**CREDIT SCORE MODEL DOCUMENTATION**

This Python code implements a machine learning pipeline to build a credit scoring model using logistic regression. The model can be used to predict the likelihood of a loan applicant defaulting on a loan based on their financial and demographic information.

Functionality:

\* Data Loading: The code can load data from a CSV file or generate synthetic data if the file is not found.

\* Preprocessing: The code handles missing values, encodes categorical variables, splits the data into features and target, splits the data into training and testing sets, and standardizes the features.

\* Model Training: The code trains a logistic regression model to classify loan applicants as likely to default (1) or not (0).

\* Model Evaluation: The code evaluates the model's performance using various metrics such as accuracy, precision, recall, F1-score, ROC AUC score, confusion matrix, and classification report.

\* Model Saving: The code saves the trained model and scaler for future use.

Code Structure:

The code is organized into several functions:

\* generate\_synthetic\_data(file\_path): Generates a synthetic dataset if the specified file path doesn't exist.

\* load\_data(file\_path): Loads data from a CSV file or generates synthetic data if the file is not found.

\* preprocess\_data(data, target\_column): Preprocesses the data by handling missing values, encoding categorical variables, splitting the data, and standardizing the features.

\* train\_model(X\_train, y\_train): Trains a logistic regression model on the provided training data.

\* evaluate\_model(model, X\_test, y\_test): Evaluates the model's performance on the testing data.

\* save\_model(model, scaler, model\_path, scaler\_path): Saves the trained model and scaler to specified file paths.

\* main(): The main function that executes the pipeline by loading data, preprocessing, training the model, evaluating the model, and saving the model and scaler.

How to Use:

\* Replace "credit\_data.csv" with the path to your actual CSV file containing the credit scoring data.

\* Ensure the CSV file has the target column named "default" (or replace with your target column name if different).

\* Run the script (python script\_name.py).

Output:

The script will print messages indicating successful data loading, model training completion, and evaluation results. The trained model and scaler will be saved to the specified file paths ("credit\_scoring\_model.pkl" and "scaler.pkl" by default).

Notes:

\* This is a basic example of a credit scoring model. Real-world credit scoring models may involve more complex techniques and feature engineering.

\* The synthetic data generation is for demonstration purposes only and may not reflect real-world credit data distributions.

Platform:

Google Colab:

Google Colab is a cloud-based platform by Google that provides a Jupyter Notebook environment for writing and executing Python code. It is widely used for data science, machine learning, and deep learning tasks due to its accessibility and simplicity. Colab supports GPU and TPU acceleration, allowing users to train models faster without requiring powerful local hardware.

Users can access files directly from Google Drive, import datasets from various sources, and install custom Python libraries. Collaboration is seamless, enabling multiple users to edit and run notebooks simultaneously. With features like markdown support, code snippets, and visualization tools.

**Code:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score

from sklearn.metrics import confusion\_matrix, classification\_report

import joblib

# Step 1: Load the dataset

def load\_data(file\_path):

try:

data = pd.read\_csv(file\_path)

print("Data loaded successfully.")

return data

except Exception as e:

print(f"Error loading data: {e}")

return None

# Step 2: Preprocess the data

def preprocess\_data(data, target\_column):

# Handle missing values

data = data.dropna()

# Encode categorical variables

data = pd.get\_dummies(data, drop\_first=True)

# Split features and target

X = data.drop(target\_column, axis=1)

y = data[target\_column]

# Split into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

return X\_train, X\_test, y\_train, y\_test, scaler

# Step 3: Train the logistic regression model

def train\_model(X\_train, y\_train):

model = LogisticRegression()

model.fit(X\_train, y\_train)

print("Model training completed.")

return model

# Step 4: Evaluate the model

def evaluate\_model(model, X\_test, y\_test):

y\_pred = model.predict(X\_test)

y\_pred\_prob = model.predict\_proba(X\_test)[:, 1]

