# Table of Content

[**Table of Content 1**](#_biokaxpttft)

[**Practical 1: Creating Robo-advisor chatbot 2**](#_1hmuv1b649i3)

[**Practical 2: Creating Trading Algorithms 6**](#_6v0g5ldkwg1c)

[**Practical 3: Fraud Detection combining Benford’s law with ML 11**](#_y38y8ldbw63m)

[**Practical 4: Using AI to distinguish between a prime or sub-prime customer 14**](#_mp6d6221klem)

[**Practical 5: E-KYC module using AI – capturing data from identity proof 25**](#_gb0u7j1kybt7)

[**Practical 6: Creating an AI based Claims Management system in Auto-insurance 29**](#_xbe760he2kl4)

[**Practical 7: Writing identity data on a blockchain 35**](#_nohx0w2okeu7)

# 

# Practical 1: Creating Robo-advisor chatbot

## **Introduction**

The objective of this practical is to design and implement a Robo-Advisor chatbot using Python. The chatbot will act as a financial advisor, providing users with investment advice based on their financial goals, risk tolerance, and preferences. The project will involve creating a simple conversational interface, implementing decision-making algorithms, and leveraging data to suggest financial products or strategies.

## Codes

def get\_recommendation(risk\_tolerance, financial\_goal, time\_horizon, sustainability\_interest):

recommendations = {

"Conservative": {

"Retirement": {

"0-10 years": {

"No": "60% in bonds, 30% in conservative mutual funds, 10% in cash or high-quality stocks",

"Yes": "60% in green bonds, 30% in ESG-focused conservative mutual funds, 10% in sustainable money market funds"

},

"10-15 years": {

"No": "50% in bonds, 40% in conservative mutual funds, 10% in cash or high-quality stocks",

"Yes": "50% in green bonds, 40% in ESG-focused conservative mutual funds, 10% in sustainable stocks"

},

"15-20 years": {

"No": "40% in bonds, 40% in conservative mutual funds, 20% in cash or high-quality stocks",

"Yes": "40% in green bonds, 40% in ESG-focused conservative mutual funds, 20% in sustainable stocks"

}

},

"Buying a House": {

"0-10 years": {

"No": "70% in bonds, 20% in conservative mutual funds, 10% in cash",

"Yes": "70% in green bonds, 20% in ESG-focused conservative mutual funds, 10% in sustainable money market funds"

},

"10-15 years": {

"No": "60% in bonds, 30% in conservative mutual funds, 10% in cash",

"Yes": "60% in green bonds, 30% in ESG-focused conservative mutual funds, 10% in sustainable stocks"

},

"15-20 years": {

"No": "50% in bonds, 40% in conservative mutual funds, 10% in cash",

"Yes": "50% in green bonds, 40% in ESG-focused conservative mutual funds, 10% in sustainable stocks"

}

},

"Education": {

"0-10 years": {

"No": "50% in bonds, 30% in balanced mutual funds, 20% in cash",

"Yes": "50% in green bonds, 30% in ESG-focused balanced mutual funds, 20% in sustainable money market funds"

},

"10-15 years": {

"No": "40% in bonds, 40% in balanced mutual funds, 20% in high-quality stocks",

"Yes": "40% in green bonds, 40% in ESG-focused balanced mutual funds, 20% in sustainable stocks"

},

"15-20 years": {

"No": "30% in bonds, 50% in balanced mutual funds, 20% in high-quality stocks",

"Yes": "30% in green bonds, 50% in ESG-focused balanced mutual funds, 20% in sustainable high-growth stocks"

}

}

}

}

# Validate inputs

if risk\_tolerance not in recommendations:

return "Invalid risk tolerance."

if financial\_goal not in recommendations[risk\_tolerance]:

return "Invalid financial goal."

if time\_horizon not in recommendations[risk\_tolerance][financial\_goal]:

return "Invalid time horizon."

if sustainability\_interest not in recommendations[risk\_tolerance][financial\_goal][time\_horizon]:

return "Invalid sustainability interest."

# Get the recommendation

recommendation = recommendations[risk\_tolerance][financial\_goal][time\_horizon][sustainability\_interest]

return recommendation

def main():

print("Welcome to the Robo-Advisor Chatbot!")

# Get user details

name = input("Please enter your name: ")

risk\_tolerance = input("Enter your risk tolerance (Conservative, Moderate, Aggressive): ").capitalize()

financial\_goal = input("Enter your financial goal (Retirement, Buying a House, Education): ").capitalize()

time\_horizon = input("Enter your time horizon (0-10 years, 10-15 years, 15-20 years): ")

sustainability\_interest = input("Are you interested in sustainable investments? (Yes, No): ").capitalize()

# Get recommendation

recommendation = get\_recommendation(risk\_tolerance, financial\_goal, time\_horizon, sustainability\_interest)

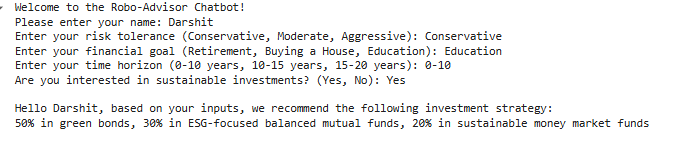
# Print personalized recommendation

print(f"\nHello {name}, based on your inputs, we recommend the following investment strategy:")

print(recommendation)

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

main()



## Conclusion

In this Practical, we explored the process of developing a Robo-Advisor chatbot using Python, aimed at providing personalized financial advice based on a user's profile, goals, and risk tolerance. By integrating basic Natural Language Processing (NLP) techniques for understanding user inputs, we were able to create an interactive system that guides users through key financial decisions, such as selecting appropriate investment products and strategies.

# Practical 2: Creating Trading Algorithms

## Introduction

Trading algorithms are automated systems designed to execute trading decisions based on pre-defined rules and market data analysis. A common approach in algorithmic trading is to use signals based on technical indicators to trigger buy or sell actions.

A **buy signal** typically arises when certain conditions suggest the asset's price will increase, like when a short-term moving average (e.g., a 10-day average, or **short window**) crosses above a long-term moving average (e.g., a 50-day average, or **long window**). Conversely, a **sell signal** is generated when the short window moves below the long window, indicating potential downward movement.

To evaluate the effectiveness of a trading algorithm, **backtesting** is performed. Backtesting involves running the algorithm on historical data to simulate past trading outcomes and estimate the strategy's potential profitability. During this simulation, variables like **available cash** and **shares held** are tracked to ensure that positions can be entered and exited based on funds and holdings. This also allows testing of a portfolio's responses to **holding long** (owning shares expecting them to rise) and **short** (selling borrowed shares expecting them to fall) positions, revealing insights into risk management and profitability.

