**Universidad Técnica Nacional**

**Sede Ciudad Quesada, San Carlos**

**Ingeniería del Software**

**ISW 911: Minería de Datos**

**Proyecto #3: Algoritmos de Machine Learning**

**Integrantes:**

**José Alí Rivas Gómez**

**José Arturo Quirós Araya**

**Hanzel Quirós Beteta**

**Docente:**

**Freddy Gerardo Rocha Boza**

**Periodo 2023**

**1er Cuatrimestre**

**Tabla de Contenido**

[INTRODUCCIÓN 3](#_Toc133326779)

[OBJETIVO GENERAL 4](#_Toc133326780)

[OBJETIVOS ESPECÍFICOS 4](#_Toc133326781)

[DESCRIPCIÓN DE LA SOLUCIÓN 5](#_Toc133326782)

[Carga de datos estandarizados 5](#_Toc133326783)

[Correlacion de las variables con datos estandarizados: 6](#_Toc133326784)

[Creación de Cluster Jerárquico 6](#_Toc133326785)

[Uso de “K-Means” 7](#_Toc133326786)

[Regresión Lineal Simple 7](#_Toc133326787)

[Regresión Lineal Múltiple 10](#_Toc133326788)

[Regresión Logística Simple 14](#_Toc133326789)

[Regresión Logística Múltiple 18](#_Toc133326790)

[Árbol de Regresión 21](#_Toc133326791)

[Árbol de Clasificación 25](#_Toc133326792)

[Bosque Aleatorio de Regresion 28](#_Toc133326793)

[Bosque Aleatorio de Clasificación 33](#_Toc133326794)

[CONCLUSIONES 37](#_Toc133326795)

# **INTRODUCCIÓN**

El proceso encargado de realizar un análisis sobre los datos con la finalidad de encontrar patrones, anomalías, datos atípicos y otros resultados de interés se le conoce como “EDA” (Exploratory Data Analysis) que permite la interpretación datos por medio de gráficos.

Con modelos entrenados se puede predecir un comportamiento a futuro, usando algoritmos de métodos aprendizaje supervisado. En este apartado se ejecutarán una serie de procedimientos tales como; Regresión Lineal y Logística, (simple y múltiple en ambos casos), Árbol de Regresión y de Clasificación, y también Bosques Aleatorios de Regresión y Clasificación, permitan resultados precisos para favorecer el análisis de datos provenientes del dataset “Happiness Index”, con el objetivo de conseguir información relevante oculta a simple vista.

# **OBJETIVO GENERAL**

* Realizar un análisis completo de los datos ejecutando una serie de procedimientos que permitan una evaluación de los resultados de acuerdo a su comportamiento.

# **OBJETIVOS ESPECÍFICOS**

* Analizar un dataset con todo el proceso EDA, incluyendo de la limpieza de los datos y normalización
* Identificar la variable dependiente del dataset en caso de que esta exista.
* Aplicar un algoritmo apropiado que ejecute una clasificación y luego una predicción
* (Cluster Jerárquico, K-Means, regresión logística).
* Desarrollar otros algoritmos que muestren información relevante para el análisis.
* Documentar cada uno de los pasos realizados.
* Interpretar los resultados obtenidos.

# **DESCRIPCIÓN DE LA SOLUCIÓN**

Para el análisis se decidió utilizar un dataset que contiene información sobre los países y sus índices de felicidad. Después de verificar la integridad de los datos y su estructura, se consideró oportuno acceder a él desde un repositorio de GitHub y realizar así los procedimientos.

Se utilizó el lenguaje Python con la herramienta Colab de Google para ejecutar los algoritmos que permiten obtener los resultados del análisis.

Carga de Librerías:import pandas as pdi  
 mport numpy as np  
 import seaborn as sns  
 import matplotlib.pyplot as plt  
 %matplotlib inline  
 from matplotlib import style  
 from mpl\_toolkits.mplot3d import Axes3D

## Carga de datos estandarizados

url = ‘[https://github.com/ArturoQuiros/MineriaBI/blob/scripts/Proyecto%203/](https://github.com/ArturoQuiros/MineriaBI/blob/scripts/Proyecto%203/datos_estandarizados_lab5.xlsx?raw=true')

[datos\_estandarizados\_lab5.xlsx?raw=true'](https://github.com/ArturoQuiros/MineriaBI/blob/scripts/Proyecto%203/datos_estandarizados_lab5.xlsx?raw=true')

datos = pd.read\_excel(url, sheet\_name=0, decimal=',') # El primer sheet

datos = datos.set\_index('Country')

datos

Convertir a dataframe:

datos=pd.DataFrame(datos)

Obtener información detallada de los datos:

datos.info( )

Obtener estadísticas globales de los datos:

datos.describe( )

Contar los nulos por columna:

datos.isnull( ).sum( )

## Correlacion de las variables con datos estandarizados:

plt.figure(figsize=(16,10))  
 df\_corr=datos

sns.heatmap(df\_corr.corr(),annot=True, cmap="YlGnBu")

plt.title("Correlacion de las variables")

plt.show( )

Resultado de variables con mayor correlación:

"Phones (per 1000)" según "GDP ($ per cápita)" 0.85

"Infant mortality (per 1000 births)" según "Birthrate" 0.86

"Infant mortality (per 1000 births)" según "Agriculture" 0.72

"Infant mortality (per 1000 births)" según "Deathrate" 0.66

"Happiness Index" según "GDP ($ per cápita)" 0.60

"Happiness Ranking" según "Infant mortality (per 1000 births)" 0.68

"Happiness Ranking" según "Birthrate" 0.60

"Agriculture" según "Birthrate" 0.68

"Population" según "Area (sq. mi.)" 0.67

"Service" según "Phones (per 1000)" 0.63

## Creación de Cluster Jerárquico

Se crearon diferentes combinaciones en los parámetros para obtener una variedad de resultados, utilizando la estructura de la siguiente función:

from scipy.cluster.hierarchy import dendrogram, linkage

plt.figure(figsize=(30,12))

data = list(zip(datos['GDP ($ per cápita)'], datos['Phones (per 1000)']))

linkage\_data = linkage(data, method='ward', metric='euclidean')

dendrogram(linkage\_data)

plt.show()

Las combinaciones realizadas son las siguientes:

**-** "Phones (per 1000)" según "GDP ($ per cápita)"

- "Infant mortality (per 1000 births)" según "Birthrate"

- "Infant mortality (per 1000 births)" según "Agriculture"

- "Infant mortality (per 1000 births)" según "Deathrate"

- "Happiness Index" según "GDP ($ per cápita)"

- "Happiness Ranking" según "Infant mortality (per 1000 births)"

- "Happiness Ranking" según "Birthrate"

- "Agriculture" según "Birthrate"

- "Population" según "Area (sq. mi.)"

