

Loan Approval Data 2025

Steps:

1. Import Necessary Libraries
2. Data Loading
3. Data Preprocessing
4. Exploratory Data Analysis & Correlation
5. Model Training
6. Model Evaluation
7. Hyperparameter Tuning
8. Feature Importance Analysis

Dataset Link: <https://www.kaggle.com/datasets/parthpatel2130/realistic-loan-approval-dataset-us-and-canada>

➤ Step 1: Import Necessary Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier,
GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

➤ Step 2: Data Loading

```
import pandas as pd

# 1. Load the dataset into a Pandas DataFrame from the CSV file
df = pd.read_csv('Loan_approval_data_2025.csv')

# 2. Display the first 5 rows to inspect the structure and column values
print("Dataset Head (First 5 Rows):")
display(df.head())

# 3. Check the basic dataset structure, total rows/columns, and data types
print("\nDataset Information & Summary:")
df.info()
```

Dataset Head (First 5 Rows):

	customer_id	age	occupation_status	years_employed	annual_income	credit_score	cre
0	CUST100000 692	40	Employed	17.2	25579		
1	CUST100001	33	Employed	7.3	43087	627	
2	CUST100002	42	Student	1.1	20840	689	
3	CUST100003	53	Student	0.5	29147	692	
4	CUST100004	32	Employed	12.5	63657	630	

Dataset Information & Summary:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 50000 entries, 0 to 49999 Data

columns (total 20 columns):

#	Column	Non-Null Count	Dtype	-
--	-----	-----	-----	0
	customer_id	50000 non-null	object	
1	age	50000 non-null	int64	
2	occupation_status	50000 non-null	object	
3	years_employed	50000 non-null	float64	
4	annual_income	50000 non-null	int64	
5	credit_score	50000 non-null	int64	
6	credit_history_years	50000 non-null	float64	
7	savings_assets	50000 non-null	int64	
8	current_debt	50000 non-null	int64	
9	defaults_on_file	50000 non-null	int64	
10	delinquencies_last_2yrs	50000 non-null	int64	
11	derogatory_marks	50000 non-null	int64	
12	product_type	50000 non-null	object	
13	loan_intent	50000 non-null	object	
14	loan_amount	50000 non-null	int64	
15	interest_rate	50000 non-null	float64	
16	debt_to_income_ratio	50000 non-null	float64	

```
17 loan_to_income_ratio      50000 non-null float64
18 payment_to_income_ratio  50000 non-null float64 19
   loan_status                50000 non-null int64
dtypes: float64(6), int64(10), object(4) memory
usage: 7.6+ MB
```

➤ Step 3: Data Preprocessing

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# 1. Drop the customer_id column as it is a unique identifier and not a useful feature for modeling
if 'customer_id' in df.columns:
    df = df.drop(columns=['customer_id'])

# 2. Identify categorical text columns and encode them into numeric format using LabelEncoder
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
print("Categorical columns encoded:", categorical_cols)

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])

# 3. Verify missing values across the dataset
print("\nTotal missing values after preprocessing:", df.isnull().sum().sum())