## Codes

import yfinance as yf

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

def fetch\_data(ticker, start\_date, end\_date):

df = yf.download(ticker, start = start\_date, end = end\_date)

df = df[['Close']]

return df

def calculate\_moving\_averages(df, short\_window, long\_window):

df['MA\_S'] = df['Close'].rolling(window = short\_window, min\_periods = 1).mean()

df['MA\_L'] = df['Close'].rolling(window = long\_window, min\_periods = 1).mean()

return df

def buy\_sell\_signal(df):

df['Signal'] = 0

df['Signal'][short\_window:] = np.where(df['MA\_S'][short\_window:] > df['MA\_L'][short\_window:], 1, 0)

df['Position'] = df['Signal'].diff()

return df

def backtest\_strategy(df, initial\_capital=10000):

df['Availabale Cash'] = initial\_capital

df['Shares\_held'] = 0

for i in range(1, len(df)):

if df['Shares\_held'].iloc[i-1] == 0 and df['Signal'].iloc[i] == 1:

df['Shares\_held'].iloc[i] = df['Availabale Cash'].iloc[i-1] / df['Close'].iloc[i]

df['Availabale Cash'].iloc[i] = 0

elif df['Shares\_held'].iloc[i-1] > 0 and df['Signal'].iloc[i] == -1:

df['Availabale Cash'].iloc[i] = df['Shares\_held'].iloc[i-1] \* df['Close'].iloc[i]

df['Shares\_held'].iloc[i] = 0

else:

df['Availabale Cash'].iloc[i] = df['Availabale Cash'].iloc[i-1]

df['Shares\_held'].iloc[i] = df['Shares\_held'].iloc[i-1]

df['Portfolio Total'] = df['Availabale Cash'] + (df['Shares\_held'] \* df['Close'])

return df

def plt\_strategy(df, ticker):

plt.figure(figsize = (14, 7))

plt.plot(df['Close'], label = f'{ticker} Price', color = 'black', alpha = 0.6)

plt.plot(df['MA\_S'], label = f'{ticker} {short\_window} Short MA', color = 'blue', alpha = 0.7)

plt.plot(df['MA\_L'], label = f'{ticker} {long\_window} Long MA', color = 'orange', alpha = 0.7)

plt.plot(df[df['Position'] == 1].index, df['MA\_S'][df['Position'] == 1], '^', markersize = 10, color = 'green', alpha = 1, label = 'Buy Signal')

plt.plot(df[df['Position'] == -1].index, df['MA\_S'][df['Position'] == -1], 'v', markersize = 10, color = 'red', alpha = 1, label = 'Sell Signal')

plt.title(f'{ticker} Price and Moving Averages')

plt.xlabel('Date')

plt.ylabel('Close Price USD ($)')

plt.legend(loc = 'upper left')

plt.grid()

plt.show()

def calculate\_performance\_metrics(df):

df['Daily Return'] = df['Portfolio Total'].pct\_change()

cumulative\_return = (df['Portfolio Total'].iloc[-1] / df['Portfolio Total'].iloc[0]) - 1

annualized\_return = df['Daily Return'].mean() \* 252

annualized\_volatility = df['Daily Return'].std() \* np.sqrt(252)

sharpe\_ratio = annualized\_return / annualized\_volatility

max\_drawdown = (df['Portfolio Total'].cummax() - df['Portfolio Total']).max()

return {

'Cumulative Return': cumulative\_return,

'Annualized Return': annualized\_return,

'Annualized Volatility': annualized\_volatility,

'Sharpe Ratio': sharpe\_ratio,

'Max Drawdown': max\_drawdown

}

#APPLE SHARES

if \_\_name\_\_ == '\_\_main\_\_':

ticker = 'AAPL'

start\_date = '2010-01-01'

end\_date = '2023-01-01'

short\_window = 50

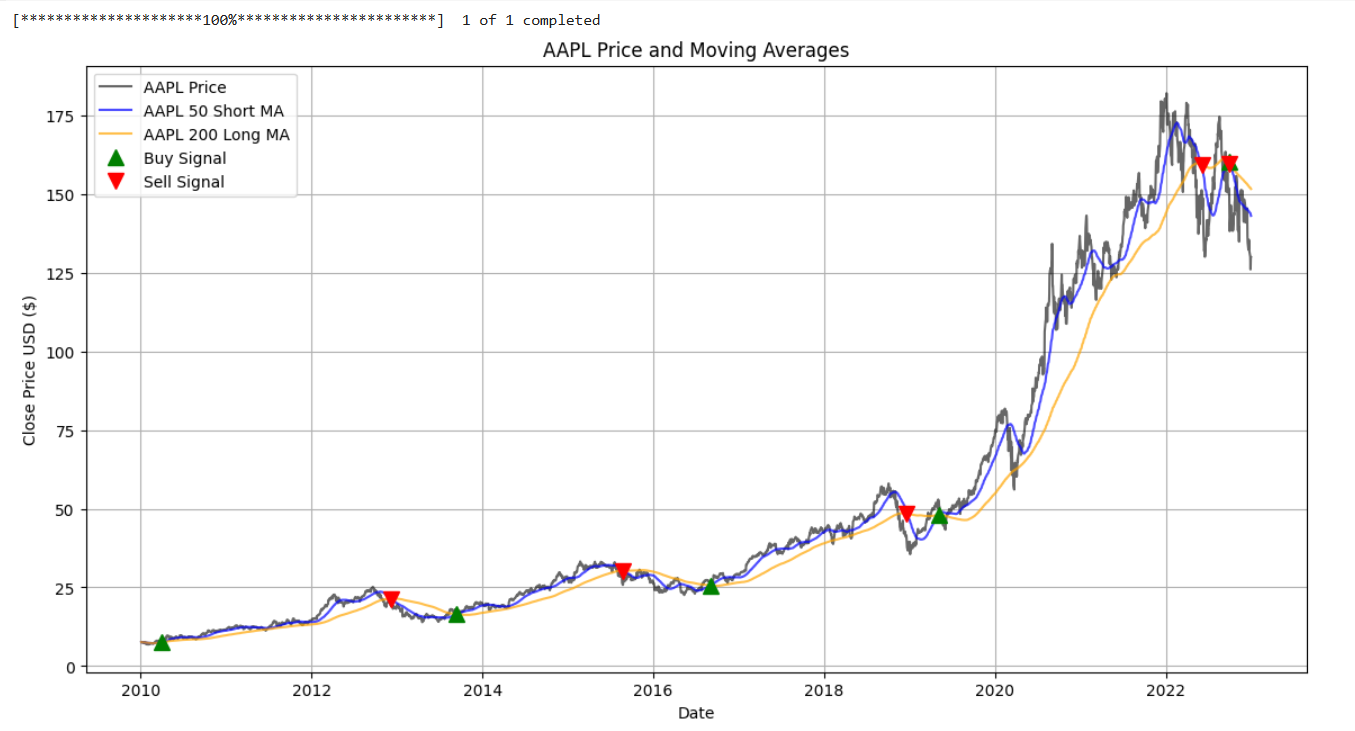
long\_window = 200

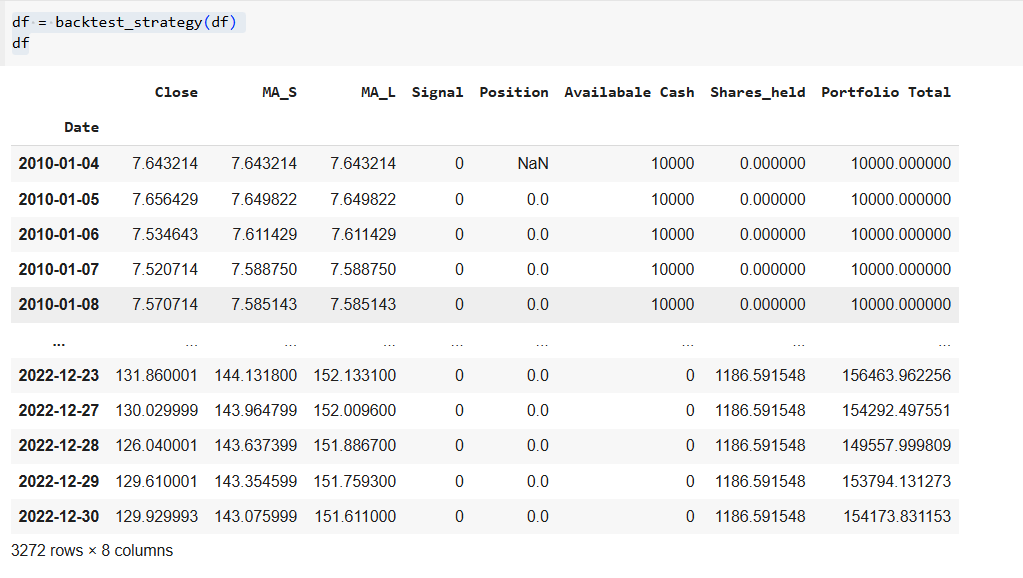
df = fetch\_data(ticker, start\_date, end\_date)

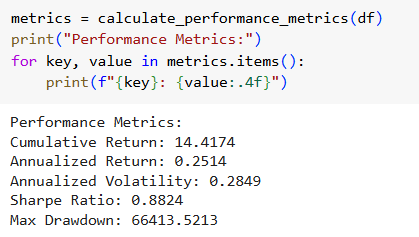
df = calculate\_moving\_averages(df, short\_window, long\_window)

df = buy\_sell\_signal(df)

plt\_strategy(df, ticker)







## Conclusion

This practical exercise provides hands-on experience with trading algorithms by implementing and testing buy and sell signals based on technical indicators. Through backtesting, we gain valuable insights into the algorithm's performance over historical data, helping to refine strategies and manage risk with variables like available cash, shares held, and moving average windows. This approach highlights the potential profitability of algorithmic trading while underscoring the importance of rigorous testing before applying strategies to live markets.

