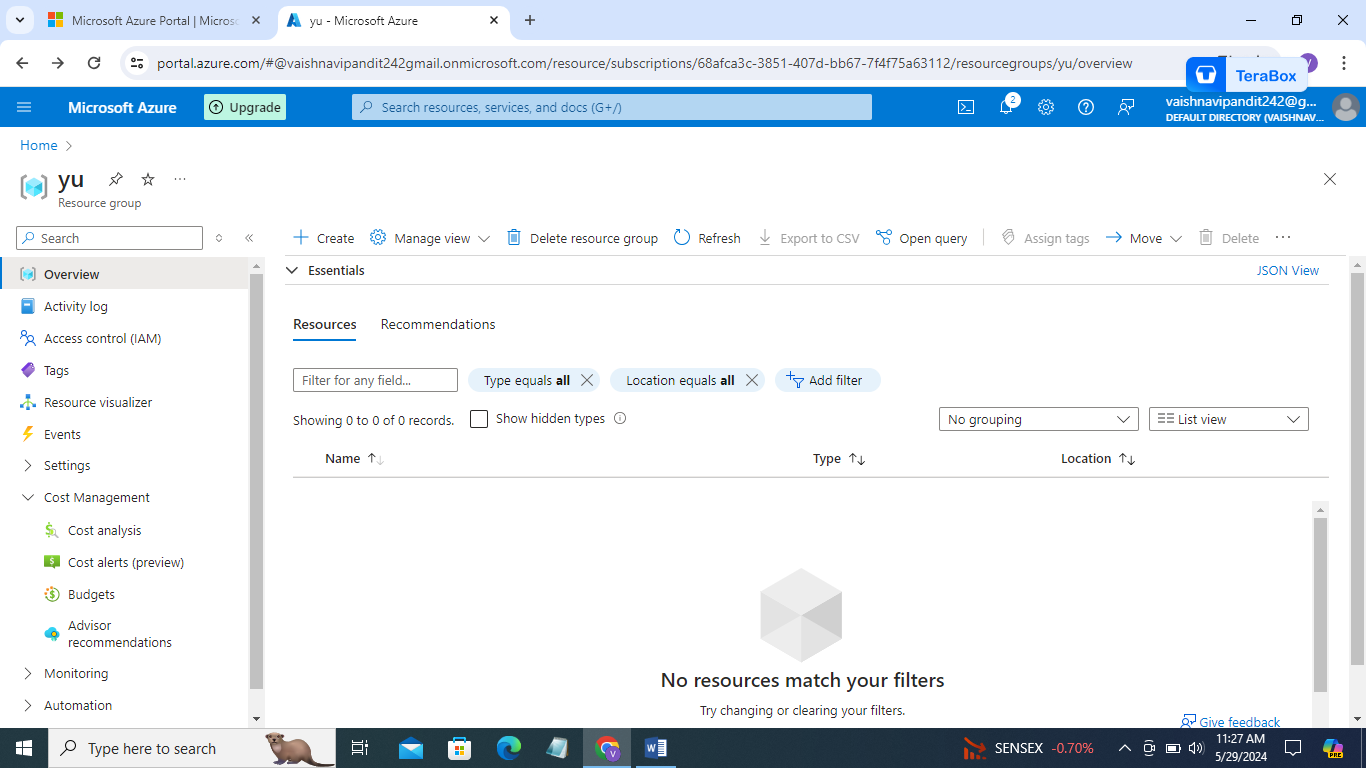
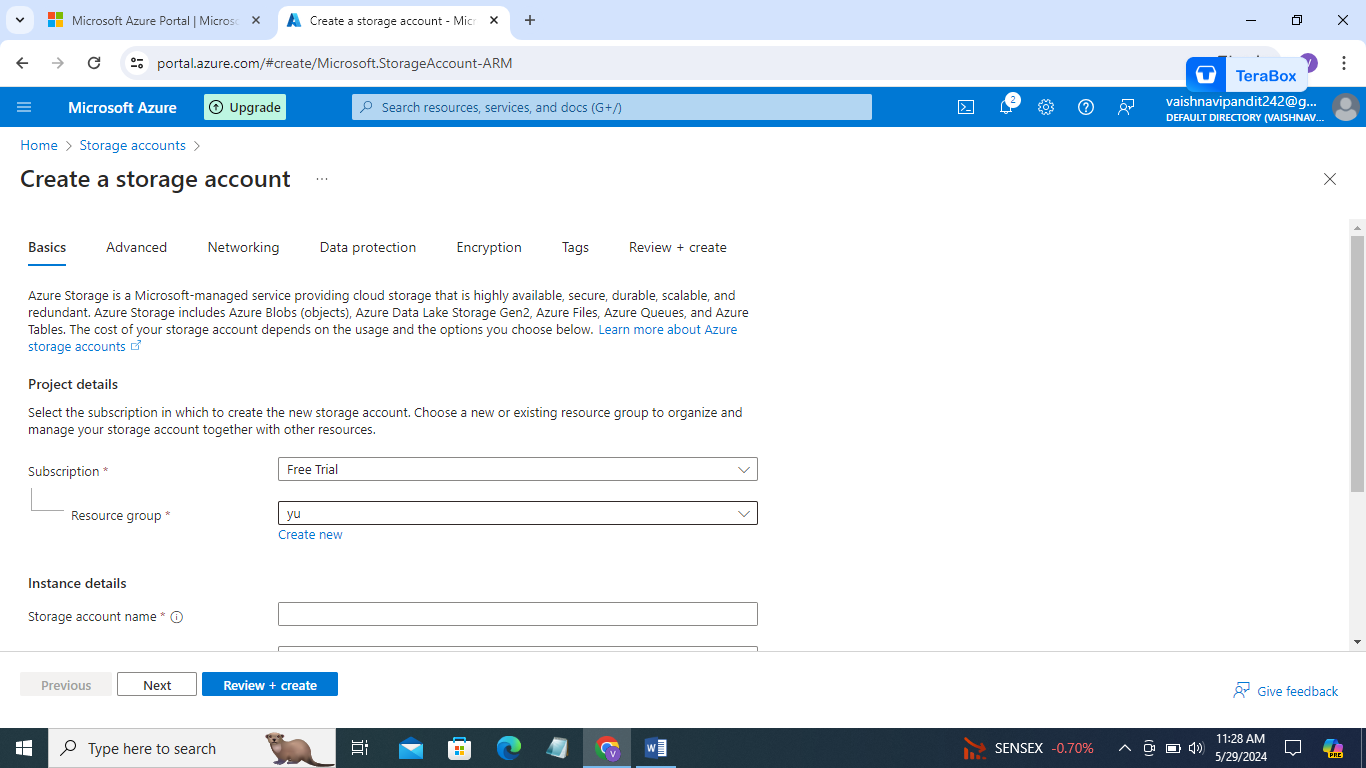
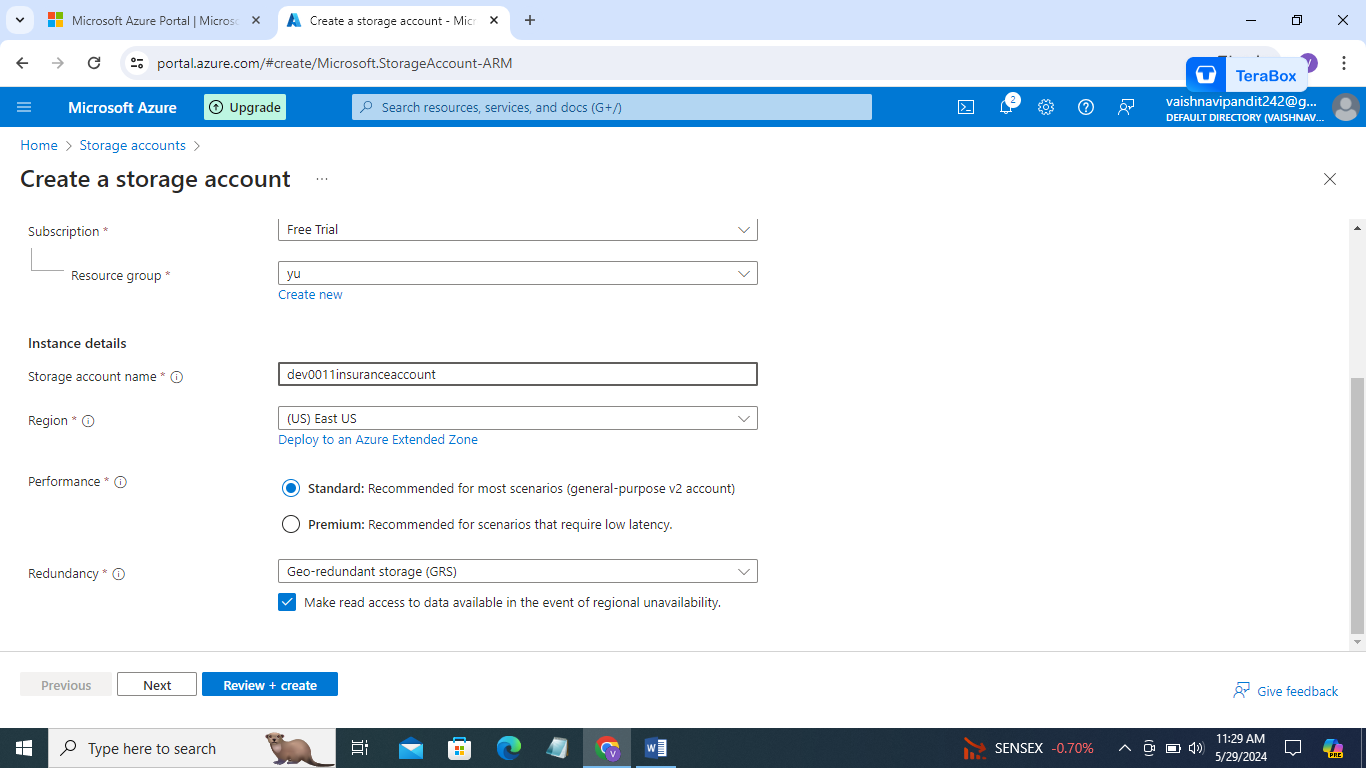
Project: Insurance data analysis

1. Create resource group

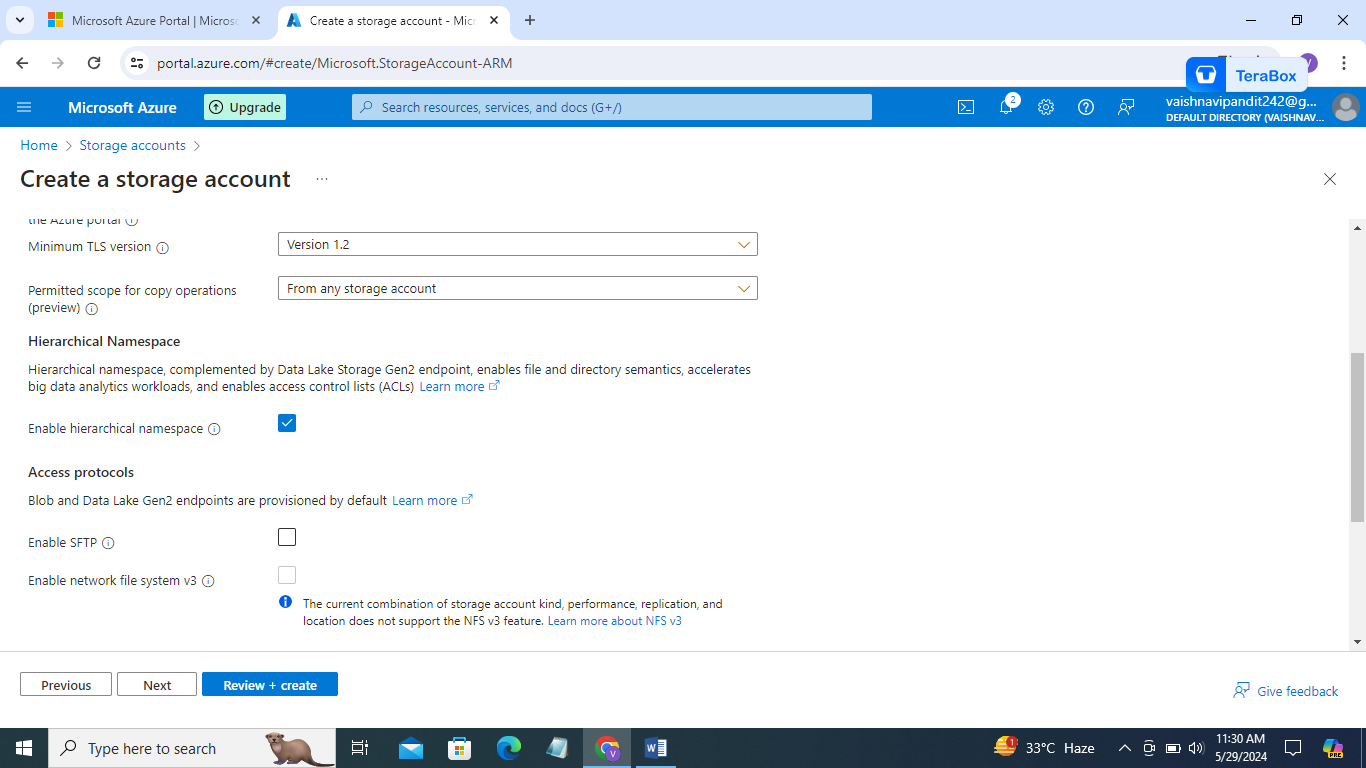


1. Now create storage account name: dev0011insuranceaccount