# 4. Display the preprocessed data dimensions and first rows
print("\nPreprocessed Dataset Shape:", df.shape)
display(df.head())
```

Categorical columns encoded: ['occupation_status', 'product_type', 'loan_intent']

Total missing values after preprocessing: 0

Preprocessed Dataset Shape: (50000, 19)

	age	occupation_status	years_employed	annual_income	credit_score	credit_history_ye
0	40	0	17.2	25579	692	
1	33	0	7.3	43087	627	
2	42	2	1.1	20840	689	
3	53	2	0.5	29147	692	
4	32	0	12.5	63657	630	

➤ Step 4: Exploratory Data Analysis & Correlation

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

# 1. Check the distribution of the target variable (loan_status) in percentages print("Loan
Status Value Counts (Proportion):")
print(df['loan_status'].value_counts(normalize=True) * 100)

# 2. Visualize the distribution of Loan Status (Fixed to avoid FutureWarning)
plt.figure(figsize=(6, 4)) sns.countplot(x='loan_status', data=df, hue='loan_status',
palette='Set2') plt.title('Distribution of Loan Status (0: Rejected, 1: Approved)')
plt.xlabel('Loan Status') plt.ylabel('Count') plt.legend([], [], frameon=False) # لإخفاء مفتاح
الألوان المكرر plt.show()

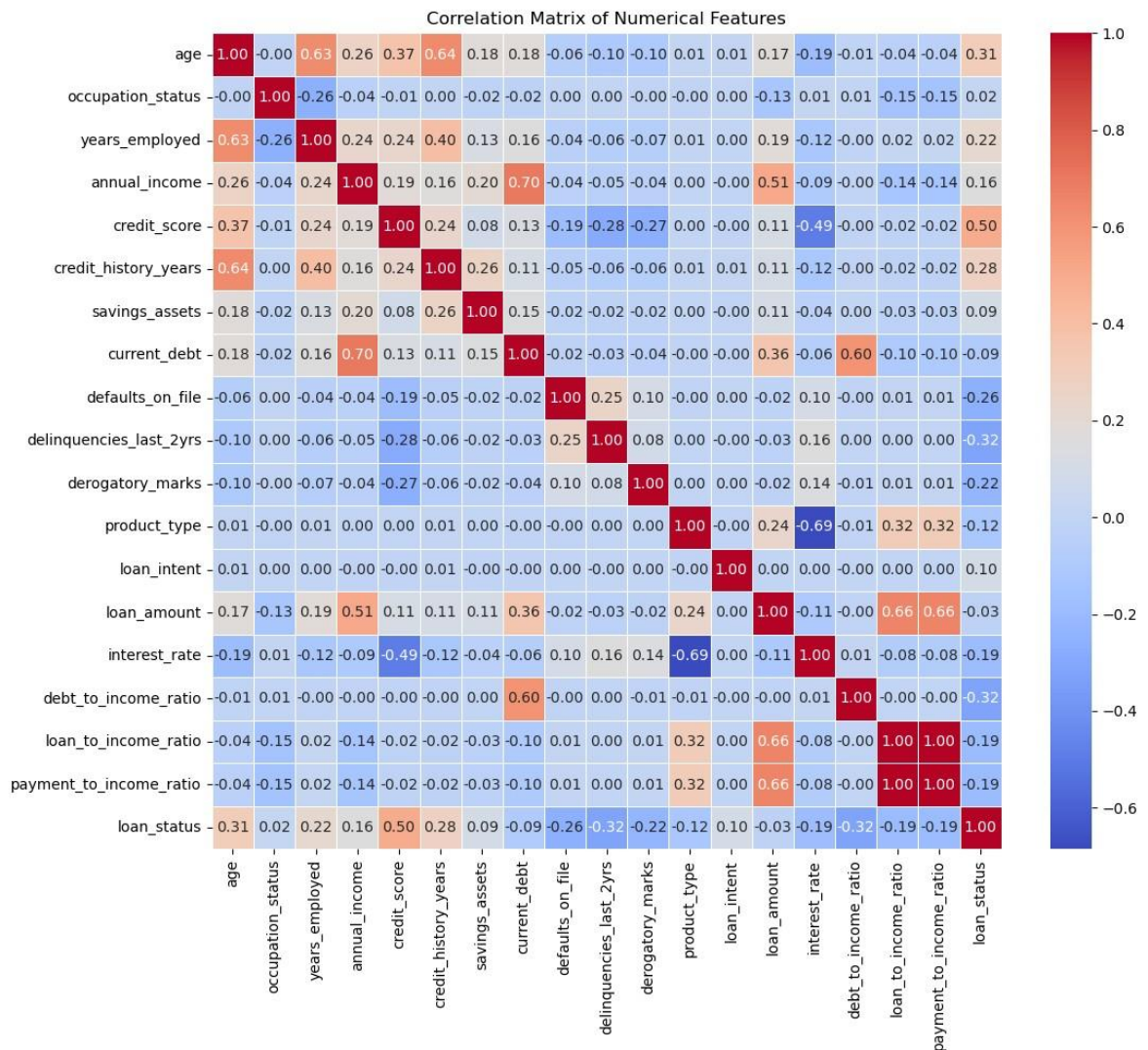
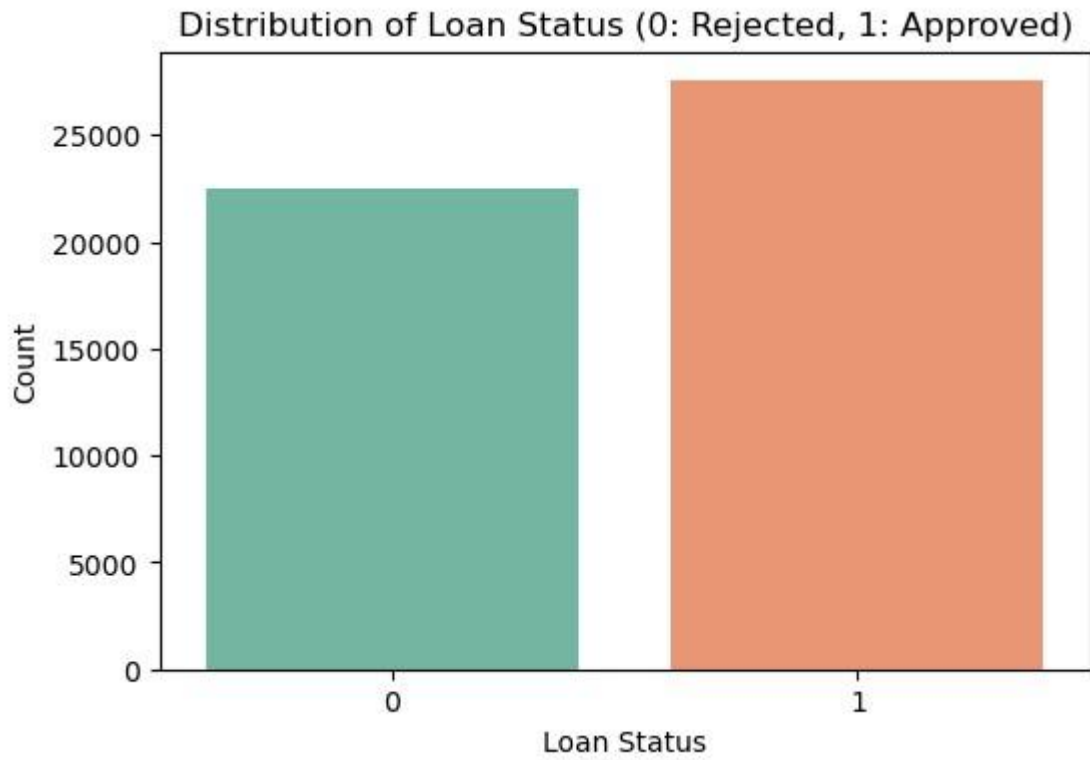
# 3. Compute correlation matrix for numerical features
plt.figure(figsize=(12, 10)) numeric_df =
df.select_dtypes(include=['int64', 'float64']) correlation_matrix =
numeric_df.corr()

# 4. Draw a heatmap for the correlation matrix
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm', linewidth
plt.title('Correlation Matrix of Numerical Features') plt.show()

# 5. Check correlation with the target variable (loan_status) print("\nCorrelation with Loan
Status:") print(correlation_matrix['loan_status'].sort_values(ascending=False))
```

```
Loan Status Value Counts (Proportion):
loan_status
1      55.046
```

```
0    44.954
Name: proportion, dtype: float64
```



```
Correlation with Loan Status:
loan_status      1.000000
credit_score      0.496316
age              0.311524
credit_history_years 0.277103
years_employed   0.218875
annual_income    0.158466
loan_intent      0.097146
savings_assets   0.094789
occupation_status 0.017855
loan_amount     -0.025987
current_debt     -0.088494
product_type     -0.119347
interest_rate    -0.185179
loan_to_income_ratio -0.193252
payment_to_income_ratio -0.193258
derogatory_marks -0.224735
defaults_on_file -0.263033
debt_to_income_ratio -0.317363
delinquencies_last_2yrs -0.318075
Name: loan_status, dtype: float64
```

