Fraud Detection in Electricity and Gas Consumption - STEG

Description of the Business Problem

- The Tunisian Company of Electricity and Gas (STEG) is a public and a non-administrative company, it is responsible for delivering electricity and gas across Tunisia.
- The company suffered tremendous losses in the order of 200 million Tunisian Dinars due to fraudulent manipulations of meters by consumers.
- Using the client’s billing history, we aim to build a model that can detect and recognize clients involved in fraudulent activities.
- The data solution is geared towards enhancing the company’s revenues and reduce the losses caused by such fraudulent activities.

Pre-Requisites

Set up file paths and directories

In [1]:
```
import os
import zipfile

import os.path
from os import path


DATA_DIR = r'C:\Users\HP\OneDrive\Desktop\Moringa bootcamp\Zindi\Fraud-detection-
TRAIN_DIR = os.path.join(DATA_DIR, 'train')
TEST_DIR = os.path.join(DATA_DIR, 'test')
OUTPUT_DIR = os.path.join(DATA_DIR, 'output')

# Ensure directories exist
for pth in [TRAIN_DIR, TEST_DIR, OUTPUT_DIR]:
    if not os.path.exists(pth):
        os.mkdir(pth)

# Define ZIP file paths
train_zip_path = os.path.join(TRAIN_DIR, "train.zip")
test_zip_path = os.path.join(TEST_DIR, "test.zip")

# Define sample submission path
sample_sub_path = os.path.join(OUTPUT_DIR,"SampleSubmission.csv")

# Unzip train.zip
if os.path.exists(train_zip_path):
    with zipfile.ZipFile(train_zip_path, 'r') as zip_ref:
        zip_ref.extractall(TRAIN_DIR)
else:
    print("train.zip not found at", train_zip_path)

# Unzip test.zip
if os.path.exists(test_zip_path):
    with zipfile.ZipFile(test_zip_path, 'r') as zip_ref:
        zip_ref.extractall(TEST_DIR)
else:
    print("test.zip not found at", test_zip_path)
```

Import Libraries

In [2]:
```
## Import the necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split,cross_val_score, StratifiedK
from sklearn.metrics import auc, roc_auc_score,precision_recall_curve,roc_curve,
from xgboost import XGBClassifier

from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE # for handling class imbalance
from imblearn.combine import SMOTEENN # for handling class imbalance

# Suppress warnings
import warnings
import joblib

warnings.simplefilter('ignore')

# Reduce joblib verbosity
joblib.parallel_backend('loky', n_jobs=1);  # Avoid triggering subprocess CPU pro
```

In [3]:
```
# Read the datasets
client_train = pd.read_csv(f'{TRAIN_DIR}/client_train.csv', low_memory=False)
invoice_train = pd.read_csv(f'{TRAIN_DIR}/invoice_train.csv', low_memory=False)

client_test = pd.read_csv(f'{TEST_DIR}/client_test.csv', low_memory=False)
invoice_test = pd.read_csv(f'{TEST_DIR}/invoice_test.csv', low_memory=False)
sample_submission = pd.read_csv(f'{DATA_DIR}/SampleSubmission.csv', low_memory=Fa
```

Data Preparation

1) Exploratory Data Analysis (EDA)

2) Feature Engineering

Exploratory Data Analysis

In [4]:
```
# compare size of the various datasets
print(invoice_train.shape, invoice_test.shape, client_train.shape, client_test.sh
```

```
(4476749, 16) (1939730, 16) (135493, 6) (58069, 5)
```

In [5]:
```
# print top rows of dataset (invoice_train)
invoice_train.head()
```

Out[5]:

	client_id	invoice_date	tarif_type	counter_number	counter_statue	counter_code	reading_re
0	train_Client_0	2014-03-24	11	1335667	0	203	
1	train_Client_0	2013-03-29	11	1335667	0	203	
2	train_Client_0	2015-03-23	11	1335667	0	203	
3	train_Client_0	2015-07-13	11	1335667	0	207	
4	train_Client_0	2016-11-17	11	1335667	0	207	

In [6]:
```
# print top rows of dataset (client_train)
client_train.head()
```

Out[6]:

	disrict	client_id	client_catg	region	creation_date	target
0	60	train_Client_0	11	101	31/12/1994	0.0
1	69	train_Client_1	11	107	29/05/2002	0.0
2	62	train_Client_10	11	301	13/03/1986	0.0
3	69	train_Client_100	11	105	11/07/1996	0.0
4	62	train_Client_1000	11	303	14/10/2014	0.0

In [7]:
```
# Get concise information of each column in dataset
invoice_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4476749 entries, 0 to 4476748
Data columns (total 16 columns):
 #   Column                Dtype
---  ------                -----
 0   client_id             object
 1   invoice_date          object
 2   tarif_type            int64
 3   counter_number        int64
 4   counter_statue        object
 5   counter_code          int64
 6   reading_remarque      int64
 7   counter_coefficient   int64
 8   consommation_level_1  int64
 9   consommation_level_2  int64
 10  consommation_level_3  int64
 11  consommation_level_4  int64
 12  old_index             int64
 13  new_index             int64
 14  months_number         int64
 15  counter_type          object
dtypes: int64(12), object(4)
memory usage: 546.5+ MB
```

In [8]:
```
# Get concise information of each column in dataset
client_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 135493 entries, 0 to 135492
Data columns (total 6 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   disrict        135493 non-null  int64
 1   client_id      135493 non-null  object
 2   client_catg    135493 non-null  int64
 3   region         135493 non-null  int64
 4   creation_date  135493 non-null  object
 5   target         135493 non-null  float64
dtypes: float64(1), int64(3), object(2)
memory usage: 6.2+ MB
```

Finding:

- There are column names that need renaming to increase ease of understanding: `disrict` , `consommation_level_1` , `reading_remarque` among others

---

In [9]:
```
# Rename columns
client_train.rename(columns={'disrict': 'district'}, inplace=True)
client_test.rename(columns={'disrict': 'district'}, inplace=True)

invoice_train.rename(columns={
    'consommation_level_1': 'Consumption_level_1',
    'consommation_level_2': 'Consumption_level_2',
    'consommation_level_3': 'Consumption_level_3',
    'consommation_level_4': 'Consumption_level_4',
    'reading_remarque': 'reading_remarks',
    'tarif_type':'tariff_type' }, inplace=True)

invoice_test.rename(columns={
    'consommation_level_1': 'Consumption_level_1',
    'consommation_level_2': 'Consumption_level_2',
    'consommation_level_3': 'Consumption_level_3',
    'consommation_level_4': 'Consumption_level_4',
    'reading_remarque': 'reading_remarks',
    'tarif_type':'tariff_type' }, inplace=True)
```

In [10]:
```
# Getting unique values on the client data
for col in client_train.columns:
    print(f"{col} - {client_train[col].nunique()}",)
print()
for col in client_test.columns:
    print(f"{col} - {client_test[col].nunique()}")
```

```
district - 4
client_id - 135493
client_catg - 3
region - 25
creation_date - 8088
target - 2

district - 4
client_id - 58069
client_catg - 3
region - 24
creation_date - 7388
```

Finding:

- There is a mismatch in the number of regions in the client datasets (Train and Test) - `Region` is an important column thus we need to reconcile the datasets

---

In [11]:
```
# Drop records in the train dataset with 'Region' feature missing in the test dat
client_train = client_train[client_train['region'].isin(client_test['region'])]
client_train.info()
```

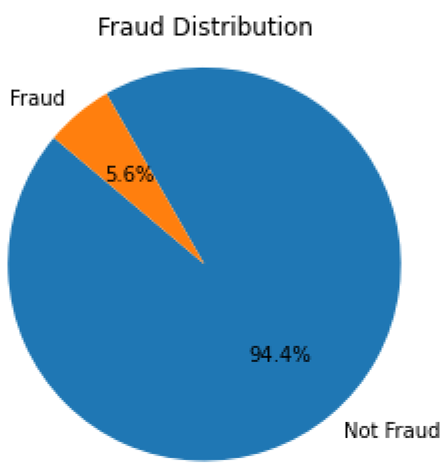
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 135491 entries, 0 to 135492
Data columns (total 6 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   district       135491 non-null  int64  
 1   client_id      135491 non-null  object 
 2   client_catg    135491 non-null  int64  
 3   region         135491 non-null  int64  
 4   creation_date  135491 non-null  object 
 5   target         135491 non-null  float64
dtypes: float64(1), int64(3), object(2)
memory usage: 7.2+ MB
```

In [12]:
```
# check for missing values
print(invoice_train.isnull().sum())
print()
print(client_train.isnull().sum())
```

