Credit Card Fraud: Model With Principal Parameter.

Previous Tasks

Import Libraries

In [1]:
```python
# Generic Libraries
import warnings
warnings.filterwarnings('ignore')
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import  RobustScaler
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display, HTML

from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score
from sklearn.metrics import  f1_score, fbeta_score, confusion_matrix
from sklearn.model_selection import GridSearchCV
```

In [2]:
```python
# Define general path:
path_general = r'C:\TFM'
path_total = os.path.join(path_general,'01_total_models')
```

In [3]:
```python
# Model Libraries.

# Cross validation
from sklearn.model_selection import cross_val_score

#------------- / Regresion Logistica /--------------
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression

#--------------/ XGBoost /--------------------------
from xgboost import XGBClassifier
import xgboost as xgb

#-------------/ AdaBoost /--------------------------
from sklearn.ensemble import AdaBoostClassifier

#-------------/ CatBoost /--------------------------
from catboost import CatBoostClassifier

#------------/ Decission Tree /----------------------
from sklearn.tree import DecisionTreeClassifier

#------------/ Random Forest /-----------------------
from sklearn.ensemble import RandomForestClassifier

#-----------/   MLP /-------------------------------
from sklearn.neural_network import MLPClassifier

#------------/ KNN /---------------------------------
from sklearn.neighbors import KNeighborsClassifier

#------------/ Naive - Bayes /-------------------------
from sklearn.naive_bayes import GaussianNB
```

Load Dataset

In [4]:
```python
# Load dataset.
df = pd.read_csv('creditcard.csv')
df = df.drop("Time", axis = 1)

y= df["Class"]
X = df.drop("Class", axis = 1)
y.shape,X.shape
```

Out[4]:
```
((284807,), (284807, 29))
```

In [5]:
```python
# Separation of the dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 42,stratify=y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[5]:
```
((227845, 29), (56962, 29), (227845,), (56962,))
```

In [6]:
```python
# Check dataset composition

print(" Fraudulent Count for Full data :  ",np.sum(y))
print(" Fraudulent Count for Train data : ",np.sum(y_train))
print(" Fraudulent Count for Test data :   ",np.sum(y_test))
```

```
 Fraudulent Count for Full data :   492
 Fraudulent Count for Train data :  394
 Fraudulent Count for Test data :    98
```

In [7]:
```python
# Save the testing set for evaluation
X_test_saved = X_test.copy()
y_test_saved = y_test.copy()
print("Saved X_test & y_test")
```

```
Saved X_test & y_test
```

In [8]:
```python
# As PCA is already performed on the dataset from V1 to V28 features, we are scaling only Amount field
scaler = RobustScaler()

# Scaling the train data
X_train[["Amount"]] = scaler.fit_transform(X_train[["Amount"]])

# Transforming the test data
X_test[["Amount"]] = scaler.transform(X_test[["Amount"]])
```

1.- Transformaciones de datos.

Dataset Original

Smote

In [9]:
```python
# Import of specific libraries
from collections import Counter
from imblearn.over_sampling import SMOTE

# Initial situation
print('Original dataset shape %s' % Counter(y_train))

# Calculate OverSampling model
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

print('Resampled dataset shape %s' % Counter(y_train_smote))
```

```
Original dataset shape Counter({0: 227451, 1: 394})
Resampled dataset shape Counter({0: 227451, 1: 227451})
```

Adasyn

In [10]:
```python
# Import of specific libraries
from imblearn.over_sampling import ADASYN

# Initial situation
print('Original dataset shape %s' % Counter(y_train))

# Calculate OverSampling model
adasyn = ADASYN(random_state=42)
X_train_adasyn, y_train_adasyn = adasyn.fit_resample(X_train, y_train)

print('Resampled dataset shape %s' % Counter(y_train_adasyn))
```

```
Original dataset shape Counter({0: 227451, 1: 394})
Resampled dataset shape Counter({1: 227458, 0: 227451})
```

Power Transformation

Original

In [11]:
```python
# - Apply : preprocessing.PowerTransformer(copy=False) to fit & transform the train & test data

from sklearn import metrics
from sklearn import preprocessing


from sklearn.preprocessing import PowerTransformer

pt= preprocessing.PowerTransformer(method='yeo-johnson', copy=True)  # creates an instance of the PowerTransformer class.
pt.fit(X_train)

X_train_pt = pt.transform(X_train)
X_test_pt = pt.transform(X_test)

y_train_pt = y_train
y_test_pt = y_test
```

Smote

In [12]:
```python
# Import of specific libraries
from collections import Counter
from imblearn.over_sampling import SMOTE

# Initial situation
print('Original dataset shape %s' % Counter(y_train_pt))

# Calculate OverSampling model
smote = SMOTE(random_state=42)
X_train_smote_pt, y_train_smote_pt = smote.fit_resample(X_train_pt, y_train_pt)

print('Resampled dataset shape %s' % Counter(y_train_smote_pt))
```

```
Original dataset shape Counter({0: 227451, 1: 394})
Resampled dataset shape Counter({0: 227451, 1: 227451})
```

Adasyn

In [13]:
```python
# Import of specific libraries
from imblearn.over_sampling import ADASYN