➤ Step 5: Model Training

```

from sklearn.model_selection import train_test_split from sklearn.preprocessing import
StandardScaler from sklearn.linear_model import LogisticRegression from sklearn.neighbors import
KNeighborsClassifier from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier,
Gradient from sklearn.tree import DecisionTreeClassifier from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# 1. Ensure DataFrame exists and split into features and target if 'df' in
locals():
    X = df.drop(columns=['loan_status'])    y =
df['loan_status']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran

# 2. Scale features    scaler =
StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)    X_test_scaled =
scaler.transform(X_test)

# 3. Define models dictionary    models = {
    "Logistic Regression": LogisticRegression(max_iter=5000, random_state=42
    "K-Nearest Neighbors (KNN)": KNeighborsClassifier(n_neighbors=5),
    "AdaBoost": AdaBoostClassifier(random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42, n_estimators=10
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "Support Vector Machine (SVM)": SVC(random_state=42)    }

# 4. Train models and evaluate accuracies
print("--- Starting Model Training and Evaluation ---")    results = {}

```

```

for name, model in models.items():
    print(f"Training model: {name}...")
    if name in ["Logistic Regression", "K-Nearest
Neighbors (KNN)", "Support
model.predict(X_test_scaled)
else:
    model.fit(X_train, y_train)
    y_pred =
model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"-> Completed! {name} Accuracy: {acc:.4f}\n")

print("=====")
print("=====")
key=lambda x: x[1], reverse=True):
    print(f"{name}: {acc:.4f}")
else:
    print("Error: DataFrame (df) is not defined.")

```

--- Starting Model Training and Evaluation --Training

model: Logistic Regression...

-> Completed! Logistic Regression Accuracy: 0.8466

Training model: K-Nearest Neighbors (KNN)... -> Completed!

K-Nearest Neighbors (KNN) Accuracy: 0.8598

Training model: AdaBoost...

