Python and Lambda

Work with RXNORM file,

1. Add header into each rff from RXNORM.xlsx
2. Add CODE\_SET & VERSION\_MONTH column with default values RxNorm and version month from downloaded filename
3. Convert dates into YYYY-MM-DD
4. Save files as txt delimited by comma(,)
5. Validate row\_count between original and converted files

Steps:

1)- we have to create 2 buckets one for data source bucket(input) and another is destination bucket (output)

2) create lambda function

3) ADD LAYER in lambda for pandas

4) edit general configuration for lambda

5)added the python code solution for our problem statement and deploy it

6)then upload excel header file in folder of data source bucket

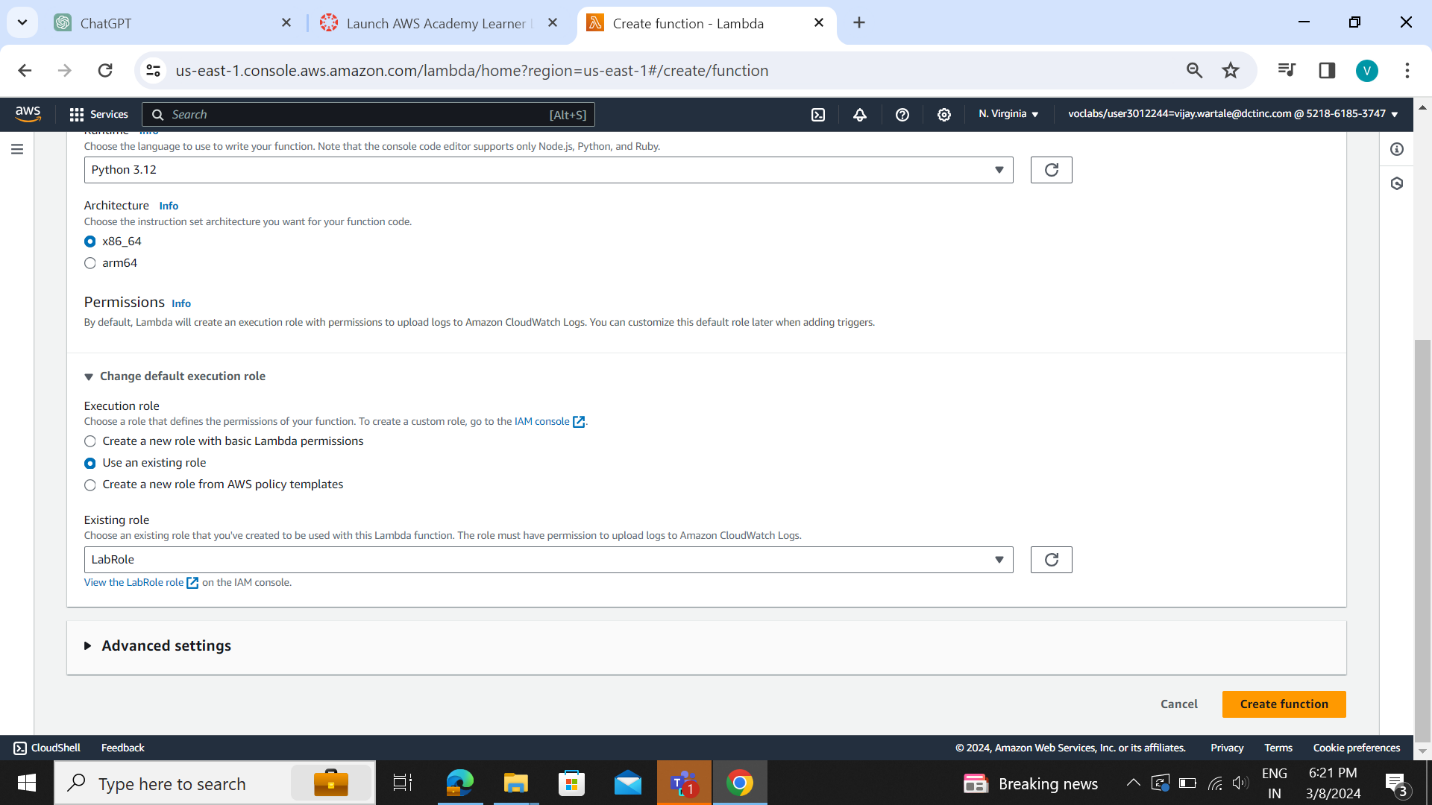
7)upload rrf.zip file in data source bucket

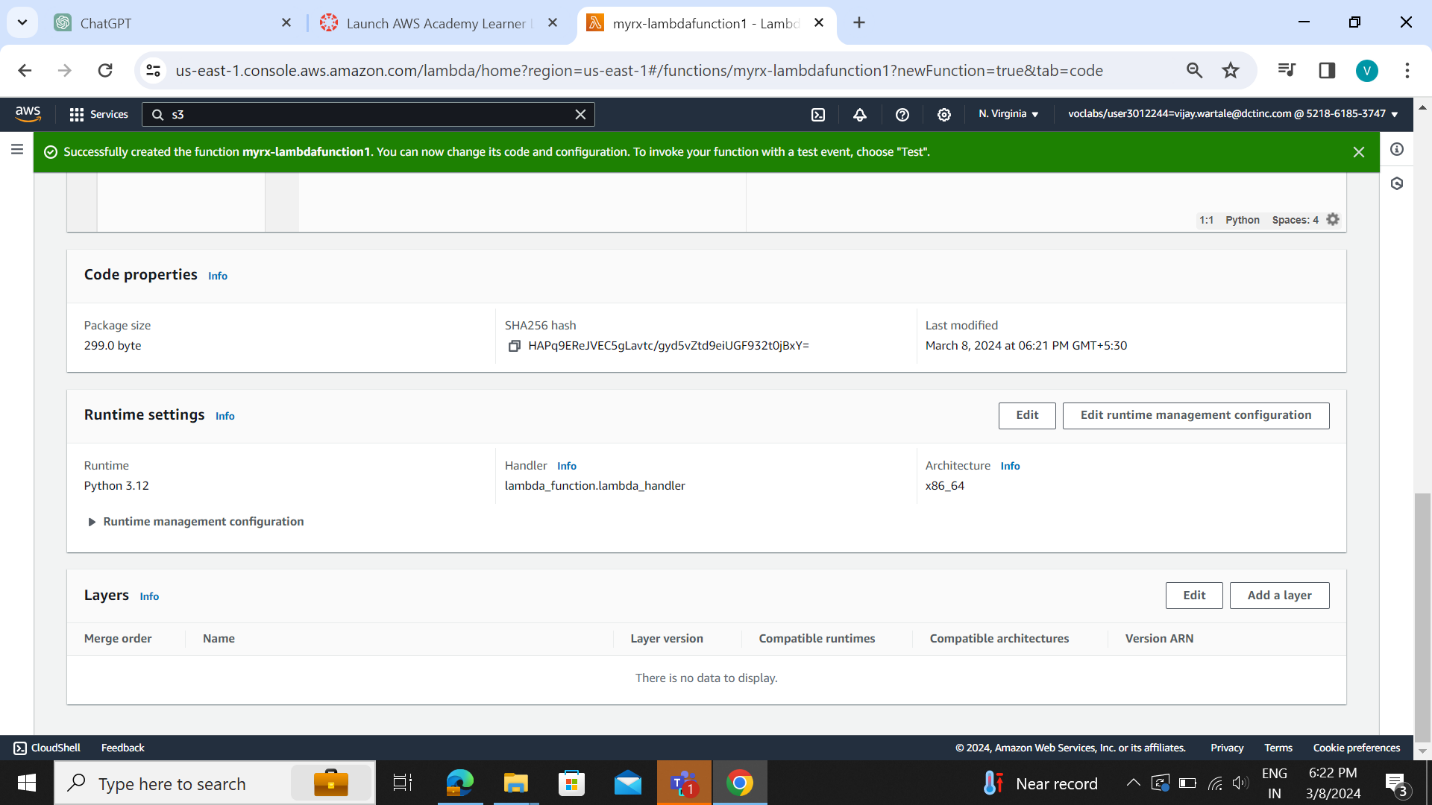
8) then we check the logs to see whether lambda function was triggerd and to see if there was any error and rectified it

