2. Creación de Modelos

En este notebook se desarrollará la fase de modelamiento del proyecto de Minería de Datos siguiendo la metodología CRISP-DM. El objetivo principal es construir y evaluar modelos de clasificación que permitan resolver el problema definido en la etapa de entendimiento del negocio.

Se implementarán siete modelos de machine learning, divididos en dos grandes categorías:

Modelos de aprendizaje supervisado:

- Árbol de decisión
- Redes neuronales (MLPClassifier)
- Máquinas de vectores de soporte (SVM)
- K-vecinos más cercanos (KNN)

Modelos de ensamble:

- Random Forest
- XGBoost
- Votación suave (Soft Voting)

Cada modelo será entrenado y evaluado utilizando una validación cruzada sobre el 70% de los datos (previamente balanceados si es necesario), y se calcularán al menos cuatro métricas de calidad para cada uno. Posteriormente, los resultados obtenidos se someterán a un análisis estadístico (ANOVA y prueba de Tukey) para identificar diferencias significativas entre los modelos y seleccionar los tres con mejor desempeño.

Finalmente, los tres modelos seleccionados serán optimizados mediante técnicas como GridSearchCV y métodos de optimización avanzados (como algoritmos genéticos u optimización bayesiana), con el fin de almacenar el mejor modelo en un pipeline listo para su despliegue.

```
!pip install imblearn==0.0
```

Run to view results

```
!pip install scikit-learn==1.1.3 imbalanced-learn==0.9.1
```

Run to view results

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, cross_validate, StratifiedKFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline
```

Run to view results

Lectura de datos preprocesados

```
#Lectura de datos
df= pd.read_csv("/work/desercion_preparado.csv")
df.info()
```

Run to view results

Mediante la descripción estadística en la preparación de datos se evidenció que hay un desbalance en la variable objetivo. Sin embargo, se consulta nuevamente.

```
print(df['Target'].value_counts(normalize=True))
```

Run to view results

2.1 Separación de datos y balanceo

```
# Separar variables
X = df.drop('Target', axis=1)
y = df['Target']
```

Run to view results

```python
#  Dividir en entrenamiento (70%) y prueba (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)
```

Run to view results

```python
#Balanceo de solo el 70%
smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)
```

Run to view results

```python
#Validación Cruzada
from sklearn.model_selection import cross_validate, StratifiedKFold

#Dataframe para comparar los modelos
comparacion_CV=pd.DataFrame()
scoring=('f1', 'accuracy','precision', 'recall')
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

Run to view results

```python
df.info()
```

Run to view results

Se realiza creación de dummies a application mode

mirar agrupación de variables categóricas!

Se normaliza variables numericas

2.2 Creación de modelos de aprendizaje supervisado

2.2.1 Árbol

```
from sklearn.tree import DecisionTreeClassifier

model_tree = DecisionTreeClassifier(
    criterion='gini',
    max_depth=10,
    min_samples_split=20,
    min_samples_leaf=10,
    random_state=42
)

scores_tree = cross_validate(
    model_tree,
    X_train_bal, y_train_bal,
    cv=cv,
    scoring=scoring,
    return_train_score=True,
    return_estimator=False
)
scores_tree = pd.DataFrame(scores_tree)
print("Resultados árbol de decisión:")
print(scores_tree[['test_f1', 'test_accuracy', 'test_precision', 'test_recall']].mean())
```

Run to view results

```
variables_numericas = [
    'Application order',
    'Daytime/evening attendance',
    'Previous qualification (grade)',
    'Admission grade',
    'Displaced',
    'Debtor',
    'Tuition fees up to date',
    'Gender',
    'Scholarship holder',
    'Age at enrollment',
    'Curricular units 1st sem (evaluations)',
    'Curricular units 1st sem (without evaluations)',
    'Curricular units 2nd sem (credited)',
    'Curricular units 2nd sem (enrolled)',
    'Curricular units 2nd sem (evaluations)',
    'Curricular units 2nd sem (approved)',
    'Curricular units 2nd sem (grade)',
    'Curricular units 2nd sem (without evaluations)',
    'Unemployment rate',
    'Inflation rate',
    'GDP'
]
```