```
client_id               0
invoice_date            0
tariff_type             0
counter_number          0
counter_statue          0
counter_code            0
reading_remarks         0
counter_coefficient     0
Consumption_level_1     0
Consumption_level_2     0
Consumption_level_3     0
Consumption_level_4     0
old_index               0
new_index               0
months_number           0
counter_type            0
dtype: int64

district         0
client_id        0
client_catg      0
region           0
creation_date    0
target           0
dtype: int64
```

Finding: There are no missing values in the train sets for both Client and Invoice data

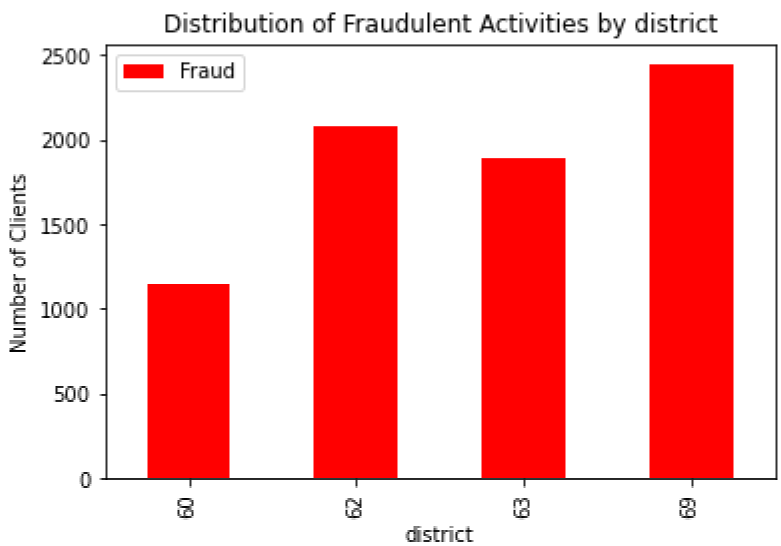
In [13]:
```
# Visualize Target Distribution (bar chart)
fraud_dist = client_train.groupby(['target'])['client_id'].count()
plt.bar(x=fraud_dist.index, height=fraud_dist.values, tick_label = [0,1])
plt.title('Fraud Distribution')
plt.show();

#Visualize Target Distribution (pie chart)
fraud_dist_2 = client_train['target'].value_counts(normalize=True)
plt.pie(fraud_dist_2, labels=['Not Fraud', 'Fraud'], autopct='%1.1f%%', startangl
plt.title('Fraud Distribution')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show();
```

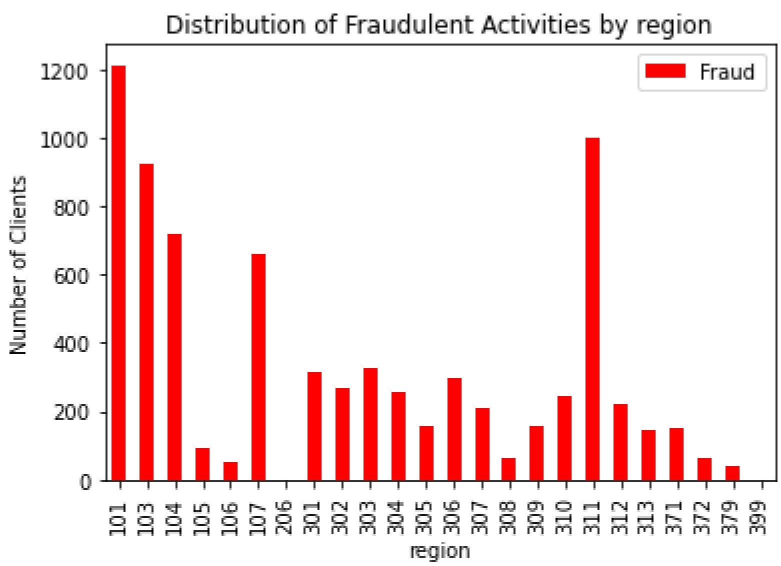


Finding:

- The Target ('Fraud') is highly imbalanced with *5.6%* of client accounts classified as fraudulent while *94.4%* of client accounts classified as not fraudulent.

In [14]:
```
# Visualize Target Distribution across Districts and Regions
for col in ['district', 'region']:
    df = client_train.groupby([col, 'target'])['client_id'].count().unstack()
    df.drop(columns=0, inplace=True)
    df.plot(kind='bar', color='red')
    plt.title(f'Distribution of Fraudulent Activities by {col}')
    plt.xlabel(col)
    plt.ylabel('Number of Clients')
    plt.legend(labels=['Fraud'])
    plt.show();
```





Findings:

- Fraudulent client accounts are reasonably well distributed across regions
- There are a few outlier regions with a much higher number of fraudulent client accounts (101-104, 107,311)

Feature Engineering

In [15]:
```
# convert date columns to date time format
for df in [invoice_train,invoice_test]:
    df['invoice_date'] = pd.to_datetime(df['invoice_date'])

for df in [client_train,client_test]:
    df['creation_date'] = pd.to_datetime(df['creation_date'])
```

In [16]:
```
# Compute the average consumption levels and number of transactions for each clie

def aggregate_by_client_id(invoice_data):
    aggs = {}
    aggs['Consumption_level_1'] = ['mean']
    aggs['Consumption_level_2'] = ['mean']
    aggs['Consumption_level_3'] = ['mean']
    aggs['Consumption_level_4'] = ['mean']

    # Aggregate the invoice data by client_id
    agg_trans = invoice_data.groupby(['client_id']).agg(aggs)

    # Flatten the MultiIndex columns
    agg_trans.columns = ['_'.join(col).strip() for col in agg_trans.columns.value
    agg_trans.reset_index(inplace=True)

    # Return a dataframe showing number of transactions for each client
    trans_count = (invoice_data.groupby('client_id')
            .size()
            .reset_index(name='transactions_count'))

    # Merge transaction count and mean consumption dataframes
    return pd.merge(trans_count, agg_trans, on='client_id', how='left')
```

In [17]:
```
# Group invoice data by client_id
agg_train = aggregate_by_client_id(invoice_train)
agg_test = aggregate_by_client_id(invoice_test)

# Merge aggregate invoice data with client data
train = pd.merge(client_train,agg_train, on='client_id', how='left')
test = pd.merge(client_test,agg_test, on='client_id', how='left')
```

In [18]:
```
train.shape, test.shape
```

Out[18]:
```
((135491, 11), (58069, 10))
```

In [19]:
```
# Encode categorical columns using OneHotEncoder
def encode_categorical_columns(df, columns):
    encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore', drop='
    encoded_cols = encoder.fit_transform(df[columns])
    encoded_df = pd.DataFrame(encoded_cols, columns=encoder.get_feature_names_out
    df = df.drop(columns=columns)
    return pd.concat([df, encoded_df], axis=1)
```

In [20]:
```
# Encode categorical columns
train = encode_categorical_columns(train,['district', 'client_catg', 'region'])
test = encode_categorical_columns(test,['district', 'client_catg', 'region'])

#drop redundant columns
#sub_client_id = test['client_id']
drop_columns = ['client_id', 'creation_date']

for col in drop_columns:
    if col in train.columns:
        train.drop([col], axis=1, inplace=True)
    if col in test.columns:
        test.drop([col], axis=1, inplace=True)
```

In [21]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 135491 entries, 0 to 135490
Data columns (total 34 columns):
 #   Column                   Non-Null Count   Dtype  
---  ------                   --------------   -----  
 0   target                   135491 non-null  float64
 1   transactions_count       135491 non-null  int64  
 2   Consumption_level_1_mean  135491 non-null  float64
 3   Consumption_level_2_mean  135491 non-null  float64
 4   Consumption_level_3_mean  135491 non-null  float64
 5   Consumption_level_4_mean  135491 non-null  float64
 6   district_62              135491 non-null  float64
 7   district_63              135491 non-null  float64
 8   district_69              135491 non-null  float64
 9   client_catg_12           135491 non-null  float64
 10  client_catg_51           135491 non-null  float64
 11  region_103               135491 non-null  float64
 12  region_104               135491 non-null  float64
 13  region_105               135491 non-null  float64
 14  region_106               135491 non-null  float64
 15  region_107               135491 non-null  float64
 16  region_206               135491 non-null  float64
 17  region_301               135491 non-null  float64
 18  region_302               135491 non-null  float64
 19  region_303               135491 non-null  float64
 20  region_304               135491 non-null  float64
 21  region_305               135491 non-null  float64
 22  region_306               135491 non-null  float64
 23  region_307               135491 non-null  float64
 24  region_308               135491 non-null  float64
 25  region_309               135491 non-null  float64
 26  region_310               135491 non-null  float64
 27  region_311               135491 non-null  float64
 28  region_312               135491 non-null  float64
 29  region_313               135491 non-null  float64
 30  region_371               135491 non-null  float64
 31  region_372               135491 non-null  float64
 32  region_379               135491 non-null  float64
 33  region_399               135491 non-null  float64
dtypes: float64(33), int64(1)
memory usage: 36.2 MB
```