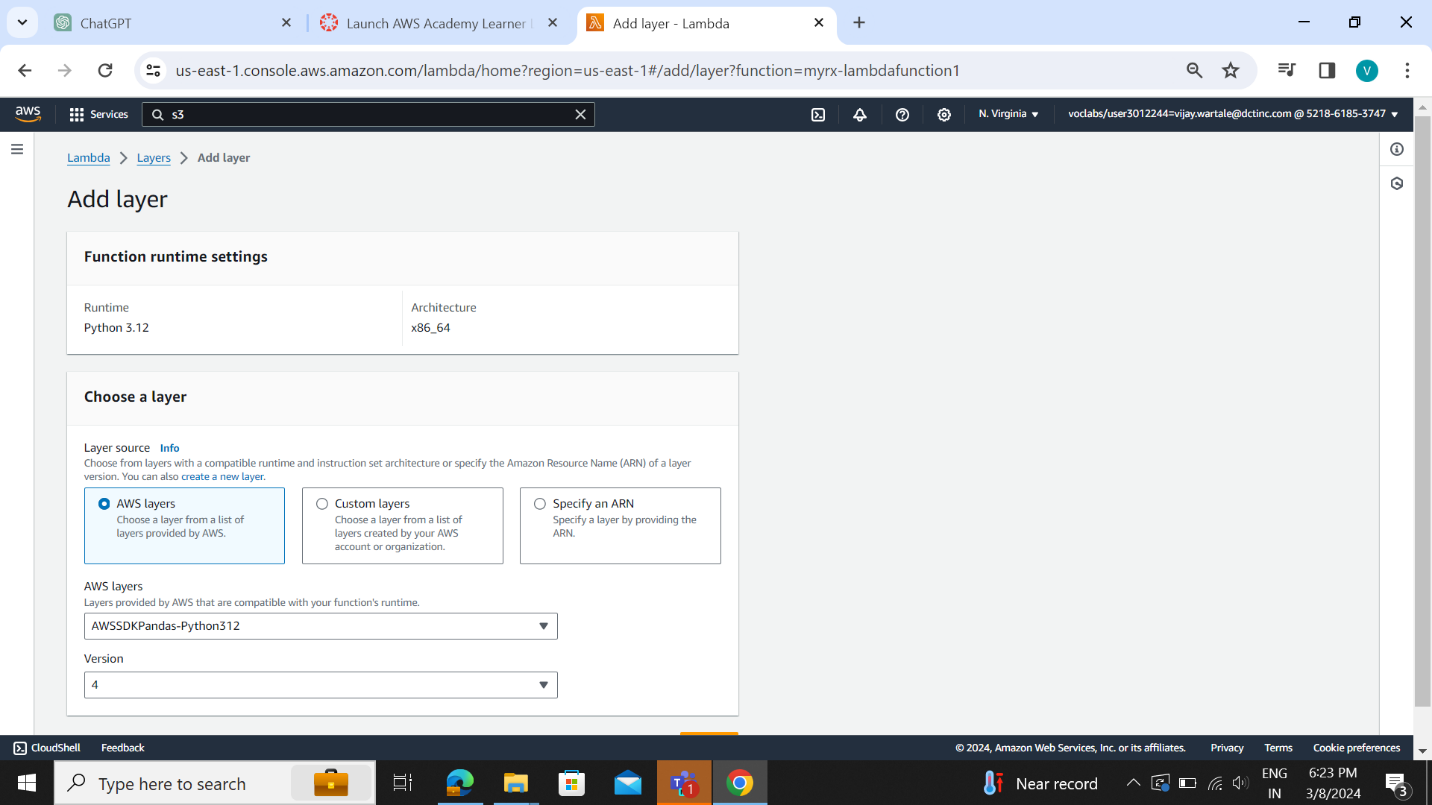
8) then we will get desired output

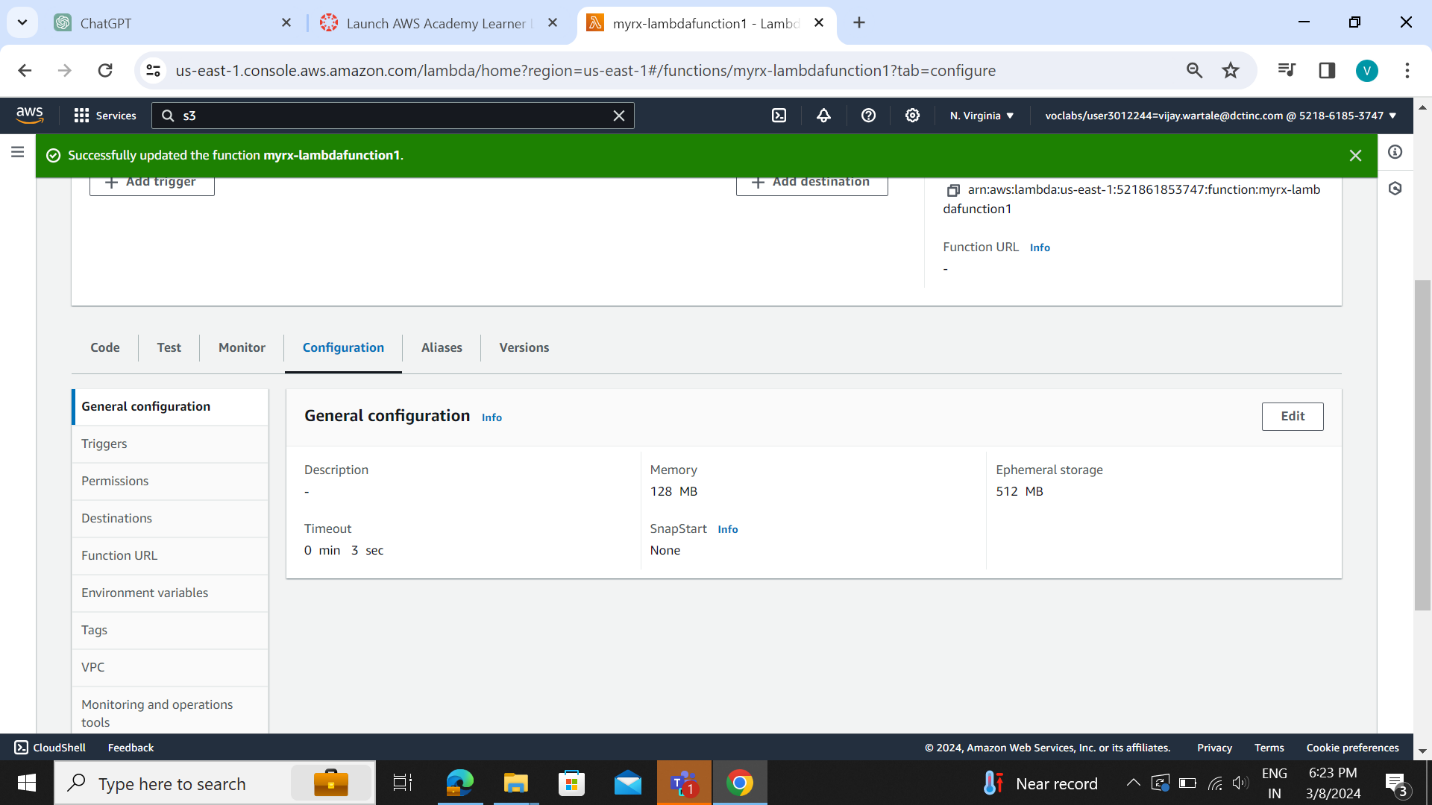
A screenshot of a computer

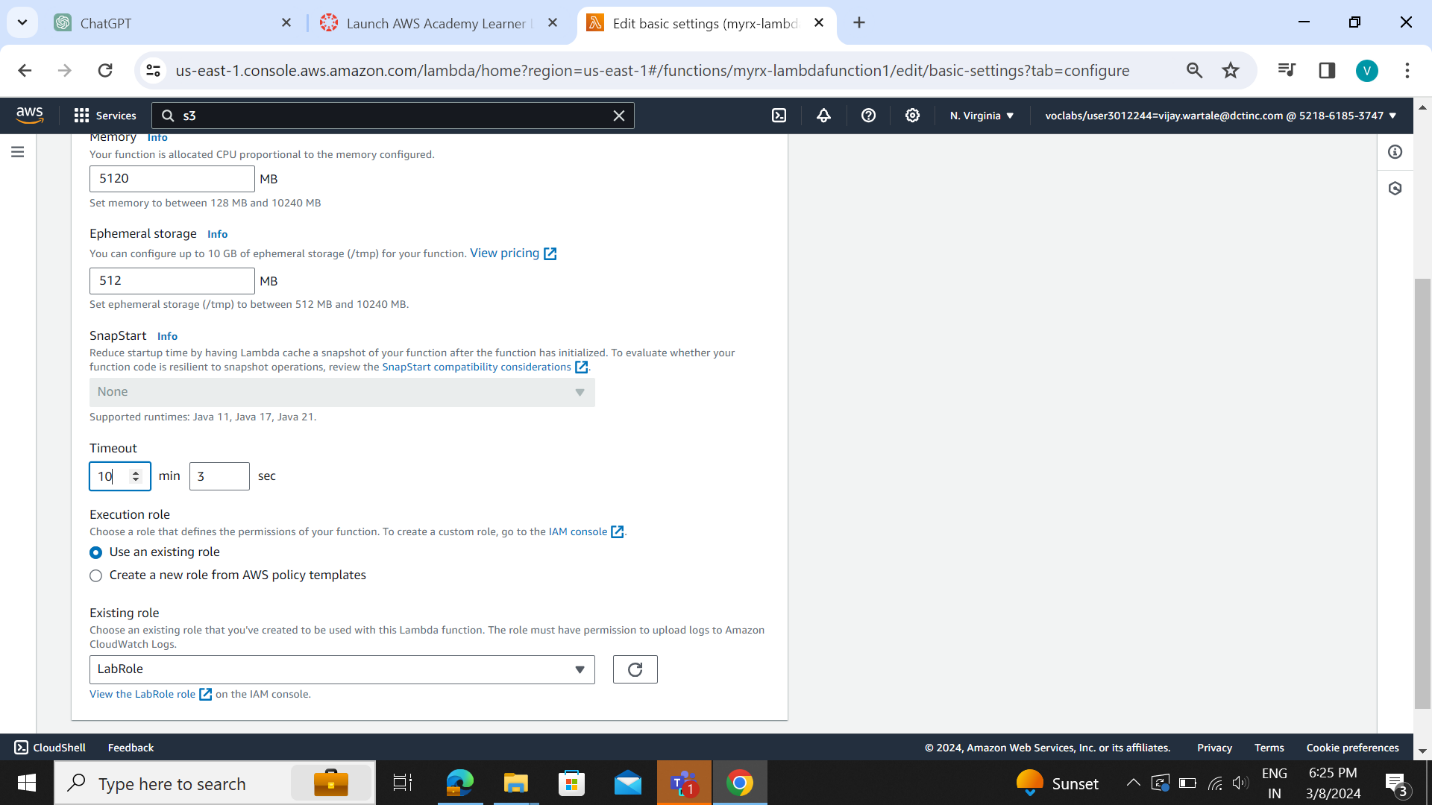
