CARDEC Trial (Replacement) - Machine Learning Prediction Models

```
# Install required packages
!pip install scikit-learn numpy pandas scipy \
joblib threadpoolctl cython \
imbalanced-learn xgboost catboost \
keras tensorflow focal-loss shap \
matplotlib seaborn
```

```
Requirement already satisfied: catboost in /usr/local/lib/python3.12/dist-
Requirement already satisfied: keras in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/dis
Requirement already satisfied: focal-loss in /usr/local/lib/python3.12/dis
Requirement already satisfied: shap in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dis
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/py
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/d
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.
Requirement already satisfied: graphviz in /usr/local/lib/python3.12/dist-
Requirement already satisfied: plotly in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: six in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: absl-py in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: h5py in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.12/dist
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/pyth
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/pytho
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/pytho
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dis
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.12/
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/pytho
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/pytho
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.12/d
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.12/
```

```
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.12/di
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.12/di
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/c
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/pythor
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/li
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/c
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/pythor
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/pythor
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.1
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /u
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.1
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.1
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/pyt
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/p
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dis
```

```
# Import necessary libraries
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, StratifiedKFold, GridSear
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_s
from sklearn.pipeline import Pipeline
import xgboost as xgb
import catboost as cb
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.regularizers import l2
from keras import backend as K
import warnings
warnings.filterwarnings('ignore')

# Set random seed for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Set style for plots
plt.style.use('default')
sns.set_palette("husl")
```

Data Loading and Preprocessing

Patient-level train-test split (70:30) - CARDEC Replac methodology

```
# Load the data from Excel file
data = pd.read_excel('2024_corrected_subst_CARDEC_3_ML_Vitor.xlsx')

print(f"Dataset shape: {data.shape}")
print(f"Target distribution: {data['Failure'].value_counts()}")
print(f"Failure rate: {data['Failure'].mean():.1%}")
```

