import pandas as pd

import random

from faker import Faker

from datetime import datetime, timedelta

fake = Faker()

#1. Transaction Data Simulation

def generate\_transaction\_data(num\_transactions, customers\_df, accounts\_df):

transactions = []

channels = ['online', 'mobile', 'ATM', 'in-branch']

transaction\_types = ['purchase', 'transfer', 'withdrawal', 'deposit']

currencies = ['USD', 'EUR', 'GBP', 'JPY']

merchants = ['Amazon', 'Walmart', 'Target', 'Best Buy', 'Costco']

categories = ['Retail', 'Grocery', 'Electronics', 'Clothing', 'Miscellaneous']

for \_ in range(num\_transactions):

customer = customers\_df.sample(1).iloc[0]

account = accounts\_df[accounts\_df['customer\_id'] == customer['customer\_id']].sample(1).iloc[0]

transaction\_id = fake.uuid4()

customer\_id = customer['customer\_id']

transaction\_date = fake.date\_time\_between(start\_date='-2y', end\_date='now')

amount = round(random.uniform(1.0, 10000.0), 2)

currency = random.choice(currencies)

transaction\_type = random.choice(transaction\_types)

channel = random.choice(channels)

merchant\_name = random.choice(merchants)

merchant\_category = random.choice(categories)

location\_country = fake.country()

location\_city = fake.city()

is\_flagged = fake.boolean(chance\_of\_getting\_true=5) # 5% chance of being flagged

transactions.append({

'transaction\_id': transaction\_id,

'customer\_id': customer\_id,

'transaction\_date': transaction\_date,

'amount': amount,

'currency': currency,

'transaction\_type': transaction\_type,

'channel': channel,

'merchant\_name': merchant\_name,

'merchant\_category': merchant\_category,

'location\_country': location\_country,

'location\_city': location\_city,

'is\_flagged': is\_flagged

})

return pd.DataFrame(transactions)

#2. Customer Data Simulation

def generate\_customer\_since(start\_year=2022):

start\_date = datetime(start\_year, 1, 1)

end\_date = datetime.now()

random\_date = start\_date + (end\_date - start\_date) \* random.random()

return random\_date

def generate\_customers(num\_customers):

customers = []

for \_ in range(num\_customers):

customer\_id = fake.uuid4()

first\_name = fake.first\_name()

last\_name = fake.last\_name()

date\_of\_birth = fake.date\_of\_birth(minimum\_age=18, maximum\_age=90)

gender = random.choice(['Male', 'Female'])

email = fake.email()

phone\_number = fake.phone\_number()

address = fake.address()

city = fake.city()

country = fake.country()

occupation = fake.job()

income\_bracket = random.choice(['Low', 'Medium', 'High'])

customer\_since = generate\_customer\_since()

customers.append({

'customer\_id': customer\_id,

'first\_name': first\_name,

'last\_name': last\_name,

'date\_of\_birth': date\_of\_birth,

'gender': gender,

'email': email,

'phone\_number': phone\_number,

'address': address,

'city': city,

'country': country,

'occupation': occupation,

'income\_bracket': income\_bracket,

'customer\_since': customer\_since

})

return pd.DataFrame(customers)

#3. Account Data Simulation

def generate\_accounts(customers\_df):

accounts = []

account\_types = ['checking', 'savings', 'credit card', 'loan']

account\_statuses = ['active', 'dormant', 'closed']

currencies = ['USD', 'EUR', 'GBP', 'JPY']

for \_, customer in customers\_df.iterrows():

for \_ in range(random.randint(1, 3)): # Each customer can have 1 to 3 accounts

account\_id = fake.uuid4()

customer\_id = customer['customer\_id']

account\_type = random.choice(account\_types)

account\_status = random.choice(account\_statuses)

open\_date = fake.date\_between(start\_date=customer['customer\_since'])

current\_balance = round(random.uniform(0.0, 100000.0), 2)

currency = random.choice(currencies)

credit\_limit = round(random.uniform(1000.0, 50000.0), 2) if account\_type == 'credit card' else 0.0

accounts.append({

'account\_id': account\_id,

'customer\_id': customer\_id,

'account\_type': account\_type,

'account\_status': account\_status,

'open\_date': open\_date,

'current\_balance': current\_balance,

'currency': currency,

'credit\_limit': credit\_limit

})

return pd.DataFrame(accounts)

