

公共交通データとの 向き合い方

渡邊 徹志 @tesshy

本日の資料



GitHub: https://github.com/tesshy/2024-09-19_odpt



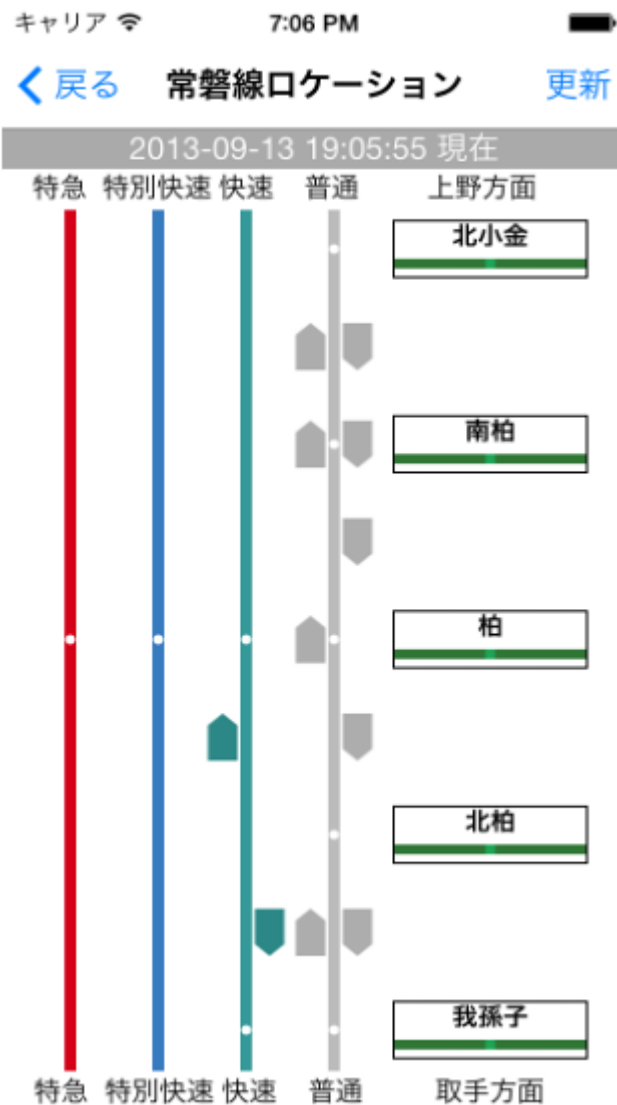
HTML: https://www.tesshy.com/2024-09-19_odpt/ViewGTFS.html

About tesshy

渡邊徹志（1984年生まれ、栃木県出身）

2009-04 - 2015-03: 東京大学大学院学際情報学府学際情報学専攻総合分析情報学コース博士後期課程坂村・越塚研究室（単位取得満期退学）
2012-04 - 2018-07: YRPユビキタス・ネットワーキング研究所研究員
2018-07 - 2019-02: 無職
2019-02 - 2024-01: 株式会社MaaS Tech Japan CTO
2024-02 - 2024-03: 無職
2024-04 - 2024-03: 株式会社team-7、複数社お手伝い、農業（主にデジタル化、運転手として）

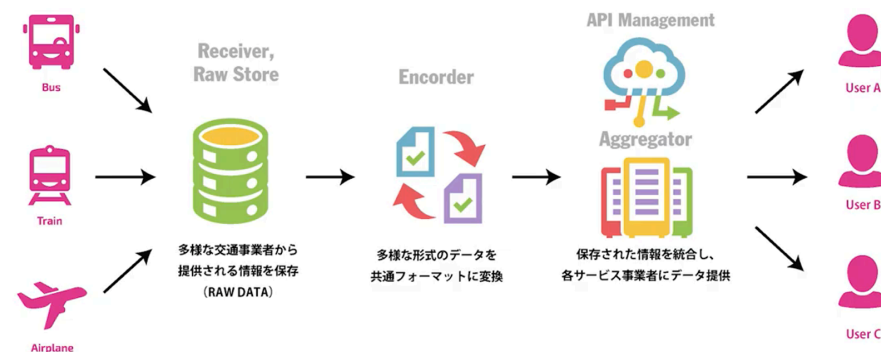
- 学士、修士は共に無線通信によるロケーションシステムの研究、以降は画像認識、AR、救急医療システム、オープンデータ、公共交通データ、etc
 - odptの最初期の仕様策定と実装は自分が担当してました、もう10年ぐらい前・・・
- データ基盤設計、データ分析、プロマネ、コンサルティング
- 最近は農作業を手伝い、バスの中で仕事してる（大型・けん引免許所持）



柏市公共交通情報連携アプリ(2013年)



東京メトロオープンデータコンテスト(2014年)



TraISARE(2022年)



愛車と共に



農業

目的

- 公共交通データをどうやって扱うか
 - 可視化を例に
- データ処理の基本を知る: ETLの初歩
 - 公共交通に限らず、幅広いデータを扱えるように
- どんなツールがあるのか・使えるのか
 - 渡邊が使っているツール類を紹介

コンテスト応募作品のアイデアとして役に立てば幸い

公共交通データを使いたい！

- 乗換案内アプリを作りたい
- バスの遅延情報を提供したい
- 公共交通を用いた到達圏の分析をしたい



よくある話

「何を実現したい」から「何が必要か」を考えましょう。手段が目的とならないように・・・。

GTFSを可視化してみる

- データを扱う上で必要なことは、まず**可視化**
 - どんなデータなのか
 - 何に使えそうか
 - どうやって処理（加工）するか
 - ~~データが間違っていないか~~
- データの民主化（Data Democratization）への第一歩
 - 従来一部の専門家だけが扱っていたものを、広く使えるように
 - 公開されるだけではなく、如何に使われるか
- 今回は**GTFS**を可視化してみる
 - GTFSの構造理解も兼ねて、プリミティブな方法を使ってみる
- 使うツールは**DuckDB(SQL)**と**Kepler.gl**だけ
 - データ処理だけに全集中

GTFS可視化レシピ

1. GTFSをダウンロードする（ここでは都営バスを対象にする）

- GTFSの仕様は現在MobilityDataが管理

2. DuckDBにGTFSを読み込む

- DuckDBはOLAP: Online Analytical Processingに強いDB
- 1ファイルにデータをまとめられるので便利
- Excelで扱いきれないような膨大なデータでも処理できる
（数十GBぐらいのデータなら一般的なスペックで処理可能）

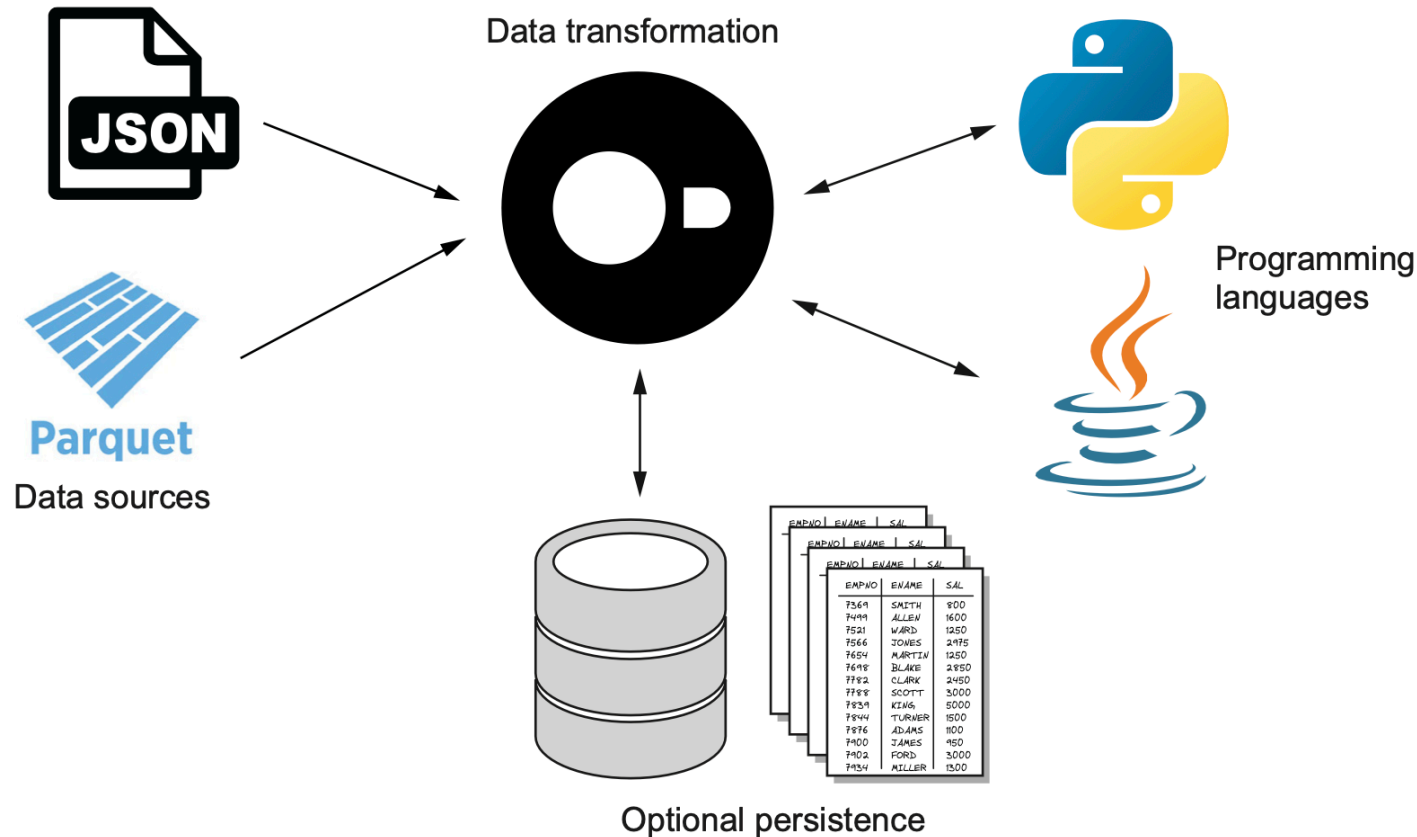
3. Kepler.glで可視化する

- GISデータをWebブラウザ上で可視化できるツール、OSSであればこれが一番軽量で使いやすい（渡邊調べ）
- 結果を1つのHTMLファイルとして出力できるので、ネットワークセキュリティの厳しい環境でも使える（重要）

ETL: Extract, Transform, Loadを実施=>現代のデータ基盤における基本

DuckDBについて

DuckDB as the hub of data ingestions and transformation



DuckDB as the hub of data ingestions and transformation

Needham, M., Hunger, M., Simons, M. (2024). [DuckDB in Action](#). Manning Publications.

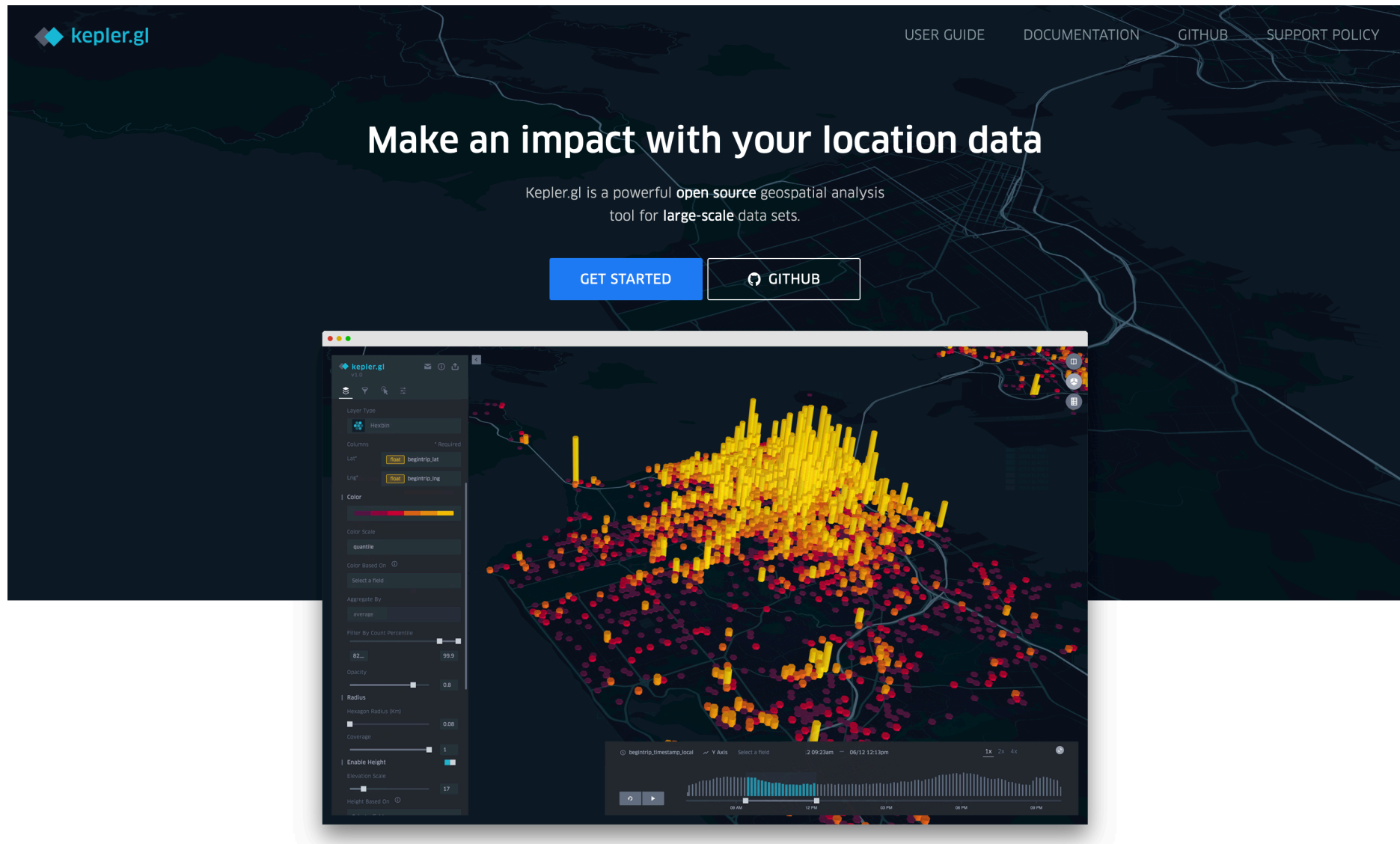
- **構造データをサクッといい感じに扱えるツール**
- 組み込み型DB、SQLiteのOLAP版と言われることもある
 - OLAP: Online Analytical Processing (大規模データ分析向け)
 - 列指向型DB (Columnar DB) であるため、データの圧縮効率が高い
 - 同じ列には同じ型で同じようなデータが入るため
- GISデータ (GeoJSON、Shapefile、etc) も読み込めて簡易なGIS演算もできる
 - **公共交通データを扱う際に超重要**
- 読み込んだデータをSQLで処理 (変換・整形) できる
 - 標準SQLはほぼサポートされている
- **dbt**や**dlt**と組み合わせて使うとETL (Extract, Transform, Load) 環境が作れる
 - DuckDB単体でもETL的なことはできるが、更新系の管理や変換ルールを管理するために使う

DuckDBのインストール

<https://duckdb.org/docs/installation>

- Windows: `winget install DuckDB.cli`
 - WSLの場合はLinuxの手順を使う
- macOS: `brew install duckdb`
 - [Homebrew](#) をインストールしてから実行
- Linux: https://github.com/duckdb/duckdb/releases/download/v1.1.0/duckdb_cli-linux-amd64.zip
 - ダウンロードして解凍できるバイナリを使う

Kepler.glについて



<https://kepler.gl>

- Webブラウザ上でGISデータをいい感じに可視化できるツール
- Uberが開発したツールがOSS化されたもの
 - deck.glをベースに構築されている
- 地理情報を含んだCSVやGeoJSONを読み込んで可視化できる
 - サーバーに情報が送られるのではなく、ブラウザの中で処理されて表示される
- レイヤーを重ねて可視化できる
 - e.g. バス停勢圏のレイヤーに、夜間人口のレイヤーを重ねる
- [Forsquare Studio](https://forsquare.com/studio/)という高機能なプロプライエタリ版がある
 - 元はUnfoldedというスタートアップが開発していたものをForsquareが買収
 - Forsquare StudioはDBに直接接続してデータを取得できるなどの拡張があるため、必要に応じてこちらを使用することも推奨
 - （最近OSSであるKepler.gl側のメンテナンスが滞っている印象がある為、たまにDemoが動かないことがある・・・）

DuckDBに GTFSを読み込む

GTFSをダウンロードする

- 都営バスのGTFSをダウンロード

```
1 curl -L -o ToeiBus-GTFS.zip https://api-public.odpt.org/api/v4/files/Toei/data/ToeiBus-GTFS.zip
```

- ダウンロードしたGTFSを解凍

```
1 unzip ToeiBus-GTFS.zip
```

- GTFSのファイル構造を眺めてみる

```
1 # ls -l
2 ToeiBus-GTFS.zip
3 agency.txt: 運行会社情報
4 agency_jp.txt
5 attributions.txt
6 calendar.txt: 運行日情報
7 calendar_dates.txt: 運行日付情報
8 fare_attributes.txt: 運賃情報
9 fare_rules.txt: 運賃ルール情報
10 feed_info.txt: フィード情報
11 office_jp.txt
12 routes.txt: 路線情報
13 shapes.txt: 路線形状情報
14 stop_times.txt: 時刻表情報
15 stops.txt: 停留所情報
16 translations.txt
17 trips.txt: 便情報
```

GTFSをDuckDBに読み込む

- DuckDBを起動する

```
1 $ duckdb ViewGTFS.duckdb
2
3 v1.0.0 1f98600c2c
4 Enter ".help" for usage hints.
5 D
```

- Spatial Extensionを有効にする（2回目以降はLOAD spatialのみで使える）

```
1 INSTALL spatial;
2 LOAD spatial;
```

- stops.txt、 shapes.txt, trips.txtを直接クエリし、その結果をテーブルとして保存

```
1 -- stops.csvをstopsテーブルに保存する
2 CREATE OR REPLACE TABLE stops AS SELECT * FROM read_csv('stops.txt');
3 CREATE OR REPLACE TABLE shapes AS SELECT * FROM read_csv('shapes.txt');
4 CREATE OR REPLACE TABLE trips AS SELECT * FROM read_csv('trips.txt');
```

- 作成したテーブルを確認してみる

```
1 SELECT * FROM stops LIMIT 5;
```

Running query in 'duckdb'

stop_id	stop_code	stop_name	stop_desc	stop_lat	stop_lon	zone_id	stop_type
0001-01	None	愛育クリニック前	None	35.653697	139.726017	0001-01	ht LC
0001-02	None	愛育クリニック前	None	35.654095	139.726541	0001-02	ht LC
0003-01	None	青戸車庫前	None	35.744787	139.843847	0003-01	ht LC
0003-02	None	青戸車庫前	None	35.745259	139.844403	0003-02	ht LC
0003-03	None	青戸車庫前	None	35.743838	139.84335	0003-03	ht LC

例外: stop_times.txtの対応

- 殆どはスキーマ自動推論に任せれば大丈夫だが、stop_times.txtだけ時刻が25:00:00などの日付を超えた表記方法が出現しtime型で読み込もうとして失敗するので、Schema (Columns) 指定する

```
1 CREATE OR REPLACE TABLE stop_times AS SELECT * FROM read_csv(  
2   'stop_times.txt',  
3   header=true,  
4   columns={  
5     'trip_id': 'VARCHAR',  
6     'arrival_time': 'VARCHAR',  
7     'departure_time': 'VARCHAR',  
8     'stop_id': 'VARCHAR',  
9     'stop_sequence': 'INTEGER',  
10    'stop_headsign': 'VARCHAR',  
11    'pickup_type': 'UTINYINT',  
12    'drop_off_type': 'UTINYINT',  
13    'shape_dist_traveled': 'DOUBLE',  
14    'timepoint': 'UTINYINT'  
15  });
```

蛇足: GTFSをDuckDBに変換する

面倒なので、GTFSをダウンロードしてDuckDBにロードするbashスクリプトを書いた

```
1 # Usage: bash GTFS2DuckDB.sh GTFS_URL
2 # bash GTFS2DuckDB.sh https://api-public.odpt.org/api/v4/files/Toei/data/ToeiBus-GTFS.zip
3
4 TITLE=$1
5 GTFS_URL=$2
6
7 GTFS_FILE=${GTFS_URL##*/}
8
9 # GTFSファイルをダウンロード
10 echo "Download: $GTFS_FILE"
11 curl -L -o $TITLE.zip $GTFS_URL
12
13 echo "Import: $GTFS_FILE to $TITLE.duckdb"
14 for basename in `unzip -Z1 $TITLE.zip`; do
15     filename=${basename%. *}
16     echo "${filename} to DuckDB"
17     # stop_timesのみスキーマ自動判定が失敗するので、手動でスキーマ定義を渡す (arrival_time, departure_timeが25:00:00のよ
18     if [ $filename = "stop_times" ]; then
19         unzip -p $TITLE.zip $basename | duckdb $TITLE.duckdb -c "CREATE OR REPLACE TABLE ${filename} AS SELEC
20     else
21         unzip -p $TITLE.zip $basename | duckdb $TITLE.duckdb -c "CREATE OR REPLACE TABLE ${filename} AS SELEC
22     fi
23 done
```

GTFSのURLを指定したら、ダウンロードしてDuckDBに読み込んで保存する
(e.g. /ToeiBus-GTFS.zip を指定したら、ToeiBus-GTFS.duckdbとして保存する)

```
1 bash GTFS2DuckDB.sh https://api-public.odpt.org/api/v4/files/Toei/data/ToeiBus-GTFS.zip
```

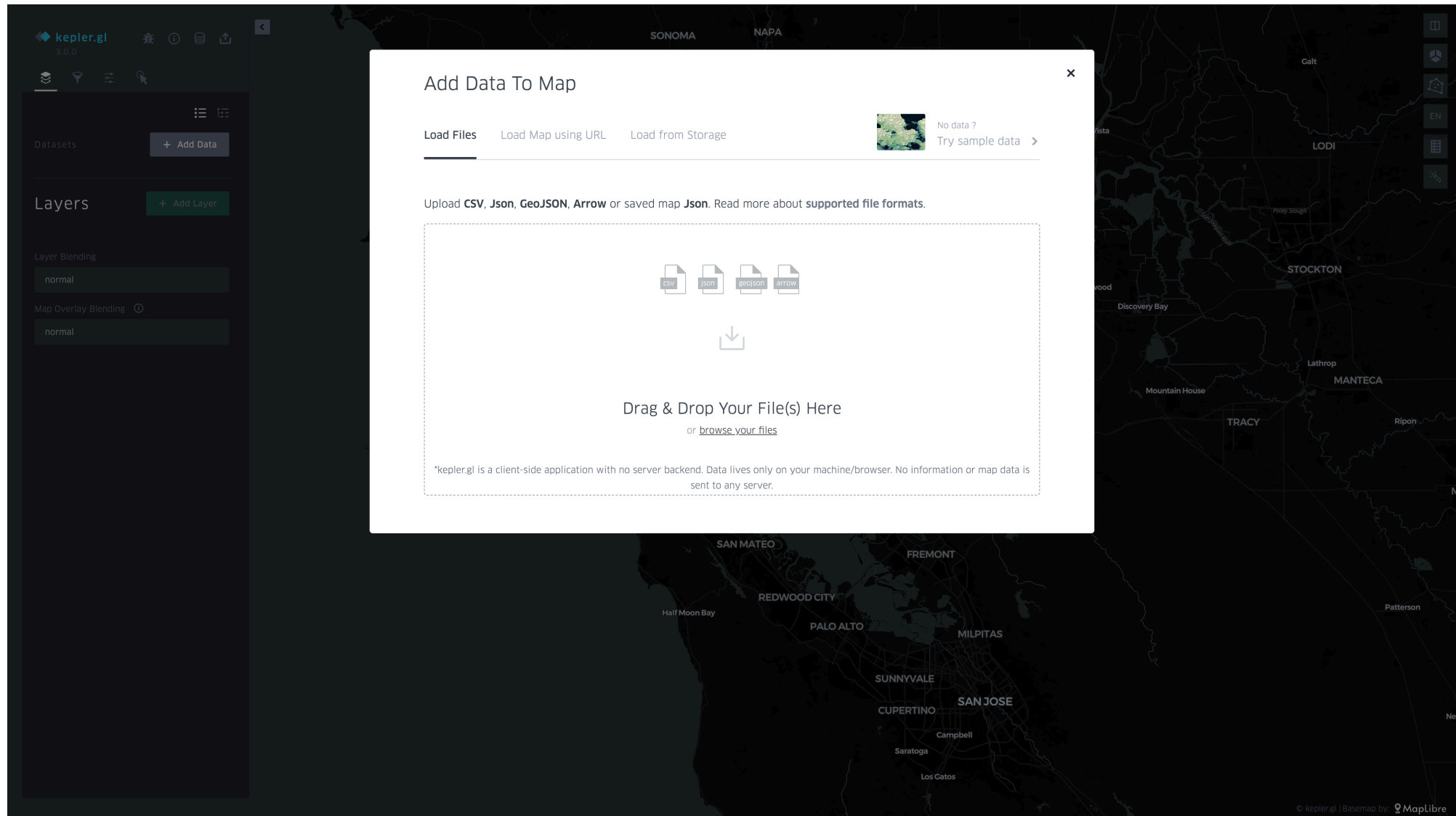
Kepler.glで可視化する

DuckDBでGTFSをKepler.glで可視化できるデータに変換する

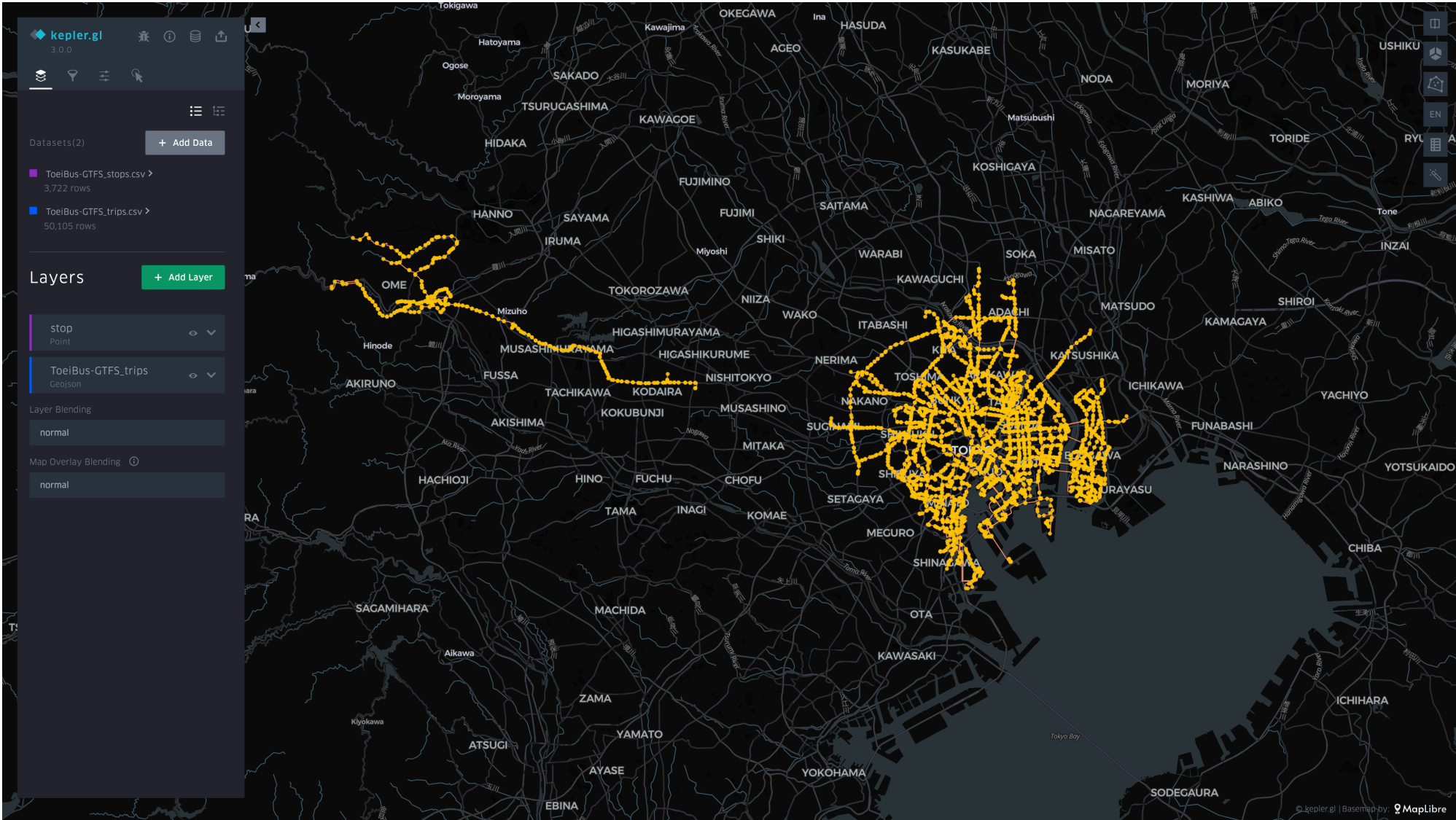
```
1 %%sql
2 -- stopsをcsvで出力する
3 COPY (
4     SELECT * FROM stops
5 ) TO 'ToeiBus-GTFS_stops.csv' WITH (FORMAT CSV, HEADER);
```

```
1 %%sql
2 -- shapesから運行路線形状をGeoJSON::LineStringで出力する
3 COPY (
4     SELECT * FROM trips
5     LEFT JOIN (
6         SELECT
7             shape_id,
8             {
9                 "type": 'LineString',
10                "coordinates": list([shape_pt_lon,shape_pt_lat])
11            }::JSON AS "geometry"
12         FROM shapes GROUP BY shape_id
13     ) AS shapes ON trips.shape_id = shapes.shape_id
14 ) TO 'ToeiBus-GTFS_trips.csv' WITH (FORMAT CSV, HEADER);
```

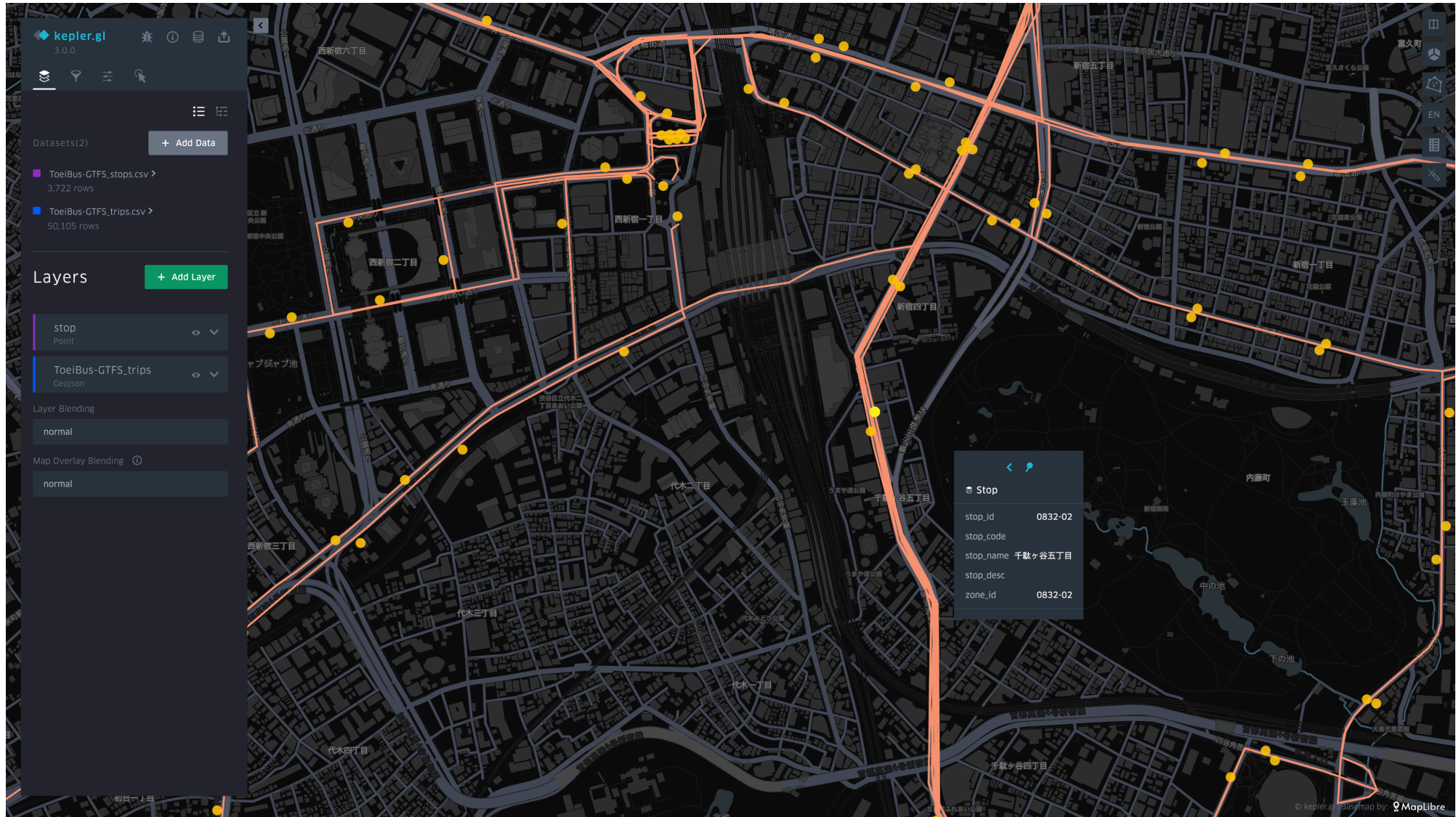
<https://kepler.gl/demo>を開いて、出力された ToeiBus-GTFS_stops.csv と ToeiBus-GTFS_trips.csv をD&Dする



停留所と路線形状が可視化される



バス停名なども保持されている



あれ・・・？

- stops（標柱）を出すだけなら、DuckDBを使わずにそのままGTFSに含まれる stops.txtをKepler.glで表示できるのでは・・・？？
 - その通り！
- なぜDuckDBを使った？？
 - ただの可視化から先に進むため
 - e.g. とある場所から半径1km以内に存在する停留所を抽出する = SQL1行書けばOK（ST_BufferとST_Containsを使う）

ちょっと複雑な例

国勢調査と組み合わせてみる

- e-Stat統計地理情報システムから国勢調査の結果を取得する
 - GIS処理しやすいように加工されたデータが提供されている
- 東京都が含まれるM5339の5次メッシュShapeデータをダウンロードする
 - 1次メッシュ枠情報: https://www.e-stat.go.jp/pdf/gis/primary_mesh_jouhou.pdf
- ダウンロードしてきたファイルを解凍し、DuckDBに読み込む

```
1 $ unzip QDDSWQ5339.zip
2 $ duckdb ViewGTFS.duckdb
3 v1.0.0 1f98600c2c
4 Enter ".help" for usage hints.
5 D load spatial;
6 D CREATE OR REPLACE TABLE mesh_5th AS SELECT * FROM ST_Read('MESH05339.shp');
```

- 東京都が含まれるM5339の5次メッシュ国勢調査結果データをダウンロードする
- ダウンロードしてきたファイルを解凍し、文字コードをCP932(ShiftJIS)からUTF-8に変換する（しなくてもなんとかなるが、なんとなく気持ち悪いので変換する）