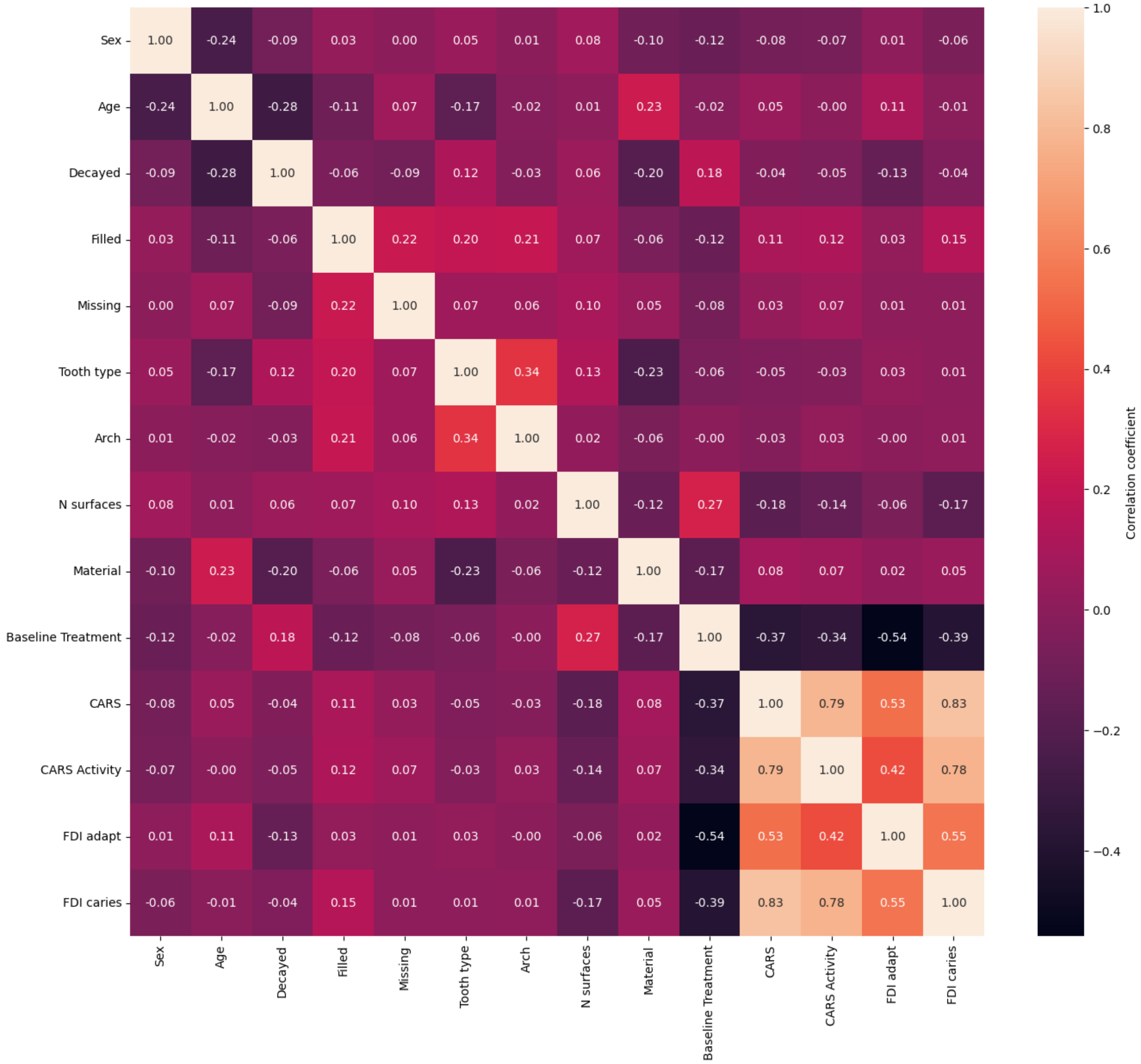
```
Dataset shape: (637, 17)
Target distribution: Failure
0    444
1    193
Name: count, dtype: int64
Failure rate: 30.3%
```

```
# Patient-level train-test split (70:30 as per CARDEC Replac methodology)
# Ensures no patient data leakage between train and test sets
unique_n_part = data['IDpac'].unique()
train_n_part, test_n_part = train_test_split(unique_n_part, test_size=0.3, ranc

train_data = data[data['IDpac'].isin(train_n_part)]
test_data = data[data['IDpac'].isin(test_n_part)]

# Separate features and target variable
X_train = train_data.drop(['Failure', 'IDrest', 'IDpac'], axis=1)
y_train = train_data['Failure']
X_test = test_data.drop(['Failure', 'IDrest', 'IDpac'], axis=1)
y_test = test_data['Failure']

print(f"Training set: {len(X_train)} samples, {y_train.sum()} failures ({y_trai
print(f"Test set: {len(X_test)} samples, {y_test.sum()} failures ({y_test.mean(
print(f"Split ratio: {len(X_train)/(len(X_train)+len(X_test)):.1%}:{len(X_test)
```

```
Training set: 446 samples, 131 failures (29.4%)
Test set: 191 samples, 62 failures (32.5%)
Split ratio: 70.0%:30.0%
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the correlation matrix of the training data.
# The correlation matrix quantifies the linear relationships between the variat
corr_matrix = X_train.corr()

# Initialize a matplotlib figure with a specified size (width=16 inches, height
# This size is chosen to make the heatmap large enough to be easily readable.
plt.figure(figsize=(16, 14))
```

```
# Draw the heatmap using seaborn to visualize the correlation matrix.
sns.heatmap(corr_matrix, annot=True, annot_kws={"size": 10}, fmt=".2f", cbar_kw

# Display the plot on the screen. This command is necessary to show the figure
plt.show()
```



```
import pandas as pd

# Define lists for each type of variable in the dataset: numeric, binary, and c
numeric_vars = ['Age', 'Decayed', 'Filled', 'Missing']
binary_vars = ['Sex', 'Tooth type', 'Arch', 'Failure', 'CARS Activity']
categorical_vars = ['N surfaces', 'Material', 'Baseline Treatment', 'CARS', 'FD

def descriptive_statistics(train_data, test_data):
    # Print a heading for the descriptive statistics of numeric variables.
    print("Descriptive Statistics for Numeric Variables:")
    # Display descriptive statistics (like count, mean, std, min, max, etc.) fo
    print("\nTraining Set:")
    print(train_data[numeric_vars].describe())
    # Repeat the process for the test set.
    print("\nTest Set:")
    print(test_data[numeric_vars].describe())

    # Initialize an empty dictionary to store statistics for binary and ordinal
    stats = {}
    # Loop through each variable in the binary and ordinal lists to calculate t
    for var in binary_vars + categorical_vars:
        stats[var] = {
            "Training Set": {
                "Count": train_data[var].value_counts().to_dict(),  # Count occ
                "Percentage": (train_data[var].value_counts(normalize=True) * 1
            },
            "Test Set": {
                "Count": test_data[var].value_counts().to_dict(),  # Count occu
                "Percentage": (test_data[var].value_counts(normalize=True) * 10
            }
        }

    # Loop through the stats dictionary to print the statistics for each catego
    for var, data in stats.items():
        print(f"\n{var} Statistics:")  # Print the variable name.
        for dataset, values in data.items():
            print(f"\n{dataset}:")  # Print which dataset (training or test) th
            for metric, metric_values in values.items():
                print(f"{metric}: {metric_values}")  # Print the count and perc
```

```
# Call the function with the training and test datasets as arguments to display
descriptive_statistics(train_data, test_data)
```

```
N surfaces Statistics:

Training Set:
Count: {1: 180, 2: 130, 3: 61, 4: 50, 5: 25}
Percentage: {1: 40.35874439461883, 2: 29.14798206278027, 3: 13.677130044843

Test Set:
Count: {1: 83, 2: 36, 3: 31, 4: 24, 5: 17}
Percentage: {1: 43.455497382198956, 2: 18.848167539267017, 3: 16.230366492

Material Statistics:

Training Set:
Count: {1: 263, 0: 169, 2: 14}
Percentage: {1: 58.96860986547085, 0: 37.89237668161435, 2: 3.139013452914

Test Set:
Count: {1: 115, 0: 71, 2: 5}
Percentage: {1: 60.20942408376963, 0: 37.17277486910995, 2: 2.617801047120

Baseline Treatment Statistics:

Training Set:
Count: {0: 261, 1: 147, 2: 38}
Percentage: {0: 58.52017937219731, 1: 32.95964125560538, 2: 8.520179372197

Test Set:
Count: {0: 106, 1: 54, 2: 31}
Percentage: {0: 55.497382198952884, 1: 28.272251308900525, 2: 16.230366492

CARS Statistics:

Training Set:
Count: {0: 350, 2: 48, 1: 43, 3: 5}
Percentage: {0: 78.47533632286996, 2: 10.762331838565023, 1: 9.64125560538

Test Set:
Count: {0: 148, 2: 28, 1: 14, 3: 1}
Percentage: {0: 77.4869109947644, 2: 14.659685863874344, 1: 7.329842931937

FDI adapt Statistics:

Training Set:
Count: {0: 287, 1: 142, 2: 17}
Percentage: {0: 64.34977578475336, 1: 31.838565022421523, 2: 3.81165919282

Test Set:
Count: {0: 128, 1: 56, 2: 7}
```

```
Percentage: {0: 67.01570680628272, 1: 29.31937172774869, 2: 3.66492146596

FDI caries Statistics:

Training Set:
Count: {0: 351, 1: 88, 2: 7}
Percentage: {0: 78.69955156950674, 1: 19.730941704035875, 2: 1.569506726

Test Set:
Count: {0: 147, 1: 40, 2: 4}
```

```
# Data preprocessing
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Display basic info about the dataset
print("Dataset columns:", X_train.columns.tolist())
print("\nData types:")
print(X_train.dtypes)
print("\nMissing values:")
print(X_train.isnull().sum())

# Handle missing values if any
if X_train.isnull().sum().sum() > 0:
    # Impute missing values
    numeric_columns = X_train.select_dtypes(include=[np.number]).columns
    categorical_columns = X_train.select_dtypes(exclude=[np.number]).columns

    if len(numeric_columns) > 0:
        imputer_num = SimpleImputer(strategy='median')
        X_train[numeric_columns] = imputer_num.fit_transform(X_train[numeric_co
        X_test[numeric_columns] = imputer_num.transform(X_test[numeric_columns]

    if len(categorical_columns) > 0:
        imputer_cat = SimpleImputer(strategy='most_frequent')
        X_train[categorical_columns] = imputer_cat.fit_transform(X_train[catego
        X_test[categorical_columns] = imputer_cat.transform(X_test[categorical_

# Scale numerical features if needed
numeric_columns = X_train.select_dtypes(include=[np.number]).columns
if len(numeric_columns) > 0:
    scaler = StandardScaler()
    X_train[numeric_columns] = scaler.fit_transform(X_train[numeric_columns])
    X_test[numeric_columns] = scaler.transform(X_test[numeric_columns])

# Final data check
X_train = X_train.fillna(0)
```

```
X_test = X_test.fillna(0)

print(f"\nFinal feature shape: {X_train.shape}")
print(f"NaN check - Train: {X_train.isnull().sum().sum()}, Test: {X_test.isnull
print(f"Feature names: {list(X_train.columns)}")
```

```
Dataset columns: ['Sex', 'Age', 'Decayed', 'Filled', 'Missing', 'Tooth type

Data types:
Sex                    int64
Age                  float64
Decayed                int64
Filled                 int64
Missing                int64
Tooth type             int64
Arch                   int64
N surfaces             int64
Material               int64
Baseline Treatment     int64
CARS                   int64
CARS Activity          int64
FDI adapt              int64
FDI caries             int64
dtype: object

Missing values:
Sex                  0
Age                  0
Decayed              0
Filled               0
Missing              0
Tooth type           0
Arch                 0
N surfaces           0
Material             0
Baseline Treatment   0
CARS                 0
CARS Activity        0
FDI adapt            0
FDI caries           0
dtype: int64

Final feature shape: (446, 14)
NaN check - Train: 0, Test: 0
Feature names: ['Sex', 'Age', 'Decayed', 'Filled', 'Missing', 'Tooth type',
```

