Maestría en Inteligencia Artificial Aplicada

Aprendizaje Automatico 1

Examen 3

Integrantes

- Andres Felipe Borrero
- Yesid Castelblanco
- Nicolas Colmenares
- Carlos Alberto Martinez

Profesores

- Santiago Ortiz
- Henry Velasco

Notas:

- Todas las respuestas, gráficas, tablas y operaciones deben ser debidamente justificadas.
- La información que sea obtenida de alguna fuente debe ser citada y referenciada en el documento a entregar.

```
# Importar librería pandas y numpy
# Pandas es una librería de Python que proporciona estructuras de
datos y herramientas de análisis de datos de alto rendimiento
import pandas as pd
# NumPy es una librería de Python que proporciona estructuras de datos
y operaciones matemáticas de alto rendimiento.
import numpy as np
#Matplotlib.pyplot es una librería de Python que proporciona
herramientas de visualización y gráficos de alta calidad.
import matplotlib.pyplot as plt
from IPython.display import Math, Latex
from IPython.core.display import Image
#Seaborn es una librería de Python que proporciona herramientas de
visualización y análisis de datos de alta calidad.
import seaborn as sns

#métricas
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from tqdm.auto import tqdm
import time
```

```
#Modelos

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_validate

#sns.set(color_codes=True) es un método de la librería Seaborn que
establece los códigos de color predeterminados para los gráficos y
diagramas. Cuando se establece color_codes=True, Seaborn utiliza una
paleta de colores predefinida y asigna un código de color a cada
categoría o variable en los gráficos y diagramas.
sns.set(color_codes=True)
#Cuando se establece rc={'figure.figsize':(10,6)}, Seaborn crea
figuras con un tamaño de 10 unidades de ancho y 6 unidades de alto.
sns.set(rc={'figure.figsize':(10,6)})
import warnings
warnings.filterwarnings('ignore')
```

Ejercicio 1

1) Considere el conjunto de datos *"Boston Housing Data"* presentados en Harrison and Rubinfeld (1978). Defina como variable respuesta a la columna *MEDV*. Realice una partición 80-20, donde el primer 80 % de los datos son datos de entrenamiento y el restante 20 % son datos para prueba.

Genere los modelos de regresión por regularización **Ridge, LASSO** y **Elastic-Net** para los datos de entrenamiento. Encuentre los valores óptimos de $\alpha*$ y $\lambda*$ junto a su respectiva gráfica de evolución de los coeficientes de regresión. Compare los modelos en términos de la selección de variables, interprete los coeficientes y escriba la ecuación ajustada de regresión para cada caso. Finalmente, realice una predicción con las observaciones de prueba y determine cual de los tres modelos es el mejor en capacidad predictiva **(RMSE)**.

```
# Instead of using load_boston, fetch the dataset directly from the
source:
data_url =
"https://github.com/cam2149/MachinelearningI/raw/32a4b5e8d5b784f79f7f5
5be8a2e6fbbfb61ccba/boston.csv"
dfBoston = pd.read_csv(data_url, sep="\s+", header=None)


header_row = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
```

```
dfBoston.columns = header_row
(dfBoston.head())
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX
\
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296.0

1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242.0

2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242.0

3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222.0

4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222.0


   PTRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

```
# Mostrar los primeros 5 datos del DataFrame
print("Primeros 5 datos:")
print(dfBoston.head())

# Mostrar los últimos 5 datos del DataFrame
print("\nÚltimos 5 datos:")
print(dfBoston.tail())
```

```
Primeros 5 datos:
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX
\
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296.0

1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242.0

2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242.0

3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222.0

4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222.0


   PTRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

```
Últimos 5 datos:
        CRIM   ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX
\
501  0.06263  0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273.0

502  0.04527  0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273.0

503  0.06076  0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273.0

504  0.10959  0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273.0

505  0.04741  0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273.0


     PTRATIO       B  LSTAT  MEDV
501     21.0  391.99   9.67  22.4
502     21.0  396.90   9.08  20.6
503     21.0  396.90   5.64  23.9
504     21.0  393.45   6.48  22.0
505     21.0  396.90   7.88  11.9

dfBoston.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB

#Considere el conjunto de datos “Boston Housing Data” presentados en
Harrison and Rubinfeld (1978). Defina como variable respuesta a la
columna MEDV.
# Realice una partición 80-20, donde el primer 80 % de los datos son
datos de entrenamiento y el restante 20 % son datos para prueba.
```

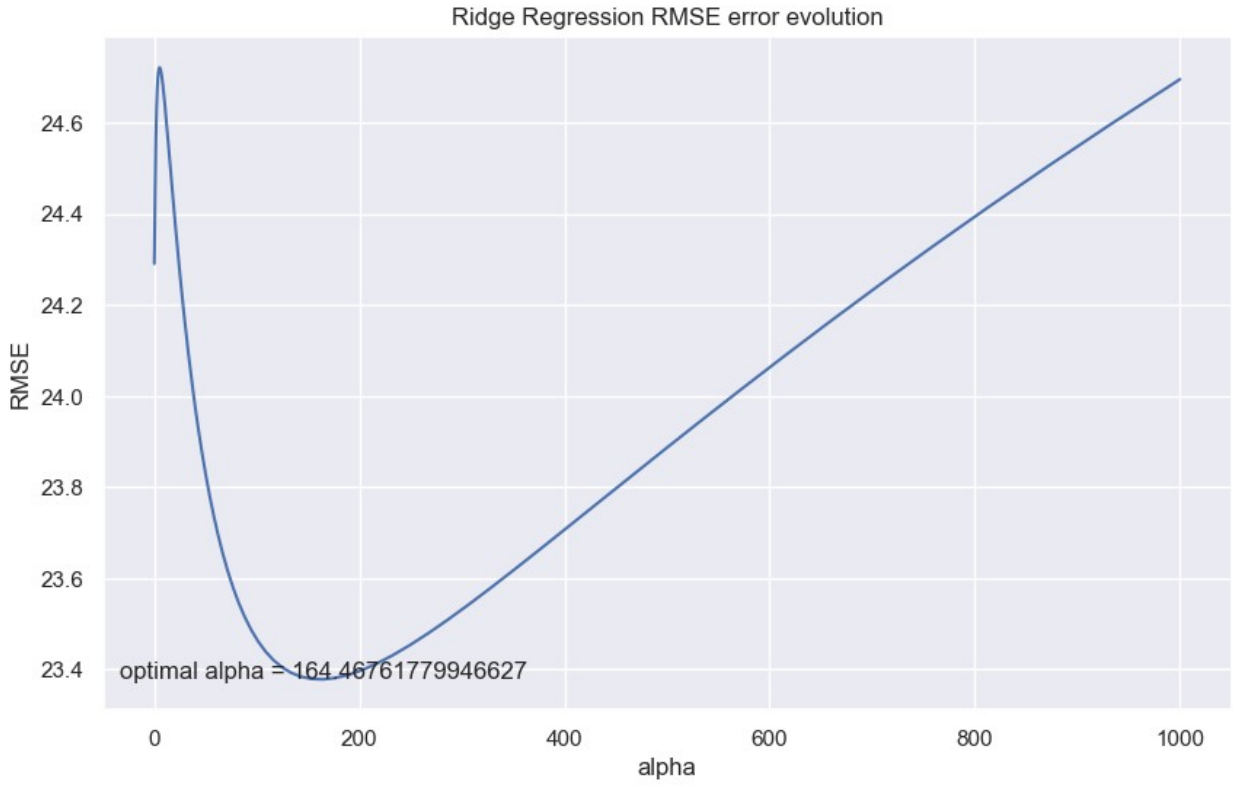
```
from sklearn.model_selection import train_test_split
# Definir la variable respuesta (MEDV) y las variables predictoras
X = dfBoston.drop('MEDV', axis=1)
y = dfBoston['MEDV']

# Realizar la partición 80-20
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Imprimir las dimensiones de los conjuntos de entrenamiento y prueba
print("Dimensiones del conjunto de entrenamiento:", X_train.shape,
y_train.shape)
print("Dimensiones del conjunto de prueba:", X_test.shape,
y_test.shape)

# We will use Ridge, Lasso and ElasticNet regression models in the
training data. For each  model we will find the optimal alpha and
lambda values and show the graph of the regression coefficients
evolution.

from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error

# Ridge Regression
ridge = Ridge()
ridge.fit(X_train, y_train)

# Find the optimal alpha value for Ridge Regression using the RMSE
error
alpha_space = np.logspace(-3, 3, 200)
ridge_scores = []
ridge_coefs = []

for alpha in alpha_space:
    ridge.alpha = alpha
    ridge.fit(X_train, y_train)
    y_pred = ridge.predict(X_test)
    ridge_scores.append(mean_squared_error(y_test, y_pred))
    ridge_coefs.append(ridge.coef_)

# Best ridge alpha
best_alpha_ridge = alpha_space[np.argmin(ridge_scores)]

# Plot the RMSE error evolution in a line plot
plt.plot(alpha_space, ridge_scores)
plt.xlabel('alpha')
plt.ylabel('RMSE')
plt.title('Ridge Regression RMSE error evolution')
# Add a label in the lowest point of the graph with the alpha value
plt.text(alpha_space[np.argmin(ridge_scores)], np.min(ridge_scores),
```

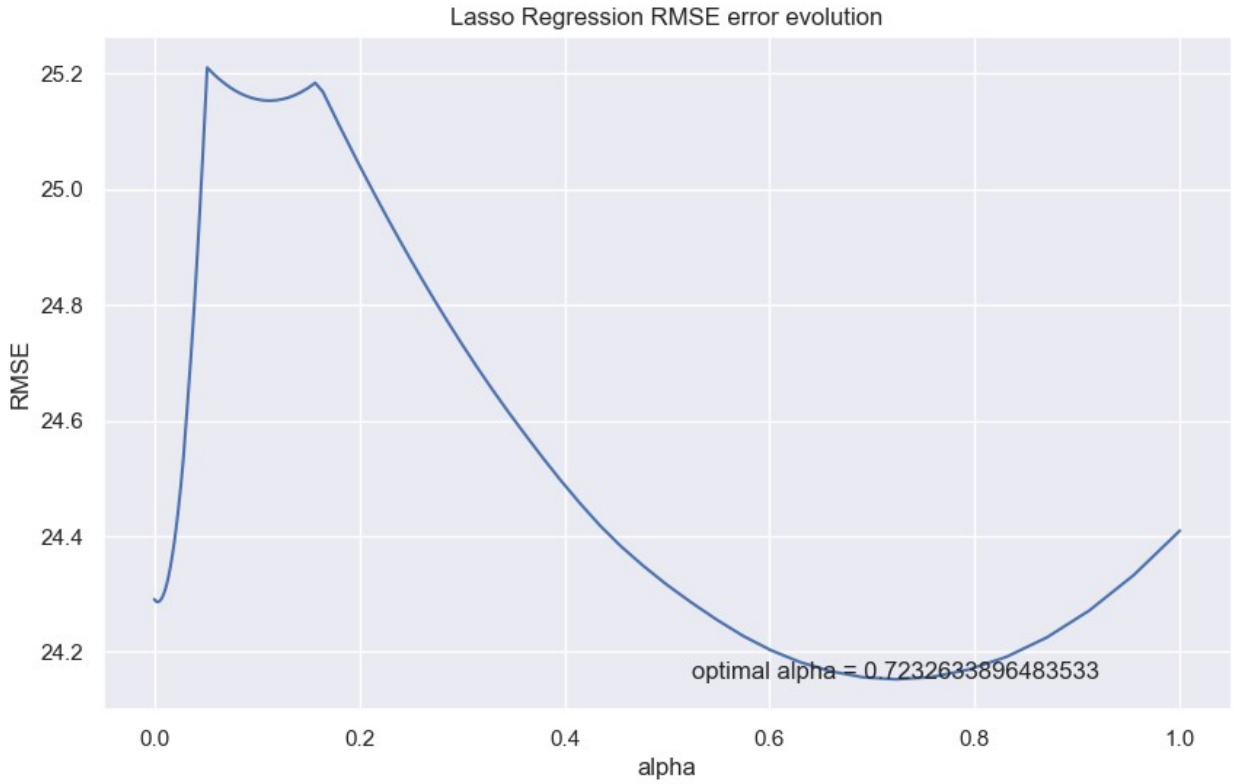
```
f'optimal alpha = {alpha_space[np.argmin(ridge_scores)]}',
ha='center')
plt.show()

# Lasso Regression
lasso = Lasso()
lasso.fit(X_train, y_train)

# Find the optimal alpha value for Lasso Regression using the RMSE
error
alpha_space = np.logspace(-4, 0, 200)
lasso_scores = []
lasso_coefs = []

for alpha in alpha_space:
    lasso.alpha = alpha
    lasso.fit(X_train, y_train)
    y_pred = lasso.predict(X_test)
    lasso_scores.append(mean_squared_error(y_test, y_pred))
    lasso_coefs.append(lasso.coef_)

# Best lasso alpha
best_alpha_lasso = alpha_space[np.argmin(lasso_scores)]

# Plot the RMSE error evolution in a line plot
plt.plot(alpha_space, lasso_scores)
plt.xlabel('alpha')
plt.ylabel('RMSE')
plt.title('Lasso Regression RMSE error evolution')
# Add a label in the lowest point of the graph with the alpha value
plt.text(alpha_space[np.argmin(lasso_scores)], np.min(lasso_scores),
f'optimal alpha = {alpha_space[np.argmin(lasso_scores)]}',
ha='center')
plt.show()

# ElasticNet Regression
elasticnet = ElasticNet()
elasticnet.fit(X_train, y_train)

# Find the optimal alpha and l1_ratio value for ElasticNet Regression
using the RMSE error
alpha_space = np.linspace(0, 1, 50)
l1_ratio_space = np.linspace(0, 1, 50)
elasticnet_scores = []
elasticnet_coefs = []

for alpha in alpha_space:
    for l1_ratio in l1_ratio_space:
        elasticnet.alpha = alpha
        elasticnet.l1_ratio = l1_ratio
```

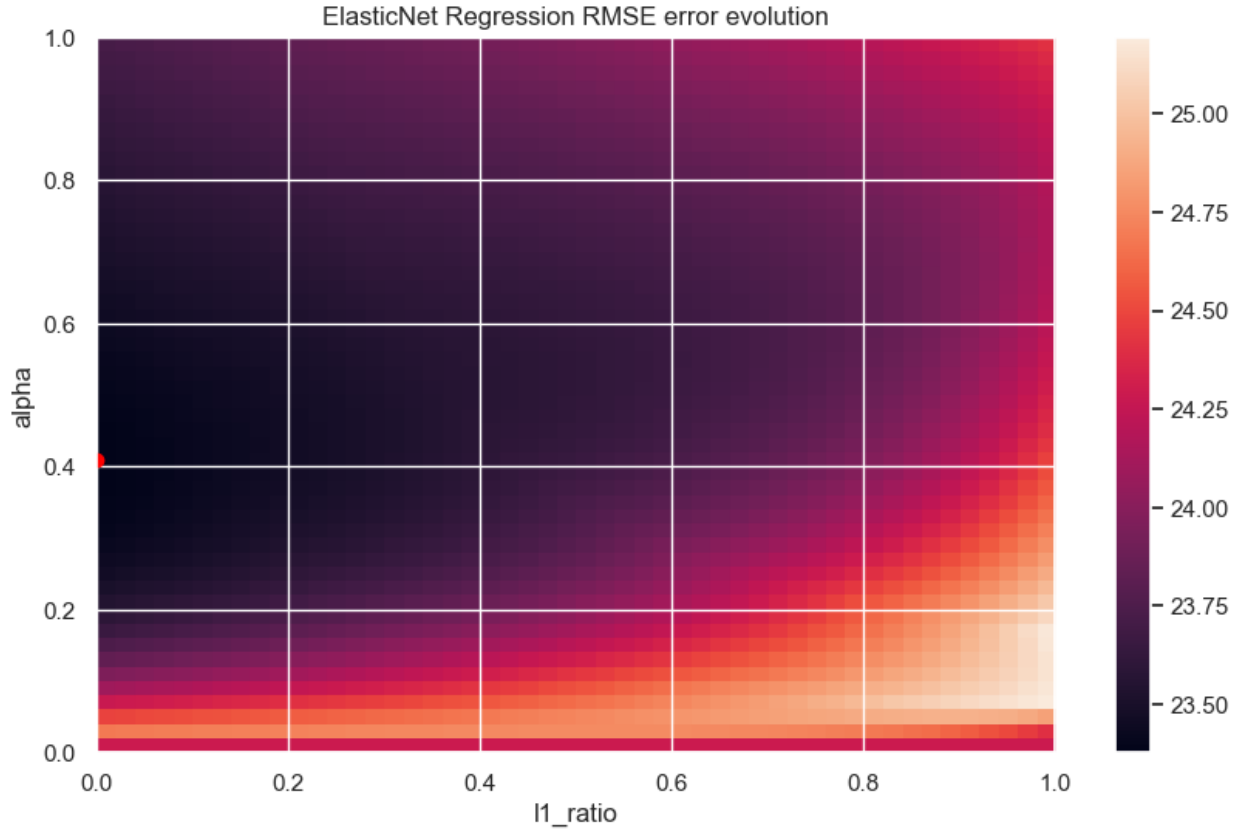
```
        elasticnet.fit(X_train, y_train)
        y_pred = elasticnet.predict(X_test)
        elasticnet_scores.append(mean_squared_error(y_test, y_pred))
        elasticnet_coefs.append(elasticnet.coef_)


# Plot the RMSE error evolution in a heatmap
elasticnet_scores = np.array(elasticnet_scores).reshape(50, 50)
plt.imshow(elasticnet_scores, origin='lower',
extent=[l1_ratio_space.min(), l1_ratio_space.max(), alpha_space.min(),
alpha_space.max()], aspect='auto')
plt.colorbar()
plt.xlabel('l1_ratio')
plt.ylabel('alpha')
plt.title('ElasticNet Regression RMSE error evolution')
# Show a marker in the lowest point of the heatmap with the optimal
alpha and l1_ratio values
optimal_alpha_en, optimal_l1_ratio_en =
np.unravel_index(np.argmin(elasticnet_scores),
elasticnet_scores.shape)
# plt.scatter(l1_ratio_space[optimal_l1_ratio],
alpha_space[optimal_alpha], color='red')
# plt.text(l1_ratio_space[optimal_l1_ratio],
alpha_space[optimal_alpha], f'optimal alpha =
{alpha_space[optimal_alpha]}\noptimal l1_ratio =
{l1_ratio_space[optimal_l1_ratio]}', ha='center')
# Show a red dot in the lowest point of the heatmap
plt.scatter(l1_ratio_space[optimal_l1_ratio_en],
alpha_space[optimal_alpha_en], color='red')
plt.show()

best_alpha_elasticnet = alpha_space[optimal_alpha_en]
best_l1_ratio_elasticnet = l1_ratio_space[optimal_l1_ratio_en]
# Compare the RMSE error of the three models with the optimal alpha
and l1_ratios values
ridge.alpha = best_alpha_ridge
ridge.fit(X_train, y_train)
ridge_y_pred = ridge.predict(X_test)
ridge_rmse = mean_squared_error(y_test, ridge_y_pred)

lasso.alpha = best_alpha_lasso
lasso.fit(X_train, y_train)
lasso_y_pred = lasso.predict(X_test)
lasso_rmse = mean_squared_error(y_test, lasso_y_pred)

elasticnet.alpha = best_alpha_elasticnet
elasticnet.l1_ratio = best_l1_ratio_elasticnet
elasticnet.fit(X_train, y_train)
elasticnet_y_pred = elasticnet.predict(X_test)
elasticnet_rmse = mean_squared_error(y_test, elasticnet_y_pred)
```

```
print(f'Ridge RMSE: {ridge_rmse}')
print(f'Lasso RMSE: {lasso_rmse}')
print(f'ElasticNet RMSE: {elasticnet_rmse}')
```

```
Dimensiones del conjunto de entrenamiento: (404, 13) (404,)
Dimensiones del conjunto de prueba: (102, 13) (102,)
```




Lasso Regression RMSE error evolution
RMSE
alpha
optimal alpha = 0.7232633896483533
25.2
25.0
24.8
24.6
24.4
24.2
0.0
0.2
0.4
0.6
0.8
1.0

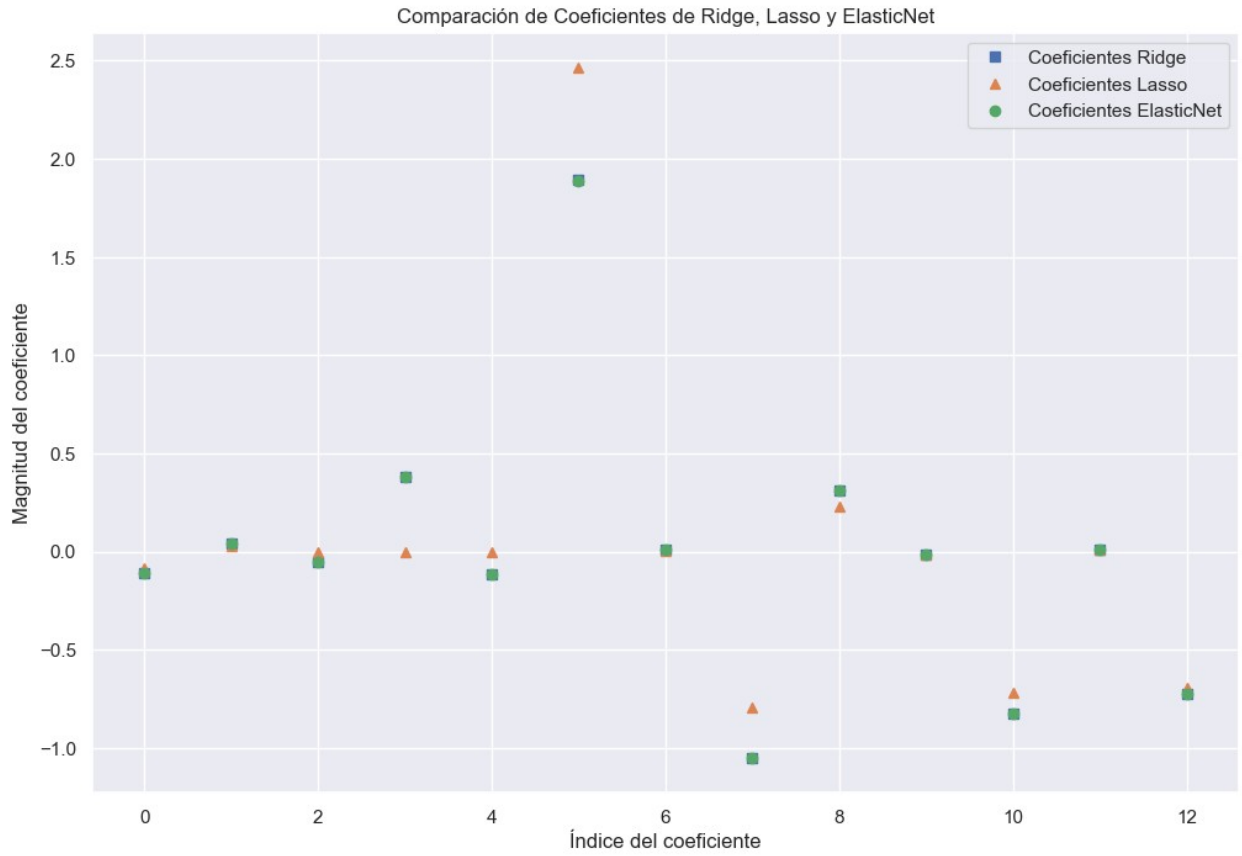

ElasticNet Regression RMSE error evolution
alpha
l1_ratio
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
25.00
24.75
24.50
24.25
24.00
23.75
23.50

```
Ridge RMSE: 23.37822100873452
Lasso RMSE: 24.15246650762958
ElasticNet RMSE: 23.37825903123421
```

```python
import numpy as np
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt


# Ridge Regression
ridge_best_model = Ridge(alpha=best_alpha_ridge).fit(X_train, y_train)

# Lasso Regression
lasso_best_model = Lasso(alpha=best_alpha_lasso).fit(X_train, y_train)

# ElasticNet Regression
elastic_net_best_model = ElasticNet(alpha=best_alpha_elasticnet,
l1_ratio=best_l1_ratio_elasticnet).fit(X_train, y_train)

# Imprimir los mejores parámetros
print("Mejor alpha para Ridge:", best_alpha_ridge)
print("Mejor alpha para Lasso:", best_alpha_lasso)
print("Mejor alpha para ElasticNet:", best_alpha_elasticnet)
print("Mejor l1_ratio para ElasticNet:", best_l1_ratio_elasticnet)

# Comparar coeficientes
ridge_coefs = ridge_best_model.coef_
lasso_coefs = lasso_best_model.coef_
elastic_net_coefs = elastic_net_best_model.coef_

# Graficar los coeficientes
plt.figure(figsize=(12, 8))
plt.plot(ridge_coefs, 's', label='Coeficientes Ridge')
plt.plot(lasso_coefs, '^', label='Coeficientes Lasso')
plt.plot(elastic_net_coefs, 'o', label='Coeficientes ElasticNet')
plt.xlabel('Índice del coeficiente')
plt.ylabel('Magnitud del coeficiente')
plt.legend()
plt.title('Comparación de Coeficientes de Ridge, Lasso y ElasticNet')
plt.show()

# Interpretación
print("Coeficientes Ridge:", ridge_coefs)
print("Coeficientes Lasso:", lasso_coefs)
print("Coeficientes ElasticNet:", elastic_net_coefs)

# Escribir la ecuación ajustada de regresión para cada modelo
def regression_equation(model, feature_names):
    intercept = model.intercept_
```

```
    coefs = model.coef_
    equation = f"y = {intercept:.4f}"
    for coef, feature in zip(coefs, feature_names):
        equation += f" + ({coef:.4f} * {feature})"
    return equation

feature_names = X.columns

ridge_equation = regression_equation(ridge_best_model, feature_names)
lasso_equation = regression_equation(lasso_best_model, feature_names)
elastic_net_equation = regression_equation(elastic_net_best_model,
feature_names)

print("Ecuación de regresión ajustada para Ridge:")
print(ridge_equation)
print("\nEcuación de regresión ajustada para Lasso:")
print(lasso_equation)
print("\nEcuación de regresión ajustada para ElasticNet:")
print(elastic_net_equation)
```

```
Mejor alpha para Ridge: 164.46761779946627
Mejor alpha para Lasso: 0.7232633896483533
Mejor alpha para ElasticNet: 0.4081632653061224
Mejor l1_ratio para ElasticNet: 0.0
```



```
Coeficientes Ridge: [-0.11058331  0.03995816 -0.0520205   0.3794143  -
0.11483231  1.89240276
  0.0071614  -1.05155008  0.309523   -0.0149496  -0.82269497
0.01159513
 -0.72621143]
Coeficientes Lasso: [-0.08569124  0.0306918  -0.          0.         -
0.          2.46446297
  0.00662688 -0.79304525  0.22883202 -0.01268625 -0.71997462
0.01183225
 -0.692407  ]
Coeficientes ElasticNet: [-0.11057984  0.03995648 -0.05203145
0.37865156 -0.11449418  1.8896236
  0.0071983  -1.05104721  0.30956454 -0.01495086 -0.82267867
0.01159323
 -0.72639433]
Ecuación de regresión ajustada para Ridge:
y = 38.0329 + (-0.1106 * CRIM) + (0.0400 * ZN) + (-0.0520 * INDUS) +
(0.3794 * CHAS) + (-0.1148 * NOX) + (1.8924 * RM) + (0.0072 * AGE) +
(-1.0516 * DIS) + (0.3095 * RAD) + (-0.0149 * TAX) + (-0.8227 *
PTRATIO) + (0.0116 * B) + (-0.7262 * LSTAT)

Ecuación de regresión ajustada para Lasso:
y = 30.3345 + (-0.0857 * CRIM) + (0.0307 * ZN) + (-0.0000 * INDUS) +
(0.0000 * CHAS) + (-0.0000 * NOX) + (2.4645 * RM) + (0.0066 * AGE) +
(-0.7930 * DIS) + (0.2288 * RAD) + (-0.0127 * TAX) + (-0.7200 *
PTRATIO) + (0.0118 * B) + (-0.6924 * LSTAT)

Ecuación de regresión ajustada para ElasticNet:
y = 38.0487 + (-0.1106 * CRIM) + (0.0400 * ZN) + (-0.0520 * INDUS) +
(0.3787 * CHAS) + (-0.1145 * NOX) + (1.8896 * RM) + (0.0072 * AGE) +
(-1.0510 * DIS) + (0.3096 * RAD) + (-0.0150 * TAX) + (-0.8227 *
PTRATIO) + (0.0116 * B) + (-0.7264 * LSTAT)
```

Conclusiones

- Se observa que la regresión optimizada de Lasso utiliza por lo general magnitudes mayores para los distintos coeficients
- Las magnitudes seleccionadas por Ridge y Elastic Net presentan valores muy similares
- El RMSE de Ridge y Elastic Net es muy similar y de un valor menor a Lasso. Esto se debe a que la optimización indicó un valor de 0 para la regularización L1 lo cual traduce a que Lasso tenga muy poca influencia. En este caso, Elastic-Net se aproxima a Ridge, utilizando solo regularización L2.
- No se evidencia una diferencia significativa entre la capacidad predictiva (RMSE) entre Ridge y Elastic-Net. Sin embargo, por la menor complejidad a la hora de entrenar y optimizar el modelo de Ridge, se recomienda el uso de este para el problema presente de Housing.

