s#Save this as Preprocess Validation File which we can import later

def preprocess\_validation(version):

v=version

blob\_files = dbutils.fs.ls("/mnt/dmro/autoloader")

# read in existing delta tables

dt = spark.read.table("dmro\_raw.detections")

tg = spark.read.table("dmro\_raw.tags").toPandas()

df = spark.read.table("dmro\_raw.defect")

# initialize dictionaries to keep track of package counts and files to exclude

package\_count\_dictionary = {}

excluded\_files = []

#creating Dictionary for storing the data as table

exclude\_files\_dict={'filename':[],

'reason':[]}

for filename in blob\_files:

f\_split = filename.path.split('/')[-1]

print(f\_split)

if f\_split != "":

f\_no\_csv = f\_split[:-4]

f\_stream = '\_'.join(f\_no\_csv.split('\_')[0:2])

f\_table\_name = f\_no\_csv.split('\_')[-1]

# count number of packages

if f\_stream in package\_count\_dictionary:

package\_count\_dictionary[f\_stream] += 1

else:

package\_count\_dictionary[f\_stream] = 1

# check for 3 matching files

matching\_files = [f for f in blob\_files if f\_stream in f.path]

if len(matching\_files) != 3:

excluded\_files.append(f\_split + " does not have 3 matching files")

exclude\_files\_dict['filename'].append(f\_split)

exclude\_files\_dict['reason'].append('does not have 3 matching files')

continue

# check defect file name

if "tags" not in f\_table\_name and "Detections" not in f\_table\_name:

if len(f\_table\_name.split('-')) < 4:

excluded\_files.append(f\_split + " defect file name is not valid")

exclude\_files\_dict['filename'].append(f\_split)

exclude\_files\_dict['reason'].append('defect file name is not valid')

continue

# check for duplicate stream id

if tg['stream\_id'].str.contains(f\_stream).any() or \

dt.filter(F.col("stream\_id") == f\_stream).count() > 0 or \

df.filter(F.col("stream\_id") == f\_stream).count() > 0:

excluded\_files.append(f\_split + " duplicate stream ID and table name")

exclude\_files\_dict['filename'].append(f\_split)

exclude\_files\_dict['reason'].append('duplicate stream ID and table name')

continue

# check for duplicate combination of 4 attributes

if "tags" not in f\_split and "Detections" not in f\_split:

sn, sv, cs, ii = f\_table\_name.split('-')[0:4]

if tg[(tg['serial\_number'] == sn) &

(tg['shop\_visit'] == sv) &

(tg['create\_station'] == cs) &

(tg['inspection\_index'] == ii)].shape[0] > 0:

excluded\_files.append(f\_split + " duplicate combination of 4 attributes")

exclude\_files\_dict['filename'].append(f\_split)

exclude\_files\_dict['reason'].append('duplicate combination of 4 attributes')

continue

# Move the excluded files to the exception folder

for file in excluded\_files:

file\_name = file.split()[0]

dbutils.fs.mv("/mnt/autoloader/" + file\_name, "/mnt/exception/" + file\_name)

#Saving the Exlcuded files info in tabular format

df\_exclude=pd.DataFrame(exclude\_files\_dict)

df\_exclude\_final=df\_exclude.style

display(df\_exclude\_final)

#return true if everything gets completed successfully

return True

##Save this file as autoloader\_readwritestream

def autoloader\_read\_writestream(version):

###for testing delete this line later

v = version

#source and exception file paths

sourcepath='/mnt/dmro/autoloader/'

exceptions\_file\_path\_tags=f"dmro.tag\_exception{v}"

exceptions\_file\_path\_defects=f"dmro.defect\_exception{v}"

exceptions\_file\_path\_detections=f"dmro.detection\_exception{v}"

# Delta table paths for the tags and defects tables

tags\_table\_path = f"dmro\_raw.tag{v}"

defects\_table\_path = f"dmro\_raw.defect{v}"

detections\_table\_path = f"dmro\_raw.detection{v}"

###tag readstream & writestream

tags\_options = {

"cloudFiles.format": "csv",

"mergeSchema": "true",

"cloudFiles.schemaLocation":f"/mnt/dmro/autoloader\_mvp/schemalocation\_mvp/tag{v}",

"inferSchema":"true",

"pathGlobFilter": "\*tags\*.csv",

"path": sourcepath

}

tags\_df = spark \

.readStream \

.format("cloudFiles") \

.options(\*\*tags\_options) \

.load()

tags\_stream = tags\_df.writeStream \

.trigger(once=True) \

.option("checkpointLocation", f"/mnt/dmro/autoloader\_mvp/checkpointlocation\_mvp/tag{v}") \

.option("delta.columnMapping.mode", "name")\

.foreachBatch(tag\_blobtoraw) \

.start()

###defect readstream & writestream

defects\_options = {

"cloudFiles.format": "csv",

"mergeSchema": "true",

"cloudFiles.schemaLocation":f"/mnt/dmro/autoloader\_mvp/schemalocation\_mvp/defect{v}",

"inferSchema":"true",

"pathGlobFilter": "\*-[0-9]-\*-[0-9].csv",

"path": sourcepath

}

defects\_df = spark \

.readStream \

.format("cloudFiles") \

.options(\*\*defects\_options) \

.load()

defects\_stream = defects\_df.writeStream \

.trigger(once=True) \

.option("checkpointLocation", f"/mnt/dmro/autoloader\_mvp/checkpointlocation\_mvp/defect{v}") \

.option("delta.columnMapping.mode", "name")\

.foreachBatch(defect\_blobtoraw) \

.start()

###detection readstream & writestream

detections\_options = {

"cloudFiles.format": "csv",

"mergeSchema": "true",

"cloudFiles.schemaLocation":f"/mnt/dmro/autoloader\_mvp/schemalocation\_mvp/detection{v}",

"inferSchema":"true",

"pathGlobFilter": "\*Detections\*.csv",

"path": sourcepath

}

detections\_df = spark \

.readStream \

.format("cloudFiles") \

.options(\*\*detections\_options) \

.load()

detections\_stream = detections\_df.writeStream \

.trigger(once=True) \

.option("checkpointLocation", f"/mnt/dmro/autoloader\_mvp/checkpointlocation\_mvp/detection{v}") \

.option("delta.columnMapping.mode", "name")\

.foreachBatch(detection\_blobtoraw) \

.start()

return True

##Save this file as blob\_to\_raw

def tag\_blobtoraw(df: DataFrame, batchId: int):

df\_timestamp = df.withColumn("load\_timestamp", F.current\_timestamp()).withColumn("processed",lit(False))

#Condition here to check if there if inspection\_index and shop\_visit is equal to 0 then save it to exception\_file\_path\_tags from where we can send email

exception\_df=df\_timestamp.filter((col('inspection\_index')==0)|(col('shop\_visit')==0))

if exception\_df.count()>0:

reason\_df = exception\_df.withColumn("reason", lit("inspection index or shop visit is 0"))

reason\_df.write.format('delta').mode('append').option('header','true').option("delta.columnMapping.mode", "name").saveAsTable(exceptions\_file\_path\_tags)

df\_timestamp.write.format("delta").mode("append").option("mergeSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable(tags\_table\_path)

return True

def detection\_blobtoraw(df: DataFrame, batchId: int):

drop\_unnamed\_col = [c for c in df.columns if "Unnamed" in c]

df = df.drop(\*drop\_unnamed\_col)

df\_timestamp = df.withColumn("load\_timestamp", F.current\_timestamp()).withColumn("processed",lit(False))

df\_stream = df\_timestamp.withColumn("stream\_id", substring(input\_file\_name(), 22, 52))

#df\_stream = df\_file\_name.withColumn("stream\_id", substring(col("input\_file"), 21, 13)).drop("input\_file")

null\_uuid\_df=df\_stream.filter(col('UUID').isNull())

#Condition here to check if there is any rows having null then save it to exception\_file\_path\_defects from where we can send email

if null\_uuid\_df.count()>0:

null\_reason\_def\_df = null\_uuid\_df.withColumn("reason", lit("UUID is NULL"))

null\_reason\_def\_df.write.format('delta').mode('append').option('header','true').option("delta.columnMapping.mode", "name").saveAsTable(exceptions\_file\_path\_detections)

df\_stream.filter(col('UUID').isNotNull()).write.format("delta").mode("append").option("mergeSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable(detections\_table\_path)

return True

def defect\_blobtoraw(df: DataFrame, batchId: int):

drop\_unnamed\_col = [c for c in df.columns if "Unnamed" in c]

df = df.drop(\*drop\_unnamed\_col)

df\_timestamp = df.withColumn("load\_timestamp", F.current\_timestamp())\

.withColumn("processed",lit(False))

df\_stream = df\_timestamp.withColumn("stream\_id", substring(input\_file\_name(), 22, 52))

null\_uuid\_df = df\_stream.filter(col('UUID').isNull())

#Condition here to check if there is any rows having null then save it to exception\_file\_path\_defects from where we can send email

if null\_uuid\_df.count()>0:

null\_reason\_det\_df = null\_uuid\_df.withColumn("reason", lit("UUID is NULL"))

null\_reason\_det\_df.write.format('delta').mode('append').option('header','true').option("delta.columnMapping.mode", "name").saveAsTable(exceptions\_file\_path\_defects)

df\_stream.filter(col('UUID').isNotNull()).write.format("delta").mode("append").option("mergeSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable(defects\_table\_path)

return True

#Save this file as raw\_to\_curated

Raw to curated

#version

v = ""

###check if UUID match in defect & detection

from pyspark.sql.functions import lit

def uuid\_defect\_detection\_match(version):

v=version

dt = spark.read.table(f"dmro\_raw.detection{v}")

dt.createOrReplaceTempView("dt")

df = spark.read.table(f"dmro\_raw.defect{v}")

df.createOrReplaceTempView("df")

##to find all non matching 1:1 uuid in detections and defects file

df\_not\_match = spark.sql("""select dt.uuid, dt.stream\_id, 'detection' as from\_table

from dt

left outer join df

on dt.uuid = df.uuid

where df.stream\_id is null and dt.processed = False

union all

select dt.uuid, dt.stream\_id, 'defect' as from\_table

from df

left outer join dt

on dt.uuid = df.uuid

where dt.stream\_id is null and df.processed = False

""")

df\_not\_match.display()

if df\_not\_match.count() != 0:

df\_not\_match = df\_not\_match.withColumn("reason", lit("the following 1-1 UUID does not match in defect & detections"))\

.withColumn("load\_timeStamp", F.current\_timestamp())

df\_not\_match.write.format("delta").mode("append").option("mergeSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable("dmro.defect\_detection\_uuid\_match")

# print("the following 1-1 UUID does not match in defect & detections: ")

# df\_not\_match.display()

return True

###inspection

def inspection\_rawtocurated(version):

v=version

#read tag table and get latest timestamp data

# df\_tag = spark.sql("select \* from dmro\_raw.tag where load\_timestamp = (select max(load\_timestamp) from dmro\_raw.tag)")

df\_tag = spark.sql(f"select \* from dmro\_raw.tag{v} where processed = False")

if df\_tag.count != 0:

exception\_df=df\_tag.filter((df\_tag['operation\_code'].isNull() == True) | (df\_tag['create\_time'].isNull() == True))

if exception\_df.count()>0:

reason\_df = exception\_df.withColumn("reason", lit("operation code or create time is null"))

reason\_df.write.format('delta').mode('append').option('header','true').option("delta.columnMapping.mode", "name").saveAsTable("dmro.tag\_exception")

#create new columns to split up time

df\_tag = df\_tag.withColumn('inspection\_date', F.date\_format(F.col('create\_time'), 'M/d/yyyy'))\

.withColumn('inspection\_time', F.date\_format(F.col('create\_time'), 'H:m:s'))

#select the necessary fields

df\_tag = df\_tag.select('part\_number', 'serial\_number', 'automated\_disposition', 'create\_station', 'inspection\_index', 'inspector\_disposition', 'inspection\_facility', 'operation\_code', 'ppi\_type', 'shop\_visit', 'stream\_id', 'inspection\_date', 'inspection\_time')

#input new current timestamp

df\_tag = df\_tag.withColumn("load\_timestamp", F.current\_timestamp())\

.withColumn("processed", lit(False))

df\_tag.schema

df\_tag.display()

#write table to curated layer

df\_tag.write.format("delta").mode("append").option("mergeSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable(f"dmro\_curated.inspection{v}")

else:

print("No New Records")

return True

###defect

def defect\_rawtocurated(version):

v= version

#read tables from raw

df\_tag = spark.read.table(f"dmro\_raw.tag{v}")

df\_tag.createOrReplaceTempView("tag")

df\_def = spark.read.table(f"dmro\_raw.defect{v}")

df\_ppi = spark.read.table("dmro\_curated.ppi\_airfoil\_region")

# df\_def.createOrReplaceTempView("defect")

df\_ppi.createOrReplaceTempView("ppi")

exception\_df=df\_def.filter((df\_def['disposition'].isNull() == True) | (df\_def['`indication type`'].isNull() == True))

if exception\_df.count()>0:

reason\_df = exception\_df.withColumn("reason", lit("disposition or indication type is null"))

reason\_df.write.format('delta').mode('append').option('header','true').option("delta.columnMapping.mode", "name").saveAsTable("dmro.defect\_exception")

df\_def = df\_def.filter((col('disposition').isNotNull()) & (col('`indication type`').isNotNull()))

df\_def.createOrReplaceTempView("defect")

# df\_def.display()

# sql statement with joins from tag to defect and ppi based on max timestamp

df = spark.sql("""select tg.part\_number, tg.serial\_number, df.`Defect Type` as defect\_type, df.size, df.disposition, df.`Indication Type` as ppi\_region, df.`Short text` as basic\_description, df.x, df.y, df.z, df.`Analytics Confidence` as analytics\_confidence, df.angle, df.elongation, df.uuid, df.stream\_id, pp.airfoil\_region

from tag as tg

join defect as df

on df.stream\_id = tg.stream\_id

left outer join ppi as pp

on pp.PPI\_region = rtrim(df.`Indication Type`) and pp.part\_number = tg.part\_number

where tg.processed = False

""")

# df = spark.sql("select tg.part\_number, df.`Indication Type` as ppi\_region, pp.airfoil\_region from tag as tg join defect as df on df.stream\_id = tg.stream\_id left outer join ppi as pp on pp.PPI\_region = df.`Indication Type`")

df.schema

df.display()

#input new current timestamp

if df.count != 0:

df = df.withColumn("load\_timestamp", F.current\_timestamp())\

.withColumn("processed", lit(False))

# df.display()

#write table to curated layer

df.write.format("delta").mode("append").option("mergeSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable(f"dmro\_curated.defect{v}")

else:

print("No New Records")

###processed true for raw layer tables

df\_tag\_raw = spark.sql(f"select \* from dmro\_raw.tag{v}")

df\_defect\_raw = spark.sql(f"select \* from dmro\_raw.defect{v}")

df\_updated\_tag = df\_tag\_raw.withColumn("processed", F.when(F.col("processed") == False, True).otherwise(F.col("processed")))

df\_updated\_defect = df\_defect\_raw.withColumn("processed", F.when(F.col("processed") == False, True).otherwise(F.col("processed")))

# df\_updated\_tag.display()

# df\_updated\_defect.display()

df\_updated\_tag.write.format("delta").mode("overwrite").option("overwriteSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable(f"dmro\_raw.tag{v}")

df\_updated\_defect.write.format("delta").mode("overwrite").option("overwriteSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable(f"dmro\_raw.defect{v}")

return True

#Function for changing the string to time

def parse\_time(date):

return datetime.strptime(date,'%H:%M:%S').time()

#registering the udf

parse\_time\_udf=udf(parse\_time)

#for changing the data type of inspection table of selected columns in curated layer

def datatype\_change\_inspection():

df\_ins = spark.read.table(f"dmro\_curated.inspection{v}")

df\_ins.withColumn("inspection\_index",df.inspection\_index.cast('int'))

df\_ins.withColumn("operation\_code",df.operation\_code.cast('int'))

df\_ins.withColumn("inspection\_date",parse\_time\_udf(df\_ins["inspection\_date"]))

df\_ins.withColumn("inspection\_time",df.inspection\_time.cast('time'))

#for changing the data type of defect table of selected columns in curated layer

def datatype\_change\_defect():

df\_ins = spark.read.table(f"dmro\_curated.inspection{v}")

df\_ins.withColumn("x",df.x.cast('double'))

df\_ins.withColumn("y",df.y.cast('double'))

df\_ins.withColumn("z",df.z.cast('double'))

df\_ins.withColumn("analytics\_confidence",df.analytics\_confidence.cast('double'))

df\_ins.withColumn("angle",df.angle.cast('time'))

df\_ins.withColumn("elongation",df.elongation.cast('double'))

#Save this notebook as curated\_to\_publish

#this function will read all the tables which is needed for curated to publish layer

def read\_curatedtables():

#version

v = ""

#read in curated tables

df\_ins = spark.read.table(f"dmro\_curated.inspection{v}")

df\_ins.createOrReplaceTempView("inspection")

df\_cast = spark.read.table("dmro\_curated.casting")

df\_cast.createOrReplaceTempView("casting")

df\_eu = spark.read.table("dmro\_curated.engine\_usage")

df\_eu.createOrReplaceTempView("engine")

df\_coat = spark.read.table("dmro\_curated.coating")

df\_coat.createOrReplaceTempView("coat")

df\_dh = spark.read.table("dmro\_curated.hole\_drill")

df\_dh.createOrReplaceTempView("hole")

df\_af = spark.read.table("dmro\_curated.airflow")

df\_af.createOrReplaceTempView("airflow")

df\_df = spark.read.table(f"dmro\_curated.defect{v}")

df\_df.createOrReplaceTempView("defect")

return True

#pivot airfoil table

def pivot\_curatedtopublish():

airflow = spark.read.table("dmro\_curated.airflow")

airflow\_group = airflow.groupby("serial\_number")\

.pivot("sequence\_number")\

.sum("flow\_deviation")

airflow\_group.createOrReplaceTempView("airflow\_group")

airflow\_group.display()

return True

#create inspection publish table

def inspection\_curatedtopublish(version):

v=version

df\_inspection = spark.sql("""select

ins.part\_number,

ins.serial\_number,

ins.automated\_disposition,

ins.inspection\_date,

ins.create\_station,

ins.shop\_visit,

ins.stream\_id,

current\_timestamp() as load\_timestamp,

cast.casting\_vendor,

eng.ESN,

eng.engine\_flight\_hours,

eng.engine\_cycles,

eng.engine\_operator,

ct.coating\_vendor,

ct.coating\_gain,

hol.hole\_drill\_vendor,

ar.`4800`,

ar.`4802`

from inspection as ins

left join

casting as cast

on ins.serial\_number=cast.serial\_number

left join

engine as eng

on eng.serial\_number=cast.serial\_number

left join

coat as ct

on ct.serial\_number=eng.serial\_number

left join

hole as hol

on hol.serial\_number =ct.serial\_number

left join

airflow\_group as ar

on ct.serial\_number=ar.serial\_number

where ins.processed = False

""")

# df = spark.sql("""SELECT \* FROM inspection""")

if df\_inspection.count() != 0:

df\_insp = df\_inspection.withColumn("system\_insp\_id", monotonically\_increasing\_id())

df\_insp.schema

df\_insp.display()

if spark.\_jsparkSession.catalog().tableExists(f"dmro\_published.inspection\_publish{v}") == False:

df\_insp.write.format("delta").mode("overwrite").option("overwriteSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable(f"dmro\_published.inspection\_publish{v}")

else:

df\_insp.write.format("delta").mode("append").option("mergeSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable(f"dmro\_published.inspection\_publish{v}")

return True

###create Defect publish table

def defect\_curatedtopublish(version):

v=version

df\_pub = spark.sql("""select

df.part\_number,

df.serial\_number,

df.defect\_type,

df.size,

df.disposition,

df.ppi\_region,

df.airfoil\_region,

df.uuid,

df.stream\_id,

current\_timestamp() as load\_timestamp,

df.x,

df.y,

df.z,

cast.casting\_vendor,

eng.ESN,

eng.engine\_flight\_hours,

eng.engine\_cycles,

eng.engine\_operator,

ct.coating\_vendor,

ct.coating\_gain,

ct.coating\_date,

hol.hole\_drill\_vendor,

ar.`4800`,

ar.`4802`,

ins.inspection\_date,

ins.automated\_disposition,

ins.create\_station,

ins.inspection\_index,

ins.shop\_visit

from defect as df

left join

casting as cast

on df.serial\_number=cast.serial\_number

left join

engine as eng

on eng.serial\_number=cast.serial\_number

left join

coat as ct

on ct.serial\_number=eng.serial\_number

left join

hole as hol

on hol.serial\_number =ct.serial\_number

left join

airflow\_group as ar

on ct.serial\_number=ar.serial\_number

left join

inspection as ins

on df.serial\_number=ins.serial\_number

where ins.processed = False and df.processed = False

""")

if df\_pub.count() != 0:

df\_pub = df\_pub.withColumn("system\_def\_id", monotonically\_increasing\_id())

df\_pub.schema

df\_pub.display()

df\_pub.write.format("delta").mode("append").option("mergeSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable(f"dmro\_published.defect\_publish{v}")

###processed true for curated layer tables

df\_tag\_curated = spark.sql(f"select \* from dmro\_curated.inspection{v}")

df\_defect\_curated = spark.sql(f"select \* from dmro\_curated.defect{v}")

df\_updated\_tag = df\_tag\_curated.withColumn("processed", F.when(F.col("processed") == False, True).otherwise(F.col("processed")))

df\_updated\_defect = df\_defect\_curated.withColumn("processed", F.when(F.col("processed") == False, True).otherwise(F.col("processed")))

# df\_updated\_tag.display()

# df\_updated\_defect.display()

df\_updated\_tag.write.format("delta").mode("overwrite").option("overwriteSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable(f"dmro\_curated.inspection{v}")

df\_updated\_defect.write.format("delta").mode("overwrite").option("overwriteSchema", "true").option("delta.columnMapping.mode", "name").saveAsTable(f"dmro\_curated.defect{v}")

return True

#Notebook which will be scheduled

#Notebook which will be scheduled

#importing every library and fucnction which is needed

from pyspark.sql.types import \*

from pyspark.sql.functions import lit, col, concat\_ws, input\_file\_name, substring, monotonically\_increasing\_id, udf

from pyspark.sql import functions as F

from datetime import datetime

#from pyspark.sql.types import StructType, StructField, StringType, IntegerType

from pyspark.sql import DataFrame

#importing autoloader\_readwritestream file which has tag,defect,detection code

import autoloader\_readwritestream,preprocess\_validation,blob\_to\_raw,raw\_to\_curated,curated\_to\_publish

#version control

version='version1'

#for preprocess validation

if preprocess\_validation(version):

#for blob to raw data transfer calling autoloader function where it will call seperate functions for tag ,defect and detection files

if autoloader\_read\_writestream(version):

#for raw to curated data transfer

if uuid\_defect\_detection\_match() && inspection\_rawtocurated(version) && defect\_rawtocurated(version)&&datatype\_change\_defect()&&datatype\_change\_inspection():

#for curated to publish data transfer

if read\_curatedtables():

pivotairfoil\_curatedtopublish()

inspection\_curatedtopublish()

defect\_curatedtopublish()

print("Blob to raw Layer Data Transfer was successful")