Python+Bash Script for Data Dictionary(.XLSX) validation with Postgres Data Warehouse

By:

Kshitij Wagle

Are your analytics facing challenges related to discrepancies and validation? Explore these situations:

- Are manual validation methods introducing errors and inefficiencies?
- Does simultaneous ETL development lead to ETL code regressions impacting your data warehouse?
- Is there confusion validating tables and fields amidst ongoing development and feature additions?
- Are unclear guidelines causing inconsistencies in documentation?

If you are facing above issues, then this article is for you. I also have identified above issues in my own experience.

For simplicity, lets consider you use following tools in your analytics team.

Data Warehouse: PostgreSQL

ETL Tool: Talend for Data Integration

Visualization and Reporting: Power BI

Documentation/Data Dictionary: ZenDesk

It is because my article will be based on above tools most of the time, and they are pretty simple to understand. Please note, the script development process explained below will use postgresql python library.

In this article I am going to demonstrate the entire steps to develop a batch script to automate your data validation to streamline your analytics workflow.

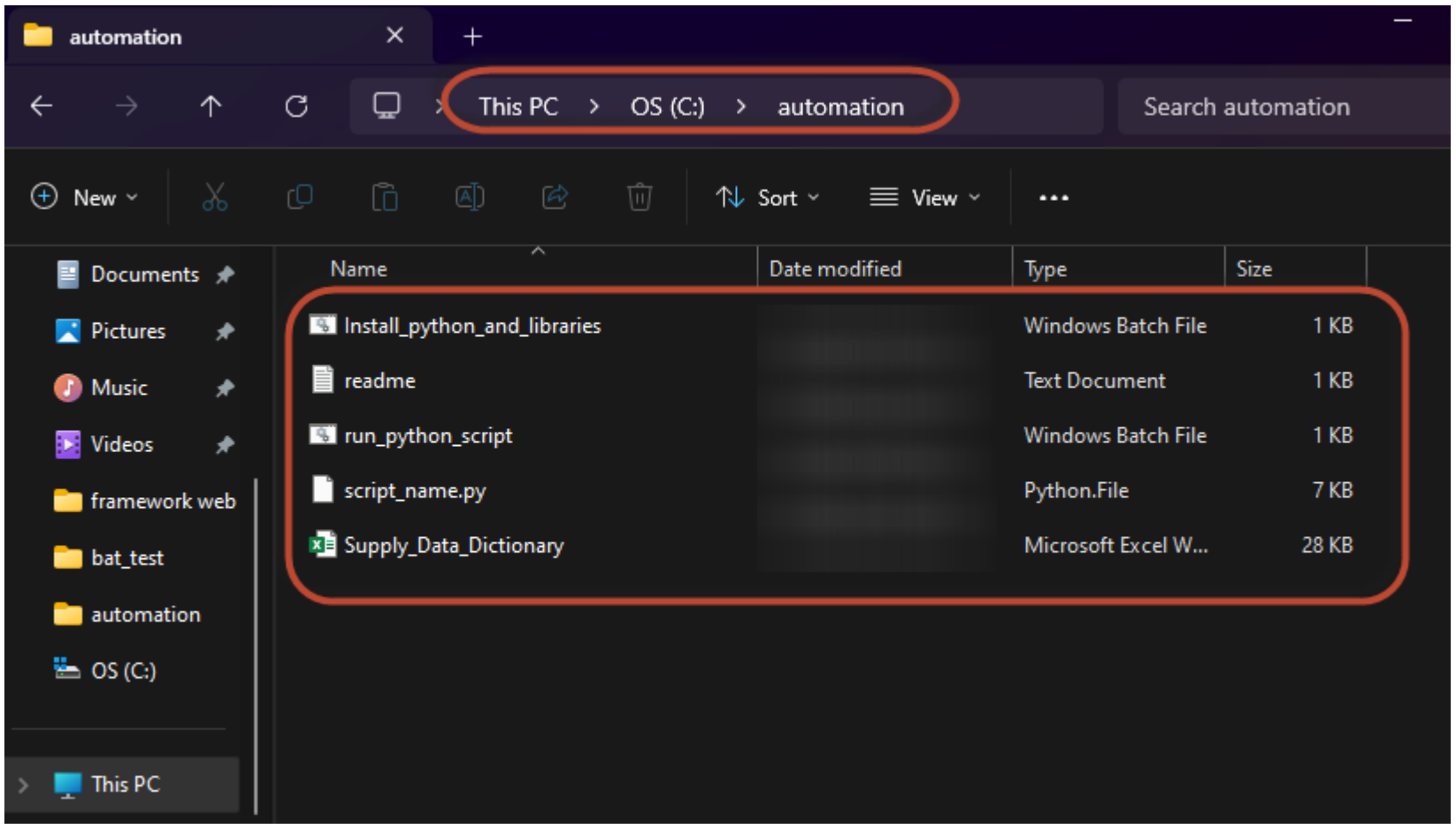
What this program does?

This program simplifies analytics challenges by offering a user-friendly interface for easy schema selection. It generates clear .xlsx reports, allowing quick identification of existing and missing tables and fields (you can copy and paste data dictionary in .xlsx format and this program detects the table and fields and compares with your data warehouse). By comparing data dictionaries with your data warehouse, it strengthens data governance and facilitates prompt resolution to the underlying issues in your your data warehouse/ETL process/documentation.

Preparation :

Download this file from GitHub: github link

Put all files and programs under: *C:/automation*, and your window should look like following:



I am assuming your table views which have prefix v_ (i.e. v_Supply_Articles) and they are in Zendesk. Then you can paste your ZenDesk data dictionary into excel file. The only thing you must consider is your table name and fields should be in first column, as this program is developed in this way.

TLDR: Quick Steps:

💡 **step 1:**

Install python and required libraries by running:----Install_python_and_libraries.bat ----

💡 **Step 2:**

Run main bat file to run the python script ---run_python_script.bat---

💡 **Step 3:**

Enter the password for your database.

💡 **Step 4:**

Enter the name of the schema you want to be in

💡 **Step 5:**
Save the report in your desired location (give the report name-- e.g. supply_report)

It will save the report.

Note: When executing the process second time, ignore first step.

Detailed Steps, and what's inside it?

Step 1: Install Python and Libraries

Program Name: *Install_python_and_libraries.bat*

Libraries required are: PostgreSQL (psycopg2) and .xlsx reader (openpyxl)

This program installs the python, required libraries, and then sets the environment variable.

```
@echo off
setlocal enabledelayedexpansion

REM Define variables
set "python_version=3.12.2"
set "install_path=C:\Python312"
set "libs=openpyxl psycopg2"

REM Install Python
echo Installing Python %python_version%...
curl -o python_installer.exe https://www.python.org/ftp/python/%python_version%/python-%python_v
start /wait python_installer.exe /quiet InstallAllUsers=1 PrependPath=1 TargetDir=%install_path%
del python_installer.exe

REM Install Python libraries
echo Installing Python libraries...
%install_path%\Scripts\pip install %libs%

REM Set environmental variables
echo Setting up environmental variables...
setx PATH "%install_path%;%install_path%\Scripts;" /M

echo Installation complete.
```

Step 2: Run batch file

Program Name: *run_python_script.bat*

This programs runs the python script , note that you need to create folder: C:\automation , and put every programs and your .xlsx data dictionary in the same folder.

```
@echo off

REM Get the current directory of the script
set "script_folder=%~dp0"

REM Change the directory to the script's folder
cd /d "%script_folder%"

REM Run the Python script
python script_name.py
```

```
pause
```

Above batch program runs following python script:

script_name.py

```
import openpyxl
import psycopg2
from psycopg2 import sql
from tkinter import filedialog
from tkinter import Tk
from datetime import datetime
import getpass  # Import the getpass module

# PostgreSQL connection details
db_server = "enter your server" #e.g. yourdwh.yourdomain.com.au
db_port = 1234 # replace 1234 with the port number
db_name = "Enter database name" # exampleDWH
db_user = "enter username" #e.g. postgres1
db_password = getpass.getpass("Enter your database password: ")  # Prompt user for password

# Function to check table existence in a specific schema
def table_exists_in_schema(cursor, schema_name, table_name):
    query = sql.SQL("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE table_schema =
        sql.Identifier(schema_name), sql.Identifier(table_name))
    cursor.execute(query, (schema_name, table_name))
    return cursor.fetchone()[0]

# Function to check field existence in a table in a specific schema
def field_exists_in_schema(cursor, schema_name, table_name, field_name):
    query = sql.SQL("SELECT EXISTS (SELECT 1 FROM information_schema.columns WHERE table_schema
        sql.Identifier(schema_name), sql.Identifier(table_name), sql.Identifier(field_name))
    cursor.execute(query, (schema_name, table_name, field_name))
    return cursor.fetchone()[0]

# Custom function for case-insensitive schema name conversion
def convert_to_valid_schema(user_input):
    valid_schemas = ["Common", "Finance", "HR", "ML", "Maintenance", "Production", "Supply", "pu
    lowercased_input = user_input.lower()

    for schema in valid_schemas:
        if lowercased_input == schema.lower():
            return schema

    print("Invalid schema name. Please enter a valid schema name.")
    exit()

# Connect to PostgreSQL
try:
    # Prompt user for the schema and convert to valid format
    user_schema = convert_to_valid_schema(input("Enter the table schema e.g Finance: "))


    # Open file dialog to select Excel file
    root = Tk()
    root.withdraw()  # Hide the main window
    xlsx_file_path = filedialog.askopenfilename(title="Select Excel file", filetypes=[("Excel fi
```

```
connection = psycopg2.connect(
    host=db_server,
    port=db_port,
    database=db_name,
    user=db_user,
    password=db_password
)

with connection.cursor() as cursor:
    views = []
    missing_tables = set()

    # Set to store unique tables and fields encountered in the Excel file
    excel_tables_and_fields = set()

    # Read Excel file and check table and field existence
    workbook = openpyxl.load_workbook(xlsx_file_path)

    for sheet_name in workbook.sheetnames:
        sheet = workbook[sheet_name]

        table_name = None
        skip_table_description = False

        for row in sheet.iter_rows(min_row=2, values_only=True):
            cell_value = row[0].strip() if row[0] else None

            if cell_value and cell_value.startswith("v_"):
                table_name = cell_value
                skip_table_description = True
                continue

            if table_name and cell_value == "Field":
                skip_table_description = False
                continue

            if skip_table_description:
                continue

            if table_name and cell_value:
                field_name = cell_value.strip()
                if table_name.startswith("v_"):
                    category_list = views
                    excel_tables_and_fields.add((table_name, field_name))
                else:
                    continue

                table_existence = table_exists_in_schema(cursor, user_schema, table_name)
                category_list.append((table_name, table_existence, field_name,
                                      field_exists_in_schema(cursor, user_schema, table_name

    # Check for missing tables and fields in the data warehouse
    cursor.execute(sql.SQL("SELECT table_name, column_name FROM information_schema.columns W
    warehouse_tables_and_fields = set(cursor.fetchall())

    missing_tables_and_fields = warehouse_tables_and_fields - excel_tables_and_fields
```

```
        # Print results based on category
        print("\nResults:")
        print("\033[94mViews:\033[0m")

        # Print existing views in alphabetical order
        views.sort(key=lambda x: x[0])
        for table, exists, field, field_existence in views:
            print(f"{table}: {'\033[92mExists\033[0m' if exists else '\033[91mDoes not exist\033
            if field:
                print(f"  - {field}: {'\033[92mExists\033[0m' if field_existence else '\033[91mD

        # Print missing tables and fields for views in alphabetical order
        missing_tables_and_fields = sorted(
            [item for item in missing_tables_and_fields if item[0].startswith("v_")],
            key=lambda x: (x[0], x[1])
        )
        print("\n\033[94mMissing Tables and Fields:\033[0m")

        current_table = None  # To keep track of the current table being printed
        for table_name, field_name in missing_tables_and_fields:
            if table_name != current_table:
                if current_table:
                    print()  # Add a newline between different tables
                current_table = table_name

            if field_name:
                print(f"{table_name} - {field_name}: \033[91mMissing in Excel file\033[0m")
            else:
                print(f"{table_name}: \033[91mMissing in Excel file\033[0m")

        # Create a new Excel workbook
        output_workbook = openpyxl.Workbook()

        # Remove the default "Sheet" created by openpyxl
        default_sheet = output_workbook["Sheet"]
        output_workbook.remove(default_sheet)

        # Create sheets in the workbook
        views_sheet = output_workbook.create_sheet("Data Dictionary")
        missing_tables_sheet = output_workbook.create_sheet("Data Warehouse")

        # Add header rows to the sheets
        views_sheet.append(["Table", "Table Existence in DWH", "Field", "Field Existence in DWH'
        missing_tables_sheet.append(["Table", "Field", "Status"])

        # Print results based on category to the 'Views' sheet
        for table, exists, field, field_existence in views:
            views_sheet.append([table, 'Exists' if exists else 'Does not exist', field, 'Exists

        # Print missing tables and fields for views to the 'Missing Tables and Fields' sheet
        for table_name, field_name in missing_tables_and_fields:
            missing_tables_sheet.append([table_name, field_name, 'Missing in Excel file' if fiel

        # Prompt user to choose where to save the .xlsx file
        output_filename = filedialog.asksaveasfilename(defaultextension=".xlsx", filetypes=[("Ex

        # Save the output Excel file
        output_workbook.save(output_filename)
```

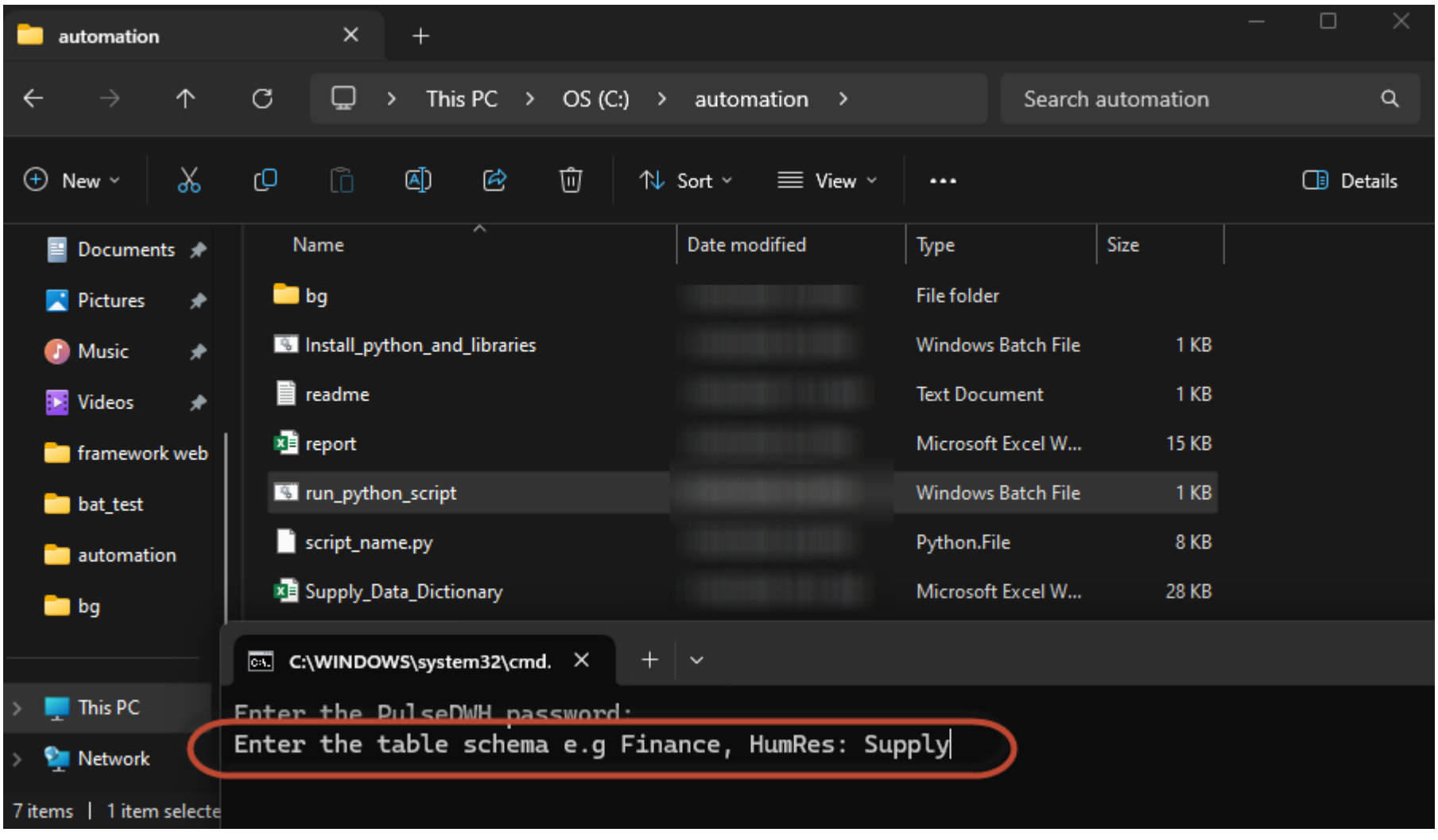
```
        print(f"\nResults have been saved to '{output_filename}'.")

except psycopg2.Error as e:
    print(f"\033[91mError: {e}\033[0m")

finally:
    if connection:
        connection.close()
```

