KAFKA ASSIGNMENT

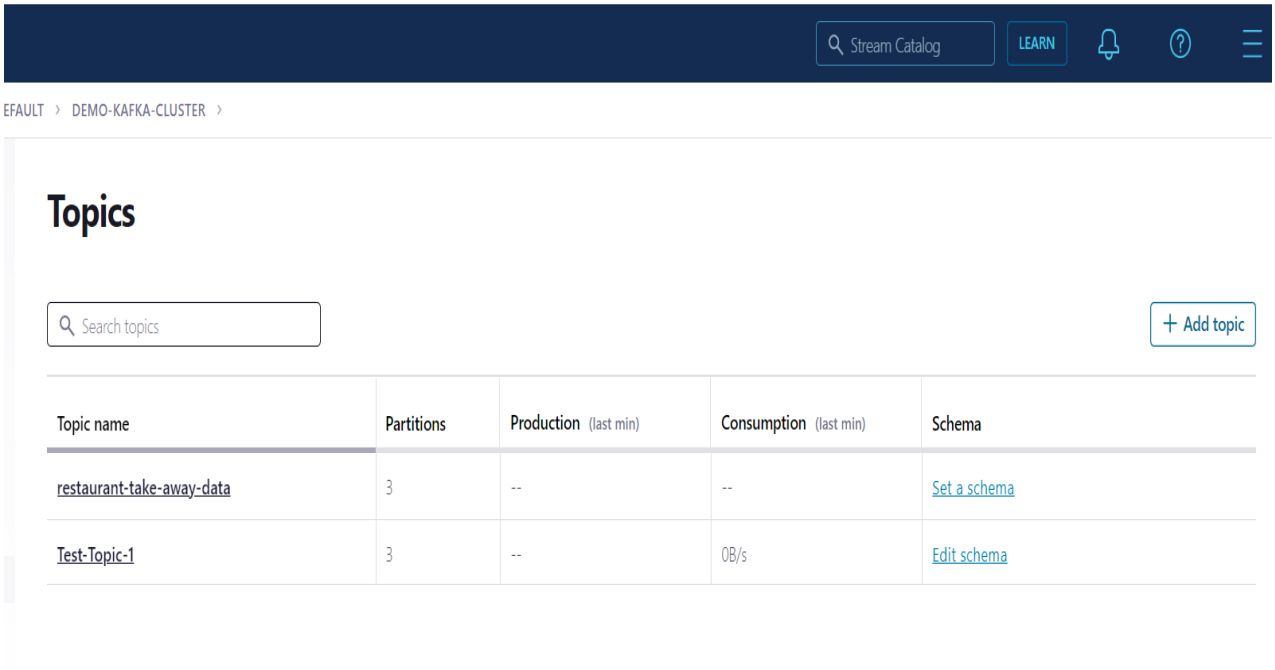
-Download restaurant data

Link: https://github.com/shashank-mishra219/Confluent-Kafka-Setup/blob/main/restaurant_orders.csv

1. Setup Confluent Kafka Account

2. Create one kafka topic named as "restaurant-take-away-data" with 3 partitions

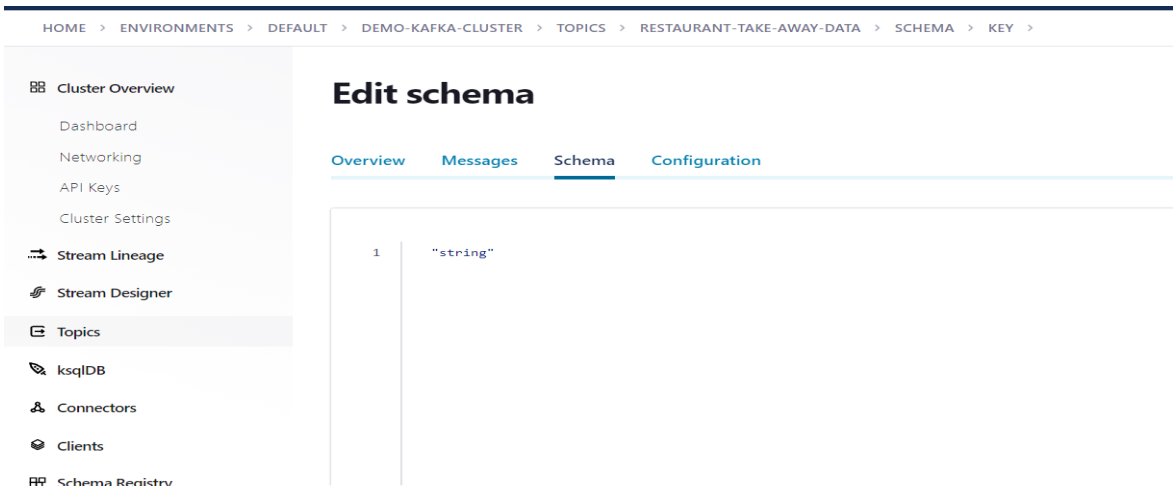
Topic 'restaurant-take-away-data ' with 3 partitions is created



