Phase-Three-Repository / Untitled.ipynb

Kemboi-koder Recommendations and Limitations bd2d5a4 · 1 hour ago

1284 lines (1284 loc) · 364 KB

Preview Code Blame Raw

Credit Card Default Classification Project

📝 Problem Statement

Lifeline Creditors is a company that has faced numerous losses due to credit card customers defaulting on their payments during the last financial year.In order to cut losses, the company has opted to reduce the quantity of high risk loans. You are tasked with developing a machine learning algorithm tha clasiffies credit card borrowers to find the ones that are most likely to default on payment the next month. This project aims to build a machine learning model to classify whether a customer will default on their credit card payment next month using UCI Loan Data a case sstudy of Taiwan. This prediction can guide risk assessment and help banks minimize financial risk.

Stakeholders:

- Credit Risk Managers
- Loan Approval Officers
- Data Analysts
- Executives of Lifeline Creditors

Step 1: Load and Explore Dataset

We start by importing the dataset and checking the structure. This helps us understand what features are available and whether any data cleaning is required. Import the relevant python libraries

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv("UCI_Credit_Card.csv")
df.head()
```

Out[2]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4
0	1	20000.0	2	2	1	24	2	2	-1	-1
1	2	120000.0	2	2	2	26	-1	2	0	0
2	3	90000.0	2	2	2	34	0	0	0	0
3	4	50000.0	2	2	1	37	0	0	0	0
4	5	50000.0	1	2	1	57	-1	0	-1	0

5 rows × 25 columns

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   ID                          30000 non-null  int64  
 1   LIMIT_BAL                   30000 non-null  float64
 2   SEX                         30000 non-null  int64  
 3   EDUCATION                   30000 non-null  int64  
 4   MARRIAGE                    30000 non-null  int64  
 5   AGE                         30000 non-null  int64  
 6   PAY_0                       30000 non-null  int64  
 7   PAY_2                       30000 non-null  int64  
 8   PAY_3                       30000 non-null  int64  
 9   PAY_4                       30000 non-null  int64  
 10  PAY_5                       30000 non-null  int64  
 11  PAY_6                       30000 non-null  int64  
 12  BILL_AMT1                   30000 non-null  float64
 13  BILL_AMT2                   30000 non-null  float64
 14  BILL_AMT3                   30000 non-null  float64
 15  BILL_AMT4                   30000 non-null  float64
 16  BILL_AMT5                   30000 non-null  float64
 17  BILL_AMT6                   30000 non-null  float64
 18  PAY_AMT1                    30000 non-null  float64
 19  PAY_AMT2                    30000 non-null  float64
 20  PAY_AMT3                    30000 non-null  float64
 21  PAY_AMT4                    30000 non-null  float64
 22  PAY_AMT5                    30000 non-null  float64
 23  PAY_AMT6                    30000 non-null  float64
 24  default.payment.next.month  30000 non-null  int64  
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
ID                          0
LIMIT_BAL                   0
SEX                         0
EDUCATION                   0
MARRIAGE                    0
AGE                         0
PAY_0                       0
PAY_2                       0
PAY_3                       0
PAY_4                       0
PAY_5                       0
PAY_6                       0
BILL_AMT1                   0
BILL_AMT2                   0
BILL_AMT3                   0
BILL_AMT4                   0
BILL_AMT5                   0
BILL_AMT6                   0
```

```
PAY_AMT1                    0
PAY_AMT2                    0
PAY_AMT3                    0
PAY_AMT4                    0
PAY_AMT5                    0
PAY_AMT6                    0
default.payment.next.month  0
dtype: int64
```

Step 2: Find relevant columns

Identify and rename the target column ('default.payment.next.month') for clarity and visualize the balance between defaulters and non-defaulters. This helps assess class imbalance, which can affect model performance.

