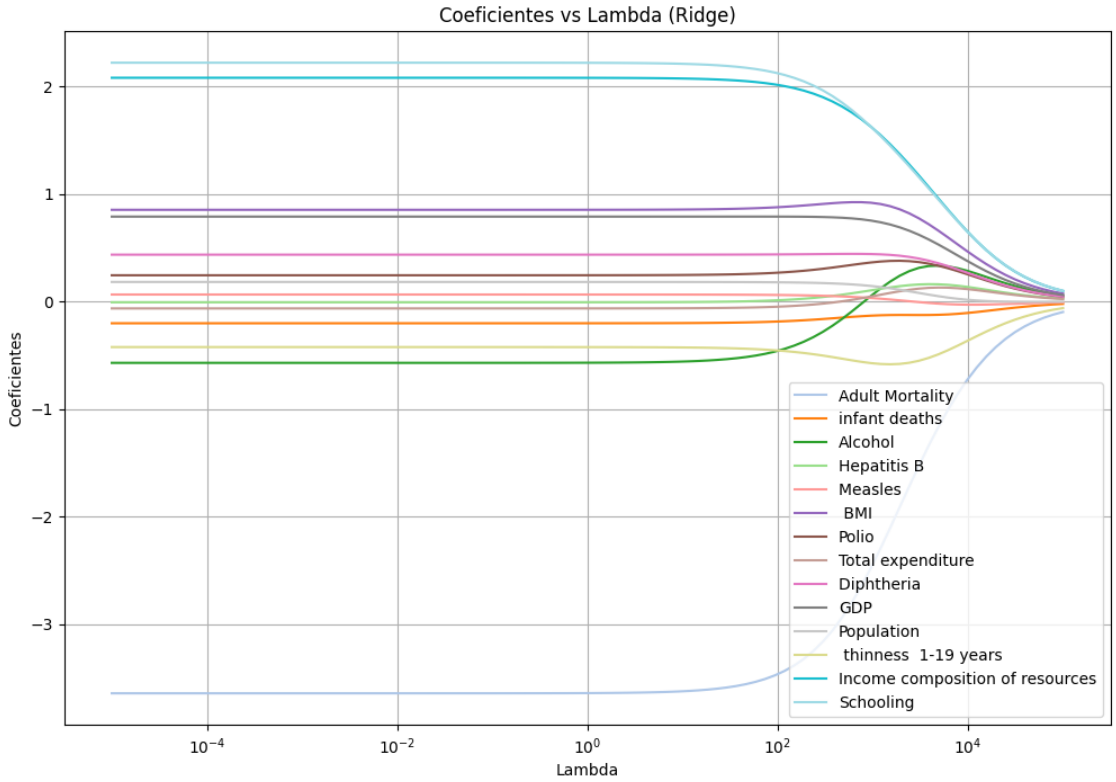
```
MSE promedio: 10.729203916001266
MAE promedio: 2.3996261131827348
R^2 promedio: 0.8598612878673034
```

[164]:
```
#Implementa regresión Ridge con descenso de gradiente, y genera el gráfico de␣
 ↪Ridge para el conjunto de datos original

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.cm import get_cmap

cmap = get_cmap('tab20')

X = data_filtered_clean[variables_independientes].values
y = data_filtered_clean['Life expectancy '].values

X_normalized = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

X = np.concatenate([np.ones((X_normalized.shape[0], 1)), X_normalized], axis=1)

# Parámetros
alpha = 0.0005
n_iterations = 10000
lambda_values = np.logspace(-5, 5, 100)

theta_inicial = np.zeros(X.shape[1])

def cost_function_ridge(X, y, theta, lambda_):
    m = len(y)
    predictions = X.dot(theta)
```

```
    cost = (1/(2*m)) * np.sum((predictions - y) ** 2) + (lambda_/(2*m)) * np.
 ↪sum(theta[1:] ** 2)
    return cost

def gradient_descent_ridge(X, y, theta, alpha, n_iterations, lambda_):
    m = len(y)
    cost_history = np.zeros(n_iterations)

    for i in range(n_iterations):
        predictions = X.dot(theta)
        errors = predictions - y
        gradient = (1/m) * X.T.dot(errors) + (lambda_/m) * np.r_[[0], theta[1:
 ↪]]  # No regularizar el intercepto
        theta -= alpha * gradient

    return theta

theta_values = []

for lambda_ in lambda_values:
    theta = gradient_descent_ridge(X, y, theta_inicial.copy(), alpha,␣
 ↪n_iterations, lambda_)
    theta_values.append(theta)

theta_values = np.array(theta_values)

plt.figure(figsize=(12, 8))

for i in range(1, theta_values.shape[1]):
    color = cmap(i / (theta_values.shape[1] - 1))  # Asignar un color diferente␣
 ↪para cada línea
    plt.plot(lambda_values, theta_values[:, i], color=color,␣
 ↪label=variables_independientes[i-1])

plt.xscale('log')
plt.xlabel('Lambda')
plt.ylabel('Coeficientes')
plt.title('Coeficientes vs Lambda (Ridge)')
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