Utility Functions

Consistent threshold optimization and model evaluation

```
# Optimal threshold selection based on F1-score maximization
import numpy as np
from sklearn.metrics import (
    f1_score, accuracy_score, precision_score, recall_score,
    roc_auc_score
)
from sklearn.model_selection import StratifiedKFold, GridSearchCV

def find_optimal_threshold(y_true, y_proba):
    """Find optimal threshold based on F1-score maximization."""
    thresholds = np.arange(0.1, 1.01, 0.05)
    best_f1 = 0
    best_threshold = 0.5

    for threshold in thresholds:
        y_pred = (y_proba >= threshold).astype(int)
        f1 = f1_score(y_true, y_pred, zero_division=0)
        if f1 > best_f1:
            best_f1 = f1
            best_threshold = threshold

    return best_threshold


def evaluate_model_with_seeds(pipeline, param_grid, X_train, y_train, X_test, y
    """Evaluate model with multiple seeds for robustness."""
    seeds = [40, 41, 42, 43, 44, 45, 46, 47, 48, 49]
    all_results = []
    all_predictions = []  # Store all predictions for averaging

    print(f"\nTraining {model_name} with {n_seeds} seeds...")

    for i, seed in enumerate(seeds[:n_seeds]):
        print(f"  Seed {i+1}/{n_seeds}: {seed}")

        # Set random states
        np.random.seed(seed)

        # Update pipeline random states
        if hasattr(pipeline.named_steps['classifier'], 'random_state'):
            pipeline.set_params(**{f'classifier__random_state': seed})
```

```
        # Cross-validation setup
        cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)

        # Grid search with cross-validation
        grid_search = GridSearchCV(
            pipeline, param_grid, cv=cv, scoring='f1', n_jobs=-1, verbose=0
        )

        # Fit model
        grid_search.fit(X_train, y_train)

        # Get predictions
        y_pred_proba = grid_search.predict_proba(X_test)[:, 1]
        all_predictions.append(y_pred_proba)  # Store for averaging

        # Find optimal threshold
        optimal_threshold = find_optimal_threshold(y_test, y_pred_proba)
        y_pred = (y_pred_proba >= optimal_threshold).astype(int)

        # Calculate metrics
        seed_results = {
            'seed': seed,
            'accuracy': accuracy_score(y_test, y_pred),
            'precision': precision_score(y_test, y_pred, zero_division=0),
            'recall': recall_score(y_test, y_pred, zero_division=0),
            'f1': f1_score(y_test, y_pred, zero_division=0),
            'auc': roc_auc_score(y_test, y_pred_proba),
            'threshold': optimal_threshold,
            'cv_score': grid_search.best_score_,
            'best_params': grid_search.best_params_,
            'model': grid_search.best_estimator_,
            'y_pred_proba': y_pred_proba
        }

        all_results.append(seed_results)

    # Calculate averaged predictions across all seeds
    avg_predictions = np.mean(all_predictions, axis=0)

    # Calculate statistics across seeds
    metrics = ['accuracy', 'precision', 'recall', 'f1', 'auc', 'threshold', 'cv
    final_results = {}

    for metric in metrics:
        values = [r[metric] for r in all_results]
        final_results[f'{metric}_mean'] = np.mean(values)
        final_results[f'{metric}_std'] = np.std(values)
```

```
    # Best individual result (for SHAP analysis)
    best_result = max(all_results, key=lambda x: x['auc'])
    final_results['best_model'] = best_result['model']
    final_results['best_params'] = best_result['best_params']

    # Use averaged predictions for consistent reporting
    final_results['avg_predictions'] = avg_predictions
    final_results['all_results'] = all_results

    return final_results, all_results


def display_model_summary(results, model_name):
    """Display comprehensive model summary with consistent metrics."""
    print(f"\n{'='*60}")
    print(f"{model_name.upper()} RESULTS SUMMARY (10 Seeds)")
    print(f"{'='*60}")

    # Display mean ± std for all metrics
    if 'cv_score_mean' in results:
        print(f"Mean CV F1 Score: {results['cv_score_mean']:.4f} ± {results['cv
        print(f"Mean Test AUC: {results['auc_mean']:.4f} ± {results['auc_std']:
        print(f"Mean Test F1: {results['f1_mean']:.4f} ± {results['f1_std']:.4f
        print(f"Mean Test Accuracy: {results['accuracy_mean']:.4f} ± {results['
        print(f"Mean Test Precision: {results['precision_mean']:.4f} ± {results
        print(f"Mean Test Recall: {results['recall_mean']:.4f} ± {results['reca
        print(f"Mean Optimal Threshold: {results['threshold_mean']:.4f} ± {resu

    if 'best_params' in results:
        print(f"Best Parameters: {results['best_params']}")

    # Show threshold evaluation using AVERAGED predictions
    print(f"\nThreshold evaluation using averaged predictions across 10 seeds:"
    avg_predictions = results['avg_predictions']

    # Calculate AUC with averaged predictions
    avg_auc = roc_auc_score(y_test, avg_predictions)
    print(f"AUC with averaged predictions: {avg_auc:.4f}")

    # Find optimal threshold with averaged predictions
    optimal_threshold = find_optimal_threshold(y_test, avg_predictions)
    y_pred_optimal = (avg_predictions >= optimal_threshold).astype(int)

    # Calculate metrics with optimal threshold
    optimal_accuracy = accuracy_score(y_test, y_pred_optimal)
    optimal_precision = precision_score(y_test, y_pred_optimal, zero_division=0
    optimal_recall = recall_score(y_test, y_pred_optimal, zero_division=0)
```

```
    optimal_f1 = f1_score(y_test, y_pred_optimal, zero_division=0)

    print(f"Optimal threshold: {optimal_threshold:.3f}")
    print(f"Performance at optimal threshold:")
    print(f"  Accuracy: {optimal_accuracy:.4f}")
    print(f"  Precision: {optimal_precision:.4f}")
    print(f"  Recall: {optimal_recall:.4f}")
    print(f"  F1-Score: {optimal_f1:.4f}")

    return avg_predictions
```