In [5]:
```
print(df['default.payment.next.month'].value_counts())
```

```
0    23364
1     6636
Name: default.payment.next.month, dtype: int64
```

In [6]:
```
df.rename(columns={'default.payment.next.month': 'defaulted'}, inplace=True)
df.head()
```

Out[6]:

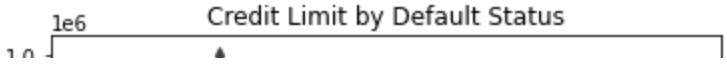
	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4
0	1	20000.0	2	2	1	24	2	2	-1	-1
1	2	120000.0	2	2	2	26	-1	2	0	0
2	3	90000.0	2	2	2	34	0	0	0	0
3	4	50000.0	2	2	1	37	0	0	0	0
4	5	50000.0	1	2	1	57	-1	0	-1	0

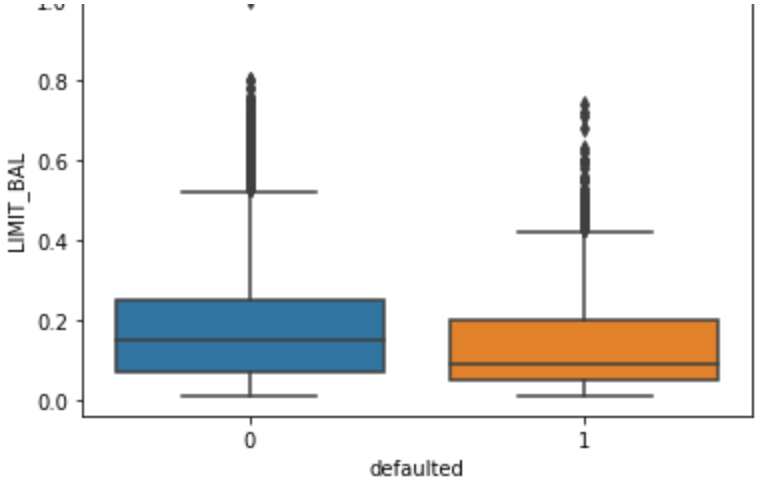
5 rows × 25 columns

Step 3: Exploratory Data Analysis

Visualize the relationships between important numeric variables (`LIMIT_BAL` and `AGE`) and the target. This helps detect patterns or distributions that may be predictive. We do this to check if the class is imbalanced

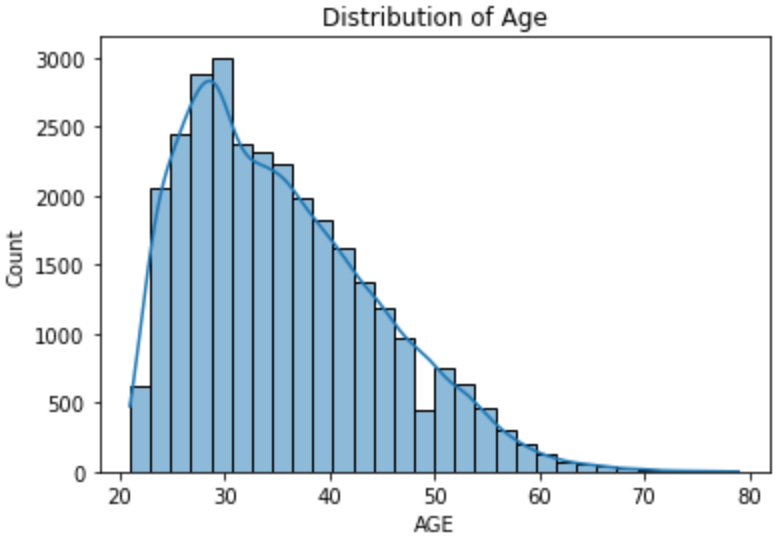
In [7]:
```
# Credit limit vs default
sns.boxplot(data=df, x='defaulted', y='LIMIT_BAL')
plt.title('Credit Limit by Default Status')
plt.show()
```





