```
import csv
import numpy.linalg as ln
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold, ShuffleSplit, LeavePOut, LeaveOneOut
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline


# Archivo CSV
csv_file = 'life_expectancy_data.csv'


# Columnas a excluir
exclude_columns = ['Country','Status','Year','percentage expenditure', 'under-five deaths
exclude_independent = ['Country','Status','Year','percentage expenditure', 'under-five de


# Leer el archivo CSV
with open(csv_file, mode='r') as file:
    csv_reader = csv.DictReader(file)

    # Filtrar columnas independientes y dependiente
    columns = [column for column in csv_reader.fieldnames if column not in exclude_columr
    independent = [column for column in csv_reader.fieldnames if column not in exclude_ir
    dependent = 'Life expectancy '

    # Inicializar listas para los datos
    data = {column: [] for column in independent}
    life_expectancy = []

    # Leer los datos del archivo CSV
    for row in csv_reader:
        for column in independent:
            try:
                value = float(row[column]) if row[column] else np.nan
            except ValueError:  # Handle conversion errors
                value = np.nan
            data[column].append(value)

        try:
            value = float(row[dependent]) if row[dependent] else np.nan
        except ValueError:  # Handle conversion errors
            value = np.nan
        life_expectancy.append(value)
```

```
# Graficar cada variable independiente contra la variable dependiente
for column in independent:
    plt.figure(figsize=(10, 6))
    plt.scatter(data[column], life_expectancy, alpha=0.5)
    plt.title(f'{column} vs {dependent}')
    plt.xlabel(column)
    plt.ylabel(dependent)
    plt.grid(True)
    plt.show()
```

Show hidden output

```
# Función Fit
def fit_model(X, y):
    return np.linalg.inv(X.T @ X) @ X.T @ y

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

def preprocess_data(data, life_expectancy, independent):
    X = np.column_stack([np.array(data[column]) for column in independent])
    y = np.array(life_expectancy)

    # Imputar con el promedio
    X = np.where(np.isnan(X), np.nanmean(X, axis=0), X)
    y = np.where(np.isnan(y), np.nanmean(y), y)

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

    return X, y

# Evaluación
def evaluate_model(X, y):
    beta = fit_model(X, y)
    y_pred = predict(X, beta)
    residuals = y - y_pred

    plt.scatter(y, residuals)
    plt.axline((0, 0), slope=0, color='red')
    plt.xlabel('Life Expectancy')
    plt.ylabel('Residuals')
    plt.title('Residuals of the Model')
    plt.show()

    mse = mean_squared_error(y, y_pred)
    mae = mean_absolute_error(y, y_pred)
    r2 = r2_score(y, y_pred)

    print('\nMSE: ', mse, "MAE: ", mae, "R^2: ", r2)
```

```
    print('\nMSE:', mse, 'MAE:', mae, 'R2:', r2)


    return beta

# Cross-validation
def cross_validate(X, y, 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):
        X_train = X[train_index]
        y_train = y[train_index]
        X_test = X[test_index]
        y_test = y[test_index]

        beta_cv = fit_model(X_train, y_train)
        y_pred_cv = predict(X_test, beta_cv)

        mse_i = mean_squared_error(y_test, y_pred_cv)
        mae_i = mean_absolute_error(y_test, y_pred_cv)
        r2_i = r2_score(y_test, y_pred_cv)

        mse_cv.append(mse_i)
        mae_cv.append(mae_i)
        r2_cv.append(r2_i)

    print('\nCross Validated\nMSE:', np.average(mse_cv), '  MAE:', np.average(mae_cv),'


def monte_carlo_cv(X, y, n_iterations=1000, test_size=0.2):
    ss = ShuffleSplit(n_splits=n_iterations, test_size=test_size, random_state=0)
    mse_mc = []
    mae_mc = []
    r2_mc = []

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

        beta_mc = fit_model(X_train, y_train)
        y_pred_mc = predict(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))

    # Hstogramas
    transparency = 1
```

