**GENERATIVE AI WITH SQL DB AND GRADIO GUI**

**import** matplotlib.pyplot **as** plt

**import** io

**import** base64

**from** langchain.agents **import** create\_sql\_agent

**from** langchain.agents.agent\_toolkits **import** SQLDatabaseToolkit

**from** langchain.sql\_database **import** SQLDatabase

**from** langchain.llms.openai **import** OpenAI

**from** langchain.agents **import** AgentExecutor

**from** langchain.agents.agent\_types **import** AgentType

**from** langchain.chat\_models **import** ChatOpenAI

**import** pyodbc

**import** gradio **as** gr

prompt**=**"Ask the Question"

**def** openai\_create(prompt):

*# Define the SQL database and OpenAI interface (assuming you have already set them up)*

db **=** SQLDatabase**.**from\_uri('mssql+pyodbc://pentaho:pentaho123@10.125.48.174:1433/ChatGPT?driver=ODBC+Driver+17+for+SQL+Server')

llm **=** ChatOpenAI(openai\_api\_key**=**"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", model**=**"gpt-3.5-turbo-16k")

toolkit **=** SQLDatabaseToolkit(db**=**db, llm**=**llm)

agent\_executor **=** create\_sql\_agent(

llm**=**llm,

toolkit**=**toolkit,

verbose**=False**,

agent\_type**=**AgentType**.**OPENAI\_FUNCTIONS,

)

a **=** agent\_executor(prompt)

plain\_text **=** "\n"**.**join([f"{key**.**upper()}: {value}" **for** key, value **in** a**.**items()])

**return** plain\_text

**def** chatgpt\_clone(input, history):

history **=** history **or** []

output **=** openai\_create(input)

history**.**append((input, output))

**return** history,history

*#block = gr.Blocks()*

block **=** gr**.**Blocks(css**=**".gradio-container {background-color: #f5f5dc}", theme**=**gr**.**themes**.**Soft(font**=**[gr**.**themes**.**GoogleFont("Sohne-Bold"),

"SohneMono-Bold", "sans-serif"], primary\_hue**=**"amber", secondary\_hue**=**"emerald",

neutral\_hue**=**"slate"))

**with** block:

**with** gr**.**Row():

**with** gr**.**Column(scale**=**1):

gr**.**HTML("""<div id='output\_image' style='display:block; margin-left: auto; margin-right: auto;

align-items: left; justify-content: center;'></div>""")

result **=** gr**.**Image("D:\Important Docs & PPTs\PythonScripts\_&\_PowerBI\power bi\Adani2.png",

label**=None**, show\_label**=False**, height**=**100, width**=**70, elem\_id**=**"output\_image",

show\_download\_button**=False**, container**=False**)

**with** gr**.**Column(scale**=**1, variant**=**"default"):

gr**.**Markdown("""<h1><center>AEML GENERATIVE AI</center></h1>""")

**with** gr**.**Column(scale**=**1):

**None**

chatbot **=** gr**.**Chatbot()

**with** gr**.**Row(equal\_height **=** **True**):

**with** gr**.**Column(scale **=** 6, min\_width **=** 900): *# to change size of text prompt increase/decrease scale*

message **=** gr**.**Textbox(placeholder**=**prompt)

state **=** gr**.**State()

**with** gr**.**Column(scale **=** 1, min\_width **=** 100): *# to change size of submit button increase/decrease scale*

submit **=** gr**.**Button("SEND 👆🏼")

submit**.**click(chatgpt\_clone, inputs**=**[message,state], outputs**=**[chatbot, state])

block**.**launch(debug **=** **True**)

**PDF SUMMARIZER USING OPENAI GPT4 AND GRADIO GUI**

import gradio as gr

from langchain.embeddings.openai import OpenAIEmbeddings

from langchain.text\_splitter import CharacterTextSplitter

from langchain.vectorstores import Chroma

from langchain.embeddings.sentence\_transformer import SentenceTransformerEmbeddings

from langchain.chains import ConversationalRetrievalChain

from langchain.chat\_models import ChatOpenAI

from chromadb.utils import embedding\_functions

from langchain.document\_loaders import PyPDFLoader

import os

import fitz

import openai

from nltk.tokenize import sent\_tokenize

from io import StringIO

import json

from io import StringIO

import openai

N=0

import fitz

from PIL import Image

openai.api\_key="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"

file\_name="hvsdhg"

def read\_pdf(file):

context = ""

# Open the PDF file

with fitz.open(file) as pdf\_file:

# Get the number of pages in the PDF file

num\_pages = pdf\_file.page\_count

# Loop through each page in the PDF file

for page\_num in range(num\_pages):

# Get the current page

page = pdf\_file[page\_num]

# Get the text from the current page

page\_text = page.get\_text()

# Append the text to context

context += page\_text

return context

def split\_text(text, chunk\_size=5000):

"""

Splits the given text into chunks of approximately the specified chunk size.

Args:

text (str): The text to split.

chunk\_size (int): The desired size of each chunk (in characters).

Returns:

List[str]: A list of chunks, each of approximately the specified chunk size.