Description automatically generated

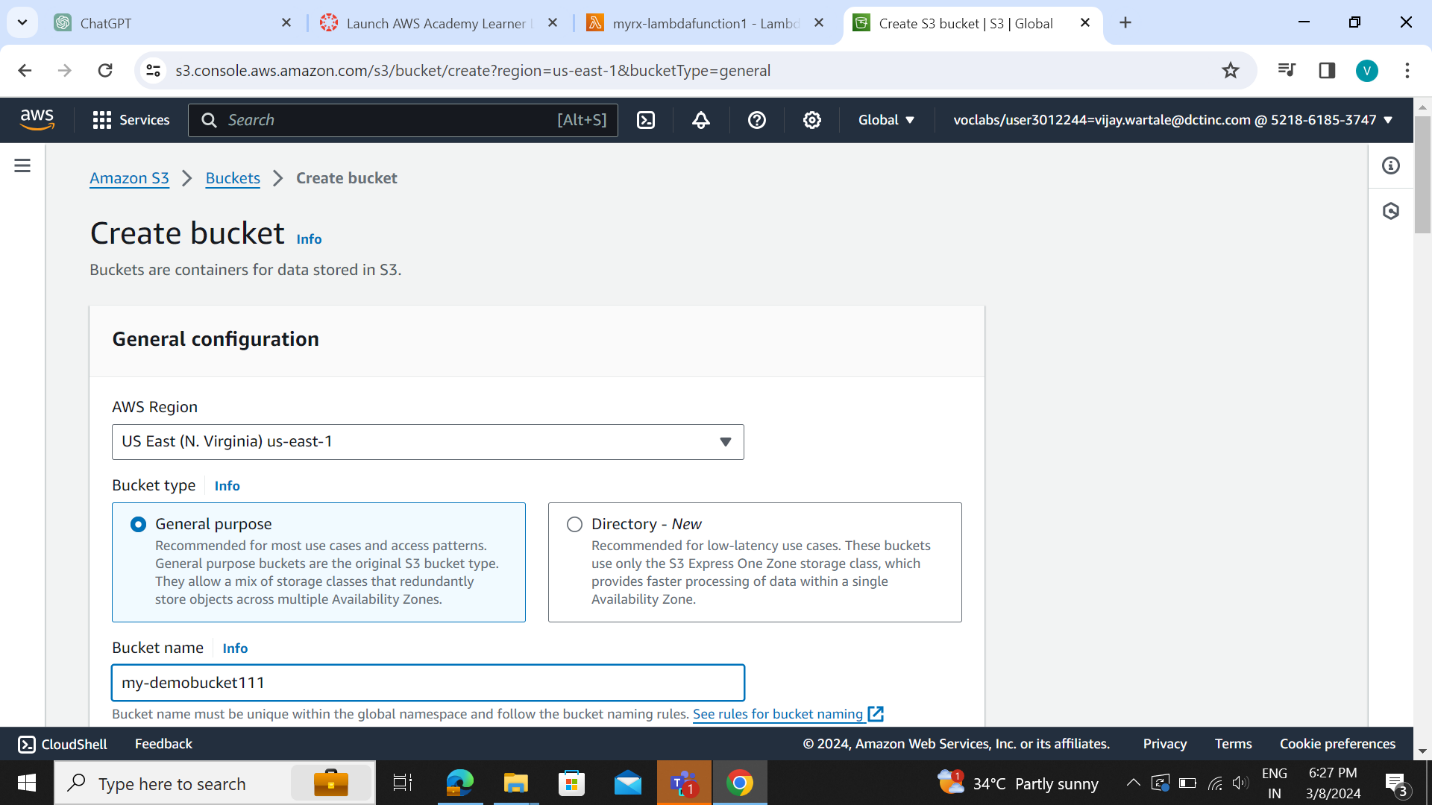


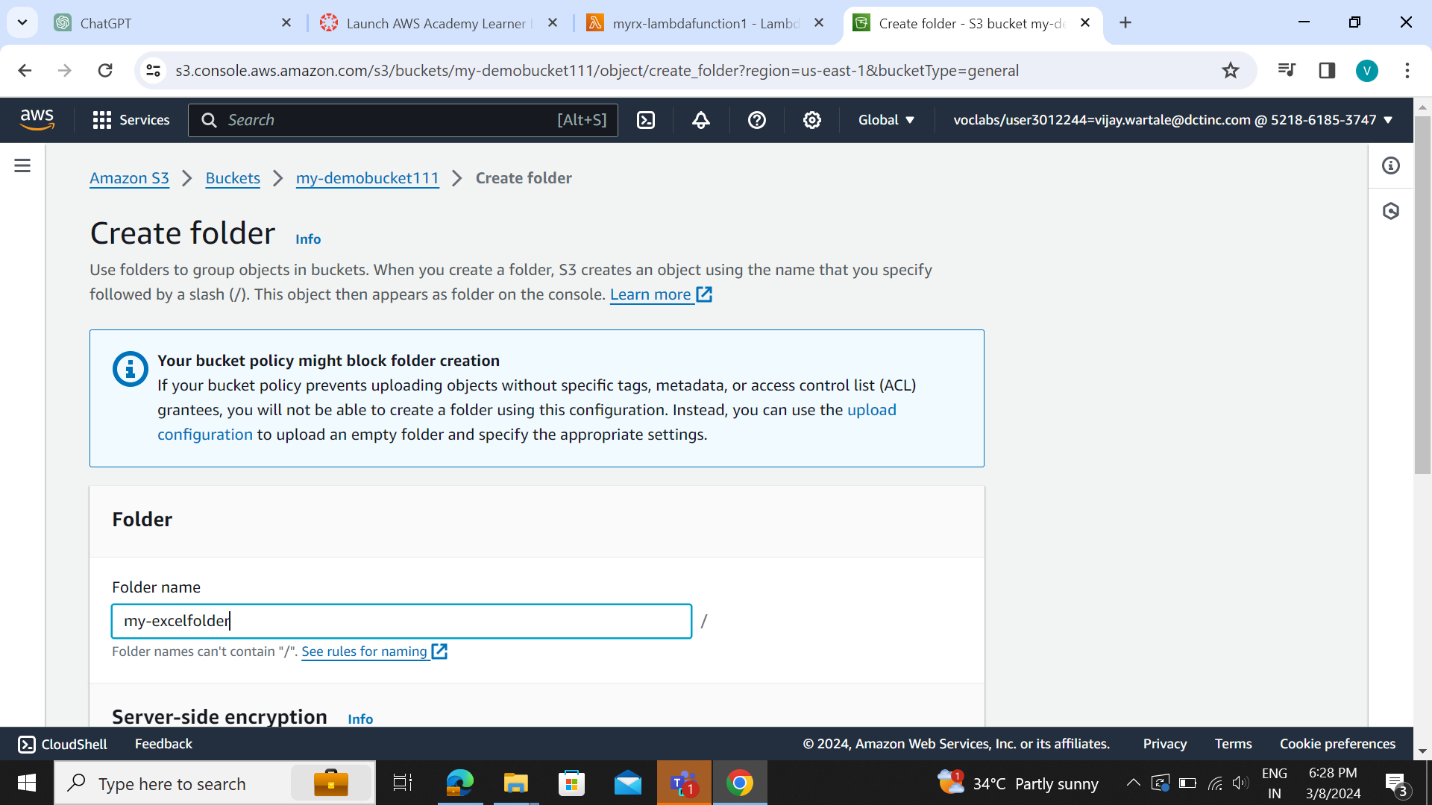


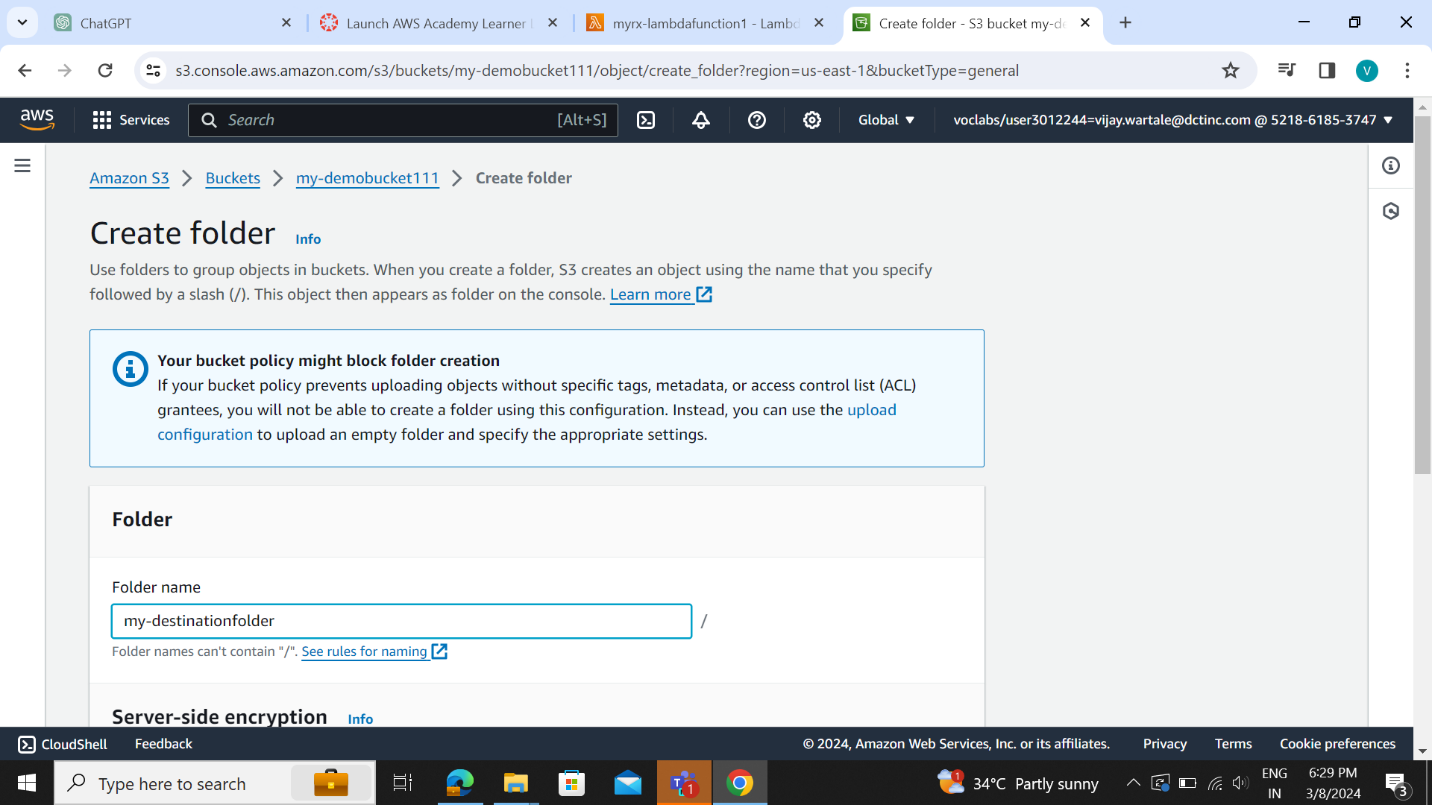


