「交通流動量 パーソントリップ発生・集中量データ」のデータベースの作り方と使い方

2023/12/24

江端智一

1. はじめに

1.1. その1

本書は、国土数値情報の「交通流動量 パーソントリップ発生・集中量データ」をSQLのデータベースにする手順を残したものです。

国土数値情報は公開データであり、誰でも無償でアクセスし使用することが可能です。研究者、政策立案者、都市計画者などが、より効果的な都市開発や交通計画を行うための貴重な情報源となっています。

しかし、一方、そのデータはファイル形式で提供されることが多く、そのフォーマット形式もバラバラです。利用者は、公開データのファイルをパースする必要が生じ、かつ、それを研究の仲間と共有するのも難

しいです。

本書の目的は、XML形式で記述された「交通流動量 パーソントリップ発生・集中量データ」を、SQLデータベースに変更するまでの手続を示して、研究室内の仲間で利用できるようにするまでのプロセスを解説することで、**ざっくりとしたデータベースの作り方**をご紹介するものです。

色々と面倒くさい手順の説明を致しますが、基本的には、**どんなデータであれ、csvファイルに変換できれば、データベース化(以下、"DB化"という)できます**。

本書を参考に、あなたの持っているデータを、研究室の学生および先生方と共有できるようにすることで、**お互いにラクができるよう**に協力しあいましょう。

1.2. その2

本データベースのハードウェアは、ラズベリーパイ4（Raspberry Pi4）です。シングルボードコンピュータ（SBC）で、教育やプロトタイピング、DIYプロジェクトなどの用途に広く使用されている小型のコンピュータです。

ラズベリーパイ4を用いた理由は、小型で持ち歩きができて、低価格で提供され、一般の人々や学校、コミュニティで手軽に利用できること、そして、広範なコミュニティに支えられており、初心者からエキスパートまで幅広いユーザーがサポートを受けながらプロジェクトを進めることができるからです。

SDカードの情報をコピーすることで、クローンを作って活用することも簡単であり、みなさんの研究活動に役立つと考えています。

2. 「交通流動量 パーソントリップ発生・集中量データ」の作成に興味のない方へ

もしDB化に興味がなく、単にDBを利用したい場合には、以下をスキップして、「「交通流動量 パーソントリップ発生・集中量データ」DBの使い方」までジャンプして下さい

本DBはすでに、江端によって作成済みです。

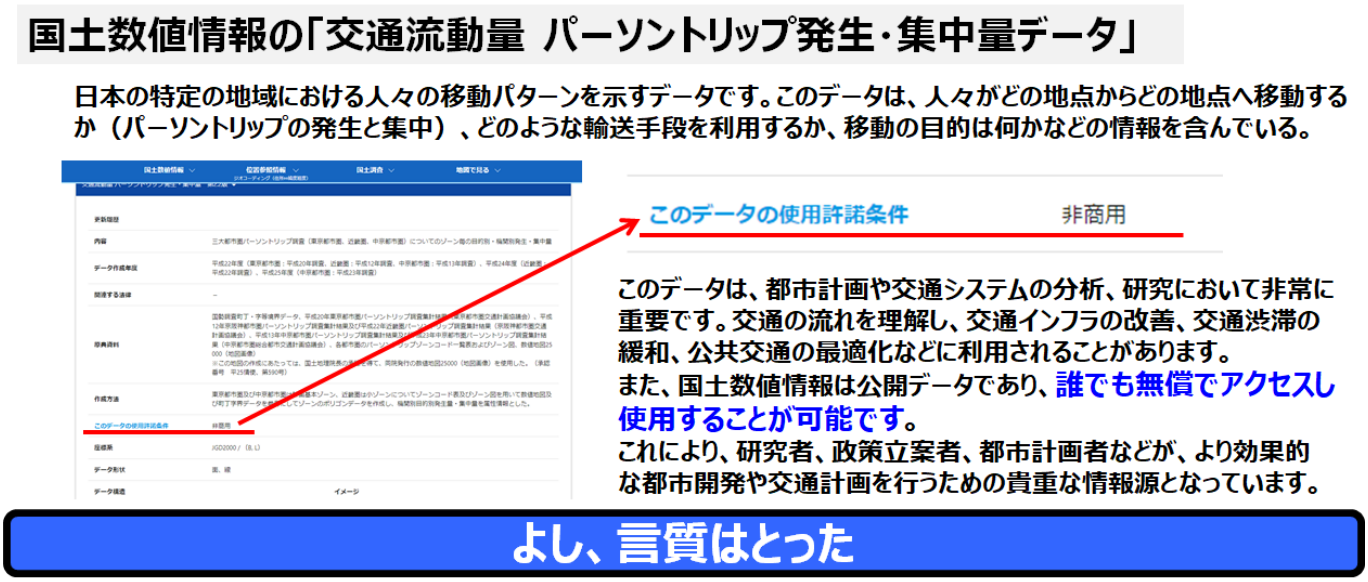
本書の「作成プロセス」は、**今後、別のデータをDB化を試みたい方への手順を示すもの**です。

3. DB化に際して注意すべき事項

データの取り扱いには十分に注意をして下さい。個人情報は、原則として個人情報保護法にもとづく取り扱いが必要になりますし、また、大学内の審査が必要な場合もあります。

使用許諾の契約の許諾者が限定されているものは、原則としてDB化してはなりません。契約違反となるからです。但し、研究室のメンバ全員に許諾されているデータであれば、この限りではありません。、

なお、本書で取り扱う、「交通流動量 パーソントリップ発生・集中量データ」は、非商用での使用が許されております。



4. 「交通流動量 パーソントリップ発生・集中量データ」の概要

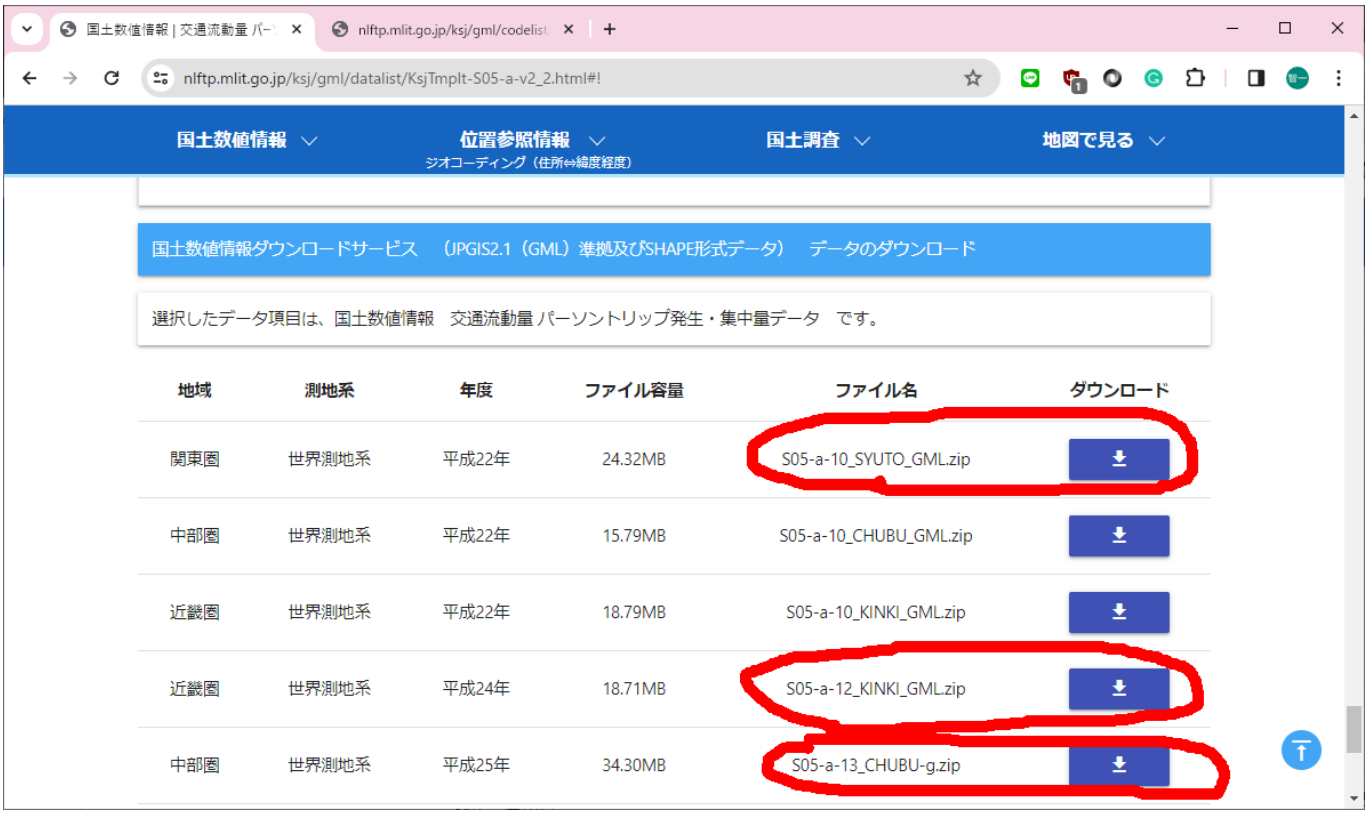
10年ごとに行われる(最近では2018年)東京圏パーソソントリップの調査データの解析結果です。