"""

chunks = []

current\_chunk = StringIO()

current\_size = 0

sentences = sent\_tokenize(text)

for sentence in sentences:

sentence\_size = len(sentence)

if sentence\_size > chunk\_size:

while sentence\_size > chunk\_size:

chunk = sentence[:chunk\_size]

chunks.append(chunk)

sentence = sentence[chunk\_size:]

sentence\_size -= chunk\_size

current\_chunk = StringIO()

current\_size = 0

if current\_size + sentence\_size < chunk\_size:

current\_chunk.write(sentence)

current\_size += sentence\_size

else:

chunks.append(current\_chunk.getvalue())

current\_chunk = StringIO()

current\_chunk.write(sentence)

current\_size = sentence\_size

if current\_chunk:

chunks.append(current\_chunk.getvalue())

return chunks

def gpt3\_completion(prompt, engine='text-davinci-003', temp=0.0, top\_p=0.3, tokens=1000):

prompt = prompt.encode(encoding='ASCII',errors='ignore').decode()

try:

response = openai.Completion.create(

engine=engine,

prompt=prompt,

temperature=temp,

top\_p=top\_p,

max\_tokens=tokens)

return response.choices[0].text.strip()

except Exception as oops:

return "GPT-3 error: %s" % oops

def summerize(message):

#print(file\_name)

# Calling the split function to split text

document=read\_pdf(file\_name)

chunks = split\_text(document)

summeries = []

for chunk in chunks:

prompt = "Please summerize the following document and answer the question: \n"

summary = gpt3\_completion(prompt + chunk)

if summary.startswith("GPT-3 error:"):

continue

summeries.append(summary)

return ''.join(summeries)

def chatgpt\_clone(input, history):

history = history or []

output = summerize(input)

history.append((input, output))

return history,history

def render\_file1(file):

global N

global file\_name

#print(file.name)

file\_name=file.name

doc = fitz.open(file.name)

page = doc[N]

# Render the page as a PNG image with a resolution of 300 DPI

pix = page.get\_pixmap(matrix=fitz.Matrix(300/72, 300/72))

image = Image.frombytes('RGB', [pix.width, pix.height], pix.samples)

return image

with gr.Blocks() as demo:

# Create a Gradio block

with gr.Column():

with gr.Row():

chatbot = gr.Chatbot(value=[], elem\_id='chatbot')

show\_img = gr.Image(label='Upload PDF')

with gr.Row():

with gr.Column():

txt = gr.Textbox(placeholder="Enter text and press enter")

state = gr.State()

with gr.Column():

submit\_btn = gr.Button('Submit')

with gr.Column():

btn = gr.UploadButton("📁 Upload a PDF", file\_types=[".pdf"])

# Event handler for uploading a PDF

btn.upload(fn=render\_file1, inputs=[btn], outputs=[show\_img])

#btn.click(fn=read\_pdf,inputs=[btn])

submit\_btn.click(chatgpt\_clone,inputs=[txt,state],outputs=[chatbot,state])

demo.queue()

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

demo.launch()

**HES PRODUCTION – API INTEGRATION**

import json

import requests

from requests.structures import CaseInsensitiveDict

import pandas as pd

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning) # No warnings like depricated method will be displayed on the screen after running this python program

warnings.filterwarnings('ignore')

import pyodbc

import pandas as pd

import configparser

import os,sys

import numpy

import datetime as da

from datetime import date

from datetime import datetime, timedelta

import shutil

file\_name = 'prop.ini'

prop\_file\_path="D:\Apoorv\MDM\scripts"

config = configparser.ConfigParser()

config.read(os.path.join(prop\_file\_path,file\_name))

conn\_strng = []

for key,value in config.items('DB\_CONN\_SMART\_METER'):

strng = '%s=%s'%(key,value)

conn\_strng.append(strng)

conn\_strng = ';'.join(conn\_strng)

cnxn = pyodbc.connect(conn\_strng)

cursor = cnxn.cursor()

count = 0.

InsertCount=0

a = da.datetime.now()

print('Start Time 1->',a)

try:

def call\_api(url,username,password, data):

# print(data)

try:

response = requests.post(url, auth=(username, password),json = data)

response.raise\_for\_status()

#print(response.text)

return response.text

except requests.exceptions.RequestsException as e:

print (f"Error: {e}")

return None

df = pd.DataFrame(columns=['Date','profileObisCode','meterId', 'OC','V','AID'])

today = date.today() - timedelta(days = 1)

StartDate=today.strftime("%Y-%m-%d 00:00:00")

today1 = date.today() - timedelta(days = 0)

EndDate=today1.strftime("%Y-%m-%d 23:59:59")

vObisCode1=["1.0.94.91.0.255","1.0.99.2.0.255","1.0.99.1.0.255","1.0.98.1.0.255"]

for vobiscode in vObisCode1:

# hes\_strng = []

# for key,value in config.items('HES'):

# strng\_HES = '%s=%s'%(key,value)

# hes\_strng.append(strng\_HES)

# hes\_strng = ';'.join(hes\_strng)

# api\_url = + vobiscode

# api\_username = "powerbi"

# api\_password = "rHn4DNLxsN^89qVg"

#api\_url = "http://10.127.2.15:7070/wsintegrations/rest/hes/meterProfileData?obisCode=" + vobiscode

api\_url = "http://10.127.4.18:8080/wsintegrations/rest/hes/meterProfileData?obisCode=" + vobiscode

api\_username = "powerbi"

api\_password = "rHn4DNLxsN^89qVg"

# meterdata = ["SQ0324421","SQ0324422","SQ0324423","SQ0324424","SQ0324417","SQ0324426","SH954925","SH954927","SH954926","SH954929",

# "SH954970","SH954928","SH954971","SH954972","SH954973","SH954974","SH954881","SH954882","SH954883","SH955089",

# "SH955091","SH953015","SH953014","SH954382","SH954384","SH952682"]

meterdata = ["9161948", "9168404", "SM30020772", "9092576", "9115769", "SM30021744", "9184841",

"SM30002848", "SM30003695", "SM30003891", "SM30007367", "9154360", "9026160",

"9179140", "SM30015068", "9113627", "SM30010776", "SM30017243", "SM30011736",

"SM30009955", "SM30010310", "SM30014915", "SM30013625", "SM30014232", "SM30014143",

"SM30028234", "SM30016602", "SM30028356", "SM30020651", "9067015", "SM30018792",

"SM30018792", "SM30022217", "SM30027291", "SM30027125", "SM30031984", "SM30032295",

"SM30031695", "SM30044401", "SM30042305", "SM30034417", "SM30043675", "SM30046013",

"SM30033216", "SM30033213", "SM30004744", "SM30016654", "SM30007647", "SM30034687",

"SM30033570", "SM30031552", "SM30041373", "SM30038277", "SM30037333", "9146503",

"SM30033414", "SM30039276", "SM30041460", "SM30019015", "SM30036435", "SM30034765",

"SM30037869", "SM30037539", "SM30049888", "SM30037403", "SM30044045", "SM30035038",

"SM30036475", "9163186", "SM30011587", "SM30049835", "9185905", "SM30000404",

"SM30036249", "SM30032375", "9167290", "SM30050709", "SM30051016", "SM30051089",

"SM30045073", "SM30050754", "SM35000368", "SM30050751", "SM35000293", "SM30050648",

"9007623", "SM30050812", "SM30051052", "SM35000359", "SM30050463", "SM30050156"]

for meterid in meterdata:

data = {

"count":1000,

"endTime": EndDate, #"2023-06-26 23:59:59",

"startIndex":1,

"startTime":StartDate, #"2023-06-23 00:00:00",

"meterList":[meterid]

}

response\_data = call\_api(api\_url,api\_username, api\_password, data)

Responsedata = json.loads(response\_data)

status = Responsedata.get('status')

if status == "SUCCESS":

profileObisCode= Responsedata['data'].get('profileObisCode')

for profile\_data in Responsedata['data']['profileData']:

meterid = profile\_data['meterId']

date=None

# Set record date

for register\_value in profile\_data['registerValues']:

oc = register\_value['OC']

v = register\_value['V']

if(oc == '0.0.1.0.0.255'):

date = v if isinstance(v, str) else None #

#date = v.split()[0] if isinstance(v, str) else None #

for register\_value in profile\_data['registerValues']:

oc = register\_value['OC']

v = register\_value['V']

aid = register\_value['AID']

df = df.append({'profileObisCode':profileObisCode,'meterId':meterid, 'OC': oc, 'V': v, 'AID': aid,'Date': date }, ignore\_index=True)

# for profile\_data in Responsedata['data']['profileData']:

# meterid = profile\_data['meterId']

# DateIndex=1

# for register\_value in profile\_data['registerValues']:

# oc = register\_value['OC']

# v = register\_value['V']

# if DateIndex==1:

# date = v.split()[0] if isinstance(v, str) else None #

# DateIndex=0

# aid = register\_value['AID']

# df = df.append({'profileObisCode':profileObisCode,'meterId':meterid, 'OC': oc, 'V': v, 'AID': aid,'Date': date }, ignore\_index=True)

else:

print("Failed to extract data",vobiscode, " for meter ", meterid)

#print(df)

df[['Date','profileObisCode','meterId','OC','V','AID']] = df[['Date','profileObisCode','meterId','OC','V','AID']].fillna(' ')

for index, row in df.iterrows():

sql\_Delete= "Delete from [Smart\_Meter].[dbo].[HES\_DATA\_PROD] where [Date] = '" + row['Date'] +"' and [ProfileObisCode] = '" + row['profileObisCode'] + "' and [MeterId] = '" + row['meterId'] + "' and [OC] = '" + row['OC'] +"'and [V] = '" + row['V'] +"'"

cnxn.execute(sql\_Delete)

cnxn.commit()

count = count + 1.

cursor.execute('INSERT INTO [Smart\_Meter].[dbo].[HES\_DATA\_PROD]('

'[Date],[ProfileObisCode], [MeterId], [OC], [V], [AID]) values'

