Loan approval project

|  |  |
| --- | --- |
| **Mahmoud Mohamed Mahmoud** | **42110489** |
| **Abanoub Refat Helmy** | **42110092** |
| **Sandy Nazeh Noshy** | **42110452** |

**📊 Loan Approval Classification Dataset Overview**

**This dataset offers a comprehensive view of loan applications along with key financial and demographic attributes that contribute to assessing loan approval or rejection. The data includes both applicant-specific and loan-specific features that can help model the probability of loan approval. Below is a quick summary:**

**🔢 Number of Records: 45,000 🧩 Total Features: 14 (mix of Categorical and Continuous types)**

## Introduction

The notebook begins by defining the objectives of the project. It appears to involve analyzing a dataset to extract meaningful insights and build predictive models.

## Setup and Dependencies

The following Python libraries are imported and used in the notebook:

* numpy
* pandas
* matplotlib
* seaborn
* Additional libraries include machine learning frameworks (e.g., scikit-learn).

Ensure these dependencies are installed before running the notebook.

## Data Preprocessing

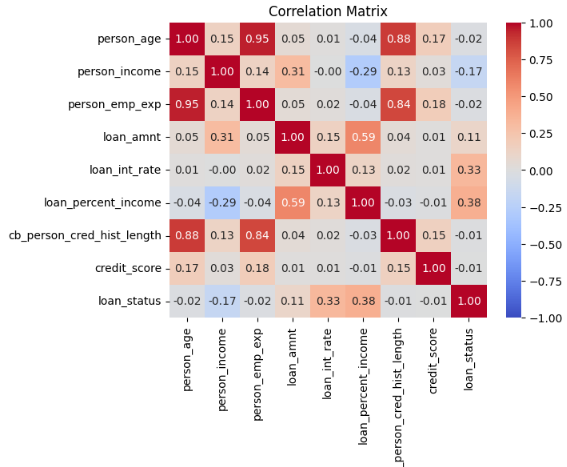
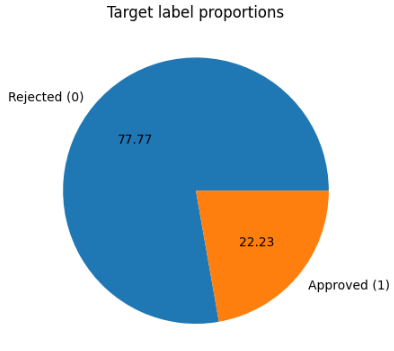
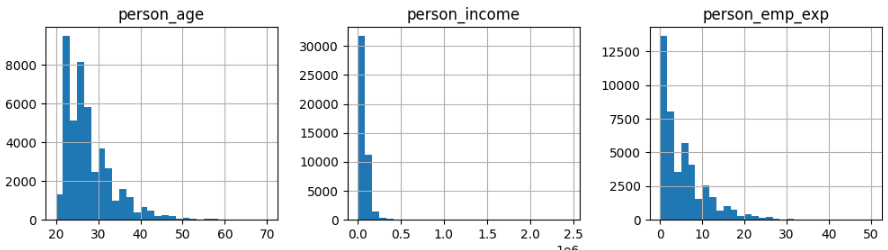
### Key Steps:

1. **Loading Data**: The dataset is loaded using pandas.read\_csv or similar methods.
2. **Cleaning**: Handling missing values, removing duplicates, and correcting inconsistent data.
3. **Feature Engineering**:
   1. Label encoding for categorical variables.
   2. Scaling applied to income and loan amount.
4. **Scaling**: Normalizing or standardizing features as required for modeling.
5. **Dealing with Outliers**: Identifying and addressing extreme values using statistical methods or domain-specific rules.

## Exploratory Data Analysis (EDA)

### Visualizations:

1. Histograms and box plots to examine distributions.
2. Correlation heatmaps to identify relationships between variables.

****

## Modeling

### Algorithms Used:

* Decision Tree
* Logistic Regression

Training and Validation:

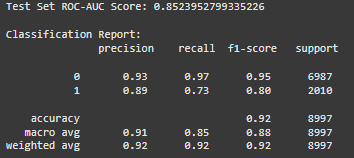
* Data is split into training and test sets using an 80:20 ratio.

For Decision Tree:

### Performance Metrics:

* **Test Set ROC-AUC Score**: 0.8524

**Classification Report:**

****

## 

## Test Set ROC-AUC Score: 0.85

## Interpretation: The ROC-AUC score of 0.85 indicates that the model has good performance in distinguishing between the two classes (approved vs not approved). A score of 1.0 would be perfect, and 0.5 would be no better than random guessing. So, 0.85 is a strong result.

