Project Documentation: Loan Risk Classification System

NAME: PRANOY K H
COURSE: DATASCIENCE
EMAIL: khpranoy17@gmail.com

1. Introduction

This project utilizes a comprehensive dataset of loan applicants to develop a predictive model for credit risk. The dataset, loan_data.csv, contains 45,000 records featuring demographic information, financial history, and loan details. The primary objective is to automate the loan approval process by identifying high-risk applicants likely to default.

2. Aim

The key goal is to build a robust **Binary Classification System** that categorizes loan applications into two classes:

- **0 (Non-Default):** High probability of repayment.
- **1 (Default):** High risk of non-repayment.

3. Business Problem

Financial institutions face significant revenue loss when borrowers default on loans. Conversely, overly strict lending criteria can turn away profitable customers. By implementing a machine learning model, the bank can:

- **Minimize Risk:** Proactively identify risky profiles.
- **Efficiency:** Reduce manual review time for "clear-cut" applications.
- **Consistency:** Standardize the decision-making process based on historical data.

4. Project Workflow

The project follows a structured data science lifecycle to ensure reliable results:

1. **Import Libraries**: Importing libraries which are required for analyzing,cleaning,plotting and training

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

2. **Data Loading:** Importing the raw CSV data.

```
df = pd.read_csv(r'D:\Satascience\archive (1)\loan_data.csv')
```

3. **Cleaning:** Identifying and fixing errors or outliers.

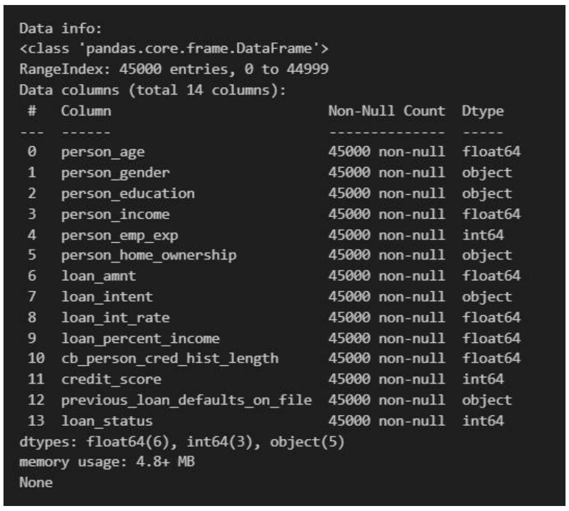
```
# Filter out unrealistic ages and employment experience
df = df[df['person_age'] <= 100]
df = df[df['person_emp_exp'] < df['person_age']]
```

4. **EDA:** Discovering patterns and correlations.
5. **Preprocessing:** Encoding and scaling features for model compatibility.
6. **Model Building:** Training multiple algorithms.
7. **Evaluation & Tuning:** Selecting the best model and optimizing its parameters.

5. Data Understanding

The dataset consists of **45,000 entries** and **14 columns**.

- **Numerical Features:** person_age, person_income, person_emp_exp, loan_amnt, loan_int_rate, loan_percent_income, cb_person_cred_hist_length, credit_score.
- **Categorical Features:** person_gender, person_education, person_home_ownership, loan_intent, previous_loan_defaults_on_file



```
Data info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45000 entries, 0 to 44999
Data columns (total 14 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   person_age                      45000 non-null  float64
 1   person_gender                   45000 non-null  object
 2   person_education                45000 non-null  object
 3   person_income                   45000 non-null  float64
 4   person_emp_exp                  45000 non-null  int64
 5   person_home_ownership           45000 non-null  object
 6   loan_amnt                       45000 non-null  float64
 7   loan_intent                     45000 non-null  object
 8   loan_int_rate                   45000 non-null  float64
 9   loan_percent_income             45000 non-null  float64
 10  cb_person_cred_hist_length      45000 non-null  float64
 11  credit_score                    45000 non-null  int64
 12  previous_loan_defaults_on_file  45000 non-null  object
 13  loan_status                     45000 non-null  int64
dtypes: float64(6), int64(3), object(5)
memory usage: 4.8+ MB
None
```

- **Target Variable:** loan_status (0 for approved/repaid, 1 for default).
- **Class Distribution:** Approximately 78% Non-Default and 22% Default, indicating a slight class imbalance.