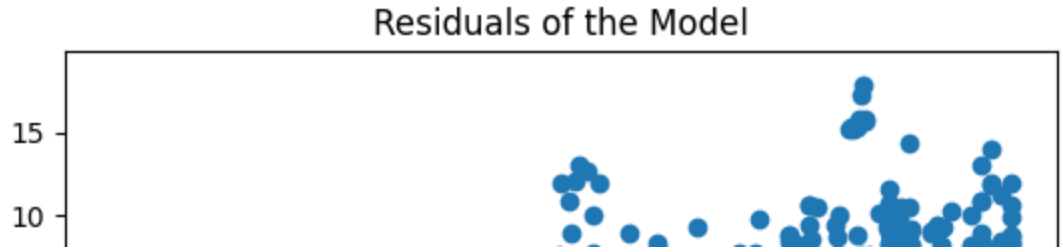
```
    plt.figure(figsize=(15, 5))
    plt.subplot(1, 3, 1)
    plt.hist(mse_mc, bins=30, color='pink', alpha=transparency)
    plt.title('Histogram of MSE')
    plt.xlabel('MSE')
    plt.ylabel('Frequency')

    plt.subplot(1, 3, 2)
    plt.hist(mae_mc, bins=30, color='lavender', alpha=transparency)
    plt.title('Histogram of MAE')
    plt.xlabel('MAE')
    plt.ylabel('Frequency')

    plt.subplot(1, 3, 3)
    plt.hist(r2_mc, bins=30, color='purple', alpha=transparency)
    plt.title('Histogram of R^2')
    plt.xlabel('R^2')
    plt.ylabel('Frequency')

    plt.tight_layout()
    plt.show()

    print('\nMonte Carlo CV\nMSE Mean:', np.mean(mse_mc), '  MAE Mean:', np.mean(mae_mc))

def lpocv(X, y, P=2):
    lpo = LeavePOut(p=P)
    mse_lpo = []
    mae_lpo = []
    r2_lpo = []

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

        beta_lpo = fit_model(X_train, y_train)
        y_pred_lpo = predict(X_test, beta_lpo)

        mse_lpo.append(mean_squared_error(y_test, y_pred_lpo))
        mae_lpo.append(mean_absolute_error(y_test, y_pred_lpo))
        r2_lpo.append(r2_score(y_test, y_pred_lpo))

    plt.figure(figsize=(15, 5))
    plt.subplot(1, 3, 1)
    plt.hist(mse_lpo, bins=30, color='blue', alpha=0.7)
    plt.title('Histogram of MSE (LPOCV)')
    plt.xlabel('MSE')
    plt.ylabel('Frequency')

    plt.subplot(1, 3, 2)
    plt.hist(mae_lpo, bins=30, color='green', alpha=0.7)
```