# Initial situation
print('Original dataset shape %s' % Counter(y_train))

# Calculate OverSampling model
adasyn = ADASYN(random_state=42)
X_train_adasyn_pt, y_train_adasyn_pt = adasyn.fit_resample(X_train_pt, y_train_pt)

print('Resampled dataset shape %s' % Counter(y_train_adasyn_pt))
```

```
Original dataset shape Counter({0: 227451, 1: 394})
Resampled dataset shape Counter({1: 227459, 0: 227451})
```

Load Model: Libraries and Functions.

In [14]:
```python
# LOAD OF MODELS.
# perfom cross validation on the X_train & y_train
from sklearn.model_selection import StratifiedKFold

# Initialize StratifiedKFold cross-validator
# perform cross validation
skf = StratifiedKFold(n_splits=3, random_state=None, shuffle=False)
#  Shuffle is False because we need a constant best model when we use GridSearchCV
```

In [15]:
```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict
```

Create dataset_list

In [16]:
```python
# Original distribution
OR_origin = ['OR origin',X_train, y_train, X_test, y_test]
OR_smote =['OR smote',X_train_smote, y_train_smote, X_test, y_test]
OR_adasyn = ['OR adasyn', X_train_adasyn, y_train_adasyn, X_test, y_test]

# Power Transformation
PT_origin = ['PT origin',X_train_pt, y_train_pt, X_test_pt, y_test_pt]
PT_smote = ['PT smote',X_train_smote_pt, y_train_smote_pt, X_test_pt, y_test_pt ]
PT_adasyn = ['PT adasyn', X_train_adasyn_pt, y_train_adasyn_pt, X_test_pt, y_test_pt]
```

Create models

In [17]:
```python
model_list = ['regression_logistic', 'adaboost', 'xgboost', 'catboost', 'decision_tree', 'random_forest', 'knn']
parameters = [
    [0.1, 0.5, 1, 1.5, 2, 2.5, 3],   # For 'regression_logistic'
    [10, 30, 50, 70, 100 ],          # For 'adaboost'
    [0.001, 0.01, 0.1, 0.5, 1, 3],   # For 'xgboost'
    [100, 200, 300, 400, 500, 600],  # For 'catboost'
    [1, 2, 3, 4, 5],                 # For 'decision_tree'
    [100, 200, 300, 400, 500, 600],  # For 'random_forest'
    [3, 5, 7]                        #knn
    ]
```

In [18]:
```python
distributions =[OR_origin, OR_smote, OR_adasyn, PT_origin, PT_smote, PT_adasyn]
complete_model = zip(model_list, parameters)
complete_model_list = list(complete_model )
complete_model_list
```

Out[18]:
```
[('regression_logistic', [0.1, 0.5, 1, 1.5, 2, 2.5, 3]),
 ('adaboost', [10, 30, 50, 70, 100]),
 ('xgboost', [0.001, 0.01, 0.1, 0.5, 1, 3]),
 ('catboost', [100, 200, 300, 400, 500, 600]),
 ('decision_tree', [1, 2, 3, 4, 5]),
 ('random_forest', [100, 200, 300, 400, 500, 600]),
 ('knn', [3, 5, 7])]
```

In [19]:
```python
def gen_models(complete_model_list, distributions, save_directory_complete_model=None):

    # test if directory is None and add path
    if save_directory_complete_model is None:
        save_directory_complete_model = os.path.join(os.getcwd(), 'total')

    #Create folder if not exits
    os.makedirs(save_directory_complete_model, exist_ok=True)


    # 1.- Iterate model
    for model_name, param_values in complete_model_list:
        #print(f"Processing model: {model_name}")

        resultados_totales = []

        # 2.- Iterate over distributions:
        for distribution in distributions:
            try:
                # Unpack distribution
                name = distribution[0]  # Nombre de la distribución
                X_train, y_train, X_val, y_val = distribution[1:]

                #print(f"  Distribution: {name}")  # Log distribución

                # Verify if data is valid:
                if X_train is None or y_train is None or X_val is None or y_val is None:
                    print(f"    Skipping due to missing data in {name}")
                    continue

                # 3.- Iterate over params:
                for param in param_values:
                    #print(f"    Training {model_name} with parameter {param}")  # Log parámetro

                    # Inicializate model and parameters
                    model_instance = None
                    if model_name == 'regression_logistic':
                        model_instance = LogisticRegression(C=param)
                        parameter_name ='C='
                    elif model_name == 'adaboost':
                        #model_instance = AdaBoostClassifier(DecisionTreeClassifier(max_depth=param))
                        model_instance = AdaBoostClassifier(DecisionTreeClassifier(max_depth=param))
                        #parameter_name = 'max_depth='
                        parameter_name = 'n_estimators='
                    elif model_name == 'xgboost':
                        model_instance = XGBClassifier(learning_rate=param)
                        parameter_name  = 'learning_rate='
                    elif model_name == 'catboost':
                        model_instance = CatBoostClassifier(iterations=param, verbose=0)
                        parameter_name  = 'iterations='
                    elif model_name == 'decision_tree':
                        model_instance = DecisionTreeClassifier(max_depth=param)
                        parameter_name  = 'max_depth='
                    elif model_name == 'random_forest':
                        model_instance = RandomForestClassifier(n_estimators=param)
                        parameter_name  = 'n_estimators'
                    elif model_name == 'mlp':
                        model_instance = MLPClassifier(hidden_layer_sizes=param)
                        parameter_name  = 'hidden_layer_sizes'
                    elif model_name == 'knn':
                        model_instance = KNeighborsClassifier(n_neighbors=param)
                        parameter_name  = 'n_neighbors'
                    else:
                        print(f"    Invalid model name: {model_name}")
                        continue