Maestría en Inteligencia Artificial Aplicada

Aprendizaje Automatico 1

Examen 3

Integrantes

- Andres Felipe Borrero
- Yesid Castelblanco
- Nicolas Colmenares
- Carlos Alberto Martinez

Profesores

- Santiago Ortiz
- Henry Velasco

Notas:

- Todas las respuestas, gráficas, tablas y operaciones deben ser debidamente justificadas.
- La información que sea obtenida de alguna fuente debe ser citada y referenciada en el documento a entregar.

Ejercicio 2

2) El conjunto de datos *"YearPredictionMSD"* contiene información sobre canciones de música po- pular y el año en que se grabaron. Incluye 515345 observaciones y 90 características, como la intensidad media del sonido, la varianza del espectro de frecuencia y la correlación entre las características espectrales. El objetivo es predecir el año en que se grabó la canción.

- Carque el conjunto de datos usando la función *read_csv* del paquete **pandas** y el como primer argumento el Link, use como segundo argumendo *header = None*.
- Divida el conjunto de datos en características o variables explicativas X y variable objetivo Y, tenga en cuenta que se quiere modelar el año en que se grabó la canción.
- Reduzca la dimensión de las variables. Para ello, use un modelo de regresión **LASSO** con un coeficiente de penalización de 10, para extraer características importantes del conjunto de variables explicativas.
- Con el conjunto de variables reducido, ajuste un modelo de regresión OLS e interprete su significancia y su R2 .
- Revise los supuestos de los errores, y con los hallazgos del ítem anterior, concluya sobre la conveniencia de usar este modelo para predecir el año de grabación de la canción.

Carque el conjunto de datos usando la función read_csv del paquete *pandas* y como primer argumento el Link, use como segundo argumendo *header = None.

El conjunto de datos "YearPredictionMSD" contiene información sobre canciones con unas caracteristicas extraidas de los archivos de audio y el objetivo es predecir el año en el que cada canción fue grabada.

```
# Cargamos la libreria Pandas

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Cargamos el conjunto de datos por medio de la función read_csv de
pandas

data_url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/00203/YearPredictionMSD.txt.zip"

df = pd.read_csv(data_url, header=None)
```

Consultamos las primeras (#) filas del dataset

```
df.head(5)
```

```
{"type":"dataframe","variable_name":"df"}
```

Consultamos las ultimas (#) filas del dataset

```
df.tail(5)
```

```
{"type":"dataframe"}
```

```
# Consultamos los tipos de datos de las variables en el DataFrame para poder decidir métodos adecuados para la visualización de los datos mediante analisis bivariado

df.dtypes

# Pandas almacena las variables categóricas como 'object' y las variables continuas se almacenan como int o float
```

```
0       int64
1     float64
2     float64
3     float64
4     float64
       ...
86    float64
87    float64
88    float64
89    float64
90    float64
Length: 91, dtype: object
```

```
# Eliminamos filas con valores NaN
df = df.dropna()

# Verificamos si hay valores faltantes

print(df.isnull().sum())
```

```
0     0
1     0
2     0
3     0
4     0
     ..
86    0
87    0
88    0
89    0
90    0
Length: 91, dtype: int64
```

VISUALIZACIÓN DE LOS DATOS DEL DATAFRAME

```
# Resumen del Dataframe útil para depuración y comprensión inicial de los datos
```

```
# Podemos ver cuántos valores no nulos hay en cada columna lo que
ayuda a identificar la presencia de valores faltantes.

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 515345 entries, 0 to 515344
Data columns (total 91 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   0       515345 non-null  int64  
 1   1       515345 non-null  float64
 2   2       515345 non-null  float64
 3   3       515345 non-null  float64
 4   4       515345 non-null  float64
 5   5       515345 non-null  float64
 6   6       515345 non-null  float64
 7   7       515345 non-null  float64
 8   8       515345 non-null  float64
 9   9       515345 non-null  float64
 10  10      515345 non-null  float64
 11  11      515345 non-null  float64
 12  12      515345 non-null  float64
 13  13      515345 non-null  float64
 14  14      515345 non-null  float64
 15  15      515345 non-null  float64
 16  16      515345 non-null  float64
 17  17      515345 non-null  float64
 18  18      515345 non-null  float64
 19  19      515345 non-null  float64
 20  20      515345 non-null  float64
 21  21      515345 non-null  float64
 22  22      515345 non-null  float64
 23  23      515345 non-null  float64
 24  24      515345 non-null  float64
 25  25      515345 non-null  float64
 26  26      515345 non-null  float64
 27  27      515345 non-null  float64
 28  28      515345 non-null  float64
 29  29      515345 non-null  float64
 30  30      515345 non-null  float64
 31  31      515345 non-null  float64
 32  32      515345 non-null  float64
 33  33      515345 non-null  float64
 34  34      515345 non-null  float64
 35  35      515345 non-null  float64
 36  36      515345 non-null  float64
 37  37      515345 non-null  float64
 38  38      515345 non-null  float64
 39  39      515345 non-null  float64
```

```
40  40      515345 non-null  float64
41  41      515345 non-null  float64
42  42      515345 non-null  float64
43  43      515345 non-null  float64
44  44      515345 non-null  float64
45  45      515345 non-null  float64
46  46      515345 non-null  float64
47  47      515345 non-null  float64
48  48      515345 non-null  float64
49  49      515345 non-null  float64
50  50      515345 non-null  float64
51  51      515345 non-null  float64
52  52      515345 non-null  float64
53  53      515345 non-null  float64
54  54      515345 non-null  float64
55  55      515345 non-null  float64
56  56      515345 non-null  float64
57  57      515345 non-null  float64
58  58      515345 non-null  float64
59  59      515345 non-null  float64
60  60      515345 non-null  float64
61  61      515345 non-null  float64
62  62      515345 non-null  float64
63  63      515345 non-null  float64
64  64      515345 non-null  float64
65  65      515345 non-null  float64
66  66      515345 non-null  float64
67  67      515345 non-null  float64
68  68      515345 non-null  float64
69  69      515345 non-null  float64
70  70      515345 non-null  float64
71  71      515345 non-null  float64
72  72      515345 non-null  float64
73  73      515345 non-null  float64
74  74      515345 non-null  float64
75  75      515345 non-null  float64
76  76      515345 non-null  float64
77  77      515345 non-null  float64
78  78      515345 non-null  float64
79  79      515345 non-null  float64
80  80      515345 non-null  float64
81  81      515345 non-null  float64
82  82      515345 non-null  float64
83  83      515345 non-null  float64
84  84      515345 non-null  float64
85  85      515345 non-null  float64
86  86      515345 non-null  float64
87  87      515345 non-null  float64
88  88      515345 non-null  float64
```

```
 89  89      515345 non-null  float64
 90  90      515345 non-null  float64
dtypes: float64(90), int64(1)
memory usage: 357.8 MB
```

Caracteristicas estadisticas del dataframe

```
# Mostrar caracteristicas estadísticas de los datos en un DataFrame
para identificar anomalias o patrones

df.describe()
```

```
{"type":"dataframe"}
```

NUMERO DE COLUMNAS Y NUMERO DE FILAS DEL DATAFRAME

```
# Obtener información sobre dimensiones de un DataFrame

df.shape

# (filas, columnas)
```

```
(515345, 91)
```

Divida el conjunto de datos en características o variables explicativas X y variable objetivo Y, tenga en cuenta que se quiere modelar el año en que se grabó la canción.

Como variable objetivo tenemos el año de grabación de la canción, las demás columnas corresponden a las caracteristicas de las canciones.

```
# Seleccionamos la primera columna que corresponde a "Year" y es la variable objetivo. Estos lo hacemos utilizando la función de pandas df.iloc[filas, columnas]

Y = df.iloc[:, 0].values

# Todas las columnas menos la primera, quedan las características o variables explicativas

X = df.iloc[:, 1:].values

# Comprobamos que (X) contiene las caracteristicas

X
```

```
array([[ 4.994357e+01,  2.147114e+01,  7.307750e+01, ..., -1.822230e+00,
        -2.746348e+01,  2.263270e+00],
       [ 4.873215e+01,  1.842930e+01,  7.032679e+01, ...,  1.204941e+01,
         5.843453e+01,  2.692061e+01],
       [ 5.095714e+01,  3.185602e+01,  5.581851e+01, ..., -5.859000e-02,
         3.967068e+01, -6.634500e-01],
       ...,
       [ 4.512852e+01,  1.265758e+01, -3.872018e+01, ..., -6.071710e+00,
         5.396319e+01, -8.093640e+00],
       [ 4.416614e+01,  3.238368e+01, -3.349710e+00, ...,  2.032240e+01,
         1.483107e+01,  3.974909e+01],
       [ 5.185726e+01,  5.911655e+01,  2.639436e+01, ..., -5.515120e+00,
         3.235602e+01,  1.217352e+01]])
```

```
# Comprobamos que (Y) contiene la variable objetivo

Y
```

```
array([2001, 2001, 2001, ..., 2006, 2006, 2005])
```

Reduzca la dimensión de las variables. Para ello, use un modelo de regresión LASSO con un coeficiente de penalización de 10, para extraer características importantes del conjunto de variables explicativas.

El conjunto de datos que usamos en nuestro ejercicio es complejo debido a que las caracteristicas presentes tienen unidades y escalas muy diferentes lo cual permite que algunas caracteristicas dominen el modelo mientras que otras sean menos significativas, debido a esto, es importante normalizar o estandarizar las caracteristicas.

```
from sklearn.preprocessing import StandardScaler

# Estandarizamos las caracteristicas

Scaler = StandardScaler()
X_scaled = Scaler.fit_transform(X)
```

Reducimos la dimensionalidad con el modelo de regresión LASSO

Usamos un modelo de regresión LASSO con un coeficiente de penalización o parámetro alpha de 10. El modelo realiza una regularización (penalización) y puede reducir algunos coeficientes a cero, de esa manera ayuda a identificar las características más importantes.

Dividimos el conjunto de datos en entrenamiento y prueba

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

Ajustamos el modelo de regresión LASSO

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error

# Crear el modelo Lasso con un coeficiente de penalización de 10
lasso = Lasso(alpha=10)
```

```
# Ajustar el modelo Lasso a los datos
lasso.fit(X_train, y_train)
```

```
Lasso(alpha=10)
```

Evaluamos el modelo

```
score = lasso.score(X_test, y_test)
print(f'R² en el conjunto de prueba: {score:.4f}')
```

```
R² en el conjunto de prueba: -0.0000
```

Examinamos los coeficientes del modelo

El modelo LASSO establece los coeficientes de algunas caracteristicas a 0 para reducir la dimensionalidad. Las caracteristicas con coeficientes diferentes a 0 con las que tienen mas relevancia.

```
# Coeficientes del modelo

coefficients = lasso.coef_
print("Coeficientes del modelo LASSO:")
print(coefficients)
```

```
Coeficientes del modelo LASSO:
[ 0.  0. -0. -0.  0. -0.  0. -0. -0.  0.  0. -0.  0.  0. -0.  0.  0.
0.
 -0.  0. -0.  0.  0. -0.  0. -0. -0.  0.  0.  0. -0.  0. -0.  0. -0. -
0.
 -0.  0.  0. -0. -0.  0. -0.  0.  0. -0.  0. -0. -0. -0.  0. -0.  0. -
0.
  0.  0. -0. -0. -0. -0. -0. -0. -0.  0. -0. -0. -0.  0. -0.  0. -0. -
0.
  0.  0.  0.  0. -0. -0.  0. -0. -0.  0.  0.  0.  0. -0.  0. -0.  0. -
0.]
```

```
# Identificamos las caracteristicas seleccionadas (coeficientes no
nulos)

selected_features = np.where(coefficients != 0)[0]
print(f'Número de características seleccionadas:
{len(selected_features)}')
print(f'Índices de las características seleccionadas:
{selected_features}')
```

```
Número de características seleccionadas: 0
Índices de las características seleccionadas: []
```

```
# Consultamos las caracteristicas seleccionadas

print(f'Coeficientes: {coefficients[selected_features]}')
```

```
Coeficientes: []
```

Reducimos las dimensiones

```
# Creamos un nuevo conjunto de datos solo con las caracteristicas
seleccionadas

X_selected = X_train[:, selected_features]
print(X_selected.shape)
```

```
(412276, 0)
```

Si obtenemos 0 como resultado tanto en el número de características seleccionadas como en los índices de características seleccionadas, esto indica que el modelo no ha seleccionado ninguna característica (es decir, todos los coeficientes son cero).

El modelo LASSO (Least Absolute Shrinkage and Selection Operator) puede hacer que todos los coeficientes se reduzcan a cero si el valor del parámetro de regularización (alpha) es muy grande. Esto sucede porque el LASSO penaliza fuertemente los coeficientes, y en situaciones extremas, puede hacer que todos los coeficientes sean cero, lo que significa que no se selecciona ninguna característica.

Teniendo en cuenta lo anterior, ajustamos el alpha dado a que alpha = 10 es demasiado grande. Lo ajustamos cuidadosamente para balancear la penalización y la capacidad predictiva del modelo, tampoco lo podemos dejar muy pequeño porque el modelo puede sobreajustarse (overfitting).

Una forma muy comun de ajuste es utilizando tecnicas de validación cruzada, para ello vamos a utilizar el modelo LassoCV el cual ajusta automaticamente el mejor valor de alpha.

```
from sklearn.linear_model import Lasso, ElasticNet, LassoCV,
ElasticNetCV
import pandas as pd
import numpy as np

modelo_lasso = LassoCV(alphas = np.logspace(-5, 2, 100),
                       fit_intercept = True,
                       #normalize = True,
                       cv = 10)

modelo_lasso.fit(X, Y)

# Imprimir el mejor valor de alpha y el coeficiente
```

```
print(f"Coeficientes del modelo: {modelo_lasso.coef_}")
print(f"Mejor alpha seleccionado: {modelo_lasso.alpha_}")
```

```
Coeficientes del modelo: [ 8.74405774e-01 -5.61905933e-02 -
4.35410653e-02  2.57668541e-03
 -1.46600182e-02 -2.19396876e-01 -6.23253677e-03 -1.00229980e-01
 -6.99138715e-02  2.45172175e-02 -1.62656132e-01 -2.29914235e-03
  4.69101974e-02  3.54928772e-04 -4.23969513e-04  6.01459000e-04
  4.77043738e-04  1.46902802e-03  1.92420543e-03  2.13384327e-03
  7.68640103e-04 -4.04820882e-04  7.54250198e-03  2.80667142e-03
 -3.54616975e-03  7.07143244e-05  1.59050740e-03  5.26923687e-04
  8.73645101e-04 -3.00878682e-04 -1.41109129e-03 -1.39883433e-03
 -5.55268104e-03  2.44647086e-03  1.83837878e-03 -5.28649664e-03
 -2.74483778e-04  6.80560172e-04  1.36830314e-03 -1.70818021e-03
 -2.01071028e-03 -7.64717873e-04 -1.40561111e-03 -2.34890863e-03
 -3.16450215e-03  6.76795146e-03  4.55124819e-04 -2.07385958e-03
  2.73147217e-04  1.93603183e-03  2.17116566e-04 -1.59976342e-03
  1.94624948e-03  4.80181894e-04 -7.44520366e-05  1.57596723e-04
 -1.89876263e-03  1.94239240e-03 -1.30708323e-03  2.24835437e-04
 -3.03322719e-03 -1.87206590e-03 -7.72908955e-03  1.18935351e-03
 -2.01610333e-03  6.59290271e-04 -1.84776604e-04 -4.28719267e-04
 -4.25280904e-03 -5.03983203e-03 -1.06292820e-03  2.38905313e-04
  7.04383496e-04  3.99118048e-03  2.99804274e-03  1.51792239e-02
  1.97043412e-04 -4.42985948e-03 -4.43329172e-05 -1.52664113e-04
 -7.97532583e-04 -5.55921228e-04  1.38430569e-03  1.00230545e-03
  2.61160779e-02  9.49760307e-05  1.15915528e-03 -3.10838241e-02
 -1.37819946e-03 -1.61427795e-03]
Mejor alpha seleccionado: 0.0093260334688322
```

En este caso, LassoCV selecciona el valor de alpha que minimiza el error de validación, y no es tan grande como para reducir todos los coeficientes a cero, pero tampoco es tan pequeño como para permitir un sobreajuste.

Mejor alpha : 0.0093

```
# Crear el modelo Lasso con un coeficiente de penalización de 10
lasso = Lasso(alpha=0.0093)

# Ajustar el modelo Lasso a los datos
lasso.fit(X_train, y_train)
```

```
Lasso(alpha=0.0093)
```

```
# Coeficientes del modelo

coefficients = lasso.coef_
print("Coeficientes del modelo LASSO:")
print(coefficients)
```

```
Coeficientes del modelo LASSO:
[ 5.23245271e+00 -2.84654225e+00 -1.48739861e+00 -0.00000000e+00
 -3.12246504e-01 -2.80257968e+00 -8.31379120e-02 -7.43421503e-01
 -7.15425656e-01  1.58889741e-01 -6.70163327e-01 -1.70992352e-02
  1.00755829e+00  6.10234375e-01 -5.22237132e-01  6.21811702e-01
  2.04929282e-01  8.10870793e-01  6.17198998e-01  6.60120869e-01
  1.26323637e-01 -0.00000000e+00  1.40861001e+00  4.19598385e-01
 -3.91859637e-01  3.05924229e-02  8.49015669e-01  1.06800056e-01
  1.22157050e-01 -2.34233001e-02 -1.24961304e-01 -9.09377730e-02
 -3.75704239e-01  1.18055776e-01  5.27877205e-02 -5.36975100e-01
 -7.88887393e-02  3.18795137e-01  3.69961945e-01 -3.47731546e-01
 -2.53169543e-01 -1.08127176e-01 -9.59730259e-02 -7.07922891e-02
 -1.11620829e-01  3.11342278e-01  2.15688704e-01 -5.32278651e-01
  6.19977937e-03  2.24434297e-01  9.33999311e-03 -9.53640780e-02
  1.22371706e-01  4.31249142e-03 -0.00000000e+00  1.41854220e-02
 -5.79982559e-01  4.97328112e-01 -2.23065841e-01  0.00000000e+00
 -1.70030110e-01 -1.00191797e-01 -2.32503100e-01  3.56093662e-01
 -4.22280066e-01  7.14910430e-02 -2.00757512e-02 -4.41342856e-02
 -4.49767963e-01 -1.47115447e-01 -2.70143163e-01  4.91389390e-02
  1.27183134e-01  2.45636726e-01  1.83754978e-01  3.65745985e-01
  2.58152554e-02 -5.99879142e-01 -0.00000000e+00 -3.11377405e-03
  0.00000000e+00 -6.34719817e-02  1.69135025e-01  1.03022128e-01
  4.21071360e-01 -6.43537462e-03  1.76707055e-01 -3.81395336e-01
 -2.34887394e-01 -2.96456904e-02]

# Identificamos las caracteristicas seleccionadas (coeficientes no
nulos)

selected_features = np.where(coefficients != 0)[0]
print(f'Número de características seleccionadas:
{len(selected_features)}')
print(f'Índices de las características seleccionadas:
{selected_features}')

Número de características seleccionadas: 84
Índices de las características seleccionadas: [ 0  1  2  4  5  6  7  8
9 10 11 12 13 14 15 16 17 18 19 20 22 23 24 25
 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49
 50 51 52 53 55 56 57 58 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
75
 76 77 79 81 82 83 84 85 86 87 88 89]

# Consultamos las caracteristicas seleccionadas

print(f'Coeficientes: {coefficients[selected_features]}')

Coeficientes: [ 5.23245271e+00 -2.84654225e+00 -1.48739861e+00 -
3.12246504e-01
 -2.80257968e+00 -8.31379120e-02 -7.43421503e-01 -7.15425656e-01
```

```
  1.58889741e-01 -6.70163327e-01 -1.70992352e-02  1.00755829e+00
  6.10234375e-01 -5.22237132e-01  6.21811702e-01  2.04929282e-01
  8.10870793e-01  6.17198998e-01  6.60120869e-01  1.26323637e-01
  1.40861001e+00  4.19598385e-01 -3.91859637e-01  3.05924229e-02
  8.49015669e-01  1.06800056e-01  1.22157050e-01 -2.34233001e-02
 -1.24961304e-01 -9.09377730e-02 -3.75704239e-01  1.18055776e-01
  5.27877205e-02 -5.36975100e-01 -7.88887393e-02  3.18795137e-01
  3.69961945e-01 -3.47731546e-01 -2.53169543e-01 -1.08127176e-01
 -9.59730259e-02 -7.07922891e-02 -1.11620829e-01  3.11342278e-01
  2.15688704e-01 -5.32278651e-01  6.19977937e-03  2.24434297e-01
  9.33999311e-03 -9.53640780e-02  1.22371706e-01  4.31249142e-03
  1.41854220e-02 -5.79982559e-01  4.97328112e-01 -2.23065841e-01
 -1.70030110e-01 -1.00191797e-01 -2.32503100e-01  3.56093662e-01
 -4.22280066e-01  7.14910430e-02 -2.00757512e-02 -4.41342856e-02
 -4.49767963e-01 -1.47115447e-01 -2.70143163e-01  4.91389390e-02
  1.27183134e-01  2.45636726e-01  1.83754978e-01  3.65745985e-01
  2.58152554e-02 -5.99879142e-01 -3.11377405e-03 -6.34719817e-02
  1.69135025e-01  1.03022128e-01  4.21071360e-01 -6.43537462e-03
  1.76707055e-01 -3.81395336e-01 -2.34887394e-01 -2.96456904e-02]
```

```python
# Creamos un nuevo conjunto de datos solo con las caracteristicas seleccionadas

X_selected = X_train[:, selected_features]
print(X_selected.shape)
```

```
(412276, 84)
```

Evaluamos el rendimiento del modelo

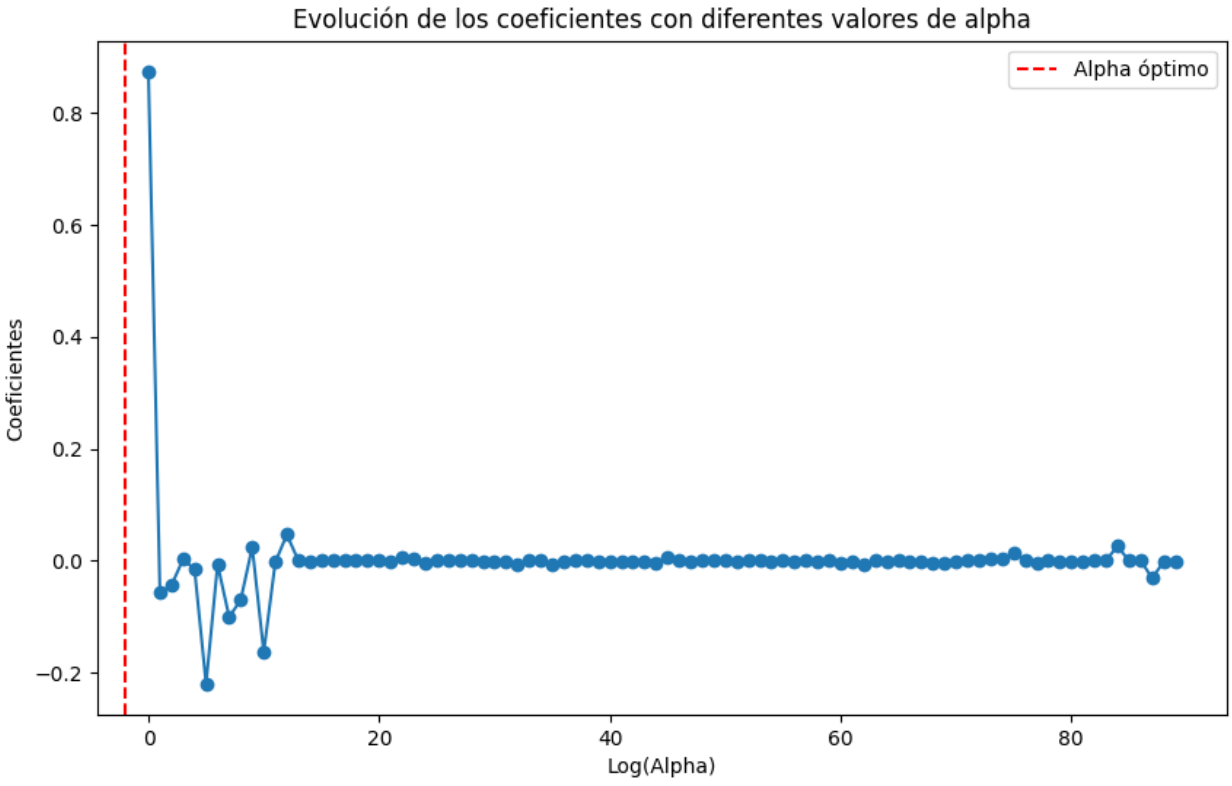
```python
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Predicciones con el modelo entrenado
y_pred = modelo_lasso.predict(X)

# Calcular RMSE
rmse = np.sqrt(mean_squared_error(Y, y_pred))
print(f'RMSE: {rmse}')

# Calcular R^2
r2 = r2_score(Y, y_pred)
print(f'R^2: {r2}')
```

```
RMSE: 9.548245793588348
R^2: 0.2369999767660692
```

RMSE : El Error Cuadrático Medio (Root Mean Squared Error) es una métrica común para evaluar modelos de regresión, ya que mide la magnitud del error.

R^2 : El coeficiente de determinación es otra métrica clave en regresión, que indica la proporción de la varianza en los datos que el modelo puede explicar. R^2 cercano a 1 indica un buen ajuste del modelo.