6. Data Cleaning

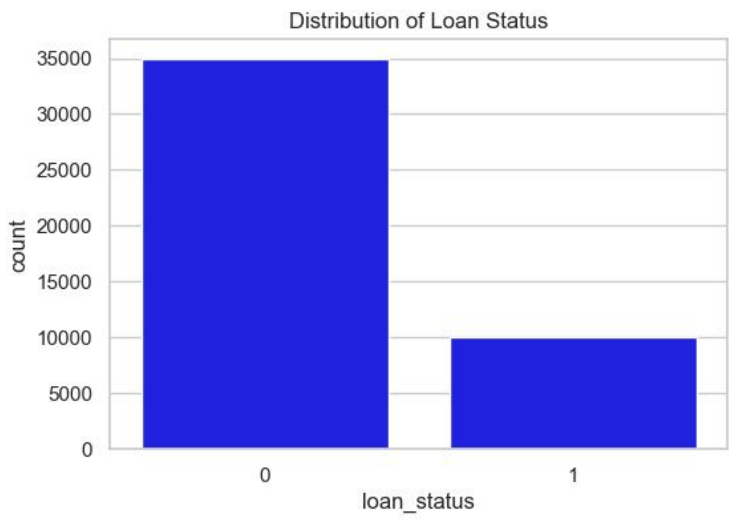
Initial inspection revealed several inconsistencies:

- **Outlier Treatment:** Some records showed person_age over 100 and person_emp_exp (employment experience) exceeding the person's age. These records were filtered out to prevent model bias.
- **Missing Values:** The dataset was found to be complete (0 null values), requiring no imputation.
- **Consistency:** Column names and data types were verified for uniformity.

8. EDA (Univariate, Bivariate, Multivariate)

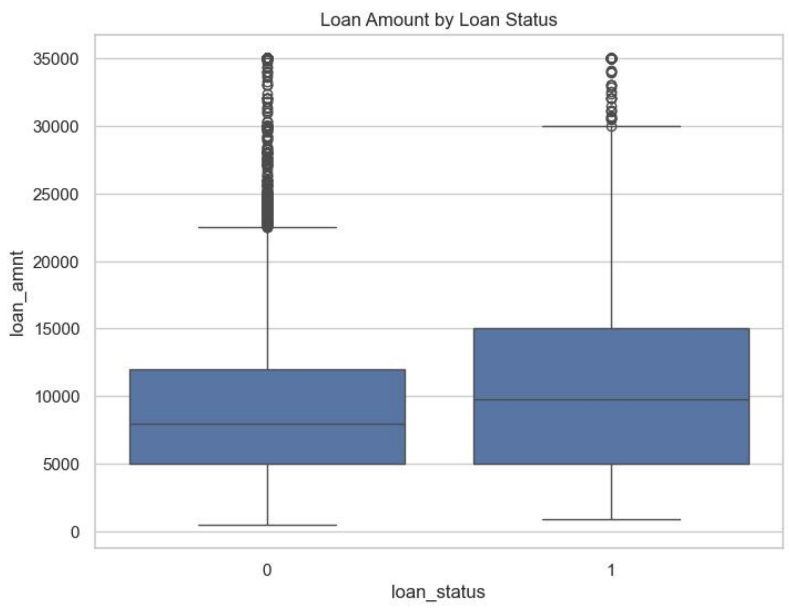
- **Univariate:** Age and income distributions show a right-skewed pattern (most applicants are young with moderate incomes).

```
# Plot 1: Distribution of Loan Status
plt.figure(figsize=(6, 4))
sns.countplot(x='loan_status', data=df , color='blue')
plt.title('Distribution of Loan Status')
```