## Classification Report

## Precision Precision for 0 (Not Approved): 0.93 This means that 93% of the cases predicted as "Not Approved" are actually correct.

## Precision for 1 (Approved): 0.89 This means that 89% of the cases predicted as "Approved" are actually correct.

## Recall Recall for 0 (Not Approved): 0.97 This means that 97% of all actual "Not Approved" cases were correctly identified by the model.

## Recall for 1 (Approved): 0.73 This means that 73% of all actual "Approved" cases were correctly identified by the model.

## F1-Score F1-Score for 0 (Not Approved): 0.95 The F1-Score is the harmonic mean of precision and recall, and for "Not Approved," this score suggests a very well-balanced performance.

## F1-Score for 1 (Approved): 0.80 For "Approved" loans, the F1-Score indicates a somewhat less balanced performance compared to "Not Approved.

## Support Support for 0 (Not Approved): 6987 This means there are 6987 instances of "Not Approved" in the test set.

## Support for 1 (Approved): 2010 There are 2010 instances of "Approved" in the test set.

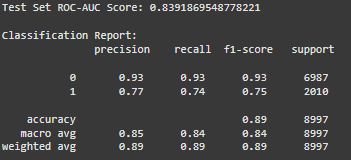
## Accuracy: 0.92 (92%) The model correctly predicted 92% of the cases in the test set

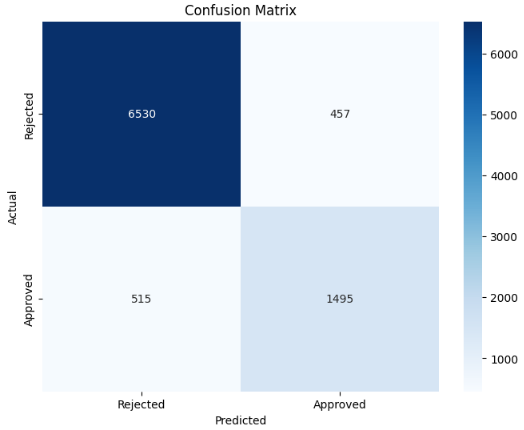
## For Logistic Regression:

### Performance Metrics:

* **Test Set ROC-AUC Score**: 0.8391

**Classification Report:**





**Interpretation:**

1. **Test Set ROC-AUC Score**: The ROC-AUC score of 0.8392 indicates good model performance in distinguishing between "Approved" and "Not Approved" classes. A score of 1.0 is perfect, and 0.5 indicates random guessing, so 0.8392 is a strong result.
2. **Precision**:
   * For Class 0 (Not Approved): 0.93, meaning 93% of predictions for "Not Approved" are correct.
   * For Class 1 (Approved): 0.77, meaning 77% of predictions for "Approved" are correct.
3. **Recall**:
   * For Class 0 (Not Approved): 0.93, meaning 93% of all actual "Not Approved" cases were correctly identified.
   * For Class 1 (Approved): 0.74, meaning 74% of all actual "Approved" cases were correctly identified.
4. **F1-Score**:
   * For Class 0 (Not Approved): 0.93, indicating a well-balanced performance between precision and recall.
   * For Class 1 (Approved): 0.75, showing slightly less balanced performance compared to Class 0.
5. **Support**:
   * For Class 0 (Not Approved): 6987 instances in the test set.
   * For Class 1 (Approved): 2010 instances in the test set.
6. **Accuracy**: The overall accuracy of 89% indicates that 89% of all test set predictions were correct.

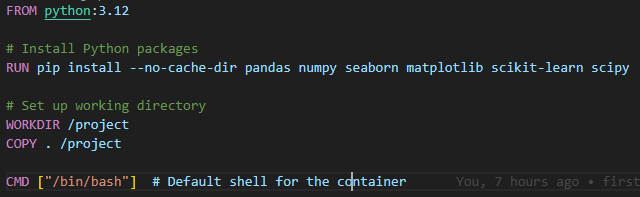
**Docker usage documentation**

This document explains the usage of Docker to create an environment for data processing, exploratory data analysis (EDA), visualization, preprocessing, and machine learning model training. The project is structured using Docker containers managed by a Docker Compose file. Below are the details of the implementation, including the Docker image, Python scripts, and Docker Compose configuration.

**Docker Image**

**Dockerfile**

The Docker image, named projectimage, is built using the following Dockerfile :



**Description**:

**Base Image**: python:3.12 provides a Python environment.

**Installed Libraries:** pandas, numpy, seaborn, matplotlib, scikit-learn, and scipy are installed to facilitate data processing, EDA, visualization, and machine learning.