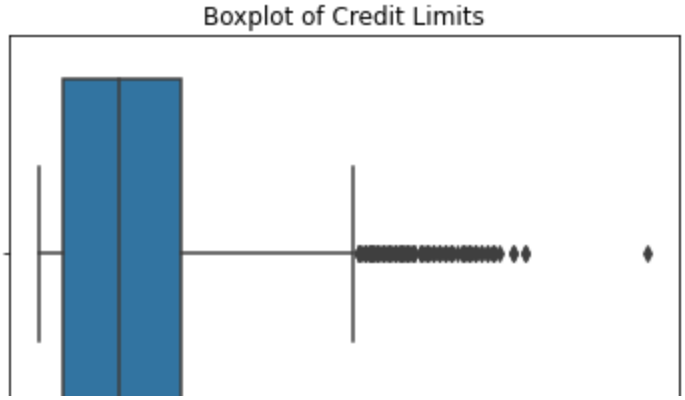
In [8]:

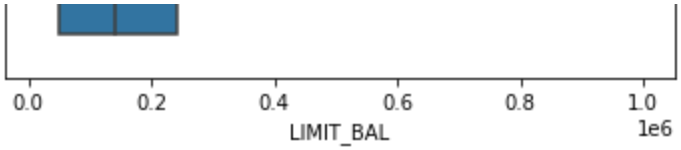
```
# Age distribution
sns.histplot(df['AGE'], kde=True, bins=30)
plt.title('Distribution of Age')
plt.show()
```



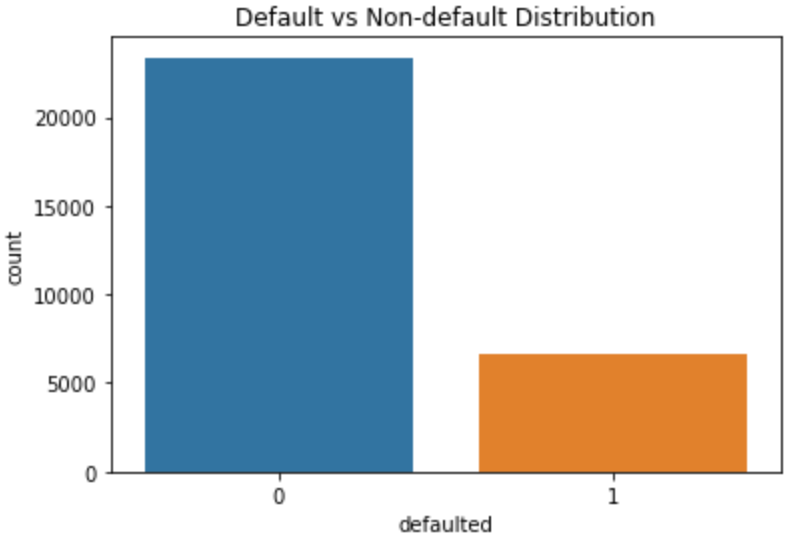
In [9]:

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

sns.boxplot(x=df['LIMIT_BAL'])
plt.title("Boxplot of Credit Limits")
plt.show()
```





In [10]:
```
#Visualize the "defaulted" class
sns.countplot(data=df, x='defaulted')
plt.title('Default vs Non-default Distribution')
plt.show()
```



Step 4: Fix the class imbalance

The "defaulted" class seems to be imbalaced. We initialize SMOTE which will create synthetic data points in the minority class in order to oversample it. We import SMOTE from imblearn and imprt the train_test_split sklearn. We then train the model on the synthetic data points.

In [11]:
```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
X = df.drop("defaulted", axis=1)
y = df["defaulted"]
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_siz

smote = SMOTE(random_state=42)
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)

from collections import Counter
print("Original:", Counter(y_train))
print("After SMOTE:", Counter(y_train_sm))
```