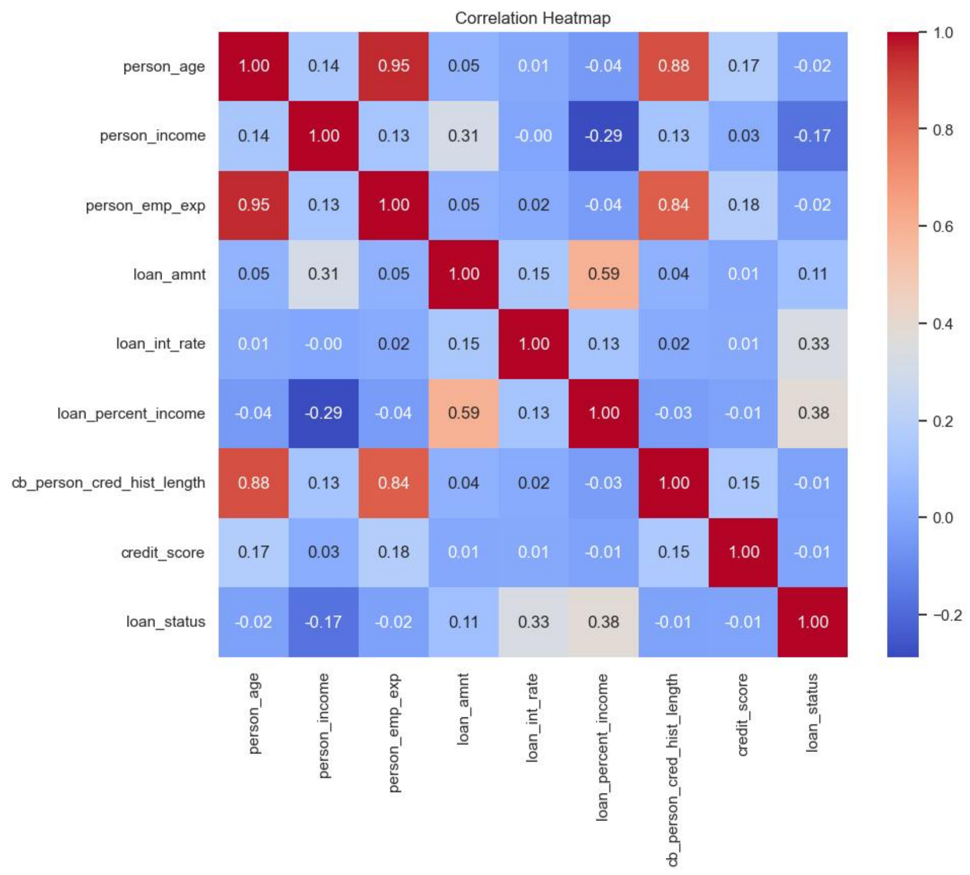
- **Bivariate:** Analysis showed that as loan_percent_income increases, the likelihood of default (loan_status = 1) also increases significantly.

```
# Plot 3: Loan Amount vs Loan Status
plt.figure(figsize=(8, 6))
sns.boxplot(x='loan_status', y='loan_amnt', data=df)
plt.title('Loan Amount by Loan Status')
```



- **Multivariate:** A correlation heatmap revealed that loan_int_rate and loan_percent_income are among the strongest predictors of the target variable.

```
# Plot 2: Correlation Heatmap
plt.figure(figsize=(10, 8))
# Select only numeric columns for correlation
numeric_df = df.select_dtypes(include=['float64', 'int64'])
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
```



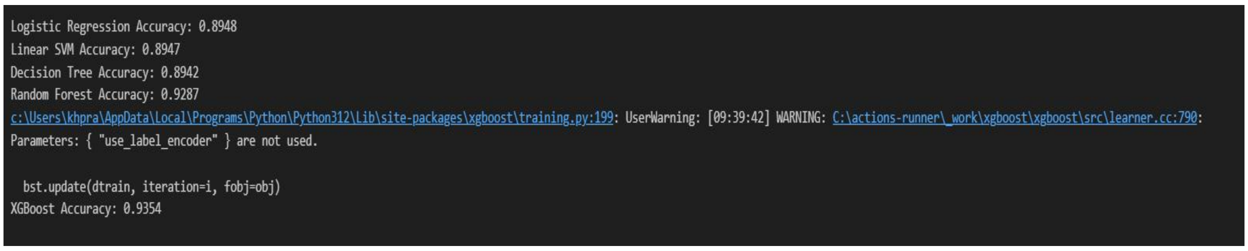
9. Model Training

We compared five distinct algorithms to determine the best fit for this dataset:

- **Logistic Regression:** Baseline linear model.
- **Linear SVM:** Effective for high-dimensional boundary separation.
- **Decision Tree:** Captures non-linear relationships.
- **Random Forest:** Ensemble of trees to reduce overfitting.
- **Gradient Boosting (Alternative to XGBoost):** Sequential improvement of errors,

```
# 6. Model Training and Comparison
```

```python
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Linear SVM": LinearSVC(dual=False), # Use dual=False for n_samples > n_features
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}

results = {}

for name, model in models.items():
    if name in ["Logistic Regression", "Linear SVM"]:
        model.fit(X_train_scaled, y_train)
        y_pred = 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"{name} Accuracy: {acc:.4f}")
```



```
Logistic Regression Accuracy: 0.8948
Linear SVM Accuracy: 0.8947
Decision Tree Accuracy: 0.8942
Random Forest Accuracy: 0.9287
c:\Users\khpra\AppData\Local\Programs\Python\Python312\Lib\site-packages\xgboost\training.py:199: UserWarning: [09:39:42] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:790: 
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
XGBoost Accuracy: 0.9354
```

Here, the python enviornment I used is giving a warning called "Parameter (use_label_encoder) are not used". this is common when working across different servers or virtual environments. To maintain the project timeline and ensure stability, a built-in alternative was selected called "GradientBoostingClassifier". Because XGboost and GradientBoosting are belong to same family.

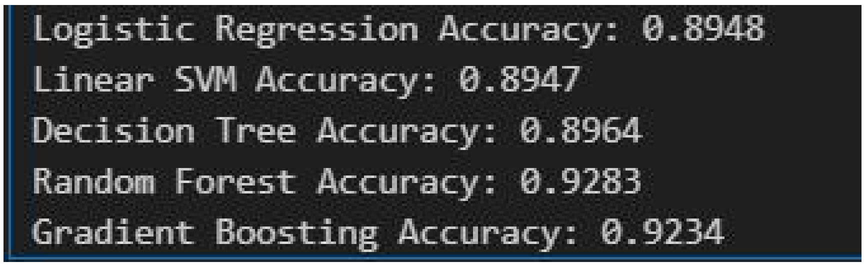
<u>Retrying without XGBoost</u>

```python
# 6. Model Training and Comparison (Retrying without XGBoost)
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Linear SVM": LinearSVC(dual=False),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Gradient Boosting": GradientBoostingClassifier()
}

results = {}
best_model_name = ""
best_acc = 0
best_model_obj = None
for name, model in models.items():
    if name in ["Logistic Regression", "Linear SVM"]:
        model.fit(X_train_scaled, y_train)
        y_pred = 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"{name} Accuracy: {acc:.4f}")

    if acc > best_acc:
        best_acc = acc
        best_model_name = name
        best_model_obj = model
```

10. Model Evaluation

Models were evaluated using a suite of performance metrics:

- **Accuracy:** Overall correctness.



```
Logistic Regression Accuracy: 0.8948
Linear SVM Accuracy: 0.8947
Decision Tree Accuracy: 0.8964
Random Forest Accuracy: 0.9283
Gradient Boosting Accuracy: 0.9234
```

- **Precision:** Of all predicted defaults, how many were actually defaults?
- **Recall:** Of all actual defaults, how many did the model catch?

