

位置データもPythonで！

PyConJP2021 2021/10/15
合同会社長目 小川 英幸

自己紹介

名前: 小川 英幸(@ogawahideyuki)

会社: 合同会社 長目 代表社員

一緒に働いてくれる方募集中！ <https://chomoku.com/recruit/>

国勢調査を触るイベントも！: <https://techplay.jp/event/832814>

趣味: 禁酒（4年目） / ジョギング / 中国語 / BTS

アラフォーからの独学でプログラミングを始め、株やら社会分析なんかに役立つことが分かりハマって早5年ちょい。証券アナリストという分析系の資格を保有。

最近は社会課題・環境問題の解決的な仕事がしたい。

登壇: PyConJP2019 / PyCon China in 北京 / PyCon mini Hiroshima 2020 / PyConJP2020 チュートリアル



5年前の写真がPyConJP Blogに!!!



[Python Boot Camp in 京都を開催しました](#)

Thank you! PyConJP!!!



執筆



どちらも可視化がインタラクティブに作れるフレームワークDashに関して執筆

祝、電子化、増刷！！！！


 朝倉書店
@AsakuraPub

【電子書籍】
『Python インタラクティブ・データビジュアライゼーション入門 —Plotly/Dashによるデータ可視化とWebアプリ構築— (@driller・小川 英幸・古木 友子 著)』
Kindleほか各種プラットフォームで電子書籍の販売を開始しました！



asakura.co.jp
インタラクティブ・データビジュアライゼーション入門...
Webサイトで公開できる対話的・探索的（読み手が自由に動かせる）可視化をPythonで実践。データ解析に便利な...

午前10:29 · 2021年10月15日 · Twitter Web App

 driller/どりらん
@patraqushe

おかげさまで
[#plotlydashbook](#) の3刷がきまりました。SNSやイベントなどで本書を盛り上げていただき、ありがとうございます
asakura.co.jp/detail.php?boo...

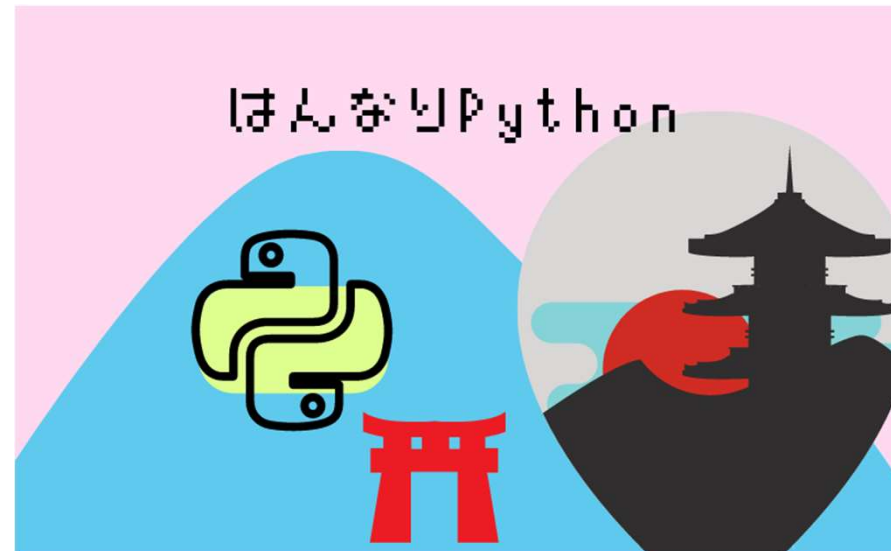


インタラクティブ・データビジュアラ...
asakura.co.jp

8:46pm · 14 Oct 2021 · Twitter Web App

はんなりPython

- 京都のコミュニティ
- オンライン化して全国へ
- 週1回くらいは開催
- 投げ銭制度を導入
- 寄付
 - PyCharity さん
 - 赤十字 さん
 - 日本学生支援機構 さん
- <https://hannari-python.connpass.com/>



トーク前にありがとう！

医療関係者ならびに社会を支える全ての方たちに

ありがとう！

トークの動機

- 仕事で多く位置データを扱った
- 我々の住むリアルワールドを扱うのに、位置データは重要だと痛感した
- 一方で、利用法などが語られることは少ない（私調べ）
- トークすることにより、色々と他の方も語り始めるのではという期待
- 基礎的な部分と活用方法を話します
- あと間違いも存在するかもしれませんので、お気づきの方はご指摘いただけますと幸いです

主要な利用パッケージ

- shapely
 - 地理空間情報を扱う
- geopandas
 - 地理空間情報を表データとして扱う
- xarray
 - 多次元データを扱う
- folium / plotly / pydeck
 - 地理空間情報の可視化

アジェンダ

- 位置データに関して
- ファイル形式
- 活用案
 - 個人の行動データ
 - 国勢調査
 - 気象データ
 - 衛星データ

アジェンダ

- 位置データに関して
- ファイル形式
- 活用案
 - 個人の行動データ
 - 国勢調査
 - 気象データ
 - 衛星データ

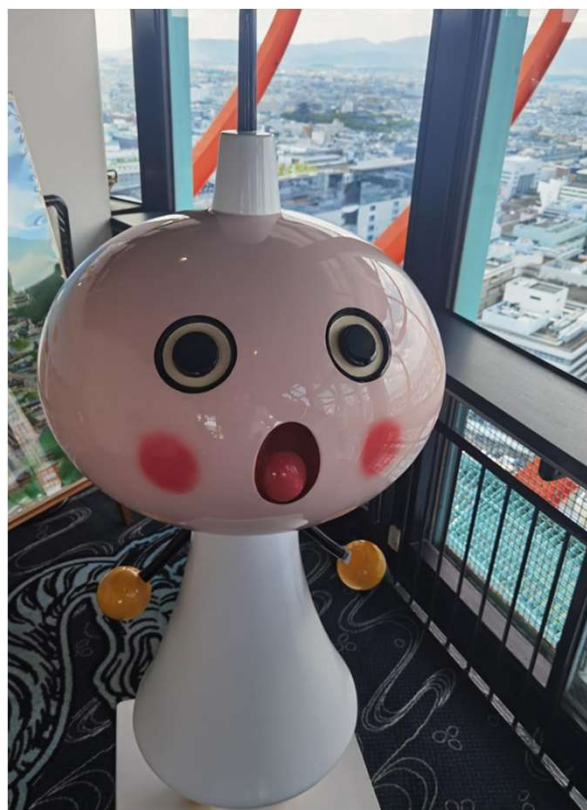
コロナ禍の私

- コロナ禍、外に出れない・・・
 - 2019年だと9月は東京でPyConJPに出て、10月はPyCon China 北京に行ったりしていた
 - しかし、不要不急といわれて・・・
- 家で仕事をしている私の行動範囲（2020/3 - 2021/10 99%）