```
Original: Counter({0: 18691, 1: 5309})
After SMOTE: Counter({0: 18691, 1: 18691})
```

The class imbalance has been resolved by oversampling the minority class.

Step 5: Data Cleaning

After doing EDA we have concluded that the data does not have null values and duplicates. Therefore, we remove unnecessary columns ('ID'). We also prepare for encoding categorical variables in the next step.

In [12]:
```
# Drop irrelevant columns
df.drop(columns=['ID'], inplace=True)
```

🔡 Step 6: Encode Categorical Variables

Convert the categorical columns (SEX, EDUCATION, MARRIAGE) into dummy variables using one-hot encoding then drop the first category to prevent multicollinearity. This prepares the model for log regression.

In [13]:
```
#One hot encode categorical columns
cat_cols = ['SEX', 'EDUCATION', 'MARRIAGE']
df = pd.get_dummies(df, columns=cat_cols, drop_first=True)
```

Step 7: Logistic Regression Model

Now that all our columns are in numeric form and class imbalance ha sbeen resolved we perform a logistic regression. Fit the trained data onto a logistic regression model to predict whether a customer will default. This model serves as a strong and interpretable baseline. We import use **sklearn** to perform the regression

In [14]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train_sm, y_train_sm)
y_pred = logreg.predict(X_test)
y_probs = logreg.predict_proba(X_test)[:, 1]
print("Classification Report:\n")
print(classification_report(y_test, y_pred))
```

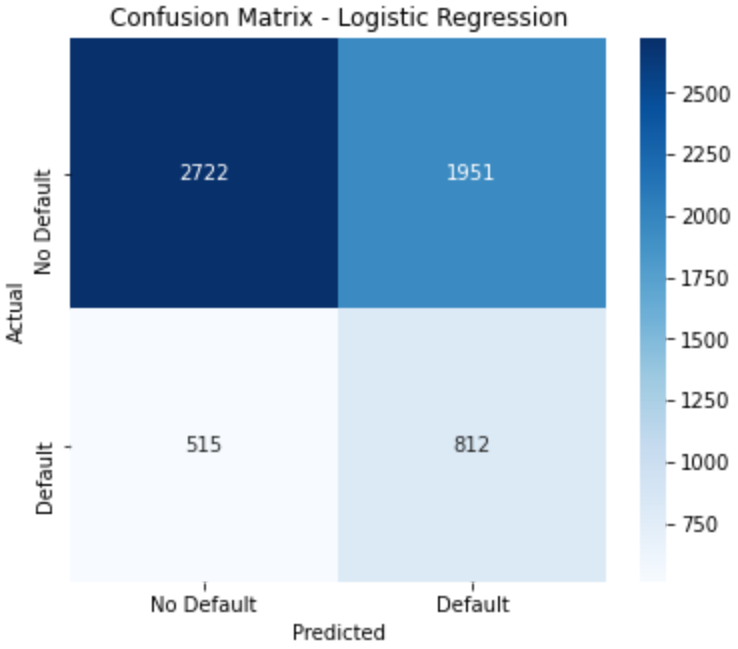
```
Classification Report:

              precision    recall  f1-score   support

           0       0.84      0.58      0.69      4673
           1       0.29      0.61      0.40      1327

    accuracy                           0.59      6000
   macro avg       0.57      0.60      0.54      6000
weighted avg       0.72      0.59      0.62      6000
```

Visualizations

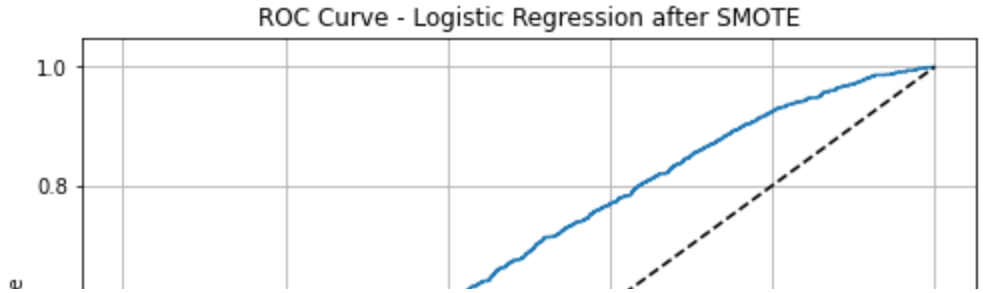
From the logistic regression we can create a confusion matrix and AUC-ROC curves to