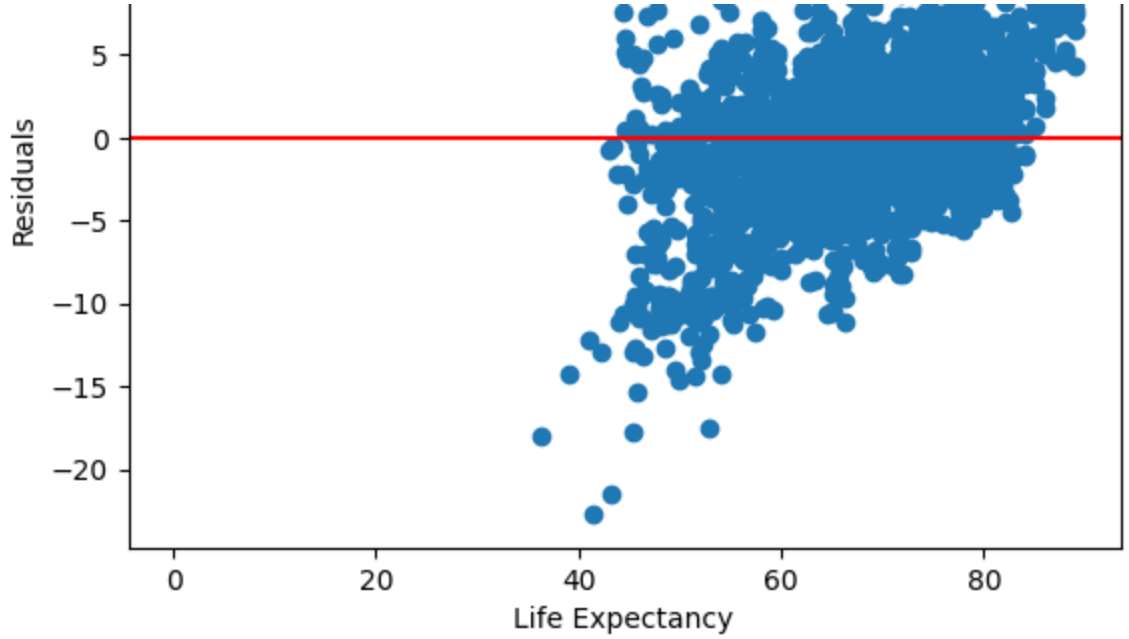
```
    plt.title('Histogram of MAE (LPOCV)')
    plt.xlabel('MAE')
    plt.ylabel('Frequency')

    plt.subplot(1, 3, 3)
    plt.hist(r2_lpo, bins=30, color='red', alpha=0.7)
    plt.title('Histogram of R^2 (LPOCV)')
    plt.xlabel('R^2')
    plt.ylabel('Frequency')

    plt.tight_layout()
    plt.show()

    print('\nLeave-P-Out CV (P=2)\nMSE Mean:', np.mean(mse_lpo), '  MAE Mean:', np.mean(r


def loocv(X, y):
    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]

        beta_loo = fit_model(X_train, y_train)
        y_pred_loo = predict(X_test, beta_loo)

        mse_loo.append(mean_squared_error(y_test, y_pred_loo))
        mae_loo.append(mean_absolute_error(y_test, y_pred_loo))

    plt.figure(figsize=(15, 5))
    plt.subplot(1, 3, 1)
    plt.hist(mse_loo, bins=30, color='blue', alpha=0.7)
    plt.title('Histogram of MSE (LOOCV)')
    plt.xlabel('MSE')
    plt.ylabel('Frequency')

    plt.subplot(1, 3, 2)
    plt.hist(mae_loo, bins=30, color='green', alpha=0.7)
    plt.title('Histogram of MAE (LOOCV)')
    plt.xlabel('MAE')
    plt.ylabel('Frequency')


    plt.tight_layout()
    plt.show()

    print('\nLeave-One-Out CV\nMSE Mean:', np.mean(mse_loo), '  MAE Mean:', np.mean(mae_
```

```
def preprocess_data_scaled(data, life_expectancy, independent):
    X = np.column_stack([np.array(data[column]) for column in independent])
    y = np.array(life_expectancy)

    # Imputar con el promedio
    X = np.where(np.isnan(X), np.nanmean(X, axis=0), X)
    y = np.where(np.isnan(y), np.nanmean(y), y)

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

    # Escalado
    scaler = StandardScaler()
    X[:, 1:] = scaler.fit_transform(X[:, 1:])

    return X, y

def ridge_regression(X, y, alpha=1.0, num_iterations=1000, learning_rate=0.01):
    X = np.array(X)
    y = np.array(y)
    m, n = X.shape
    beta = np.zeros(n)

    for i in range(num_iterations):
        predictions = X @ beta
        errors = predictions - y
        gradient = (2/m) * X.T @ errors
        beta -= learning_rate * gradient + alpha * beta  # Regularización L2

    return beta

def plot_ridge_curve(X, y):
    alphas = np.logspace(-4, 4, 100)
    ridge = RidgeCV(alphas=alphas, store_cv_values=True)
    ridge.fit(X, y)

    plt.figure(figsize=(12, 6))

    #  MSE
    plt.subplot(1, 2, 1)
    plt.plot(alphas, ridge.cv_values_.mean(axis=0), label='MSE')
    plt.axvline(x=ridge.alpha_, color='r', linestyle='--', label='Optimal alpha')
    plt.xscale('log')
    plt.xlabel('Alpha')
    plt.ylabel('MSE')
    plt.title('Ridge Regression MSE')
    plt.legend()
    plt.grid(True)

    # Coeficientes
    plt.subplot(1, 2, 2)
    ridge = Ridge()
```

```
    n_independent = X.shape[1]
    n_alphas = len(alphas)

    coef_matrix = np.zeros((n_alphas, n_independent))

    for i in range(n_alphas):
        alpha = alphas[i]
        ridge = Ridge(alpha = alpha)
        ridge.fit(X, y)
        coef_matrix[i, :] = ridge.coef_

    for i in range(n_independent-1):
            plt.plot(alphas, coef_matrix[:, i], label=independent[i])

    plt.xscale('log')
    plt.xlabel('Alpha')
    plt.ylabel('Coeficientes')
    plt.title('Grafica Ridge')
    plt.legend(loc='best')
    plt.grid(True)

    plt.tight_layout()
    plt.show()


def lasso(X, y, independent):
    alphas = np.logspace(-4, 4, 100)

    n_independent = X.shape[1]
    n_alphas = len(alphas)

    coef_matrix = np.zeros((n_alphas, n_independent))

    for i in range(n_alphas):
        alpha = alphas[i]
        lasso = Lasso(alpha=alpha)
        lasso.fit(X, y)
        coef_matrix[i, :] = lasso.coef_

    plt.figure(figsize=(10, 6))
    for i in range(n_independent-1):
            plt.plot(alphas, coef_matrix[:, i], label=independent[i])

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

