

REPORT:

SAMPLING DATA:

Basically here I created my own code to generate a file of more than 15 GB as a sample:

```
import json

def copy_json_batchwise(input_file, output_file, batch_size=1000):
    with open(input_file, 'r') as infile:
        with open(output_file, 'w') as outfile:
            for _ in range(2200): # Run for 10 iterations
                # Read a batch of entries from the input file
                batch = []
                for _ in range(batch_size):
                    line = infile.readline()
                    if not line:
                        break
                    batch.append(line.strip())

                # If no more entries, break the loop
                if not batch:
                    break

                # Write the batch to the output file
                for entry in batch:
                    outfile.write(entry)
                    outfile.write('\n') # Add newline after each entry

                # Ensure the output file ends with proper JSON format
                outfile.write("]\n") # Close the JSON array

if __name__ == "__main__":
    input_file = 'data.json' # Replace with the path to your input JSON file
    output_file = 'i211377_sample.json' # Replace with the path to the output JSON file
    copy_json_batchwise(input_file, output_file)
```

BATCH PREPROCESSING DATA:

Here we removed extra columns just considered some of them especially also_buy, removed those items which do not have also buy values means which was null.

```

import json

def preprocess_batch(data):
    preprocessed_data = []
    for entry in data:
        # Check if the "also_buy" field is not empty
        if entry.get("also_buy"):
            # Clean and format each entry
            cleaned_entry = {
                "title": entry.get("title", ""),
                "brand": entry.get("brand", ""),
                "also_buy": entry.get("also_buy", []),
                "asin": entry.get("asin", ""),
                # Add more keys as needed
            }
            preprocessed_data.append(cleaned_entry)
    return preprocessed_data

def main():
    chunk_size = 100 # Number of entries to read at a time
    with open('i211377_sample.json', 'r') as file:
        with open('preprocessed_data.json', 'w') as outfile:
            outfile.write("[\n") # Write opening bracket of the JSON array
            first_entry = True # Flag to track if it's the first entry being written
            # Read the file in chunks
            for i in range(5): # For testing purposes, consider removing this limit for the full file
                chunk = []
                for _ in range(chunk_size):
                    chunk = []
                    for _ in range(chunk_size):
                        line = file.readline()
                        if not line:
                            break
                        entry = json.loads(line)
                        preprocessed_entry = preprocess_batch([entry])
                        if preprocessed_entry: # Check if the entry is not empty after preprocessing
                            if not first_entry:
                                outfile.write(",\n") # Add comma and newline if it's not the first entry
                            else:
                                first_entry = False
                            json.dump(preprocessed_entry[0], outfile, indent=4)
                    outfile.write("\n") # Write closing bracket of the JSON array

if __name__ == "__main__":
    main()

```

APPLYING APRIORI ALGORITHM:

Started and run mongoDB and mongoDB compass as CUI 😊

Connect Edit View Collection Help

localhost:27017 ...

My Queries Databases config association_rules

kafka_data > association_rules

Documents 12 Aggregations Schema Indexes 2 Validation

Type a query: { field: 'value' } or [Generate query](#)

EXPLAIN Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE 1-3 of 3

```

1  _id: ObjectId('662be67e70d17074abd9b678')
2  rule: "99ee69e1-bccc-4eba-9b47-c854b817b01c"
3  confidence: Array (empty)

1  _id: ObjectId('662be6d070d17074abd9b679')
2  rule: "72b10920-302d-4e37-80f2-c43f30dc4a51"
3  confidence: Array (empty)

1  _id: ObjectId('662be70170d17074abd9b67a')
2  rule: "12ef9aef-2e68-4d20-9776-b4ac57d17259"
3  confidence: Array (empty)