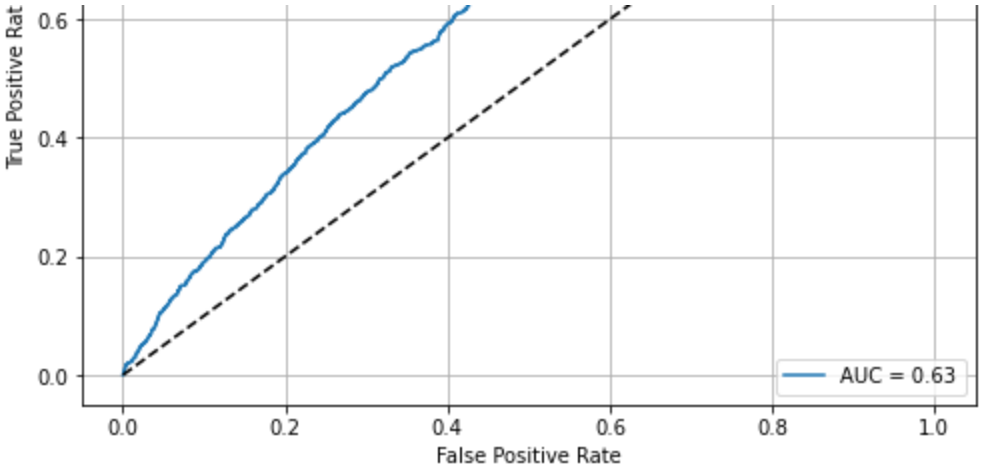
help us understand the model better.

In [15]:
```
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', xticklabels=['No Defa
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```



In [16]:
```
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve

fpr, tpr, _ = roc_curve(y_test, y_probs)
auc_score = roc_auc_score(y_test, y_probs)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'AUC = {auc_score:.2f}')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression after SMOTE')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```





Summary

-The AUC (area under the curve) is 0.63. This shows that there is some predictive power beyond 0.5 (**the dotted line**) that indicates random guessing. -The model is better at detecting defaulters (recall = 61%) than at correctly predicting them (precision = 29%). -Overal recall of the minority class after SMOTE is good but there's still a high false positive rate. This aligns with our goals but a recall of 0.61 is still low.

Step 7: Change the decision threshold

In order to get a better balance between the prcision and recall, we need to tune the prediction threshold from the default of 0.5 to one that better suites our objectives. We first calculate the different Precision and Recall for different prediction thresholds then determine the sweet spot where we are not favoring either precision or recall too heavily.

In [17]:

```
import numpy as np
from sklearn.metrics import precision_score, recall_score, f1_score

y_probs = logreg.predict_proba(X_test)[:, 1]
thresholds = np.arange(0.1, 0.9, 0.05)
results = []
for thresh in thresholds:
    y_pred_thresh = (y_probs >= thresh).astype(int)
    precision = precision_score(y_test, y_pred_thresh)
    recall = recall_score(y_test, y_pred_thresh)
    f1 = f1_score(y_test, y_pred_thresh)
    results.append((thresh, precision, recall, f1))

results_df = pd.DataFrame(results, columns=["Threshold", "Precision", "Recall"
print(results_df)
```

