Pythonではじめる人流データ解析実践

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

目次

データの紹介　Agoop ポイント型流動人口データ

データ概要

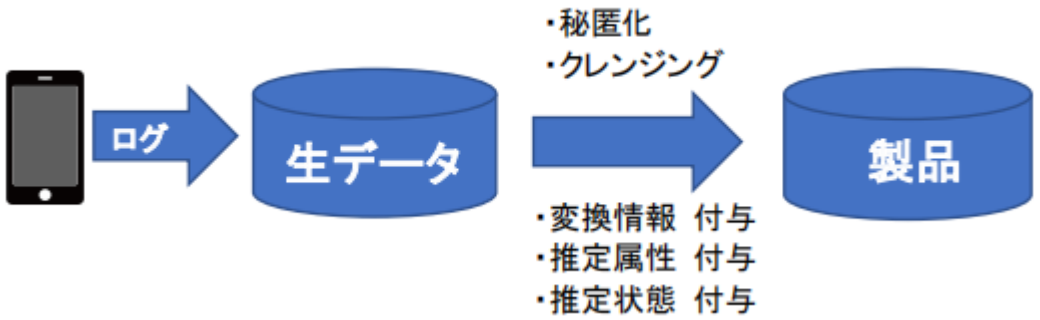
- スマホアプリを通じて収集された仙台市全体の人流データ
- 東北大学がAgoop社より購入
- 2019年（6月・12月）、2020年（6月・12月）、2021年（1月～12月）、2022年（6月・12月）、2023年（6月・12月）、2024年（6月）
- データは一日単位でCSVファイル形式に保存

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

データの紹介　Agoop ポイント型流動人口データ

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

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)			
2	年	year	整数(4 桁)		年	
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	
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			

ユーザーを識別するID

時間情報

位置情報

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月の一日平均　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	2dacccc2fe	2020	12	1	2	0	0	38.22777	140.8817	Android	Japan	44010	Japan

目次

Python

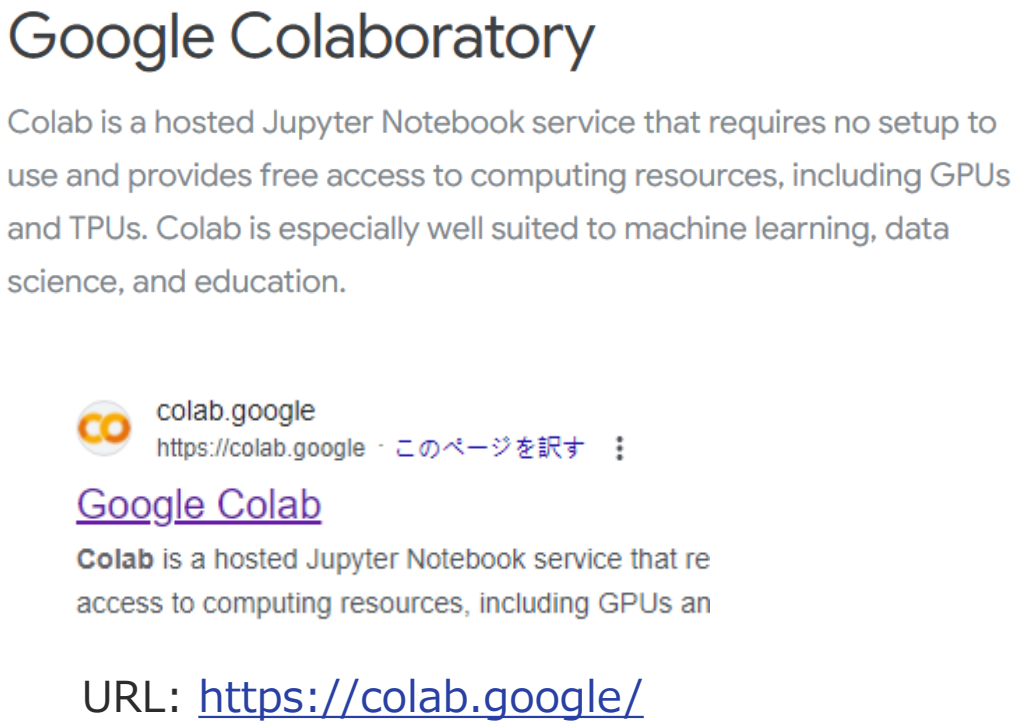
シンプルで強力なプログラミング言語。データ解析や機械学習に広く使用されている。

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。



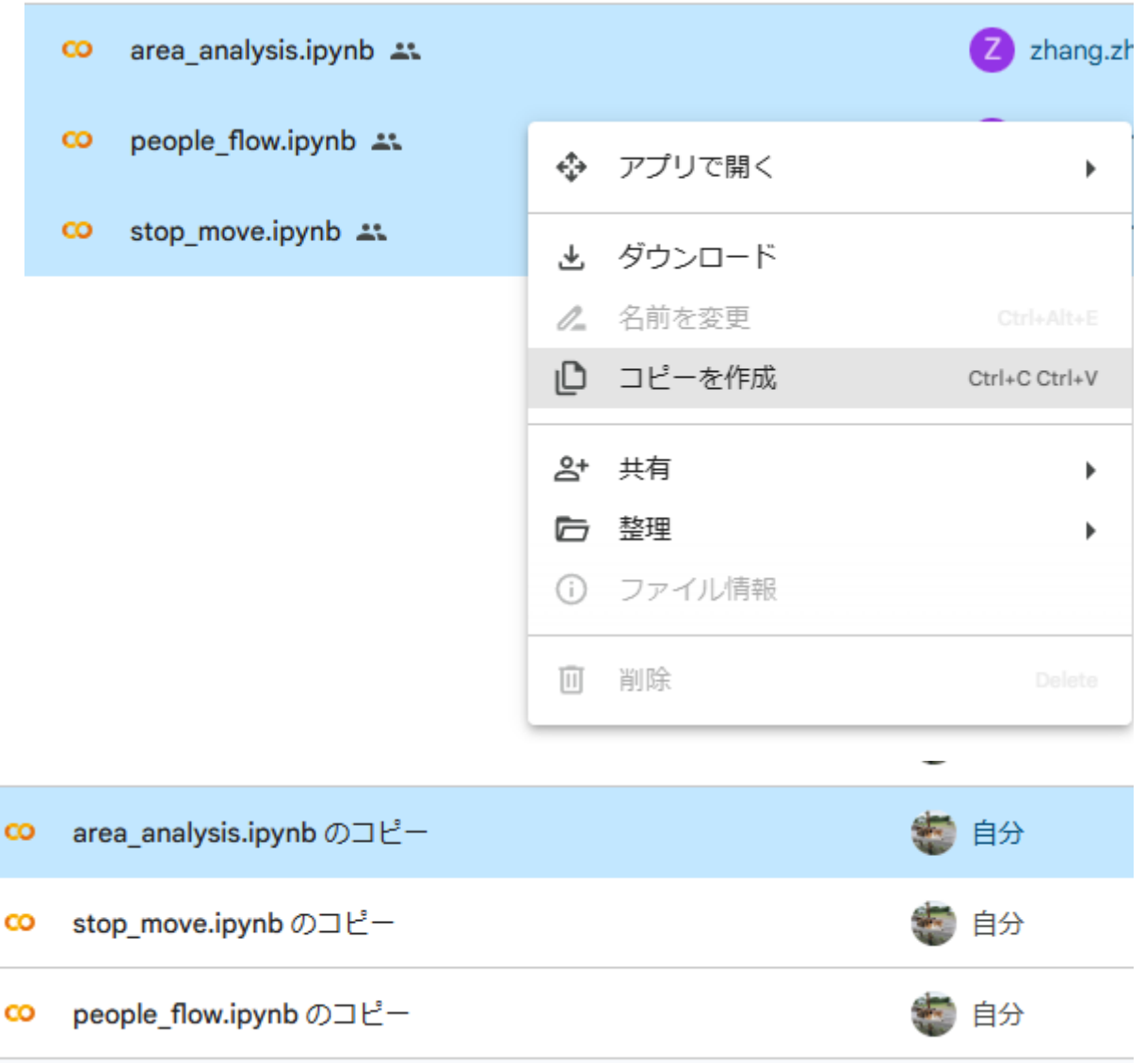
共有したファイルの使い方

- 共有ファイルを確認する
  - people_flow.ipynb
  - area_analysis.ipynb
  - stop_move.ipynb
  - .csv