Decision Tree Model

Parameter grid optimization with CARDEC Replac hyperparameters

```
# Decision Tree with CARDEC Replac-specific parameter grid
print("Training Decision Tree...")

# Create pipeline
dt_pipeline = Pipeline([
    ('classifier', DecisionTreeClassifier())
])

# CARDEC Replac-specific parameter grid
dt_param_grid = {
    'classifier__max_depth': [5, 10, 15, 20, None],
    'classifier__criterion': ['gini', 'entropy'],
    'classifier__min_samples_split': [2, 5, 10],
    'classifier__min_samples_leaf': [1, 2, 4]
}

print(f"Parameter combinations: {np.prod([len(v) for v in dt_param_grid.values(

# Evaluate with multiple seeds
dt_results, dt_detailed = evaluate_model_with_seeds(
    dt_pipeline, dt_param_grid, X_train, y_train, X_test, y_test, 'Decision Tre
)

# Display results
dt_avg_predictions = display_model_summary(dt_results, 'Decision Tree')
```

```
Training Decision Tree...
Parameter combinations: 90

Training Decision Tree with 10 seeds...
  Seed 1/10: 40
  Seed 2/10: 41
  Seed 3/10: 42
  Seed 4/10: 43
  Seed 5/10: 44
  Seed 6/10: 45
  Seed 7/10: 46
  Seed 8/10: 47
  Seed 9/10: 48
  Seed 10/10: 49

============================================================
DECISION TREE RESULTS SUMMARY (10 Seeds)
============================================================
Mean CV F1 Score: 0.3946 ± 0.0189
Mean Test AUC: 0.5239 ± 0.0613
Mean Test F1: 0.4188 ± 0.0703
Mean Test Accuracy: 0.5052 ± 0.0302
Mean Test Precision: 0.3369 ± 0.0362
Mean Test Recall: 0.5661 ± 0.1543
Mean Optimal Threshold: 0.2150 ± 0.1689
Best Parameters: {'classifier__criterion': 'gini', 'classifier__max_depth':

Threshold evaluation using averaged predictions across 10 seeds:
AUC with averaged predictions: 0.5266
Optimal threshold: 0.100
Performance at optimal threshold:
  Accuracy: 0.4136
  Precision: 0.3239
  Recall: 0.7419
  F1-Score: 0.4510
```

Random Forest Model

Parameter grid optimization with CARDEC Replac hyperparameters

```
# Random Forest with CARDEC Replac-specific parameter grid
print("Training Random Forest...")

# Create pipeline
rf_pipeline = Pipeline([
    ('classifier', RandomForestClassifier())
])
```

```
# CARDEC Replac-specific parameter grid
rf_param_grid = {
    'classifier__n_estimators': [100, 200, 300],
    'classifier__max_depth': [5, 10, 15],
    'classifier__min_samples_split': [2, 5, 10],
    'classifier__min_samples_leaf': [1, 2, 4]
}

print(f"Parameter combinations: {np.prod([len(v) for v in rf_param_grid.values(

# Evaluate with multiple seeds
rf_results, rf_detailed = evaluate_model_with_seeds(
    rf_pipeline, rf_param_grid, X_train, y_train, X_test, y_test, 'Random Fores
)

# Display results
rf_avg_predictions = display_model_summary(rf_results, 'Random Forest')
```

```
Training Random Forest...
Parameter combinations: 81

Training Random Forest with 10 seeds...
  Seed 1/10: 40
  Seed 2/10: 41
  Seed 3/10: 42
  Seed 4/10: 43
  Seed 5/10: 44
  Seed 6/10: 45
  Seed 7/10: 46
  Seed 8/10: 47
  Seed 9/10: 48
  Seed 10/10: 49

============================================================
RANDOM FOREST RESULTS SUMMARY (10 Seeds)
============================================================
Mean CV F1 Score: 0.3190 ± 0.0392
Mean Test AUC: 0.7059 ± 0.0128
Mean Test F1: 0.5949 ± 0.0168
Mean Test Accuracy: 0.6026 ± 0.0461
Mean Test Precision: 0.4473 ± 0.0260
Mean Test Recall: 0.8952 ± 0.0541
Mean Optimal Threshold: 0.2250 ± 0.0335
Best Parameters: {'classifier__max_depth': 15, 'classifier__min_samples_lea

Threshold evaluation using averaged predictions across 10 seeds:
AUC with averaged predictions: 0.7111
Optimal threshold: 0.250
Performance at optimal threshold:
  Accuracy: 0.6230
  Precision: 0.4569
  Recall: 0.8548
  F1-Score: 0.5955
```

