|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | A black text with a white background  Description automatically generated |

CONFIGURAZIONE

|  |  |
| --- | --- |
| Nome stream | Governance Metadata |
| Nome Progettualità | Governance Metadata |
| Sottotitolo Progettualità |  |

|  |  |
| --- | --- |
|  |  |
| Data del documento: | 18/12/23 |

|  |  |
| --- | --- |
| Cliente | Ministero della Giustizia |
| Progetto |  |
| Documento |  |
| Ambito |  |
| Versione | 1.0 |
| Stato | Prima versione |
| Società | Accenture |
| Data | 18/12/23 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revisioni | | | | |
| Ver. | Data | Descrizione | Autore | Approvazione |
| 1.0 | 18/12/23 | Prima versione documento | Nursaule Myrzashova |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |
| --- | --- |
| Lista di Distribuzione | |
| D.G.S.I.A. (RUP) | Nome Cognome |
| D.G.S.I.A. (DEC) | Nome Cognome |
|  |  |
|  |  |

**Summary**

[**Documentazione Progetto** 4](#_Toc153898054)

[**1** **SOFTWARE PREREQUISITES** 5](#_Toc153898055)

[**1.1** **Development prerequisites** 5](#_Toc153898056)

[**2** **Setup Pyapacheatlas** 6](#_Toc153898057)

[**3** **Project folder and python files description** 6](#_Toc153898058)

[3.1 atlas\_type.py 6](#_Toc153898059)

[3.2 auth.py 7](#_Toc153898060)

[3.3 business\_metadata.py 7](#_Toc153898061)

[3.4 column\_entities.py 9](#_Toc153898062)

[3.5 lineage.py 13](#_Toc153898063)

[3.6 preproccessing.py 15](#_Toc153898064)

[3.7 table\_entities.py 17](#_Toc153898065)

[3.8 main.py 19](#_Toc153898066)

## **Documentazione Progetto**

|  |  |
| --- | --- |
| **Documento** | **Descrizione** |
| Accordo Quadro del 04/03/2022 e relativi Allegati |  |
| Ministero Giustizia - Contratto Esecutivo |  |
| Piano Operativo |  |
| Piano dei Fabbisogni |  |
| Documenti di requisiti dell’Amministrazione |  |
| Altri allegati |  |

# **SOFTWARE PREREQUISITES**

1. Apache Atlas 2.3.0
2. Cloudera 7.1.7
3. Python 3.11.4 or higher

## **Development prerequisites**

The following software is required for development:

* Python 3.11.4 or higher
* Visual Studio Code for python ([Download Visual Studio Code](https://code.visualstudio.com/download))
* Docker engine for running the Apache Atlas image.
* (Optional) Apache Ranger 2.3.0 for data governance access/modify control option.

# **Setup Pyapacheatlas**

For fast downloading the pyapacheatlas, run the next command in terminal:

python -m pip install pyapacheatlas

Basic Auth for Apache Atlas

You'll need to collect your username and password as well as the Atlas API endpoint. The code below expects you to have ATLAS\_USERNAME and ATLAS\_PASSWORD stored as environment variables or hard coded in the script.

In our case was used Basic Authentication method with the next endpoint:

https://atlas.dai-training.com/api/atlas/v2

A computer screen shot of text

Description automatically generated

# **Project folder and python files description**

## atlas\_type.py

Contains code for retrieving names of tables with “hive\_table” type from Hive Data Platform and convert them into the list with proper format for the next use.

Code contains beeline command to connect to the hive and retrieve the only hive tables’ names from the storage. After that the name were formatted from Json format into the list format by eliminating spaces and “|” chars from the given output.

A black background with colorful text

Description automatically generated

## auth.py

Contains authentication method and endpoint for pyapacheatlas. The code can be found in SETUP PYAPACHEATLAS section.

## business\_metadata.py

Contains implementation of the next functions:

1. create\_bisness\_metadata\_table
2. create\_buisness\_metadata\_for\_column.

The following functions create business metadata with corresponding attributes from excel file in Profile/Administration field in Apache Atlas. The implementation code you can find bellow:

A computer screen shot of a computer code

Description automatically generated

A computer screen shot of text

Description automatically generated

In the **cerate\_bisness\_metadata\_table** function we removed 4 fields from the “Istruzioni” dataset ('Nome del Dataset', 'Descrizione Dataset', "Nome dell'Attributo", "Descrizione dell'attributo"), because it’s not business metadata required fields but technical ones. Also, we replaced not supported characters as ‘/’ and ‘à’ in the names of attributes as Atlas API do not accept this chars. In the end we append new formatted attributes in **attributeDefs** list and created Business Metadata for tables with these attributes.

In the **create\_bisness\_metadata\_for\_column** function the two parameters are received: **business\_metadata\_dataset** and **column\_keys**. We checked which column attributes belong to business metadata, and by this parameter added columns’ business metadata camps in the **matched\_list.** After converting attributes’ names to applicable format, we add each attribute in attributeDefs list by converting into AtlasAttributeDef type. In the end we created Business Metadata for columns with these attributes.

The result example:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## column\_entities.py

Contains implementation of the next functions:

1. create\_entities\_column(formatted\_column\_dataset, dataset\_dict\_records)
2. delete\_entitties(entities)
3. delete\_column\_from\_atlas(formated\_column\_dataset, client, dataset\_dict\_records)
4. **create\_entities\_column** function receives two parameters: **formatted\_column\_dataset** – dataset containing all column data in dictionary format with table to which they are belong and **dataset\_dict\_records** which contains tables name with their generated qualified names.

First the function checks if tables from dataset\_dict\_records belong to the type of hive\_table by checking if table belongs to the list that we received from hive storage in atlas\_type.py script above. If its not belong to this list, we assign another type to the table (for example cudu\_table in future).

Then we check if columns already exist in the Atlas system by sending its generated qualified name to Atlas, else we create Atlas entity with hive\_column type and send to the server.

By sending new column to the server, the next attributes were created:

1. technical attributes – type, data\_type, description, isOptional
2. relationship attributes – table to which the column is belong
3. business attributes – Dati Sensibili, Range or Domino Valori
4. custom attributes (user-defined attributes) – nullabillity

The code:

def create\_entities\_column(formated\_column\_dataset, dataset\_dict\_records):

    list\_of\_qualified\_names\_with\_type = {}

    dict\_name\_guid = {}

    for value in dataset\_dict\_records:

        if value["Nome del Dataset"].lower() in list\_hive\_table\_type:

            value['Atlas Type'] = "hive\_table"

        else:

            # print("it false hive\_table")

            value['Atlas Type'] = "hive\_table"

        list\_of\_qualified\_names\_with\_type[value['QualifiedName']] = value['Atlas Type']

    for qName in list\_of\_qualified\_names\_with\_type:

        entities = client.get\_entity(

        qualifiedName=qName,

        typeName=list\_of\_qualified\_names\_with\_type[qName]

        )

        e = entities.get("entities")

        # print(json.dumps(e, indent=2))

        if e != None:

            dict\_name\_guid[qName] = e[0]["guid"]

    for table in formated\_column\_dataset:

        for column in formated\_column\_dataset[table]:

            if table.lower() in list\_hive\_table\_type:

                column['Atlas Type'] = "hive\_column"

            else:

                # print("it false hive\_column")

                column['Atlas Type'] = "hive\_column"

            name = column["Nome dell'Attributo"]

            entity\_from\_server = client.get\_entity(

            qualifiedName = f'siamm://{table.lower()}\_{name}\_col',

            typeName = column['Atlas Type']

            )

            if entity\_from\_server.get("entities") != None:

                for entity in entity\_from\_server.get("entities"):

                    jsonData =json.dumps(entity, indent=2)

                    readJSON = json.loads(jsonData)

                    # print(readJSON)

                    print(f"Entity {readJSON['attributes']['name']} column already exists in Atlas")

            else:

                for qualified\_name in dict\_name\_guid:

                    if f'siamm://{table.lower()}' == qualified\_name:

                        name = name

                        ae = AtlasEntity(

                        name = name,

                        typeName = column['Atlas Type'],

                        qualified\_name = f'siamm://{table.lower()}\_{name}\_col',

                        attributes={

                            "type" : 'string',

                            "data\_type": column['Formato'],

                            "description": column["Descrizione dell'attributo"],

                            "isOptional": True

                        },

                        relationshipAttributes = {'table': {'typeName': 'hive\_table', 'guid': dict\_name\_guid[qualified\_name], 'qualifiedName': f'siamm://{table.lower()}'}},

                        guid = -1

                        )

                        ae.addBusinessAttribute(SIAMM\_business\_metadata\_for\_column\_main={"Dati Sensibili":column["Dati Sensibili"],

                                                                            "Range or Dominio Valori":column["Range/Dominio Valori"],

                                                                            })

                        ae.addCustomAttribute(nullability=column["Nullability"])

                        resp = client.upload\_entities([ae])

                        guid = resp["guidAssignments"]["-1"]

                        jsonData3 = json.dumps(resp,indent=2)

                        print(jsonData3)

                        print(json.dumps(client.get\_single\_entity(guid),indent=2))

                        readJSON3 = json.loads(jsonData3)

                        print(f"Column was created")

Result example:

A screenshot of a computer

Description automatically generated

1. **delete\_entitties** function receives parameter entities (Atlas Entity type) and delete this entity from the server by its guid.

def delete\_entitties(entities):

    for entity in entities:

        guid\_from\_server = entity["guid"]

        delete\_response = client.delete\_entity(guid=guid\_from\_server)

        jsonDelete = json.dumps(delete\_response, indent=2)

        print(jsonDelete)

        readJSON = json.loads(jsonDelete)

        print("Column was deleted...")

1. **delete\_column\_from\_atlas** function looks for the columns that were deleted from the excel file and delete these columns from the Atlas dashboard. We check that table exists in the Atlas by its qualified name and then retrieve all columns that already in the Atlas. Then we check all the columns in excel file and find the difference between existing columns in excel and Atlas. If there are columns in the **not\_matched\_list** we get these columns and delete them.

def delete\_column\_from\_atlas(formated\_column\_dataset, dataset\_dict\_records):

    print("Script is looking for deleted columns in exel and delete them from database, be patient...")

    list\_of\_qualified\_names = []

    dict\_qualified\_name = []

    existing\_columns\_qual\_name =[]

    qualified\_names\_in\_exel =[]

    for value in dataset\_dict\_records:

        list\_of\_qualified\_names.append(value['QualifiedName'])

    for qulified\_name in list\_of\_qualified\_names:

        entities = client.get\_entity(

        qualifiedName=qulified\_name,

        typeName="hive\_table"

        )

        e = entities.get("entities")

        # print(json.dumps(e, indent=2))

        if e != None:

            dict\_qualified\_name.append(e[0]["relationshipAttributes"]["columns"])

            for value in dict\_qualified\_name:

                for i in value:

                    existing\_columns\_qual\_name.append(i["qualifiedName"])

    existing\_columns\_qual\_name = list(set(existing\_columns\_qual\_name))

    for table in formated\_column\_dataset:

        for column in formated\_column\_dataset[table]:

            name = column["Nome dell'Attributo"]

            qualifiedName = f'siamm://{table.lower()}\_{name}\_col'

            qualified\_names\_in\_exel.append(qualifiedName)

    non\_matched\_list = list(set(existing\_columns\_qual\_name)-set(qualified\_names\_in\_exel))

    # print(non\_matched\_list)

    if len(non\_matched\_list) != 0:

        entities\_non\_matched\_list = client.get\_entity(

            qualifiedName=non\_matched\_list,

            typeName="hive\_column"

            )

        # print(entities\_non\_matched\_list)

        if any(entities\_non\_matched\_list):

            delete\_entitties(entities\_non\_matched\_list.get("entities"))

        else:

            print("No columns were deleted in excel file")

## lineage.py

Contains implementation of the next functions:

1. create\_lineage\_for\_two\_tables(client)
2. create\_lineage\_for\_ingestion(client)

The first function creates sample lineage for JOIN process between two tables ds\_siamm\_dc\_beneficiario\_tab\_fat and ds\_siamm\_dc\_personafisicasa\_tab\_dim by showing the result in the ds\_siamm\_dc\_parte\_procedimento\_tab\_fat table.

def create\_lineage\_for\_two\_tables(client):

    entities = client.get\_entity(

        qualifiedName=["siamm://ds\_siamm\_dc\_beneficiario\_tab\_fat", "siamm://ds\_siamm\_dc\_personafisicasa\_tab\_dim", "siamm://ds\_siamm\_dc\_parte\_procedimento\_tab\_fat"],

        typeName="hive\_table"

    )

    beneficiario\_entity = entities.get("entities")[0]

    personaFisica\_entity = entities.get("entities")[1]

    parte\_procedimento = entities.get("entities")[2]

    process = AtlasProcess(

        name="JOIN sample process in lineage",

        typeName="Process",

        qualified\_name="pyapacheatlas://democustomprocess",

        inputs=[beneficiario\_entity, personaFisica\_entity],

        outputs=[parte\_procedimento],

        guid="-1"

    )

    results = client.upload\_entities(

        batch=[process]

    )

    print(json.dumps(results, indent=2))

Result example:

A diagram of a diagram

Description automatically generated

The second function creates sample lineage between DS\_SIAMM\_DR\_PERSONAFISICASA\_TAB\_DIM and DS\_SIAMM\_DQ\_PERSONAFISICASA\_TAB\_DIM tables by connecting them and adding Ingestion process, Data quality process.

def create\_lineage\_for\_ingestion(client):

    entity\_from\_server = client.get\_entity(

        qualifiedName=["siamm://ds\_siamm\_dr\_personafisicasa\_tab\_dim"],

        typeName="hive\_table"

    )

    entities = client.get\_entity(

        qualifiedName=["siamm://ds\_siamm\_dc\_personafisicasa\_tab\_dim"],

        typeName="hive\_table"

    )

    personaFisica\_entity = entities.get("entities")[0]

    input01 = AtlasEntity(

        name="DS\_SIAMM\_DR\_PERSONAFISICASA\_TAB\_DIM",

        typeName="hive\_table",

        qualified\_name="siamm://ds\_siamm\_dr\_personafisicasa\_tab\_dim",

        guid="-10000003"

    )

    input02 = AtlasEntity(

        name="DS\_SIAMM\_DQ\_PERSONAFISICASA\_TAB\_DIM",

        typeName="hive\_table",

        qualified\_name="siamm://ds\_siamm\_dq\_personafisicasa\_tab\_dim",

        guid="-1000000"

    )

    process1 = AtlasProcess(

        name="Ingestion process",

        typeName="Process",

        qualified\_name="pyapacheatlas://democustomprocess2",

        inputs=[input01],

        outputs=[input02],

        guid="-100001"

    )

    process2 = AtlasProcess(

        name="Data quality process",

        typeName="Process",

        qualified\_name="pyapacheatlas://democustomprocess3",

        inputs=[input02],

        outputs=[personaFisica\_entity],

        guid="-1000002"

    )

    results = client.upload\_entities(

        batch=[input01, input02, process1, process2]

    )

    print(json.dumps(results, indent=2))

Result example:

A screenshot of a computer

Description automatically generated

## preproccessing.py

Contains implementation of the next functions:

1. create\_qualifiedname(dataset)
2. read\_exel\_tech\_business\_metadata(path)
3. divide\_columns\_by\_type(list\_of\_metadata)
4. convert\_column\_to\_dict\_all(path, sheet\_name)
5. dataframe\_formatting(columns\_dict\_records)
6. create\_qualifiedname function takes the names of the table and create qualified name for each one for future send to the Atlas server.

def create\_qualifiedname(dataset):

    for value in dataset:

        qualifiedName = f"siamm://{value['Nome del Dataset'].lower()}"

        value['QualifiedName'] =  qualifiedName

    print("The qualified names were created for each table...")

    return dataset

1. read\_exel\_tech\_business\_metadata function reads excel fields from sheet “Istruzioni” and convert them to dictionary in proper format.

def read\_exel\_tech\_business\_metadata(path):

    siamm\_dataframe = pd.read\_excel(path, sheet\_name="Istruzioni", header=None)

    dataframe\_dict\_records = siamm\_dataframe.to\_dict(orient='records')

    values\_dict = {}

    for values in dataframe\_dict\_records[3:]:

        values\_dict[values[0]] = values[5]

    print("The Excel file with metadata was transformed to a dictionary...")

    return values\_dict

1. divide\_columns\_by\_type function divides the dictionary on technical and business metadata by its keys names and return two lists.

def divide\_columns\_by\_type(list\_of\_metadata):

    business\_fillds\_list = []

    tech\_operational\_fillds\_list = []

    for key in list\_of\_metadata:

        if list\_of\_metadata[key] == 'Business':

            business\_fillds\_list.append(key)

        elif list\_of\_metadata[key] == 'Tecnico' or list\_of\_metadata[key] == 'Operational':

            tech\_operational\_fillds\_list.append(key)

    # print(business\_fillds\_list)

    # print(tech\_operational\_fillds\_list)

    print("Meta data was devided on technial and business fields...")

    return business\_fillds\_list, tech\_operational\_fillds\_list

1. convert\_column\_to\_dict\_all function convert excel column or table field into dictionary depending of the excel page name.