In [22]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 58069 entries, 0 to 58068
Data columns (total 33 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   transactions_count        58069 non-null  int64  
 1   Consumption_level_1_mean  58069 non-null  float64
 2   Consumption_level_2_mean  58069 non-null  float64
 3   Consumption_level_3_mean  58069 non-null  float64
 4   Consumption_level_4_mean  58069 non-null  float64
 5   district_62               58069 non-null  float64
 6   district_63               58069 non-null  float64
 7   district_69               58069 non-null  float64
 8   client_catg_12            58069 non-null  float64
 9   client_catg_51            58069 non-null  float64
 10  region_103                58069 non-null  float64
 11  region_104                58069 non-null  float64
 12  region_105                58069 non-null  float64
 13  region_106                58069 non-null  float64
 14  region_107                58069 non-null  float64
 15  region_206                58069 non-null  float64
 16  region_301                58069 non-null  float64
 17  region_302                58069 non-null  float64
 18  region_303                58069 non-null  float64
 19  region_304                58069 non-null  float64
 20  region_305                58069 non-null  float64
 21  region_306                58069 non-null  float64
 22  region_307                58069 non-null  float64
 23  region_308                58069 non-null  float64
 24  region_309                58069 non-null  float64
 25  region_310                58069 non-null  float64
 26  region_311                58069 non-null  float64
 27  region_312                58069 non-null  float64
 28  region_313                58069 non-null  float64
 29  region_371                58069 non-null  float64
 30  region_372                58069 non-null  float64
 31  region_379                58069 non-null  float64
 32  region_399                58069 non-null  float64
dtypes: float64(32), int64(1)
memory usage: 15.1 MB
```

Data Modelling and Evaluation

- We will build 2 baseline models (`Logistic Regression and Decision Trees`) to compare both parametric and non-parametric models.
- We will then enhance these models with tuned hyperparameters and possibly introduce a more robust model if deemed necessary

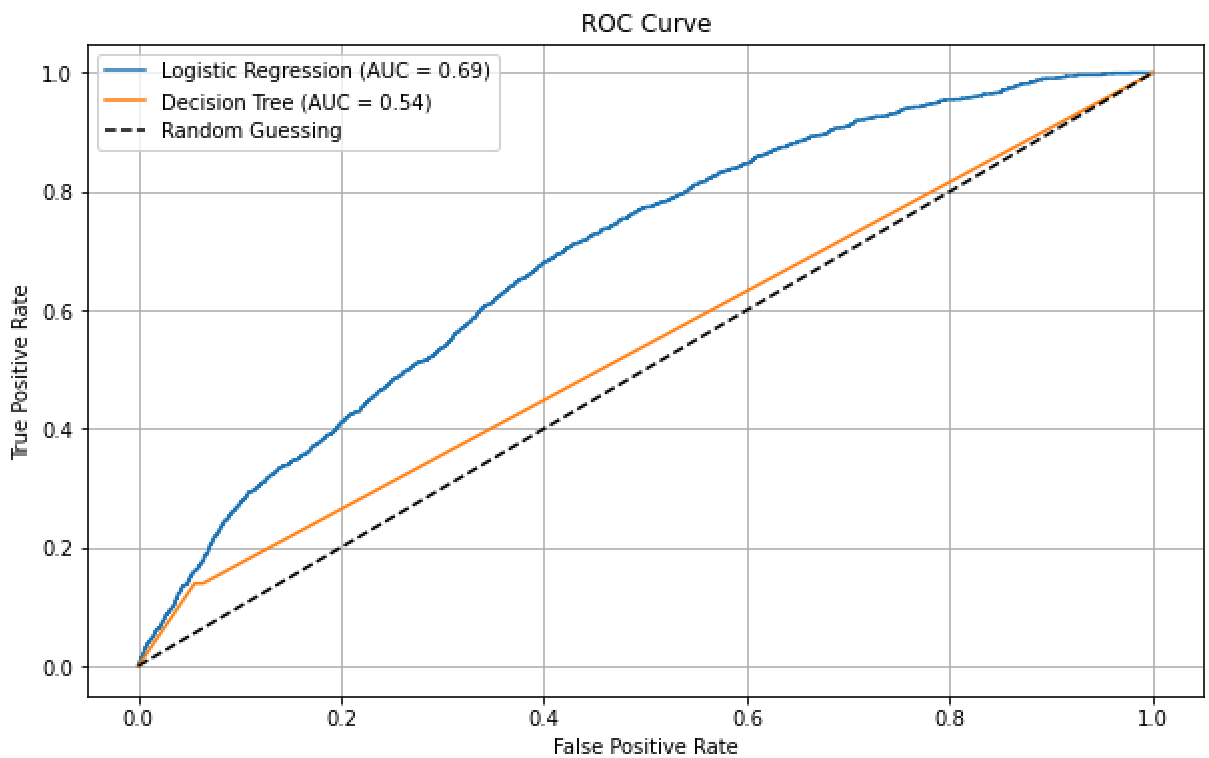
Train + Evaluate Base Models

In [23]:
```
# Define the Target and feature variables
X = train.drop(columns=['target'])
y = train['target']

# Split the train dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

# Logistic Regression Classifier
log_model = LogisticRegression(max_iter=1000, random_state=42)
log_model.fit(X_train, y_train)

# Decision Tree Classifier
decision_model = DecisionTreeClassifier(random_state=42)
decision_model.fit(X_train, y_train)


# Predict on the train set
y_pred_log_train = log_model.predict(X_train)
y_probab_log_train = log_model.predict_proba(X_train)[:, 1]

y_pred_decision_train = decision_model.predict(X_train)
y_probab_decision_train = decision_model.predict_proba(X_train)[:, 1]


# Predict on the test set
y_pred_log_test = log_model.predict(X_test)
y_probab_log_test = log_model.predict_proba(X_test)[:, 1]

y_pred_decision_test = decision_model.predict(X_test)
y_probab_decision_test = decision_model.predict_proba(X_test)[:, 1]


# Evaluate the classifiers
def evaluate_model(y_train, y_test, y_pred_train,y_pred_test,y_probab_train,y_pro
    print(f"Evaluation for {model_name} Model:")
    print("-"*60)
    print("Classification Report (train):")
    print(classification_report(y_train, y_pred_train))
    print("ROC-AUC Score(train):", roc_auc_score(y_train, y_probab_train))
    print("-"*60)
    print("Classification Report (test):")
    print(classification_report(y_test, y_pred_test))
    print("ROC-AUC Score(test):", roc_auc_score(y_test, y_probab_test))
    print("_"*60)


evaluate_model(y_train, y_test, y_pred_log_train,y_pred_log_test,y_probab_log_tra
evaluate_model(y_train, y_test, y_pred_decision_train,y_pred_decision_test,y_prob
```

```
Evaluation for Logistic Regression Model:
------------------------------------------------------------
Classification Report (train):
              precision    recall  f1-score   support

         0.0       0.94      1.00      0.97    102377
         1.0       0.26      0.00      0.01      6015

    accuracy                           0.94    108392
   macro avg       0.60      0.50      0.49    108392
weighted avg       0.91      0.94      0.92    108392

ROC-AUC Score(train): 0.6852343080130762
------------------------------------------------------------
Classification Report (test):
              precision    recall  f1-score   support

         0.0       0.94      1.00      0.97     25548
         1.0       0.23      0.00      0.01      1551

    accuracy                           0.94     27099
   macro avg       0.59      0.50      0.49     27099
weighted avg       0.90      0.94      0.92     27099

ROC-AUC Score(test): 0.6898968927353545
```

```
Evaluation for Decision Tree Model:
------------------------------------------------------------
Classification Report (train):
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    102377
         1.0       1.00      0.99      1.00      6015

    accuracy                           1.00    108392
   macro avg       1.00      1.00      1.00    108392
weighted avg       1.00      1.00      1.00    108392