                    # Train model
                    model_instance.fit(X_train, y_train)

                    # Get predictions
                    y_pred = model_instance.predict(X_val)

                    # Calculate metrics
                    roc_auc = roc_auc_score(y_val, y_pred)
                    accuracy = accuracy_score(y_val, y_pred)
                    precision = precision_score(y_val, y_pred)
                    recall = recall_score(y_val, y_pred)
                    f1 = f1_score(y_val, y_pred)
                    f2 = fbeta_score(y_val, y_pred, beta=2)
                    confusion = confusion_matrix(y_val, y_pred)

                    # Save results in DataFrame:
                    results_df = pd.DataFrame({
                        'Model': [model_name],
                        'Description': [name],
                        'Parameter':[parameter_name + str(param)],
                        'ROC-AUC': [roc_auc],
                        'Accuracy': [accuracy],
                        'Precision': [precision],
                        'Recall': [recall],
                        'F1 Score': [f1],
                        'F2 Score': [f2],
                        'Confusion Matrix': [confusion],
                    })

                    resultados_totales.append(results_df)

            except Exception as e:
                print(f"  Error processing distribution {name}: {str(e)}")
                continue

        # Save results:
        if resultados_totales:
            df_resultados_final = pd.concat(resultados_totales, ignore_index=True)
            save_path = os.path.join(save_directory_complete_model, f"{model_name}_total.csv")
            df_resultados_final.to_csv(save_path, index=False)
            display(HTML(f"<h2 style='text-align: center;font-size:60px;'> Modelo: {model_name}</h2>"))
            display(df_resultados_final)
            print(f"\n\n\nResults for {model_name} saved to {save_path}")
        else:
            print(f"No results generated for {model_name}")
```

In [ ]:
```python
gen_models(complete_model_list, distributions, path_total)
```

Modelo: regression_logistic

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	regression_logistic	OR origin	C=0.1	0.821314	0.999157	0.828947	0.642857	0.724138	0.673077	[[56851, 13], [35, 63]]
1	regression_logistic	OR origin	C=0.5	0.826416	0.999175	0.831169	0.653061	0.731429	0.682303	[[56851, 13], [34, 64]]
2	regression_logistic	OR origin	C=1	0.826416	0.999175	0.831169	0.653061	0.731429	0.682303	[[56851, 13], [34, 64]]
3	regression_logistic	OR origin	C=1.5	0.826416	0.999175	0.831169	0.653061	0.731429	0.682303	[[56851, 13], [34, 64]]
4	regression_logistic	OR origin	C=2	0.826416	0.999175	0.831169	0.653061	0.731429	0.682303	[[56851, 13], [34, 64]]
5	regression_logistic	OR origin	C=2.5	0.826416	0.999175	0.831169	0.653061	0.731429	0.682303	[[56851, 13], [34, 64]]
6	regression_logistic	OR origin	C=3	0.826416	0.999175	0.831169	0.653061	0.731429	0.682303	[[56851, 13], [34, 64]]
7	regression_logistic	OR smote	C=0.1	0.946056	0.973649	0.056854	0.918367	0.107079	0.227848	[[55371, 1493], [8, 90]]
8	regression_logistic	OR smote	C=0.5	0.946038	0.973614	0.056782	0.918367	0.106952	0.227618	[[55369, 1495], [8, 90]]
9	regression_logistic	OR smote	C=1	0.946029	0.973596	0.056747	0.918367	0.106888	0.227503	[[55368, 1496], [8, 90]]
10	regression_logistic	OR smote	C=1.5	0.946038	0.973614	0.056782	0.918367	0.106952	0.227618	[[55369, 1495], [8, 90]]
11	regression_logistic	OR smote	C=2	0.946038	0.973614	0.056782	0.918367	0.106952	0.227618	[[55369, 1495], [8, 90]]