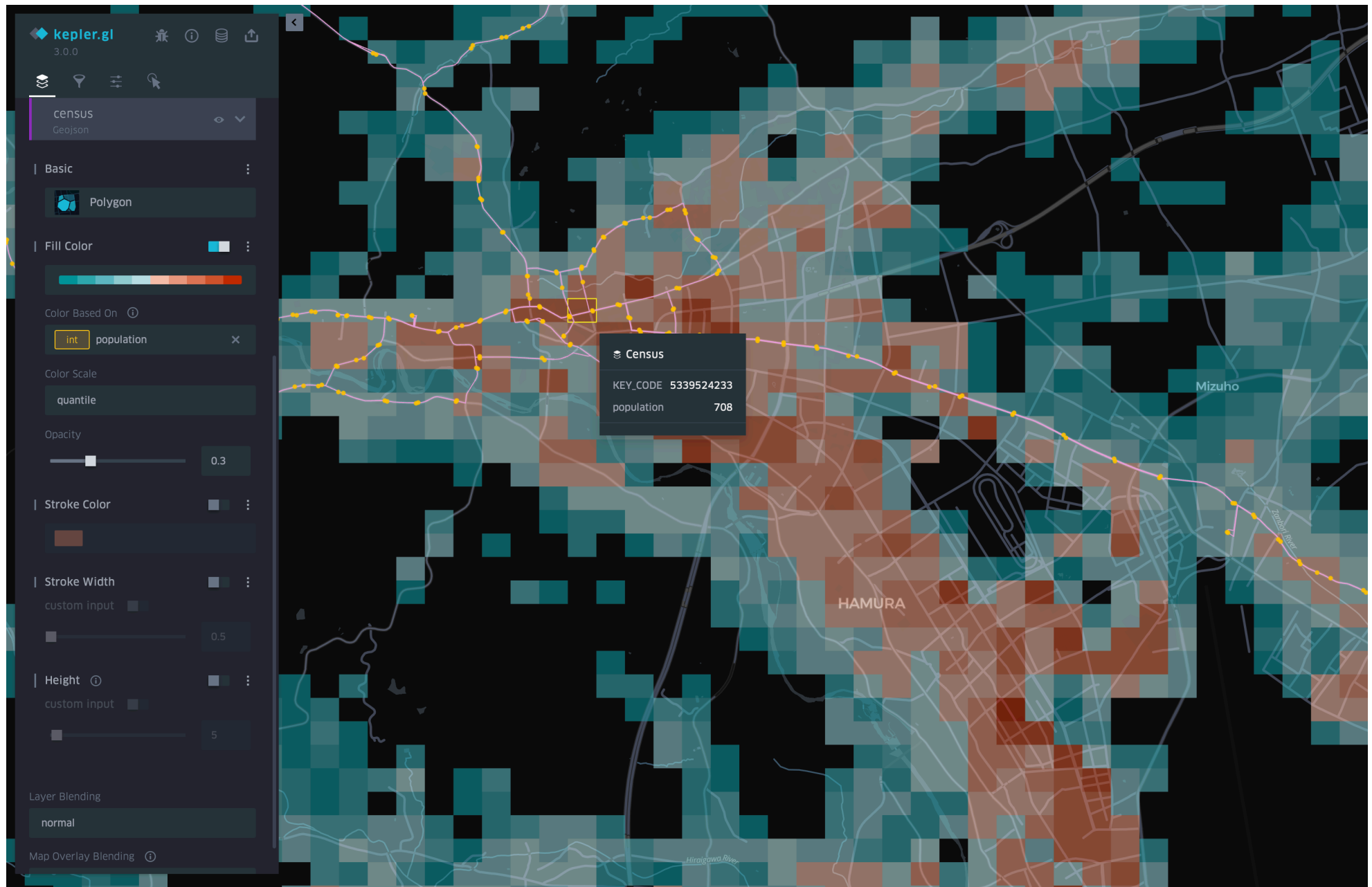
```
1 $ unzip -p tblT001142Q5339.zip | iconv -c -f cp932 -t utf-8 > tblT001142Q5339_utf8.txt
```

- 変換したデータをDuckDBに読み込む

```
1 $ duckdb ViewGTFS.duckdb
2 v1.0.0 1f98600c2c
3 Enter ".help" for usage hints.
4 D load spatial;
5 -- 2行の日本語ヘッダーがレコードとして含まれているが気にしない
6 D CREATE OR REPLACE TABLE census AS SELECT * FROM read_csv('tblT001142Q5339_utf8.txt', header = true);
```