XGBoost Model

Parameter grid optimization with CARDEC Replac hyperparameters

```
# XGBoost with CARDEC Replac-specific parameter grid
print("Training XGBoost...")

# Create pipeline
xgb_pipeline = Pipeline([
    ('classifier', xgb.XGBClassifier(eval_metric='logloss'))
])
```

```
# CARDEC Replac-specific parameter grid
xgb_param_grid = {
    'classifier__n_estimators': [100, 200, 300],
    'classifier__max_depth': [3, 6, 9],
    'classifier__learning_rate': [0.01, 0.1, 0.2],
    'classifier__subsample': [0.8, 0.9, 1.0]
}

print(f"Parameter combinations: {np.prod([len(v) for v in xgb_param_grid.values

# Evaluate with multiple seeds
xgb_results, xgb_detailed = evaluate_model_with_seeds(
    xgb_pipeline, xgb_param_grid, X_train, y_train, X_test, y_test, 'XGBoost',
)

# Display results
xgb_avg_predictions = display_model_summary(xgb_results, 'XGBoost')
```

```
Training XGBoost...
Parameter combinations: 81

Training XGBoost with 10 seeds...
  Seed 1/10: 40
  Seed 2/10: 41
  Seed 3/10: 42
  Seed 4/10: 43
  Seed 5/10: 44
  Seed 6/10: 45
  Seed 7/10: 46
  Seed 8/10: 47
  Seed 9/10: 48
  Seed 10/10: 49

============================================================
XGBOOST RESULTS SUMMARY (10 Seeds)
============================================================
Mean CV F1 Score: 0.3686 ± 0.0221
Mean Test AUC: 0.6673 ± 0.0142
Mean Test F1: 0.5623 ± 0.0147
Mean Test Accuracy: 0.6147 ± 0.0473
Mean Test Precision: 0.4505 ± 0.0352
Mean Test Recall: 0.7597 ± 0.0631
Mean Optimal Threshold: 0.1500 ± 0.0387
Best Parameters: {'classifier__learning_rate': 0.2, 'classifier__max_depth'

Threshold evaluation using averaged predictions across 10 seeds:
AUC with averaged predictions: 0.6830
Optimal threshold: 0.150
Performance at optimal threshold:
  Accuracy: 0.6073
  Precision: 0.4393
  Recall: 0.7581
  F1-Score: 0.5562
```