日本の特定の地域における人々の移動パターンを示すデータです。このデータは、人々がどの地点からどの地点へ移動するか（パーソントリップの発生と集中）、どのような輸送手段を利用するか、移動の目的は何かなどの情報を含んでいます。

データはこちらにあります

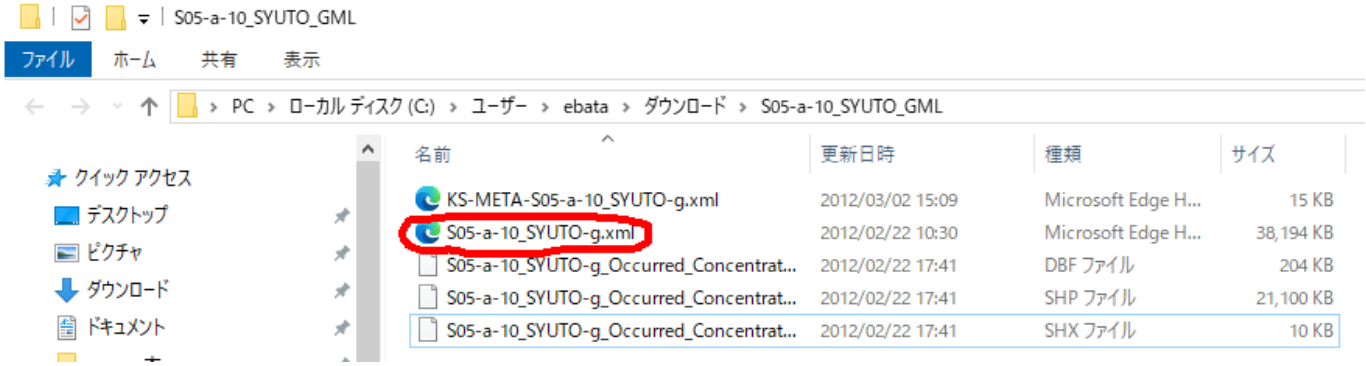
https://nlftp.mlit.go.jp/ksj/gml/datalist/KsjTmplt-S05-a-v2_2.html#!

このページのここからダウンロードして下さい。



ダウンロードの前に、アンケートがあります(スキップもできます)。

上記3つのファイルをダウンロードして、別個のディレクトリで解凍して下さい。



以下、S05-a-10_SYUTO_GMLのディレクトリの中に入っている"**S05-a-10_SYUTO-g.xml**"を例題として取り扱います。

4.1. 本データのローカルホスト(PC,ラズパイ等)へのインストール

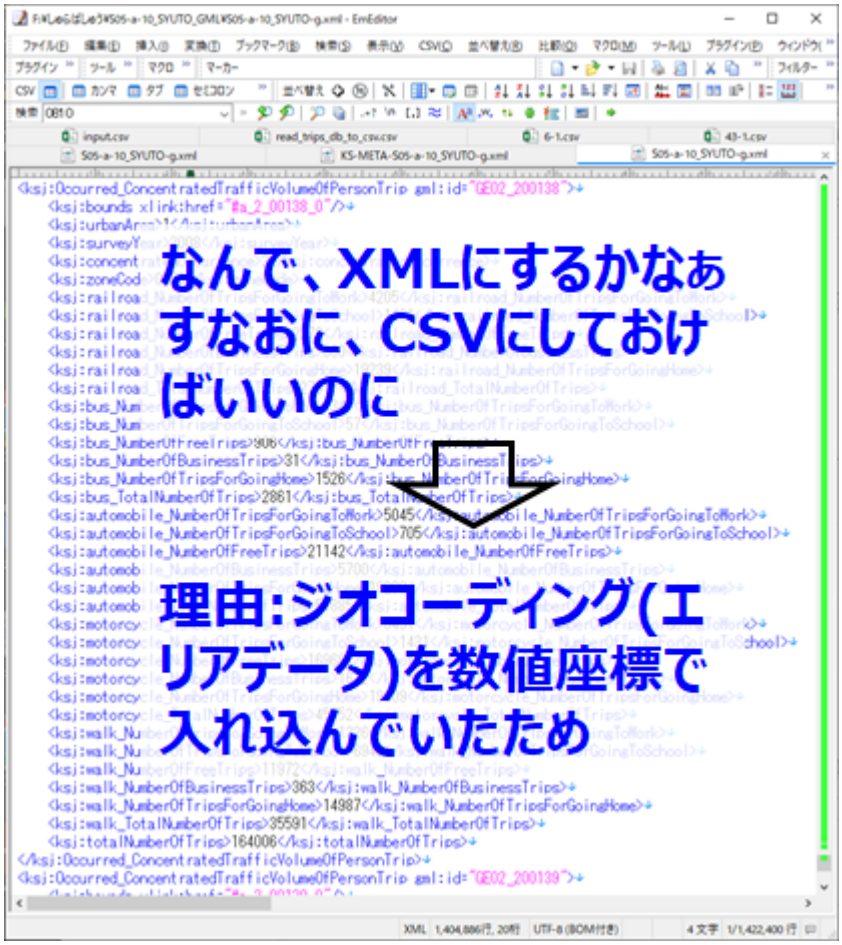
ダウンロードして解凍したディレクトリは、以下のように配置していて下さい。

```
person_trip1/
├── S05-a-10_SYUTO_GML
├── S05-a-12_KINKI_GML
├── S05-a-13_CHUBU-g
└── docker-compose.yml(後で説明します)
```

4.2. 「交通流動量 パーソントリップ発生・集中量データ」の問題点

4.2.1. XML形式での提供

提供されているデータは、XMLで記載されており、簡単にcsvファイルに変換できません(理由は以下の図中に記載)。



4.2.2. 中ゾーンでのデータ提供

意図的はどうかは不明ですが、小ゾーンでなく、それを纏めた中ゾーンで提供されており、粒度は小ゾーンよりは荒いです。

5. 「交通流動量 パーソントリップ発生・集中量データ」のDBの作成方法

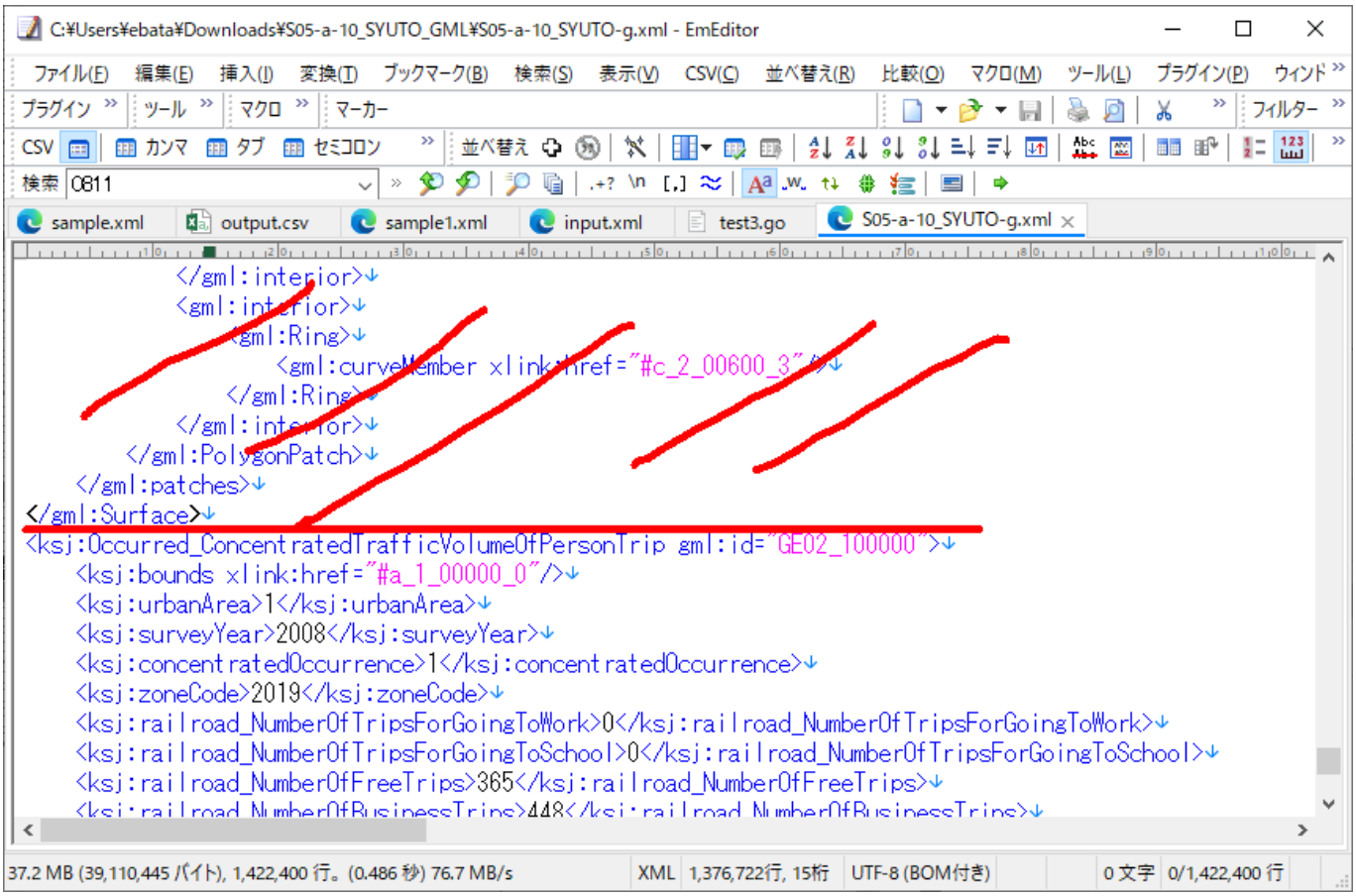
5.1. xmlファイルからcsvファイルへの変換

CSVファイルを作成できれば、postgresqlにインポートすることは、一般的に行われていることで、簡単にできます。

"**S05-a-10_SYUTO-g.xml**"は、ジオデータやら、名前空間やらが入っていますので、このままパースをすると結構な頻度で失敗します(私は失敗しました)。

このファイルを、"**sample1.xml**"という名前でコピペします。

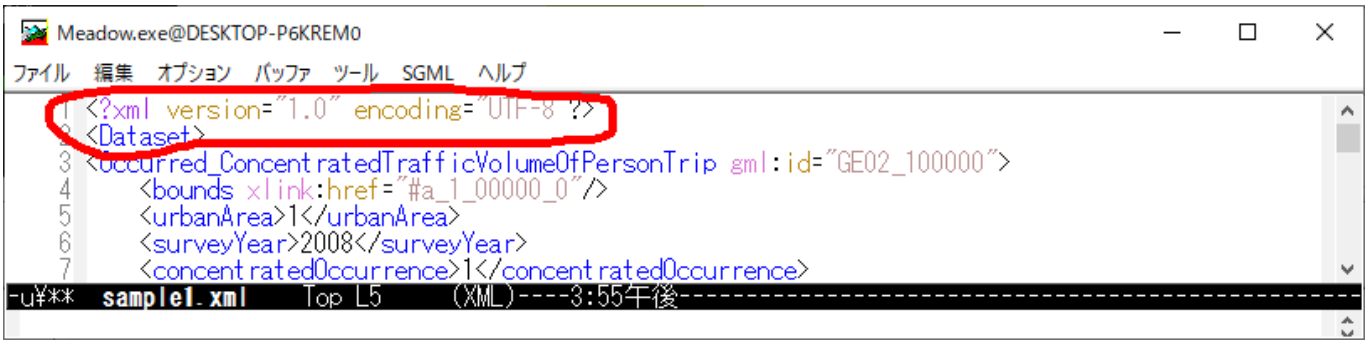
まず、前半をバッサリ削除して下さい(私の見ているファイルでは、行番号は、1,376,722行あたりです)



次に、残ったファイルの中から、"ksj:"を全削除して下さい(名前空間です)。

このファイルの先頭に、

```
<?xml version="1.0" encoding="UTF-8"?>
<Dataset>
```

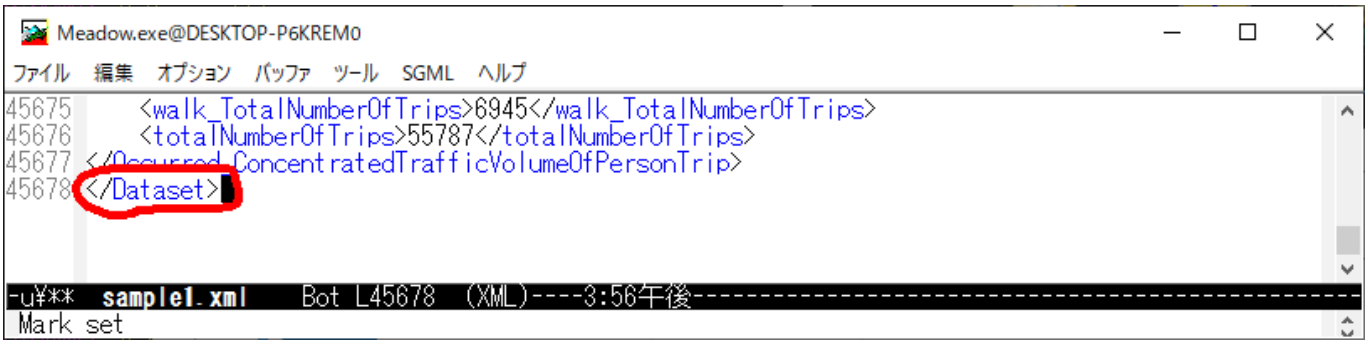


を加えて下さい。

そして、最終行に、

```
</Dataset>
```

が付いていることも確認して下さい。



これで、"**sample1.xml**"の完成です。

本書では、(あまりメジャーとは言えない)GO言語の使用を前提とします。XMLファイルからcsvファイルの変更は、Pythonなどでも可能ですし、多分そういうサービスもあると思いますので、それを利用して頂いても構いません。

以下の"s05_xml2csv.go"をコピペするか、こちら[https://wp.kobore.net/%e6%b1%9f%e7%ab%af%e3%81%95%e3%82%93%e3%81%ae%e6%8a%80%e8%a1%93%e3%83%a1%e3%83%a2/post-12919/]からダウンロードして下さい("sample1.xml"もダウンロードできます)。

```
package main

import (
    "encoding/csv"
    "encoding/xml"
    "fmt"
    "os"
)

// XMLデータの構造体定義
type Dataset struct {
    Items []OccurredConcentratedTrafficVolumeOfPersonTrip
`xml:"Occurred_ConcentratedTrafficVolumeOfPersonTrip"`
}

type OccurredConcentratedTrafficVolumeOfPersonTrip struct {
    ID                                  string `xml:"id,attr"`
    UrbanArea                           int    `xml:"urbanArea"`
    SurveyYear                          int    `xml:"surveyYear"`
    ConcentratedOccurrence              int    `xml:"concentratedOccurrence"`
    ZoneCode                            int    `xml:"zoneCode"`
    Railroad_NumberOfTripsForGoingToWork    int
`xml:"railroad_NumberOfTripsForGoingToWork"`
    Railroad_NumberOfTripsForGoingToSchool  int
`xml:"railroad_NumberOfTripsForGoingToSchool"`
    Railroad_NumberOfFreeTrips              int
`xml:"railroad_NumberOfFreeTrips"`
    Railroad_NumberOfBusinessTrips          int
`xml:"railroad_NumberOfBusinessTrips"`
    Railroad_NumberOfTripsForGoingHome      int
`xml:"railroad_NumberOfTripsForGoingHome"`
```

```
    Railroad_TotalNumberOfTrips              int
`xml:"railroad_TotalNumberOfTrips"`
    Bus_NumberOfTripsForGoingToWork          int
`xml:"bus_NumberOfTripsForGoingToWork"`
    Bus_NumberOfTripsForGoingToSchool        int
`xml:"bus_NumberOfTripsForGoingToSchool"`
    Bus_NumberOfFreeTrips                    int    `xml:"bus_NumberOfFreeTrips"`
    Bus_NumberOfBusinessTrips                int
`xml:"bus_NumberOfBusinessTrips"`
    Bus_NumberOfTripsForGoingHome            int
`xml:"bus_NumberOfTripsForGoingHome"`
    Bus_TotalNumberOfTrips                   int    `xml:"bus_TotalNumberOfTrips"`
    Automobile_NumberOfTripsForGoingToWork   int
`xml:"automobile_NumberOfTripsForGoingToWork"`
    Automobile_NumberOfTripsForGoingToSchool int
`xml:"automobile_NumberOfTripsForGoingToSchool"`
    Automobile_NumberOfFreeTrips             int
`xml:"automobile_NumberOfFreeTrips"`
    Automobile_NumberOfBusinessTrips         int
`xml:"automobile_NumberOfBusinessTrips"`
    Automobile_NumberOfTripsForGoingHome     int
`xml:"automobile_NumberOfTripsForGoingHome"`
    Automobile_TotalNumberOfTrips            int
`xml:"automobile_TotalNumberOfTrips"`
    Motorcycle_NumberOfTripsForGoingToWork   int
`xml:"motorcycle_NumberOfTripsForGoingToWork"`
    Motorcycle_NumberOfTripsForGoingToSchool int
`xml:"motorcycle_NumberOfTripsForGoingToSchool"`
    Motorcycle_NumberOfFreeTrips             int
`xml:"motorcycle_NumberOfFreeTrips"`
    Motorcycle_NumberOfBusinessTrips         int
`xml:"motorcycle_NumberOfBusinessTrips"`
    Motorcycle_NumberOfTripsForGoingHome     int
`xml:"motorcycle_NumberOfTripsForGoingHome"`
    Motorcycle_TotalNumberOfTrips            int
`xml:"motorcycle_TotalNumberOfTrips"`
    Walk_NumberOfTripsForGoingToWork         int
`xml:"walk_NumberOfTripsForGoingToWork"`
    Walk_NumberOfTripsForGoingToSchool       int
`xml:"walk_NumberOfTripsForGoingToSchool"`
    Walk_NumberOfFreeTrips                   int    `xml:"walk_NumberOfFreeTrips"`
    Walk_NumberOfBusinessTrips               int
`xml:"walk_NumberOfBusinessTrips"`
    Walk_NumberOfTripsForGoingHome           int
`xml:"walk_NumberOfTripsForGoingHome"`
    Walk_TotalNumberOfTrips                  int
`xml:"walk_TotalNumberOfTrips"`
    TotalNumberOfTrips                       int    `xml:"totalNumberOfTrips"`
}

func main() {
    // XMLファイルを読み込む
    xmlData, err := os.Open("sample1.xml")
    if err != nil {
```

```
        fmt.Println("Error opening XML file:", err)
        return
    }
    defer xmlData.Close()

    // XMLデータをデコード
    var dataset Dataset
    decoder := xml.NewDecoder(xmlData)
    if err := decoder.Decode(&dataset); err != nil {
        fmt.Println("Error decoding XML:", err)
        return
    }

    // CSVファイルにデータを書き込み
    csvFile, err := os.Create("output.csv")
    if err != nil {
        fmt.Println("Error creating CSV file:", err)
        return
    }
    defer csvFile.Close()

    csvWriter := csv.NewWriter(csvFile)
    defer csvWriter.Flush()

    // CSVヘッダを書き込み
    headers := []string{
        "ID", "UrbanArea", "SurveyYear", "ConcentratedOccurrence", "ZoneCode",
        "Railroad_NumberOfTripsForGoingToWork",
"Railroad_NumberOfTripsForGoingToSchool", "Railroad_NumberOfFreeTrips",
        "Railroad_NumberOfBusinessTrips", "Railroad_NumberOfTripsForGoingHome",
"Railroad_TotalNumberOfTrips",
        "Bus_NumberOfTripsForGoingToWork", "Bus_NumberOfTripsForGoingToSchool",
"Bus_NumberOfFreeTrips",
        "Bus_NumberOfBusinessTrips", "Bus_NumberOfTripsForGoingHome",
"Bus_TotalNumberOfTrips",
        "Automobile_NumberOfTripsForGoingToWork",
"Automobile_NumberOfTripsForGoingToSchool", "Automobile_NumberOfFreeTrips",
        "Automobile_NumberOfBusinessTrips",
"Automobile_NumberOfTripsForGoingHome", "Automobile_TotalNumberOfTrips",
        "Motorcycle_NumberOfTripsForGoingToWork",
"Motorcycle_NumberOfTripsForGoingToSchool", "Motorcycle_NumberOfFreeTrips",
        "Motorcycle_NumberOfBusinessTrips",
"Motorcycle_NumberOfTripsForGoingHome", "Motorcycle_TotalNumberOfTrips",
        "Walk_NumberOfTripsForGoingToWork", "Walk_NumberOfTripsForGoingToSchool",
"Walk_NumberOfFreeTrips",
        "Walk_NumberOfBusinessTrips", "Walk_NumberOfTripsForGoingHome",
"Walk_TotalNumberOfTrips",
        "TotalNumberOfTrips",
    }

    if err := csvWriter.Write(headers); err != nil {
        fmt.Println("Error writing CSV headers:", err)
        return
    }
```

```
    // データをCSVに書き込み
    //for _, person := range root.Persons {
    for _, item := range dataset.Items {

        // fmt.Println("pass1")
        // fmt.Println(item)

        //record := []string{item.ID, fmt.Sprintf("%d", item.UrbanArea),
fmt.Sprintf("%d", item.SurveyYear), fmt.Sprintf("%d",
item.ConcentratedOccurrence), fmt.Sprintf("%d", item.ZoneCode)}

        record := []string{
            item.ID,
            fmt.Sprintf("%d", item.UrbanArea),
            fmt.Sprintf("%d", item.SurveyYear),
            fmt.Sprintf("%d", item.ConcentratedOccurrence),
            fmt.Sprintf("%d", item.ZoneCode),
            fmt.Sprintf("%d", item.Railroad_NumberOfTripsForGoingToWork),
            fmt.Sprintf("%d", item.Railroad_NumberOfTripsForGoingToSchool),
            fmt.Sprintf("%d", item.Railroad_NumberOfFreeTrips),
            fmt.Sprintf("%d", item.Railroad_NumberOfBusinessTrips),
            fmt.Sprintf("%d", item.Railroad_NumberOfTripsForGoingHome),
            fmt.Sprintf("%d", item.Railroad_TotalNumberOfTrips),
            fmt.Sprintf("%d", item.Bus_NumberOfTripsForGoingToWork),
            fmt.Sprintf("%d", item.Bus_NumberOfTripsForGoingToSchool),
            fmt.Sprintf("%d", item.Bus_NumberOfFreeTrips),
            fmt.Sprintf("%d", item.Bus_NumberOfBusinessTrips),
            fmt.Sprintf("%d", item.Bus_NumberOfTripsForGoingHome),
            fmt.Sprintf("%d", item.Bus_TotalNumberOfTrips),
            fmt.Sprintf("%d", item.Automobile_NumberOfTripsForGoingToWork),
            fmt.Sprintf("%d", item.Automobile_NumberOfTripsForGoingToSchool),
            fmt.Sprintf("%d", item.Automobile_NumberOfFreeTrips),
            fmt.Sprintf("%d", item.Automobile_NumberOfBusinessTrips),
            fmt.Sprintf("%d", item.Automobile_NumberOfTripsForGoingHome),
            fmt.Sprintf("%d", item.Automobile_TotalNumberOfTrips),
            fmt.Sprintf("%d", item.Motorcycle_NumberOfTripsForGoingToWork),
            fmt.Sprintf("%d", item.Motorcycle_NumberOfTripsForGoingToSchool),
            fmt.Sprintf("%d", item.Motorcycle_NumberOfFreeTrips),
            fmt.Sprintf("%d", item.Motorcycle_NumberOfBusinessTrips),
            fmt.Sprintf("%d", item.Motorcycle_NumberOfTripsForGoingHome),
            fmt.Sprintf("%d", item.Motorcycle_TotalNumberOfTrips),
            fmt.Sprintf("%d", item.Walk_NumberOfTripsForGoingToWork),
            fmt.Sprintf("%d", item.Walk_NumberOfTripsForGoingToSchool),
            fmt.Sprintf("%d", item.Walk_NumberOfFreeTrips),
            fmt.Sprintf("%d", item.Walk_NumberOfBusinessTrips),
            fmt.Sprintf("%d", item.Walk_NumberOfTripsForGoingHome),
            fmt.Sprintf("%d", item.Walk_TotalNumberOfTrips),
            fmt.Sprintf("%d", item.TotalNumberOfTrips),
        }

        if err := csvWriter.Write(record); err != nil {
            fmt.Println("Error writing CSV record:", err)
            return
```

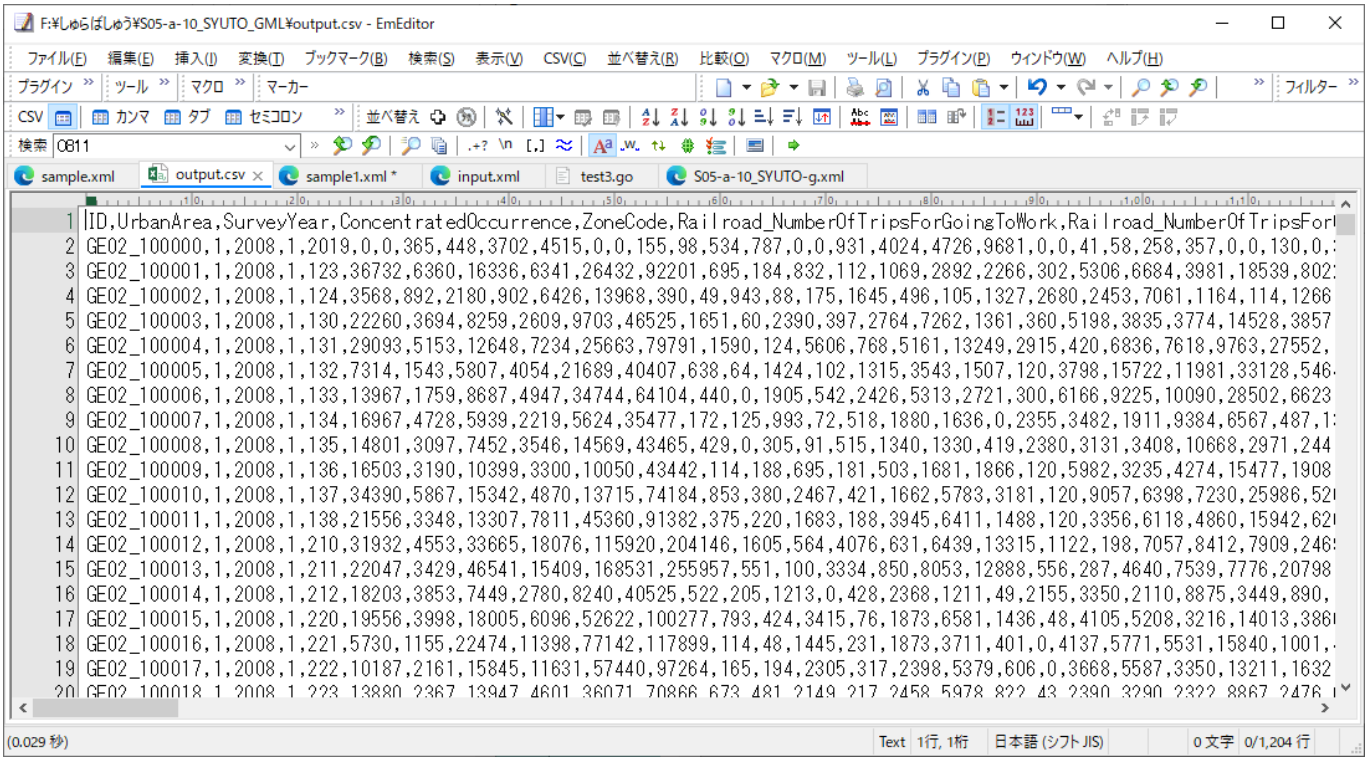
```
        }

    }

    fmt.Println("CSV file successfully created.")
}
```

その後、

```
go run s05_xml2csv.go
```

を実行することで、"**output.csv**"が生成されます。

これと同じ処理を、**S05-a-12_KINKI_GML**、**S05-a-13_CHUBU-g**のディレクトリでも実施して下さい。



これで、csvファイルの完成です。

6. システム構築方法

本システムでは、ラズパイ4に「Raspberry Pi OS」をインストールしていますが、別にこのディストリビューションに限定する必要はありません。Dockerが使えれば問題ありません(Dockerが使えないLinuxのディストリビューションなどはありません)。

本書では、ディストリビューションのインストール方法については割愛します。

6.1. Dockerのインストール方法

本書では、「Raspberry Pi OS」を前提に以下を説明しあす。

ターミナルを開き、以下のコマンドを実行してDockerの公式スクリプトをダウンロードします。

```
curl -fsSL https://get.docker.com -o get-docker.sh
```

スクリプトを実行可能に設定します。

```
chmod +x get-docker.sh
```

スクリプトを実行してDockerをインストールします。

```
sudo ./get-docker.sh
```

インストールが完了したら、piユーザーをDockerグループに追加して、Dockerをsudoなしで実行できるようにします。

```
sudo usermod -aG docker pi
```

注意: ユーザー名が"pi"でない場合は、該当するユーザー名に置き換えてください(本システムでは、"tuel"としました)。ここで一度、ラズパイ4を再起動すると良いようです。

Dockerが正常にインストールされたかどうかを確認します。

```
docker --version
```

これで、Raspberry Pi OSにDockerが簡単にインストールされ、piユーザーがDockerコマンドを実行できるようになります。

6.2. Docker-composeのインストール方法

Docker Composeのバイナリをダウンロードします。公式のGitHubリリースページから最新バージョンを確認し、ダウンロードします。以下のコマンドで最新バージョンをダウンロードできますが、バージョン番号を確認して最新のものを使用してください。

```
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

ダウンロードしたバイナリに実行権限を付与します。

```
sudo chmod +x /usr/local/bin/docker-compose
```

Docker Composeのバージョンを確認してインストールが成功したことを確認します。

```
docker-compose --version
```

6.3. ファイルのコピー

ローカルで作成したファイルを格納したディレクトリを、ラズパイ4にコピーして下さい。

```
person_trip1/
├── S05-a-10_SYUTO_GML
├── S05-a-12_KINKI_GML
├── S05-a-13_CHUBU-g
└── docker-compose.yml(以下で説明します)
```

で作成した、ファイル群をラズパイにコピーして下さい。

私の場合、USBメモリを使って、

```
/home/tuel
```

にコピーしました。

6.4. docker-composeによるPostgresqlDBの作成

person_trip1に、pt_dbのディレクトリを作成して下さい。

また、person_trip1に、以下のdocker-compose.ymlを作成して下さい。

```
version: '3'

services:
  db:
    image: postgres:14
    container_name: postgres

    ports:
      - 15432:5432
    volumes:
      - ./pt_db:/pt_db
      - db-data:/var/lib/postgresql/data
    environment:
      - POSTGRES_PASSWORD=4039ynu
```

```
    restart: always

volumes:
  db-data:

# pt_dbというディレクトリを掘っておくこと
# docker exec -it postgres bash // コンテナへのログイン方法
```

その後、以下のコマンドでDockerコンテナを作成して下さい。

```
docker-compose up -d
```

その後、画面に色々な表示がされますが、上手くいけばDockerコンテナが作成されます。 その後、"restart: always"を有効にするために、以下のコマンドも投入しておいて下さい。

```
docker compose build
```

その後、PostgresqlのDockerコンテナの起動は

```
docker ps
```

で確認できます。

```
CONTAINER ID   IMAGE         COMMAND                  CREATED       STATUS
PORTS                                       NAMES
b2e7b75cb7f8   postgres:14   "docker-entrypoint.s…"   5 hours ago   Up 5 hours
0.0.0.0:15432->5432/tcp, :::15432->5432/tcp   postgres
```

6.5. psqlのクライアントのインストール

postgresqlを操作する為に、psqlのクライアントをインストールしておきます。

パッケージリストを更新します。

```
sudo apt update
```

psql クライアントをインストールします。

```
sudo apt install postgresql-client
```

インストールが完了したら、psql コマンドを使用してPostgreSQLデータベースに接続できます。接続するには以下のコマンドを使用します。

```
psql -U postgres -h localhost -p 15432
```

パスワードは、"**4039gun**"でログインできます。

6.6. DBとテーブルの作成

6.6.1. DBの作成

```
psql -U postgres -h localhost -p 15432
```

パスワードは、"**4039gun**"でログインした後、以下のコマンドでデータベースを作成して下さい。

```
postgres=# create database s05;
```

6.6.2. テーブルの作成

データベースの作成に成功したら、次のデータベース(**s05**)にログインして下さい。

```
\c s05
```

以下スクリプトで、"**a_10_syuto_gml**"のテーブルを作成して下さい(コピペしてリターンキーを押下するだけで作成されます)。

```
CREATE TABLE a_10_syuto_gml (
    ID VARCHAR(255),
    UrbanArea INT,
    SurveyYear INT,
    ConcentratedOccurrence INT,
    ZoneCode INT,
    Railroad_NumberOfTripsForGoingToWork INT,
    Railroad_NumberOfTripsForGoingToSchool INT,
    Railroad_NumberOfFreeTrips INT,
    Railroad_NumberOfBusinessTrips INT,
    Railroad_NumberOfTripsForGoingHome INT,
    Railroad_TotalNumberOfTrips INT,
    Bus_NumberOfTripsForGoingToWork INT,
    Bus_NumberOfTripsForGoingToSchool INT,
    Bus_NumberOfFreeTrips INT,
    Bus_NumberOfBusinessTrips INT,
```

```
    Bus_NumberOfTripsForGoingHome INT,
    Bus_TotalNumberOfTrips INT,
    Automobile_NumberOfTripsForGoingToWork INT,
    Automobile_NumberOfTripsForGoingToSchool INT,
    Automobile_NumberOfFreeTrips INT,
    Automobile_NumberOfBusinessTrips INT,
    Automobile_NumberOfTripsForGoingHome INT,
    Automobile_TotalNumberOfTrips INT,
    Motorcycle_NumberOfTripsForGoingToWork INT,
    Motorcycle_NumberOfTripsForGoingToSchool INT,
    Motorcycle_NumberOfFreeTrips INT,
    Motorcycle_NumberOfBusinessTrips INT,
    Motorcycle_NumberOfTripsForGoingHome INT,
    Motorcycle_TotalNumberOfTrips INT,
    Walk_NumberOfTripsForGoingToWork INT,
    Walk_NumberOfTripsForGoingToSchool INT,
    Walk_NumberOfFreeTrips INT,
    Walk_NumberOfBusinessTrips INT,
    Walk_NumberOfTripsForGoingHome INT,
    Walk_TotalNumberOfTrips INT,
    TotalNumberOfTrips INT
);
```

次に、S05-a-10_SYUTO_GMLで作成したoutput.csvをデータベースに、次のコマンドでインポートします。

```
\copy a_10_syuto_gml from /home/tuel/person_trip1/S05-a-10_SYUTO_GML/output.csv
delimiter ',' csv header;
```

これで、output.csvをデータベースに格納できます。

S05-a-12_KINKI_GML, S05-a-13_CHUBU-g についても同様の処理を行います。

作成されたDBのテーブルは以下のようになります。

```
s05=# \dt
                リレーション一覧
 スキーマ |      名前      |  タイプ  |  所有者
----------+----------------+----------+----------
 public   | a_10_syuto_gml | テーブル | postgres
 public   | a_12_kinki_gml | テーブル | postgres
 public   | a_13_chubu_gml | テーブル | postgres
(3 行)
```

以上で、DB化が完成します。

7. 「交通流動量 パーソントリップ発生・集中量データ」DBの使い方

7.1. アクセス方法

ラズベリーパイ4（Raspberry Pi 4）(以下、ラズパイ4という)のログインは以下の通りです(必要ない場合もあります)。