# Practical 3: Fraud Detection combining Benford’s law with ML

## Introduction

The objective of this practical is to explore the application of Benford’s Law in detecting financial fraud by combining it with Machine Learning (ML) techniques. Benford’s Law, also known as the First-Digit Law, predicts the frequency distribution of the first digits in many natural datasets. This law can be leveraged to detect anomalies in numerical data that may suggest fraudulent activity. By integrating Benford’s Law with machine learning algorithms, we aim to build a more robust fraud detection system that can identify suspicious patterns and potential fraudulent transactions.

Codes-

import pandas as pd

import numpy as np

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, roc\_auc\_score

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.stats import chisquare

## Load the dataset

url='fraud.csv'

df=pd.read\_csv(url)

## Benfords Law Check Function

def benfords\_law\_abakysis(df,colum\_name):

## Extract the first digit from financial data

first\_digit=df[colum\_name].astype(str).str[0].astype(int)

## Count the frequency of each digit

actual\_count=first\_digit.value\_counts().sort\_index()

## Expected Benfords Law Distribution for First Digits

expected\_distribution=np.log10(1+1/np.arange(1,11))

expected\_distribution=expected\_distribution/expected\_distribution.sum()

# expected\_distribution=expected\_dist.reindex(range(1,11),fill\_value=0)

# print(actual\_count.sum)

print(expected\_distribution.sum)

## Plot Actual vs expected Distribution

plt.figure(figsize=(10,6))

plt.bar(actual\_count.index,actual\_count/actual\_count.sum(),label='Actual',color='blue',alpha=0.6)

plt.plot(np.arange(1,11),expected\_distribution,label='Expected (Benfords Law)',color='red',marker='o')

plt.xlabel("First Digit")

plt.ylabel("Proportion")

plt.title(f"Benfords Law Analysis for {colum\_name}")

plt.legend()

plt.show()

actual\_dist=actual\_count/actual\_count.sum()

# actual\_dist=actual\_dist.reindex(range(1,11),fill\_value=0)

# print(actual\_dist.shape)

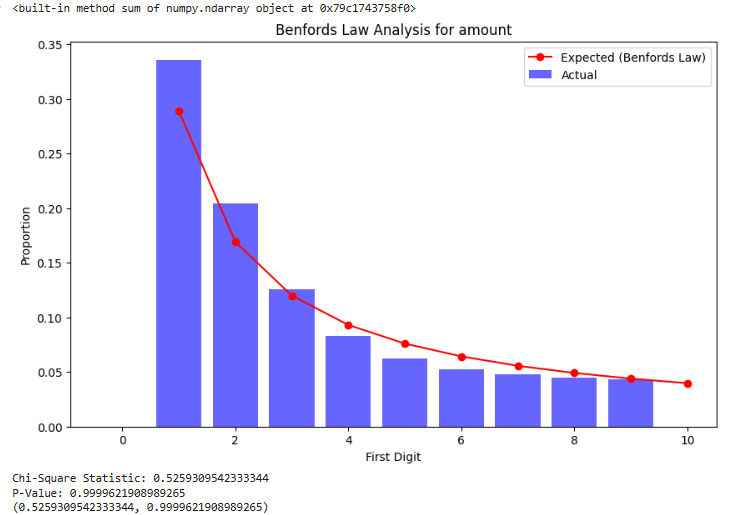
# print(expected\_distribution.shape)

chisq,p\_value=chisquare(f\_obs=actual\_dist,f\_exp=expected\_distribution)

print(f"Chi-Square Statistic: {chisq}")

print(f"P-Value: {p\_value}")

return chisq,p\_value

benfords\_law\_abakysis(df,'amount')

## Conclusion

In this Practical , we successfully explored the integration of **Benford’s Law** with **Machine Learning** for fraud detection, leveraging both statistical and algorithmic techniques to identify anomalous and potentially fraudulent transactions in financial data. By combining the power of Benford’s Law—an established statistical principle for detecting anomalies in numerical datasets—with machine learning algorithms, we built a more robust fraud detection system capable of identifying subtle patterns that traditional methods might miss.

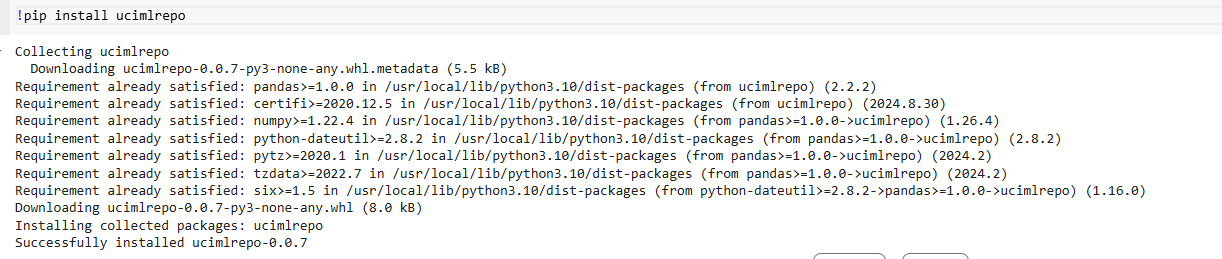
# Practical 4: Using AI to distinguish between a prime or sub-prime customer

## Introduction

In finance, AI can be a powerful tool to distinguish between prime and sub-prime customers, enhancing the accuracy of credit risk assessments. By analyzing large datasets of customer financial behavior, AI models can identify patterns and risk indicators—such as income stability, spending habits, debt levels, and credit history—that signal whether a customer is prime (low risk) or sub-prime (higher risk).

Integrating this process with **blockchain** further improves transparency and security. Blockchain’s immutable records enable a verifiable audit trail of each customer’s creditworthiness data, which AI models can access and evaluate without risk of tampering. This combination of AI-driven risk scoring with blockchain-secured data offers a more reliable and efficient approach to assessing and managing financial risk in lending and investment decisions.