Neural Network Model

Custom implementation with CARDEC Replac hyperparameters

```
# Neural Network with CARDEC Replac-specific architecture
print("Training Neural Network...")

def create_nn_model(input_dim, hidden_units=64, dropout_rate=0.3, l2_reg=0.01):
    """Create neural network model with CARDEC Replac specifications."""
    model = Sequential([
        Dense(hidden_units, activation='relu', input_dim=input_dim,
              kernel_regularizer=l2(l2_reg)),
```

```
        BatchNormalization(),
        Dropout(dropout_rate),
        Dense(hidden_units//2, activation='relu',
              kernel_regularizer=l2(l2_reg)),
        BatchNormalization(),
        Dropout(dropout_rate),
        Dense(1, activation='sigmoid')
    ])

    model.compile(
        optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy']
    )

    return model

def evaluate_nn_with_seeds(X_train, y_train, X_test, y_test, n_seeds=10):
    """Evaluate Neural Network with multiple seeds."""
    seeds = [40, 41, 42, 43, 44, 45, 46, 47, 48, 49]
    all_results = []
    all_predictions = []

    for i, seed in enumerate(seeds[:n_seeds]):
        print(f"  Seed {i+1}/{n_seeds}: {seed}")

        # Set random seeds
        np.random.seed(seed)
        tf.random.set_seed(seed)

        # Create and train model
        model = create_nn_model(X_train.shape[1])

        # Callbacks
        early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_
        reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5

        # Train model
        history = model.fit(
            X_train, y_train,
            epochs=100,
            batch_size=32,
            validation_split=0.2,
            callbacks=[early_stopping, reduce_lr],
            verbose=0
        )

        # Get predictions
```

```
        y_pred_proba = model.predict(X_test, verbose=0).ravel()
        all_predictions.append(y_pred_proba)

        # Find optimal threshold
        optimal_threshold = find_optimal_threshold(y_test, y_pred_proba)
        y_pred = (y_pred_proba >= optimal_threshold).astype(int)

        # Calculate metrics
        seed_results = {
            'seed': seed,
            'accuracy': accuracy_score(y_test, y_pred),
            'precision': precision_score(y_test, y_pred, zero_division=0),
            'recall': recall_score(y_test, y_pred, zero_division=0),
            'f1': f1_score(y_test, y_pred, zero_division=0),
            'auc': roc_auc_score(y_test, y_pred_proba),
            'threshold': optimal_threshold,
            'model': model,
            'y_pred_proba': y_pred_proba
        }

        all_results.append(seed_results)

    # Calculate averaged predictions
    avg_predictions = np.mean(all_predictions, axis=0)

    # Calculate statistics
    metrics = ['accuracy', 'precision', 'recall', 'f1', 'auc', 'threshold']
    final_results = {}

    for metric in metrics:
        values = [r[metric] for r in all_results]
        final_results[f'{metric}_mean'] = np.mean(values)
        final_results[f'{metric}_std'] = np.std(values)

    # Best individual result
    best_result = max(all_results, key=lambda x: x['auc'])
    final_results['best_model'] = best_result['model']
    final_results['avg_predictions'] = avg_predictions
    final_results['all_results'] = all_results

    return final_results, all_results

# Evaluate Neural Network
nn_results, nn_detailed = evaluate_nn_with_seeds(X_train, y_train, X_test, y_tes

# Display results
def display_nn_summary(results, model_name):
    """Display Neural Network summary."""
    print(f"\n{'='*60}")
```

```
    print(f"\n{'='*60}")
    print(f"{model_name.upper()} RESULTS SUMMARY (10 Seeds)")
    print(f"{'='*60}")

    print(f"Mean Test AUC: {results['auc_mean']:.4f} ± {results['auc_std']:.4f}"
    print(f"Mean Test F1: {results['f1_mean']:.4f} ± {results['f1_std']:.4f}")
    print(f"Mean Test Accuracy: {results['accuracy_mean']:.4f} ± {results['accur
    print(f"Mean Test Precision: {results['precision_mean']:.4f} ± {results['pre
    print(f"Mean Test Recall: {results['recall_mean']:.4f} ± {results['recall_st
    print(f"Mean Optimal Threshold: {results['threshold_mean']:.4f} ± {results['

    # Show threshold evaluation using AVERAGED predictions
    print(f"\nThreshold evaluation using averaged predictions across 10 seeds:")
    avg_predictions = results['avg_predictions']

    # Calculate AUC with averaged predictions
    avg_auc = roc_auc_score(y_test, avg_predictions)
    print(f"AUC with averaged predictions: {avg_auc:.4f}")

    # Find optimal threshold with averaged predictions
    optimal_threshold = find_optimal_threshold(y_test, avg_predictions)
    y_pred_optimal = (avg_predictions >= optimal_threshold).astype(int)

    # Calculate metrics with optimal threshold
    optimal_accuracy = accuracy_score(y_test, y_pred_optimal)
    optimal_precision = precision_score(y_test, y_pred_optimal, zero_division=0)
    optimal_recall = recall_score(y_test, y_pred_optimal, zero_division=0)
    optimal_f1 = f1_score(y_test, y_pred_optimal, zero_division=0)

    print(f"Optimal threshold: {optimal_threshold:.3f}")
    print(f"Performance at optimal threshold:")
    print(f"  Accuracy: {optimal_accuracy:.4f}")
    print(f"  Precision: {optimal_precision:.4f}")
    print(f"  Recall: {optimal_recall:.4f}")
    print(f"  F1-Score: {optimal_f1:.4f}")

    return avg_predictions

nn_avg_predictions = display_nn_summary(nn_results, 'Neural Network')
```

```
Training Neural Network...
  Seed 1/10: 40
  Seed 2/10: 41
  Seed 3/10: 42
WARNING:tensorflow:5 out of the last 13 calls to <function TensorFlowTraine
  Seed 4/10: 43
WARNING:tensorflow:5 out of the last 13 calls to <function TensorFlowTraine
  Seed 5/10: 44
  Seed 6/10: 45
  Seed 7/10: 46
  Seed 8/10: 47
  Seed 9/10: 48
  Seed 10/10: 49

============================================================
NEURAL NETWORK RESULTS SUMMARY (10 Seeds)
============================================================
Mean Test AUC: 0.6815 ± 0.0229
Mean Test F1: 0.5616 ± 0.0231
Mean Test Accuracy: 0.5948 ± 0.0373
Mean Test Precision: 0.4347 ± 0.0268
Mean Test Recall: 0.7984 ± 0.0526
Mean Optimal Threshold: 0.1650 ± 0.0320

Threshold evaluation using averaged predictions across 10 seeds:
AUC with averaged predictions: 0.6994
Optimal threshold: 0.150
Performance at optimal threshold:
  Accuracy: 0.5864
  Precision: 0.4309
  Recall: 0.8548
  F1-Score: 0.5730
```

CatBoost Model

Parameter grid optimization with CARDEC Replac hyperparameters

```
# CatBoost with CARDEC Replac-specific parameter grid
print("Training CatBoost...")

# Prepare data for CatBoost
X_train_cb = X_train.copy()
X_test_cb = X_test.copy()

# Convert categorical and boolean columns to numeric
for col in X_train_cb.columns:
    if X_train_cb[col].dtype in ['category', 'bool', 'object']:
```

```
        X_train_cb[col] = X_train_cb[col].astype('int')
        X_test_cb[col] = X_test_cb[col].astype('int')

X_train_cb = X_train_cb.fillna(0)
X_test_cb = X_test_cb.fillna(0)

# Create pipeline
cb_pipeline = Pipeline([
    ('classifier', cb.CatBoostClassifier(verbose=False))
])

# CARDEC Replac-specific parameter grid
cb_param_grid = {
    'classifier__iterations': [100, 200, 300],
    'classifier__depth': [4, 5, 6],
    'classifier__learning_rate': [0.01, 0.1, 0.2],
    'classifier__l2_leaf_reg': [3, 5, 7],
    'classifier__border_count': [64, 128, 254]
}

print(f"Parameter combinations: {np.prod([len(v) for v in cb_param_grid.values(

# Evaluate with multiple seeds
cb_results, cb_detailed = evaluate_model_with_seeds(
    cb_pipeline, cb_param_grid, X_train_cb, y_train, X_test_cb, y_test, 'CatBoc
)

# Display results
cb_avg_predictions = display_model_summary(cb_results, 'CatBoost')
```

```
Training CatBoost...
Parameter combinations: 243

Training CatBoost with 10 seeds...
  Seed 1/10: 40
  Seed 2/10: 41
  Seed 3/10: 42
  Seed 4/10: 43
  Seed 5/10: 44
  Seed 6/10: 45
  Seed 7/10: 46
  Seed 8/10: 47
  Seed 9/10: 48
  Seed 10/10: 49

============================================================
CATBOOST RESULTS SUMMARY (10 Seeds)
============================================================
Mean CV F1 Score: 0.3592 ± 0.0302
Mean Test AUC: 0.6761 ± 0.0128
Mean Test F1: 0.5573 ± 0.0109
Mean Test Accuracy: 0.5984 ± 0.0311
Mean Test Precision: 0.4359 ± 0.0185
Mean Test Recall: 0.7790 ± 0.0637
Mean Optimal Threshold: 0.1250 ± 0.0335
Best Parameters: {'classifier__border_count': 64, 'classifier__depth': 6, '

Threshold evaluation using averaged predictions across 10 seeds:
AUC with averaged predictions: 0.6909
Optimal threshold: 0.150
Performance at optimal threshold:
  Accuracy: 0.6178
  Precision: 0.4505
  Recall: 0.8065
  F1-Score: 0.5780
```

