import pandas as pd

import hashlib

from sklearn.cluster import KMeans

import numpy as np

def hash\_identifier(identifier):

return hashlib.sha256(identifier.encode()).hexdigest()

def generate\_synthetic\_data(original\_data, num\_samples):

synthetic\_data = []

for \_ in range(num\_samples):

sample = original\_data.sample(n=1)

noise = np.random.normal(0, 0.1, sample.shape)

synthetic\_sample = sample + noise

synthetic\_data.append(synthetic\_sample)

return pd.concat(synthetic\_data, ignore\_index=True)

# 加载数据

publisher\_data = pd.read\_csv('publisher\_data.csv')

advertiser\_data = pd.read\_csv('advertiser\_data.csv')

# 对用户标识符进行哈希处理

publisher\_data['user\_id'] = publisher\_data['user\_id'].apply(hash\_identifier)

advertiser\_data['user\_id'] = advertiser\_data['user\_id'].apply(hash\_identifier)

# 合并数据集

merged\_data = pd.merge(publisher\_data, advertiser\_data, on='user\_id', how='inner')

# 聚类分析

features = merged\_data[['feature1', 'feature2', 'feature3']]

kmeans = KMeans(n\_clusters=3)

kmeans.fit(features)

merged\_data['cluster'] = kmeans.labels\_

# 生成合成数据

synthetic\_data = generate\_synthetic\_data(merged\_data, num\_samples=100)