北: MKボウル上賀茂



南: 京都駅



西: 金閣寺



東: 銀閣寺



グーグルマップを使って見る

- 京都の人しか分かんというわけで、グーグルで見てください！
 - Googlemap: マイプレイス
 - <https://goo.gl/maps/cFh66Zmg4j7cgfsG7>
 - google は賢い
- グーグルだと場所の名称でプロットされる
 - ex. 銀閣寺
 - 住所: 〒606-8402 京都府京都市左京区銀閣寺町2
 - 多分 地名 -> 住所 -> 座標でプロットされてると推測
 - 銀閣寺 座標で検索 -> 緯度経度 35.0270° N, 135.7983° E

ジオコーディング

- 位置データをプロットする際に住所を渡さない
- 緯度経度などの座標で指定する
- 地名・住所などから緯度経度への変換をジオコーディングと呼ぶ
- ジオコーディング事例(googleのgeocoding API: 要API_KEY)

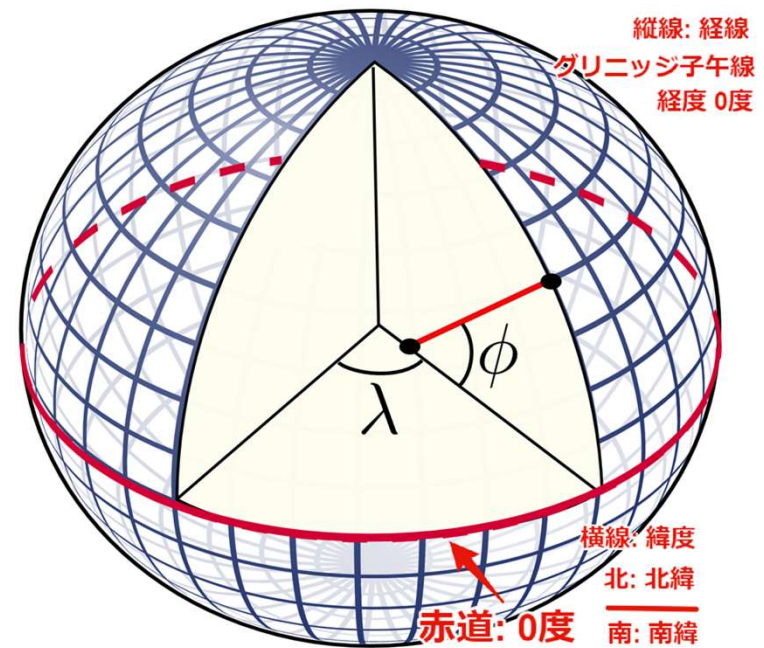
```
[8] 1 import requests

[29] 1 url = 'https://maps.googleapis.com/maps/api/geocode/json'
     2 params = {'address': '銀閣寺', 'key': id_key}
     3 r = requests.get(url, params=params)
     4 result = r.json()['results']
     5 result[0]['geometry']

{'location': {'lat': 35.0270213, 'lng': 135.7982058},
 'location_type': 'ROOFTOP',
 'viewport': {'northeast': {'lat': 35.0283702802915, 'lng': 135.7995547802915},
 'southwest': {'lat': 35.02567231970851, 'lng': 135.7968568197085}}}
```

座標

- 緯度経度
 - 緯度
 - 南北を表現（90度まで）
 - 赤道を0度
 - 経度
 - 東西を表現（180度まで）
 - 本初子午線を0度: グリニッジ天文台から102.478m東
- 小数で表現される場合もあるが度・分・秒で表現されることも多く、分秒を60で割る必要
- 緯度経度を数値で表現すると位置が特定できる
- 一方で、表現が複数存在し・・・



Peter Mercator, Public domain, ウィキメディア・コモンズ経由で

座標参照系（CRS）

- 位置を表現するための定義が複数存在する
 - でこぼこ楕円の地球をどうするかを定義する測地系
 - 地球を球体で表現するか？平面で表現するかの座標系
 - 多数存在する測地系と座標系の組み合わせ => 座標参照系
 - GeoRepositoryというサイトによると、6547のCRSがあるようだ
 - **The International Association of Oil & Gas Producers (IOGP)** による運営だそうだ
 - この辺りに注意しないと、全然位置データを活かせないといかなる
- 正確な情報は本などを参照ください

Shapely

- 地理空間情報の表現。shapelyを使う
 - <https://shapely.readthedocs.io/en/stable/>
- Point(点)、LineString(線)、Polygon(面) を使って空間を表現する
 - x, y の順に数値を渡す(経度・緯度)
 - それぞれをまとめてしまう Multi* もある
 - Point : MultiPoint
 - LineString : MultiString
 - Polygon : MultiPolygon
- ここからのノートブック
 - Colab:
https://colab.research.google.com/drive/1CU8jzoLs8Hv0NNmHqES3gtYqneX_PvNJ?usp=sharing

位置データに関して: 行動範囲をPointで表現

- 緯度経度で表現
- MultiPointにすると、4つの点で表示される

[29] 1 df

	地名	geometry
0	銀閣寺	POINT (135.79821 35.02702)
1	金閣寺	POINT (135.72924 35.03937)
2	MKボウル	POINT (135.74609 35.06682)
3	京都駅	POINT (135.75877 34.98585)



```
1 center = shapely.geometry.MultiPoint(df.geometry.values)
2 center
```



位置データに関して: 行動範囲をLineStringで表現

- Pointを複数LineStringに渡すと線で表現される
- 東西と南北を線にしてみる

```
[32] 1 touzai = shapely.geometry.LineString([df.loc[0, 'geometry'], df.loc[1, 'geometry']])  
     2 nanboku = shapely.geometry.LineString([df.loc[2, 'geometry'], df.loc[3, 'geometry']])
```

```
[35] 1 print(touzai, nanboku)
```

```
LINESTRING (135.7982058 35.0270213, 135.7292431 35.03937) LINESTRING (135.746086 35.0668248, 135.7587667 34.985849)
```

```
[34] 1 shapely.geometry.MultiLineString([touzai, nanboku])
```



位置データに関して: 行動範囲をPolygonで表現

- Pointを複数Polygonに渡すと面で表現される
- 4点を面に変換する

```
[57] 1 polygon = shapely.geometry.Polygon([
      2                                     df.loc[0, 'geometry'],
      3                                     df.loc[2, 'geometry'],
      4                                     df.loc[1, 'geometry'],
      5                                     df.loc[3, 'geometry']
      6                                     ])
```