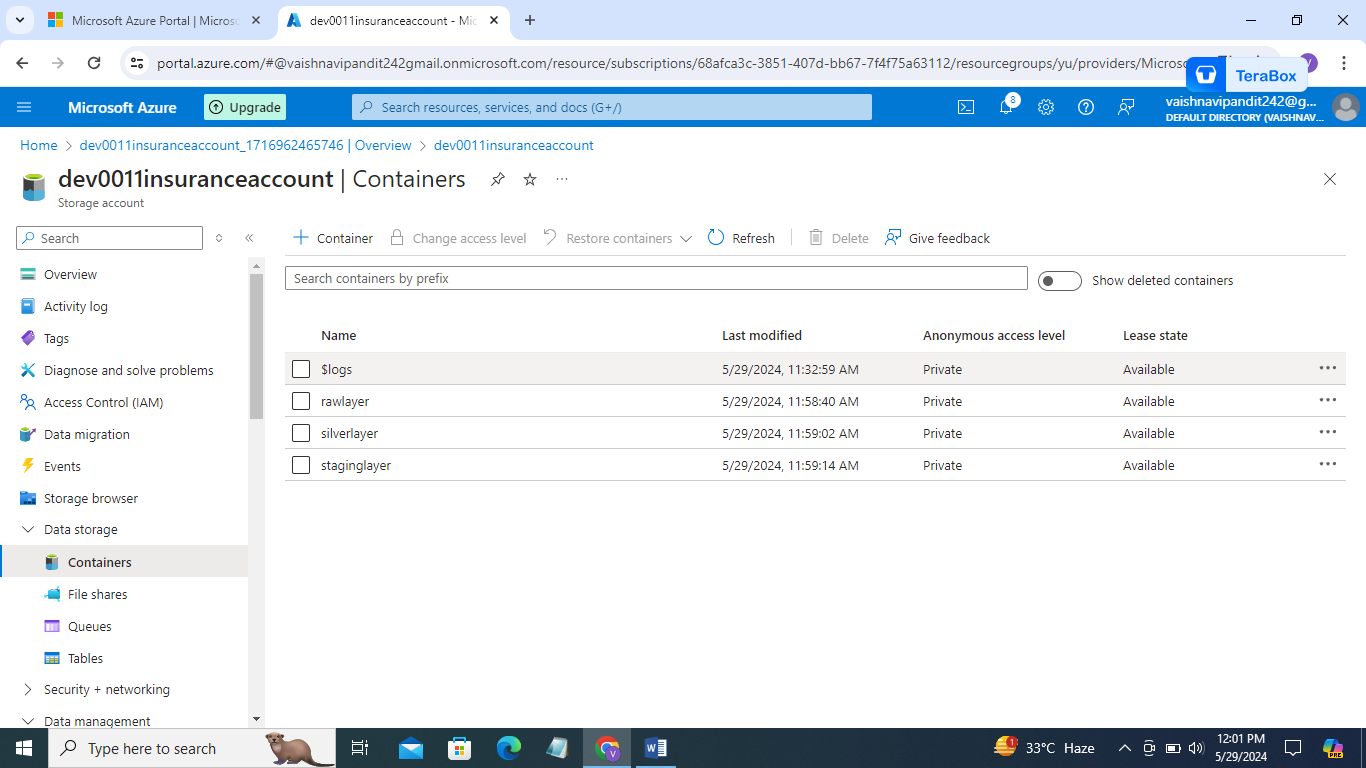




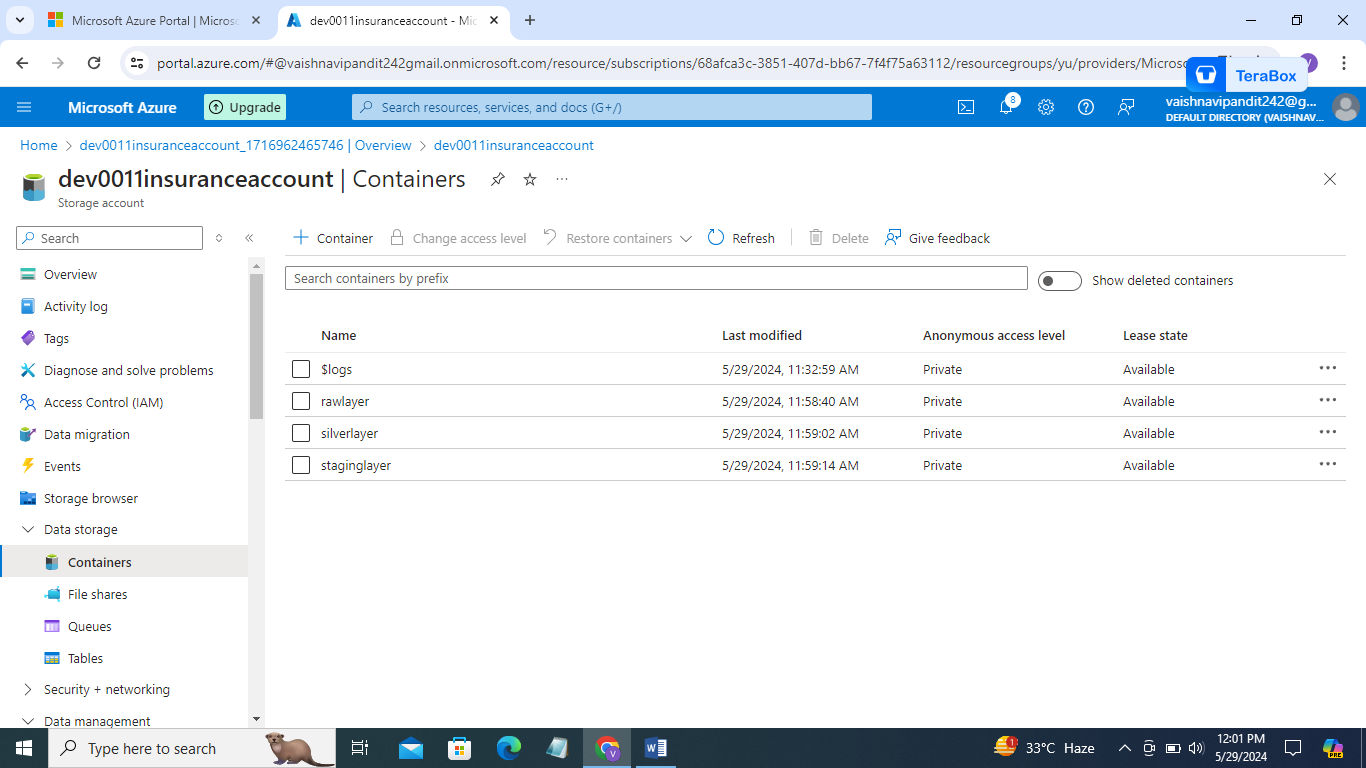
1. Enable heriarchical storage in advance and create



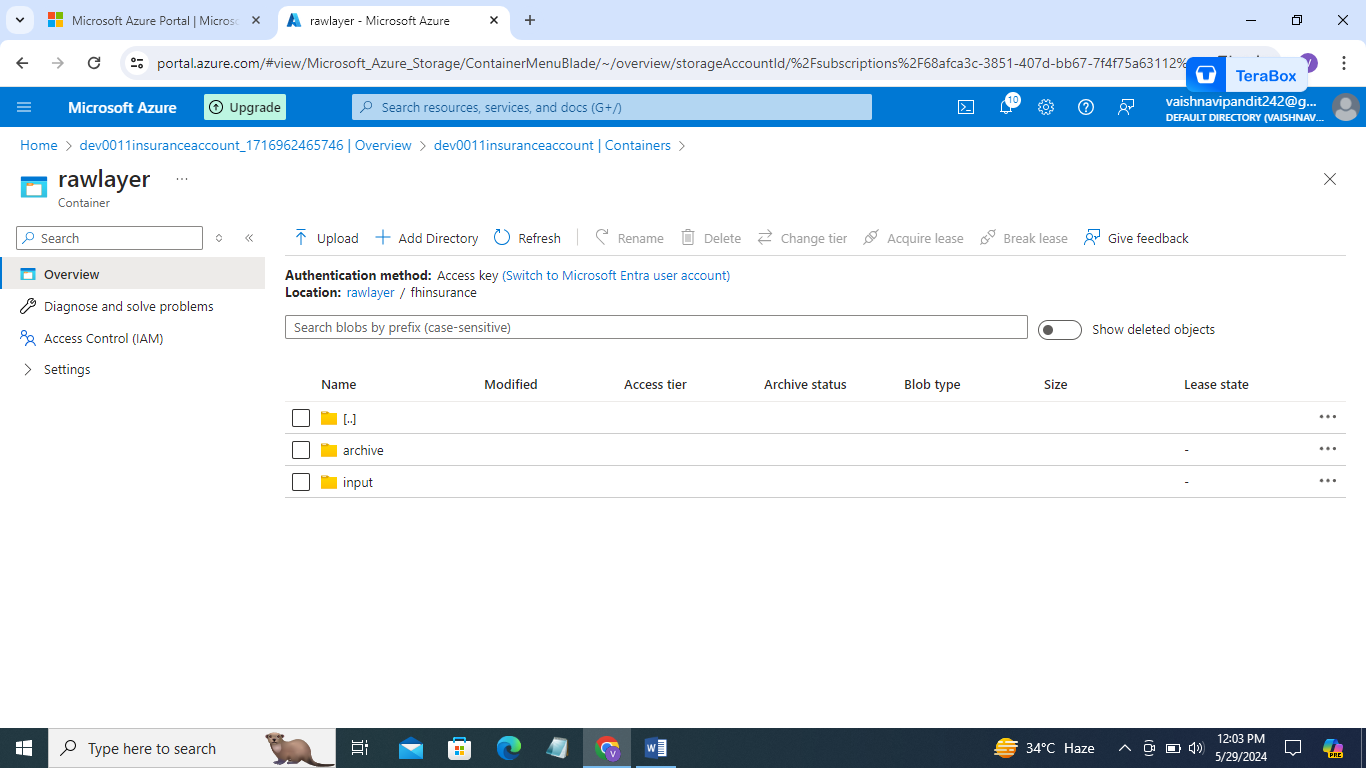
1. Go to storage account create containers rawlayer,silverlayer,staginglayer



1. In conatainer create directories fhinsurance,mpinsurance for differnet types of insurance

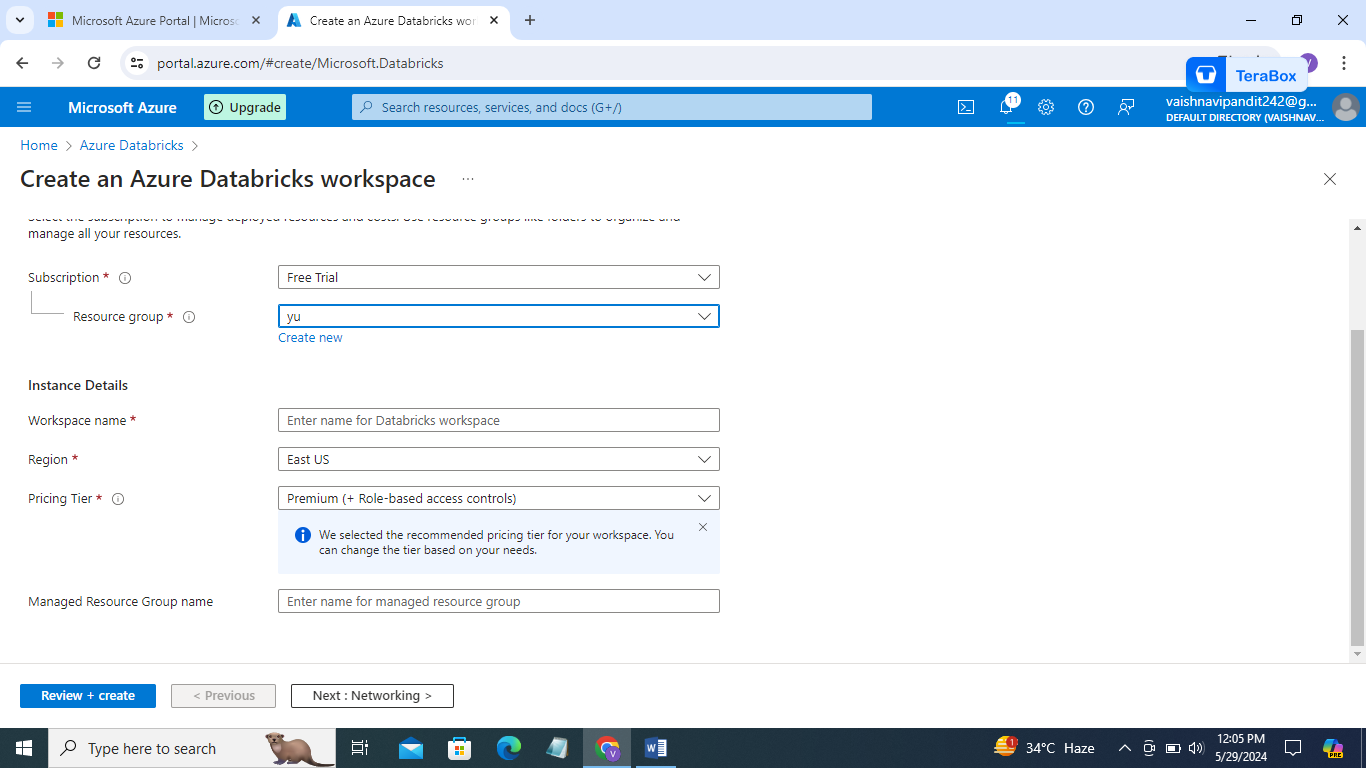


1. Now go to the rawlayer create directories input, archive

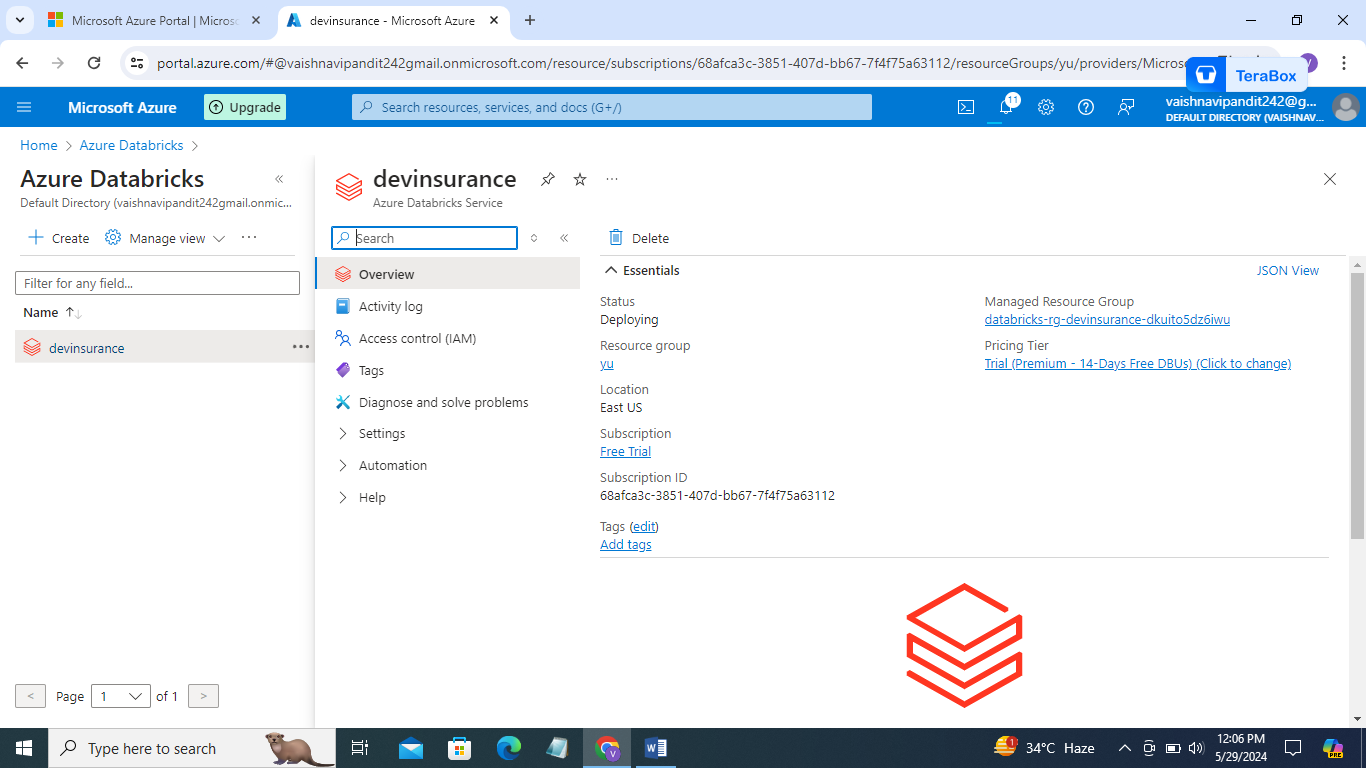


1. Now go to azure databricks

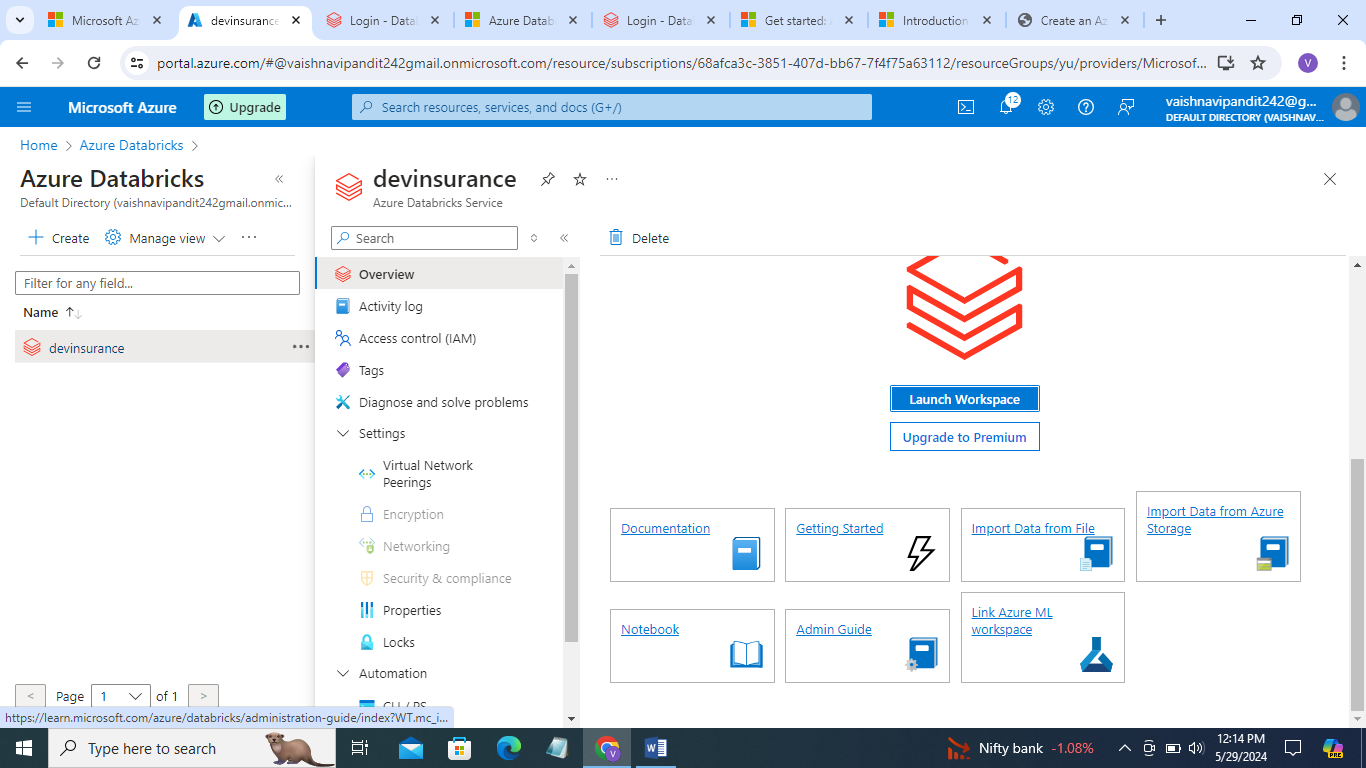
Create service



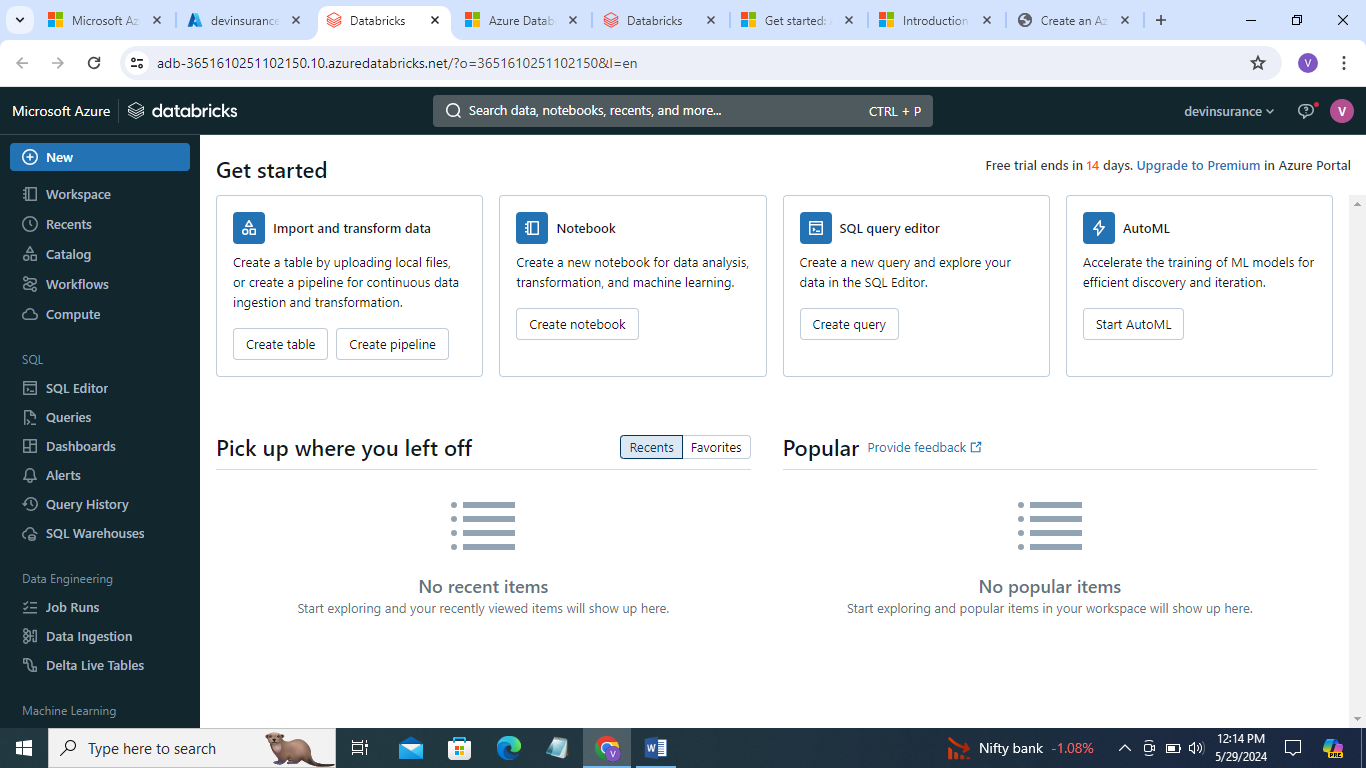
1. Now go to the databricks



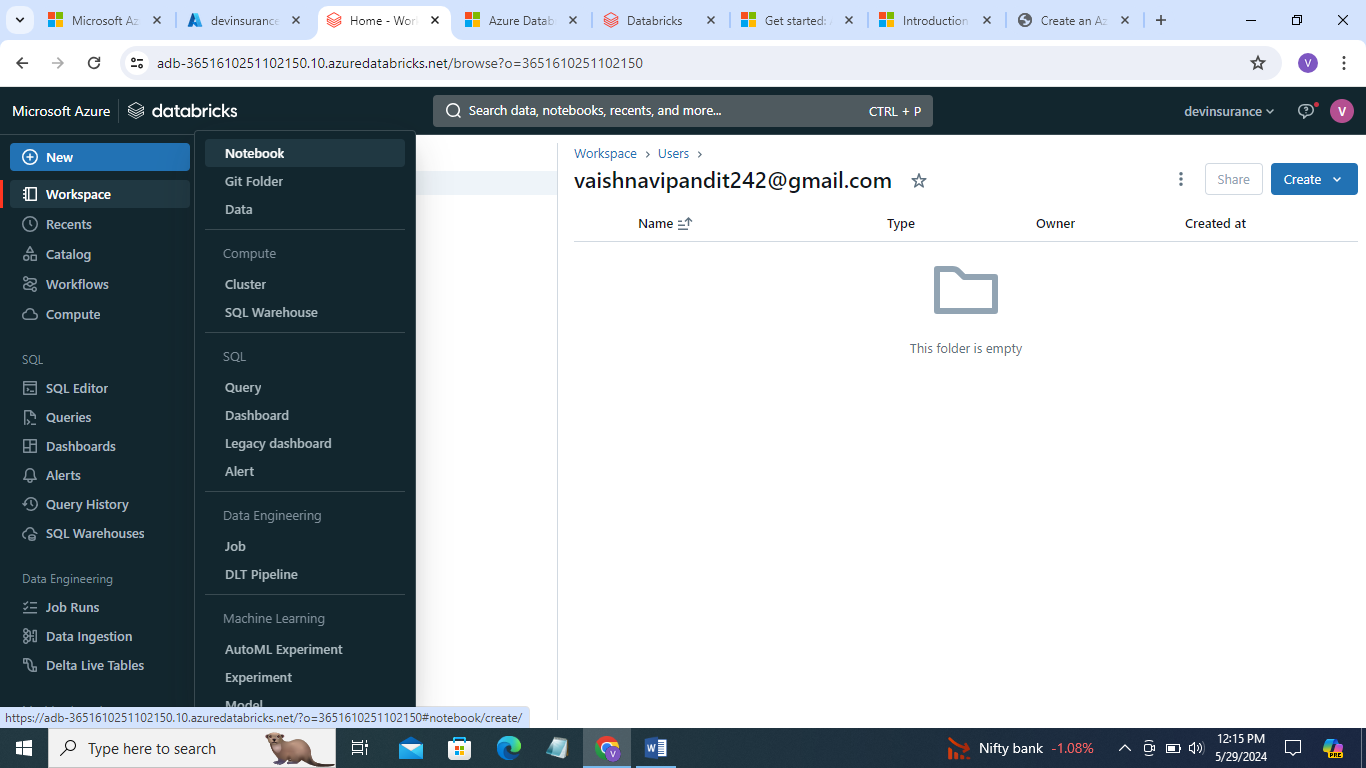
1. Go down you will find launch workspace option click that



