**Project Overview: Combining Data, Training ML Models, and Integrating Kafka and Spark for Real-Time Prediction**

**1. Combining Multiple CSV Files into One**

The first step in our project is combining three separate CSV files into one consolidated file. This is done using pandas, which provides efficient data manipulation tools. The combined file is then saved as final\_risk\_factor.csv.

python

CopyEdit

import pandas as pd

# Read the three CSV files

df1 = pd.read\_csv('/path/to/bcsc\_risk\_factors\_summarized1\_092020.csv')

df2 = pd.read\_csv('/path/to/bcsc\_risk\_factors\_summarized2\_092020.csv')

df3 = pd.read\_csv('/path/to/bcsc\_risk\_factors\_summarized3\_092020.csv')

# Concatenate the DataFrames

df = pd.concat([df1, df2, df3], ignore\_index=True)

# Check for missing values

print(df.isnull().any())

# Drop unnecessary columns

df = df.drop(columns=['count', 'year'])

# Save the final dataset

df.to\_csv('final\_risk\_factor.csv', index=False)

print("CSV files combined and saved as 'final\_risk\_factor.csv'.")

**2. Training a Bloom Filter to Identify Unique Records**

A Bloom filter is a space-efficient probabilistic data structure that helps identify whether an element is a member of a set. Here, we use the Bloom filter to check for the uniqueness of records based on specific columns from the dataset.

python

CopyEdit

from pybloom\_live import BloomFilter

import pickle

import pandas as pd

# Load the final CSV data

df = pd.read\_csv('final\_risk\_factor.csv')

# Initialize the Bloom filter

bloom\_filter = BloomFilter(capacity=13000000, error\_rate=0.01)

# Function to generate a unique key for each row based on certain columns

def generate\_key(row):

return f"{row['age\_group\_5\_years']}\_{row['race\_eth']}\_{row['first\_degree\_hx']}\_{row['age\_menarche']}\_{row['age\_first\_birth']}\_{row['BIRADS\_breast\_density']}\_{row['current\_hrt']}\_{row['menopaus']}\_{row['bmi\_group']}\_{row['biophx']}"

# Add unique records to the Bloom filter

for \_, row in df.iterrows():

if row['breast\_cancer\_history'] == 1: # Only consider records where breast\_cancer\_history is 1

unique\_val = generate\_key(row)

bloom\_filter.add(unique\_val)

# Save the trained Bloom filter

with open("bloom\_filter.pkl", "wb") as f:

pickle.dump(bloom\_filter, f)

print("✅ Bloom filter trained and saved!")

**3. Training Multiple Machine Learning Models**

We train three different machine learning models: Logistic Regression, Random Forest, and XGBoost, using the final\_risk\_factor.csv dataset. The models are trained to predict the presence of breast cancer based on the available risk factors.

python

CopyEdit

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

import joblib

from xgboost import XGBClassifier

# Load the final dataset

df = pd.read\_csv('final\_risk\_factor.csv')

# Define features and target

X = df.drop(columns=['breast\_cancer\_history'])

y = df['breast\_cancer\_history']

# Split the data into training and testing sets

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

# Logistic Regression

lr = LogisticRegression()

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

lr\_pred = lr.predict(X\_test)

lr\_acc = accuracy\_score(lr\_pred, y\_test)

print("Logistic Regression Accuracy:", lr\_acc)

# Random Forest

rf = RandomForestClassifier()

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

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

rf\_acc = accuracy\_score(rf\_pred, y\_test)

print("Random Forest Accuracy:", rf\_acc)

# XGBoost

y\_train = y\_train.replace(9, 2)

y\_test = y\_test.replace(9, 2)

xgboost = XGBClassifier(use\_label\_encoder=False)

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

xgboost\_pred = xgboost.predict(X\_test)

xgboost\_acc = accuracy\_score(xgboost\_pred, y\_test)

print("XGBoost Accuracy:", xgboost\_acc)

# Save the trained model

joblib.dump(xgboost, 'model.pkl')

**4. Integrating Kafka for Real-Time Data Streaming**

We use Kafka to stream data in real time. The generate\_data() function creates random data, and the generate\_data\_2() function processes only those records that have a breast\_cancer\_history of 1.