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred))

print("Recall:", recall\_score(y\_test, y\_pred))

print("F1 Score:", f1\_score(y\_test, y\_pred))

print("ROC AUC Score:", roc\_auc\_score(y\_test, y\_pred\_prob))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

# Step 5: Save the model and scaler

def save\_model(model, scaler, model\_path, scaler\_path):

joblib.dump(model, model\_path)

joblib.dump(scaler, scaler\_path)

print(f"Model saved to {model\_path}")

print(f"Scaler saved to {scaler\_path}")

# Main function

def main():

# File path to your dataset

file\_path = "credit\_data.csv" # Replace with your file path

target\_column = "default" # Replace with your target column name

data = load\_data(file\_path)

if data is not None:

X\_train, X\_test, y\_train, y\_test, scaler = preprocess\_data(data, target\_column)

model = train\_model(X\_train, y\_train)

evaluate\_model(model, X\_test, y\_test)

save\_model(model, scaler, "credit\_scoring\_model.pkl", "scaler.pkl")

if \_name\_ == "\_main\_":

main()

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score

from sklearn.metrics import confusion\_matrix, classification\_report

import joblib

import os

# Step 1: Generate synthetic data (if no file provided)

def generate\_synthetic\_data(file\_path):

if not os.path.exists(file\_path):

print("Dataset not found. Generating synthetic dataset...")

np.random.seed(42)

num\_samples = 1000

data = {

'age': np.random.randint(18, 70, num\_samples),

'income': np.random.randint(20000, 150000, num\_samples),

'loan\_amount': np.random.randint(1000, 50000, num\_samples),

'credit\_score': np.random.randint(300, 850, num\_samples),

'employment\_years': np.random.randint(0, 30, num\_samples),

'default': np.random.choice([0, 1], num\_samples, p=[0.7, 0.3])

}

df = pd.DataFrame(data)

df.to\_csv(file\_path, index=False)

print(f"Synthetic dataset saved to {file\_path}")

# Step 2: Load the dataset

def load\_data(file\_path):

generate\_synthetic\_data(file\_path)

try:

data = pd.read\_csv(file\_path)

print("Data loaded successfully.")

return data

except Exception as e:

print(f"Error loading data: {e}")

return None

# Step 3: Preprocess the data

def preprocess\_data(data, target\_column):

# Handle missing values

data = data.dropna()

# Encode categorical variables

data = pd.get\_dummies(data, drop\_first=True)

# Split features and target

X = data.drop(target\_column, axis=1)

y = data[target\_column]

# Split into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

return X\_train, X\_test, y\_train, y\_test, scaler

# Step 4: Train the logistic regression model

def train\_model(X\_train, y\_train):

model = LogisticRegression()

model.fit(X\_train, y\_train)

print("Model training completed.")

return model

# Step 5: Evaluate the model

def evaluate\_model(model, X\_test, y\_test):

y\_pred = model.predict(X\_test)

y\_pred\_prob = model.predict\_proba(X\_test)[:, 1]

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred))

print("Recall:", recall\_score(y\_test, y\_pred))

print("F1 Score:", f1\_score(y\_test, y\_pred))

print("ROC AUC Score:", roc\_auc\_score(y\_test, y\_pred\_prob))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

# Step 6: Save the model and scaler

def save\_model(model, scaler, model\_path, scaler\_path):

joblib.dump(model, model\_path)

joblib.dump(scaler, scaler\_path)

print(f"Model saved to {model\_path}")

print(f"Scaler saved to {scaler\_path}")

# Main function

def main():

# File path to your dataset

file\_path = "credit\_data.csv" # Replace with your file path

target\_column = "default" # Replace with your target column name

data = load\_data(file\_path)

if data is not None:

X\_train, X\_test, y\_train, y\_test, scaler = preprocess\_data(data, target\_column)

model = train\_model(X\_train, y\_train)

evaluate\_model(model, X\_test, y\_test)

save\_model(model, scaler, "credit\_scoring\_model.pkl", "scaler.pkl")

if \_name\_ == "\_main\_":

main()