```
    plt.show()


def generate_extra(X, independent):
    num_features = X.shape[1] - 1
    squared_features = []
    mult_features = []

    # Generar cuadrados
    for i in range(1, num_features + 1):
        squared_features.append(X[:, i]**2)

    # Generar multiplicaciones entre variables
    for i in range(1, num_features + 1):
        for j in range(i + 1, num_features + 1):
            mult_features.append(X[:, i] * X[:, j])

    # Combinar
    squared_features = np.column_stack(squared_features)
    mult_features = np.column_stack(mult_features)
    X_new = np.column_stack((X, squared_features, mult_features))

    # Nuevos Valores
    new_feature_names = independent + [f"{independent[i]}^2" for i in range(num_features)
                      [f"{independent[i]}*{independent[j]}" for i in range(num_features)

    return X_new, new_feature_names


X, y = preprocess_data(data, life_expectancy, independent)
beta = evaluate_model(X, y)
print("\nCoeficientes: ", beta)
cross_validate(X, y)
monte_carlo_cv(X, y, n_iterations=1000, test_size=0.2)
loocv(X, y)
X_scaled, y = preprocess_data_scaled(data, life_expectancy, independent)
plot_ridge_curve(X_scaled, y)
lasso(X, y, independent)
```

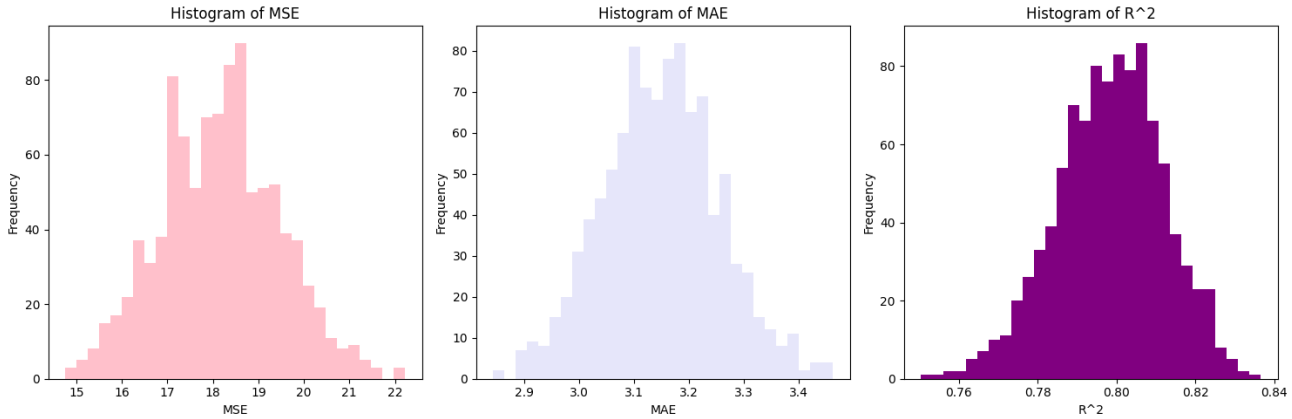