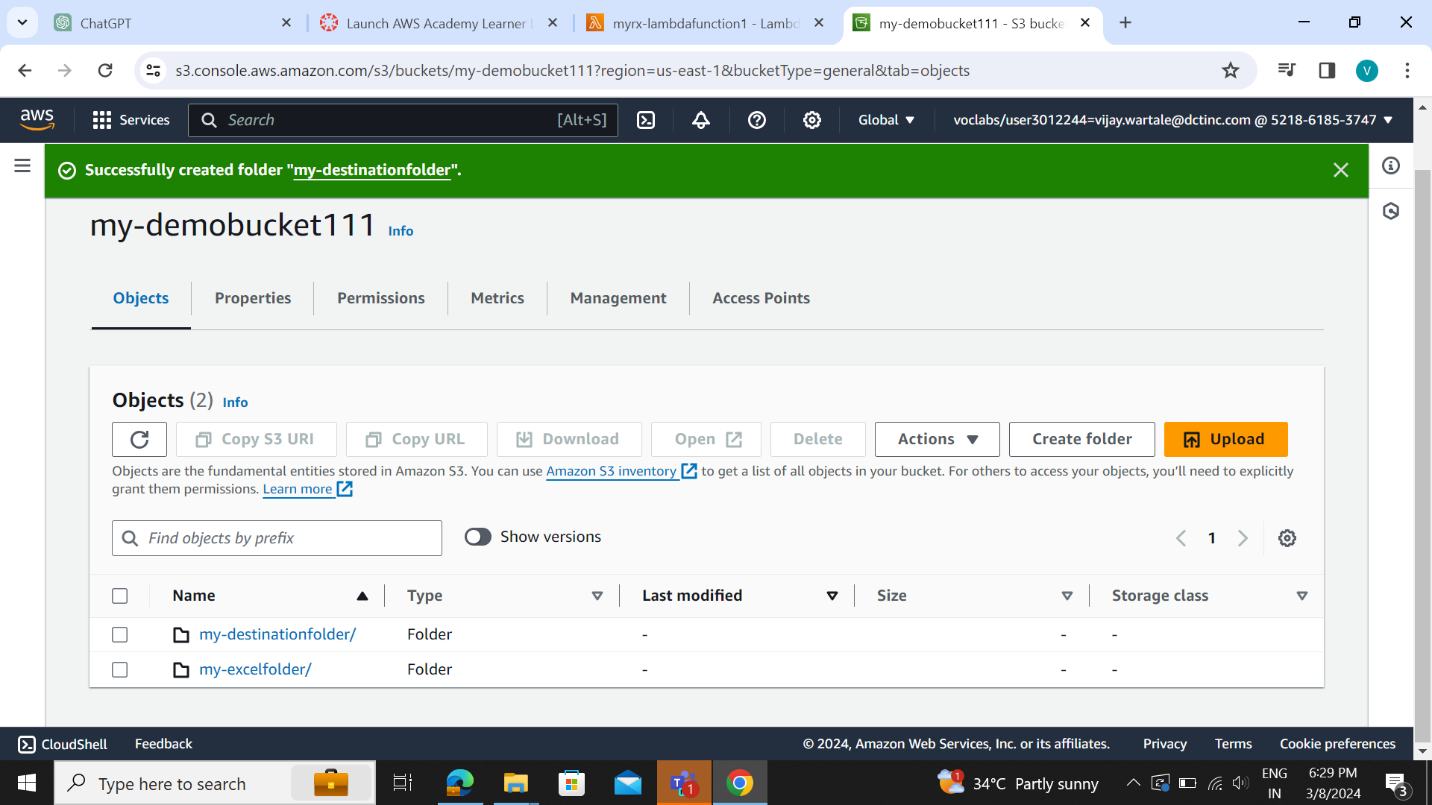


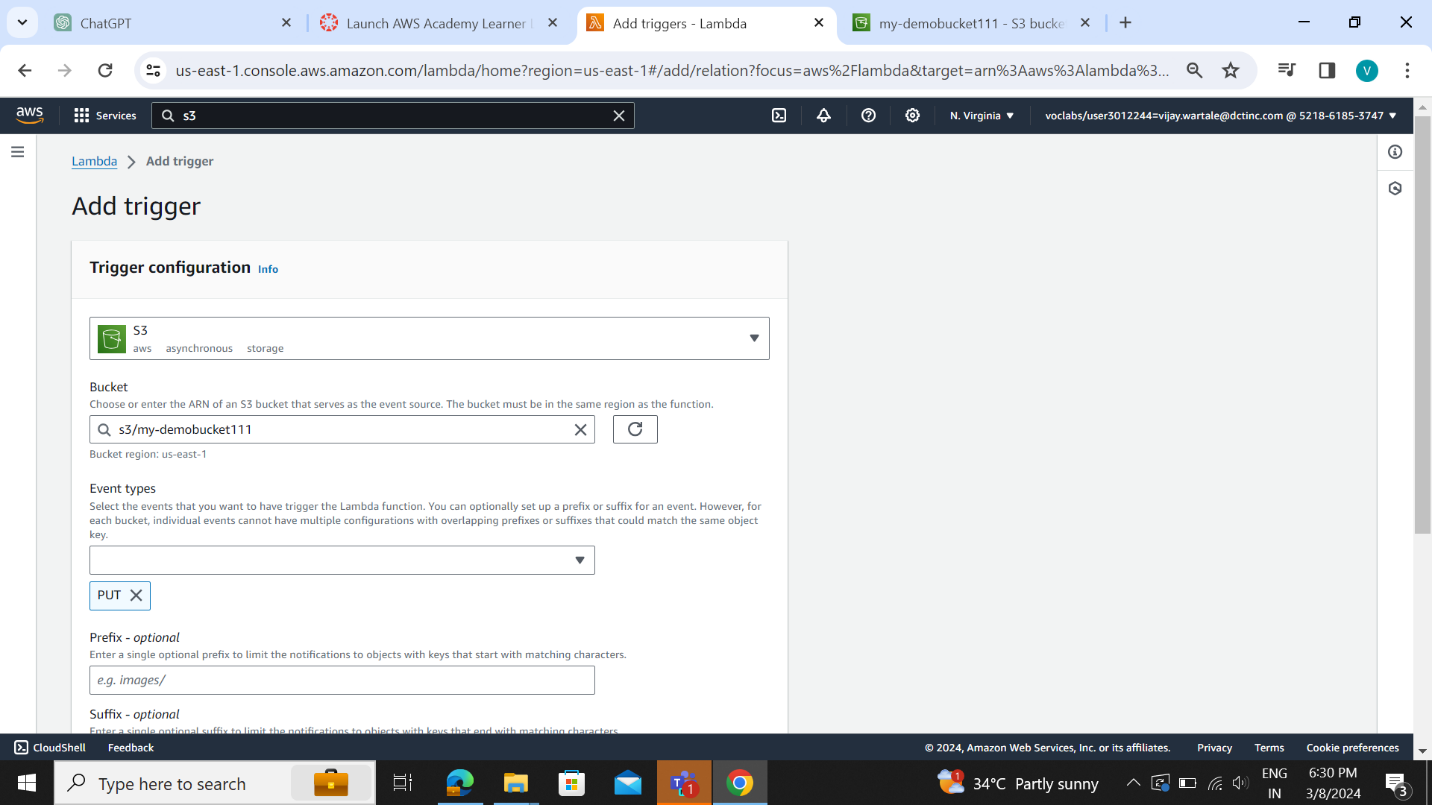


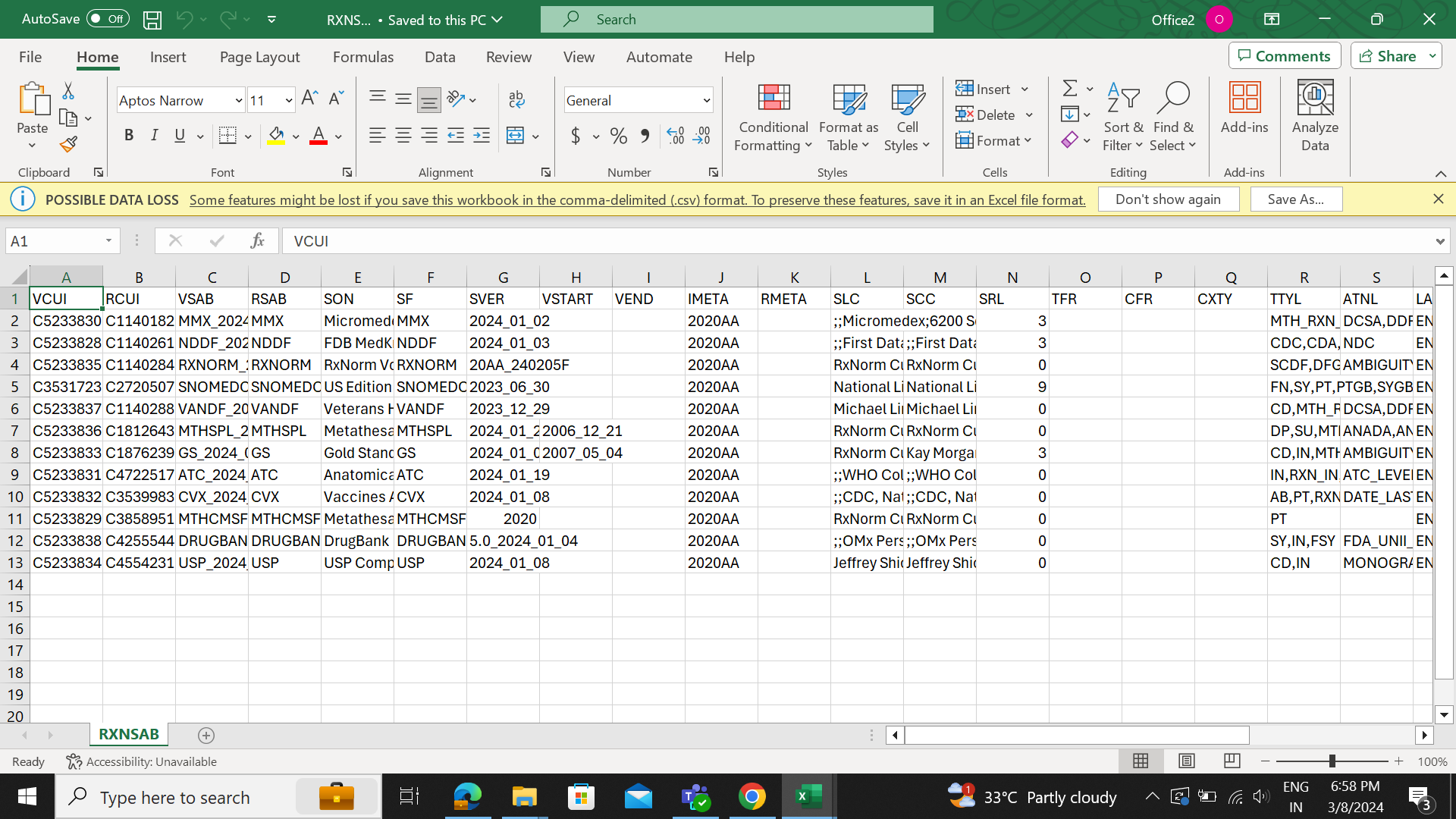
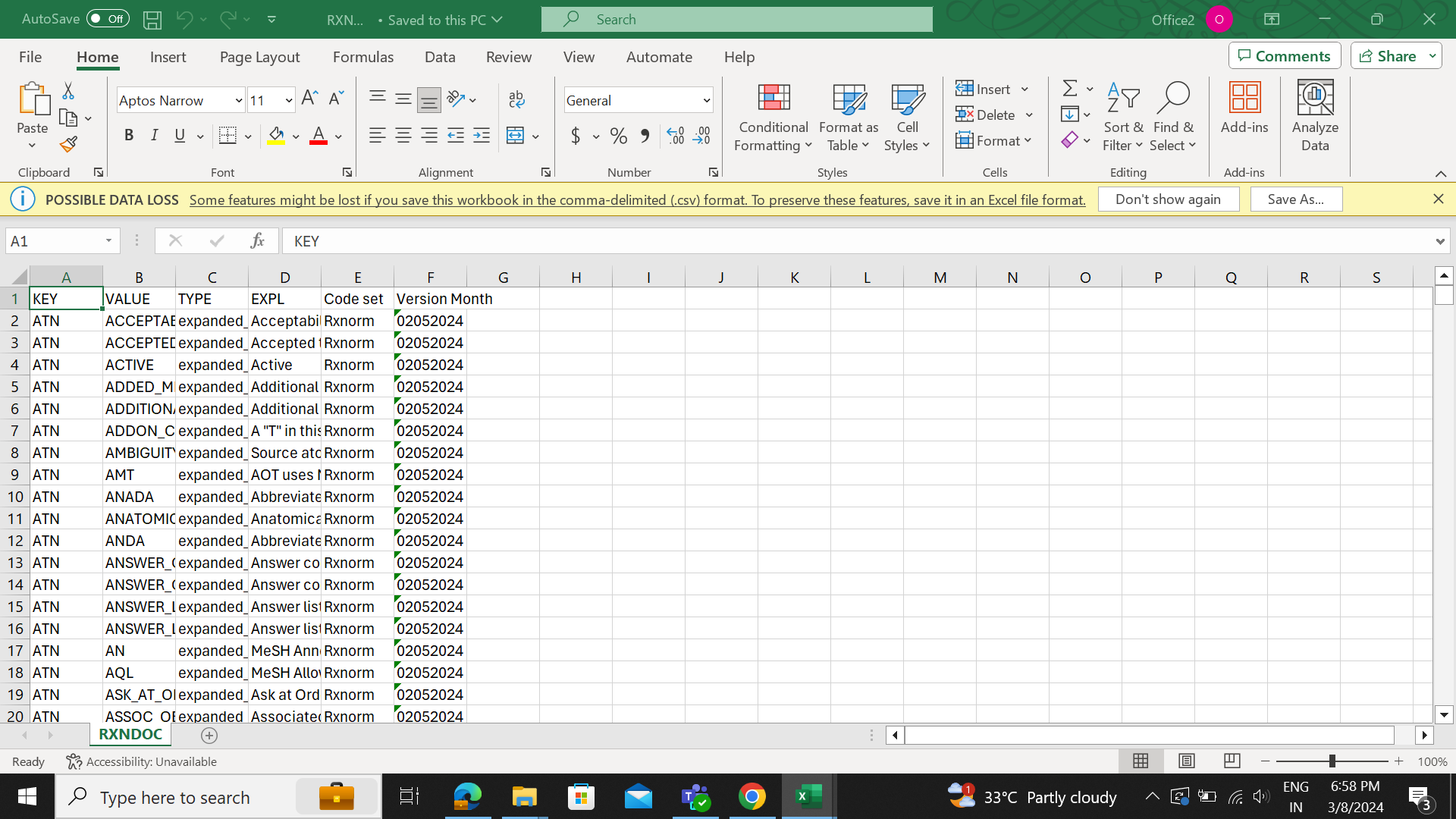