12	regression_logistic	OR smote	C=2.5	0.946038	0.973614	0.056782	0.918367	0.106952	0.227618	[[55369, 1495], [8, 90]]
13	regression_logistic	OR smote	C=3	0.946047	0.973632	0.056818	0.918367	0.107015	0.227733	[[55370, 1494], [8, 90]]
14	regression_logistic	OR adasyn	C=0.1	0.922704	0.916857	0.018880	0.928571	0.037007	0.087299	[[52135, 4729], [7, 91]]
15	regression_logistic	OR adasyn	C=0.5	0.922686	0.916822	0.018872	0.928571	0.036992	0.087265	[[52133, 4731], [7, 91]]
16	regression_logistic	OR adasyn	C=1	0.922704	0.916857	0.018880	0.928571	0.037007	0.087299	[[52135, 4729], [7, 91]]
17	regression_logistic	OR adasyn	C=1.5	0.922678	0.916804	0.018868	0.928571	0.036984	0.087248	[[52132, 4732], [7, 91]]
18	regression_logistic	OR adasyn	C=2	0.922713	0.916874	0.018884	0.928571	0.037014	0.087315	[[52136, 4728], [7, 91]]
19	regression_logistic	OR adasyn	C=2.5	0.922722	0.916892	0.018888	0.928571	0.037022	0.087332	[[52137, 4727], [7, 91]]
20	regression_logistic	OR adasyn	C=3	0.922686	0.916822	0.018872	0.928571	0.036992	0.087265	[[52133, 4731], [7, 91]]
21	regression_logistic	PT origin	C=0.1	0.836612	0.999192	0.825000	0.673469	0.741573	0.699153	[[56850, 14], [32, 66]]
22	regression_logistic	PT origin	C=0.5	0.836603	0.999175	0.814815	0.673469	0.737430	0.697674	[[56849, 15], [32, 66]]
23	regression_logistic	PT origin	C=1	0.836603	0.999175	0.814815	0.673469	0.737430	0.697674	[[56849, 15], [32, 66]]
24	regression_logistic	PT origin	C=1.5	0.836603	0.999175	0.814815	0.673469	0.737430	0.697674	[[56849, 15], [32, 66]]
25	regression_logistic	PT origin	C=2	0.836603	0.999175	0.814815	0.673469	0.737430	0.697674	[[56849, 15], [32, 66]]
26	regression_logistic	PT origin	C=2.5	0.836603	0.999175	0.814815	0.673469	0.737430	0.697674	[[56849, 15], [32, 66]]
27	regression_logistic	PT origin	C=3	0.836603	0.999175	0.814815	0.673469	0.737430	0.697674	[[56849, 15], [32, 66]]
28	regression_logistic	PT smote	C=0.1	0.944860	0.971262	0.052356	0.918367	0.099064	0.213169	[[55235, 1629], [8, 90]]
29	regression_logistic	PT smote	C=0.5	0.944816	0.971174	0.052204	0.918367	0.098793	0.212665	[[55230, 1634], [8, 90]]
30	regression_logistic	PT smote	C=1	0.944807	0.971156	0.052174	0.918367	0.098738	0.212565	[[55229, 1635], [8, 90]]
31	regression_logistic	PT smote	C=1.5	0.944798	0.971139	0.052144	0.918367	0.098684	0.212465	[[55228, 1636], [8, 90]]
32	regression_logistic	PT smote	C=2	0.944790	0.971121	0.052113	0.918367	0.098630	0.212364	[[55227, 1637], [8, 90]]
33	regression_logistic	PT smote	C=2.5	0.944790	0.971121	0.052113	0.918367	0.098630	0.212364	[[55227, 1637], [8, 90]]
34	regression_logistic	PT smote	C=3	0.944790	0.971121	0.052113	0.918367	0.098630	0.212364	[[55227, 1637], [8, 90]]
35	regression_logistic	PT adasyn	C=0.1	0.920339	0.912134	0.017882	0.928571	0.035088	0.083014	[[51866, 4998], [7, 91]]
36	regression_logistic	PT adasyn	C=0.5	0.920330	0.912117	0.017878	0.928571	0.035081	0.082999	[[51865, 4999], [7, 91]]
37	regression_logistic	PT adasyn	C=1	0.920330	0.912117	0.017878	0.928571	0.035081	0.082999	[[51865, 4999], [7, 91]]
38	regression_logistic	PT adasyn	C=1.5	0.920330	0.912117	0.017878	0.928571	0.035081	0.082999	[[51865, 4999], [7, 91]]
39	regression_logistic	PT adasyn	C=2	0.920330	0.912117	0.017878	0.928571	0.035081	0.082999	[[51865, 4999], [7, 91]]
40	regression_logistic	PT adasyn	C=2.5	0.920330	0.912117	0.017878	0.928571	0.035081	0.082999	[[51865, 4999], [7, 91]]
41	regression_logistic	PT adasyn	C=3	0.920330	0.912117	0.017878	0.928571	0.035081	0.082999	[[51865, 4999], [7, 91]]