- "Service" según "Phones (per 1000)"

Estos parámetros se aplicaron nuevamente en el siguiente algoritmo, usando la herramienta “K-Means”.

## Uso de “K-Means”

Se aplicó la clusterización con K-Means, y también el algoritmo del método “Elbow” (codo) para evaluar la cantidad mínima necesaria de variables. En el siguiente ejemplo se ejecutó la clusterización, seguido del método “Elbow”: - ("Phones (per 1000)" según "GDP ($ per cápita)"):

from sklearn.cluster import KMeans

plt.figure(figsize=(15,10))

plt.scatter(datos['GDP ($ per cápita)'], datos['Phones (per 1000)'])

plt.xlabel('GDP ($ per cápita)')

plt.ylabel('Phones (per 1000)')

plt.show( )

data = list(zip(datos['GDP ($ per cápita)'], datos['Phones (per 1000)']))

inertias = [ ]

for i in range(1,11):

kmeans = KMeans(n\_clusters=i)

kmeans.fit(data)

inertias.append(kmeans.inertia\_)

plt.figure(figsize=(15,10))

plt.plot(range(1,11), inertias, marker='o')

plt.title('Elbow method')

plt.xlabel('Number of clusters')

plt.ylabel('Inertia')

plt.show()

Se presentan las variables con mayor correlación de los resultados obtenidos:

"Phones (per 1000)" según "GDP ($ per cápita)" 0.85

"Infant mortality (per 1000 births)" según "Birthrate" 0.86

"Infant mortality (per 1000 births)" según "Agriculture" 0.72

"Infant mortality (per 1000 births)" según "Deathrate" 0.66

"Happiness Index" según "GDP ($ per cápita)" 0.60

"Happiness Ranking" según "Infant mortality (per 1000 births)" 0.68

"Happiness Ranking" según "Birthrate" 0.60

"Agriculture" según "Birthrate" 0.68

"Population" según "Area (sq. mi.)" 0.67

"Service" según "Phones (per 1000)" 0.63

## Regresión Lineal Simple

Se realizó esta técnica por ser rápida y eficaz, para analizar las relaciones entre variables dependientes con otras independientes y obtener predicciones con sus datos.

Se va a predecir "Infant mortality (per 1000 births)" segun "Birthrate"

Tratamiento de datos

import pandas as pd

import numpy as np

Gráficos

import matplotlib.pyplot as plt

from matplotlib import style

import seaborn as sns

Preprocesado y modelado

from scipy.stats import pearsonr

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import r2\_score

from sklearn.metrics import mean\_squared\_error

import statsmodels.api as sm

import statsmodels.formula.api as smf

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')

Configuración warnings

import warnings

warnings.filterwarnings('ignore')

Algoritmo para obtener el gráfico de la distribución "Infant mortality (per 1000 births)" según "Birthrate"

fig, ax = plt.subplots(figsize=(15, 10))datos.plot(

x = 'Birthrate',

y = 'Infant mortality (per 1000 births)',

c = 'firebrick',

kind = "scatter",

ax = ax

)

ax.set\_title('Distribución de Birthrate y Infant mortality (per 1000 births)');

Correlación lineal entre las dos variables

corr\_test = pearsonr(x = datos['Birthrate'], y = datos['Infant mortality (per 1000 births)'])

print("Coeficiente de correlación de Pearson: ", corr\_test[0])

print("P-value: ", corr\_test[1]))

Coeficiente de correlación de Pearson:

0.857156713236685

1. value:

1.169392229892433e-69

División de los datos en train y test

X = datos[['Birthrate']]

y = datos['Infant mortality (per 1000 births)']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X.values.reshape(-1,1),

y.values.reshape(-1,1),

train\_size = 0.8,

random\_state = 1234,

shuffle = True

)

Creación del modelo utilizando matrices

\*A la matriz de predictores se le añadió una columna de “1s” para el intercept del modelo

X\_train = sm.add\_constant(X\_train, prepend=True)

modelo = sm.OLS(endog=y\_train, exog=X\_train,)

modelo = modelo.fit()

print(modelo.summary())

Intervalos de confianza para los coeficientes del modelo

modelo.conf\_int(alpha=0.05)

Resultado:

array([[-0.05781233, 0.09071995],  
 [ 0.78402999, 0.93391535]])

Predicciones con intervalo de confianza del 95%

predicciones = modelo.get\_prediction(exog = X\_train).summary\_frame(alpha=0.05)

predicciones.head(20)

Predicciones con intervalo de confianza del 95%

predicciones = modelo.get\_prediction(exog = X\_train).summary\_frame(alpha=0.05)

predicciones['x'] = X\_train[:, 1]

predicciones['y'] = y\_train

predicciones = predicciones.sort\_values('x')

Gráfico del modelo

fig, ax = plt.subplots(figsize=(15, 10))

ax.scatter(predicciones['x'], predicciones['y'], marker='o', color = "gray")

ax.plot(predicciones['x'], predicciones["mean"], linestyle='-', label="OLS")

ax.plot(predicciones['x'], predicciones["mean\_ci\_lower"], linestyle='--', color='red', label="95% CI")

ax.plot(predicciones['x'], predicciones["mean\_ci\_upper"], linestyle='--', color='red')

ax.fill\_between(predicciones['x'], predicciones["mean\_ci\_lower"], predicciones["mean\_ci\_upper"], alpha=0.1)

ax.legend( );

Error de test del modelo

X\_test = sm.add\_constant(X\_test, prepend=True)

predicciones = modelo.predict(exog = X\_test)

rmse = mean\_squared\_error(

y\_true = y\_test,

y\_pred = predicciones,

squared = False

)

print("")

print(f"El error (rmse) de test es: {rmse}")