python

CopyEdit

import random

import json

from kafka import KafkaProducer

import pandas as pd

# Kafka producer setup

producer = KafkaProducer(

bootstrap\_servers='localhost:9092',

value\_serializer=lambda v: json.dumps(v).encode('utf-8')

)

# Generate random data

def generate\_data():

return {

"age\_group\_5\_years": random.randint(1, 13),

"race\_eth": random.choice([1, 2, 3, 4, 5, 6, 9]),

"first\_degree\_hx": random.choice([0, 1, 9]),

"age\_menarche": random.choice([0, 1, 2, 9]),

"age\_first\_birth": random.choice([0, 1, 2, 3, 4, 9]),

"BIRADS\_breast\_density": random.choice([1, 2, 3, 4, 9]),

"current\_hrt": random.choice([0, 1, 9]),

"menopaus": random.choice([1, 2, 3, 9]),

"bmi\_group": random.choice([1, 2, 3, 4, 9]),

"biophx": random.choice([0, 1, 9]),

}

# Load the final risk factor dataset

df = pd.read\_csv('final\_risk\_factor.csv')

# Generate data based on actual records with breast cancer history

def generate\_data\_2():

for \_, rows in df.iterrows():

if rows['breast\_cancer\_history'] == 1:

yield {

"age\_group\_5\_years": int(rows['age\_group\_5\_years']),

"race\_eth": int(rows['race\_eth']),

"first\_degree\_hx": int(rows['first\_degree\_hx']),

"age\_menarche": int(rows['age\_menarche']),

"age\_first\_birth": int(rows['age\_first\_birth']),

"BIRADS\_breast\_density": int(rows['BIRADS\_breast\_density']),

"current\_hrt": int(rows['current\_hrt']),

"menopaus": int(rows['menopaus']),

"bmi\_group": int(rows['bmi\_group']),

"biophx": int(rows['biophx'])

}

# Send data to Kafka

data = generate\_data\_2()

for record in data:

producer.send('ehr\_risk\_topic', record)

print(f"Sent data: {record}")

producer.flush()

**5. Integrating Spark for Real-Time Predictions**

In this step, we integrate Spark to consume the real-time data from Kafka and apply the trained Bloom filter and ML model for predictions.

python

CopyEdit

from pyspark.sql import SparkSession

import pickle

import joblib

import json

from pyspark.sql.functions import udf

from pyspark.sql.types import IntegerType

from pyspark.sql.functions import col

# Load the Bloom filter and ML model

with open('bloom\_filter.pkl', 'rb') as f:

filter = pickle.load(f)

model = joblib.load('model.pkl')

# Initialize Spark session

spark = SparkSession.builder.appName('EHR\_RISK\_DATA\_BF\_ML').config(

'spark.jars.packages', 'org.apache.spark:spark-sql-kafka-0-10\_2.12:3.5.5').getOrCreate()

# Read Kafka data stream

df = spark.readStream.format('kafka').option('kafka.bootstrap.servers', 'localhost:9092').option('subscribe', 'ehr\_risk\_topic').load()

df = df.selectExpr('CAST(value AS STRING)')

# Define UDF to apply Bloom filter and ML model

def bloom\_ml\_model(value):

rows = json.loads(value)

unique\_val = f"{rows['age\_group\_5\_years']}\_{rows['race\_eth']}\_{rows['first\_degree\_hx']}\_{rows['age\_menarche']}\_{rows['age\_first\_birth']}\_{rows['BIRADS\_breast\_density']}\_{rows['current\_hrt']}\_{rows['menopaus']}\_{rows['bmi\_group']}\_{rows['biophx']}"

if unique\_val not in filter:

return 0

features = [

rows['age\_group\_5\_years'], rows['race\_eth'], rows['first\_degree\_hx'],

rows['age\_menarche'], rows['age\_first\_birth'], rows['BIRADS\_breast\_density'],

rows['current\_hrt'], rows['menopaus'], rows['bmi\_group'], rows['biophx']

]

prediction = model.predict([features])

return int(prediction[0])

# Register UDF

predicted\_udf = udf(bloom\_ml\_model, IntegerType())

# Apply UDF to the Kafka data

df = df.withColumn('final\_prediction', predicted\_udf(col("value")))

# Write the result to the console

query = df.select("value", "final\_prediction") \

.writeStream \

.outputMode("append") \

.format("console") \

.option("truncate", "false") \

.option("numRows", 10000).start()

query.awaitTermination()