Run to view results

2.2.2 Red neuronal

```python
from sklearn.neural_network import MLPClassifier
from sklearn.compose import ColumnTransformer

# Crear transformador que solo escale variables numéricas
preprocessor_rn = ColumnTransformer([
    ('scaler', MinMaxScaler(), variables_numericas)
], remainder='passthrough')

# Pipeline con normalización selectiva para evitar data leaking
pipeline_rn = ImbPipeline([
    ('preprocessor', preprocessor_rn),
    ('classifier', MLPClassifier(
        activation="relu",
        hidden_layer_sizes=(16),
        learning_rate='constant',
        learning_rate_init=0.02,
        momentum=0.3,
        max_iter=500,
        verbose=False,
        random_state=42
    ))
])

scores_rn = cross_validate(
    pipeline_rn,
    X_train_bal, y_train_bal,
    cv=cv,
    scoring=scoring,
    return_train_score=True,
    return_estimator=False
)
scores_rn = pd.DataFrame(scores_rn)
print("Resultados Red Neuronal:")
print(scores_rn[['test_f1', 'test_accuracy', 'test_precision', 'test_recall']].mean())
```

Run to view results

2.2.3 SVM (Support Vector Machine)

2.2.4 KNN

```python
from sklearn.svm import SVC

# Crear transformador que only escale variables numéricas
preprocessor_svm = ColumnTransformer([
    ('scaler', MinMaxScaler(), variables_numericas)
], remainder='passthrough')

pipeline_svm = ImbPipeline([
    ('preprocessor', preprocessor_svm),
    ('classifier', SVC(
        C=1.0,
        kernel='rbf',
        gamma='scale',
        probability=True,
        random_state=42
    ))
])

scores_svm = cross_validate(
    pipeline_svm,
    X_train_bal, y_train_bal,
    cv=cv,
    scoring=scoring,
    return_train_score=True,
    return_estimator=False
)
scores_svm = pd.DataFrame(scores_svm)
print("Resultados SVM:")
print(scores_svm[['test_f1', 'test_accuracy', 'test_precision', 'test_recall']].mean())
```

Run to view results

```python
from sklearn.neighbors import KNeighborsClassifier

# Crear transformador que solo escale variables numéricas
preprocessor_knn = ColumnTransformer([
    ('scaler', MinMaxScaler(), variables_numericas)
], remainder='passthrough')

pipeline_knn = ImbPipeline([
    ('preprocessor', preprocessor_knn),
    ('classifier', KNeighborsClassifier(n_neighbors=1, metric='euclidean'))
])

scores_knn = cross_validate(
    pipeline_knn,
    X_train_bal, y_train_bal,
    cv=cv,
    scoring=scoring,
    return_train_score=True,
    return_estimator=False
)
scores_knn = pd.DataFrame(scores_knn)
print("Resultados KNN:")
print(scores_knn[['test_f1', 'test_accuracy', 'test_precision', 'test_recall']].mean())
```

Run to view results

```
!pip install xgboost==3.0.0
```

Run to view results

2.3 Creación de modelos de ensamble

```
import xgboost as xgb
model_xgb = xgb.XGBClassifier(
    max_depth=10,
    learning_rate=0.1,
    n_estimators=100,
    subsample=0.8,
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)

scores_xgboost = cross_validate(
    model_xgb,
    X_train_bal, y_train_bal,
    cv=cv,
    scoring=scoring,
    return_train_score=True,
    return_estimator=False
)
scores_xgboost = pd.DataFrame(scores_xgboost)
print("Resultados XGBoost:")
print(scores_xgboost[['test_f1', 'test_accuracy', 'test_precision', 'test_recall']].mean())
```