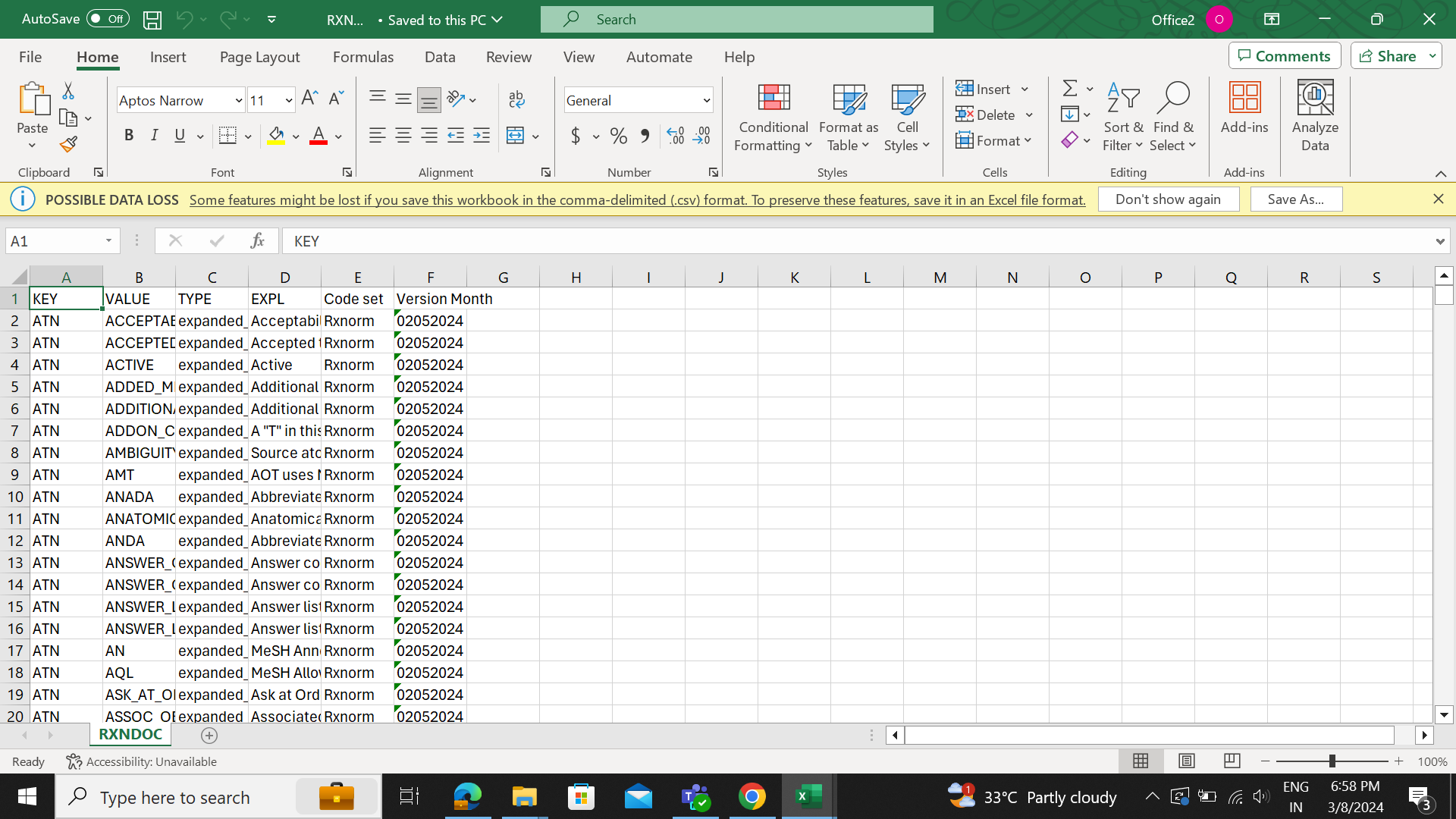
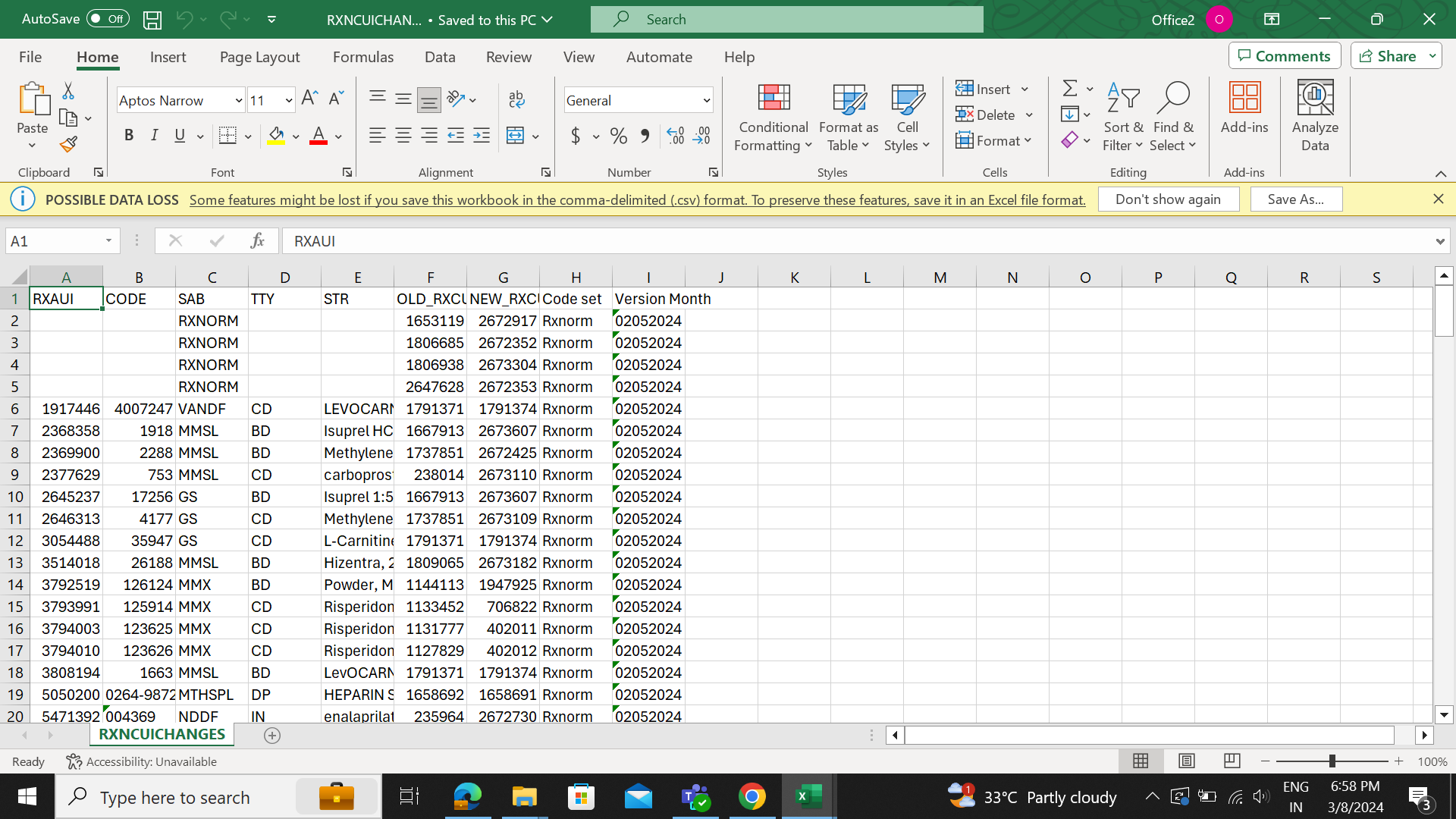
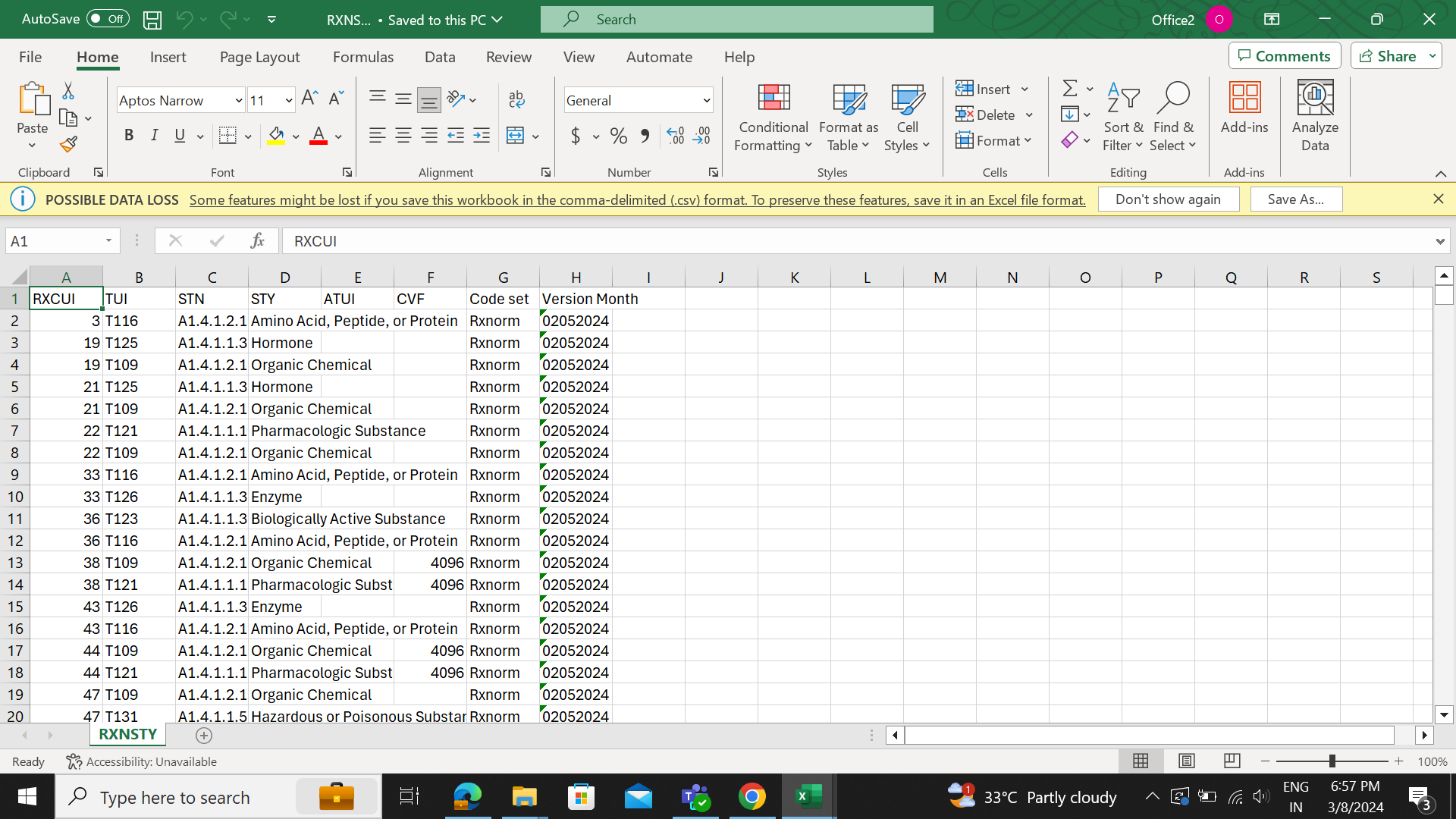
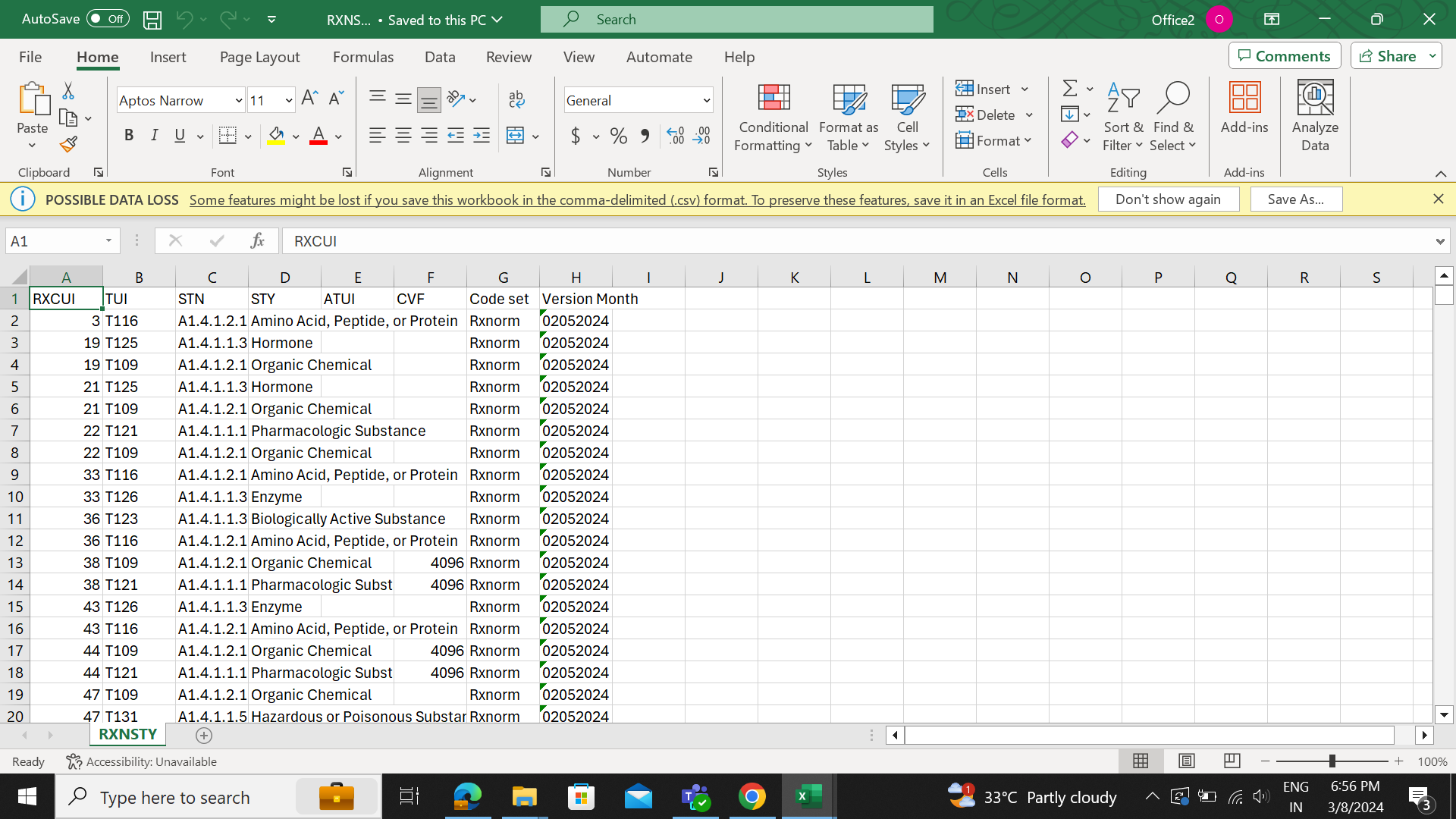