'(?,?,?,?,?,?)',

row['Date'],row['profileObisCode'],row['meterId'],row['OC'],row['V'],row['AID']

)

if count >= 1000:

cnxn.commit()

InsertCount=InsertCount+1

#cursor.close()

print("Records inserted in [Smart\_Meter].[dbo].[HES\_DATA\_PROD] -->" ,InsertCount\*1000)

b = da.datetime.now()

print(b)

count = 0

except pyodbc.Error as e:

msg = str(e)

raise e

b = da.datetime.now()

c = b-a

print("=======================================================")

print('Completion Time 2->',b)

print("Time taken in minutes -->", round(c.seconds/60,1))

cnxn.commit()

cursor.close()

cnxn.close()

df.iloc[0:0]

**MDM PRODUCTION CODE – API INTEGRATION**

import pyodbc

import pandas as pd

import requests

import urllib3

urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)

import uuid

import xmltodict

import configparser as configparser

import time

from dateutil import parser

from dateutil import tz

from datetime import datetime

prop\_file\_path = r"D:\Apoorv\MDM\scripts\prop.ini"

# Read the database connection information from the 'prop.ini' file

config = configparser.ConfigParser()

config.read(prop\_file\_path)

server\_name = config.get('DB\_CONN\_SMART\_METER', 'SERVER')

database\_name = config.get('DB\_CONN\_SMART\_METER', 'DATABASE')

driver = config.get('DB\_CONN\_SMART\_METER', 'DRIVER')

username = config.get('DB\_CONN\_SMART\_METER', 'UID')

password = config.get('DB\_CONN\_SMART\_METER', 'PWD')

connection\_string = f'DRIVER={driver};SERVER={server\_name};DATABASE={database\_name};UID={username};PWD={password}'

# Recording the start time

a = time.time()

try:

connection = pyodbc.connect(connection\_string)

cursor = connection.cursor()

# Execute the SQL query to fetch the top rows

query = "SELECT TOP 20 Meter\_Object\_ID FROM Smart\_Meter.dbo.ZAMI\_MTRS\_LTP2;"

df1 = pd.read\_sql\_query(query, connection)

# URL and SOAP envelope template

url = "https://gcpaeamei01p.adani.com:8151/em-mws/services/MeterReadService"

envelope\_template = """

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

<soapenv:Header/>

<soapenv:Body>

<RequestMessage xmlns="http://www.emeter.com/energyip/meterreadinterface">

<Header>

<verb>get</verb>

<noun>MeterRead</noun>

<revision>1</revision>

<dateTime>2022-12-07T09:30:15</dateTime>

<messageID>{}</messageID>

</Header>

<Request>

<startTime>2023-08-12T00:00:00.000</startTime>

<endTime>2023-08-13T00:00:00.000</endTime>

<objectIdType>METER\_X\_UNIVERSAL\_ID</objectIdType>

<objectId>{}</objectId>

<organisationCode>AEML</organisationCode>

<measurementProfile>Consolidated consumption trend</measurementProfile>

</Request>

</RequestMessage>

</soapenv:Body>

</soapenv:Envelope>

"""

username = "mws"

password = "mW$1234!!Prod"

headers = {

'Content-Type': 'text/xml; charset=utf-8',

}

# Initialize DataFrame to store the extracted data for all meters

df\_columns=['StartTime', 'EndTime', 'IntervalLength', 'Value', 'NoData', 'Flags',

'ServicePoint\_mRID', 'Meter\_mRID', 'ReadingType\_mRID', 'MeasurementType',

'Unit', 'TOUCode', 'ChannelNumber']

# Initialize the counter and insert count

count = 0

insert\_count = 0

missed\_meters = []

data\_list = []

# Fetching the results and directly processing them in the loop

for index, row in df1.iterrows():

meter\_object\_id = row.Meter\_Object\_ID

#print("1")

# Generate a random UUID

message\_id = str(uuid.uuid4())

# Fill in the meter object ID and message ID in the envelope template

envelope = envelope\_template.format(message\_id, meter\_object\_id)

response = requests.post(url, data=envelope, headers=headers, auth=(username, password), verify=False)

current\_time = datetime.now()

print(f"Processed time: {current\_time}")

print(f"Fetching API record for Meter ID: {meter\_object\_id}")

if response.status\_code == 200:

try:

# Convert XML response to dictionary

data\_dict = xmltodict.parse(response.content)

#print("2")

# Extracting data from the data\_dict

meter\_reading = data\_dict['soap:Envelope']['soap:Body']['ReplyMessage']['Payload']['MeterReading']

service\_point\_mrid = meter\_reading['ServicePoint']['mRID']

meter\_mrid = meter\_reading['Meter']['mRID']

interval\_blocks = meter\_reading['IntervalBlock']

# Handling the case where there is only one interval block (convert to a list)

if not isinstance(interval\_blocks, list):

interval\_blocks = [interval\_blocks]