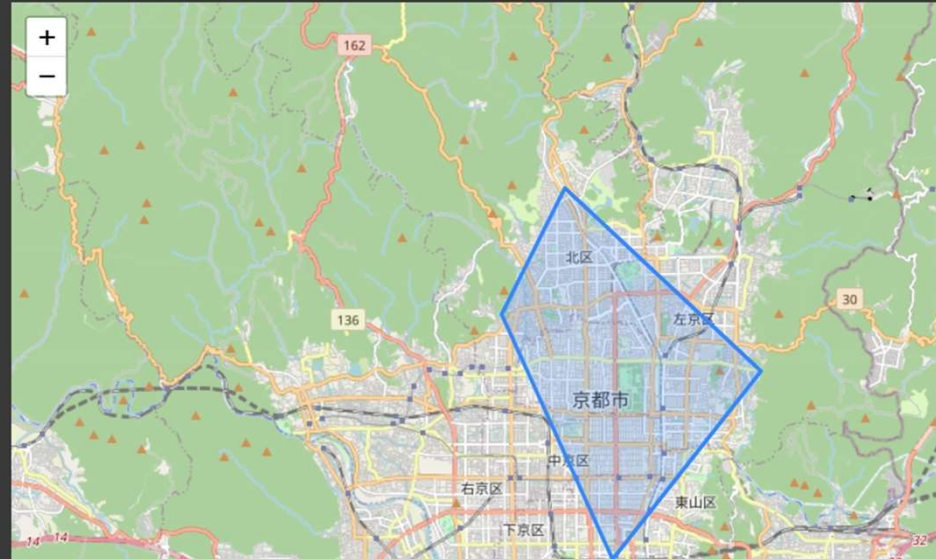
```
[58] 1 polygon
```



位置データに関して: プロットする

- foliumで可視化する

```
[77] 1 polygon_kyoto = shapely.geometry.Polygon([
2                                     df.loc[0, 'geometry'],
3                                     df.loc[2, 'geometry'],
4                                     df.loc[1, 'geometry'],
5                                     df.loc[3, 'geometry']
6 ])
7 center = polygon_kyoto.centroid
8 map = folium.Map([center.y, center.x], zoom_start=12)
9 folium.GeoJson(polygon_kyoto).add_to(map)
10 map
```



座標参照系を変換して距離を測る

- 距離を測ったりする場合、座標参照系変更する
- 今回の場合 EPSG4326 => EPSG6674
- そうすることにより線分の距離を計測できる
- 色々な属性で色々データが取得できる

```
[59] 1 df_kyoto = df.to_crs("EPSG:6674")
```

```
[60] 1 df_kyoto
```

	地名	geometry
0	銀閣寺	POINT (-18413.537 -107922.597)
1	金閣寺	POINT (-24702.618 -106537.855)
2	MKボウル	POINT (-23158.194 -103496.311)
3	京都駅	POINT (-22023.357 -112481.852)

```
[70] 1 # 銀閣寺 - 金閣寺の距離
2 shapely.geometry.LineString([df_kyoto.loc[0, 'geometry'],
3                               df_kyoto.loc[1, 'geometry']
4                               ]).length
```

```
6439.7249422724335
```



```
1 # 京都駅 - MKボウルの距離
2 shapely.geometry.LineString([df_kyoto.loc[2, 'geometry'],
3                               df_kyoto.loc[3, 'geometry']
4                               ]).length
```

```
9056.920195253735
```

アジェンダ

- 位置データに関して
- ファイル形式
- 活用案
 - 個人の行動データ
 - 国勢調査
 - 気象データ
 - 衛星データ

アジェンダ

- 位置データに関して
- ファイル形式
- 活用案
 - 個人の行動データ
 - 国勢調査
 - 気象データ
 - 衛星データ

2つのデータ形式

- ラスタデータ
 - 格子状にデータが格納されている
 - ファイルフォーマット: GeoTIFFなど
 - ライブラリ: rasterio, **xarray**
- ベクタデータ
 - ポイントやラインストリング、ポリゴンから作られる座標データ
 - ファイルフォーマット: Shape、GeoJSON、KMLなど
 - Shapeは3-5個くらいのファイルから構成される
 - ライブラリ: **geopandas**

ラスタデータ

- データ: 国土数値情報: 土地利用細分メッシュ（ラスタ版）データ
 - https://nlftp.mlit.go.jp/ksj/gml/datalist/KsjTmplt-L03-b_r.html
 - ここでは京都近辺を扱う: 5235
 - zipを解凍してtifファイルをxarrayのopen_rasterio関数で読み込み
 - データは各地点の100mメッシュに土地被覆分類の値が格納されている
 - 実際の読み込みとオブジェクト

```
[21] 1 da = xr.open_rasterio('/content/L03-b-14_5235.tif')  
      2 da
```

xarray.DataArray (band: 1, y: 800, x: 800)

[640000 values with dtype=uint8]

▼ Coordinates:

band	(band)	int64	1
y	(y)	float64	35.33 35.33 35.33 ... 34.67 34.67
x	(x)	float64	135.0 135.0 135.0 ... 136.0 136.0

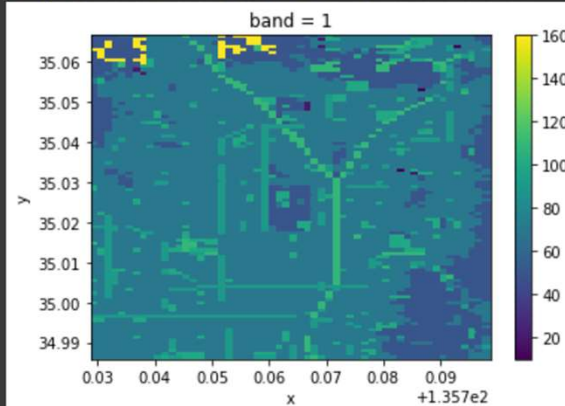
► Attributes: (12)

ラスタデータ 2

- 私の1年半の行動範囲を切り出す
 - plotメソッドで可視化できる
 - xarrayを使うと、データの切り出しも容易

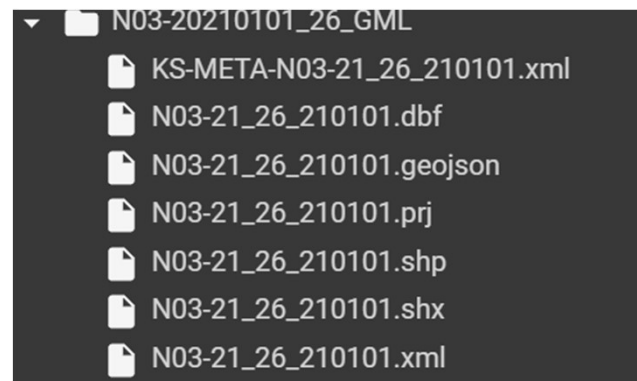
```
[38] 1 ogawa_area = da.sel(x=slice(kyoto_bounds[0], kyoto_bounds[2]), y=slice(kyoto_bounds[3], kyoto_bounds[1]))  
     2 ogawa_area.plot()
```

<matplotlib.collections.QuadMesh at 0x7f39e88380d0>



ベクターデータ

- 利用データ: 国土数値情報: 行政区域データ
 - https://nlftp.mlit.go.jp/ksj/gml/datalist/KsjTmplt-N03-v3_0.html#!
 - 例によって今回扱うのは京都
 - ZIPにはshape, geojson両方が含まれる
 - foliumで可視化する

