

✓ Reto | Mercadotecnia Telefónica con Aprendizaje Supervisado

✓ Introducción

El telemarketing ha sido utilizado por empresas para comunicarse directamente con clientes potenciales. Con el apoyo de la inteligencia artificial, su impacto ha mejorado significativamente.

Un banco desea evaluar el éxito de su programa de telemarketing para promocionar un plan de inversión a largo plazo, utilizando aprendizaje supervisado. Este análisis busca identificar las características de los clientes más propensos a adquirir dicho plan.

Objetivo

Desarrollar un modelo de aprendizaje supervisado que prediga si un cliente adquirirá un plan de inversión bancaria tras una entrevista telefónica.

✓ Librerías

```
import numpy as np
import pandas as pd

import plotly.graph_objects as go
import plotly.subplots as sp

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import (
    MinMaxScaler,
    LabelEncoder,
)

from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, learning_curve
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

▼ Funciones Auxiliares

```
def signed_sqrt(x):
    return np.sign(x) * np.sqrt(np.abs(x))

class SafeTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, method="auto"):
        self.method = method

    def fit(self, X, y=None):
        self.method_ = (
            "signed_sqrt"
            if np.any(X < 0)
            else "log1p" if self.method == "auto" else self.method
        )
        return self

    def transform(self, X):
        if self.method_ == "log1p":
```

```

        return np.log1p(X)
    elif self.method_ == "signed_sqrt":
        return signed_sqrt(X)
    else:
        raise ValueError("Método desconocido en SafeTransformer.")

def process_data(data, original_numeric_columns, skew_threshold=0.5, std_threshold=0.5):
    data_clean = data.replace([np.inf, -np.inf], np.nan).fillna(0)
    skewness = data_clean[original_numeric_columns].skew()
    std = data_clean[original_numeric_columns].std()

    to_transform, to_scale, to_transform_and_scale = [], [], []

    for col in original_numeric_columns:
        col_skew, col_std = skewness[col], std[col]
        if abs(col_skew) > skew_threshold and col_std > std_threshold:
            to_transform_and_scale.append(col)
        elif abs(col_skew) > skew_threshold:
            to_transform.append(col)
        elif col_std > std_threshold:
            to_scale.append(col)

    scaler = MinMaxScaler()
    transformers = []

    if to_transform:
        transformers.append(("transform_only", SafeTransformer(), to_transform))
    if to_scale:
        transformers.append(("scale_only", scaler, to_scale))
    if to_transform_and_scale:
        transformers.append(
            (
                "transform_and_scale",
                Pipeline([("safe_transform", SafeTransformer()), ("scaler", scaler)]),
                to_transform_and_scale,
            )
        )

    preprocessor = ColumnTransformer(transformers=transformers, remainder="passthru")
    X_processed_array = preprocessor.fit_transform(data_clean)

    transformed_columns = to_transform + to_scale + to_transform_and_scale
    passthrough_columns = [
        col for col in data.columns if col not in transformed_columns
    ]

```

```

]
final_columns = transformed_columns + passthrough_columns

X_processed_df = pd.DataFrame(
    X_processed_array, columns=final_columns, index=data.index
)

summary = pd.DataFrame(
    {
        "Variable": transformed_columns,
        "Acción": (
            ["Transformar"] * len(to_transform)
            + ["Escalar"] * len(to_scale)
            + ["Transformar + Escalar"] * len(to_transform_and_scale)
        ),
    }
)

return X_processed_df, summary

def plot_hist(original_data, transformed_data, columns):
    n_vars = len(columns)
    fig = sp.make_subplots(
        rows=n_vars,
        cols=2,
        subplot_titles=[f"{col} - Original" for col in columns]
        + [f"{col} - Transformado" for col in columns],
        vertical_spacing=0.08,
    )

    for idx, col in enumerate(columns):
        fig.add_trace(
            go.Histogram(x=original_data[col], opacity=0.7), row=idx + 1, col=1
        )
        fig.add_trace(
            go.Histogram(x=transformed_data[col], opacity=0.7), row=idx + 1, col=2
        )

    fig.update_layout(
        height=300 * n_vars,
        width=800,
        title_text="Distribuciones Original vs Transformado (Grid 2x2)",
        showlegend=False,
        template="plotly_white",
    )

```

```

    )

fig.show()

def plot_learning_curves(
    estimator, X, y, cv=3, scoring="accuracy", train_sizes=np.linspace(0.1, 1.0, 10),
):
    train_sizes, train_scores, val_scores = learning_curve(
        estimator,
        X,
        y,
        cv=cv,
        scoring=scoring,
        train_sizes=train_sizes,
        n_jobs=-1,
        shuffle=True,
        random_state=42,
    )

    fig = go.Figure()

    fig.add_trace(
        go.Scatter(
            x=train_sizes,
            y=np.mean(train_scores, axis=1),
            mode="lines+markers",
            name="Training Score",
            error_y=dict(type="data", array=np.std(train_scores, axis=1), visible=True)
        )
    )

    fig.add_trace(
        go.Scatter(
            x=train_sizes,
            y=np.mean(val_scores, axis=1),
            mode="lines+markers",
            name="Validation Score",
            error_y=dict(type="data", array=np.std(val_scores, axis=1), visible=True)
        )
    )

    fig.update_layout(
        title="Learning Curves (Training vs Validation)",
        xaxis_title="Training Set Size",

```