-> Completed! AdaBoost Accuracy: 0.8875

Training model: Decision Tree...

-> Completed! Decision Tree Accuracy: 0.8737

Training model: Random Forest...

-> Completed! Random Forest Accuracy: 0.9127

Training model: Gradient Boosting...

-> Completed! Gradient Boosting Accuracy: 0.9207

Training model: Support Vector Machine (SVM)... -> Completed!

Support Vector Machine (SVM) Accuracy: 0.8907

=====

Final Model Performance

=====

Gradient Boosting: 0.9207

Random Forest: 0.9127

Support Vector Machine (SVM): 0.8907

AdaBoost: 0.8875

Decision Tree: 0.8737

K-Nearest Neighbors (KNN): 0.8598

Logistic Regression: 0.8466

➤ Step 6: Model Evaluation

```

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

# 1. Detailed Evaluation for Each Trained Model print("--- Detailed Model
Evaluation ---")

for name, model in models.items():
    print(f"\n===== {name} =====")

    # Select the correct test set based on whether the model requires scaling    if name in
["Logistic Regression", "K-Nearest Neighbors (KNN)", "Support Vec        y_pred =
model.predict(X_test_scaled)    else:        y_pred = model.predict(X_test)

    # Print Classification Report (Precision, Recall, F1-Score, Support)
print("Classification Report:")    print(classification_report(y_test, y_pred))

    # Generate and plot Confusion Matrix    plt.figure(figsize=(5, 4))    cm =
confusion_matrix(y_test, y_pred)    sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues', cbar=False)    plt.title(f'Confusion Matrix - {name}')
plt.xlabel('Predicted Label')    plt.ylabel('True Label')    plt.show()

```

--- Detailed Model Evaluation ---

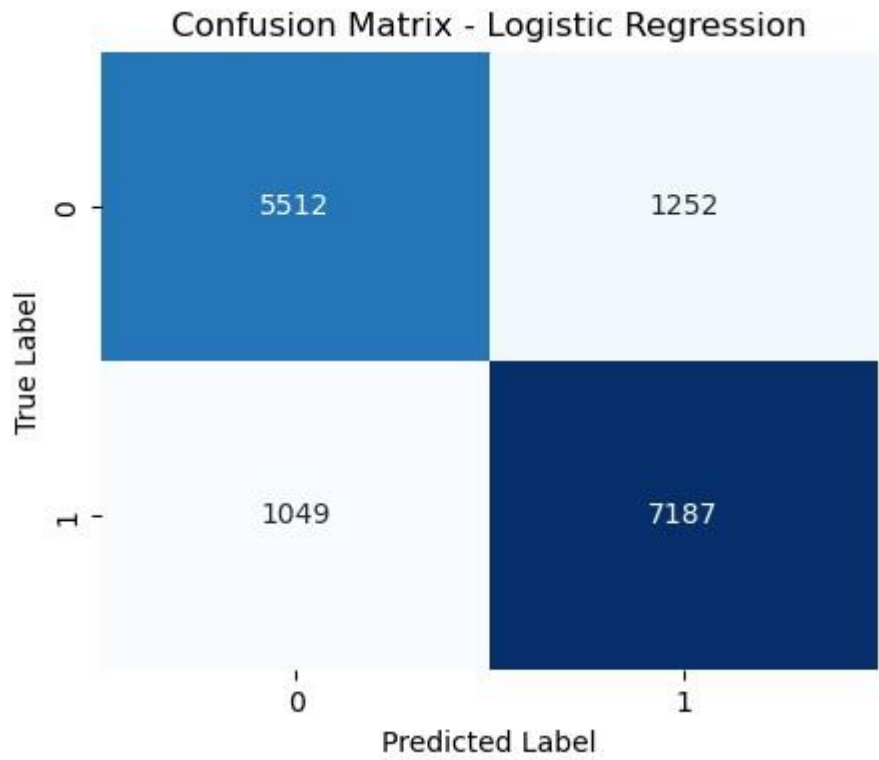
```

===== Logistic Regression =====
Classification Report:          precision    recall
f1-score   support

      0      0.84      0.81      0.83      6764
      1      0.85      0.87      0.86      8236

 accuracy          0.85      15000
 macro avg          0.85      0.84      0.84      15000
 weighted avg          0.85      0.85      0.85      15000

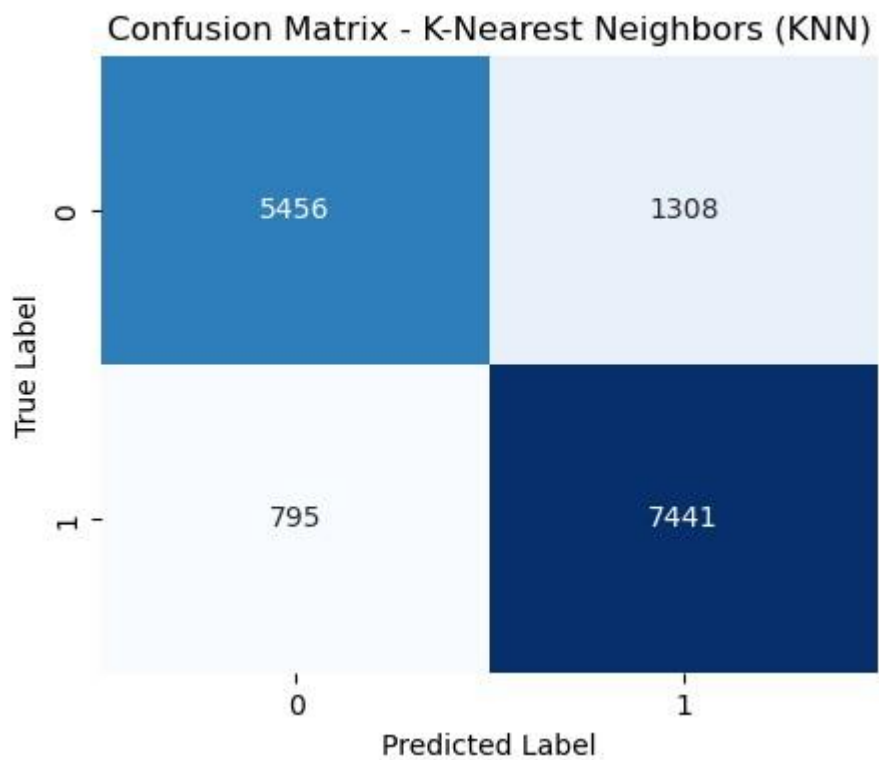
```