Resultado

El error (rmse) de test es: 0.5176387872304924

En otros modelos se realizaron análisis con los siguientes parámetros:

"Population" según "Area (sq. mi.)"  
 "Phones (per 1000)" según "GDP ($ per cápita)"

## Regresión Lineal Múltiple

Se va a predecir "Happiness Index" segun las otras columnas

Se preparan los datos y el entorno

Tratamiento de datos

import pandas as pd

import numpy as np

Gráficos

import matplotlib.pyplot as plt

from matplotlib import style

import seaborn as sns

Preprocesado y modelado

from scipy.stats import pearsonr

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import r2\_score

from sklearn.metrics import mean\_squared\_error

import statsmodels.api as sm

import statsmodels.formula.api as smf

from statsmodels.stats.anova import anova\_lm

from scipy import stats

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')

Configuración warnings

import warnings

warnings.filterwarnings('ignore')

Correlación entre columnas numéricas

def tidy\_corr\_matrix(corr\_mat): #convierte matriz de correlación de pandas en formato tidy

corr\_mat = corr\_mat.stack().reset\_index()

corr\_mat.columns = ['variable\_1','variable\_2','r']

corr\_mat = corr\_mat.loc[corr\_mat['variable\_1'] != corr\_mat['variable\_2'], :]

corr\_mat['abs\_r'] = np.abs(corr\_mat['r'])

corr\_mat = corr\_mat.sort\_values('abs\_r', ascending=False)

return(corr\_mat)

corr\_matrix = datos.select\_dtypes(include=['float64', 'int']).corr(method='pearson')

tidy\_corr\_matrix(corr\_matrix).head(20)

Crear Heatmap matriz de las correlaciones

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(15,15))

sns.heatmap(  
 corr\_matrix,

annot = True,

cbar = False,

annot\_kws = {"size": 8},

vmin = -1,

vmax = 1,

center = 0,

cmap = sns.diverging\_palette(20, 220, n=200),

square = True,

ax = ax

)

ax.set\_xticklabels(

ax.get\_xticklabels(),

rotation = 45,

horizontalalignment = 'right',

)

ax.tick\_params(labelsize = 10)

Creación de gráfico de distribución para cada variable numérica, ajustando número de subplots en función del número de columnas

fig, axes = plt.subplots(nrows=5, ncols=5, figsize=(27, 15))

axes = axes.flat

columnas\_numeric = datos.select\_dtypes(include=['float64', 'int']).columns

for i, colum in enumerate(columnas\_numeric):

sns.histplot(

data = datos,

x = colum,

stat = "count",

kde = True,

color = (list(plt.rcParams['axes.prop\_cycle'])\*5)[i]["color"],

line\_kws= {'linewidth': 2},

alpha = 0.3,

ax = axes[i]

)

axes[i].set\_title(colum, fontsize = 10, fontweight = "bold")

axes[i].tick\_params(labelsize = 8)

axes[i].set\_xlabel("")

fig.tight\_layout()

plt.subplots\_adjust(top = 0.9)

fig.suptitle('Distribución variables numéricas', fontsize = 10, fontweight = "bold");

División de los datos en train y test

# X = datos[['GDP ($ per cápita)', 'Phones (per 1000)', 'Literacy (%)', 'Service', 'Climate', 'Coastline (coast/area ratio)', 'Net migration', 'Pop. Density (per sq. mi.)']]

X = datos.drop(columns = ['Happiness Index'])

y = datos['Happiness Index']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X,

y.values.reshape(-1,1),  
 train\_size = 0.8,

random\_state = 1234,

shuffle = True

)

Creación del modelo utilizando matrices, a la matriz de predictores se le añade una columna de 1s para el intercept del modelo, se le pide que muestre un resumen del análisis

X\_train = sm.add\_constant(X\_train, prepend=True)  
modelo = sm.OLS(endog=y\_train, exog=X\_train,)  
modelo = modelo.fit()  
print(modelo.summary())

Intervalos de confianza para los coeficientes del modelo

intervalos\_ci = modelo.conf\_int(alpha=0.05)  
intervalos\_ci.columns = ['2.5%', '97.5%']  
intervalos\_ci

Diagnóstico errores (residuos) de las predicciones de entrenamiento

y\_train = y\_train.flatten()  
 prediccion\_train = modelo.predict(exog = X\_train)  
 residuos\_train = prediccion\_train - y\_train

Se realiza la configuación de los gráficos

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(18, 16))  
 axes[0, 0].scatter(y\_train, prediccion\_train, edgecolors=(0, 0, 0), alpha = 0.4)  
 axes[0, 0].plot([y\_train.min(), y\_train.max()], [y\_train.min(), y\_train.max()],'k--', color = 'black', lw=2)  
 axes[0, 0].set\_title('Valor predicho vs valor real', fontsize = 10, fontweight = "bold")  
 axes[0, 0].set\_xlabel('Real')  
 axes[0, 0].set\_ylabel('Predicción')  
 axes[0, 0].tick\_params(labelsize = 7)  
 axes[0, 1].scatter(list(range(len(y\_train))), residuos\_train, edgecolors=(0, 0, 0), alpha = 0.4)  
 axes[0, 1].axhline(y = 0, linestyle = '--', color = 'black', lw=2)  
 axes[0, 1].set\_title('Residuos del modelo', fontsize = 10, fontweight = "bold")  
 axes[0, 1].set\_xlabel('id')  
 axes[0, 1].set\_ylabel('Residuo')  
 axes[0, 1].tick\_params(labelsize = 7)

sns.histplot(   
 data = residuos\_train,  
 stat = "density",  
 kde = True,  
 line\_kws= {'linewidth': 1},  
 color = "firebrick",  
 alpha = 0.3,  
 ax = axes[1, 0]  
 )

axes[1, 0].set\_title('Distribución residuos del modelo', fontsize = 10, fontweight = "bold")  
 axes[1, 0].set\_xlabel("Residuo")  
 axes[1, 0].tick\_params(labelsize = 7)  
 sm.qqplot(  
 residuos\_train,  
 fit = True,  
 line = 'q',  
 ax = axes[1, 1],   
 color = 'firebrick',  
 alpha = 0.4,  
 lw = 2  
 )

axes[1, 1].set\_title('Q-Q residuos del modelo', fontsize = 10, fontweight = "bold")  
 axes[1, 1].tick\_params(labelsize = 7)  
 axes[2, 0].scatter(prediccion\_train, residuos\_train, edgecolors=(0, 0, 0), alpha = 0.4)  
 axes[2, 0].axhline(y = 0, linestyle = '--', color = 'black', lw=2)  
 axes[2, 0].set\_title('Residuos del modelo vs predicción', fontsize = 10, fontweight = "bold")  
 axes[2, 0].set\_xlabel('Predicción')  
 axes[2, 0].set\_ylabel('Residuo')  
 axes[2, 0].tick\_params(labelsize = 7)