```

        yaxis_title=scoring.capitalize(),
        height=500,
        width=800,
        legend=dict(x=0.05, y=0.95),
        template="plotly_white",
    )

fig.show()


```

✓ Explorcion de Datos

```

data = pd.read_csv("bank_marketing.csv")
display(data.head())

```




	age	job	marital	education	default	balance	housing	loan	contact
0	31	self-employed	married	tertiary	no	2666	no	no	cellular
1	29	unemployed	single	unknown	no	1584	no	no	cellular
2	41	blue-collar	married	secondary	no	2152	yes	no	cellular
3	50	blue-collar	married	secondary	no	84	yes	no	cellular
4	40	admin.	married	secondary	no	0	no	no	cellular

```
from tabulate import tabulate
```

```
print("Dataset Info:")
```


```
info = [  
    ["Number of Rows", data.shape[0]],  
    ["Number of Columns", data.shape[1]],  
    ["Column Names", "", ".join(data.columns)],  
    ["Missing Values", data.isnull().sum().sum()],  
    ["Data Types", data.dtypes.value_counts().to_dict()],  
]
```

```
print(tabulate(info, headers=["Property", "Value"], tablefmt="fancy_grid"))
```

 Dataset Info:

Property	Value
Number of Rows	9000
Number of Columns	17
Column Names	age, job, marital, education, default, balance, housing,
Missing Values	0
Data Types	{dtype('O'): 10, dtype('int64'): 7}

```
display("\nDescripción estadística:\n", data.describe())
```

 '\nDescripción estadística:\n'

	age	balance	day	duration	campaign	pdays	
count	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000
mean	41.090556	1482.262778	15.619556	353.832778	2.520111	50.511333	
std	11.664253	3031.013197	8.345305	336.945158	2.737758	107.691963	
min	18.000000	-3058.000000	1.000000	3.000000	1.000000	-1.000000	
25%	32.000000	109.000000	8.000000	131.000000	1.000000	-1.000000	
50%	39.000000	519.000000	15.000000	240.500000	2.000000	-1.000000	
75%	49.000000	1646.500000	21.000000	462.000000	3.000000	-1.000000	
max	95.000000	81204.000000	31.000000	3253.000000	58.000000	850.000000	


```
print("\nValores únicos por variable:\n", data.nunique())
print(
    "\nDistribución de la variable objetivo:\n", data["y"].value_counts(normalize:
)
```



Valores únicos por variable:

age	74
job	12
marital	3
education	4
default	2
balance	3476
housing	2
loan	2
contact	3
day	31
month	12
duration	1327
campaign	34
pdays	437
previous	31
poutcome	4
y	2

dtype: int64

Distribución de la variable objetivo:

y	
no	0.579222
yes	0.420778

Name: proportion, dtype: float64

✓ Preparacion de datos

```
original_numeric_columns = [
    "age",
    "balance",
    "day",
    "duration",
    "campaign",
    "pdays",
    "previous",
]
```

```

data_encoded = data.copy()
label_encode_cols = ["marital", "default", "housing", "loan", "contact"]
one_hot_encode_cols = ["job", "education", "month", "poutcome"]

le = LabelEncoder()
for col in label_encode_cols:
    data_encoded[col] = le.fit_transform(data_encoded[col].astype(str))

data_encoded = pd.get_dummies(
    data_encoded,
    columns=one_hot_encode_cols,
    drop_first=False, # Keep all levels, no drop
)


data_encoded["y"] = data_encoded["y"].map({"no": 0, "yes": 1}).astype(int)

bool_cols = data_encoded.select_dtypes(include=["bool"]).columns
data_encoded[bool_cols] = data_encoded[bool_cols].astype(int)

assert all(
    data_encoded.dtypes.apply(lambda x: np.issubdtype(x, np.number))
), "Non-numeric columns remain!"