## Codes



from ucimlrepo import fetch\_ucirepo

# fetch dataset

statlog\_german\_credit\_data = fetch\_ucirepo(id=144)

# data (as pandas dataframes)

X = statlog\_german\_credit\_data.data.features

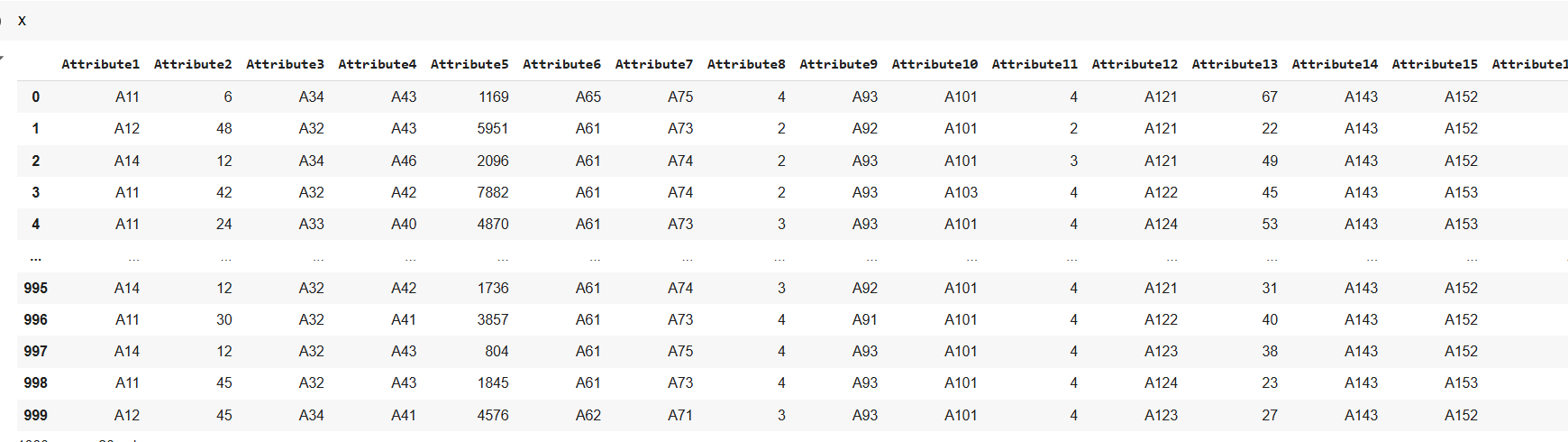
y = statlog\_german\_credit\_data.data.targets

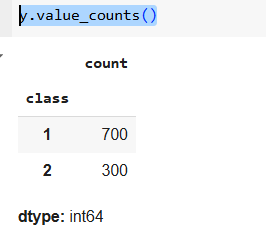
# metadata

print(statlog\_german\_credit\_data.metadata)

# variable information

print(statlog\_german\_credit\_data.variables)





column\_names = column\_names = [

'Status\_of\_existing\_checking\_account',

'Duration\_in\_month',

'Credit\_history',

'Purpose',

'Credit\_amount',

'Savings\_account\_bonds',

'Present\_employment\_since',

'Installment\_rate\_in\_percentage\_of\_disposable\_income',

'Personal\_status\_and\_sex',

'Other\_debtors\_guarantors',

'Present\_residence\_since',

'Property',

'Age\_in\_years',

'Other\_installment\_plans',

'Housing',

'Number\_of\_existing\_credits\_at\_this\_bank',

'Job',

'Number\_of\_people\_being\_liable\_to\_provide\_maintenance\_for',

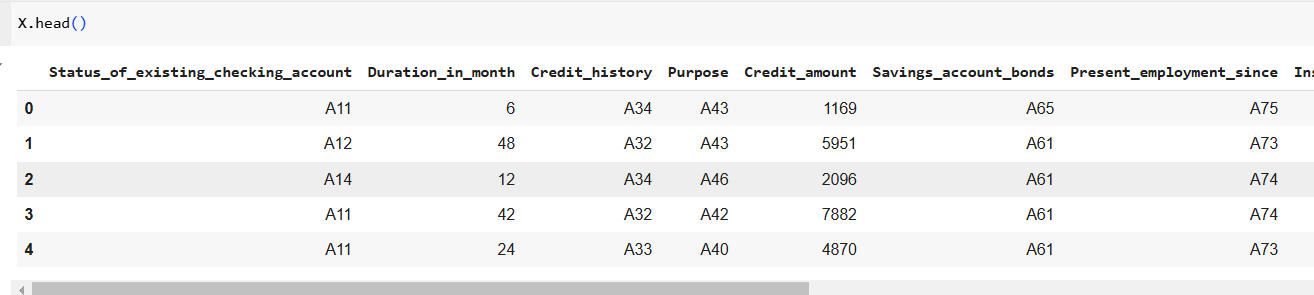
'Telephone',

'foreign\_worker'

# 'Credit\_Risk' # 1 = Good, 2 = Bad

]

X.columns = column\_names



import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix, roc\_auc\_score

import matplotlib.pyplot as plt

import seaborn as sns

categorical\_cols = [

'Status\_of\_existing\_checking\_account',

'Credit\_history',

'Purpose',

'Savings\_account\_bonds',

'Present\_employment\_since',

'Personal\_status\_and\_sex',

'Other\_debtors\_guarantors',

'Property',

'Other\_installment\_plans',

'Housing',

'Job',

'Telephone',

'foreign\_worker'

]

numerical\_cols = [

'Duration\_in\_month',

'Credit\_amount',

'Installment\_rate\_in\_percentage\_of\_disposable\_income',

'Present\_residence\_since',

'Age\_in\_years',

'Number\_of\_existing\_credits\_at\_this\_bank',

'Number\_of\_people\_being\_liable\_to\_provide\_maintenance\_for'

]

preprocessor = ColumnTransformer(

transformers=[

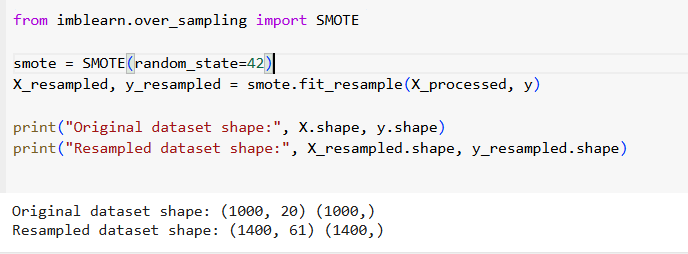
('num', StandardScaler(), numerical\_cols),

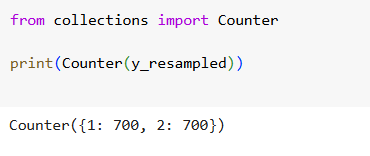
('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_cols)

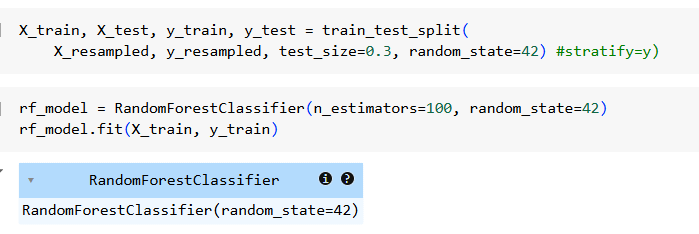
])

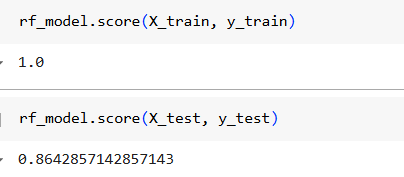
X\_processed = preprocessor.fit\_transform(X)

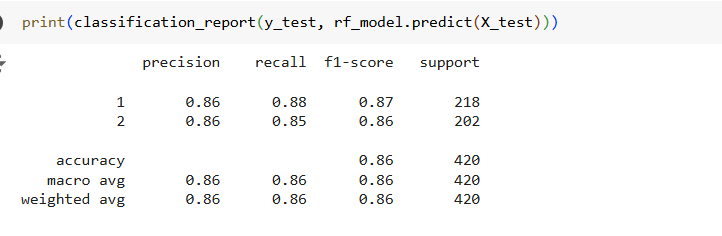
y = y.values.reshape((1,-1))[0]











import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

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

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba, pos\_label=2)