分析およびデータ処理のためのコードが含まれている。

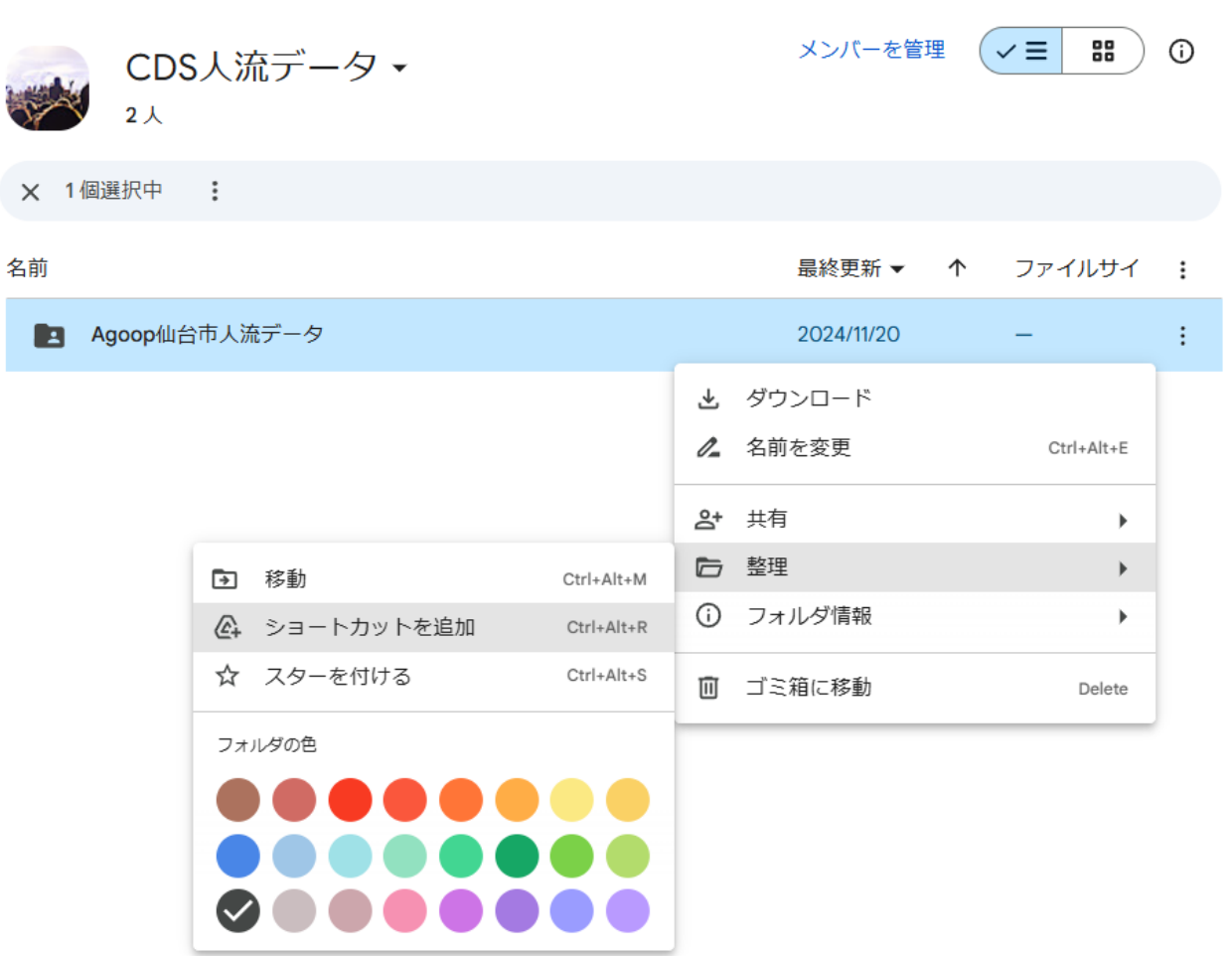
仙台市の人流データ

- 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



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] import pandas as pd

    # 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)
```

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  fe78af49b43c2bf0c46f6f9cc62fdc1a73bcb50f425c03...  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
```

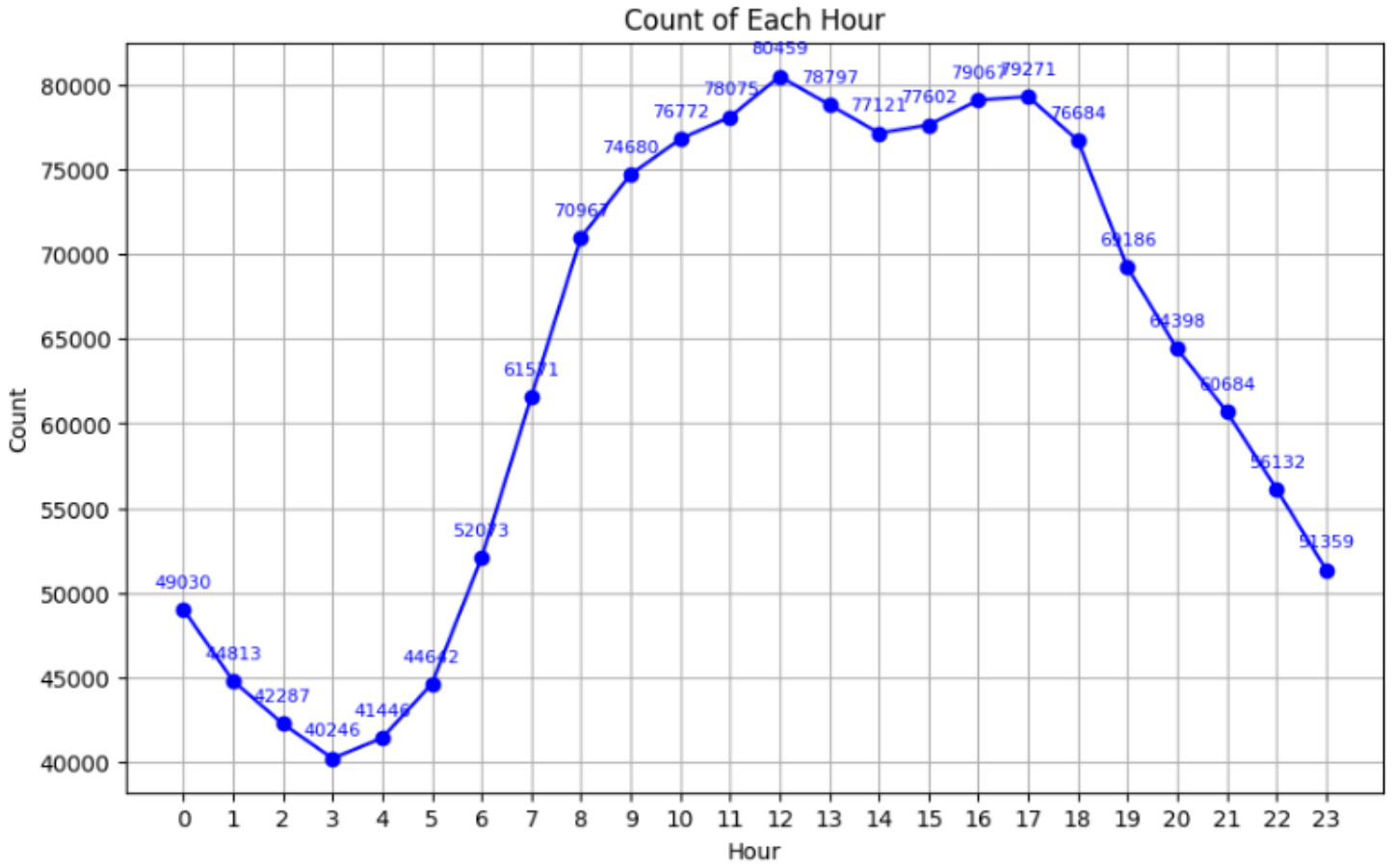
データの視覚化を行うためのライブラリで、さまざまな種類のグラフや図を簡単に作成できる。

グラフのサイズを10インチ×6インチに設定

```
plt.figure(figsize=(10, 6))
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、12時から15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)
```

hour	Male Count	Female Count	Nan Count
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

- 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）

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))
```

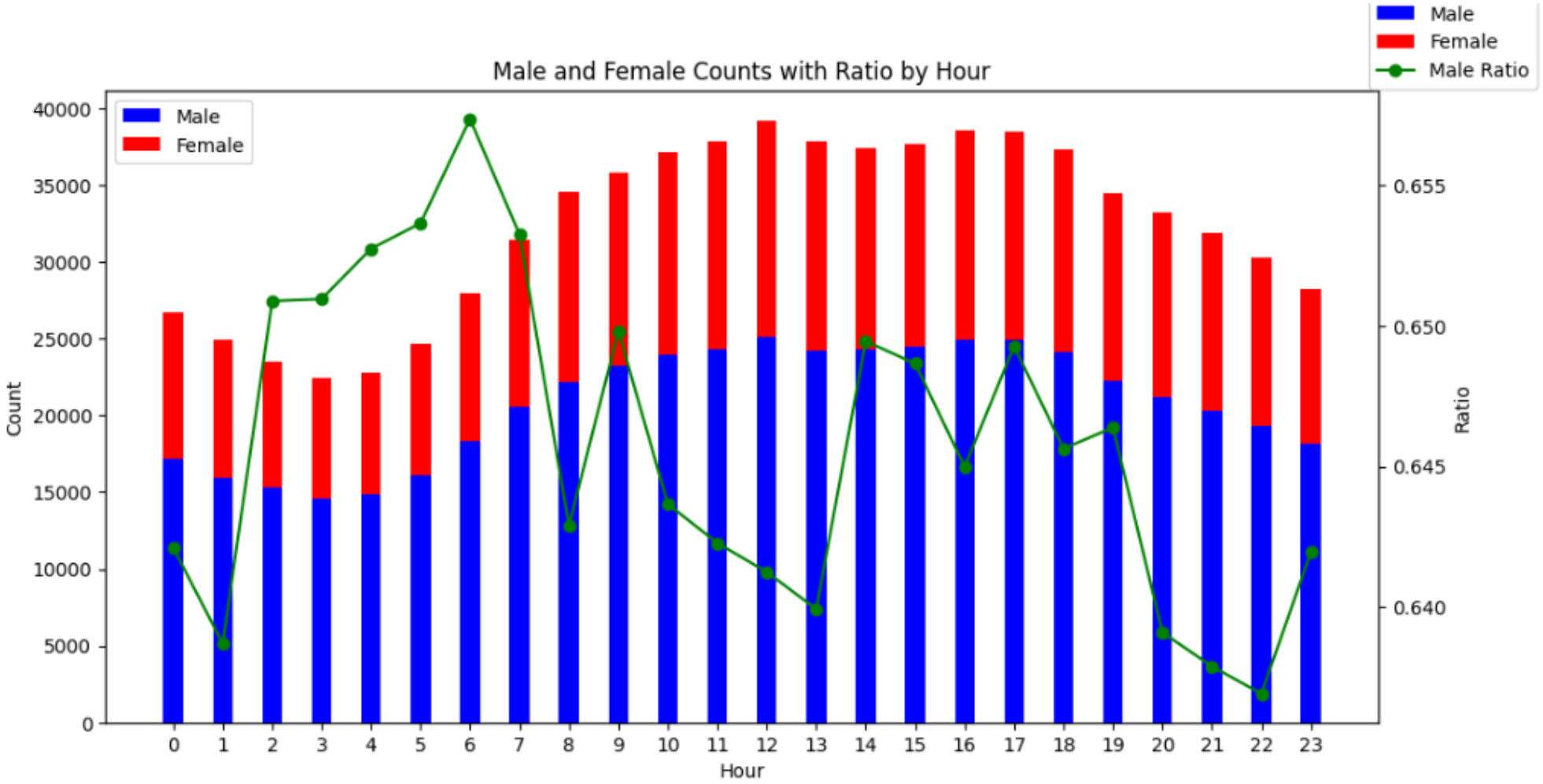
  - 男女人数を視覚的に比較するために、棒グラフを使用する。

```
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()
```

視覚化結果



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],
```

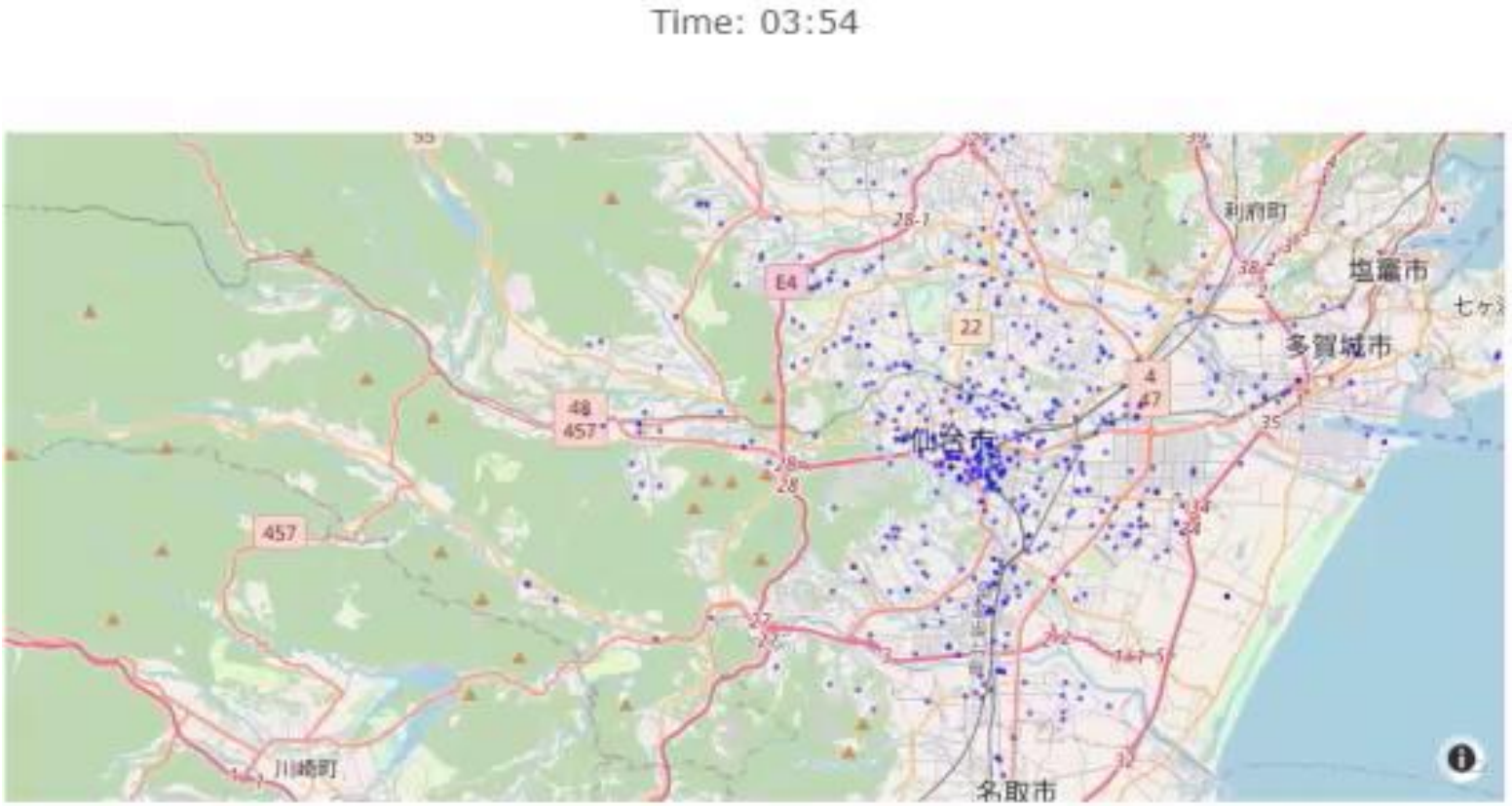
Dashをインストール

```
!pip install dash
```

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

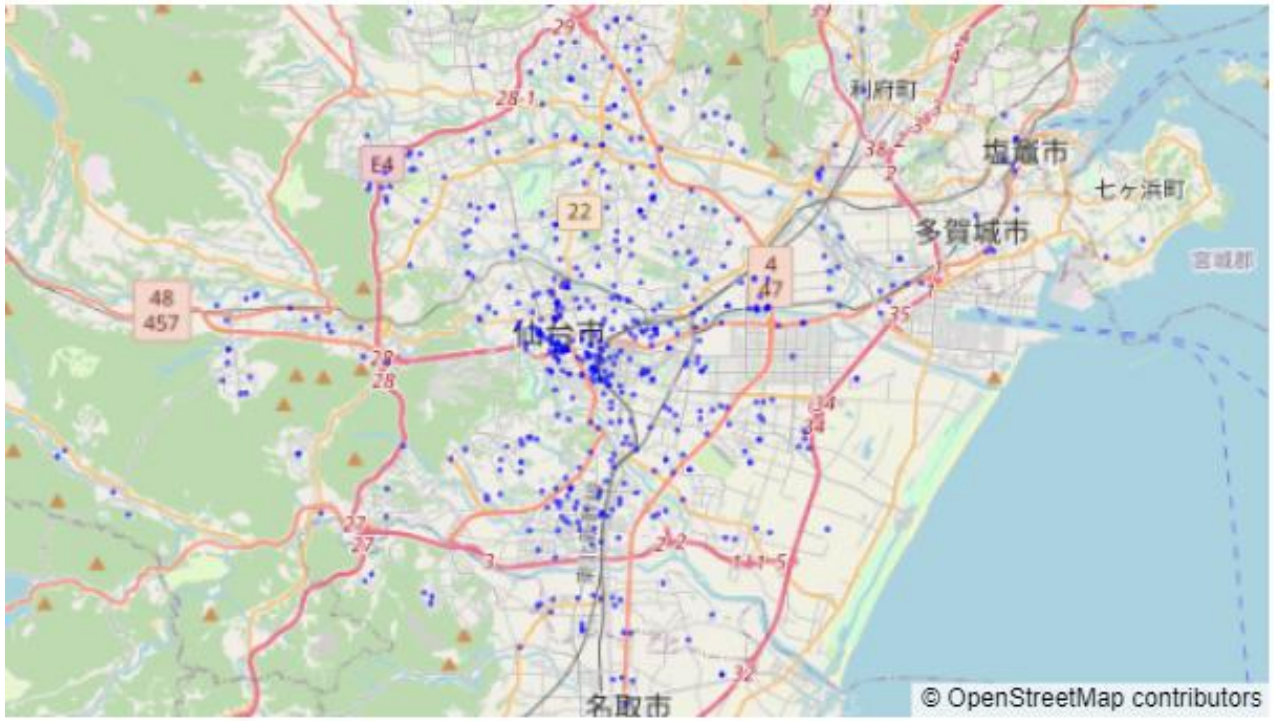
アニメーションを作成

```
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
app1 = dash.Dash(__name__)

# アプリケーションのレイアウト
app1.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
])

@app1.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__':
    app1.run_server(debug=True)
```

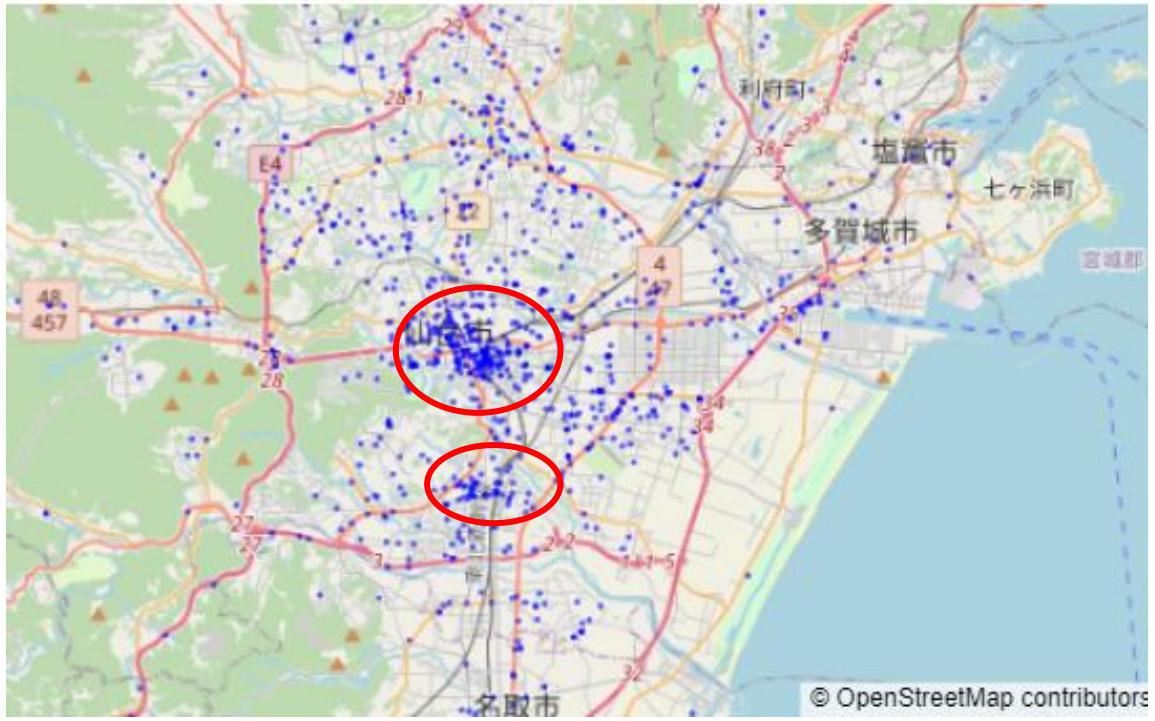

Time: 03:54
E4
22
48
457
457
仙台市
塩竈市
多賀城市
名取市
利府町
川崎町

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

繁華街の人流解析

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

2019年6月8日（コロナ禍以前）のデータから情報の読み取り

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

```
Mounted at /content/drive
```

```
[2] import pandas as pd

    # 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（繁華街）の人流データを統計解析

国分町の緯度と経度の範囲を地図上で確認して、以下のように設定する

```
koku_lat_min=38.26031
koku_lat_max=38.26872
koku_lon_min=140.86825
koku_lon_max=140.87000
```

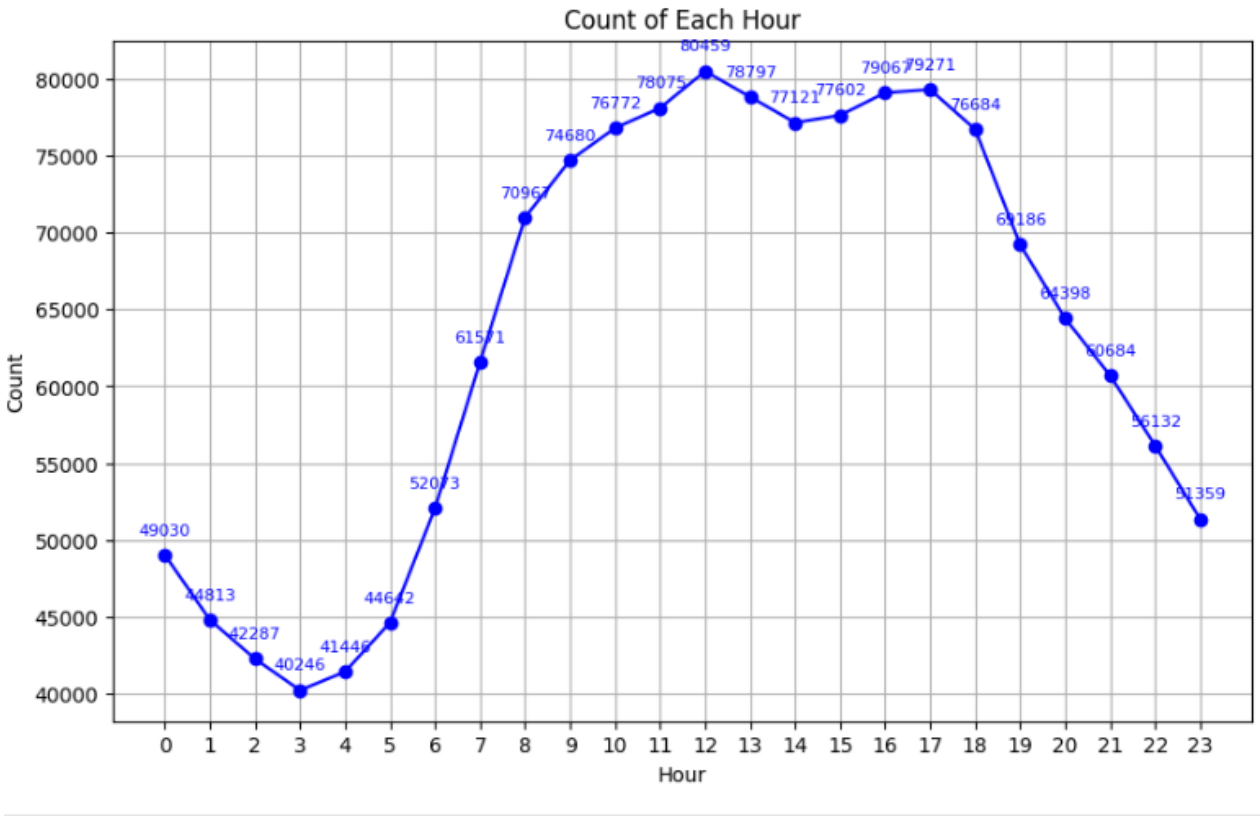
人流データから国分町の人流情報を抽出

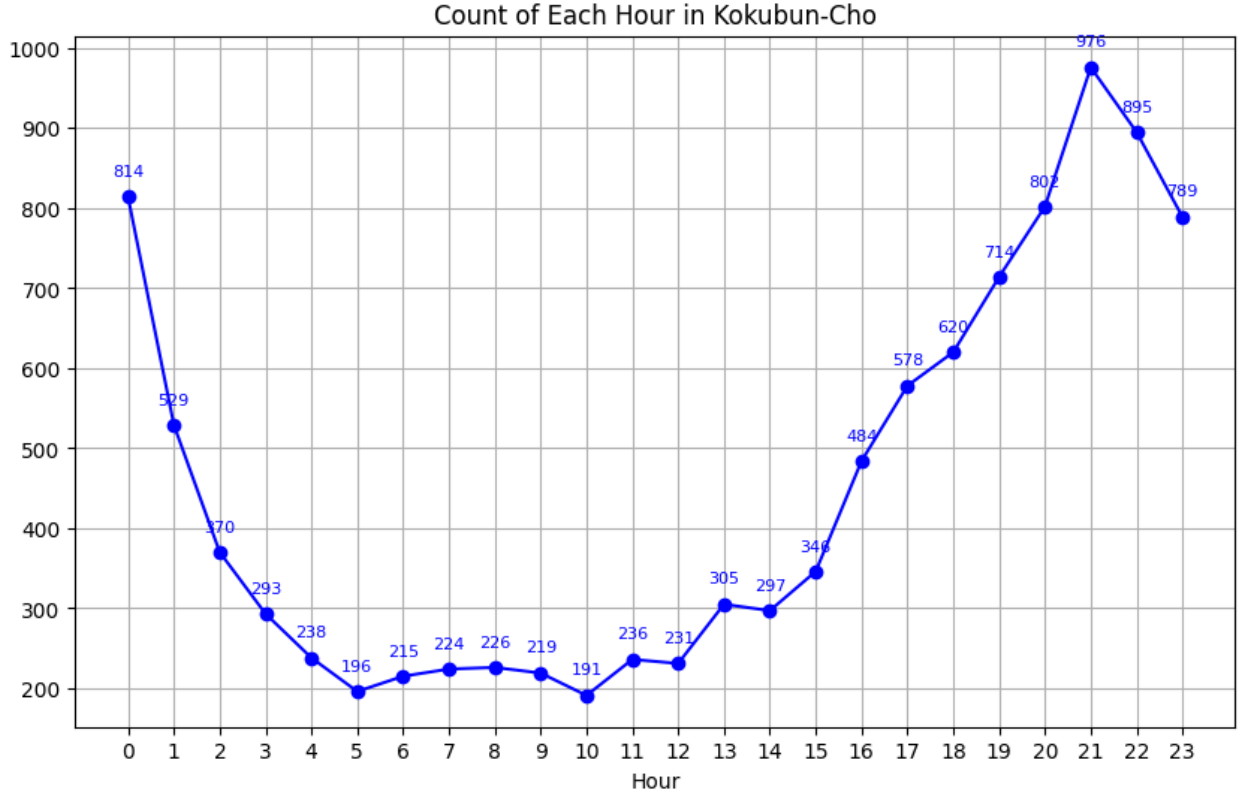
```
df_kokubun=data[  (data['latitude']  >  koku_lat_min)  &  (data['latitude']  <  koku_lat_max)  &  \
                  (data['longitude']  >  koku_lon_min)  &  (data['longitude']  <  koku_lon_max)]
print(df_kokubun.head())
```

```
                                                dailyid  year  month  day  \
157  9a157db529295b34fbc191bacce9bf2f7b3e2162b09037...  2019      6    8
232  21e84dffd561c41ddd02cf5c0122253015beecff08e3f0...  2019      6    8
246  9c23a80f09c326bb8d78155ce26e035eaacf8fa4251f13...  2019      6    8
314  5076e83ed38da541fa6c2c2d5b55e4be3dc2f27e99156c...  2019      6    8
326  f2cd13c1aba9b6c1f16e1df3a8019a283195fa157415d6...  2019      6    8

     dayofweek  hour  minute   latitude   longitude       os  ...   course  \
157          6     0       0  38.262091  140.869005  Android  ...  334.983
232          6     0       0  38.264158  140.869494  Android  ...  344.349
246          6     0       0  38.264426  140.869274  Android  ...      NaN
314          6     0       0  38.265826  140.868741  Android  ...      NaN
326          6     0       0  38.263684  140.869743  Android  ...      NaN
```

時間ごとの統計（練習問題１）
視覚化結果　　（練習問題２）

繁華街の人流解析　エリア別での視覚化結果





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。

国分町の人流散布図マップを作成

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

      #  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}
```

人数が少ないため、時間ごとで表示

```
!pip install dash
```

Dashをインストール

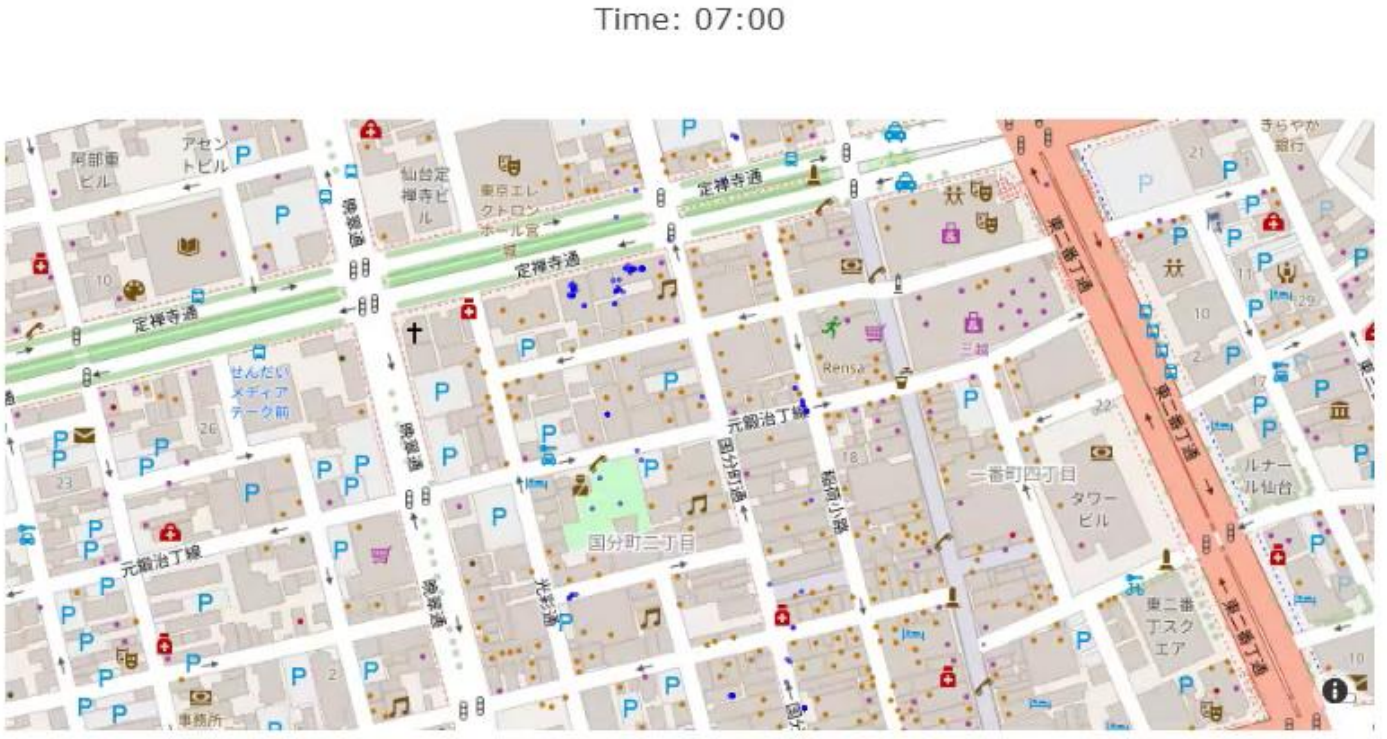
```
import  dash
import  dash_core_components  as  dcc
import  dash_html_components  as  html
from  dash.dependencies  import  Input,  Output
import  plotly.graph_objs  as  go

max_total_hours  =  df_kokubun['hour'].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=1000,  n_intervals=0),    #  1秒ごとに更新

])

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

繁華街の人流解析

人流散布図マップを作成


```
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=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: {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(debug=True)
```

国分町のエリアを表示



繁華街の人流解析

人流散布図マップ

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

人流散布図マップ

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。

目次

エリア間の人流解析

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。

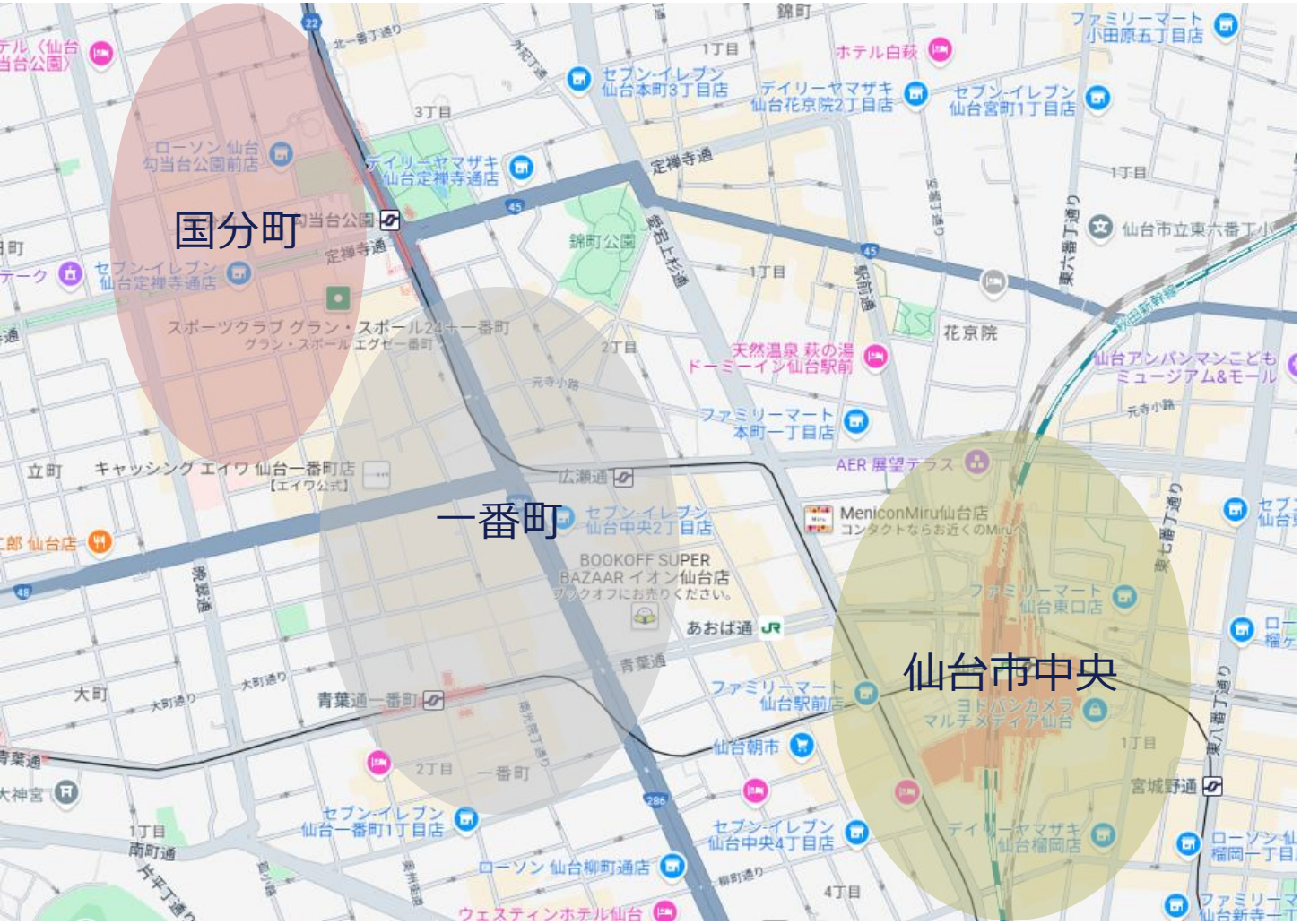
移動人数

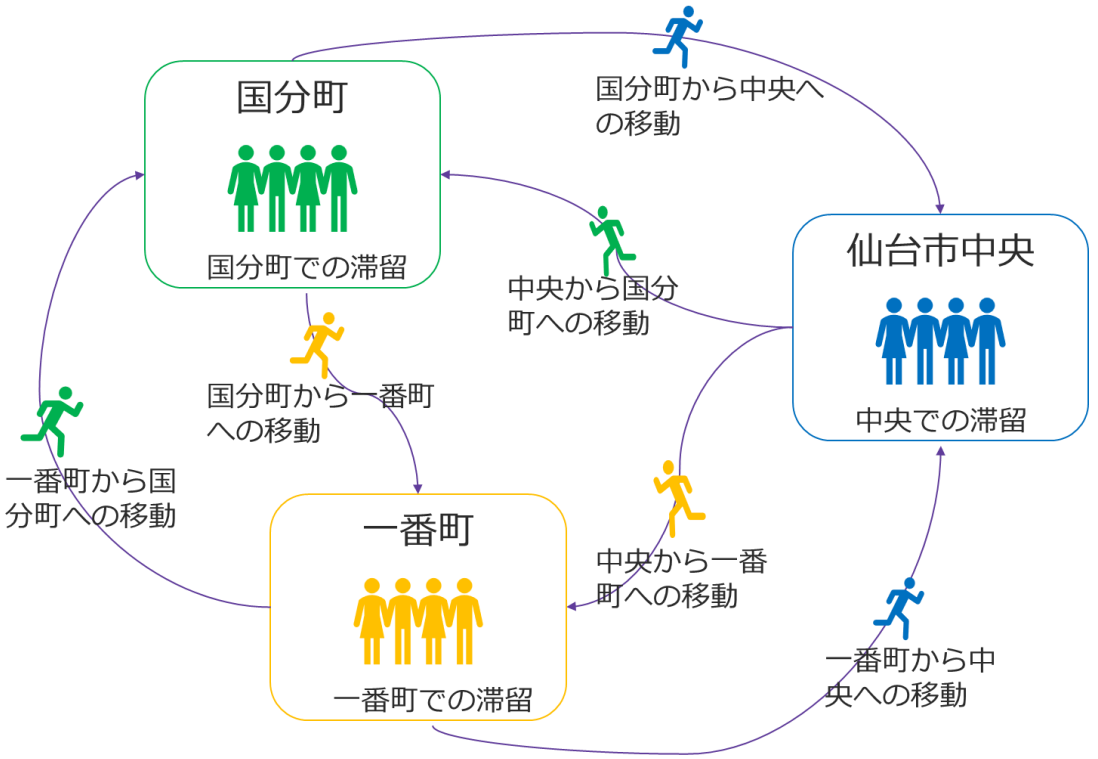
時間帯毎に異なるエリア間を移動する人々の数である。時間帯毎・エリア間毎の移動のパターンを解析することで、どの時間帯にどのエリア間の移動が活発か、交通の流れを明らかにすることができる。

各エリアにおける滞留と、各エリア間の移動を解析

エリア間の人流解析

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





エリア間の人流解析

滞留

stop_move.ipynb

データを読み取り

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[2] import pandas as pd

    # CSVの読み取り
    file_path = '/content/drive/MyDrive/Agoop仙台市人流データ/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
```

エリア間の人流解析

stop_move.ipynb

各時間・各エリアの滞留人数の集計

```python
# 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)
```

```
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]
```

エリア間の人流解析

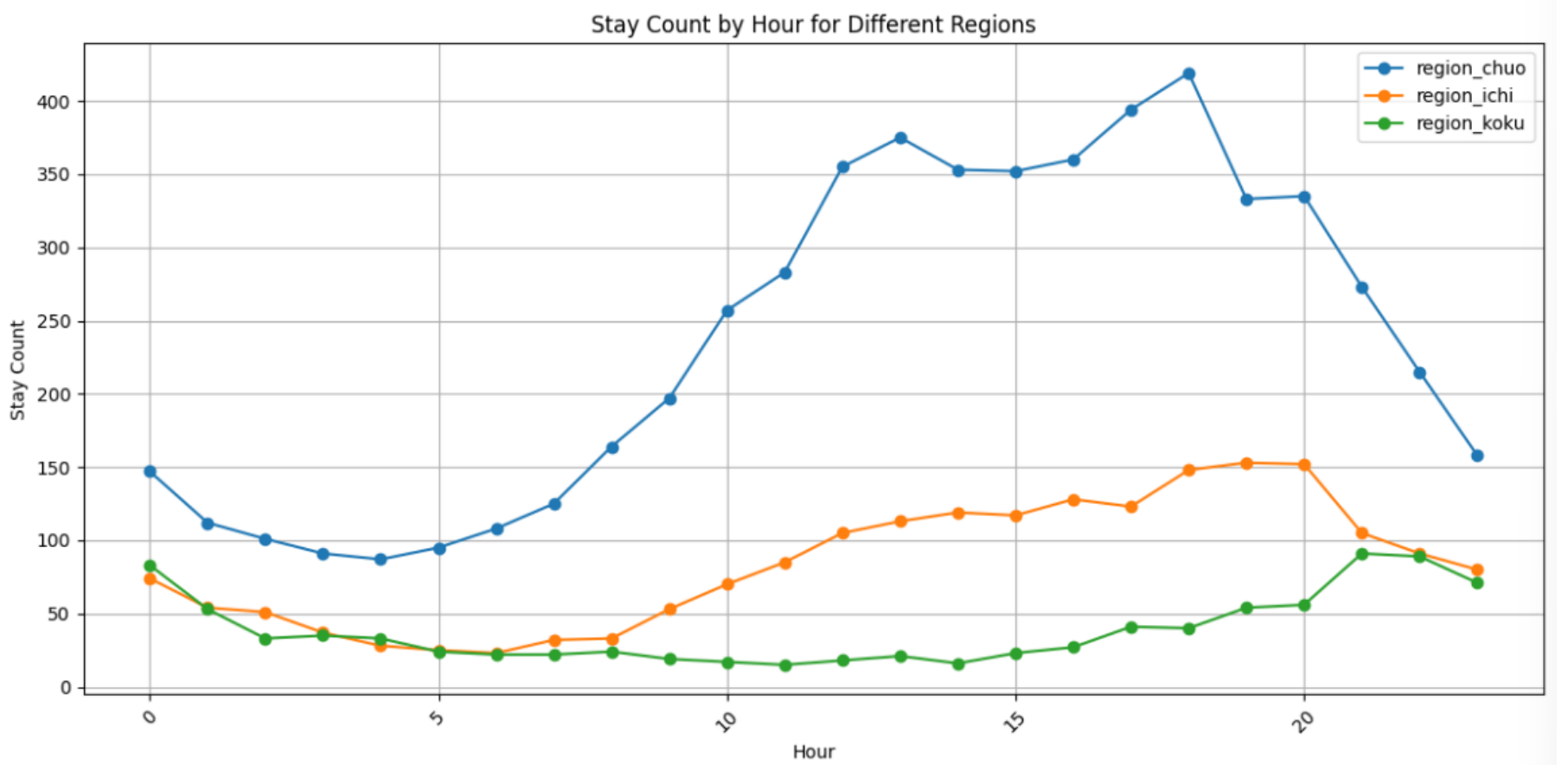
stop_move.ipynb

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）

エリア間の人流解析

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]
```

エリア間の人流解析

滞留と移動データを折れ線グラフで表示

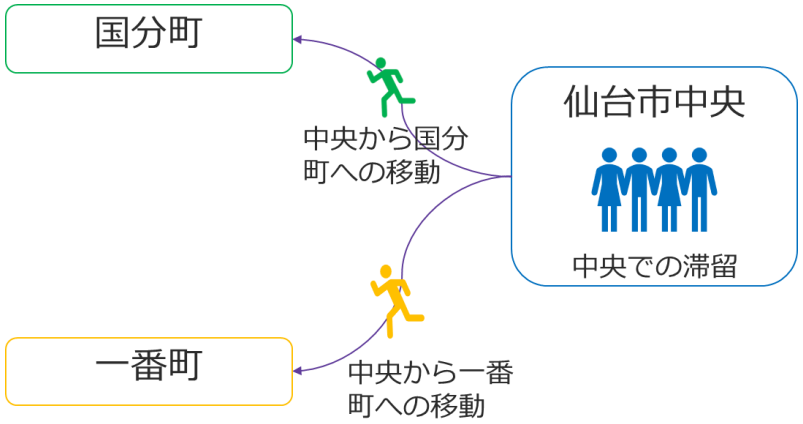
stop_move.ipynb

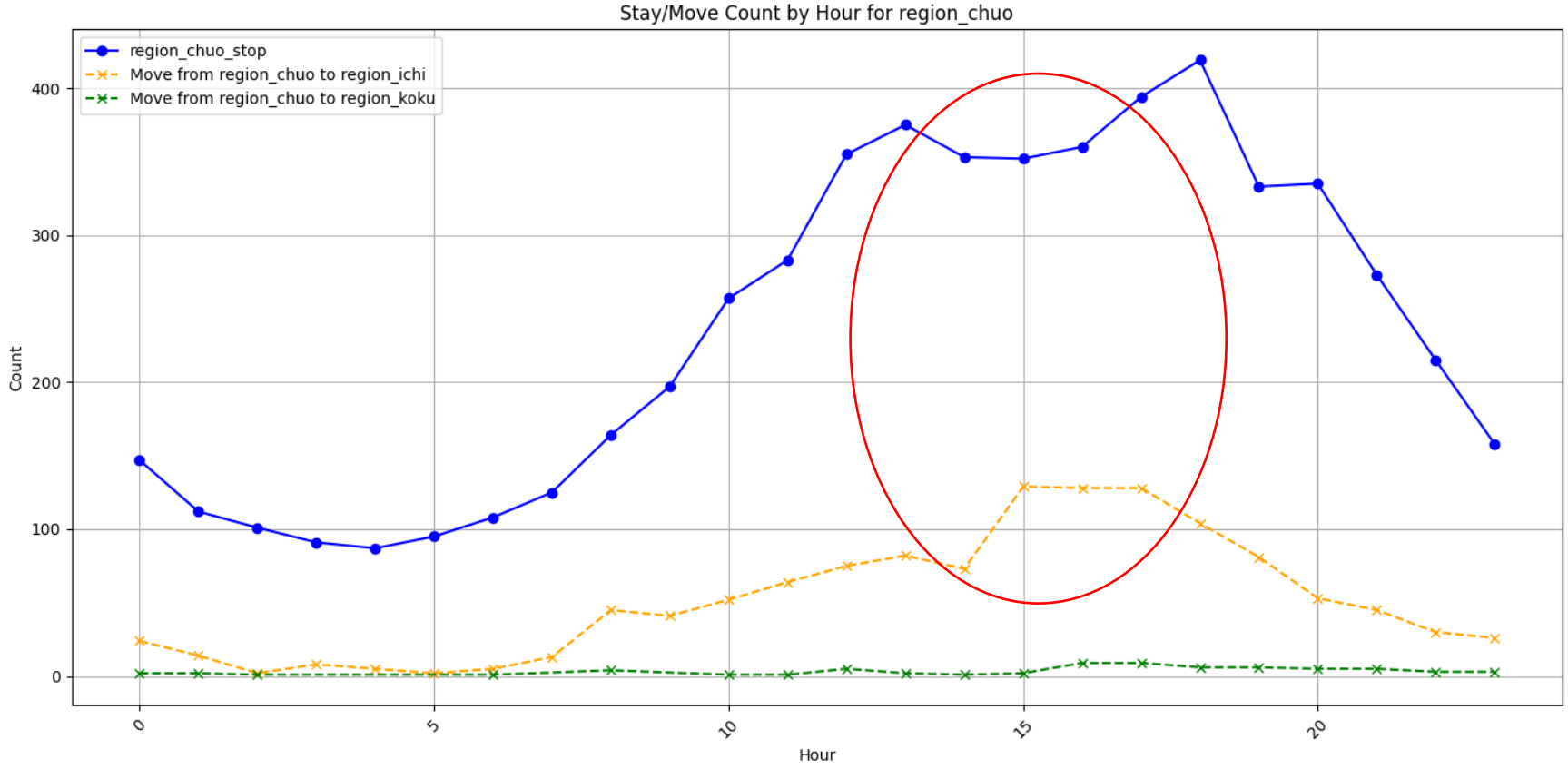
```
# 滞留データの折れ線グラフを描画
regions = stay_counts['region'].unique()
colors = {
    'region_chuo': 'blue',
    'region_ichi': 'orange',
    'region_koku': 'green'
}