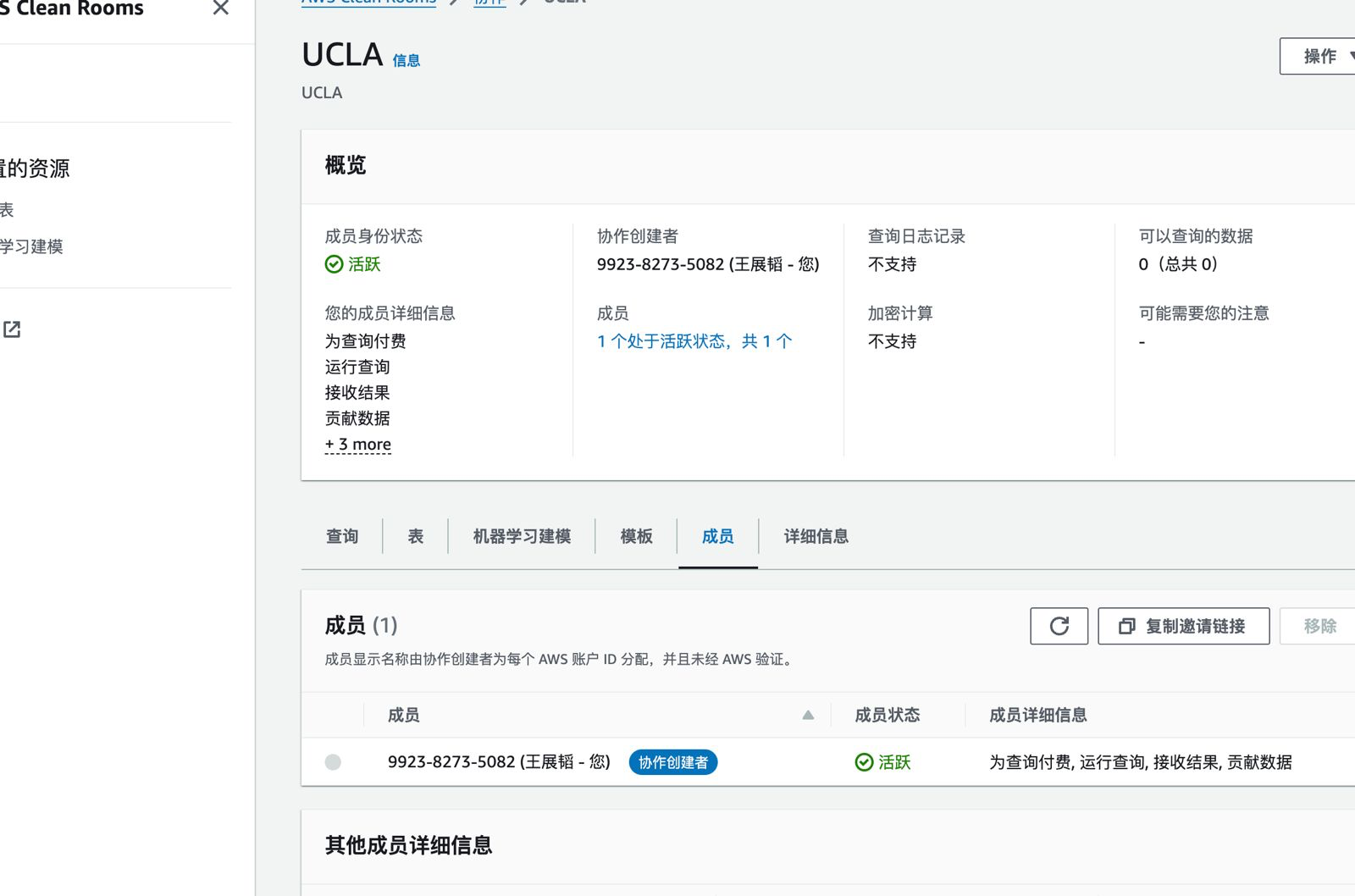
# 保存并分析合成数据

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

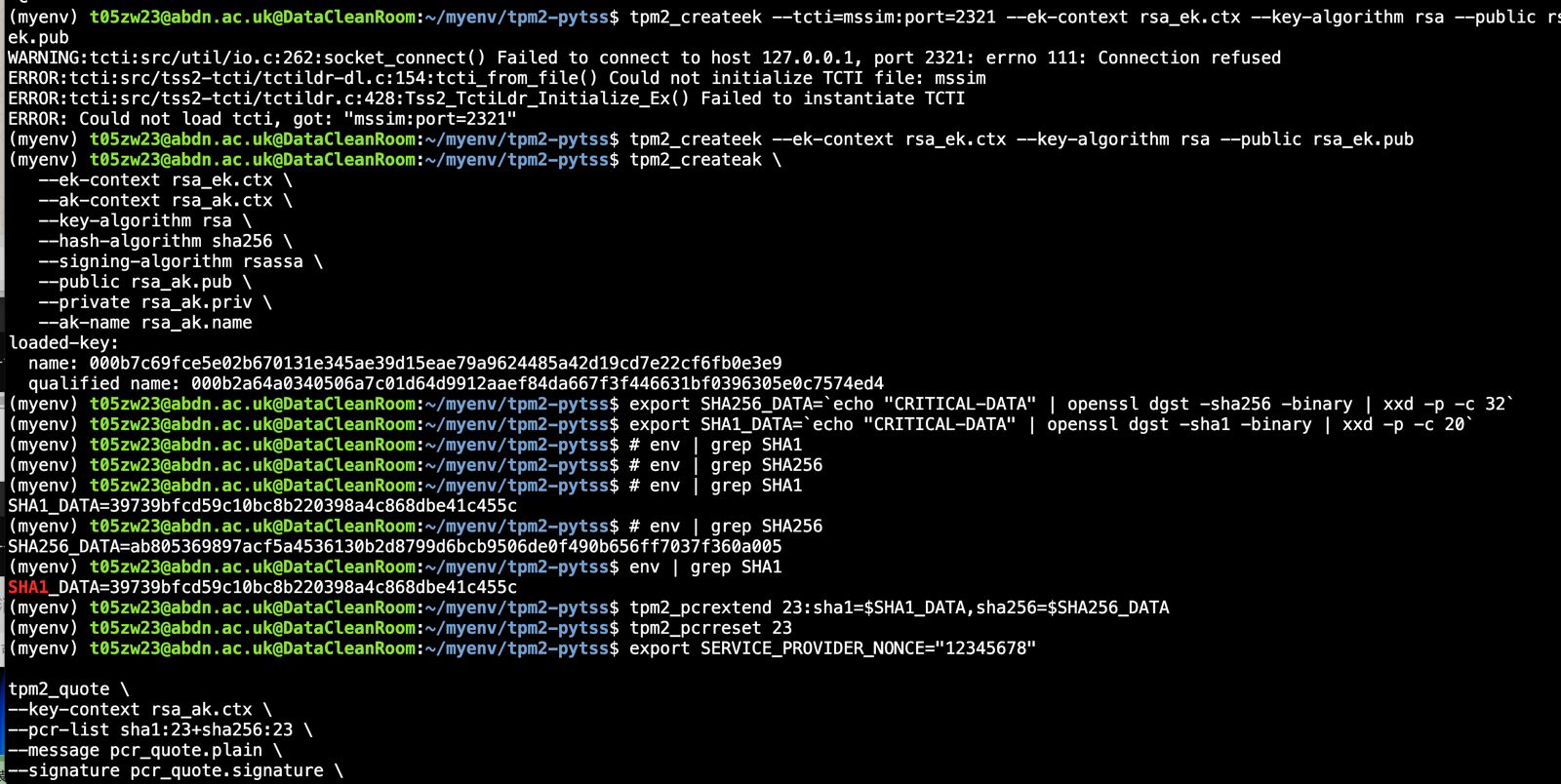
summary = synthetic\_data.describe()