```
# Graficar los coeficientes
plt.figure(figsize=(10, 6))
plt.plot(modelo_lasso.coef_, marker='o', linestyle='-')
plt.xscale('linear')
plt.xlabel('Log(Alpha)')
plt.ylabel('Coeficientes')
plt.title('Evolución de los coeficientes con diferentes valores de
alpha')
plt.axvline(np.log10(best_alpha), color='r', linestyle='--',
label='Alpha óptimo')
plt.legend()
plt.show()
```



Con el conjunto de variables reducido, ajuste un modelo de regresión OLS e interprete su significancia y su R^2 ❑$_a$ d j.

```
import statsmodels.api as sm

# Crear el conjunto de datos reducido con las características seleccionadas
X_selected_train = X_train[:, selected_features]
X_selected_test = X_test[:, selected_features]

# Añadir una constante para el modelo de statsmodels (intercepto)
X_selected_train_const = sm.add_constant(X_selected_train)
X_selected_test_const = sm.add_constant(X_selected_test)
```

Ajustamos el modelo OLS

```
# Ajustar el modelo OLS con el conjunto de datos reducido

ols_model = sm.OLS(y_train, X_selected_train_const).fit()

# Resumen del modelo

print(ols_model.summary())
```

```
                            OLS Regression Results

==============================================================================
========
Dep. Variable:                          y   R-squared:
0.237
Model:                                OLS   Adj. R-squared:
0.237
Method:                     Least Squares   F-statistic:
1522.
Date:                   Thu, 14 Nov 2024   Prob (F-statistic):
0.00
Time:                           02:33:07   Log-Likelihood:                -
1.5155e+06
No. Observations:                412276   AIC:
3.031e+06
Df Residuals:                    412191   BIC:
3.032e+06
Df Model:                            84

Covariance Type:            nonrobust
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       1998.3950      0.015   1.34e+05      0.000    1998.366    1998.424
x1             5.2885      0.026    201.972      0.000       5.237       5.340
x2            -2.8991      0.023   -124.569      0.000      -2.945      -2.854
x3            -1.4864      0.023    -64.042      0.000      -1.532      -1.441
x4            -0.3392      0.018    -18.828      0.000      -0.375      -0.304
x5            -2.8517      0.032    -90.291      0.000      -2.914      -2.790
x6            -0.1050      0.021     -5.065      0.000      -0.146      -0.064
x7            -0.7953      0.026    -31.107      0.000      -0.845      -0.745
x8            -0.7536      0.021    -35.614      0.000      -0.795      -0.712
x9             0.1876      0.027      7.035      0.000       0.135       0.240
x10           -0.7071      0.026    -27.038      0.000      -0.758      -0.656
x11           -0.0159      0.018     -0.860      0.390      -0.052       0.020
x12            1.0356      0.020     51.323      0.000       0.996       1.075
x13            0.6158      0.024     25.221      0.000       0.568       0.664
x14           -0.5587      0.028    -19.928      0.000      -0.614      -0.504
x15            0.6458      0.039     16.388      0.000       0.569       0.723
x16            0.2110      0.023      9.182      0.000       0.166       0.256
x17            0.8868      0.043     20.530      0.000       0.802       0.971
x18            0.6105      0.030     20.248      0.000       0.551       0.670
x19            0.6560      0.034     19.297      0.000       0.589       0.723
x20            0.1294      0.032      4.073      0.000       0.067       0.192
```

```
x21             1.4353      0.043     33.502      0.000        1.351
1.519
x22             0.4424      0.028     15.791      0.000        0.388
0.497
x23            -0.4244      0.020    -21.692      0.000       -0.463
-0.386
x24             0.0501      0.019      2.629      0.009        0.013
0.087
x25             0.8650      0.029     29.978      0.000        0.808
0.922
x26             0.1244      0.019      6.562      0.000        0.087
0.161
x27             0.1345      0.021      6.265      0.000        0.092
0.177
x28            -0.0495      0.023     -2.193      0.028       -0.094
-0.005
x29            -0.1376      0.021     -6.514      0.000       -0.179
-0.096
x30            -0.1086      0.020     -5.481      0.000       -0.147
-0.070
x31            -0.3910      0.026    -15.094      0.000       -0.442
-0.340
x32             0.1327      0.021      6.240      0.000        0.091
0.174
x33             0.0696      0.025      2.838      0.005        0.022
0.118
x34            -0.5597      0.024    -23.483      0.000       -0.606
-0.513
x35            -0.1020      0.020     -5.058      0.000       -0.141
-0.062
x36             0.3145      0.021     15.150      0.000        0.274
0.355
x37             0.3731      0.021     17.780      0.000        0.332
0.414
x38            -0.3644      0.021    -17.636      0.000       -0.405
-0.324
x39            -0.2400      0.019    -12.376      0.000       -0.278
-0.202
x40            -0.1171      0.021     -5.503      0.000       -0.159
-0.075
x41            -0.1044      0.019     -5.354      0.000       -0.143
-0.066
x42            -0.0859      0.018     -4.722      0.000       -0.122
-0.050
x43            -0.1305      0.020     -6.584      0.000       -0.169
-0.092
x44             0.3523      0.025     14.068      0.000        0.303
0.401
x45             0.2225      0.018     12.526      0.000        0.188
```

```
0.257
x46            -0.5490      0.021    -25.805      0.000      -0.591
-0.507
x47             0.0533      0.024      2.267      0.023       0.007
0.099
x48             0.2432      0.019     12.983      0.000       0.207
0.280
x49             0.0140      0.021      0.666      0.506      -0.027
0.055
x50            -0.1143      0.022     -5.216      0.000      -0.157
-0.071
x51             0.1429      0.019      7.460      0.000       0.105
0.180
x52             0.0395      0.021      1.893      0.058      -0.001
0.080
x53             0.0293      0.021      1.421      0.155      -0.011
0.070
x54            -0.5867      0.021    -28.111      0.000      -0.628
-0.546
x55             0.5163      0.025     20.996      0.000       0.468
0.565
x56            -0.2155      0.021    -10.440      0.000      -0.256
-0.175
x57            -0.1863      0.021     -8.875      0.000      -0.227
-0.145
x58            -0.0994      0.017     -5.703      0.000      -0.133
-0.065
x59            -0.2614      0.025    -10.252      0.000      -0.311
-0.211
x60             0.3727      0.018     20.564      0.000       0.337
0.408
x61            -0.4567      0.023    -19.612      0.000      -0.502
-0.411
x62             0.0848      0.019      4.382      0.000       0.047
0.123
x63            -0.0339      0.022     -1.567      0.117      -0.076
0.009
x64            -0.0489      0.022     -2.238      0.025      -0.092
-0.006
x65            -0.4615      0.021    -22.053      0.000      -0.502
-0.420
x66            -0.1701      0.019     -9.005      0.000      -0.207
-0.133
x67            -0.2780      0.021    -13.486      0.000      -0.318
-0.238
x68             0.0545      0.020      2.742      0.006       0.016
0.093
x69             0.1186      0.024      5.011      0.000       0.072
0.165
```

```
x70             0.2565      0.018     14.275      0.000       0.221
0.292
x71             0.1821      0.018      9.999      0.000       0.146
0.218
x72             0.3983      0.021     18.869      0.000       0.357
0.440
x73             0.0448      0.020      2.293      0.022       0.007
0.083
x74            -0.6090      0.021    -28.883      0.000      -0.650
-0.568
x75            -0.0218      0.019     -1.162      0.245      -0.059
0.015
x76            -0.0779      0.020     -3.936      0.000      -0.117
-0.039
x77             0.1730      0.022      8.006      0.000       0.131
0.215
x78             0.1079      0.018      6.079      0.000       0.073
0.143
x79             0.4237      0.019     22.486      0.000       0.387
0.461
x80             0.0023      0.019      0.123      0.902      -0.034
0.039
x81             0.1961      0.018     10.616      0.000       0.160
0.232
x82            -0.4079      0.022    -18.920      0.000      -0.450
-0.366
x83            -0.2534      0.017    -14.938      0.000      -0.287
-0.220
x84            -0.0412      0.018     -2.257      0.024      -0.077
-0.005
==============================================================================
Omnibus:                   115558.847   Durbin-Watson:
1.998
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
375725.568
Skew:                          -1.426   Prob(JB):
0.00
Kurtosis:                       6.707   Cond. No.
10.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

Interpretación de su significancia y R^2 adj

R^2 **= 0.237**, este valor es relativamente bajo, lo que sugiere que las características que se han utilizado en el modelo no explican bien la variabilidad de la variable dependiente. Esto nos puede indicar que el modelo es poco adecuado para predecir la variable o que la relación entre las variables explicativas y la variable dependiente es débil.

.

R^2 **ajustado** = 0.237, el R^2 ajustado es igual a R^2 dado a que el número de variables en el modelo no ha aumentado lo suficiente como para que el R^2 ajustado penalice de manera significativa el número de características. El R^2 ajustado ajusta el valor de R^2 penalizando la inclusión de variables poco importantes, y en este caso no hay una gran diferencia entre los dos, lo que podría ser una señal de que las variables seleccionadas están aportando algo de valor.

Revise los supuestos de los errores, y con los hallazgos del ítem anterior, concluya sobre la conveniencia de usar este modelo para predecir el año de grabación de la canción.

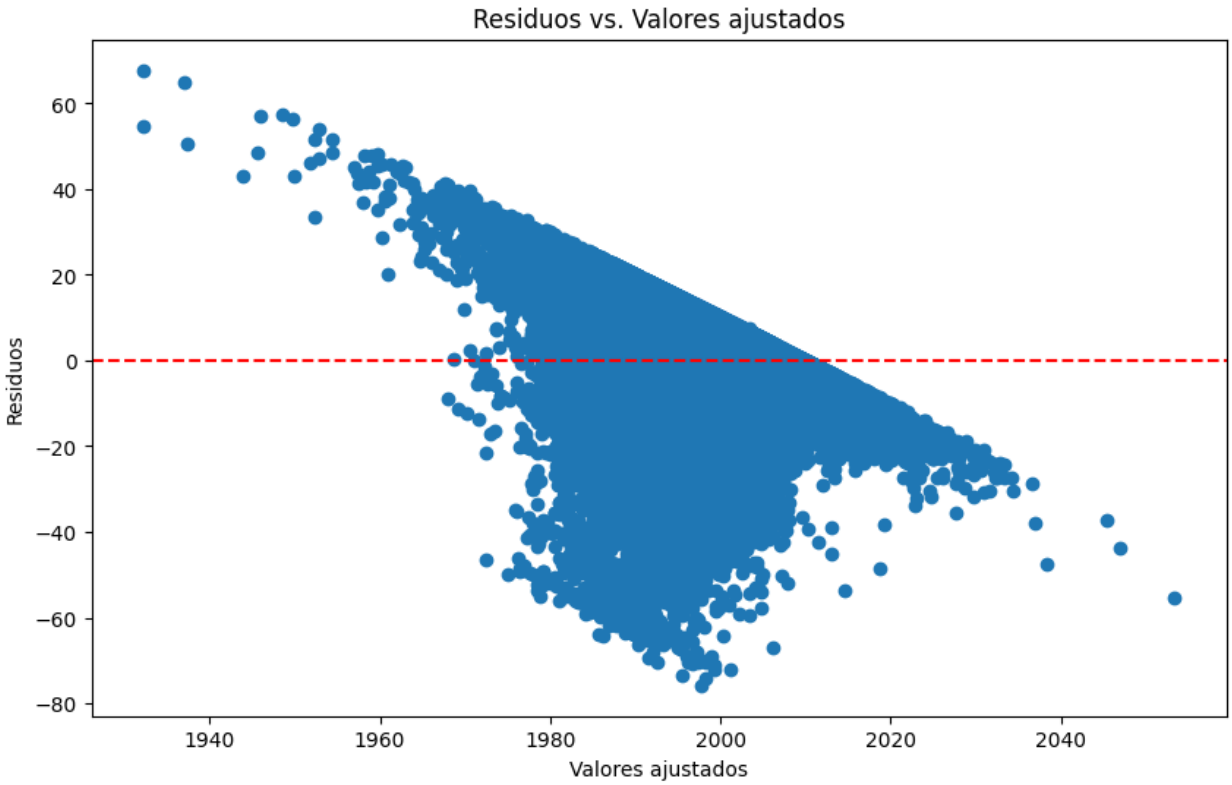
Los supuestos de los errores en la regresión OLS son condiciones necesarias para que las estimaciones del modelo sean válidas y para que las inferencias estadísticas (como los valores p de los coeficientes) sean confiables.

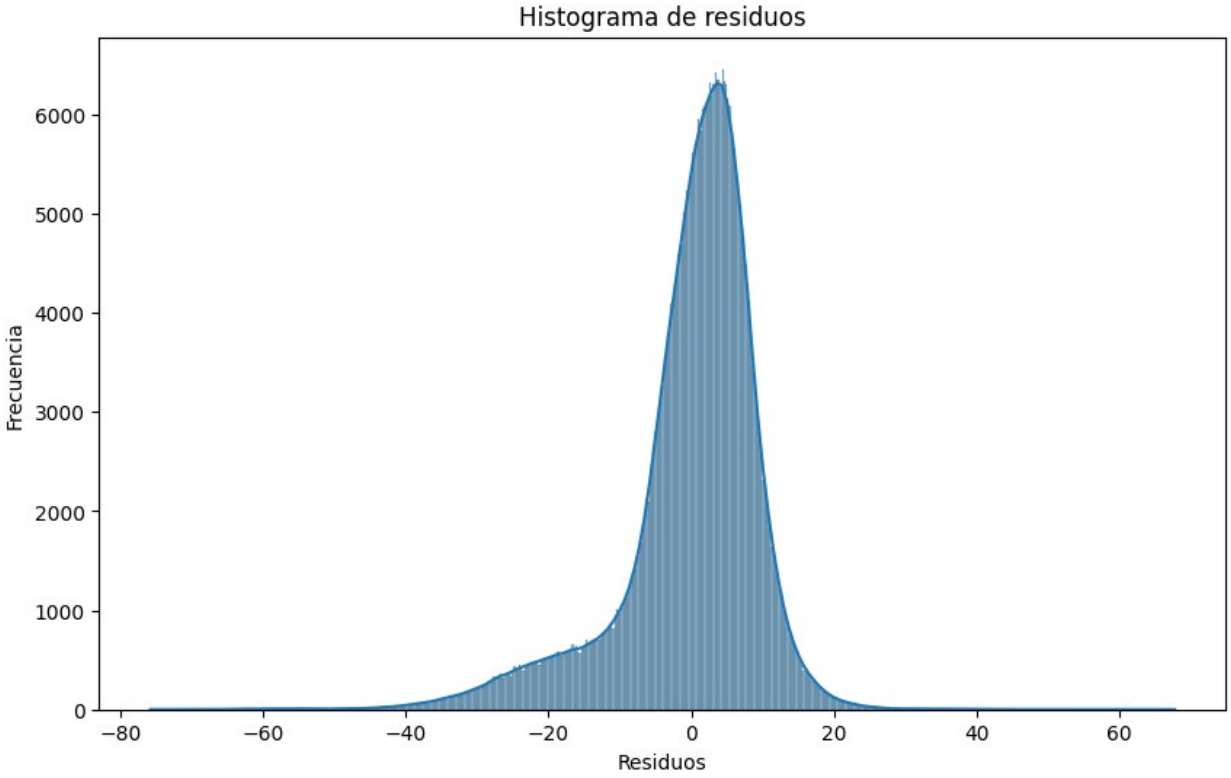
Supuestos de los errores en la regresión OLS

Linealidad: Si observamos que los residuos no siguen una distribución aleatoria (por ejemplo, si hay patrones no lineales), este supuesto podría no cumplirse. Esto indicaría que el modelo no está capturando adecuadamente las relaciones entre las variables y que podría ser necesario un modelo más complejo (por ejemplo, regresión polinómica o un modelo no lineal).

```
import scipy.stats as stats

# Residuos del modelo OLS
residuos = ols_model.resid

# 1. Gráfico de residuos vs. valores ajustados
fitted_values = ols_model.fittedvalues

plt.figure(figsize=(10, 6))
plt.scatter(fitted_values, residuos)
plt.axhline(0, color='red', linestyle='--')
plt.title('Residuos vs. Valores ajustados')
plt.xlabel('Valores ajustados')
plt.ylabel('Residuos')
plt.show()
```

```
# 2. Histograma de residuos
plt.figure(figsize=(10, 6))
sns.histplot(residuos, kde=True)
plt.title('Histograma de residuos')
plt.xlabel('Residuos')
plt.ylabel('Frecuencia')
plt.show()

# 3. Gráfico Q-Q de residuos
plt.figure(figsize=(10, 6))
stats.probplot(residuos, dist="norm", plot=plt)
plt.title('Gráfico Q-Q de residuos')
plt.show()
```




Histograma de residuos
Frecuencia
Residuos
0
1000
2000
3000
4000
5000
6000
−80
−60
−40
−20
0
20
40
60

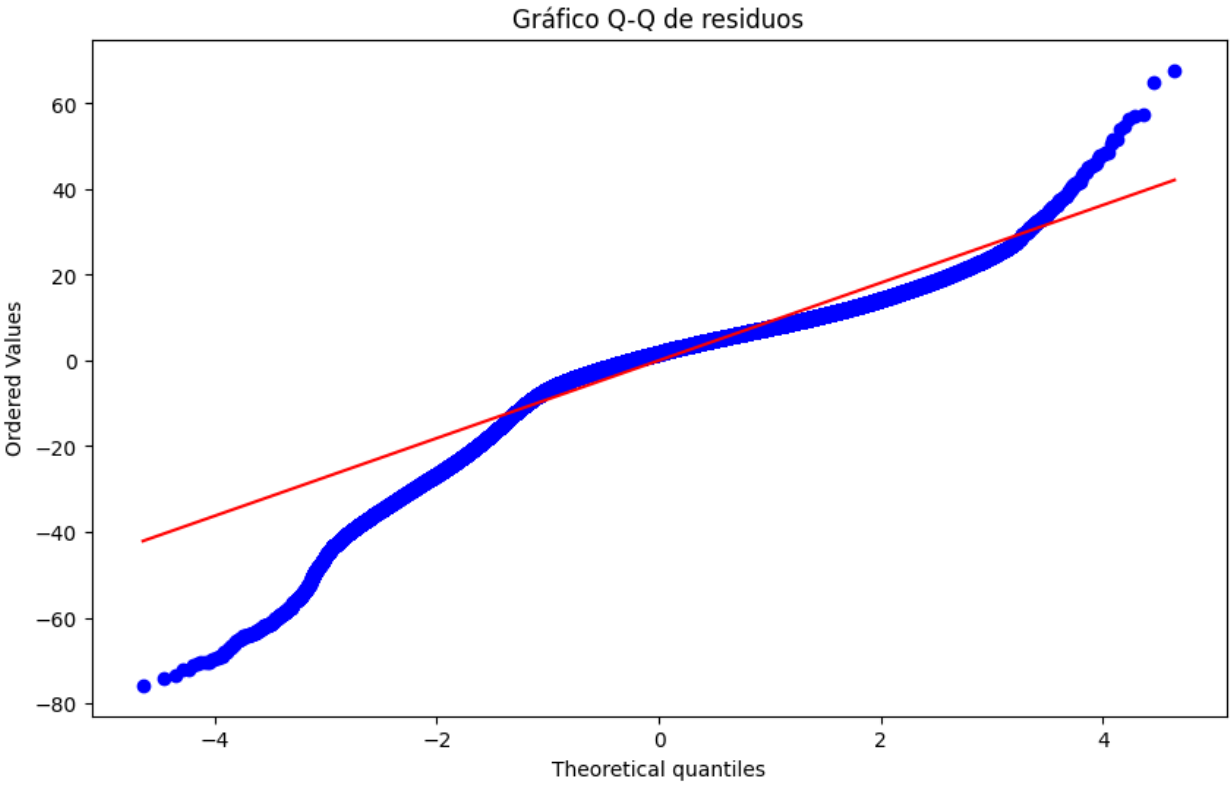


Gráfico Q-Q de residuos
Ordered Values
Theoretical quantiles
−80
−60
−40
−20
0
20
40
60
−4
−2
0
2
4

CONCLUSIONES

Al validar el gráfico Q-Q podemos observar que los puntos no siguen una linea recta lo cual indica que los residuos no siguen una distribución normal, teniendo en cuenta ese resultado es necesario realizar una transformación sobre los datos o considerar otro tipo de modelo.

El resultado de significancia y R^2 muestra valores relativamente bajos, lo que sugiere que las características que se han utilizado en el modelo no explican bien la variabilidad de la variable dependiente. Esto nos puede indicar que el modelo es poco adecuado para predecir la variable o que la relación entre las variables explicativas y la variable dependiente es débil.

Es necesario usar otras técnicas de regularización que nos puedan dar un mejor resultado.

Se debe verificar si es útil probar otros modelos que nos arrojen mejores resultados, como modelos de árbol de decisión o redes neuronales, que pueden manejar mejor relaciones no lineales y problemas de multicolinealidad.

LASSO asume que hay una relación lineal entre las variables independientes y la variable dependiente. Si las relaciones son no lineales, LASSO puede no capturar correctamente estas relaciones, ya que se basa en una regresión lineal por lo cual es una buena opción aplicar otros modelos, por ejemplo ElasticNet.

Maestría en Inteligencia Artificial Aplicada

Aprendizaje Automatico 1

Examen 3

Integrantes

- Yesid Castelblanco
- Andres Felipe Borrero
- Carlos Alberto Martinez Ramirez
- Nicolas Colmenares

Profesores

- Santiago Ortiz
- Henry Velasco

Notas:

- Todas las respuestas, gráficas, tablas y operaciones deben ser debidamente justificadas.
- La información que sea obtenida de alguna fuente debe ser citada y referenciada en el documento a entregar.

```
# Importar librería pandas y numpy
# Pandas es una librería de Python que proporciona estructuras de
datos y herramientas de análisis de datos de alto rendimiento
import pandas as pd
# NumPy es una librería de Python que proporciona estructuras de datos
y operaciones matemáticas de alto rendimiento.
import numpy as np
#Matplotlib.pyplot es una librería de Python que proporciona
herramientas de visualización y gráficos de alta calidad.
import matplotlib.pyplot as plt
from IPython.display import Math, Latex
from IPython.core.display import Image
#Seaborn es una librería de Python que proporciona herramientas de
visualización y análisis de datos de alta calidad.
import seaborn as sns

from sklearn.model_selection import train_test_split


#métricas
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import accuracy_score
from tqdm.auto import tqdm
import time

#Modelos

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import ElasticNet

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_validate

#sns.set(color_codes=True) es un método de la librería Seaborn que
establece los códigos de color predeterminados para los gráficos y
diagramas. Cuando se establece color_codes=True, Seaborn utiliza una
paleta de colores predefinida y asigna un código de color a cada
categoría o variable en los gráficos y diagramas.
sns.set(color_codes=True)
#Cuando se establece rc={'figure.figsize':(10,6)}, Seaborn crea
figuras con un tamaño de 10 unidades de ancho y 6 unidades de alto.
sns.set(rc={'figure.figsize':(10,6)})
import warnings
warnings.filterwarnings('ignore')
```

Ejercicio 3

3) El conjunto de datos conocido como *"California Housing Dataset"* puede ser cargado del paquete **sklearn**. La variable objetivo es el valor medio de la vivienda para los distritos de California, expresado en cientos de miles de dólares ($100000). Este conjunto de datos se derivó del censo de EE.UU. de 1990, usando como unidad de censo el grupo de bloques. Un grupo de bloques es la unidad geográfica más pequeña para la que La Oficina del Censo de EE.UU. publica datos de muestra (un grupo de bloque generalmente tiene una población de 600 a 3000 personas).

Un hogar es un grupo de personas que residen dentro de una casa. Dado que el promedio. El número de habitaciones y dormitorios en este conjunto de datos se proporciona por hogar, estas

columnas pueden tomar valores sorprendentemente grandes para grupos de bloques con pocos hogares y muchas casas vacías, como centros vacacionales.

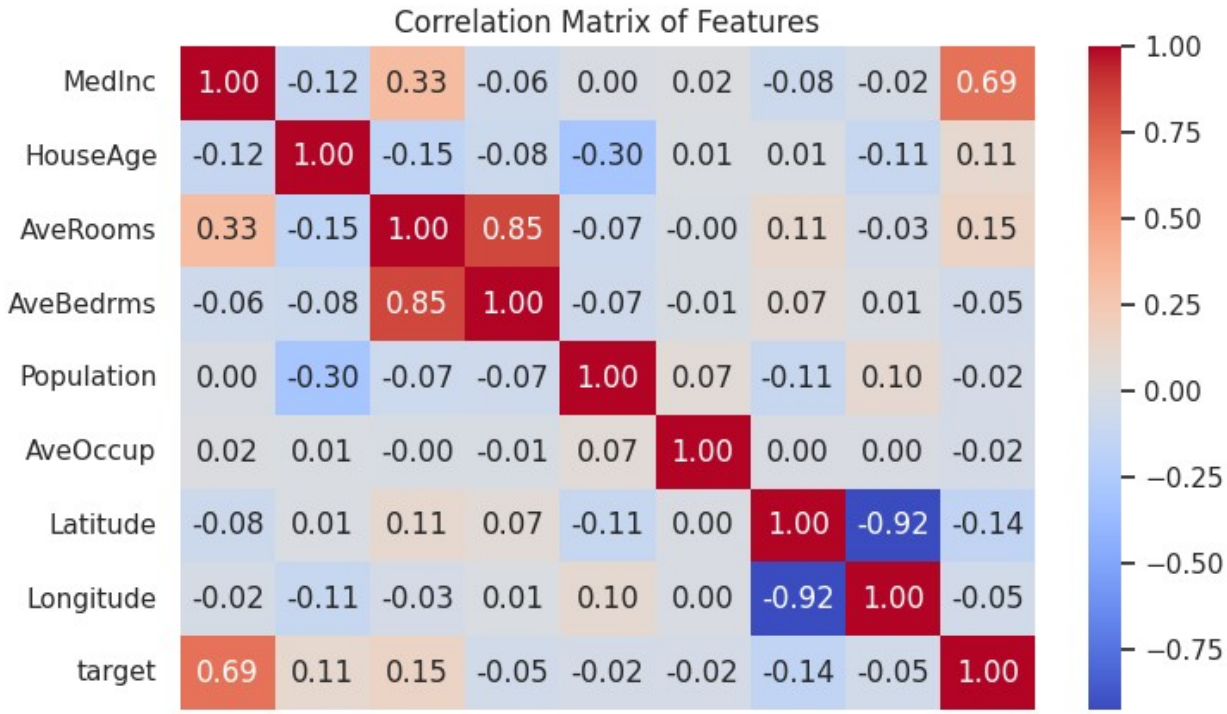
- Lea el conjunto de datos usando la función **fetch_california_housing** del paquete **sklearn.datasets**, guardelos en una variable llamada **california_housing** y con el comando `print(california_housing.DESCR)` observe la descripción general del dataset y en especial qué es cada una de las variables de entrada.