Step 3: Enter the Schema

You are required to enter the schema name to compare the data dictionary with data warehouse. If you want to validate Supply Data Dictionary with Data Warehouse, then enter ‘Supply’



Step 4: Select Data Dictionary

In the data dictionary within the XLSX file, the tables and fields must be placed in the first column for this program to function correctly.

How to create it?

You can copy and paste the Zendesk data dictionary into an Excel sheet (.xlsx) and provide the file name for the associated data dictionary module.

For example, I created an Excel sheet where I pasted the data dictionary from Zendesk(Data Dictionary for Supply Module), and named it "Supply_Data_Dictionary," and structured it as follows.

Note: In this structure, tables and fields are in the first column. The program disregards table descriptions and headers but reads the table name and its fields.

AutoSave Off | Supply_Data_Dictionary | Search

Table Dictionary

Supply Tables

v_Supply_Articles

Field	Data Type	Description	Key_Type
Article_ID	Integer	Unique identifier for each supply article.	PK
Title	Character	Title of the supply article.	
Content	Text	The actual content of the supply article.	
Author_ID	Integer	Foreign Key referencing the v_Supply_Authors table.	FK
Category_ID	Integer	Foreign Key referencing the v_Supply_Categories table.	FK
Publication_Date	DateTime	Date and time when the article was published.	

v_Supply_Authors

Field	Data Type	Description	Key_Type
Author_ID	Integer	Unique identifier for each supply author.	PK
Author_Name	Character	Name of the supply author.	
Email	Character	Email address of the supply author.	
Bio	Text	Short biography or description of the supply author.	