ROC-AUC Score(train): 0.9999939882849993
------------------------------------------------------------
Classification Report (test):
              precision    recall  f1-score   support

         0.0       0.95      0.94      0.95     25548
         1.0       0.13      0.14      0.14      1551

    accuracy                           0.90     27099
   macro avg       0.54      0.54      0.54     27099
weighted avg       0.90      0.90      0.90     27099

ROC-AUC Score(test): 0.5382818294171642
```

In [24]:
```
# Plot ROC curves for both models
def plot_roc_curve(y_test, y_probab_log_test, y_probab_decision_test):
    fpr_log, tpr_log, _ = roc_curve(y_test, y_probab_log_test)
    fpr_decision, tpr_decision, _ = roc_curve(y_test, y_probab_decision_test)

    plt.figure(figsize=(10, 6))
    plt.plot(fpr_log, tpr_log, label='Logistic Regression (AUC = {:.2f})'.format(
    plt.plot(fpr_decision, tpr_decision, label='Decision Tree (AUC = {:.2f})'.for
    plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend()
    plt.grid()
    plt.show();

plot_roc_curve(y_test, y_probab_log_test, y_probab_decision_test)
```



Summary findings on the Untuned Models

Logistic Regression

- The following are the summary metrics for the model:
  - Train and Test Accuracy scores of **~94%**
  - Recall and F1 score of **0-1%** on test data for the minority class (Class 1: Fraud)
  - AUC Score of **~68%** on both test and training data
- These results indicate that the logistic regression model is performing extremely poorly in detecting fraud (Class 1) with a recall score of *zero*.

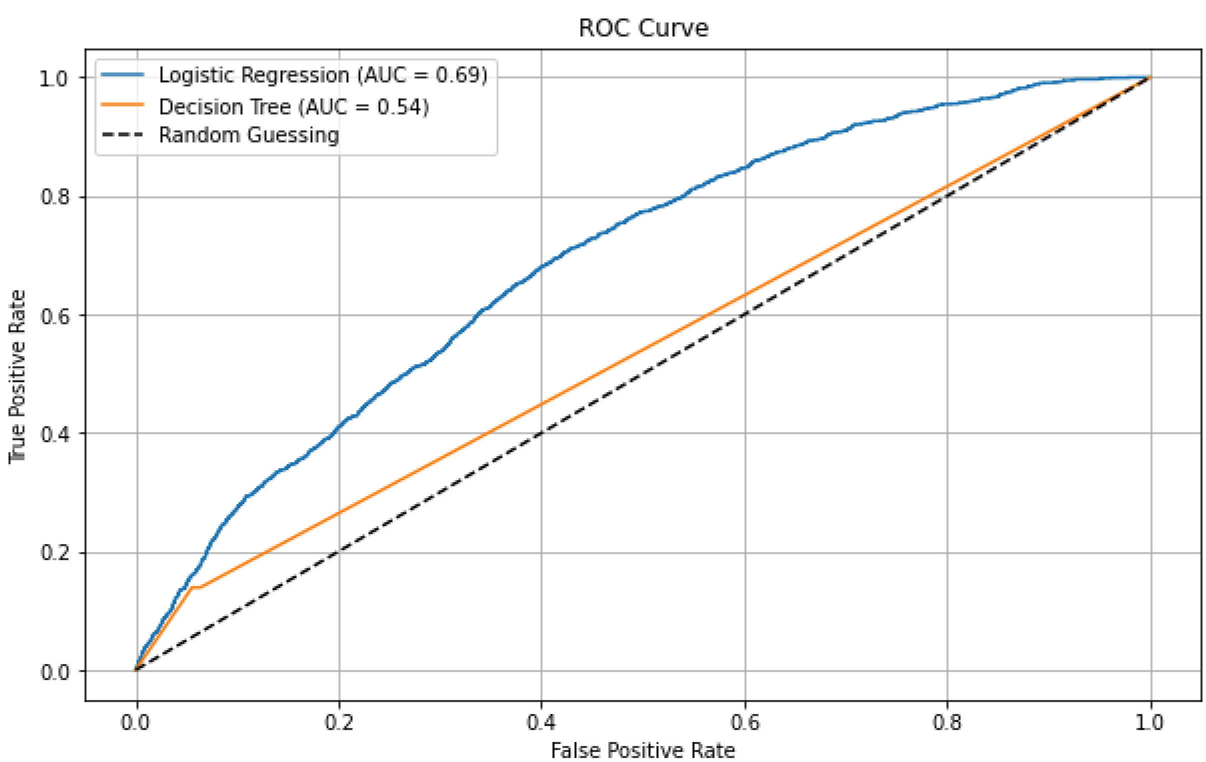
- Accuracy scores are extremely misleading (***94%***) which results from us dealing with a highly imbalanced dataset
- The model performs much better than a random model with stable generalization (***AUC: ~68%*** for both train and test data sets indicating *low overfitting*)

Decision Tree

- The following are the summary metrics for the model:
  - Train Accuracy of **100%** and Test Accuracy of **~90%**
  - Recall and F1 score of **13-14%** on test data for the minority class
  - AUC score of **~100%** on training data (near-perfect model) and **~54%** on test data (slightly better than random)
- These results indicate classic *overfitting* (*Accuracy/AUC: 100%*) on the training dataset with the model basically memorizing the training data
- The model fails to generalize to the test data and barely performs better than a random model (*ROC-AUC: ~54%*)
- These results also indicate a highly imbalanced dataset.

Logistic vs Decision Tree

- The ROC curve indicates that the logistic regression model outperforms the decision tree model in terms of making correct classifications



Summary statistics:

	UNTUNED MODELS			
	Logistic Regression		Decision Tree	
	Train	Test	Train	Test
Accuracy	0.94	0.94	1.00	0.90
Precision (Minority Class)	0.26	0.23	1.00	0.13
Recall (Minority Class)	0.00	0.00	0.99	0.14
F1 score	0.01	0.01	1.00	0.14
AUC score	0.68	0.68	1.00	0.54

Next Steps: Dealing with Class imbalance + Hyperparameter tuning

- To deal with class imbalance and improve model performance, we will do the following:

  1) Use the `precision-recall curve` to find ***better classification thresholds*** (default=0.5 is rarely optimal on imbalanced data)

  2) Use class weights in our models by adding a new field `class_weight = 'balanced'`

  3) Apply oversampling/undersampling techniques on the minority/majority class (**SMOTE**/**SMOTEEN**)

Train + Evaluate Tuned Models

Pipeline 1: Manual Resampling

In [25]:
```
# 1. Define the Target and feature variables
X = train.drop(columns=['target'])
y = train['target']

# 2. Split the train dataset
# Stratify the split to maintain class diversity in both train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0

# 3. Scaling the features - important for models like Logistic Regression and XGE
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 4. Handle class imbalance with SMOTEEN (random under-sampling + SMOTE over-samp
smote = SMOTEENN(random_state=42) # Combination of SMOTE and Edited Nearest Neigh
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train

# After resampling
scale_pos_weight = y_train_resampled.value_counts()[0] / y_train_resampled.value_
print(f"Scale Pos Weight: {scale_pos_weight}")
```

```
  File "C:\Users\HP\anaconda3\envs\learn-env\lib\site-packages\joblib\externals
\loky\backend\context.py", line 257, in _count_physical_cores
    cpu_info = subprocess.run(
  File "C:\Users\HP\anaconda3\envs\learn-env\lib\subprocess.py", line 489, in r
un
    with Popen(*popenargs, **kwargs) as process:
  File "C:\Users\HP\anaconda3\envs\learn-env\lib\subprocess.py", line 854, in _
_init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "C:\Users\HP\anaconda3\envs\learn-env\lib\subprocess.py", line 1307, in
_execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,

Scale Pos Weight: 0.8577078935235738
```

Finding:

- Since our `scale_pos_weight = 0.85` after resampling, this means that our classes are mostly balanced (with the minority class slightly overrepresented)

Train all models on resampled training data

In [26]:
```
# 5. Train Logistic Regression (with class weights, scaling and SMOTE)
log_model = LogisticRegression(class_weight='balanced', random_state=42)
log_model.fit(X_train_resampled, y_train_resampled)

# 6. Train Decision Tree (with class weights, scaling, SMOTE and tuned hyperparam
decision_model = DecisionTreeClassifier(class_weight='balanced', max_depth=10, mi
decision_model.fit(X_train_resampled, y_train_resampled)

