Geographic Data in Python

A Mahfouz | a.mahfouz@mail.utoronto.ca

Toronto Data Workshop

25 June 2020

Agenda

- About me
- Project description
- Google Directions API: APIs, time, build-your-own-geojson
- OpenTripPlanner: more about spatial data representation, simple spatial operations

About Me

- Master of Information student at the University of Toronto
- Geographer by training
- Still not used to writing about me sections

Project Description

- Transit analysis for schools in multiple cities in the US
- What would their students' commutes look like?
- What areas of the city are within reasonable commuting distance via transit?

Toolset

- Python
- Google Directions API
- OpenTripPlanner
- QGIS

Skills/Concepts

- Working with APIs
- UNIX time and timezones 😱
- Build-your-own dataset
- Spatial data representation
- Common file formats
- Basic spatial operations

Inputs

School data

School Name	School Address	AM Bell Time	PM Bell Time
PS X	601 N Mesa	7:30 AM	2:30 PM

Student data

ID	Grade	Address	Stop
123456789	9	711 Hills	St Vrain & E Father Rahm

A Note on Geocoding

- AKA converting addresses into coordinates
- Unnecessary for the Directions API
- Google's geocoder is pretty error-tolerant
- Some other options: Nominatim, Mapbox, HERE

Working with the Google Directions API

- We'll just use `requests`
  - Versatile library for HTTP requests
- There is a Python client (`googlemaps`) but it doesn't return the full API response
- Request parameters
  - API key
  - `origin`
  - `destination`
  - `mode`
  - `departure_time` or `arrival_time`

Do the Time Warp

- We have bell times as text (e.g., "7:30 AM") but need UNIX timestamps (seconds since midnight 1/1/1970 UTC)
- Specific travel dates aren't important, but constraints apply:
  - Must be a non-holiday weekday
  - Cannot be too far into the past or future
- Schools are in all different time zones

Do the Time Warp

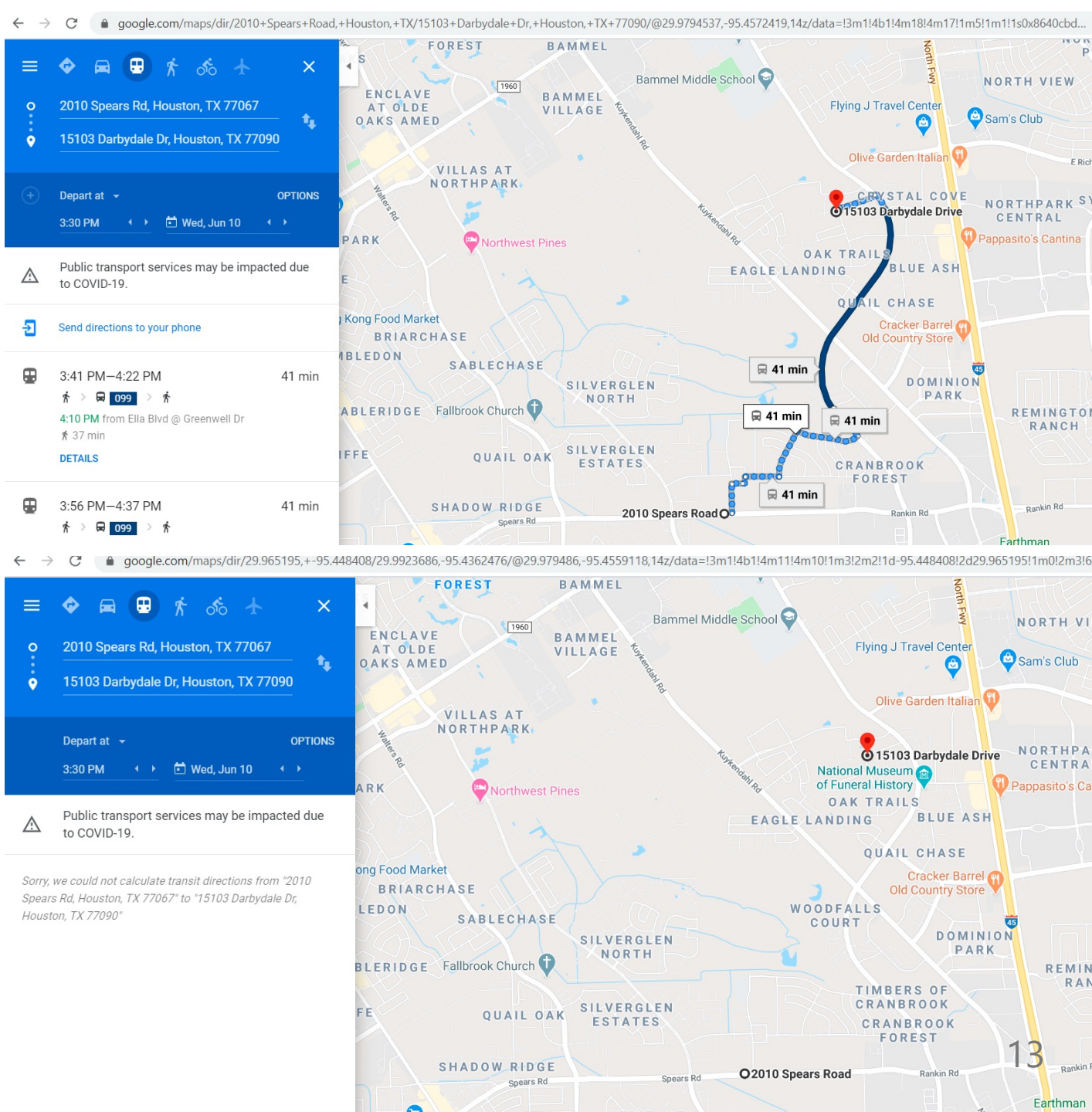
```
from datetime import date, datetime
from dateutil import parser, tz
from dateutil.relativedelta import WE, relativedelta

def get_next_wednesday():
    today = date.today()
    delta = relativedelta(days=1, weekday=WE(1))
    next_wednesday = today + delta
    return next_wednesday


def create_timestamp(str_time, tz):
    """
    Args:
        str_time (str): Time to convert. Expects hours and minutes; seconds are optional.
          If AM/PM is not specified, a 24-hour clock is assumed.
        tz (str): 3-character timezone string.
    Returns:
        int: a UNIX timestamp (seconds from January 1, 1970 UTC) representing str_time,
        next Wednesday from when the function is called.
    """

    tzinfo = {'EDT': tz.gettz('US/Mountain'),
              'CDT': tz.gettz('US/Central'),
              'MDT': tz.gettz('US/Mountain')}
    str_time = str_time + ' ' + tz
    n = get_next_wednesday()
    full_dt = str(n) + ' ' + str_time
    timestamp = datetime.timestamp(parser.parse(full_dt, tzinfos=tzinfo))
    return int(timestamp)
```