-Setup key (string) & value (json) schema in the confluent schema registry

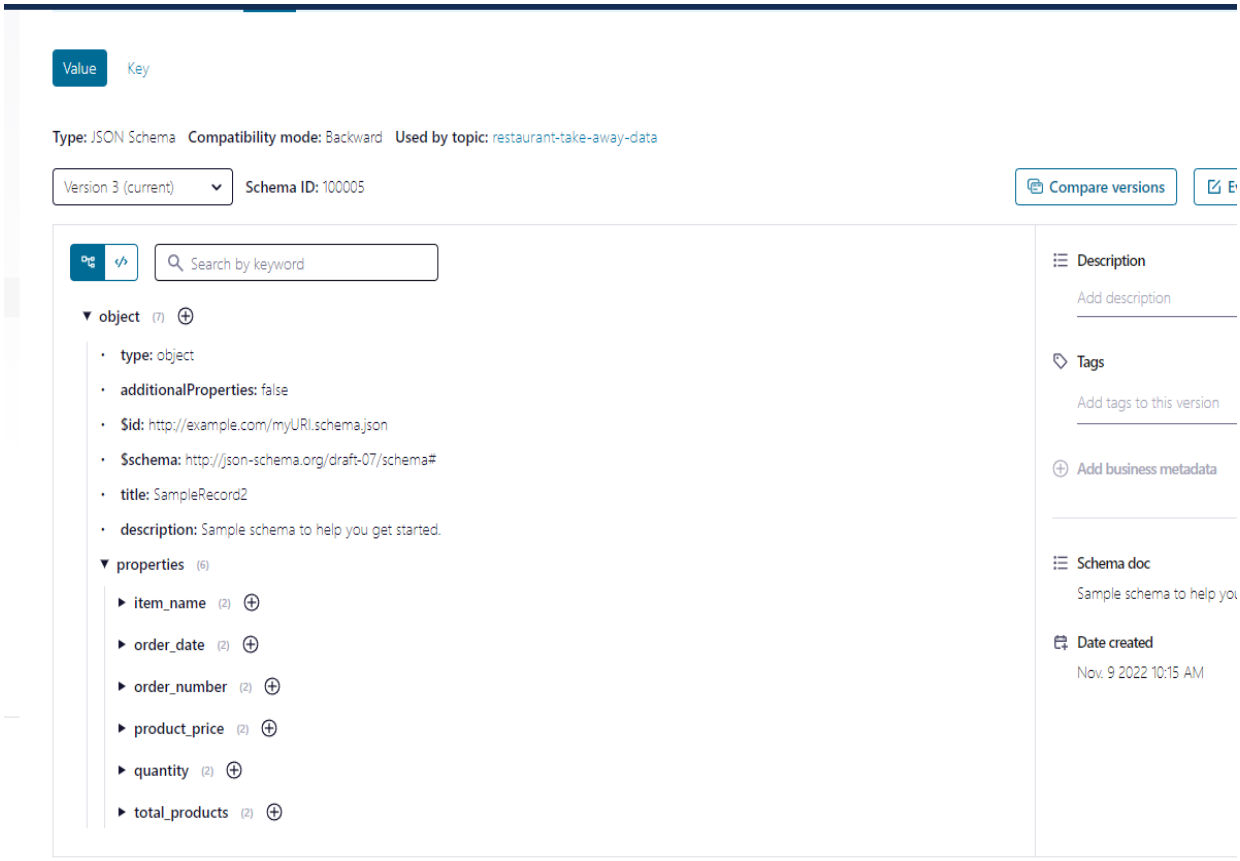
Key Schema:

```
"string"
```



Value Schema:

```
{
  "$id": "http://example.com/myURI.schema.json",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "additionalProperties": false,
  "description": "Sample schema to help you get started.",
  "properties": { "item_name": {
      "description": "The type(v) type is used.",
      "type": "string"
    },
    "order_date": {
      "description": "The type(v) type is used.",
      "type": "string"
    },
    "order_number": {
      "description": "The type(v) type is used.",
      "type": "number"
    },
    "product_price": {
      "description": "The type(v) type is used.",
      "type": "number"
    },
    "quantity": {
      "description": "The type(v) type is used.",
      "type": "number"
    },
    "total_products": {
      "description": "The type(v) type is used.",
      "type": "number"
    }
  },
  "title": "SampleRecord2",
  "type": "object"
}
```



-Write a kafka producer program (python or any other language) to read data records from restaurant data csv file, make sure schema is not hardcoded in the producer code, read the latest version of schema and schema_str from schema registry and use it for data serialization.

From producer code, publish data in Kafka Topic one by one and use dynamic key while publishing the records into the Kafka Topic