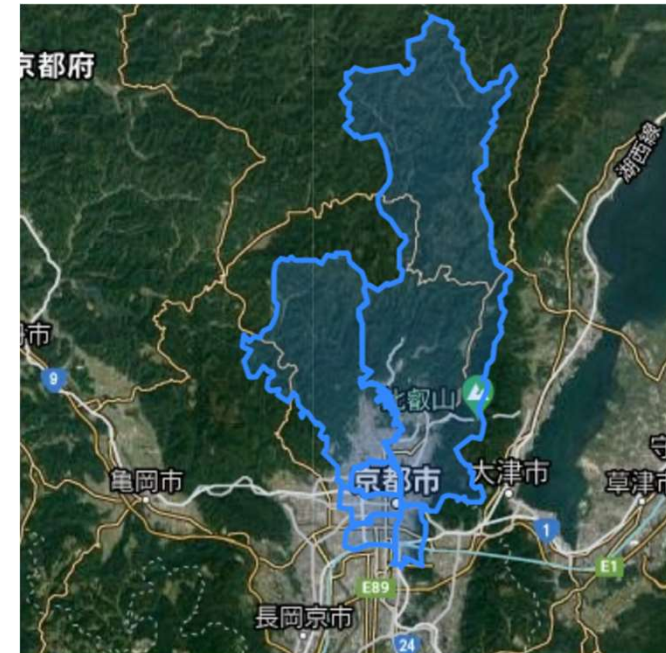


ベクターデータ 2

- 私の行動範囲の区はどれか調べる
 - intersectsメソッドを使う
 - そこだけをfoliumを使って可視化
 - 水色の部分がベクターデータ
 - グーグルの衛星画像を用いて観察する
 - 個人的に左京区の長さに驚き

```
1 data['me'] = data['geometry'].map(lambda x: x.intersects(kyoto_poly))
2 my_data = data[data['me'] == True]
3 my_data
```

	N03_001	N03_002	N03_003	N03_004	N03_007	geometry	me
0	京都府	None	京都市	北区	26101	POLYGON (((135.72539 35.17080, 135.72552 35.170...	True
1	京都府	None	京都市	上京区	26102	POLYGON (((135.75062 35.03822, 135.75079 35.038...	True
2	京都府	None	京都市	左京区	26103	POLYGON (((135.80481 35.31708, 135.80586 35.316...	True
3	京都府	None	京都市	中京区	26104	POLYGON (((135.73195 35.02251, 135.73195 35.022...	True
4	京都府	None	京都市	東山区	26105	POLYGON (((135.78466 35.01035, 135.78468 35.009...	True
5	京都府	None	京都市	下京区	26106	POLYGON (((135.77017 35.00428, 135.77019 35.003...	True



アジェンダ

- 位置データに関して
- ファイル形式
- 活用案
 - 個人の行動データ
 - 国勢調査
 - 気象データ
 - 衛星データ

アジェンダ

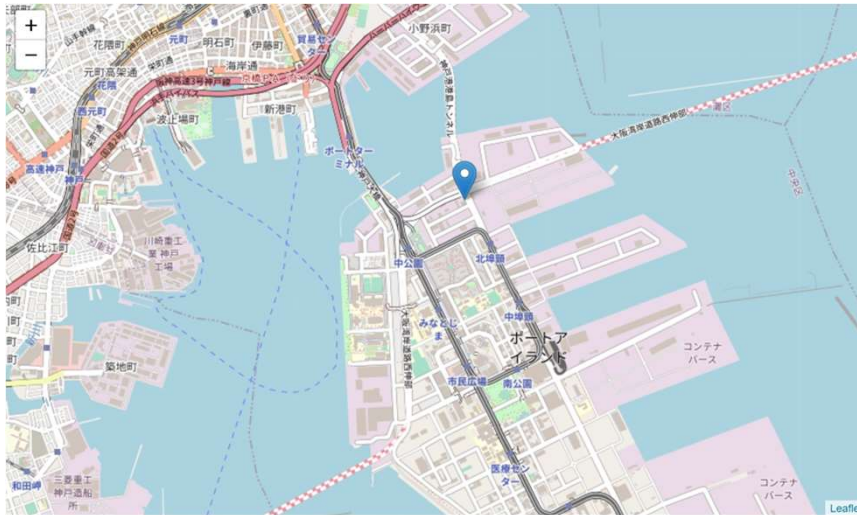
- 位置データに関して
- ファイル形式
- 活用案
 - 個人の行動データ
 - 国勢調査
 - 気象データ
 - 衛星データ

個人の行動データ(1)

- データ: GPS trajectory linked data project のリポジトリにあるデータ
 - <https://github.com/koujikozaiki/GPS2LOD> (CC4.0)
 - CSVファイル
 - 内容: あるカンファレンス時の行動
 - 時間と位置データ、高度、スピードなどのデータがある
 - みんな真面目に行動していたか、データを観察してみます
 - colab: https://colab.research.google.com/drive/1_LNM-AKdpcuJb-vOzPKJt3O81o1jOGia?usp=sharing
- データ処理
 - 今回活用するのはスピード、位置情報、個人
 - 30分おきのデータに変換（元データは時間がバラバラなため、今回発表用の加工）
 - 小数で存在する位置情報をshapely.geometry.Point型に
 - DataFrame => GeoDataFrameに
 - CSV => GeoJson ファイルに

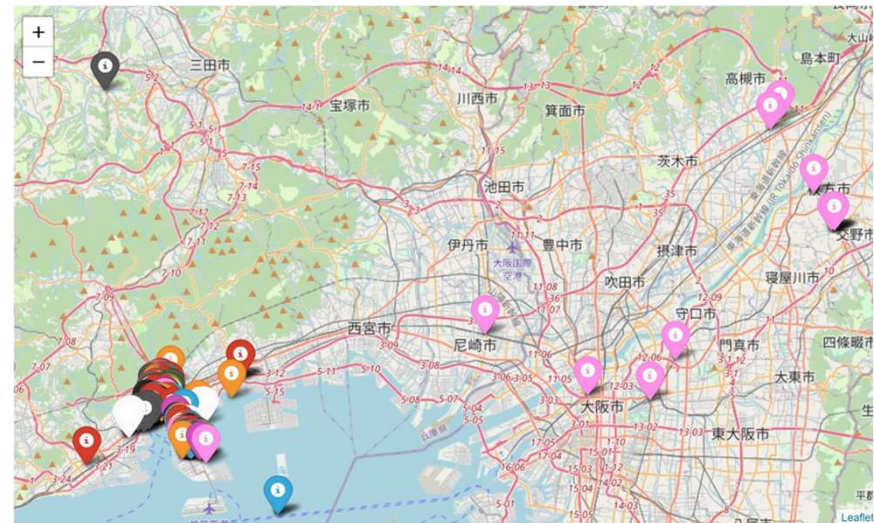
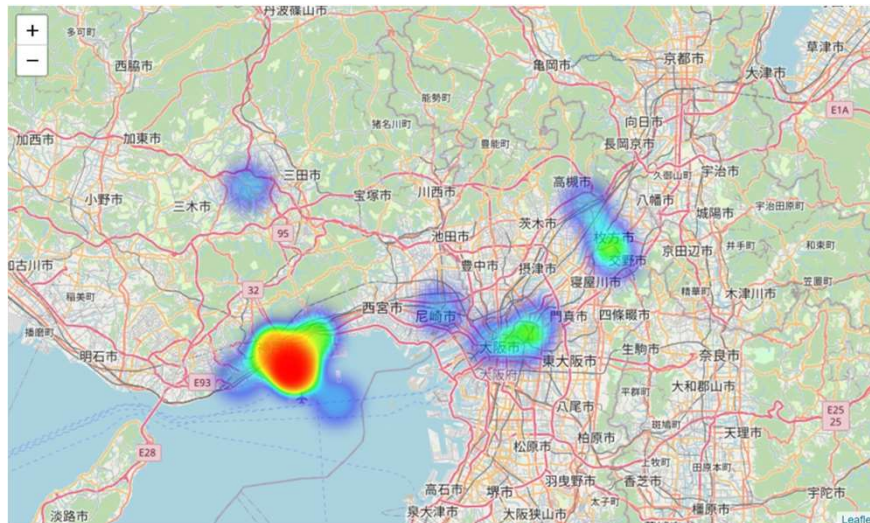
個人の行動データ(2)

- 全員の行動の中心点は？
 - MultiPointのcentroid属性で取得できるが、移動の中心になる・・・（左）
 - floatである位置データからmedianを作った（右）：ビンゴ！



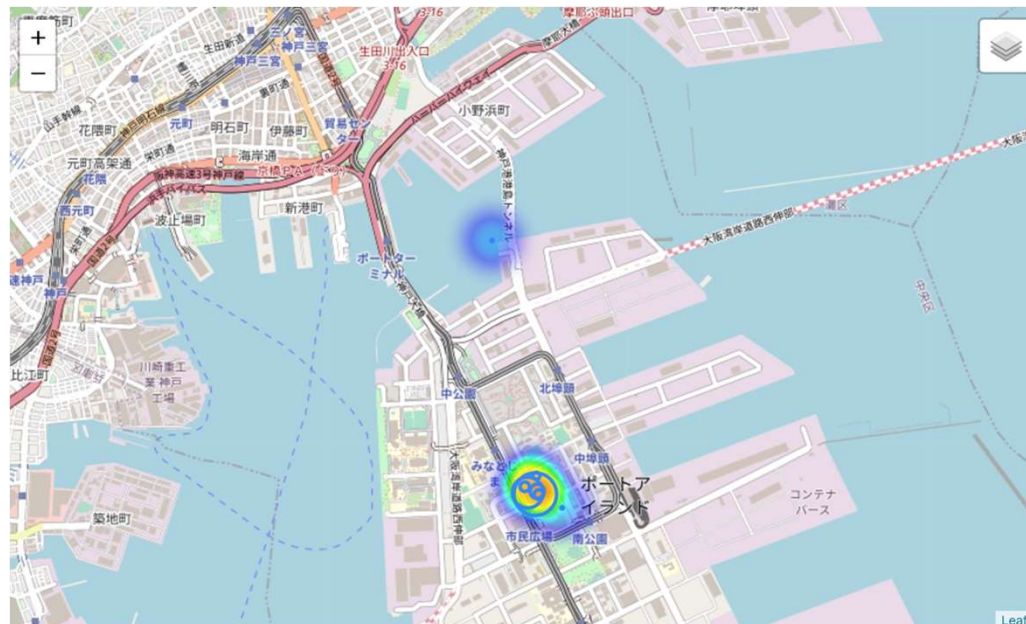
個人の行動データ(3)

- 全ての人の行動を観察する
- 頻度が多かった場所などをヒートマップを使って観察
- 各個人を色で分けて観察



個人の行動データ(4)

- ある時点の各個人のスピードを円の大きさに表現
- 集まり具合をヒートマップで表現



個人の行動データ(5)

- 各個人の1日の緯度距離を比較
- 座標参照系を変更して、メートルで表現 (EPSG4326 => EPSG6673)

各個人移動距離: 2016-10-18



アジェンダ

- 位置データに関して
- ファイル形式
- 活用案
 - 個人の行動データ
 - 国勢調査
 - 気象データ
 - 衛星データ

アジェンダ

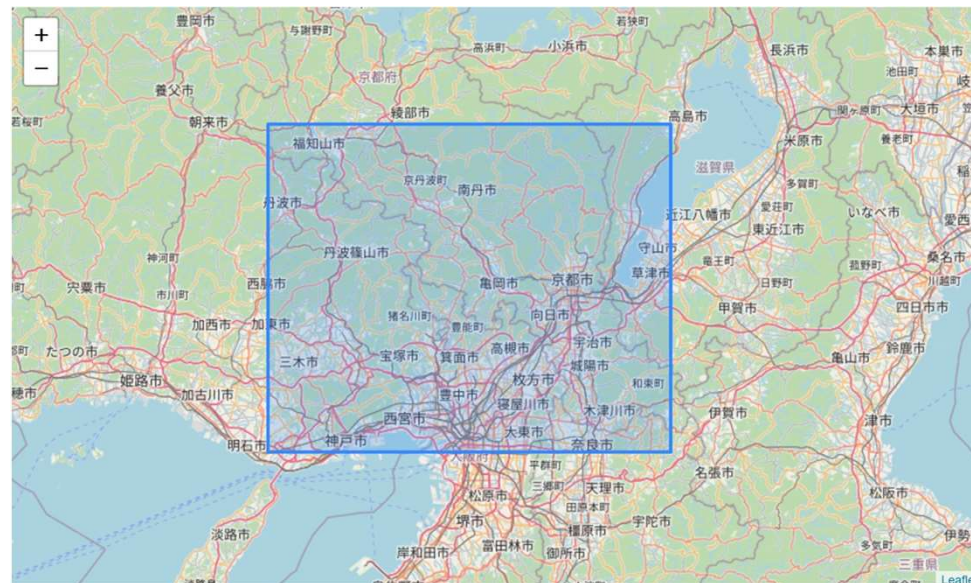
- 位置データに関して
- ファイル形式
- 活用案
 - 個人の行動データ
 - 国勢調査
 - 気象データ
 - 衛星データ

国勢調査(1)

- 国勢調査は国の最も重要な統計調査
 - 各家庭の構成人数や年齢などのデータも得れる
 - 全国、町・字で分けられたもの、1km、500m、250mメッシュでデータが作成されている
 - 統計データと境界データが別に置かれており、地域のデータはその2つをマージして作成する
 - csvとshapeファイルで取得できる（zipに固められている）
 - <https://www.e-stat.go.jp/gis>
- 今回は人口など基本統計に関する事項を使って、マーケティングする事例を示す
 - 0から14歳にマーケティング
 - 外国人居住者の方にマーケティング
- Colab:
https://colab.research.google.com/drive/1QspBpoW9BO_ofXmrLOX3a04YD5SE3eH?usp=sharing

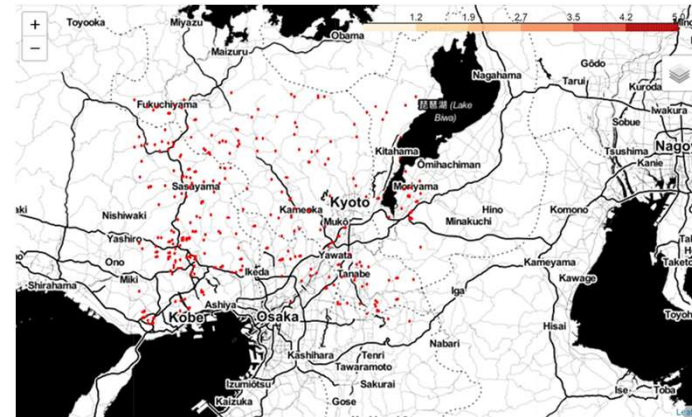
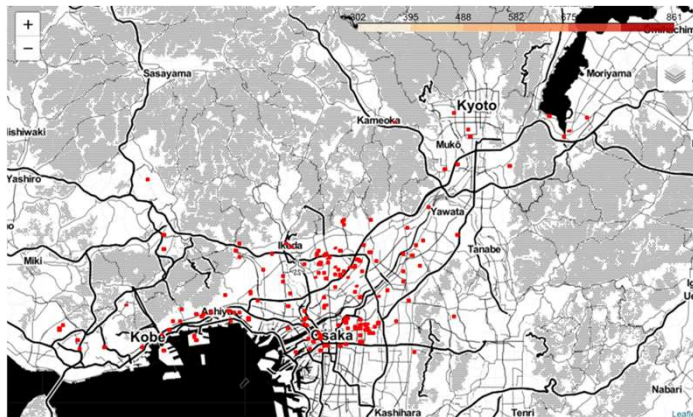
国勢調査(2)

- 京都の中心部が含まれるデータを作りました
 - 250メートルメッシュで、人口総数、男女比、年齢構成、世帯数などのデータ
 - KEY_CODEで位置データとマージした



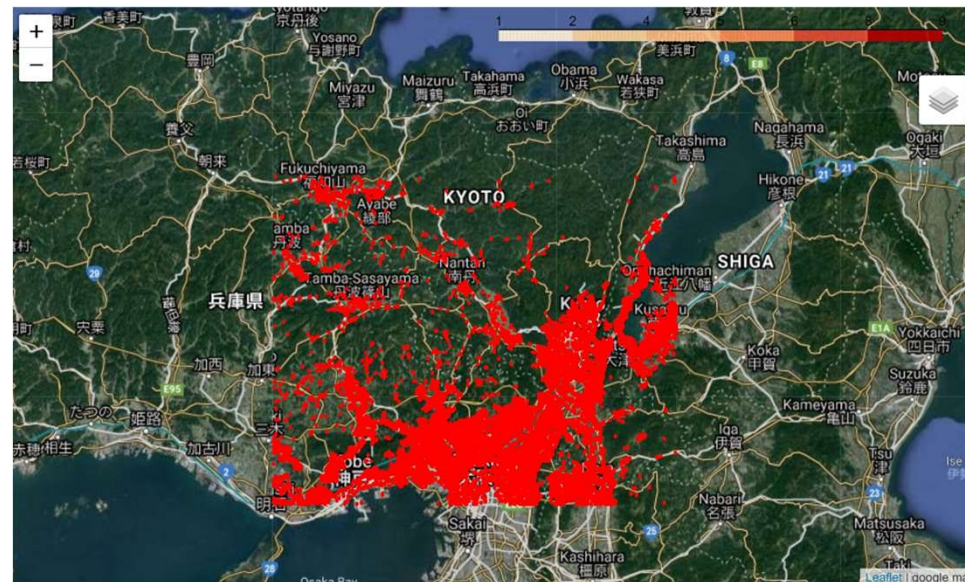
国勢調査(3)

- 0～14歳人口を使って、若い層が多そうなエリアを探す
 - 総数とレシオを作る（300人以上、40%）
 - この辺りはPandas的な処理で作成できる



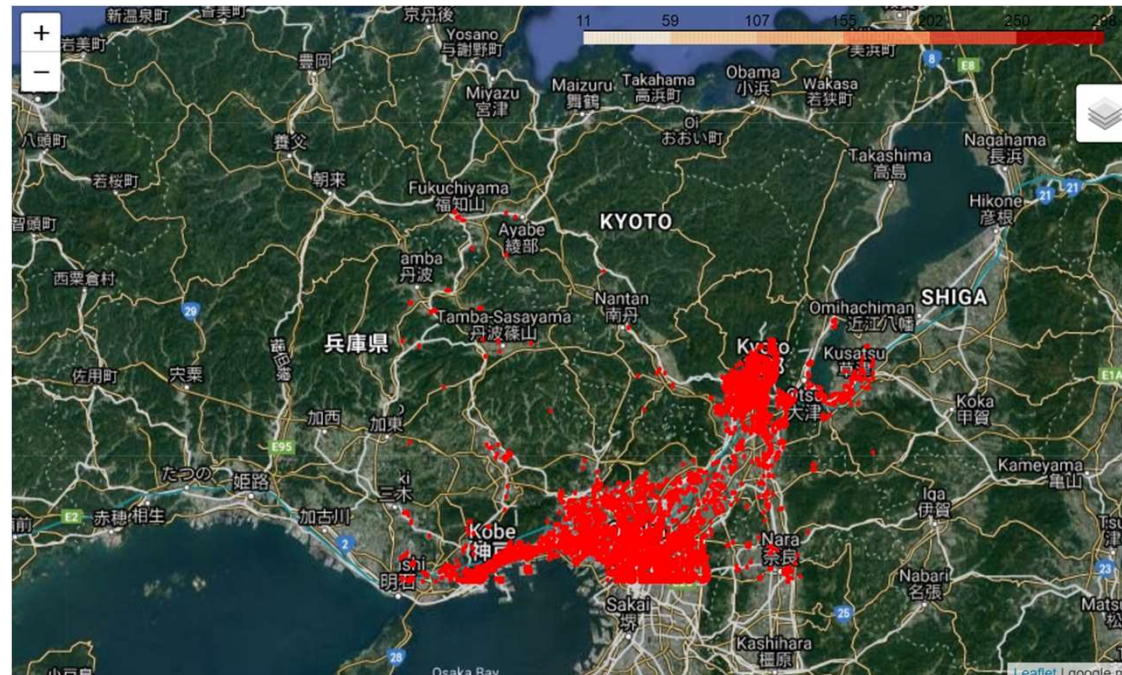
国勢調査(4)

- コロナ禍で困っている外国人在住者に情報を届ける
- 近くに外国人の方がおられない方ほど情報の取得が難しそう
- 250メートルメッシュで10人以下の地域を探す



国勢調査(5)