Making the Call

```
# simplified example code
def format_params(record, ampm):
    params = {'key': GKEY, 'mode': 'transit'}
    if ampm=='am':
        # add origin, destination, arrival time for AM
        params['origin'] = '{}'.format(record['home_address'])
        params['destination'] = '{}'.format(record['school_address'])
        # format arrival time to be school session time next Weds
        params['arrival_time'] = create_timestamp(record['am_bell_time'], record['tz'])
    return params


def query_dir_api(params):
    data = {}
    r = requests.get(dir_api, params=params)
    if r.status_code == 200:
        data = r.json()
    return data


def batch_process(df, ampm):
    all_journeys = []
    for idx, row in df.iterrows():
        params = format_params(row, ampm)
        response = query_dir_api(params)
        all_journeys.append(response)
    return all_journeys
```

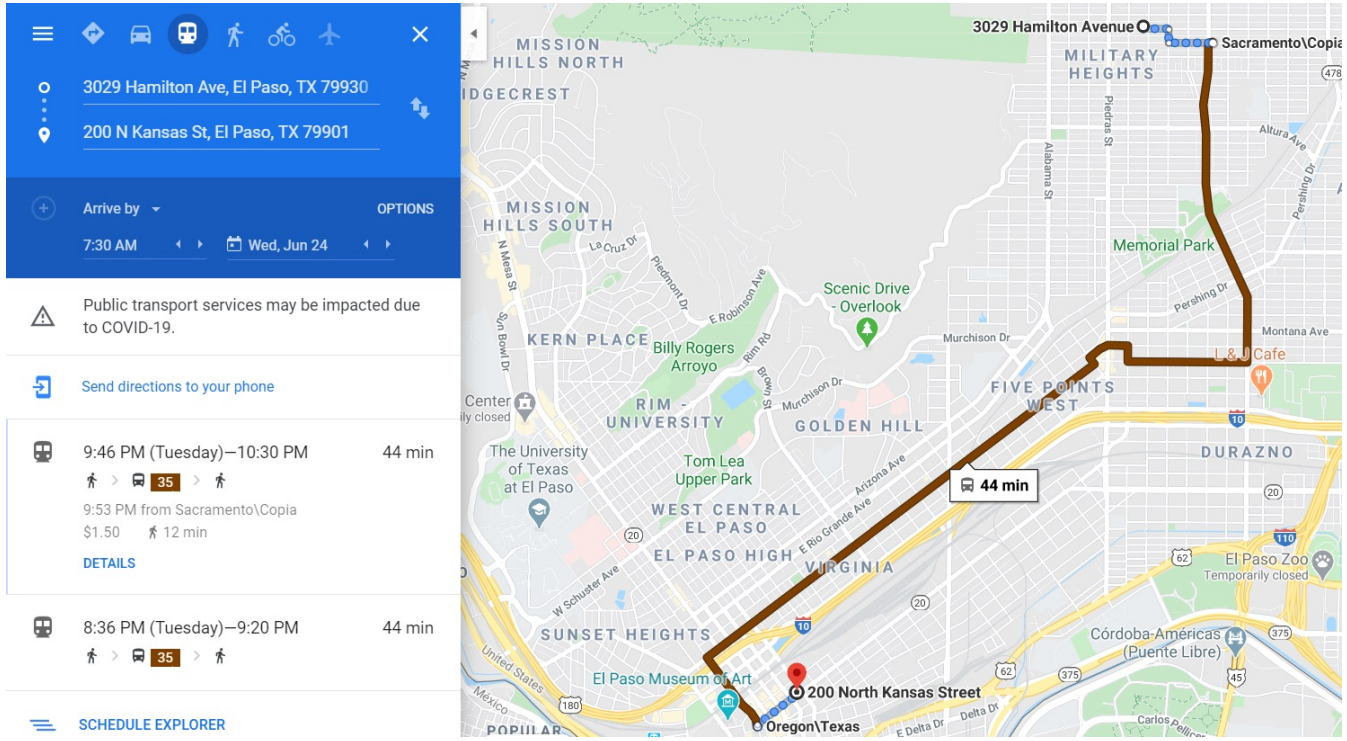
Caveat 1

Better to pass in an address than coordinates



Caveat 2

Results may meet the letter but not the spirit of the request



Google Directions Results


```
{
    "geocoded_waypoints": [...],
    "routes": [
        {
          "legs": [
              {
                  "arrival_time": {"text": "3:16pm", "time_zone": "America/Denver", "value": 1593641805},
                  "departure_time": {...},
                  "distance": {"text": "7.2 mi", "value": 11586},
                  "duration": {"text": "43 mins", "value": 2603},
                  "end_address": "6209 Weems Ct, El Paso, TX 79905, USA",
                  "end_location": {...},
                  "start_address": "601 N Mesa St, El Paso, TX 79901, USA",
                  "start_location": {...},
                  "steps": [...]
               }
          ],
          "overview_polyline": {
               "points": "woz`Evy}hS~F_FfFgEb@e@zBiB@M`BuAe@yCm@sEu@wESoAIe@IMi@eAa@w@HIIHgAoBsCeF
               iBgDk@b@_@ZeA~@qC|BwByDaRu\\_OyWmGaLuLiTqDwGg@aAGi@DaFD_DBoAHe@?UU@]B{A@yDDqB@]Cu@W
               UOm@w@a@iAe@aAGM_@]m@[SEg@CmBAsOCwKCAbGCj@RbAn@lA|AbC}@t@{@yAKSRQK[Uu@MY@SDI?[Ac@j@
               ?xC?jCD`AChED|G@zIBvDLtBBxHIDk@FeG?eAC@?uC@gKDqXF}X@yR?iC?o@AaD@}A@yCNoAZm@vAmBfAsA
               `@o@Lo@De@`@iFh@mGRaCDUbAcEnAgFpA}FzAsIRaARkA\\_CdBcMJ[PUNKHJJDN?LELSBOAOEMGKE}@BkD
               Hy@JIDM@SAOIORgBt@sFTqB^cDvAiKdBgNxA{K|@yGn@{E\\kCB@B@DYLy@YCQEALyBEoCG?eJAk@"
            }
        }
    ],
    "status": "OK"
}
```

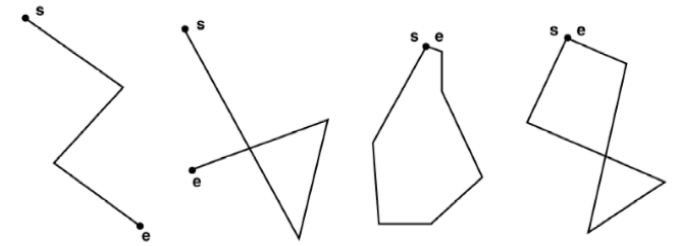
Google Directions Results Components

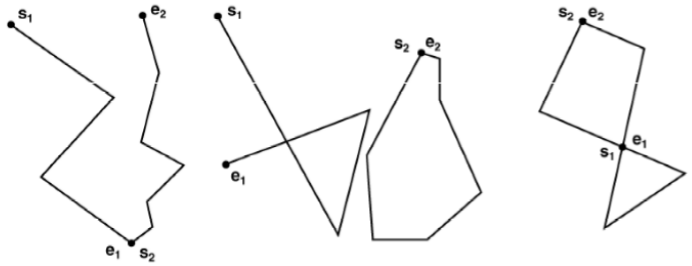
- `status`
- `geocoded_waypoints`
  - Geocoder status message
  - If a location was found, Google's place ID for it
  - If coordinates were passed, the reverse geocoding result
- `routes`
  - By default only one route is returned
  - `legs` : itinerary information
  - `overview_polyline` : geographic shape

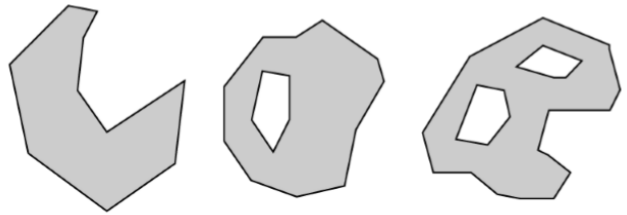
Great! But we can't map this.