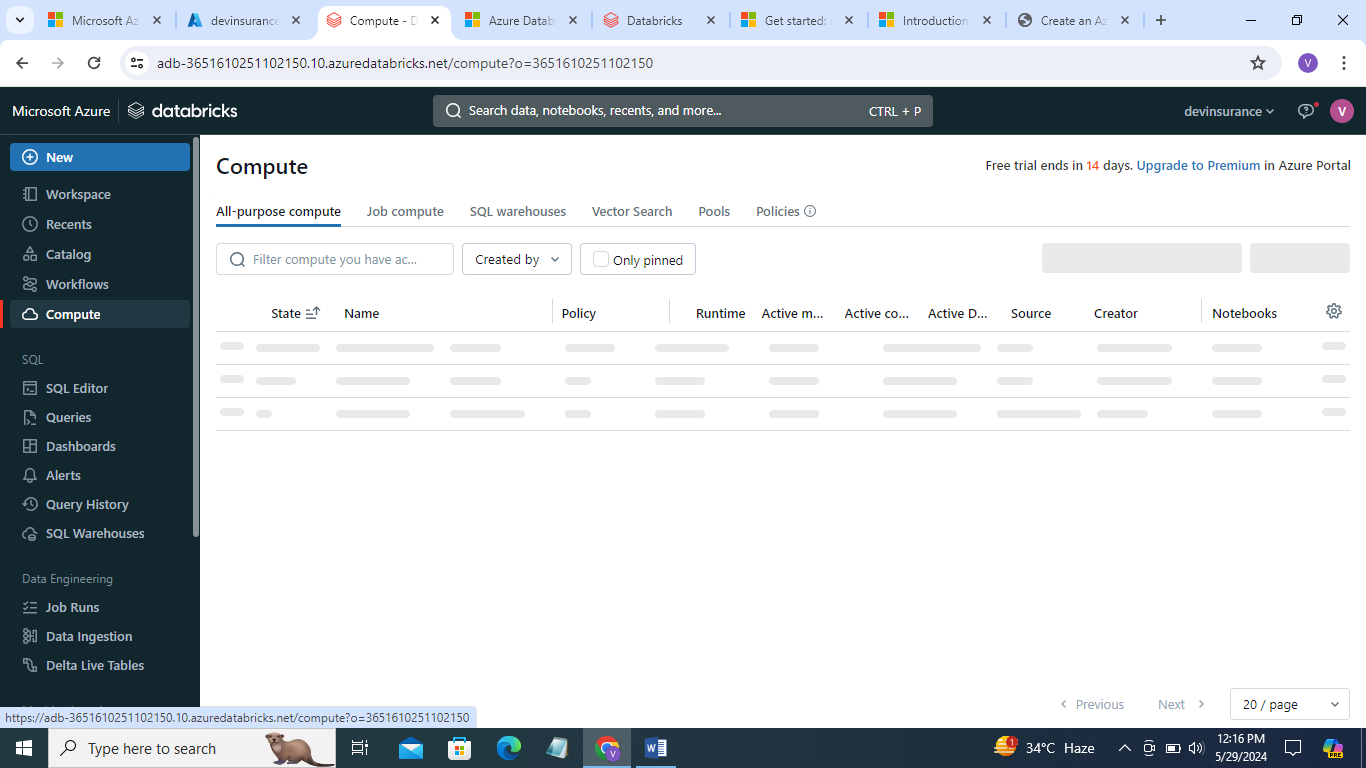
1. Now go to workspace



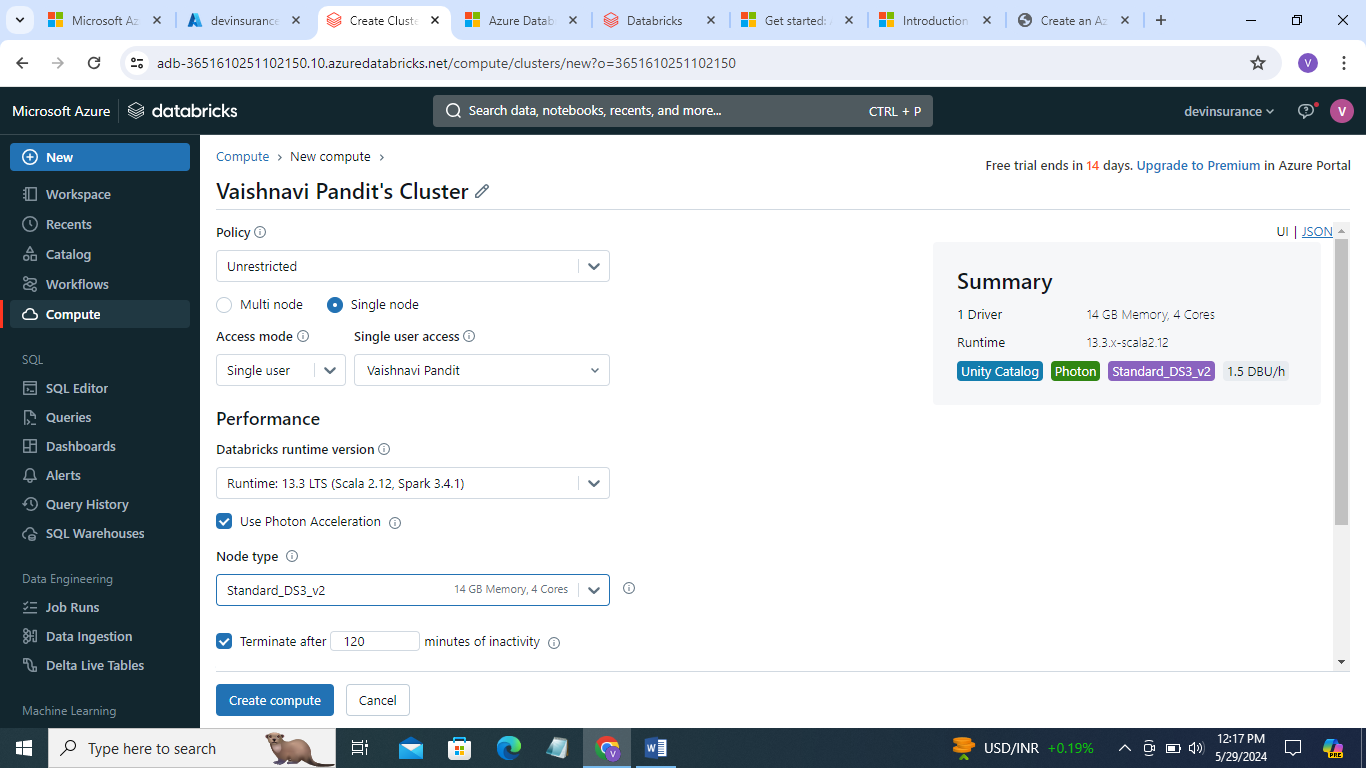
1. Create notebook clicking on new



1. Now go to the compute



1. Create compute add single node



1. Now go to the notebook

And write the code to flattern the raw data into structured data

This output data will be in table format.

Account key= DoaxB6QXcYCJB0aEW3REkQ69kSZKBC8hbsoaVzy84mfi+ORxwxD5RPvaQeftrrwkxn8GvKfZ21ew+ASt/TWSqQ==

Get account key from access keys in container

from pyspark.sql import \*

from pyspark.sql.functions import \*

spark.conf.set('fs.azure.account.key.dev0011insuranceaccount.dfs.core.windows.net','DoaxB6QXcYCJB0aEW3REkQ69kSZKBC8hbsoaVzy84mfi+ORxwxD5RPvaQeftrrwkxn8GvKfZ21ew+ASt/TWSqQ==')

raw\_path='abfss://rawlayer@dev0011insuranceaccount.dfs.core.windows.net/fhinsurance/input/farmers-protest-tweets-2021-03-5.json'

df=spark.read.format('json').load(raw\_path)

display(df)

df.printSchema()

def flatten(df):

   # compute Complex Fields (Lists and Structs) in Schema

   complex\_fields = dict([(field.name, field.dataType)

                             for field in df.schema.fields

                             if type(field.dataType) == ArrayType or  type(field.dataType) == StructType])

   while len(complex\_fields)!=0:

      col\_name=list(complex\_fields.keys())[0]

      print ("Processing :"+col\_name+" Type : "+str(type(complex\_fields[col\_name])))

      # if StructType then convert all sub element to columns.

      # i.e. flatten structs

      if (type(complex\_fields[col\_name]) == StructType):

         expanded = [col(col\_name+'.'+k).alias(col\_name+'\_'+k) for k in [ n.name for n in  complex\_fields[col\_name]]]

         df=df.select("\*", \*expanded).drop(col\_name)

      # if ArrayType then add the Array Elements as Rows using the explode function

      # i.e. explode Arrays

      elif (type(complex\_fields[col\_name]) == ArrayType):

         df=df.withColumn(col\_name,explode\_outer(col\_name))

      # recompute remaining Complex Fields in Schema

      complex\_fields = dict([(field.name, field.dataType)

                             for field in df.schema.fields

                             if type(field.dataType) == ArrayType or  type(field.dataType) == StructType])

   return df

df\_flatten = flatten(df)

df\_flatten.show()

or

def read\_nested\_json(df):

column\_list = []

for column\_name in df.schema.names:

if isinstance(df.schema[column\_name].dataType, ArrayType):

df = df.withColumn(column\_name, explode(column\_name))

column\_list.append(column\_name)

elif isinstance(df.schema[column\_name].dataType, StructType):

for field in df.schema[column\_name].dataType.fields:

column\_list.append(col(column\_name + "." + field.name).alias(column\_name + "\_" + field.name))

else:

column\_list.append(column\_name)

df = df.select(column\_list)

return df

def flatten(df):

read\_nested\_json\_flag = True

while read\_nested\_json\_flag:

df = read\_nested\_json(df)

read\_nested\_json\_flag = False

for column\_name in df.schema.names:

if isinstance(df.schema[column\_name].dataType, ArrayType):

read\_nested\_json\_flag = True

elif isinstance(df.schema[column\_name].dataType, StructType):

read\_nested\_json\_flag = True

return df

ndf = flatten(df)

ndf.printSchema()