**Working Directory**: The project files are copied into /project, the working directory within the container.

**Default Command:** The container opens a Bash shell by default.

**Python Scripts**

**1. load.py**

**Purpose:** Loads the dataset loan\_data.csv, processes it, and outputs:

loaded.csv as the initial processed dataset.

A file head.txt containing the dataset's header, saved in output/data/.

**Next Step**: Calls eda.py using os.system("python eda.py").

**2. eda.py**

**Purpose:** Performs exploratory data analysis (EDA) on loaded.csv.

**Output:** Generates analysis results saved in output/eda/.

**Next Step:** Calls vis.py using os.system("python vis.py").

**3. vis.py**

**Purpose:** Creates visualizations based on the dataset.

**Output:** Saves visualization files in output/vis/.

**Next Step**: Calls dpre.py using os.system("python dpre.py").

**4. dpre.py**

**Purpose:** Preprocesses loaded.csv and outputs:

A preprocessed dataset named res\_dpre.csv.

**Action:** Prints "preprocessing is done".

**5. model.py**

**Purpose**: Contains a function decision\_tree\_tuning\_and\_evaluation to:

Perform hyperparameter tuning for a decision tree model.

Evaluate the model's performance.

**Docker Compose Configuration**

**The Docker Compose file creates and manages three containers**: data\_processing, model\_training, and model\_training\_1. A shared volume output is used to store data persistently across containers.

**Docker Compose File**

services:

data\_processing:

build:

context: .

dockerfile: project.dockerfile

container\_name: data\_processing

volumes:

- ./output:/project/output

command: /bin/bash -c "

python load.py &&

cp /project/res\_dpre.csv /project/output/final.csv"

model\_training:

build:

context: .

dockerfile: project.dockerfile

container\_name: model\_training

volumes:

- ./output:/project/output

depends\_on:

- data\_processing

entrypoint: /bin/bash -c "

while [ ! -f /project/output/final.csv ]; do

echo 'Waiting for /project/output/res\_dpre.csv to be created...';

sleep 5;

done &&

python model.py /project/output/final.csv"

model\_training\_1:

build:

context: .

dockerfile: project.dockerfile

container\_name: model\_training\_1

volumes:

- ./output:/project/output

depends\_on:

- data\_processing

entrypoint: /bin/bash -c "

while [ ! -f /project/output/final.csv ]; do

echo 'Waiting for /project/output/res\_dpre.csv to be created...';

sleep 5;

done &&

python model.py /project/output/final.csv"

**Explanation of Services**

**1. data\_processing Service**

**Build:** Uses the project.dockerfile to create the Docker image.

Command:

Executes python load.py to process the dataset.

Copies res\_dpre.csv to the output directory, renaming it as final.csv for further processing.

**2. model\_training Service**

**Depends On:** Waits for the data\_processing container to complete.

**Entrypoint:**

Implements a while loop to repeatedly check if final.csv exists in the output directory.

Purpose of the while loop:

Ensures model\_training doesn't start until data\_processing has completed.

Prints "Waiting for /project/output/res\_dpre.csv to be created..." every 5 seconds until the file is available.

Runs python model.py using final.csv as input.

**3. model\_training\_1 Service**

**Functionality:** Identical to model\_training, providing redundancy or parallel training capabilities.

Workflow Summary

The data\_processing container processes the dataset and saves the preprocessed file as final.csv in the output directory.

The model\_training and model\_training\_1 containers wait for final.csv to be created. Once available:

They execute the decision\_tree\_tuning\_and\_evaluation function in model.py to train and evaluate the model.

A shared output volume ensures all containers can access intermediate and final outputs.

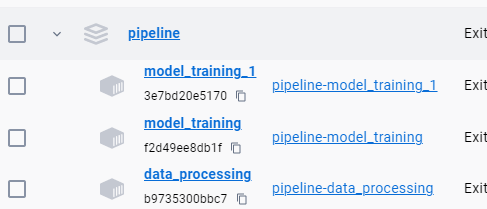
**Key Outputs**

output/data/head.txt: Contains the dataset header.

output/eda/: Stores results from eda.py.

output/vis/: Stores visualizations from vis.py.

output/final.csv: The preprocessed dataset, used as input for model training.



**Git Integration**

A Bash script, git\_script.sh, is provided to manually update the GitHub repository whenever changes occur in the output folder within the pipeline directory.

**Script Details**

**Purpose:** Detects changes in the output folder and pushes updates to the GitHub repository.

**Execution:** Run the script manually to ensure the repository stays synchronized with the latest results.

**Notes**

The while loop in model\_training services ensures synchronization by checking for the existence of final.csv before proceeding.

The shared output volume enables seamless data sharing between containers.

This setup provides a modular and automated pipeline for data processing, analysis, and machine learning model training using Docker.