- 10人以下が予想以上に多かったので、10人以上も調べてみた



アジェンダ

- 位置データに関して
- ファイル形式
- 活用案
 - 個人の行動データ
 - 国勢調査
 - 気象データ
 - 衛星データ

アジェンダ

- 位置データに関して
- ファイル形式
- 活用案
 - 個人の行動データ
 - 国勢調査
 - 気象データ
 - 衛星データ

気象データ with xarray

- データはPTree 気象データ 「(c) JAXA・気象庁」
 - Colab: <https://colab.research.google.com/drive/1J3QU3Hv2iO1EKYGKma96CkkFeVqqzrD?usp=sharing>
- データ解説: https://www.eorc.jaxa.jp/ptree/userguide_j.html
 - 非営利利用のみ。
 - 営利の場合購入しましょう！
- データはNetCDF形式
 - 多次元アレイ
 - xarrayを使って読み込みます
- やること
 - 太陽光量の一日の変化を見る

```
3 da = xr.open_dataset(cdfs[0])
4 da
```

xarray.Dataset

Dimensions: (band: 6, geometry: 17, latitude: 2601, longitude: 2701, time: 1)

Coordinates:

latitude	(latitude)	float32	50.0 49.99 49.98 ... 24.01 24.0	📄 📄
longitude	(longitude)	float32	123.0 123.0 123.0 ... 150.0 150.0	📄 📄

Data variables:

band_id	(band)	int32	...	📄 📄
start_time	(time)	datetime64[ns]	...	📄 📄
end_time	(time)	datetime64[ns]	...	📄 📄
geometry_par...	(geometry)	float64	...	📄 📄
TAOT_02	(latitude, longitude)	float32	...	📄 📄
TAAE	(latitude, longitude)	float32	...	📄 📄
PAR	(latitude, longitude)	float32	...	📄 📄
SWR	(latitude, longitude)	float32	...	📄 📄
UVA	(latitude, longitude)	float32	...	📄 📄
UVB	(latitude, longitude)	float32	...	📄 📄
QA_flag	(latitude, longitude)	float32	...	📄 📄

Attributes: (13)

気象データ with xarray(2)

- xarrayの使い方は直感的で分かりやすい
 - pandasが使えたら行けると思う
 - UIも分かりやすい
 - ファイルをまとめて読み込むことも！
 - 35-35.5 / 135-135.5のデータを使う

```
4 da.sel(latitude=slice(35.5, 35),
5         longitude=slice(135, 135.5))['SWR']
```

xarray.DataArray 'SWR' (latitude: 51, longitude: 51)

```
array([[320.95, 376.35, 461.35, ..., 48.9, 37.15, 4.65],
       [269.1, 372.75, 436.85, ..., 47.65, 28.1, 13.],
       [367.9, 384.55002, 395.05002, ..., 17.1, 4.45, 22.75],
       ...,
       [500.7, 505.6, 505.35, ..., 500.55002, 493.05002, 501.75],
       [495.75, 495.55002, 501.75, ..., 389.30002, 471.65, 523.35004],
       [489.2, 489.05002, 496.45, ..., 390.1, 472.1, 472.30002]],
      dtype=float32)
```

Coordinates:

latitude	(latitude)	float32	35.5 35.49 35.48 ... 35.01 35.0
longitude	(longitude)	float32	135.0 135.0 135.0 ... 135.5 135.5

```
1 data = xr.open_mfdataset(cdfs, combine='nested', concat_dim='counts', parallel=True)
```

1 data

xarray.Dataset

21個のファイルを読んだ

Dimensions: (band: 6, counts: 21, geometry: 17, latitude: 2601, longitude: 2701, time: 1)

Coordinates:

latitude	(latitude)	float32	50.0 49.99 49.98 ... 24....
longitude	(longitude)	float32	123.0 123.0 123.0 ... 1...

Data variables:

band_id	(counts, band)	int32	dask.array<chunksize=(...
start_time	(counts, time)	datetime64[ns]	dask.array<chunksize=(...
end_time	(counts, time)	datetime64[ns]	dask.array<chunksize=(...
geometry_par...	(counts, geometry)	float64	dask.array<chunksize=(...
TAOT_02	(counts, latitude, longitude)	float32	dask.array<chunksize=(...
TAAE	(counts, latitude, longitude)	float32	dask.array<chunksize=(...
PAR	(counts, latitude, longitude)	float32	dask.array<chunksize=(...
SWR	(counts, latitude, longitude)	float32	dask.array<chunksize=(...

気象データ with xarray and plotly

- Plotlyを使って時間ごとのデータを可視化する
 - 動いた方が体感できる！！



気象データ with xarray and pydeck

- 一日の合計を表現する
 - 場所ごとにグループバイして1日分を合計
 - pydeckを利用する



アジェンダ

- 位置データに関して
- ファイル形式
- 活用案
 - 個人の行動データ
 - 国勢調査
 - 気象データ
 - 衛星データ

アジェンダ

- 位置データに関して
- ファイル形式
- 活用案
 - 個人の行動データ
 - 国勢調査
 - 気象データ
 - 衛星データ

衛星データ

- 利用データ Sentinel2
 - Colab: <https://colab.research.google.com/drive/1hajaElBbUwGWvm5JbnJ6y0WhsMDwc3Yi?usp=sharing>
 - [ライセンス](#) Copernicus Open Access Hub ([リンク](#))
 - 色々ファイルがあるが、ラスターデータがjp2ファイルにある
 - そのファイルをrioxarrayを使って読み込み処理する
 - 1pixel / 10mのTCIイメージを読み込む
 - 地元の田上山を観察する（都合1年前しかさかのぼれないので、その2つを比較する）

```
1 # ファイルの読み込み True Color Image
2 tci_path = '/content/drive/Shared drives/work-note/ogawa/document/pycon jp2021/data/tanakami_sat/S2A_MS
3 da = riox.open_rasterio(tci_path)
4 da
```

xarray.DataArray (band: 3, y: 10980, x: 10980)

[361681200 values with dtype=uint8]

▼ Coordinates:

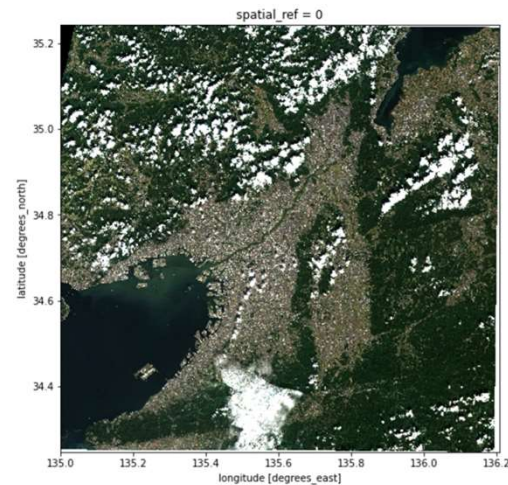
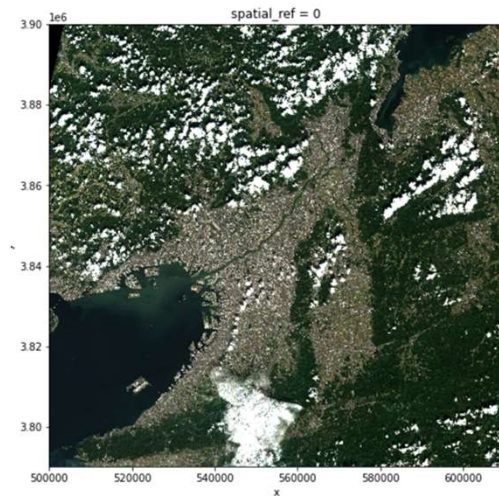
band	(band)	int64	1 2 3	
x	(x)	float64	5e+05 5e+05 ... 6.098e+05 6.098e+05	
y	(y)	float64	3.9e+06 3.9e+06 ... 3.79e+06	
spatial_ref	()	int64	0	

