Penerapan Fungsi Reduce dan Filter pada Pemrosesan Data Optimasi Kinerja Kepolisian

Rahma Neliyana[1)], Eka Fidiya Putri[2)], Izza Lutfia[3)], Dea Mutia Risani[4)], Dinda Nababan[5)]
Program Studi Sains Data, Fakultas Sains, Institut Teknologi Sumatera
Email : rahma.122450036@student.itera.ac.id[1)], eka.122450045@student.itera.ac.id[2)], izza.122450090@student.itera.ac.id[3)], dea.122450099@student.itera.ac.id[4)], dinda.122450120@student.itera.ac.id[5)]

1. Pendahuluan

Dalam dunia digital ini, data yang efisien sudah menjadi kebutuhan mendasar dalam segala sektor, termasuk kepolisian. Peningkatan volume data yang dapat dihasilkan dari aktivitas kepolisian sehari-hari. Salah satu pendekatan yang dapat diterapkan adalah penggunaan fungsi reduce dan filter dalam pemrosesan data.

Penelitian ini bertujuan untuk mengkaji penerapan fungsi reduce dan filter dalam pemrosesan data di lingkungan kepolisian, serta mengevaluasi dampaknya terhadap optimasi kinerja institusi tersebut. Melalui pendekatan ini, diharapkan dapat diperoleh model pemrosesan data yang tidak hanya meningkatkan efisiensi operasional tetapi juga mendukung pengambilan keputusan yang lebih berbasis data dan proaktif dalam menjaga keamanan dan ketertiban masyarakat.

2. Metode

Pada pemrosesan data optimasi kinerja kepolisian, kami menggunakan beberapa metode untuk mendapatkan program yang dituju dan benar agar programnya koefisien, adapun metode yang kami gunakan yaitu :

2.1. Filter

Filter akan memilah nilai yang ditentukan lalu dikembalikan menjadi sebuah array baru[1]. Dan filter memiliki fungsi untuk mengembalikan objek filter yang berisi elemen-elemen yang memenuhi kondisi yang ditentukan.

2.2. Reduce

Operasi *reduce* memungkinkan Anda menghitung nilai tunggal berdasarkan fungsi yang digunakan untuk menggabungkan semua daftar elemen. Cara kerja dari fungsi ini yaitu hasil operasi pada elemen ke-i, akan dijadikan parameter untuk operasi pada elemen berikutnya (i+1), sehingga hasil operasi reduce adalah akumulasi operasi pada semua elemen. Pendeknya, *value* yang tadinya banyak akan menjadi satu saja. Dalam *HOF reduce*, dapat ditambahkan parameter kedua yaitu *initial value* sebagai nilai permulaan sebelum *reduce* dieksekusi[2].

2.3. Lambda

Lambda fungsi yang didefinisikan tanpa nama. Fungsi biasa diawali dengan def, akan tetapi lambda *function* diawali dengan lambda. pada dasarnya fungsi lambda memiliki perilaku yang sama dengan fungsi biasa, kita bisa memasukkan berapapun parameter akan tetapi hanya dapat memiliki satu ekspresi atau perintah dan mengembalikan nilainya. bahkan fungsi ini ditulis dalam satu baris kode saja[3].

3. Pembahasan

3.1. Kode dan Hasil Pemrograman

Berikut merupakan kode dan hasil pemrograman yang telah kami buat. Secara keseluruhan kode ini menjelaskan tentang penerapan *Reduce* dan *Filter* pada pemrosesan data.

```
import pandas as pd

lewat_excel = '/content/911_Calls_for_Service_(Last_30_Days).csv'

data_excel = pd.read_csv(lewat_excel)

print(data_excel.head())
```

Gambar 1. Mengimport dataset

Gambar 1 menunjukan pengimportan dataset dengan menggunakan library pandas as pd. Kemudian dataset tersebut dibaca dengan menggunakan tipe data csv.

```
# Membuat list dari dictionary kosong
daftar_kamus = []

# Menambahkan dictionary kosong ke dalam list
daftar_kamus.append({})
```