# 7. Train XGBoost (with class weights, scaling and SMOTE)
xgb_model = XGBClassifier(eval_metric='logloss', random_state=42)
xgb_model.fit(X_train_resampled, y_train_resampled)

# 8. Find better classification thresholds
best_thresholds_dict = {}

for name, y_probab in [("Logistic Regression", log_model.predict_proba(X_test_sca
                    ("Decision Tree", decision_model.predict_proba(X_test_scaled)
                    ("XGBoost", xgb_model.predict_proba(X_test_scaled)[:, 1])]:



    # Calculate precision, recall, and thresholds
    prec, rec, thresholds = precision_recall_curve(y_test, y_probab)

    f1_scores = 2 * (prec * rec) / (prec + rec + 1e-10)
    best_idx = f1_scores.argmax()
    best_threshold = thresholds[best_idx]

    best_thresholds_dict[name] = round(best_threshold,3)

best_thresholds_dict

# 9. Evaluate all the models

for name, model in [("Logistic Regression", log_model),
                    ("Decision Tree", decision_model),
                    ("XGBoost", xgb_model)]:
    print("_"*80)
    print(f"Evaluation for {name} Model:")
    print("_"*80)

    # Function to apply a threshold to predicted probabilities
    threshold = best_thresholds_dict[name]

    def apply_threshold(y_proba, threshold):
        return (y_proba > threshold).astype(int)

    # Cross-validation setup
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    # Loop through train and test sets for evaluation
    for label, X_eval, y_eval in [("train", X_train_resampled, y_train_resampled)
                                  ("test", X_test_scaled, y_test)]:

        # Predict on the evaluation set
        #y_pred = model.predict(X_eval)
        y_probab = model.predict_proba(X_eval)[:, 1]
```

```
    y_pred = apply_threshold(y_probab, threshold)

    # Perform cross-validation
    cv_scores = cross_val_score(model, X_eval, y_eval, cv=cv, scoring='roc_au

    # Print classification report and ROC-AUC score
    print("-"*80)
    print(f"Classification Report ({label}):")
    print("-"*80)
    print(classification_report(y_eval, y_pred))
    print("-"*80)
    print(f"ROC-AUC Score ({label}): {roc_auc_score(y_eval, y_probab):.3f}")
    print(f"Cross-Validated ROC-AUC Scores ({label}): {cv_scores}")
    print(f"Mean Cross-Validated ROC-AUC Score ({label}): {np.mean(cv_scores)
```

```
________________________________________________________________________________
_
Evaluation for Logistic Regression Model:
________________________________________________________________________________
_
--------------------------------------------------------------------------------
-
Classification Report (train):
--------------------------------------------------------------------------------
-
              precision    recall  f1-score   support

         0.0       0.55      0.89      0.68     70694
         1.0       0.80      0.37      0.51     82422

    accuracy                           0.61    153116
   macro avg       0.67      0.63      0.59    153116
weighted avg       0.68      0.61      0.59    153116

--------------------------------------------------------------------------------
-
ROC-AUC Score (train): 0.776
Cross-Validated ROC-AUC Scores (train): [0.76986881 0.7814549  0.77846086 0.776
70168 0.773     ]
Mean Cross-Validated ROC-AUC Score (train): 0.776
--------------------------------------------------------------------------------
-
Classification Report (test):
--------------------------------------------------------------------------------
-
              precision    recall  f1-score   support

         0.0       0.96      0.85      0.90     25586
         1.0       0.13      0.37      0.19      1513

    accuracy                           0.82     27099
   macro avg       0.54      0.61      0.54     27099
weighted avg       0.91      0.82      0.86     27099

--------------------------------------------------------------------------------
-
ROC-AUC Score (test): 0.704
Cross-Validated ROC-AUC Scores (test): [0.6961335  0.68776117 0.69831036 0.6938
7939 0.71947618]
Mean Cross-Validated ROC-AUC Score (test): 0.699
________________________________________________________________________________
_
Evaluation for Decision Tree Model:
________________________________________________________________________________
_
--------------------------------------------------------------------------------
-
Classification Report (train):
--------------------------------------------------------------------------------
-
              precision    recall  f1-score   support
```

```
         0.0       0.73      0.88      0.80     70694
         1.0       0.87      0.72      0.79     82422

    accuracy                           0.79    153116
   macro avg       0.80      0.80      0.79    153116
weighted avg       0.81      0.79      0.79    153116

-------------------------------------------------------------------------------
-
ROC-AUC Score (train): 0.904
Cross-Validated ROC-AUC Scores (train): [0.89080739 0.89747661 0.8926707  0.895
76971 0.89227131]
Mean Cross-Validated ROC-AUC Score (train): 0.894
-------------------------------------------------------------------------------
-
Classification Report (test):
-------------------------------------------------------------------------------
-
              precision    recall  f1-score   support

         0.0       0.96      0.80      0.88     25586
         1.0       0.13      0.49      0.21      1513

    accuracy                           0.79     27099
   macro avg       0.55      0.65      0.54     27099
weighted avg       0.92      0.79      0.84     27099

-------------------------------------------------------------------------------
-
ROC-AUC Score (test): 0.713
Cross-Validated ROC-AUC Scores (test): [0.66911129 0.64800145 0.65393521 0.6679
1888 0.65709419]
Mean Cross-Validated ROC-AUC Score (test): 0.659
_______________________________________________________________________________
_
Evaluation for XGBoost Model:
_______________________________________________________________________________
_
-------------------------------------------------------------------------------
-
Classification Report (train):
-------------------------------------------------------------------------------
-
              precision    recall  f1-score   support

         0.0       0.93      0.94      0.94     70694
         1.0       0.95      0.94      0.94     82422

    accuracy                           0.94    153116
   macro avg       0.94      0.94      0.94    153116
weighted avg       0.94      0.94      0.94    153116

-------------------------------------------------------------------------------
-
ROC-AUC Score (train): 0.986
Cross-Validated ROC-AUC Scores (train): [0.98168277 0.98157325 0.98179679 0.980
46998 0.98043377]
```

```
Mean Cross-Validated ROC-AUC Score (train): 0.981
-------------------------------------------------------------------------------
-
Classification Report (test):
-------------------------------------------------------------------------------
-
              precision    recall  f1-score   support

         0.0       0.96      0.87      0.91     25586
         1.0       0.15      0.37      0.21      1513

    accuracy                           0.84     27099
   macro avg       0.55      0.62      0.56     27099
weighted avg       0.91      0.84      0.87     27099

-------------------------------------------------------------------------------
-
ROC-AUC Score (test): 0.736
Cross-Validated ROC-AUC Scores (test): [0.71245397 0.72056808 0.71215085 0.7059
8555 0.71891481]
Mean Cross-Validated ROC-AUC Score (test): 0.714
```

- In our first pipeline, we carry out resampling (using SMOTEENN) once before model training. SMOTE also uses full training data
- This comes with a risk of data leakage and inflated performance on the training set as seen in the table below

	TUNED MODELS					
	Logistic Regression		Decision Tree		XGBoost (before GridSearch)	
	Train	*Test*	*Train*	*Test*	*Train*	*Test*
Accuracy	0.82	0.82	0.79	0.79	0.95	0.84
Precision (Minority Class)	0.80	0.13	0.87	0.13	0.95	0.15
Recall (Minority Class)	0.37	0.37	0.72	0.49	0.94	0.37
F1 score	0.51	0.19	0.79	0.21	0.94	0.21
AUC score	0.78	0.70	0.90	0.71	0.99	0.74
AUC_cv	0.78	0.70	0.89	0.72	0.98	0.74

- As a result of this data leakage, later evaluation (cross validation) is not honest and the Train/Test gap is very wide (poor generalization)

Pipeline 2: Using `ImbPipeline`

- To mitigate this issue of inflated model performance, we can build our pipeline using `ImbPipeline` which allows us to do the following:

  a) Apply SMOTEENN within each CV fold

  b) Apply SMOTE only within each training fold - meaning no leakage from the validation set into synthetic data

In [27]:
```
# 1. Define target and features
X = train.drop(columns=['target'])
y = train['target']

# 2. Train/test split (stratified)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0

# 3. Define models
models = {
    "Logistic Regression": LogisticRegression(class_weight='balanced', random_sta
    "Decision Tree": DecisionTreeClassifier(class_weight='balanced', max_depth=10
    "XGBoost": XGBClassifier(eval_metric='logloss', random_state=42)
}