```
C:\Users\pfeli\AppData\Local\Temp\ipykernel_16284\106165138.py:7:
MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib
3.7 and will be removed two minor releases later. Use
``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap(obj)``
instead.
```

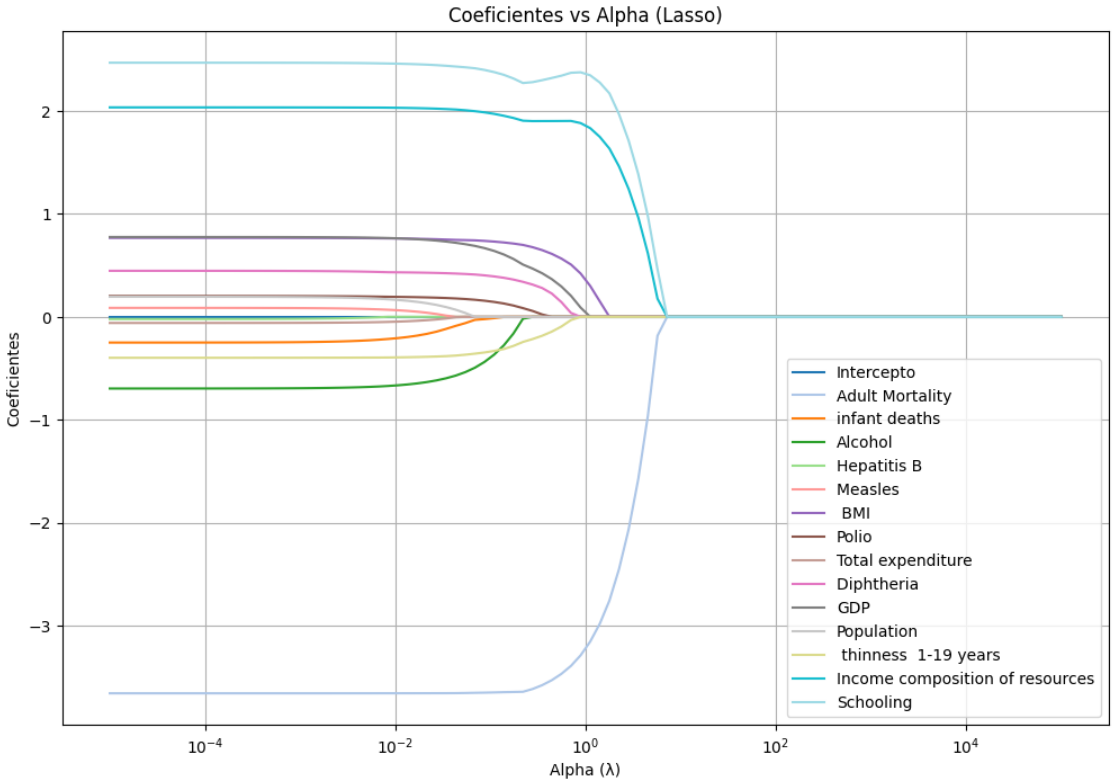
```
cmap = get_cmap('tab20')
```



[165]:
```
#Utiliza una librería para generar el gráfico de Lasso para el conjunto de␣
 ↪datos original

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.cm import get_cmap
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler

cmap = get_cmap('tab20')

X = data_filtered_clean[variables_independientes].values
y = data_filtered_clean['Life expectancy '].values

scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

X_with_intercept = np.concatenate([np.ones((X_normalized.shape[0], 1)),␣
 ↪X_normalized], axis=1)
```

```
# Parámetros
alpha_values = np.logspace(-5, 5, 100)

coefficients = []

for alpha in alpha_values:
    lasso = Lasso(alpha=alpha, max_iter=10000)
    lasso.fit(X_with_intercept, y)
    coefficients.append(lasso.coef_)

coefficients = np.array(coefficients)

variables_independientes_con_intercepto = ['Intercepto'] +␣
 ↪variables_independientes

plt.figure(figsize=(12, 8))
for i in range(coefficients.shape[1]):
    color = cmap(i / (coefficients.shape[1] - 1))
    plt.plot(alpha_values, coefficients[:, i], color=color,␣
 ↪label=variables_independientes_con_intercepto[i])

plt.xscale('log')
plt.xlabel('Alpha ( )')
plt.ylabel('Coeficientes')
plt.title('Coeficientes vs Alpha (Lasso)')
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

```
C:\Users\pfeli\AppData\Local\Temp\ipykernel_16284\1435286719.py:7:
MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib
3.7 and will be removed two minor releases later. Use
``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap(obj)``
instead.
  cmap = get_cmap('tab20')
```



¿Qué variables son más relevantes para el modelo? Schooling, Income composition of resources y Adult Mortality son las variables más relevantes, ya que son las que más se alejan de 0.

Viendo los resultados de regresión, desarrolla una conclusión sobre los siguientes puntos: ¿Consideras que el modelo de regresión lineal es efectivo para modelar los datos del problema? ¿Por qué? ¿Observas una variabilidad importante en los valores de R2, MSE y MAE cuando aplicas validación cruzada? Detalla tu respuesta. ¿Qué modelo es mejor para los datos del problema, el lineal o el cuadrático? ¿Por qué? ¿Qué variables son más relevantes para el modelo según Ridge y Lasso? ¿Encuentras alguna relación interesante entre la variable de respuesta y los predictores?

Considero que el modelo de regresión lineal es aceptable, pero no muy efectivo, ya que muestra un R^2 promedio de 0.75. Cuando aplico validación cruzada, se observa una variabilidad leve en los valores de MAE, mientras que la variabilidad en R^2 y MSE es más notable. La variabilidad en el MSE entre los diferentes pliegues es considerable, con un rango de 17.10 a 21.52. Aunque el MAE también muestra variabilidad, esta es menor en comparación con el MSE, con un rango de 3.01 a 3.20. Finalmente, la variabilidad en el R^2 es significativa, con un rango de 0.73 a 0.79. Esto podría indicar que el modelo tiene dificultades para generalizar, podría estar sobreajustado a ciertos subconjuntos o no estar capturando adecuadamente la variabilidad en los datos.