Se eliminan los axes vacíos

fig.delaxes(axes[2,1])  
 fig.tight\_layout()  
 plt.subplots\_adjust(top=0.9)  
 fig.suptitle('Diagnóstico residuos', fontsize = 12, fontweight = "bold");

Normalidad de los residuos Shapiro-Wilk test

shapiro\_test = stats.shapiro(residuos\_train)  
 shapiro\_test

Resultado:

ShapiroResult(statistic=0.8506371378898621, pvalue=1.2076745853351056e-12)

Normalidad de los residuos D'Agostino's K-squared test

k2, p\_value = stats.normaltest(residuos\_train)  
 print(f"Estadítico= {k2}, p-value = {p\_value}")

Resultado:

Estadítico= 111.97673657046448, p-value = 4.836827543602868e-25

Predicciones con intervalo de confianza

predicciones = modelo.get\_prediction(exog = X\_train).summary\_frame(alpha=0.05)  
 predicciones.head(20)

Error de test del modelo

X\_test = sm.add\_constant(X\_test, prepend=True)  
 predicciones = modelo.predict(exog = X\_test)  
 rmse = mean\_squared\_error(  
 y\_true = y\_test,  
 y\_pred = predicciones,  
 squared = False  
 )  
 print(f"El error (rmse) de test es: {rmse}")

Resultado  
 El error (rmse) de test es: 0.11444112951476224

## Regresión Logística Simple

Se va a predecir la nueva columna "Happy" segun "GDP ($ per capita)"

Tratamiento de datos

import pandas as pd  
 import numpy as np

Gráficos

import matplotlib.pyplot as plt  
 from matplotlib import style  
 import seaborn as sns

Preprocesado y modelado

from sklearn.linear\_model import LogisticRegression  
 from sklearn.model\_selection import train\_test\_split  
 from sklearn.metrics import accuracy\_score  
 import statsmodels.api as sm  
 import statsmodels.formula.api as smf  
 from statsmodels.stats.weightstats import ttest\_ind

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')

Configuración warnings

import warnings  
 warnings.filterwarnings('ignore')

Obtener valores mínimos de “Happiness Index”

datos["Happiness Index"].min()

Resultado  
 -2.753194297283495

Obtener valores máximos de “Happiness Index”

datos["Happiness Index"].max()

Resultado

1.981312292792781

Se copian los datos\_transformed y se les agrega la columna “Happy”, según el “Happines Index”

datos2 = datos.copy()  
 datos2["Happy"] = np.where(datos2["Happiness Index"] >= 0.2, 1, 0)  
 datos2

Número de observaciones por clase

datos2.Happy.value\_counts().sort\_index()

Resultado

0 118  
 1 119  
 Name: Happy, dtype: int64

Gráfico

fig, ax = plt.subplots(figsize=(15, 10))  
sns.violinplot(  
 x = 'Happy',  
 y = 'GDP ($ per cápita)',  
 data = datos2,  
 #color = "white",  
 ax = ax  
 )  
ax.set\_title('Distribución GDP ($ per cápita) por Happy');

División de los datos en train y test

X = datos2[['GDP ($ per cápita)']]  
y = datos2['Happy']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X.values.reshape(-1,1),  
 y.values.reshape(-1,1),  
 train\_size = 0.8,  
 random\_state = 1234,  
 shuffle = True  
 )

Creación del modelo utilizando matrices

A la matriz de predictores se le tiene que añadir una columna de 1s para el intercept del modelo

X\_train = sm.add\_constant(X\_train, prepend=True)  
modelo = sm.Logit(endog=y\_train, exog=X\_train,)  
modelo = modelo.fit()  
print(modelo.summary())

Intervalos de confianza para los coeficientes del modelo

intervalos\_ci = modelo.conf\_int(alpha=0.05)  
intervalos\_ci = pd.DataFrame(intervalos\_ci)  
intervalos\_ci.columns = ['2.5%', '97.5%']  
intervalos\_ci

Resultados

2.5% 97.5%  
0 -0.007155 0.841251  
1 1.343475 2.681321

Predicción de probabilidades

predicciones = modelo.predict(exog = X\_train)  
 predicciones[:4]

Resultado

array([0.18622342, 0.76943424, 0.28460102, 0.8907481 ])

Clasificación predicha

clasificacion = np.where(predicciones<0.5, 0, 1)  
clasificacion

Resultado

array([0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1,  
 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1,  
 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,  
 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1,  
 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0,  
 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,  
 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0,  
 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0])

Predicciones en todo el rango de X

grid\_X = np.linspace(  
 start = min(datos2['GDP ($ per cápita)']),  
 stop = max(datos2['GDP ($ per cápita)']),  
 num = 200  
 ).reshape(-1,1)  
grid\_X = sm.add\_constant(grid\_X, prepend=True)  
predicciones = modelo.predict(exog = grid\_X)

Gráfico del modelo

fig, ax = plt.subplots(figsize=(15, 10))  
ax.scatter(  
 X\_train[(y\_train == 1).flatten(), 1],  
 y\_train[(y\_train == 1).flatten()].flatten()  
)  
ax.scatter(  
 X\_train[(y\_train == 0).flatten(), 1],  
 y\_train[(y\_train == 0).flatten()].flatten()  
)  
ax.plot(grid\_X[:, 1], predicciones, color = "gray")  
ax.set\_title("Modelo regresión logística simple")  
ax.set\_ylabel("P(Happy = 1 | GDP ($ per cápita))")  
ax.set\_xlabel("GDP ($ per cápita)");

Test de precisión del modelo

X\_test = sm.add\_constant(X\_test, prepend=True)  
predicciones = modelo.predict(exog = X\_test)  
clasificacion = np.where(predicciones<0.5, 0, 1)  
accuracy = accuracy\_score(  
 y\_true = y\_test,  
 y\_pred = clasificacion,  
 normalize = True  
 )  
print(f"El accuracy de test es: {100\*accuracy}%")

Resultado

El accuracy de test es: 85.41666666666666%

## Regresión Logística Múltiple

Se realizará una predicción sobre la nueva columna "Happy" según las otras columnas (excepto "Happiness Index")

Tratamiento de datos

import pandas as pd  
import numpy as np

Gráficos

import matplotlib.pyplot as plt  
from matplotlib import style  
import seaborn as sns

Preprocesado y modelado

from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score  
import statsmodels.api as sm  
import statsmodels.formula.api as smf

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')

Configuración warnings

import warnings  
warnings.filterwarnings('ignore')  
print("Número de observaciones por clase")  
print(datos2['Happy'].value\_counts())  
print("")  
print("Porcentaje de observaciones por clase")  
print(100 \* datos2['Happy'].value\_counts(normalize=True))

Resultado

Número de observaciones por clase  
1 119  
0 118  
Name: Happy, dtype: int64

Porcentaje de observaciones por clase  
1 50.21097  
0 49.78903  
Name: Happy, dtype: float64

División de los datos en train y test

X = datos2.drop(columns = ['Happy', 'Happiness Index'])  
y = datos2['Happy']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X,  
 y.values.reshape(-1,1),  
 train\_size = 0.8,  
 random\_state = 1234,  
 shuffle = True  
 )

Creación del modelo utilizando matrices

A la matriz de predictores se le tiene que añadir una columna de 1s para el intercept del modelo

X\_train = sm.add\_constant(X\_train, prepend=True)  
modelo = sm.Logit(endog=y\_train, exog=X\_train,)  
modelo = modelo.fit(maxiter = 3)  
print(modelo.summary())

Predicciones con intervalo de confianza

predicciones = modelo.predict(exog = X\_train)

Clasificación predicha

clasificacion = np.where(predicciones<0.5, 0, 1)  
clasificacion

Resultado

array([0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1,  
 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0,  
 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,  
 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,  
 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1,  
 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0,  
 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1,  
 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,  
 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1])

Accuracy test del modelo

X\_test = sm.add\_constant(X\_test, prepend=True)  
predicciones = modelo.predict(exog = X\_test)  
clasificacion = np.where(predicciones<0.5, 0, 1)  
accuracy = accuracy\_score(  
 y\_true = y\_test,  
 y\_pred = clasificacion,  
 normalize = True  
 )  
print("")  
print(f"El accuracy de test es: {100\*accuracy}%")

Resultado

El accuracy de test es: 95.83333333333334%

Matriz de confusión de las predicciones de test

confusion\_matrix = pd.crosstab(  
 y\_test.ravel(),  
 clasificacion,  
 rownames=['Real'],  
 colnames=['Predicción']  
)  
confusion\_matrix

Resultado

Predicción 0 1  
Real

0 24 2  
 1 0 22

## Árbol de Regresión

Se va a predecir "Happiness Index" según las otras columnas

Tratamiento de datos

import numpy as np  
import pandas as pd

Gráficos

import matplotlib.pyplot as plt

Preprocesado y modelado

from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeRegressor  
from sklearn.tree import plot\_tree  
from sklearn.tree import export\_graphviz  
from sklearn.tree import export\_text  
from sklearn.model\_selection import GridSearchCV  
from sklearn.metrics import mean\_squared\_error

Configuración warnings

import warnings  
warnings.filterwarnings('once')

División de los datos en train y test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 datos.drop(columns = "Happiness Index"),  
 datos['Happiness Index'],  
 random\_state = 123  
 )

Creación del modelo

modelo = DecisionTreeRegressor(  
 max\_depth = 3,  
 random\_state = 123  
 )

Entrenamiento del modelo

modelo.fit(X\_train, y\_train)

Resultado

DecisionTreeRegressor  
DecisionTreeRegressor(max\_depth=3, random\_state=123)

Estructura del árbol creado

fig, ax = plt.subplots(figsize=(15, 10))  
print(f"Profundidad del árbol: {modelo.get\_depth()}")  
print(f"Número de nodos terminales: {modelo.get\_n\_leaves()}")  
plot = plot\_tree(  
 decision\_tree = modelo,  
 feature\_names = datos.drop(columns = "Happiness Index").columns,  
 class\_names = 'Happiness Index',  
 filled = True,  
 impurity = False,  
 fontsize = 10,  
 precision = 2,  
 ax = ax  
 )

Dimensiones del árbol

Profundidad del árbol: 3  
Número de nodos terminales: 8

Determinación de los predictores más importantes en el análisis

importancia\_predictores = pd.DataFrame(  
 {'predictor': datos.drop(columns = "Happiness Index").columns,  
 'importancia': modelo.feature\_importances\_}  
 )  
print("Importancia de los predictores en el modelo")  
print("-------------------------------------------")  
importancia\_predictores.sort\_values('importancia', ascending=False)

Resultado

Importancia de los predictores en el modelo  
-------------------------------------------

predictor importancia  
19 Happiness Ranking 1.0  
1 Population 0.0  
18 Service 0.0  
17 Industry 0.0  
16 Agriculture 0.0  
15 Deathrate 0.0  
14 Birthrate 0.0  
13 Climate 0.0  
12 Other (%) 0.0  
11 Crops (%) 0.0  
0 Region 0.0  
9 Phones (per 1000) 0.0  
8 Literacy (%) 0.0  
7 GDP ($ per cápita) 0.0  
6 Infant mortality (per 1000 births) 0.0  
5 Net migration 0.0  
4 Coastline (coast/area ratio) 0.0  
3 Pop. Density (per sq. mi.) 0.0  
2 Area (sq. mi.) 0.0  
10 Arable (%) 0.0