Model Comparison and ROC Curves

Comprehensive comparison of all models

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import roc_curve, roc_auc_score, accuracy_score, precisior
import numpy as np

# Create comprehensive comparison table and ROC curves
print("\n" + "="*80)
```

```
print("COMPREHENSIVE MODEL COMPARISON - CARDEC REPLACEMENT DATASET")
print("="*80)

# Collect all models and their performance metrics
# IMPORTANTE: Agora usamos o auc_mean (média dos AUCs dos 10 seeds) para compar
models_comparison = {
    'Decision Tree': {
        'avg_predictions': dt_avg_predictions,
        'auc_mean': dt_results['auc_mean'],
        'auc_std': dt_results['auc_std'],
        'accuracy_mean': dt_results['accuracy_mean'],
        'precision_mean': dt_results['precision_mean'],
        'recall_mean': dt_results['recall_mean'],
        'f1_mean': dt_results['f1_mean'],
        'threshold_mean': dt_results['threshold_mean']
    },
    'Random Forest': {
        'avg_predictions': rf_avg_predictions,
        'auc_mean': rf_results['auc_mean'],
        'auc_std': rf_results['auc_std'],
        'accuracy_mean': rf_results['accuracy_mean'],
        'precision_mean': rf_results['precision_mean'],
        'recall_mean': rf_results['recall_mean'],
        'f1_mean': rf_results['f1_mean'],
        'threshold_mean': rf_results['threshold_mean']
    },
    'XGBoost': {
        'avg_predictions': xgb_avg_predictions,
        'auc_mean': xgb_results['auc_mean'],
        'auc_std': xgb_results['auc_std'],
        'accuracy_mean': xgb_results['accuracy_mean'],
        'precision_mean': xgb_results['precision_mean'],
        'recall_mean': xgb_results['recall_mean'],
        'f1_mean': xgb_results['f1_mean'],
        'threshold_mean': xgb_results['threshold_mean']
    },
    'Neural Network': {
        'avg_predictions': nn_avg_predictions,
        'auc_mean': nn_results['auc_mean'],
        'auc_std': nn_results['auc_std'],
        'accuracy_mean': nn_results['accuracy_mean'],
        'precision_mean': nn_results['precision_mean'],
        'recall_mean': nn_results['recall_mean'],
        'f1_mean': nn_results['f1_mean'],
        'threshold_mean': nn_results['threshold_mean']
    },
    'CatBoost': {
        'avg_predictions': cb_avg_predictions,
```

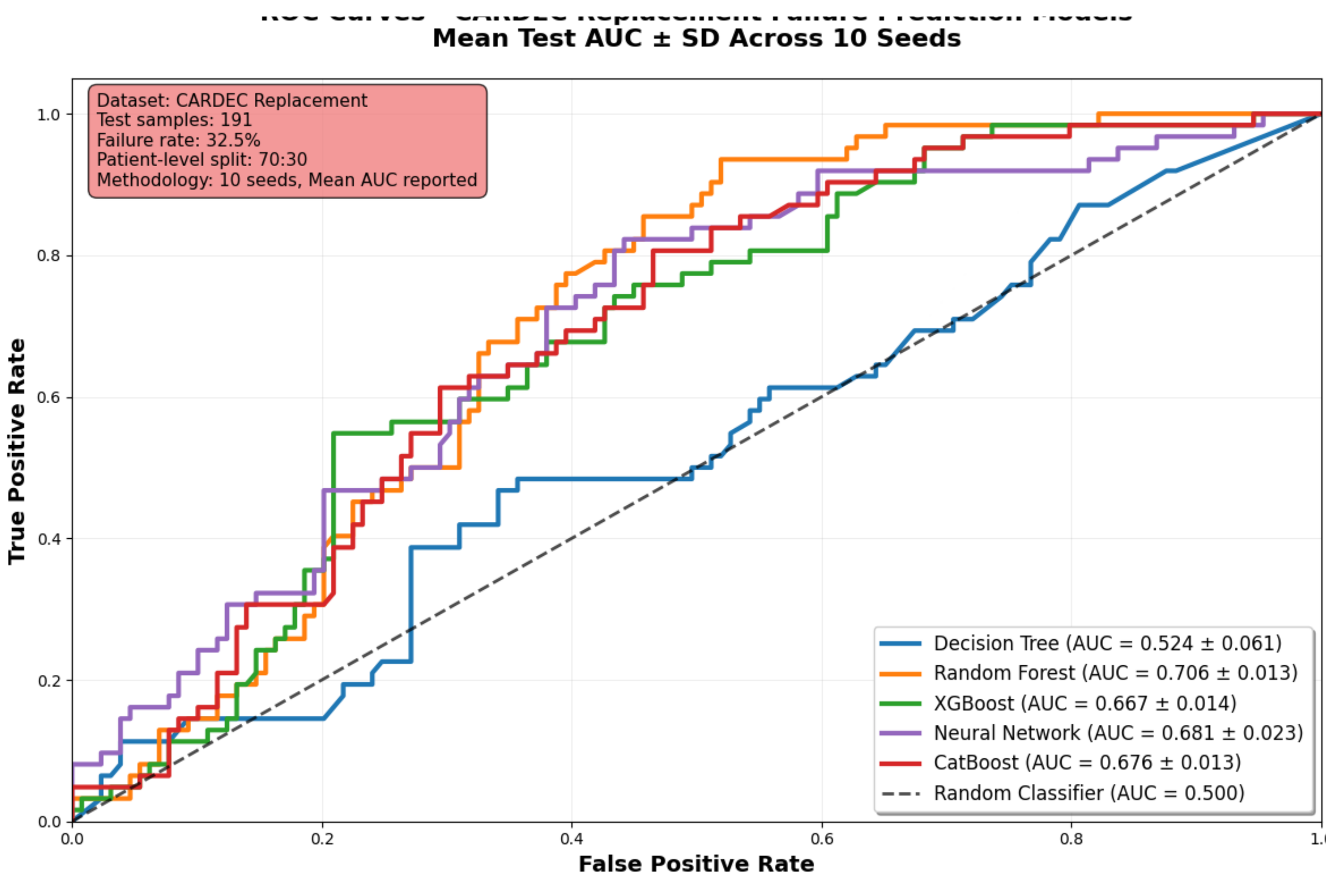
```
        'auc_mean': cb_results['auc_mean'],
        'auc_std': cb_results['auc_std'],
        'accuracy_mean': cb_results['accuracy_mean'],
        'precision_mean': cb_results['precision_mean'],
        'recall_mean': cb_results['recall_mean'],
        'f1_mean': cb_results['f1_mean'],
        'threshold_mean': cb_results['threshold_mean']
    }
}