# 4. Define threshold per model (from previous tuning)
best_thresholds = {
    "Logistic Regression": 0.697,
    "Decision Tree": 0.661,
    "XGBoost": 0.545
    }

# 5. Setup cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# 6. Evaluation function
def evaluate_model(name, model, threshold):
    print("="*80)
    print(f"Evaluating: {name}")
    print("="*80)

    # Create pipeline: Scaling + SMOTE + Model
    pipe = ImbPipeline([
        ('scaler', StandardScaler()),
        ('smoteen', SMOTEENN(random_state=42)),
        ('model', model)
    ])

    # Fit pipeline on training data
    pipe.fit(X_train, y_train)

    # Predict probabilities on train and test sets
    for label, X_eval, y_eval in [("Train", X_train, y_train), ("Test", X_test, y
        y_proba = pipe.predict_proba(X_eval)[:, 1]
        y_pred = (y_proba > threshold).astype(int)

        print(f"Classification Report ({label}):")
        print("-"*80)
        print(classification_report(y_eval, y_pred))
        print(f"AUC Score ({label}): {roc_auc_score(y_eval, y_proba):.3f}")
        print("-"*80)

    # Cross-validated AUC
    y_cv_proba = cross_val_predict(pipe, X_train, y_train, cv=cv, method='predict
    y_cv_pred = (y_cv_proba > threshold).astype(int)

    print(f"Cross-Validated AUC Score: {roc_auc_score(y_train, y_cv_proba):.3f}")
    print("="*80)
```

```
# 7. Run evaluation for each model
for name, model in models.items():
    threshold = best_thresholds[name]
    evaluate_model(name, model, threshold)
```

```
===============================================================================
=
Evaluating: Logistic Regression
===============================================================================
=
Classification Report (Train):
-------------------------------------------------------------------------------
-
              precision    recall  f1-score   support

         0.0       0.96      0.85      0.90    102339
         1.0       0.12      0.35      0.18      6053

    accuracy                           0.82    108392
   macro avg       0.54      0.60      0.54    108392
weighted avg       0.91      0.82      0.86    108392

AUC Score (Train): 0.701
-------------------------------------------------------------------------------
-
Classification Report (Test):
-------------------------------------------------------------------------------
-
              precision    recall  f1-score   support

         0.0       0.96      0.85      0.90     25586
         1.0       0.13      0.37      0.19      1513

    accuracy                           0.82     27099
   macro avg       0.54      0.61      0.54     27099
weighted avg       0.91      0.82      0.86     27099

AUC Score (Test): 0.704
-------------------------------------------------------------------------------
-
Cross-Validated AUC Score: 0.699
===============================================================================
=
===============================================================================
=
Evaluating: Decision Tree
===============================================================================
=
Classification Report (Train):
-------------------------------------------------------------------------------
-
              precision    recall  f1-score   support

         0.0       0.97      0.81      0.88    102339
         1.0       0.14      0.54      0.23      6053

    accuracy                           0.79    108392
   macro avg       0.56      0.68      0.55    108392
weighted avg       0.92      0.79      0.84    108392

AUC Score (Train): 0.762
-------------------------------------------------------------------------------
```

```
-
Classification Report (Test):
--------------------------------------------------------------------------------
-
              precision    recall  f1-score   support

         0.0       0.96      0.80      0.88     25586
         1.0       0.13      0.49      0.21      1513

    accuracy                           0.79     27099
   macro avg       0.55      0.65      0.54     27099
weighted avg       0.92      0.79      0.84     27099

AUC Score (Test): 0.713
--------------------------------------------------------------------------------
-
Cross-Validated AUC Score: 0.717
================================================================================
=
================================================================================
=
Evaluating: XGBoost
================================================================================
=
Classification Report (Train):
--------------------------------------------------------------------------------
-
              precision    recall  f1-score   support

         0.0       0.97      0.88      0.92    102339
         1.0       0.20      0.50      0.28      6053

    accuracy                           0.86    108392
   macro avg       0.58      0.69      0.60    108392
weighted avg       0.92      0.86      0.89    108392

AUC Score (Train): 0.800
--------------------------------------------------------------------------------
-
Classification Report (Test):
--------------------------------------------------------------------------------
-
              precision    recall  f1-score   support

         0.0       0.96      0.87      0.91     25586
         1.0       0.15      0.37      0.21      1513

    accuracy                           0.84     27099
   macro avg       0.55      0.62      0.56     27099
weighted avg       0.91      0.84      0.87     27099

AUC Score (Test): 0.736
--------------------------------------------------------------------------------
-
Cross-Validated AUC Score: 0.736
================================================================================
=
```

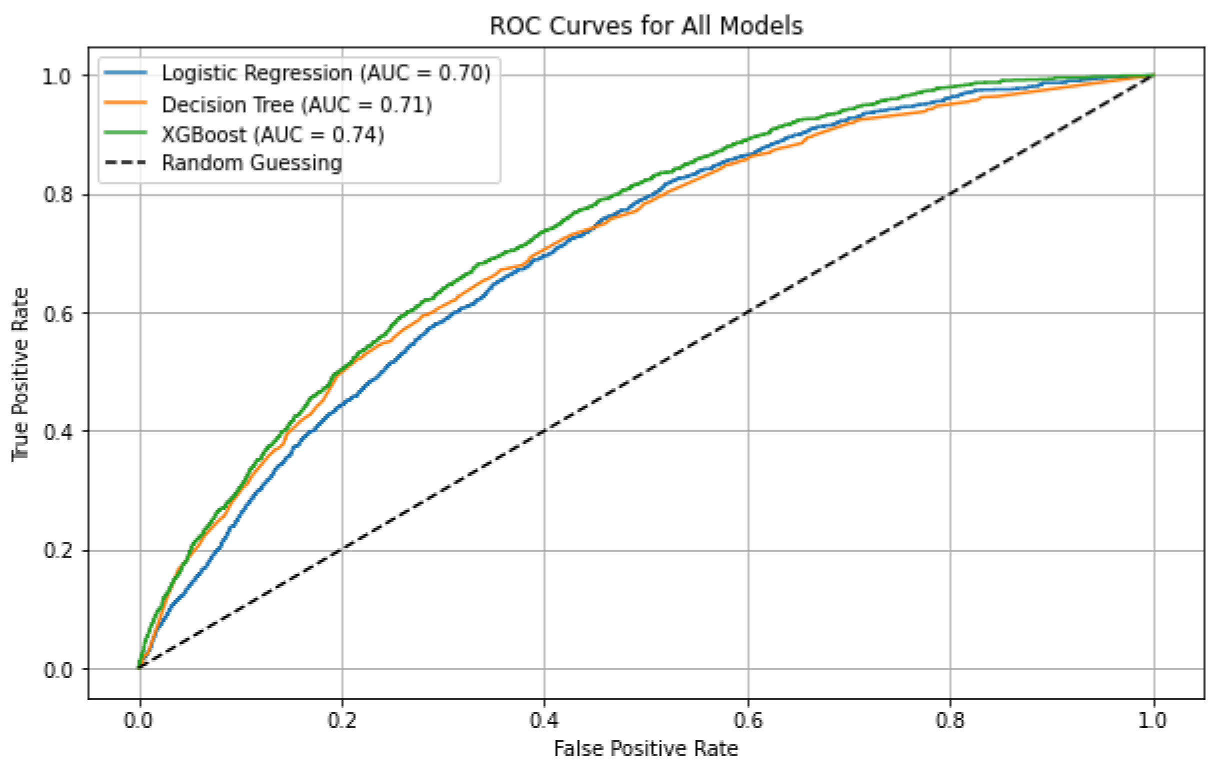
- Model results from Pipeline 2 thus end up looking like this:

	TUNED MODELS					
	Logistic Regression		Decision Tree		XGBoost (before GridSearch)	
	Train	*Test*	*Train*	*Test*	*Train*	*Test*
Accuracy	0.82	0.82	0.79	0.79	0.86	0.84
Precision (Minority Class)	0.12	0.13	0.14	0.13	0.20	0.15
Recall (Minority Class)	0.35	0.37	0.54	0.49	0.50	0.37
F1 score	0.18	0.19	0.23	0.21	0.28	0.21
AUC score	0.70	0.70	0.76	0.71	0.80	0.74
AUC_cv		0.70		0.72		0.74

- We can see that this pipeline gives a truer picture of model performance on the training dataset - the Train/Test gap is less wide making the model more reliable (great generalization)

Choosing the best-performing model

In [28]:
```
# Plot the ROC curves for all models
def plot_roc_curves(models, X_test_scaled, y_test):
    plt.figure(figsize=(10, 6))

    for name, model in models.items():
        y_probab = model.predict_proba(X_test_scaled)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_probab)
        plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc_score(y_test, y_probab)

    plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curves for All Models')
    plt.legend()
    plt.grid()
    plt.show();