# Extracting data from IReading within each IntervalBlock

for interval\_block in interval\_blocks:

reading\_type\_mrid = interval\_block['ReadingType']['mRID']

measurement\_type = interval\_block['ReadingType']['measurementType']

unit = interval\_block['ReadingType']['unit']

tou\_code = interval\_block['ReadingType']['touCode']

channel\_number = interval\_block['ReadingType']['channelNumber']

interval\_length = interval\_block['ReadingType']['intervalLength']

readings = interval\_block['IReading']

# Creating a list of tuples to represent individual readings

for reading in readings:

# Converting StartTime and EndTime to IST format

start\_time\_str = reading['startTime']

end\_time\_str = reading['endTime']

# Using dateutil to parse the datetime strings and handle timezones

start\_time = parser.parse(start\_time\_str)

end\_time = parser.parse(end\_time\_str)

# Create time zones for UTC and IST

utc\_tz = tz.tzutc()

ist\_tz = tz.gettz('Asia/Kolkata')

# Convert to IST

start\_time\_ist = start\_time.astimezone(ist\_tz)

end\_time\_ist = end\_time.astimezone(ist\_tz)

# Remove the timezone information

start\_time\_ist = start\_time\_ist.replace(tzinfo=None)

end\_time\_ist = end\_time\_ist.replace(tzinfo=None)

#print("8")

data\_list.append((

start\_time\_ist,

end\_time\_ist,

reading['intervalLength'],

reading['value'],

reading['Quality'].get('noData', 'N/A'),

reading.get('flags', 'N/A'),

service\_point\_mrid,

meter\_mrid,

reading\_type\_mrid,

measurement\_type,

unit,

tou\_code,

channel\_number,

))

df = pd.DataFrame(data\_list, columns = df\_columns)

# count = 0

# #print("3")

# # Inside the loop, delete and insert data for the current meter

# for index, row in df.iterrows():

# # Perform delete using query where StartTime, EndTime, and meter\_mrid are the same

# delete\_query = f"DELETE FROM Smart\_Meter.dbo.MDM\_METER\_DATA WHERE Meter\_mRID = '{row['Meter\_mRID']}' AND Unit = '{row['Unit']}' AND MeasurementType = '{row['MeasurementType']}' AND ServicePoint\_mRID = '{row['ServicePoint\_mRID']}' AND StartTime = '{row['StartTime']}' AND EndTime = '{row['EndTime']}'"

# cursor.execute(delete\_query)

# connection.commit()

# insert\_query = '''

# INSERT INTO Smart\_Meter.dbo.MDM\_METER\_DATA (StartTime, EndTime, IntervalLength, Value, NoData, Flags,

# ServicePoint\_mRID, Meter\_mRID, ReadingType\_mRID, MeasurementType,

# Unit, TOUCode, ChannelNumber)

# VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

# '''

# # Pass the values as a tuple for insertion

# values\_tuple = (

# row['StartTime'], row['EndTime'], row['IntervalLength'], row['Value'], row['NoData'], row['Flags'], row['ServicePoint\_mRID'],

# row['Meter\_mRID'], row['ReadingType\_mRID'], row['MeasurementType'], row['Unit'], row['TOUCode'], row['ChannelNumber'],

# )

# cursor.execute(insert\_query, values\_tuple)

# count = count+1

# if count > 300:

# insert\_count = insert\_count+1

# connection.commit()

# print("Records Inserted",{insert\_count\*300})

# count = 0

# #print("4")

# connection.commit()

except KeyError as e:

# The 'Payload' key is missing in the API response, skip this meter and log the missed meter\_object\_id

print(f"Error processing Meter ID {meter\_object\_id}: {e}")

missed\_meters.append((

"2099-12-31 00:00:00",

"2099-12-31 00:00:00",

"0",

"0",

"true",

"1",

"0",

meter\_object\_id,

"Not Found",

"Not Found",

"Not Found",

"0",

"0",

))

else:

# API response status code is not 200, skip this meter and log the missed meter\_object\_id

print(f"Error fetching data for Meter ID {meter\_object\_id}. Status Code: {response.status\_code}")

missed\_meters.append((

"2099-12-31 00:00:00",

"2099-12-31 00:00:00",

"0",

"0",

"true",

"1",

"0",

meter\_object\_id,

"Not Found",

"Not Found",

"Not Found",

"0",

"0",

))

# Create a DataFrame from missed\_meters list

df\_missed = pd.DataFrame(missed\_meters, columns=df\_columns)

# print("Inserting the missed meters into MDM\_Meter\_Data....")

# count = 0

# for index, row in df\_missed.iterrows():

# # Perform delete using query where StartTime, EndTime, and meter\_mrid are the same

# delete\_query2 = f"DELETE FROM Smart\_Meter.dbo.MDM\_METER\_DATA WHERE Meter\_mRID = '{row['Meter\_mRID']}' AND Unit = '{row['Unit']}' AND MeasurementType = '{row['MeasurementType']}' AND ServicePoint\_mRID = '{row['ServicePoint\_mRID']}' AND StartTime = '{row['StartTime']}' AND EndTime = '{row['EndTime']}'"