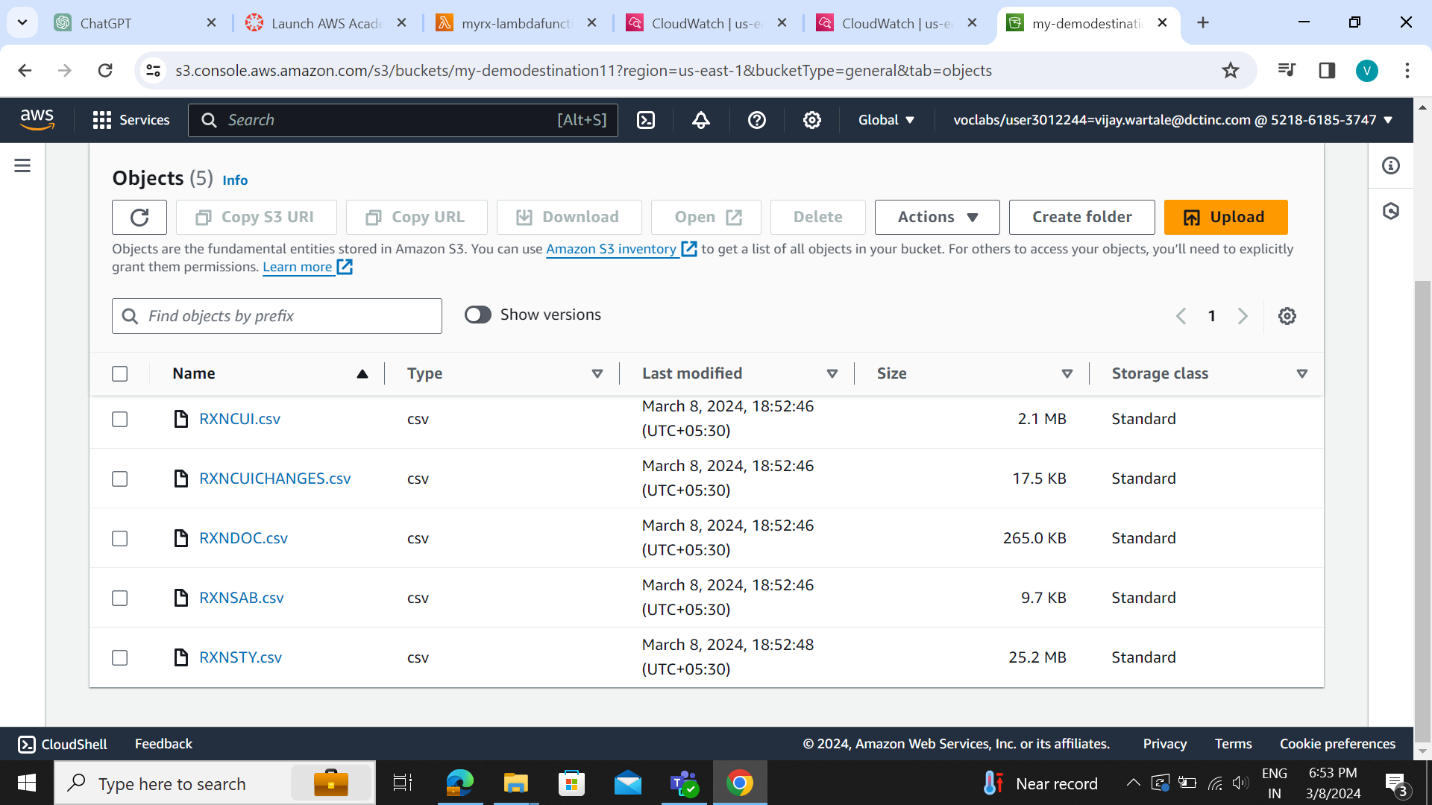
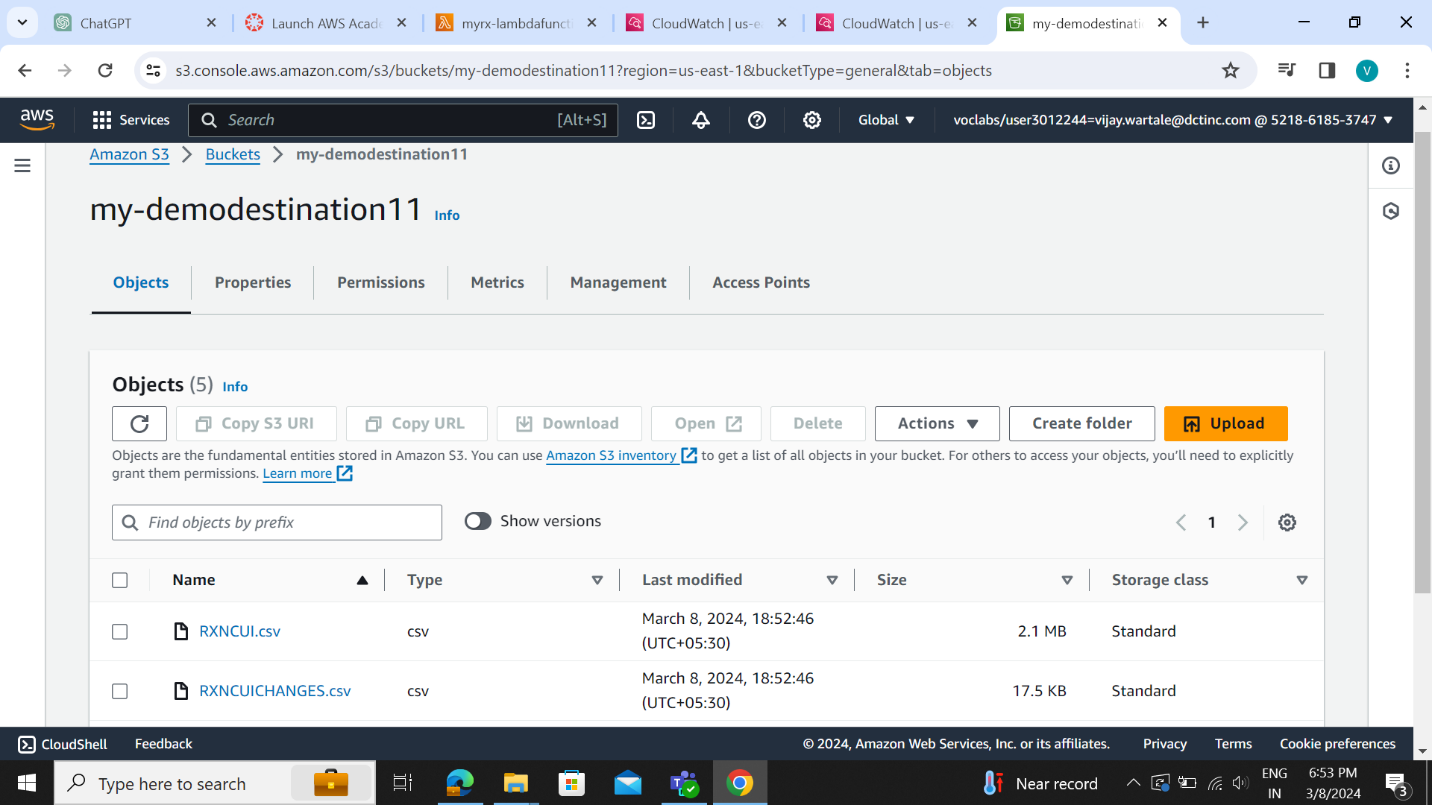
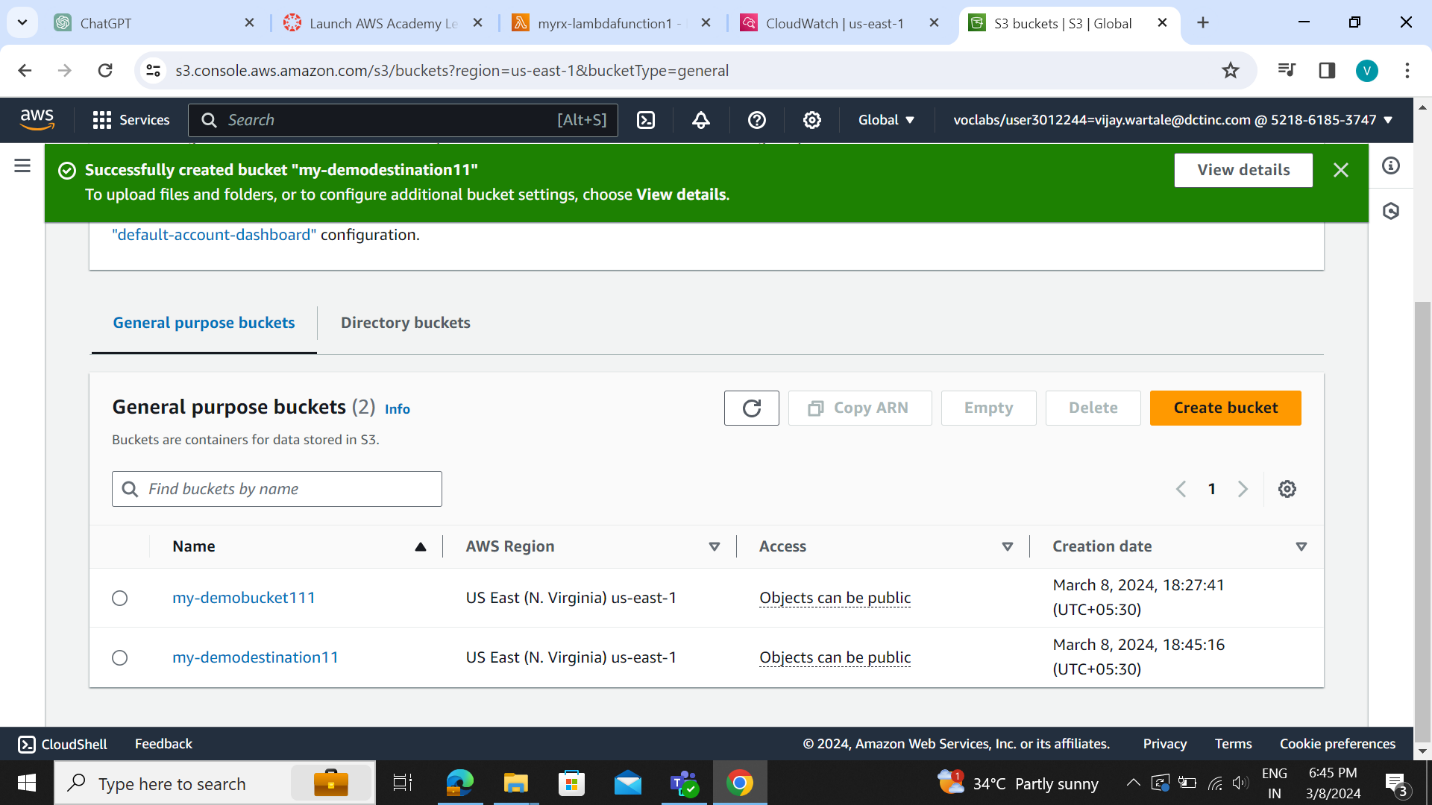
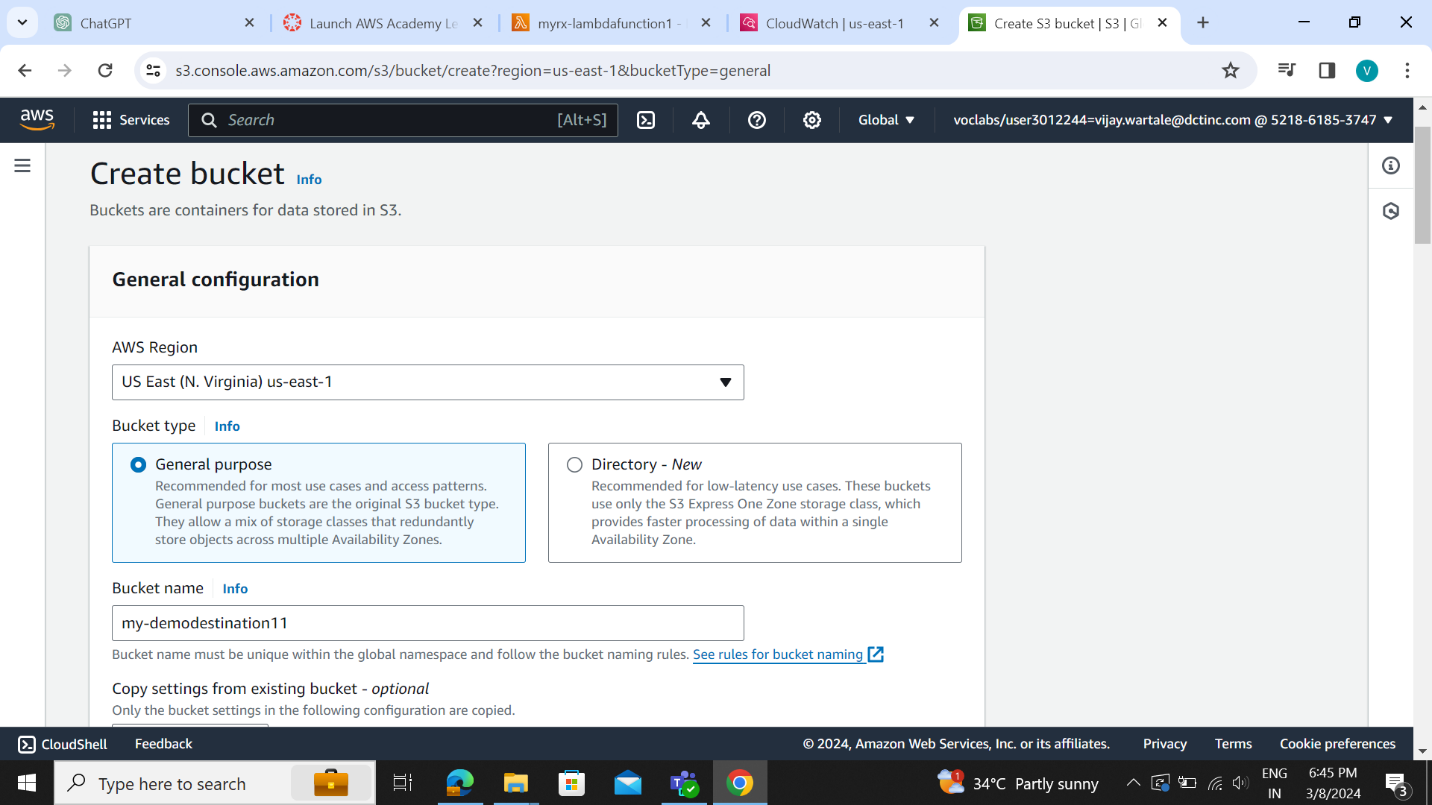
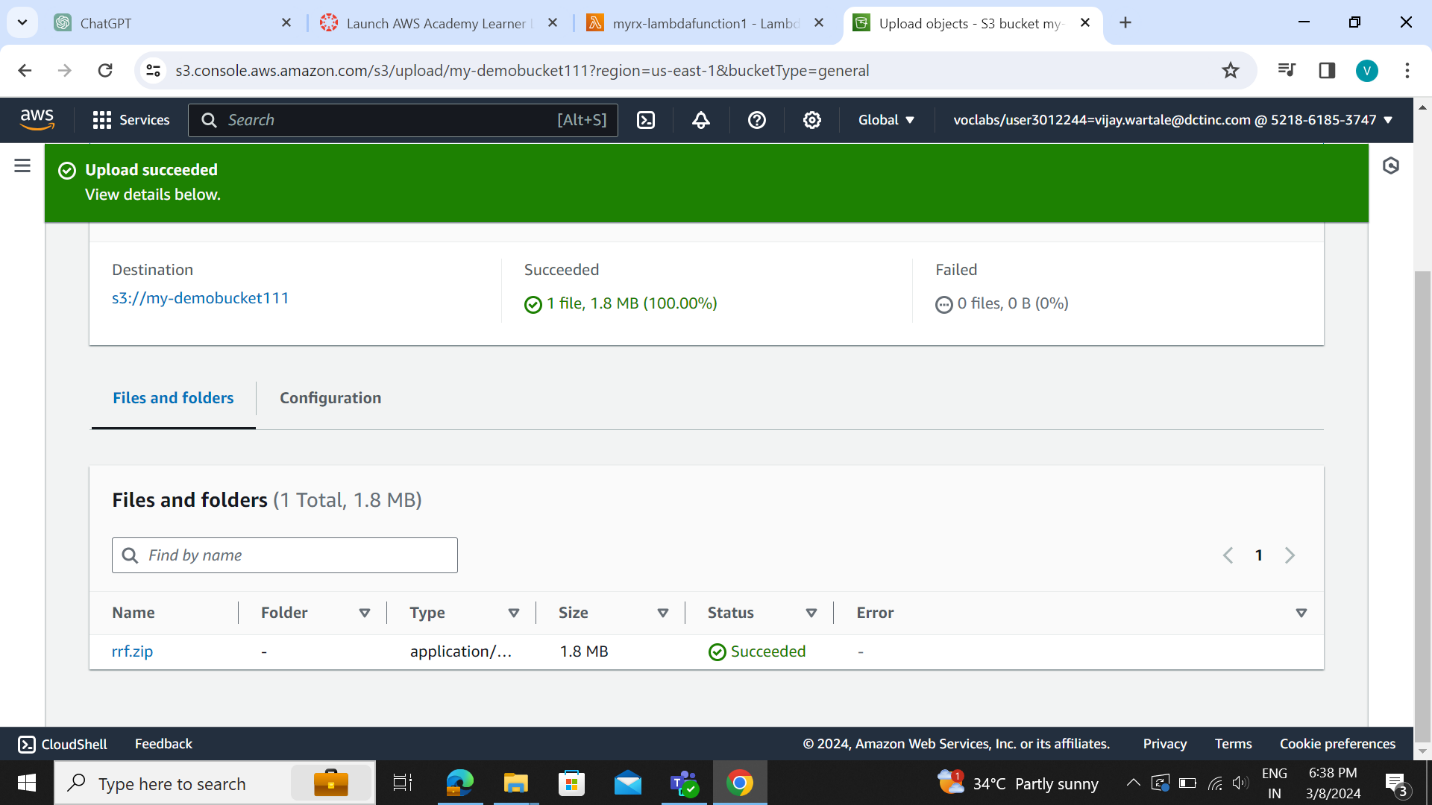