Pruning (const complexity pruning) por validación cruzada

Valores de ccp\_alpha evaluados

param\_grid = {'ccp\_alpha':np.linspace(0, 80, 20)}

Búsqueda por validación cruzada

El árbol se crece al máximo posible para luego aplicar el pruning

grid = GridSearchCV(  
 estimator = DecisionTreeRegressor(  
 max\_depth = None,  
 min\_samples\_split = 2,  
 min\_samples\_leaf = 1,  
 random\_state = 123  
 ),  
 param\_grid = param\_grid,  
 cv = 10,  
 refit = True,  
 return\_train\_score = True  
 )  
grid.fit(X\_train, y\_train)  
fig, ax = plt.subplots(figsize=(15, 10))  
scores = pd.DataFrame(grid.cv\_results\_)  
scores.plot(x='param\_ccp\_alpha', y='mean\_train\_score', yerr='std\_train\_score', ax=ax)  
scores.plot(x='param\_ccp\_alpha', y='mean\_test\_score', yerr='std\_test\_score', ax=ax)  
ax.set\_title("Error de validacion cruzada vs hiperparámetro ccp\_alpha");

Mejor valor ccp\_alpha encontrado

grid.best\_params\_

Resultado

{'ccp\_alpha': 0.0}

Estructura del árbol final

modelo\_final = grid.best\_estimator\_  
print(f"Profundidad del árbol: {modelo\_final.get\_depth()}")  
print(f"Número de nodos terminales: {modelo\_final.get\_n\_leaves()}")  
fig, ax = plt.subplots(figsize=(15, 10))  
plot = plot\_tree(  
 decision\_tree = modelo\_final,  
 feature\_names = datos.drop(columns = "Happiness Index").columns,  
 class\_names = 'Happiness Index',  
 filled = True,  
 impurity = False,  
 ax = ax  
)

Dimensiones del árbol

Profundidad del árbol: 17  
Número de nodos terminales: 126

Error de test del modelo inicial

predicciones = modelo.predict(X = X\_test)  
rmse = mean\_squared\_error(  
 y\_true = y\_test,  
 y\_pred = predicciones,  
 squared = False  
 )  
print(f"El error (rmse) de test es: {rmse}")

Resultado

El error (rmse) de test es: 0.1274498442783775

Error de test del modelo final (tras aplicar pruning)

predicciones = modelo\_final.predict(X = X\_test)  
rmse = mean\_squared\_error(  
 y\_true = y\_test,  
 y\_pred = predicciones,  
 squared = False  
 )  
print(f"El error (rmse) de test es: {rmse}")

Resultado

El error (rmse) de test es: 0.051340362310297524

## Árbol de Clasificación

Se va a predecir la nueva columna "Happy" según las otras columnas (excepto "Happiness Index")

Tratamiento de datos

import numpy as np  
import pandas as pd  
import statsmodels.api as sm

Gráficos

import matplotlib.pyplot as plt

Preprocesado y modelado

from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.tree import plot\_tree  
from sklearn.tree import export\_graphviz  
from sklearn.tree import export\_text  
from sklearn.model\_selection import GridSearchCV  
from sklearn.compose import ColumnTransformer  
from sklearn.preprocessing import OneHotEncoder  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import confusion\_matrix

Configuración warnings

import warnings  
warnings.filterwarnings('once')

División de los datos en train y test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 datos2.drop(columns = ['Happy', 'Happiness Index']),  
 datos2['Happy'],  
 random\_state = 123  
)

One-hot-encoding de las variables categóricas

Se identifica el nombre de las columnas numéricas y categóricas

cat\_cols = X\_train.select\_dtypes(include=['object', 'category']).columns.to\_list()  
numeric\_cols = X\_train.select\_dtypes(include=['float64', 'int']).columns.to\_list()

Se aplica one-hot-encoding solo a las columnas categóricas

preprocessor = ColumnTransformer(  
 [('onehot', OneHotEncoder(handle\_unknown='ignore'), cat\_cols)],  
 remainder='passthrough'  
 )

Una vez que se ha definido el objeto ColumnTransformer, con el método fit(), se aprenden las

transformaciones con los datos de entrenamiento y se aplican a los dos conjuntos con transform().

Ambas operaciones a la vez con fit\_transform().

X\_train\_prep = preprocessor.fit\_transform(X\_train)  
X\_test\_prep = preprocessor.transform(X\_test)

Creación del modelo

modelo = DecisionTreeClassifier(  
 max\_depth = 5,  
 criterion = 'gini',  
 random\_state = 123  
)

Entrenamiento del modelo

modelo.fit(X\_train\_prep, y\_train)

Estructura del árbol creado

fig, ax = plt.subplots(figsize=(15, 10))  
print(f"Profundidad del árbol: {modelo.get\_depth()}")  
print(f"Número de nodos terminales: {modelo.get\_n\_leaves()}")  
plot = plot\_tree(  
 decision\_tree = modelo,  
 feature\_names = labels.tolist(),  
 class\_names = 'Happy',  
 filled = True,  
 impurity = False,  
 fontsize = 7,  
 ax = ax  
 )

Error de test del modelo

predicciones = modelo.predict(X = X\_test\_prep,)  
print("Matriz de confusión")  
print("-------------------")  
confusion\_matrix(  
 y\_true = y\_test,  
 y\_pred = predicciones  
)

Matriz de confusión

array([[22, 0],  
 [ 0, 38]])

Accuracy = accuracy\_score(  
 y\_true = y\_test,  
 y\_pred = predicciones,  
 normalize = True  
 )  
print(f"El accuracy de test es: {100 \* accuracy} %")

Resultado

El accuracy de test es: 100.0 %

Post pruning (const complexity pruning) por validación cruzada

Valores de ccp\_alpha evaluados

param\_grid = {'ccp\_alpha':np.linspace(0, 5, 10)}

Búsqueda por validación cruzada

grid = GridSearchCV(  
 # El árbol se crece al máximo posible antes de aplicar el pruning  
 estimator = DecisionTreeClassifier(  
 max\_depth = None,  
 min\_samples\_split = 2,  
 min\_samples\_leaf = 1,  
 random\_state = 123  
 ),  
 param\_grid = param\_grid,  
 scoring = 'accuracy',  
 cv = 10,  
 refit = True,  
 return\_train\_score = True  
 )

grid.fit(X\_train\_prep, y\_train)  
fig, ax = plt.subplots(figsize=(15, 10))  
scores = pd.DataFrame(grid.cv\_results\_)  
scores.plot(x='param\_ccp\_alpha', y='mean\_train\_score', yerr='std\_train\_score', ax=ax)  
scores.plot(x='param\_ccp\_alpha', y='mean\_test\_score', yerr='std\_test\_score', ax=ax)  
ax.set\_title("Error de validacion cruzada vs hiperparámetro ccp\_alpha");