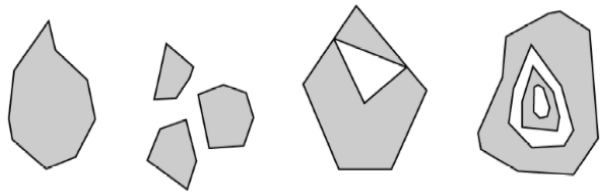
Simple Features

- ISO/OGC standard for spatial data representation
- Geometry types
  - Point and Multipoint
  - LineString and MultiLineString
  - Polygon and MultiPolygon
  - GeometryCollection
- Most geographic information systems use this model
  - Notable exceptions: raster data, OpenStreetMap









geoJSON

- Standardized format for representing simple features

Skeleton:

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "geometry": {
         "type": "LineString",
         "coordinates": [...]
        },
      "properties": {...}
    }
  ]
}
```

Decoding Polylines

- https://developers.google.com/maps/documentation/utilities/polylinealgorithm
- `polyline` to the rescue

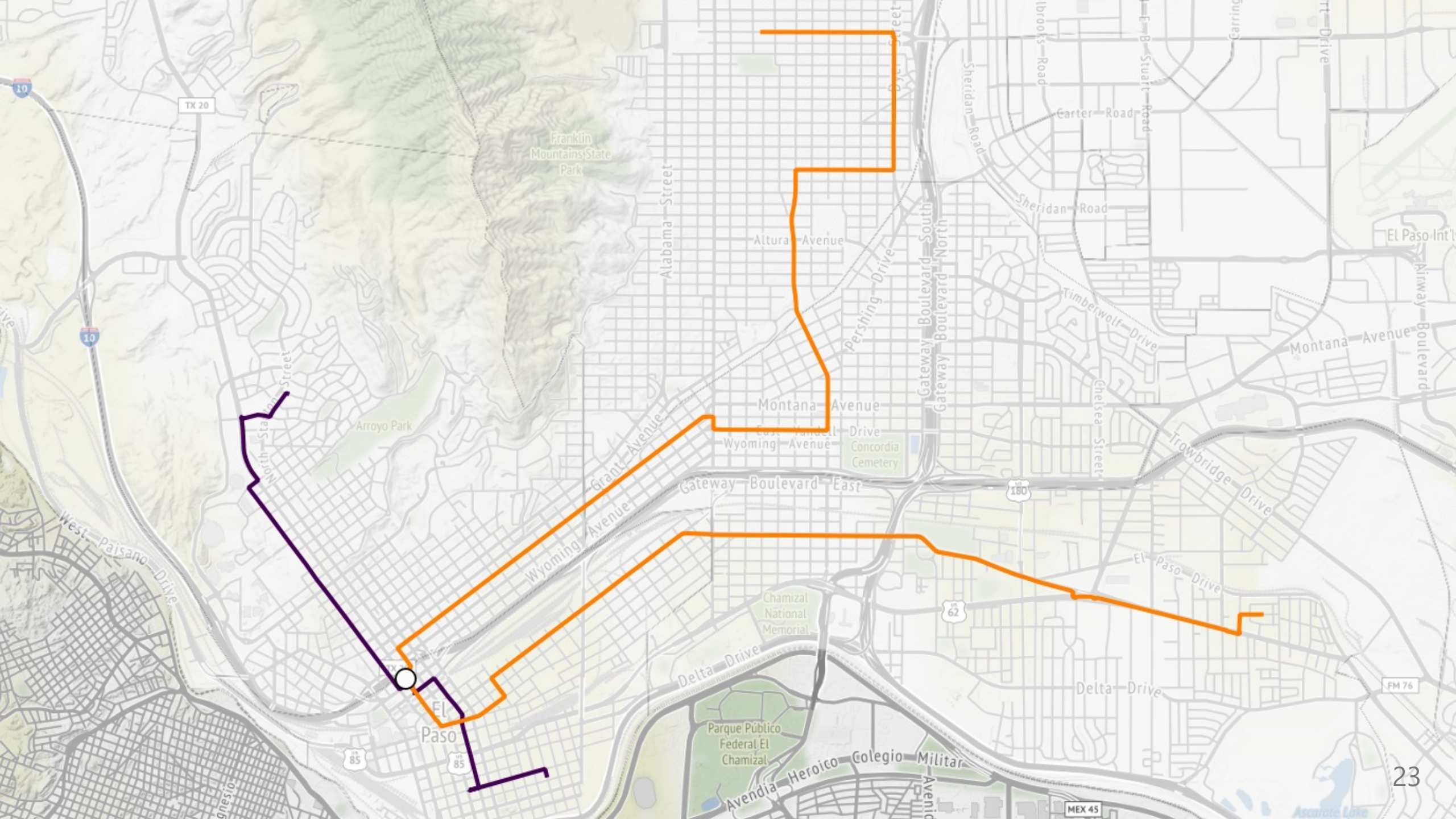
```
import polyline

def extract_overview_line(api_response):
  route_line = api_response['routes'][0]['overview_polyline']['points']
  pts = polyline.decode(route_line, geojson=True)
  return pts
```

Extracting Attributes

