Water Utility Intelligence Model: Payment Compliance Prediction (Classification)

Predict which customers are likely to default in future.

Analyst: Elizabeth Nyadimo

Date: 2025

Water billing compliance remains a persistent challenge for utility companies across Kenya. While many have adopted digital payment channels and modern infrastructure, issues like revenue leakage, customer defaults, and dormant accounts continue to undermine financial sustainability.

This analysis takes a predictive approach—going beyond historical reporting to forecast which customers are likely to default. By modeling payment compliance based on consumption behavior, telco-linked transactions, and account activity, we aim to provide utilities with an early warning system.

The ultimate goal is to support proactive engagement, smarter credit control, and revenue recovery strategies—turning raw operational data into real-time, actionable intelligence.

Goal:

- **Predict whether a customer is compliant (Paid) or non-compliant (Not Paid) using:**
- **Features: consumption, telco, customer type, zone**

- **Target: payment_compliance (binary: Paid = 1, Not Paid = 0)**
- **Models: Logistic Regression, Random Forest, XGBoost**

Step 1: Import necessary libraries

In [35]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
%pip install xgboost
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
Requirement already satisfied: xgboost in c:\users\hp\anaconda3\lib\site-packages
(3.0.2)Note: you may need to restart the kernel to use updated packages.

Requirement already satisfied: numpy in c:\users\hp\anaconda3\lib\site-packages (fro
m xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\hp\anaconda3\lib\site-packages (fro
m xgboost) (1.11.4)
```

Step 2: Load the Dataset

In [36]:

```
df=pd.read_excel("final_water_data.xlsx")

# Quick Look
print(df.shape)
df.head()
```

```
(1301, 62)
```

Out[36]:

	name	telco_category	meter_type	status	customer_type	zone	line	january_pr
0	Person_1	Safaricom	REGULAR	ACTIVE	INDIVIDUAL	Zone_1	Line_1	
1	Person_2	Safaricom	LAISON	ACTIVE	INDIVIDUAL	Zone_2	Line_2	
2	Person_3	Safaricom	LAISON	ACTIVE	INDIVIDUAL	Zone_1	Line_1	
3	Person_4	Safaricom	LAISON	ACTIVE	INDIVIDUAL	Zone_1	Line_1	
4	Person_5	Safaricom	LAISON	ACTIVE	INDIVIDUAL	Zone_1	Line_1	

5 rows × 62 columns

Step 3: Define Target Variables

- ### Redefine billing and payment columns

In [37]:
```
bill_columns = [col for col in df.columns if '_billing' in col]
pay_columns = [col for col in df.columns if '_payment' in col]

df['total_billing_5m'] = df[bill_columns].sum(axis=1)
df['total_payment_5m'] = df[pay_columns].sum(axis=1)
df['payment_compliance'] = (df['total_payment_5m'] >= df['total_billing_5m']).astyp
```

Step 4: Feature Engineering - Average Consumption (Jan to May)

In [38]:
```
consumption_columns = [col for col in df.columns if '_consumption' in col]
df['avg_consumption'] = df[consumption_columns].mean(axis=1)
```

Step 5: Select Features for Modeling

In [44]:
```
features = ['avg_consumption', 'telco_category','meter_type','status','customer_typ
X = df[features]
y = df['payment_compliance']
```

Step 6: Encode Categorical Variables

In [45]:
```
X_encoded = X.copy()
label_encoders = {}
for col in ['telco_category','meter_type','status','customer_type', 'zone','line']:
    le = LabelEncoder()
    X_encoded[col] = le.fit_transform(X_encoded[col].astype(str))
    label_encoders[col] = le  # Save encoders if needed later
```

Step 7: Scale Numeric Features

In [46]:
```
scaler = StandardScaler()
X_encoded['avg_consumption'] = scaler.fit_transform(X_encoded[['avg_consumption']])
```

Step 8: Train-Test Split

In [47]:
```
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, ra
```

Step 9: Train Models

- ### Logistic Regression

In [48]:
```
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