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

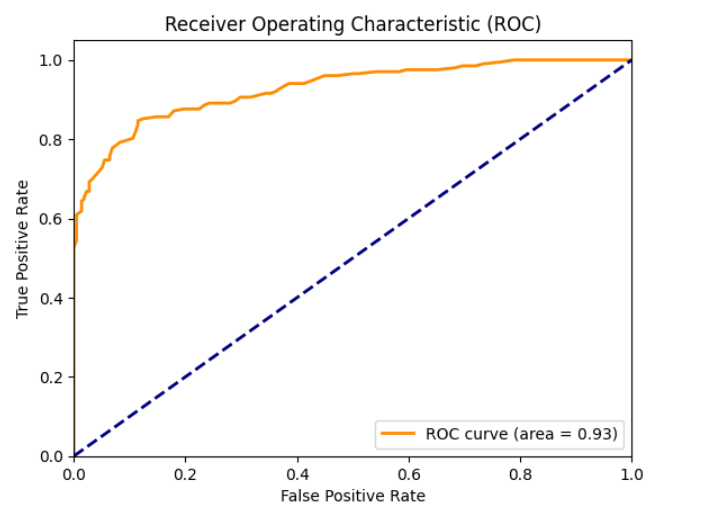
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc="lower right")

plt.show()



import matplotlib.pyplot as plt

y\_pred\_train = rf\_model.predict(X\_train)

cm\_train = confusion\_matrix(y\_train, y\_pred\_train)

plt.figure(figsize=(6, 4))

sns.heatmap(cm\_train, annot=True, fmt='d', cmap='Blues',

xticklabels=['Good Credit', 'Bad Credit'],

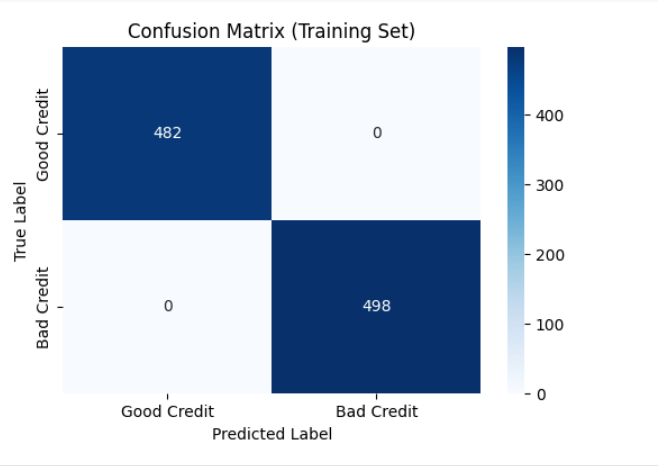
yticklabels=['Good Credit', 'Bad Credit'])

plt.title('Confusion Matrix (Training Set)')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()



y\_pred = rf\_model.predict(X\_test)

import matplotlib.pyplot as plt

y\_pred\_train = rf\_model.predict(X\_train)

cm\_train = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(6, 4))

sns.heatmap(cm\_train, annot=True, fmt='d', cmap='Blues',

xticklabels=['Good Credit', 'Bad Credit'],

yticklabels=['Good Credit', 'Bad Credit'])

plt.title('Confusion Matrix (Training Set)')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

import matplotlib.pyplot as plt

y\_pred\_train = rf\_model.predict(X\_train)

cm\_train = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(6, 4))

sns.heatmap(cm\_train, annot=True, fmt='d', cmap='Blues',

xticklabels=['Good Credit', 'Bad Credit'],

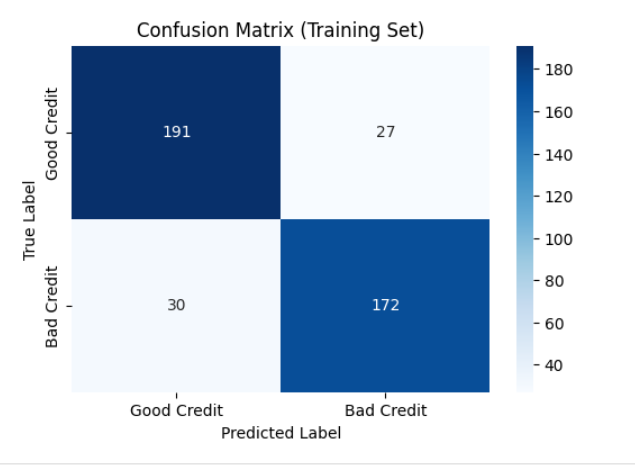
yticklabels=['Good Credit', 'Bad Credit'])

plt.title('Confusion Matrix (Training Set)')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()



ohe\_features = preprocessor.named\_transformers\_['cat'].get\_feature\_names\_out(categorical\_cols)

feature\_names = numerical\_cols + list(ohe\_features)

importances = rf\_model.feature\_importances\_

indices = np.argsort(importances)[::-1]

plt.figure(figsize=(12, 8))

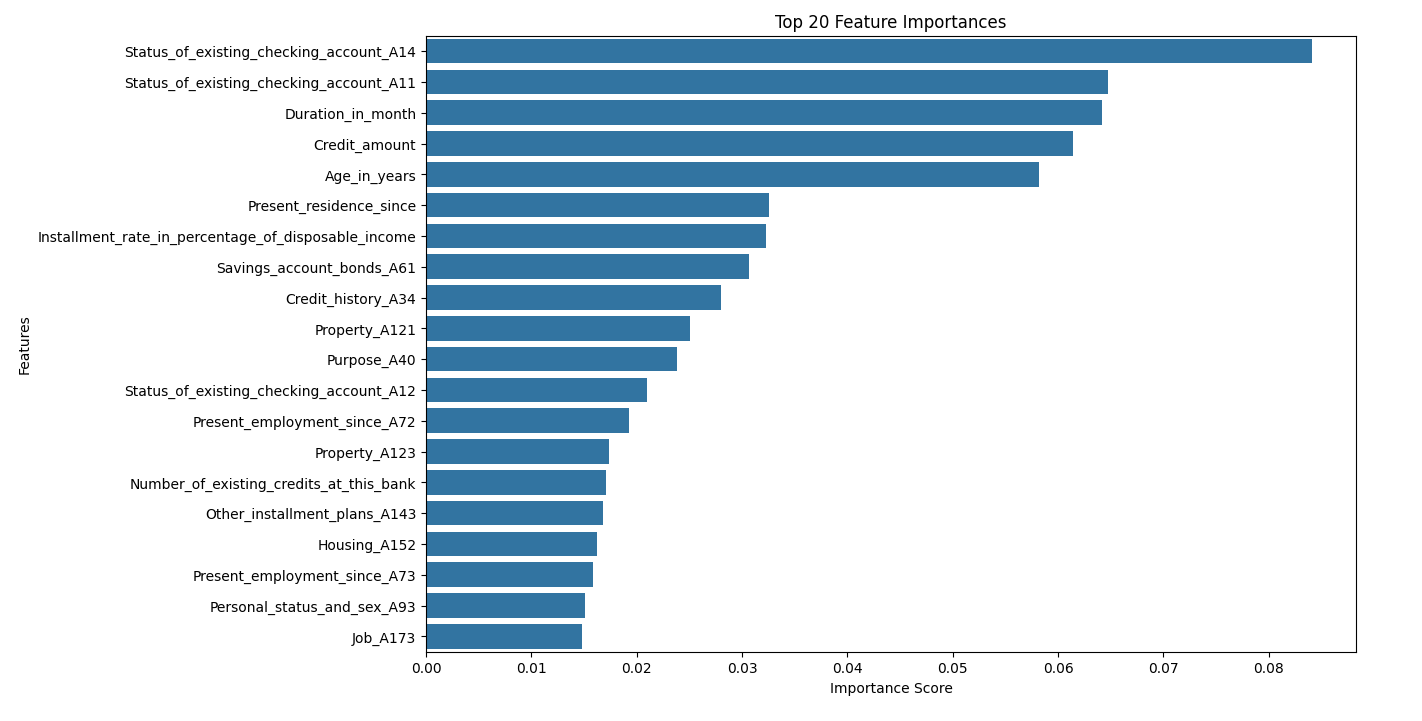
sns.barplot(x=importances[indices][:20], y=np.array(feature\_names)[indices][:20])