- メッシュコードをJOINしてGeoJSON::ポリゴンとして保存する

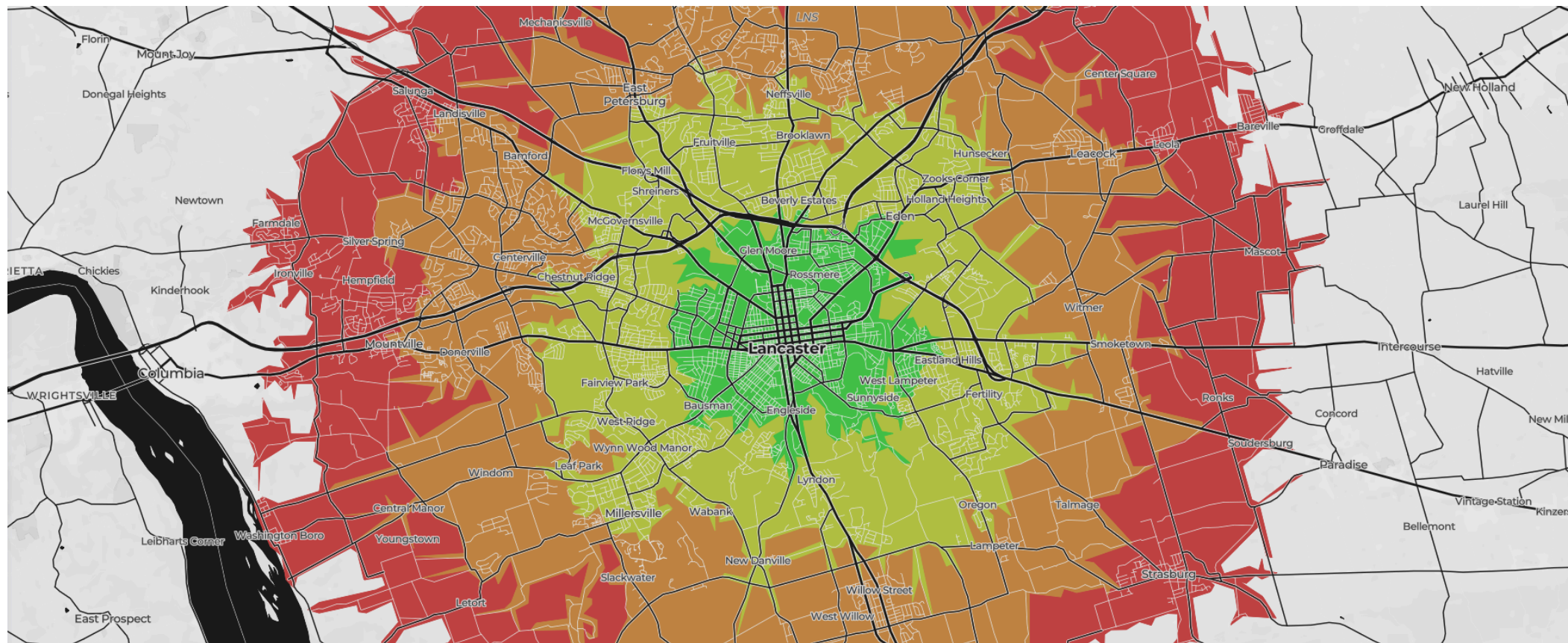
```
1 -- メッシュ内総人口はT001142001カラムに保存されているので、メッシュコードのポリゴンと人口をJOINしてGeoJSONとして保存する
2 D COPY (
3     SELECT mesh_5th.KEY_CODE, census.T001142001 AS population, geom FROM mesh_5th
4     LEFT JOIN census ON mesh_5th.KEY_CODE = census.KEY_CODE
5 ) TO 'census.geojson' WITH (FORMAT GDAL, DRIVER 'GeoJSON', LAYER_CREATION_OPTIONS 'WRITE_BBOX=YES');
```



国勢調査データとGTFsを重ねて表示した例

経路探索と組み合わせてみる

- GTFSを読み込んで、経路探索を行うツールを使う
 - [OpenTripPlanner](#)
 - [GraphHopper](#)
 - [Valhalla](#)（できるっぽいけど、特定のレジストリに依存するらしく自由に読み込ませる方法が不明）
- [@takoyaki3333333](#) が[OpenTripPlannerを使ったサンプル](#)を公開
- 経路探索エンジンを使うと何ができる？
 - 経路案内: 但し乗換所要時間などが精緻でないので、ナビゲーションとして使うには要注意
 - 経路推定: 乗車券が時刻t1にAバス停からBバス停で使われた場合に乗車したバス路線の推定
 - 等時線(Isochrone)分析: 特定の出発地点からある時間で到達可能な範囲を示したもの



例: Isochrone

まとめ

- GTFSをDuckDBに読み込んでKepler.glで可視化する方法を紹介
- 可視化はデータ活用の第一歩
 - 可視化をすることで、データを直感的かつ俯瞰的に見ることができる
 - 可視化をするためにデータを処理することで、応用ができるようになる
- 様々なデータを組み合わせることで、新たな価値が生まれる？！
 - 組み合わせる為には、対象データの「空間」を揃える必要がある
 - 今回は地理空間に揃えた
 - 他に時間軸、属性軸なども考えられる
 - このような処理を行う際、DuckDBはお手軽で非常に強力なツール
 - 今後の為にもSQLは使えるとヨシッ！

Let's データ利活用！

Appendix: ツール・サービス紹介

- [DuckDB](#): ローカルOLAP環境、GISデータも扱える、今後のデータ処理におけるデファクトはこれだと思ってる
- [Kepler.gl](#): Uberが開発したWebGLを使った高速な地理情報可視化環境
- [OpenTripPlanner](#): オープンソースの経路探索エンジン、GTFSを読み込んで経路探索ができる
- [Valhalla](#): Mapzenが開発した経路探索エンジン、OSMのデータを使って経路探索できる
- [GraphHopper](#): 歴史ある経路探索エンジン、GTFSを読み込んで経路探索ができる
- [gtfs-realtime-bindings](#): GTFS-RTを扱う時に使うライブラリ、これがデファクトだと思う（方法はいっぱいあるけど）
- [Visual Studio Code](#): Microsoftが開発した軽量なIDE、PythonやSQLの開発にも使える、最早デファクトエディタ（エディタ戦争の終焉）
- [Jupyter Lab](#): Pythonのインタラクティブな開発環境、データ分析にはよく使われる
- [Quarto](#): Markdownで記述したドキュメントを、PDFやHTML、docxに出力できる。内部でPandocを使用。このプレゼンもQuarto製、TeXの時代は終わったのかもしれない・・・
- [uv](#): Rustで実装された、Pythonパッケージ管理システム。最近venv環境構築にも対応したのでpipenvから乗り換えた（脱Lock地獄）
- [Polars](#): Rustで実装された高速なデータフレームライブラリ、PythonのPandasと互換性がある
- [GeoPandas](#): Pandasの拡張で、GISデータを扱うためのライブラリ
- [deck.gl](#): Uberが開発したWebGLを使った高速な地理情報可視化ライブラリ、Python Bindingもある
- [Pandas](#): Pythonのデータフレームライブラリ、ど定番。
- [Dusk](#): Pythonの分散処理ライブラリ、Pandasで処理を書いてしまいデータが大きすぎて処理できない場合の救世主
- [dbt](#): ETLツール、SQLでデータ変換を記述できる
- [dlt](#): Data Load Tool、データロードを管理するためのツール
- [Apache Superset](#): Airbnbが開発したBIツール、OSSのBIツールの中ではGISデータの扱いが比較的得意
- [Forsquare Studio](#): Kepler.glの高機能版
- [Databricks](#): Apache Sparkをベースにしたクラウド型データ基盤、データ分析環境。DuckDBで管理しきれなくなったらこちらに移行検討を（最初から検討した方がいいケースもあり）