En mi opinión, el modelo cuadrático en este ejemplo es superior, ya que muestra una reducción significativa en el MSE y MAE, además de una mejora en R^2, lo que indica un mejor ajuste a los datos.

En el análisis realizado, las variables más relevantes para el modelo según Ridge fueron Schooling,

Income Composition of Resources, Adult Mortality, BMI, y Alcohol, mientras que para Lasso, las variables más destacadas fueron Schooling, Income Composition of Resources, Adult Mortality, BMI, y GDP.

Entre las relaciones más interesantes, se observa que Schooling e Income Composition of Resources son consistentemente importantes en ambos modelos de regularización, lo que sugiere que estos factores están fuertemente asociados con la expectativa de vida. Además, el impacto de variables como Adult Mortality y BMI también es notable, lo que subraya la importancia de la mortalidad y la salud general en la predicción de la longevidad.

3 ==========

3.1 EJERCICIO 2

4 ==========

Todas las variables predictoras, menos X5, X9, X13, X17, la variable total_UPDRS como variable a predecir.

[200]:
```
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
import numpy as np
```

[201]:
```
pip install --upgrade scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\pfeli\appdata\local\pack
ages\pythonsoftwarefoundation.python.3.10_qbz5n2kfra8p0\localcache\local-
packages\python310\site-packages (1.5.1)
Requirement already satisfied: numpy>=1.19.5 in c:\users\pfeli\appdata\local\pac
kages\pythonsoftwarefoundation.python.3.10_qbz5n2kfra8p0\localcache\local-
packages\python310\site-packages (from scikit-learn) (1.26.1)
Requirement already satisfied: scipy>=1.6.0 in c:\users\pfeli\appdata\local\pack
ages\pythonsoftwarefoundation.python.3.10_qbz5n2kfra8p0\localcache\local-
packages\python310\site-packages (from scikit-learn) (1.11.3)
Requirement already satisfied: joblib>=1.2.0 in c:\users\pfeli\appdata\local\pac
kages\pythonsoftwarefoundation.python.3.10_qbz5n2kfra8p0\localcache\local-
packages\python310\site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\pfeli\appdata\lo
cal\packages\pythonsoftwarefoundation.python.3.10_qbz5n2kfra8p0\localcache\local
-packages\python310\site-packages (from scikit-learn) (3.2.0)
Note: you may need to restart the kernel to use updated packages.
```

[202]:
```
file_path = file_path = "C:
  ↪\\Users\\pfeli\\Downloads\\ACT1MACHINE\\parkinsons_updrs.csv"
data = pd.read_csv(file_path)
```

[203]:
```
print(data.columns)
```

```
Index(['subject#', 'age', 'sex', 'test_time', 'motor_UPDRS', 'total_UPDRS',
       'Jitter(%)', 'Jitter(Abs)', 'Jitter:RAP', 'Jitter:PPQ5', 'Jitter:DDP',
       'Shimmer', 'Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5',
       'Shimmer:APQ11', 'Shimmer:DDA', 'NHR', 'HNR', 'RPDE', 'DFA', 'PPE'],
      dtype='object')
```

[204]:
```
columns_of_interest = [
    'age', 'sex', 'test_time',
       'Jitter(%)', 'Jitter(Abs)', 'Jitter:PPQ5', 'Jitter:DDP',
       'Shimmer', 'Shimmer:APQ3', 'Shimmer:APQ5',
       'Shimmer:APQ11', 'NHR', 'HNR', 'RPDE', 'PPE'
]

data_filtered = data[['total_UPDRS',] + columns_of_interest]
```

[208]:
```
data_filtered_clean = data_filtered.dropna()

X = data_filtered.drop(columns=['total_UPDRS'])
y = data_filtered['total_UPDRS']
```

[218]:
```
#Evalúa con validación cruzada un modelo de regresión lineal para las variables␣
 ↪asignadas según tu matrícula utilizando alguna librería o framework.

from sklearn.model_selection import KFold
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

kf = KFold(n_splits=5, shuffle=True, random_state=42)

regr = linear_model.LinearRegression()

mse_scores = []
mae_scores = []
r2_scores = []

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    regr.fit(X_train, y_train)

    y_pred = regr.predict(X_test)

    mse_scores.append(mean_squared_error(y_test, y_pred))
    mae_scores.append(mean_absolute_error(y_test, y_pred))
    r2_scores.append(r2_score(y_test, y_pred))
```

```
print("Coeficientes del modelo: ", regr.coef_)
print("MSE medio:", np.mean(mse_scores))
print("MAE medio:", np.mean(mae_scores))
print("R^2 medio:", np.mean(r2_scores))
```

```
Coeficientes del modelo:  [ 3.45152189e-01 -2.92854819e+00  1.69018168e-02
-7.22357902e+01
 -8.77638294e+04 -4.03724735e+02  3.23174447e+02  5.24962276e+01
 -1.04500386e+02 -1.21046734e+02  3.67052188e+01  1.69588891e+01
 -5.27957027e-01  5.08551035e+00  1.07426233e+01]
MSE medio: 98.01454202014695
MAE medio: 8.238240864753518
R^2 medio: 0.14405853909475325
```