```
ラズパイ4の起動方法
■ユーザ名:"tuel"
■パスワード:"4039ynu"
で起動します。
```

以下のコマンドで(あるいは他のSQLクライアントから)、データベースにアクセスして下さい。パスワードは"**4039ynu**"です。

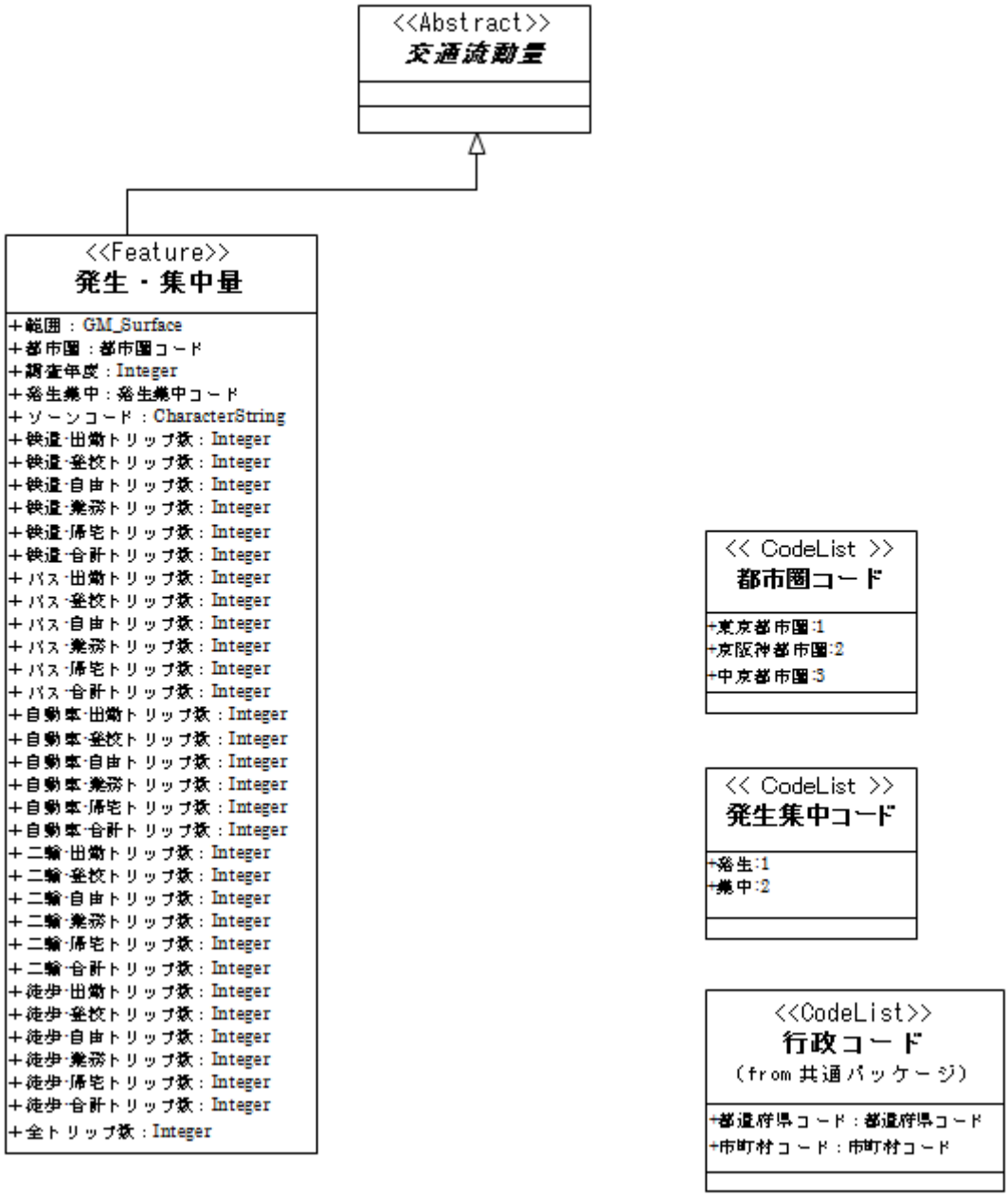
```
$ psql -U postgres -h localhost -p 15432
Password for user postgres:
```

リモートからアクセスする場合は以下のIPアドレスを使って下さい。