# cursor.execute(delete\_query2)

# connection.commit()

# insert\_query2 = '''

# INSERT INTO Smart\_Meter.dbo.MDM\_METER\_DATA (StartTime, EndTime, IntervalLength, Value, NoData, Flags,

# ServicePoint\_mRID, Meter\_mRID, ReadingType\_mRID, MeasurementType,

# Unit, TOUCode, ChannelNumber)

# VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

# '''

# # Pass the values as a tuple for insertion

# values\_tuple2 = (

# row['StartTime'], row['EndTime'], row['IntervalLength'], row['Value'], row['NoData'], row['Flags'], row['ServicePoint\_mRID'],

# row['Meter\_mRID'], row['ReadingType\_mRID'], row['MeasurementType'], row['Unit'], row['TOUCode'], row['ChannelNumber'],

# )

# cursor.execute(insert\_query2, values\_tuple2)

# count = count+1

# print("4")

# if count > 300:

# insert\_count = insert\_count+1

# connection.commit()

# print("Records Inserted",{insert\_count\*300})

# count = 0

# print("5")

# connection.commit()

print("All records inserted successfully!")

cursor.close()

connection.close()

# Recording the end time

b = time.time()

# total time taken for the program

total\_time = b - a

print(f"Total time taken: {(total\_time/60):.2f} minutes")

except pyodbc.Error as e:

print("Error connecting to the database:", e)

**XML to EXCEL converter using PYQT5 GUI**

import sys

import xml.etree.ElementTree as ET

import pandas as pd

from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton, QFileDialog, QVBoxLayout, QWidget, QTableWidget, QTableWidgetItem

class XmlDataExtractorApp(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.init\_ui()

def init\_ui(self):

self.setWindowTitle("XML Data Extractor")

self.setFixedSize(800, 600) # Set the window size

self.open\_button = QPushButton("Open XML Files", self)

self.open\_button.clicked.connect(self.open\_files)

self.table\_widget = QTableWidget(self)

self.table\_widget.setColumnCount(14) # Increase the column count

self.table\_widget.setHorizontalHeaderLabels(["readingTypeId", "endTime", "value", "mRID", "Channel Number",

"pathname", "idtype", "source", "flags", "Interval Length",

"toubinnumber", "measurementtype", "Unit", "intervalStartTime"]) # Add a new column

self.export\_button = QPushButton("Export to Excel", self)

self.export\_button.clicked.connect(self.export\_to\_excel)

layout = QVBoxLayout()

layout.addWidget(self.open\_button)

layout.addWidget(self.table\_widget)

layout.addWidget(self.export\_button)

container = QWidget()

container.setLayout(layout)

self.setCentralWidget(container)

def open\_files(self):

options = QFileDialog.Options()

options |= QFileDialog.ReadOnly

file\_paths, \_ = QFileDialog.getOpenFileNames(self, "Open XML Files", "", "XML Files (\*.xml)", options=options)

self.dataframes = []

for file\_path in file\_paths:

with open(file\_path, "r") as file:

xml\_content = file.read()

df = self.extract\_data\_from\_xml(xml\_content)

self.dataframes.append(df)

self.combined\_df = pd.concat(self.dataframes, ignore\_index=True)

self.populate\_table\_widget()

def extract\_data\_from\_xml(self, xml\_content):

root = ET.fromstring(xml\_content)

namespace = {'ns': 'http://www.emeter.com/energyip/amiinterface'}

data\_list = []

for interval\_block in root.findall(".//ns:IntervalBlock", namespace):

reading\_type\_element = interval\_block.find(".//ns:readingTypeId", namespace)

if reading\_type\_element is not None:

reading\_type = reading\_type\_element.text

else:

reading\_type = None

# Extract data from ReadingType

reading\_type\_elem = interval\_block.find(".//ns:ReadingType", namespace)

measurement\_type\_elem = reading\_type\_elem.find(".//ns:measurementType", namespace)

tou\_bin\_number\_elem = reading\_type\_elem.find(".//ns:touBinNumber", namespace)

unit\_elem = interval\_block.find(".//ns:unit", namespace)

channel\_number\_elem = reading\_type\_elem.find(".//ns:channelNumber", namespace)

measurement\_type = measurement\_type\_elem.text if measurement\_type\_elem is not None else None

tou\_bin\_number = tou\_bin\_number\_elem.text if tou\_bin\_number\_elem is not None else None

unit = unit\_elem.text if unit\_elem is not None else None

channel\_number = channel\_number\_elem.text if channel\_number\_elem is not None else None

# Check if intervalLength is present before accessing its text

interval\_length\_elem = reading\_type\_elem.find(".//ns:intervalLength", namespace)

interval\_length = interval\_length\_elem.text if interval\_length\_elem is not None else None