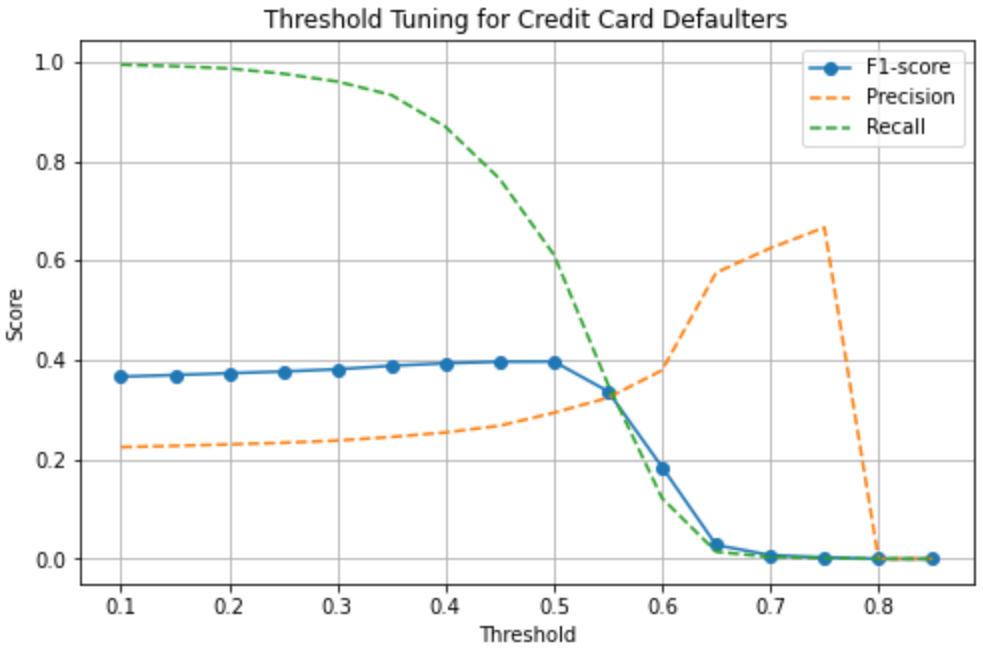
```
   Threshold  Precision    Recall        F1
0       0.10   0.224778  0.993971  0.366644
1       0.15   0.227351  0.990957  0.369850
2       0.20   0.230336  0.986436  0.373466
3       0.25   0.233544  0.975885  0.376892
```

```
3        0.25   0.233544  0.973883  0.376892
4        0.30   0.237997  0.960060  0.381437
5        0.35   0.245246  0.932931  0.388392
6        0.40   0.254301  0.868877  0.393448
7        0.45   0.267829  0.764130  0.396636
8        0.50   0.293883  0.611907  0.397066
9        0.55   0.324061  0.351168  0.337071
10       0.60   0.379391  0.122080  0.184721
11       0.65   0.575758  0.014318  0.027941
12       0.70   0.625000  0.003768  0.007491
13       0.75   0.666667  0.001507  0.003008
14       0.80   0.000000  0.000000  0.000000
15       0.85   0.000000  0.000000  0.000000
```

```
C:\Users\User\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [18]:

```
# Plot F1 vs threshold
plt.figure(figsize=(8, 5))
plt.plot(results_df["Threshold"], results_df["F1"], marker='o', label='F1-scor
plt.plot(results_df["Threshold"], results_df["Precision"], linestyle='--', lab
plt.plot(results_df["Threshold"], results_df["Recall"], linestyle='--', label=
plt.xlabel("Threshold")
plt.ylabel("Score")
plt.title("Threshold Tuning for Credit Card Defaulters")
plt.legend()
plt.grid(True)
plt.show()
```



In [19]:

```
optimal_threshold = 0.37
y_pred_final = (y_probs >= optimal_threshold).astype(int)
print(classification_report(y_test, y_pred_final))
```

```
              precision    recall  f1-score   support
```

```
              0       0.90      0.22      0.35      4673
              1       0.25      0.91      0.39      1327