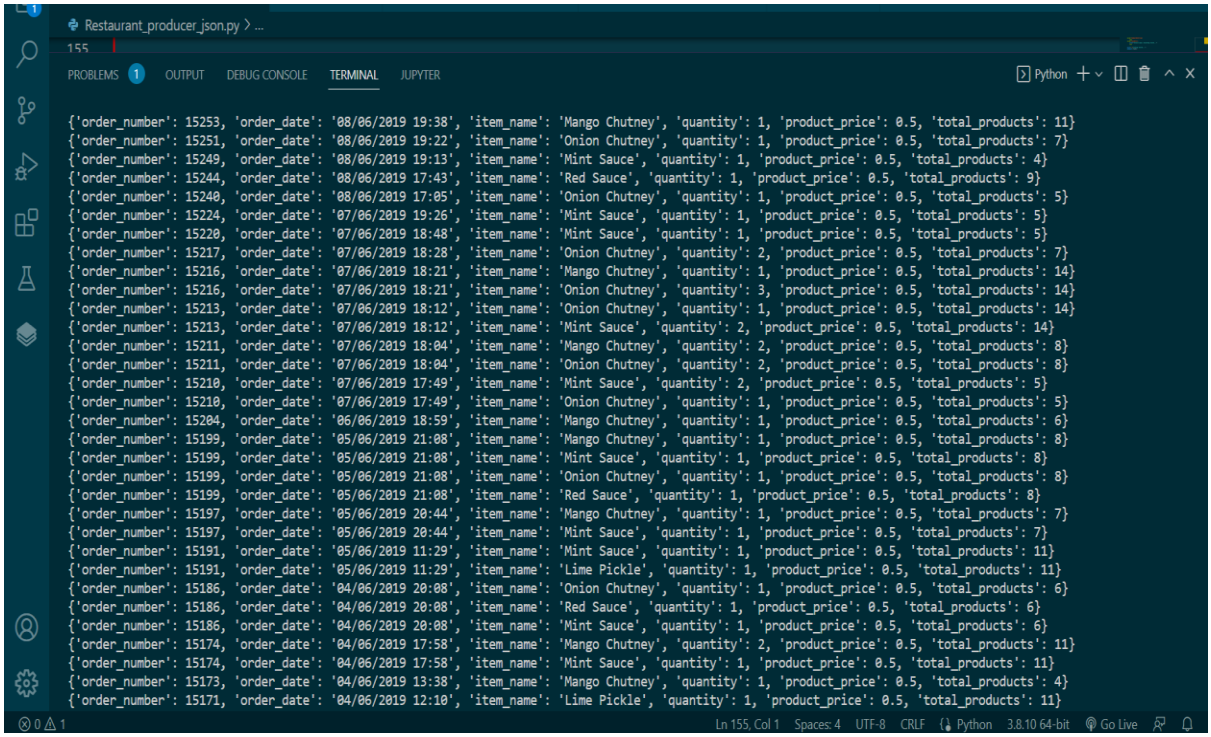
Consumer Code:

```
import argparse
from uuid import uuid4
from six.moves import input
from confluent_kafka import Producer
from confluent_kafka.serialization import StringSerializer,
SerializationContext, MessageField
from confluent_kafka.schema_registry import SchemaRegistryClient
from confluent_kafka.schema_registry.json_schema import JSONSerializer
#from confluent_kafka.schema_registry import *
import pandas as pd
from typing import List

FILE_PATH = "/Users/namrt/OneDrive/Desktop/Kafka/restaurant_orders.csv"
columns=['order_number', 'order_date', 'item_name', 'quantity',
'product_price', 'total_products']
```

```
API_KEY = '3DKHZA77TCORTQVM'
ENDPOINT_SCHEMA_URL  = 'https://psrc-zj6ny.us-east-2.aws.confluent.cloud'
API_SECRET_KEY =
'0c+PCR+sp1cEL0ns6tBTZl8bA9Z+AmpYEPSgQ4DfnsiwUKPRSR2vZs2Cx6f8IfoL'
BOOTSTRAP_SERVER = 'pkc-ymrq7.us-east-2.aws.confluent.cloud:9092'
SECURITY_PROTOCOL = 'SASL_SSL'  # authetication mechanism for encrypted and
secured connection
SSL_MACHENISM = 'PLAIN'
SCHEMA_REGISTRY_API_KEY = 'G6WDT4VOCBSLA4CO'
SCHEMA_REGISTRY_API_SECRET =
'28e5JcKaaXek7dbw6SCOoaX0T0UNnuYUg3wopsPqRHy9LrEfhqGB9pmquUbT1Pqe'

# for kafka cluster connectivity
def sasl_conf():

    sasl_conf = {'sasl.mechanism': SSL_MACHENISM,
                 # Set to SASL_SSL to enable TLS support.
                #  'security.protocol': 'SASL_PLAINTEXT'}
                'bootstrap.servers':BOOTSTRAP_SERVER,
                'security.protocol': SECURITY_PROTOCOL,
                'sasl.username': API_KEY,
                'sasl.password': API_SECRET_KEY
                }
    return sasl_conf


# for schema registry connectivity
def schema_config():
    return {'url':ENDPOINT_SCHEMA_URL,

    'basic.auth.user.info':f"{SCHEMA_REGISTRY_API_KEY}:{SCHEMA_REGISTRY_API_SE
CRET}"

    }


class Restaurant:
    def __init__(self,record:dict):
        for k,v in record.items():
            setattr(self,k,v)

        self.record=record

    @staticmethod
    def dict_to_restaurant(data:dict,ctx):
        return Restaurant(record=data)

    def __str__(self):
```

```
        return f"{self.record}"


def get_restaurant_instance(file_path):
    df=pd.read_csv(file_path)
    df=df.iloc[:,:]
    restaurants:List[Restaurant]=[]
    for data in df.values:
        restaurant=Restaurant(dict(zip(columns,data)))
        restaurants.append(restaurant)
        yield restaurant