Estructura del árbol final

modelo\_final = grid.best\_estimator\_  
print(f"Profundidad del árbol: {modelo\_final.get\_depth()}")  
print(f"Número de nodos terminales: {modelo\_final.get\_n\_leaves()}")

Dimensiones del Árbol

Profundidad del árbol: 1  
Número de nodos terminales: 2

Error de test del modelo final

predicciones = modelo\_final.predict(X = X\_test\_prep)  
accuracy = accuracy\_score(  
 y\_true = y\_test,  
 y\_pred = predicciones,  
 normalize = True  
 )  
print(f"El accuracy de test es: {100 \* accuracy} %")

Resultado

El accuracy de test es: 100.0 %

## Bosque Aleatorio de Regresion

Se va a predecir "Happiness Index" segun las otras columnas

Tratamiento de datos

import numpy as np  
import pandas as pd

Gráficos

import matplotlib.pyplot as plt

Preprocesado y modelado

from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import RepeatedKFold  
from sklearn.model\_selection import GridSearchCV  
from sklearn.model\_selection import ParameterGrid  
from sklearn.inspection import permutation\_importance  
import multiprocessing

Configuración warnings

import warnings  
warnings.filterwarnings('once')

División de los datos en train y test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
datos.drop(columns = ["Happiness Index"]),  
datos['Happiness Index'],  
random\_state = 123  
)

Creación del modelo

modelo = RandomForestRegressor(  
 n\_estimators = 500,  
 criterion = 'friedman\_mse',  
 max\_depth = None,  
 max\_features = 'auto',  
 oob\_score = False,  
 n\_jobs = -1,  
 random\_state = 123  
 )

Entrenamiento del modelo

modelo.fit(X\_train, y\_train)

RandomForestRegressor(criterion='friedman\_mse', max\_features='auto',  
 n\_estimators=500, n\_jobs=-1, random\_state=123)

Error de test del modelo inicial

predicciones = modelo.predict(X = X\_test)  
rmse = mean\_squared\_error(  
 y\_true = y\_test,  
 y\_pred = predicciones,  
 squared = False  
 )  
print(f"El error (rmse) de test es: {rmse}")

Resultado

El error (rmse) de test es: 0.04092078371067556

Script de Validación empleando el Out-of-Bag error

train\_scores = [ ]

oob\_scores = [ ]

Valores evaluados

estimator\_range = range(1, 500, 5)

Bucle para entrenar un modelo con cada valor de n\_estimators y extraer su error de entrenamiento y de Out-of-Bag.

for n\_estimators in estimator\_range:  
 modelo = RandomForestRegressor(  
 n\_estimators = n\_estimators,  
 criterion = 'friedman\_mse',  
 max\_depth = None,  
 max\_features = 'auto',  
 oob\_score = True,  
 n\_jobs = -1,  
 random\_state = 123  
 )  
 modelo.fit(X\_train, y\_train)  
 train\_scores.append(modelo.score(X\_train, y\_train))  
 oob\_scores.append(modelo.oob\_score\_)

Gráfico con la evolución de los errores

fig, ax = plt.subplots(figsize=(15, 10))  
ax.plot(estimator\_range, train\_scores, label="train scores")  
ax.plot(estimator\_range, oob\_scores, label="out-of-bag scores")  
ax.plot(estimator\_range[np.argmax(oob\_scores)], max(oob\_scores),  
 marker='o', color = "red", label="max score")  
ax.set\_ylabel("R^2")  
ax.set\_xlabel("n\_estimators")  
ax.set\_title("Evolución del out-of-bag-error vs número árboles")  
plt.legend();  
print(f"Valor óptimo de n\_estimators: {estimator\_range[np.argmax(oob\_scores)]}")

Validación empleando el Out-of-Bag error

train\_scores = [ ]  
oob\_scores = [ ]

Valores evaluados

max\_features\_range = range(1, X\_train.shape[1] + 1, 1)

Bucle para entrenar un modelo con cada valor de max\_features y extraer su error de entrenamiento y de Out-of-Bag.

for max\_features in max\_features\_range:  
 modelo = RandomForestRegressor(  
 n\_estimators = 471,  
 criterion = 'friedman\_mse',  
 max\_depth = None,  
 max\_features = max\_features,  
 oob\_score = True,  
 n\_jobs = -1,  
 random\_state = 123  
 )  
 modelo.fit(X\_train, y\_train)  
 train\_scores.append(modelo.score(X\_train, y\_train))  
 oob\_scores.append(modelo.oob\_score\_)

Gráfico con la evolución de los errores

fig, ax = plt.subplots(figsize=(15, 10))  
ax.plot(max\_features\_range, train\_scores, label="train scores")  
ax.plot(max\_features\_range, oob\_scores, label="out-of-bag scores")  
ax.plot(max\_features\_range[np.argmax(oob\_scores)], max(oob\_scores),  
 marker='o', color = "red")  
ax.set\_ylabel("R^2")  
ax.set\_xlabel("max\_features")  
ax.set\_title("Evolución del out-of-bag-error vs número de predictores")  
plt.legend();  
print(f"Valor óptimo de max\_features: {max\_features\_range[np.argmax(oob\_scores)]}")

Resultado

Valor óptimo de max\_features: 20

División de los datos en train y test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 datos.drop(columns = "Happiness Index"),  
 datos['Happiness Index'],  
 random\_state = 123  
 )

Creación del modelo

modelo = RandomForestRegressor(  
 n\_estimators = 471,  
 criterion = 'friedman\_mse',  
 max\_depth = None,  
 max\_features = 20,  
 oob\_score = False,  
 n\_jobs = -1,  
 random\_state = 123  
 )

Entrenamiento del modelo

modelo.fit(X\_train, y\_train)

RandomForestRegressor(criterion='friedman\_mse', max\_features=20,  
 n\_estimators=471, n\_jobs=-1, random\_state=123)