display(data_encoded.head())

```



	age	marital	default	balance	housing	loan	contact	day	duration	campaign
0	31	1	0	2666	0	0	0	10	318	
1	29	2	0	1584	0	0	0	6	245	
2	41	1	0	2152	1	0	0	17	369	
3	50	1	0	84	1	0	0	17	18	
4	40	1	0	0	0	0	0	28	496	

5 rows x 45 columns

Durante el procesamiento de datos, se aplicaron dos tipos de codificación para las variables categóricas:

- **Label Encoding:** Utilizado para variables binarias o con pocos niveles (e.g., marital, default, housing, loan, contact). Convierte categorías a valores enteros, manteniendo simplicidad y evitando inflar la dimensión del dataset.
- **One-Hot Encoding:** Aplicado a variables categóricas multiclase (e.g., job, education, month, outcome). Representa cada nivel como una columna independiente, evitando imponer un orden artificial entre categorías nominales y mejorando la representación para modelos sensibles a relaciones numéricas no reales.

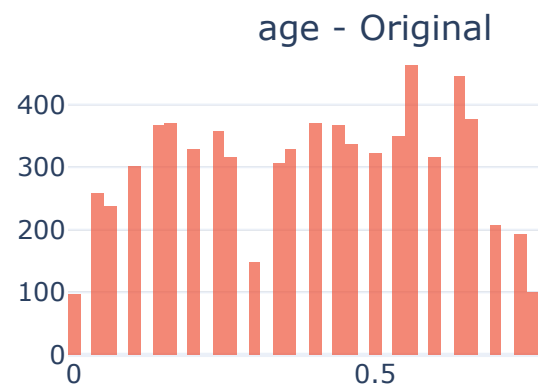
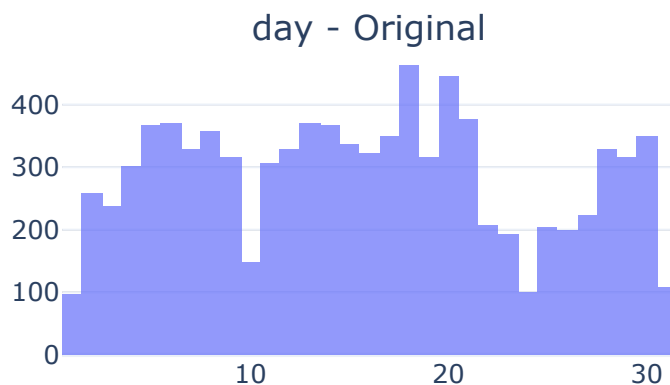
```
X = data_encoded.drop(columns=["y"])
y = data_encoded["y"]
```

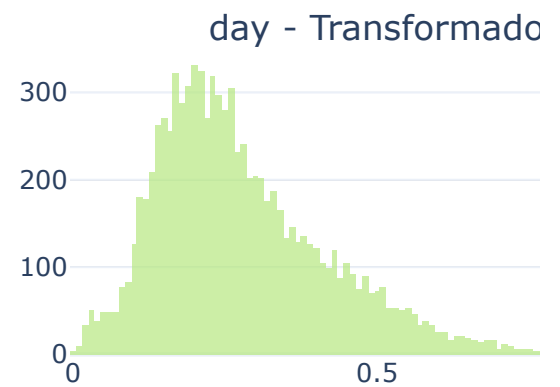
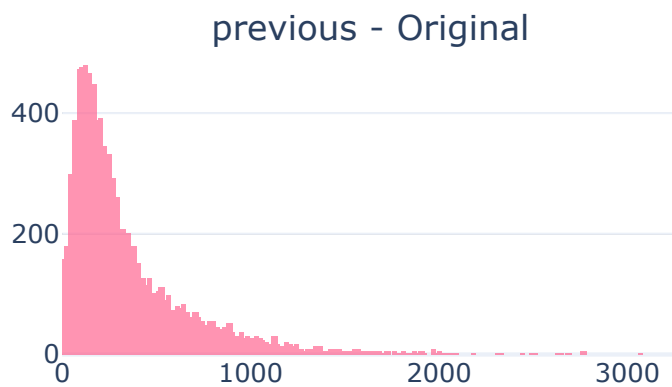
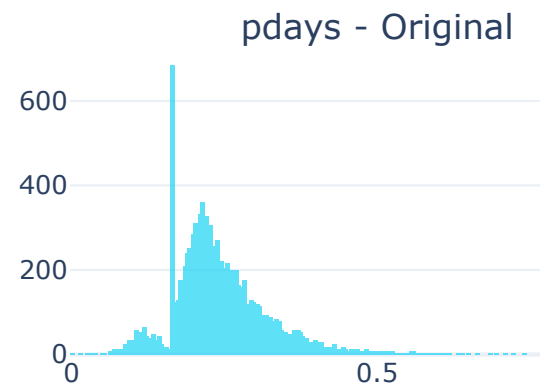
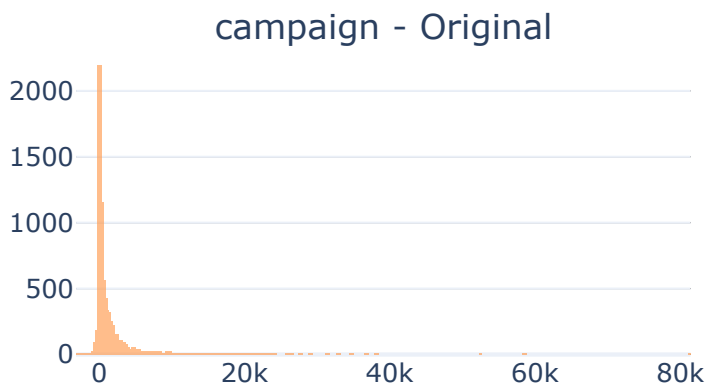