#4. External Data Simulation - Credit Bureau Data

def generate\_credit\_data(customers\_df):

credit\_data = []

for \_, customer in customers\_df.iterrows():

customer\_id = customer['customer\_id']

credit\_score = random.randint(300, 850)

number\_of\_credit\_accounts = random.randint(0, 10)

total\_credit\_limit = round(random.uniform(1000.0, 50000.0), 2)

total\_credit\_used = round(random.uniform(0.0, total\_credit\_limit), 2)

number\_of\_late\_payments = random.randint(0, 5)

bankruptcies = random.randint(0, 2)

credit\_data.append({

'customer\_id': customer\_id,

'credit\_score': credit\_score,

'number\_of\_credit\_accounts': number\_of\_credit\_accounts,

'total\_credit\_limit': total\_credit\_limit,

'total\_credit\_used': total\_credit\_used,

'number\_of\_late\_payments': number\_of\_late\_payments,

'bankruptcies': bankruptcies

})

return pd.DataFrame(credit\_data)

# Watchlist Data

def generate\_watchlist\_data(num\_entries):

watchlist = []

risk\_categories = ['Low', 'Medium', 'High']

entity\_types = ['Individual', 'Organization']

sources = ['OFAC', 'EU', 'UN', 'Local Authorities']

for \_ in range(num\_entries):

entity\_id = fake.uuid4()

entity\_name = fake.company() if random.choice([True, False]) else fake.name()

entity\_type = random.choice(entity\_types)

risk\_category = random.choice(risk\_categories)

listed\_date = fake.date\_this\_decade()

source = random.choice(sources)

watchlist.append({

'entity\_id': entity\_id,

'entity\_name': entity\_name,

'entity\_type': entity\_type,

'risk\_category': risk\_category,

'listed\_date': listed\_date,

'source': source

})

return pd.DataFrame(watchlist)

# Main script

def main():

# Generate customer data

customers\_df = generate\_customers(1000)

customers\_df.to\_csv('customers.csv', index=False)

# Generate account data

accounts\_df = generate\_accounts(customers\_df)

accounts\_df.to\_csv('accounts.csv', index=False)

# Generate transaction data

transactions\_df = generate\_transaction\_data(10000, customers\_df, accounts\_df)

transactions\_df.to\_csv('transactions.csv', index=False)

# Generate credit bureau data

credit\_data\_df = generate\_credit\_data(customers\_df)

credit\_data\_df.to\_csv('credit\_data.csv', index=False)

# Generate watchlist data

watchlist\_df = generate\_watchlist\_data(1000)

watchlist\_df.to\_csv('watchlist.csv', index=False)

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

main()

import pandas as pd

import random

from faker import Faker

fake = Faker()

# Country to currency and city mapping

country\_currency\_mapping = {

'United States': {'currency': 'USD', 'cities': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix']},

'India': {'currency': 'INR', 'cities': ['Delhi', 'Mumbai', 'Hyderabad', 'Chennai', 'Bangalore']},

'United Kingdom': {'currency': 'GBP', 'cities': ['London', 'Birmingham', 'Manchester', 'Glasgow', 'Liverpool']},

'Japan': {'currency': 'JPY', 'cities': ['Tokyo', 'Osaka', 'Nagoya', 'Sapporo', 'Fukuoka']},

'Germany': {'currency': 'EUR', 'cities': ['Berlin', 'Hamburg', 'Munich', 'Cologne', 'Frankfurt']}

}

# Transaction data generation

def generate\_transaction\_data(num\_transactions, customers\_df, accounts\_df):

transactions = []

channels = ['online', 'mobile', 'ATM', 'in-branch']

transaction\_types = ['purchase', 'transfer', 'withdrawal', 'deposit']

merchants = ['Amazon', 'Walmart', 'Target', 'Best Buy', 'Costco']

categories = ['Retail', 'Grocery', 'Electronics', 'Clothing', 'Miscellaneous']

for \_ in range(num\_transactions):

customer = customers\_df.sample(1).iloc[0]

account = accounts\_df[accounts\_df['customer\_id'] == customer['customer\_id']].sample(1).iloc[0]

transaction\_id = fake.uuid4()

customer\_id = customer['customer\_id']

transaction\_date = fake.date\_time\_between(start\_date='-2y', end\_date='now')

amount = round(random.uniform(1.0, 10000.0), 2)

country\_info = country\_currency\_mapping.get(customer['country'], {'currency': 'USD', 'cities': ['New York']})

currency = country\_info['currency']

transaction\_type = random.choice(transaction\_types)

channel = random.choice(channels)

merchant\_name = random.choice(merchants)

merchant\_category = random.choice(categories)

location\_country = customer['country']

location\_city = random.choice(country\_info['cities'])

is\_flagged = fake.boolean(chance\_of\_getting\_true=5) # 5% chance of being flagged

transactions.append({

'transaction\_id': transaction\_id,

'customer\_id': customer\_id,

'transaction\_date': transaction\_date,

'amount': amount,

'currency': currency,

'transaction\_type': transaction\_type,

'channel': channel,

'merchant\_name': merchant\_name,

'merchant\_category': merchant\_category,

'location\_country': location\_country,

'location\_city': location\_city,

'is\_flagged': is\_flagged

})

return pd.DataFrame(transactions)