[182]:
```
#Encuentra el número óptimo de predictores para el modelo utilizando el método␣
 ↪filter y validación cruzada.
# Una vez que tengas el número óptimo, muestra las características␣
 ↪seleccionadas.

from sklearn.feature_selection import SelectKBest, f_regression

def evaluate_k_best(X, y, k):
    selector = SelectKBest(score_func=f_regression, k=k)
    X_new = selector.fit_transform(X, y)
    scores = cross_val_score(LinearRegression(), X_new, y, cv=5, scoring='r2')
    return np.mean(scores)

max_features = X.shape[1]
best_k = 0
best_score = float('-inf')

for k in range(1, max_features + 1):
    score = evaluate_k_best(X, y, k)
    if score > best_score:
        best_score = score
        best_k = k

print(f'\nNúmero óptimo de predictores: {best_k}')
selector = SelectKBest(score_func=f_regression, k=best_k)
X_new = selector.fit_transform(X, y)
selected_features = X.columns[selector.get_support()]
print("Características seleccionadas:", selected_features.tolist())
```

```
Número óptimo de predictores: 2
Características seleccionadas: ['age', 'HNR']
```

[183]:
```
#Repite el paso anterior pero con selección de características secuencial␣
↪(Wrapper). Reporta los predictores óptimos encontrados por el método.

from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
import numpy as np

model = LinearRegression()

sfs = SequentialFeatureSelector(
    estimator=model,
    n_features_to_select='auto',
    direction='forward',
    scoring='r2',
    cv=5,
    n_jobs=-1
)

sfs.fit(X, y)

selected_features = X.columns[sfs.get_support()]

optimal_number_of_features = len(selected_features)

print(f'Número óptimo de predictores: {optimal_number_of_features}')
print('Características seleccionadas:', selected_features.tolist())

X_selected = sfs.transform(X)
scores = cross_val_score(model, X_selected, y, cv=5, scoring='r2')
```

```
Número óptimo de predictores: 7
Características seleccionadas: ['age', 'test_time', 'Jitter(%)', 'Jitter:PPQ5',
'Jitter:DDP', 'NHR', 'HNR']
```

[185]:
```
#Haz el mismo proceso del paso 2, pero ahora con el método de selección de␣
↪características recursivo.

from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
import numpy as np

model = LinearRegression()

rfe = RFE(estimator=model, n_features_to_select=1)
```

```
rfe.fit(X, y)

optimal_number_of_features = np.sum(rfe.support_)
selected_features = X.columns[rfe.support_]

print(f'Número óptimo de predictores: {optimal_number_of_features}')
print('Características seleccionadas:', selected_features.tolist())

X_selected = rfe.transform(X)
scores = cross_val_score(model, X_selected, y, cv=5, scoring='r2')
```

```
Número óptimo de predictores: 1
Características seleccionadas: ['Jitter(Abs)']
```

K VECINOS

[219]:
```
#k-vecinos más cercanos

from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

X = data_filtered.drop(columns=['total_UPDRS']).values
y = data_filtered['total_UPDRS'].values

knn = KNeighborsRegressor(n_neighbors=5)

kf = KFold(n_splits=5, shuffle=True, random_state=42)

mse_scores = []
mae_scores = []
r2_scores = []

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    knn.fit(X_train, y_train)

    y_pred = knn.predict(X_test)

    mse_scores.append(mean_squared_error(y_test, y_pred))
    mae_scores.append(mean_absolute_error(y_test, y_pred))
    r2_scores.append(r2_score(y_test, y_pred))

print("MSE medio:", np.mean(mse_scores))
print("MAE medio:", np.mean(mae_scores))
```

```
print("R^2 medio:", np.mean(r2_scores))
```

```
MSE medio: 56.599011362152375
MAE medio: 5.502855996595744
R^2 medio: 0.5057975589691985
```

[225]:
```
# Filter y validación cruzada

from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsRegressor
import numpy as np

X = data_filtered.drop(columns=['total_UPDRS']).values
y = data_filtered['total_UPDRS'].values

def evaluate_k_best_knn(X, y, k):
    selector = SelectKBest(score_func=f_regression, k=k)
    X_new = selector.fit_transform(X, y)
    scores = cross_val_score(KNeighborsRegressor(), X_new, y, cv=5,␣
 ↪scoring='r2')
    return np.mean(scores)

max_features = X.shape[1]
best_k_knn = 0
best_score_knn = float('-inf')

for k in range(1, max_features + 1):
    score = evaluate_k_best_knn(X, y, k)
    if score > best_score_knn:
        best_score_knn = score
        best_k_knn = k
print(f'\nNúmero óptimo de predictores (KNN): {best_k_knn}')

selector_knn = SelectKBest(score_func=f_regression, k=best_k_knn)
X_new_knn = selector_knn.fit_transform(X, y)

selected_features_knn = data_filtered.drop(columns=['total_UPDRS']).
 ↪columns[selector_knn.get_support()]
print("Características seleccionadas (KNN):", selected_features_knn.tolist())
```

```
Número óptimo de predictores (KNN): 6
Características seleccionadas (KNN): ['age', 'sex', 'Shimmer:APQ11', 'HNR',
'RPDE', 'PPE']
```

