```
# Librerias para la Manipulación de datos y visualización
import pandas as pd
import numpy as np
import plotly.express as px
import matplotlib.pyplot as plt
from matplotlib import rcParams
import seaborn as sns
from datetime import datetime, timedelta
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.application import MIMEApplication
import pickle
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

#LIbreria para pre-procesar datos (antes de entrenar el modelo)
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

#Librerias de modelos
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import tree

#Librerias para evaluar modelos
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.model_selection import KFold
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Importar el modelo de Clasificación
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, roc_curve, precision_recall_curve
from sklearn.preprocessing import StandardScaler
```

```
file_path = '/content/drive/MyDrive/SAP_Descuadres_Input/Historico_Consolidado_Copec.csv'
RUTA_SALIDA_MODELO = '/content/drive/MyDrive/SAP_Descuadres_Input/modelo_descuadres.pkl'
RUTA_SALIDA_REPORTES = '/content/drive/MyDrive/SAP_Descuadres_Input/reportes/'
```

```
df = pd.read_csv(file_path, sep=';')
print(df.head())
df.info()
```

```
  Fecha Contable Centro Producto Copec  Material Cantidad        Fecha Version
0     2025-10-01   C003          MUEVO         3    0,001  2025-12-22 21:53:12
1     2025-10-01   C006          MUEVO         3   -0,001  2025-12-22 21:53:12
2     2025-10-01   C006            TAE        16    -7619  2025-12-22 21:53:12
3     2025-10-01   C006            TCT        16     7619  2025-12-22 21:53:12
4     2025-10-01   C008            TAE        16   -24115  2025-12-22 21:53:12
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 570912 entries, 0 to 570911
Data columns (total 6 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   Fecha Contable  570912 non-null  object
 1   Centro          570912 non-null  object
 2   Producto Copec  531192 non-null  object
 3   Material        570912 non-null  int64
 4   Cantidad        570912 non-null  object
 5   Fecha Version   570912 non-null  object
dtypes: int64(1), object(5)
memory usage: 26.1+ MB
```

```
## NORMALIZACIÓN: CONFIGURACIÓN, LIMPIEZA DE DATOS Y CORRECCIÓN DE TIPOS

# Normalización de productos (TCT y TAE -> TCT_TAE)

df['Producto Copec'] = df['Producto Copec'].replace(['TCT', 'TAE'], 'TCT_TAE')
print(df['Producto Copec'].value_counts())

# Limpieza de la columna Cantidad
```

```
if df['Cantidad'].dtype == 'object':
   df['Cantidad'] = df['Cantidad'].str.replace(',', '.', regex=False)
   df['Cantidad'] = pd.to_numeric(df['Cantidad'], errors='coerce').fillna(0)

# Conversión de fechas

df['Fecha Contable'] = pd.to_datetime(df['Fecha Contable'], format='%Y-%m-%d', errors='coerce')
df['Fecha Version'] = pd.to_datetime(df['Fecha Version'], errors='coerce')

# Convertir 'Material' a string para OHE

df['Material'] = df['Material'].astype(str)

# 6.4 Agregación por día/centro/producto

df = df.groupby(['Fecha Contable', 'Centro', 'Material', 'Producto Copec'], as_index=False).agg({
    'Cantidad': 'sum',
    'Fecha Version': 'max'
})
print(f"Datos después de agregación: {len(df):,} registros")
```

```
Producto Copec
MUEVO      250829
TCT_TAE    191352
CUPON       61683
FFAA        26112
VIRTUAL      1205
DDRR            7
TR              4
Name: count, dtype: int64
Datos después de agregación: 314,576 registros
```

```
## INGENIERÍA DE CARACTERÍSTICAS (FEATURE ENGINEERING)

# Features temporales

df['Dia_Semana'] = df['Fecha Contable'].dt.dayofweek
df['Dia_Mes'] = df['Fecha Contable'].dt.day
df['Es_Fin_Semana'] = (df['Dia_Semana'] >= 5).astype(int)
df['Es_Fin_Mes'] = (df['Dia_Mes'] >= 28).astype(int)
df['Mes'] = df['Fecha Contable'].dt.month
df['Periodo'] = df['Fecha Contable'].dt.to_period('M')

# 7.2 Features de cantidad

df['Cantidad_Abs'] = df['Cantidad'].abs()
df['Es_Cantidad_Negativa'] = (df['Cantidad'] < 0).astype(int)

# 7.3 Features de agregación por Centro

centro_stats = df.groupby('Centro')['Cantidad'].agg([
    ('Centro_Mean', 'mean'),
    ('Centro_Std', 'std'),
    ('Centro_Count', 'count')
]).reset_index()
df = df.merge(centro_stats, on='Centro', how='left')
df['Centro_Std'] = df['Centro_Std'].fillna(0)

# 7.4 Features de agregación por Producto

producto_stats = df.groupby('Producto Copec')['Cantidad'].agg([
    ('Producto_Mean', 'mean'),
    ('Producto_Std', 'std')
]).reset_index()
df = df.merge(producto_stats, on='Producto Copec', how='left')
df['Producto_Std'] = df['Producto_Std'].fillna(0)

print(df.head())
```

```
  Fecha Contable Centro Material Producto Copec  Cantidad       Fecha Version  \
0     2022-11-19   G348        1          CUPON   -23.980 2025-12-22 21:44:25
1     2022-11-19   G348       13          CUPON  -871.600 2025-12-22 21:44:25
2     2022-11-19   G348        3          CUPON   -39.970 2025-12-22 21:44:25
3     2022-11-19   G403       13          CUPON  -155.390 2025-12-22 21:44:25
4     2022-11-20   G348        1          CUPON   177.055 2025-12-22 21:44:25

   Dia_Semana  Dia_Mes  Es_Fin_Semana  Es_Fin_Mes  Mes  Periodo  Cantidad_Abs  \
0           5       19              1           0   11  2022-11        23.980
1           5       19              1           0   11  2022-11       871.600
2           5       19              1           0   11  2022-11        39.970
3           5       19              1           0   11  2022-11       155.390
4           6       20              1           0   11  2022-11       177.055
```

```
   Es_Cantidad_Negativa  Centro_Mean  Centro_Std  Centro_Count  Producto_Mean  \
0                     1    39.210481  596.524115          2775     -38.650195
1                     1    39.210481  596.524115          2775     -38.650195
2                     1    39.210481  596.524115          2775     -38.650195
3                     1    12.772253  201.689622           237     -38.650195
4                     0    39.210481  596.524115          2775     -38.650195

   Producto_Std
0    733.593455
1    733.593455
2    733.593455
3    733.593455
4    733.593455
```

```
# CREACIÓN DE VARIABLE OBJETIVO (Y)

TARGET_COLUMN = 'Y_ALERTA_RIESGO'
UMBRAL = 0.05


# Calcular suma por periodo y centro
df_sum = df.groupby(['Periodo', 'Centro'])['Cantidad'].sum().reset_index()
df_sum.rename(columns={'Cantidad': 'Suma_Cantidad_Centro_Mes_Producto'}, inplace=True)

# Detectar descuadre: si |Suma| > UMBRAL_RIESGO
df_sum[TARGET_COLUMN] = (
    (df_sum['Suma_Cantidad_Centro_Mes_Producto'] < -UMBRAL) |
    (df_sum['Suma_Cantidad_Centro_Mes_Producto'] > UMBRAL)
).astype(int)

# Fusionar con df original
df = df.merge(df_sum[['Periodo', 'Centro',TARGET_COLUMN]],
              on=['Periodo', 'Centro'], how='left')

print(df['Y_ALERTA_RIESGO'].value_counts())
print(f"Porcentaje de descuadres: {df['Y_ALERTA_RIESGO'].mean()*100:.2f}%")
df.head()
```

```
Y_ALERTA_RIESGO
0    195793
1    118783
Name: count, dtype: int64
Porcentaje de descuadres: 37.76%
```

	Fecha Contable	Centro	Material	Producto Copec	Cantidad	Fecha Version	Dia_Semana	Dia_Mes	Es_Fin_Semana	Es_Fin_Mes	Mes	Per
0	2022-11-19	G348	1	CUPON	-23.980	2025-12-22 21:44:25	5	19	1	0	11	20
1	2022-11-19	G348	13	CUPON	-871.600	2025-12-22 21:44:25	5	19	1	0	11	20
2	2022-11-19	G348	3	CUPON	-39.970	2025-12-22 21:44:25	5	19	1	0	11	20
3	2022-11-19	G403	13	CUPON	-155.390	2025-12-22 21:44:25	5	19	1	0	11	20
4	2022-11-20	G348	1	CUPON	177.055	2025-12-22 21:44:25	6	20	1	0	11	20

```
## PREPARAR X e Y PARA ENTRENAMIENTO

# One-Hot Encoding

features_categoricas = ['Centro', 'Producto Copec', 'Material']
df_encoded = pd.get_dummies(df, columns=features_categoricas, drop_first=True)
df_encoded.columns

# Separar X e y

columnas_drop = ['Y_ALERTA_RIESGO', 'Fecha Contable', 'Fecha Version', 'Periodo']
X = df_encoded.drop(columns=columnas_drop, errors='ignore')
y = df_encoded['Y_ALERTA_RIESGO']

print(f"Dimensiones de X: {X.shape}")
print(f"Dimensiones de y: {y.shape}")
```

```
Dimensiones de X: (314576, 734)
Dimensiones de y: (314576,)
```

```
# DIVISIÓN TRAIN/TEST Y ESCALADO

# 1. División del dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size= 0.2,
    random_state= 5,
    stratify=y
)

# 2. Escalado de Variables Numéricas
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(f"Tamaño de X_train (Escalado): {X_train_scaled.shape}")
```

```
Tamaño de X_train (Escalado): (251660, 734)
```

```
from sklearn.metrics import mean_absolute_percentage_error, make_scorer

# ENTRENAMIENTO Y ANÁLISIS DEL MODELO

modelo_gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)

modelo_gbc.fit(X_train_scaled, y_train)

# Predecir sobre el conjunto de prueba
y_pred_proba_smote = modelo_gbc.predict_proba(X_test_scaled)[:, 1]
y_pred = modelo_gbc.predict(X_test_scaled)

# Evaluar el modelo utilizando RMSE
mape = np.sqrt(mean_absolute_percentage_error(y_test, y_pred))
print(f'MAPE del modelo de Gradient Boosting: {mape:.2f}')
```

```
MAPE del modelo de Gradient Boosting: 15780710.39
```

```
# 13. VALIDACIÓN CRUZADA
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(
    modelo_gbc,
    X_train_scaled,
    y_train,
    cv=5,
    scoring='roc_auc',
    n_jobs=-1
)

print(f"\n📊 Resultados de Validación Cruzada:")
print(f"   ROC-AUC promedio: {cv_scores.mean():.4f}")
print(f"   Desviación estándar: {cv_scores.std():.4f}")
print(f"   Scores individuales: {[f'{s:.4f}' for s in cv_scores]}")
```

```
📊 Resultados de Validación Cruzada:
   ROC-AUC promedio: 0.8062
   Desviación estándar: 0.0020
   Scores individuales: ['0.8094', '0.8035', '0.8049', '0.8065', '0.8064']
```

```
# 14. EVALUACIÓN EN TEST
# ------------------------------------------------------------------------
print("\n" + "="*70)
print("📊 EVALUACIÓN DEL MODELO EN TEST")
print("="*70)

# Predicciones
y_pred = modelo_gbc.predict(X_test_scaled)
y_pred_proba = modelo_gbc.predict_proba(X_test_scaled)[:, 1]

# Métricas
roc_auc = roc_auc_score(y_test, y_pred_proba)
cm = confusion_matrix(y_test, y_pred)

print(f"\n📈 ROC-AUC Score: {roc_auc:.4f}")
```

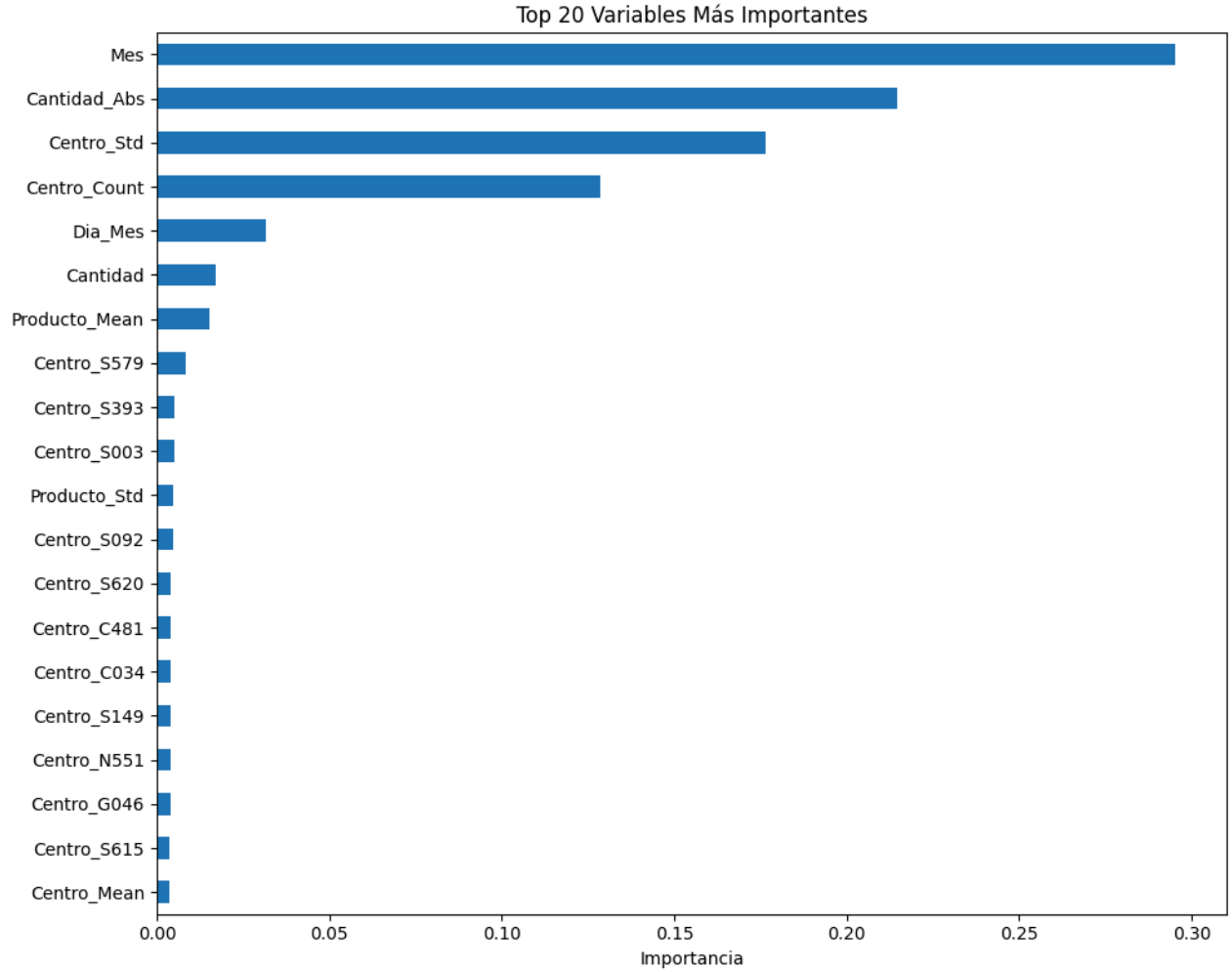
```
print("\n📊 Matriz de Confusión:")
cm_df = pd.DataFrame(cm,
                     index=['Real: Normal', 'Real: Descuadre'],
                     columns=['Pred: Normal', 'Pred: Descuadre'])
print(cm_df)

# Métricas adicionales
TN, FP, FN, TP = cm.ravel()
print(f"\n📌 Métricas Detalladas:")
print(f"   True Negatives (TN):  {TN:,}")
print(f"   False Positives (FP): {FP:,}")
print(f"   False Negatives (FN): {FN:,}")
print(f"   True Positives (TP):  {TP:,}")

precision = TP / (TP + FP) if (TP + FP) > 0 else 0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0
f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
tasa_fn = FN / (FN + TP) if (FN + TP) > 0 else 0

print(f"\n   Precisión: {precision:.4f}")
print(f"   Recall (Sensibilidad): {recall:.4f}")
print(f"   F1-Score: {f1:.4f}")
print(f"   ⚠️ Tasa de Falsos Negativos: {tasa_fn:.4f} ({tasa_fn*100:.2f}%)")

# Reporte completo
print("\n📋 Classification Report:")
print(classification_report(y_test, y_pred,
                            target_names=['Normal', 'Descuadre']))
```

```
======================================================================
📊 EVALUACIÓN DEL MODELO EN TEST
======================================================================

📈 ROC-AUC Score: 0.8091

📊 Matriz de Confusión:
                Pred: Normal  Pred: Descuadre
Real: Normal            35680             3479
Real: Descuadre         12308            11449

📌 Métricas Detalladas:
   True Negatives (TN):  35,680
   False Positives (FP): 3,479
   False Negatives (FN): 12,308
   True Positives (TP):  11,449

   Precisión: 0.7669
   Recall (Sensibilidad): 0.4819
   F1-Score: 0.5919
   ⚠️ Tasa de Falsos Negativos: 0.5181 (51.81%)

📋 Classification Report:
              precision    recall  f1-score   support

      Normal       0.74      0.91      0.82     39159
   Descuadre       0.77      0.48      0.59     23757

    accuracy                           0.75     62916
   macro avg       0.76      0.70      0.71     62916
weighted avg       0.75      0.75      0.73     62916
```

```
# 15. IMPORTANCIA DE CARACTERÍSTICAS
# ----------------------------------------------------------------------------
print("\n" + "="*70)
print("🔝 IMPORTANCIA DE CARACTERÍSTICAS")
print("="*70)

feature_importance = pd.Series(
    modelo_gbc.feature_importances_,
    index=X_train.columns
).sort_values(ascending=False)

print("\n📊 Top 20 Variables Más Importantes:")
print(feature_importance.head(20).to_string())

# Visualización
plt.figure(figsize=(10, 8))
feature_importance.head(20).plot(kind='barh')
plt.xlabel('Importancia')
plt.title('Top 20 Variables Más Importantes')
plt.gca().invert_yaxis()
```

```
plt.tight_layout()
plt.show()
```

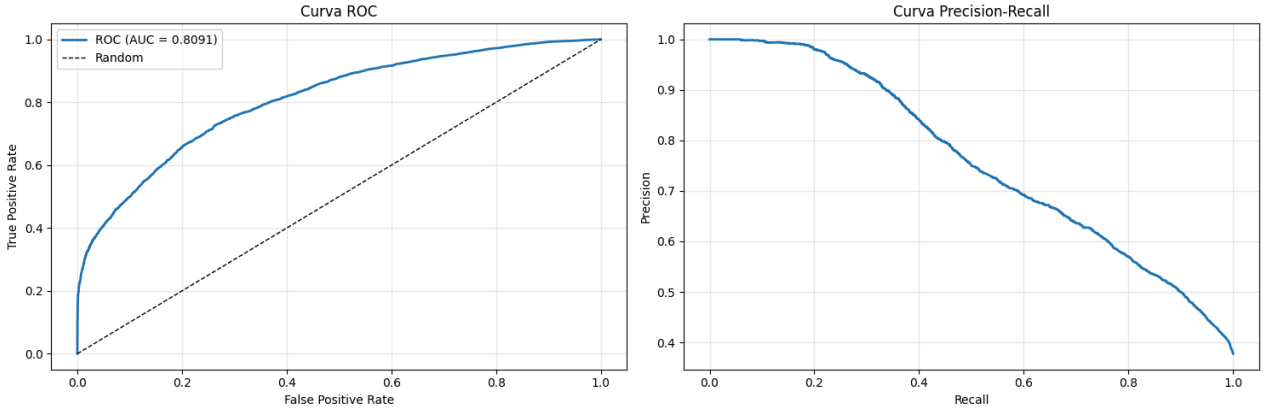
```
======================================================================
🔝 IMPORTANCIA DE CARACTERÍSTICAS
======================================================================

📊 Top 20 Variables Más Importantes:
Mes              0.295387
Cantidad_Abs     0.214803
Centro_Std       0.176371
Centro_Count     0.128677
Dia_Mes          0.031683
Cantidad         0.017217
Producto_Mean    0.015144
Centro_S579      0.008396
Centro_S393      0.005202
Centro_S003      0.005085
Producto_Std     0.004885
Centro_S092      0.004792
Centro_S620      0.004056
Centro_C481      0.003922
Centro_C034      0.003914
Centro_S149      0.003863
Centro_N551      0.003856
Centro_G046      0.003855
Centro_S615      0.003775
Centro_Mean      0.003742
```




```
# 16. CURVAS ROC Y PRECISION-RECALL
# ----------------------------------------------------------------------------
print("\n" + "="*70)
print("📈 CURVAS DE EVALUACIÓN")
print("="*70)

# Crear figura con 2 subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Curva ROC
fpr, tpr, thresholds_roc = roc_curve(y_test, y_pred_proba)
axes[0].plot(fpr, tpr, label=f'ROC (AUC = {roc_auc:.4f})', linewidth=2)
```

```
axes[0].plot([0, 1], [0, 1], 'k--', label='Random', linewidth=1)
axes[0].set_xlabel('False Positive Rate')
axes[0].set_ylabel('True Positive Rate')
axes[0].set_title('Curva ROC')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Curva Precision-Recall
precision_curve, recall_curve, thresholds_pr = precision_recall_curve(y_test, y_pred_proba)
axes[1].plot(recall_curve, precision_curve, linewidth=2)
axes[1].set_xlabel('Recall')
axes[1].set_ylabel('Precision')
axes[1].set_title('Curva Precision-Recall')
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

```
======================================================================
📈 CURVAS DE EVALUACIÓN
======================================================================
```




```
# 17. GENERAR PREDICCIONES EN TODO EL DATASET
# ------------------------------------------------------------------------------
print("\n" + "="*70)
print("🔮 GENERANDO PREDICCIONES EN TODO EL DATASET")
print("="*70)

# Crear dataset para predicción (usando todas las filas)
df_future = df.copy()

# Preparar features
df_future_encoded = pd.get_dummies(df_future,
                                   columns=['Centro', 'Producto Copec', 'Material'],
                                   drop_first=True)

# Alinear columnas con entrenamiento
X_future = df_future_encoded.reindex(columns=X_train.columns, fill_value=0)

# Escalar y predecir
X_future_scaled = scaler.transform(X_future)
future_probabilities = modelo_gbc.predict_proba(X_future_scaled)[:, 1]

# Agregar probabilidades al dataframe
df_future['Probabilidad_Descuadre'] = future_probabilities

print(f"✅ Predicciones generadas para {len(df_future):,} registros")
```

```
======================================================================
🔮 GENERANDO PREDICCIONES EN TODO EL DATASET
======================================================================
✅ Predicciones generadas para 314,576 registros
```

```
# 18. RANKING DE RIESGO
# ------------------------------------------------------------------------------
print("\n" + "="*70)
print("🏆 RANKING DE CENTROS CON MAYOR RIESGO")
print("="*70)

# Define the Config class and its attributes
class Config:
```

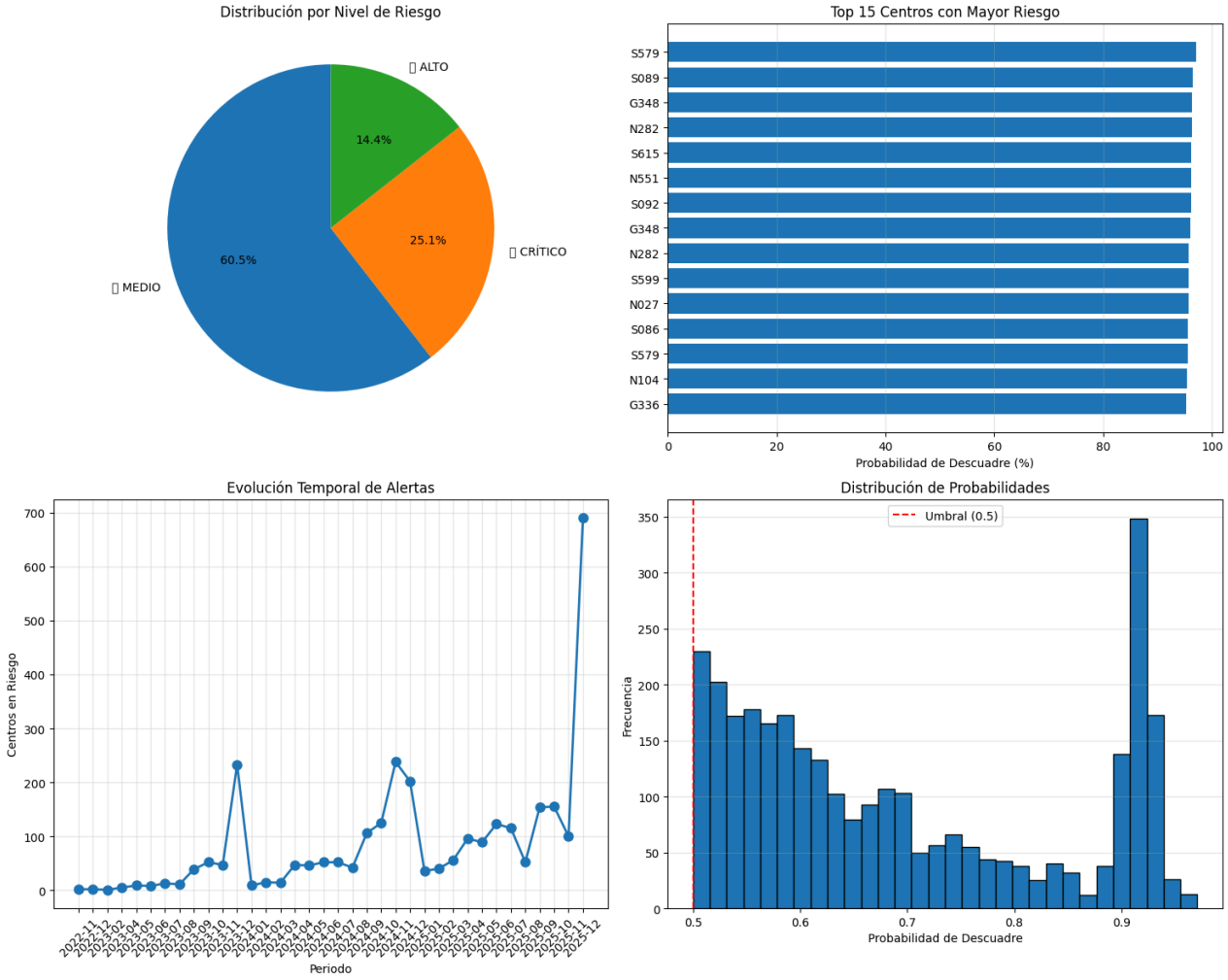
```
    UMBRAL_ALERTA = 0.5 # Example threshold, adjust as needed
    UMBRAL_RIESGO = 0.05 # Reusing the UMBRAL defined earlier
    RUTA_SALIDA_MODELO = '/content/drive/MyDrive/SAP_Descuadres_Input/modelo_des
    RUTA_SALIDA_REPORTES = '/content/drive/MyDrive/SAP_Descuadres_Input/reportes

# Agrupar por periodo y centro
ranking = df_future.groupby(['Periodo', 'Centro']).agg(
    Probabilidad_Maxima=('Probabilidad_Descuadre', 'max'),
    Probabilidad_Promedio=('Probabilidad_Descuadre', 'mean'),
    Num_Transacciones=('Centro', 'size')
).reset_index()

# Filtrar solo alertas de riesgo (probabilidad > umbral)
ranking_filtrado = ranking[ranking['Probabilidad_Maxima'] > Config.UMBRAL_ALERTA

# Clasificar nivel de riesgo
ranking_filtrado['Nivel_Riesgo'] = pd.cut(
    ranking_filtrado['Probabilidad_Maxima'],
    bins=[0.5, 0.7, 0.85, 1.0],
    labels=['🟡 MEDIO', '🟠 ALTO', '🔴 CRÍTICO']
)

# Ordenar por probabilidad
ranking_filtrado = ranking_filtrado.sort_values('Probabilidad_Maxima', ascending

# Formatear para visualización
ranking_display = ranking_filtrado.copy()
ranking_display['Probabilidad_Maxima'] = (ranking_display['Probabilidad_Maxima']
ranking_display['Probabilidad_Promedio'] = (ranking_display['Probabilidad_Promed
ranking_display['Periodo'] = ranking_display['Periodo'].astype(str)

print(f"\n⚠️ Total de Centros en Riesgo: {len(ranking_filtrado)}")
print(f"\n📊 Distribución por Nivel de Riesgo:")
print(ranking_filtrado['Nivel_Riesgo'].value_counts())

print(f"\n🔝 Top 20 Centros con Mayor Riesgo:")
print(ranking_display.head(20).to_string(index=False))
```

```
======================================================================
🏆 RANKING DE CENTROS CON MAYOR RIESGO
======================================================================

⚠️ Total de Centros en Riesgo: 3076

📊 Distribución por Nivel de Riesgo:
Nivel_Riesgo
🟡 MEDIO      1861
🔴 CRÍTICO     772
🟠 ALTO        443
Name: count, dtype: int64

🔝 Top 20 Centros con Mayor Riesgo:
Periodo Centro Probabilidad_Maxima Probabilidad_Promedio  Num_Transacciones Nivel_Riesgo
2025-12   S579              97.08%                94.82%                 40    🔴 CRÍTICO
2025-12   S089              96.45%                90.26%                 61    🔴 CRÍTICO
2025-12   G348              96.39%                93.16%                 50    🔴 CRÍTICO
2025-12   N282              96.26%                88.79%                 65    🔴 CRÍTICO
2025-12   S615              96.19%                85.47%                 40    🔴 CRÍTICO
2025-12   N551              96.18%                89.21%                 48    🔴 CRÍTICO
2025-12   S092              96.13%                88.33%                 53    🔴 CRÍTICO
2022-12   G348              96.03%                 87.5%                192    🔴 CRÍTICO
2024-12   N282              95.71%                78.92%                 48    🔴 CRÍTICO
2025-12   S599              95.69%                86.41%                 24    🔴 CRÍTICO
2025-12   N027              95.66%                83.51%                 58    🔴 CRÍTICO
2025-12   S086              95.62%                82.98%                 51    🔴 CRÍTICO
2024-12   S579              95.61%                88.21%                125    🔴 CRÍTICO
2025-12   N104              95.48%                90.86%                 29    🔴 CRÍTICO
2025-12   G336              95.32%                93.23%                 44    🔴 CRÍTICO
2025-12   N197              95.31%                87.79%                 51    🔴 CRÍTICO
2025-12   S931              95.22%                 85.6%                 54    🔴 CRÍTICO
2025-12   C096              95.11%                80.98%                 49    🔴 CRÍTICO
2025-12   G781              95.05%                92.19%                 47    🔴 CRÍTICO
2024-12   C148              95.05%                81.63%                  5    🔴 CRÍTICO
```

```
# 19. VISUALIZACIONES DEL RANKING
# ------------------------------------------------------------------------------
print("\n" + "="*70)
print("📊 VISUALIZACIONES")
print("="*70)

# Crear figura con múltiples gráficos
fig, axes = plt.subplots(2, 2, figsize=(15, 12))

# Gráfico 1: Distribución de niveles de riesgo
nivel_counts = ranking_filtrado['Nivel_Riesgo'].value_counts()
axes[0, 0].pie(nivel_counts.values, labels=nivel_counts.index, autopct='%1.1f%%', startangle=90)
```

```
axes[0, 0].set_title('Distribución por Nivel de Riesgo')

# Gráfico 2: Top 15 centros
top15 = ranking_filtrado.head(15)
axes[0, 1].barh(range(len(top15)), top15['Probabilidad_Maxima'] * 100)
axes[0, 1].set_yticks(range(len(top15)))
axes[0, 1].set_yticklabels(top15['Centro'])
axes[0, 1].set_xlabel('Probabilidad de Descuadre (%)')
axes[0, 1].set_title('Top 15 Centros con Mayor Riesgo')
axes[0, 1].invert_yaxis()
axes[0, 1].grid(True, alpha=0.3, axis='x')

# Gráfico 3: Timeline de riesgos
timeline = ranking_filtrado.groupby('Periodo').size()
axes[1, 0].plot(range(len(timeline)), timeline.values, marker='o', linewidth=2, markersize=8)
axes[1, 0].set_xlabel('Periodo')
axes[1, 0].set_ylabel('Centros en Riesgo')
axes[1, 0].set_title('Evolución Temporal de Alertas')
axes[1, 0].grid(True, alpha=0.3)
axes[1, 0].set_xticks(range(len(timeline)))
axes[1, 0].set_xticklabels([str(p) for p in timeline.index], rotation=45)

# Gráfico 4: Distribución de probabilidades
axes[1, 1].hist(ranking_filtrado['Probabilidad_Maxima'], bins=30, edgecolor='black')
axes[1, 1].set_xlabel('Probabilidad de Descuadre')
axes[1, 1].set_ylabel('Frecuencia')
axes[1, 1].set_title('Distribución de Probabilidades')
axes[1, 1].axvline(Config.UMBRAL_ALERTA, color='red', linestyle='--', label=f'Umbral ({Config.UMBRAL_ALERTA})')
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()
```

```
======================================================================
📊 VISUALIZACIONES
======================================================================
```



✦ Gemini

```
# 20. GUARDAR MODELO Y RESULTADOS
# ------------------------------------------------------------------------
import os # Import the os module

print("\n" + "="*70)
print("💾 GUARDANDO MODELO Y RESULTADOS")
print("="*70)

# Guardar modelo y objetos necesarios
modelo_completo = {
    'modelo': modelo_gbc,
    'scaler': scaler,
    'columnas_entrenamiento': X_train.columns.tolist(),
    'umbral_riesgo': Config.UMBRAL_RIESGO,
    'umbral_alerta': Config.UMBRAL_ALERTA,
    'metricas': {
        'roc_auc': roc_auc,
        'tasa_fn': tasa_fn,
        'precision': precision,
        'recall': recall,
        'f1': f1
    },
    'feature_importance': feature_importance.to_dict(),
    'fecha_entrenamiento': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
}

# Guardar el modelo
with open(Config.RUTA_SALIDA_MODELO, 'wb') as f:
    pickle.dump(modelo_completo, f)
```

```
print(f"✅ Modelo guardado en: {Config.RUTA_SALIDA_MODELO}")

# Guardar ranking en Excel
fecha_actual = datetime.now().strftime('%Y%m%d_%H%M%S')
ruta_excel = f"{Config.RUTA_SALIDA_REPORTES}ranking_riesgo_{fecha_actual}.xlsx"

# Ensure the directory exists before saving the Excel file
os.makedirs(Config.RUTA_SALIDA_REPORTES, exist_ok=True)

with pd.ExcelWriter(ruta_excel, engine='openpyxl') as writer:
    # Hoja 1: Ranking completo
    ranking_display.to_excel(writer, sheet_name='Ranking Riesgo', index=False)

    # Hoja 2: Resumen
    resumen = pd.DataFrame({
```