# Customer data generation

def generate\_customers(num\_customers):

customers = []

countries = list(country\_currency\_mapping.keys())

for \_ in range(num\_customers):

customer\_id = fake.uuid4()

first\_name = fake.first\_name()

last\_name = fake.last\_name()

date\_of\_birth = fake.date\_of\_birth(minimum\_age=18, maximum\_age=90)

gender = random.choice(['Male', 'Female'])

email = fake.email()

phone\_number = fake.phone\_number()

address = fake.street\_address()

country = random.choice(countries)

city = random.choice(country\_currency\_mapping[country]['cities'])

occupation = fake.job()

income\_bracket = random.choice(['Low', 'Medium', 'High'])

customer\_since = fake.date\_between(start\_date='-10y', end\_date='now')

customers.append({

'customer\_id': customer\_id,

'first\_name': first\_name,

'last\_name': last\_name,

'date\_of\_birth': date\_of\_birth,

'gender': gender,

'email': email,

'phone\_number': phone\_number,

'address': address,

'city': city,

'country': country,

'occupation': occupation,

'income\_bracket': income\_bracket,

'customer\_since': customer\_since

})

return pd.DataFrame(customers)

# Account data generation

def generate\_accounts(customers\_df):

accounts = []

account\_types = ['checking', 'savings', 'credit card', 'loan']

account\_statuses = ['active', 'dormant', 'closed']

for \_, customer in customers\_df.iterrows():

for \_ in range(random.randint(1, 3)): # Each customer can have 1 to 3 accounts

account\_id = fake.uuid4()

customer\_id = customer['customer\_id']

account\_type = random.choice(account\_types)

account\_status = random.choice(account\_statuses)

open\_date = fake.date\_between(start\_date=customer['customer\_since'])

current\_balance = round(random.uniform(0.0, 100000.0), 2)

currency = country\_currency\_mapping.get(customer['country'], {'currency': 'USD'})['currency']

credit\_limit = round(random.uniform(1000.0, 50000.0), 2) if account\_type == 'credit card' else 0.0

accounts.append({

'account\_id': account\_id,

'customer\_id': customer\_id,

'account\_type': account\_type,

'account\_status': account\_status,

'open\_date': open\_date,

'current\_balance': current\_balance,

'currency': currency,

'credit\_limit': credit\_limit

})

return pd.DataFrame(accounts)

# Credit data generation

def generate\_credit\_data(customers\_df):

credit\_data = []

for \_, customer in customers\_df.iterrows():

credit\_score = random.randint(300, 850)

number\_of\_credit\_accounts = random.randint(1, 10)

total\_credit\_limit = round(random.uniform(1000.0, 50000.0), 2)

total\_credit\_used = round(random.uniform(0.0, total\_credit\_limit), 2)

number\_of\_late\_payments = random.randint(0, 5)

bankruptcies = random.randint(0, 1)

credit\_data.append({

'customer\_id': customer['customer\_id'],

'credit\_score': credit\_score,

'number\_of\_credit\_accounts': number\_of\_credit\_accounts,

'total\_credit\_limit': total\_credit\_limit,

'total\_credit\_used': total\_credit\_used,

'number\_of\_late\_payments': number\_of\_late\_payments,

'bankruptcies': bankruptcies

})

return pd.DataFrame(credit\_data)

# Watchlist data generation

def generate\_watchlist\_data(num\_entities):

watchlist = []

entity\_types = ['Individual', 'Organization']

risk\_categories = ['Low', 'Medium', 'High']

sources = ['OFAC', 'UN', 'EU', 'Interpol']

for \_ in range(num\_entities):

entity\_id = fake.uuid4()

entity\_name = fake.name() if random.choice(entity\_types) == 'Individual' else fake.company()

entity\_type = random.choice(entity\_types)

risk\_category = random.choice(risk\_categories)

listed\_date = fake.date\_between(start\_date='-10y', end\_date='now')

source = random.choice(sources)

watchlist.append({

'entity\_id': entity\_id,

'entity\_name': entity\_name,

'entity\_type': entity\_type,

'risk\_category': risk\_category,

'listed\_date': listed\_date,

'source': source

})

return pd.DataFrame(watchlist)