[230]:
```
#Wrapper y validación cruzada

from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score
import numpy as np

X = data_filtered.drop(columns=['total_UPDRS'])
y = data_filtered['total_UPDRS']

model = KNeighborsRegressor(n_neighbors=5)

sfs = SequentialFeatureSelector(
    estimator=model,
    n_features_to_select='auto',
    direction='forward',
    scoring='r2',
    cv=5,
    n_jobs=-1
)

sfs.fit(X, y)

selected_features = X.columns[sfs.get_support()]

optimal_number_of_features = len(selected_features)

print(f'Número óptimo de predictores: {optimal_number_of_features}')
print('Características seleccionadas:', selected_features.tolist())

X_selected = sfs.transform(X)

scores = cross_val_score(model, X_selected, y, cv=5, scoring='r2')
```

```
Número óptimo de predictores: 7
Características seleccionadas: ['age', 'Jitter(Abs)', 'Shimmer', 'Shimmer:APQ3',
'Shimmer:APQ5', 'NHR', 'RPDE']
```

[233]:
```
#Recursive y validación cruzada

from sklearn.feature_selection import RFE
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
import numpy as np

model_lr = LinearRegression()
```

```
rfe = RFE(estimator=model_lr, n_features_to_select=1)

rfe.fit(X, y)

optimal_number_of_features = np.sum(rfe.support_)
selected_features = X.columns[rfe.support_]

print(f'Número óptimo de predictores: {optimal_number_of_features}')
print('Características seleccionadas:', selected_features.tolist())

X_selected = X[selected_features]

model_knn = KNeighborsRegressor()

scores = cross_val_score(model_knn, X_selected, y, cv=5, scoring='r2')
```

```
Número óptimo de predictores: 1
Características seleccionadas: ['Jitter(Abs)']
```

DECISION TREE REGRESSOR

[235]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

X = data_filtered.drop(columns=['total_UPDRS']).values
y = data_filtered['total_UPDRS'].values

dt_regressor = DecisionTreeRegressor()

kf = KFold(n_splits=5, shuffle=True, random_state=42)

mse_scores = []
mae_scores = []
r2_scores = []

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    dt_regressor.fit(X_train, y_train)

    y_pred = dt_regressor.predict(X_test)

    mse_scores.append(mean_squared_error(y_test, y_pred))
    mae_scores.append(mean_absolute_error(y_test, y_pred))
```

```
    r2_scores.append(r2_score(y_test, y_pred))

print("DecisionTreeRegressor - MSE medio:", np.mean(mse_scores))
print("DecisionTreeRegressor - MAE medio:", np.mean(mae_scores))
print("DecisionTreeRegressor - R^2 medio:", np.mean(r2_scores))
```

```
DecisionTreeRegressor - MSE medio: 4.979562885737872
DecisionTreeRegressor - MAE medio: 0.5610437617021278
DecisionTreeRegressor - R^2 medio: 0.9563755416627708
```

[237]:
```
# Filter y validación cruzada

from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.model_selection import cross_val_score
import numpy as np

def evaluate_k_best_dt(X, y, k):
    selector = SelectKBest(score_func=f_regression, k=k)
    X_new = selector.fit_transform(X, y)
    scores = cross_val_score(DecisionTreeRegressor(), X_new, y, cv=5,␣
 ↪scoring='r2')
    return np.mean(scores)

X = data_filtered.drop(columns=['total_UPDRS']).values
y = data_filtered['total_UPDRS'].values

max_features = X.shape[1]
best_k = 0
best_score = float('-inf')

for k in range(1, max_features + 1):
    score = evaluate_k_best_dt(X, y, k)
    if score > best_score:
        best_score = score
        best_k = k

print(f'DecisionTreeRegressor - Número óptimo de predictores: {best_k}')

selector = SelectKBest(score_func=f_regression, k=best_k)
X_new = selector.fit_transform(X, y)
selected_features = data_filtered.drop(columns=['total_UPDRS']).
 ↪columns[selector.get_support()]
print(f'DecisionTreeRegressor - Características seleccionadas:',␣
 ↪selected_features.tolist())
```

```
DecisionTreeRegressor - Número óptimo de predictores: 1
DecisionTreeRegressor - Características seleccionadas: ['age']
```

[241]:
```
# Wrapper y validación cruzada
from sklearn.feature_selection import SequentialFeatureSelector

X = data_filtered.drop(columns=['total_UPDRS'])
y = data_filtered['total_UPDRS']

sfs = SequentialFeatureSelector(
    estimator=DecisionTreeRegressor(),
    n_features_to_select='auto',
    direction='forward',
    scoring='r2',
    cv=5,
    n_jobs=-1
)

sfs.fit(X, y)
selected_features = X.columns[sfs.get_support()]
optimal_number_of_features = len(selected_features)

print(f'DecisionTreeRegressor - Número óptimo de predictores:␣
 ↪{optimal_number_of_features}')
print(f'DecisionTreeRegressor - Características seleccionadas:',␣
 ↪selected_features.tolist())

X_selected = sfs.transform(X)
scores = cross_val_score(DecisionTreeRegressor(), X_selected, y, cv=5,␣
 ↪scoring='r2')
```

```
DecisionTreeRegressor - Número óptimo de predictores: 7
DecisionTreeRegressor - Características seleccionadas: ['age', 'sex',
'test_time', 'Jitter(%)', 'Jitter(Abs)', 'Shimmer:APQ5', 'RPDE']
```

[243]:
```
# Recursive y validación cruzada

from sklearn.feature_selection import RFE

X = data_filtered.drop(columns=['total_UPDRS'])
y = data_filtered['total_UPDRS']

model_lr = DecisionTreeRegressor()
rfe = RFE(estimator=model_lr, n_features_to_select=1)

rfe.fit(X, y)
optimal_number_of_features = np.sum(rfe.support_)
selected_features = X.columns[rfe.support_]
```