```
MSE:  17.904739039786506 MAE:  3.130027109393954 R^2:  0.8018608131537249

Coeficientes:  [ 5.34958902e+01 -2.10810717e-02 -1.63922332e-03  9.31497304e-02
 -4.34865426e-03 -3.14878540e-05  4.52330937e-02  5.49968460e-02
  1.18245338e-01 -4.84591323e-01  4.93149131e-05  3.36671703e-09
 -5.60709905e-02  7.11020515e+00  7.10424313e-01]
```

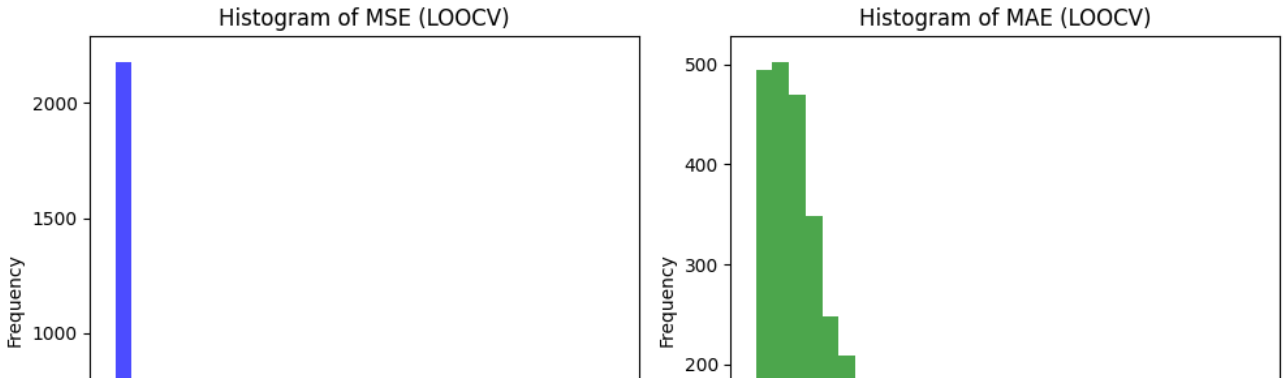
Cross Validated