Out[48]:
```
LogisticRegression
LogisticRegression()
```

- ## Random Forest

In [49]:
```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

Out[49]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

- ## XGBoost

In [50]:
```
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42
xgb.fit(X_train, y_train)
```

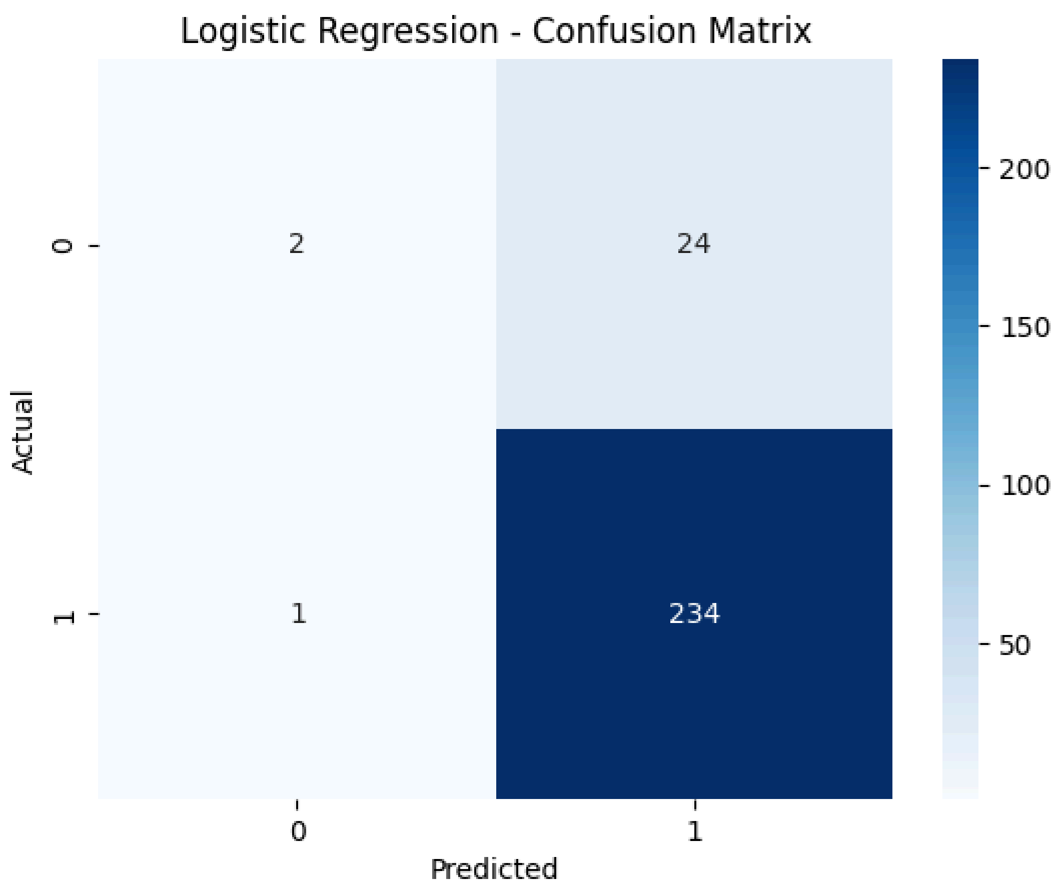
Out[50]:
```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds
=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin
=None,
```

Step 10: Evaluate Models

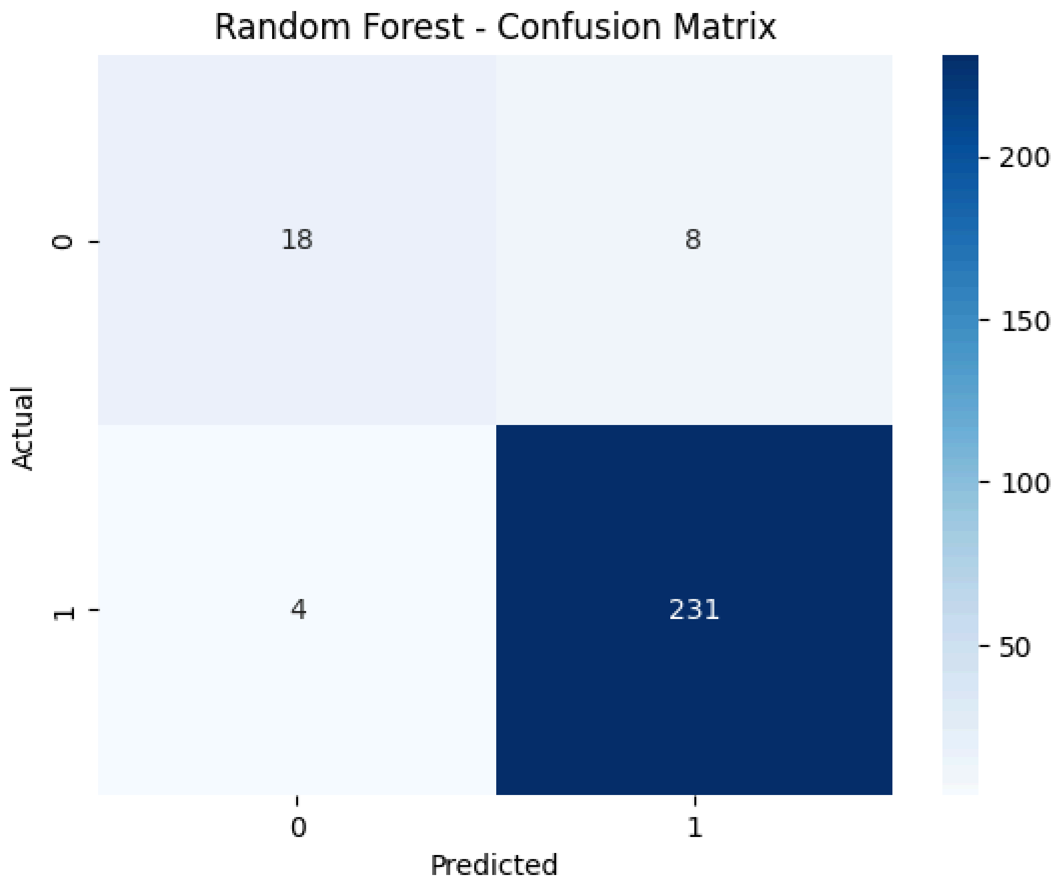
In [51]:
```
models = {'Logistic Regression': lr, 'Random Forest': rf, 'XGBoost': xgb}

