## Assumptions Made:

* Task Intention: The initial task lacked clarity and detail, requiring reliance on best practices.
* Use of Docker Compose: Employing Docker Compose for service orchestration allowed for secure communication via automatic network creation.
* Data Fetching Method: CSV data is assumed to be pre-downloaded for simplicity, rather than dynamically streaming data over the internet.
* Dynamic CSV Parsing: The producer dynamically parses CSV into headers and content for message creation.
* Because the assignment said single-threaded, that Spring Boot would be overkill for the task’s purposes.

## Things to Improve Performance and Security

* Explore Kafka Connect and Streams as options to ease the transfer of large amounts of data.
* Increase partition / replication factors.
* Enable confluence balancer.
* Encrypt the producer, consumer and zookeeper. This can be done by enabling SSL, for example.
* Create KeyStore and Truststore to leverage security checks in tandem with encryption and setting passwords on them.

## Challenges Encountered:

* Ambiguity: Ambiguous initial instructions required interpretation, leading a lot of time spent to exploratory approaches in setting up the whole system.
* Service Deployment: Lack of broker or managing service hindered direct topic creation, necessitating Docker Compose utilization.
* Data Retrieval: Dynamic data fetching from kaggle posed challenges due to potential complexity and security concerns.
* Not having worked with Kafka before: meant that a lot of time was spent learning it from the ground up first which ate into time
* Time: Ultimately, this all meant that the task couldn’t be completed in it’s entirety, missing out communication from the Consumer back to the Producer

## Things to Improve:

* Entire structure of the app, segregating into different classes perhaps and maybe even different packages.
* Error handling, as it is very sparse currently due to time limitations
* Implement testing of the methods used to manipulate csv data. Could not do as ran out of time.
* Containerising the whole thing, including the producer and consumer pps to run on the same docker network would make things easier to execute.
* Using spring boot to simplify the set up for each application and leveraging native kafka and gradle support to speed up the dev process.
* Using spring would then make it easier to add a controller / scheduled event to control when the messages get pushed.

## Extension of Solution

To extend the solution to report on import/export by commodity in addition to country\_or\_area, several options can be considered:

Modify Data Structure:

* Expand the JSON message structure to include commodity information along with country\_or\_area.
* Update producer to parse and include commodity information in the JSON message.

Aggregate by Commodity:

* Maintain separate data structures or maps for import/export totals grouped by commodity.

Topic Partitioning:

* Allows for more efficient consumption and processing of messages related to specific country\_or\_area and commodity combinations.