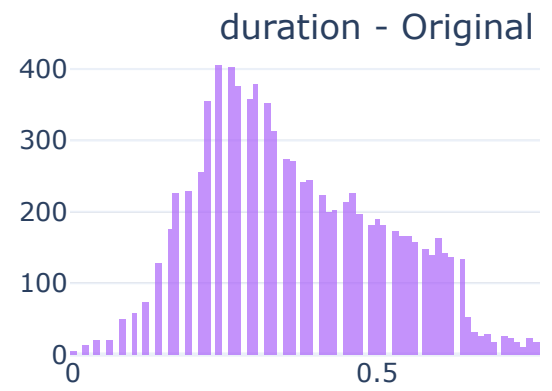
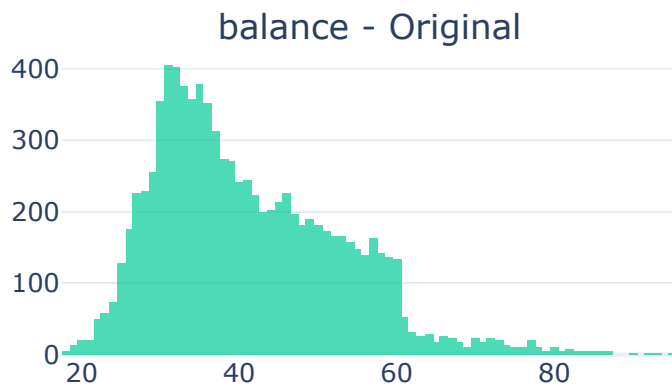
```
X_before_processing = X.copy()
X_processed, acciones = process_data(X, original_numeric_columns)
```

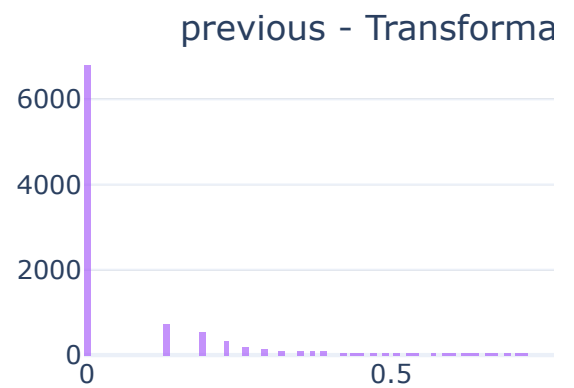
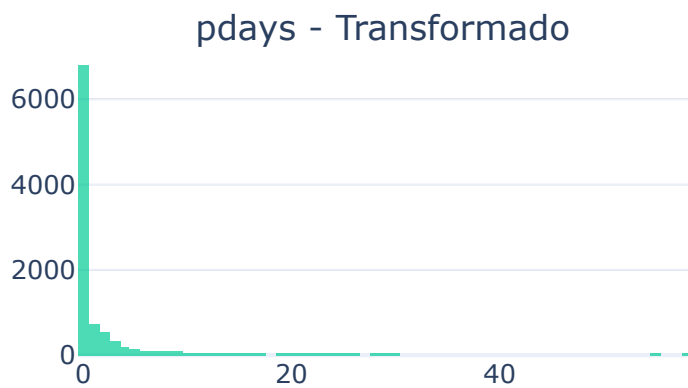
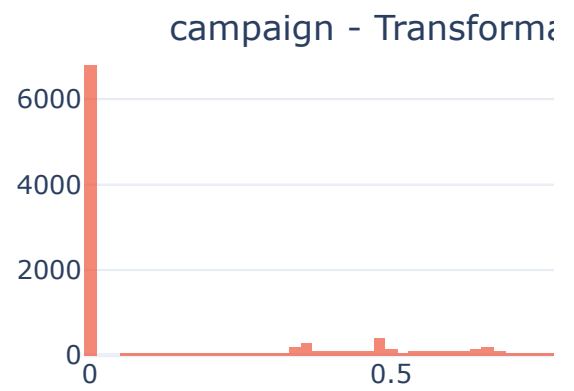
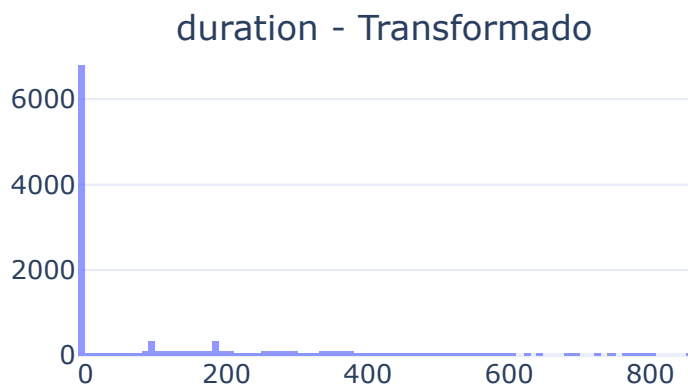
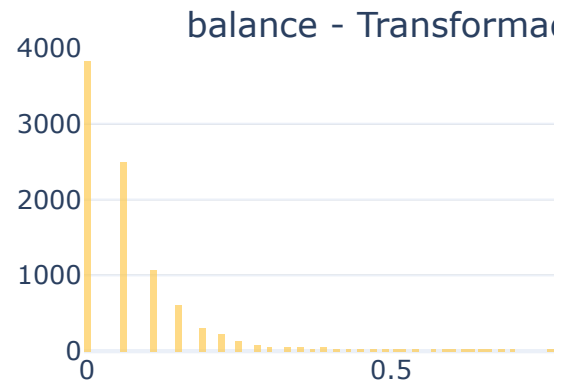
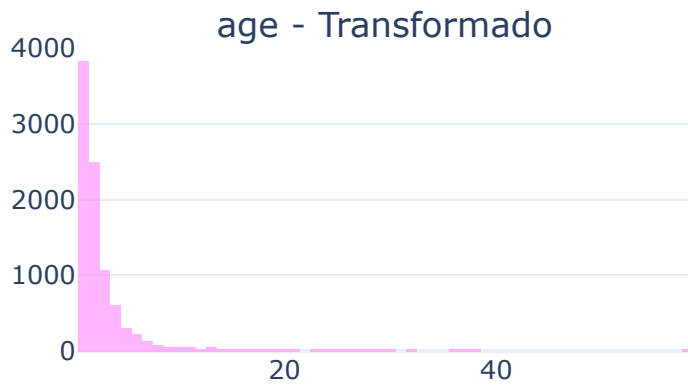
```
plot_hist(
    original_data=X_before_processing,
    transformed_data=X_processed,
    columns=acciones["Variable"].tolist(),
)
```



Distribuciones Original vs Transformado (Grid 2x2)







Durante el análisis exploratorio, se identificaron variables numéricas con alto sesgo (skewness) y desviación estándar (std). Para mejorar el desempeño de los modelos, se aplicaron las siguientes estrategias:

- **Transformar:** Variables con sesgo $> \pm 0.5$ fueron transformadas (log1p o raíz cuadrada segura) para normalizar su distribución.
- **Escalar:** Variables con alta dispersión fueron escaladas con MinMaxScaler (0 a 1), crucial para redes neuronales.
- **Transformar y Escalar:** Variables que cumplían ambas condiciones fueron transformadas y luego escaladas.

✓ Particionar datos