===== K-Nearest Neighbors (KNN) =====

Classification Report: precision recall
f1-score support

	0	0.87	0.81	0.84	6764
1	0.85	0.90	0.88		8236
accuracy				0.86	15000
macro avg	0.86	0.86	0.86	0.86	15000 weighted
avg	0.86	0.86	0.86	0.86	15000



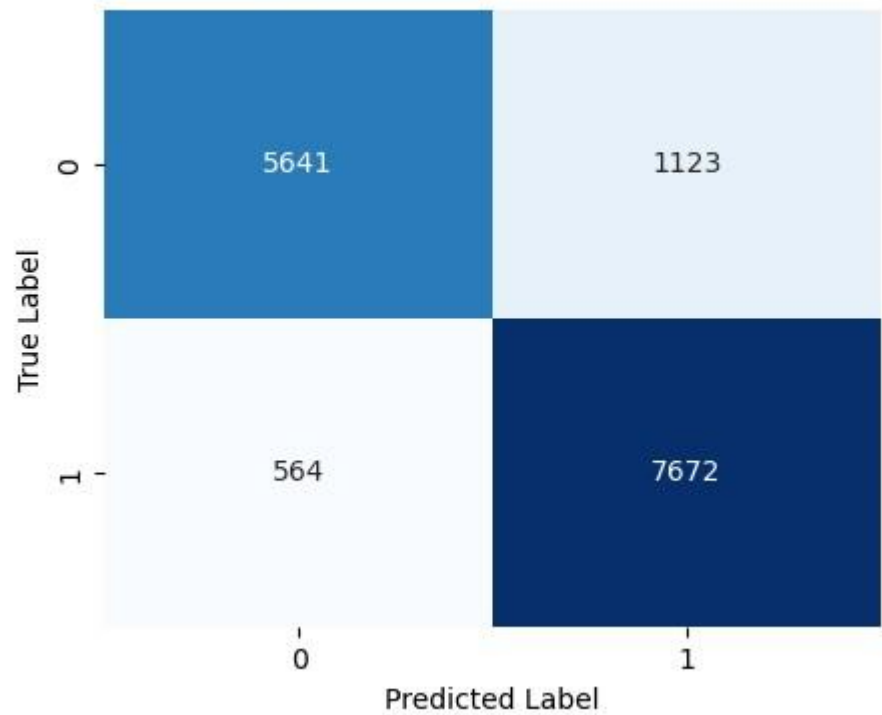
===== AdaBoost =====

Classification Report: precision recall
f1-score support

	0	0.91	0.83	0.87	6764
1	0.87	0.93	0.90		8236

accuracy			0.89	15000
macro avg	0.89	0.88	0.89	15000
weighted avg	0.89	0.89	0.89	15000