# Extract data from IReading (in a loop for each interval)

i\_reading\_elements = interval\_block.findall(".//ns:IReading", namespace)

for i\_reading\_elem in i\_reading\_elements:

end\_time\_elem = i\_reading\_elem.find(".//ns:endTime", namespace)

interval\_length\_elem = i\_reading\_elem.find(".//ns:intervalLength", namespace)

value\_elem = i\_reading\_elem.find(".//ns:value", namespace)

flags\_elem = i\_reading\_elem.find(".//ns:flags", namespace)

end\_time = end\_time\_elem.text if end\_time\_elem is not None else None

interval\_length\_ireading = interval\_length\_elem.text if interval\_length\_elem is not None else None

value = value\_elem.text if value\_elem is not None else None

flags = flags\_elem.text if flags\_elem is not None else None

# Extract data from ServiceDeliveryPoint

service\_delivery\_point = root.find(".//ns:ServiceDeliveryPoint", namespace)

path\_name = service\_delivery\_point.find(".//ns:pathName", namespace).text

id\_type = service\_delivery\_point.find(".//ns:idType", namespace).text

# Extract data from Header

source = root.find(".//ns:Header/ns:source", namespace).text

# Extract data from Meter

meter = root.find(".//ns:Meter", namespace)

mrID\_elem = meter.find(".//ns:mRID", namespace)

mrID = mrID\_elem.text if mrID\_elem is not None else None

# Add the interval start time using the 'endTime' as the base

interval\_start\_time = end\_time\_elem.text.replace('T', 'T00:00:00.000+05:30')

data\_list.append({

"readingTypeId": reading\_type,

"endTime": end\_time,

"value": value,

"mRID": mrID,

"channelnumber": channel\_number,

"pathname": path\_name,

"idtype": id\_type,

"source": source,

"flags": flags,

"intervalLength": interval\_length\_ireading,

"toubinnumber": tou\_bin\_number,

"measurementtype": measurement\_type,

"unit": unit,

"intervalStartTime": interval\_start\_time

})

df = pd.DataFrame(data\_list)

return df

def populate\_table\_widget(self):

self.table\_widget.setRowCount(len(self.combined\_df))

for row\_idx, row in self.combined\_df.iterrows():

for col\_idx, value in enumerate(row):

item = QTableWidgetItem(str(value))

self.table\_widget.setItem(row\_idx, col\_idx, item)

def export\_to\_excel(self):

export\_path, \_ = QFileDialog.getSaveFileName(self, "Save Excel File", "", "Excel Files (\*.xlsx)")

if export\_path:

self.combined\_df.to\_excel(export\_path, index=False)

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

app = QApplication(sys.argv)

window = XmlDataExtractorApp()

window.show()

sys.exit(app.exec\_())

**DATA INSERTION – PYTHON TO SQL**

import pandas as pd

import pyodbc

import configparser as configparser

# Read the Excel file into a pandas DataFrame

excel\_file\_path = 'D:\Apoorv\MeterMDM.xlsx' # Replace with the actual file path

df = pd.read\_excel(excel\_file\_path)

prop\_file\_path = r"D:\Apoorv\MDM\scripts\prop.ini"

# Read the database connection information from the 'prop.ini' file

config = configparser.ConfigParser()

config.read(prop\_file\_path)

server\_name = config.get('DB\_CONN\_SMART\_METER', 'SERVER')

database\_name = config.get('DB\_CONN\_SMART\_METER', 'DATABASE')

driver = config.get('DB\_CONN\_SMART\_METER', 'DRIVER')

username = config.get('DB\_CONN\_SMART\_METER', 'UID')

password = config.get('DB\_CONN\_SMART\_METER', 'PWD')

connection\_string = f'DRIVER={driver};SERVER={server\_name};DATABASE={database\_name};UID={username};PWD={password}'

# Establish a connection to the database

connection = pyodbc.connect(connection\_string)

cursor = connection.cursor()

# Insert the data into the MDM\_Meter\_Master table

table\_name = 'MDM\_Meter\_Master'

for index, row in df.iterrows():

# Assuming the columns in the Excel file are the same as in the table

# If not, you may need to map the columns accordingly

insert\_query = f"""

INSERT INTO MDM\_Meter\_Master (Meter\_Object\_ID)

VALUES (?)

"""

values = tuple(row) # Assuming the order of columns in the DataFrame is the same as in the table

cursor.execute(insert\_query, values)

connection.commit()

# Close the connection

cursor.close()

connection.close()

print('Data inserted successfully!')

**METER REPLACEMENT – 3MNTH\_PRIOR, 3MNTH\_NXT, PERCENT\_CHG**

import pyodbc

import configparser as configparser

import time

import pandas as pd

#Load Configurations

prop\_file\_path = r"D:\Apoorv\Meter\_Replacement\prop.ini"

config = configparser.ConfigParser()

config.read(prop\_file\_path)

server\_name = config.get('METER\_REPLACEMENT', 'SERVER')

database\_name = config.get('METER\_REPLACEMENT', 'DATABASE')

driver = config.get('METER\_REPLACEMENT', 'DRIVER')

username = config.get('METER\_REPLACEMENT', 'UID')

password = config.get('METER\_REPLACEMENT', 'PWD')

connection\_string = f'DRIVER={driver};SERVER={server\_name};DATABASE={database\_name};UID={username};PWD={password}'

a = time.time()

try:

connection = pyodbc.connect(connection\_string)

cursor = connection.cursor()