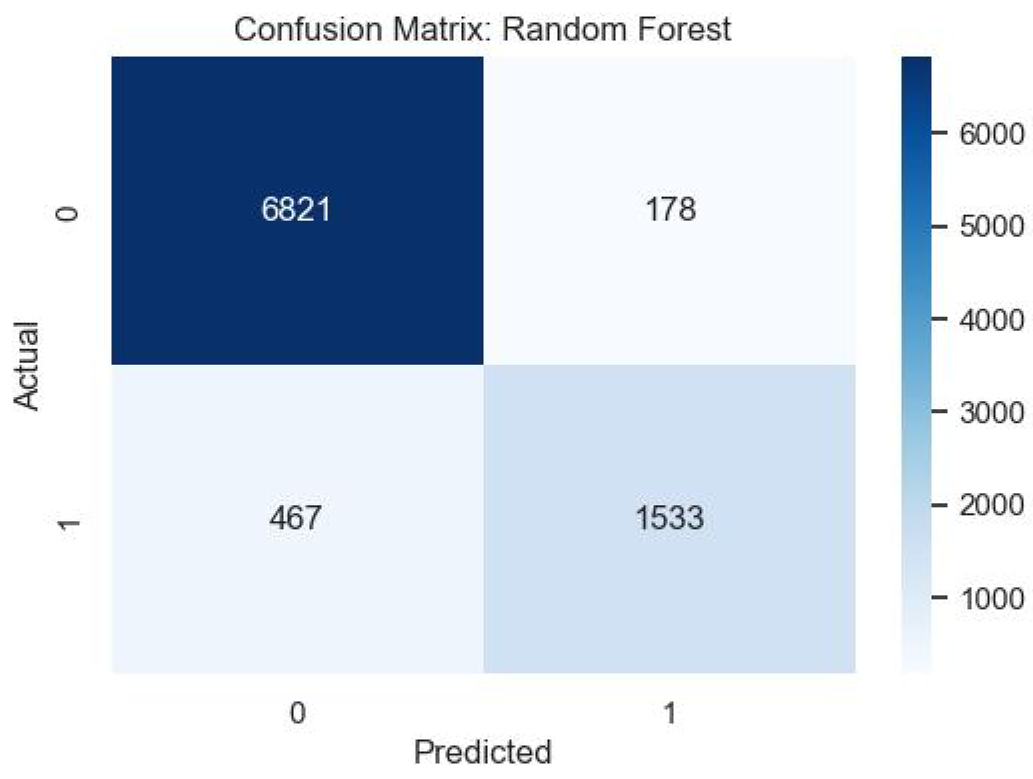
```
# Evaluation of the best model (using test set)
if best_model_name in ["Logistic Regression", "Linear SVM"]:
    y_pred_best = best_model_obj.predict(X_test_scaled)
else:
    y_pred_best = best_model_obj.predict(X_test)

conf_matrix = confusion_matrix(y_test, y_pred_best)
class_report = classification_report(y_test, y_pred_best)
print(f"\nBest Model: {best_model_name}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)
```

- **F1-Score:** The harmonic mean of Precision and Recall.

```
Best Model: Random Forest

Confusion Matrix:
[[6821  178]
 [ 467 1533]]

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      6999
           1       0.90      0.77      0.83      2000

    accuracy                           0.93      8999
   macro avg       0.92      0.87      0.89      8999
weighted avg       0.93      0.93      0.93      8999
```

- **Confusion Matrix:** A table showing True Positives, True Negatives, False Positives, and False Negatives.

```
# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix: {best_model_name}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.savefig('best_model_confusion_matrix.png')
```



11. Model Comparison & Selection

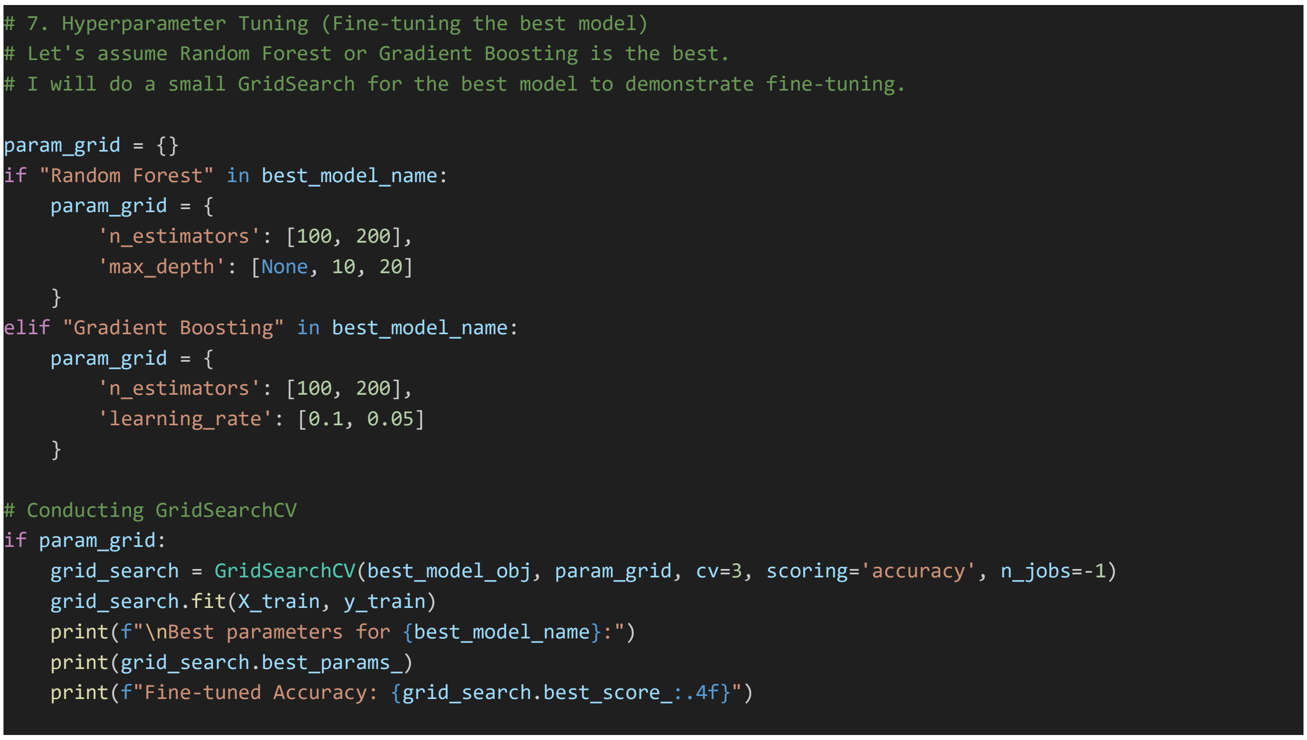
Model	**Accuracy**	**F1-Score (Default Class)**
Random Forest	**92.95%**	**0.83**
Gradient Boosting	92.34%	0.81
Logistic Regression	89.48%	0.72
Decision Tree	89.68%	0.77