def convert\_column\_to\_dict\_all(path, sheet\_name):

    dataset\_siamm\_dataframe = pd.read\_excel(path, sheet\_name=sheet\_name)

    if sheet\_name == "Colonne":

        dataset\_siamm\_dataframe.fillna('', inplace=True)

    elif sheet\_name == "Dataset":

        dataset\_siamm\_dataframe["Note"].fillna(0, inplace=True)

        dataset\_siamm\_dataframe.fillna(0, inplace=True)

    new\_dataframe = dataset\_siamm\_dataframe.to\_dict(orient='records')

    print("Columns were transformed to a dictionary...")

    return new\_dataframe

1. dataframe\_formatting function group all columns by their table names.

def dataframe\_formatting(columns\_dict\_records):

    grouped\_dictionary = {}

    for value in columns\_dict\_records:

        name = value.pop('Nome del Dataset')

        if name in grouped\_dictionary:

            grouped\_dictionary[name].append(value)

        else:

            grouped\_dictionary[name] = [value]

    return grouped\_dictionary

## table\_entities.py

Contains implementation of the next functions:

* create\_entities\_dataset(dataset,technical\_and\_operational\_metadata,business\_metadata\_dataset)

This function creates table atlas entities from dictionary by formatting the dictionary and deleting repetitive camps.

def create\_entities\_dataset(dataset, technical\_and\_operational\_metadata, business\_metadata\_dataset):

    print("Starting of table creation....")

    technical\_and\_operational\_metadata.remove("Formato")

    technical\_and\_operational\_metadata.remove('Nullability')

    technical\_and\_operational\_metadata.remove('Atlas Type')

    formated\_business\_dataset = []

    for name\_of\_attr in business\_metadata\_dataset:

        name\_of\_attr = name\_of\_attr.replace('/',' or ')

        name\_of\_attr = name\_of\_attr.replace('à','a')

        formated\_business\_dataset.append(name\_of\_attr)

    # print(formated\_business\_dataset)

    for value in dataset:

        value['Dominio or Area di riferimento'] = value['Dominio/Area di riferimento']

        del value['Dominio/Area di riferimento']

        value['Descrizione entita di business'] = value['Descrizione entità di business']

        del value['Descrizione entità di business']

        value['Nome entita di Business'] = value['Nome entità di Business']

        del value['Nome entità di Business']

        value['Profondita Storica'] = value['Profondità Storica']

        del value['Profondità Storica']

        if value["Nome del Dataset"].lower() in list\_hive\_table\_type:

            # print("it true hive\_table")

            value['Atlas Type'] = "hive\_table"

        else:

            # print("it false hive\_table")

            value['Atlas Type'] = "hive\_table"

        entity\_from\_server = client.get\_entity(

        qualifiedName= value['QualifiedName'],

        typeName=value['Atlas Type']

        )

        if entity\_from\_server.get("entities") != None:

            for entity in entity\_from\_server.get("entities"):

                jsonData =json.dumps(entity, indent=2)

                readJSON = json.loads(jsonData)

                # print(readJSON)

                print(f"Entity {readJSON['attributes']['name']} already exists in Atlas")

                # delete\_entitties(entity\_from\_server.get("entities"))

                # entity\_from\_server = client.get\_entity(

                # qualifiedName= value['QualifiedName'],

                # typeName=value['Atlas Type']

                # )

                # if entity\_from\_server.get("entities") != None:

                #     for entity in entity\_from\_server.get("entities"):

                #         jsonData =json.dumps(entity, indent=2)

                #         readJSON = json.loads(jsonData)

                #         # print(readJSON)

                #         print(f"Entity {readJSON['attributes']['name']} already exists in Atlas")

                        # delete\_entitties(entity\_from\_server.get("entities"))

                # rerìwrite this part od code

        ae = AtlasEntity(

        name = value['Nome del Dataset'],

        typeName = value['Atlas Type'],

        qualified\_name = value['QualifiedName'],

        attributes={

                        "description": value["Descrizione Dataset"],

                        "isOptional": True

                    },

        guid = -1

        )

        business\_attr = {}

        custom\_attr = {}

        for attr in value:

            business\_attr[attr]=value[attr]

        for c\_attr in technical\_and\_operational\_metadata:

            custom\_attr[c\_attr] = value[c\_attr]