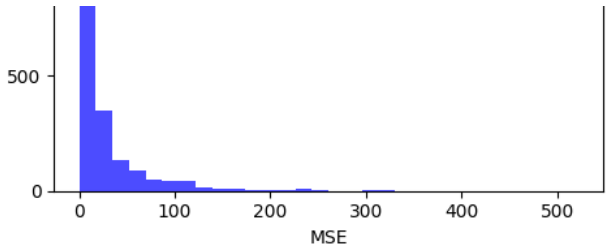
```
MSE: 18.09164535350007   MAE: 3.148347403977894   R^2: 0.7990182987367039
```

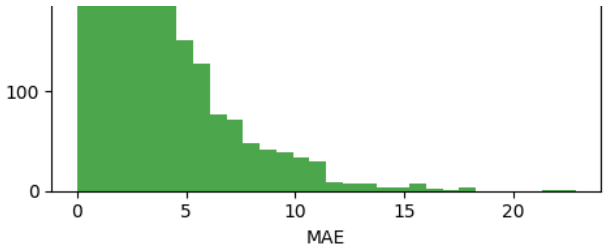


Monte Carlo CV

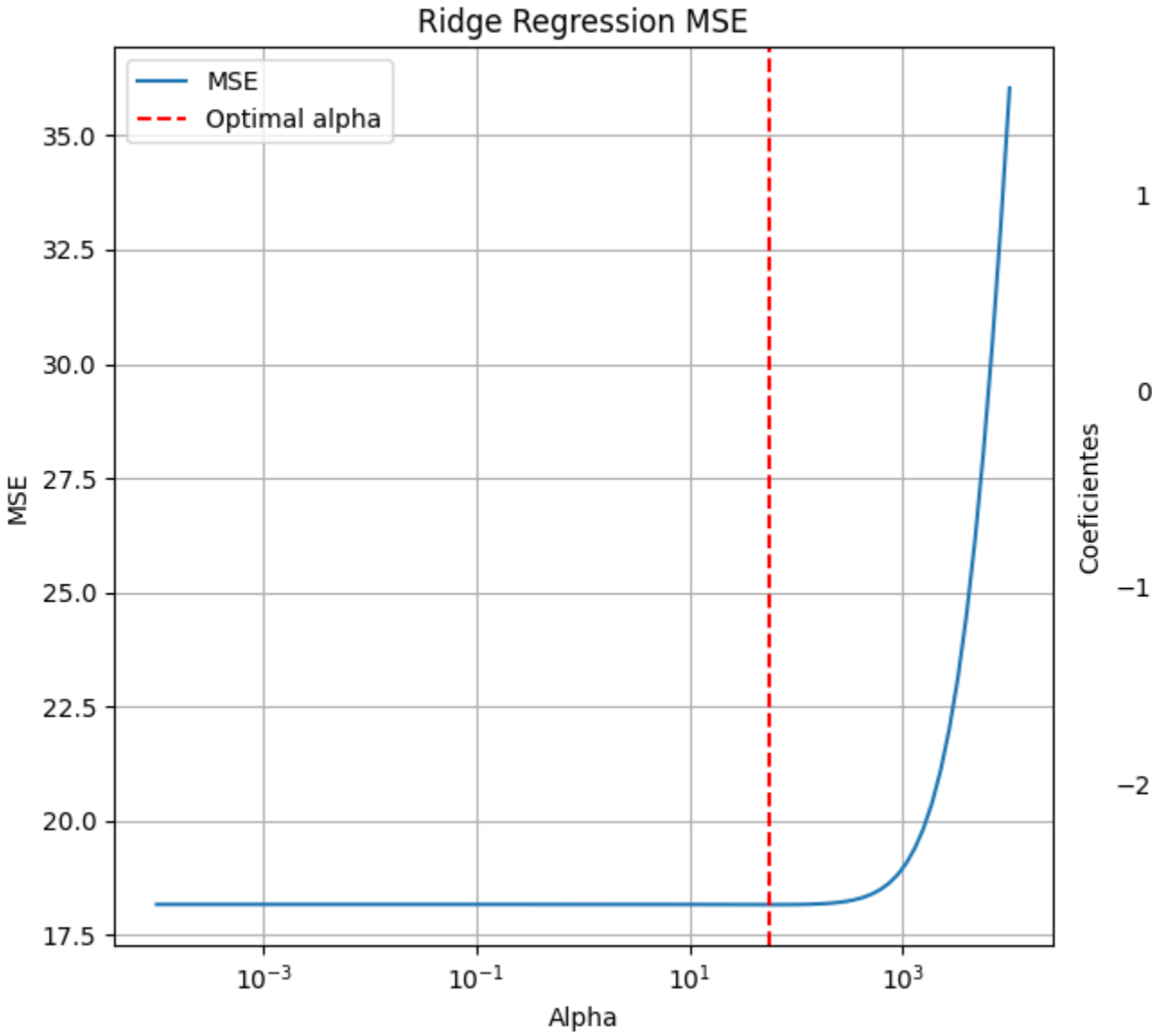
```
MSE Mean: 18.17079142985833   MAE Mean: 3.150464249272709   R^2 Mean: 0.798424128677
```

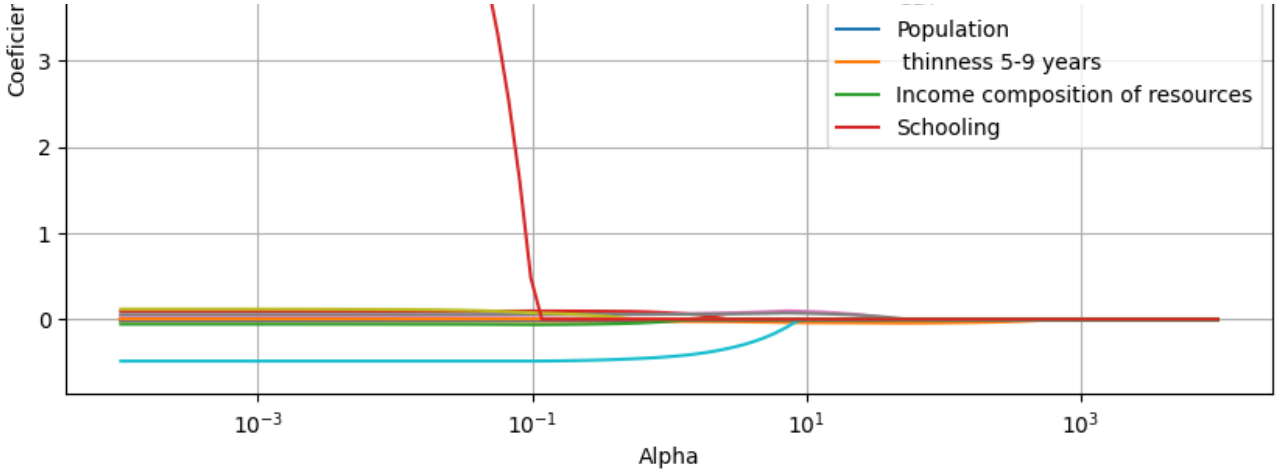




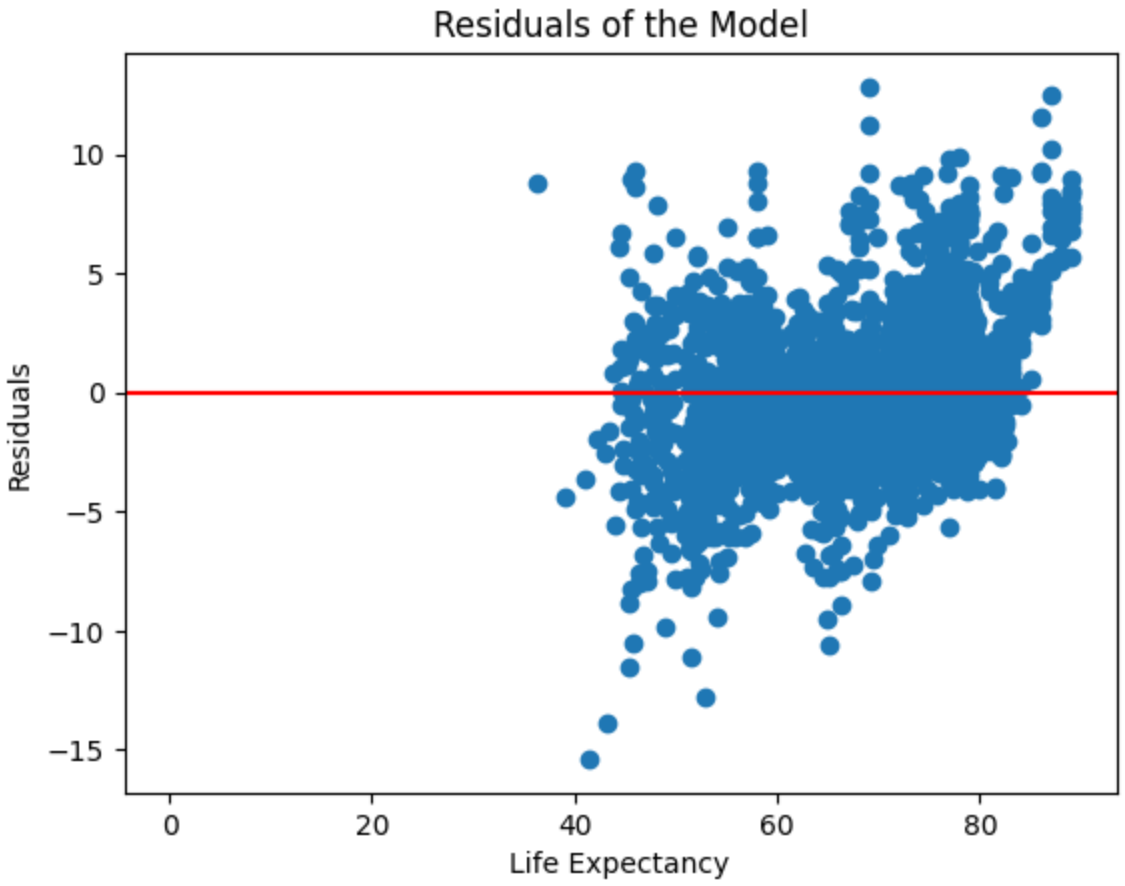