print(summary)

# AWS data clean room



# NPM:





# Javascript:

const fetch = require('node-fetch');

const fs = require('fs');

const csv = require('csv-parser');

const crypto = require('crypto');

const { KMeans } = require('sklearn');

const tf = require('@tensorflow/tfjs-node');

// 哈希函数

function hashIdentifier(identifier) {

return crypto.createHash('sha256').update(identifier).digest('hex');

}

// 读取CSV文件并处理数据

async function loadData(url) {

try {

const response = await fetch(url);

const data = [];

await new Promise((resolve, reject) => {

response.body.pipe(csv())

.on('data', (row) => {

// 对用户标识符进行哈希处理

row.user\_id = hashIdentifier(row.user\_id);

data.push(row);

})

.on('end', () => {

console.log('CSV file successfully processed');

resolve();

})

.on('error', reject);

});

return data;

} catch (error) {

console.error('Error loading data:', error);

throw error;

}

}

// 生成合成数据

function generateSyntheticData(originalData, numSamples) {

const syntheticData = [];

for (let i = 0; i < numSamples; i++) {

const sample = originalData[Math.floor(Math.random() \* originalData.length)];

const noise = sample.map(value => value + Math.random() \* 0.1 - 0.05);

syntheticData.push(noise);

}

return syntheticData;

}

// 创建和训练模型

async function trainModel(data) {

const model = tf.sequential();

model.add(tf.layers.dense({ units: 64, activation: 'relu', inputShape: [data[0].length - 1] }));

model.add(tf.layers.dense({ units: 32, activation: 'relu' }));

model.add(tf.layers.dense({ units: 1, activation: 'sigmoid' }));

model.compile({ optimizer: 'adam', loss: 'binaryCrossentropy', metrics: ['accuracy'] });

const xs = tf.tensor2d(data.map(d => d.slice(0, -1)));

const ys = tf.tensor2d(data.map(d => d[d.length - 1]), [data.length, 1]);

await model.fit(xs, ys, { epochs: 10 });

return model;

}

// 主函数

async function main() {

// 确保使用实际的Azure Blob Storage URL

const publisherDataUrl = 'https://myrealstorage.blob.core.windows.net/myrealcontainer/train\_data\_feeds.csv';

const advertiserDataUrl = 'https://myrealstorage.blob.core.windows.net/myrealcontainer/train\_data\_ads.csv';

try {

const publisherData = await loadData(publisherDataUrl);

const advertiserData = await loadData(advertiserDataUrl);

// 合并数据集

const mergedData = publisherData.map((pubRow, index) => ({

...pubRow,

...advertiserData[index]

}));

// 生成合成数据

const syntheticData = generateSyntheticData(mergedData, 100);

// 保存合成数据

fs.writeFileSync('synthetic\_data.json', JSON.stringify(syntheticData, null, 2));

console.log('Synthetic data generated and saved to synthetic\_data.json');

// 训练模型

const model = await trainModel(syntheticData);

console.log('Model trained successfully');

} catch (error) {

console.error('An error occurred:', error);

}

}

main().catch(error => {

console.error('Unhandled error in main:', error);

});