        # print(custom\_attr)

        # for i in custom\_attr:

        #     print(i)

        ae.addBusinessAttribute(SIAMM\_business\_metadata\_hive\_table\_main1=business\_attr)

        ae.addCustomAttribute(sistemaSorgente = custom\_attr["Sistema Sorgente"],dataStore = custom\_attr["Data Store"],schemaTarget = custom\_attr["Schema Target"],primaryAndForeignKey = custom\_attr["Primary Key and Foreign Key"],layerApplicativo = custom\_attr["Layer applicativo"],chiaveDiPartizionamento = custom\_attr["Chiave di partizionamento"],frequenzaAggiornamento = custom\_attr["Frequenza d’aggiornamento "],sizing = custom\_attr["Sizing"],dataCreazione = custom\_attr["Data Creazione"],updateDate = custom\_attr["Update date"])

        resp = client.upload\_entities([ae])

        guid = resp["guidAssignments"]["-1"]

        jsonData2 = json.dumps(resp,indent=2)

        print(jsonData2)

        print(json.dumps(client.get\_single\_entity(guid),indent=2))

        readJSON2 = json.loads(jsonData2)

Sample result:

A screenshot of a computer

Description automatically generated

## main.py

Executes each created function in its order.

Head of the script contains path with different excel files for testing:

- adding columns to the table

- delete columns

- edit attributes

The PATH constant is required for easy alterations of the path in functions.

The execution steps:

1. Reads excel “Istruzioni” sheet and save the fields in dictionary named **list\_of\_metadata**.
2. Divides **list\_of\_metadata** dictionary on technical and business metadata lists. Save lists in the tuple named **tuple\_of\_metadatatype.**
3. Devides tuple **tuple\_of\_metadatatype** on two separated lists **business\_metadata\_dataset** list and **technical\_and\_operational\_metadata** list.
4. Take tables with attributes from “Dataset” page in excel and convert it into the dictionary named **dataset\_dict\_records**
5. Create qualified names to each table from **dataset\_dict\_records and save them into dataset\_with\_q\_name.**

from preproccessing import read\_exel\_tech\_business\_metadata, divide\_columns\_by\_type, convert\_column\_to\_dict\_all, create\_qualifiedname, dataframe\_formatting

from business\_metadata import create\_bisness\_metadata\_table, create\_bisness\_metadata\_for\_column

from table\_entities import create\_entities\_dataset

from column\_entities import create\_entities\_column, delete\_column\_from\_atlas

from lineage import create\_lineage\_for\_two\_tables, create\_lineage\_for\_ingestion

path\_main\_exel = 'develop\MdG-MetadataManagementSIAMM.xlsx'

path\_example= 'develop\MdG-MetadataManagementSIAMM\_example.xlsx'

path\_test\_add\_column = "develop\col\_add\_test.xlsx"

path\_delete\_column = "develop\col\_delete\_test.xlsx"

path\_add\_del\_col = "develop\col\_del\_add\_test.xlsx"

path\_edit\_attr = 'develop\diff\_tech\_attr\_test.xlsx'

path\_full\_dataset\_modify\_table\_attr = "develop\modify\_full\_test\_dataset.xlsx"

PATH = path\_main\_exel

list\_of\_metadata = read\_exel\_tech\_business\_metadata(PATH)

tuple\_of\_metadatatype = divide\_columns\_by\_type(list\_of\_metadata)

business\_metadata\_dataset = tuple\_of\_metadatatype[0]

technical\_and\_operational\_metadata =tuple\_of\_metadatatype[1]

dataset\_dict\_records = convert\_column\_to\_dict\_all(PATH, sheet\_name="Dataset" )

dataset\_with\_q\_name = create\_qualifiedname(dataset\_dict\_records)

columns\_dict\_records = convert\_column\_to\_dict\_all(PATH, sheet\_name="Colonne" )

formated\_column\_dataset = dataframe\_formatting(columns\_dict\_records)

for table in formated\_column\_dataset:

    for attr in formated\_column\_dataset[table]:

        column\_keys = list(attr.keys())

# print(column\_keys)

# delete\_column\_from\_atlas(formated\_column\_dataset, dataset\_with\_q\_name)

create\_bisness\_metadata\_table(business\_metadata\_dataset)

create\_bisness\_metadata\_for\_column(column\_keys, business\_metadata\_dataset)

create\_entities\_dataset(dataset\_with\_q\_name, technical\_and\_operational\_metadata, business\_metadata\_dataset)

create\_entities\_column(formated\_column\_dataset, dataset\_with\_q\_name)

# create\_lineage\_for\_two\_tables(client)

# create\_lineage\_for\_ingestion(client)