```
def extract_properties(result):
    if len(result['routes']) > 0:
        leg = result['routes'][0]['legs'][0]
        start_address = leg['start_address']
        end_address = leg['end_address']
        # account for walking-only routes not having departure and arrival times
        dep_time = leg.get('departure_time', {}).get('text')
        arr_time = leg.get('arrival_time', {}).get('text')
        duration_minutes = round(leg['duration']['value'] / 60, 1)
        dist_miles = round(leg['distance']['value'] * 0.00062137, 2)
        properties = {'origin': start_address,
                      'dest': end_address,
                      'departure_time': dep_time,
                      'arrival_time': arr_time,
                      'total_minutes': duration_minutes,
                      'total_miles': dist_miles,
                      'notes': ''}
    else:
        properties = {'notes': result['status']}
    return properties
```

Putting It All Together

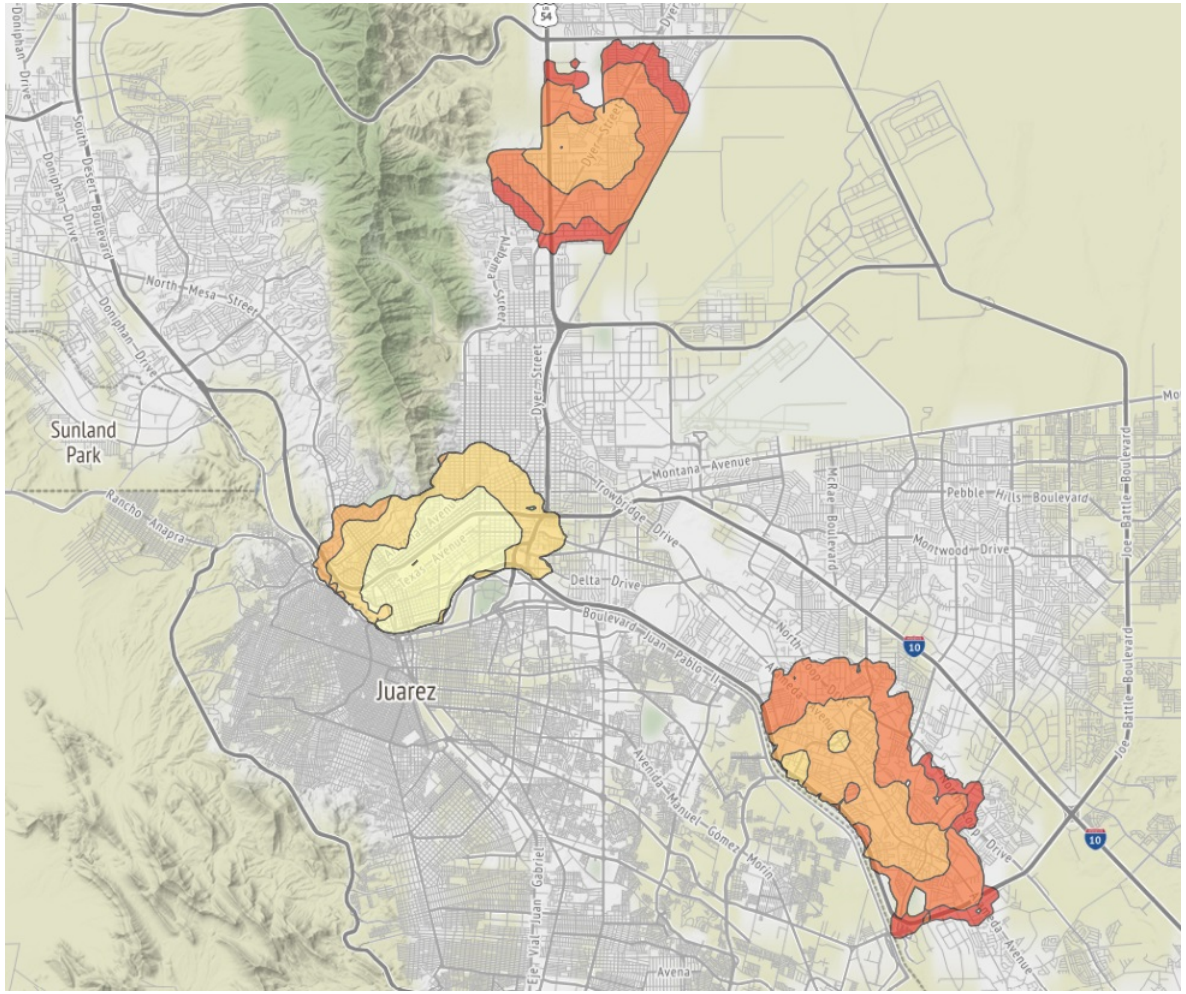
```
def make_feature(result):
    """Convert a Google Directions API result into a geoJSON feature."""
    obj = {'type': 'Feature'}
    if len(result['routes']) > 0:
        geom = extract_overview_line(result)
        obj['geometry'] = {'type': 'LineString',
                           'coordinates': geom}
    else:
        obj['geometry'] = {'type': 'GeometryCollection',
                           'coordinates': []}
    obj['properties'] = extract_properties(result)
    return obj


def gen_geojson(data):
    """Turn a list of Directions results into a geoJSON"""
    out_geojson = {
        "type": "FeatureCollection",
        "features": []
    }
    for journey in data:
        feature = make_feature(journey)
        out_geojson['features'].append(feature)