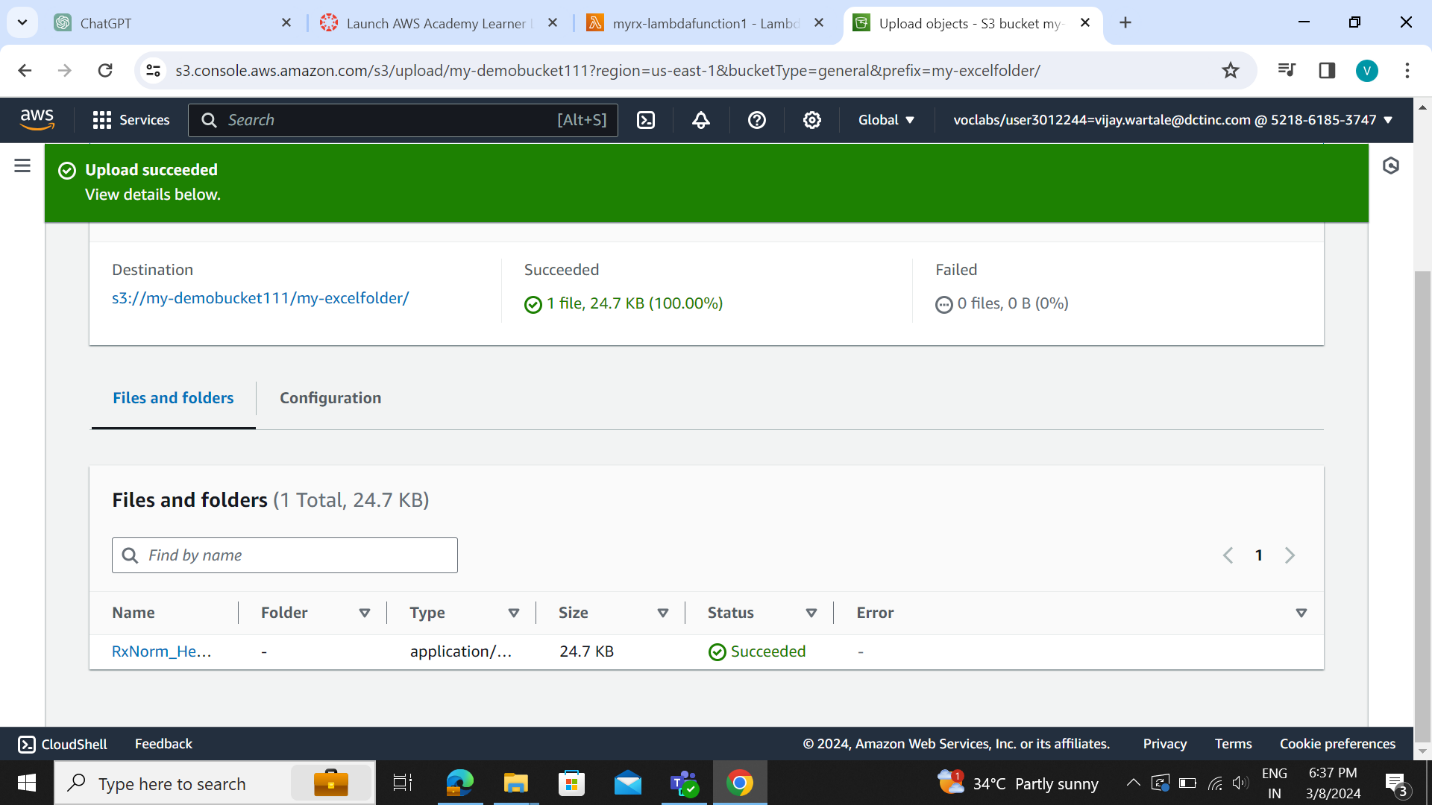
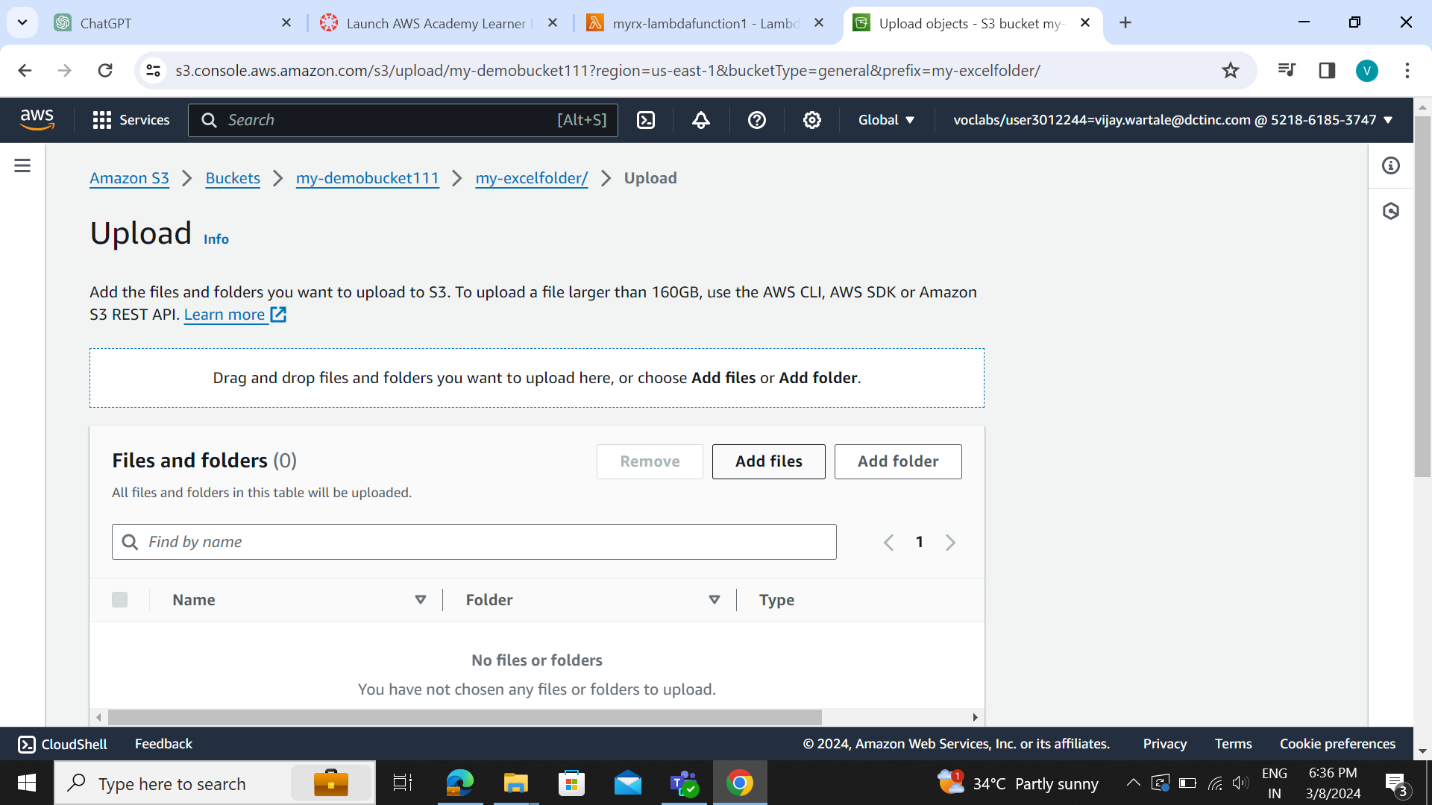
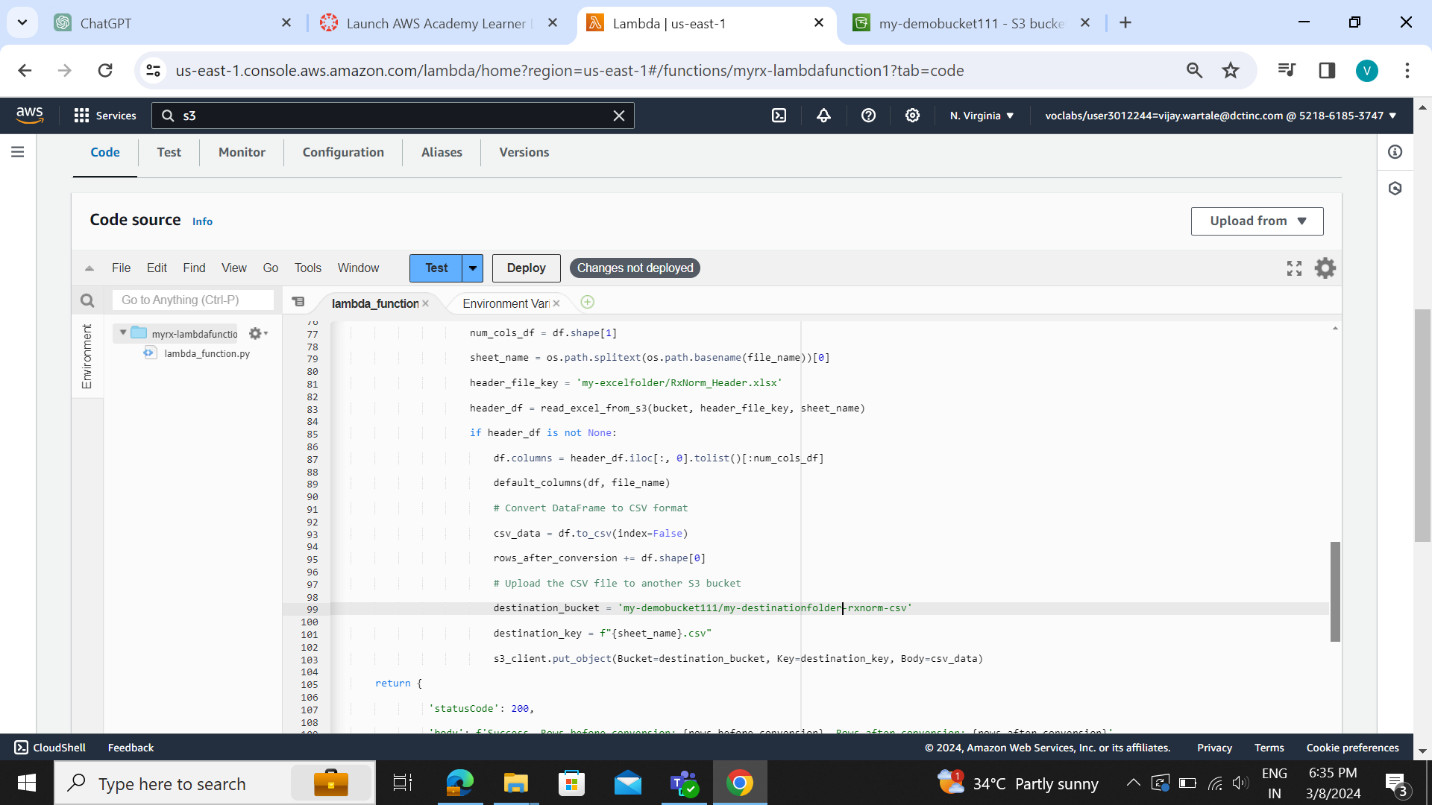
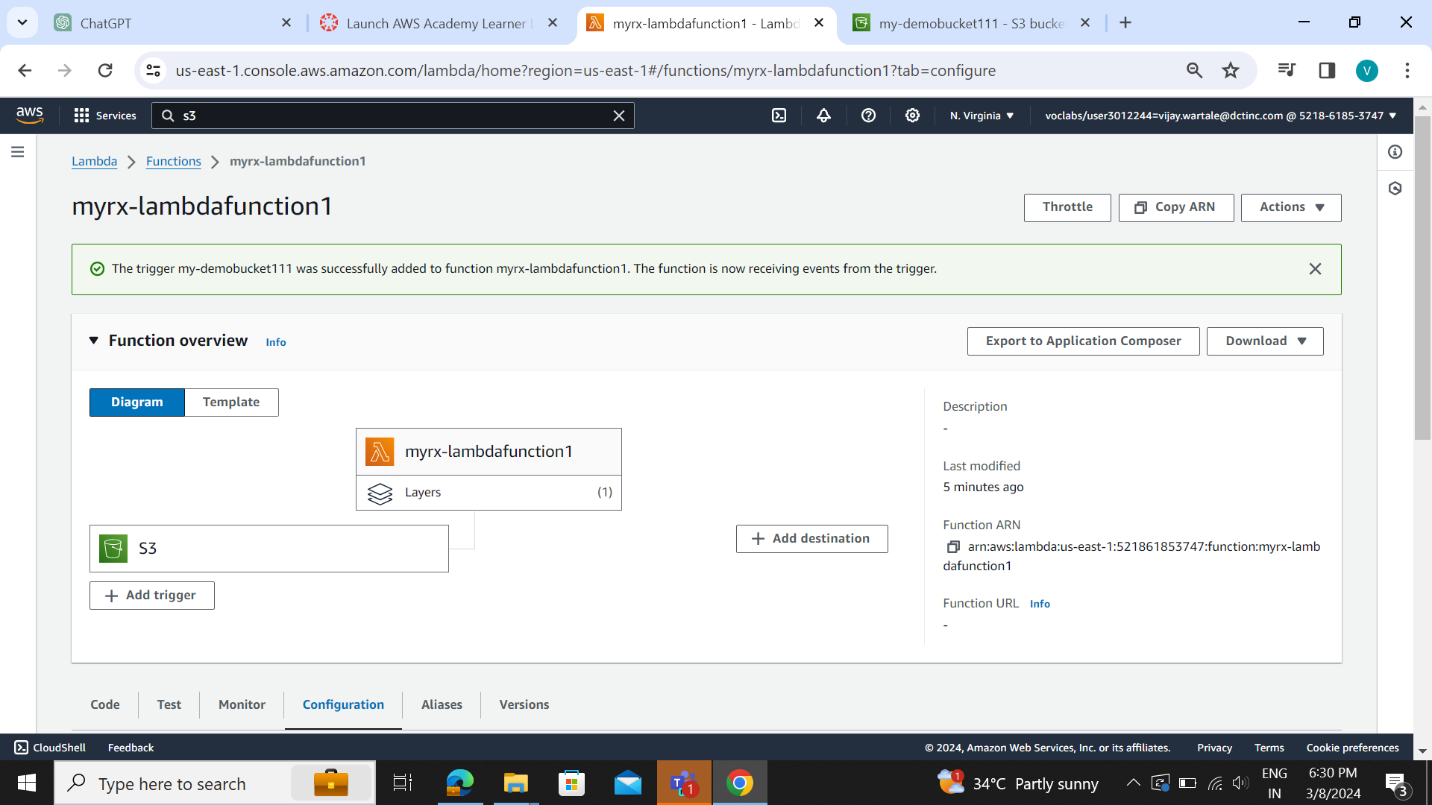