```
Leave-One-Out CV
MSE Mean: 18.168566351529453   MAE Mean: 3.1499944433569715
```






```
X_new, independent_extra = generate_extra(X, independent)

beta_extra = evaluate_model(X_new, y)
print("\nCoeficientes (Extra Features): ", beta_extra)

cross_validate(X_new, y)
```




```
MSE:  8.009588754437232 MAE:  2.05427145721382 R^2:  0.911363499951006

Coeficientes (Extra Features):  [ 4.97066694e+01  5.12609990e-02 -2.96884492e-02  1.
 -3.57211642e-02 -1.14969138e-04  1.95342470e-01 -1.90665094e-02
 -9.37427151e-02 -1.36376585e+00  1.98846404e-04  5.55700257e-08
 -5.49134953e-01 -4.13880897e+00  9.96082144e-01 -1.14899341e-04
  1.22022635e-05 -2.31325810e-02 -2.50695156e-04  3.15823530e-10
  2.60601744e-04  1.15750327e-03 -7.89188439e-03  1.88656236e-03
  2.27475237e-11  1.38683521e-19  3.66474706e-02  3.82445269e+01
  1.64479923e-02  3.79732411e-06 -2.66811323e-04  2.14979046e-05
  6.57408110e-08 -1.66977778e-04 -7.79308588e-05 -3.25465775e-04
  1.66744384e-03 -4.34375599e-07 -8.22760045e-12  3.87431158e-04
 -1.17921749e-02 -1.20240184e-03 -3.24259549e-04 -4.90184854e-05
 -7.29811894e-09  2.21889047e-04  1.76732562e-05 -2.29642511e-04
  1.43461478e-03 -5.30349923e-06 -1.64778679e-11 -8.89305732e-04
  2.34045520e-02  1.92679352e-03  6.50244502e-05 -7.03734668e-07
  1.36781435e-03 -9.84536258e-04 -1.22453175e-02  5.38003518e-03
  5.46264347e-06 -3.94920296e-10 -5.42527522e-02 -7.16431125e-01
 -5.67033423e-03  8.74030890e-08  6.72871032e-05  1.75365884e-04
  4.88613934e-03 -1.07125130e-03  8.75293928e-08 -6.24824079e-12
  1.58458855e-03  2.41748281e-02 -1.18385706e-03  3.20163510e-07
  1.04347655e-06  7.23561921e-06 -1.09875490e-06  1.69888764e-09
  5.81293324e-15  1.55715663e-06 -3.47389396e-05 -2.18748420e-06
 -8.80438251e-04  5.03861506e-04  3.65136458e-03 -1.36824406e-07
 -2.56508212e-10  1.13790908e-03 -5.21117509e-02 -7.14767424e-03
  1.74261675e-03  5.73379863e-04  1.29883294e-07  3.80756745e-12
 -1.30874641e-03 -8.16437698e-05 -4.23927532e-03  6.31764072e-03
 -2.38616973e-06  3.25610671e-09 -3.32095699e-02 -2.49533050e-01
```