    return out_geojson
```

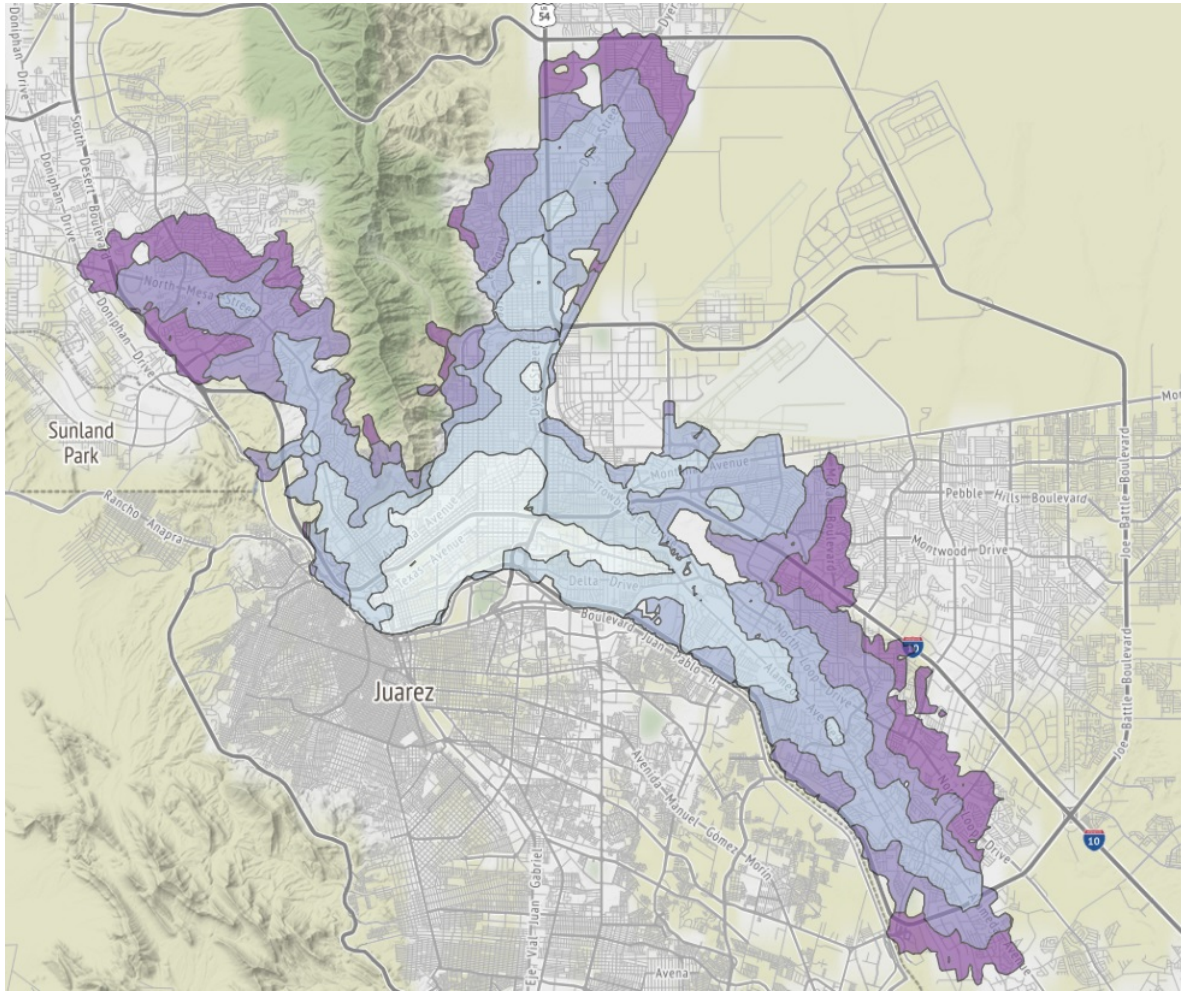

Franklin Mountains State Park
Alabama Street
Altura Avenue
Pershing Drive
Gateway Boulevard South
Gateway Boulevard North
Sheridan Road
Carter Road
Timberwolf Drive
Montana Avenue
Airway Boulevard
El Paso Int'l
Chelsea Street
Trowbridge Drive
Arroyo Park
North Stanton Street
Grant Avenue
Wyoming Avenue
Concordia Cemetery
Gateway Boulevard East
West Paisano Drive
El Paso Drive
Chamizal National Memorial
Delta Drive
Parque Público Federal El Chamizal
Avenida Heroico Colegio Militar
TX 20
US 180
US 62
US 85
FM 76
MEX 45

That works for individual trips. What if I want something more general?

OpenTripPlanner

- Open source multimodal routing software written in Java
- Works with OpenStreetMap and GTFS data
- APIs for routing and isochrone creation
- Excellent R resources: `otpr`, `opentripplanner`, tutorial


Sunland Park
Juarez
Dyer Street
Montana Avenue
Pebble Hills Boulevard
Montwood Drive
Joe Battle Boulevard
Delta Drive
Trowbridge Drive
McRae Boulevard
North Mesa Street
Doniphan Drive
South Desert Boulevard
Rancho Anapra
Avenida Manuel Gomez Morin
Avena
10
54


Sunland Park
Juarez
Dyer Street
Montana Avenue
Pebble Hills Boulevard
Montwood Drive
Joe Battle Boulevard
Delta Drive
Trowbridge Drive
McRae Boulevard
North Mesa Street
Doniphan Drive
South Desert Boulevard
Rancho Anapra
Avenida Manuel Gomez Morin
Avena
10
54

One Little Problem...

Feature (0) has invalid geometry. Please fix the geometry or change the Processing setting to the "Ignore invalid input features" option.
Execution failed after 0.05 seconds

Geopandas to the Rescue

- Python library for spatial data manipulation
- Think `pandas` with simple features support
  - Read/write spatial data files
  - Project coordinate reference systems
  - Perform spatial operations

Geopandas DataFrames

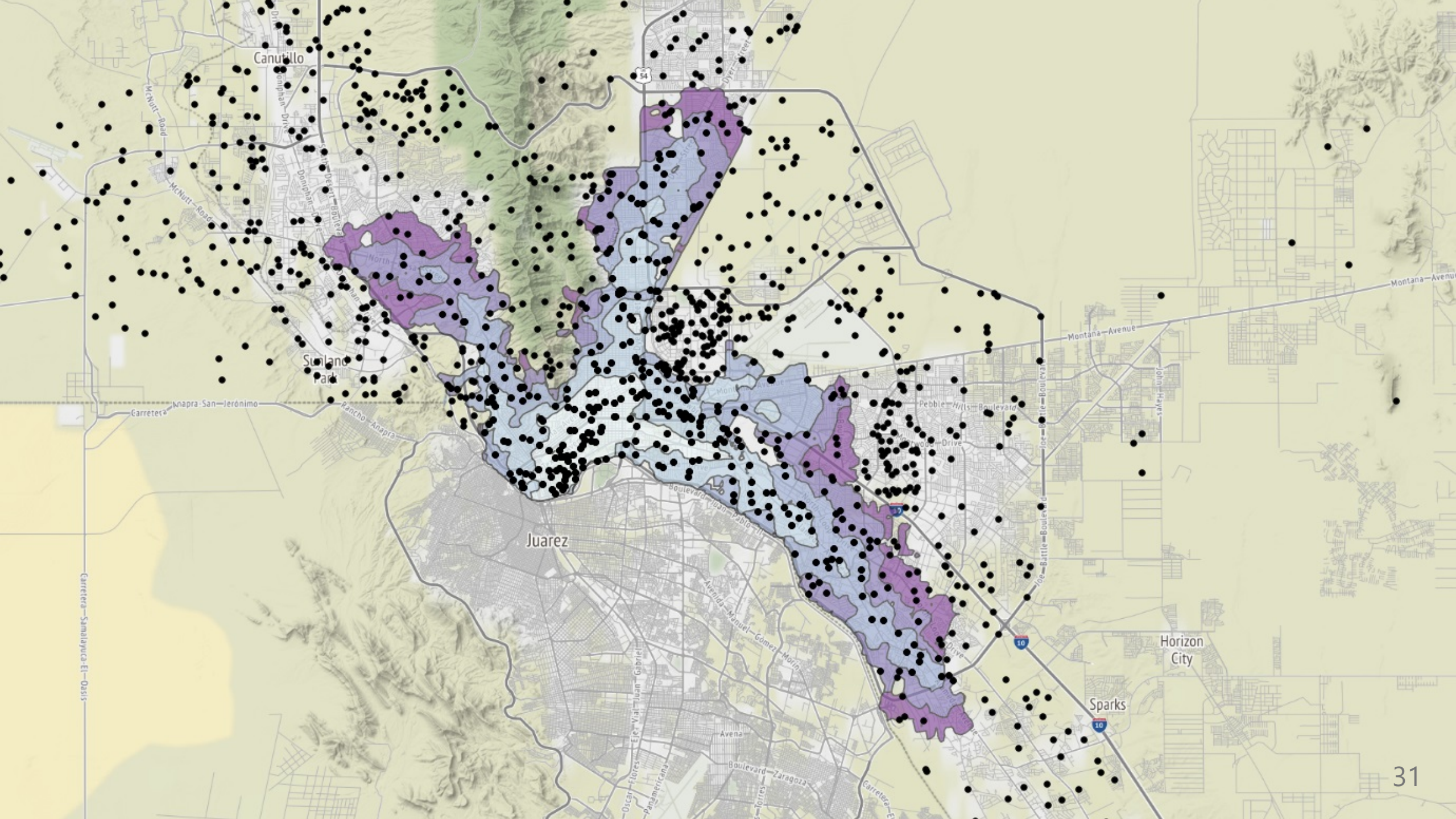
```
import geopandas as gpd

isos = gpd.read_file('pm_isochrones.geojson')
print(isos)
```

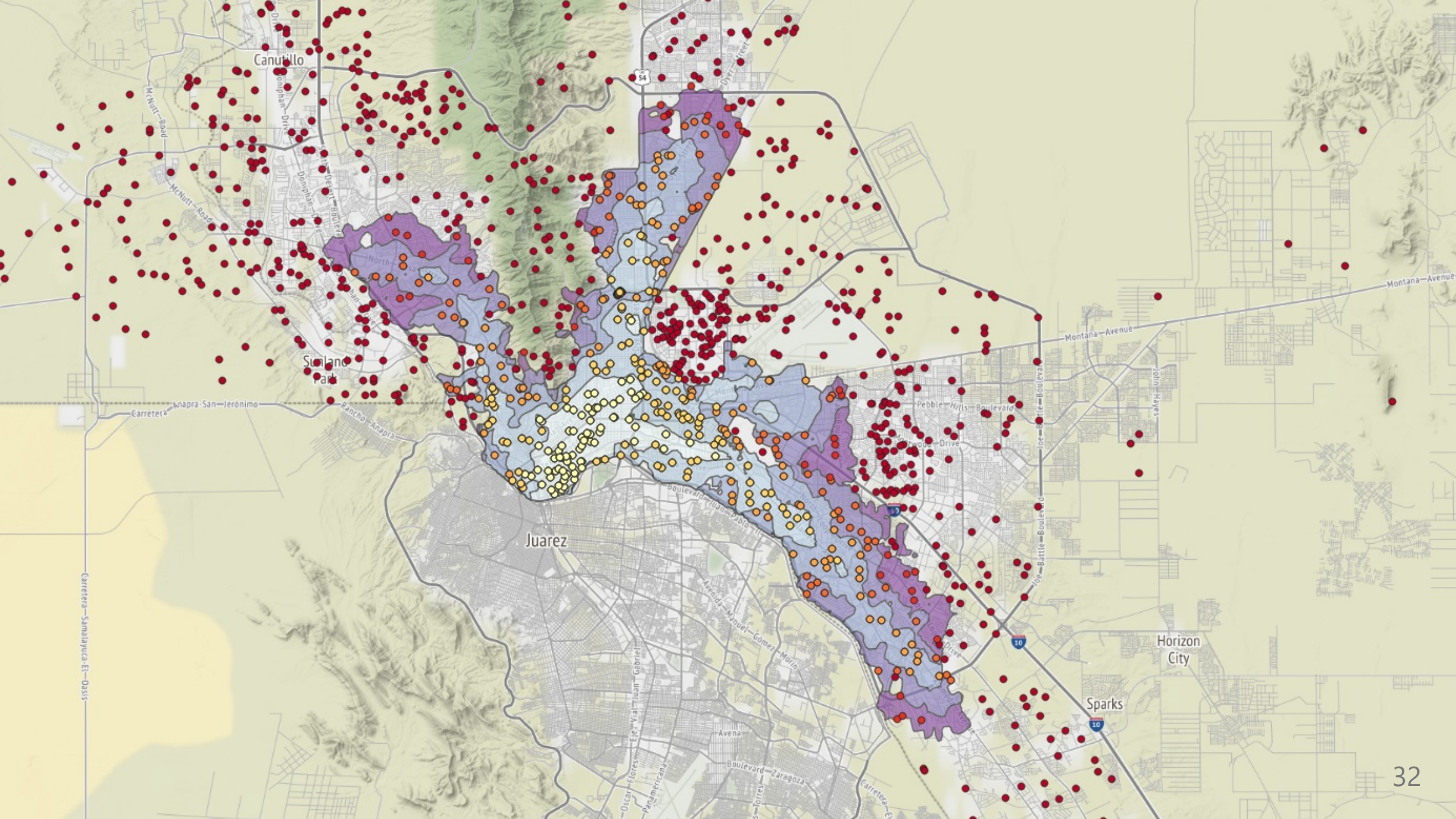
```
                                  id  time                                           geometry
0  fid-1681139f_172e865dd6a_-7ffc  5400  MULTIPOLYGON (((-106.53650 31.81310, -106.5356...
1  fid-1681139f_172e865dd6a_-7ffd  4500  MULTIPOLYGON (((-106.53650 31.81310, -106.5356...
2  fid-1681139f_172e865dd6a_-7ffe  3600  MULTIPOLYGON (((-106.52230 31.81850, -106.5217...
3  fid-1681139f_172e865dd6a_-7fff  2700  MULTIPOLYGON (((-106.44180 31.79880, -106.4426...
4  fid-1681139f_172e865dd6a_-8000  1800  MULTIPOLYGON (((-106.46880 31.78890, -106.4695...
```

Shape Shifting

- One operation that *does* work on malformed polygons: **buffering**
- Buffer creates an area *x* distance away from a shape
- It also fixes our geometry errors

```
import geopandas as gpd

isos = gpd.read_file('pm_isochrones.geojson')
isos['geometry'] = isos.buffer(0)
isos.to_file('fixed_pm_isochrones.geojson', driver='GeoJSON')
```


Canutillo
Juarez
Horizon City
Sparks
Montana Avenue


Canutillo
Montana Avenue
Juarez
Horizon City
Sparks

Spatial Joining in Geopandas

```
import geopandas as gpd

isos = gpd.read_file('fixed_pm_isochrones.geojson')
pts = gpd.read_file('demo_pts.geojson')

pts_in_isos = gpd.sjoin(pts, isos, how='left', op='within')
pts_in_isos.to_file('pm_joined.geojson', driver='GeoJSON')
```

Thank you!

Useful links and references

Google Documentation

- Directions API: https://developers.google.com/maps/documentation/directions/start
- Polyline encoding: https://developers.google.com/maps/documentation/utilities/polylinealgorithm

OpenTripPlanner

- OTP documentation: http://docs.opentripplanner.org/en/latest/
- Isochrone API reference: http://dev.opentripplanner.org/apidoc/1.4.0/resource_LIsochrone.html
- Marcus Young's OTP tutorial in R: https://github.com/marcusyoung/otp-tutorial

Useful links and references

Python Libraries

- requests: https://requests.readthedocs.io/en/master/
- geopandas: https://geopandas.org/index.html
- dateutil: https://dateutil.readthedocs.io/en/stable/
- polyline: https://pypi.org/project/polyline/

Spatial Data

- Simple Features: https://www.ogc.org/standards/sfa
- geoJSON: https://geojson.org/
- Spatial reference systems: https://epsg.io/

Useful links and references

GIS Software

- QGIS: https://www.qgis.org/en/site/

Misc

- This presentation was made with Marp for VS Code:
  - https://marketplace.visualstudio.com/items?itemName=marp-team.marp-vscode
  - https://marpit.marp.app/