for name, model in models.items():
    y_pred = model.predict(X_test)
    print(f"\n{name} Classification Report:\n")
    print(classification_report(y_test, y_pred))

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"{name} - Confusion Matrix")
```

```
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

```
Logistic Regression Classification Report:

              precision    recall  f1-score   support

           0       0.67      0.08      0.14        26
           1       0.91      1.00      0.95       235

    accuracy                           0.90       261
   macro avg       0.79      0.54      0.54       261
weighted avg       0.88      0.90      0.87       261
```

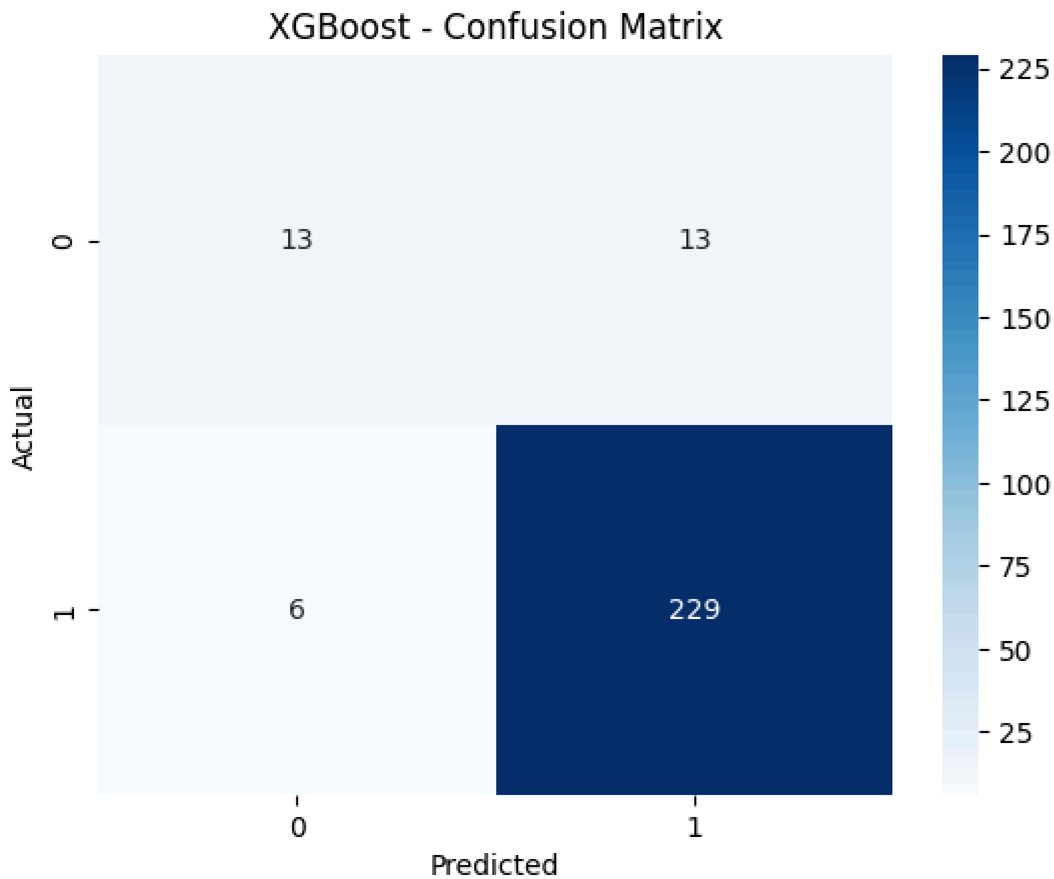



```
Random Forest Classification Report:

              precision    recall  f1-score   support

           0       0.82      0.69      0.75        26
           1       0.97      0.98      0.97       235

    accuracy                           0.95       261
   macro avg       0.89      0.84      0.86       261
weighted avg       0.95      0.95      0.95       261
```



XGBoost Classification Report:

	precision	recall	f1-score	support
0	0.68	0.50	0.58	26
1	0.95	0.97	0.96	235
accuracy			0.93	261
macro avg	0.82	0.74	0.77	261
weighted avg	0.92	0.93	0.92	261



B. TIME SERIES FORECASTING (OPTIONALg

Aggregate to Monthly Total (System-wide Forecasting)

- Reshape Data to Monthly Time Series

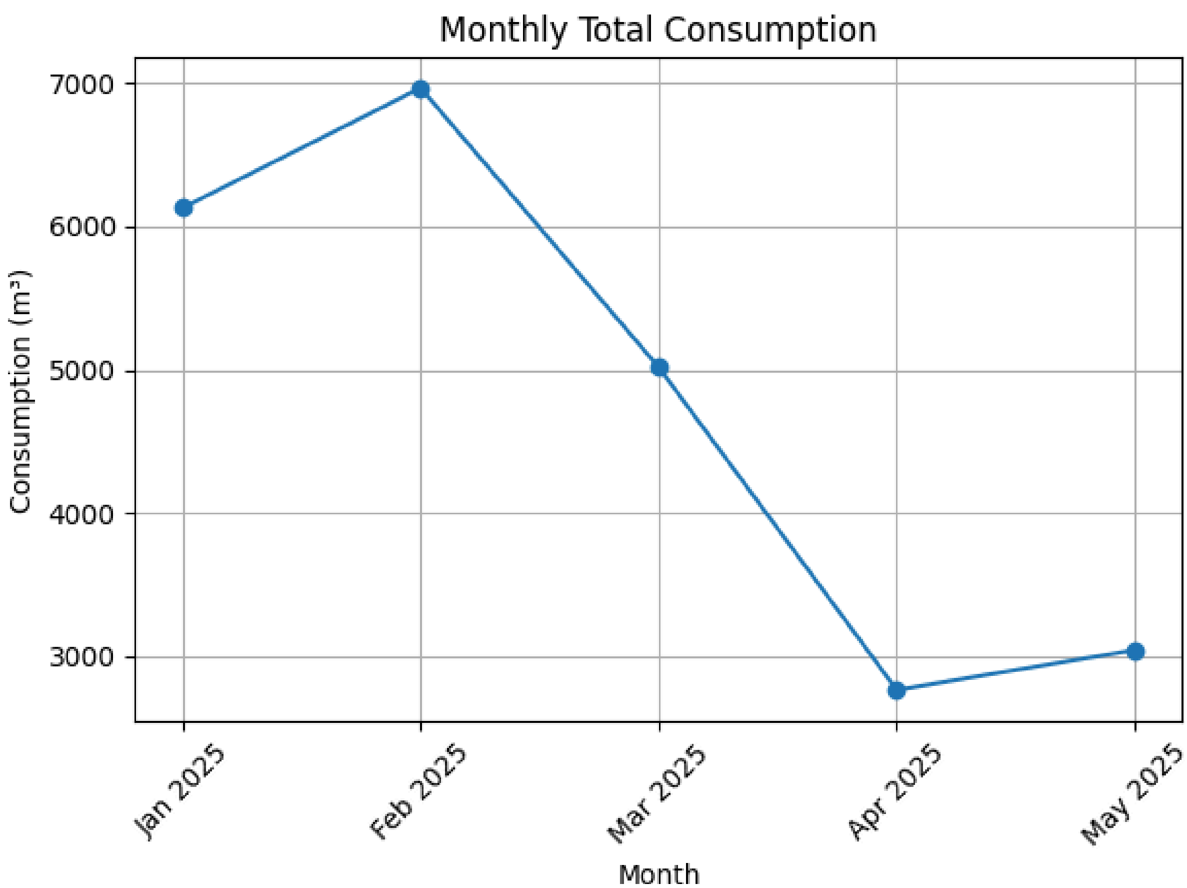
In [52]:
```
monthly_consumption = df[['january_consumption', 'february_consumption', 'march_con
monthly_consumption.index = ['2025-01', '2025-02', '2025-03', '2025-04', '2025-05']
monthly_consumption = monthly_consumption.reset_index()
monthly_consumption.columns = ['month', 'consumption']
monthly_consumption['month'] = pd.to_datetime(monthly_consumption['month'])
# Format month as "Jan 2025", etc.
monthly_consumption['month_label'] = monthly_consumption['month'].dt.strftime('%b %
```

- Plot

In [53]:
```
monthly_consumption = df[['january_consumption', 'february_consumption', 'march_con
monthly_consumption.index = ['2025-01', '2025-02', '2025-03', '2025-04', '2025-05']
monthly_consumption = monthly_consumption.reset_index()
monthly_consumption.columns = ['month', 'consumption']
monthly_consumption['month'] = pd.to_datetime(monthly_consumption['month'])
```

```
# Format month as "Jan 2025", etc.
monthly_consumption['month_label'] = monthly_consumption['month'].dt.strftime('%b %
```

In [54]:
```
import matplotlib.pyplot as plt

plt.plot(monthly_consumption['month_label'], monthly_consumption['consumption'], ma
plt.title("Monthly Total Consumption")
plt.xlabel("Month")
plt.ylabel("Consumption (m³)")
plt.grid(True)
plt.xticks(rotation=45)       # Rotate for better visibility
plt.tight_layout()            # Prevent label cutoff
plt.show()
```



Random Forest Modeling with Class Weight Adjustment

In [55]:
```
rf_balanced = RandomForestClassifier(
    n_estimators=100,
    random_state=42,
    class_weight='balanced'
)
rf_balanced.fit(X_train, y_train)
```

Out[55]:

```
▾ RandomForestClassifier
RandomForestClassifier(class_weight='balanced', random_state=42)
```

In [56]:

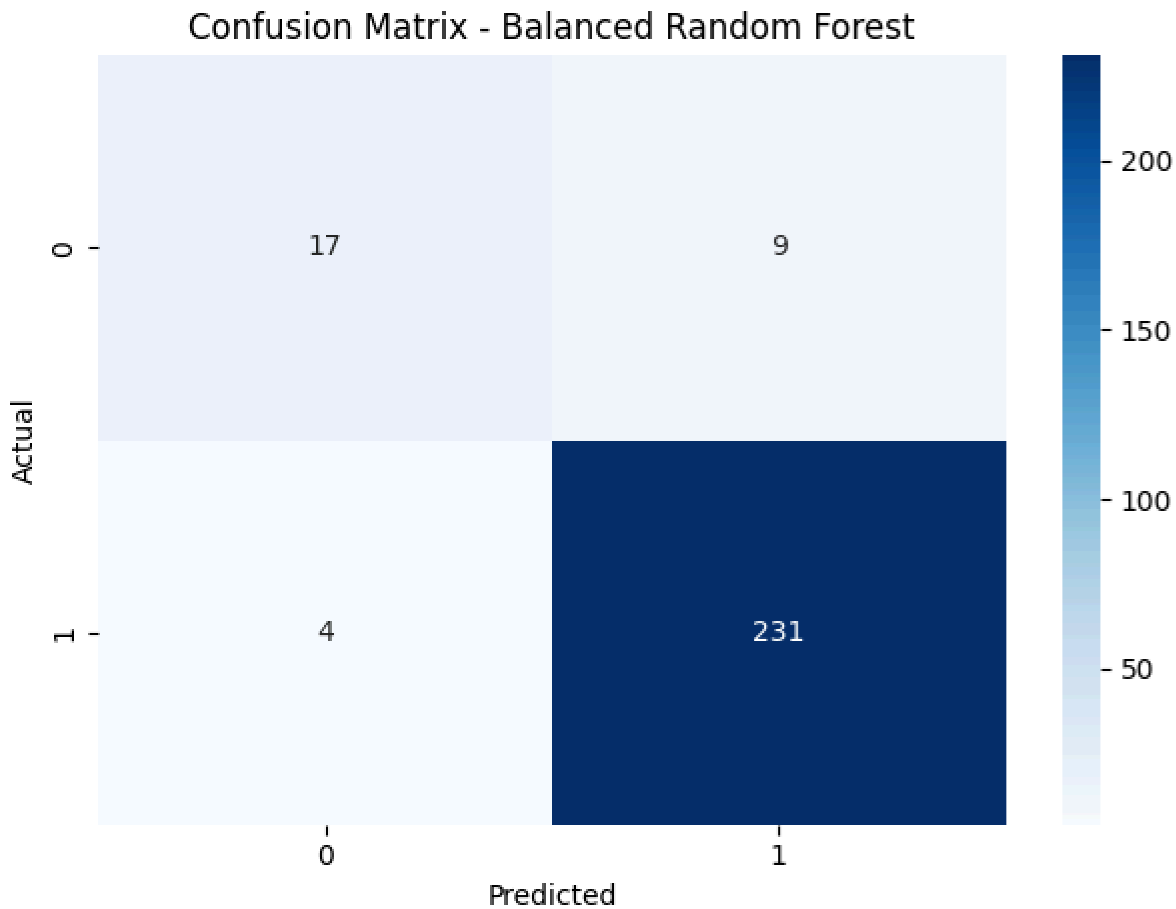
```python
y_pred = rf_balanced.predict(X_test)
```

- ## Evaluate the Model

In [57]:

```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Classification report
print("🔍 Balanced Random Forest Classification Report:\n")
print(classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix - Balanced Random Forest")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

```
🔍 Balanced Random Forest Classification Report:

              precision    recall  f1-score   support

           0       0.81      0.65      0.72        26
           1       0.96      0.98      0.97       235

    accuracy                           0.95       261
   macro avg       0.89      0.82      0.85       261
weighted avg       0.95      0.95      0.95       261
```



In [58]:
```
# Check Feature Importance (Optional but Insightful)

import pandas as pd

# Get feature importances
feature_importance = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': rf_balanced.feature_importances_
}).sort_values(by='Importance', ascending=False)

# Display
print("🌟 Feature Importance:")
print(feature_importance)
```

```
🌟 Feature Importance:
           Feature  Importance
0  avg_consumption    0.451514
2       meter_type    0.362034
6             line    0.056704
1   telco_category    0.053829
3           status    0.039148
5             zone    0.020542
4    customer_type    0.016228
```

In [59]:
```
import joblib
joblib.dump(rf_balanced, "rf_payment_compliance_model.pkl")
```

Out[59]: ['rf_payment_compliance_model.pkl']

```
In [ ]:
```

```
In [ ]:
```