       accuracy                           0.37      6000
      macro avg       0.57      0.56      0.37      6000
   weighted avg       0.75      0.37      0.36      6000
```

-Recall for defaulters (class 1) jumped from 61% to 91%. This means that the algorithim will flag more defaulters than the previous algorithm at the cost of false alarms.

-Precision and accuracy becomes unrellaible. This trade-off is acceptable since our main goal is to get as many defaulters as possible.

Step 8: Vizualizations and Evaluations

✔ Decision Tree with Tuned Threshold

We will now apply a Decision Tree classifier using the optimal threshold identified from our threshold tuning plot. We will retrain the model, apply the threshold, and evaluate performance.

Decision tree anlysis

-This decision is trained on credit card default data from the UCI Credit Card Default dataset, with 29,000 samples split between defaulters (11,669) and non-defaulters (17,124).

MARRIAGE (≤1.5): The root split suggests marital status is the strongest predictor

PAY Variables: Payment history features dominate the tree structure (PAY_0, PAY_2: Recent payment behavior)

EDUCATION (≤2.5): Educational attainment influences risk. Graduate school (1), University (2), High school(3),Others (4)

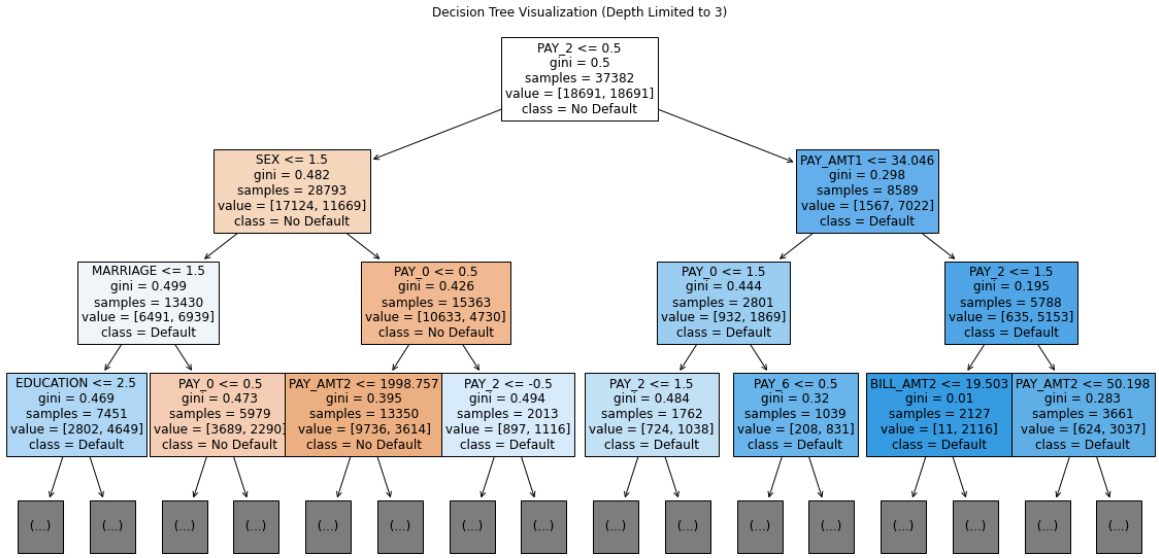
BILL_AMT2 & PAY_AMT2: Financial capacity indicators. Bill amounts show spending patterns

The tree highlights that recent repayment history (`PAY_*` variables) and demographic features (like gender or marital status) influence the likelihood of default.

In [20]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Train Decision Tree model
dt = DecisionTreeClassifier(random_state=42, max_depth=5)
dt.fit(X_train_sm, y_train_sm)
y_probs_dt = dt.predict_proba(X_test)[:, 1]
```

```
optimal_threshold = 0.37
y_pred_dt_thresh = (y_probs_dt >= optimal_threshold).astype(int)

plt.figure(figsize=(20, 10))
plot_tree(dt, feature_names=X.columns, fontsize=12, class_names=["No Default",
plt.title("Decision Tree Visualization (Depth Limited to 3)")
plt.show()
```



✔ ROC Curve

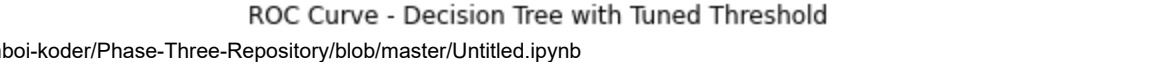
The ROC (Receiver Operating Characteristic) curve evaluates the model's ability to distinguish between classes at various threshold levels.