Confusion Matrix - AdaBoost

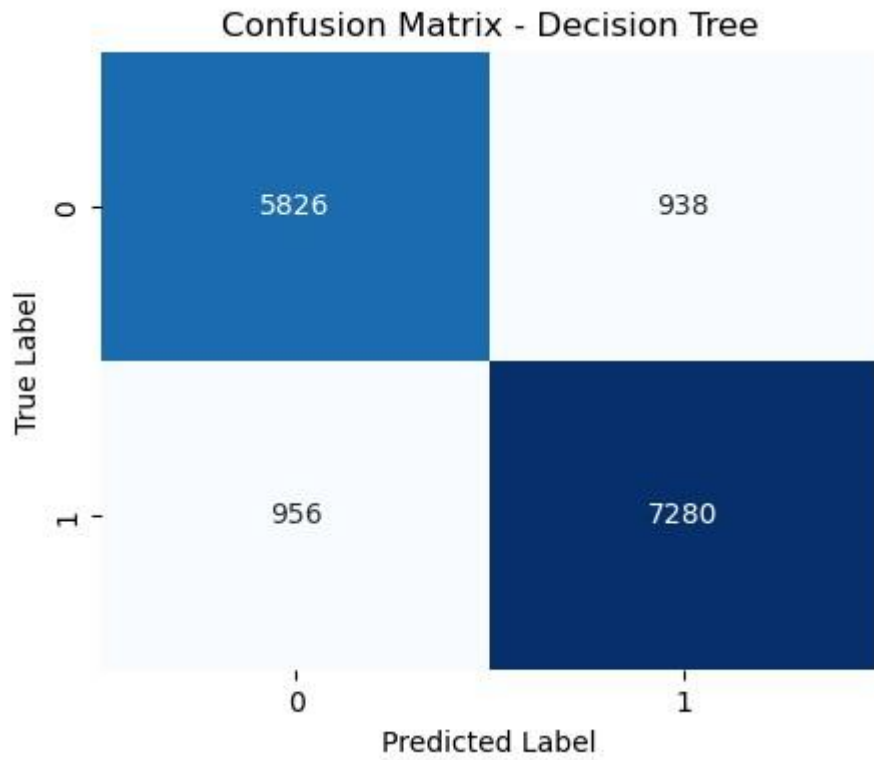


===== Decision Tree =====

Classification Report: precision recall
f1-score support

	0	0.86	0.86	0.86	6764
1	0.89	0.88	0.88		8236

accuracy			0.87	15000
macro avg	0.87	0.87	0.87	15000
weighted avg	0.87	0.87	0.87	15000

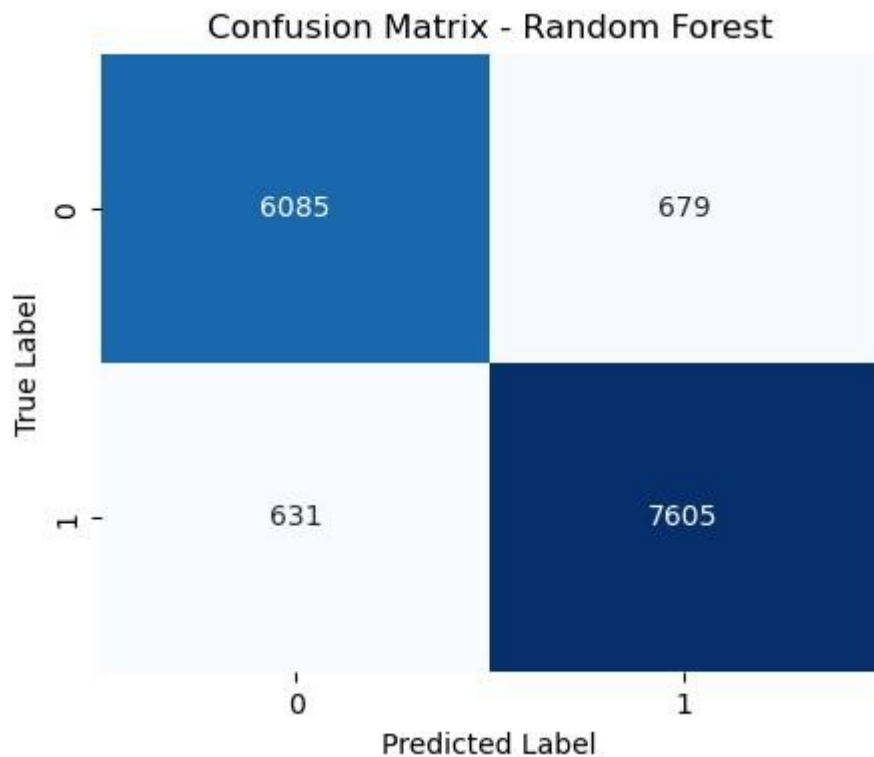


```

===== Random Forest =====
Classification Report:
              precision    recall  f1-score   support

     0       0.91      0.90      0.90      6764
     1       0.92      0.92      0.92      8236

 accuracy          0.91      15000
 macro avg         0.91      0.91      0.91      15000
 weighted avg      0.91      0.91      0.91      15000
  