```
Results for regression_logistic saved to C:\TFM\01_total_models\regression_logistic_total.csv
```

Modelo: adaboost

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	adaboost	OR origin	n_estimators=10	0.882609	0.999508	0.937500	0.765306	0.842697	0.794492	[[56859, 5], [23, 75]]
1	adaboost	OR origin	n_estimators=30	0.887518	0.999140	0.737864	0.775510	0.756219	0.767677	[[56837, 27], [22, 76]]
2	adaboost	OR origin	n_estimators=50	0.882451	0.999192	0.765306	0.765306	0.765306	0.765306	[[56841, 23], [23, 75]]
3	adaboost	OR origin	n_estimators=70	0.882468	0.999228	0.781250	0.765306	0.773196	0.768443	[[56843, 21], [23, 75]]
4	adaboost	OR origin	n_estimators=100	0.887518	0.999140	0.737864	0.775510	0.756219	0.767677	[[56837, 27], [22, 76]]
5	adaboost	OR smote	n_estimators=10	0.933568	0.999561	0.876289	0.867347	0.871795	0.869121	[[56852, 12], [13, 85]]
6	adaboost	OR smote	n_estimators=30	0.902973	0.999491	0.887640	0.806122	0.844920	0.821206	[[56854, 10], [19, 79]]
7	adaboost	OR smote	n_estimators=50	0.891617	0.997156	0.353211	0.785714	0.487342	0.631148	[[56723, 141], [21, 77]]
8	adaboost	OR smote	n_estimators=70	0.896684	0.997103	0.349776	0.795918	0.485981	0.634146	[[56719, 145], [20, 78]]
9	adaboost	OR smote	n_estimators=100	0.881369	0.997033	0.339367	0.765306	0.470219	0.611746	[[56718, 146], [23, 75]]
10	adaboost	OR adasyn	n_estimators=10	0.908067	0.999491	0.879121	0.816327	0.846561	0.828157	[[56853, 11], [18, 80]]
11	adaboost	OR adasyn	n_estimators=30	0.902991	0.999526	0.908046	0.806122	0.854054	0.824635	[[56856, 8], [19, 79]]
12	adaboost	OR adasyn	n_estimators=50	0.887694	0.999491	0.915663	0.775510	0.839779	0.800000	[[56857, 7], [22, 76]]
13	adaboost	OR adasyn	n_estimators=70	0.861313	0.997665	0.401130	0.724490	0.516364	0.623902	[[56758, 106], [27, 71]]
14	adaboost	OR adasyn	n_estimators=100	0.866353	0.997560	0.389189	0.734694	0.508834	0.623917	[[56751, 113], [26, 72]]
15	adaboost	PT origin	n_estimators=10	0.908128	0.999614	0.952381	0.816327	0.879121	0.840336	[[56860, 4], [18, 80]]
16	adaboost	PT origin	n_estimators=30	0.872229	0.999122	0.744898	0.744898	0.744898	0.744898	[[56839, 25], [25, 73]]
17	adaboost	PT origin	n_estimators=50	0.877331	0.999140	0.747475	0.755102	0.751269	0.753564	[[56839, 25], [24, 74]]
18	adaboost	PT origin	n_estimators=70	0.867136	0.999122	0.750000	0.734694	0.742268	0.737705	[[56840, 24], [26, 72]]
19	adaboost	PT origin	n_estimators=100	0.877322	0.999122	0.740000	0.755102	0.747475	0.752033	[[56838, 26], [24, 74]]
20	adaboost	PT smote	n_estimators=10	0.928501	0.999614	0.913043	0.857143	0.884211	0.867769	[[56856, 8], [14, 84]]
21	adaboost	PT smote	n_estimators=30	0.908102	0.999561	0.919540	0.816327	0.864865	0.835073	[[56857, 7], [18, 80]]
22	adaboost	PT smote	n_estimators=50	0.907117	0.997595	0.402010	0.816327	0.538721	0.676819	[[56745, 119], [18, 80]]
23	adaboost	PT smote	n_estimators=70	0.902015	0.997577	0.398990	0.806122	0.533784	0.669492	[[56745, 119], [19, 79]]
24	adaboost	PT smote	n_estimators=100	0.896842	0.997419	0.380488	0.795918	0.514851	0.653266	[[56737, 127], [20, 78]]
25	adaboost	PT adasyn	n_estimators=10	0.918288	0.999561	0.901099	0.836735	0.867725	0.848861	[[56855, 9], [16, 82]]
26	adaboost	PT adasyn	n_estimators=30	0.902991	0.999526	0.908046	0.806122	0.854054	0.824635	[[56856, 8], [19, 79]]
27	adaboost	PT adasyn	n_estimators=50	0.836629	0.999228	0.846154	0.673469	0.750000	0.702128	[[56852, 12], [32, 66]]
28	adaboost	PT adasyn	n_estimators=70	0.881510	0.997314	0.365854	0.765306	0.495050	0.628141	[[56734, 130], [23, 75]]
29	adaboost	PT adasyn	n_estimators=100	0.876531	0.997542	0.389474	0.755102	0.513889	0.635739	[[56748, 116], [24, 74]]

