Tagging unmarked Ohio trails using data

12/4/24

Jacob Nicholson

Many applications hikers use to tour American national parks rely on data from Open Street Maps, an open source editable map. These applications often translate Data from OSM(OpenSteetMaps) in a somewhat unfiltered way that displays private and unoffical trails as equal to recomened and offical trails maintained by the National Park. This reportedly causes users of trail apps derived from OSM such as All Trails to traverse these trails, putting themseleves and the wildilife in harms way. Becuase of this, OSM created the Trial Stewardship Initative in 2021, as a mission for volunteers to correctly tag the trails in national parks accordingly and encourage the applications to display the trails differently. At the moment, this tagging is done by hand and tagging an entire park for a team takes months. In This report, I will be exploring ways hoping to find ways to automate this tagging, starting with Ohio using existing data from OSM's overpass API and ODNR's(Ohio Department of National Recources) ArcGIS REST Service.

For more information on the Trail Stewardship Initiative this presentation by OSM is availilbe: https://openstreetmap.us/events/state-of-the-map-us/2024/osm-us-trails-stewardship-initiative/

Gathering data

OSM data

I entered this search in the turbo wizard on https://overpass-turbo.eu/, Overpass being Open Street Maps API in order to revieve all listed paths in Ohio
/* This has been generated by the overpass-turbo wizard. The original search was: “highway=path in "Ohio"” */ [out:json][timeout:25]; // fetch area “Ohio” to search in {{geocodeArea:Ohio}}->.searchArea; // gather results nwr"highway"="path"; // print results out geom;

I then rendered this file using jupyterlab-geojson 3.4.0

ONDR data

Entered an SQL Query to
https://gis2.ohiodnr.gov/arcgis/rest/services/OIT_Services/Trails_WayPoints_POI/MapServer/2
Where: 1=1

Out Fields: *

Return Geometry: true

Format: GEOjson

resultOffset: 0

resultRecordCount: 1000

This data is used in Ohio official Trial hiking app, and should serve as a good source for the offical and maintained trails

In [2]:

```
geojson_file = 'export.geojson'

ONDRgeojson_file = 'query.geojson'
```

In [3]:

```
import pandas as pd
import json
with open('export.geojson') as fo:
    data_str = fo.read()
    data = json.loads(data_str)
df = pd.DataFrame(data)
import geopandas as gpd
import folium
from folium.plugins import FloatImage
```

For this report, I'll be using geopandas to create dataframes from geojson files and clean them, then render them on a map with folium.

In [4]:

```
export_gdf = gpd.read_file("export.geojson")
query_gdf = gpd.read_file("query.geojson")
```

Cleaning Data

Part of the issue the Trail Stewardship Initiative is trying to tackle is that OSM does not have a very percise standard for the differentiating between the type of ways they collect, just that they exist. So, with the given keys, I will have to try to select the several that could refer to the trail being an offical and maintained Ohio trail, and sort out the rest based on the Trail Stewardship Initiative's recommended nomenclature. ODNR's dataset has quality information and can serve as what we should expect our map of offical OSM trails to look like.

In [5]:

```
#dataframe of OSM data
#export_gdf.head(1)
```

In [6]:

```
#dataframe of ODNR's data
#query_gdf.head(1)
```

In [7]:

```
#ODNR columns
query_gdf.columns.tolist()
```

Out[7]:

```
['OBJECTID',
 'TRAIL_NAME',
 'LENGTH',
 'USE_CODE',
 'STATUS',
 'DIFF_LEVEL',
 'ACCESS',
 'SURF_TYPE',
 'DIVISION',
 'LANDS_NAME',
 'RuleID_2',
 'Shape.STLength()',
 'Trails_jnFld',
 'Total_Length',
 'geometry']
```

The Data collected from ODNR's ArcGIS REST API has specific catagories are filled for all entries as scene in this list of columns.