# Define the models to plot
models_to_plot = {
    "Logistic Regression": log_model,
    "Decision Tree": decision_model,
    "XGBoost": xgb_model
}

# Plot the ROC curves
plot_roc_curves(models_to_plot, X_test_scaled, y_test)
```



Summary Findings on the Tuned Models (Pipeline 2)

Logistic Regression (Tuned)

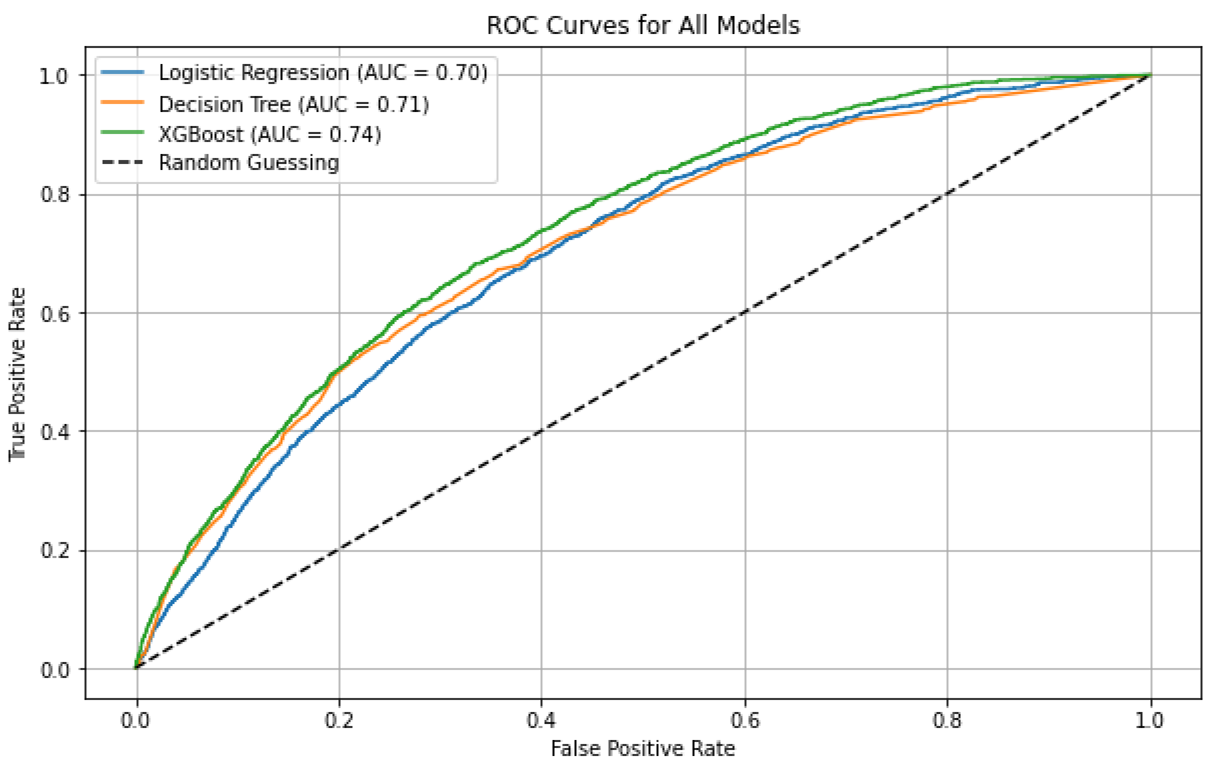
- The following are the summary metrics for the tuned model:
  - Train/Test Accuracy: **82%**
  - Recall: **37%**, F1 score: **19%** - on test data for the minority class (Class 1: Fraud)
  - AUC Score: **70%** on test data
- The tuned model shows a significant increase in performance on all metrics: recall (increase of ***36%***), F1 score (increase of ***18%***)

Decision Tree (Tuned)

- The following are the summary metrics for the tuned model:
  - Train/Test Accuracy: **79%**
  - Recall: **49%**, F1 score: **21%** - on test data for the minority class (Class 1: Fraud)
  - AUC Score: **71%** on test data
- The tuned model shows a significant increase in performance on all metrics when compared to the base models: recall (increase of ***35%***), F1 score (increase of ***7%***)

XGBoost (Tuned)

- The following are the summary metrics for the tuned model:
  - Train Accuracy: **86%**, Test Accuracy: **84%**
  - Recall: **37%**, F1 score: **21%** - on test data for the minority class (Class 1: Fraud)
  - AUC Score: **74%** on test data
- As per the ROC curves, `XGBoost` outperforms all other models in terms of predictive power (`AUC = 0.74`)



- In terms of the other classification metrics (accuracy, F1-score), `XGBoost` also marginally outperforms other models (1st in accuracy, 1st in F1 score)

Summary statistics

	TUNED MODELS					
	Logistic Regression		Decision Tree		XGBoost (before GridSearch)	
	Train	*Test*	*Train*	*Test*	*Train*	*Test*
Accuracy	0.82	0.82	0.79	0.79	0.94	0.84
Precision (Minority Class)	0.12	0.13	0.14	0.13	0.20	0.15
Recall (Minority Class)	0.35	0.37	0.54	0.49	0.50	0.37
F1 score	0.18	0.19	0.23	0.21	0.28	0.21
AUC score	0.70	0.70	0.76	0.71	0.80	0.74
AUC_cv		0.70		0.72		0.74

Using `GridSearchCV` on XGBoost (further tuning)

In [29]:
```
# 1. Define target and features
X = train.drop(columns=['target'])
y = train['target']

# 2. Train/test split (stratified)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=(

# 3. Define models and their parameter grids
model_grids = {
    "XGBoost": {
        "model": XGBClassifier(eval_metric='logloss', use_label_encoder=False, ra
        "params": {
            'model__n_estimators': [100],
            'model__max_depth': [3, 5, 7],
            'model__scale_pos_weight': [1, 2, 5]  # adjust based on class imbalar
        }
    }
}

# 4. Set up cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# 5. Helper: find best threshold based on F1
def find_best_threshold(y_true, y_prob):
    prec, rec, thresholds = precision_recall_curve(y_true, y_prob)
    f1 = 2 * (prec * rec) / (prec + rec + 1e-10)
    best_idx = np.argmax(f1)
    return thresholds[best_idx], f1[best_idx]

# 6. Main evaluation loop
for name, config in model_grids.items():
    print("="*100)
    print(f"Running GridSearchCV for: {name}")
    print("="*100)

    # Build pipeline: scaler + SMOTE + model
    pipe = ImbPipeline([
        ('scaler', StandardScaler()),
        ('smoteen', SMOTEENN(random_state=42)),
        ('model', config['model'])
    ])

    # Grid search
    grid = GridSearchCV(pipe, config['params'], scoring='roc_auc', cv=cv, n_jobs=
    grid.fit(X_train, y_train)

    print(f"Best Parameters: {grid.best_params_}")
    print(f"Best Cross-Validated AUC: {grid.best_score_:.3f}")

    # Evaluate on test set
    best_model = grid.best_estimator_

    # Predict probabilities
    y_test_proba = best_model.predict_proba(X_test)[:, 1]
    y_train_proba = best_model.predict_proba(X_train)[:, 1]