Gambar 2. Membuat dictionary

Gambar 2 menunjukan pembuatan list dari dictionary kosong dengan menggunakan tanda kurung siku dan kemudian list dictionary kosong tersebut ditambahkan ke dalam list dengan perintah *append.*

```
import csv
from functools import reduce

def csv_to_dict(row):
    def parse_float(value):
        try:
            return float(value)
        except ValueError:
            return 0.0

    def parse_int(value):
        try:
            return int(value)
        except ValueError:
            return 0  # Mengembalikan 0 jika tidak bisa dikonversi menjadi integer

    return {
        'X': parse_float(row['\ufeffX']),
        'Y': parse_float(row['Y']),
        'incident_id': parse_int(row['incident_id']),
        'agency': row['agency'],
        'incident_address': row['incident_address'],
        'zip_code': row['zip_code'],
        'priority': parse_int(row['priority']),
        'callcode': row['callcode'],
        'calldescription': row['calldescription'],
        'category': row['category'],
        'traveltime': parse_float(row['traveltime']),
        'totalresponsetime': parse_float(row['totalresponsetime']),
        'time_on_scene': parse_float(row['time_on_scene']),
        'totaltime': parse_float(row['totaltime']),
        'neighborhood': row['neighborhood'],
        'block_id': parse_float(row['block_id']),
        'council_district': parse_float(row['council_district']),
        'longitude': parse_float(row['longitude']),
        'latitude': parse_float(row['latitude']),
        'ObjectId': parse_int(row['ObjectId'])
    }
```

Gambar 3. Menjalankan Perintah dengan Menggunakan Functools Reduce

Gambar 3 menunjukan penggunaan functools reduce dimana fungsi ‘csv_to_dict’ dirancang untuk memparsing sebuah kamus secara robust, mengkonversi kolom numerik menjadi float atau integer sesuai kebutuhan, dan mengembalikan nilai nil jika terjadi kesalahan konversi. Fungsi ini juga menangani kemungkinan adanya BOM di header kolom pertama. Pendekatan ini memastikan bahwa data proses dan diubah dengan bersih dan konsisten, yang sangat berguna untuk mempertegas pemrosesan atau analisis data selanjutnya.

```
# Misalnya kita membaca data dari file CSV
data = []
with open('/content/911_Calls_for_Service_(Last_30_Days).csv', newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        data.append(csv_to_dict(row))

# Filter baris yang memiliki data yang hilang di kolom Zip atau Neighborhood
filtered_data = list(filter(lambda x: x['zip_code'] != '' and x['neighborhood'] != '', data))

# Hitung total waktu respons rata-rata, waktu pengiriman rata-rata, dan total waktu rata-rata untuk kepolisian Detroit
total_time_respond = reduce(lambda acc, val: acc + val['traveltime'], filtered_data, 0)
total_time_arrival = reduce(lambda acc, val: acc + val['totalresponsetime'], filtered_data, 0)
total_response_time = reduce(lambda acc, val: acc + val['totaltime'], filtered_data, 0)

average_time_respond = total_time_respond / len(filtered_data)
average_time_arrival = total_time_arrival / len(filtered_data)
average_response_time = total_response_time / len(filtered_data)

print("Total Average Time to Respond:", average_time_respond)
print("Total Average Time to Arrival:", average_time_arrival)
print("Total Average Response Time:", average_response_time)
```

```
Total Average Time to Respond: 2.2458019938298404
Total Average Time to Arrival: 8.023254124066208
Total Average Response Time: 28.634880548686127
```

Gambar 4. Menghitung Rata-rata Respond, Arrival dan Waktu

Gambar 4 menunjukan bahwa kode tersebut membaca dan memparsing data dari file csv. Memfilter baris yang memiliki nilai kosong di dalam ‘zip_code’ atau ‘neighborhood’ dan menghitung dan mencetak waktu responds rata-rata, waktu pengiriman rata-rata, dan total waktu rata-rata untuk panggilan layanan 911 di Detroit.

```python
from functools import reduce

# Membagi list dictionary menjadi sublist dictionary berdasarkan nilai 'neighborhood'
neighborhood_data = {}
for entry in filtered_data:
    neighborhood = entry['neighborhood']
    if neighborhood not in neighborhood_data:
        neighborhood_data[neighborhood] = []
    neighborhood_data[neighborhood].append(entry)

# Menghitung total waktu respons rata-rata, waktu pengiriman rata-rata, dan total waktu rata-rata untuk setiap neighborhood
neighborhood_statistics = []
for neighborhood, data in neighborhood_data.items():
    total_time_respond = reduce(lambda acc, val: acc + val['traveltime'], data, 0)
    total_time_arrival = reduce(lambda acc, val: acc + val['totalresponsetime'], data, 0)
    total_response_time = reduce(lambda acc, val: acc + val['totaltime'], data, 0)
    average_time_respond = total_time_respond / len(data)
    average_time_arrival = total_time_arrival / len(data)
    average_response_time = total_response_time / len(data)
    neighborhood_statistics.append({
        'neighborhood': neighborhood,
        'total_average_time_respond': average_time_respond,
        'total_average_time_arrival': average_time_arrival,
        'total_average_response_time': average_response_time,
    })

# Menambahkan item dictionary untuk menyertakan data populasi untuk semua Detroit dalam daftar gabungan Anda
for stat in neighborhood_statistics:
    stat['population'] = len(neighborhood_data[stat['neighborhood']])

# Menyimpan hasil perhitungan ke dalam list dictionary
print(neighborhood_statistics)
```