In [8]:
```
#OSM columns, many only have a few features using them
limited_columns = export_gdf.columns.tolist()[:500]
print(limited_columns)
```

```
['id', '@id', 'FIXME', 'PDF_lineColor', 'Trail', 'abandoned', 'abandoned:highway', 'aband
oned:railway', 'access', 'access:conditional', 'access:disabled', 'alt_name', 'amenity',
'area', 'athletics', 'atv', 'bicycle', 'bicycle_road', 'bridge', 'bridge:name', 'bridge:r
ef', 'bridge:structure', 'capacity', 'check_date', 'check_date:surface', 'comment', 'cons
truction', 'covered', 'created_by', 'crossing', 'crossing:island', 'crossing:markings',
'cutting', 'cycleway', 'description', 'disused:highway', 'disused:leisure', 'dog', 'ele',
'embankment', 'fee', 'fixme', 'foot', 'footway', 'ford', 'golf', 'golf_cart', 'handrail',
'heritage', 'highway', 'hiking', 'horse', 'horse_scale', 'incline', 'informal', 'key', 'l
anduse', 'lanes', 'layer', 'lcn', 'lcn_ref', 'leisure', 'length', 'length_unit', 'level',
'line', 'lit', 'man_made', 'manufacturer', 'material', 'maxspeed', 'maxweight', 'maxwidt
h', 'motor_vehicle', 'motorcar', 'motorcycle', 'mtb', 'mtb:description', 'mtb:scale', 'mt
b:scale:imba', 'mtb:scale:uphill', 'name', 'name_1', 'name_2', 'network', 'noname', 'not
e', 'note:old_railway_operator', 'official_name', 'old_name', 'old_railway_operator', 'ol
d_ref', 'oneway', 'operator', 'ownership', 'par', 'parking', 'path', 'piste:grooming', 'p
iste:type', 'portage', 'proposed', 'railway', 'ref', 'ref:color', 'route', 'sac_scale',
'seasonal', 'segregated', 'service', 'sidewalk', 'ski', 'smoothness', 'snowmobile', 'sour
ce', 'sport', 'steps', 'stroller', 'surface', 'symbol', 'tactile_paving', 'tiger:cfcc',
'tiger:county', 'tiger:name_base', 'tiger:name_base_1', 'tiger:name_direction_prefix', 't
iger:name_direction_suffix', 'tiger:name_type', 'tiger:name_type_1', 'tiger:reviewed', 't
iger:separated', 'tiger:source', 'tiger:tlid', 'tiger:upload_uuid', 'tiger:zip_left', 'ti
ger:zip_left_1', 'tiger:zip_left_2', 'tiger:zip_left_3', 'tiger:zip_right', 'tiger:zip_ri
ght_1', 'tiger:zip_right_3', 'time', 'tracktype', 'trail_visibility', 'trailblazed', 'tun
nel', 'type', 'usage', 'vehicle', 'website', 'wheelchair', 'width', 'wikidata', 'wikipedi
a', 'geometry']
```

OSM's data, while more abundant, has many overlapping and infrequently used catagories, making defining trails from one another difficult.

In [9]:
```
# Count unique operators and their frequencies
operator_counts = export_gdf['operator'].value_counts()

# Display the counts
print(operator_counts)
```

```
operator
Cincinnati Off-Road Alliance          33
Ohio Department of Natural Resources   7
Wayne National Forest                  6
ODNR                                   3
Western Wildlife Corridor              1
Cleveland Metroparks                   1
Name: count, dtype: int64
```

In [10]:
```
filtered_export_gdf = export_gdf[export_gdf['operator'].notna() & (export_gdf['operator'
#filtered_export_gdf.head()
```

In [11]:
```
# Count unique operators and their frequencies
official_name_counts = export_gdf['official_name'].value_counts()

# Display the counts
print(official_name_counts)
```

```
official_name
North Country Trail–Athens East       12
North Country Trail–Athens Central     1
Name: count, dtype: int64
```

In [12]:

```
# Filter DataFrame for specific operator values
filtered_gdf_Offname = export_gdf[export_gdf['official_name'].isin(['North Country Trail

# Display the filtered DataFrame
#filtered_gdf_Offname.head()
```

In [13]:

```
# Count unique operators and their frequencies
formal_counts = export_gdf['informal'].value_counts()

# Display the counts
print(formal_counts)
```

```
informal
yes    304
no       7
Name: count, dtype: int64
```

In [14]:

```
# Filter DataFrame for specific operator values
filtered_gdf_informal = export_gdf[export_gdf['informal'].isin(['no'])]

# Display the filtered DataFrame
#filtered_gdf_informal.head()
```

In [15]:

```
# Count unique operators and their frequencies
access_counts = export_gdf['access'].value_counts()

# Display the counts
print(access_counts)
```

```
access
private        424
no             117
permissive     107
yes             56
customers       40
unknown          4
destination      2
designated       1
discouraged      1
Name: count, dtype: int64
```

In [16]:

```
# Filter DataFrame for specific operator values
filtered_gdf_access = export_gdf[export_gdf['access'].isin(['permissive', 'yes'])]

# Display the filtered DataFrame
#filtered_gdf_access.head()
```

In [17]:

```
# Count unique operators and their frequencies
ref_counts = export_gdf['ref'].value_counts()

# Display the counts
print(ref_counts)
```

```
ref
BT           30
TR 1024      12
GO            9
TR 1001       2
CH 47         2
T-286         1
TR 173        1
TR 1029       1
SR 56         1
TR 1002       1
TR 1001A      1
Name: count, dtype: int64
```

In [18]:

```
filtered_export_gdf2 = export_gdf[export_gdf['ref'].notna() & (export_gdf['ref'] != "")]
```

In [19]:

```
# Concatenate the DataFrames row-wise (stacking them)
merged_gdf = pd.concat([filtered_gdf_access, filtered_gdf_Offname, filtered_export_gdf,

# Display the merged DataFrame
#merged_gdf.head(100)
```

Comparing Cleaned Data to Offical Data

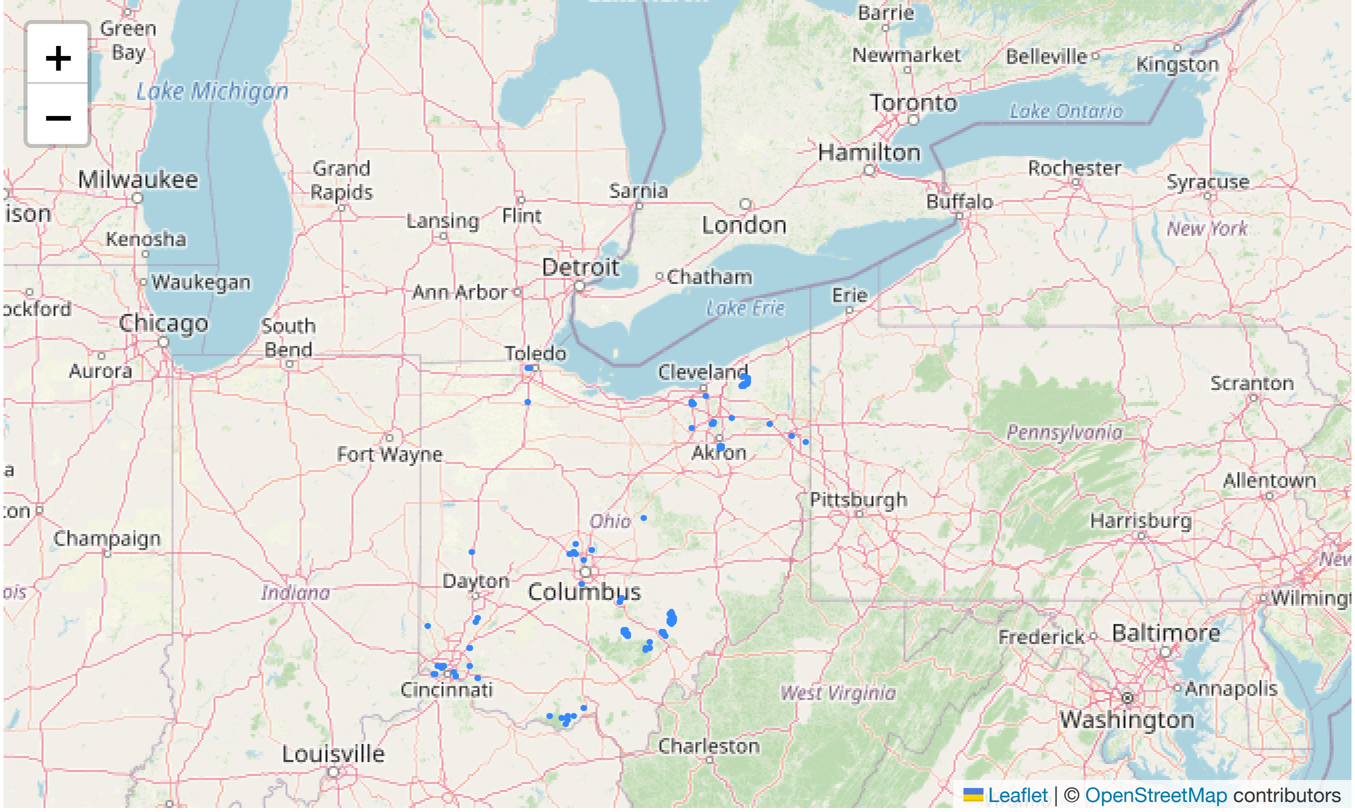
After combing through the OSM data and combining all of the keys that could refer to the data being Offcial, I rendered it using jupyterlab-geojson 3.4.0 as a good way to visual compare the data.

Cleaned OSM data

In [32]:

```
geojson_data = merged_gdf.to_json()  # Convert the GeoDataFrame to GeoJSON

# Initialize a Folium map
m = folium.Map(location=[40.0, -79.0], zoom_start=6)  # Centering the map roughly on Ohi

# Add the GeoJSON data to the map
folium.GeoJson(data=geojson_data).add_to(m)

# Save or display the map
m.save("map_with_merged_gdf.html")
m
```

Out[32]:

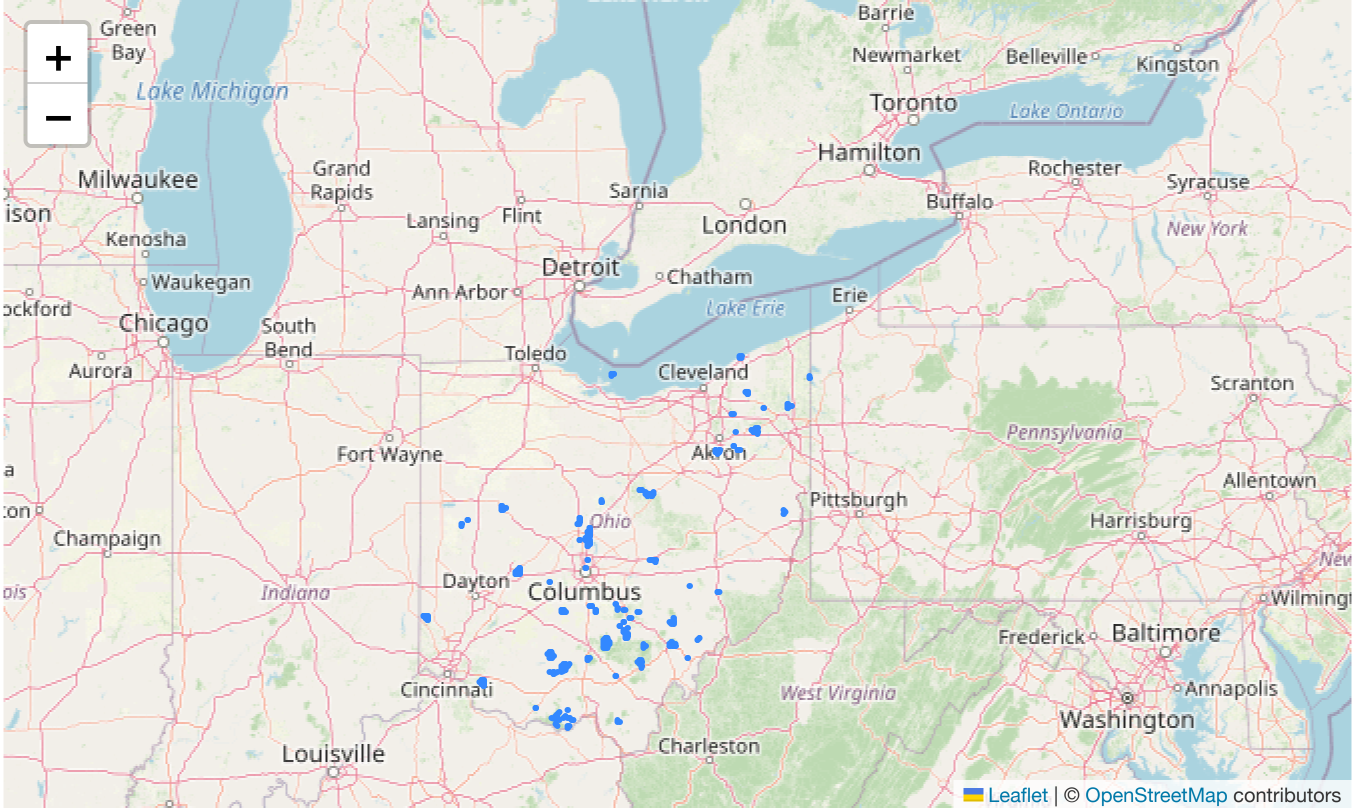


Official ODNR data

In [34]:

```
# Path to the GeoJSON file
ONDRgeojson_file = 'query.geojson'

# Load the GeoJSON file
with open(ONDRgeojson_file, 'r') as f:
    geojson_data = json.load(f)

# Initialize a Folium map
m = folium.Map(location=[40.0, -79.0], zoom_start=6)  # Centering the map roughly on Ohi

# Add the GeoJSON data to the map
folium.GeoJson(data=geojson_data).add_to(m)

# Save or display the map)
m
```

Out[34]:

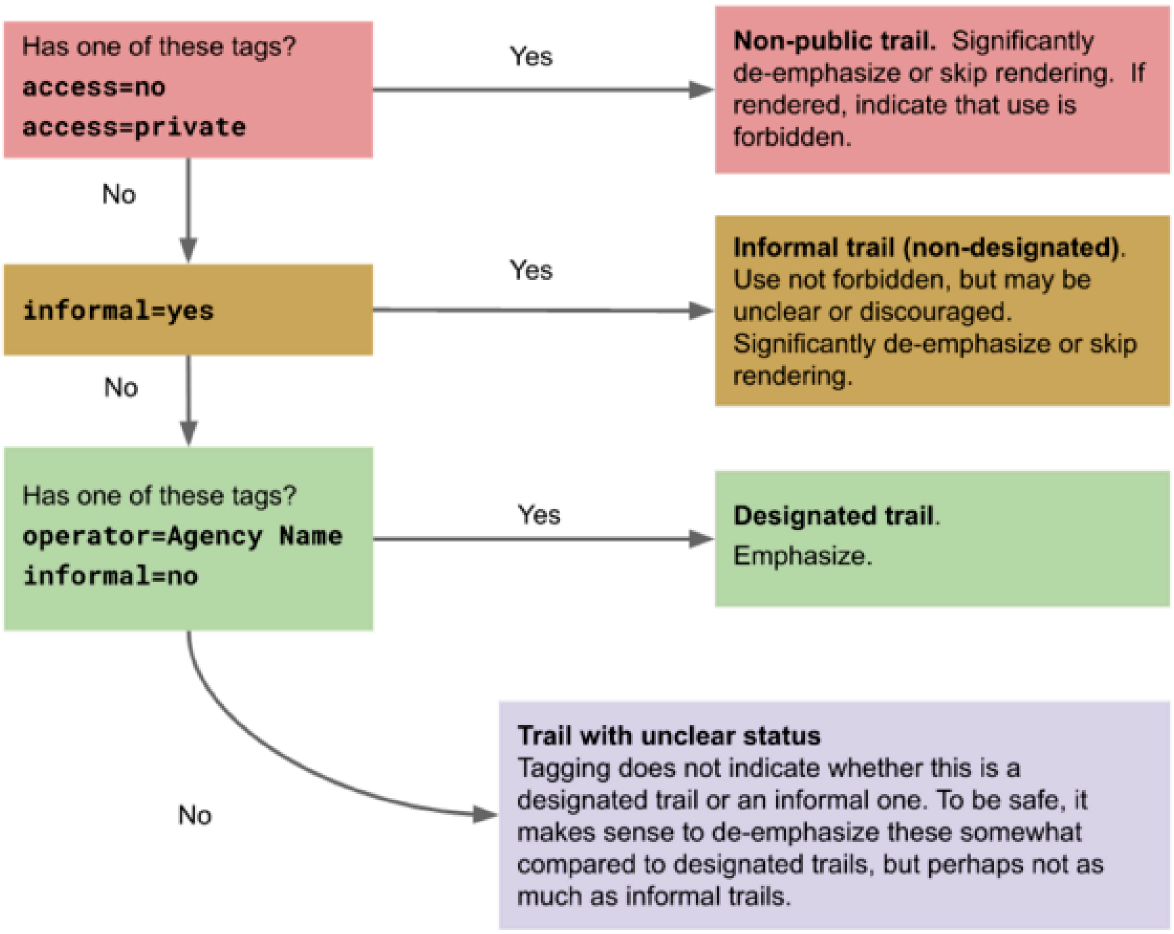


While the cleaned OSM data is a little light, it matches the pattern of the offical ODNR data pretty well.

Sorting trails and creating a map

OSM's ways, the lines between two geographicla points, are tagged with information. The tag the indeicates something is a trail, is Highway = path, export.geojson contains every way listed as Highway = Path in Ohio. If the trail has other tags, it can help indicate what kind of trail. If it has a name and is not listed as informal, it is likely and offical path, if it is informal, then it is likely as social trail or in otherwords, and trail used by people but not intended by the park. If Access is tagged as no then the trail is closed or private, if the path only has the tag Highway = Path, then it is likely not part of a maintained park system. I then organized and color coded this data set of trails based on the Trails Stewardship Initiative recomended nomenclature using folium, as the previous render allowed the user to click on the way, but not customize color.

Potential trail rendering style decision tree



Has one of these tags?
`access=no`
`access=private`

Non-public trail. Significantly de-emphasize or skip rendering. If rendered, indicate that use is forbidden.

`informal=yes`

Informal trail (non-designated). Use not forbidden, but may be unclear or discouraged. Significantly de-emphasize or skip rendering.

Has one of these tags?
`operator=Agency Name`
`informal=no`

Designated trail.
Emphasize.

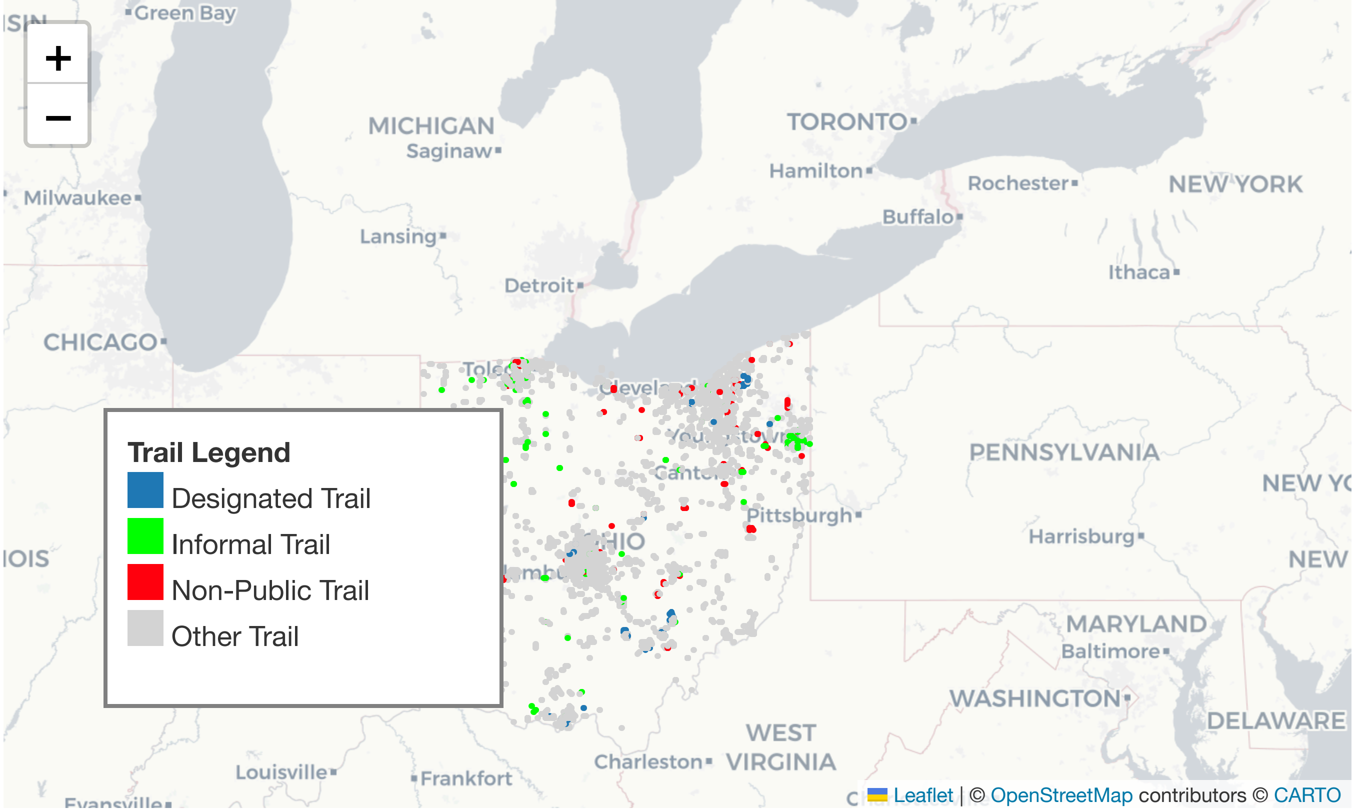
Trail with unclear status
Tagging does not indicate whether this is a designated trail or an informal one. To be safe, it makes sense to de-emphasize these somewhat compared to designated trails, but perhaps not as much as informal trails.

The choice to include informal, non-designated, or non-public trails will depend on a map's purpose and intended audience. Consider best practices for outdoor ethics of mapping when making this decision. If informal, non-designated, or non-public trails are included, designated trails should be emphasized and other types de-emphasized through variations in line weight, scale visibility, symbology, and/or labeling.

In [35]:

```python
# Path to your GeoJSON file
geojson_file = 'export.geojson'

# Load the GeoJSON data
with open(geojson_file, 'r') as file:
    geojson_data = json.load(file)

# Define a function to style the GeoJSON features dynamically
def style_function(feature):
    properties = feature.get('properties', {})
    # Designated Trails
    # Check for an operator that is not null
    if properties.get('operator') is not None:
        return {'color': '#1f78b4'}  # Blue

    # Check for an official name that is not null
    elif properties.get('official name') is not None:
        return {'color': '#1f78b4'}  # Blue
    # Check for access that equals 'permissive' or 'yes'
    elif properties.get('access') in ['permissive', 'yes']:
        return {'color': '#1f78b4'}  # Blue

    # Check for a ref that is not null
    elif properties.get('ref') is not None:
        return {'color': '#1f78b4'}  # Blue

    elif properties.get('informal') in ['no']:
        return {'color': '#1f78b4'}  # Blue

    # Informal Trails
    elif properties.get('informal') in ['yes']:
        return {'color': '#00ff00'}  # Green

    # Non-Public Trails
    elif properties.get('access') in ['private', 'no']:
        return {'color': '#ff000d'}  # Red

    # Unclear
    # Default style for other trails
    return {'color': '#D3D3D3'}  # Blue
# Initialize a map centered on Ohio
m = folium.Map(location=[40.0, -79.0], zoom_start=6, tiles="cartodb positron")

# Add the GeoJSON data to the map with the style function
folium.GeoJson(
    geojson_data,
    style_function=style_function
).add_to(m)

# Add a legend (HTML and CSS)
legend_html = """
<div style="
    position: fixed;
    bottom: 50px;
    left: 50px;
    width: 200px;
    height: 150px;
    background-color: white;
    border:2px solid grey;
    z-index:9999;
    font-size:14px;
    padding: 10px;
">
    <b>Trail Legend</b><br>
    <i style="background: #1f78b4; width: 18px; height: 18px; display: inline-block;"></
    <i style="background: #00ff00; width: 18px; height: 18px; display: inline-block;"></
    <i style="background: #ff000d; width: 18px; height: 18px; display: inline-block;"></
    <i style="background: #D3D3D3; width: 18px; height: 18px; display: inline-block;"></
</div>
"""
#legend

m.get_root().html.add_child(folium.Element(legend_html))

# Save to an HTML file or display in a Jupyter Notebook
m
```

Out[35]:



Conclusion

With this data, we can gain a better understanding of the current state of tagging for Ohio trails on Open Street Maps, and where it might need to improve. Part of the issue is that OSM is a forever ongoing project that might lack the specific and marked data that a complete source like ODNR could provide. Manual re-tagging might need to be done to correct missed tagged and umarked trails. We also have a solid structure for trail apps to use when deciphering what data from OSM to recomend to users as the trails they can trust to Hike.