```

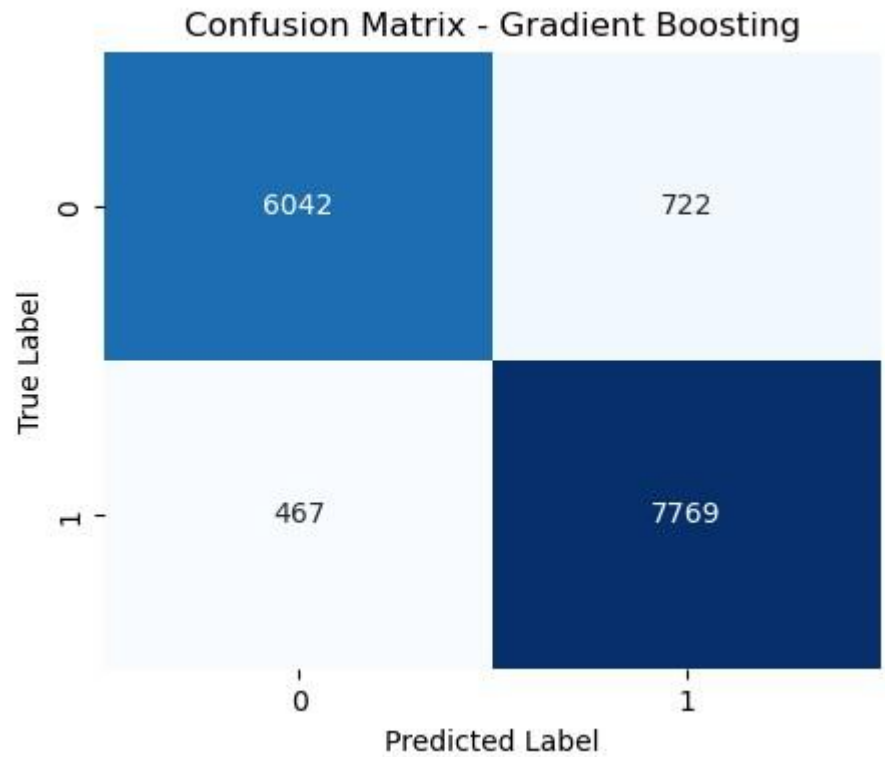


===== Gradient Boosting =====

Classification Report: precision recall
f1-score support

	0	0.93	0.89	0.91	6764
1	0.91	0.94	0.93		8236

accuracy			0.92	15000
macro avg	0.92	0.92	0.92	15000 weighted
avg	0.92	0.92	0.92	15000

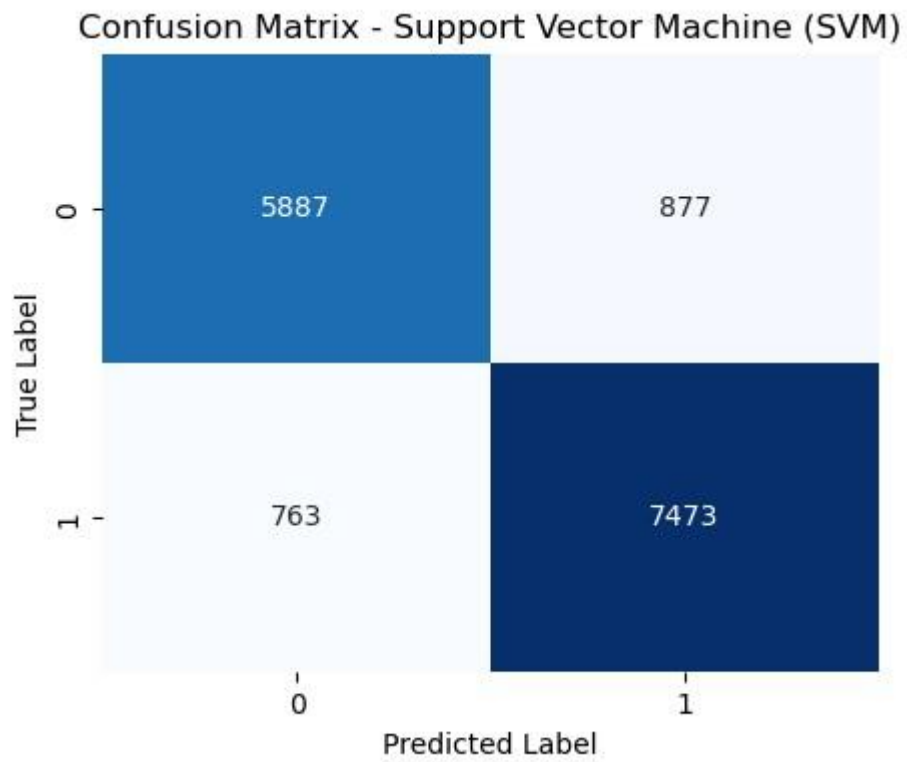


===== Support Vector Machine (SVM) =====

Classification Report: precision recall
f1-score support

	0	0.89	0.87	0.88	6764
1	0.89	0.91	0.90		8236

accuracy			0.89	15000
macro avg	0.89	0.89	0.89	15000 weighted
avg	0.89	0.89	0.89	15000