```
Results for adaboost saved to C:\TFM\01_total_models\adaboost_total.csv
```

Modelo: xgboost

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	xgboost	OR origin	learning_rate=0.001	0.892778	0.999473	0.895349	0.785714	0.836957	0.805439	[[56855, 9], [21, 77]]
1	xgboost	OR origin	learning_rate=0.01	0.908084	0.999526	0.898876	0.816327	0.855615	0.831601	[[56855, 9], [18, 80]]
2	xgboost	OR origin	learning_rate=0.1	0.908111	0.999579	0.930233	0.816327	0.869565	0.836820	[[56858, 6], [18, 80]]
3	xgboost	OR origin	learning_rate=0.5	0.913204	0.999579	0.920455	0.826531	0.870968	0.843750	[[56857, 7], [17, 81]]
4	xgboost	OR origin	learning_rate=1	0.908102	0.999561	0.919540	0.816327	0.864865	0.835073	[[56857, 7], [18, 80]]
5	xgboost	OR origin	learning_rate=3	0.688512	0.998402	0.552239	0.377551	0.448485	0.403050	[[56834, 30], [61, 37]]
6	xgboost	OR smote	learning_rate=0.001	0.915927	0.974509	0.055191	0.857143	0.103704	0.219436	[[55426, 1438], [14, 84]]
7	xgboost	OR smote	learning_rate=0.01	0.926868	0.986184	0.098952	0.867347	0.177638	0.339728	[[56090, 774], [13, 85]]
8	xgboost	OR smote	learning_rate=0.1	0.937360	0.996963	0.348178	0.877551	0.498551	0.672926	[[56703, 161], [12, 86]]
9	xgboost	OR smote	learning_rate=0.5	0.943623	0.999298	0.750000	0.887755	0.813084	0.856299	[[56835, 29], [11, 87]]
10	xgboost	OR smote	learning_rate=1	0.923197	0.999192	0.728070	0.846939	0.783019	0.820158	[[56833, 31], [15, 83]]
11	xgboost	OR smote	learning_rate=3	0.927191	0.956322	0.034295	0.897959	0.066066	0.148749	[[54386, 2478], [10, 88]]
12	xgboost	OR adasyn	learning_rate=0.001	0.912929	0.927847	0.021012	0.897959	0.041064	0.096070	[[52764, 4100], [10, 88]]
13	xgboost	OR adasyn	learning_rate=0.01	0.917646	0.957603	0.034552	0.877551	0.066486	0.149254	[[54461, 2403], [12, 86]]
14	xgboost	OR adasyn	learning_rate=0.1	0.935909	0.994066	0.208738	0.877551	0.337255	0.534826	[[56538, 326], [12, 86]]
15	xgboost	OR adasyn	learning_rate=0.5	0.923232	0.999263	0.754545	0.846939	0.798077	0.826693	[[56837, 27], [15, 83]]
16	xgboost	OR adasyn	learning_rate=1	0.913028	0.999228	0.750000	0.826531	0.786408	0.810000	[[56837, 27], [17, 81]]
17	xgboost	OR adasyn	learning_rate=3	0.795130	0.692655	0.005004	0.897959	0.009953	0.024476	[[39367, 17497], [10, 88]]
18	xgboost	PT origin	learning_rate=0.001	0.892778	0.999473	0.895349	0.785714	0.836957	0.805439	[[56855, 9], [21, 77]]
19	xgboost	PT origin	learning_rate=0.01	0.908084	0.999526	0.898876	0.816327	0.855615	0.831601	[[56855, 9], [18, 80]]
20	xgboost	PT origin	learning_rate=0.1	0.908111	0.999579	0.930233	0.816327	0.869565	0.836820	[[56858, 6], [18, 80]]
21	xgboost	PT origin	learning_rate=0.5	0.913204	0.999579	0.920455	0.826531	0.870968	0.843750	[[56857, 7], [17, 81]]
22	xgboost	PT origin	learning_rate=1	0.908102	0.999561	0.919540	0.816327	0.864865	0.835073	[[56857, 7], [18, 80]]
23	xgboost	PT origin	learning_rate=3	0.688512	0.998402	0.552239	0.377551	0.448485	0.403050	[[56834, 30], [61, 37]]
24	xgboost	PT smote	learning_rate=0.001	0.949663	0.970682	0.051881	0.928571	0.098272	0.212022	[[55201, 1663], [7, 91]]
25	xgboost	PT smote	learning_rate=0.01	0.942165	0.986219	0.101970	0.897959	0.183143	0.350598	[[56089, 775], [10, 88]]
26	xgboost	PT smote	learning_rate=0.1	0.942462	0.996980	0.350806	0.887755	0.502890	0.679688	[[56703, 161], [11, 87]]
27	xgboost	PT smote	learning_rate=0.5	0.933427	0.999280	0.752212	0.867347	0.805687	0.841584	[[56836, 28], [13, 85]]
28	xgboost	PT smote	learning_rate=1	0.923241	0.999280	0.761468	0.846939	0.801932	0.828343	[[56838, 26], [15, 83]]
29	xgboost	PT smote	learning_rate=3	0.914628	0.961746	0.037761	0.867347	0.072371	0.160802	[[54698, 2166], [13, 85]]
30	xgboost	PT adasyn	learning_rate=0.001	0.933078	0.947737	0.029421	0.918367	0.057016	0.130397	[[53895, 2969], [8, 90]]
31	xgboost	PT adasyn	learning_rate=0.01	0.922194	0.956515	0.034078	0.887755	0.065636	0.147708	[[54398, 2466], [11, 87]]
32	xgboost	PT adasyn	learning_rate=0.1	0.930614	0.993662	0.196305	0.867347	0.320151	0.515152	[[56516, 348], [13, 85]]
33	xgboost	PT adasyn	learning_rate=0.5	0.923206	0.999210	0.734513	0.846939	0.786730	0.821782	[[56834, 30], [15, 83]]
34	xgboost	PT adasyn	learning_rate=1	0.912993	0.999157	0.723214	0.826531	0.771429	0.803571	[[56833, 31], [17, 81]]
35	xgboost	PT adasyn	learning_rate=3	0.762556	0.566606	0.003794	0.959184	0.007558	0.018674	[[32181, 24683], [4, 94]]