    # Find optimal threshold on training data
```

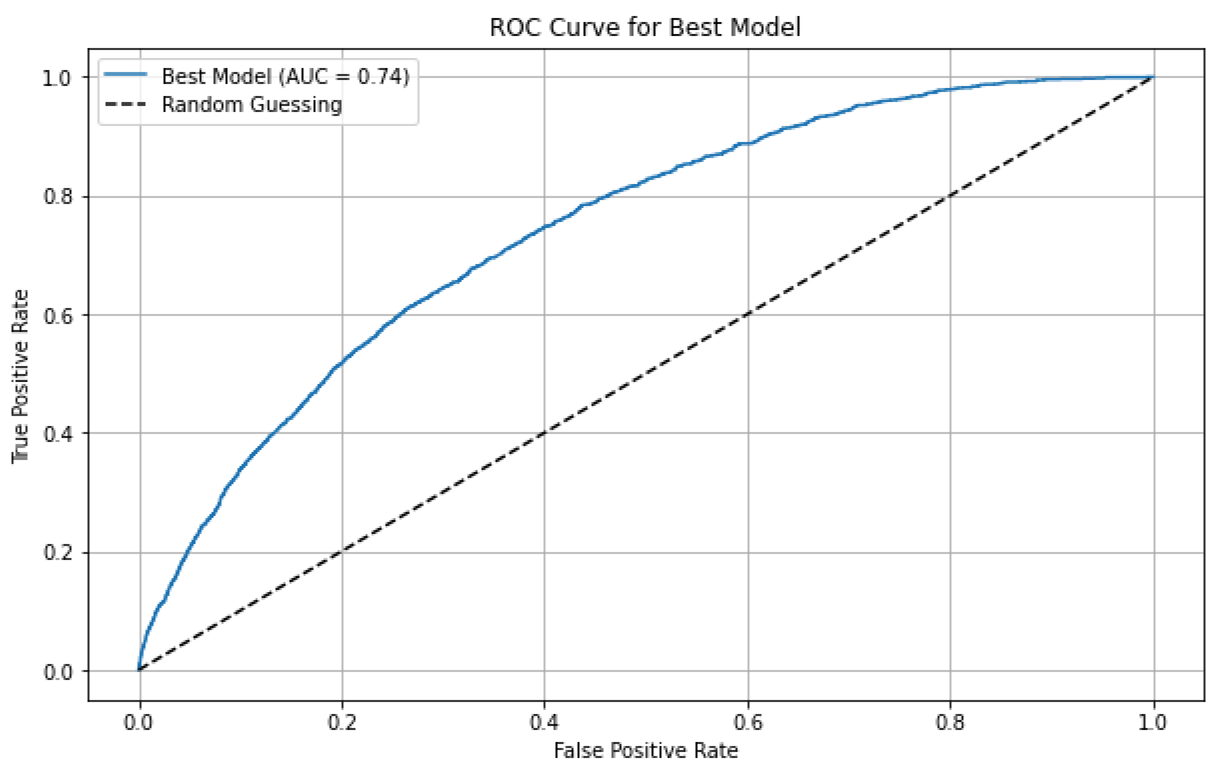
```
    threshold, best_f1 = find_best_threshold(y_train, y_train_proba)
    print(f"Best F1 Threshold (train): {threshold:.3f} (F1: {best_f1:.3f})")

    # Apply threshold
    y_test_pred = (y_test_proba > threshold).astype(int)

    # Report
    print("\nTest Classification Report:")
    print(classification_report(y_test, y_test_pred))
    print(f"Test ROC-AUC: {roc_auc_score(y_test, y_test_proba):.3f}")

    print("="*100 + "\n")
```

```
================================================================================
====================
Running GridSearchCV for: XGBoost
================================================================================
====================
[02:04:56] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release_1.2.
0\src\learner.cc:516:
Parameters: { use_label_encoder } might not be used.

  This may not be accurate due to some parameters are only used in language bin
dings but
  passed down to XGBoost core.  Or some parameters are not used but slip throug
h this
  verification. Please open an issue if you find above cases.


Best Parameters: {'model__max_depth': 3, 'model__n_estimators': 100, 'model__sc
ale_pos_weight': 1}
Best Cross-Validated AUC: 0.739
Best F1 Threshold (train): 0.714 (F1: 0.228)

Test Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      0.87      0.92     25586
         1.0       0.15      0.39      0.22      1513

    accuracy                           0.85     27099
   macro avg       0.56      0.63      0.57     27099
weighted avg       0.92      0.85      0.88     27099

Test ROC-AUC: 0.742
================================================================================
====================
```

In [33]:
```
# Predict probabilities
y_train_proba = best_model.predict_proba(X_train)[:, 1]

# Apply threshold
y_train_pred = (y_train_proba > threshold).astype(int)

# Report
print("\nTrain Classification Report:")
print(classification_report(y_train, y_train_pred))
print(f"Train ROC-AUC: {roc_auc_score(y_train, y_train_proba):.3f}")

print("="*100 + "\n")
```

```
Train Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      0.87      0.92    102339
         1.0       0.16      0.41      0.23      6053

    accuracy                           0.85    108392
   macro avg       0.56      0.64      0.57    108392
weighted avg       0.92      0.85      0.88    108392

Train ROC-AUC: 0.755
================================================================================
====================
```

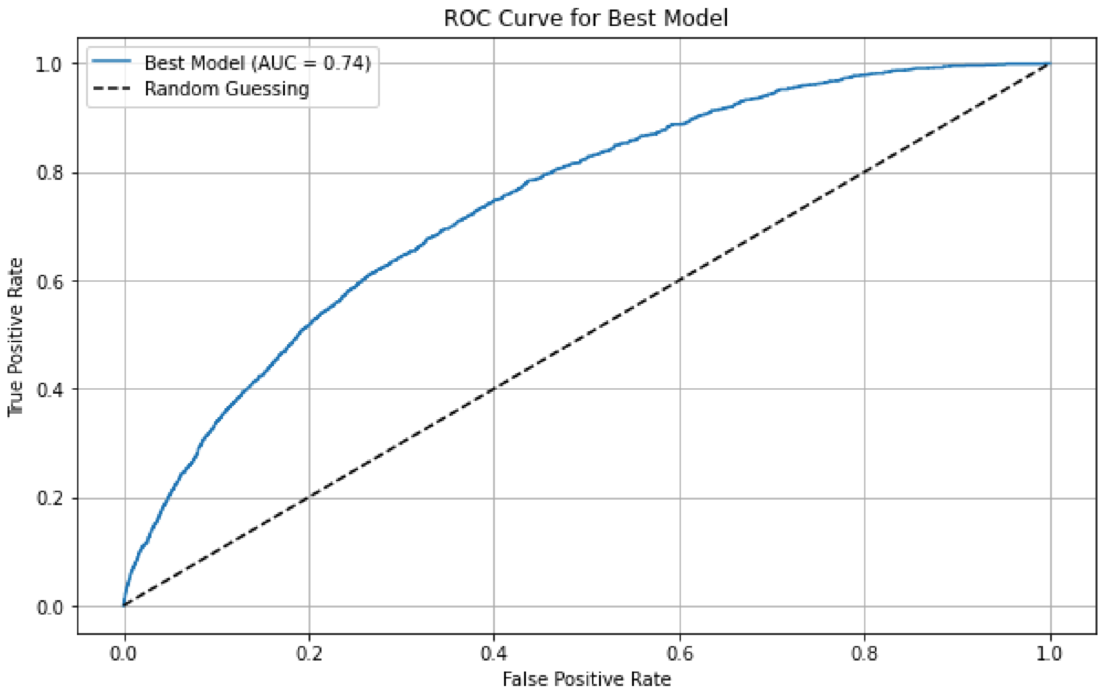
In [30]:
```
# Plot the ROC curve for the best model
def plot_best_model_roc_curve(best_model, X_test, y_test):
    y_probab = best_model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_probab)

    plt.figure(figsize=(10, 6))
    plt.plot(fpr, tpr, label=f'Best Model (AUC = {roc_auc_score(y_test, y_probab)
    plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve for Best Model')
    plt.legend()
    plt.grid()
    plt.show();

# Plot ROC curve for the best model
best_model = grid.best_estimator_
plot_best_model_roc_curve(best_model, X_test, y_test)
```



Summary Findings on the Best Model

(XGBoost - GridSearchCV)

- The following are the summary metrics for the tuned model:
  - Train/Test Accuracy: **85%**
  - Recall: **39%**, F1 score: **22%** - on test data for the minority class (Class 1: Fraud)
  - AUC Score: **74%** on test data
  - As per the ROC curve, XGBoost tuned via GridSearchCV also has an `AUC = 0.74`



Summary Statistics

	XGBoost (After GridSearch)	
	Train	*Test*
Accuracy	0.85	0.85
Precision (Minority Class)	0.16	0.15
Recall (Minority Class)	0.41	0.39
F1 score	0.23	0.22
AUC score	0.76	0.74
AUC_cv		0.74

Conclusion and Next Steps

Conclusions

- The model with the highest predictive power/performance is:
  - Name: **Tuned XGBoost**
  - Tuning technique: **GridSearchCV**
  - Best Model (ROC)AUC Score: **74%**
  - Best Model F1 Score: **22%**

Next Steps:

- **Feature transparency**: Identify the top features most predictive of fraud using SHAP: region or district? Consumption threshold?

- Incorporate Precision-Recall AUC (better than ROC AUC for imbalance)
- Deploy the model onto the client systems

In [ ]:

```

```