```
print(f'DecisionTreeRegressor - Número óptimo de predictores:␣
 ↪{optimal_number_of_features}')
print(f'DecisionTreeRegressor - Características seleccionadas:',␣
 ↪selected_features.tolist())

X_selected = X[selected_features]
scores = cross_val_score(DecisionTreeRegressor(), X_selected, y, cv=5,␣
 ↪scoring='r2')
```

```
DecisionTreeRegressor - Número óptimo de predictores: 1
DecisionTreeRegressor - Características seleccionadas: ['age']
```

GRADIENT BOOSTING REGRESSOR

[245]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

X = data_filtered.drop(columns=['total_UPDRS']).values
y = data_filtered['total_UPDRS'].values

gbr = GradientBoostingRegressor()

kf = KFold(n_splits=5, shuffle=True, random_state=42)

mse_scores = []
mae_scores = []
r2_scores = []

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    gbr.fit(X_train, y_train)

    y_pred = gbr.predict(X_test)

    mse_scores.append(mean_squared_error(y_test, y_pred))
    mae_scores.append(mean_absolute_error(y_test, y_pred))
    r2_scores.append(r2_score(y_test, y_pred))

print("GradientBoostingRegressor - MSE medio:", np.mean(mse_scores))
print("GradientBoostingRegressor - MAE medio:", np.mean(mae_scores))
print("GradientBoostingRegressor - R^2 medio:", np.mean(r2_scores))
```

```
GradientBoostingRegressor - MSE medio: 24.928502176070637
GradientBoostingRegressor - MAE medio: 3.9552129022430718
```

```
GradientBoostingRegressor - R^2 medio: 0.7818041766028119
```

[247]:
```
# Filter y validación cruzada

from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.model_selection import cross_val_score
import numpy as np

def evaluate_k_best_gbr(X, y, k):
    selector = SelectKBest(score_func=f_regression, k=k)
    X_new = selector.fit_transform(X, y)
    scores = cross_val_score(GradientBoostingRegressor(), X_new, y, cv=5,␣
 ↪scoring='r2')
    return np.mean(scores)

X = data_filtered.drop(columns=['total_UPDRS']).values
y = data_filtered['total_UPDRS'].values

max_features = X.shape[1]
best_k = 0
best_score = float('-inf')

for k in range(1, max_features + 1):
    score = evaluate_k_best_gbr(X, y, k)
    if score > best_score:
        best_score = score
        best_k = k

print(f'GradientBoostingRegressor - Número óptimo de predictores: {best_k}')

selector = SelectKBest(score_func=f_regression, k=best_k)
X_new = selector.fit_transform(X, y)
selected_features = data_filtered.drop(columns=['total_UPDRS']).
 ↪columns[selector.get_support()]
print(f'GradientBoostingRegressor - Características seleccionadas:',␣
 ↪selected_features.tolist())
```

```
GradientBoostingRegressor - Número óptimo de predictores: 3
GradientBoostingRegressor - Características seleccionadas: ['age', 'HNR',
'RPDE']
```

[249]:
```
#Wrapper y validación cruzada
from sklearn.feature_selection import SequentialFeatureSelector

X = data_filtered.drop(columns=['total_UPDRS'])
y = data_filtered['total_UPDRS']
```

```
sfs = SequentialFeatureSelector(
    estimator=GradientBoostingRegressor(),
    n_features_to_select='auto',
    direction='forward',
    scoring='r2',
    cv=5,
    n_jobs=-1
)

sfs.fit(X, y)
selected_features = X.columns[sfs.get_support()]
optimal_number_of_features = len(selected_features)

print(f'GradientBoostingRegressor - Número óptimo de predictores:␣
 ↪{optimal_number_of_features}')
print(f'GradientBoostingRegressor - Características seleccionadas:',␣
 ↪selected_features.tolist())

X_selected = sfs.transform(X)
scores = cross_val_score(GradientBoostingRegressor(), X_selected, y, cv=5,␣
 ↪scoring='r2')
```

```
GradientBoostingRegressor - Número óptimo de predictores: 7
GradientBoostingRegressor - Características seleccionadas: ['Jitter(%)',
'Jitter:PPQ5', 'Jitter:DDP', 'Shimmer', 'Shimmer:APQ3', 'Shimmer:APQ11', 'PPE']
```

[251]:
```
#Recursive y validación cruzada
from sklearn.feature_selection import RFE

X = data_filtered.drop(columns=['total_UPDRS'])
y = data_filtered['total_UPDRS']

model_lr = GradientBoostingRegressor()
rfe = RFE(estimator=model_lr, n_features_to_select=1)

rfe.fit(X, y)
optimal_number_of_features = np.sum(rfe.support_)
selected_features = X.columns[rfe.support_]

print(f'GradientBoostingRegressor - Número óptimo de predictores:␣
 ↪{optimal_number_of_features}')
print(f'GradientBoostingRegressor - Características seleccionadas:',␣
 ↪selected_features.tolist())

X_selected = X[selected_features]
scores = cross_val_score(GradientBoostingRegressor(), X_selected, y, cv=5,␣
 ↪scoring='r2')
```

```
GradientBoostingRegressor - Número óptimo de predictores: 1
GradientBoostingRegressor - Características seleccionadas: ['age']
```