```
Results for xgboost saved to C:\TFM\01_total_models\xgboost_total.csv
```

Modelo: catboost

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	catboost	OR origin	iterations=100	0.913221	0.999614	0.941860	0.826531	0.880435	0.847280	[[56859, 5], [17, 81]]
1	catboost	OR origin	iterations=200	0.913239	0.999649	0.964286	0.826531	0.890110	0.850840	[[56861, 3], [17, 81]]
2	catboost	OR origin	iterations=300	0.903017	0.999579	0.940476	0.806122	0.868132	0.829832	[[56859, 5], [19, 79]]
3	catboost	OR origin	iterations=400	0.897915	0.999561	0.939759	0.795918	0.861878	0.821053	[[56859, 5], [20, 78]]
4	catboost	OR origin	iterations=500	0.908146	0.999649	0.975610	0.816327	0.888889	0.843882	[[56862, 2], [18, 80]]
5	catboost	OR origin	iterations=600	0.908128	0.999614	0.952381	0.816327	0.879121	0.840336	[[56860, 4], [18, 80]]
6	catboost	OR smote	iterations=100	0.933093	0.998613	0.562914	0.867347	0.682731	0.782689	[[56798, 66], [13, 85]]
7	catboost	OR smote	iterations=200	0.933225	0.998876	0.625000	0.867347	0.726496	0.804924	[[56813, 51], [13, 85]]
8	catboost	OR smote	iterations=300	0.922951	0.998701	0.584507	0.846939	0.691667	0.777154	[[56805, 59], [15, 83]]
9	catboost	OR smote	iterations=400	0.923056	0.998912	0.638462	0.846939	0.728070	0.795019	[[56817, 47], [15, 83]]
10	catboost	OR smote	iterations=500	0.933216	0.998859	0.620438	0.867347	0.723404	0.803403	[[56812, 52], [13, 85]]
11	catboost	OR smote	iterations=600	0.922871	0.998543	0.549669	0.846939	0.666667	0.764273	[[56796, 68], [15, 83]]
12	catboost	OR adasyn	iterations=100	0.907389	0.998139	0.476190	0.816327	0.601504	0.714286	[[56776, 88], [18, 80]]
13	catboost	OR adasyn	iterations=200	0.907389	0.998139	0.476190	0.816327	0.601504	0.714286	[[56776, 88], [18, 80]]
14	catboost	OR adasyn	iterations=300	0.922784	0.998367	0.515528	0.846939	0.640927	0.750452	[[56786, 78], [15, 83]]
15	catboost	OR adasyn	iterations=400	0.917796	0.998578	0.557823	0.836735	0.669388	0.760668	[[56799, 65], [16, 82]]
16	catboost	OR adasyn	iterations=500	0.922863	0.998525	0.546053	0.846939	0.664000	0.762868	[[56795, 69], [15, 83]]
17	catboost	OR adasyn	iterations=600	0.927499	0.997612	0.407767	0.857143	0.552632	0.702341	[[56742, 122], [14, 84]]
18	catboost	PT origin	iterations=100	0.913221	0.999614	0.941860	0.826531	0.880435	0.847280	[[56859, 5], [17, 81]]
19	catboost	PT origin	iterations=200	0.913239	0.999649	0.964286	0.826531	0.890110	0.850840	[[56861, 3], [17, 81]]
20	catboost	PT origin	iterations=300	0.903017	0.999579	0.940476	0.806122	0.868132	0.829832	[[56859, 5], [19, 79]]
21	catboost	PT origin	iterations=400	0.897915	0.999561	0.939759	0.795918	0.861878	0.821053	[[56859, 5], [20, 78]]
22	catboost	PT origin	iterations=500	0.908146	0.999649	0.975610	0.816327	0.888889	0.843882	[[56862, 2], [18, 80]]
23	catboost	PT origin	iterations=600	0.908128	0.999614	0.952381	0.816327	0.879121	0.840336	[[56860, 4], [18, 80]]
24	catboost	PT smote	iterations=100	0.912597	0.998367	0.515924	0.826531	0.635294	0.737705	[[56788, 76], [17, 81]]
25	catboost	PT smote	iterations=200	0.912597	0.998367	0.515924	0.826531	0.635294	0.737705	[[56788, 76], [17, 81]]
26	catboost	PT smote	iterations=300	0.922951	0.998701	0.584507	0.846939	0.691667	0.777154	[[56805, 59], [15, 83]]
27	catboost	PT smote	iterations=400	0.917884	0.998754	0.598540	0.836735	0.697872	0.775047	[[56809, 55], [16, 82]]
28	catboost	PT smote	iterations=500	0.923003	0.998806	0.610294	0.846939	0.709402	0.785985	[[56811, 53], [15, 83]]
29	catboost	PT smote	iterations=600	0.933216	0.998859	0.620438	0.867347	0.723404	0.803403	[[56812, 52], [13, 85]]
30	catboost	PT adasyn	iterations=100	0.917356	0.997700	0.416244	0.836735	0.555932	0.696095	[[56749, 115], [16, 82]]
31	catboost	PT adasyn	iterations=200	0.912553	0.998280	0.500000	0.826531	0.623077	0.731047	[[56783, 81], [17, 81]]
32	catboost	PT adasyn	iterations=300	0.922616	0.998034	0.461111	0.846939	0.597122	0.725524	[[56767, 97], [15, 83]]
33	catboost	PT adasyn	iterations=400	0.912694	0.998560	0.554795	0.826531	0.663934	0.752788	[[56799, 65], [17, 81]]
34	catboost	PT adasyn	iterations=500	0.917849	0.998683	0.581560	0.836735	0.686192	0.769231	[[56805, 59], [16, 82]]
35	catboost	PT adasyn	iterations=600	0.922792	0.998385	0.518750	0.846939	0.643411	0.751812	[[56787, 77], [15, 83]]