Run to view results

2.3.1 Random forest

```
from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier(
    n_estimators=100,
    max_depth=15,
    min_samples_split=10,
    min_samples_leaf=5,
    max_features='sqrt',
    random_state=42,
    n_jobs=-1
)

scores_rf = cross_validate(
    model_rf,
    X_train_bal, y_train_bal,
    cv=cv,
    scoring=scoring,
    return_train_score=True,
    return_estimator=False
)
scores_rf = pd.DataFrame(scores_rf)
print("Resultados Random Forest:")
print(scores_rf[['test_f1', 'test_accuracy', 'test_precision', 'test_recall']].mean())
```

Run to view results

2.3.2 XGBoost

```
def calcular_promedios(scores, nombre_modelo):
    return {
        'Modelo': nombre_modelo,
        'F1': scores['test_f1'].mean(),
        'Accuracy': scores['test_accuracy'].mean(),
        'Precision': scores['test_precision'].mean(),
        'Recall': scores['test_recall'].mean()
    }

resultados = pd.DataFrame([
    calcular_promedios(scores_tree, "Árbol de decisión"),
    calcular_promedios(scores_rn, "Red Neuronal"),
    calcular_promedios(scores_svm, "SVM"),
    calcular_promedios(scores_knn, "KNN"),
    calcular_promedios(scores_rf, "Random Forest"),
    calcular_promedios(scores_xgboost, "XGBoost"),
    calcular_promedios(scores_voting, "Soft Voting")
])

resultados = resultados.sort_values('F1', ascending=False)
print(resultados.to_markdown(index=False))
```

Run to view results

2.3.3 Soft Voting

```python
from sklearn.ensemble import VotingClassifier

# Crear transformadores para modelos que necesitan normalización
preprocessor_voting = ColumnTransformer([
    ('scaler', MinMaxScaler(), variables_numericas)
], remainder='passthrough')

# Para Soft Voting, reutilizamos modelos ya definidos y creamos pipelines solo donde sea necesario
estimators = [
    ('tree', model_tree),  # Uso del árbol ya definido
    ('rf', model_rf),      # Uso del random forest ya definido
    ('svm', Pipeline([
        ('preprocessor', preprocessor_voting),
        ('classifier', SVC(
            C=1.0, kernel='rbf', gamma='scale',
            probability=True, random_state=42
        ))
    ])),
    ('mlp', Pipeline([
        ('preprocessor', ColumnTransformer([
            ('scaler', MinMaxScaler(), variables_numericas)
        ], remainder='passthrough')),
        ('classifier', MLPClassifier(
            activation="relu", hidden_layer_sizes=(16),
            learning_rate='constant', learning_rate_init=0.02,
            momentum=0.3, max_iter=500, verbose=False,
            random_state=42
        ))
    ])),
    ('xgb', model_xgb)  # Uso de XGBoost ya definido
]

model_voting = VotingClassifier(
    estimators=estimators,
    voting='soft',  # Usa probabilidades
    n_jobs=-1
)

scores_voting = cross_validate(
    model_voting,
    X_train_bal, y_train_bal,
    cv=cv,
    scoring=scoring,
    return_train_score=True,
    return_estimator=False
)
scores_voting = pd.DataFrame(scores_voting)
print("Resultados Soft Voting:")
print(scores_voting[['test_f1', 'test_accuracy', 'test_precision', 'test_recall']].mean())
```

Run to view results

```
from scipy import stats
from statsmodels.stats.multicomp import pairwise_tukeyhsd

# Crear un DataFrame con los resultados de F1-score de los modelos
resultados_f1 = pd.DataFrame({
    'Árbol de decisión': scores_tree['test_f1'],
    'Red Neuronal': scores_rn['test_f1'],
    'SVM': scores_svm['test_f1'],
    'KNN': scores_knn['test_f1'],
    'Random Forest': scores_rf['test_f1'],
    'XGBoost': scores_xgboost['test_f1'],
    'Soft Voting': scores_voting['test_f1']
})

# Reorganizar los datos para ANOVA
resultados_melt = resultados_f1.melt(var_name='Modelo', value_name='F1')
```