# Execute the SQL query to fetch data from Order\_Details table in SAP database

query1 = """SELECT [Service Order\_CS\_ORDER],

[Business Partner\_BPARTNER],

[Contract Account Number\_/BIC/ZVKONT],

[Equipment Number\_EQUIPMENT],

[Maintenance Activity Type\_PMACTTYPE],

[Technical completion date\_/BIC/ZIDAT2],

[Maintenance planner group\_PMPLANGRP],

[Medium description\_TXTMD],

[Installation Number: Utilities Industry\_UCINSTALLA],

[Serial Number\_UCSERIALNR],

[Order type\_COORD\_TYPE],

[Maintenance processing stage\_/BIC/ZIPHAS],

[Work Center/Resource\_WORKCENTER],

[Regio Group\_/BIC/ZREGIOGRP],

[Regio Area\_/BIC/ZREGIOAR],

[Material\_MATERIAL]

from [SAP].dbo.[Order\_Details] where

[Maintenance Activity Type\_PMACTTYPE]='E07' and [Technical completion date\_/BIC/ZIDAT2] >= '2022-10-01';"""

df1 = pd.read\_sql\_query(query1, connection)

result\_data\_list = []

for index, row in df1.iterrows():

contract\_account\_number = row['Contract Account Number\_/BIC/ZVKONT']

technical\_completion\_date = row['Technical completion date\_/BIC/ZIDAT2']

# SQL query to fetch data from ZBIBD\_HIST\_Full table in SAB Database

# Here we map it with Order\_Details table to fetch common consumers

query2 = f"""

SELECT

ZH.[Contract Account Number\_CONSUMER\_NO] AS ConsumerID,

ZH.[End of Billing Period\_BILL\_TO\_DATE] AS To\_date,

ZH.[MMM-YY : (Ex - FEB-05 for Feburary 2005)\_BILL\_MONTH] AS BillMonth,

ZH.[Places Before Decimal Point in Meter Reading\_TOT\_UNITS\_BILLED] AS TotalUnitsBilled

FROM [SAP].[dbo].[ZBIBD\_HIST\_Full] ZH

WHERE ZH.[Contract Account Number\_CONSUMER\_NO] = '{contract\_account\_number}'

AND ZH.[End of Billing Period\_BILL\_TO\_DATE] BETWEEN DATEADD(MONTH, -3, '{technical\_completion\_date}')

AND DATEADD(MONTH, 3, '{technical\_completion\_date}')

"""

df2 = pd.read\_sql\_query(query2, connection)

df2['To\_date'] = pd.to\_datetime(df2['To\_date'])

# Function to calculate 3-month rolling averages

def calculate\_rolling\_average(consumer\_data):

rolling\_avg = []

for i in range(len(consumer\_data) - 2):

rolling\_avg.append(consumer\_data[i:i+3].mean())

return rolling\_avg

result\_data = []

grouped = df2.groupby('ConsumerID')

for consumer\_id, group in grouped:

group = group.sort\_values('To\_date')

tech\_completion\_date = row['Technical completion date\_/BIC/ZIDAT2']

# Calculate 3 months rolling averages based on comparison with tech\_completion\_date

prior\_group = group[group['To\_date'] < tech\_completion\_date]

next\_group = group[group['To\_date'] > tech\_completion\_date]

prior\_rolling\_avg = calculate\_rolling\_average(prior\_group['TotalUnitsBilled'].values)

next\_rolling\_avg = calculate\_rolling\_average(next\_group['TotalUnitsBilled'].values)

# Calculating the percentage change with the Prior and Next data

for i in range(min(len(prior\_rolling\_avg), len(next\_rolling\_avg))):

prior\_val = prior\_rolling\_avg[i] if i < len(prior\_rolling\_avg) else None

next\_val = next\_rolling\_avg[i] if i < len(next\_rolling\_avg) else None

if prior\_val is not None and next\_val is not None:

prior\_avg = sum(prior\_rolling\_avg) / len(prior\_rolling\_avg)

next\_avg = sum(next\_rolling\_avg) / len(next\_rolling\_avg)

if prior\_avg != 0:

percentage\_change = ((next\_avg - prior\_avg) / prior\_avg) \* 100

else:

percentage\_change = float('inf') if next\_avg != 0 else float('nan')

else:

percentage\_change = None

result\_data.append({

'ConsumerID': consumer\_id,

'MeterNo': row['Serial Number\_UCSERIALNR'],

'Installation\_Number': row['Installation Number: Utilities Industry\_UCINSTALLA'],

'Technical\_Completion\_Date': tech\_completion\_date,

'3Mnths\_Prior': prior\_val,

'3Mnths\_Next': next\_val,

'Percentage\_Chng': percentage\_change

})

result\_df = pd.DataFrame(result\_data)

result\_data\_list.append(result\_df)

# Appending all the data from loop into a single dataframe

df3 = pd.concat(result\_data\_list, ignore\_index=True)

# Recording the end time

b = time.time()

# total time taken for the program

total\_time = b - a

print(f"Total time taken: {(total\_time/60):.2f} minutes")

except pyodbc.Error as e:

print("Error connecting to the database:", e)