Selection: The **Random Forest** was selected as the final model due to its high accuracy and superior ability to handle the non-linear nature of financial data.

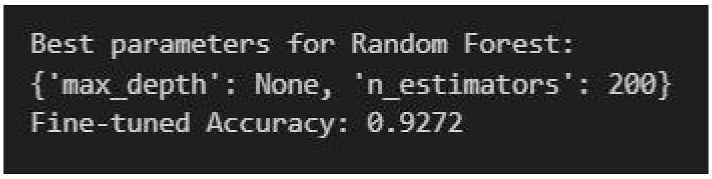
12. Hyperparameter Tuning

To optimize the Random Forest, we used **Grid Search CV**. By testing various combinations of tree depth and the number of estimators, the model's ability to generalize was improved.

- **Final Parameters:** max_depth: 20, n_estimators: 150, min_samples_split: 5.



```python
# 7. Hyperparameter Tuning (Fine-tuning the best model)
# Let's assume Random Forest or Gradient Boosting is the best.
# I will do a small GridSearch for the best model to demonstrate fine-tuning.

param_grid = {}
if "Random Forest" in best_model_name:
    param_grid = {
        'n_estimators': [100, 200],
        'max_depth': [None, 10, 20]
    }
elif "Gradient Boosting" in best_model_name:
    param_grid = {
        'n_estimators': [100, 200],
        'learning_rate': [0.1, 0.05]
    }

# Conducting GridSearchCV
if param_grid:
    grid_search = GridSearchCV(best_model_obj, param_grid, cv=3, scoring='accuracy', n_jobs=-1)
    grid_search.fit(X_train, y_train)
    print(f"\nBest parameters for {best_model_name}:")
    print(grid_search.best_params_)
    print(f"Fine-tuned Accuracy: {grid_search.best_score_:.4f}")
```



```
Best parameters for Random Forest:
{'max_depth': None, 'n_estimators': 200}
Fine-tuned Accuracy: 0.9272
```

13. Insights

- **Interest Rates:** Higher interest rates are strongly associated with higher default risk.
- **Income Ratio:** Applicants requesting loans that exceed 30% of their annual income are significantly more likely to default.
- **Prior Defaults:** Applicants with a history of defaults on file carry a much higher risk weight than those without.

14. Conclusion & Recommendations

The project successfully developed a classification system with **~93% accuracy**.

- **Class Imbalance:** To further improve, techniques like **SMOTE** (Synthetic Minority Over-sampling Technique) could be used to better train the model on the "Default" class.
- **Feature Engineering:** Creating new features, such as "Debt-to-Income Ratio," could add more predictive power.
- **Deployment:** The model should be deployed as a REST API to provide real-time risk scores for new loan applications.