Run to view results

2.4 Anova

```python
# Realizar ANOVA
anova_result = stats.f_oneway(
    resultados_f1['Árbol de decisión'],
    resultados_f1['Red Neuronal'],
    resultados_f1['SVM'],
    resultados_f1['KNN'],
    resultados_f1['Random Forest'],
    resultados_f1['XGBoost'],
    resultados_f1['Soft Voting']
)

print("Resultados ANOVA:")
print(f"F-statistic: {anova_result.statistic:.4f}")
print(f"p-value: {anova_result.pvalue:.4f}")

# Interpretación
if anova_result.pvalue < 0.05:
    print("\nHay diferencias significativas entre al menos un par de modelos (p < 0.05).")
    print("Procedemos con la prueba de Tukey para identificar qué modelos difieren.")

    # Realizar prueba de Tukey
    tukey = pairwise_tukeyhsd(
        endog=resultados_melt['F1'],
        groups=resultados_melt['Modelo'],
        alpha=0.05
    )

    print("\nResultados de la prueba de Tukey:")
    print(tukey.summary())

    # Visualización gráfica
    tukey.plot_simultaneous()
    plt.ylabel('Modelo')
    plt.xlabel('Diferencia en F1-score')
    plt.title('Comparación de modelos con prueba de Tukey')
    plt.show()
else:
    print("\nNo hay diferencias significativas entre los modelos (p ≥ 0.05).")
```

Run to view results

2.3 Comparación de métricas de modelos

```
!pip install statsmodels==0.14.4
```

Run to view results

3. Hiperparametrización

TOP 3 RECOMENDADO (equilibrando performance y costo):
🥇 1. XGBoost

F1-score: ~0.91 (el más alto)
Costo: Alto, pero justificado por su excelente performance
Ventajas: Mejor modelo individual, robusto, maneja bien overfitting

🥈 2. Random Forest

F1-score: ~0.89 (muy bueno)
Costo: Medio (paralelizable, eficiente)
Ventajas: Excelente relación performance/costo, interpretable, estable

🥉 3. SVM

F1-score: ~0.89 (muy bueno)
Costo: Medio
Ventajas: Sólido desempeño, buena generalización

```
pip install scikit-optimize
```

Run to view results

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import ColumnTransformer
from skopt import BayesSearchCV
from skopt.space import Real, Categorical, Integer
import joblib
import warnings
warnings.filterwarnings('ignore')

# Cargar datos
df = pd.read_csv("/work/desercion_preparado.csv")
X = df.drop('Target', axis=1)
y = df['Target']

# Dividir datos (ya balanceados previamente)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# Definir variables numéricas (según documento)
variables_numericas = [
    'Application order', 'Daytime/evening attendance', 'Previous qualification (grade)',
    'Admission grade', 'Displaced', 'Debtor', 'Tuition fees up to date', 'Gender',
    'Scholarship holder', 'Age at enrollment', 'Curricular units 1st sem (evaluations)',
    'Curricular units 1st sem (without evaluations)', 'Curricular units 2nd sem (credited)',
    'Curricular units 2nd sem (enrolled)', 'Curricular units 2nd sem (evaluations)',
    'Curricular units 2nd sem (approved)', 'Curricular units 2nd sem (grade)',
    'Curricular units 2nd sem (without evaluations)', 'Unemployment rate',
    'Inflation rate', 'GDP'
]

# Transformador para escalar solo variables numéricas
preprocessor = ColumnTransformer([
    ('scaler', MinMaxScaler(), variables_numericas)
], remainder='passthrough')

# Configuración común
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
scoring = 'f1'
```

Run to view results

```
# Pipeline para XGBoost
pipeline_xgb = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42))
])