# Generate all data

def generate\_data(num\_customers, num\_transactions, num\_watchlist\_entities):

customers\_df = generate\_customers(num\_customers)

accounts\_df = generate\_accounts(customers\_df)

transactions\_df = generate\_transaction\_data(num\_transactions, customers\_df, accounts\_df)

credit\_data\_df = generate\_credit\_data(customers\_df)

watchlist\_data\_df = generate\_watchlist\_data(num\_watchlist\_entities)

return customers\_df, accounts\_df, transactions\_df, credit\_data\_df, watchlist\_data\_df

# Save data to CSV

# def save\_data\_to\_csv(customers\_df, accounts\_df, transactions\_df, credit\_data\_df, watchlist\_data\_df):

# customers\_df.to\_csv('customers.csv', index=False)

# accounts\_df.to\_csv('accounts.csv', index=False)

# transactions\_df.to\_csv('transactions.csv', index=False)

# credit\_data\_df.to\_csv('credit\_data.csv', index=False)

# watchlist\_data\_df.to\_csv('watchlist\_data.csv', index=False)

import os

def save\_data\_to\_csv(customers\_df, accounts\_df, transactions\_df, credit\_data\_df, watchlist\_data\_df):

download\_path = r'C:\Users\sunilsh\Desktop\Capstone'

customers\_df.to\_csv(os.path.join(download\_path, 'customers.csv'), index=False)

accounts\_df.to\_csv(os.path.join(download\_path, 'accounts.csv'), index=False)

transactions\_df.to\_csv(os.path.join(download\_path, 'transactions.csv'), index=False)

credit\_data\_df.to\_csv(os.path.join(download\_path, 'credit\_data.csv'), index=False)

watchlist\_data\_df.to\_csv(os.path.join(download\_path, 'watchlist\_data.csv'), index=False)

# Main function

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

num\_customers = 1000

num\_transactions = 10000

num\_watchlist\_entities = 100

customers\_df, accounts\_df, transactions\_df, credit\_data\_df, watchlist\_data\_df = generate\_data(num\_customers, num\_transactions, num\_watchlist\_entities)

save\_data\_to\_csv(customers\_df, accounts\_df, transactions\_df, credit\_data\_df, watchlist\_data\_df)

# Read CSV files

customers\_df = pd.read\_csv('customers.csv')

accounts\_df = pd.read\_csv('accounts.csv')

transactions\_df = pd.read\_csv('transactions.csv')

credit\_data\_df = pd.read\_csv('credit\_data.csv')

watchlist\_df = pd.read\_csv('watchlist.csv')

import boto3

from boto3.s3.transfer import S3Transfer

import os

pip install boto3

def upload\_files(file\_paths, bucket\_name, folder, region\_name='us-east-1'):

"""

Upload multiple files to an S3 bucket.

Parameters:

- file\_paths (list): List of paths to the local files.

- bucket\_name (str): The name of the S3 bucket.

- folder (str): The folder in the S3 bucket where the files will be uploaded.

- region\_name (str): The AWS region where the S3 bucket is located.

"""

client = boto3.client(

's3',

aws\_access\_key\_id='AKIAU6GD3NDD4YM2N5WA',

aws\_secret\_access\_key='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',

region\_name='us-east-1' # Fixed: region name should be a string

)

transfer = S3Transfer(client)

for file\_path in file\_paths:

file\_name = os.path.basename(file\_path)

s3\_key = folder + file\_name

try:

transfer.upload\_file(file\_path, bucket\_name, s3\_key)

print(f'{file\_path} uploaded to s3://{bucket\_name}/{s3\_key}')

except Exception as e:

print(f'Error uploading {file\_path}: {e}')

# List of files to upload

files\_to\_upload = [

r"C:\Users\sunilsh\Desktop\Capstone\transactions.csv",

r"C:\Users\sunilsh\Desktop\Capstone\accounts.csv",

r"C:\Users\sunilsh\Desktop\Capstone\customers.csv",

r"C:\Users\sunilsh\Desktop\Capstone\credit\_data.csv",

r"C:\Users\sunilsh\Desktop\Capstone\watchlist\_data.csv"

]

# Call the function to upload multiple files

upload\_files(files\_to\_upload, 'captrail3', 'capfolder3/', 'us-east-1')