chatterboc

- chat1
- chat1
- bro
- asdasd
- sadfzxcv

chat1

```
import polars as pl

chunk_size = 1000  # Adjust based on available memory
dataframes = []

for i in range(0, len(a_list), chunk_size):
    chunk = a_list[i:i + chunk_size]
    dataframes.append(pl.DataFrame(chunk, schema=["stsdf", "abc"]))

combined_df = pl.concat(dataframes)
```

chat1

```
import polars as pl

def flag_offline_periods(df):
    # Count entries per bucket
    counts = df.groupby('timestamp_bucket').agg(pl.count('*').alias('count'))

    # Identify buckets with more than 20 entries
    high_count_buckets = counts.filter(pl.col('count') > 20)['timestamp_bucket']

    # Create a function to generate offline periods
    def generate_offline_periods(high_count_buckets):
        offline_periods = []
        for bucket in high_count_buckets:
            offline_end = bucket + pl.duration(seconds=30)
            offline_periods.append((bucket, offline_end))
        return offline_periods

    # Generate offline periods
    offline_periods = generate_offline_periods(high_count_buckets)

    # Merge overlapping periods
    merged_periods = []
    for start, end in sorted(offline_periods):
        if merged_periods and start <= merged_periods[-1][1]:
            merged_periods[-1] = (merged_periods[-1][0], max(merged_periods[-1][1], end))
        else:
            merged_periods.append((start, end))

    # Create a function to check if a timestamp is within offline periods
    def is_offline(timestamp):
        return any(start <= timestamp < end for start, end in merged_periods)

    # Apply the offline check to each row
    df = df.with_columns(
        pl.when(pl.col('timestamp_bucket').apply(is_offline))
        .then(pl.lit('OFFLINE'))
        .otherwise(pl.lit('ONLINE'))
        .alias('status')
    )
```

```
    return df

# Assuming your DataFrame is called 'df'
df = flag_offline_periods(df)
```

bro

```
#!/usr/bin/env bash

# Usage: ./group_by_date_hour_symbol_source.sh path_to_logfile

LOGFILE="$1"

awk '
  {
    # Split the first field (20250209-13:45:02.123456) on the dash.
    # dt[1] will hold "20250209", dt[2] will hold "13:45:02.123456".
    split($1, dt, "-")
    date_str = dt[1]  # e.g. "20250209"

    # Now split dt[2] ("13:45:02.123456") on the colon to get the hour.
    split(dt[2], time_parts, ":")
    hour = time_parts[1]  # e.g. "13"

    symbol = $2   # e.g. "USDCAD"
    source = $3   # e.g. "MyLiquidityProvider"

    # Increment a count in an associative array, keyed by date, hour, symbol, source.
    counts[date_str, hour, symbol, source]++
  }
  END {
    # Print header (optional)
    # print "DATE", "HOUR", "SYMBOL", "SOURCE", "COUNT"

    # Loop over all keys in counts.
    for (key in counts) {
      # The key format is "date_str SUBSEP hour SUBSEP symbol SUBSEP source".
      split(key, parts, SUBSEP)
      date_str = parts[1]
      hour     = parts[2]
      symbol   = parts[3]
      source   = parts[4]
      count    = counts[key]

      print date_str, hour, symbol, source, count
    }
  }
' "$LOGFILE" | sort
```

asdasd

```
@echo off
start wt new-tab --profile "Command Prompt" --title "Program" cmd.exe /k "program.exe" ^
; split-pane --horizontal cmd.exe /k "stuff.exe" ^
; split-pane --vertical cmd.exe /k "morestuff.exe" ^
; new-tab --profile "Git Bash" --title "UAT Servers" bash.exe -c "uat1 && connectthis && ls" ^
; split-pane --horizontal bash.exe -c "uat2 && connectthis && ls" ^
; split-pane --vertical bash.exe -c "uat3 && connectthis && ls"
```

```
@echo off
start wt new-tab --profile "Command Prompt" --title "Program" --tabColor "#0C0C52" cmd.exe /k
"program.exe" ^
; split-pane --horizontal cmd.exe /k "stuff.exe" ^
; split-pane --vertical cmd.exe /k "morestuff.exe" ^
; new-tab --profile "Git Bash" --title "UAT Servers" --tabColor "#8B4513" bash.exe -c "uat1 &&
connectthis && ls" ^
; split-pane --horizontal bash.exe -c "uat2 && connectthis && ls" ^
; split-pane --vertical bash.exe -c "uat3 && connectthis && ls"
```

sadfzxcv

```
#!/bin/bash
#
# This script loads all tables in /stuff/kdb, runs a hardcoded multi-line Q-syntax query,
# and writes the results into /home/stuff/output with a timestamped filename.

# Move to the kdb directory
cd /stuff/kdb || {
  echo "Error: Failed to cd into /stuff/kdb. Check that it exists."
  exit 1
}

# Hardcoded multi-line Q/SQL-like query
# (Adjust the actual code to your use case)
read -r -d '' MULTI_LINE_SQL <<'EQUERY'
select from trades
 where sym in `AAPL`MSFT,
       time within 09:30 16:00
EQUERY

# Generate a timestamp
TIMESTAMP=$(date +"%Y%m%d_%H%M%S")

# Hardcoded output directory (ensure this directory exists or create it beforehand)
OUTPUT_DIR="/home/stuff/output"

# Construct the output filename (e.g., output_20250310_143042.csv)
OUTPUT_CSV="${OUTPUT_DIR}/output_${TIMESTAMP}.csv"

# Run q with 1GB workspace
q -w 1000 <<EOF
  \l .
  results: $MULTI_LINE_SQL
  \`$OUTPUT_CSV 0: results
  \\
EOF

echo "Query results saved to: $OUTPUT_CSV"
```