# Parámetros para GridSearch
param_grid_xgb = {
    'classifier__max_depth': [3, 5, 7, 10],
    'classifier__learning_rate': [0.01, 0.1, 0.2],
    'classifier__n_estimators': [50, 100, 200],
    'classifier__subsample': [0.6, 0.8, 1.0],
    'classifier__colsample_bytree': [0.6, 0.8, 1.0]
}

# GridSearch para XGBoost
grid_xgb = GridSearchCV(
    estimator=pipeline_xgb,
    param_grid=param_grid_xgb,
    cv=cv,
    scoring=scoring,
    n_jobs=-1,
    verbose=1
)

grid_xgb.fit(X_train, y_train)

print("Mejores parámetros XGBoost:", grid_xgb.best_params_)
print("Mejor F1-score XGBoost:", grid_xgb.best_score_)
```

Run to view results

```
# Pipeline para Random Forest
pipeline_rf = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42, n_jobs=-1))
])

# Parámetros para GridSearch
param_grid_rf = {
    'classifier__n_estimators': [50, 100, 200],
    'classifier__max_depth': [None, 5, 10, 20],
    'classifier__min_samples_split': [2, 5, 10],
    'classifier__min_samples_leaf': [1, 2, 4],
    'classifier__max_features': ['sqrt', 'log2', None]
}

# GridSearch para Random Forest
grid_rf = GridSearchCV(
    estimator=pipeline_rf,
    param_grid=param_grid_rf,
    cv=cv,
    scoring=scoring,
    n_jobs=-1,
    verbose=1
)

grid_rf.fit(X_train, y_train)

print("Mejores parámetros Random Forest:", grid_rf.best_params_)
print("Mejor F1-score Random Forest:", grid_rf.best_score_)
```

Run to view results

```
# Pipeline para SVM
pipeline_svm = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', SVC(probability=True, random_state=42))
])

# Parámetros para GridSearch
param_grid_svm = {
    'classifier__C': [0.1, 1, 10, 100],
    'classifier__kernel': ['linear', 'rbf', 'poly'],
    'classifier__gamma': ['scale', 'auto', 0.1, 1]
}

# GridSearch para SVM
grid_svm = GridSearchCV(
    estimator=pipeline_svm,
    param_grid=param_grid_svm,
    cv=cv,
    scoring=scoring,
    n_jobs=-1,
    verbose=1
)

grid_svm.fit(X_train, y_train)

print("Mejores parámetros SVM:", grid_svm.best_params_)
print("Mejor F1-score SVM:", grid_svm.best_score_)
```

Run to view results

Optimización bayesiana

XGBoost con BayesSearchCV

```
# Espacio de búsqueda para optimización bayesiana
search_space_xgb = {
    'classifier__max_depth': Integer(3, 10),
    'classifier__learning_rate': Real(0.01, 0.3, prior='log-uniform'),
    'classifier__n_estimators': Integer(50, 300),
    'classifier__subsample': Real(0.5, 1.0),
    'classifier__colsample_bytree': Real(0.5, 1.0),
    'classifier__gamma': Real(0, 5),
    'classifier__reg_alpha': Real(0, 10),
    'classifier__reg_lambda': Real(0, 10)
}

# Optimización bayesiana para XGBoost
bayes_xgb = BayesSearchCV(
    estimator=pipeline_xgb,
    search_spaces=search_space_xgb,
    cv=cv,
    scoring=scoring,
    n_jobs=-1,
    n_iter=50,
    verbose=1,
    random_state=42
)

bayes_xgb.fit(X_train, y_train)

print("Mejores parámetros XGBoost (Bayes):", bayes_xgb.best_params_)
print("Mejor F1-score XGBoost (Bayes):", bayes_xgb.best_score_)
```

Run to view results

Random Forest con BayesSearchCV