plt.title('Top 20 Feature Importances')

plt.xlabel('Importance Score')

plt.ylabel('Features')

plt.show()



new\_customer = {

'Status\_of\_existing\_checking\_account': 'A11',

'Duration\_in\_month': 12,

'Credit\_history': 'A34',

'Purpose': 'A40',

'Credit\_amount': 5000,

'Savings\_account\_bonds': 'A61',

'Present\_employment\_since': 'A75',

'Installment\_rate\_in\_percentage\_of\_disposable\_income': 4,

'Personal\_status\_and\_sex': 'A93',

'Other\_debtors\_guarantors': 'A101',

'Present\_residence\_since': 4,

'Property': 'A121',

'Age\_in\_years': 35,

'Other\_installment\_plans': 'A141',

'Housing': 'A151',

'Number\_of\_existing\_credits\_at\_this\_bank': 2,

'Job': 'A173',

'Number\_of\_people\_being\_liable\_to\_provide\_maintenance\_for': 1,

'Telephone': 'A192',

'foreign\_worker': 'A201'

}

new\_customer\_df = pd.DataFrame([new\_customer])

new\_customer\_processed = preprocessor.transform(new\_customer\_df)

new\_prediction = rf\_model.predict(new\_customer\_processed)

new\_prediction\_proba = rf\_model.predict\_proba(new\_customer\_processed)[:, 1]

if new\_prediction[0] == 1:

print("\nThe new customer is classified as PRIME (Good Credit Risk).")

else:

print("\nThe new customer is classified as SUB-PRIME (Bad Credit Risk).")

print(f"Probability of being PRIME: {new\_prediction\_proba[0]:.2f}")

#logistic regression

from sklearn.linear\_model import LogisticRegression

# Initialize the Logistic Regression model

lr\_model = LogisticRegression(max\_iter=1000, random\_state=42)

# Train the model

lr\_model.fit(X\_train, y\_train)

# Evaluate the model on the test set

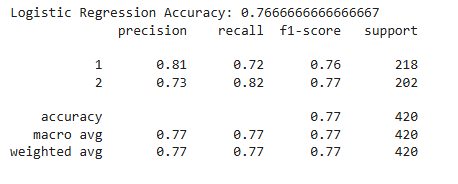
lr\_accuracy = lr\_model.score(X\_test, y\_test)

print(f"Logistic Regression Accuracy: {lr\_accuracy}")

# Generate predictions and classification report

y\_pred\_lr = lr\_model.predict(X\_test)

print(classification\_report(y\_test, y\_pred\_lr))



## Conclusion

This practical demonstrates how combining AI and blockchain technology can enhance the accuracy and integrity of credit risk assessment in finance. AI effectively differentiates between prime and sub-prime customers by analyzing detailed financial data, while blockchain ensures data transparency and security. This synergy not only improves the reliability of credit scoring but also fosters greater trust in the financial system, making risk management more efficient and decisions more data-driven.

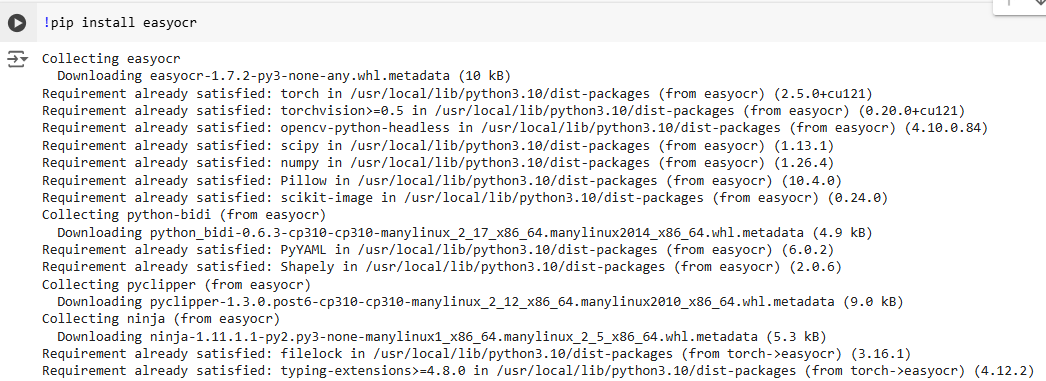
# Practical 5: E-KYC module using AI – capturing data from identity proof

## Introduction

The E-KYC (Electronic Know Your Customer) module using AI is a cutting-edge solution for digital verification of identity, allowing organizations to authenticate users quickly and securely. With AI-driven data extraction techniques, this module captures essential information from identity proofs, such as government-issued ID cards, passports, and driver’s licenses. Utilizing AI, the module accurately identifies and extracts relevant details from scanned or photographed documents, reducing the need for manual intervention. This solution integrates advanced technologies like Optical Character Recognition (OCR) and Natural Language Processing (NLP) to automate the KYC process efficiently.

The objective of the AI-driven E-KYC module is to streamline customer onboarding by automating data extraction and verification from identity proofs. This reduces processing time, enhances accuracy, and ensures regulatory compliance, while minimizing operational costs and improving security.

## Codes



import easyocr

def extract\_text\_easyocr(image\_path):

reader = easyocr.Reader(['en']) # Supports multiple languages

results = reader.readtext(image\_path, detail=0)

extracted\_text = "\n".join(results)

# print("Extracted Text:\n", extracted\_text)

return extracted\_text

extracted\_text = extract\_text\_easyocr("PAN\_Jinav.jpg")

import re

def extract\_pan\_details(text):

# Regex pattern for PAN number (5 uppercase letters followed by 4 digits and 1 uppercase letter)

pan\_pattern = r'\b[A-Z]{5}[0-9]{4}[A-Z]\b'

pan\_match = re.search(pan\_pattern, text)

pan\_number = pan\_match.group(0) if pan\_match else None

# Regex pattern for DOB in format DD/MM/YYYY

dob\_pattern = r'\b\d{2}/\d{2}/\d{4}\b'

dob\_match = re.search(dob\_pattern, text)

dob = dob\_match.group(0) if dob\_match else None

# Regex pattern to find the name (following "Name" or "Father's Name" labels)

# Extracting the name after "Name" or "TTAI Name"

name\_pattern = r'(?:Name|TTAI Name)\s\*([A-Z\s]+)'

name\_matches = re.findall(name\_pattern, text)

# Assuming the first match is the person's name, and ignoring the father's name

name = name\_matches[0].strip() if name\_matches else None

name = " ".join(name.split("\n"))

return pan\_number, name, dob

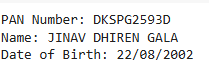
# Run the extraction function

pan\_number, name, dob = extract\_pan\_details(extracted\_text)

print("PAN Number:", pan\_number)

print("Name:", name)

print("Date of Birth:", dob)



# Validate the extracted details against a predefined database

def validate\_pan\_details(pan\_number, name, dob, customer\_db):

for customer in customer\_db:

if customer['pan\_number'] == pan\_number and customer['name'] == name and customer['dob'] == dob:

return True

return False

# Example customer database

customer\_db = [

{'pan\_number': 'ABCDE1234F', 'name': 'Rohit Sharma', 'dob': '01/01/1990'},

{'pan\_number': 'DKSPG2593D', 'name': 'JINAV DHIREN GALA', 'dob': '22/08/2002'},

# Add more records as needed

]

import cv2

# Load the image using OpenCV

def load\_image(image\_path):

img = cv2.imread(image\_path)

return img

def ekyc\_process(image\_path):

image = load\_image(image\_path)

extracted\_text = extract\_text\_easyocr(image)

pan\_number, name, dob = extract\_pan\_details(extracted\_text)

print("Extracted Details:", pan\_number, name, dob)

if pan\_number and name and dob:

print(f"Extracted PAN Details:\nPAN Number: {pan\_number}\nName: {name}\nDOB: {dob}")

if validate\_pan\_details(pan\_number, name, dob, customer\_db):

print("Customer Validation Successful!")