```
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X_processed, y, test_size=0.2, random_state=42, stratify=y
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.25, random_state=42, stratify=y_train_val
)

print(
    f"Tamaños:\nEntrenamiento: {X_train.shape}\nValidación: {X_val.shape}\nPrueba"
)
```

```
➡ Tamaños:
Entrenamiento: (5400, 44)
Validación: (1800, 44)
Prueba: (1800, 44)
```

✓ Modelos

Regresión Logística Base

```
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train, y_train)
y_val_pred_logreg = log_reg.predict(X_val)
acc_logreg = accuracy_score(y_val, y_val_pred_logreg)
print("\nRegresión Logística Base - Accuracy Validación:", acc_logreg)
```



Regresión Logística Base - Accuracy Validación: 0.8205555555555556

Regresión Logística Ajustada

```
param_grid_logreg = {
    "C": [0.01, 0.1, 1, 10, 100],
    "penalty": ["l2"],
    "solver": ["lbfgs", "saga", "newton-cg"],
}

grid_search_logreg = GridSearchCV(
    LogisticRegression(max_iter=1000, random_state=42),
    param_grid=param_grid_logreg,
    scoring="accuracy",
    cv=3,
    verbose=2,
    n_jobs=-1,
)
grid_search_logreg.fit(X_train, y_train)
best_logreg_model = grid_search_logreg.best_estimator_
y_val_pred_best_logreg = best_logreg_model.predict(X_val)
acc_best_logreg = accuracy_score(y_val, y_val_pred_best_logreg)
print("\nRegresión Logística Ajustada - Accuracy Validación:", acc_best_logreg)
```



Fitting 3 folds for each of 15 candidates, totalling 45 fits

Regresión Logística Ajustada - Accuracy Validación: 0.8222222222222222

Red Neuronal Base