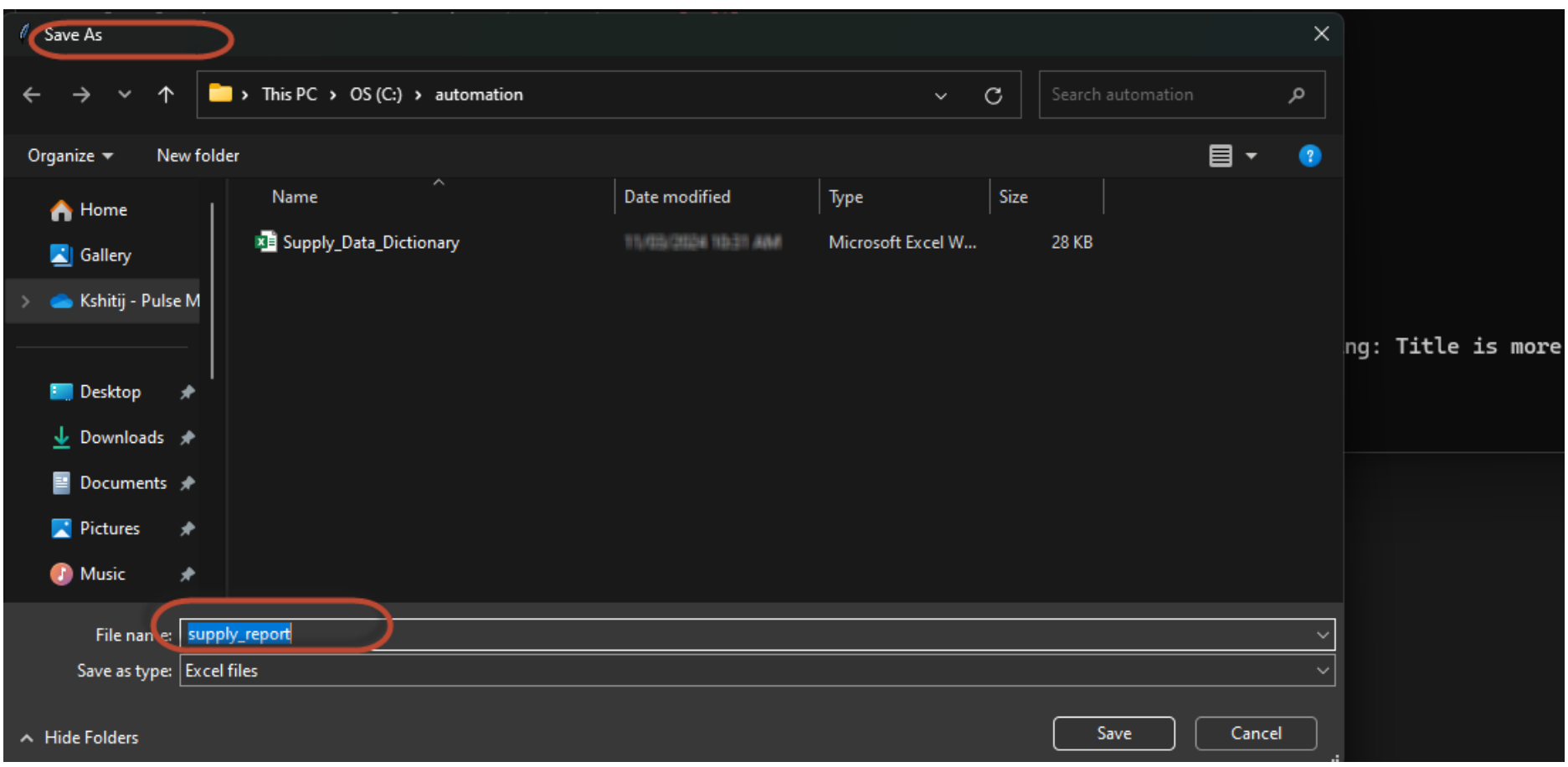
v_Supply_Categories

Field	Data Type	Description	Key_Type
Category_ID	Integer	Unique identifier for each supply category.	PK
Category_Name	Character	Name of the supply category.	
Description	Text	Description of the supply category.	

Sheet4

After selecting the file, you will get following result with save as window:

Enter the name for the report (e.g. supply_report) and select the folder you want to save it. it will then saves the report in .xlsx format with multiple sheets.



The file is saved in following location:

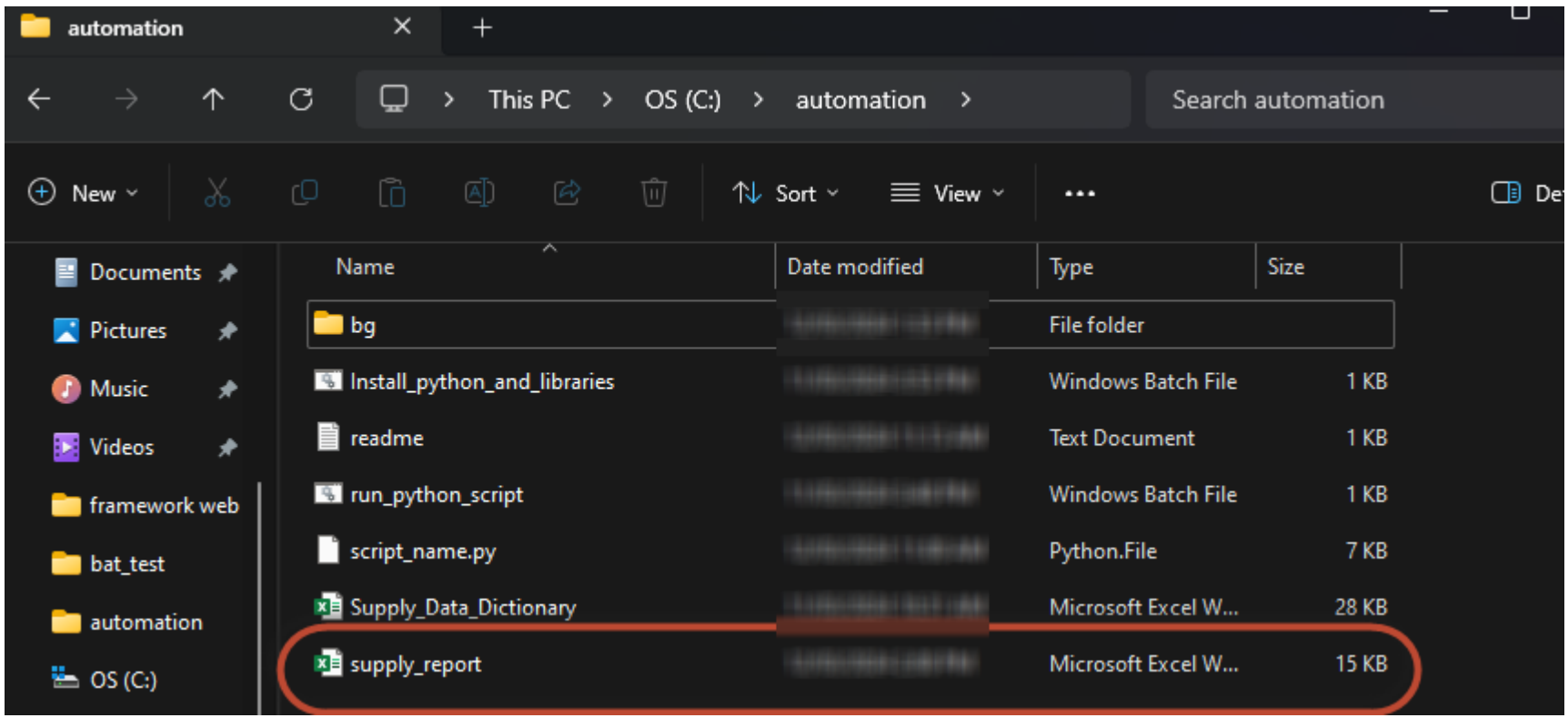
```
C:\Users\kwagle\AppData\Roaming\Python\Python312\site-packages\openpyxl\workbook\child.py:99: UserWarning: Title is more
 than 31 characters. Some applications may not be able to read the file
  warnings.warn("Title is more than 31 characters. Some applications may not be able to read the file")

Results have been saved to 'C:/automation/supply_report.xlsx'.
Press any key to continue . . .
```

Step 5: View the report

The report will have two sheets

1. Data Dictionary: The report in this section contains information whether the tables and fields that exists in data dictionary exists or not in Data Warehouse (Compares Data Dictionary against Data Warehouse)
2. Data Warehouse: It provides report about the missing tables and fields if they are not in data dictionary. (i.e. compares Data Warehouse against Data Dictionary)
3.



Conclusion:

In summary, the automated Data Dictionary validation streamlines and improves the validation process, empowering team members to effectively utilize the program. It addresses challenges such as manual validation, ETL code regression, ambiguity in validating tables and fields, and inconsistent documentation practices. The user-friendly interface and detailed reports contribute to enhanced efficiency and data governance, providing clear insights for team members and ensuring the reliability of the validation system