```

ObjectId
String
Array

After the I created a producer which produces the values 😊

```

muhammad-Latitude-E5470:~/kafka$ python3 producer.py
68', 'B07CP532DP', 'B01FVRKZ4H', 'B0179ATZ9A', 'B073P7PHCD', 'B076B832TX', 'B072XTT9', 'B010V0WTP2', 'B071PFP967', 'B07H2Y255K', 'B012MKK0J6', 'B019M8S', 'B01M4KDSI7', 'B0159XAQ60', 'B01E377QRU', 'B0034XRCAG', 'B07C2M6K6Z', ['B018YRBB80', 'B07FD9HMPM', 'B017M55D14', 'B07KX6PPW6', 'B017', 'B0169ZHJDK', 'B00NSF70KM', 'B00MQ2R4E0', 'B00NV1VFP0', 'B01LOUFRQ', 'B01LYDMB6U', 'B01CIW8NAG', 'B00BIJC8I4', 'B00KU1NLGO', 'B019ZAYUB0', 'B01', 'IA'], ['B00VBVXVPI'], ['B01AHZSZ9A', 'B01I809NCO', 'B07219C7LQ', 'B06ZZBQMT4', 'B06XKWCOTT', 'B01H43Z5GY', 'B01H8BRVXY', 'B015W134LS', 'B01NBT', 'B0748C68ZX', 'B07CMD59N4', 'B01JRD11JE', 'B00125S044'], ['B06XY5N95G', 'B01LY4VKTL', 'B01EKRMG8C', 'B004SLKRY6', 'B074Z3QGMX', 'B01JOVOFRE', 'B0', 'B0758YYWMB', 'B06XKWCOTT', 'B016ENOSXQ', 'B0723CQH2L'], ['B077M99VNM', 'B07HVRPXHY', 'B07CP7DLL5', 'B07FDF7NZK', 'B0041T42NC', 'B07CKXYN', 'B0KKI3WG', 'B07C8LM5GQ', 'B00KXKH15', 'B07GT4H3SK', 'B00V6HPY7A', 'B076DSYLC3', 'B00S9OXZ3W', 'B07G612DWD', 'B017KE80G0', 'B00MOGZU6M', 'B000', 'B0091CC10G', 'B07JPC4LN6'], ['B00MEX510K'], ['B07CM33RK6', 'B009A882D0', 'B005TS13Y0', 'B001RNO30W', 'B00318CB52', 'B01LF765JS', 'B00W9DYAIY', 'NOIS4A', 'B07C5F6YDF', 'B012537H06', 'B003AIKE6E', 'B004GGU9QY', 'B01LX9T201', 'B003AIKE3C', 'B007P28FN0', 'B07BT98MVN', 'B00P2D5U7I', 'B001SE', 'B01M3R75G6', 'B00W9DYBDS', 'B00KX22PBC', 'B07D1TRJ94', 'B007KFQ3E0', 'B01301T4NA', 'B000000XB8', 'B01D10RT22', 'B002YAGIOG', 'B07BRDZVK2'], ['M3', 'B0756WNSN1', 'B07KWDN37M', 'B079VXGSP2', 'B07CB883LS', 'B07CQMP2NF', 'B07BNZMNTZ', 'B0774BMLF1', 'B0767LQ87P', 'B07CKRHC35', 'B01BHK034U', 'B0MCXB', 'B01LTHM35C', 'B075G5ZMMK', 'B01708W316'], ['B07CM33RK6', 'B009A882D0', 'B005TS13Y0', 'B001RNO30W', 'B00318CB52', 'B01LF765JS', 'B00W', 'B01HNOIS4A', 'B07C5F6YDF', 'B012537H06', 'B003AIKE6E', 'B004GGU9QY', 'B01LX9T201', 'B003AIKE3C', 'B007P28FN0', 'B07BT98MVN', 'B00P2D5U7I', 'B0', '7C', 'B01M3R75G6', 'B00W9DYBDS', 'B00KX22PBC', 'B07D1TRJ94', 'B007KFQ3E0', 'B01301T4NA', 'B000000XB8', 'B01D10RT22', 'B002YAGIOG', 'B07BRDZVK2', '7CM33RK6', 'B009A882D0', 'B005TS13Y0', 'B001RNO30W', 'B00318CB52', 'B01LF765JS', 'B00W9DYAIY', 'B01HNOIS4A', 'B07C5F6YDF', 'B012537H06', 'B003', 'B004GGU9QY', 'B01LX9T201', 'B003AIKE3C', 'B007P28FN0', 