```
$ psql -U postgres -h 133.34.90.40 -p 15432
Password for user postgres:
```

7.2. データスキーマーの構成

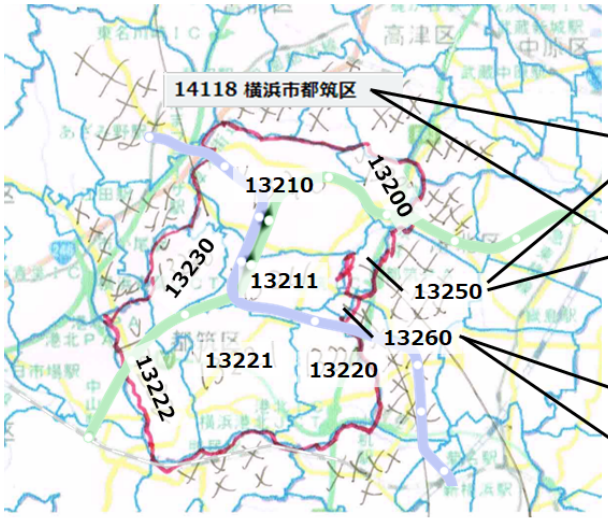
データスキーマは以下の通りです。



こちらのページにスキーマの詳しい記載があります。https://nlftp.mlit.go.jp/ksj/gml/datalist/KsjTmplt-S05-a-v2_2.html#

8. SQLクエリのサンプル ーー 後程差し替え

作成したデータベースの代表エントリー



■PT2018 (ゾーンコードは上5桁のみで処理が必要)

出発地：区分	Dep_point_class
出発地：完全桁数	Dep_loc_Comp_digits
出発地：ゾーンコード	Dep_Loc_Zone
出発地：JISコード（5桁）	Dep_point_code
到着地：区分	Arr_place_class
到着地：完全桁数	Arr_Place_Comp_Digits
到着地：ゾーンコード	Arr_Place_Zone_Code
到着地：JISコード（5桁）	Arr_Place_JIS_Code

■PT2008 (都筑区のJISコード(5桁)がない)

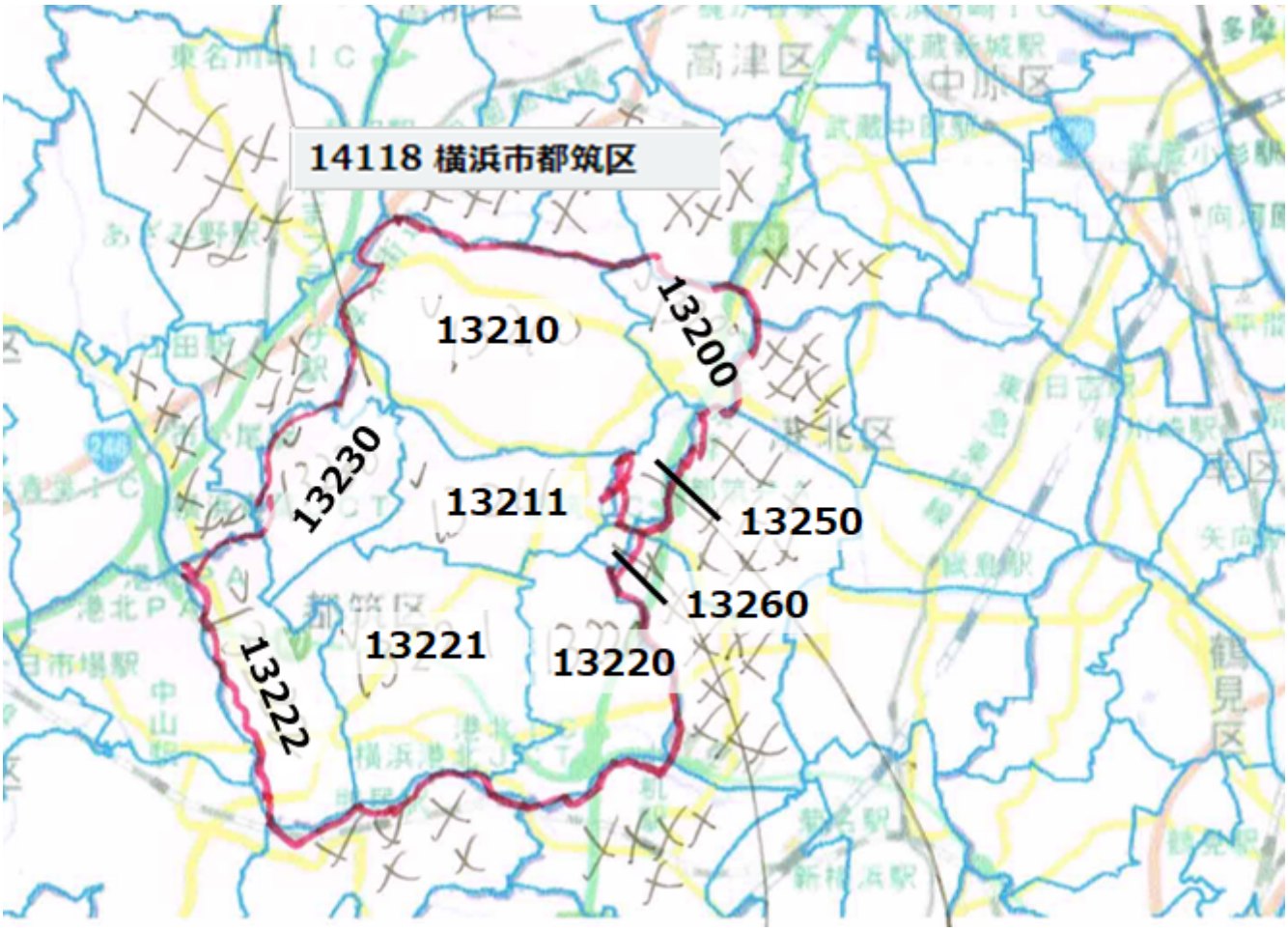
出発地	出発地区分	Dep_point_class
	完全桁数	Dep_loc_Comp_digits
	小ゾーン	Dep_Loc_Zone
	施設の種類	Facility_Type
到着地	到着地区分	Arr_place_class
	完全桁数	Arr_Place_Comp_Digits
	小ゾーン	Arr_Place_Zone_Code
	施設の種類	Type_of_facility

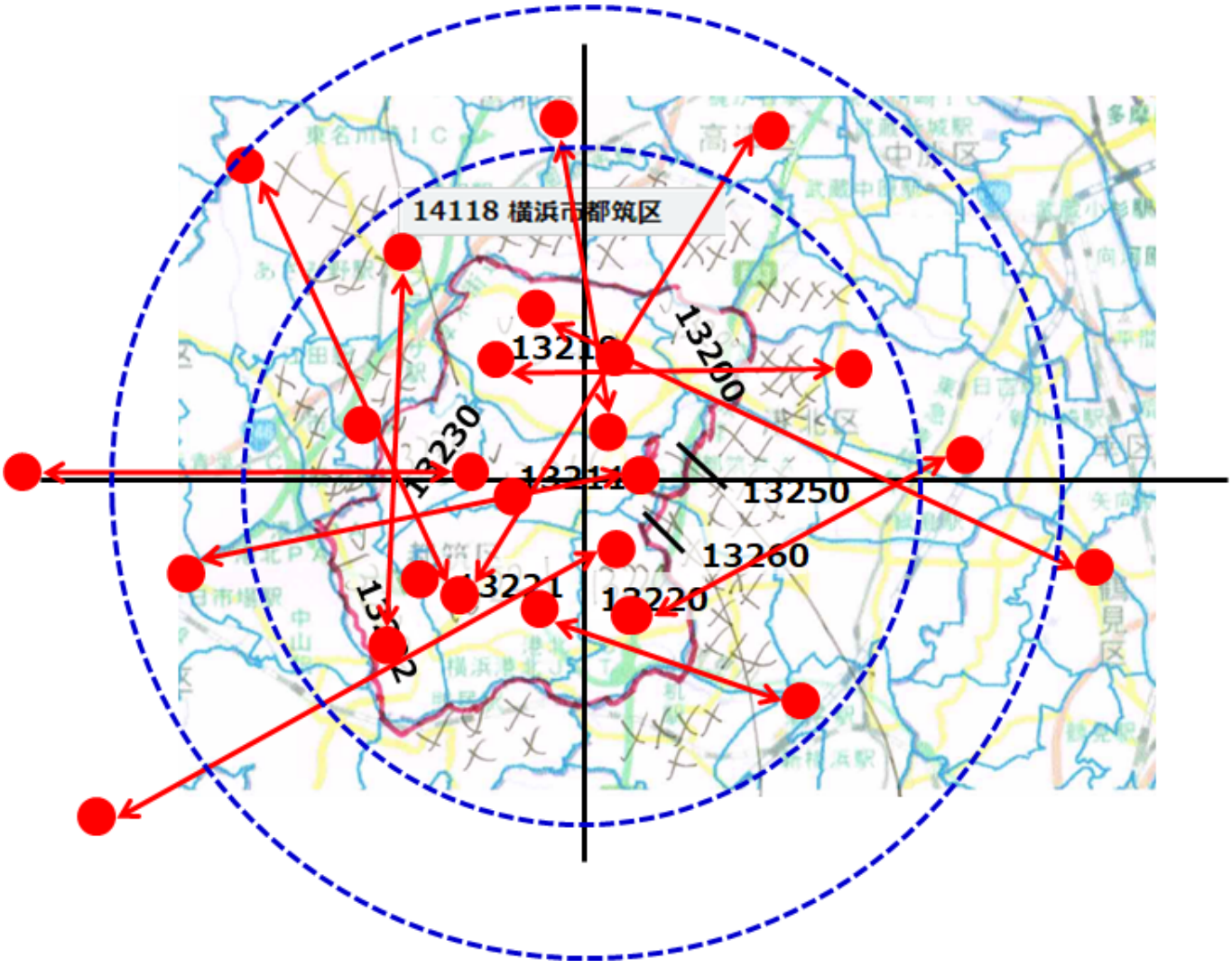
出発:pt2018=# select count(*) from ms2611 where left(dep_loc_zone::TEXT,5) = '13200' and Dep_point_code = 14118;

到着: pt2018=# select count(*) from ms2611 where left(Arr_Place_Zone_Code::TEXT,5) = '13220' and Arr_Place_JIS_Code = 14118;

■出発は都筑区(のみ)pt2018=# select count(*) from ms2611 where Dep_point_code = 14118;count-------4399(1 row)

■出発は都筑区で、マストラ降車が、都筑区外select count(*) from ms2611 where Dep_point_code = 14118 and
left(mastra_drop_zone::TEXT,5) != '13210' and
left(mastra_drop_zone::TEXT,5) != '13200' and
left(mastra_drop_zone::TEXT,5) != '13240' and
left(mastra_drop_zone::TEXT,5) != '13230' and
left(mastra_drop_zone::TEXT,5) != '13211' and
left(mastra_drop_zone::TEXT,5) != '13250' and
left(mastra_drop_zone::TEXT,5) != '13260' and
left(mastra_drop_zone::TEXT,5) != '13222' and
left(mastra_drop_zone::TEXT,5) != '13221' and
left(mastra_drop_zone::TEXT,5) != '13220';count-------1106(1 row)


14118 横浜市都筑区
13210
13200
13230
13211
13250
13260
13221
13220
13222


14118 横浜市都筑区
13200
13230
13250
13260
13220

東京圏パーソントリップデータデータベースの使い方

- 発行日 2023年12月24日
- 発行者 江端 智一