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

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/
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
```

P-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, 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