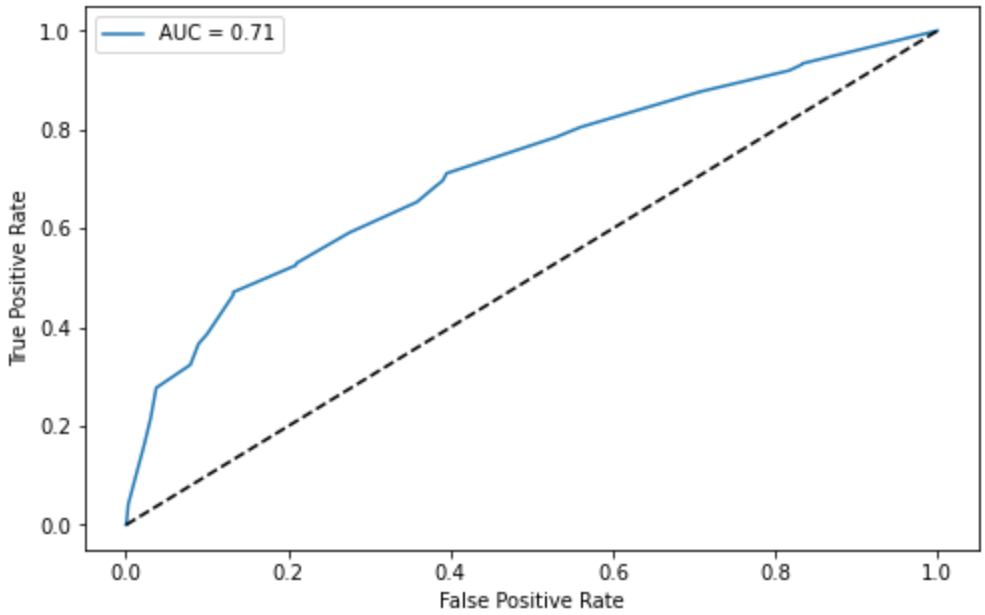
- **X-axis**: False Positive Rate (FPR).
- **Y-axis**: True Positive Rate (TPR), also known as recall.
- **AUC (Area Under Curve)**: 0.71

An AUC of 0.71 suggests the model has a **fair** ability to discriminate between default and non-default than a random classifier would score 0.5, while a perfect model would score 1.0.

In [23]:

```
# Plot ROC Curve
from sklearn.metrics import roc_auc_score
roc_auc_dt = roc_auc_score(y_test, y_probs_dt)
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_probs_dt)
plt.figure(figsize=(8,5))
plt.plot(fpr_dt, tpr_dt, label=f'AUC = {roc_auc_dt:.2f}')
plt.plot([0,1], [0,1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree with Tuned Threshold')
plt.legend()
plt.show()
```





✔ Confusion Matrix

Interpretation:

- **True Positives (TP)** = 907 → Correctly predicted defaults.
- **True Negatives (TN)** = 2896 → Correctly predicted non-defaults.
- **False Positives (FP)** = 1777 → Non-defaults incorrectly predicted as defaults.
- **False Negatives (FN)** = 420 → Defaults missed by the model.

This matrix reflects a balanced trade-off:

- The model captures **a large portion of defaults** (good recall).
- However, there is still a **notable rate of false positives**, which could result in wrongly flagged customers.
- Aligns with our goal of **risk mitigation**, where catching defaults is more critical than some over-warning.

In [ ]:
```
# Confusion Matrix
cm_dt = confusion_matrix(y_test, y_pred_dt_thresh)
plt.figure(figsize=(6, 5))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Default', 'Default'],
            yticklabels=['No Default', 'Default'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Decision Tree with Threshold')
plt.show()
```

In [ ]:
```
# Classification Report
print("\nClassification Report:\n", classification_report(y_test, y_pred_dt_th
```

In [ ]:

```
# ROC AUC
roc_auc_dt = roc_auc_score(y_test, y_probs_dt)
print("ROC AUC Score:", roc_auc_dt)
```

🔎Summary

-The AUC (**area under the curve**) is **0.71** which is an improvement from **0.63** in the previous model meaning that this model is more accurate at a threshold of **0.37** from the previous default threshold of **0.5**

-The model is better at detecting defaulters (recall = 68%) form the previous which was (recall=61%).

-Precision has inrceased to **34%** from **29%** in the previous model which means that this new model is better at correctly predicting deafaulters(**True Positives**).

-Overal recall of the minority class after threshold tuning has increased as well as the precision meaning the **false positive rate has reduced**.

-In conclusion the new model is better at detecting and predicting possible defaulters while reducing the number of non-defaulters flagged as compared to the previous model.