- Separe las variables explicativas **X** de la variable respuesta **Y**, para acceder a ellas use los comandos `california_housing.data` y `california_housing.target`. Considere la conveniencia de incluir las variables Longitud y Latitud al modelo. Haga un análisis exploratorio de las correlaciones entre las variables y comente al respecto.
- Ajuste un modelo de regresión **Elastic-Net** con un coeficiente de penalización pequeño, iterativamente ajuste este valor para eliminar variables explicativas y corregir el problema de multicolinealidad, en cada iteración calcule las correlaciones de las variables explicativas y pare cuando no se encuentren correlaciones altas.

```
from sklearn.datasets import fetch_california_housing
california_housing = fetch_california_housing()
print(california_housing.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

:Number of Instances: 20640

:Number of Attributes: 8 numeric, predictive attributes and the target

:Attribute Information:
    - MedInc        median income in block group
    - HouseAge      median house age in block group
    - AveRooms      average number of rooms per household
    - AveBedrms     average number of bedrooms per household
    - Population    block group population
    - AveOccup      average number of household members
    - Latitude      block group latitude
    - Longitude     block group longitude

:Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California
districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per
census
block group. A block group is the smallest geographical unit for which
the U.S.
Census Bureau publishes sample data (a block group typically has a
population
of 600 to 3,000 people).
```

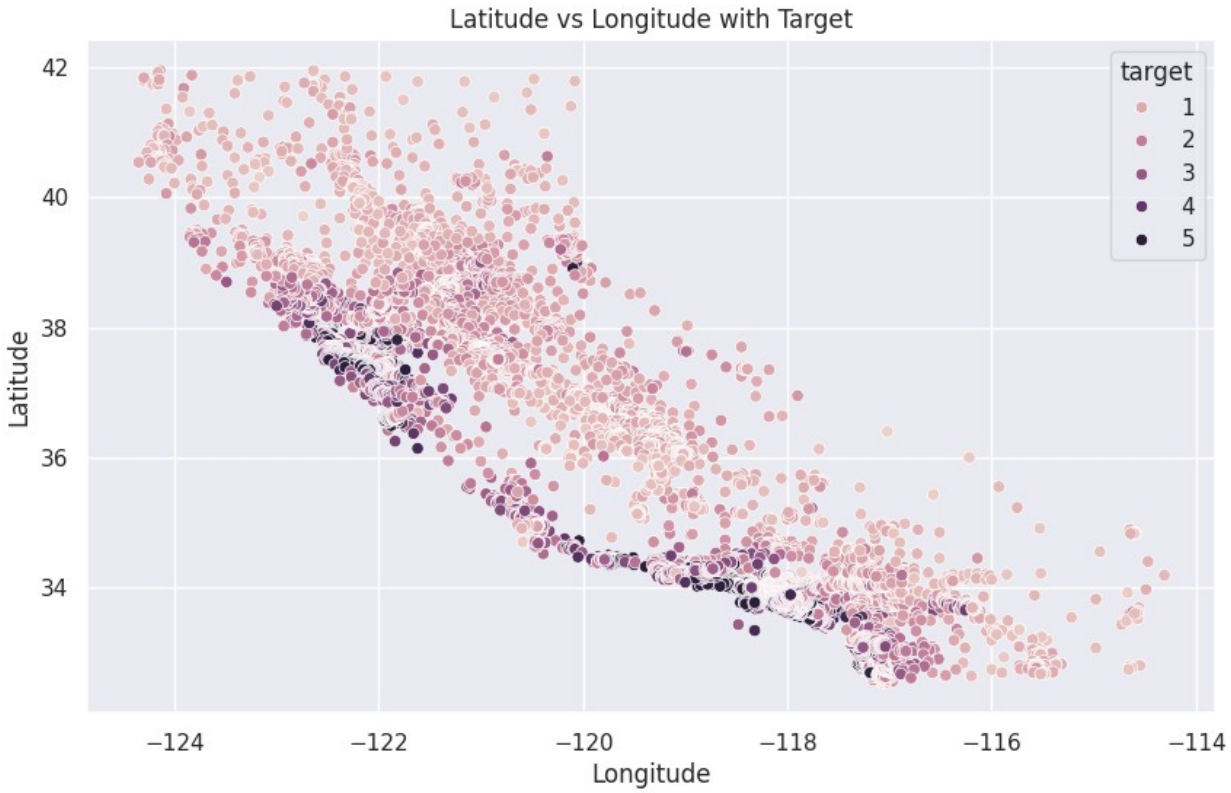
```
A household is a group of people residing within a home. Since the
average
number of rooms and bedrooms in this dataset are provided per
household, these
columns may take surprisingly large values for block groups with few
households
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. rubric:: References

- Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
  Statistics and Probability Letters, 33 (1997) 291-297


X = california_housing.data
Y = california_housing.target

df = pd.DataFrame(X, columns=california_housing.feature_names)
df['target'] = Y

# Calculate the correlation matrix
correlation_matrix = df.corr()

# Create the heatmap
plt.figure(figsize=(8, 5))
ax = sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
ax.set(xlabel="",ylabel="")
ax.xaxis.set_visible(False)

plt.title('Correlation Matrix of Features')

plt.show()
```



En la matriz de correlación, se observa una fuerte correlación entre las variables "AveRooms" y "AveBedrms", así como entre la latitud y la longitud. Entre las demás variables, la correlación es leve o nula.

La correlación entre las variables explicativas y la variable objetivo es débil para todas, excepto para la variable **MedInc**.

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Longitude', y='Latitude', hue='target')
plt.title('Latitude vs Longitude with Target')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



Considerando el caso de estudio en el que la ubicación de una vivienda influye en su valor, y basándonos en el gráfico anterior que muestra que ciertas regiones presentan valores similares, se decidió incluir la longitud y la latitud en el modelo.

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)

# Función para calcular la correlación entre variables explicativas
def calculate_correlations(X):
    correlation_threshold = 0.8  # Umbral para alta correlación
    corr_matrix = pd.DataFrame(X).corr().abs()
    return
np.any(corr_matrix.values[np.triu_indices_from(corr_matrix.values, 1)]
> correlation_threshold)

df_x_train = pd.DataFrame(X_train,
columns=california_housing.feature_names)

# Parámetros iniciales
alpha = 0.01   # Penalización inicial pequeña
l1_ratio = 0.5  # Proporción entre lasso y ridge en Elastic-Net

# Ajuste iterativo del modelo
while True:
    print(f"Iniciando el modelo con una penalización de {alpha}")
    # Definir y ajustar el modelo Elastic-Net
```

```
    model = ElasticNet(alpha=alpha, l1_ratio=l1_ratio, max_iter=10000,
random_state=42)
    model.fit(X_train, y_train)

    # Filtrar variables con coeficientes no nulos (las que quedan en
el modelo)
    non_zero_coefficients = model.coef_ != 0
    X_train_filtered = df_x_train.iloc[:, non_zero_coefficients]

    # Calcular la correlación entre variables explicativas restantes
    if not calculate_correlations(X_train_filtered):
        print("No hay correlaciones altas entre las variables
explicativas. Proceso terminado.")
        break

    # Aumentar la penalización y continuar
    alpha += 0.01
    print(f"Aumentando penalización a {alpha}")

print(f"Número de variables restantes: {X_train_filtered.shape[1]}")
print(f"Columnas restantes: {X_train_filtered.columns.values}")
print(f"alpha optimo: {alpha:.2f}")
```

```
Número de variables restantes: 5
Columnas restantes: ['MedInc' 'HouseAge' 'Population' 'AveOccup'
'Latitude']
alpha optimo: 0.27
```

Maestría en Inteligencia Artificial Aplicada

Aprendizaje Automatico 1

Examen 3

Integrantes

- Andres Felipe Borrero
- Yesid Castelblanco
- Nicolas Colmenares
- Carlos Alberto Martinez

Profesores

- Santiago Ortiz
- Henry Velasco

Notas:

- Todas las respuestas, gráficas, tablas y operaciones deben ser debidamente justificadas.
- La información que sea obtenida de alguna fuente debe ser citada y referenciada en el documento a entregar.

Ejercicio 4

4) El fichero de datos *"Dengue_Data.xlsx"* contiene información epidemiológica de los casos de Dengue en el Departamento de Antioquia. Estos datos contienen tanto información socio-económica como clínica de las personas que resultaron infectadas y desarrollaron Dengue o Dengue Hemorrágico. Para una completa descripción de los datos y/o fenómeno estudiado, remítase al siguiente artículo *Identification of Hazard and Socio-Demographic Patterns of Dengue Infections in a Colombian Subtropical Region from 2015 to 2020: Cox Regression Models and Statistical Analysis.

*Realizar.

- Utilizando solo las variables socio-demográficas, ajuste un modelo Logit y los modelos Logit-Ridge, Logit-LASSO y Logit-Enet (con sus parámetros óptimos, por supuesto) para predecir si una persona va a desarrollar "DENGUE" o "DENGUE GRAVE". Interprete los resultados de cada modelo y compárelos; defina que variables son las más importantes par predecir el estado categórico modelado, muestre los gráficos de penalidad y de evolución de coeficientes. * Concluya sobre el fenómeno estudiado y a información del artículo.
- Realice el mismo ejercicio anterior, solo que ahora considere como variables explicativas las variables de tipo clínico/médico para modelar si una persona requiere o no ser

hospitali- zada. Realice los mismos análisis y procedimientos. Concluya en función de la información presentada en el artículo.

```
# Tratamiento de datos
#
==============================================================================
========
# Pandas es una librería de Python que proporciona estructuras de
datos y herramientas de análisis de datos de alto rendimiento
import pandas as pd
# NumPy es una librería de Python que proporciona estructuras de datos
y operaciones matemáticas de alto rendimiento.
import numpy as np

# Gráficos
#
==============================================================================
========
#Matplotlib.pyplot es una librería de Python que proporciona
herramientas de visualización y gráficos de alta calidad.
import matplotlib.pyplot as plt
from matplotlib import style
from IPython.display import Math, Latex
from IPython.core.display import Image
#Seaborn es una librería de Python que proporciona herramientas de
visualización y análisis de datos de alta calidad.
import seaborn as sns

# Métricas
#
==============================================================================
========
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import precision_score, recall_score,
auc,roc_curve, f1_score

from tqdm.auto import tqdm
import time

# Preprocesado y modelado
#
==============================================================================
========
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score

# Configuración seaborn
#
================================================================================
#sns.set(color_codes=True) es un método de la librería Seaborn que
establece los códigos de color predeterminados para los gráficos y
diagramas. Cuando se establece color_codes=True, Seaborn utiliza una
paleta de colores predefinida y asigna un código de color a cada
categoría o variable en los gráficos y diagramas.
sns.set(color_codes=True)
#Cuando se establece rc={'figure.figsize':(10,6)}, Seaborn crea
figuras con un tamaño de 10 unidades de ancho y 6 unidades de alto.
sns.set(rc={'figure.figsize':(10,6)})

# Configuración warnings
#
================================================================================
import warnings
warnings.filterwarnings('ignore')

# Configuración matplotlib
#
================================================================================
plt.rcParams['image.cmap'] = "bwr"
#plt.rcParams['figure.dpi'] = "100"
plt.rcParams['savefig.bbox'] = "tight"
style.use('ggplot') or plt.style.use('ggplot')
```

#Datos Contiene información epidemiológica de los casos de Dengue en el Departamento de Antioquia. Estos datos contienen tanto información socio-económica como clínica de las personas que resultaron infectadas y desarrollaron Dengue o Dengue Hemorrágico

```
url =
'https://github.com/cam2149/MachinelearningI/raw/19a9b08e636dd80915a47
27c674a04c64a962b59/Dengue_Data.xlsx'
dfDengue = pd.read_excel(url, sheet_name='datos_2')

dfDengue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50397 entries, 0 to 50396
Data columns (total 48 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   edad_                            50397 non-null  int64
 1   Grupos edad                      50397 non-null  object
 2   sexo_                            50397 non-null  object
 3   area_                            50397 non-null  object
 4   area_.1                          50397 non-null  object
 5   area_.2                          50397 non-null  object
 6   ocupacion_                       50397 non-null  object
 7   per_etn_                         50397 non-null  object
 8   gp_discapa                       49661 non-null  object
 9   gp_desplaz                       49676 non-null  object
 10  gp_migrant                       49696 non-null  object
 11  gp_carcela                       49681 non-null  object
 12  gp_gestan                        24841 non-null  object
 13  gp_indigen                       49691 non-null  object
 14  gp_pobicbf                       49688 non-null  object
 15  gp_mad_com                       24815 non-null  object
 16  gp_desmovi                       49695 non-null  object
 17  gp_vic_vio                       49701 non-null  object
 18  gp_otros                         50377 non-null  object
 19  fec_con_                         50386 non-null  datetime64[ns]
 20  ini_sin_                         50384 non-null  datetime64[ns]
 21  tpo_consulta(consulta-inicio)    50384 non-null  float64
 22  pac_hos_                         50397 non-null  object
 23  fec_hos_                         14960 non-null  datetime64[ns]
 24  tpo_deterioro(hosp-consul)       14959 non-null  float64
 25  fiebre                           50388 non-null  object
 26  cefalea                          50388 non-null  object
 27  dolrretroo                       50388 non-null  object
 28  malgias                          50388 non-null  object
 29  artralgia                        50388 non-null  object
 30  erupcionr                        50388 non-null  object
 31  dolor_abdo                       50384 non-null  object
 32  vomito                           50384 non-null  object
 33  diarrea                          50384 non-null  object
 34  somnolenci                       49643 non-null  object
 35  hipotensio                       50384 non-null  object
 36  hepatomeg                        50384 non-null  object
 37  hem_mucosa                       49645 non-null  object
 38  hipotermia                       49646 non-null  object
 39  caida_plaq                       49648 non-null  object
 40  acum_liqui                       49647 non-null  object
 41  aum_hemato                       49645 non-null  object
 42  extravasac                       289 non-null    object
 43  hemorr_hem                       289 non-null    object
 44  choque                           1103 non-null   object
```

```
 45  daño_organ                    289 non-null    object
 46  nom_eve                       50397 non-null  object
 47  Region                        50397 non-null  object
dtypes: datetime64[ns](3), float64(2), int64(1), object(42)
memory usage: 18.5+ MB
```

```
##Ajuste de los valores de los encabezados
newHeaders = 'Edad', 'GrupoEdad', 'Genero', 'Area', 'Area1',
'Area2','Ocupacion', 'Etnia', 'Discapacidad', 'Desplazados',
'Inmigrantes',      'Prisionero', 'Embarazada', 'Indigena', 'Pobreza',
'Gestante','Desmovilizado', 'VictimaViolencia', 'Otros',
'FechaConsulta', 'InicioSintomas','TipoConsulta', 'Hospitalizado',
'FechaHospitalizacion','TiempodeDeterioro', 'Fiebre', 'DolorCabeza',
'DolorRetroOrbitario',                'DolorMuscular',
'DolorArticular', 'ErupcionCutanea', 'DolorAbdominal', 'Vomitos',
'Diarrea','Somnolencia', 'Hipotension', 'Hepatomegalia', 'HemMucosa',
'Hipotermia','CaidaPlaquetas', 'AcumulacionLiquido',
'AumentoHematocritos', 'Extravasacion', 'Hemorroides','Choque',
'DanioOrgano', 'Evento', 'Region'

dfDengue.columns = newHeaders
```

#Análisis exploratorio de los datos.

```
dfDengue.describe()
```


```
{"summary":"{\n  \"name\": \"dfDengue\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"Edad\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
17804.809772904533,\n        \"min\": 1.0,\n        \"max\": 50397.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
29.609857729626764,\n          42.0,\n          50397.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"FechaConsulta\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"1970-01-01 00:00:00.000050386\",\n        \"max\": \"2021-02-09
00:00:00\",\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          \"50386\",\n          \"2017-01-25 06:06:35.252649728\",\
n          \"2017-07-19 00:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"InicioSintomas\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"1970-01-01 00:00:00.000050384\",\n        \"max\": \"2021-01-02
00:00:00\",\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          \"50384\",\n          \"2017-01-20 20:32:11.470307840\",\
n          \"2017-07-15 00:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TipoConsulta\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

```
17793.75950747473,\n        \"min\": 0.0,\n        \"max\": 50384.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
4.410348523340743,\n          5.0,\n          50384.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"FechaHospitalizacion\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"1970-01-01 00:00:00.000014960\",\n        \"max\": \"2021-01-11
00:00:00\",\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          \"14960\",\n          \"2017-03-31 07:13:03.529411584\",\
n          \"2018-08-06 00:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TiempodeDeterioro\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5270.042531406434,\n        \"min\": 0.0,\n        \"max\": 14959.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
5.136840697907614,\n          6.0,\n          14959.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
dfDengue.head(3)
```

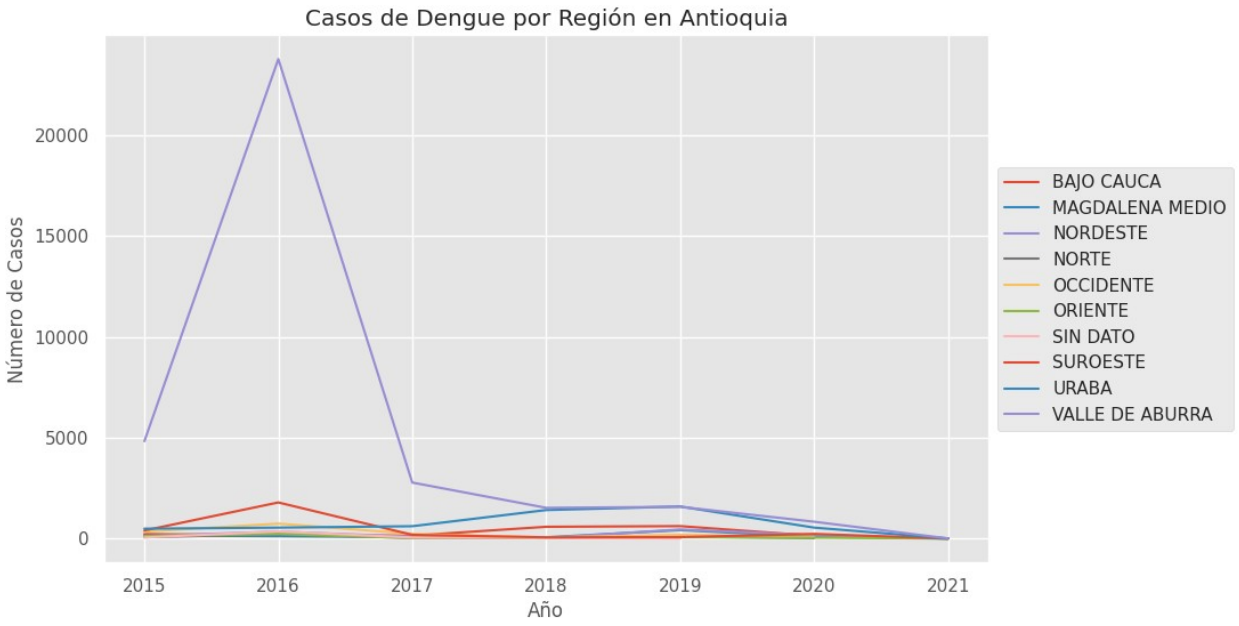

```
{"type":"dataframe","variable_name":"dfDengue"}
```


```
dfDengue.tail()
```

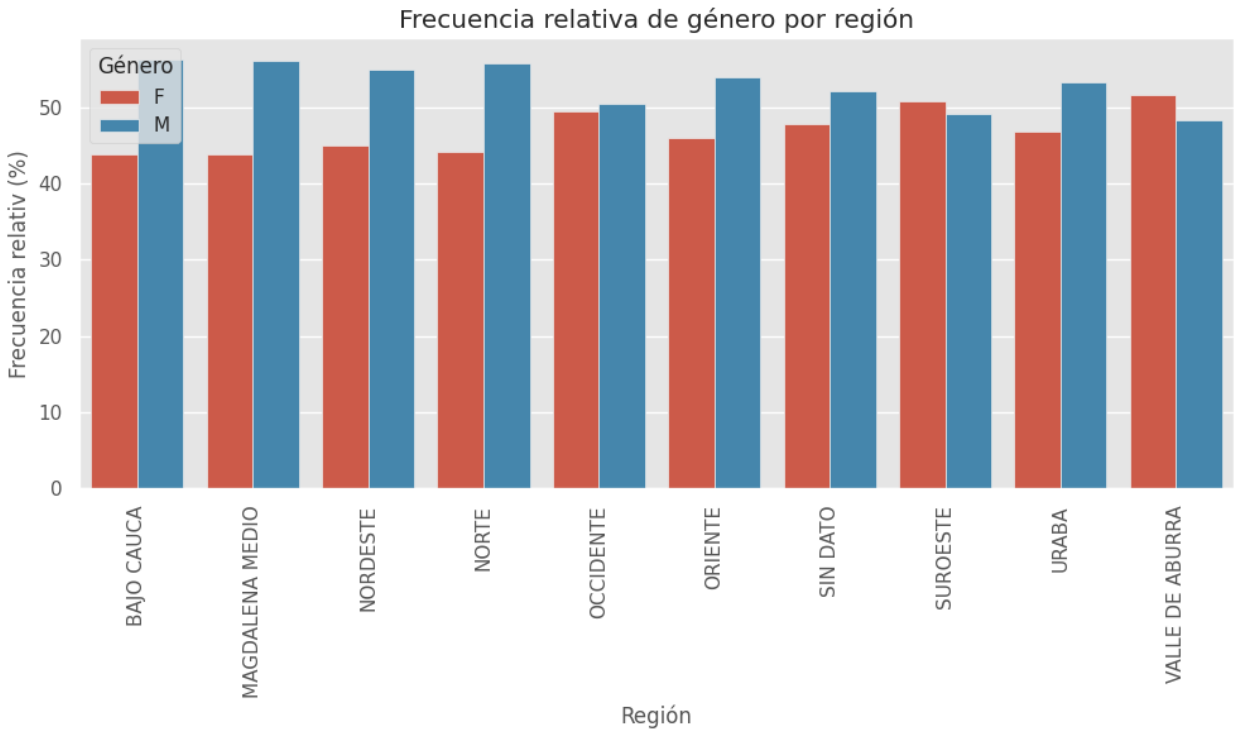

```
{"type":"dataframe"}
```


```
# Grafico de los casos registrados por region en todos los años por
region
casos_por_region_y_año = dfDengue.groupby(['Region',
dfDengue['FechaConsulta'].dt.year])['Region'].count().unstack()

# Crear una figura y ejes
fig, ax = plt.subplots(figsize=(10, 6))

# Iterar sobre las regiones y crear un gráfico de líneas para cada una
for region in casos_por_region_y_año.index:
  ax.plot(casos_por_region_y_año.columns,
casos_por_region_y_año.loc[region], label=region)

# Ajustar la leyenda para que las regiones
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels, loc='center left', bbox_to_anchor=(1, 0.5),
ncol=1)

# Configurar el título y las etiquetas de los ejes
ax.set_title('Casos de Dengue por Región en Antioquia')
ax.set_xlabel('Año')
ax.set_ylabel('Número de Casos')
```

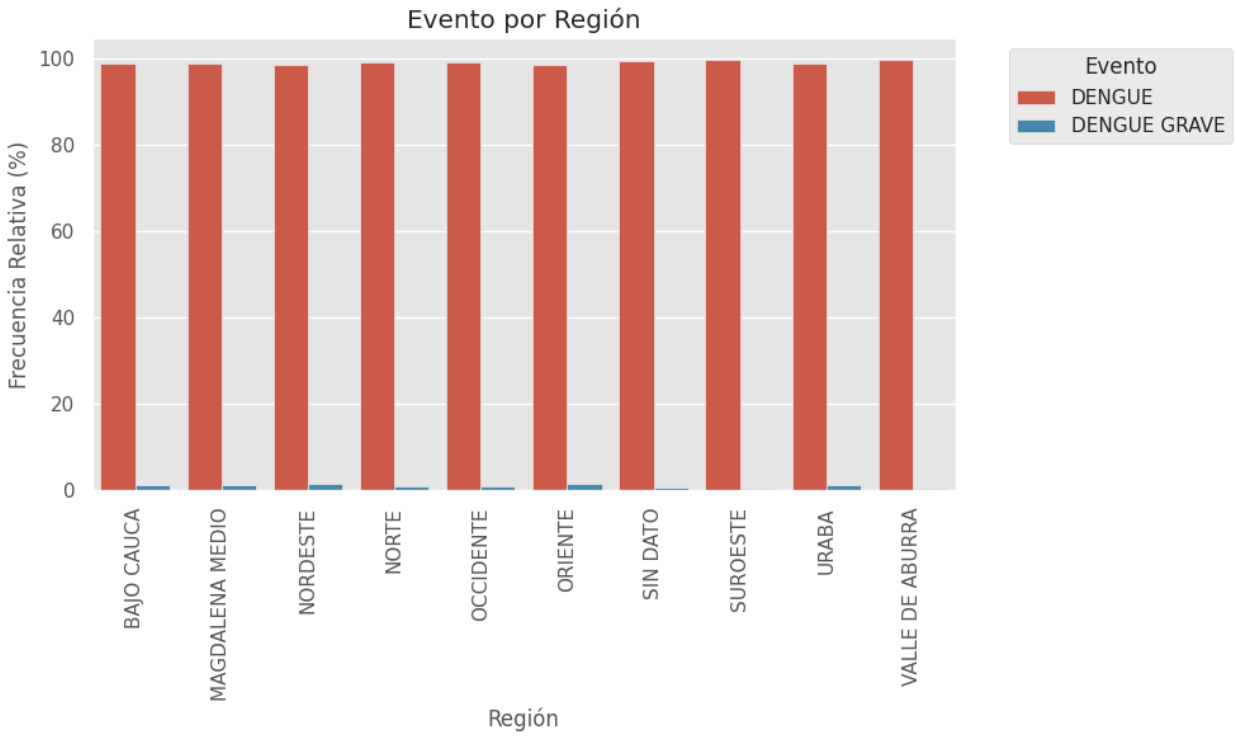
```
# Mostrar el gráfico
plt.show()
```




```
# Grafico de la frecencia relatica de los casos de denge registrados agrupados por Región y genero
genero_freq_by_region = dfDengue.groupby(['Region', 'Genero'])['Genero'].count() / dfDengue.groupby('Region')['Genero'].count() * 100
genero_freq_by_region = genero_freq_by_region.reset_index(name='Relative Frequency')
# Crear una figura y ejes
sns.barplot(x='Region', y='Relative Frequency', hue='Genero', data=genero_freq_by_region)
# Ajustar la leyenda para que las regiones
plt.title('Frecuencia relativa de género por región')
plt.xlabel('Región')
plt.ylabel('Frecuencia relativ (%)')
plt.xticks(rotation=90)
plt.legend(title='Género')
plt.tight_layout()
# Mostrar el gráfico
plt.show()
```




```
# Grafico de la frecencia relativa de los casos de Denge registrados
agrupados por Región
nombre_evento_freq_by_region = dfDengue.groupby(['Region', 'Evento'])
['Evento'].count() / dfDengue.groupby('Region')['Evento'].count() *
100
nombre_evento_freq_by_region =
nombre_evento_freq_by_region.reset_index(name='Relative Frequency')
# Crear una figura y ejes
sns.barplot(x='Region', y='Relative Frequency', hue='Evento',
data=nombre_evento_freq_by_region)
# Ajustar la leyenda para que las regiones
plt.title('Evento por Región')
plt.xlabel('Región')
plt.ylabel('Frecuencia Relativa (%)')
plt.xticks(rotation=90)
plt.legend(title='Evento', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
# Mostrar el gráfico
plt.show()
```



#Limpieza de Datos

```
dfDengue.isna().sum()
```

```
Edad                    0
GrupoEdad               0
Genero                  0
Area                    0
Area1                   0
Area2                   0
Ocupacion               0
Etnia                   0
Discapacidad          736
Desplazados           721
Inmigrantes           701
Prisionero            716
Embarazada          25556
Indigena              706
Pobreza               709
Gestante            25582
Desmovilizado         702
VictimaViolencia      696
Otros                  20
FechaConsulta          11
InicioSintomas         13
TipoConsulta           13
Hospitalizado           0
```

```
FechaHospitalizacion    35437
TiempodeDeterioro       35438
Fiebre                      9
DolorCabeza                 9
DolorRetroOrbitario         9
DolorMuscular               9
DolorArticular              9
ErupcionCutanea             9
DolorAbdominal             13
Vomitos                    13
Diarrea                    13
Somnolencia               754
Hipotension                13
Hepatomegalia              13
HemMucosa                 752
Hipotermia                751
CaidaPlaquetas            749
AcumulacionLiquido        750
AumentoHematocritos       752
Extravasacion           50108
Hemorroides             50108
Choque                  49294
DanioOrgano             50108
Evento                      0
Region                      0
dtype: int64
```