'B07BT98MVN', 'B00P2D5U7I', 'B001SE07C', 'B01M3R75G6', 'B00W9DYBDS', 'B00KX22PBC', '94', 'B007KFQ3E0', 'B01301T4NA', 'B000000XB8', 'B01D10RT22', 'B002YAGIOG', 'B07BRDZVK2'], ['B07CM33RK6', 'B009A882D0', 'B005TS13Y0', 'B001RNO3', '0318CB52', 'B01LF765JS', 'B00W9DYAIY', 'B01HNOIS4A', 'B07C5F6YDF', 'B012537H06', 'B003AIKE6E', 'B004GGU9QY', 'B01LX9T201', 'B003AIKE3C', 'B007', 'B07BT98MVN', 'B00P2D5U7I', 'B001SE07C', 'B01M3R75G6', 'B00W9DYBDS', 'B00KX22PBC', 'B07D1TRJ94', 'B007KFQ3E0', 'B01301T4NA', 'B000000XB8', 'B01D10RT22', 'B002YAGIOG', 'B07BRDZVK2'], ['B01AYZJUR4'] 3XB2H1', 'B01EFPJ95M', 'B075MZPWL1', 'B01CP8Y2I0', 'B06X96749N', 'B077FQPID5', 'B079N979JC', 'B01CD2CVTU', 'B07CVJ3CVT', 'B0777FQ086', 'B073ML', 'B075JH6VC3', 'B06ZYX6718', 'B078MSR1H1', 'B074QW8VTG', 'B06VQHTJV', 'B06XJCPX69', 'B07CNZY5C8', 'B0755TPY6L', 'B0130IAIIB', 'B07DLRRKY7', 'B0', 'B0777HWSFC', 'B00RCM5KFK', 'B079XMR27F', 'B07H14XZV8', 'B01L1Q1QC0', 'B01D4X6ZM0', 'B077NF1D7B', 'B07JZ5YMBN', 'B072LN3Y3D', 'B072VFPVM8', 'ZQMA', 'B07939SZVM', 'B07CL5H9RB', 'B00MY1DPK0', 'B0797MYW9S', 'B01K019L0K', 'B01F49BULE', 'B07FQ6FRRN', 'B07GJNQBMH', 'B07D72TBCJ', 'B00TFAOI', '75SD9MJF', 'B07C4Z56HD', 'B07D6GFMS', 'B07CW54TY2', 'B07B3TWQWR', 'B01HE3V7UW', 'B078VVLQ6J', '0385320434', 'B079PTBMXP', 'B07DKB669T', 'B07C', 'B0154RNB54', 'B07KGPYSMF', 'B07D28CY2Y', 'B00LFU25NY', 'B00BY53XAG', 'B00BT7XQ7W', 'B07CSP6M2C', 'B07BKPWCFK', 'B07KN9D8Y2', 'B077MRNGXC', 'SXDC', 'B01MF9ASUD', 'B01MV7MP9F', 'B01G3E1XT8', 'B074B95CRC', 'B072217Y5P', 'B0765P1VRW', 'B07KFPWL21', 'B01J3Y02GC', 'B07CKVHT3', 'B071HLSQU', '79Q6VCP7', 'B07BKTH925', 'B074ZZMKVR', 'B075YRR5L9', 'B01FA9UTLA', 'B06W2NGKGG', 'B074GLRYX', 'B0038292LA', 'B07C3PYXV9', 'B0799FG2N1', 'B009', 'B0783RDZQT', 'B06XW6HZBS', 'B01DNRAH16', 'B014B20XEA', 'B00GN3K918', 'B07L8T7WD', 'B07C9CPLJT', 'B07D1TRJ94', 'B00CE5SEK', 'B00J7FWM60', '1V', 'B00S1YR5DK', 'B00FXA9TKE', 'B074NZK8V9', 'B01DL9V488', 'B07KNBYDLS', 'B004NEHRKK', 'B071RYG2HF', 'B072JX4P31', 'B01CADK2MK', 'B01M0XVIUL', 'AIKE0K', 'B06XQSRHJZ', 'B07BRC7ZY4', 'B00C04DQF0', 'B075P11G4P', 'B01N3D4H2Z', 'B009USLJBM', 'B00IMB19A4', 'B01AA0ETJM', 'B07BJF788', 'B00Y8D', 'B07M946CPT', 'B071NMVFFV', 'B079H0GNJZ', 'B071K12Y65', 'B07K415W8V', 'B00P8VODGS', 'B06X8NP49V', 'B00ISRKNFM', 'B071JV30H6', 'B07CHM3VNG', 'B0

```

And a consumer too which consumes these values which I printed on each iteration:

