

【2026年3月】3週連続集中開催

Web3.0関連技術 データ・ブロックチェーン・生成AI ハンズオンセミナー

Pythonではじめる人流データ解析実践 ～仙台市の実データを用いた解析と可視化演習～

講師 東北大学データ駆動科学・AI教育研究センター 助教 張 彰

3月5日（木）14:00～17:00

Pythonではじめる人流データ解析実践

～仙台市の実データを用いた解析と可視化演習～

データ駆動科学・AI教育研究センター
データ基盤・セキュリティ教育研究部門

助教 張 彰

このセミナーでは、Python（Google Colaboratory）を使って人流データの解析と視覚化を行いながら、データ処理の基本的な手法を学ぶ。

- データの読み込みと前処理
- 人流データの解析
- 視覚化手法（グラフなど）
- 実践演習：Pythonによるデータ分析

学習目標

- データ分析能力の向上
- Pythonプログラミングの実践的応用
- 視覚化技術の習得
- 実社会でのデータ活用の重要性を学ぶ





目次

人流データの紹介	 P5
基礎的な解析	 P12
繁華街の人流解析	 P28
エリア間の人流解析	 P45
練習問題例	 P57



目次

人流データの紹介 P5

データ概要

- スマホアプリのGPS機能を通じて収集された仙台市全体の人流データ
- 東北大学がAgoop社より購入
- 2019年（6月・12月）、2020年（6月・12月）、2021年（1月～12月）、2022年（6月・12月）、2023年（6月・12月）、2024年（6月）
- データは一日単位でCSVファイル形式に保存



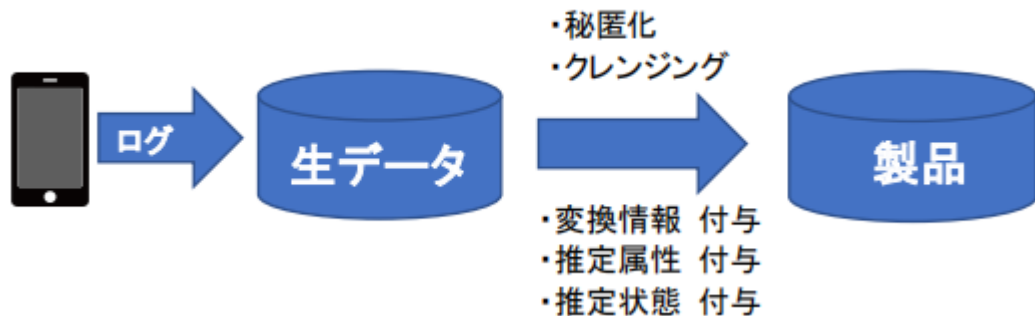
データの収集方法

Agoop社*が提供する開発ツールを搭載したスマートフォンのアプリから収集

*Agoop社は、スマホアプリから大量の位置情報・センサー情報を収集し、位置情報ビッグデータとして第三者への販売を行なっている企業



データに適用している処理



処理名	概要
クレンジング	不正ログ等を除外する処理
秘匿化	推定した居住地周辺のログを秘匿化する処理
変換情報 付与	生データから得られる情報を別の形に変換し付加する処理 例) 緯度、経度から都道府県コードを付与
推定属性 付与	生データから推計される属性を付加 例) ユーザの推定居住国等
推定状態 付与	生データから推計されるユーザの状態を付加 例) 推定スピード等

データ規模（1日のデータサイズ）

- 2019年: 約 300MB
- 2020年: 約 200MB
- 新型コロナウイルスの影響により外出が減少したため、データ容量も減少？
- 2021年: 約 500MB
- 2022年: 約 900MB
- 2023年: 約 800MB
- 2024年: 約 800MB

新型コロナウイルスの影響により外出が減少したため、データ容量も減少？

データの内容

No	タイトル	CSV 項目	データ型	NULL	単位	推定
1	デイリーID	dailyid	文字列(96 桁)			← ユーザーを識別するID
2	年	year	整数(4 桁)		年	} ← 時間情報
3	月	month	整数(1-2 桁)		月	
4	日	day	整数(1-2 桁)		日	
5	曜日	dayofweek	整数(1 桁)			
6	時間	hour	整数(1-2 桁)		時	} ← 位置情報
7	分	minute	整数(1-2 桁)		分	
8	緯度	latitude	小数点以下 6 桁			
9	経度	longitude	小数点以下 6 桁			
10	OS	os	文字列			

4	推定勤務地 市区町村コード	workplace_citycode	文字列(5 桁)	有		○	Ex)"01101","
5	性別	gender	文字列	有			"m","f" ← 性別情報

性別によるユーザー数が違い

例えば、2019年6月の一平均 男性:約3250名、女性:約1875名

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	dailyid	year	month	day	dayofweek	hour	minute	latitude	longitude	os	home_cour	plmn	plmn_count
2	2d52c88c9	2020	12	1	2	0	0	38.37638	141.1491	Android	Japan	44010	Japan
3	2dacc0cc2fe	2020	12	1	2	0	0	38.22777	140.8817	Android	Japan	44010	Japan

データ概要

- 基地局ベースの位置情報に基づいて集計された人流データ（国分町付近）
- 東北大学がNTTドコモビジネス（株）より購入
- 2019年（6月・12月）、2020年（6月・12月）、2021年（6月・12月）、2022年（6月・12月）、2023年（6月・12月）、2024年（6月・12月）
- データはCSVファイル形式に保存

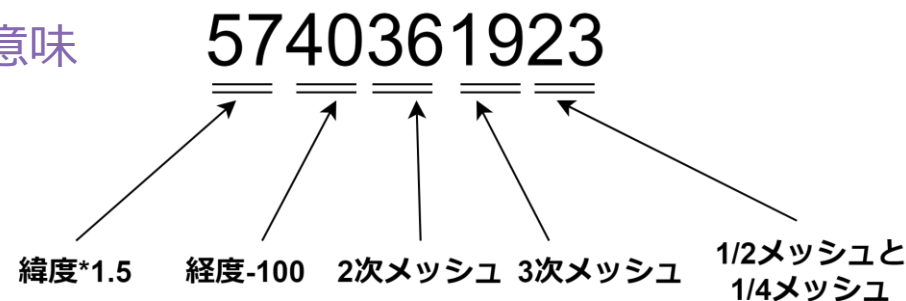
データ規模（全体）

- 総数 2.15MB
- 性別・年代別 25.3MB
- 居住地別 20.0MB

データの内容

時間情報		位置情報	居住地情報	年代情報	性別情報	人数	
date	day_of_week	time	area	residence	age	gender	population
20190601		16 0000	5740361923	-1	-1	-1	2295
20190601		16 0000	5740361924	-1	-1	-1	1120
20190601		16 0000	5740361932	-1	-1	-1	3180
20190601		16 0000	5740361934	-1	-1	-1	2500
20190601		16 0000	5740361941	-1	-1	-1	5810
20190601		16 0000	5740361943	-1	-1	-1	2854
20190601		16 0100	5740361923	-1	-1	-1	1944
20190601		16 0100	5740361924	-1	-1	-1	954
20190601		16 0100	5740361932	-1	-1	-1	2708
20190601		16 0100	5740361934	-1	-1	-1	2200
20190601		16 0100	5740361941	-1	-1	-1	4610
20190601		16 0100	5740361943	-1	-1	-1	2228
20190601		16 0200	5740361923	-1	-1	-1	1635
20190601		16 0200	5740361924	-1	-1	-1	812
20190601		16 0200	5740361932	-1	-1	-1	2235
20190601		16 0200	5740361934	-1	-1	-1	1841
20190601		16 0200	5740361941	-1	-1	-1	3535
20190601		16 0200	5740361943	-1	-1	-1	1690
20190601		16 0300	5740361923	-1	-1	-1	1420

位置情報の意味



本データは個人の移動履歴を追跡するものではなく、一定時間ごとに集計された人口分布データである。

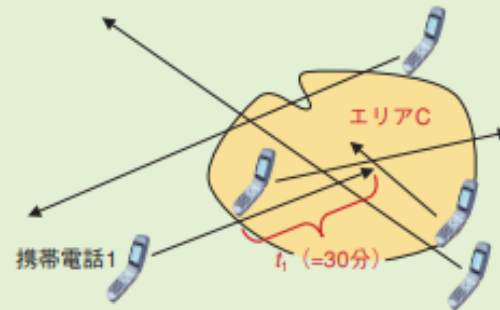
データの集計方法

収集対象：各基地局ごとに周期的に確認される携帯電話の在圏（存在）情報

非識別化处理：個人情報 は完全に除去

推計処理：**在圏数推計**、拡大推計、エリア変換処理

- たとえば、ある1時間に、「基地局エリアC」を5台の携帯電話が通過もしくは滞在したとする。
- ここで、携帯電話1～5は、この1時間のうちに、それぞれ $t_1 \sim t_5$ の時間だけエリアCに在圏していたとする。
- たとえば、携帯電話1の在圏時間は、 $t_1=30$ 分だったとする。



在圏数推計の考え方：

携帯電話 i は、1時間（60分）の間にそれぞれ t_i （分）在圏していた。

⇒上記1時間のある時点において、携帯電話 i がエリアCに在圏している確率は、 $t_i/60$ である。

（たとえば、携帯電話1が在圏している確率は $30/60 = 1/2$ ）

⇒すなわち、エリアCに在圏する携帯電話数（在圏数）の期待値は、

$t_1/60 + t_2/60 + \dots + t_5/60 = \sum t_i/60$ （台）となる。

図1 在圏数推計の考え方

*寺田雅之, 永田智大, 岩澤俊弥, 小林基成：モバイル空間統計における推計技術，NTT DOCOMO テクニカル・ジャーナル, Vol.20, No.3, pp.11-16, 2012.

データの集計方法

収集対象：各基地局ごとに周期的に確認される携帯電話の在圏（存在）情報

非識別化処理：個人情報情報は完全に除去

推計処理：在圏数推計、**拡大推計**、エリア変換処理

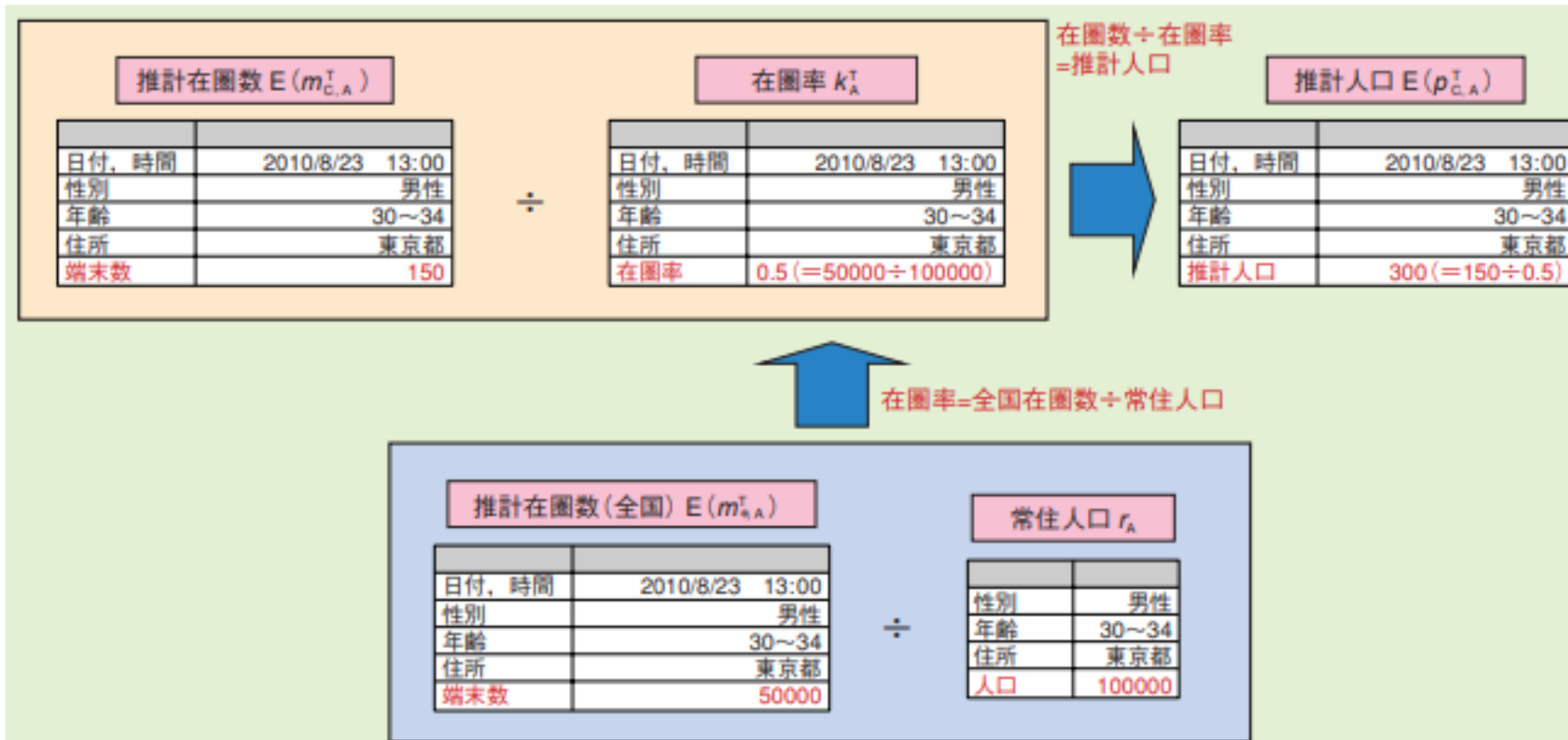


図2 拡大推計処理のイメージ図

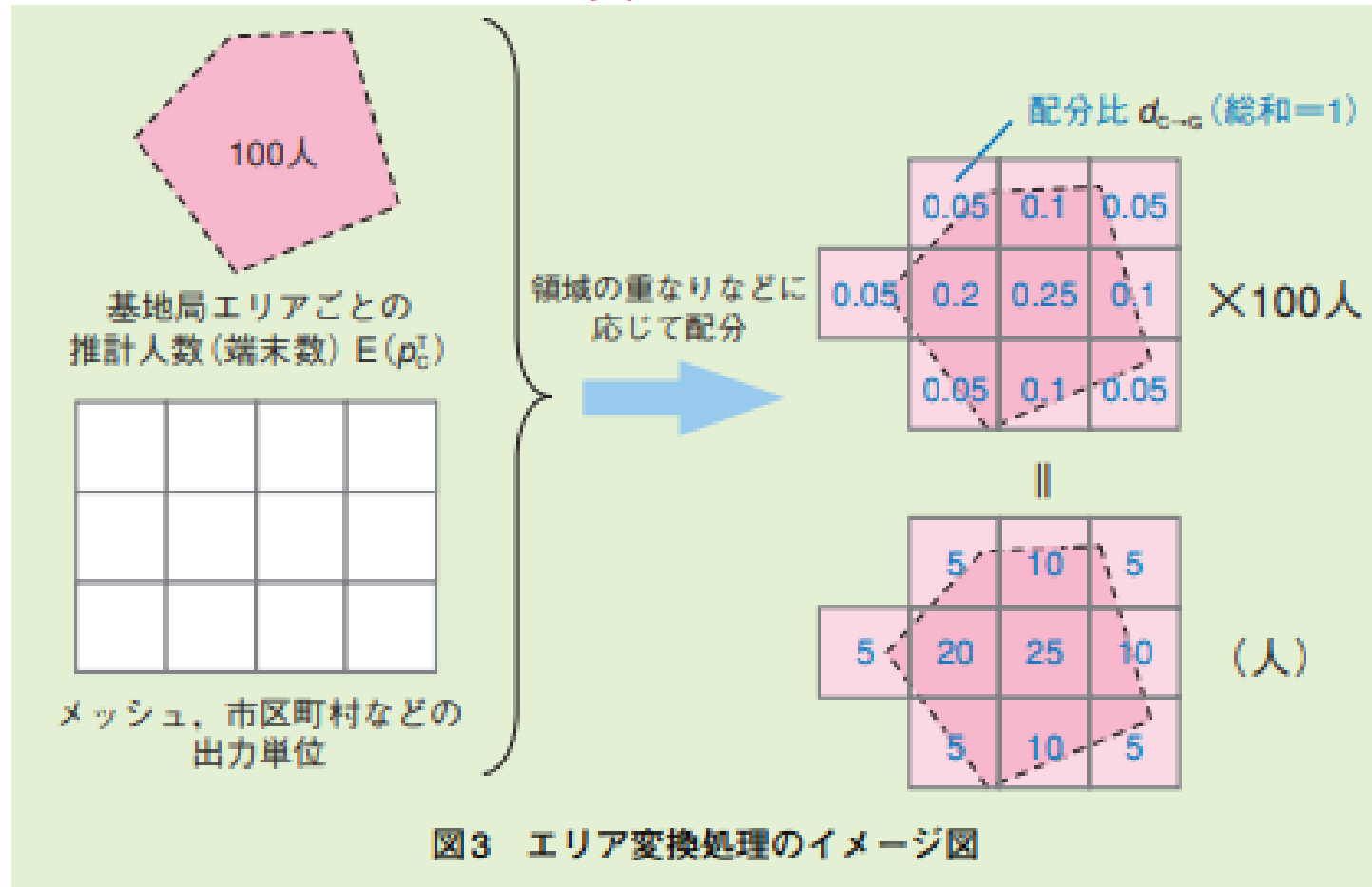
*寺田雅之, 永田智大, 岩澤俊弥, 小林基成: モバイル空間統計における推計技術, NTT DOCOMO テクニカル・ジャーナル, Vol.20, No.3, pp.11-16, 2012.

データの集計方法

収集対象：各基地局ごとに周期的に確認される携帯電話の在圏（存在）情報

非識別化処理：個人情報情報は完全に除去

推計処理：在圏数推計、拡大推計、**エリア変換処理**



*寺田雅之, 永田智大, 岩澤俊弥, 小林基成: モバイル空間統計における推計技術, NTT DOCOMO テクニカル・ジャーナル, Vol.20, No.3, pp.11-16, 2012.



目次

人流データの紹介 P5

基礎的な解析 P12

Python

シンプルで強力なプログラミング言語。データ解析や機械学習に広く使用されている。

Google Colaboratory

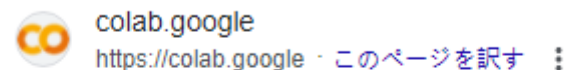
ブラウザ上でPythonコードを実行できる無料のクラウドサービス。環境設定不要で、強力なGPUも利用可能。

利用方法

Googleアカウント（ID）でログインすることで、Google Driveと連携し、データを保存・共有できる。

Google Colaboratory

Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.



Google Colab

Colab is a hosted Jupyter Notebook service that re access to computing resources, including GPUs an

URL: <https://colab.google/>

共有したファイルの使い方

- 共有ファイルを確認する
 - people_flow.ipynb
 - area_analysis.ipynb 分析およびデータ処理のためのコードが含まれている。
 - stop_move.ipynb
 - .CSV 仙台市の人流データ
- 共有されたipynbファイルを自身のGoogle Driveにコピーする。
 - マイドライブにコピーしたファイルはそのまま使える



 stop_move.ipynb のコピー	 自分
 people_flow_cmp_answer.ipynb のコピー	 自分
 area_analysis.ipynb のコピー	 自分
 people_flow.ipynb のコピー	 自分

共有したファイルの使い方

- csvファイルは公開不可なので、共有ファイルフォルダーにそのまま保管し、ショートカットのように使用する。



people_flow.ipynb

Colabの使い方

- ① .ipynbファイルをダブルクリック
- ② 開いたノートブックで、Pythonコードを実行できる
- ③ Google Driveのマウント

実行

```
from google.colab import drive
drive.mount('/content/drive')
```

pandas: Pythonのデータ解析ライブラリで、データ操作や分析に便利な機能を提供する。

④ データの読み込み

```
[2]
✓ 14 秒
import pandas as pd

# CSVの読み取り
file_path = '/content/drive/MyDrive/人流データ解析実践/201906/20190608.csv' # CSVファイルの場所
data = pd.read_csv(file_path)
```

⑤ データの確認:

```
[ ] # データ構造を確認
print(data.head())
```

```
dailyid year month day \
0 9719913cce620fc1ec334a4aaa7850e8df04c829372891... 2019 6 8
1 d55d67d9b0b3381676bd9aad67ce6c224935bb3d1a173f... 2019 6 8
2 8cf1a8258b8bd9cf36216b6b5c6563052dd514c043412c... 2019 6 8
3 d55d67d9b0b3381676bd9aad67ce6c224935bb3d1a173f... 2019 6 8
4 fe78af49b43c2bf0c46f6f9cc62fdcl1a73bcb50f425c03... 2019 6 8
```

```
dayofweek hour minute latitude longitude os ... course \
0 6 0 0 38.228120 140.908557 Android ... 86.308
1 6 0 0 38.276416 140.880986 iOS ... 147.656
2 6 0 0 38.229460 140.869599 Android ... NaN
3 6 0 0 38.276335 140.881131 iOS ... 253.476
4 6 0 0 38.238607 140.852856 Android ... NaN
```

```
estimated_course_flag prefcode citycode mesh100mid home_prefcode \
0 2.0 4 4103.0 5740277236 4.0
1 1.0 4 4101.0 5740373014 4.0
2 NaN 4 4104.0 5740267955 4.0
3 1.0 4 4101.0 5740373014 4.0
4 NaN 4 4104.0 5740268862 4.0
```

```
home_citycode workplace_prefcode workplace_citycode gender
0 4103.0 6.0 6203.0 m
1 4101.0 4.0 4101.0 NaN
2 4104.0 4.0 4101.0 m
3 4101.0 4.0 4101.0 NaN
4 4104.0 3.0 3206.0 f
```

[5 rows x 31 columns]

目的

ある1日の仙台市全体の人流データ量の時間推移と位置分布を把握する。

時間帯ごとの統計

```
count = data['hour'].value_counts().sort_index(ascending=True)
print(count)
```

hour 列内のユニークな値（時間）の出現回数をカウント
ascending=True: カウントされた結果をインデックス（時間）で昇順にソート

hour	
0	49030
1	44813
2	42287
3	40246
4	41446
5	44642
6	52073
7	61571
8	70967
9	74680
10	76772
11	78075
12	80459
13	78797
14	77121
15	77602
16	79067
17	79271
18	76684
19	69186
20	64398
21	60684
22	56132
23	51359

グラフによる視覚化

```
import matplotlib.pyplot as plt
```

データの視覚化を行うためのライブラリで、さまざまな種類のグラフや図を簡単に作成できる。

```
plt.figure(figsize=(10, 6))
```

グラフのサイズを10インチ×6インチに設定

```
plt.plot(count.index, count.values, marker='o', linestyle='-', color='b')
```

```
for i, value in enumerate(count.values):
```

```
    plt.annotate(f'{value}', (count.index[i], value),
```

```
                  textcoords="offset points", xytext=(0, 10), ha='center',
```

```
                  fontsize=8, color='b')
```

```
plt.xlabel('Hour')
```

```
plt.ylabel('Count')
```

```
plt.title('Count of Each Hour')
```

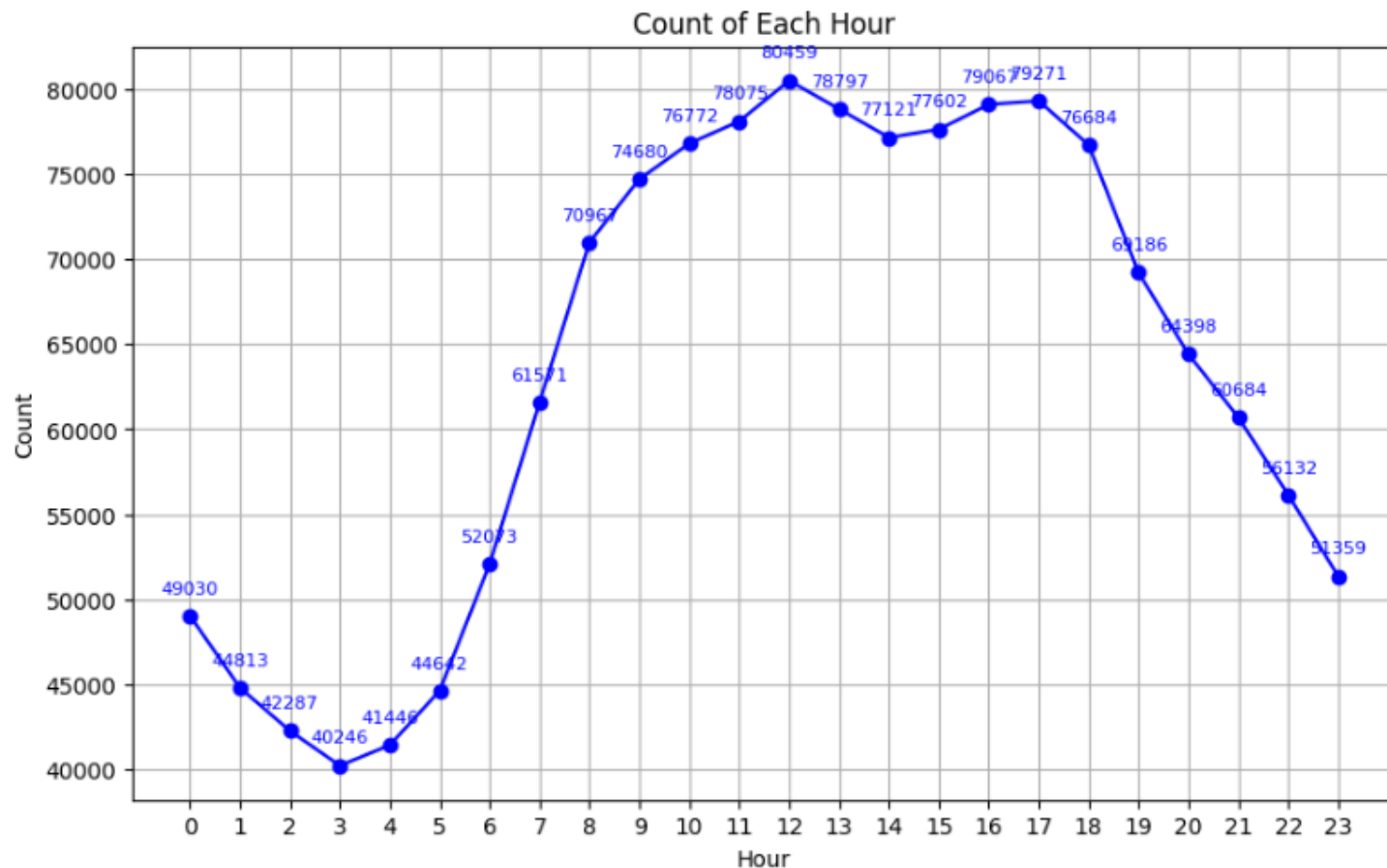
```
plt.grid(True)
```

```
plt.xticks(count.index)
```

```
plt.show()
```

時間（count.index）と人数（count.values）を使って線グラフを描画する。点の形状は円（marker='o'）、線のスタイルは実線（linestyle='-）、色は青（color='b'）である。

各データポイントの上に人数の値を表示する。オフセット（xytext=(0, 10)）を使って、値がポイントから少し離れた位置に表示される



結果の考察

この日（2019年6月8日・土）のデータでは、11時から17時の間に人数が多く、深夜は人数が少ないことがわかる。

（一般的な感覚の通り）

ユーザー性別の統計

```
unique_dailyid_male = male_df['dailyid'].nunique()
```

```
print(f"男性: {unique_dailyid_male}")
```

男性: 3093

```
unique_dailyid_female = female_df['dailyid'].nunique()
```

```
print(f"女性: {unique_dailyid_female}")
```

女性: 1888

- ユーザー数は、男女で異なることがある。

```
male_df = data[data['gender'] == 'm']#男性
female_df = data[data['gender'] == 'f']#女性
nan_df = data[~data['gender'].isin(['m', 'f'])]#性別不明

male_count = male_df['hour'].value_counts().sort_index(ascending=True)
female_count = female_df['hour'].value_counts().sort_index(ascending=True)
nan_count = nan_df['hour'].value_counts().sort_index(ascending=True)
df_combined = pd.DataFrame({
    'Male Count': male_count,
    'Female Count': female_count,
    'Nan Count': nan_count
})
totals = df_combined.sum()
df_combined.loc['Total'] = totals
print(df_combined)
```

- 時間を問わず、男性が多い
- トータルで男性比率 = 64.6% (性別不明を除外)

	Male Count	Female Count	Nan Count
hour			
0	17148	9558	22324
1	15920	9006	19887
2	15297	8205	18785
3	14595	7826	17825
4	14910	7932	18604
5	16104	8533	20005
6	18368	9574	24131
7	20573	10920	30078
8	22215	12340	36412
9	23273	12542	38865
10	23933	13250	39589
11	24342	13558	40175
12	25157	14075	41227
13	24256	13649	40892
14	24283	13107	39731
15	24477	13258	39867
16	24906	13709	40452
17	24969	13488	40814
18	24100	13228	39356
19	22271	12184	34731
20	21224	11986	31188
21	20319	11535	28830
22	19309	11008	25815
23	18133	10114	23112
Total	500082	274585	752695

トータルで男性比率 = 64.6%

その比率は時間帯によって変わったりするのか？

各時間帯の男女人数および男性の割合の統計

- 男性の人数を総人数で割り、男性の割合を計算する。

```
import numpy as np
import matplotlib.pyplot as plt

ratio = male_count / (female_count+male_count)  #男性の割合

hours = male_count.index
```

グラフによる視覚化

- 棒グラフと折れ線グラフの作成

```
fig, ax1 = plt.subplots(figsize=(12, 6))
```

- 男女人数を視覚的に比較するために、棒グラフを使用する。

```
bar_width = 0.4
indices = np.arange(len(hours))

p1 = ax1.bar(indices, male_count, bar_width, color='blue', label='Male')
p2 = ax1.bar(indices, female_count, bar_width, bottom=male_count, color='red', label='Female')

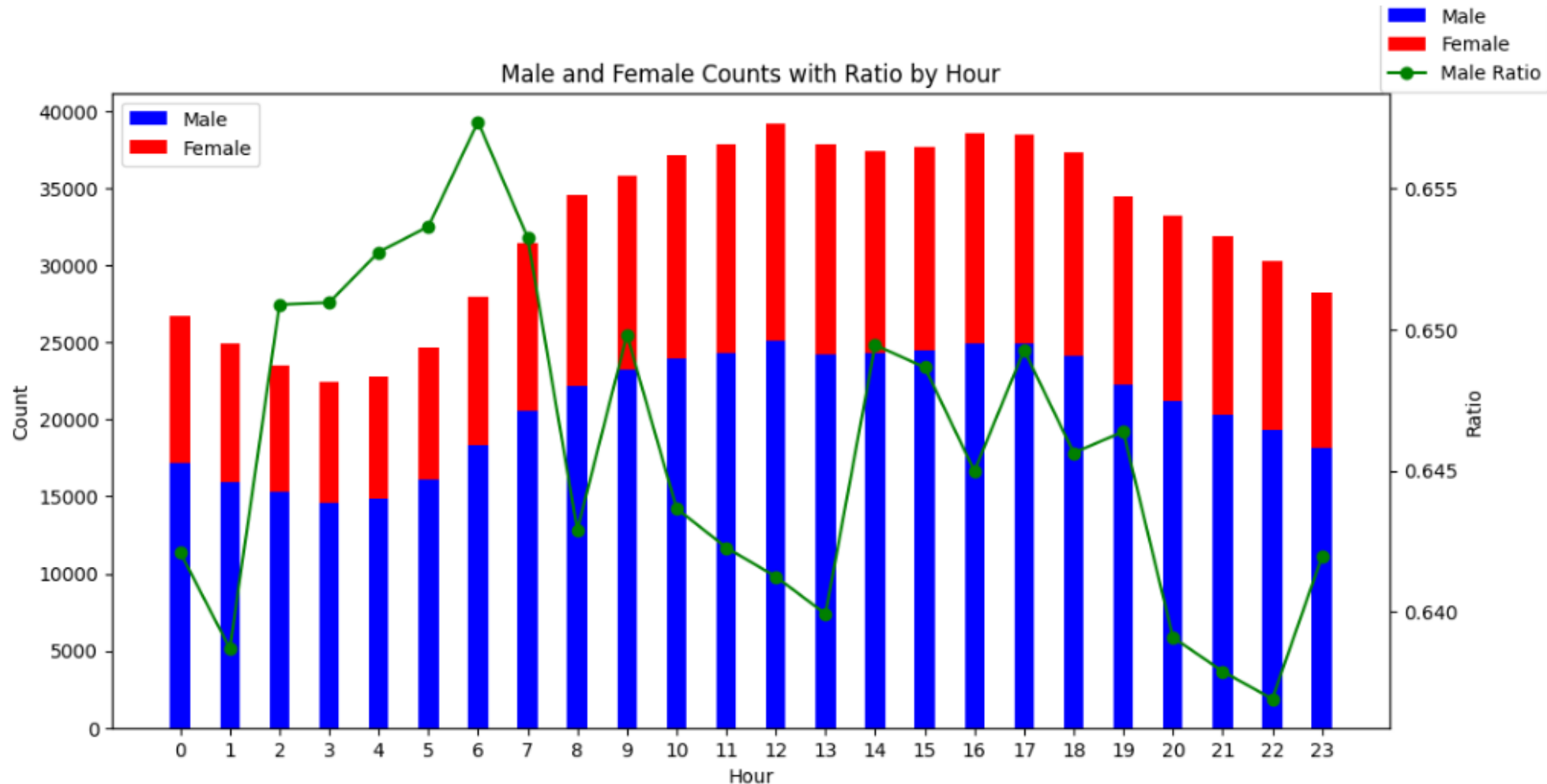
ax1.set_xlabel('Hour')
ax1.set_ylabel('Count')
ax1.set_title('Male and Female Counts with Ratio by Hour')
ax1.set_xticks(indices)
ax1.set_xticklabels(hours)
ax1.legend(loc='upper left')
```

- 時間に伴う男性の割合の変化を示すために、折れ線グラフを使用する。

```
# 割合の折れ線グラフ
ax2 = ax1.twinx()
ax2.plot(indices, ratio, color='green', marker='o', label='Male Ratio')

ax2.set_ylabel('Ratio')
fig.legend(loc='upper right')

plt.show()
```



統計結果から、深夜・早朝の時間帯において、男性の比率が高まることが確認できる。ただし、その比率の変動は一日を通して0.635から0.655の間であり、大きくはない。

位置情報（緯度と経度）のリストを作成

```
data['total_minutes'] = data['hour'] * 60 + data['minute']
grouped_data = data.groupby('total_minutes')

# 総分数をキー、その秒内にすべてのデバイスの位置リストを値とする辞書を作成する
locations_per_minute = {minute: group[['longitude', 'latitude']].values.tolist() for minute, group in grouped_data}
locations_per_minute_vlaue = {minute: group[['latitude', 'longitude']].values for minute, group in grouped_data}
```

locations_per_minute: 一分間ごとの位置情報をリスト形式で保存

- キー: その分に該当する総分数（例: 午後1時30分 → 810分）
- 値: その分内に記録されたデバイスの位置リスト（緯度・経度）

```
print(locations_per_minute_vlaue)

{0: array([[ 38.22812, 140.908557],
           [ 38.276416, 140.880986],
           [ 38.22946, 140.869599],
           ...,
           [ 38.313901, 140.842273],
           [ 38.237997, 140.843708],
           [ 38.282599, 140.875539]]), 1: array([[ 35.577497, 139.5941 ],
           [ 38.411745, 140.37876 ],
           [ 38.240475, 140.904092],
           ...,
           [ 38.222893, 140.888873],
           [ 38.231032, 140.780362],
           [ 38.262696, 140.889207]]), 2: array([[ 37.882375, 138.973094],
           [ 38.301497, 140.986611],
           [ 38.436392, 140.926218],
```

Dashをインストール

[14]
✓ 5 秒

```
!pip install dash==2.14.2
```

PythonでインタラクティブなWebアプリケーションを構築するためのフレームワークである

アニメーションを作成

```
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.graph_objs as go

max_total_minutes = data['total_minutes'].max()

# Dashアプリケーションの作成
app = dash.Dash(__name__)

# アプリケーションのレイアウト
app.layout = html.Div([
    dcc.Graph(id='scatter-map', style={'width': '80vw', 'height': '80vh'}),
    dcc.Interval(id='interval-component', interval=200, n_intervals=0) # 1秒ごとに更新
])

@app.callback(
    Output('scatter-map', 'figure'),
    Input('interval-component', 'n_intervals')
)
```

```
def update_graph(n):
    minute = n % (max_total_minutes + 1) # ループ再生
    locations = locations_per_minute.get(minute, [])

    # 経緯度データ
    latitudes = [loc[1] for loc in locations]
    longitudes = [loc[0] for loc in locations]

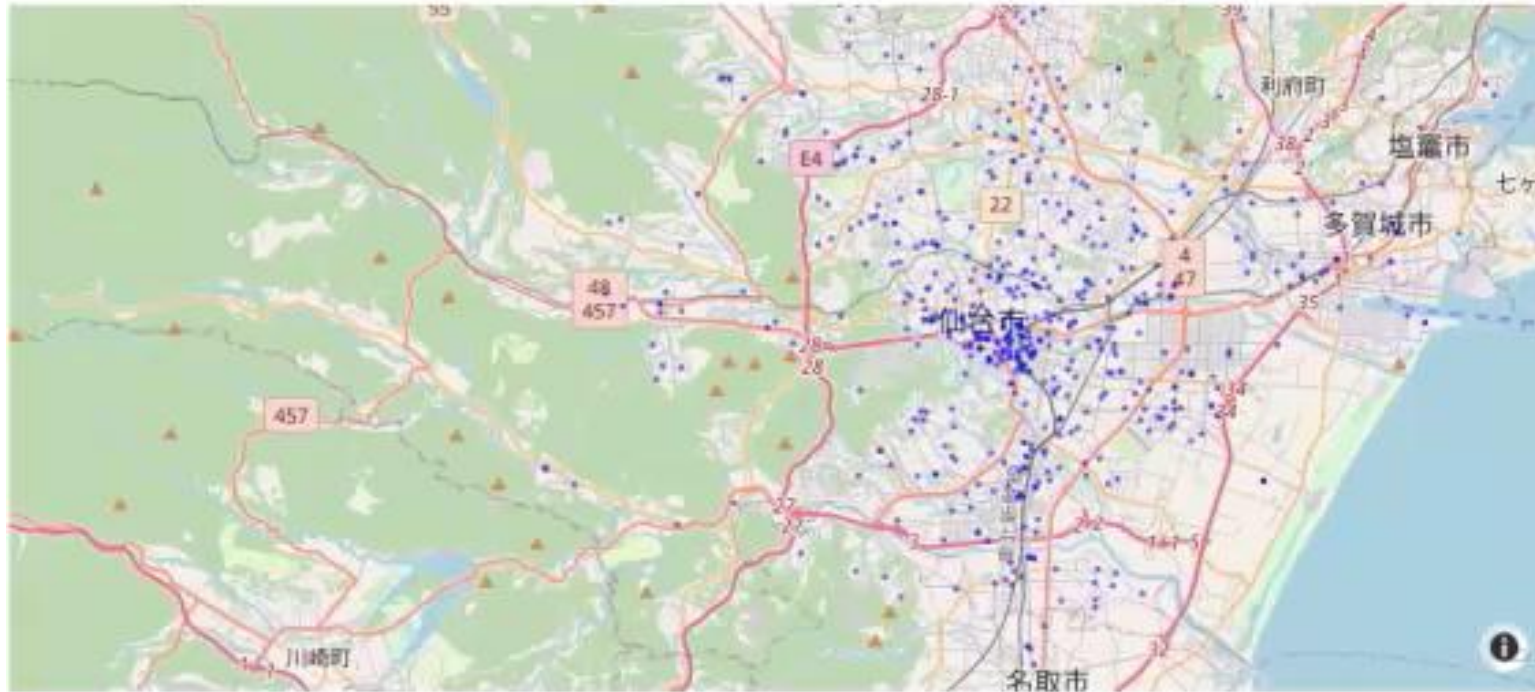
    # 散布図マップの作成
    scattermap = go.Scattermapbox(
        lat=latitudes,
        lon=longitudes,
        mode='markers',
        marker=dict(size=4, color='blue', opacity=0.6)
    )

    # 地図のレイアウトを設定し、OpenStreetMapを使用する
    layout = go.Layout(
        title=f'Time: {minute // 60:02d}:{minute % 60:02d}',
        autosize=True,
        hovermode='closest',
        mapbox=dict(
            style='open-street-map', # OpenStreetMap
            bearing=0,
            center=dict(lat=38.26, lon=140.8), # 地図の中心点を設定する
            pitch=0,
            zoom=10
        ),
    )

    return {'data': [scattermap], 'layout': layout}

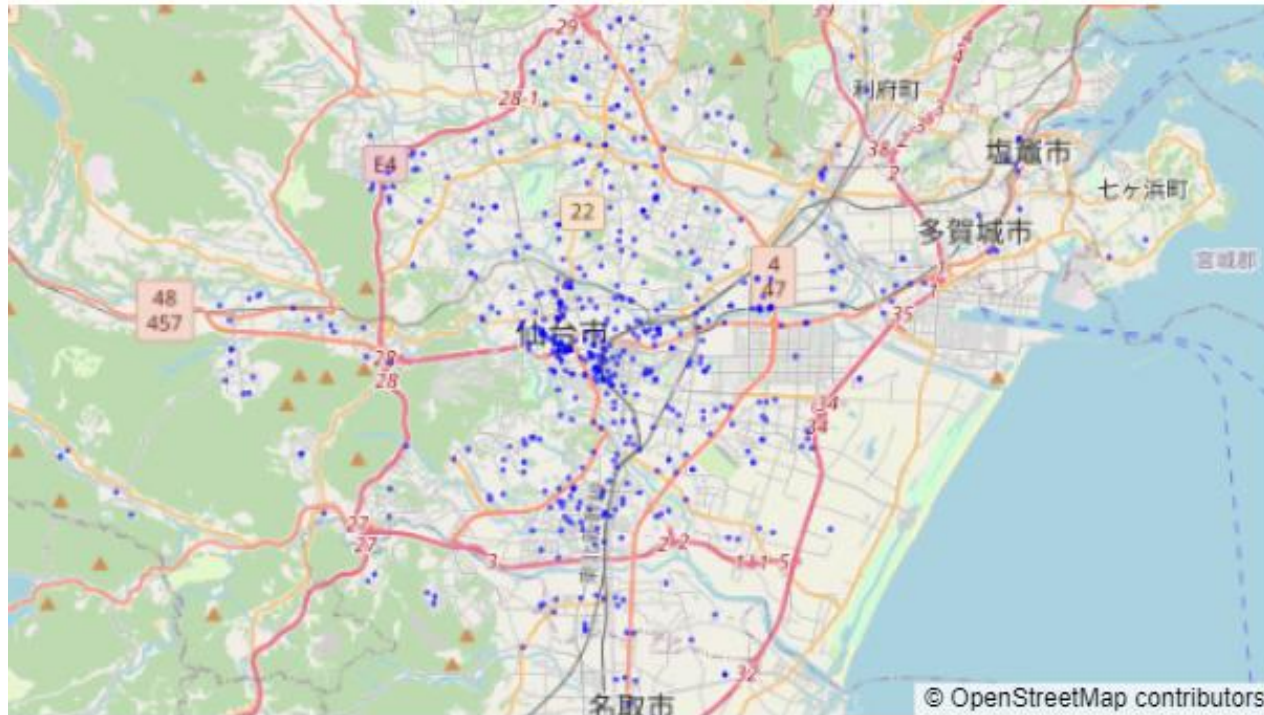
if __name__ == '__main__':
    app.run_server(debug=True)
```

Time: 03:54

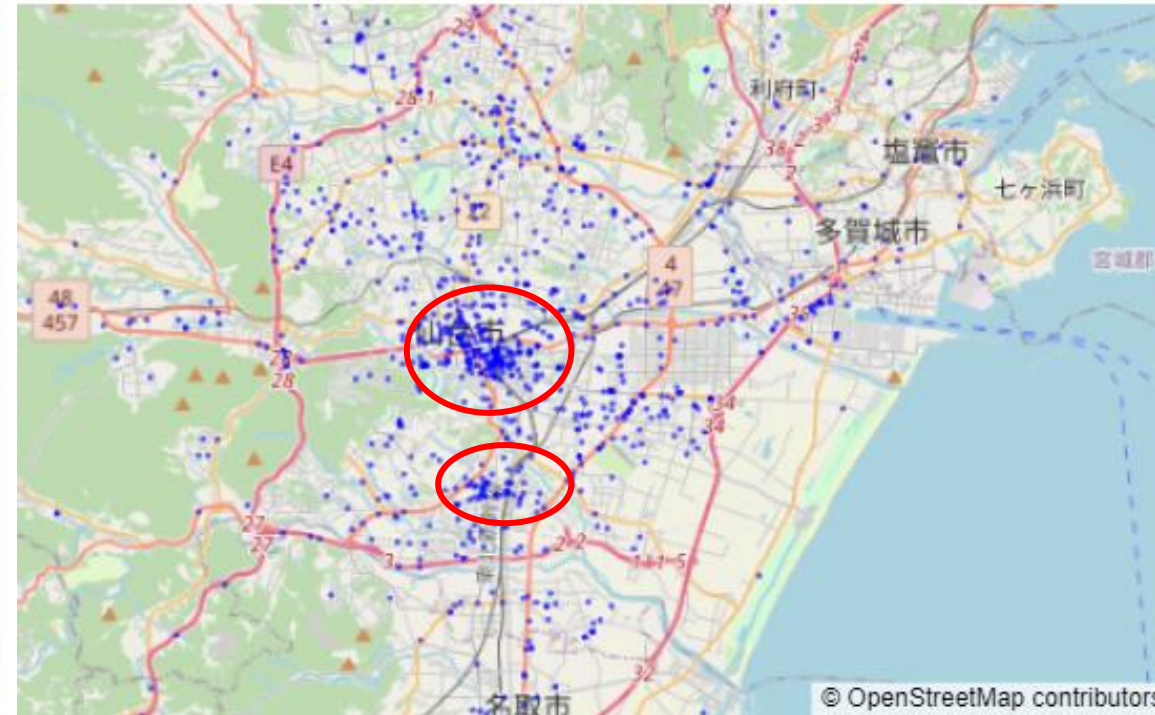


夜間（左図）よりも、昼（右図）の時間帯に全体的な人数（青点）が多く、また人流が商業施設(仙台市中央、長町)などの場所に集中していることがわかる。（一般的な感覚の通り）

Time: 01:31



Time: 12:22





目次

人流データの紹介	 P5
基礎的な解析	 P12
繁華街の人流解析	 P28

繁華街の人流動向は、仙台市全体とどのように異なるのか？

- 繁華街の人流動向を解析する
- 繁華街特有の特徴を明らかにする



area_analysis.ipynb

モバイル空間統計データから国分町付近の人流情報を読み取り

[1]
37
秒

```
from google.colab import drive
drive.mount('/content/drive')
```

▼

Mounted at /content/drive

```
import pandas as pd
```

```
# CSVの読み取り
```

```
file_path = '/content/drive/MyDrive/人流データ解析実践/モバイル空間統計データ/01_国内人口分布_総数.csv'
df = pd.read_csv(file_path)
df.head()
```

	date	day_of_week	time	area	residence	age	gender	population
0	20190601	16	0	5740361923	-1	-1	-1	2295
1	20190601	16	0	5740361924	-1	-1	-1	1120
2	20190601	16	0	5740361932	-1	-1	-1	3180
3	20190601	16	0	5740361934	-1	-1	-1	2500
4	20190601	16	0	5740361941	-1	-1	-1	5810

繁華街の人流解析

area_analysis.ipynb

国分町（繁華街）の人流データを統計解析

指定した日付のデータを抽出する関数を作成

[3]

```
import matplotlib.pyplot as plt

def plot_hourly_population(date_value):

    # ある日の情報を抽出
    df_day = df[df["date"] == date_value]
    if df_day.empty:
        print(f"⚠ {date_value} ")
        return

    # 時間ごと統計
    hourly_count = df_day.groupby("time")["population"].sum()
    hours = (hourly_count.index // 100).astype(int)

    plt.figure(figsize=(10, 6))
    plt.plot(hours, hourly_count.values, marker='o', linestyle='-', color='b')

    for i, value in enumerate(hourly_count.values):
        plt.annotate(
            f'{value:,}',
            (hours[i], value),
            textcoords="offset points",
            xytext=(0, 10),
            ha='center', fontsize=8, color='b'
        )

    plt.xlabel('Hour')
    plt.ylabel('Total Population')
    plt.title(f'Hourly Population on {date_value}')
    plt.grid(True)
    plt.xticks(hours, [f"{h:02d}" for h in hours])

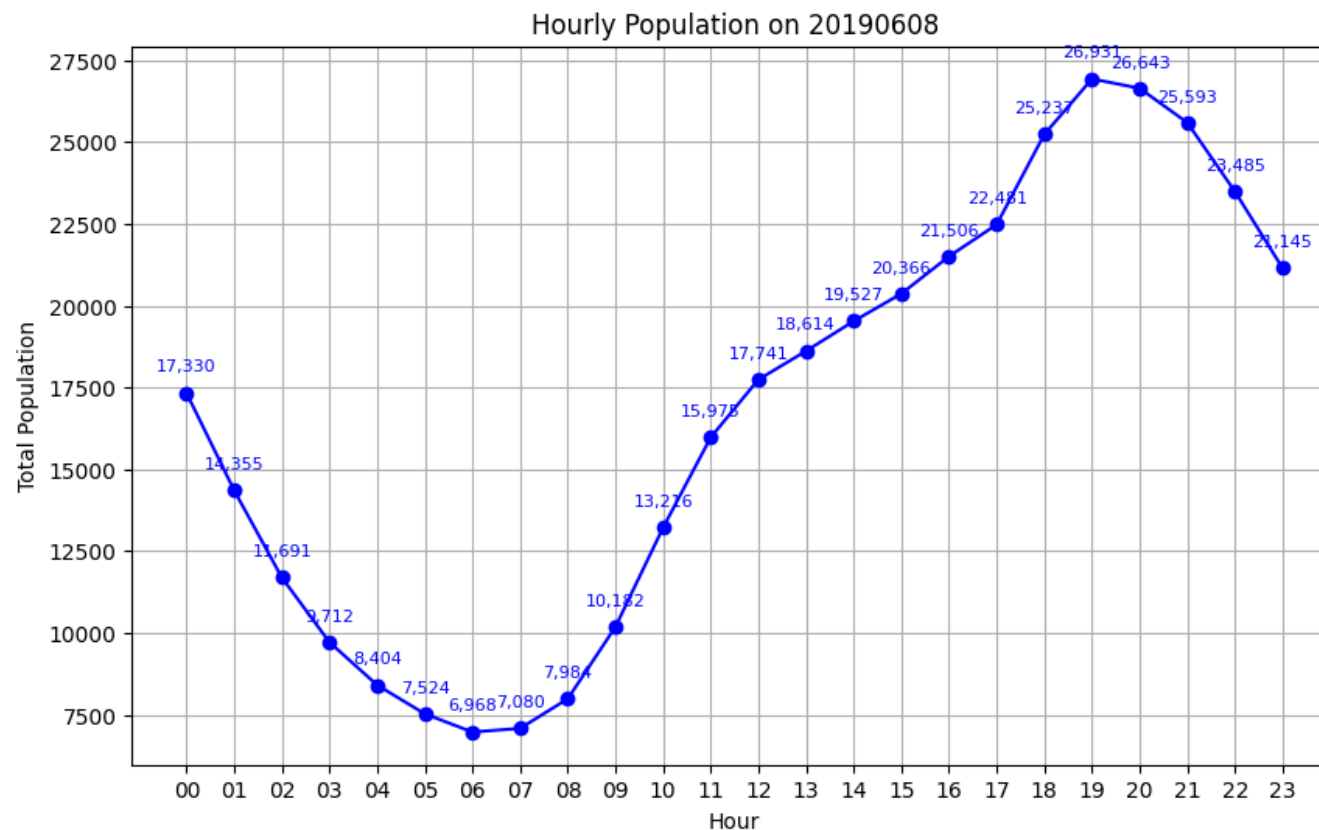
    plt.show()
```

area_analysis.ipynb

国分町（繁華街）の人流データを統計解析

2019年6月8日（コロナ禍以前）の情報の読み取り

```
[4]  
✓ 0 秒  
plot_hourly_population(20190608)
```



area_analysis.ipynb

```

1 def meshcode_to_latlon_center(mesh):
    """
    10桁の4分の1地域メッシュコード(約250m)を
    「メッシュの中心」の緯度・経度(度)に変換する
    """
    code = str(int(mesh)) # 保険で int → str
    code = code.replace("-", "")

    lat = 0.0
    lon = 100.0 # 日本の地域メッシュは経度100度スタート

    # 1次メッシュ(上2桁=緯度*1.5, 下2桁=経度-100)
    if len(code) >= 4:
        lat += int(code[0:2]) / 1.5
        lon += int(code[2:4])

    # 2次メッシュ(5桁目=南から0-7, 6桁目=西から0-7)
    if len(code) >= 6:
        lat += int(code[4]) / 12.0 # 5分 = 1/12度
        lon += int(code[5]) / 8.0 # 7.5分 = 1/8度

    # 3次メッシュ(7桁目=南から0-9, 8桁目=西から0-9)
    if len(code) >= 8:
        lat += int(code[6]) / 120.0 # 30秒 = 1/120度
        lon += int(code[7]) / 80.0 # 45秒 = 1/80度

    # 2分の1メッシュ(9桁目 = 1~4: SW, SE, NW, NE)
    if len(code) >= 9:
        d = int(code[8]) - 1 # 0~3 に変換
        lat += (d // 2) / 240.0 # 15秒 = 1/240度
        lon += (d % 2) / 160.0 # 22.5秒 = 1/160度

    # 4分の1メッシュ(10桁目 = 1~4: SW, SE, NW, NE)
    if len(code) >= 10:
        d = int(code[9]) - 1 # 0~3
        lat += (d // 2) / 480.0 # 7.5秒 = 1/480度
        lon += (d % 2) / 320.0 # 11.25秒 = 1/320度

    # ここまでの lat, lon は「メッシュの南西端 (SW)」
    # 4分の1メッシュのサイズ
    delta_lat = 7.5 / 3600.0 # 7.5秒 ≈ 0.0020833度
    delta_lon = 11.25 / 3600.0 # 11.25秒 ≈ 0.003125度

    lat_center = lat + delta_lat / 2.0
    lon_center = lon + delta_lon / 2.0

    return lat_center, lon_center

```

各エリアの人流推移の視覚化

- メッシュの中心を緯度経度に変換

[8]
0
秒

```

df["lat"], df["lon"] = zip(*df["area"].apply(meshcode_to_latlon_center))
print(df["lat"], df["lon"])

```

```

0      38.261458
1      38.261458
2      38.263542
3      38.265625
4      38.263542

...
52699   38.261458
52700   38.263542
52701   38.265625
52702   38.263542
52703   38.265625
Name: lat, Length: 52704, dtype: float64 0      140.870313
1      140.873438
2      140.867188
3      140.867188
4      140.870313

...
52699   140.873438
52700   140.867188
52701   140.867188
52702   140.870313
52703   140.870313
Name: lon, Length: 52704, dtype: float64

```

area_analysis.ipynb

[9]

```
import plotly.express as px
```

```
def make_daily_animation(date_value):
```

```
    # Step 1. ある日の情報を抽出
```

```
    df1 = df[df["date"] == date_value].copy()
```

```
    if df1.empty:
```

```
        print(f"▲ {date_value} ")
```

```
        return
```

```
    # Step 2: 緯度経度の計算
```

```
    df1["lat"], df1["lon"] = zip(*df1["area"].apply(meshcode_to_latlon_center))
```

```
    # Step 3: 時間ごとの統計
```

```
    total_pop_hour = df1.groupby("time")["population"].sum().to_dict()
```

```
    # Step 4: Heatmap
```

```
    fig = px.density_mapbox(
```

```
        df1,
```

```
        lat="lat",
```

```
        lon="lon",
```

```
        z="population",
```

```
        radius=120,
```

```
        opacity=0.8,
```

```
        animation_frame="time",
```

```
        center={"lat": df1["lat"].mean(), "lon": df1["lon"].mean()},
```

```
        zoom=15,
```

```
        mapbox_style="open-street-map",
```

```
        height=700,
```

```
        range_color=(0, df1["population"].max())
```

```
    )
```

```
    # Step 5: タイトル
```

```
    for frame in fig.frames:
```

```
        hour = int(frame.name)
```

```
        pop = total_pop_hour.get(hour, 0)
```

```
        frame.layout.title = f"{hour//100}:{hour%100:02d} - 総人数: {pop:,}"
```

```
    first_hour = min(total_pop_hour.keys())
```

```
    fig.update_layout(
```

```
        title=f"{first_hour//100}:{first_hour%100:02d} - 総人数: {total_pop_hour[first_hour]:,}"
```

```
    )
```

```
    fig.show()
```

各エリアの人流推移の視覚化

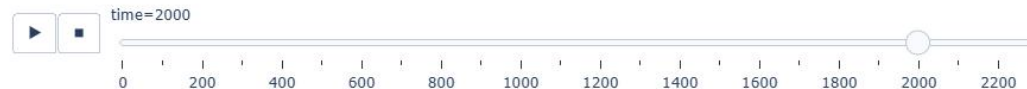
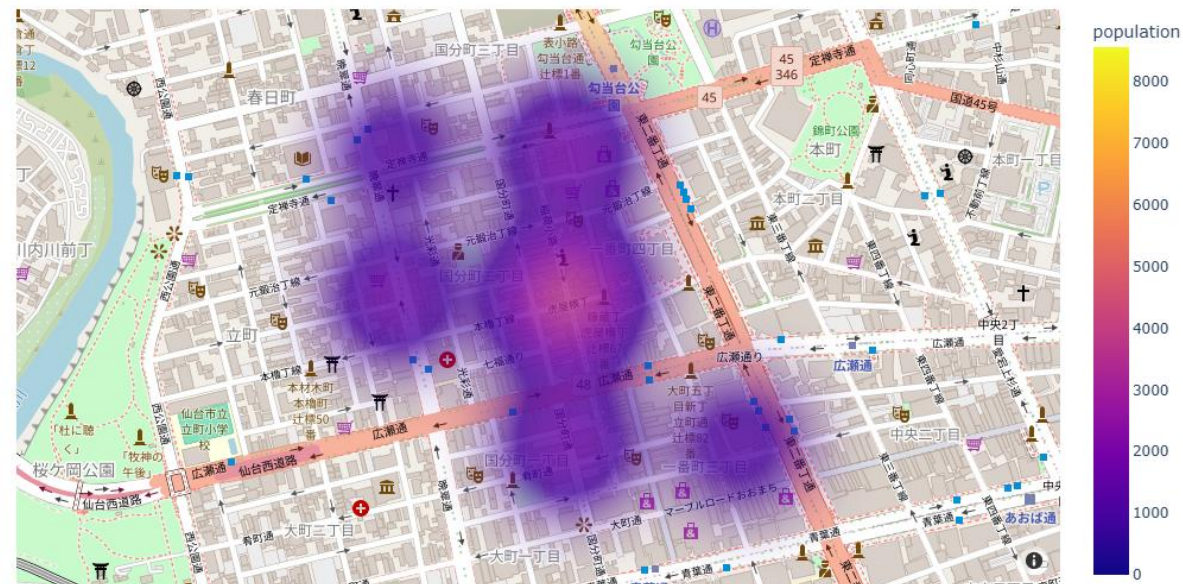
[12]

0 秒

▼

```
make_daily_animation(20190608)
```

20:00 — 総人数: 26,643



area_analysis.ipynb

Agoop ポイント型流動人口データから情報の読み取り

[13]
9
秒

```
# CSVの読み取り
file_path = '/content/drive/MyDrive/Agoop仙台市人流データ/201906/20190608.csv' # CSVファイルの場所
data = pd.read_csv(file_path)
```

国分町（前と同じエリア）の人流情報を抽出

[]

```
df_kokubun = data[
    (
        (data['latitude'].between(38.26245, 38.26665)) &
        (data['longitude'].between(140.8655, 140.8719))
    )
    |
    (
        (data['latitude'].between(38.26035, 38.26245)) &
        (data['longitude'].between(140.8719, 140.8750))
    )
]

print(df_kokubun.head())
```

```
...
12  3a554536111f34f442971fec47dec47dec5c4c93bc63d87... 2019 6 8
78  da50c724b6727b5d2b1c407fd0ac8818b2e21ea64820af... 2019 6 8
102 3a554536111f34f442971fec47dec47dec5c4c93bc63d87... 2019 6 8
173 821f201d2d9cd5834c92abc3b14581aac46aa86366cd3f... 2019 6 8
174 931a8aebb33db9cc97d0295cbec36dc6fac17df3132c3c... 2019 6 8

   dayofweek  hour  minute  latitude  longitude  os  ...  course \
12          6    0        0  38.261210  140.873904  Android  ...  19.983
78          6    0        0  38.262869  140.867404  Android  ...    NaN
102         6    0        0  38.261210  140.873915  Android  ...  270.000
173         6    0        0  38.263607  140.870526  Android  ...  211.759
174         6    0        0  38.262461  140.871198  Android  ...   91.275

   estimated_course_flag  prefcode  citycode  mesh100mid  home_prefcode \
12                    2.0         4    4101.0  5740361939         4.0
78                    NaN         4    4101.0  5740361953         4.0
102                   2.0         4    4101.0  5740361939         4.0
173                   2.0         4    4101.0  5740361966         4.0
174                   2.0         4    4101.0  5740361946         4.0

   home_citycode  workplace_prefcode  workplace_citycode  gender
12          4101.0                4.0                4101.0    m
78          4101.0                4.0                4101.0    m
102         4101.0                4.0                4101.0    m
173         4104.0                4.0                4101.0    m
174         4102.0                4.0                4101.0    f
```

[5 rows x 31 columns]

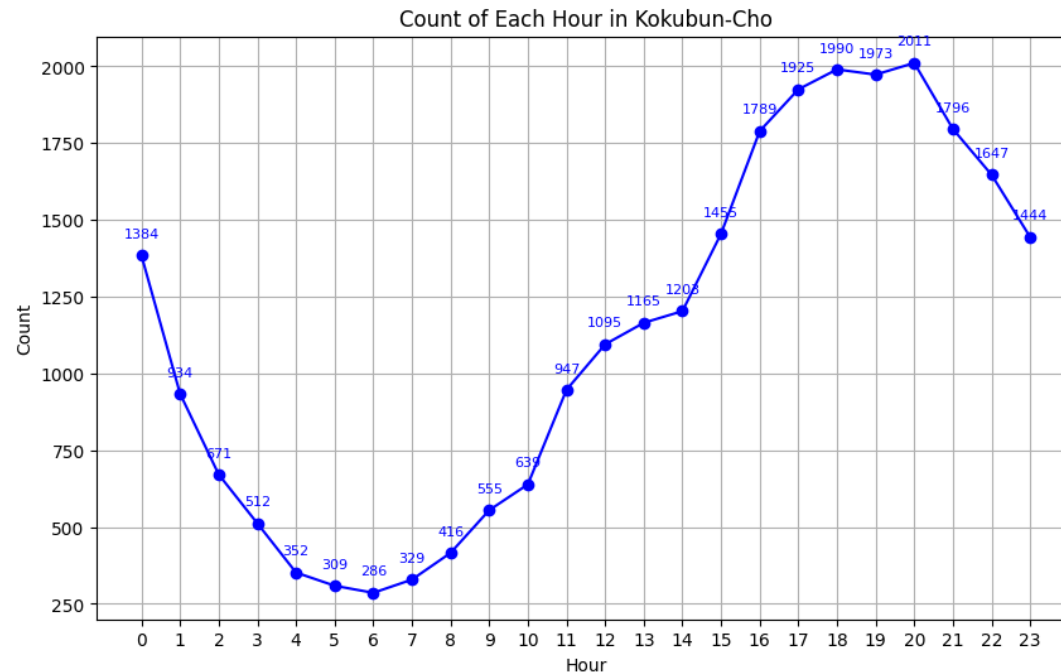
国分町（繁華街）の人流データを統計解析

時間ごとで統計

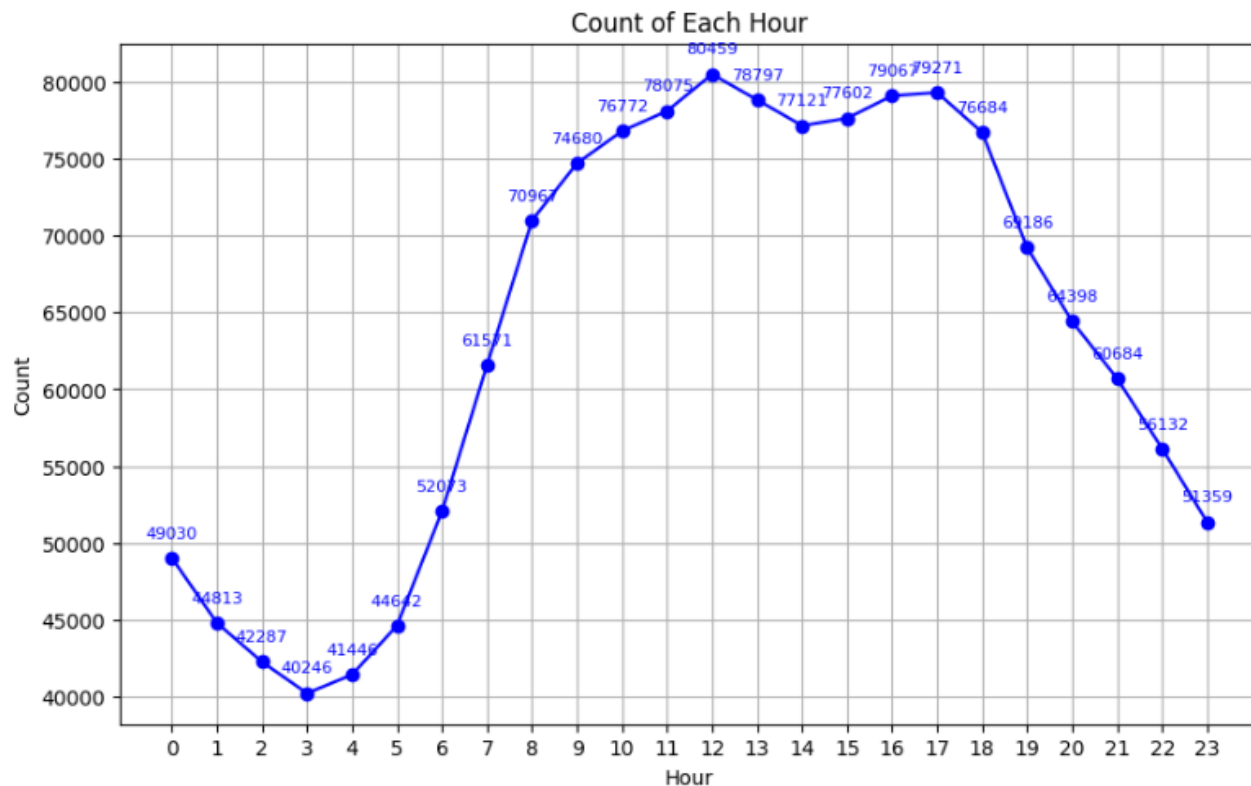
#時間ごとの統計

```
all_hours = pd.Series(range(24))
koku_count = df_kokubun['hour'].value_counts().sort_index(ascending=True)
koku_count = koku_count.reindex(all_hours, fill_value=0)
```

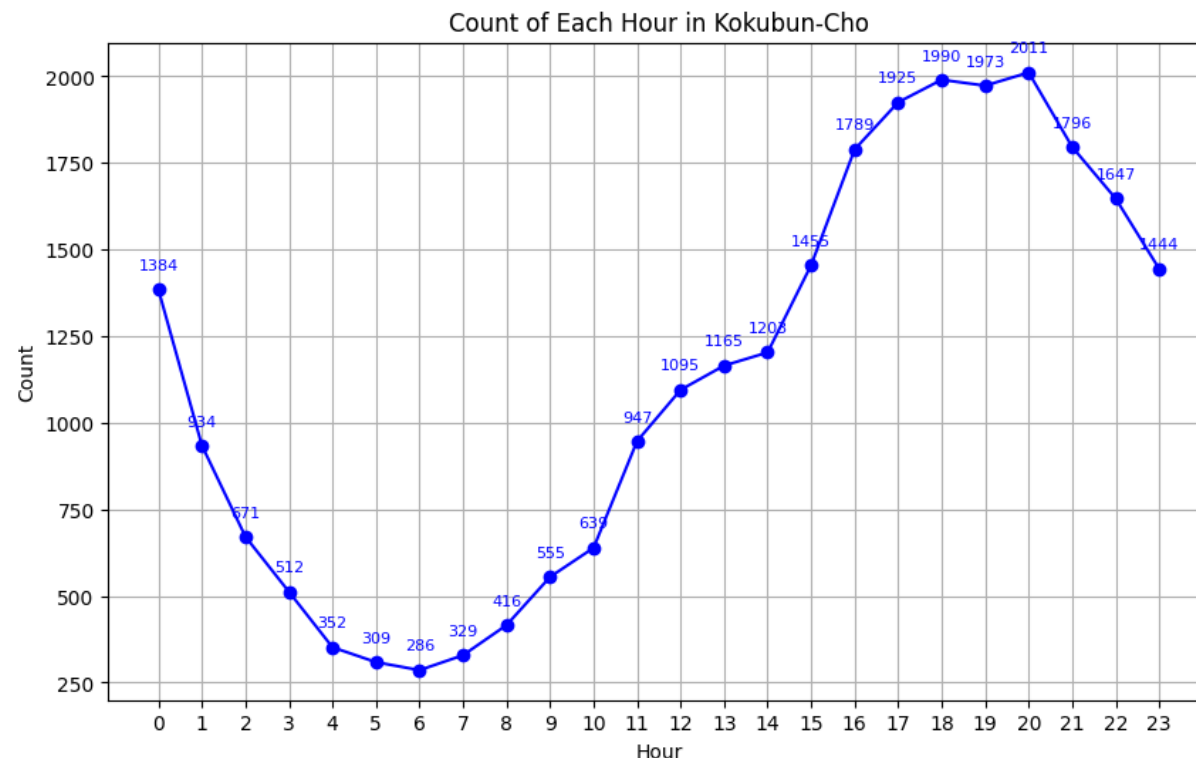
```
plt.figure(figsize=(10, 6))
plt.plot(koku_count.index, koku_count.values, marker='o', linestyle='-', color='b')
for i, value in enumerate(koku_count.values):
    plt.annotate(f'{value}', (koku_count.index[i], value),
                 textcoords="offset points", xytext=(0, 10), ha='center',
                 fontsize=8, color='b')
plt.xlabel('Hour')
plt.ylabel('Count')
plt.title('Count of Each Hour in Kokubun-Cho')
plt.grid(True)
plt.xticks(koku_count.index)
plt.show()
```



仙台市 (Agoop)

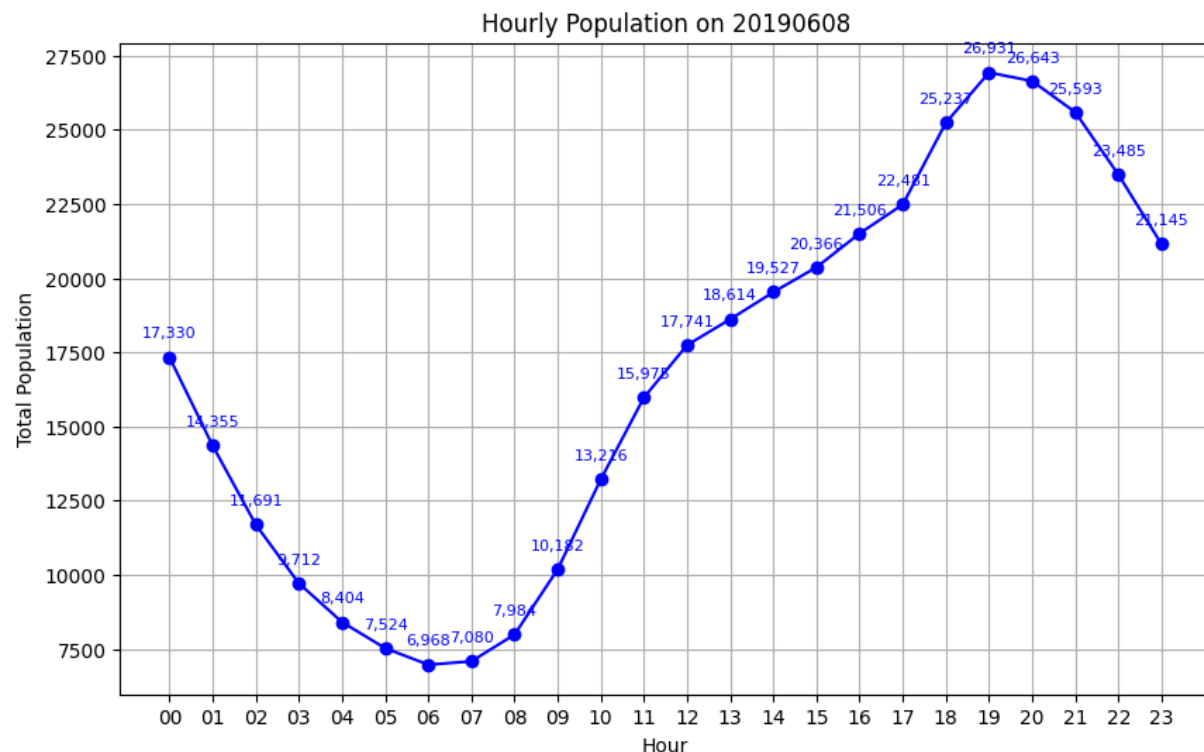


国分町 (Agoop)

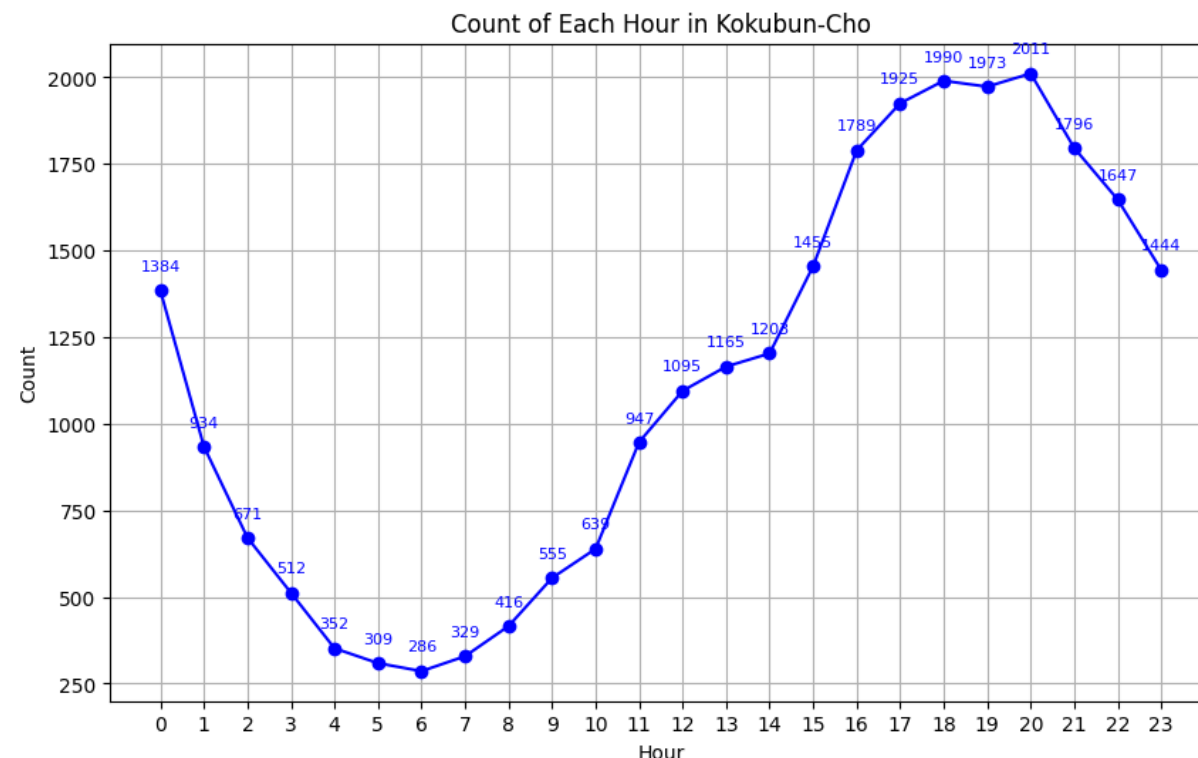


仙台市全体での結果(左図)とは異なり、国分町 (右図) は日中よりも夜間の人流が増大する。国分町は夜間の活動が盛んなエリアであることがわかる。

国分町（モバイル空間データ）



国分町（Agoop）



両データの時間的トレンドは類似しているが、絶対人数には大きな差が見られる。これは、Agoop社（右図）のデータが拡大推計を行っていないため、母集団が限定されていることに起因すると考えられる。

相関係数の計算

インデックスの確認

```
[13] print(population_dcm.index.tolist())
print(koku_count.index.tolist())

[0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
```

インデックスを整合させた2つのデータを用いて、相関係数を算出

```
[18] 0 秒
✓ population_dcm.index = koku_count.index
combined = pd.DataFrame({
    'dcm': population_dcm,
    'agoop': koku_count
}).dropna()

corr = combined['dcm'].corr(combined['agoop'])
print(f"相関係数: {corr:.4f}")

... 相関係数: 0.9803
```

相関係数が0.98と非常に高い値を示したことから、時間帯ごとの人流の変動傾向がほぼ一致していることが確認された。これにより、両データソースの計測結果に高い整合性があると言える。

国分町の人流散布図マップを作成

```
[10] grouped_data = df_kokubun.groupby('hour')
```

人数が少ないため、時間ごとで表示

```
# 総時間数をキー、その時間内にすべてのデバイスの位置リストを値とする辞書を作成する
```

```
locations_per_hour = {hour: group[['longitude', 'latitude']].values.tolist() for hour, group in grouped_data}
```

```
locations_per_hour_vlaue = {hour: group[['latitude', 'longitude']].values for hour, group in grouped_data}
```

```
!pip install dash==2.14.2
```

Dashをインストール

```
from dash import Dash, html
```

```
import dash_core_components as dcc
```

```
from dash.dependencies import Input, Output
```

```
import plotly.graph_objs as go
```

```
max_total_hours = df_kokubun['hour'].max()
```

```
# Dashアプリケーションの作成
```

```
app = Dash(__name__)
```

```
# アプリケーションのレイアウト
```

```
app.layout = html.Div([
```

```
    dcc.Graph(id='scatter-map', style={'width': '80vw', 'height': '80vh'}),
```

```
    dcc.Interval(id='interval-component', interval=1000, n_intervals=0), # 1秒ごとに更新
```

```
])
```

```
koku_lat_min=38.26035
```

```
koku_lat_max=38.26665
```

```
koku_lon_min=140.8655
```

```
koku_lon_max=140.8750
```

```
@app.callback(
    Output('scatter-map', 'figure'),
    Input('interval-component', 'n_intervals')
)
def update_graph(n):
    hour = n % (max_total_hours + 1) # ループ再生
    locations = locations_per_hour.get(hour, [])

    # 経緯度データ
    latitudes = [loc[1] for loc in locations]
    longitudes = [loc[0] for loc in locations]

    # 中心点を計算
    center_lat = (koku_lat_min + koku_lat_max) / 2
    center_lon = (koku_lon_min + koku_lon_max) / 2

    # 散布図マップの作成
    scattermap = go.Scattermapbox(
        lat=latitudes,
        lon=longitudes,
        mode='markers',
        marker=dict(size=3, color='blue', opacity=0.6)
    )

    # 地図のレイアウトを設定し、OpenStreetMapを使用する
    layout = go.Layout(
        title=f'Time: {hour :02d}:00',
        autosize=True,
        hovermode='closest',
        mapbox=dict(
            style='open-street-map', # OpenStreetMap
            bearing=0,
            center=dict(lat=center_lat, lon=center_lon), # 地図の中心点を設定する
            pitch=0,
            zoom=15,
            bounds=dict( # 表示範囲を緯度経度の最大最小で設定
                west=koku_lon_min-0.004,
                east=koku_lon_max+0.004,
                south=koku_lat_min-0.004,
                north=koku_lat_max+0.004
            )
        )
    ),
)

return {'data': [scattermap], 'layout': layout}

if __name__ == '__main__':
    app.run_server(mode="inline", debug=False)
```

国分町のエリアを表示

Time: 10:00

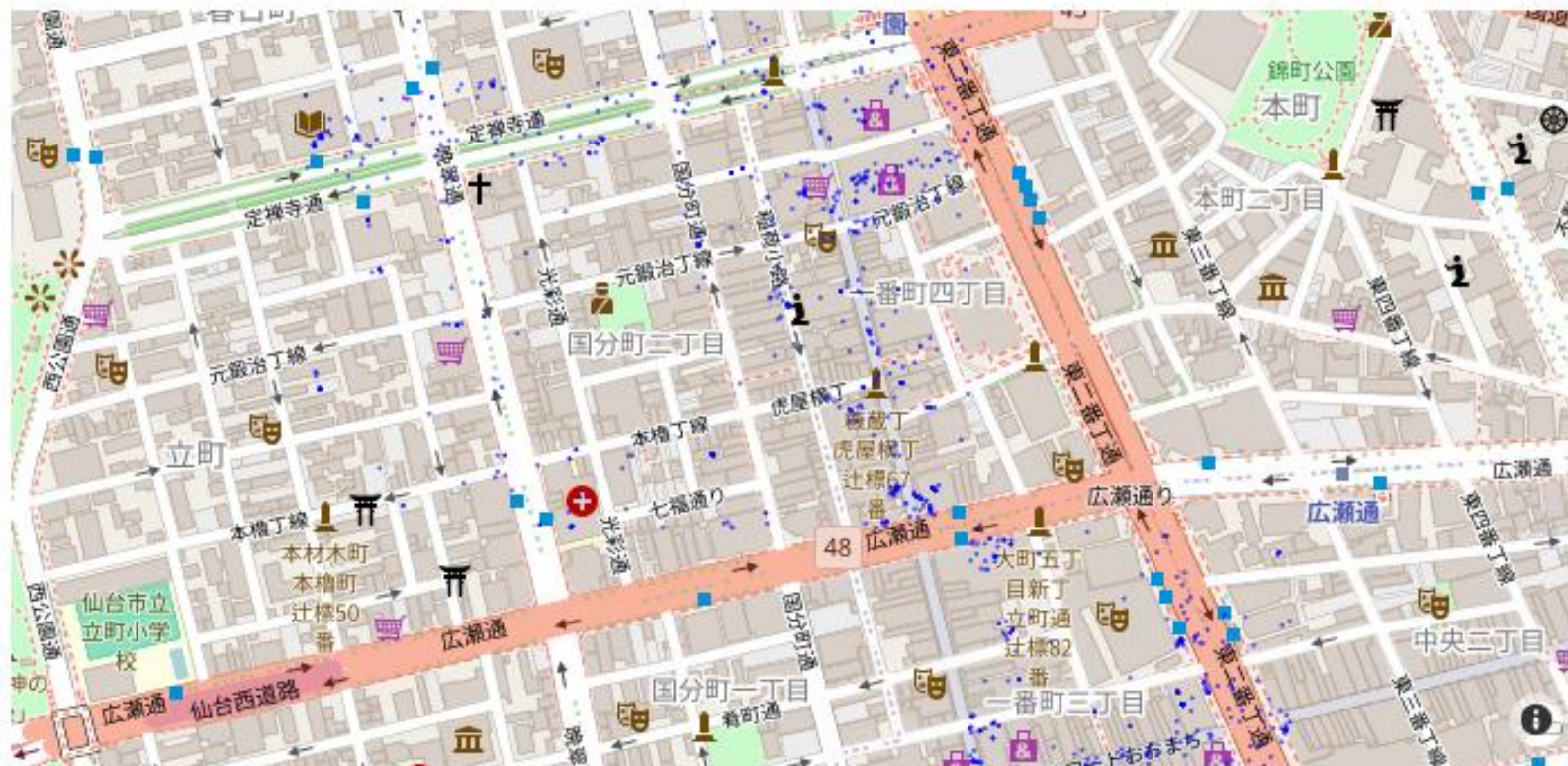


Time: 20:00



20時台には地図上に多くの青い点を確認できる。これは、この時間帯は多くの人々が外で食事や遊びを楽しんでいることを示唆している。

Time: 12:00



昼の12時台では、青い点が大半は道路上に分布している。これは、この時間帯の当該エリアは、目的地ではなく移動経路として利用されていることを示唆している。

モバイル空間統計データ

- ✓ サンプル規模が大きい（ほぼ全契約者をカバー）
- ✓ 拡大推計により実人口に近い値を推定可能
- ◆ 地理解像度は比較的低い（基地局単位）
- ◆ 個々人の詳細な移動経路は追跡できない

Agoop ポイント型流動人口データ

- ✓ 地理解像度が高い（数m～数十mレベル）
- ✓ 詳細な移動経路の分析が可能
- ✓ 微視的な人流パターンの把握に適する
- ◆ サンプル規模が小規模
- ◆ GPSアプリ利用者に限定される
- ◆ 利用状況に依存するためサンプルに偏りがある



目次

人流データの紹介	 P5
基礎的な解析	 P12
繁華街の人流解析	 P28
エリア間の人流解析	 P45

滞留人数

時間帯毎に同じエリアに留まる人々の数である。滞留データからは、人々がどのエリアに集まりやすいか、また時間帯毎の混雑状況を把握することができる。

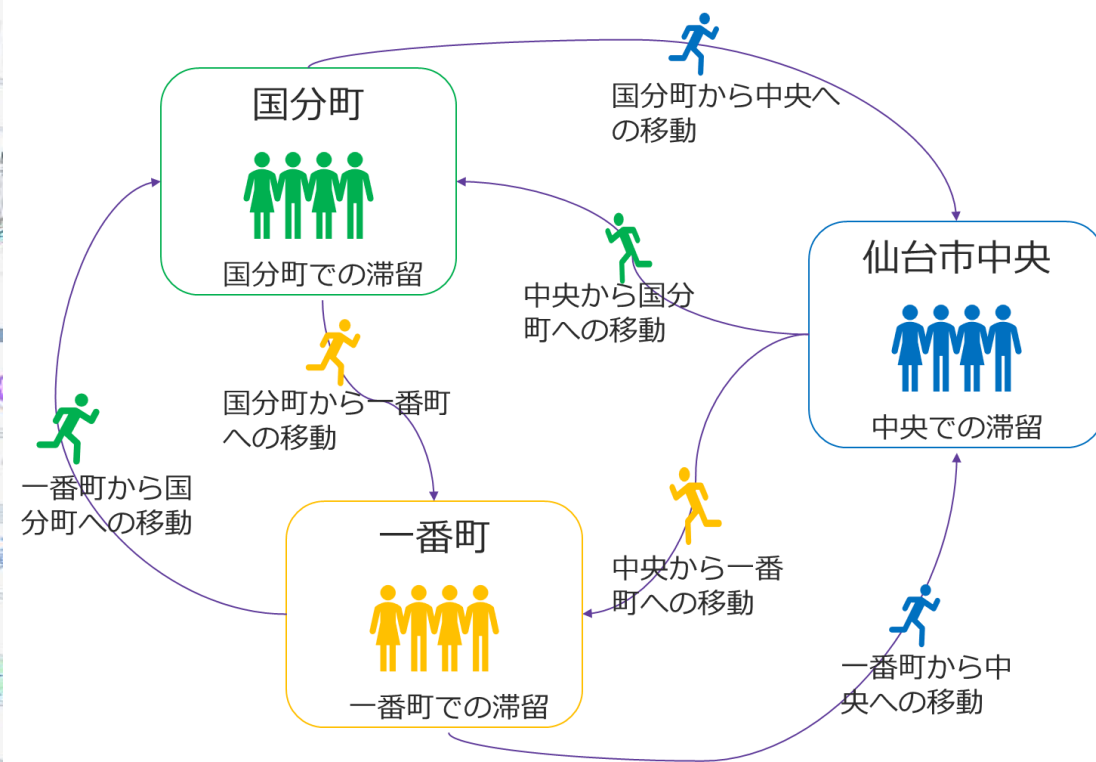
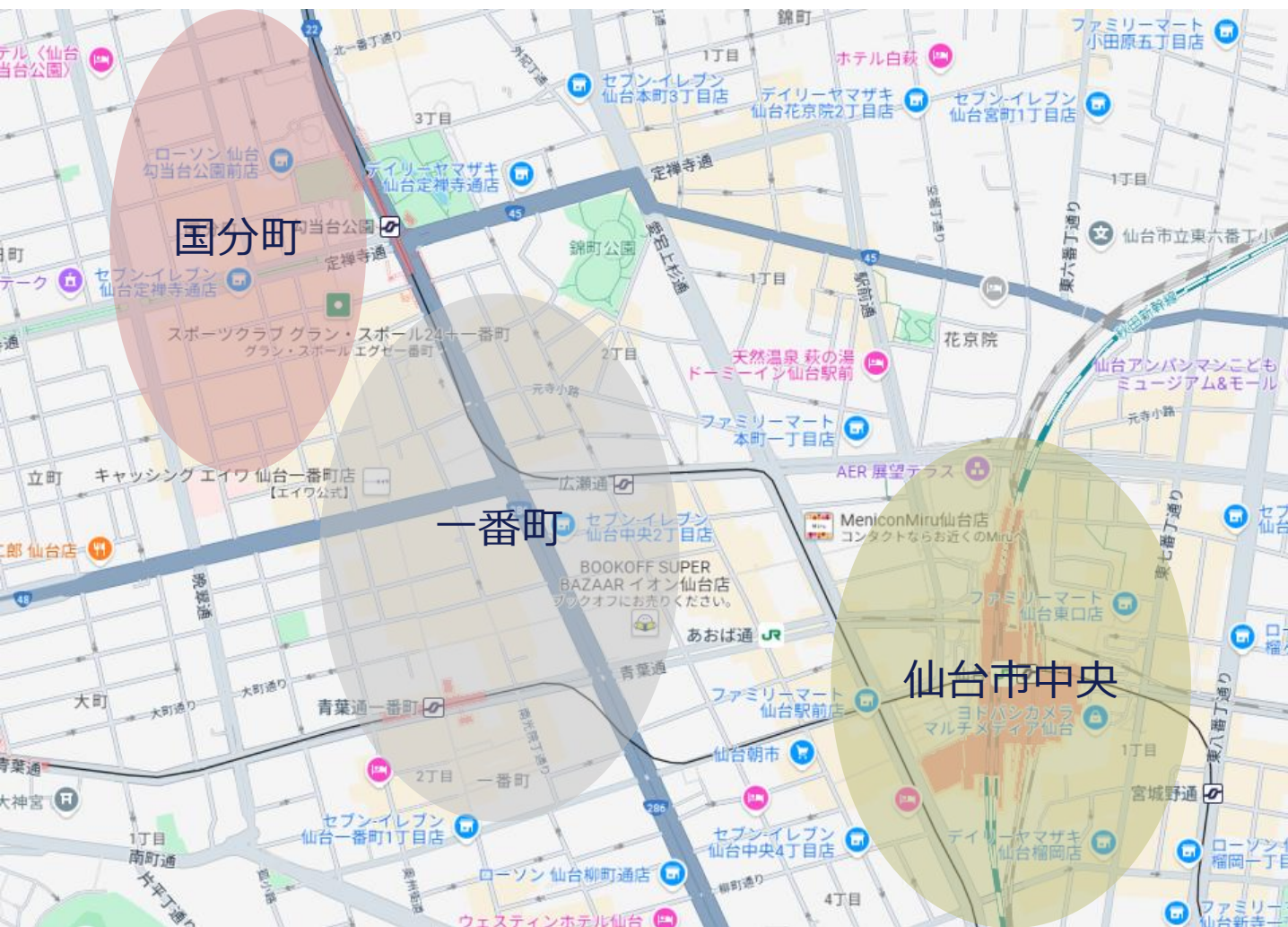
移動人数

時間帯毎に異なるエリア間を移動する人々の数である。時間帯毎・エリア間毎の移動のパターンを解析することで、どの時間帯にどのエリア間の移動が活発か、交通の流れを明らかにすることができる。

各エリアにおける滞留と、各エリア間の移動を解析



国分町、仙台市中央、一番町の各エリアにおける滞留と、各エリア間の移動を解析



stop_move.ipynb

データを読み取り

```
[1] 18 秒
✓ from google.colab import drive
  drive.mount('/content/drive')

... Mounted at /content/drive

[2] 13 秒
✓ import pandas as pd

# CSVの読み取り
file_path = '/content/drive/MyDrive/人流データ解析実践/201906/20190608.csv'
df = pd.read_csv(file_path)
```

緯度と経度に基づいて国分町、一番町、仙台市中央の各エリアの人流をリスト化

```
region_koku = {'lat_min': 38.26031, 'lat_max': 38.26872, 'lon_min': 140.86825, 'lon_max': 140.87000}
region_ichi = {'lat_min': 38.25633, 'lat_max': 38.26872, 'lon_min': 140.87000, 'lon_max': 140.87354}
region_chuo = {'lat_min': 38.25633, 'lat_max': 38.26282, 'lon_min': 140.87354, 'lon_max': 140.88391}

def get_region_from_coordinates(lat, lon):
    if (lat >= region_koku['lat_min']) and (lat <= region_koku['lat_max']) \
        and (lon >= region_koku['lon_min']) and (lon <= region_koku['lon_max']):
        return 'region_koku'
    elif (lat >= region_ichi['lat_min']) and (lat <= region_ichi['lat_max']) \
        and (lon >= region_ichi['lon_min']) and (lon <= region_ichi['lon_max']):
        return 'region_ichi'
    elif (lat >= region_chuo['lat_min']) and (lat <= region_chuo['lat_max']) \
        and (lon >= region_chuo['lon_min']) and (lon <= region_chuo['lon_max']):
        return 'region_chuo'
    else:
        return None
```

stop_move.ipynb

各時間・各エリアの滞留人数の集計

滞留データ:

	hour	region	stay_count
0	0	region_chuo	147
1	0	region_ichi	74
2	0	region_koku	83
3	1	region_chuo	112
4	1	region_ichi	54
..	...	...	...
67	22	region_ichi	91
68	22	region_koku	89
69	23	region_chuo	158
70	23	region_ichi	80
71	23	region_koku	71

[72 rows x 3 columns]

```
# エリア情報の追加
df['region'] = df.apply(lambda row: get_region_from_coordinates(row['latitude'], row['longitude']), axis=1)

# データの時間順ソート
df = df.sort_values(by=['dailyid', 'hour', 'minute'])

# 各 dailyid における時間ごとのエリア数の集計
region_counts = df.groupby(['dailyid', 'hour'])['region'].nunique().reset_index(name='unique_regions')

# 単一エリア内のレコードをフィルタリング
stay_df = region_counts[region_counts['unique_regions'] == 1]

# 滞留エリアの情報の統合:
stay_df = pd.merge(stay_df, df[['dailyid', 'hour', 'region']], on=['dailyid', 'hour'], how='left').drop_duplicates(subset=['dailyid', 'hour'])

# 各時間ごとのエリアの滞留人数の集計
stay_counts = stay_df.groupby(['hour', 'region'])['dailyid'].count().reset_index(name='stay_count')

print("滞留データ:")
print(stay_counts)
```

stop_move.ipynb

各エリアの滞留人数を折れ線グラフで表示

```
import matplotlib.pyplot as plt

# 各エリアの滞留人数を折れ線グラフで表示
plt.figure(figsize=(12, 6))

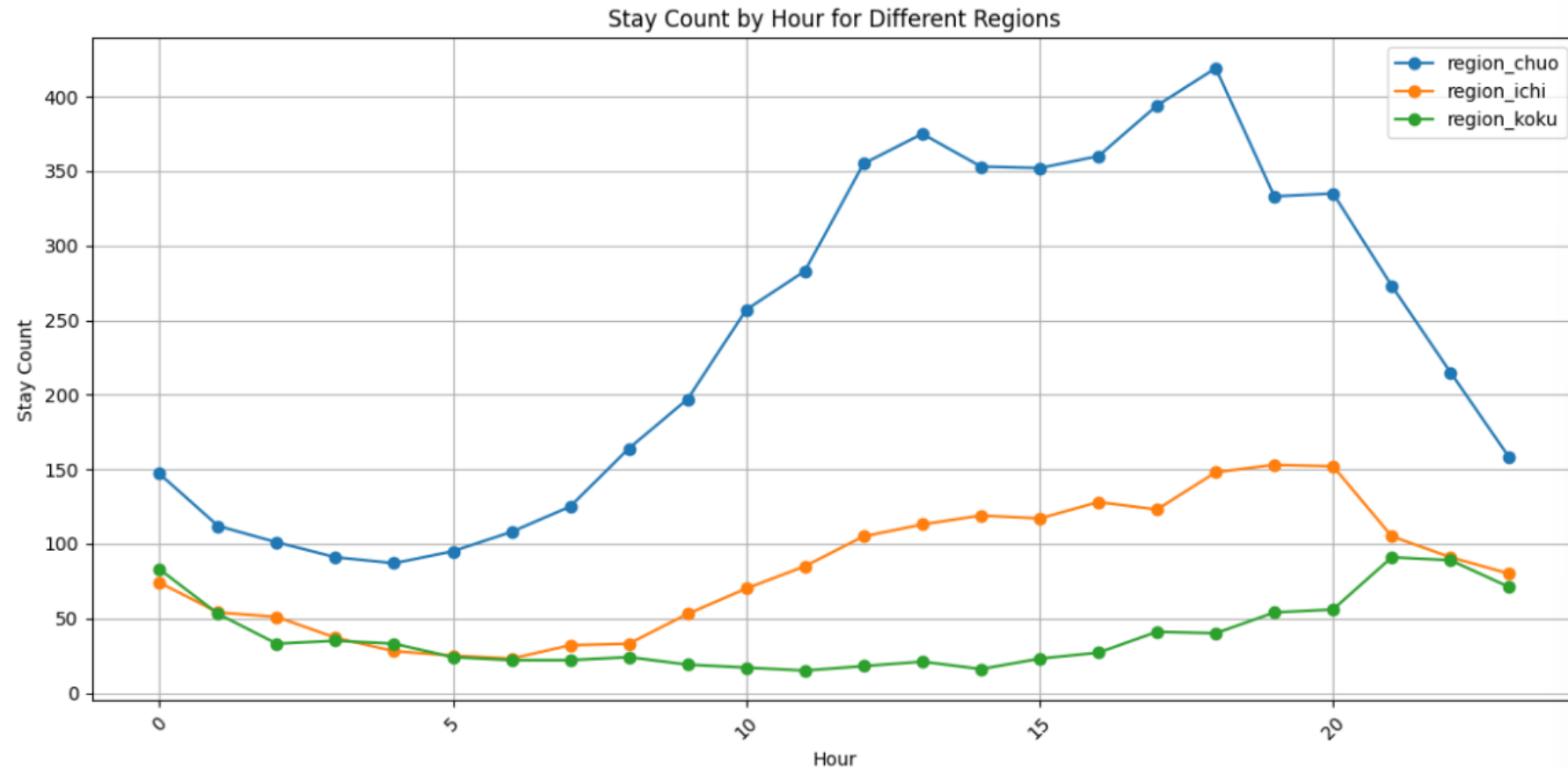
for region in stay_counts['region'].unique():
    region_data = stay_counts[stay_counts['region'] == region]
    plt.plot(region_data['hour'], region_data['stay_count'], marker='o', label=region)

plt.xlabel('Hour')
plt.ylabel('Stay Count')
plt.title('Stay Count by Hour for Different Regions')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```

stop_move.ipynb

滞留人数の折れ線グラフ



仙台市中央 ([region_chuo](#)) : 12時から18時の間が混雑 (推測される原因: 昼食やショッピング)

一番町 ([region_ichi](#)) : 18時から20時の間が混雑 (推測される原因: 飲食店やショッピング)

国分町 ([region_koku](#)) : 21時から0時の間が混雑 (推測される原因: 飲み会や遊び)

stop_move.ipynb

エリア間の移動人数の集計

```
# 各 `dailyid` の毎時内での前のエリアを計算
df['prev_region'] = df.groupby('dailyid')['region'].shift(1)

# 統計範囲外の移動をフィルタリング ('region_other' の移動は統計しない)
movement_df = df[df['region'].isin(['region_koku', 'region_ichi', 'region_chuo']) &
                  df['prev_region'].isin(['region_koku', 'region_ichi', 'region_chuo'])]

# エリア間の移動量を統計
movement_counts = movement_df.groupby(['hour', 'prev_region', 'region']).size().reset_index(name='movement_count')

print("移動データ:")
print(movement_counts)
```

移動データ:

	hour	prev_region	region	movement_count
0	0	region_chuo	region_chuo	1089
1	0	region_chuo	region_ichi	24
2	0	region_chuo	region_koku	2
3	0	region_ichi	region_chuo	23
4	0	region_ichi	region_ichi	627
...	...	...	...	...
200	23	region_ichi	region_ichi	757
201	23	region_ichi	region_koku	39
202	23	region_koku	region_chuo	6
203	23	region_koku	region_ichi	39
204	23	region_koku	region_koku	713

[205 rows x 4 columns]

滞留と移動データを折れ線グラフで表示

stop_move.ipynb

```
# 滞留データの折れ線グラフを描画
regions = stay_counts['region'].unique()
colors = {
    'region_chuo': 'blue',
    'region_ichi': 'orange',
    'region_koku': 'green'
}

for region in regions:
    plt.figure(figsize=(14, 7))
    region_data = stay_counts[stay_counts['region'] == region]
    plt.plot(region_data['hour'], region_data['stay_count'], color=colors[region], marker='o', label=f'{region}_stop')

    movement_counts_filtered = movement_counts[(movement_counts['prev_region'] != movement_counts['region']) \
                                                & (movement_counts['prev_region'] == region)]
    pairs = movement_counts_filtered[['prev_region', 'region']].drop_duplicates()

    # 移動データ
    for _, pair in pairs.iterrows():
        prev_region = pair['prev_region']
        curr_region = pair['region']

        # 各ペアのエリア間の移動データを取得
        pair_data = movement_counts_filtered[
            (movement_counts_filtered['prev_region'] == prev_region) &
            (movement_counts_filtered['region'] == curr_region)
        ]

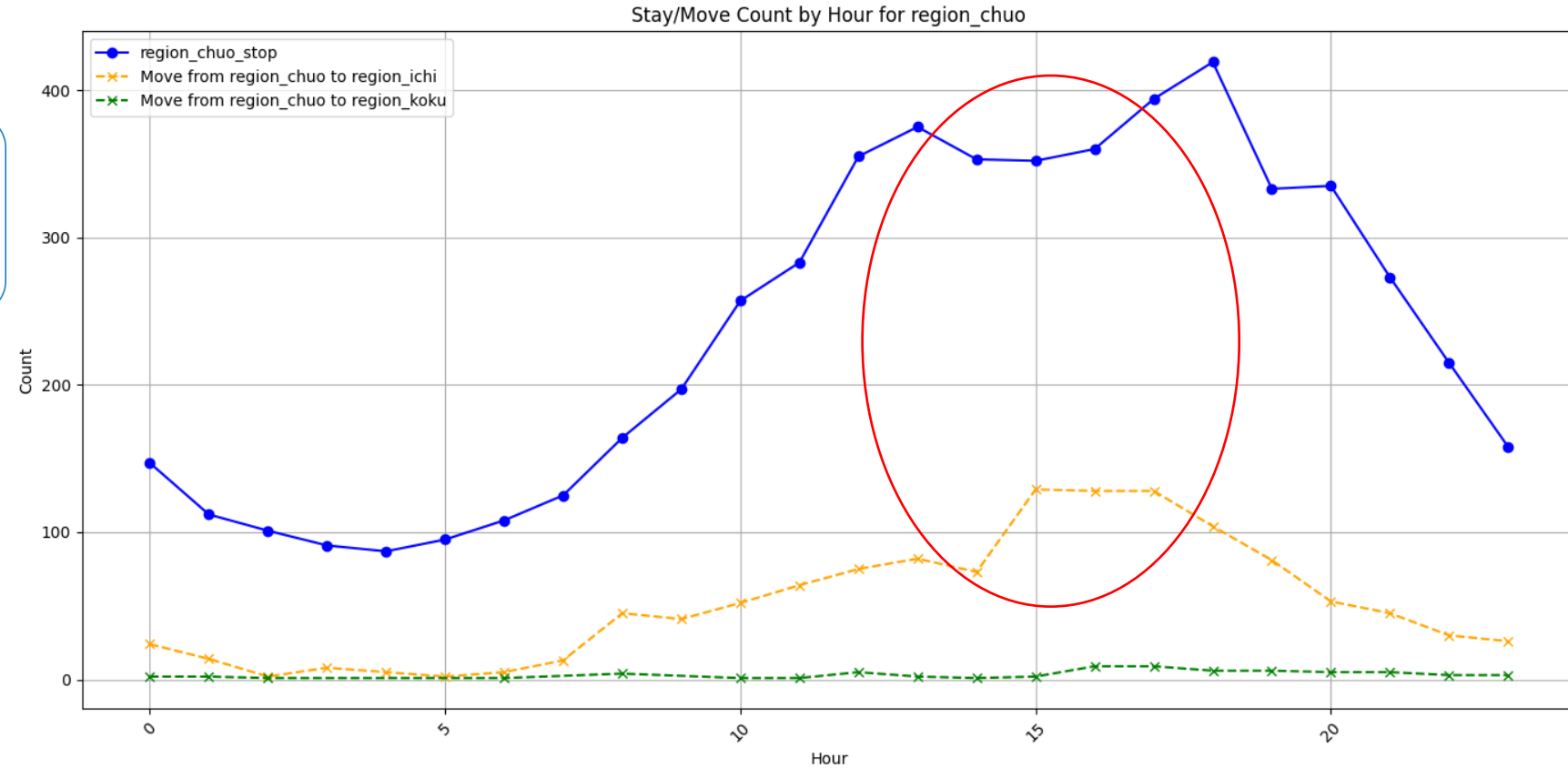
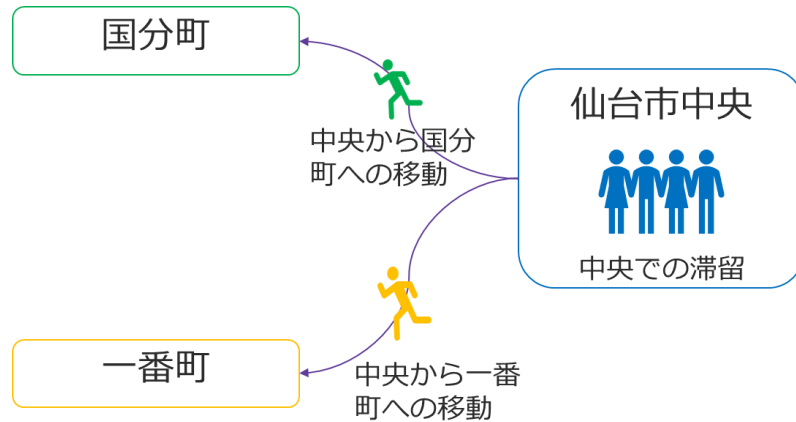
        plt.plot(pair_data['hour'], pair_data['movement_count'], marker='x', \
                 linestyle='—', color=colors[curr_region], label=f'Move from {prev_region} to {curr_region}')

plt.xlabel('Hour')
plt.ylabel('Count')
plt.title(f'Stay/Move Count by Hour for {prev_region}')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()

# グラフを表示
plt.show()
```

stop_move.ipynb

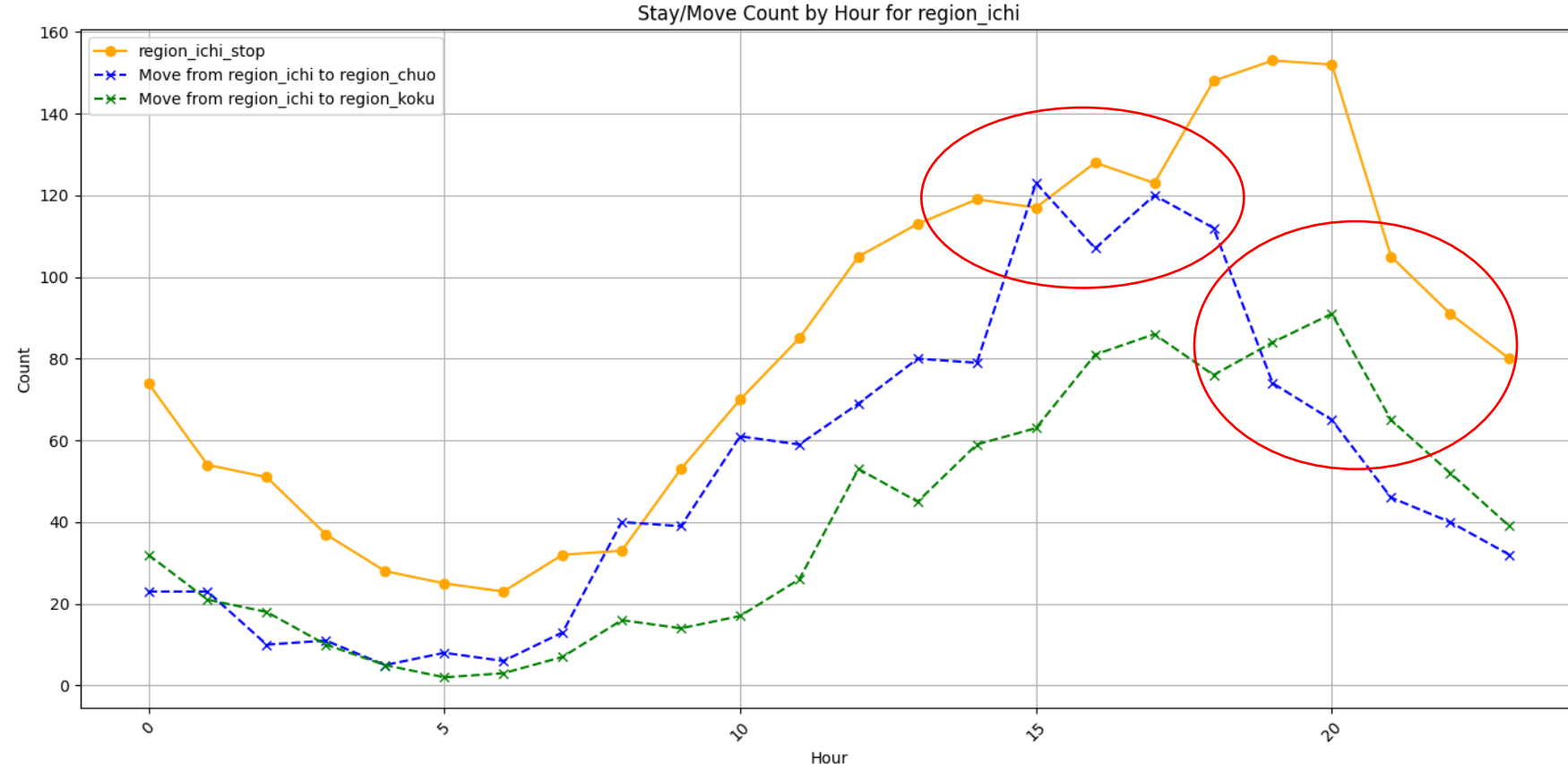
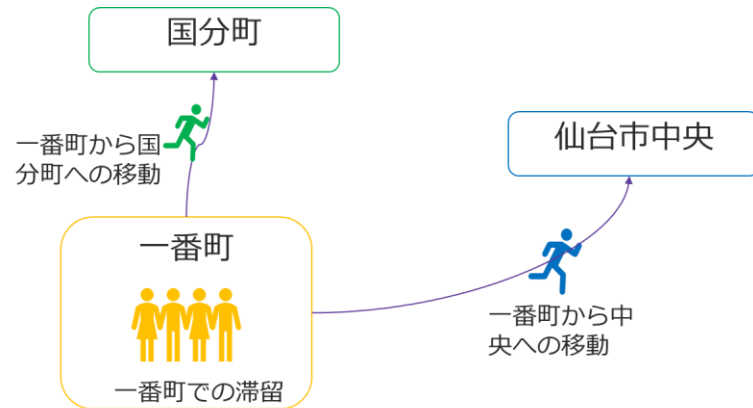
滞留と移動の折れ線グラフ（仙台市中央）



昼食後の時間帯に一番町への移動が増加していることは、周辺の飲食店やカフェでの食事を終えた人々が、買い物や観光を目的として移動していることを示唆している。

stop_move.ipynb

滞留と移動の折れ線グラフ（一番町）

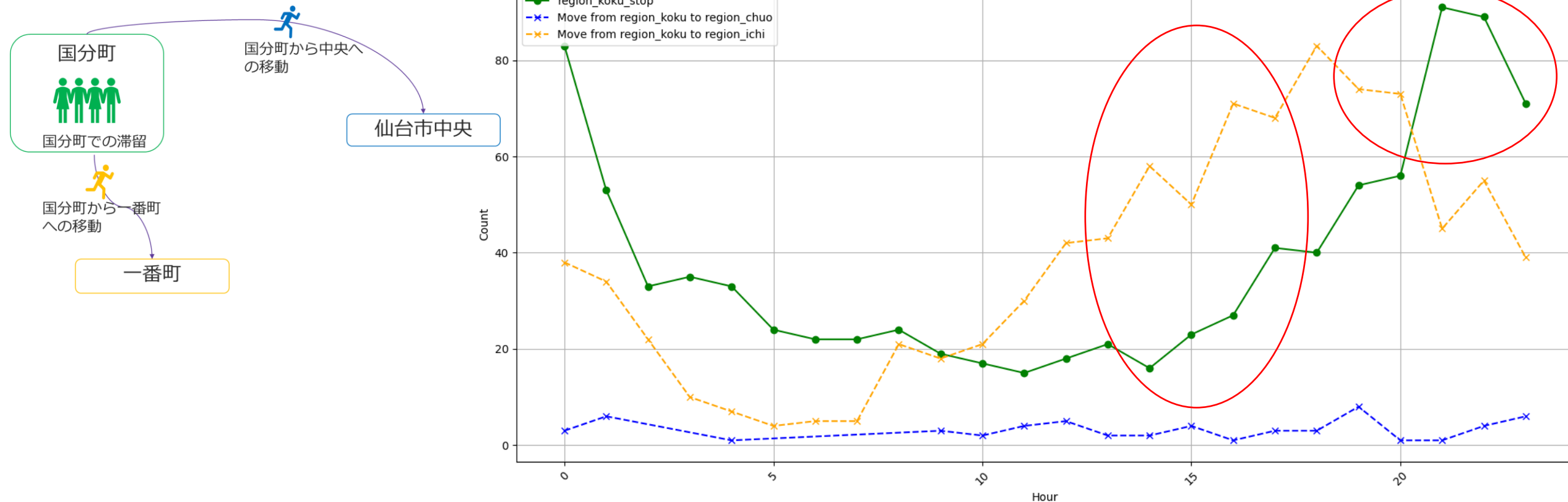


15時から18時にかけて、一番町から仙台市中央への人移動が見られる。しかし、一番町の人数が減少しないことは、一番町での活動や交流が続いていることを示唆している。

17時から21時にかけて、国分町への移動が増加している。これは、夕食や飲み会を目的に人々が国分町に移動していることを示唆している。

stop_move.ipynb

滞留と移動の折れ線グラフ（国分町）



国分町は多くの飲食店やバー、クラブなどが集中しているため、17時以降に人々が集まる傾向が強い。日中に、一番町に向かう人の増加するが、滞留人数は大きく減っていない。これは、国分町を通過する人数が多いことを示唆している。

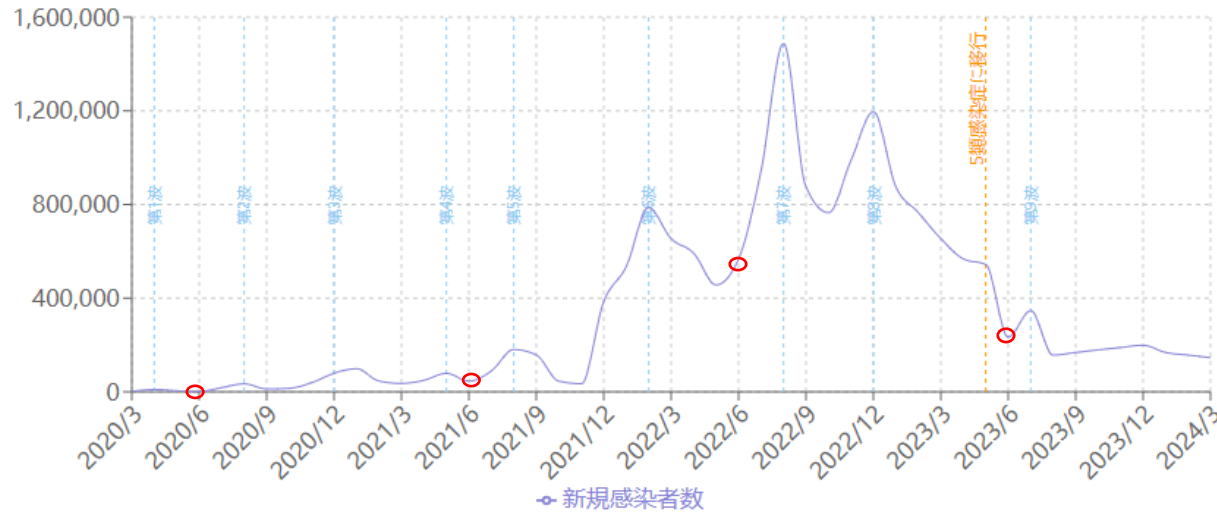


目次

人流データの紹介	...	P5
基礎的な解析	...	P12
繁華街の人流解析	...	P28
エリア間の人流解析	...	P45
練習問題例	...	P57

2019年～2024年の各年の6月のデータを比較して、コロナ禍以前、コロナ禍、コロナ禍後の人流がどのように変化したかを考察してください。

- 全体の人流・毎時人流の推移の視覚化・考察
- 繁華街人流の推移の視覚化・考察



データ出典:

- 厚生労働省「オープンデータ」<https://www.mhlw.go.jp/stf/covid-19/open-data.html>
- 2023年5月8日以降: 厚生労働省「感染症発生動向調査」

緊急事態宣言・まん延防止等重点措置（宮城県）

1回目: 2020/4/16～2020/5/14

2回目: 2021/4/5～2021/5/11

3回目: 2021/8/20～2021/9/30

制限緩和

2022年3月1日、外国人の新規入国制限の緩和措置を実施

2022年10月11日、入国者総数の上限が撤廃

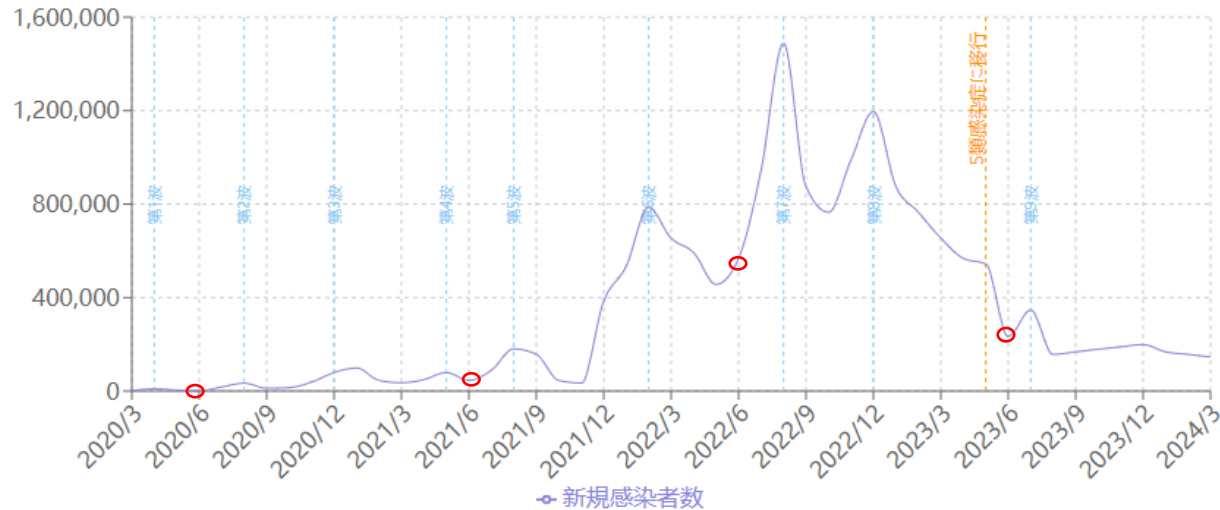
The background of the slide is a solid dark purple color. It is decorated with a repeating pattern of stylized, light purple leaves and branches. The leaves are simple, rounded shapes, and the branches are thin, curved lines. The pattern is distributed across the entire slide, with some branches curving around the edges.

Pythonによる人流データ解析と視覚化

練習問題例の解答例

2019年～2024年の各年の6月のデータを比較して、コロナ禍以前、コロナ禍、コロナ禍後の人流がどのように変化したかを考察してください。

- 全体の人流・毎時人流の推移の視覚化・考察
- 繁華街人流の推移の視覚化・考察



データ出典:

- 厚生労働省「オープンデータ」<https://www.mhlw.go.jp/stf/covid-19/open-data.html>
- 2023年5月8日以降：厚生労働省「感染症発生動向調査」
- 新型コロナウイルス感染症の発生状況及び対応状況について <https://www.pref.miyagi.jp/documents/37399/20220228.pdf>

緊急事態宣言・まん延防止等重点措置（宮城県）

1回目：2020/4/16～2020/5/14

2回目：2021/4/5～2021/5/11

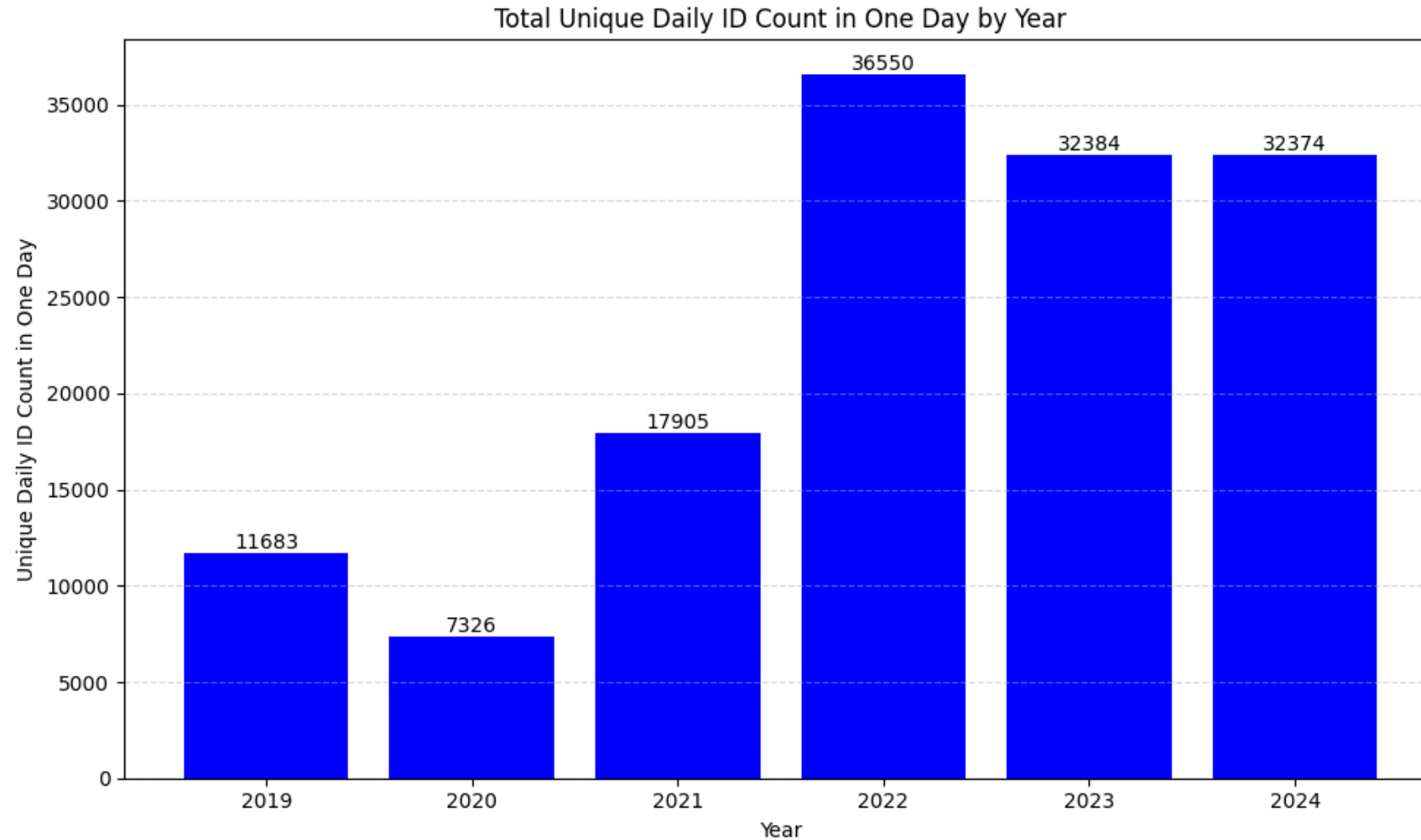
3回目：2021/8/20～2021/9/30

制限緩和

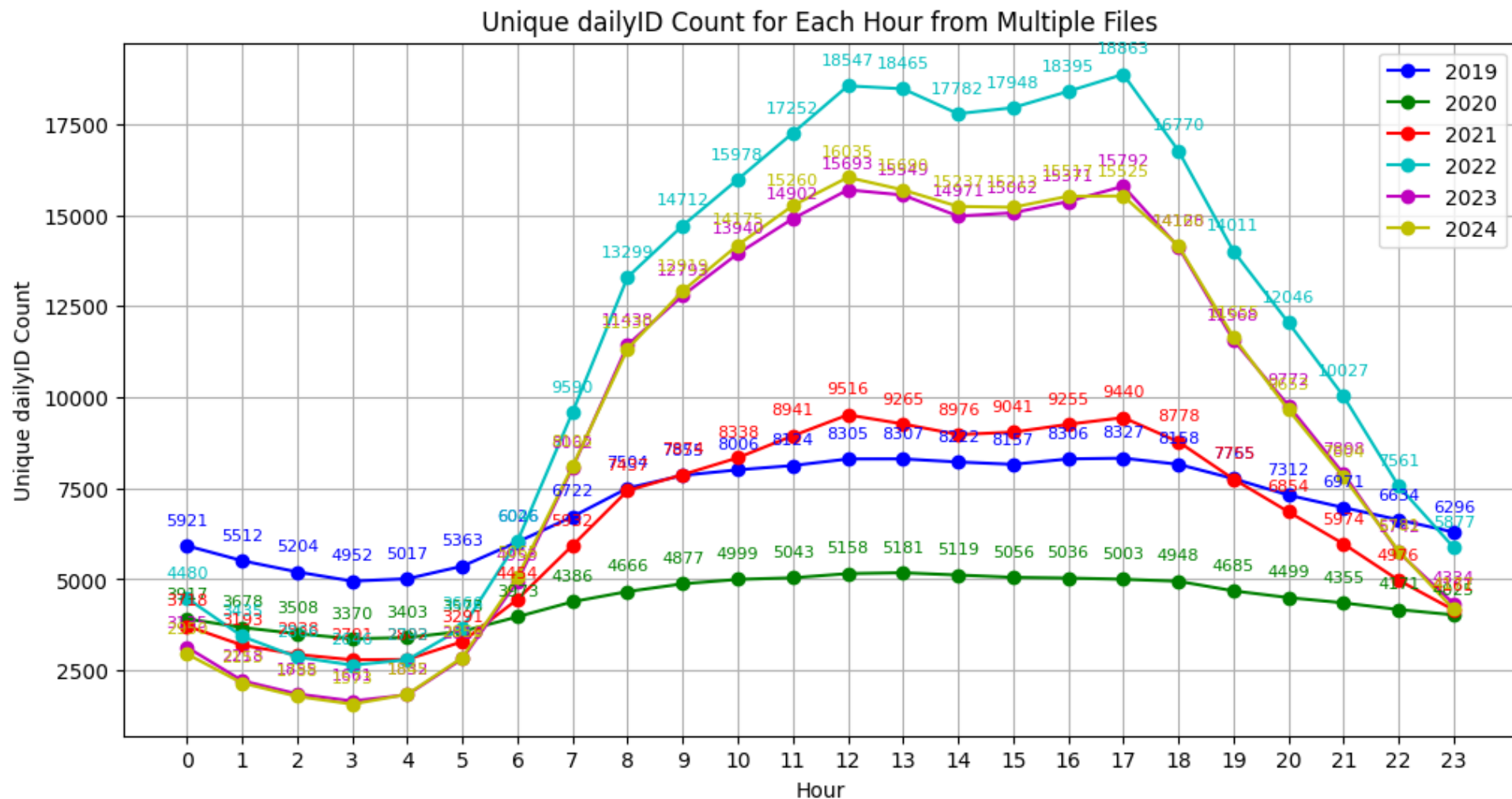
2022年3月1日、外国人の新規入国制限の緩和措置を実施

2022年10月11日、入国者総数の上限が撤廃

2019年～2024年 終日データ（6月・第2土曜日）



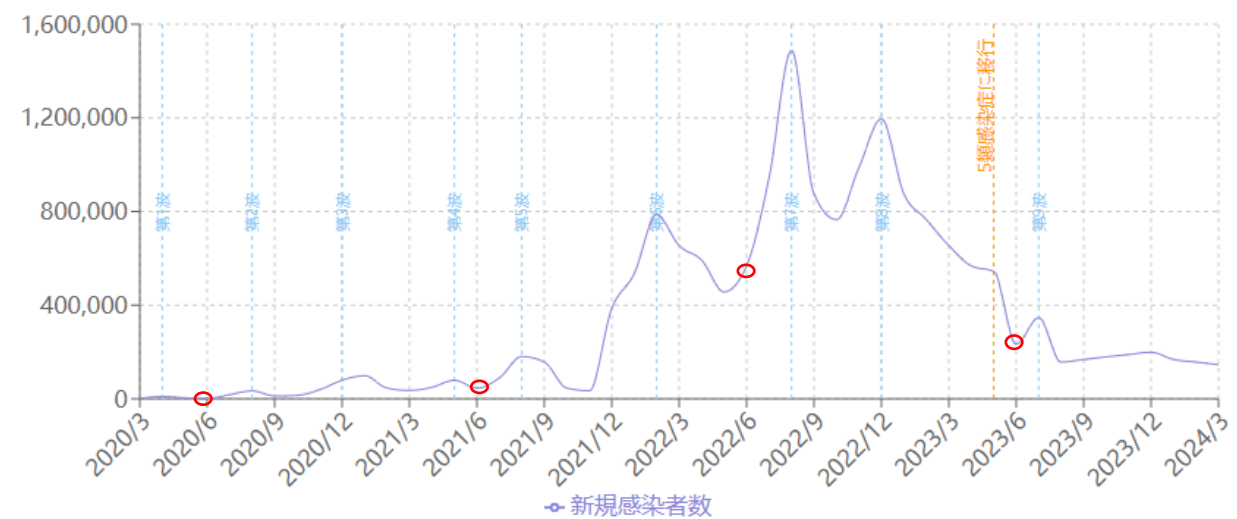
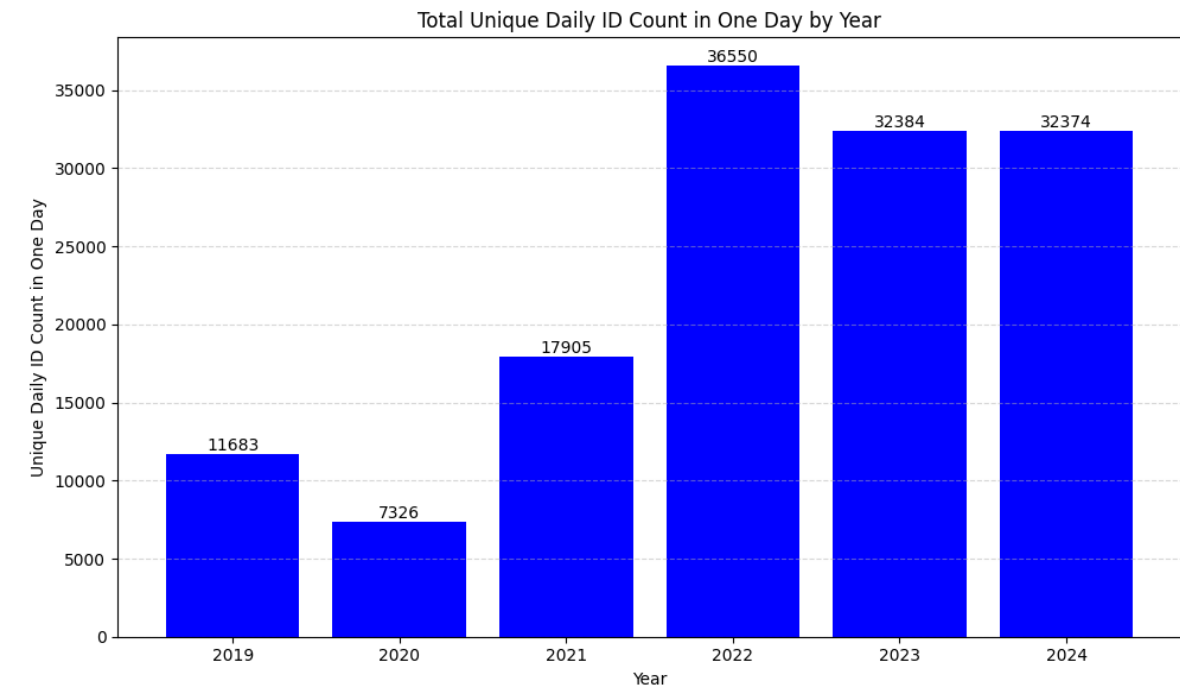
2019年～2024年 終日データ (6月・第2土曜日)

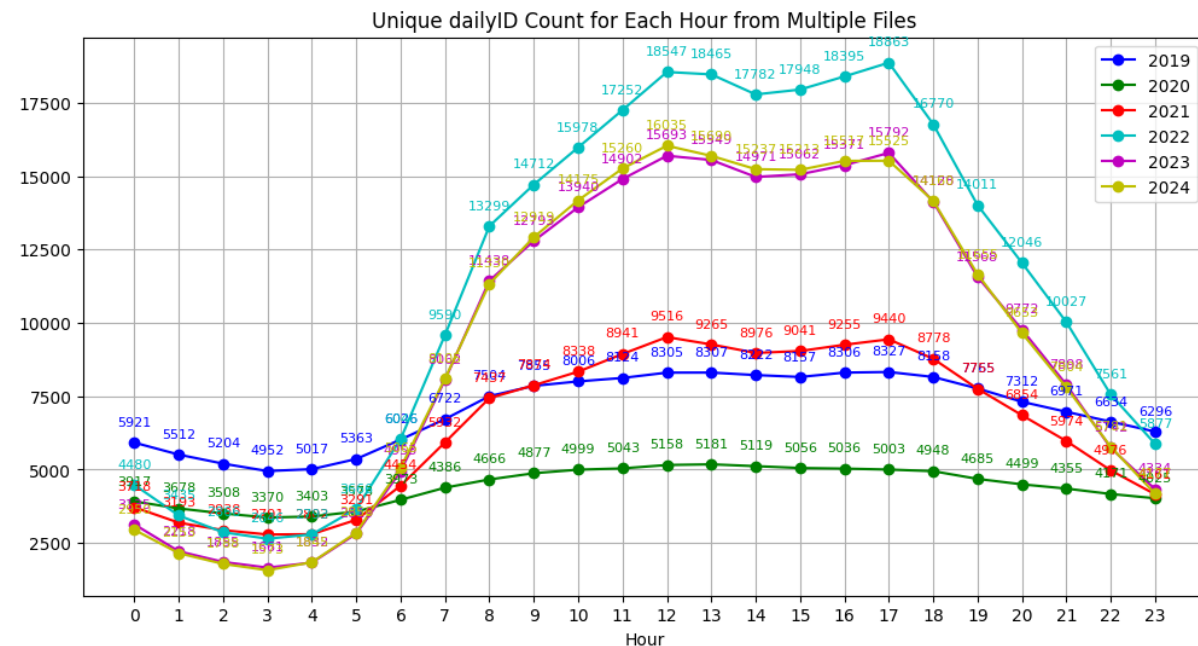
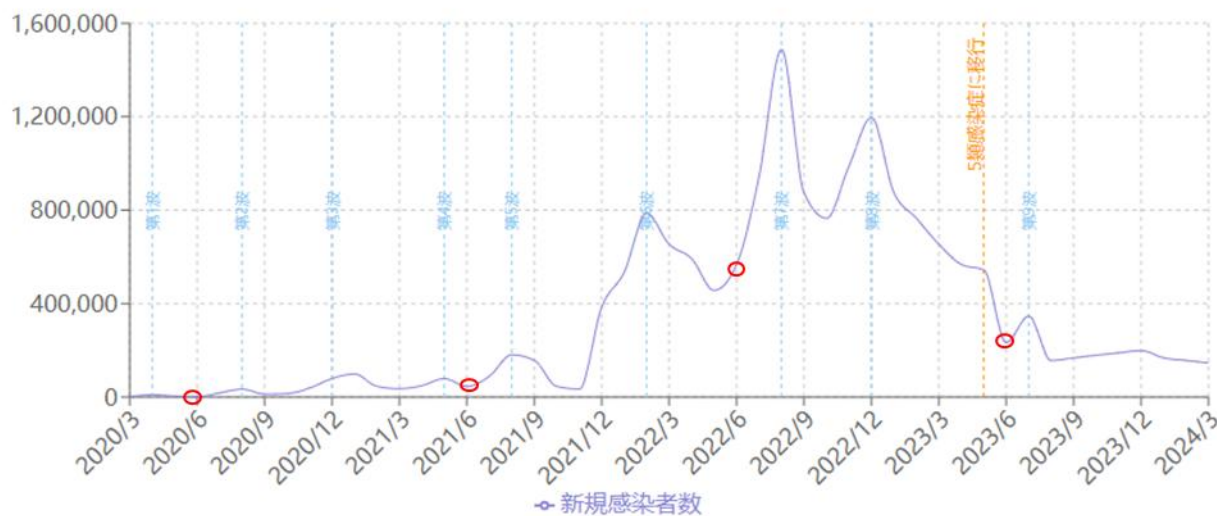


全体の人流の推移:

2020年は2019年に比べて大幅に減少し、2021年には若干回復し、2022年には人流が急増している。その後、2023年と2024年は2022年に比べてわずかに減少し、安定している。

2020年に人流が大幅に減少した主な理由は、新型コロナウイルスの感染拡大に伴う政府の緊急事態宣言や移動制限などの対策により、人々は外出を控えるようになったからと考えられる。2021年にはワクチン接種の普及や感染状況の改善により、人流が徐々に回復した。2022年には、社会的活動がさらに大幅に再開され、人々の移動が急増したと考えられる。その結果、新規感染者も増大している。その後、2023年と2024年には、経済や生活習慣が安定し、人流が安定した状態に戻ったと考えられる。なお、人流の増加には、統計に使用されたアプリのユーザー数の増加も影響していると考えられる。





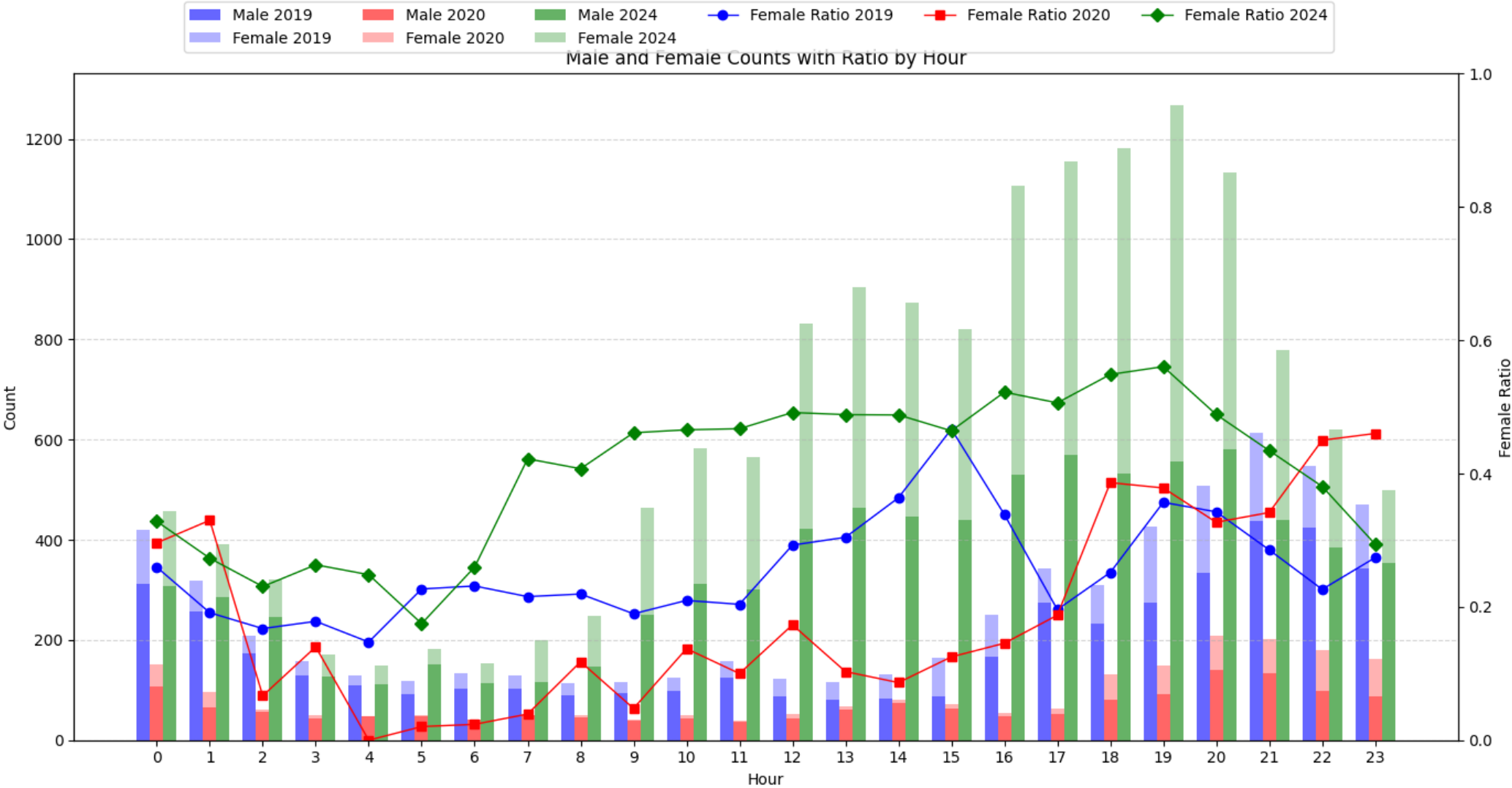
毎時人流の推移:

2019年と2020年は昼夜の人流の変動がそれほど大きくなかったが、2021年以降、日中の人流が増大し、夜間の人流が減少することで、昼夜の人流の変動が大きくなっている。

2019年において、昼夜の変動がそれほど激しくなかった理由は、コロナ禍以前は夜間の外出が活発だったからと推測される。また、2020年は、新型コロナウイルスの感染拡大により、緊急事態宣言や移動制限などが発令され、昼夜を問わず外出が控えられたからだと推測される。

2021年以降、ワクチンの普及や感染状況の緩和に伴い、人々の活動が再開されると、日中の外出が大幅に増大したが、夜間の外出はそれほど回復しなかった。これにより、2021年以降は昼夜の人流の変動が顕著になったと推測される。

国分町の人流変化 2019年（コロナ前） 2020年（コロナ中） 2024年（コロナ後）

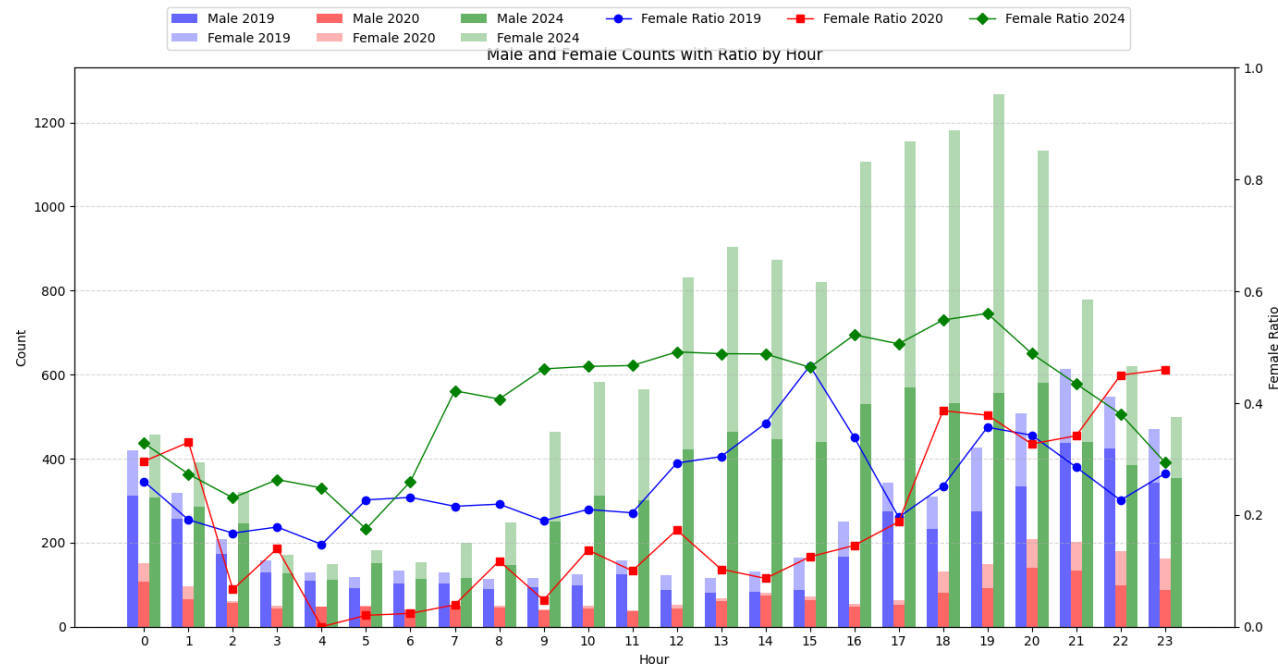


国分町の人流分析：

コロナ禍期間中、国分町の終日人流が大幅に減少した。しかし、コロナ禍後は人流が急激に増加し、活動が再び活発化した。興味深い点として、コロナ禍後の人流のピークの時間帯はコロナ禍前より早まっている。これは、夜間の活動が縮小され、より早い時間帯に人々が集中するようになったことが一因と考えられる。

男女比の推移：

コロナ禍期間中、特に日中に街を訪れる女性の割合がコロナ禍前より大きく減少し、男性比率が増大した。一方、コロナ禍後は日中に街を訪れる女性の割合が、男性以上に大幅に増加した。これは、国分町がコロナ禍を通して、従来の夜間中心の商業エリアから、昼夜を通じて多様な来街者を受け入れる複合的な商業エリアへと発展したと推測される。



The background of the slide is a solid dark purple color. It is decorated with a repeating pattern of stylized, light purple leaves and branches. The leaves are simple, rounded shapes, and the branches are thin, curved lines. The pattern is distributed across the entire slide, with some branches curving from the top left towards the bottom right.

Pythonによる人流データ解析と視覚化