```
mlp = MLPClassifier(
    hidden_layer_sizes=(32, 16),
    activation="relu",
    solver="adam",
    learning_rate_init=0.001,
```

```
max_iter=300,  
alpha=0.0001,  
early_stopping=True,  
n_iter_no_change=10,  
random_state=42,  
verbose=True,  
)  
mlp.fit(X_train, y_train)  
y_val_pred_mlp = mlp.predict(X_val)  
acc_mlp = accuracy_score(y_val, y_val_pred_mlp)  
print("\nRed Neuronal Base (MLP) - Accuracy Validación:", acc_mlp)
```

```
↔ Iteration 1, loss = 0.65386926  
Validation score: 0.648148  
Iteration 2, loss = 0.62065156  
Validation score: 0.687037  
Iteration 3, loss = 0.59240163  
Validation score: 0.716667  
Iteration 4, loss = 0.56784289  
Validation score: 0.738889  
Iteration 5, loss = 0.54517315  
Validation score: 0.759259  
Iteration 6, loss = 0.52061470  
Validation score: 0.770370  
Iteration 7, loss = 0.49562813  
Validation score: 0.779630  
Iteration 8, loss = 0.47111506  
Validation score: 0.825926  
Iteration 9, loss = 0.45046820  
Validation score: 0.814815  
Iteration 10, loss = 0.42988816  
Validation score: 0.831481  
Iteration 11, loss = 0.41404909  
Validation score: 0.851852  
Iteration 12, loss = 0.39726679  
Validation score: 0.857407  
Iteration 13, loss = 0.38769023  
Validation score: 0.851852  
Iteration 14, loss = 0.37769919  
Validation score: 0.855556  
Iteration 15, loss = 0.37164997  
Validation score: 0.872222  
Iteration 16, loss = 0.36473636  
Validation score: 0.859259  
Iteration 17, loss = 0.36341943  
Validation score: 0.866667  
Iteration 18, loss = 0.35846083  
Validation score: 0.866667  
Iteration 19, loss = 0.35355192
```



```
Validation score: 0.861111
Iteration 20, loss = 0.34934298
Validation score: 0.872222
Iteration 21, loss = 0.34728552
Validation score: 0.866667
Iteration 22, loss = 0.34535458
Validation score: 0.864815
Iteration 23, loss = 0.34498607
Validation score: 0.861111
Iteration 24, loss = 0.34179969
Validation score: 0.857407
Iteration 25, loss = 0.33906688
Validation score: 0.862963
Iteration 26, loss = 0.33862386
Validation score: 0.861111
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs
```

Red Neuronal Base (MLP) – Accuracy Validación: 0.8338888888888889

Red Neuronal Ajustada

```

param_grid_mlp = {
    "hidden_layer_sizes": [(32,), (64,), (32, 16), (64, 32)],
    "activation": ["relu", "tanh"],
    "solver": ["adam"],
    "learning_rate_init": [0.001, 0.01],
}

grid_search_mlp = GridSearchCV(
    MLPClassifier(max_iter=300, early_stopping=True, random_state=42),
    param_grid=param_grid_mlp,
    scoring="accuracy",
    cv=3,
    verbose=2,
    n_jobs=-1,
)
grid_search_mlp.fit(X_train, y_train)
best_mlp_model = grid_search_mlp.best_estimator_
y_val_pred_best_mlp = best_mlp_model.predict(X_val)
acc_best_mlp = accuracy_score(y_val, y_val_pred_best_mlp)
print("\nRed Neuronal Ajustada (MLP) - Accuracy Validación:", acc_best_mlp)

```

↔ Fitting 3 folds for each of 16 candidates, totalling 48 fits

Red Neuronal Ajustada (MLP) - Accuracy Validación: 0.8311111111111111

✓ Comparacion de Modelos

```

model_scores = {
    "Logistic Regression Base": acc_logreg,
    "Logistic Regression Ajustado": acc_best_logreg,
    "Red Neuronal Base": acc_mlp,
    "Red Neuronal Ajustado": acc_best_mlp,
}

model_scores_df = pd.DataFrame(
    list(model_scores.items()), columns=["Modelo", "Accuracy"]
)