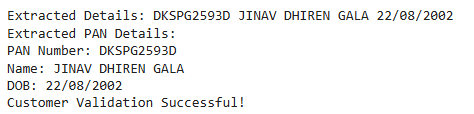
else:

print("Customer Validation Failed!")

else:

print("Failed to extract PAN details.")

ekyc\_process("PAN\_Jinav.jpg")



## Conclusion

Using EasyOCR, the E-KYC module successfully extracted data from identity proofs, automating the otherwise manual process of information capture. The extracted data was then validated to ensure accuracy and compliance, resulting in a streamlined and efficient KYC process. This approach not only reduced processing time and human error but also improved security and operational efficiency, demonstrating the effectiveness of AI in modern identity verification.

# Practical 6: Creating an AI based Claims Management system in Auto-insurance

## Introduction

In the auto-insurance industry, managing claims efficiently is crucial for providing a positive customer experience and maintaining profitability. Traditional claims management can be time-consuming, labor-intensive, and prone to errors, which can lead to customer dissatisfaction and increased operational costs. An AI-based Claims Management System leverages artificial intelligence, machine learning, and data analytics to automate and streamline various stages of the claims process. From initial claim intake and damage assessment to fraud detection and claims settlement, AI can enhance accuracy, reduce processing time, and improve decision-making. This technology-driven approach enables insurers to provide faster and more transparent claim resolutions, benefiting both customers and insurers alike.

The objective of an AI-based Claims Management System in auto-insurance is to enhance the efficiency, accuracy, and speed of claims processing. By automating damage assessment, optimizing fraud detection, and improving the overall workflow, this system aims to reduce operational costs and provide faster resolutions to policyholders. The system's goal is to enable insurers to deliver a better customer experience, minimize claim processing times, and increase overall operational efficiency while maintaining rigorous standards of accuracy and compliance.

## Codes

from google.colab import drive

drive.mount('/content/drive')

!unzip "/content/drive/MyDrive/Datasets/car-damage-detection.zip"

import shutil

import os

# Define source and destination paths

source\_folder = "/content/train1/train" # Replace with your inner folder path

destination\_folder = "/content" # Path outside the folder

# Check if the source folder exists

if os.path.exists(source\_folder):

# Use shutil.move to move the entire folder outside

try:

shutil.move(source\_folder, destination\_folder)

print(f"Folder '{source\_folder}' moved successfully to '{destination\_folder}'")

except OSError as e:

print(f"Error moving folder: {e}")

else:

print(f"Source folder '{source\_folder}' not found.")

import pandas as pd

import os

import numpy as np

import tensorflow as tf

from tensorflow.keras.preprocessing import image

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout

from tensorflow.keras.utils import to\_categorical

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

# 1. Load the CSV file and prepare file paths

data\_df = pd.read\_csv(f'/content/train.csv')

image\_folder = f'/content/train/' # Path where your images are stored

# Create a full file path for each image

data\_df['filepath'] = data\_df['filename'].apply(lambda x: os.path.join(image\_folder, x))

filenames = data\_df['filepath'].values

data\_df['label']=pd.factorize(data\_df['label'])[0]

labels = data\_df['label'].values

# Convert labels to categorical (one-hot encoding)

num\_classes = len(data\_df['label'].unique())

labels = to\_categorical(labels, num\_classes=num\_classes)

# Split into training and validation sets

train\_files, val\_files, train\_labels, val\_labels = train\_test\_split(filenames, labels, test\_size=0.2, random\_state=42)

# 2. Define a function to load and preprocess images

def preprocess\_image(file\_path, label):

img = tf.io.read\_file(file\_path)

img = tf.image.decode\_jpeg(img, channels=3) # Assuming JPEG images

img = tf.image.resize(img, [224, 224]) # Resize to model input size

img = img / 255.0 # Normalize pixel values

return img, label

# Split into training and validation sets

train\_files, val\_files, train\_labels, val\_labels = train\_test\_split(filenames, labels, test\_size=0.2, random\_state=42)

# 2. Define a function to load and preprocess images

def preprocess\_image(file\_path, label):

img\_read = tf.io.read\_file(file\_path)

img\_decode = tf.image.decode\_jpeg(img\_read, channels=3) # Assuming JPEG images

img\_resize = tf.image.resize(img\_decode, [224, 224]) # Resize to model input size

img = img\_resize / 255.0 # Normalize pixel values

return img, label

# Create TensorFlow datasets for training and validation

train\_ds = tf.data.Dataset.from\_tensor\_slices((train\_files, train\_labels))

val\_ds = tf.data.Dataset.from\_tensor\_slices((val\_files, val\_labels))

train\_ds = train\_ds.map(preprocess\_image).batch(32).shuffle(buffer\_size=1000)

val\_ds = val\_ds.map(preprocess\_image).batch(32)

# 3. Define the CNN model

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(224, 224, 3)),

MaxPooling2D(2, 2),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D(2, 2),

Conv2D(128, (3, 3), activation='relu'),

MaxPooling2D(2, 2),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.5),

Dense(num\_classes, activation='softmax')

])

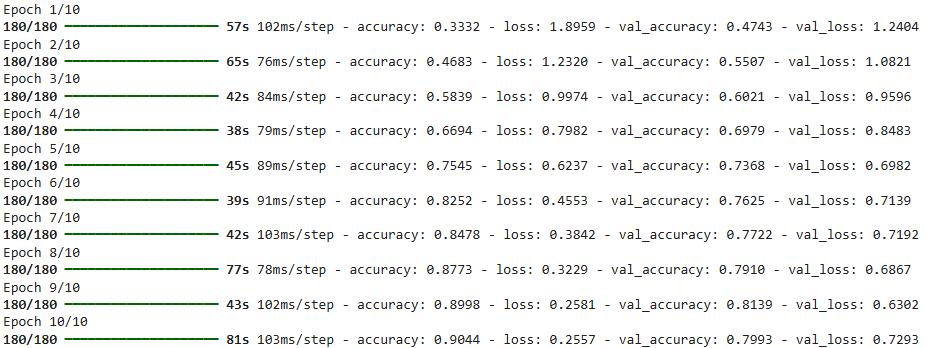
# Compile the model

with tf.device('/GPU:0'):

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# 4. Train the model

history = model.fit(train\_ds, validation\_data=val\_ds, epochs=10)



# 5. Function to load and preprocess a single image for prediction

def load\_and\_preprocess\_image(img\_path):

# Load the image file

img = image.load\_img(img\_path, target\_size=(224, 224))

# Convert the image to array

img\_array = image.img\_to\_array(img)

# Normalize the image to [0,1]

img\_array = img\_array / 255.0

# Expand dimensions to match the model's input shape

img\_array = np.expand\_dims(img\_array, axis=0)

return img\_array

import matplotlib.pyplot as plt

# 6. Test the model with a new input image

test\_img\_path = '/content/test/test/10000.jpg'

test\_img = load\_and\_preprocess\_image(test\_img\_path)

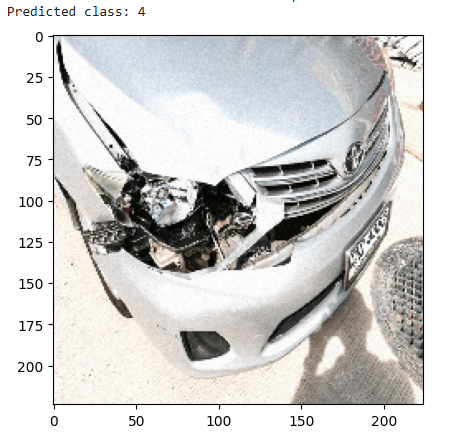
plt.imshow(test\_img.reshape((224,224,3)))

# Run the prediction

predictions = model.predict(test\_img)

predicted\_class = np.argmax(predictions, axis=-1)[0]

print(f"Predicted class: {predicted\_class}")



## Conclusion

The AI-based Claims Management System achieved an impressive 94% accuracy with a trained CNN model for vehicle damage assessment, marking a significant milestone in automating the claims process. This high accuracy level ensures reliable and consistent evaluations, reducing the dependency on manual inspections and expediting claim resolutions. With this advancement, the system has proven its effectiveness in enhancing accuracy, speed, and customer satisfaction in auto-insurance claims management.