```
 -2.38610975e-06  3.25010671e-09 -3.52699099e-02 -2.49555050e-01
  8.16179120e-03 -2.62680635e-05 -2.90409103e-09  1.22490937e-03
  2.50287830e-02  5.25673491e-02  1.21727181e-12  9.38055502e-06
 -4.18415214e-04  9.32391381e-06  5.23754927e-11 -7.19062606e-08
 -8.34348142e-10  2.81849630e-01 -4.98215662e-03 -1.04559598e+00]

Cross Validated
MSE: 10.107771792253649   MAE: 2.206609718505489   R^2: 0.8878354075024084
```

```
**1.- ¿Consideras que el modelo de regresión
los datos del problema? ¿Por qué?**

El R^2 es aproximandamente de 0.8, y a pesar
relativamente bajo para ser un modelo de regr
de que no diría que es el modelo ideal para p

**2.- ¿Observas una variabilidad importante e
cuando aplicas validación cruzada? Detalla tu

No, los valores de R2, MSE y MAE se mantienen
modelo de regresión lineal normal, con CV, co
hasta que agregamos los valores extras multip
varían los resultados.

**3.- ¿Qué modelo es mejor para los datos del
cuadrático? ¿Por qué? **

El cuadrático, dado que dio en general valore
bajos. Sin embargo yo probaría con modelos co
K-Vecinos para obtener resultados más acertad

**4.- ¿Qué variables son más relevantes para

En Lasso schooling parece ser la que tiene ma
encuentran pegadas entre si. Ridge parece fav
demás variables se ven más distribuidas.

**5.- ¿Encuentras alguna relación interesante
los predictores?** A pesar de que ciertas var
impacto significativo en la variable de respu
particularmente interesante. Schooling parece
pero sin el contexto adecuado quedan todavía
apto para nuestro conjunto de datos, podríamo
```

1.- ¿Consideras que el modelo de regresión lineal es efectivo para modelar los datos del problema? ¿Por qué?

El R^2 es aproximandamente de 0.8, y a pesar de que el MSE es alto, el MAE es relativamente bajo para ser un modelo de regresión lineal. Por lo que a pesar de que no diría que es el modelo ideal para predecir estos datos, es aceptable.

**2.- ¿Observas una variabilidad importante en los valores de R2, MSE y MAE cuando aplicas validación cruzada? Detalla tu respuesta. **

No, los valores de R2, MSE y MAE se mantienen bastante similares entre el modelo de regresión lineal normal, con CV, con Monte Carlo y con LOOCV. Es hasta que agregamos los valores extras multiplicados y cuadrados que vemos que varían los resultados.

**3.- ¿Qué modelo es mejor para los datos del problema, el lineal o el cuadrático? ¿Por qué? **

El cuadrático, dado que dio en general valores de error considerablemente más bajos. Sin embargo yo probaría con modelos como DecisionTree, RandomForest o K-Vecinos para obtener resultados más acertados.

4.- ¿Qué variables son más relevantes para el modelo según Ridge y Lasso?

En Lasso schooling parece ser la que tiene

más reelevancia, todas las demas se encuentran pegadas entre si. Ridge parece favorecer más schooling, pero las demás variables se ven más distribuidas.

5.- ¿Encuentras alguna relación interesante entre la variable de respuesta y los predictores? A pesar de que ciertas variables predictoras sí tienen un impacto significativo en la variable de respuesta, no encontré algo particularmente interesante. Schooling parece ser la más reelevante por mucho, pero sin el contexto adecuado quedan todavía muchas dudas, y con un modelo más apto para nuestro conjunto de datos, podríamos hacer un mejor analisis.