▼ Attributes:

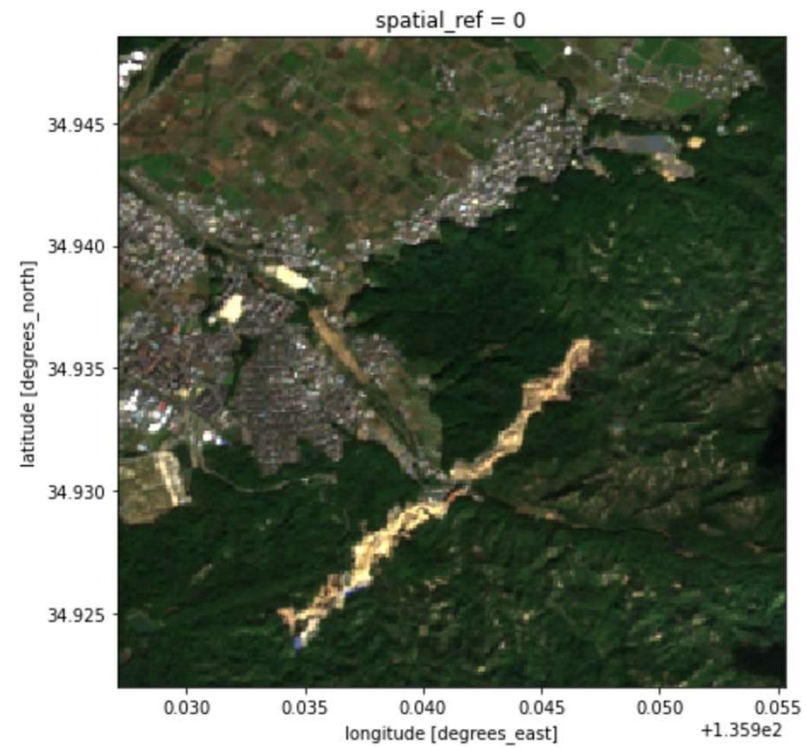
scale_factor :	1.0
add_offset :	0.0

衛星データ2

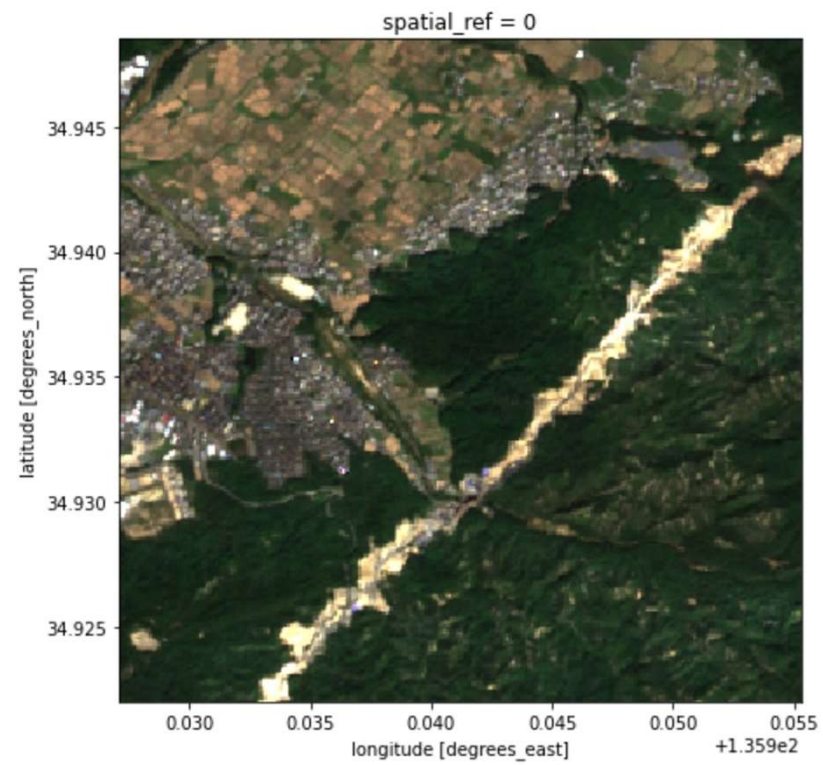
- そのまま可視化
 - 大阪・京都・奈良・滋賀辺りの衛星画像
 - 位置情報が経度・緯度でないので変更する (EPSG:32653 => EPSG:4326)
 - 前もって準備していた田上山の見たい部分の位置データ呼び出し、切り取る



衛星データ3 : 2020/10/12



衛星データ4 : 2021/10/2



まとめ

- Pythonで位置データを活用するのは容易
- 一方で、正確な知識はかなり複雑で、この辺りは専門家と協力して触りたいところ
- データが整備されれば、社会課題の解決などに協力できそう
- 「どう行動していくか？」が、データが触れたり、コードが書けたりする私たちの課題かもしれません。

まとめ 2



from ghibli
<https://www.ghibli.jp/works/mononoke/#frame>

ご清聴ありがとうございました



参考資料

books(only japanese)

- 地図リテラシー入門 羽田康祐 ペレ出版
- その問題、デジタル地図が解決します 中島円 ペレ出版
- GIS地理情報システム 矢野桂司 創元社

for study

- GEO-PYTHON(University of Helsinki) [Link](#)

Packeges

- shapely
 - Document: <https://shapely.readthedocs.io/en/stable/>
- geopandas
 - Document: <https://geopandas.org/>
- xarray
 - Document: <http://xarray.pydata.org/en/stable/>
- folium
 - Document: <https://python-visualization.github.io/folium/>
- plotly
 - Document: <https://plotly.com/python/>
- pydeck
 - Document: <https://deckgl.readthedocs.io/en/latest/>

Data

- Behavior data of 11 people
 - GPS trajectory linked data project [Link](#)
- kokuse-chosa
 - eStat Statistics GIS <https://www.e-stat.go.jp/gis>
 - kokuse-chosa (2015) 250meter mesh
- Himawari
 - JAXA HIMAWARI MONITOR: [Link](#)
- Sentinel2
 - Copernicus Open Access Hub: [Link](#)