fig = go.Figure()
fig.add_trace(
    go.Bar(
        x=model_scores_df["Modelo"],
        y=model_scores_df["Accuracy"],
    )
)

```

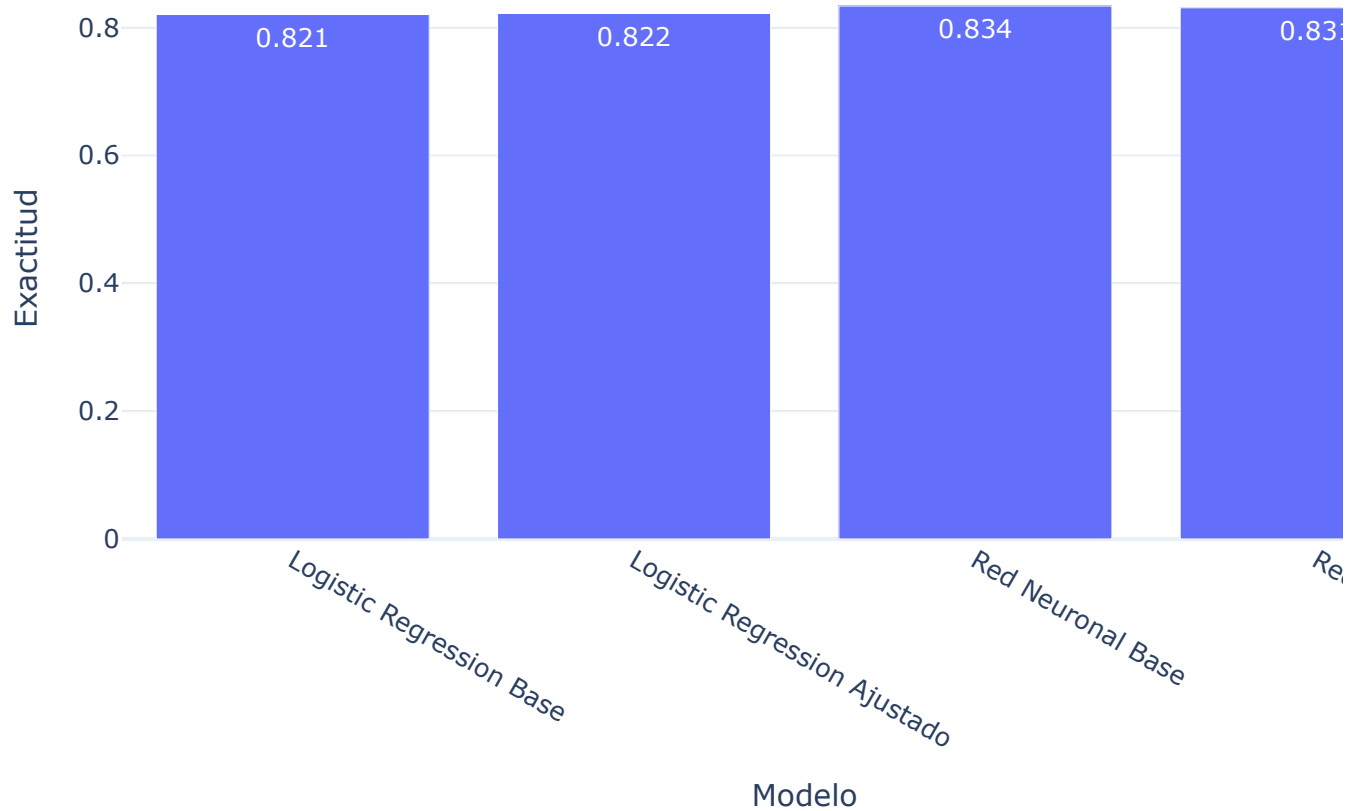
```
        text=model_scores_df["Accuracy"].round(3),
        textposition="auto",
    )
)
fig.update_layout(
    title="Comparación de Modelos - Exactitud en Validación",
    xaxis_title="Modelo",
    yaxis_title="Exactitud",
    height=500,
    width=800,
    template="plotly_white",
)
fig.show()

best_model_name = model_scores_df.loc[model_scores_df["Accuracy"].idxmax(), "Modelo"]
print("\n>>> Mejor Modelo en Validación:", best_model_name)

if best_model_name == "Regresion Logistica Base":
    final_model = log_reg
elif best_model_name == "Regresion Logistica Ajustado":
    final_model = best_logreg_model
elif best_model_name == "Red Neuronal Base":
    final_model = mlp
elif best_model_name == "Red Neuronal Ajustado":
    final_model = best_mlp_model
```



Comparación de Modelos - Exactitud en Validación



```
>>> Mejor Modelo en Validación: Red Neuronal Base
```

✓ Evaluacion en datos de prueba