ADABOOST REGRESSOR

[252]:
```python
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

X = data_filtered.drop(columns=['total_UPDRS']).values
y = data_filtered['total_UPDRS'].values

ada_regressor = AdaBoostRegressor()

kf = KFold(n_splits=5, shuffle=True, random_state=42)

mse_scores = []
mae_scores = []
r2_scores = []

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    ada_regressor.fit(X_train, y_train)

    y_pred = ada_regressor.predict(X_test)

    mse_scores.append(mean_squared_error(y_test, y_pred))
    mae_scores.append(mean_absolute_error(y_test, y_pred))
    r2_scores.append(r2_score(y_test, y_pred))

print("AdaBoostRegressor - MSE medio:", np.mean(mse_scores))
print("AdaBoostRegressor - MAE medio:", np.mean(mae_scores))
print("AdaBoostRegressor - R^2 medio:", np.mean(r2_scores))
```

```
AdaBoostRegressor - MSE medio: 70.77736410504073
AdaBoostRegressor - MAE medio: 7.350269384743511
AdaBoostRegressor - R^2 medio: 0.3813266021230778
```

[253]:
```python
# Filter y validación cruzada

from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.model_selection import cross_val_score
import numpy as np

def evaluate_k_best_ada(X, y, k):
```

```
    selector = SelectKBest(score_func=f_regression, k=k)
    X_new = selector.fit_transform(X, y)
    scores = cross_val_score(AdaBoostRegressor(), X_new, y, cv=5, scoring='r2')
    return np.mean(scores)

X = data_filtered.drop(columns=['total_UPDRS']).values
y = data_filtered['total_UPDRS'].values

max_features = X.shape[1]
best_k = 0
best_score = float('-inf')

for k in range(1, max_features + 1):
    score = evaluate_k_best_ada(X, y, k)
    if score > best_score:
        best_score = score
        best_k = k

print(f'AdaBoostRegressor - Número óptimo de predictores: {best_k}')

selector = SelectKBest(score_func=f_regression, k=best_k)
X_new = selector.fit_transform(X, y)
selected_features = data_filtered.drop(columns=['total_UPDRS']).
 ↪columns[selector.get_support()]
print(f'AdaBoostRegressor - Características seleccionadas:', selected_features.
 ↪tolist())
```

```
AdaBoostRegressor - Número óptimo de predictores: 1
AdaBoostRegressor - Características seleccionadas: ['age']
```

[254]:
```
#Wrapper y validación cruzada

from sklearn.feature_selection import SequentialFeatureSelector

X = data_filtered.drop(columns=['total_UPDRS'])
y = data_filtered['total_UPDRS']

sfs = SequentialFeatureSelector(
    estimator=AdaBoostRegressor(),
    n_features_to_select='auto',
    direction='forward',
    scoring='r2',
    cv=5,
    n_jobs=-1
)

sfs.fit(X, y)
```

```
selected_features = X.columns[sfs.get_support()]
optimal_number_of_features = len(selected_features)

print(f'AdaBoostRegressor - Número óptimo de predictores:␣
 ↪{optimal_number_of_features}')
print(f'AdaBoostRegressor - Características seleccionadas:', selected_features.
 ↪tolist())

X_selected = sfs.transform(X)
scores = cross_val_score(AdaBoostRegressor(), X_selected, y, cv=5, scoring='r2')
```

```
AdaBoostRegressor - Número óptimo de predictores: 7
AdaBoostRegressor - Características seleccionadas: ['age', 'Jitter(%)',
'Shimmer', 'Shimmer:APQ3', 'Shimmer:APQ5', 'Shimmer:APQ11', 'NHR']
```

[256]:
```
#Recursive y validación cruzada

from sklearn.feature_selection import RFE

X = data_filtered.drop(columns=['total_UPDRS'])
y = data_filtered['total_UPDRS']

model_lr = AdaBoostRegressor()
rfe = RFE(estimator=model_lr, n_features_to_select=1)

rfe.fit(X, y)
optimal_number_of_features = np.sum(rfe.support_)
selected_features = X.columns[rfe.support_]

print(f'AdaBoostRegressor - Número óptimo de predictores:␣
 ↪{optimal_number_of_features}')
print(f'AdaBoostRegressor - Características seleccionadas:', selected_features.
 ↪tolist())

X_selected = X[selected_features]
scores = cross_val_score(AdaBoostRegressor(), X_selected, y, cv=5, scoring='r2')
```

```
AdaBoostRegressor - Número óptimo de predictores: 1
AdaBoostRegressor - Características seleccionadas: ['age']
```

RANDOM FOREST

[257]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

X = data_filtered.drop(columns=['total_UPDRS']).values
```

```
y = data_filtered['total_UPDRS'].values

rf_regressor = RandomForestRegressor()

kf = KFold(n_splits=5, shuffle=True, random_state=42)

mse_scores = []
mae_scores = []
r2_scores = []

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    rf_regressor.fit(X_train, y_train)

    y_pred = rf_regressor.predict(X_test)

    mse_scores.append(mean_squared_error(y_test, y_pred))
    mae_scores.append(mean_absolute_error(y_test, y_pred))
    r2_scores.append(r2_score(y_test, y_pred))

print("RandomForestRegressor - MSE medio:", np.mean(mse_scores))
print("RandomForestRegressor - MAE medio:", np.mean(mae_scores))
print("RandomForestRegressor - R^2 medio:", np.mean(r2_scores))
```