```
search_space_rf = {
    'classifier__n_estimators': Integer(50, 300),
    'classifier__max_depth': Integer(3, 20),
    'classifier__min_samples_split': Integer(2, 20),
    'classifier__min_samples_leaf': Integer(1, 10),
    'classifier__max_features': Categorical(['sqrt', 'log2', None]),
    'classifier__bootstrap': Categorical([True, False])
}

bayes_rf = BayesSearchCV(
    estimator=pipeline_rf,
    search_spaces=search_space_rf,
    cv=cv,
    scoring=scoring,
    n_jobs=-1,
    n_iter=50,
    verbose=1,
    random_state=42
)

bayes_rf.fit(X_train, y_train)

print("Mejores parámetros Random Forest (Bayes):", bayes_rf.best_params_)
print("Mejor F1-score Random Forest (Bayes):", bayes_rf.best_score_)
```

Run to view results

SVM con BayesSearchCV

```
search_space_svm = {
    'classifier__C': Real(1e-3, 1e3, prior='log-uniform'),
    'classifier__kernel': Categorical(['linear', 'rbf', 'poly']),
    'classifier__gamma': Real(1e-3, 1e3, prior='log-uniform'),
    'classifier__degree': Integer(2, 5)  # Solo para kernel poly
}

bayes_svm = BayesSearchCV(
    estimator=pipeline_svm,
    search_spaces=search_space_svm,
    cv=cv,
    scoring=scoring,
    n_jobs=-1,
    n_iter=50,
    verbose=1,
    random_state=42
)

bayes_svm.fit(X_train, y_train)

print("Mejores parámetros SVM (Bayes):", bayes_svm.best_params_)
print("Mejor F1-score SVM (Bayes):", bayes_svm.best_score_)
```

Run to view results

Evaluación y selección del mejor modelo

```python
# Recolectar resultados
results = {
    'XGBoost_Grid': grid_xgb.best_score_,
    'XGBoost_Bayes': bayes_xgb.best_score_,
    'RF_Grid': grid_rf.best_score_,
    'RF_Bayes': bayes_rf.best_score_,
    'SVM_Grid': grid_svm.best_score_,
    'SVM_Bayes': bayes_svm.best_score_
}

# Identificar el mejor modelo
best_model_name = max(results, key=results.get)
best_score = results[best_model_name]

print(f"\nEl mejor modelo es: {best_model_name} con un F1-score de: {best_score:.4f}")

# Asignar el mejor modelo
if 'XGBoost' in best_model_name:
    best_model = bayes_xgb if 'Bayes' in best_model_name else grid_xgb
elif 'RF' in best_model_name:
    best_model = bayes_rf if 'Bayes' in best_model_name else grid_rf
else:
    best_model = bayes_svm if 'Bayes' in best_model_name else grid_svm
```

Run to view results

Almacenamiento del mejor modelo para despliegue

```python
# Guardar el mejor modelo
joblib.dump(best_model.best_estimator_, 'mejor_modelo_desercion.pkl')

# Guardar también el preprocesador por separado (opcional)
joblib.dump(preprocessor, 'preprocessor_desercion.pkl')

print("Mejor modelo guardado correctamente para despliegue.")
```

Run to view results

```
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler
import xgboost
from xgboost import XGBClassifier
import joblib

variables_int = [
    'Application order',
    'Daytime/evening attendance',
    'Displaced',
    'Debtor',
    'Tuition fees up to date',
    'Gender',
    'Scholarship holder',
    'Age at enrollment',
    'Curricular units 1st sem (evaluations)',
    'Curricular units 1st sem (without evaluations)',
    'Curricular units 2nd sem (credited)',
    'Curricular units 2nd sem (enrolled)',
    'Curricular units 2nd sem (evaluations)',
    'Curricular units 2nd sem (approved)',
    'Curricular units 2nd sem (without evaluations)'
]

variables_float = [
    'Previous qualification (grade)',
    'Admission grade',
    'Curricular units 2nd sem (grade)',
    'Unemployment rate',
    'Inflation rate',
    'GDP'
]