Error de test del modelo final

predicciones = modelo.predict(X = X\_test)  
rmse = mean\_squared\_error(  
 y\_true = y\_test,  
 y\_pred = predicciones,  
 squared = False  
 )  
print(f"El error (rmse) de test es: {rmse}")

Resultado

El error (rmse) de test es: 0.04079805918451147

Gráfico

fig, ax = plt.subplots(figsize=(10, 12))  
df\_importancia = df\_importancia.sort\_values('importances\_mean', ascending=True)  
ax.barh(  
 df\_importancia['feature'],  
 df\_importancia['importances\_mean'],  
 xerr=df\_importancia['importances\_std'],  
 align='center',  
 alpha=0  
)  
ax.plot(  
 df\_importancia['importances\_mean'],  
 df\_importancia['feature'],  
 marker="D",  
 linestyle="",  
 alpha=0.8,  
 color="r"  
)  
ax.set\_title('Importancia de los predictores (train)')  
ax.set\_xlabel('Incremento del error tras la permutación');

## Bosque Aleatorio de Clasificación

Se realiza una predicción usando la nueva columna "Happy" según las otras columnas

Tratamiento de datos

import numpy as np  
import pandas as pd  
import statsmodels.api as sm

Gráficos

import matplotlib.pyplot as plt

Preprocesado y modelado

from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import ConfusionMatrixDisplay  
from sklearn.metrics import classification\_report  
from sklearn.compose import ColumnTransformer  
from sklearn.preprocessing import OneHotEncoder  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import RepeatedKFold  
from sklearn.model\_selection import GridSearchCV  
from sklearn.model\_selection import ParameterGrid  
from sklearn.inspection import permutation\_importance  
import multiprocessing

Configuración warnings

import warnings  
warnings.filterwarnings('once')

División de los datos en train y test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
datos2.drop(columns = ['Happy', 'Happiness Index']),  
datos2['Happy'],  
random\_state = 123  
)

One-hot-encoding de las variables categóricas

Se identifica el nobre de las columnas numéricas y categóricas

cat\_cols = X\_train.select\_dtypes(include=['object', 'category']).columns.to\_list()  
numeric\_cols = X\_train.select\_dtypes(include=['float64', 'int']).columns.to\_list()

Se aplica one-hot-encoding solo a las columnas categóricas

preprocessor = ColumnTransformer(  
 [('onehot', OneHotEncoder(handle\_unknown='ignore'), cat\_cols)],  
 remainder='passthrough'  
 )

Una vez que se ha definido el objeto ColumnTransformer, con el método fit() se aprenden las transformaciones con los datos de entrenamiento y se aplican a los dos conjuntos con transform().   
Ambas operaciones a la vez con fit\_transform().

X\_train\_prep = preprocessor.fit\_transform(X\_train)  
X\_test\_prep = preprocessor.transform(X\_test)

Convertir el output del ColumnTransformer en dataframe y añadir nombre columnas

labels = np.concatenate([numeric\_cols, cat\_cols])

Conversión a dataframe

X\_train\_prep = pd.DataFrame(X\_train\_prep, columns=labels)  
X\_test\_prep = pd.DataFrame(X\_test\_prep, columns=labels)  
X\_train\_prep.info()  
modelo = RandomForestClassifier(  
 oob\_score = True,  
 n\_jobs = -1,  
 random\_state = 123,  
 n\_estimators = 471,  
 max\_depth = 5,  
 criterion = 'gini',  
 max\_features = 20  
 )  
modelo.fit(X\_train\_prep, y\_train)

RandomForestClassifier(max\_depth=5, max\_features=20, n\_estimators=471,  
 n\_jobs=-1, oob\_score=True, random\_state=123)

Error de test del modelo final

predicciones = modelo.predict(X = X\_test\_prep)  
predicciones[:40]

Resultado

array([1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,  
 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1])

mat\_confusion = confusion\_matrix(  
 y\_true = y\_test,  
 y\_pred = predicciones  
 )

accuracy = accuracy\_score(  
 y\_true = y\_test,  
 y\_pred = predicciones,  
 normalize = True  
 )

print("Matriz de confusión")  
print("-------------------")  
print(mat\_confusion)  
print("")  
print(f"El accuracy de test es: {100 \* accuracy} %")

Resultado

Matriz de confusión  
[[22 0]  
[ 0 38]]

El accuracy de test es: 100.0 %

print(  
 classification\_report(  
 y\_true = y\_test,  
 y\_pred = predicciones  
 )  
)

precision recall f1-score support  
 0 1.00 1.00 1.00 22  
 1 1.00 1.00 1.00 38  
 accuracy 1.00 60  
 macro avg 1.00 1.00 1.00 60  
weighted avg 1.00 1.00 1.00 60

Gráfico

fig, ax = plt.subplots(figsize=(10, 12))  
df\_importancia = df\_importancia.sort\_values('importances\_mean', ascending=True)  
ax.barh(  
 df\_importancia['feature'],  
 df\_importancia['importances\_mean'],  
 xerr=df\_importancia['importances\_std'],  
 align='center',  
 alpha=0  
)  
ax.plot(  
 df\_importancia['importances\_mean'],  
 df\_importancia['feature'],  
 marker="D",  
 linestyle="",  
 alpha=0.8,  
 color="r"  
)  
ax.set\_title('Importancia de los predictores (train)')  
ax.set\_xlabel('Incremento del error tras la permutación');

# **CONCLUSIONES**

* Se obtuvo una alta correlación entre las variables "Infant mortality" y "Birthrate" de un 0.86, la más alta de todas.
* "Phones” y "GDP" tienen la segunda correlación más alta, 0.85.
* Se determinó utilizar las variables dependientes “Infant mortality”, “Population”, “Phones”, “Happiness Index” y “Happy”.
* Los algoritmos del Cluster Jerárquico, K-Means y Regresión Logística se ejecutaron satisfactoriamente, revelando información vital para decidir los parámetros en algoritmos posteriores.
* Los algoritmos que tienen una mayor exactitud en las predicciones son Bosques Aleatorios de Clasificación y Regresión, obteniendo resultados de un 100% y error (rmse) de 0.04. Aunque los Árboles de Clasificación y Regresión también dieron muy buenos resultados, 100% y error (rmse) de 0.05