```
RandomForestRegressor - MSE medio: 3.095935194305712
RandomForestRegressor - MAE medio: 0.7196687194893624
RandomForestRegressor - R^2 medio: 0.9729955130881839
```

[258]:
```
# Filter y validación cruzada

from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.model_selection import cross_val_score
import numpy as np

def evaluate_k_best_rf(X, y, k):
    selector = SelectKBest(score_func=f_regression, k=k)
    X_new = selector.fit_transform(X, y)
    scores = cross_val_score(RandomForestRegressor(), X_new, y, cv=5,␣
 ↪scoring='r2')
    return np.mean(scores)

X = data_filtered.drop(columns=['total_UPDRS']).values
y = data_filtered['total_UPDRS'].values

max_features = X.shape[1]
```

```
best_k = 0
best_score = float('-inf')

for k in range(1, max_features + 1):
    score = evaluate_k_best_rf(X, y, k)
    if score > best_score:
        best_score = score
        best_k = k

print(f'RandomForestRegressor - Número óptimo de predictores: {best_k}')

selector = SelectKBest(score_func=f_regression, k=best_k)
X_new = selector.fit_transform(X, y)
selected_features = data_filtered.drop(columns=['total_UPDRS']).
 ↪columns[selector.get_support()]
print(f'RandomForestRegressor - Características seleccionadas:',␣
 ↪selected_features.tolist())
```

```
RandomForestRegressor - Número óptimo de predictores: 1
RandomForestRegressor - Características seleccionadas: ['age']
```

[259]:
```
#Wrapper y validación cruzada

from sklearn.feature_selection import SequentialFeatureSelector

X = data_filtered.drop(columns=['total_UPDRS'])
y = data_filtered['total_UPDRS']

sfs = SequentialFeatureSelector(
    estimator=RandomForestRegressor(),
    n_features_to_select='auto',
    direction='forward',
    scoring='r2',
    cv=5,
    n_jobs=-1
)

sfs.fit(X, y)
selected_features = X.columns[sfs.get_support()]
optimal_number_of_features = len(selected_features)

print(f'RandomForestRegressor - Número óptimo de predictores:␣
 ↪{optimal_number_of_features}')
print(f'RandomForestRegressor - Características seleccionadas:',␣
 ↪selected_features.tolist())

X_selected = sfs.transform(X)
```

```
scores = cross_val_score(RandomForestRegressor(), X_selected, y, cv=5,␣
 ↪scoring='r2')
```

```
RandomForestRegressor - Número óptimo de predictores: 7
RandomForestRegressor - Características seleccionadas: ['sex', 'Jitter(Abs)',
'Shimmer', 'Shimmer:APQ3', 'Shimmer:APQ11', 'NHR', 'PPE']
```

[260]:
```
#Recursive y validación cruzada

from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

X = data_filtered.drop(columns=['total_UPDRS'])
y = data_filtered['total_UPDRS']

model_lr = RandomForestRegressor()
rfe = RFE(estimator=model_lr, n_features_to_select=1)

rfe.fit(X, y)
optimal_number_of_features = np.sum(rfe.support_)
selected_features = X.columns[rfe.support_]

print(f'RandomForestRegressor - Número óptimo de predictores:␣
 ↪{optimal_number_of_features}')
print(f'RandomForestRegressor - Características seleccionadas:',␣
 ↪selected_features.tolist())

X_selected = X[selected_features]
scores = cross_val_score(RandomForestRegressor(), X_selected, y, cv=5,␣
 ↪scoring='r2')
```

```
RandomForestRegressor - Número óptimo de predictores: 1
RandomForestRegressor - Características seleccionadas: ['age']
```

Consideras que el modelo de regresión lineal es adecuado para los datos. ¿Por qué? ¿Qué método de selección de características consideras que funciona bien con los datos? ¿Por qué? Del proceso de selección de características, ¿puedes identificar algunas que sean sobresalientes? ¿Qué información relevantes observas de dichas características? ¿Los modelos de regresión no lineal funcionaron mejor que el lineal? ¿Por qué? ¿Se puede concluir algo interesante sobre los resultados de modelar estos datos con regresión? Argumenta tu respuesta.

Dados los resultados obtenidos, considero que el modelo de regresión lineal no es el más adecuado para este conjunto de datos. Los bajos valores de R-cuadrada, junto con los elevados valores de MSE y MAE, indican que el modelo lineal no logra capturar de manera satisfactoria las relaciones entre las variables.

El método wrapper, utilizando SequentialFeatureSelector, parece ser el más efectivo. Este método

selecciona un mayor número de características, lo que es beneficioso para los modelos no lineales que pueden aprovechar la información de múltiples características para mejorar su rendimiento.

Las características más relevantes y seleccionadas frecuentemente son age y Jitter(Abs). Age es importante porque el riesgo de desarrollar Parkinson aumenta con la edad, mientras que Jitter(Abs) es un síntoma típico de la enfermedad, lo que refuerza su relevancia en la predicción.

En promedio, los modelos de regresión no lineal funcionaron mucho mejor que el lineal. Los que destacaron fueron Decision Tree Regressor con un R-cuadrada de 0.96 y Random Forest Regressor con un R-cuadrada de 0.97. Esto se debe a que los modelos de regresión no lineal pueden capturar relaciones más complejas entre las variables, lo que les permite ajustarse mejor a los patrones no lineales en los datos.

En conclusión, la enfermedad de Parkinson es bastante compleja y necesita modelos más sofisticados para entenderla bien. La edad y el temblor en la voz son dos datos muy importantes para predecirla, pero hay otros factores que también influyen.