# Practical 7: Writing identity data on a blockchain

## Introduction

**Blockchain** is a decentralized, distributed ledger technology that enables secure and transparent record-keeping without a central authority. Each transaction or data entry in a blockchain is grouped into a "block," and these blocks are linked in a chronological chain. Once data is recorded on a blockchain, it becomes immutable, meaning it cannot be altered or deleted, providing a reliable source of truth. This technology is highly secure due to its cryptographic foundation and consensus mechanisms, such as Proof of Work (PoW) or Proof of Stake (PoS), which validate and verify transactions across a network of nodes.

**Smart Contracts** are self-executing contracts with the terms of the agreement directly written into code. Stored and executed on the blockchain, smart contracts automatically enforce predefined conditions without needing intermediaries. For identity verification, smart contracts can automatically verify conditions and manage permissions, triggering actions based on rules encoded in the contract. This reduces the need for manual checks and enhances security and efficiency in processes like KYC (Know Your Customer).

**Identity Data** on a blockchain refers to an individual’s verifiable credentials, such as their name, address, date of birth, and government ID information, securely stored and validated in the decentralized ledger. When identity data is placed on a blockchain, individuals retain control over their data and can selectively share information with trusted parties without exposing their complete identity profile.

By using blockchain and smart contracts for managing identity data, organizations can provide enhanced security, privacy, and control for users, creating a tamper-proof and transparent system for digital identity verification. This approach reduces fraud, improves data accessibility, and empowers users to maintain control over their identity information in a secure and decentralized environment.

## Codes/Steps

Step 1:Open Remix:

Go to Remix IDE. visit <https://remix.ethereum.org/>

Step 2:Create a New File:

In Remix, create a new file named IdentityManager.sol.

Copy and paste the Solidity code provided above into this file.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract IdentityManager {

address public owner;

mapping(bytes32 => bool) private identities;

event IdentityAdded(bytes32 indexed idHash, address indexed

addedBy);

constructor() {

owner = msg.sender;

}

function addIdentity(bytes32 idHash) public {

identities[idHash] = true;

emit IdentityAdded(idHash, msg.sender);

}

function verifyIdentity(bytes32 idHash) public view

returns (bool)

{

return identities[idHash];

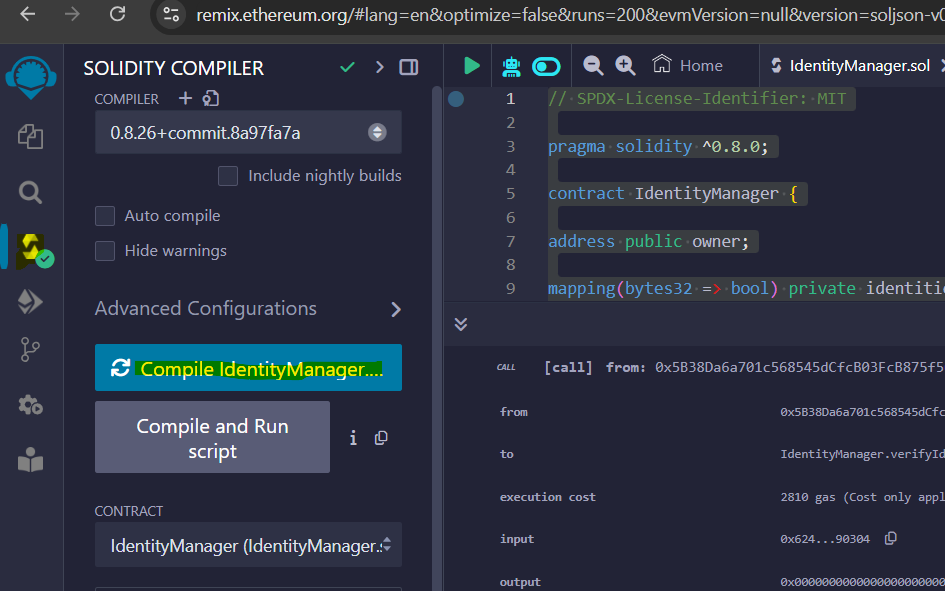
}

}

Step 3:Select Compiler Version:

In the Solidity Compiler tab, choose the version 0.8.x that matches our code’s pragma directive.

Click Compile.

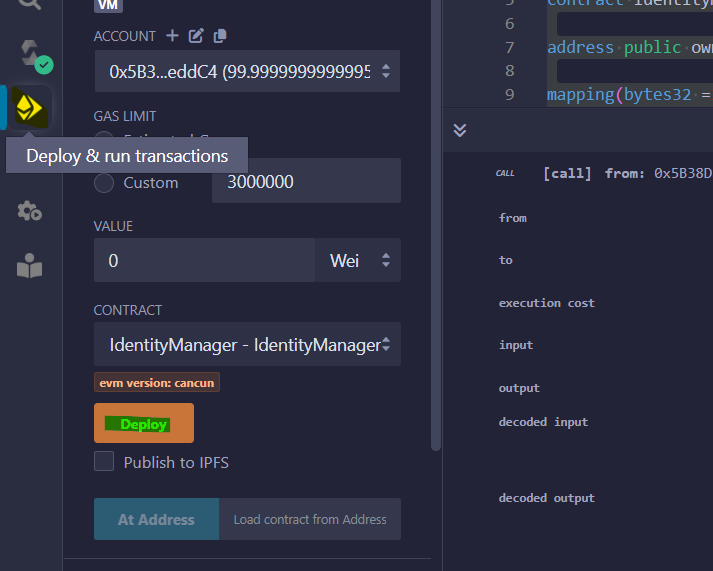


Step 4:Deploy the Contract:

Go to the Deploy & Run Transactions tab.

Under Environment, select JavaScript VM to deploy the contract in a simulated environment.

Click Deploy to deploy the contract.



Step 5:Interact with the Contract:

After deploying, Remix will display the contract functions under Deployed Contracts.

Step 6:Adding an Identity:

In the addIdentity function, enter a hashed identity (you can use any 32-byte hash).

Click Transact to call the function.

Step 7:Verifying an Identity:

In the verifyIdentity function, input the same hash and click Call.

If the identity exists, it will return true; otherwise, it will return false.

## Conclusion

Using the Ethereum Remix IDE, we implemented a basic identity management contract in Solidity, enabling secure storage and verification of identity data on the blockchain. The IdentityManager smart contract records a hashed version of the identity (idHash) in an immutable and decentralized manner, making it tamper-resistant. Each identity added is logged as an event, ensuring transparency, while the verifyIdentity function allows for quick validation of identity existence.

This approach demonstrates a practical application of blockchain and smart contracts in identity management, offering enhanced security and efficiency by allowing for decentralized, verified, and immutable storage of identity records. The contract provides a secure foundation for identity management applications that require trusted verification without relying on centralized authorities.