```
y_test_pred_final = final_model.predict(X_test)
cm_final = confusion_matrix(y_test, y_test_pred_final)
```

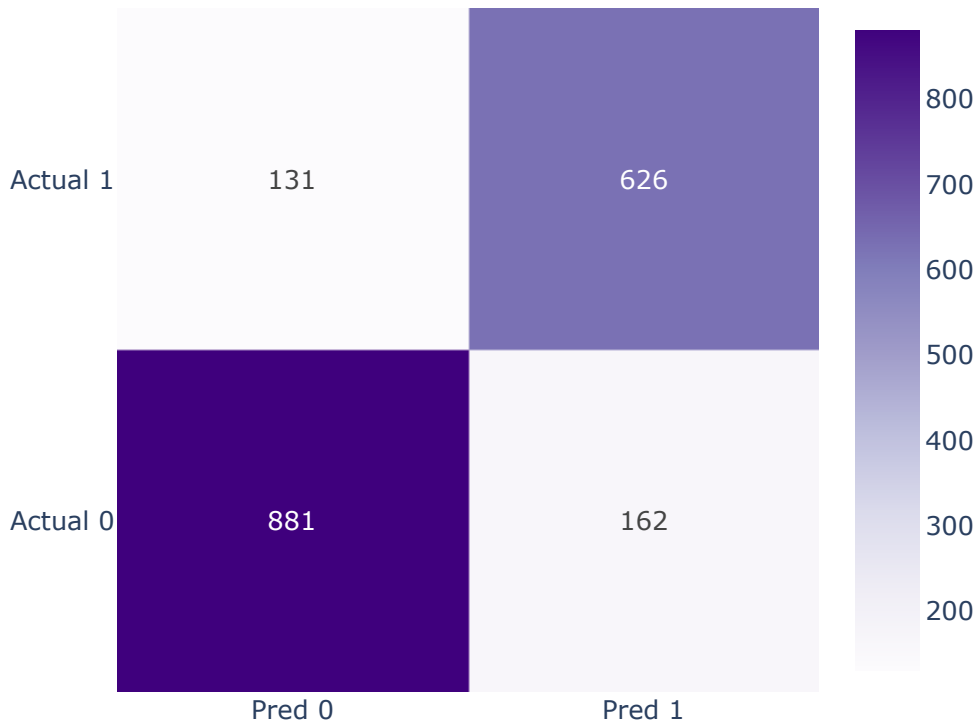
```
fig = go.Figure(
    data=go.Heatmap(
        z=cm_final,
        x=["Pred 0", "Pred 1"],
        y=["Actual 0", "Actual 1"],
        colorscale="Purples",
        showscale=True,
        text=cm_final,
```

```
        texttemplate="%{text}",
    )
)
fig.update_layout(
    title=f"Matriz de Confusión - {best_model_name} (Datos de Prueba)",
    width=500,
    height=500,
    template="plotly_white",
)
fig.show()

print(f"\nReporte de Clasificacion - {best_model_name} (Datos de Prueba Finales):")
print(classification_report(y_test, y_test_pred_final))
print(
    f"Exactitud - {best_model_name} (Datos de Prueba Finales):",
    accuracy_score(y_test, y_test_pred_final),
)
```



Matriz de Confusión - Red Neuronal Base (Datos de Prue



Reporte de Clasificacion - Red Neuronal Base (Datos de Prueba Finales):				
	precision	recall	f1-score	support
0	0.87	0.84	0.86	1043
1	0.79	0.83	0.81	757
accuracy			0.84	1800
macro avg	0.83	0.84	0.83	1800
weighted avg	0.84	0.84	0.84	1800

Exactitud - Red Neuronal Base (Datos de Prueba Finales): 0.8372222222222222

Curvas de Aprendizaje

```
plot_learning_curves(  
    estimator=mlp,  
    X=X_train_val,  
    y=y_train_val,  
    cv=3,  
    scoring="accuracy",  
)
```



Learning Curves (Training vs Validation)



Conclusiones del Proyecto

Trabajar en este proyecto me ayudó a ver de primera mano cómo el machine learning puede transformar procesos tradicionales, como el telemarketing, en estrategias mucho más eficientes y basadas en datos.

Lo que aprendí sobre Inteligencia Artificial:

- **Automatización de decisiones:** Pude reemplazar la intuición con modelos que toman decisiones basadas en datos reales.
- **Descubrimiento de patrones ocultos:** Encontré relaciones en los datos que, de otra forma, habrían pasado desapercibidas.
- **Mejor enfoque en las campañas:** Logré dirigir los esfuerzos hacia los clientes más propensos a comprar, aprovechando mejor los recursos.

La importancia de preparar bien los datos

Me di cuenta de que un buen modelo empieza con un buen procesamiento de datos. En este proyecto trabajé mucho en:

- **Transformaciones:** Para corregir distribuciones sesgadas.
- **Escalado:** Para manejar variables muy dispersas, algo que fue clave especialmente para las redes neuronales.
- **Codificación:** Usé Label Encoding y One-Hot Encoding para convertir las variables categóricas en algo que los modelos pudieran entender.

¿Qué modelo funcionó mejor?

- **Regresión Logística:** Fue un buen punto de partida, simple pero bastante competitivo.
- **Redes Neuronales:** Al final, las redes neuronales fueron las que mejor capturaron las relaciones complejas, superando a la regresión.

Personalmente disfrute mucho de poder darle un proceso ordenado y justificado a los modelos tomando decisiones basadas en datos en cada paso pensado en cadena que beneficiaría mejor mi siguiente paso desde el actual.