En el conjunto de datos se indetifican una gran cantidad de datos nulos Discapacidad 736, Desplazados 721, Inmigrantes 701, Prisionero 716, Embarazada 25556, Indigena 706, Pobreza 709, Gestante 25582, Desmovilizado 702, VictimaViolencia 696, Otros 20, FechaConsulta 11, InicioSintomas 13, TipoConsulta 13, FechaHospitalizacion 35437, TiempodeDeterioro 35438, Fiebre 9, DolorCabeza 9, DolorRetroOrbitario 9, DolorMuscular 9, DolorArticular 9, ErupcionCutanea 9, DolorAbdominal 13, Vomitos 13, Diarrea 13, Somnolencia 754, Hipotension 13, Hepatomegalia 13, HemMucosa 752, Hipotermia 751, CaidaPlaquetas 749, AcumulacionLiquido 750, AumentoHematocritos 752, Extravasacion 50108, Hemorroides 50108, Choque 49294, DanioOrgano 50108

- Eliminar filas que todos sus datos son nulos
- Se realizara el replace correspondiente por valor 0, a las se identifican como respuesta [Si o No] o [Sí o No]
- La columna Evento se hará el reemplazo {"DENGUE": 1, "DENGUE GRAVE": 2}

- La Columna Genero se hará el reemplazo {"M": 1, "F": 0}

```
# Eliminar filas con datos nulos
countCollumns = len(dfDengue)
dfDengue.dropna(how='all', inplace=True)

# Describir el efecto de la eliminación
```

```
print("Número de filas antes de eliminar datos nulos:", countCollumns)
print("Número de filas después de eliminar datos nulos:",
len(dfDengue))
print("Número de filas eliminadas:", len(dfDengue) - countCollumns)
```

```
Número de filas antes de eliminar datos nulos: 50397
Número de filas después de eliminar datos nulos: 50397
Número de filas eliminadas: 0
```

```
# Eliminar columnas con una cantidad significativa de valores nulos,
más del 50%
threshold = 0.5  # Umbral para la proporción de valores nulos
countCollumns = len(dfDengue.columns) # Cantidad de columnas del
dataset
dfDengue.dropna(thresh=len(dfDengue) * threshold, axis=1,
inplace=True)

# Imprimir la cantidad de columnas eliminadas
print(f"Se eliminaron {len(dfDengue.columns) - countCollumns} columnas
con más del {threshold * 100}% de valores nulos.")

# Mostrar las columnas restantes
print("Columnas restantes:", dfDengue.columns.tolist())
```

```
Se eliminaron -8 columnas con más del 50.0% de valores nulos.
Columnas restantes: ['Edad', 'GrupoEdad', 'Genero', 'Area', 'Area1',
'Area2', 'Ocupacion', 'Etnia', 'Discapacidad', 'Desplazados',
'Inmigrantes', 'Prisionero', 'Indigena', 'Pobreza', 'Desmovilizado',
'VictimaViolencia', 'Otros', 'FechaConsulta', 'InicioSintomas',
'TipoConsulta', 'Hospitalizado', 'Fiebre', 'DolorCabeza',
'DolorRetroOrbitario', 'DolorMuscular', 'DolorArticular',
'ErupcionCutanea', 'DolorAbdominal', 'Vomitos', 'Diarrea',
'Somnolencia', 'Hipotension', 'Hepatomegalia', 'HemMucosa',
'Hipotermia', 'CaidaPlaquetas', 'AcumulacionLiquido',
'AumentoHematocritos', 'Evento', 'Region']
```

```
# Eliminar columnas de fechas no significativas para la prediccion de
la variable objetivo Evento
columns_to_drop = ['FechaConsulta', 'InicioSintomas']
dfDengue.drop(columns = columns_to_drop, inplace=True)
# Mostrar las columnas restantes
print("Columnas restantes:", dfDengue.columns.tolist())
```

```
Columnas restantes: ['Edad', 'GrupoEdad', 'Genero', 'Area', 'Area1',
'Area2', 'Ocupacion', 'Etnia', 'Discapacidad', 'Desplazados',
'Inmigrantes', 'Prisionero', 'Indigena', 'Pobreza', 'Desmovilizado',
'VictimaViolencia', 'Otros', 'TipoConsulta', 'Hospitalizado',
'Fiebre', 'DolorCabeza', 'DolorRetroOrbitario', 'DolorMuscular',
'DolorArticular', 'ErupcionCutanea', 'DolorAbdominal', 'Vomitos',
'Diarrea', 'Somnolencia', 'Hipotension', 'Hepatomegalia', 'HemMucosa',
```

```
'Hipotermia', 'CaidaPlaquetas', 'AcumulacionLiquido',
'AumentoHematocritos', 'Evento', 'Region']

# Recorrer todos los campos y si su valor es NaN convertir en un 0, si
el en valor es No en 0 o si el valor es Si en 1
excluded_columns = ['Evento', 'Genero']
for column in dfDengue.columns:
  if column not in excluded_columns:
    # Reemplazar valoes NaN con 0
    dfDengue[column].fillna(0, inplace=True)
    # Reemplazar "No" con 0 y "Si" con 1
    dfDengue[column] = dfDengue[column].replace({"No": 0, "Si": 1,
"Sí" : 1})

# Reemplazar "DENGUE" con 1 and "DENGUE GRAVE" con 2
dfDengue['Evento'] = dfDengue['Evento'].replace({"DENGUE": 0, "DENGUE
GRAVE": 1})
dfDengue['Genero'] = dfDengue['Genero'].replace({'M': 1, 'F': 0})

for column in dfDengue.columns:
  if pd.api.types.is_numeric_dtype(dfDengue[column]):
    # Convertir la columna a tipo int si es posible
    try:
      dfDengue[column] = dfDengue[column].astype(int)
    except ValueError:
      # Si no se puede convertir a int, mantener el tipo original
      pass

print(dfDengue.isna().sum())
```

```
Edad                  0
GrupoEdad             0
Genero                0
Area                  0
Area1                 0
Area2                 0
Ocupacion             0
Etnia                 0
Discapacidad          0
Desplazados           0
Inmigrantes           0
Prisionero            0
Indigena              0
Pobreza               0
Desmovilizado         0
VictimaViolencia      0
Otros                 0
TipoConsulta          0
Hospitalizado         0
Fiebre                0
```

```
DolorCabeza             0
DolorRetroOrbitario     0
DolorMuscular           0
DolorArticular          0
ErupcionCutanea         0
DolorAbdominal          0
Vomitos                 0
Diarrea                 0
Somnolencia             0
Hipotension             0
Hepatomegalia           0
HemMucosa               0
Hipotermia              0
CaidaPlaquetas          0
AcumulacionLiquido      0
AumentoHematocritos     0
Evento                  0
Region                  0
dtype: int64
```

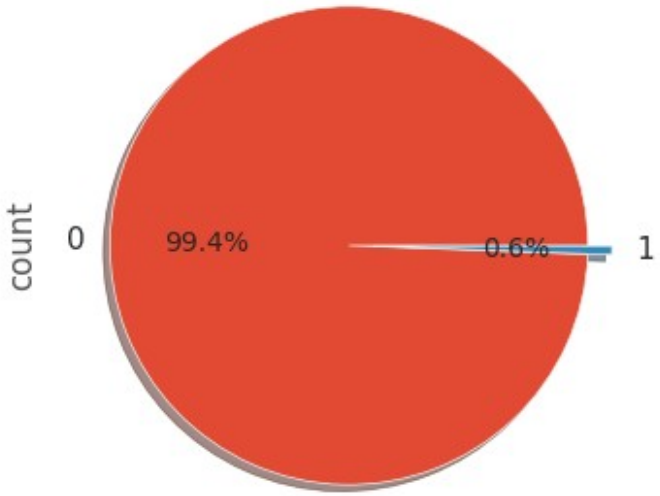
```
dfDengue.describe()
```

```
{"type":"dataframe"}
```

```
unique_nombre_evento_counts = dfDengue['Evento'].value_counts()
print(unique_nombre_evento_counts)
```

```
Evento
0    50101
1      296
Name: count, dtype: int64
```

```
plt.figure(figsize=[4,4])
dfDengue['Evento'].value_counts().plot.pie(explode=[0,0.1],autopct='%1
.1f%%',shadow=True)
plt.show()
```




```
# Se seleccionan las columnas a estandarizar
columns_to_standardize = ['GrupoEdad', 'Area', 'Area1', 'Area2',
'Ocupacion', 'Etnia', 'Region']
#scaler = StandardScaler()

# Convertir valores de cadena en representaciones numéricas usando
codificación de etiquetas
# Esto es necesario antes de escalar porque StandardScaler trabaja con
datos numéricos.
for column in columns_to_standardize:
  if dfDengue[column].dtype == 'object':
    unique_values = dfDengue[column].unique()
    value_mapping = {value: index for index, value in
enumerate(unique_values)}

    # Reemplazar valores de cadena con etiquetas numéricas
    dfDengue[column] = dfDengue[column].map(value_mapping)

# Se ajusta y transforma las columnas seleccionadas.
#dfDengue[columns_to_standardize] =
scaler.fit_transform(dfDengue[columns_to_standardize])

# Calculo de la matriz de correlación.
#correlation_matrix = dfDengue.corr()
#(correlation_matrix)
```

#Modelos *Socio*-demográficas

###Filtrando las variables socio-demográficas

Para poder evaluar la capacidad predictiva de cada modelo, se dividen las observaciones disponibles en dos grupos: uno de entrenamiento (70%) y otro de test (30%).

```
dfDengue.info()
```

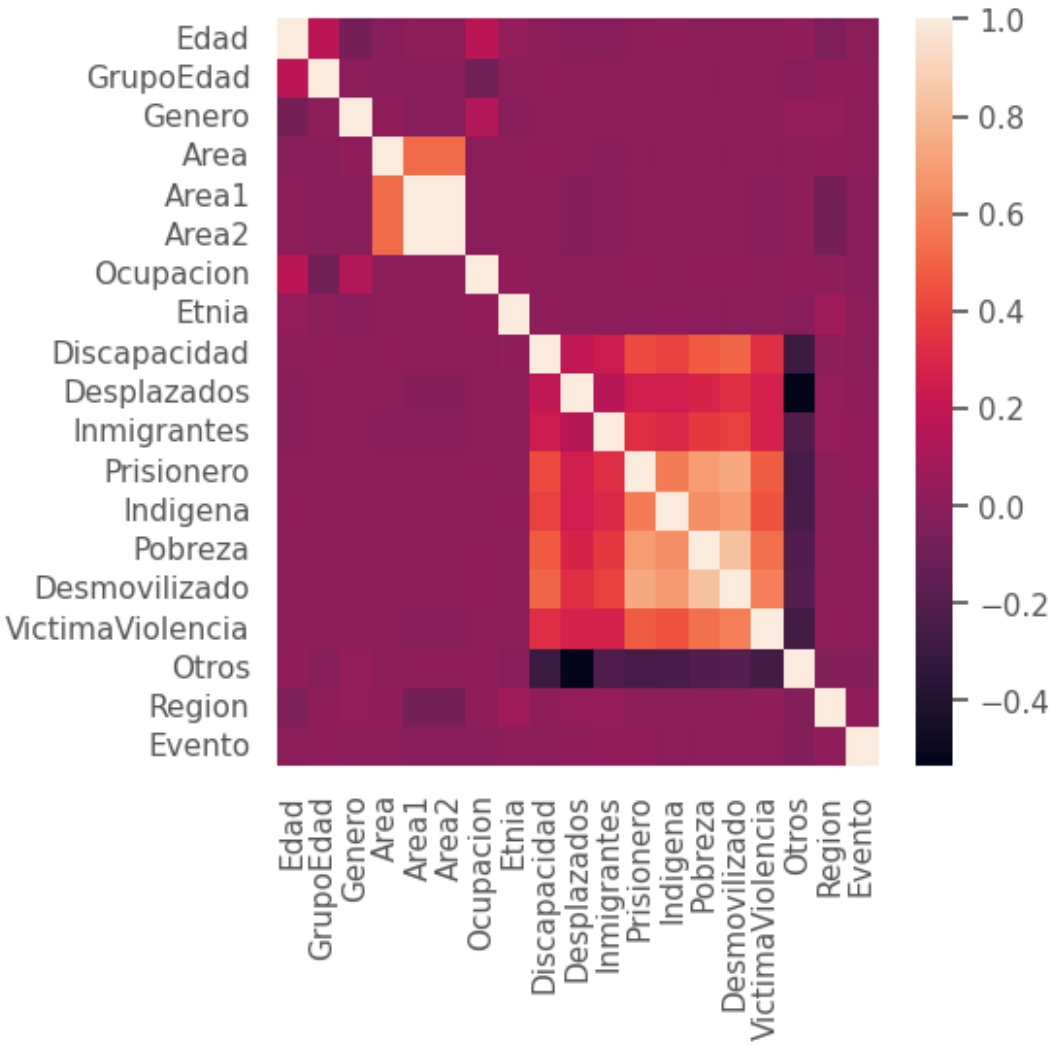
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50397 entries, 0 to 50396
Data columns (total 38 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Edad                  50397 non-null  int64
 1   GrupoEdad             50397 non-null  int64
 2   Genero                50397 non-null  int64
 3   Area                  50397 non-null  int64
 4   Area1                 50397 non-null  int64
 5   Area2                 50397 non-null  int64
 6   Ocupacion             50397 non-null  int64
 7   Etnia                 50397 non-null  int64
 8   Discapacidad          50397 non-null  int64
 9   Desplazados           50397 non-null  int64
 10  Inmigrantes           50397 non-null  int64
 11  Prisionero            50397 non-null  int64
 12  Indigena              50397 non-null  int64
 13  Pobreza               50397 non-null  int64
 14  Desmovilizado         50397 non-null  int64
 15  VictimaViolencia      50397 non-null  int64
 16  Otros                 50397 non-null  int64
 17  TipoConsulta          50397 non-null  int64
 18  Hospitalizado         50397 non-null  int64
 19  Fiebre                50397 non-null  int64
 20  DolorCabeza           50397 non-null  int64
 21  DolorRetroOrbitario   50397 non-null  int64
 22  DolorMuscular         50397 non-null  int64
 23  DolorArticular        50397 non-null  int64
 24  ErupcionCutanea       50397 non-null  int64
 25  DolorAbdominal        50397 non-null  int64
 26  Vomitos               50397 non-null  int64
 27  Diarrea               50397 non-null  int64
 28  Somnolencia           50397 non-null  int64
 29  Hipotension           50397 non-null  int64
 30  Hepatomegalia         50397 non-null  int64
 31  HemMucosa             50397 non-null  int64
 32  Hipotermia            50397 non-null  int64
 33  CaidaPlaquetas        50397 non-null  int64
 34  AcumulacionLiquido    50397 non-null  int64
 35  AumentoHematocritos   50397 non-null  int64
 36  Evento                50397 non-null  int64
 37  Region                50397 non-null  int64
dtypes: int64(38)
memory usage: 14.6 MB
```

```
print("Columnas", dfDengue.columns.tolist())
```

```
Columnas ['Edad', 'GrupoEdad', 'Genero', 'Area', 'Area1', 'Area2',
'Ocupacion', 'Etnia', 'Discapacidad', 'Desplazados', 'Inmigrantes',
'Prisionero', 'Indigena', 'Pobreza', 'Desmovilizado',
'VictimaViolencia', 'Otros', 'TipoConsulta', 'Hospitalizado',
'Fiebre', 'DolorCabeza', 'DolorRetroOrbitario', 'DolorMuscular',
'DolorArticular', 'ErupcionCutanea', 'DolorAbdominal', 'Vomitos',
'Diarrea', 'Somnolencia', 'Hipotension', 'Hepatomegalia', 'HemMucosa',
'Hipotermia', 'CaidaPlaquetas', 'AcumulacionLiquido',
'AumentoHematocritos', 'Evento', 'Region']
```

```
socio_demo_vars = ['Edad', 'GrupoEdad', 'Genero', 'Area', 'Area1',
'Area2',
                   'Ocupacion', 'Etnia', 'Discapacidad', 'Desplazados',
                   'Inmigrantes', 'Prisionero', 'Indigena', 'Pobreza',
                   'Desmovilizado', 'VictimaViolencia', 'Otros',
'Region',
                   'Evento']

df_socio_demo = dfDengue[socio_demo_vars]

correlation_matrix = df_socio_demo.corr()

fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (5, 5))
sns.heatmap(correlation_matrix, ax = ax)
```

```
<Axes: >
```




```
dfDengueSocioDemo = dfDengue[socio_demo_vars].copy()

# Definir variables predictoras (X) y variable objetivo (y)
X = dfDengueSocioDemo.drop('Evento', axis=1)
y = (dfDengueSocioDemo['Evento']).astype(int)
y = y.values.ravel()

# Escalar las variables predictoras
#scaler = StandardScaler()
#X_scaled = scaler.fit_transform(X)

from sklearn.preprocessing import MinMaxScaler

# Dividir los datos en conjuntos de entrenamiento y prueba (opcional,
para evaluar el modelo)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=0.3, random_state=42)

#scaler = MinMaxScaler()  # Instancia de la clase MinMaxScaler
#X_train = scaler.fit_transform(X_train)
#X_test = scaler.transform(X_test)

# Porcentaje de los valores unicos de , y_train
unique_y_train_counts =
pd.Series(y_train).value_counts(normalize=True) * 100
unique_y_train_counts
```

```
0    99.410381
1     0.589619
Name: proportion, dtype: float64
```

##Modelo LogisticRegression sin regularización

```
#LogisticRegression
logitreg_model = LogisticRegression(penalty=None)
logitreg_model.fit(X_train, y_train)
logitreg_y_pred = logitreg_model.predict(X_test)

print("Usaremos estas métricas como comparación de referencia para
cualquier mejora que obtengamos con la regularizacion Logit-Ridge,
Logit-LASSO y Logit-Enet")
print("==========================================================.")
logitreg_model_precision_score = precision_score(y_test,
logitreg_y_pred, average='macro')
print("Precisión del Modelo LogisticRegression con el set de Pruebas:
{:.4f}".format(logitreg_model_precision_score))
logitreg_model_recall_score = recall_score(y_test, logitreg_y_pred,
average='macro')
print('Recall del Modelo LogisticRegression con el set de Pruebas:
{:.4f}'.format(logitreg_model_recall_score))
logitreg_model_f1_score = f1_score(y_test, logitreg_y_pred,
average='macro')
print('F1 Score del Modelo LogisticRegression con el set de Pruebas:
{:.4f}'.format(logitreg_model_f1_score))
print("==========================================================.")
scores =
cross_val_score(logitreg_model,X_train,y_train,cv=5,scoring='roc_auc',
n_jobs=-1)
logitreg_model_roc_auc = scores.mean()
print("AUC Cross Val Score del set de entrenamiento:
{:.4f}".format(logitreg_model_roc_auc))
acc =
cross_val_score(logitreg_model,X_train,y_train,cv=5,scoring='accuracy'
,n_jobs=-1)
logitreg_model_accv = acc.mean()
print("Accuracy Cross Val Score del Modelo LogisticRegression para el
```

```
set de entrenamiento: {:.4f}".format(logitreg_model_accv))
logitreg_model_score_train = logitreg_model.score(X_train, y_train)
print("Accuracy del Modelo LogisticRegression del set de entrenamiento
set: {:.4f}".format(logitreg_model_score_train))
logitreg_model_score_test = logitreg_model.score(X_test, y_test)
print("Accuracy del Modelo LogisticRegression del set de pruebas:
{:.4f}".format(logitreg_model_score_test))
# Error de test del modelo
#
==========================================================================
========
logitreg_model_rmse_ols = mean_squared_error(y_test, logitreg_y_pred,
squared = False )
print("El error (rmse) de test es:
{:.4f}".format(logitreg_model_rmse_ols))
```

```
Usaremos estas métricas como comparación de referencia para cualquier
mejora que obtengamos con la regularizacion Logit-Ridge, Logit-LASSO y
Logit-Enet
=========================================================.
Precisión del Modelo LogisticRegression con el set de Pruebas: 0.4971
Recall del Modelo LogisticRegression con el set de Pruebas: 0.5000
F1 Score del Modelo LogisticRegression con el set de Pruebas: 0.4985
=========================================================.
AUC Cross Val Score del set de entrenamiento: 0.5569
Accuracy Cross Val Score del Modelo LogisticRegression para el set de
entrenamiento: 0.9941
Accuracy del Modelo LogisticRegression del set de entrenamiento set:
0.9941
Accuracy del Modelo LogisticRegression del set de pruebas: 0.9942
El error (rmse) de test es: 0.0763
```

```
residuals_logitreg_model = y_test - logitreg_y_pred
print("Media de los resideales:", np.mean(residuals_logitreg_model))
print("Desviación estándar de los residuales:",
np.std(residuals_logitreg_model))
```

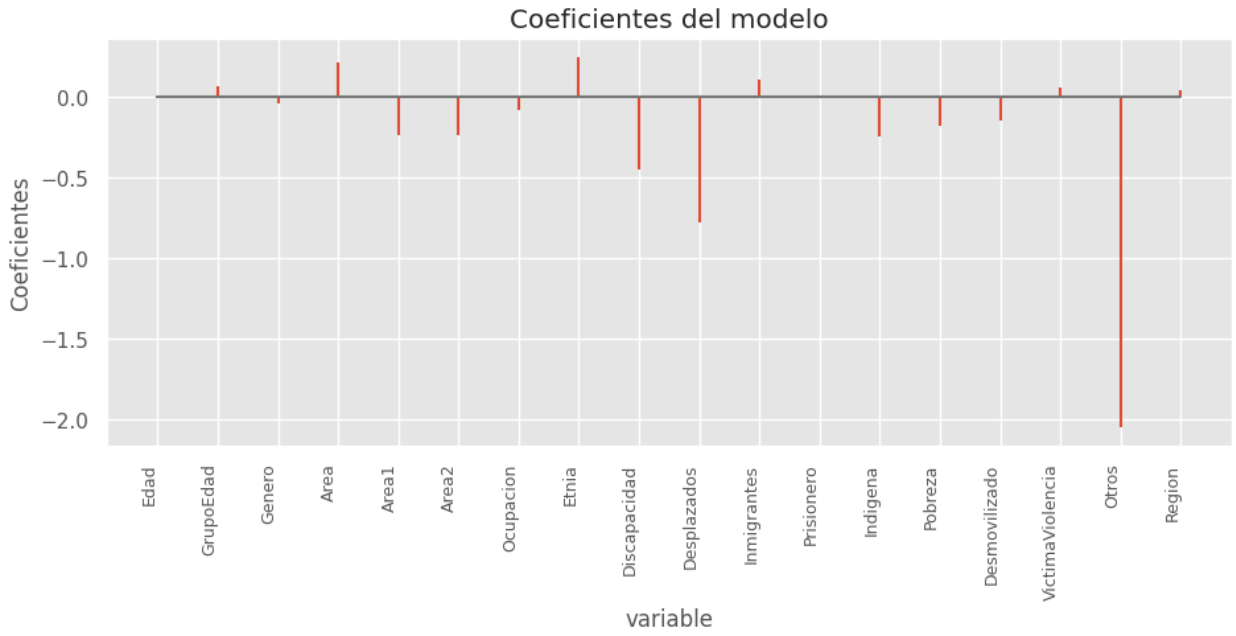
```
Media de los resideales: 0.00582010582010582
Desviación estándar de los residuales: 0.07606728724194514
```

Las predicciones del modelo final se alejan en promedio 0.07606 unidades del valor real.