Gambar 5. Menjalankan Perintah Neighborhood

Gambar 5 menjelaskan bahwa Kode ini mengelompokkan data berdasarkan ‘neighborhood’, menghitung berbagai statistik waktu rata-rata untuk setiap ‘neighborhood’, menambahkan data populasi, dan mencetak hasilnya. ‘reduce’ digunakan untuk mengakumulasi nilai-nilai waktu dalam proses penghitungan statistik.

[{'neighborhood': 'Eden Gardens', 'total_average_time_respond': 2.132478632478633, 'total_average_time_arrival': 13.271794871794876, 'total_average_response_time': 39.54829059829062, 'population': 234}

Gambar 6. Output Neighborhood

Gambar 6 menunjukan bahwa list ‘neighborhood_statistics’ yang berisi statistika rata-rata waktu respons, waktu kedatangan, dan total waktu untuk setiap ‘neighborhood’, beserta jumlah populasi.

```python
import json

# Tulis list dictionary ke dalam file JSON
output_file_path = 'output.json'
with open(output_file_path, 'w') as output_file:
    json.dump(neighborhood_statistics, output_file, indent=4)

print("File JSON berhasil dibuat:", output_file_path)
```

```
File JSON berhasil dibuat: output.json
```

```python
from google.colab import files

# Unduh file JSON
files.download('output.json')
```

Gambar 7. Mengekstrak File Menjadi JSON

Gambar 7 menjelaskan bahwa code pertama berfungsi untuk menyimpan data yang terdapat dalam list dictionary `neighborhood_statistics` ke dalam sebuah file dengan format JSON. Dengan menggunakan modul `json`, file tersebut dibuka dalam mode tulis dan data ditulis ke dalam file `output.json` dengan format yang rapi menggunakan indentasi 4 spasi. Setelah file berhasil dibuat, pesan konfirmasi dicetak ke konsol. Code kedua digunakan dalam lingkungan Google Colab untuk mengunduh file `output.json` yang telah dibuat sebelumnya. Dengan mengimpor modul `files` dari `google.colab`, fungsi `files.download` dipanggil untuk memulai proses unduhan file tersebut, memungkinkan pengguna untuk menyimpan file JSON ke perangkat lokal mereka.

4. Kesimpulan

Penelitian ini mengevaluasi penerapan fungsi reduce dan filter dalam pemrosesan data untuk optimasi kinerja kepolisian. Penerapan metode ini diharapkan dapat meningkatkan efisiensi operasional dan mendukung pengambilan keputusan yang lebih berbasis data di lingkungan kepolisian. Secara keseluruhan, penelitian ini menunjukkan bahwa penerapan fungsi reduce dan filter dapat memberikan dampak positif terhadap optimasi kinerja kepolisian dengan menyediakan metode yang efisien dan efektif untuk pemrosesan data. Hal ini mendukung keputusan yang lebih tepat dan berbasis data dalam menjaga keamanan dan ketertiban masyarakat.

Referensi

[1] A. Nurdin, "Higher Order Function di Javascript," DEV, 13 Mei 2021. [Online]. Available: https://dev.to/iamalinurdin/higher-order-function-di-javascript-4jbg. [Accessed 2024].

[2] A. E. Pamungkas, "Berkenalan dengan Javascript Higher Order Function untuk Array," Akhdani Reka Solusi, 30 Mei 2018. [Online]. Available: https://blog.akhdani.co.id/2018/05/berkenalan-dengan-javascript-higher-order-function-untuk-array/. [Accessed 2024].

[3] Revanza, M. G. (2020). Struktur Data Dan Bahasa Pemrograman.