

FASTQ File Processing & Analysis Report

Step 1: Initial Setup

- **Environment:** Python 3.9.21 in a virtual environment
- **Goal:** Read and analyze a large compressed `.fastq.gz` file (~3.2 GB)
- **Data Format:** FASTQ (used in bioinformatics to store sequences with quality scores)

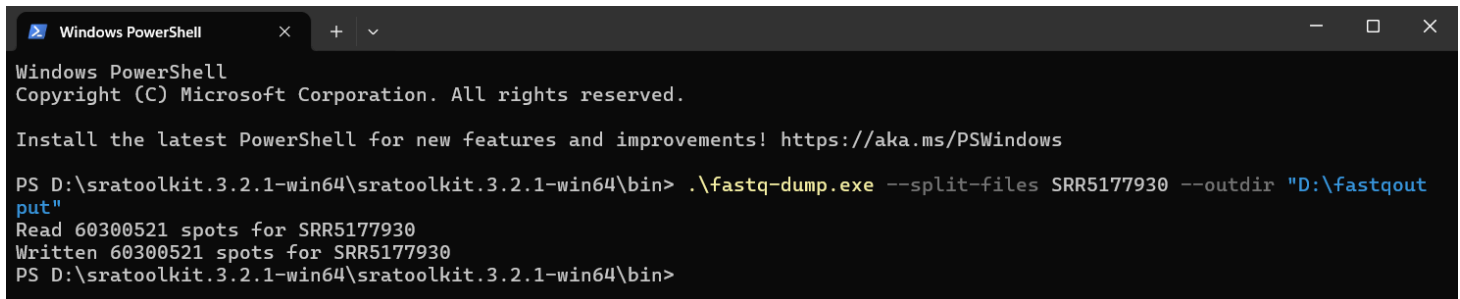
Step 2: Reading FASTQ with Biopython

We used `gzip + Bio.SeqIO` to read compressed FASTQ files directly, & to check the total number of records.

```
#Total Records in File
count = 0
with gzip.open(file_path, "rt") as handle:
    for _ in SeqIO.parse(handle, "fastq"):
        count += 1
print(f"Total records: {count}")
```

Total records: 60300521

to cross check:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\sratoolkit.3.2.1-win64\sratoolkit.3.2.1-win64\bin> .\fastq-dump.exe --split-files SRR5177930 --outdir "D:\fastqout"
Read 60300521 spots for SRR5177930
Written 60300521 spots for SRR5177930
PS D:\sratoolkit.3.2.1-win64\sratoolkit.3.2.1-win64\bin>
```

Step 3: Batch Conversion to Parquet

Since the file had **60M+ records**, we avoided loading everything into memory by splitting it into batches of 100,000:

```

import pandas as pd
import gzip
from Bio import SeqIO

batch_size = 100000
batch_num = 0
file_path = "your_file.fastq.gz"

with gzip.open(file_path, "rt") as handle:
    records = []
    for i, record in enumerate(SeqIO.parse(handle, "fastq")):
        records.append({
            "id": record.id,
            "sequence": str(record.seq),
            "quality": record.letter_annotations["phred_quality"]
        })

    if (i + 1) % batch_size == 0:
        df = pd.DataFrame(records)
        df.to_parquet(f"reads_batch_{batch_num}.parquet")
        print(f"Saved batch {batch_num}")
        records = []
        batch_num += 1

    if records:
        df = pd.DataFrame(records)
        df.to_parquet(f"reads_batch_{batch_num}.parquet")
        print(f"Saved final batch {batch_num}")

```

Result: 603 .parquet files created, each containing ~100,000 reads.

Files were then organized inside a folder: batches/

Step 4: Reading Parquet with DuckDB

We installed DuckDB:

```
pip install duckdb
```

Then queried all Parquet batches:

```
import duckdb

con = duckdb.connect()

df = con.execute("""
    SELECT id, sequence
    FROM 'batches/reads_batch_*.parquet'
""").fetchdf()
```

DuckDB handled filtering and selection directly on disk — super fast and RAM-efficient.

Output of duckdb: could not fetch the third column due to 60M records - memory error; but can fetch 60M sequences.

```
id \
0          SRR5177930.1
1          SRR5177930.2
2          SRR5177930.3
3          SRR5177930.4
4          SRR5177930.5
...
60300516   SRR5177930.9999996
60300517   SRR5177930.9999997
60300518   SRR5177930.9999998
60300519   SRR5177930.9999999
60300520   SRR5177930.10000000

sequence

0          ATGGCCCGAGGGAGACCCCTGCTGTCCGGTGTGCTAGTCCCTTTT...
1          ATGCTGGCCAGAGCCCAGAGGGAGAGGGCTCATCGGTCCATGGAGA...
2          ATGGTAAAGCATAGGGGCCATGCTAAAGAAACCACCACCAAGGAGA...
3          ATGAAATTAACTTTGGTGTCTGGGACAGTGATATTCTCATTCAAGC...
4          ATCTCAGAAAGGACAGAGGAAACTCTTCCTAATGACTGGCTGATGC...
...
60300516   ATACCCAAGCCCTGCTGCCTGCCCCAGTGGTCTGTGCCCAAAGACA...
60300517   ATCCACCAACCCCGCCAGCTGCTTGTGGCATAGTCCAGGATGTTG...
60300518   ATTGTCGGAGCTGAGCTGGGCCGGGGAGCTCTCCAGGGGCAGGAAG...
60300519   ATCATGATCAATTATCTGAGCAGAATTCTAGCTAGGTTAAGTAAGG...
60300520   ATACTAGATGGGATTTTGTCTCCCTCCATCTCCACTGCCAGGCAAT...

[60300521 rows x 2 columns]
```

Key Lessons

- **Don't read full FASTQ into memory** — batch + parquet is better.
- **Use Parquet** for storage efficiency and fast I/O.
- **DuckDB** is good for SQL-style querying over many files.
- `quality` column is complex (list of ints) — handle carefully in queries.