```
# Coeficientes del modelo
#
==========================================================================
========
coefficients = logitreg_model.coef_
intercept = logitreg_model.intercept_
print("")
print("Coeficientes:", coefficients)
```

```
print("Intercepto:", intercept)
print("Coeficiente:", list(zip(X.columns,
logitreg_model.coef_.flatten(), )))
print("")
dfDenge_coeficientes = pd.DataFrame({'predictor': X.columns, 'coef':
logitreg_model.coef_[0]})

fig, ax = plt.subplots(figsize=(11, 4))
ax.stem(dfDenge_coeficientes.predictor, dfDenge_coeficientes.coef,
markerfmt=' ')
plt.xticks(rotation=90, ha='right', size=9)
ax.set_xlabel('variable')
ax.set_ylabel('Coeficientes')
ax.set_title('Coeficientes del modelo');
```

```
Coeficientes: [[-1.54510525e-03  6.54530628e-02 -4.09205583e-02
2.20197015e-01
  -2.35360049e-01 -2.35360049e-01 -7.60621429e-02  2.54159682e-01
  -4.43823838e-01 -7.74711066e-01  1.11342924e-01  1.05945993e-03
  -2.39990210e-01 -1.75328665e-01 -1.41207491e-01  6.37135302e-02
  -2.04477180e+00  4.43297000e-02]]
Intercepto: [-3.29017292]
Coeficiente: [('Edad', -0.0015451052468487803), ('GrupoEdad',
0.0654530627822511), ('Genero', -0.04092055828169914), ('Area',
0.22019701489119795), ('Area1', -0.235360048768328), ('Area2', -
0.235360048768328), ('Ocupacion', -0.07606214290418802), ('Etnia',
0.25415968153293556), ('Discapacidad', -0.4438238376477244),
('Desplazados', -0.7747110656182455), ('Inmigrantes',
0.11134292404479129), ('Prisionero', 0.0010594599269353076),
('Indigena', -0.23999020955650044), ('Pobreza', -0.1753286648850158),
('Desmovilizado', -0.14120749109131336), ('VictimaViolencia',
0.06371353022397562), ('Otros', -2.044771799225836), ('Region',
0.04432969998261621)]
```



##Modelo LogisticRegression con regularización Logit-Lasso

```
lasso_model = LogisticRegression(random_state=42,
                                 C = 0.1, ### parametro de
penalización, valores menores a uno penalizacion alta
                                 class_weight= 'balanced',
                                 penalty= 'l1', ## Elgir como
penalizar: l1, l2, elasticnet, None
                                 solver= 'liblinear' ### Algoritmo de
optimización
                                 )
lasso_model.fit(X_train, y_train) #This line is added to train the
model
y_pred_lasso_model = lasso_model.predict(X_test)

print("=========================================================.")
lasso_model_precision_score = precision_score(y_test,
y_pred_lasso_model, average='macro')
print("Precisión del Modelo LogisticRegression con LASSO el set de
Pruebas: {:.4f}".format(lasso_model_precision_score))
lasso_model_recall_score = recall_score(y_test, y_pred_lasso_model,
average='macro')
print('Recall del Modelo LogisticRegression con con LASSO el set de
Pruebas: {:.4f}'.format(lasso_model_recall_score))
lasso_model_f1_score = f1_score(y_test, y_pred_lasso_model,
average='macro')
print('F1 Score del Modelo LogisticRegression con LASSO con el set de
Pruebas: {:.4f}'.format(lasso_model_f1_score))
print("=========================================================.")
scores =
```

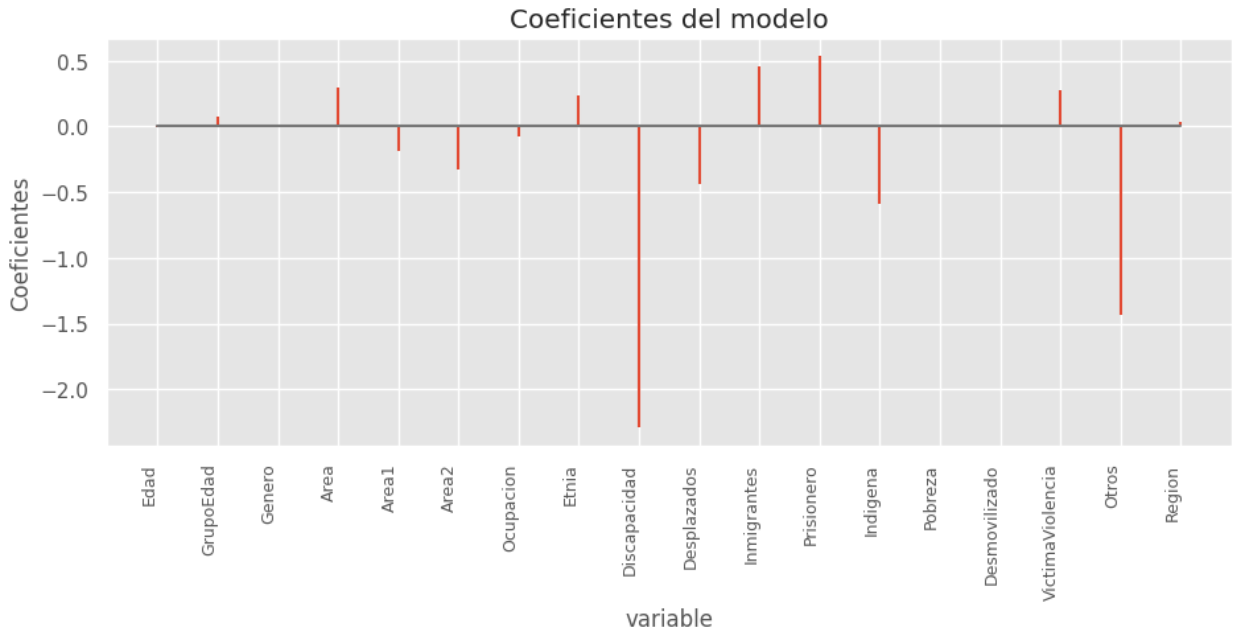
```
cross_val_score(lasso_model,X_train,y_train,cv=5,scoring='roc_auc',n_j
obs=-1)
lasso_model_roc_auc = scores.mean()
print("AUC Cross Val Score del set de entrenamiento:
{:.4f}".format(lasso_model_roc_auc))
acc =
cross_val_score(lasso_model,X_train,y_train,cv=5,scoring='accuracy',n_
jobs=-1)
lasso_model_accv = acc.mean()
print("Accuracy Cross Val Score del Modelo LogisticRegression con
LASSO para el set de entrenamiento: {:.4f}".format(lasso_model_accv))
lasso_model_score_train = lasso_model.score(X_train, y_train)
print("Accuracy del Modelo LogisticRegression con LASSO del set de
entrenamiento set: {:.4f}".format(lasso_model_score_train))
lasso_model_score_test = lasso_model.score(X_test, y_test)
print("Accuracy del Modelo LogisticRegression con LASSO del set de
pruebas: {:.4f}".format(lasso_model_score_test))
```

```
============================================================.
Precisión del Modelo LogisticRegression con LASSO el set de Pruebas:
0.5026
Recall del Modelo LogisticRegression con con LASSO el set de Pruebas:
0.6022
F1 Score del Modelo LogisticRegression con LASSO con el set de
Pruebas: 0.3972
============================================================.
AUC Cross Val Score del set de entrenamiento: 0.5622
Accuracy Cross Val Score del Modelo LogisticRegression con LASSO para
el set de entrenamiento: 0.6471
Accuracy del Modelo LogisticRegression con LASSO del set de
entrenamiento set: 0.6319
Accuracy del Modelo LogisticRegression con LASSO del set de pruebas:
0.6358
```

```
# Coeficientes del modelo
#
================================================================================
========
coefficients = lasso_model.coef_
intercept = lasso_model.intercept_
print("")
print("Coeficientes:", coefficients)
print("Intercepto:", intercept)
print("")
dfDenge_coeficientes = pd.DataFrame({'predictor': X.columns, 'coef':
lasso_model.coef_[0]})

fig, ax = plt.subplots(figsize=(11, 4))
ax.stem(dfDenge_coeficientes.predictor, dfDenge_coeficientes.coef,
markerfmt=' ')
```

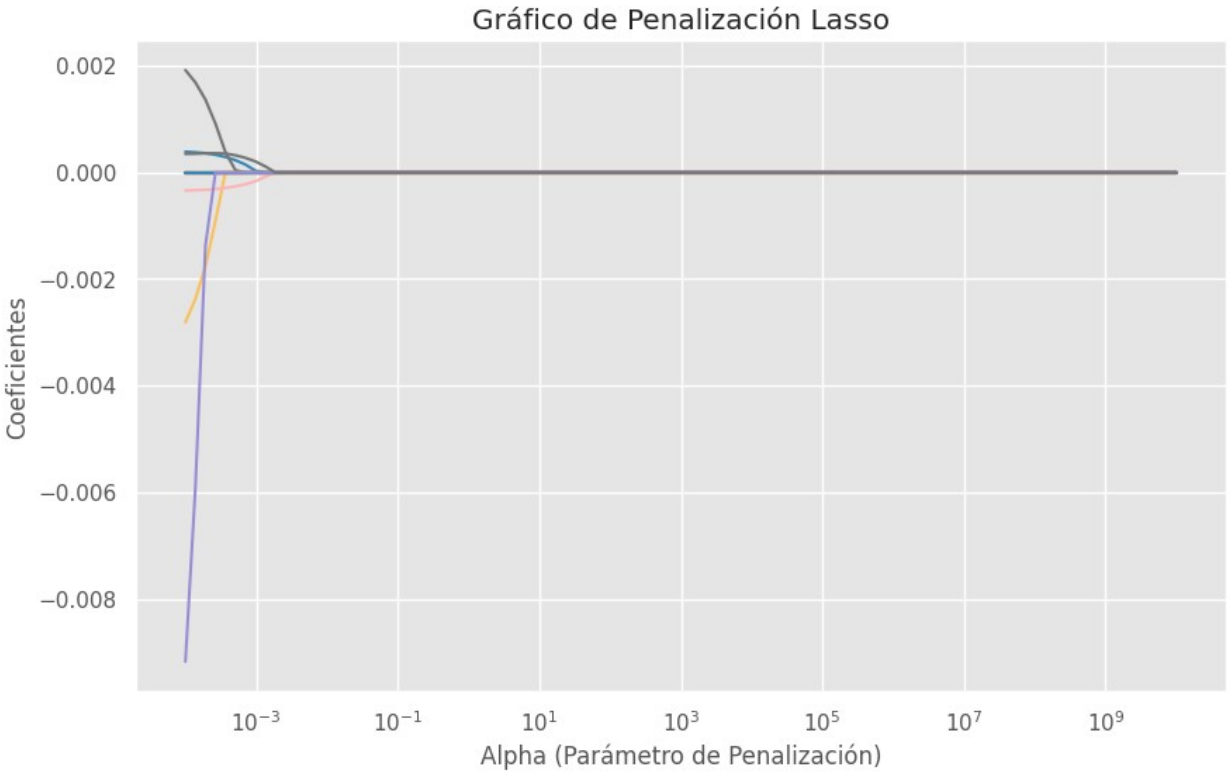
```
plt.xticks(rotation=90, ha='right', size=9)
ax.set_xlabel('variable')
ax.set_ylabel('Coeficientes')
ax.set_title('Coeficientes del modelo');
```

```
Coeficientes: [[-8.41241928e-04  7.64660276e-02  1.28285979e-02
2.94470176e-01
  -1.89015515e-01 -3.32073209e-01 -7.68615000e-02  2.30436913e-01
  -2.28236552e+00 -4.37924317e-01  4.53427499e-01  5.31884863e-01
  -5.89717241e-01  0.00000000e+00  0.00000000e+00  2.74833296e-01
  -1.43041969e+00  3.80265179e-02]]
Intercepto: [1.16362624]
```




```
# Crear una lista de valores de alpha para la penalización Lasso
alphas = np.logspace(-4, 10, 100)
# Crear una lista para almacenar los coeficientes para cada valor de
alpha
coefs = []
# Iterar sobre los valores de alpha y ajustar el modelo Lasso
for alpha in alphas:
    lasso_model = Lasso(alpha=alpha)
    lasso_model.fit(X_train, y_train)
    coefs.append(lasso_model.coef_)

# Convertir la lista de coeficientes a un array NumPy
coefs = np.array(coefs)

# Graficar la penalización Lasso
```

```
plt.figure(figsize=(10, 6))
ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale('log')
plt.xlabel('Alpha (Parámetro de Penalización)')
plt.ylabel('Coeficientes')
plt.title('Gráfico de Penalización Lasso')
#plt.legend(X.columns)
plt.show()
```



##Modelo LogisticRegression con regularización Logit-Ridge

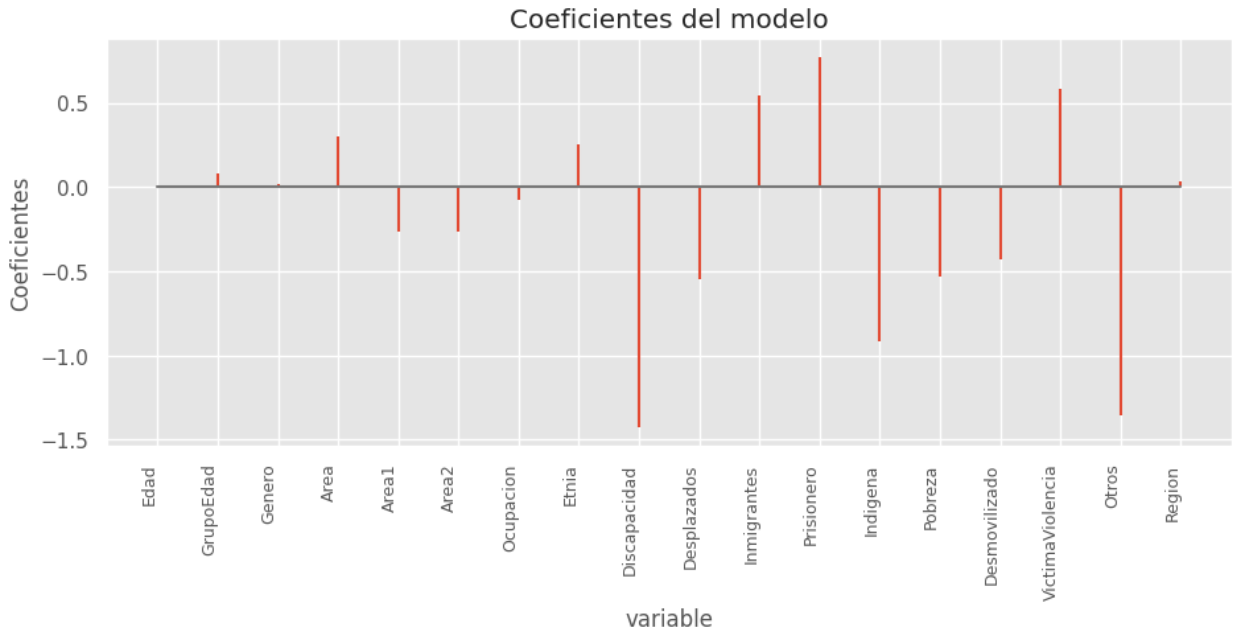
```
# Ajustar modelo Logit-Ridge
ridge_model = LogisticRegression(C = 0.1, ### parametro de
penalización, valores menores a uno penalizacion alta
                                 class_weight= 'balanced',
                                 penalty= 'l2',  ## Elgir como
penalizar: l1, l2, elasticnet, None
                                 solver= 'liblinear', ### Algoritmo de
optimización
                                 random_state=42)
ridge_model.fit(X_train, y_train)
y_pred_ridge_model = ridge_model.predict(X_test)
```

```
print("============================================================.")
ridge_model_precision_score = precision_score(y_test,
y_pred_ridge_model, average='macro')
print("Precisión del Modelo LogisticRegression con Ridge el set de
Pruebas: {:.4f}".format(ridge_model_precision_score))
ridge_model_recall_score = recall_score(y_test, y_pred_ridge_model,
average='macro')
print('Recall del Modelo LogisticRegression con con Ridge el set de
Pruebas: {:.4f}'.format(ridge_model_recall_score))
ridge_model_f1_score = f1_score(y_test, y_pred_ridge_model,
average='macro')
print('F1 Score del Modelo LogisticRegression con Ridge con el set de
Pruebas: {:.4f}'.format(ridge_model_f1_score))
print("============================================================.")
scores =
cross_val_score(ridge_model,X_train,y_train,cv=5,scoring='roc_auc',n_j
obs=-1)
ridge_model_roc_auc = scores.mean()
print("AUC Cross Val Score del set de entrenamiento:
{:.4f}".format(ridge_model_roc_auc))
acc =
cross_val_score(ridge_model,X_train,y_train,cv=5,scoring='accuracy',n_
jobs=-1)
ridge_model_accv = acc.mean()
print("Accuracy Cross Val Score del Modelo LogisticRegression con
Ridge para el set de entrenamiento: {:.4f}".format(ridge_model_accv))
ridge_model_score_train = ridge_model.score(X_train, y_train)
print("Accuracy del Modelo LogisticRegression con Ridge del set de
entrenamiento set: {:.4f}".format(ridge_model_score_train))
ridge_model_score_test = ridge_model.score(X_test, y_test)
print("Accuracy del Modelo LogisticRegression con Ridge del set de
pruebas: {:.4f}".format(ridge_model_score_test))
```

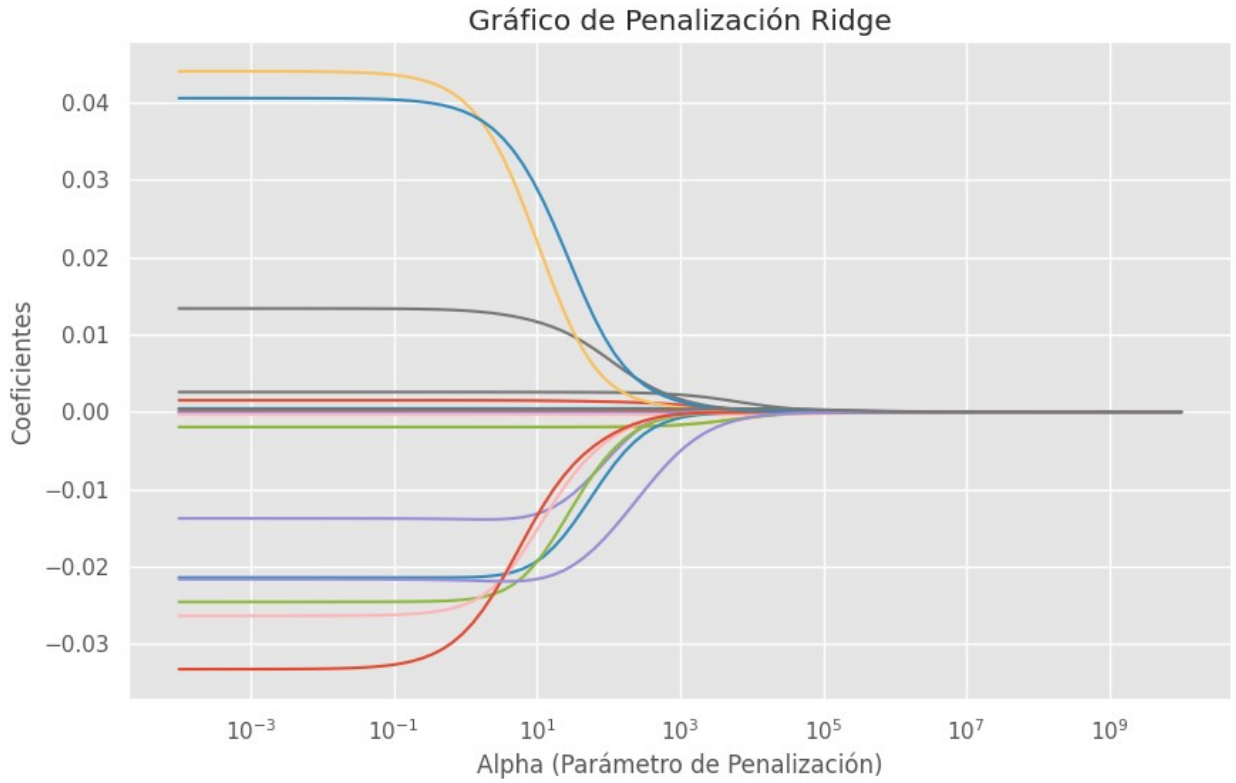
```
============================================================.
Precisión del Modelo LogisticRegression con Ridge el set de Pruebas:
0.5025
Recall del Modelo LogisticRegression con con Ridge el set de Pruebas:
0.6013
F1 Score del Modelo LogisticRegression con Ridge con el set de
Pruebas: 0.3964
============================================================.
AUC Cross Val Score del set de entrenamiento: 0.5613
Accuracy Cross Val Score del Modelo LogisticRegression con Ridge para
el set de entrenamiento: 0.6434
Accuracy del Modelo LogisticRegression con Ridge del set de
entrenamiento set: 0.6300
Accuracy del Modelo LogisticRegression con Ridge del set de pruebas:
0.6340
```

```
# Coeficientes del modelo Ridge
#
==========================================================================
========
coefficients = ridge_model.coef_
intercept = ridge_model.intercept_
print("")
print("Coeficientes:", coefficients)
print("Intercepto:", intercept)
print("")
dfDenge_coeficientes = pd.DataFrame({'predictor': X.columns, 'coef':
ridge_model.coef_[0]})

fig, ax = plt.subplots(figsize=(11, 4))
ax.stem(dfDenge_coeficientes.predictor, dfDenge_coeficientes.coef,
markerfmt=' ')
plt.xticks(rotation=90, ha='right', size=9)
ax.set_xlabel('variable')
ax.set_ylabel('Coeficientes')
ax.set_title('Coeficientes del modelo');
```

```
Coeficientes: [[-8.50439631e-04  7.77482534e-02  1.60169129e-02
2.99254371e-01
  -2.64200840e-01 -2.64200840e-01 -7.70977715e-02  2.50401010e-01
  -1.42189378e+00 -5.45012447e-01  5.47257181e-01  7.74827506e-01
  -9.18566471e-01 -5.30221476e-01 -4.33187073e-01  5.85668265e-01
  -1.35313354e+00  3.67836632e-02]]
Intercepto: [1.06641327]
```



```python
# Crear una lista de valores de alpha para la penalización Lasso
alphas = np.logspace(-4, 10, 100)
# Crear una lista para almacenar los coeficientes para cada valor de
alpha
coefs = []
# Iterar sobre los valores de alpha y ajustar el modelo Lasso
for alpha in alphas:
    ridge_model = Ridge(alpha=alpha)
    ridge_model.fit(X_train, y_train)
    coefs.append(ridge_model.coef_)

# Convertir la lista de coeficientes a un array NumPy
coefs = np.array(coefs)

# Graficar la penalización Lasso
plt.figure(figsize=(10, 6))
ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale('log')
plt.xlabel('Alpha (Parámetro de Penalización)')
plt.ylabel('Coeficientes')
plt.title('Gráfico de Penalización Ridge')
#plt.legend(X.columns)
plt.show()
```



```
# Ajustar modelo Logit-Enet
enet_model = LogisticRegression(
    C = 0.1,
    class_weight= 'balanced',
    penalty= 'elasticnet',
    solver= 'saga',
    random_state=42,
    l1_ratio=0.5
    )

enet_model.fit(X_train, y_train)
y_pred_enet_model = enet_model.predict(X_test)

print("============================================================.")
enet_model_precision_score = precision_score(y_test,
y_pred_enet_model, average='macro')
print("Precisión del Modelo LogisticRegression con Logit-Enet el set
de Pruebas: {:.4f}".format(enet_model_precision_score))
enet_model_recall_score = recall_score(y_test, y_pred_enet_model,
average='macro')
print('Recall del Modelo LogisticRegression con con Logit-Enet el set
de Pruebas: {:.4f}'.format(enet_model_recall_score))
enet_model_f1_score = f1_score(y_test, y_pred_enet_model,
average='macro')
print('F1 Score del Modelo LogisticRegression con Logit-Enet con el
set de Pruebas: {:.4f}'.format(enet_model_f1_score))
print("============================================================.")
scores =
cross_val_score(enet_model,X_train,y_train,cv=5,scoring='roc_auc',n_jo
bs=-1)
enet_model_roc_auc = scores.mean()
print("AUC Cross Val Score del set de entrenamiento:
{:.4f}".format(enet_model_roc_auc))
acc =
cross_val_score(enet_model,X_train,y_train,cv=5,scoring='accuracy',n_j
obs=-1)
enet_model_accv = acc.mean()
print("Accuracy Cross Val Score del Modelo LogisticRegression con
Logit-Enet para el set de entrenamiento:
{:.4f}".format(enet_model_accv))
enet_model_score_train = enet_model.score(X_train, y_train)
print("Accuracy del Modelo LogisticRegression con Logit-Enet del set
de entrenamiento set: {:.4f}".format(enet_model_score_train))
enet_model_score_test = enet_model.score(X_test, y_test)
print("Accuracy del Modelo LogisticRegression con Logit-Enet del set
de pruebas: {:.4f}".format(enet_model_score_test))
```

```
============================================================.
Precisión del Modelo LogisticRegression con Logit-Enet el set de
Pruebas: 0.5044
```

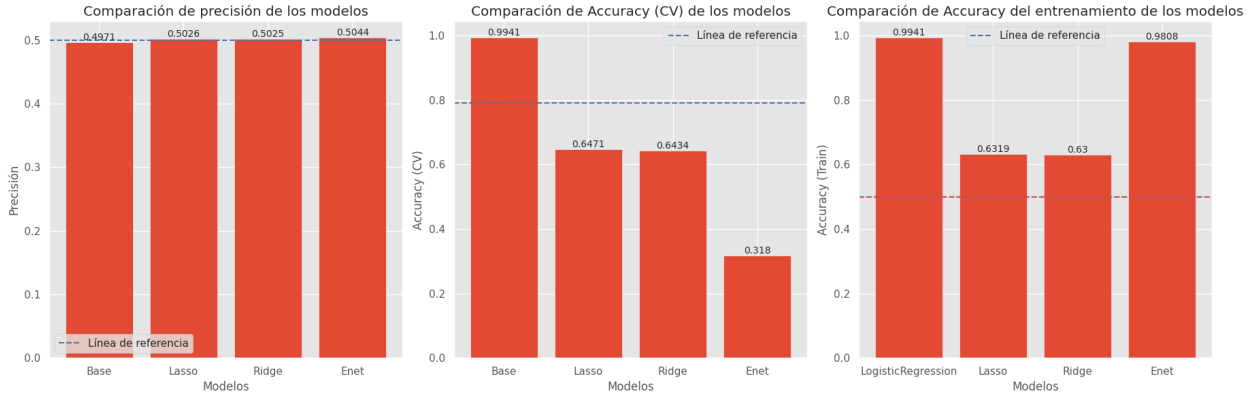
```
Recall del Modelo LogisticRegression con con Logit-Enet el set de
Pruebas: 0.5102
F1 Score del Modelo LogisticRegression con Logit-Enet con el set de
Pruebas: 0.5053
========================================================.
AUC Cross Val Score del set de entrenamiento: 0.5287
Accuracy Cross Val Score del Modelo LogisticRegression con Logit-Enet
para el set de entrenamiento: 0.3180
Accuracy del Modelo LogisticRegression con Logit-Enet del set de
entrenamiento set: 0.9808
Accuracy del Modelo LogisticRegression con Logit-Enet del set de
pruebas: 0.9808
```

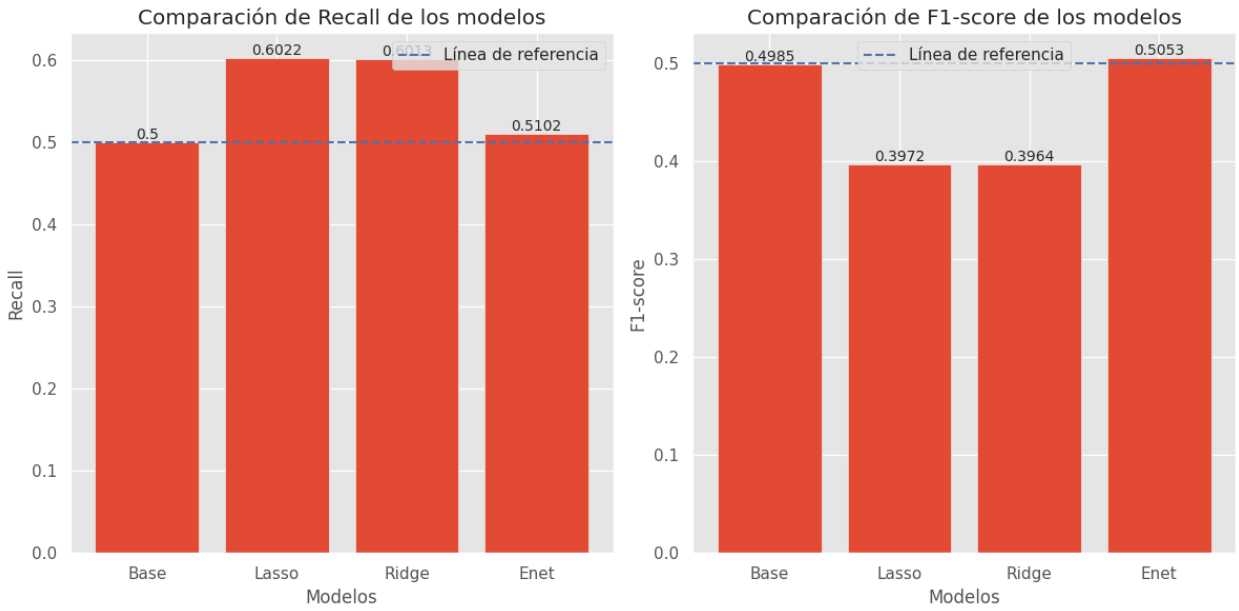
```python
# Datos de recall para cada modelo
recall_scores = {
    'Base': logitreg_model_recall_score,
    'Lasso': lasso_model_recall_score,
    'Ridge': ridge_model_recall_score,
    'Enet': enet_model_recall_score,
}

# Datos de precisión para cada modelo
precision_scores = {
    'Base': logitreg_model_precision_score,
    'Lasso': lasso_model_precision_score,
    'Ridge': ridge_model_precision_score,
    'Enet': enet_model_precision_score,
}

# Datos de accuracy para cada modelo
accuracy_scores = {
    'Base': logitreg_model_accv,
    'Lasso': lasso_model_accv,
    'Ridge': ridge_model_accv,
    'Enet': enet_model_accv,
}

# Datos de accuracy para el entrenamiento de cada modelo
accuracy_scores_train = {
    'LogisticRegression': logitreg_model_score_train,
    'Lasso': lasso_model_score_train,
    'Ridge': ridge_model_score_train,
    'Enet': enet_model_score_train,
}