def restaurant_to_dict(restaurant:Restaurant, ctx):
    """
    Returns a dict representation of a User instance for serialization.
    Args:
        user (User): User instance.
        ctx (SerializationContext): Metadata pertaining to the serialization
            operation.
    Returns:
        dict: Dict populated with user attributes to be serialized.
    """

    # User._address must not be serialized; omit from dict
    return restaurant.record


def delivery_report(err, msg):
    """
    Reports the success or failure of a message delivery.
    Args:
        err (KafkaError): The error that occurred on None on success.
        msg (Message): The message that was produced or failed.
    """

    if err is not None:
        print("Delivery failed for User record {}: {}".format(msg.key(), err))
        return
    print('User record {} successfully produced to {} [{}] at offset
{}'.format(
        msg.key(), msg.topic(), msg.partition(), msg.offset()))


def main(topic):
    schema_registry_conf = schema_config()
    schema_registry_client = SchemaRegistryClient(schema_registry_conf)
    schema_str = schema_registry_client.get_latest_version('restaurant-take-
away-data-value').schema.schema_str # for taking latest schema
```

```
    string_serializer = StringSerializer('utf_8')  # utf-8 -> encoding -> to
handle all unique codes in any particular string
    json_serializer = JSONSerializer(schema_str,
schema_registry_client,restaurant_to_dict)

    producer = Producer(sasl_conf())

    print("Producing user records to topic {}. ^C to exit.".format(topic))
    #while True:
        # Serve on_delivery callbacks from previous calls to produce()
    producer.poll(0.0)
    try:
        for restaurant in get_restaurant_instance(file_path=FILE_PATH):

            print(restaurant)
            producer.produce(topic=topic,
                             key=string_serializer(str(uuid4()),
restaurant_to_dict),
                             value=json_serializer(restaurant,
SerializationContext(topic, MessageField.VALUE)),
                             on_delivery=delivery_report)
            #break
            # just for testing purpose
    except KeyboardInterrupt:
        pass
    except ValueError:
        print("Invalid input, discarding record...")
        pass

    print("\nFlushing records...")
    producer.flush()

main("restaurant-take-away-data")
```



-Write kafka consumer code and create two copies of same consumer code and save it with different names (kafka_consumer_1.py & kafka_consumer_2.py), again make sure lates schema version and schema_str is not hardcoded in the consumer code, read it automatically from the schema registry to deserialize the data.

Now test two scenarios with your consumer code:

a.) Use "group.id" property in consumer config for both consumers and mention different group_ids in kafka_consumer_1.py & kafka_consumer_2.py,

apply "earliest" offset property in both consumers and run these two consumers from two different terminals. Calculate how many records each consumer consumed and printed on the terminal

b.) Use "group.id" property in consumer config for both consumers and mention same group_ids in kafka_consumer_1.py & kafka_consumer_2.py,

apply "earliest" offset property in both consumers and run these two consumers from two different terminals. Calculate how many records each consumer consumed and printed on the terminal

```
import argparse
from confluent_kafka import Consumer
from confluent_kafka.serialization import SerializationContext, MessageField
from confluent_kafka.schema_registry.json_schema import JSONDeserializer
from confluent_kafka.schema_registry import SchemaRegistryClient


API_KEY = '3DKHZA77TCORTQVM'
ENDPOINT_SCHEMA_URL  = 'https://psrc-zj6ny.us-east-2.aws.confluent.cloud'
```

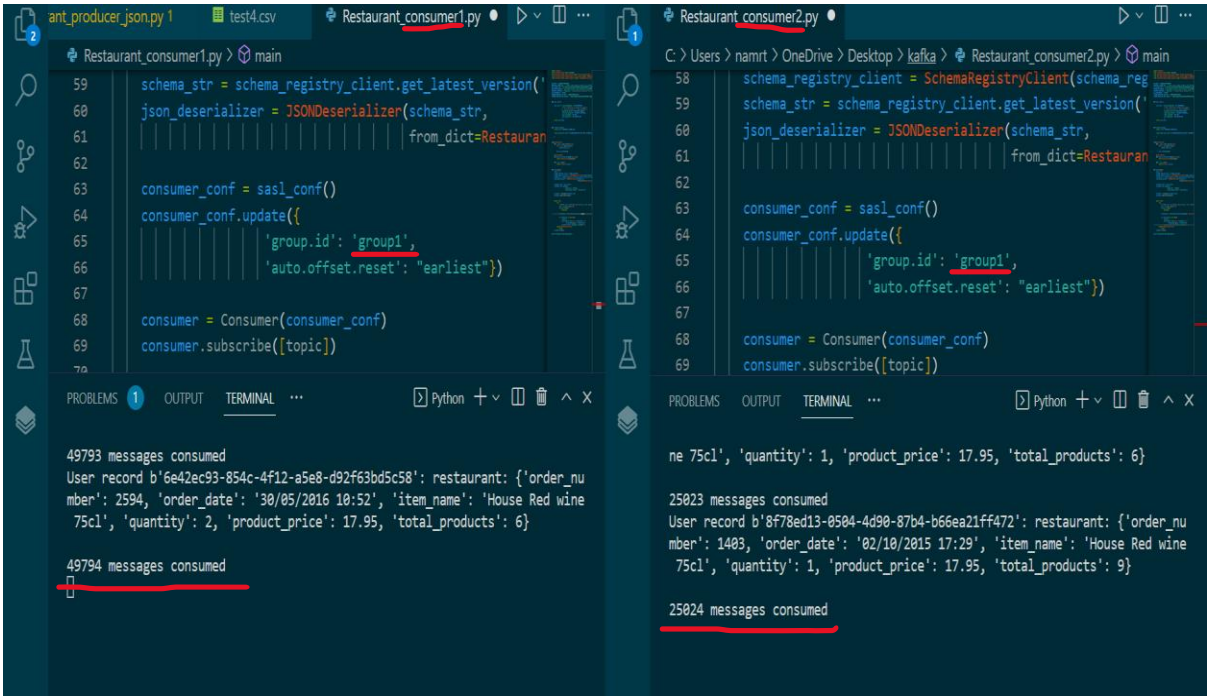
```
API_SECRET_KEY =
'0c+PCR+sp1cEL0ns6tBTZl8bA9Z+AmpYEPSgQ4DfnsiwUKPRSR2vZs2Cx6f8IfoL'
BOOTSTRAP_SERVER = 'pkc-ymrq7.us-east-2.aws.confluent.cloud:9092'
SECURITY_PROTOCOL = 'SASL_SSL'  # authetication mechanism for encrypted and
secured connection
SSL_MACHENISM = 'PLAIN'
SCHEMA_REGISTRY_API_KEY = 'G6WDT4VOCBSLA4CO'
SCHEMA_REGISTRY_API_SECRET =
'28e5JcKaaXek7dbw6SCOoaX0T0UNnuYUg3wopsPqRHy9LrEfhqGB9pmquUbT1Pqe'


def sasl_conf():

    sasl_conf = {'sasl.mechanism': SSL_MACHENISM,
                 # Set to SASL_SSL to enable TLS support.
                #  'security.protocol': 'SASL_PLAINTEXT'}
                'bootstrap.servers':BOOTSTRAP_SERVER,
                'security.protocol': SECURITY_PROTOCOL,
                'sasl.username': API_KEY,
                'sasl.password': API_SECRET_KEY
                }
    return sasl_conf



def schema_config():
    return {'url':ENDPOINT_SCHEMA_URL,

    'basic.auth.user.info':f"{SCHEMA_REGISTRY_API_KEY}:{SCHEMA_REGISTRY_API_SE
CRET}"

    }


class Restaurant:
    def __init__(self,record:dict):
        for k,v in record.items():
            setattr(self,k,v)

        self.record=record

    @staticmethod
    def dict_to_restaurant(data:dict,ctx):
        return Restaurant(record=data)

    def __str__(self):
        return f"{self.record}"
```

```
def main(topic):
    count = 0
    schema_registry_conf = schema_config()
    schema_registry_client = SchemaRegistryClient(schema_registry_conf)
    schema_str = schema_registry_client.get_latest_version('restaurant-take-
away-data-value').schema.schema_str # for taking latest schema
    json_deserializer = JSONDeserializer(schema_str,
                                         from_dict=Restaurant.dict_to_restaura
nt)

    consumer_conf = sasl_conf()
    consumer_conf.update({
                     'group.id': 'group1',
                     'auto.offset.reset': "earliest"})

    consumer = Consumer(consumer_conf)
    consumer.subscribe([topic])


    while True:
        try:
            # SIGINT can't be handled when polling, limit timeout to 1 second.
            msg = consumer.poll(1.0)
            if msg is None:
                continue

            restaurant = json_deserializer(msg.value(),
SerializationContext(msg.topic(), MessageField.VALUE))

            if restaurant is not None:
                count+=1
                print("User record {}: restaurant: {}\n"
                      .format(msg.key(), restaurant))
                print(f"{count} messages consumed")
        except KeyboardInterrupt:
            break
    consumer.close()

main("restaurant-take-away-data")
```

For testing, we need to change group id only, rest code will remain same.

```
    consumer_conf.update({
                     'group.id': 'group1',
                     'auto.offset.reset': "earliest"})
```

a) group id for consumer1 and consumer2 is same.
   Task gets distributed among the number of consumers we have. Load sharing results in fast processing.



Total messages: 74818
Consumer1: 49794
Consumer2: 25024

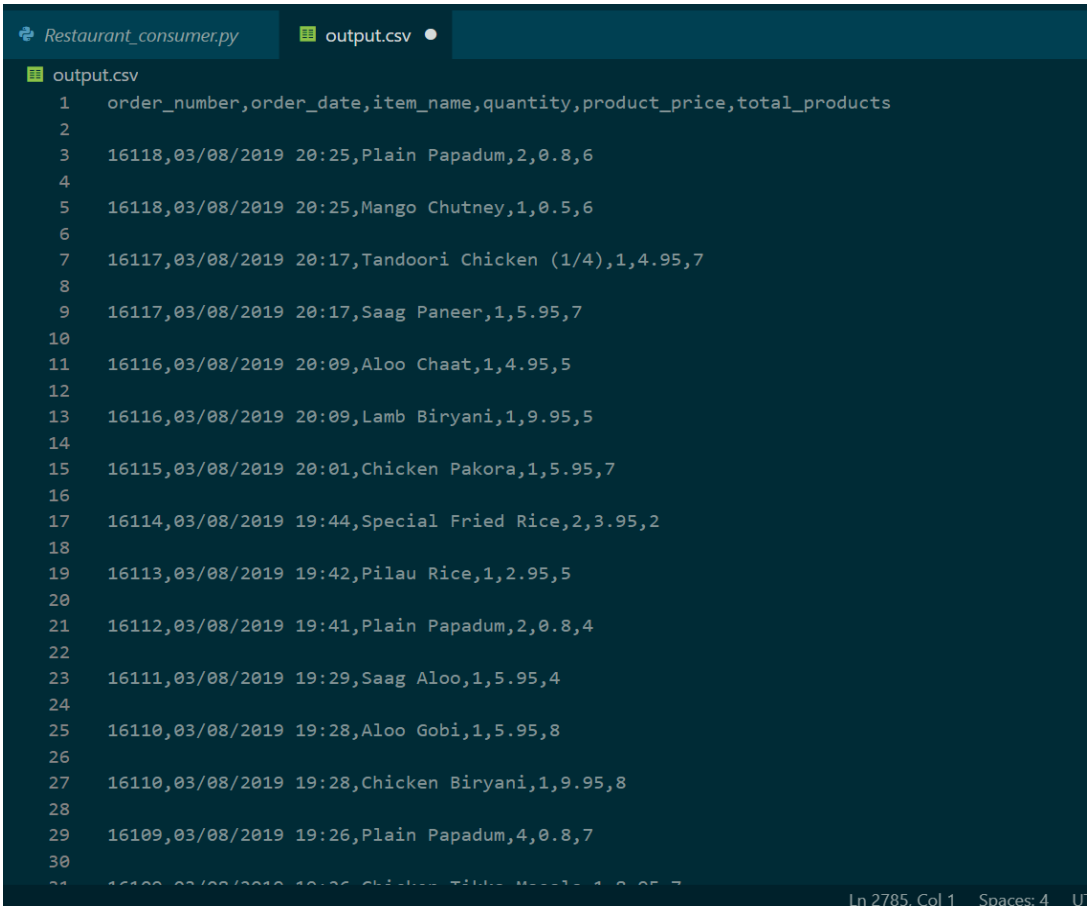
b) Group if for consumer1 and consumer2 is different.
Both will perform their own task.
Total messages: 74818
Consumer1: 74818
Consumer2: 74818

-Once above questions are done, write another kafka consumer to read data from kafka topic and from the consumer code create one csv file "output.csv" and append consumed records output.csv file

```
import argparse
import csv
from confluent_kafka import Consumer
from confluent_kafka.serialization import SerializationContext, MessageField
from confluent_kafka.schema_registry.json_schema import JSONDeserializer
from confluent_kafka.schema_registry import SchemaRegistryClient


API_KEY = '3DKHZA77TCORTQVM'
ENDPOINT_SCHEMA_URL  = 'https://psrc-zj6ny.us-east-2.aws.confluent.cloud'
API_SECRET_KEY =
'0c+PCR+sp1cEL0ns6tBTZl8bA9Z+AmpYEPSgQ4DfnsiwUKPRSR2vZs2Cx6f8IfoL'
BOOTSTRAP_SERVER = 'pkc-ymrq7.us-east-2.aws.confluent.cloud:9092'
```

```
SECURITY_PROTOCOL = 'SASL_SSL'  # authentication mechanism for encrypted and
secured connection
SSL_MACHENISM = 'PLAIN'
SCHEMA_REGISTRY_API_KEY = 'G6WDT4VOCBSLA4CO'
SCHEMA_REGISTRY_API_SECRET =
'28e5JcKaaXek7dbw6SCOoaX0T0UNnuYUg3wopsPqRHy9LrEfhqGB9pmquUbT1Pqe'


def sasl_conf():

    sasl_conf = {'sasl.mechanism': SSL_MACHENISM,
                 # Set to SASL_SSL to enable TLS support.
                #  'security.protocol': 'SASL_PLAINTEXT'}
                'bootstrap.servers':BOOTSTRAP_SERVER,
                'security.protocol': SECURITY_PROTOCOL,
                'sasl.username': API_KEY,
                'sasl.password': API_SECRET_KEY
                }
    return sasl_conf



def schema_config():
    return {'url':ENDPOINT_SCHEMA_URL,

    'basic.auth.user.info':f"{SCHEMA_REGISTRY_API_KEY}:{SCHEMA_REGISTRY_API_SE
CRET}"

    }


class Restaurant:
    def __init__(self,record:dict):
        for k,v in record.items():
            setattr(self,k,v)

        self.record=record

    @staticmethod
    def dict_to_restaurant(data:dict,ctx):
        return Restaurant(record=data)

    def __str__(self):
        return f"{self.record}"


def main(topic):
    count = 0
    schema_registry_conf = schema_config()
```

```
    schema_registry_client = SchemaRegistryClient(schema_registry_conf)
    schema_str = schema_registry_client.get_latest_version('restaurant-take-
away-data-value').schema.schema_str # for taking latest schema
    json_deserializer = JSONDeserializer(schema_str,
                                         from_dict=Restaurant.dict_to_restaura
nt)

    consumer_conf = sasl_conf()
    consumer_conf.update({
                     'group.id': 'group1',
                     'auto.offset.reset': "earliest"})

    consumer = Consumer(consumer_conf)
    consumer.subscribe([topic])
    final = []
    columns=['order_number', 'order_date', 'item_name', 'quantity',
'product_price', 'total_products']


    while True:
        try:
            # SIGINT can't be handled when polling, limit timeout to 1 second.
            msg = consumer.poll(1.0)
            if msg is None:
                with open('output.csv', 'w') as csvfile:
                    writer = csv.DictWriter(csvfile, fieldnames = columns)
                    writer.writeheader()
                    writer.writerows(final)
                continue

            restaurant = json_deserializer(msg.value(),
SerializationContext(msg.topic(), MessageField.VALUE))

            if restaurant is not None:
                count+=1
                print("User record {}: restaurant: {}"
                      .format(msg.key(), restaurant))
                final.append(restaurant.record)
                print(f"{count} messages consumed\n")
        except KeyboardInterrupt:
            break
    consumer.close()

main("restaurant-take-away-data")
```


Restaurant_consumer.py
output.csv
output.csv
order_number,order_date,item_name,quantity,product_price,total_products
16118,03/08/2019 20:25,Plain Papadum,2,0.8,6
16118,03/08/2019 20:25,Mango Chutney,1,0.5,6
16117,03/08/2019 20:17,Tandoori Chicken (1/4),1,4.95,7
16117,03/08/2019 20:17,Saag Paneer,1,5.95,7
16116,03/08/2019 20:09,Aloo Chaat,1,4.95,5
16116,03/08/2019 20:09,Lamb Biryani,1,9.95,5
16115,03/08/2019 20:01,Chicken Pakora,1,5.95,7
16114,03/08/2019 19:44,Special Fried Rice,2,3.95,2
16113,03/08/2019 19:42,Pilau Rice,1,2.95,5
16112,03/08/2019 19:41,Plain Papadum,2,0.8,4
16111,03/08/2019 19:29,Saag Aloo,1,5.95,4
16110,03/08/2019 19:28,Aloo Gobi,1,5.95,8
16110,03/08/2019 19:28,Chicken Biryani,1,9.95,8
16109,03/08/2019 19:26,Plain Papadum,4,0.8,7
Ln 2785, Col 1
Spaces: 4