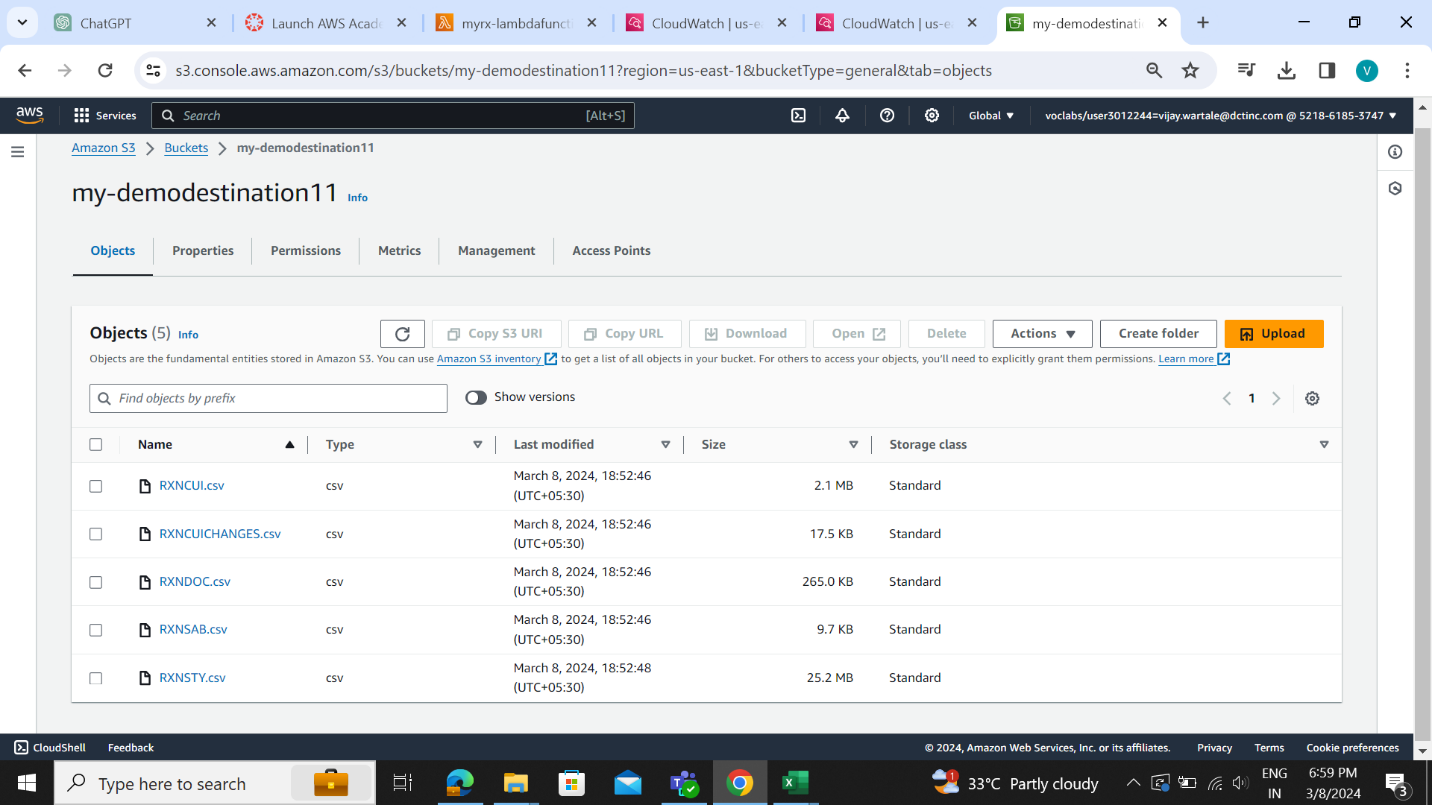












Code:

import os

import boto3

import zipfile

import io

import logging

from datetime import datetime

import pandas as pd

logger = logging.getLogger()

logger.setLevel(logging.INFO)

s3\_client = boto3.client('s3')

def default\_columns(df, file\_name):

# if 'RXNATOMARCHIVE' in file\_name:

# df['CREATED\_TIMESTAMP'] = df['CREATED\_TIMESTAMP'].apply(lambda x: datetime.strptime(x, "%Y-%m-%d %I:%M:%S %p") if pd.notnull(x) else x)

# df['UPDATED\_TIMESTAMP'] = df['UPDATED\_TIMESTAMP'].apply(lambda x: datetime.strptime(x, "%Y-%m-%d %I:%M:%S %p") if pd.notnull(x) else x)

# df['LAST\_RELEASED'] = df['LAST\_RELEASED'].fillna('').astype(str).apply(lambda x: datetime.strptime(x, '%Y-%m-%d').strftime('%Y-%m-%d') if pd.notnull(x) else x)

# elif 'RXNSAB' in file\_name:

# df['VSTART'] = df['VSTART'].apply(lambda x: datetime.strptime(x, '%Y-%m-%d').strftime('%Y-%m-%d') if pd.notnull(x) else x)

# df['VEND'] = df['VEND'].apply(lambda x: datetime.strptime(x, '%Y-%m-%d').strftime('%Y-%m-%d') if pd.notnull(x) else x)

df['Code set'] = 'Rxnorm'

df['Version Month'] = "02052024"

def lambda\_handler(event, context):

try:

logger.info("Received event: %s", event)

rows\_before\_conversion = 0

rows\_after\_conversion = 0

# Get the S3 bucket and object key from the event

bucket = event['Records'][0]['s3']['bucket']['name']

key = event['Records'][0]['s3']['object']['key']

# Download the zip file from S3

response = s3\_client.get\_object(Bucket=bucket, Key=key)

zip\_data = response['Body'].read()

# Extract the zip file

with zipfile.ZipFile(io.BytesIO(zip\_data)) as zip\_ref:

for file\_name in zip\_ref.namelist():

if 'rrf/RX' in file\_name:

# Extract the file

with zip\_ref.open(file\_name) as file\_data:

df = pd.read\_csv(file\_data, delimiter='|')

rows\_before\_conversion += len(df)

# Get header names from the corresponding Excel sheet

num\_cols\_df = df.shape[1]

sheet\_name = os.path.splitext(os.path.basename(file\_name))[0]

header\_file\_key = 'my-excelfolder/RxNorm\_Header.xlsx'

header\_df = read\_excel\_from\_s3(bucket, header\_file\_key, sheet\_name)

if header\_df is not None:

df.columns = header\_df.iloc[:, 0].tolist()[:num\_cols\_df]

default\_columns(df, file\_name)

# Convert DataFrame to CSV format

csv\_data = df.to\_csv(index=False)

rows\_after\_conversion += df.shape[0]

# Upload the CSV file to another S3 bucket

destination\_bucket = 'my-demodestination11'

destination\_key = f"{sheet\_name}.csv"

s3\_client.put\_object(Bucket=destination\_bucket, Key=destination\_key, Body=csv\_data)

return {

'statusCode': 200,

'body': f'Success. Rows before conversion: {rows\_before\_conversion}, Rows after conversion: {rows\_after\_conversion}'

}

except Exception as e:

logger.error("Error: %s", e)

return {

'statusCode': 500,

'body': f'Error: {e}'

}

def read\_excel\_from\_s3(bucket, key, sheet\_name):

try:

response = s3\_client.get\_object(Bucket=bucket, Key=key)

excel\_bytes = response['Body'].read()

df = pd.read\_excel(io.BytesIO(excel\_bytes), sheet\_name=sheet\_name, header=None)

return df

except Exception as e:

logger.error("Error reading Excel file: %s", e)

return None

# def get\_last\_updated\_version\_month(filename):

# parts = filename.split('\_')

# version\_month = parts[-1].split('.')[0]

# month = version\_month[:2]

# year = version\_month[4:]

# return f"{month}\_{year}"