variables_bool = [
    'Marital status_Divorced',
    'Marital status_FactoUnion',
    'Marital status_Separated',
    'Marital status_Single',
    'Application mode_Admisión Especial',
    'Application mode_Admisión Regular',
    'Application mode_Admisión por Ordenanza',
    'Application mode_Cambios/Transferencias',
    'Application mode_Estudiantes Internacionales',
    'Application mode_Mayores de 23 años',
    'Course_Agricultural & Environmental Sciences',
    'Course_Arts & Design',
    'Course_Business & Management',
    'Course_Communication & Media',
    'Course_Education',
    'Course_Engineering & Technology',
    'Course_Health Sciences',
    'Course_Social Sciences',
    'Previous qualification_Higher Education',
    'Previous qualification_Other',
    'Previous qualification_Secondary Education',
    'Previous qualification_Technical Education',
    'Nacionality_Colombian',
    'Nacionality_Cuban',
    'Nacionality_Dutch',
    'Nacionality_English',
    'Nacionality_German',
    'Nacionality_Italian',
    'Nacionality_Lithuanian',
    'Nacionality_Moldovan',
    'Nacionality_Mozambican',
    'Nacionality_Portuguese',
    'Nacionality_Romanian',
    'Nacionality_Santomean',
    'Nacionality_Turkish',
    "Mother's qualification_Basic_or_Secondary",
    "Mother's qualification_Other_or_Unknown",
    "Mother's qualification_Postgraduate",
    "Mother's qualification_Technical_Education",
    "Father's qualification_Basic_or_Secondary",
    "Father's qualification_Other_or_Unknown",
    "Father's qualification_Postgraduate",
    "Mother's occupation_Administrative/Clerical",
```

```
    "Mother's occupation_Skilled Manual Workers",
    "Mother's occupation_Special Cases",
    "Mother's occupation_Technicians/Associate Professionals",
    "Mother's occupation_Unskilled Workers",
    "Father's occupation_Administrative/Clerical",
    "Father's occupation_Professionals",
    "Father's occupation_Skilled Manual Workers",
    "Father's occupation_Special Cases",
    "Father's occupation_Technicians/Associate Professionals"
]

# Variables que necesitan escalado (numéricas continuas)
variables_a_escalar = variables_float + [
    'Application order',
    'Age at enrollment',
    'Curricular units 1st sem (evaluations)',
    'Curricular units 1st sem (without evaluations)',
    'Curricular units 2nd sem (credited)',
    'Curricular units 2nd sem (enrolled)',
    'Curricular units 2nd sem (evaluations)',
    'Curricular units 2nd sem (approved)',
    'Curricular units 2nd sem (without evaluations)'
]

# Crear el transformador para escalar solo las variables seleccionadas
preprocessor = ColumnTransformer([
    ('scaler', MinMaxScaler(), variables_a_escalar)
], remainder='passthrough')  # Las variables booleanas y otras int no necesitan escalado

# Configurar el modelo XGBoost con los mejores parámetros encontrados
best_xgb_params = {
    'colsample_bytree': 1.0,
    'gamma': 0.0,
    'learning_rate': 0.08905618676080333,
    'max_depth': 10,
    'n_estimators': 64,
    'reg_alpha': 0.0,
    'reg_lambda': 0.0,
    'subsample': 0.5,
    'use_label_encoder': False,
    'eval_metric': 'logloss',
    'random_state': 42
}

# Crear el pipeline completo
final_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', XGBClassifier(**best_xgb_params))
])

df = pd.read_csv("desercion_preparado.csv")
X = df.drop('Target', axis=1)
y = df['Target']

final_pipeline.fit(X, y)

# Guardar el pipeline completo para despliegue
joblib.dump(final_pipeline, 'pipeline_final_desercion.pkl')

# Opcional: Guardar también la lista de columnas esperadas
joblib.dump(X.columns.tolist(), 'columnas_esperadas.pkl')
```

Run to view results