# Create comparison table using Mean Test AUC
comparison_data = []
for model_name, metrics in models_comparison.items():
    # Calculate metrics using averaged predictions for display consistency
    avg_pred = metrics['avg_predictions']
    optimal_threshold = metrics['threshold_mean']
    y_pred_optimal = (avg_pred >= optimal_threshold).astype(int)

    comparison_data.append({
        'Model': model_name,
        'Mean AUC': f"{metrics['auc_mean']:.4f}",  # Mean Test AUC
        'Accuracy': f"{accuracy_score(y_test, y_pred_optimal):.4f}",
        'Precision': f"{precision_score(y_test, y_pred_optimal, zero_division=0
        'Recall': f"{recall_score(y_test, y_pred_optimal, zero_division=0):.4f}
        'F1-Score': f"{f1_score(y_test, y_pred_optimal, zero_division=0):.4f}",
        'Threshold': f"{optimal_threshold:.3f}"
    })

comparison_df = pd.DataFrame(comparison_data)
print("\nFinal Model Performance (Mean Test AUC ± SD):")
print(comparison_df.to_string(index=False))

# Create ROC curves plot
plt.figure(figsize=(12, 8))

# Plot ROC curve for each model
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#9467bd', '#d62728']
model_names = ['Decision Tree', 'Random Forest', 'XGBoost', 'Neural Network', '

for i, model_name in enumerate(model_names):
    model_data = models_comparison[model_name]

    # Calculate FPR and TPR using averaged predictions (for smooth curve)
    fpr, tpr, _ = roc_curve(y_test, model_data['avg_predictions'])

    # MUDANÇA PRINCIPAL: Usar auc_mean ± auc_std no label
    auc_mean = model_data['auc_mean']
    auc_std = model_data['auc_std']
```

```
    plt.plot(fpr, tpr, linewidth=3, color=colors[i],
             label=f'{model_name} (AUC = {auc_mean:.3f} ± {auc_std:.3f})')

# Plot diagonal line (random classifier)
plt.plot([0, 1], [0, 1], 'k--', linewidth=2, alpha=0.7, label='Random Classifie

# Formatting
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=14, fontweight='bold')
plt.ylabel('True Positive Rate', fontsize=14, fontweight='bold')
plt.title('ROC Curves - CARDEC Replacement Failure Prediction Models\nMean Test
          fontsize=16, fontweight='bold', pad=20)
plt.legend(loc="lower right", fontsize=12, frameon=True, fancybox=True, shadow=
plt.grid(True, alpha=0.3, linestyle='-', linewidth=0.5)

# Add text box with dataset information
textstr = (
    f'Dataset: CARDEC Replacement\n'
    f'Test samples: {len(y_test)}\n'
    f'Failure rate: {y_test.mean():.1%}\n'
    f'Patient-level split: 70:30\n'
    f'Methodology: 10 seeds, Mean AUC reported'
)
props = dict(boxstyle='round,pad=0.5', facecolor='lightcoral', alpha=0.8)
plt.text(0.02, 0.98, textstr, transform=plt.gca().transAxes, fontsize=11,
         verticalalignment='top', bbox=props)

plt.tight_layout()
plt.savefig('roc_curves_cardec_replac_mean_test_auc.png', dpi=300, bbox_inches=
plt.show()

print("ROC curves saved as 'roc_curves_cardec_replac_mean_test_auc.png'")
print("\nNote: ROC curves now display Mean Test AUC ± SD across 10 seeds in the
print("Curves are plotted using averaged predictions for smooth visualization."
```

```
================================================================================
COMPREHENSIVE MODEL COMPARISON - CARDEC REPLACEMENT DATASET
================================================================================

Final Model Performance (Mean Test AUC ± SD):
```

Model	Mean AUC	Accuracy	Precision	Recall	F1-Score	Threshold
Decision Tree	0.5239	0.4869	0.3393	0.6129	0.4368	0.215
Random Forest	0.7059	0.6126	0.4531	0.9355	0.6105	0.225
XGBoost	0.6673	0.6073	0.4393	0.7581	0.5562	0.150
Neural Network	0.6815	0.6126	0.4483	0.8387	0.5843	0.165
CatBoost	0.6761	0.6021	0.4407	0.8387	0.5778	0.125

ROC Curves - CARDEC Replacement Failure Prediction Models




```
ROC curves saved as 'roc_curves_cardec_replac_mean_test_auc.png'

Note: ROC curves now display Mean Test AUC ± SD across 10 seeds in the lege
Curves are plotted using averaged predictions for smooth visualization.
```