```
Results for catboost saved to C:\TFM\01_total_models\catboost_total.csv
```

Modelo: decision_tree

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	decision_tree	OR origin	max_depth=1	0.862007	0.999052	0.724490	0.724490	0.724490	0.724490	[[56837, 27], [27, 71]]
1	decision_tree	OR origin	max_depth=2	0.882451	0.999192	0.765306	0.765306	0.765306	0.765306	[[56841, 23], [23, 75]]
2	decision_tree	OR origin	max_depth=3	0.882460	0.999210	0.773196	0.765306	0.769231	0.766871	[[56842, 22], [23, 75]]
3	decision_tree	OR origin	max_depth=4	0.892725	0.999368	0.836957	0.785714	0.810526	0.795455	[[56849, 15], [21, 77]]
4	decision_tree	OR origin	max_depth=5	0.892787	0.999491	0.905882	0.785714	0.841530	0.807128	[[56856, 8], [21, 77]]
5	decision_tree	OR smote	max_depth=1	0.923309	0.968909	0.046612	0.877551	0.088523	0.192222	[[55105, 1759], [12, 86]]
6	decision_tree	OR smote	max_depth=2	0.919462	0.981567	0.075000	0.857143	0.137931	0.277778	[[55828, 1036], [14, 84]]
7	decision_tree	OR smote	max_depth=3	0.918932	0.909326	0.017337	0.928571	0.034038	0.080659	[[51706, 5158], [7, 91]]
8	decision_tree	OR smote	max_depth=4	0.921390	0.975247	0.057355	0.867347	0.107595	0.226788	[[55467, 1397], [13, 85]]
9	decision_tree	OR smote	max_depth=5	0.924480	0.961079	0.037942	0.887755	0.072773	0.162011	[[54658, 2206], [11, 87]]
10	decision_tree	OR adasyn	max_depth=1	0.887238	0.866385	0.011572	0.908163	0.022853	0.055054	[[49262, 7602], [9, 89]]
11	decision_tree	OR adasyn	max_depth=2	0.887315	0.836031	0.009760	0.938776	0.019320	0.046853	[[47530, 9334], [6, 92]]
12	decision_tree	OR adasyn	max_depth=3	0.907382	0.865928	0.012039	0.948980	0.023776	0.057287	[[49232, 7632], [5, 93]]
13	decision_tree	OR adasyn	max_depth=4	0.907873	0.877076	0.012980	0.938776	0.025605	0.061497	[[49868, 6996], [6, 92]]
14	decision_tree	OR adasyn	max_depth=5	0.910711	0.892911	0.014715	0.928571	0.028972	0.069191	[[50771, 6093], [7, 91]]
15	decision_tree	PT origin	max_depth=1	0.862007	0.999052	0.724490	0.724490	0.724490	0.724490	[[56837, 27], [27, 71]]
16	decision_tree	PT origin	max_depth=2	0.882451	0.999192	0.765306	0.765306	0.765306	0.765306	[[56841, 23], [23, 75]]
17	decision_tree	PT origin	max_depth=3	0.882468	0.999228	0.781250	0.765306	0.773196	0.768443	[[56843, 21], [23, 75]]
18	decision_tree	PT origin	max_depth=4	0.892725	0.999368	0.836957	0.785714	0.810526	0.795455	[[56849, 15], [21, 77]]
19	decision_tree	PT origin	max_depth=5	0.887694	0.999491	0.915663	0.775510	0.839779	0.800000	[[56857, 7], [22, 76]]
20	decision_tree	PT smote	max_depth=1	0.924012	0.970314	0.048725	0.877551	0.092324	0.199351	[[55185, 1679], [12, 86]]
21	decision_tree	PT smote	max_depth=2	0.923579	0.979618	0.068938	0.867347	0.127724	0.261538	[[55716, 1148], [13, 85]]
22	decision_tree	PT smote	max_depth=3	0.923874	0.919192	0.019415	0.928571	0.038036	0.089585	[[52268, 4596], [7, 91]]
23	decision_tree	PT smote	max_depth=4	0.932328	0.956410	0.034725	0.908163	0.066892	0.150592	[[54390, 2474], [9, 89]]
24	decision_tree	PT smote	max_depth=5	0.932521	0.956796	0.035026	0.908163	0.067450	0.151722	[[54412, 2452], [9, 89]]
25	decision_tree	PT adasyn	max_depth=1	0.912724	0.937608	0.023967	0.887755	0.046674	0.108155	[[53321, 3543], [11, 87]]
26	decision_tree	PT adasyn	max_depth=2	0.832843	0.727274	0.005890	0.938776	0.011706	0.028727	[[41335, 15529], [6, 92]]
27	decision_tree	PT adasyn	max_depth=3	0.906024	0.883554	0.013548	0.928571	0.026706	0.064003	[[50238, 6626], [7, 91]]
28	decision_tree	PT adasyn	max_depth=4	0.905525	0.872389	0.012508	0.938776	0.024688	0.059378	[[49601, 7263], [6, 92]]
29	decision_tree	PT adasyn	max_depth=5	0.894332	0.890717	0.013962	0.897959	0.027496	0.065721	[[50649, 6215], [10, 88]]

```
Results for decision_tree saved to C:\TFM\01_total_models\decision_tree_total.csv
```

In [ ]: