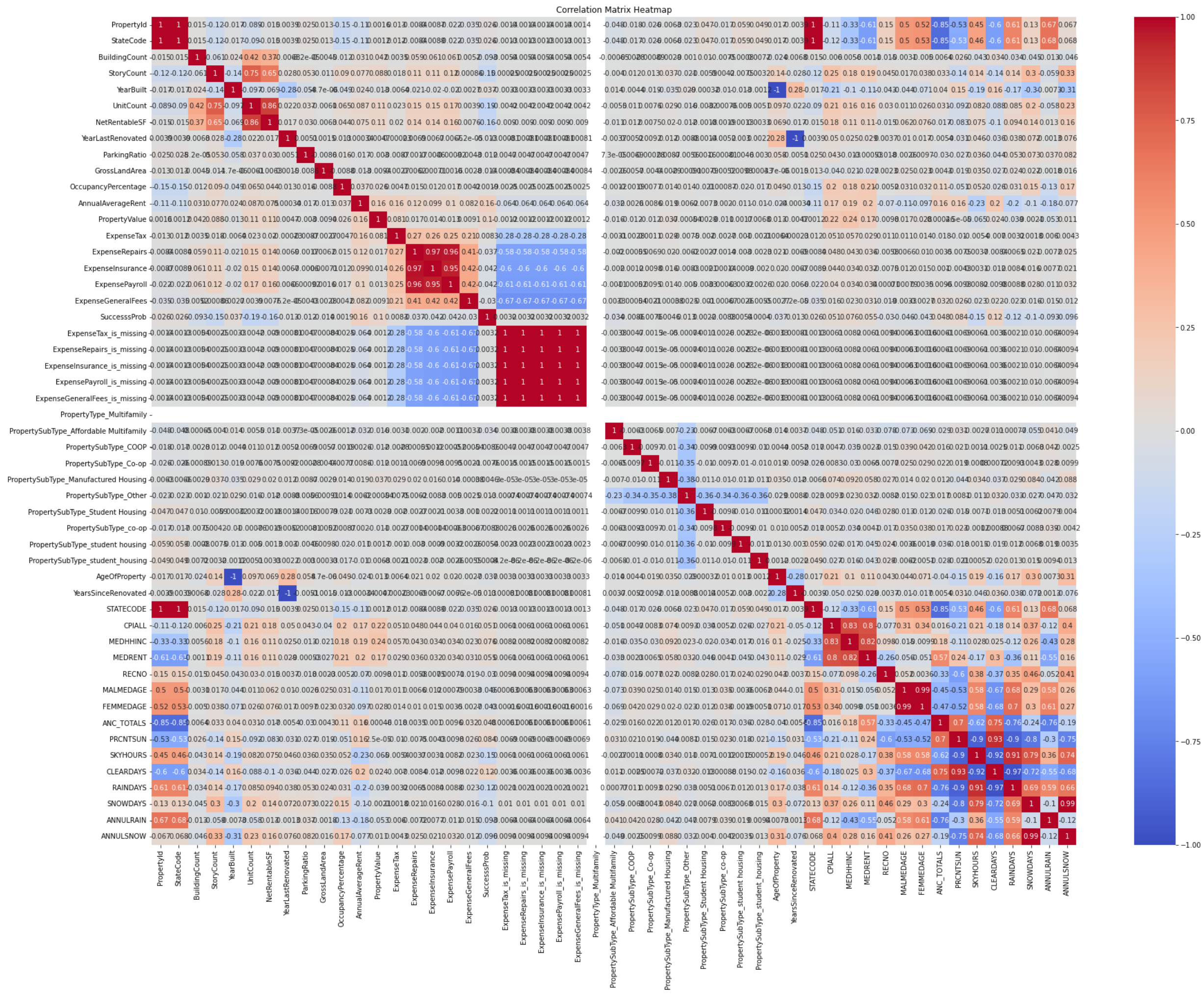
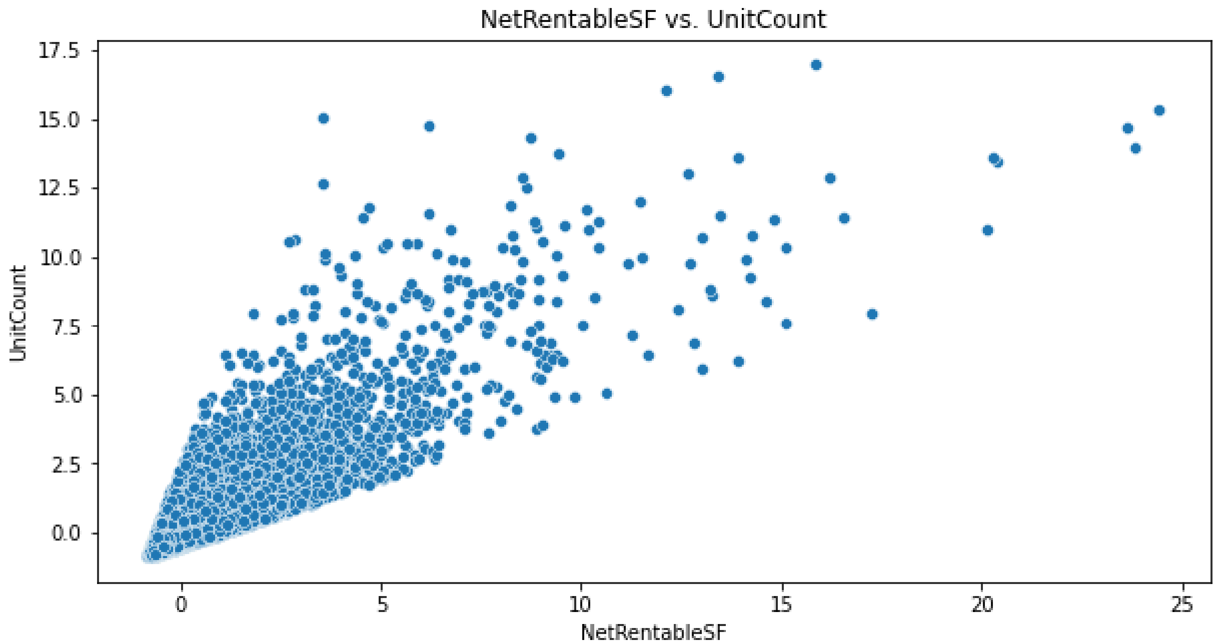
```python
plt.title('Correlation Matrix Heatmap')
plt.show()

```



In [238]:

```python
plt.figure(figsize=(10, 5))
sns.scatterplot(x='NetRentableSF', y='UnitCount', data=df)
plt.title('NetRentableSF vs. UnitCount')
plt.show()

```



In [239]:

```python
#Model building
from sklearn.model_selection import train_test_split

# Define your features and target variable
X = df.drop('SuccesssProb', axis=1)  # assuming 'SuccesssProb' is your targe
y = df['SuccesssProb']

# Convert 'SuccesssProb' into a binary variable
y = (df['SuccesssProb'] > 0.5).astype(int)

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Create a logistic regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)


```

Out[239]: LogisticRegression()

In [240]:
```python
from sklearn.linear_model import LogisticRegression

# Create a logistic regression model
model_lr = LogisticRegression()

# Train the model on the training data
model_lr.fit(X_train, y_train)

```

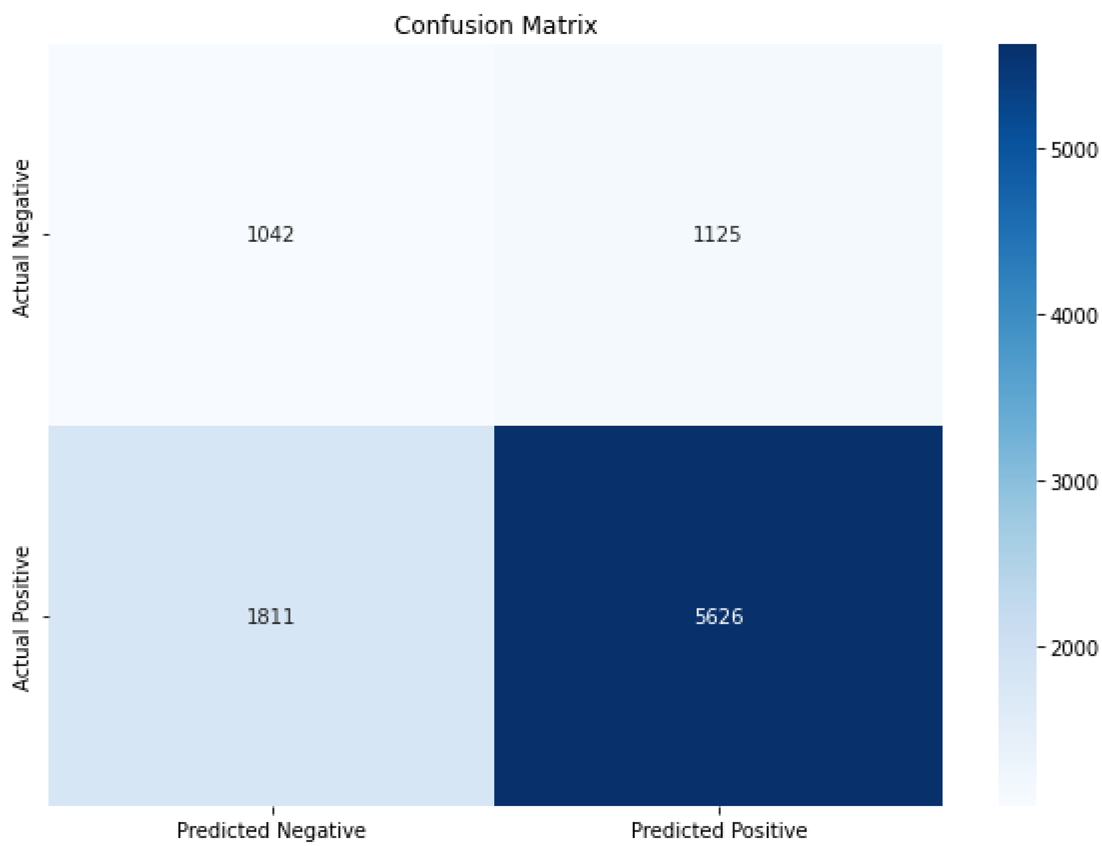
Out[240]: LogisticRegression()

In [241]:
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f

# Make predictions on the test set
y_pred = model_lr.predict(X_test)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

# Print metrics
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
print(f'ROC AUC Score: {roc_auc}')

```

```
Accuracy: 0.7747813411078717
Precision: 0.7751460767946577
Recall: 0.9989242974317601
F1 Score: 0.8729216849773809
ROC AUC Score: 0.5022309535151417
```

In [242]:

```python
from sklearn.metrics import confusion_matrix

# Make probability predictions on the test set
y_prob = model.predict_proba(X_test)[:, 1]

# Classify properties as successful or unsuccessful based on the optimal dec
y_pred = (y_prob > min_threshold).astype(int)

# Create a confusion matrix
cm  = confusion_matrix(y_test, y_pred)

# Create a DataFrame from the confusion matrix for better visualization
cm_df = pd.DataFrame(cm, columns=['Predicted Negative', 'Predicted Positive'

plt.figure(figsize=(10, 7))

sns.heatmap(cm_df, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.show()

```



In [243]:

```python
#cost benifit analysis
import numpy as np

# Define the costs and benefits
benefit = 1000000
cost = 3000000

# Calculate the probabilities of the positive class
y_prob = model.predict_proba(X_test)[:, 1]

# Calculate the expected cost for different decision thresholds
thresholds = np.linspace(0, 1, 100)
expected_costs = []
for threshold in thresholds:
    y_pred = (y_prob > threshold).astype(int)
    fp = np.sum((y_test == 0) & (y_pred == 1))
    tp = np.sum((y_test == 1) & (y_pred == 1))
    expected_cost = fp * cost - tp * benefit
    expected_costs.append(expected_cost)

# Find the threshold with the minimum expected cost
min_cost = min(expected_costs)
min_threshold = thresholds[expected_costs.index(min_cost)]

print(f'Minimum Expected Cost: {min_cost}')
print(f'Optimal Decision Threshold: {min_threshold}')

```

```
Minimum Expected Cost: -2251000000
Optimal Decision Threshold: 0.7575757575757577
```

In [244]:

```python
#A negative value indicates that the benefits outweigh the costs.
```

The threshold value of approximately 0.76. is the optimal decision threshold for classifying a property as successful or unsuccessful, based on minimizing the expected cost.

This means that if the model predicts a success probability of greater than 0.76 for a property, we should classify it as successful; otherwise, you should classify it as unsuccessful.

Generally, the accuracy is 50%, but based on our conditions of 1:3 ratio of profit to loss, we ended up at 0.76 as threshold

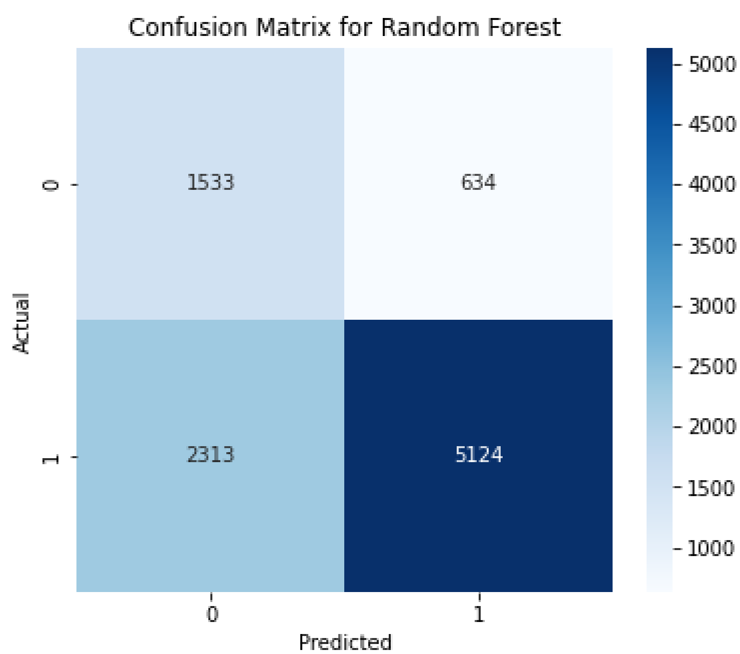
In [245]:

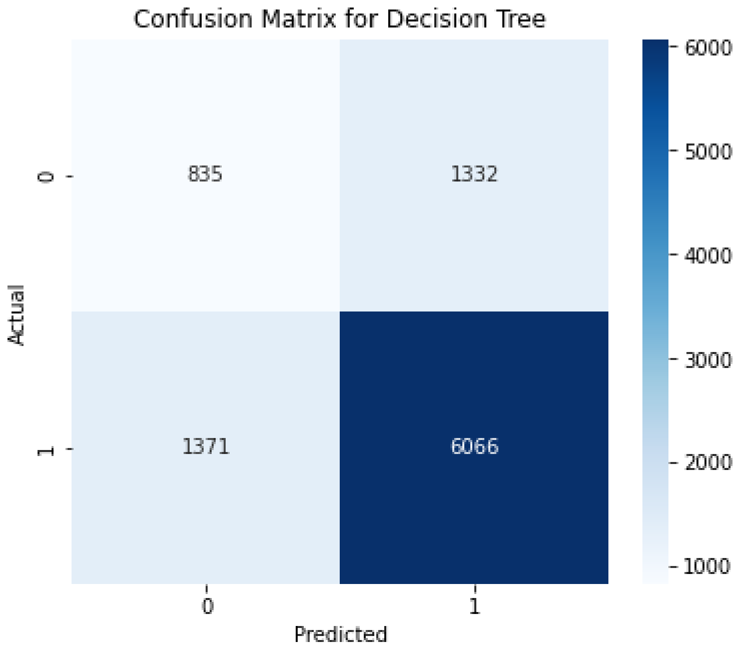
```
from sklearn.ensemble import RandomForestClassifier

# Initialize the model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions
y_pred_rf = rf_model.predict(X_test)


from sklearn.tree import DecisionTreeClassifier

# Initialize the model
dt_model = DecisionTreeClassifier(random_state=42)

# Train the model
dt_model.fit(X_train, y_train)

# Make predictions
y_pred_dt = dt_model.predict(X_test)


import xgboost as xgb

# Initialize the model
xgb_model = xgb.XGBClassifier(random_state=42)

# Train the model
xgb_model.fit(X_train, y_train)

# Make predictions
y_pred_xgb = xgb_model.predict(X_test)

```

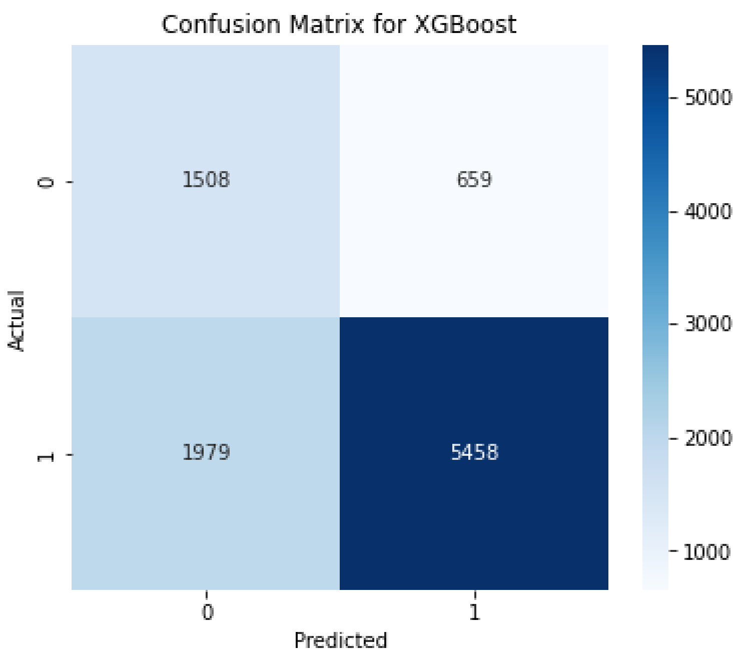
```
[14:49:57] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defa
ult evaluation metric used with the objective 'binary:logistic' was changed fro
m 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
```

In [246]:

```python
from sklearn.metrics import confusion_matrix

# Make probability predictions on the test set
y_prob_rf = rf_model.predict_proba(X_test)[:, 1]
y_prob_dt = dt_model.predict_proba(X_test)[:, 1]
y_prob_xgb = xgb_model.predict_proba(X_test)[:, 1]

# Classify properties as successful or unsuccessful based on the optimal dec
y_pred_rf = (y_prob_rf > min_threshold).astype(int)
y_pred_dt = (y_prob_dt > min_threshold).astype(int)
y_pred_xgb = (y_prob_xgb > min_threshold).astype(int)

# Create confusion matrices
cm_rf = confusion_matrix(y_test, y_pred_rf)
cm_dt = confusion_matrix(y_test, y_pred_dt)
cm_xgb = confusion_matrix(y_test, y_pred_xgb)

import seaborn as sns
import matplotlib.pyplot as plt

# Create a list of models and their confusion matrices
models = ['Random Forest', 'Decision Tree', 'XGBoost']
cms = [cm_rf, cm_dt, cm_xgb]

# Plot the confusion matrix for each model
for model, cm in zip(models, cms):
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix for {model}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

```




Confusion Matrix for Decision Tree
835
1332
1371
6066
Actual
Predicted
0
1
6000
5000
4000
3000
2000
1000


Confusion Matrix for XGBoost
1508
659
1979
5458
Actual
Predicted
0
1
5000
4000
3000
2000
1000

In [247]:

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score



# Calculate metrics for Random Forest
tn_rf, fp_rf, fn_rf, tp_rf = cm_rf.ravel()
specificity_rf = tn_rf / (tn_rf + fp_rf)

# Calculate metrics for Decision Tree
tn_dt, fp_dt, fn_dt, tp_dt = cm_dt.ravel()
specificity_dt = tn_dt / (tn_dt + fp_dt)

# Calculate metrics for XGBoost
tn_xgb, fp_xgb, fn_xgb, tp_xgb = cm_xgb.ravel()
specificity_xgb = tn_xgb / (tn_xgb + fp_xgb)



# Calculate metrics for Random Forest
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)

# Calculate metrics for Decision Tree
accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt)
recall_dt = recall_score(y_test, y_pred_dt)

# Calculate metrics for XGBoost
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
precision_xgb = precision_score(y_test, y_pred_xgb)
recall_xgb = recall_score(y_test, y_pred_xgb)



# Print the metrics for each model
print('\nMetrics for Random Forest:')
print(f'Accuracy: {accuracy_rf:.4f}')
print(f'Precision: {precision_rf:.4f}')
print(f'Recall (Sensitivity): {recall_rf:.4f}')
print(f'Specificity: {specificity_rf:.4f}')

print('\nMetrics for Decision Tree:')
print(f'Accuracy: {accuracy_dt:.4f}')
print(f'Precision: {precision_dt:.4f}')
print(f'Recall (Sensitivity): {recall_dt:.4f}')
print(f'Specificity: {specificity_dt:.4f}')

print('\nMetrics for XGBoost:')
print(f'Accuracy: {accuracy_xgb:.4f}')
print(f'Precision: {precision_xgb:.4f}')
print(f'Recall (Sensitivity): {recall_xgb:.4f}')
print(f'Specificity: {specificity_xgb:.4f}')
```

```
Metrics for Random Forest:
Accuracy: 0.6931
Precision: 0.8899
Recall (Sensitivity): 0.6890
Specificity: 0.7074

Metrics for Decision Tree:
Accuracy: 0.7186
Precision: 0.8200
Recall (Sensitivity): 0.8157
Specificity: 0.3853

Metrics for XGBoost:
Accuracy: 0.7253
Precision: 0.8923
Recall (Sensitivity): 0.7339
Specificity: 0.6959
```

In [248]:

```
model_rf = rf_model
model_dt = dt_model
model_xgb = xgb_model
```

In [249]:

```python
#Probability thresholds for various models:

import numpy as np

# Define the costs and benefits
benefit = 1000000
cost = 3000000

# Calculate the probabilities of the positive class for each model

y_prob_rf = model_rf.predict_proba(X_test)[:, 1]
y_prob_dt = model_dt.predict_proba(X_test)[:, 1]
y_prob_xgb = model_xgb.predict_proba(X_test)[:, 1]

# Calculate the expected cost for different decision thresholds for each mod
thresholds = np.linspace(0, 1, 100)



# Random Forest
expected_costs_rf = []
for threshold in thresholds:
    y_pred_rf = (y_prob_rf > threshold).astype(int)
    fp_rf = np.sum((y_test == 0) & (y_pred_rf == 1))
    tp_rf = np.sum((y_test == 1) & (y_pred_rf == 1))
    expected_cost_rf = fp_rf * cost - tp_rf * benefit
    expected_costs_rf.append(expected_cost_rf)

# Decision Tree
expected_costs_dt = []
for threshold in thresholds:
    y_pred_dt = (y_prob_dt > threshold).astype(int)
    fp_dt = np.sum((y_test == 0) & (y_pred_dt == 1))
    tp_dt = np.sum((y_test == 1) & (y_pred_dt == 1))
    expected_cost_dt = fp_dt * cost - tp_dt * benefit
    expected_costs_dt.append(expected_cost_dt)

# XGBoost
expected_costs_xgb = []
for threshold in thresholds:
    y_pred_xgb = (y_prob_xgb > threshold).astype(int)
    fp_xgb = np.sum((y_test == 0) & (y_pred_xgb == 1))
    tp_xgb = np.sum((y_test == 1) & (y_pred_xgb == 1))
    expected_cost_xgb = fp_xgb * cost - tp_xgb * benefit
    expected_costs_xgb.append(expected_cost_xgb)

# Find the threshold with the minimum expected cost for each model


min_cost_rf = min(expected_costs_rf)
min_threshold_rf = thresholds[expected_costs_rf.index(min_cost_rf)]

min_cost_dt = min(expected_costs_dt)
min_threshold_dt = thresholds[expected_costs_dt.index(min_cost_dt)]

min_cost_xgb = min(expected_costs_xgb)
min_threshold_xgb = thresholds[expected_costs_xgb.index(min_cost_xgb)]
```

```python

# Print the minimum expected costs and optimal decision thresholds for each 

print('Random Forest:')
print(f'Minimum Expected Cost: {min_cost_rf}')
print(f'Optimal Decision Threshold: {min_threshold_rf}\n')

print('Decision Tree:')
print(f'Minimum Expected Cost: {min_cost_dt}')
print(f'Optimal Decision Threshold: {min_threshold_dt}\n')

print('XGBoost:')
print(f'Minimum Expected Cost: {min_cost_xgb}')
print(f'Optimal Decision Threshold: {min_threshold_xgb}\n')


```

```
Random Forest:
Minimum Expected Cost: -3229000000
Optimal Decision Threshold: 0.7474747474747475

Decision Tree:
Minimum Expected Cost: -2070000000
Optimal Decision Threshold: 0.0

XGBoost:
Minimum Expected Cost: -3509000000
Optimal Decision Threshold: 0.7777777777777778
```

Conclusion

In [253]:

```python
import pandas as pd

# Create a dictionary with the model metrics
model_metrics = {
    'Model': ['Logistic Regression', 'Random Forest', 'Decision Tree', 'XGBo
    'Accuracy': [accuracy, accuracy_rf, accuracy_dt, accuracy_xgb],
    'Optimal Threshold': [min_threshold, min_threshold_rf, min_threshold_dt,
}

# Create a DataFrame from the dictionary
metrics_table = pd.DataFrame(model_metrics)

# Print the table
print(metrics_table)

```

```
                 Model  Accuracy  Optimal Threshold
0  Logistic Regression  0.774781           0.757576
1        Random Forest  0.693149           0.747475
2        Decision Tree  0.718555           0.000000
3              XGBoost  0.725323           0.777778
```

Based on the EDA and the modeling contingent on the cost benigit contriants provided in the Question

Here are the best models:

Logistic Regression: 77% accuracy, threshold is 75%

XGboost: 72% accuracy , threshold is 77%

In [ ]:
```
 
```

In [ ]:
```
 
```