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）

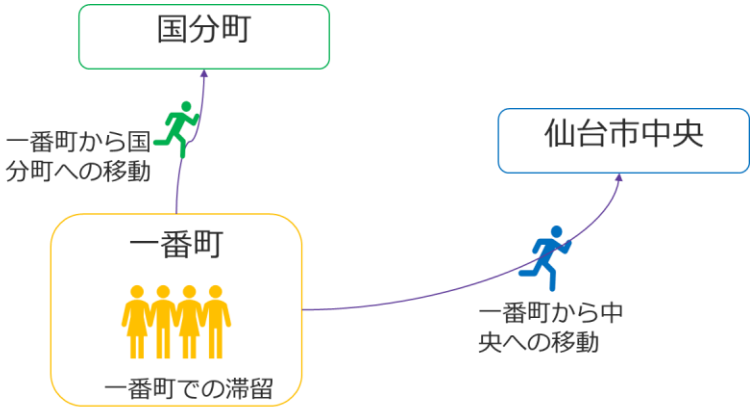


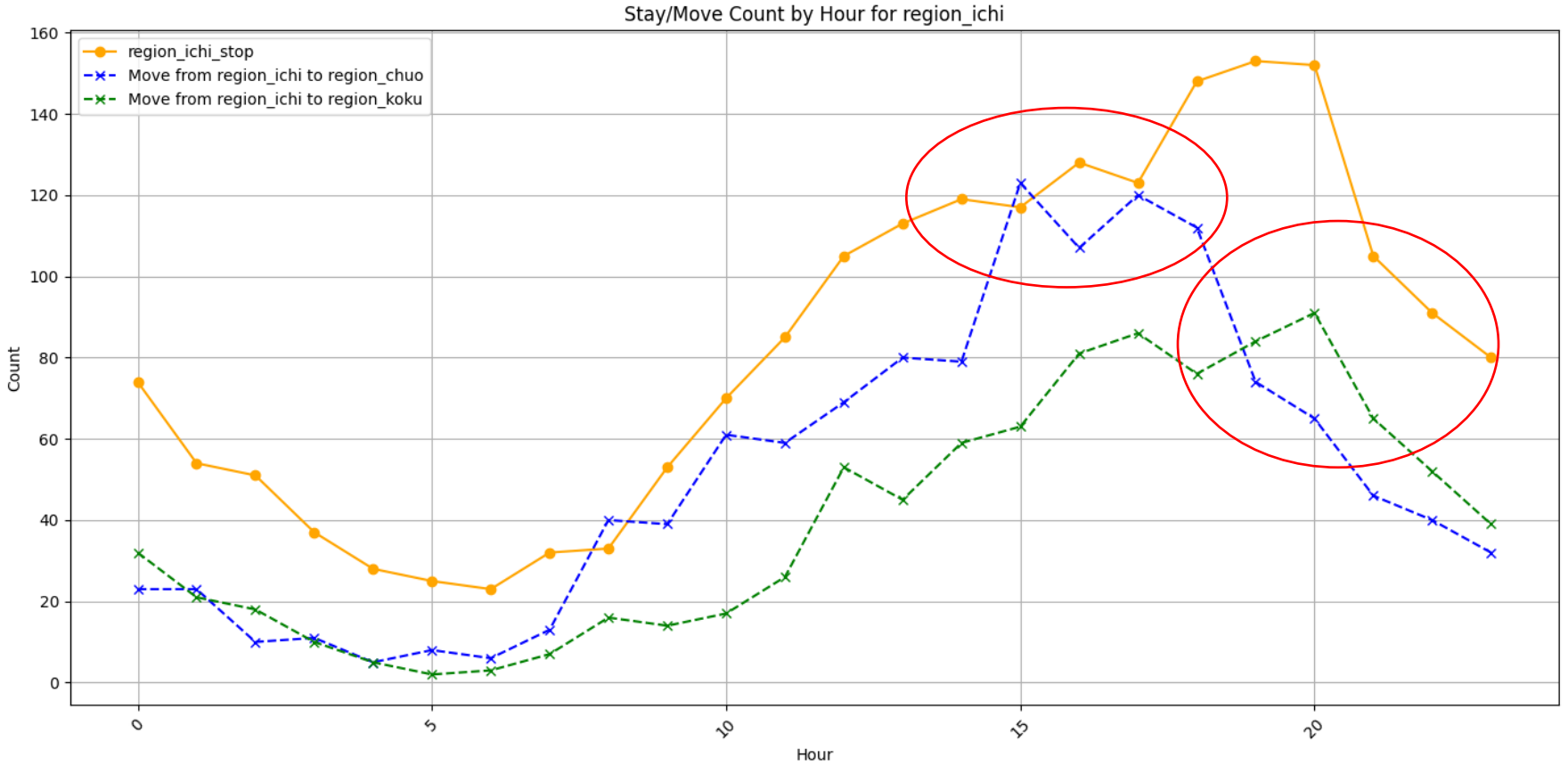


昼食後の時間帯に一番町への移動が増加していることは、周辺の飲食店やカフェでの食事を終えた人々が、買い物や観光を目的として移動していることを示唆している。

stop_move.ipynb

滞留と移動の折れ線グラフ（一番町）

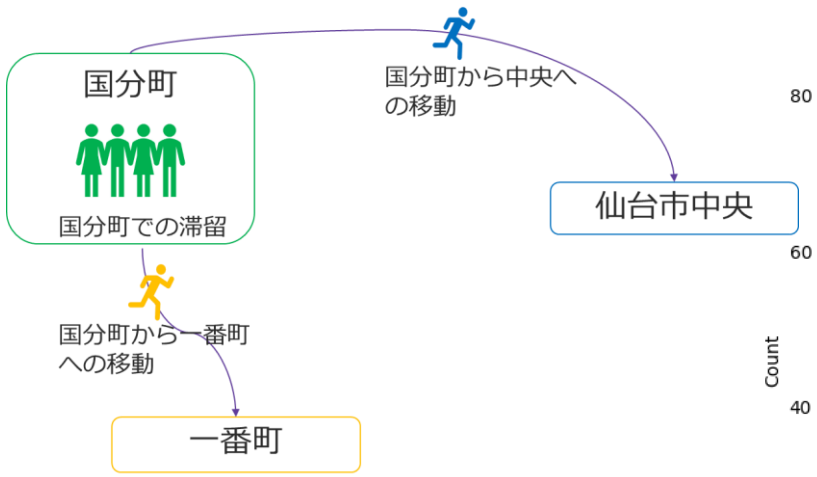


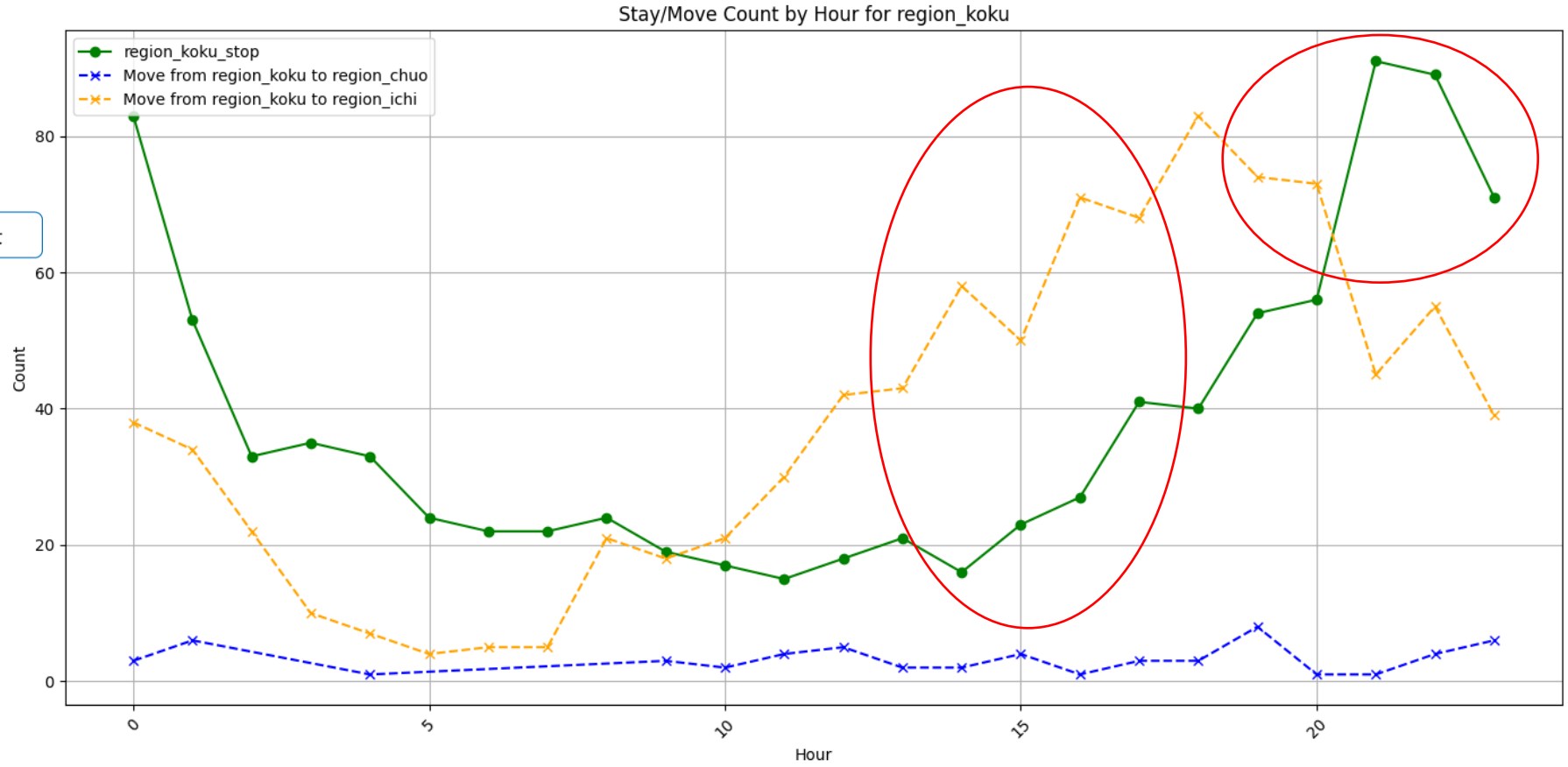


15時から18時にかけて、一番町から仙台市中央への人の移動が見られる。しかし、一番町の人数が減少しないことは、一番町での活動や交流が続いていることを示唆している。
17時から21時にかけて、国分町への移動が増加している。これは、夕食や飲み会を目的に人々が国分町に移動していることを示唆している。

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。

Matplotlibの他の視覚化方法を試す

stop_move.ipynb

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

```
def plot_hourly_movements_and_stays(hourly_movement_data, stay_data, hour):
    # グラフのサイズを設定
    fig, ax = plt.subplots(figsize=(10, 6))

    # 円の半径と位置
    circle_radius = 0.1
    region_positions = {
        'region_koku': (0.2, 0.5),
        'region_ichi': (0.5, 0.7),
        'region_chuo': (0.8, 0.5)
    }
    region_colors = {
        'region_koku': 'brown',
        'region_ichi': 'gray',
        'region_chuo': 'olive'
    }
    # 円を描画し、円の中に滞留人数を表示
    for region, pos in region_positions.items():
        circle = plt.Circle(pos, circle_radius, \
            color=region_colors[region], alpha=0.5, label=region)
        ax.add_artist(circle)

        # 現在の時間の滞留人数を取得
        stay_count = stay_data.get(region, 0)
        ax.text(pos[0], pos[1], f'{region}\nStay: {stay_count}',\
                fontsize=12, ha='center', va='center', color='black')
```

```
    # 矢印と数字を描画
    arrows = []
    for start_region, end_region in [('region_koku', 'region_ichi'), \
      ('region_koku', 'region_chuo'), ('region_ichi', 'region_chuo')]:
        count = hourly_movement_data.get((start_region, end_region), 0)
        start_pos = region_positions[start_region]
        end_pos = region_positions[end_region]

        # 矢印の方向を計算
        direction = np.array(end_pos) - np.array(start_pos)
        direction_norm = np.linalg.norm(direction)
        direction_unit = direction / direction_norm

        # 矢印のオフセットを計算
        offset = 0.02
        arrow_offset = offset * direction_unit
        arrow_side_offset = offset * \
        np.array([-direction_unit[1], direction_unit[0]])  # 側面のオフセット

        # 行きの矢印を描画
        arrows.append((start_pos + arrow_offset - \
            arrow_side_offset, end_pos + arrow_offset, '->', 'blue'))
        # 帰りの矢印（少し位置をずらして）を描画
        arrows.append((end_pos - arrow_offset + \
        arrow_side_offset, start_pos - arrow_offset, '->', 'red'))
```

エリア間の人流解析

stop_move.ipynb

丸と矢印を使って地域間の流れを示し、滞留人数を丸の中に表示

```
# 矢印を描画
for start, end, style, color in arrows:
        ax.annotate('', xy=end, xytext=start,
                            arrowprops=dict(arrowstyle=style, lw=2, color=color, alpha=0.7))

# 流量数量を描画
for (start_region, end_region), count in hourly_movement_data.items():
        start_pos = region_positions[start_region]
        end_pos = region_positions[end_region]
        mid_point = ((start_pos[0] + end_pos[0]) / 2, (start_pos[1] + end_pos[1]) / 2)
        # 矢印の方向に応じて上下に数字を配置し、円との重なりを避ける
        direction = np.array(end_pos) - np.array(start_pos)
        direction_unit = direction / np.linalg.norm(direction)
        if direction_unit[1] > 0:
                text_pos = (mid_point[0], mid_point[1] + 0.02 * direction_unit[0])
        else:
                text_pos = (mid_point[0], mid_point[1] + 0.02 * direction_unit[0])
        ax.text(text_pos[0], text_pos[1], str(count), fontsize=10, ha='center', va='center', color='black')
```

```
# 軸とタイトルを設定
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.set_aspect('equal')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_title(f'Movement and Stay Counts Visualization for Hour {hour}')

# 軸を非表示にする
ax.axis('off')

plt.legend()
plt.show()
```

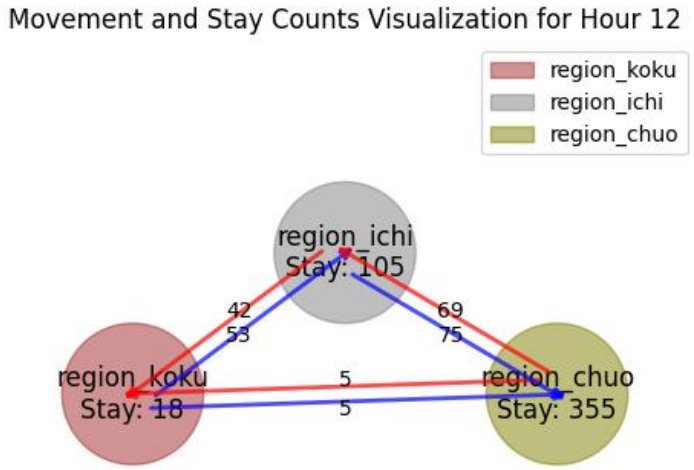
各時間帯のエリア間の移動人数と滞留人数を計算

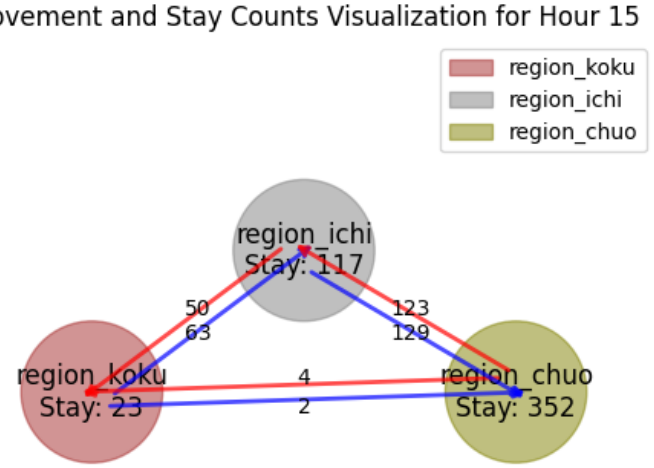
stop_move.ipynb

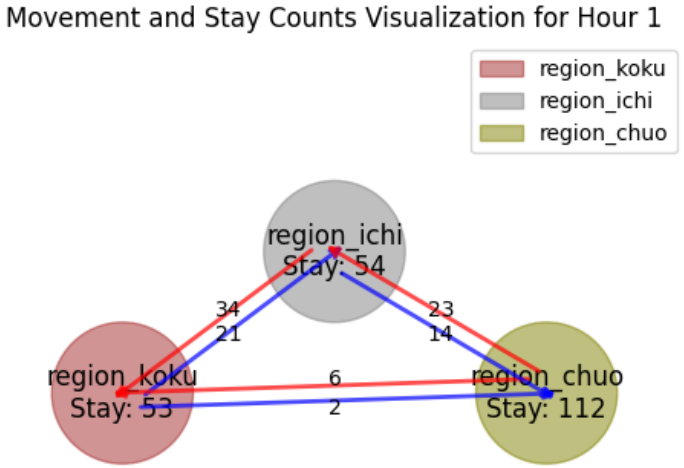
```
# 各時間のエリア間の移動量データを計算
def calculate_hourly_movement_counts(movement_counts):
    hourly_data = {}
    for hour in movement_counts['hour'].unique():
        data_for_hour = movement_counts[movement_counts['hour'] == hour]
        # データを辞書形式に変換：((start_region, end_region): count)
        hour_movement_dict = {}
        for start_region in ['region_koku', 'region_ichi', 'region_chuo']:
            for end_region in ['region_koku', 'region_ichi', 'region_chuo']:
                if start_region != end_region:
                    count = data_for_hour.loc[(data_for_hour['prev_region'] == start_region) & ¥
                     (data_for_hour['region'] == end_region), 'movement_count'].sum()
                    hour_movement_dict[(start_region, end_region)] = count
        hourly_data[hour] = hour_movement_dict
    return hourly_data
# 各時間のエリアの滞留人数を計算
def calculate_hourly_stay_counts(stay_counts):
    hourly_stay_data = {}
    for hour in stay_counts['hour'].unique():
        data_for_hour = stay_counts[stay_counts['hour'] == hour]
        hour_stay_dict = {}
        for region in ['region_koku', 'region_ichi', 'region_chuo']:
            stay_count = data_for_hour.loc[data_for_hour['region'] == region, 'stay_count'].sum()
            hour_stay_dict[region] = stay_count
        hourly_stay_data[hour] = hour_stay_dict
    return hourly_stay_data
hourly_movement_data = calculate_hourly_movement_counts(movement_counts)
hourly_stay_data = calculate_hourly_stay_counts(stay_counts)
# 各時間のデータを描画
for hour in hourly_movement_data.keys():
    plot_hourly_movements_and_stays(hourly_movement_data[hour], hourly_stay_data[hour], hour)
```

stop_move.ipynb

視覚化結果







日中（12時台と15時台）は、国分町から一番町への移動人数が多く、滞留人数を上回っていることが確認できる。
また、一番町を経由しない国分町と仙台市中央の間を往復する人数は少ないことが確認できる。
この視覚化手法は、エリア間の人の移動や滞留の状況を理解する用途に役立つ。

目次

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
- 繋華街人流の推移の視覚化・考察

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

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」