# Crear una figura con subplots
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6))
```

```
# Gráfico 1: Precisión
models = list(precision_scores.keys())
scores = list(precision_scores.values())

axes[0].bar(models, scores)
#axes[0].set_ylim(0.5, 1)
axes[0].set_xlabel('Modelos')
axes[0].set_ylabel('Precisión')
axes[0].set_title('Comparación de precisión de los modelos')
for i, score in enumerate(scores):
  axes[0].text(i, score, str(round(score, 4)), ha='center',
va='bottom')

axes[0].axhline(y=0.5, color='b', linestyle='--', label='Línea de
referencia')
axes[0].legend()


# Gráfico 2: Accuracy (Cross-Validation)
models = list(accuracy_scores.keys())
scores = list(accuracy_scores.values())

axes[1].bar(models, scores)
#axes[1].set_ylim(0.5, 1)
axes[1].set_xlabel('Modelos')
axes[1].set_ylabel('Accuracy (CV)')
axes[1].set_title('Comparación de Accuracy (CV) de los modelos')
for i, score in enumerate(scores):
  axes[1].text(i, score, str(round(score, 4)), ha='center',
va='bottom')

axes[1].axhline(y=0.79, color='b', linestyle='--', label='Línea de
referencia')
axes[1].legend()

# Gráfico 3: Accuracy del entrenamiento
models = list(accuracy_scores_train.keys())
scores = list(accuracy_scores_train.values())

axes[2].bar(models, scores)
#axes[2].set_ylim(0.5, 1)
axes[2].set_xlabel('Modelos')
axes[2].set_ylabel('Accuracy (Train)')
axes[2].set_title('Comparación de Accuracy del entrenamiento de los
modelos')
for i, score in enumerate(scores):
  axes[2].text(i, score, str(round(score, 4)), ha='center',
va='bottom')
```

```
axes[2].axhline(y=0.5, color='b', linestyle='--', label='Línea de
referencia')
axes[2].legend()

# Ajustar los subplots para evitar superposiciones
plt.tight_layout()

# Agregar una línea horizontal para comparar
plt.axhline(y=0.5, color='r', linestyle='--')

# Mostrar la figura
plt.show()

# Datos de recall para cada modelo
recall_scores = {
    'Base': logitreg_model_recall_score,
    'Lasso': lasso_model_recall_score,
    'Ridge': ridge_model_recall_score,
    'Enet': enet_model_recall_score,
}

# Datos de F1-score para cada modelo
f1_scores = {
    'Base': logitreg_model_f1_score,
    'Lasso': lasso_model_f1_score,
    'Ridge': ridge_model_f1_score,
    'Enet': enet_model_f1_score,
}

# Crear una figura con subplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

# Gráfico 1: Recall
models = list(recall_scores.keys())
scores = list(recall_scores.values())

axes[0].bar(models, scores)
#axes[0].set_ylim(0.5, 1)
axes[0].set_xlabel('Modelos')
axes[0].set_ylabel('Recall')
axes[0].set_title('Comparación de Recall de los modelos')
for i, score in enumerate(scores):
  axes[0].text(i, score, str(round(score, 4)), ha='center',
va='bottom')

axes[0].axhline(y=0.5, color='b', linestyle='--', label='Línea de
referencia')
axes[0].legend()
```

```
# Gráfico 2: F1-score
models = list(f1_scores.keys())
scores = list(f1_scores.values())

axes[1].bar(models, scores)
#axes[1].set_ylim(0.5, 1)
axes[1].set_xlabel('Modelos')
axes[1].set_ylabel('F1-score')
axes[1].set_title('Comparación de F1-score de los modelos')
for i, score in enumerate(scores):
  axes[1].text(i, score, str(round(score, 4)), ha='center',
va='bottom')

axes[1].axhline(y=0.5, color='b', linestyle='--', label='Línea de
referencia')
axes[1].legend()
# Ajustar los subplots para evitar superposiciones
plt.tight_layout()

# Mostrar la figura
plt.show()
```





El estudio realizado analiza la capacidad predictiva de diferentes modelos de regresión logística para predecir la ocurrencia de eventos de dengue utilizando variables sociodemográficas. Se evaluaron modelos sin regularización y con regularización LASSO, Ridge y Elastic Net.

Obtenemos una precisión del modelo usando average='macro' dado que, el uso de macro-promedio es una buena medida para evaluar el rendimiento del modelo cuando se trabaja con un problema de clasificación con clases desbalanceadas.

La precesión del modelo en relación del set de pruebas con la predicción es de **0.4971**. Indica que el modelo no tiene un buen desempeño, y la precisión en la predicción de la variable objetivo **"Evento"**' no es relativamente baja. El que alcanzó un mejor desempeño de los 4 fue el modelo usando Lasso con una precesión de **0.5026**

En el contexto del modelo para predecir "Dengue" o "Dengue Grave", un F1 Score de 0,5 sugiere que el modelo tiene un nivel promedio de rendimiento en cuanto a la capacidad de clasificar correctamente y minimizar tanto los falsos positivos como los falsos negativos lo que también se evidencia en el cálculo anterior de la precisión del modelo. El que alcanzó un mejor desempeño de los 4 fue el modelo usando Elastic Net con un valor de **0.5053**

En el caso de la predicción de enfermedades como el dengue, un Recall bajo podría ser un problema, porque significa que se podrían estar pasando por alto casos positivos reales, en nuestro caso un Recall de 0.5 para la clase Evento significaría que el modelo solo identifica la mitad de los casos reales de dengue. El que alcanzó un mejor desempeño de los 4 fue el modelo usando Lasso con un valor de **0.6022** Para el caso de predicción de enfermedades como se presenta en el caso de estudio, un Recall alto es deseable para poder identificar correctamente la mayoría de los casos positivos reales.

- Se revisa la proporción de los datos de los Eventos antes y después del split del dataset y se observa que se conserva la proporción.

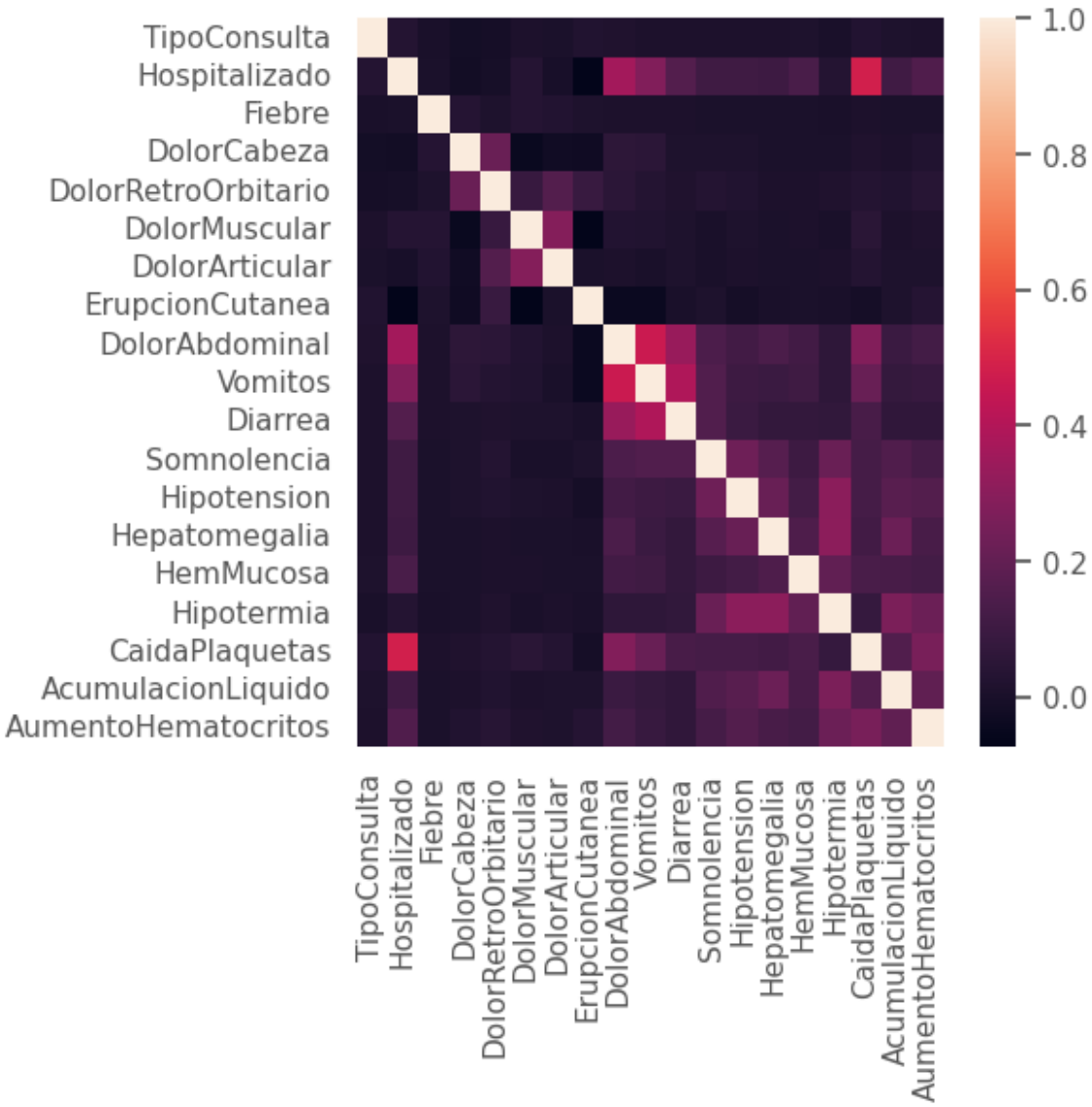
- Los modelos de regresión logística no demostraron un buen desempeño menor que aceptable en la predicción de eventos de dengue, con valores de precisión, Recall y F1-score cercanos a 0.5 en la mayoría de los casos.
- Los modelos con regularización mejoraron el desempeño en los datos de entrenamiento, pero se observó una disminución considerable en la precisión en los datos de prueba. Esto indica un posible sobreajuste (overfitting).

#Modelos *Clínico*-médico

###Filtrando las variables clínico-medicas

Para poder evaluar la capacidad predictiva de cada modelo, se dividen las observaciones disponibles en dos grupos: uno de entrenamiento (70%) y otro de test (30%).

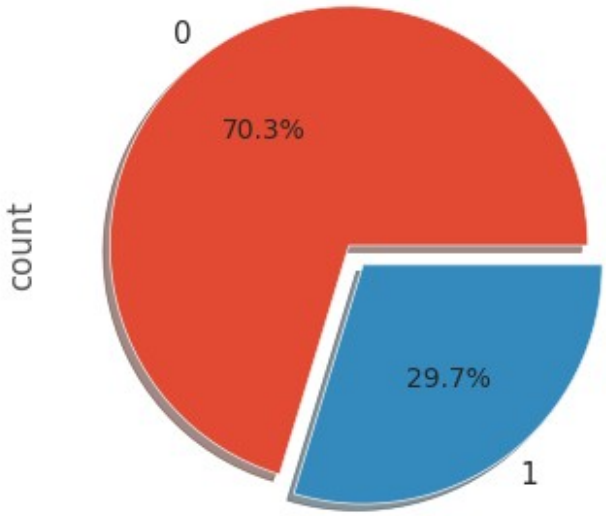
```
clinico_med_vars =  ['TipoConsulta', 'Hospitalizado', 'Fiebre',
'DolorCabeza', 'DolorRetroOrbitario', 'DolorMuscular',
'DolorArticular', 'ErupcionCutanea', 'DolorAbdominal', 'Vomitos',
'Diarrea', 'Somnolencia', 'Hipotension', 'Hepatomegalia', 'HemMucosa',
'Hipotermia', 'CaidaPlaquetas', 'AcumulacionLiquido',
'AumentoHematocritos']

df_socio_clinic = dfDengue[clinico_med_vars]

correlation_matrix = df_socio_clinic.corr()

fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (5, 5))
sns.heatmap(correlation_matrix, ax = ax)

<Axes: >
```




```
unique_nombre_Hospitalizado_counts =
dfDengue['Hospitalizado'].value_counts()
print(unique_nombre_Hospitalizado_counts)
```

```
Hospitalizado
0    35437
1    14960
Name: count, dtype: int64
```

```
plt.figure(figsize=[4,4])
dfDengue['Hospitalizado'].value_counts().plot.pie(explode=[0,0.1],auto
pct='%1.1f%%',shadow=True)
plt.show()
```




```
dfDengueClinicoMed = dfDengue[clinico_med_vars].copy()

# Definir variables predictoras (X) y variable objetivo (y)
X = dfDengueClinicoMed.drop('Hospitalizado', axis=1)
y = (dfDengueClinicoMed['Hospitalizado']).astype(int)
y = y.values.ravel()

# Escalar las variables predictoras
#scaler = StandardScaler()
#X_scaled = scaler.fit_transform(X)

from sklearn.preprocessing import MinMaxScaler

# Dividir los datos en conjuntos de entrenamiento y prueba (opcional,
para evaluar el modelo)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

scaler = MinMaxScaler() #saga solver requires features to be scaled
for model conversion

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Porcentaje de los valores unicos de , y_train
unique_y_train_counts =
pd.Series(y_train).value_counts(normalize=True) * 100
unique_y_train_counts
```

```
0    70.252572
1    29.747428
Name: proportion, dtype: float64
```

##Modelo LogisticRegression sin regularización

```
#LogisticRegression
logitreg_model = LogisticRegression(penalty=None)
logitreg_model.fit(X_train, y_train)
y_pred = logitreg_model.predict(X_test)

print("Usaremos estas métricas como comparación de referencia para
cualquier mejora que obtengamos con la regularizacion Logit-Ridge,
Logit-LASSO y Logit-Enet")
print("============================================================.")
logitreg_model_precision_score = precision_score(y_test, y_pred,
average='macro')
print("Precisión del Modelo LogisticRegression con el set de Pruebas:
{:.4f}".format(logitreg_model_precision_score))
logitreg_model_recall_score = recall_score(y_test, y_pred,
average='macro')
print('Recall del Modelo LogisticRegression con el set de Pruebas:
{:.4f}'.format(logitreg_model_recall_score))
logitreg_model_f1_score = f1_score(y_test, y_pred, average='macro')
print('F1 Score del Modelo LogisticRegression con el set de Pruebas:
{:.4f}'.format(logitreg_model_f1_score))
print("============================================================.")
scores =
cross_val_score(logitreg_model,X_train,y_train,cv=5,scoring='roc_auc',
n_jobs=-1)
logitreg_model_roc_auc = scores.mean()
print("AUC Cross Val Score del set de entrenamiento:
{:.4f}".format(logitreg_model_roc_auc))
acc =
cross_val_score(logitreg_model,X_train,y_train,cv=5,scoring='accuracy'
,n_jobs=-1)
logitreg_model_accv = acc.mean()
print("Accuracy Cross Val Score del Modelo LogisticRegression para el
set de entrenamiento: {:.4f}".format(logitreg_model_accv))
logitreg_model_score_train = logitreg_model.score(X_train, y_train)
print("Accuracy del Modelo LogisticRegression del set de entrenamiento
set: {:.4f}".format(logitreg_model_score_train))
logitreg_model_score_test = logitreg_model.score(X_test, y_test)
print("Accuracy del Modelo LogisticRegression del set de pruebas:
{:.4f}".format(logitreg_model_score_test))
# Error de test del modelo
#
================================================================================
========
```

```
logitreg_model_rmse_ols = mean_squared_error(y_test, y_pred, squared =
False )
print("El error (rmse) de test es:
{:.4f}".format(logitreg_model_rmse_ols))
```

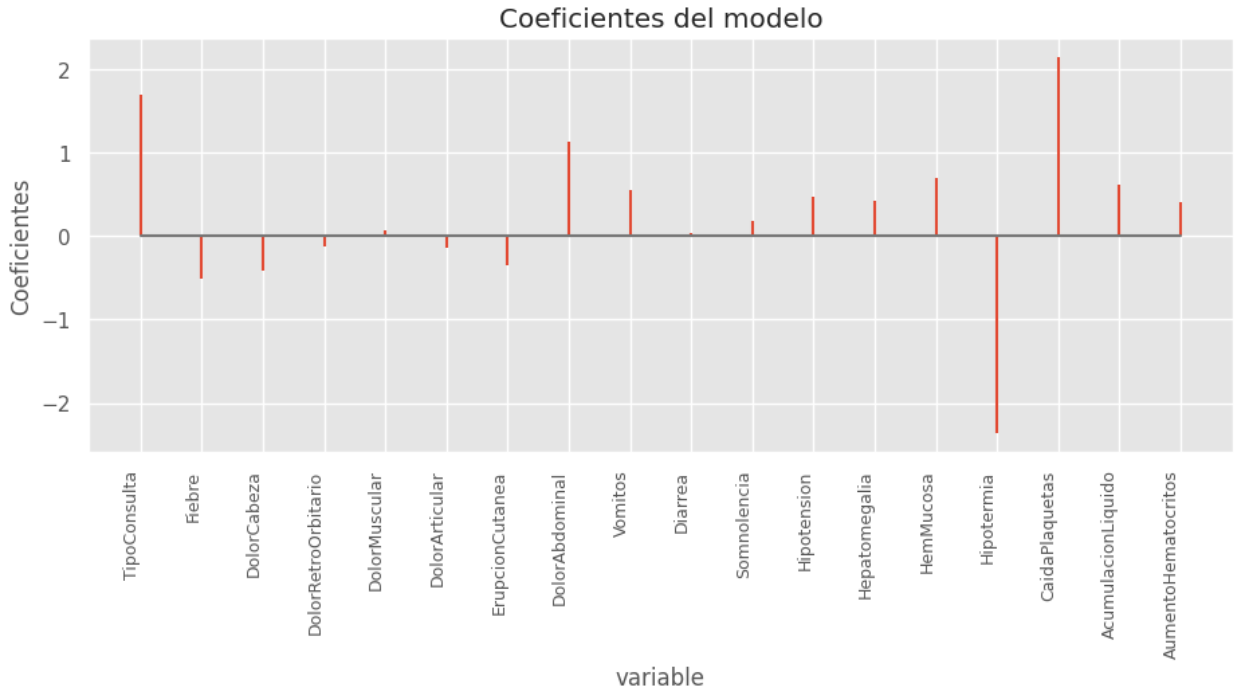
```
Usaremos estas métricas como comparación de referencia para cualquier
mejora que obtengamos con la regularizacion Logit-Ridge, Logit-LASSO y
Logit-Enet
==========================================================.
Precisión del Modelo LogisticRegression con el set de Pruebas: 0.7636
Recall del Modelo LogisticRegression con el set de Pruebas: 0.7138
F1 Score del Modelo LogisticRegression con el set de Pruebas: 0.7298
==========================================================.
AUC Cross Val Score del set de entrenamiento: 0.8299
Accuracy Cross Val Score del Modelo LogisticRegression para el set de
entrenamiento: 0.7936
Accuracy del Modelo LogisticRegression del set de entrenamiento set:
0.7941
Accuracy del Modelo LogisticRegression del set de pruebas: 0.7946
El error (rmse) de test es: 0.4532
```

```
residuals_logitreg_model = y_test - y_pred
print("Media de los resideales:", np.mean(residuals_logitreg_model))
print("Desviación estándar de los residuales:",
np.std(residuals_logitreg_model))
```

```
Media de los resideales: 0.08029100529100529
Desviación estándar de los residuales: 0.4460679711575807
```

```
# Coeficientes del modelo
#
================================================================================
========
coefficients = logitreg_model.coef_
intercept = logitreg_model.intercept_
print("")
print("Coeficientes:", coefficients)
print("Intercepto:", intercept)
print("")
dfDenge_coeficientes = pd.DataFrame({'predictor': X.columns, 'coef':
logitreg_model.coef_[0]})

fig, ax = plt.subplots(figsize=(11, 4))
ax.stem(dfDenge_coeficientes.predictor, dfDenge_coeficientes.coef,
markerfmt=' ')
plt.xticks(rotation=90, ha='right', size=9)
ax.set_xlabel('variable')
ax.set_ylabel('Coeficientes')
ax.set_title('Coeficientes del modelo');
```

```
Coeficientes: [[ 1.69405147 -0.51559075 -0.40726208 -0.12744185
0.06727224 -0.14579614
  -0.34648925  1.13377242  0.55321914  0.04239808  0.18277696
0.47300195
   0.42347637  0.69171068 -2.35293212  2.15647518  0.61619306
0.40618646]]
Intercepto: [-0.8835427]
```



##Modelo LogisticRegression con regularización Logit-Lasso

```
lasso_model = LogisticRegression(random_state=42,
                                 C = 0.1, ### parametro de
penalización, valores menores a uno penalizacion alta
                                 class_weight= 'balanced',
                                 penalty= 'l1', ## Elgir como
penalizar: l1, l2, elasticnet, None
                                 solver= 'liblinear' ### Algoritmo de
optimización
                                 )
lasso_model.fit(X_train, y_train) #This line is added to train the
model
y_pred_lasso_model = lasso_model.predict(X_test)

print("==========================================================.")
lasso_model_precision_score = precision_score(y_test,
y_pred_lasso_model, average='macro')
```

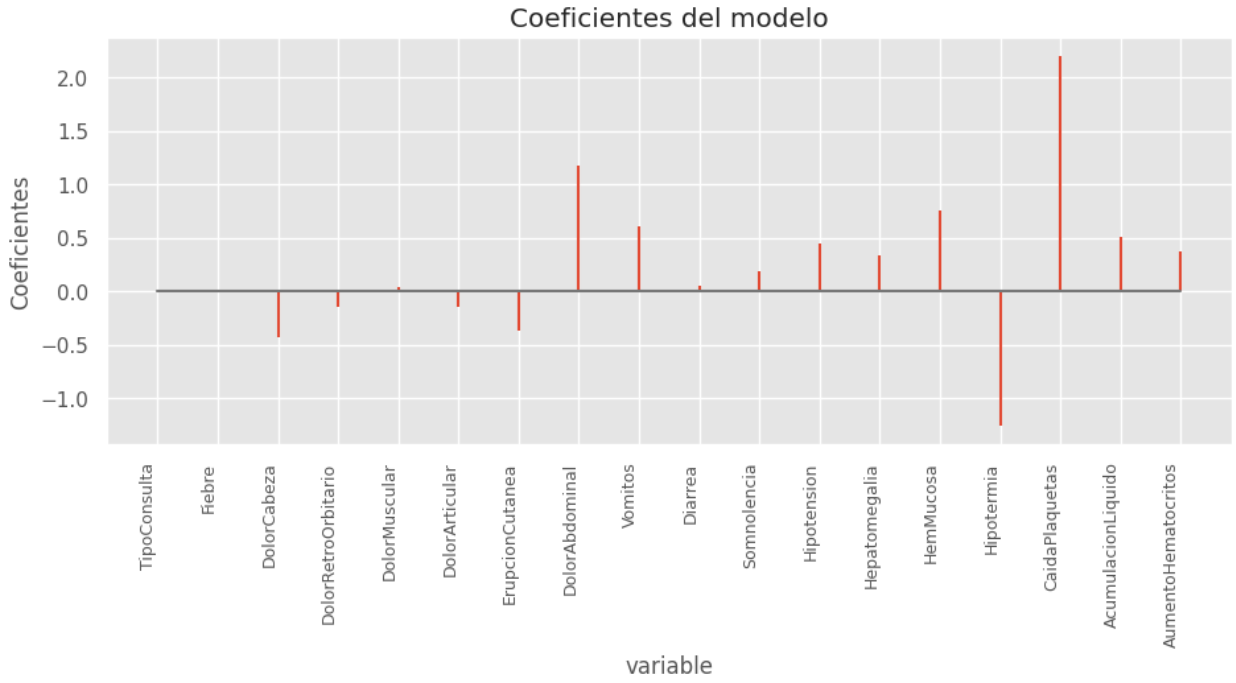
```
print("Precisión del Modelo LogisticRegression con LASSO el set de
Pruebas: {:.4f}".format(lasso_model_precision_score))
lasso_model_recall_score = recall_score(y_test, y_pred_lasso_model,
average='macro')
print('Recall del Modelo LogisticRegression con con LASSO el set de
Pruebas: {:.4f}'.format(lasso_model_recall_score))
lasso_model_f1_score = f1_score(y_test, y_pred_lasso_model,
average='macro')
print('F1 Score del Modelo LogisticRegression con LASSO con el set de
Pruebas: {:.4f}'.format(lasso_model_f1_score))
print("=========================================================.")
scores =
cross_val_score(lasso_model,X_train,y_train,cv=5,scoring='roc_auc',n_j
obs=-1)
lasso_model_roc_auc = scores.mean()
print("AUC Cross Val Score del set de entrenamiento:
{:.4f}".format(lasso_model_roc_auc))
acc =
cross_val_score(lasso_model,X_train,y_train,cv=5,scoring='accuracy',n_
jobs=-1)
lasso_model_accv = acc.mean()
print("Accuracy Cross Val Score del Modelo LogisticRegression con
LASSO para el set de entrenamiento: {:.4f}".format(lasso_model_accv))
lasso_model_score_train = lasso_model.score(X_train, y_train)
print("Accuracy del Modelo LogisticRegression con LASSO del set de
entrenamiento set: {:.4f}".format(lasso_model_score_train))
lasso_model_score_test = lasso_model.score(X_test, y_test)
print("Accuracy del Modelo LogisticRegression con LASSO del set de
pruebas: {:.4f}".format(lasso_model_score_test))
```

```
=========================================================.
Precisión del Modelo LogisticRegression con LASSO el set de Pruebas:
0.7428
Recall del Modelo LogisticRegression con con LASSO el set de Pruebas:
0.7639
F1 Score del Modelo LogisticRegression con LASSO con el set de
Pruebas: 0.7507
=========================================================.
AUC Cross Val Score del set de entrenamiento: 0.8291
Accuracy Cross Val Score del Modelo LogisticRegression con LASSO para
el set de entrenamiento: 0.7828
Accuracy del Modelo LogisticRegression con LASSO del set de
entrenamiento set: 0.7834
Accuracy del Modelo LogisticRegression con LASSO del set de pruebas:
0.7828
```

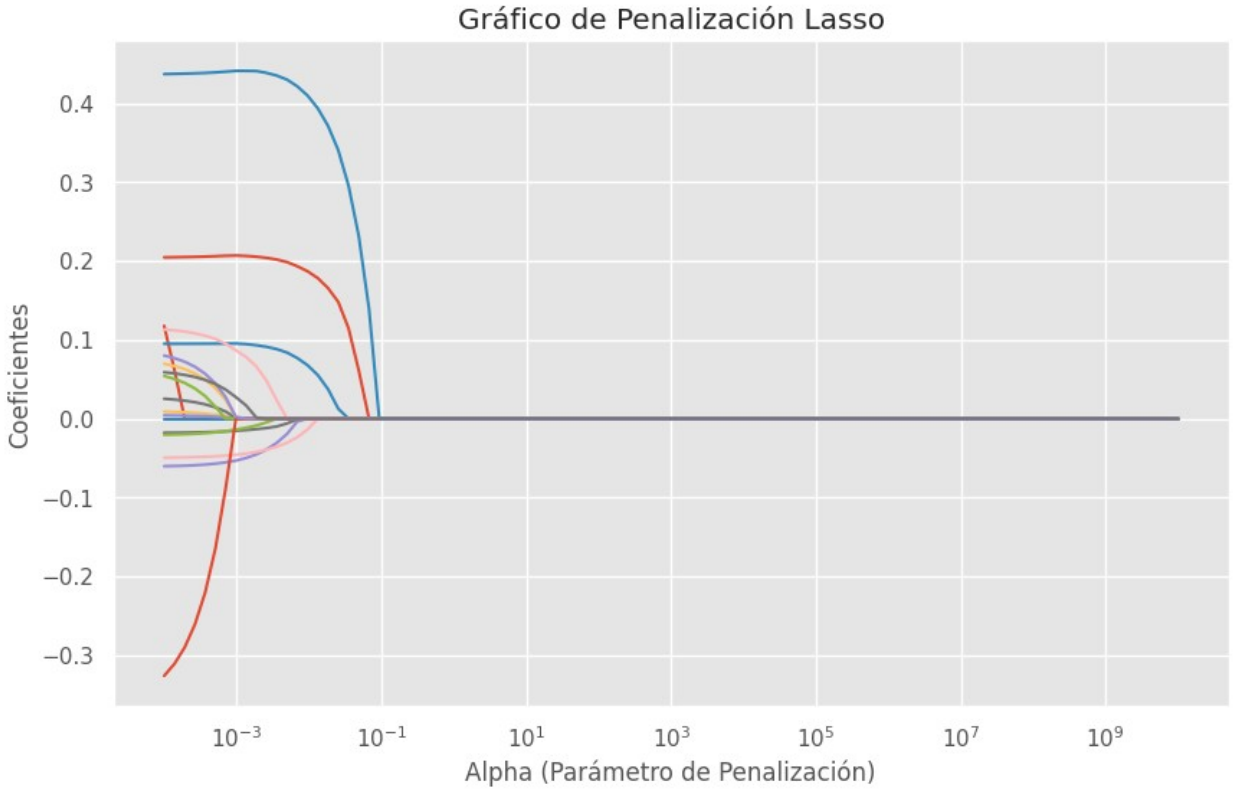
```
# Coeficientes del modelo
#
================================================================================
========
```

```
coefficients = lasso_model.coef_
intercept = lasso_model.intercept_
print("")
print("Coeficientes:", coefficients)
print("Intercepto:", intercept)
print("")
dfDenge_coeficientes = pd.DataFrame({'predictor': X.columns, 'coef':
lasso_model.coef_[0]})

fig, ax = plt.subplots(figsize=(11, 4))
ax.stem(dfDenge_coeficientes.predictor, dfDenge_coeficientes.coef,
markerfmt=' ')
plt.xticks(rotation=90, ha='right', size=9)
ax.set_xlabel('variable')
ax.set_ylabel('Coeficientes')
ax.set_title('Coeficientes del modelo');
```

```
Coeficientes: [[ 0.          0.         -0.43122807 -0.14295073
0.04016511 -0.15179909
  -0.37611032  1.17714069  0.61114703  0.04966828  0.17948778
0.43923717
   0.33945589  0.75325382 -1.25269399  2.20402649  0.50326122
0.36486878]]
Intercepto: [-0.49647861]
```



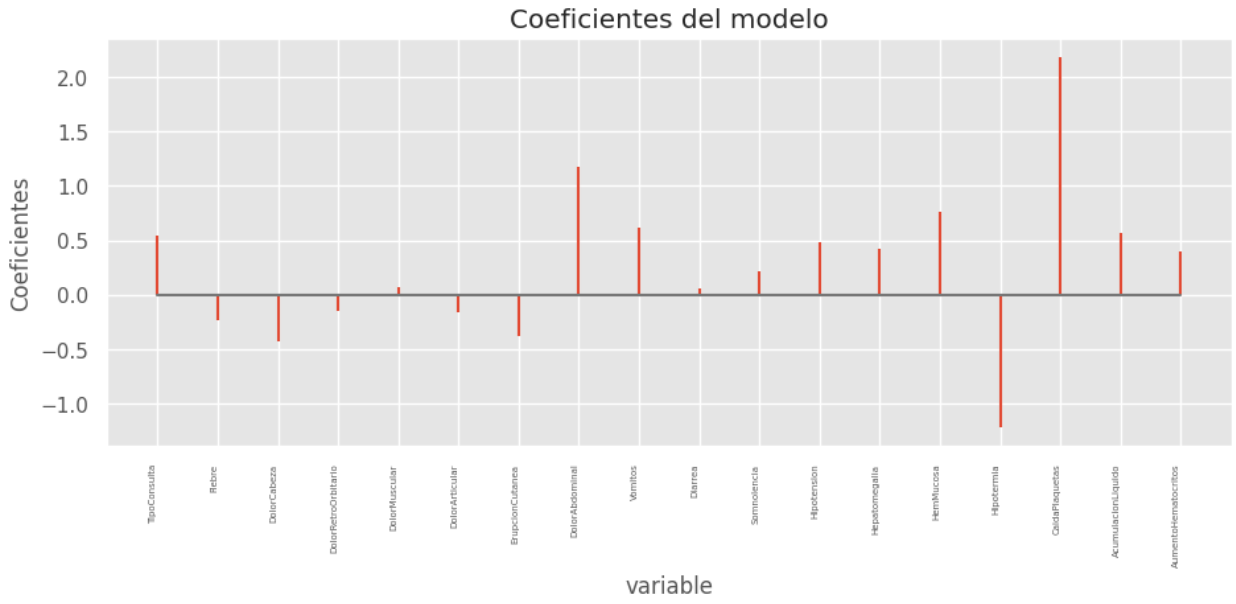
```
# Crear una lista de valores de alpha para la penalización Lasso
alphas = np.logspace(-4, 10, 100)
# Crear una lista para almacenar los coeficientes para cada valor de
alpha
coefs = []
# Iterar sobre los valores de alpha y ajustar el modelo Lasso
for alpha in alphas:
    lasso_model = Lasso(alpha=alpha)
    lasso_model.fit(X_train, y_train)
    coefs.append(lasso_model.coef_)

# Convertir la lista de coeficientes a un array NumPy
coefs = np.array(coefs)

# Graficar la penalización Lasso
plt.figure(figsize=(10, 6))
ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale('log')
plt.xlabel('Alpha (Parámetro de Penalización)')
plt.ylabel('Coeficientes')
plt.title('Gráfico de Penalización Lasso')
#plt.legend(X.columns)
plt.show()
```



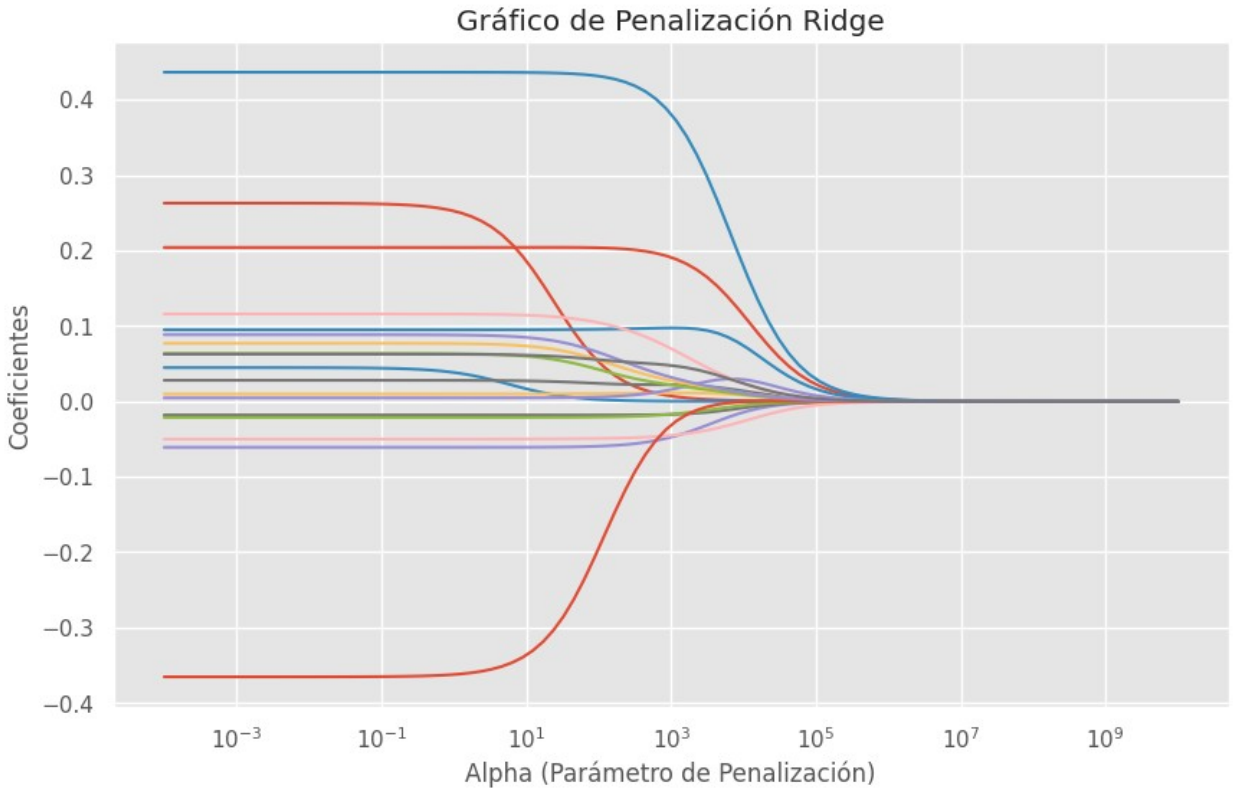
##Modelo LogisticRegression con regularización Logit-Ridge

```
# Ajustar modelo Logit-Ridge
ridge_model = LogisticRegression(C = 0.1, ### parametro de
penalización, valores menores a uno penalizacion alta
                                 class_weight= 'balanced',
                                 penalty= 'l2',  ## Elgir como
penalizar: l1, l2, elasticnet, None
                                 solver= 'liblinear', ### Algoritmo de
optimización
                                 random_state=42)
ridge_model.fit(X_train, y_train)
y_pred_ridge_model = ridge_model.predict(X_test)



print("=========================================================.")
ridge_model_precision_score = precision_score(y_test,
y_pred_ridge_model, average='macro')
print("Precisión del Modelo LogisticRegression con Ridge el set de
Pruebas: {:.4f}".format(ridge_model_precision_score))
ridge_model_recall_score = recall_score(y_test, y_pred_ridge_model,
average='macro')
print('Recall del Modelo LogisticRegression con con Ridge el set de
Pruebas: {:.4f}'.format(ridge_model_recall_score))
ridge_model_f1_score = f1_score(y_test, y_pred_ridge_model,
average='macro')
print('F1 Score del Modelo LogisticRegression con Ridge con el set de
Pruebas: {:.4f}'.format(ridge_model_f1_score))
print("=========================================================.")
scores =
cross_val_score(ridge_model,X_train,y_train,cv=5,scoring='roc_auc',n_j
obs=-1)
ridge_model_roc_auc = scores.mean()
print("AUC Cross Val Score del set de entrenamiento:
{:.4f}".format(ridge_model_roc_auc))
acc =
cross_val_score(ridge_model,X_train,y_train,cv=5,scoring='accuracy',n_
jobs=-1)
ridge_model_accv = acc.mean()
print("Accuracy Cross Val Score del Modelo LogisticRegression con
Ridge para el set de entrenamiento: {:.4f}".format(ridge_model_accv))
ridge_model_score_train = ridge_model.score(X_train, y_train)
print("Accuracy del Modelo LogisticRegression con Ridge del set de
entrenamiento set: {:.4f}".format(ridge_model_score_train))
ridge_model_score_test = ridge_model.score(X_test, y_test)
print("Accuracy del Modelo LogisticRegression con Ridge del set de
pruebas: {:.4f}".format(ridge_model_score_test))
```

```
============================================================.
Precisión del Modelo LogisticRegression con Ridge el set de Pruebas:
0.7434
Recall del Modelo LogisticRegression con con Ridge el set de Pruebas:
0.7648
F1 Score del Modelo LogisticRegression con Ridge con el set de
Pruebas: 0.7514
============================================================.
AUC Cross Val Score del set de entrenamiento: 0.8301
Accuracy Cross Val Score del Modelo LogisticRegression con Ridge para
el set de entrenamiento: 0.7834
Accuracy del Modelo LogisticRegression con Ridge del set de
entrenamiento set: 0.7835
Accuracy del Modelo LogisticRegression con Ridge del set de pruebas:
0.7833


# Coeficientes del modelo Ridge
#
================================================================================
========
coefficients = ridge_model.coef_
intercept = ridge_model.intercept_
print("")
print("Coeficientes:", coefficients)
print("Intercepto:", intercept)
print("")
dfDenge_coeficientes = pd.DataFrame({'predictor': X.columns, 'coef':
ridge_model.coef_[0]})

fig, ax = plt.subplots(figsize=(11, 4))
ax.stem(dfDenge_coeficientes.predictor, dfDenge_coeficientes.coef,
markerfmt=' ')
plt.xticks(rotation=90, ha='right', size=5)
ax.set_xlabel('variable')
ax.set_ylabel('Coeficientes')
ax.set_title('Coeficientes del modelo');


Coeficientes: [[ 0.5463972  -0.23306253 -0.42715123 -0.14659113
0.06568775 -0.15902555
  -0.37722438  1.16928313  0.61195134  0.06088231  0.2101375
0.48090201
   0.41656167  0.75791719 -1.21152098  2.18637327  0.56335092
0.40153449]]
Intercepto: [-0.28951663]
```




```
# Crear una lista de valores de alpha para la penalización Lasso
alphas = np.logspace(-4, 10, 100)
# Crear una lista para almacenar los coeficientes para cada valor de
alpha
coefs = []
# Iterar sobre los valores de alpha y ajustar el modelo Lasso
for alpha in alphas:
    ridge_model = Ridge(alpha=alpha)
    ridge_model.fit(X_train, y_train)
    coefs.append(ridge_model.coef_)

# Convertir la lista de coeficientes a un array NumPy
coefs = np.array(coefs)

# Graficar la penalización Lasso
plt.figure(figsize=(10, 6))
ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale('log')
plt.xlabel('Alpha (Parámetro de Penalización)')
plt.ylabel('Coeficientes')
plt.title('Gráfico de Penalización Ridge')
#plt.legend(X.columns)
plt.show()
```




```
# Ajustar modelo Logit-Enet
enet_model = LogisticRegression(
    C = 0.1,
    class_weight= 'balanced',
    penalty= 'elasticnet',
    solver= 'saga',
    random_state=42,
    l1_ratio=0.5
    )

enet_model.fit(X_train, y_train)
y_pred_enet_model = enet_model.predict(X_test)

print("============================================================.")
enet_model_precision_score = precision_score(y_test,
y_pred_enet_model, average='macro')
print("Precisión del Modelo LogisticRegression con Logit-Enet el set
de Pruebas: {:.4f}".format(enet_model_precision_score))
enet_model_recall_score = recall_score(y_test, y_pred_enet_model,
average='macro')
print('Recall del Modelo LogisticRegression con con Logit-Enet el set
de Pruebas: {:.4f}'.format(enet_model_recall_score))
enet_model_f1_score = f1_score(y_test, y_pred_enet_model,
average='macro')
```

```
print('F1 Score del Modelo LogisticRegression con Logit-Enet con el
set de Pruebas: {:.4f}'.format(enet_model_f1_score))
print("============================================================.")
scores =
cross_val_score(enet_model,X_train,y_train,cv=5,scoring='roc_auc',n_jo
bs=-1)
enet_model_roc_auc = scores.mean()
print("AUC Cross Val Score del set de entrenamiento:
{:.4f}".format(enet_model_roc_auc))
acc =
cross_val_score(enet_model,X_train,y_train,cv=5,scoring='accuracy',n_j
obs=-1)
enet_model_accv = acc.mean()
print("Accuracy Cross Val Score del Modelo LogisticRegression con
Logit-Enet para el set de entrenamiento:
{:.4f}".format(enet_model_accv))
enet_model_score_train = enet_model.score(X_train, y_train)
print("Accuracy del Modelo LogisticRegression con Logit-Enet del set
de entrenamiento set: {:.4f}".format(enet_model_score_train))
enet_model_score_test = enet_model.score(X_test, y_test)
print("Accuracy del Modelo LogisticRegression con Logit-Enet del set
de pruebas: {:.4f}".format(enet_model_score_test))
```

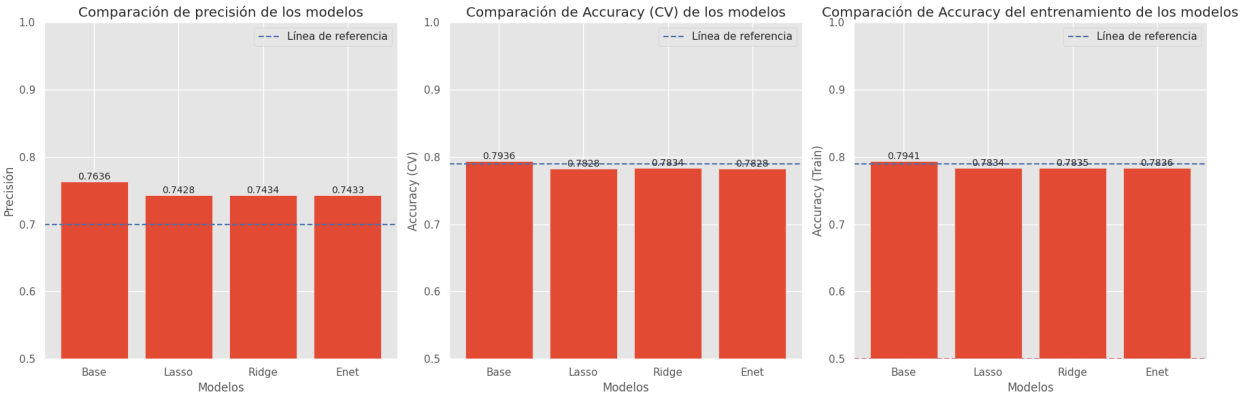
```
============================================================.
Precisión del Modelo LogisticRegression con Logit-Enet el set de
Pruebas: 0.7433
Recall del Modelo LogisticRegression con con Logit-Enet el set de
Pruebas: 0.7643
F1 Score del Modelo LogisticRegression con Logit-Enet con el set de
Pruebas: 0.7512
============================================================.
AUC Cross Val Score del set de entrenamiento: 0.8301
Accuracy Cross Val Score del Modelo LogisticRegression con Logit-Enet
para el set de entrenamiento: 0.7828
Accuracy del Modelo LogisticRegression con Logit-Enet del set de
entrenamiento set: 0.7836
Accuracy del Modelo LogisticRegression con Logit-Enet del set de
pruebas: 0.7833
```

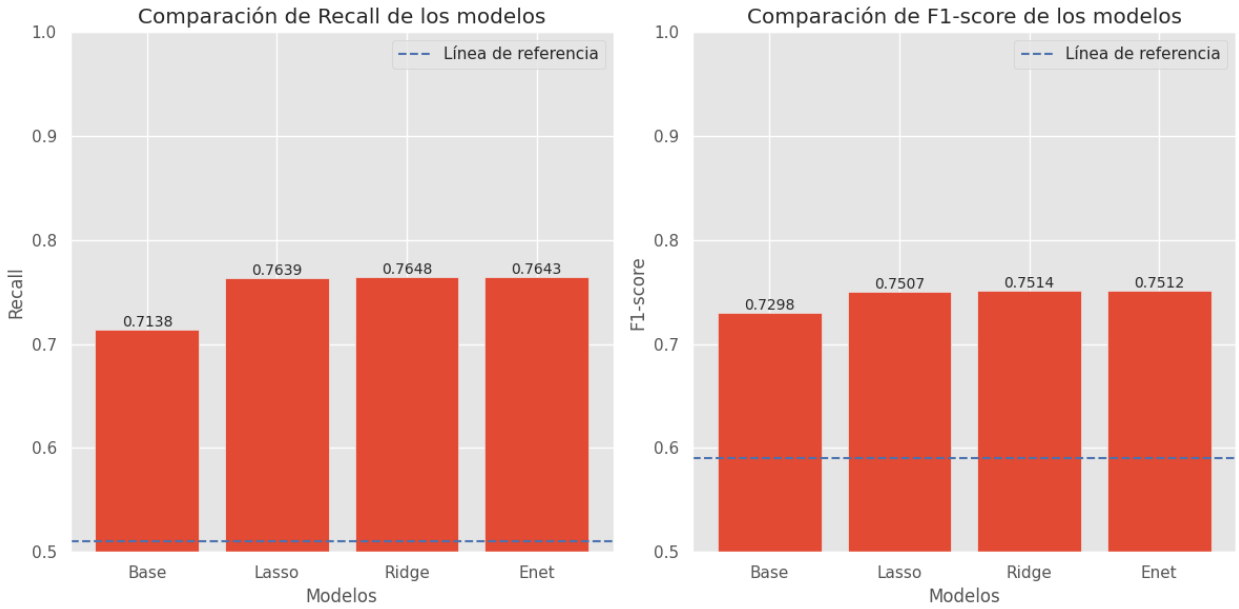
```
# Datos de recall para cada modelo
recall_scores = {
    'Base': logitreg_model_recall_score,
    'Lasso': lasso_model_recall_score,
    'Ridge': ridge_model_recall_score,
    'Enet': enet_model_recall_score,
}

# Datos de precisión para cada modelo
```

```
precision_scores = {
    'Base': logitreg_model_precision_score,
    'Lasso': lasso_model_precision_score,
    'Ridge': ridge_model_precision_score,
    'Enet': enet_model_precision_score,
}

# Datos de accuracy para cada modelo
accuracy_scores = {
    'Base': logitreg_model_accv,
    'Lasso': lasso_model_accv,
    'Ridge': ridge_model_accv,
    'Enet': enet_model_accv,
}

# Datos de accuracy para el entrenamiento de cada modelo
accuracy_scores_train = {
    'Base': logitreg_model_score_train,
    'Lasso': lasso_model_score_train,
    'Ridge': ridge_model_score_train,
    'Enet': enet_model_score_train,
}


# Crear una figura con subplots
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6))


# Gráfico 1: Precisión
models = list(precision_scores.keys())
scores = list(precision_scores.values())

axes[0].bar(models, scores)
axes[0].set_ylim(0.5, 1)
axes[0].set_xlabel('Modelos')
axes[0].set_ylabel('Precisión')
axes[0].set_title('Comparación de precisión de los modelos')
for i, score in enumerate(scores):
  axes[0].text(i, score, str(round(score, 4)), ha='center',
va='bottom')

axes[0].axhline(y=0.70, color='b', linestyle='--', label='Línea de
referencia')
axes[0].legend()

# Gráfico 2: Accuracy (Cross-Validation)
models = list(accuracy_scores.keys())
scores = list(accuracy_scores.values())

axes[1].bar(models, scores)
```

```
axes[1].set_ylim(0.5, 1)
axes[1].set_xlabel('Modelos')
axes[1].set_ylabel('Accuracy (CV)')
axes[1].set_title('Comparación de Accuracy (CV) de los modelos')
for i, score in enumerate(scores):
  axes[1].text(i, score, str(round(score, 4)), ha='center',
va='bottom')

axes[1].axhline(y=0.79, color='b', linestyle='--', label='Línea de
referencia')
axes[1].legend()
# Gráfico 3: Accuracy del entrenamiento
models = list(accuracy_scores_train.keys())
scores = list(accuracy_scores_train.values())

axes[2].bar(models, scores)
axes[2].set_ylim(0.5, 1)
axes[2].set_xlabel('Modelos')
axes[2].set_ylabel('Accuracy (Train)')
axes[2].set_title('Comparación de Accuracy del entrenamiento de los
modelos')
for i, score in enumerate(scores):
  axes[2].text(i, score, str(round(score, 4)), ha='center',
va='bottom')

axes[2].axhline(y=0.79, color='b', linestyle='--', label='Línea de
referencia')
axes[2].legend()
# Ajustar los subplots para evitar superposiciones
plt.tight_layout()

# Agregar una línea horizontal para comparar
plt.axhline(y=0.5, color='r', linestyle='--')
# Mostrar la figura
plt.show()

# Datos de recall para cada modelo
recall_scores = {
    'Base': logitreg_model_recall_score,
    'Lasso': lasso_model_recall_score,
    'Ridge': ridge_model_recall_score,
    'Enet': enet_model_recall_score,
}

# Datos de F1-score para cada modelo
f1_scores = {
    'Base': logitreg_model_f1_score,
    'Lasso': lasso_model_f1_score,
    'Ridge': ridge_model_f1_score,
    'Enet': enet_model_f1_score,
```

```
}

# Crear una figura con subplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6) )

# Gráfico 1: Recall
models = list(recall_scores.keys())
scores = list(recall_scores.values())

axes[0].bar(models, scores)
axes[0].set_ylim(0.5, 1)
axes[0].set_xlabel('Modelos')
axes[0].set_ylabel('Recall')
axes[0].set_title('Comparación de Recall de los modelos')
for i, score in enumerate(scores):
  axes[0].text(i, score, str(round(score, 4)), ha='center',
va='bottom')

axes[0].axhline(y=0.51, color='b', linestyle='--', label='Línea de
referencia')
axes[0].legend()
# Gráfico 2: F1-score
models = list(f1_scores.keys())
scores = list(f1_scores.values())

axes[1].bar(models, scores)
axes[1].set_ylim(0.5, 1)
axes[1].set_xlabel('Modelos')
axes[1].set_ylabel('F1-score')
axes[1].set_title('Comparación de F1-score de los modelos')
for i, score in enumerate(scores):
  axes[1].text(i, score, str(round(score, 4)), ha='center',
va='bottom')

axes[1].axhline(y=0.59, color='b', linestyle='--', label='Línea de
referencia')
axes[1].legend()
# Ajustar los subplots para evitar superposiciones
plt.tight_layout()

# Mostrar la figura
plt.show()
```





El estudio realizado analiza la capacidad predictiva de diferentes modelos de regresión logística para predecir la ocurrencia de eventos de dengue utilizando variables clínico-medicas. Se evaluaron modelos sin regularización y con regularización LASSO, Ridge y Elastic Net.

Obtenemos una precisión del modelo usando average='macro' dado que, el uso de macro-promedio es una buena medida para evaluar el rendimiento del modelo cuando se trabaja con un problema de clasificación con clases desbalanceadas ya que no tiene en cuenta el desbalanceo. La precisión del modelo en relación del set de pruebas con la predicción es de **0.7636** Indica que el modelo no tiene un buen desempeño, y la precisión en la predicción de la variable objetivo **"Evento"**' no es relativamente baja. El que alcanzó un mejor desempeño entre LASSO, Ridge y Elastic Net fue el modelo usando Ridge con una precesión de **0.7434**

En el contexto del modelo para predecir si el paciente es "Hospitalizado", un F1 Score superior a 0.7 sugiere que el modelo tiene un buen rendimiento general en la predicción de la variable objetivo en este caso, "Hospitalizado". El que alcanzó un mejor desempeño en el modelo usando Ridge con un valor de **0.7514** diferenciandose de los demás por milecimas. Aunque el F1 Score es bueno, siempre es posible buscar mejoras en el modelo, como la optimización de los hiperparámetros o la inclusión de nuevas variables predictoras.

En el caso de la predicción si el paciente es "Hospitalizado", un Recall del **0.7138** significa que el modelo es capaz de identificar correctamente el 76.38% de predecir los casos positivos de pacientes a ser hospitalizados, aunque es un buen indicador significa que se podrían estar pasando por alto casos positivos reales en 23.57%. El que alcanzó un mejor desempeño fue Ridge con un valor de **0.7648**

- Se revisa la proporción de los datos de los Eventos antes y después del split del dataset y se observa que se conserva la proporción.
- Los modelos de regresión logística no demostraron un buen desempeño menor que aceptable en la predicción de eventos de dengue, con valores de precisión, Recall y F1-score cercanos a 0.70 en la mayoría de los casos.

En conclusion, el uso del regularización con Ridge fue mejor en todos los casos y aunque el resultado es bastante bueno, pero se puede mejorar aún más para evitar falsos negativos y garantizar una mejor predicción de los casos de hospitalización, se tendria que realizar un mejor ajuste de los hiperparametros para encontrar el mejor resultado