➤ Step 7: Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV from sklearn.ensemble
import RandomForestClassifier

# 1. Define the parameter grid param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5]
}

# 2. Initialize model and GridSearchCV with verbose=1 to show progress rf =
RandomForestClassifier(random_state=42) grid_search = GridSearchCV( estimator=rf,
param_grid=param_grid, cv=3, scoring='accuracy', n_jobs=-1, verbose=1 #
This prints fitting progress to the output
) print("--- Starting Hyperparameter Tuning (GridSearchCV)... Please wait ---")

# 3. Fit the model
grid_search.fit(X_train, y_train)

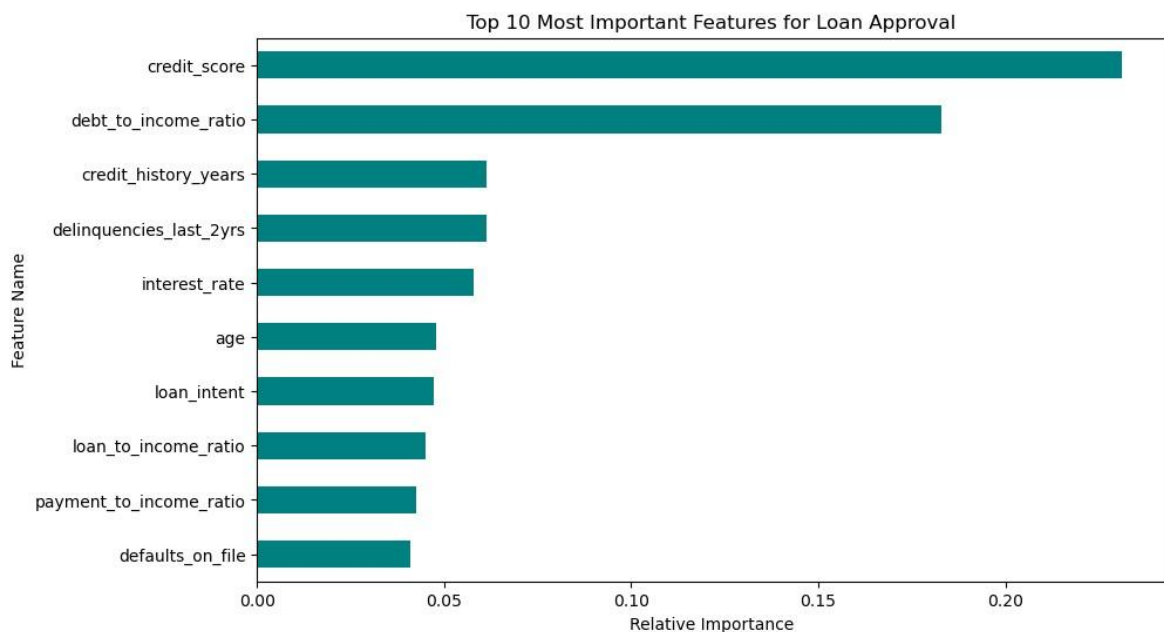
# 4. Print results
print("\n--- Tuning Completed Successfully! ---") print("Best Parameters Found:",
grid_search.best_params_) print(f"Best Accuracy After Tuning:
{grid_search.best_score_:.4f}")
```

```
--- Starting Hyperparameter Tuning (GridSearchCV)... Please wait --Fitting 3
folds for each of 18 candidates, totalling 54 fits

--- Tuning Completed Successfully! ---
Best Parameters Found: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 200} Best Accuracy After Tuning: 0.9063
```

➤ Step 8: Feature Importance Analysis

```
best_model = grid_search.best_estimator_  
feature_importances = pd.Series(best_model.feature_importances_, index=X.column  
  
plt.figure(figsize=(10, 6)) feature_importances.nlargest(10).plot(kind='barh',  
color='teal') plt.title('Top 10 Most Important Features for Loan Approval')  
plt.xlabel('Relative Importance') plt.ylabel('Feature Name')  
plt.gca().invert_yaxis() plt.show()
```



--- The End ---