projet

May 6, 2024

[68]:
```
import geopandas as gpd
import pandas as pd

import folium
from shapely.geometry import Point
```

[4]:
```
sites=gpd.read_file('data/country_sites.csv')
countries_info = gpd.read_file('data/country_sites.csv')
```

[6]:
```
sites['coordinates'] = sites['coordinates'].str.strip('[]')
```

[7]:
```
# Define a function to create Point geometries from coordinates
def create_point(coord):
    try:
        # Split the coordinate pair by comma and convert to float
        coords = coord.split(',')
        longitude = float(coords[0])
        latitude = float(coords[1])
        return Point(longitude, latitude)
    except (ValueError, IndexError):
        # Handle invalid coordinates
        return None

# Apply the function to create 'geometry' column
sites['geometry'] = sites['coordinates'].apply(create_point)
```

[8]:
```
egypt_data = sites[sites['country'] == 'Egypt'].copy()
nile=gpd.read_file("Nile River.shp")
egypt=gpd.read_file('Egypt.shp')
egypt_data.crs=nile.crs
```

[9]:
```
buffer_distance = 0.1
buffer_1000m=egypt_data.copy()
buffer_1000m.drop(columns=['geometry'])
buffer_1000m['geometry'] = egypt_data['geometry'].buffer(buffer_distance)
```

```
C:\Users\sayeh omar\AppData\Local\Temp\ipykernel_8900\3936645512.py:4:
UserWarning: Geometry is in a geographic CRS. Results from 'buffer' are likely
```

```
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS
before this operation.

  buffer_1000m['geometry'] = egypt_data['geometry'].buffer(buffer_distance)
```

[10]:
```
buffer_distance = 0.2
buffer_2000m=egypt_data.copy()
buffer_2000m.drop(columns=['geometry'])
buffer_2000m['geometry'] = egypt_data['geometry'].buffer(buffer_distance)
```

```
C:\Users\sayeh omar\AppData\Local\Temp\ipykernel_8900\2544810035.py:4:
UserWarning: Geometry is in a geographic CRS. Results from 'buffer' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS
before this operation.

  buffer_2000m['geometry'] = egypt_data['geometry'].buffer(buffer_distance)
```

[11]:
```
buffer_distance = 0.01
buffer_100m=egypt_data.copy()
buffer_100m.drop(columns=['geometry'])
buffer_100m['geometry'] = egypt_data['geometry'].buffer(buffer_distance)
```

```
C:\Users\sayeh omar\AppData\Local\Temp\ipykernel_8900\2744043746.py:4:
UserWarning: Geometry is in a geographic CRS. Results from 'buffer' are likely
incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS
before this operation.

  buffer_100m['geometry'] = egypt_data['geometry'].buffer(buffer_distance)
```

[55]:
```
# Create Folium Map with base map as base layer
map = folium.Map(location=[30.036749, 31.231509], zoom_start=8)

# Add base map layer to Folium map
folium.GeoJson(egypt).add_to(map)

# Add river shapefile layer to Folium map
folium.GeoJson(nile, name='River',color='red').add_to(map)

# Add GeoPandas DataFrame layer to Folium map
for idx, row in egypt_data.iterrows():
    folium.Marker([row['geometry'].y, row['geometry'].x],␣
 ↪popup=row['place_name']).add_to(map)

for idx, row in buffer_100m.iterrows():
    folium.GeoJson(row['geometry'],color=('red')).add_to(map)

for idx, row in buffer_2000m.iterrows():
    folium.GeoJson(row['geometry'],color=('green')).add_to(map)
```

```
for idx, row in buffer_1000m.iterrows():
    folium.GeoJson(row['geometry'],color=('yellow')).add_to(map)


# Display the Map
map
```

[55]: <folium.folium.Map at 0x1e8c4f47920>

[56]:
```
#map.save('Buffer-Zones.html')
```

[14]:
```
#output_folder = 'waste places egypt'
#output_filename = 'waste_places_egypt.shp'

# Export the GeoDataFrame as a shapefile
#egypt_data.to_file(f'{output_folder}/{output_filename}', driver='ESRI␣
↪Shapefile')
```

[39]:
```
# Perform spatial intersection between the buffer zones and the Nile River
intersections1000m = gpd.overlay( nile,buffer_1000m, how='intersection'␣
↪,keep_geom_type=False)

# Perform spatial intersection between the buffer zones and the Nile River
intersections2000m = gpd.overlay(buffer_2000m, nile,␣
↪how='intersection',keep_geom_type=False)

# Perform spatial intersection between the buffer zones and the Nile River
intersections100m = gpd.overlay(buffer_100m, nile,␣
↪how='intersection',keep_geom_type=False)
```

[57]:
```
import folium
from folium import GeoJson

# Create Folium Map with base map as base layer
m = folium.Map(location=[30.036749, 31.231509], zoom_start=8)

# Create a FeatureGroup for each layer
base_group = folium.FeatureGroup(name='Base Layer')
river_group = folium.FeatureGroup(name='River Layer')
waste_sites_group = folium.FeatureGroup(name='Waste Sites')
buffer_100m_group = folium.FeatureGroup(name='100m Buffer')
buffer_1000m_group = folium.FeatureGroup(name='1000m Buffer')
buffer_2000m_group = folium.FeatureGroup(name='2000m Buffer')

# Add base map layer to base_group
GeoJson(egypt).add_to(base_group)
```

```python
# Add river shapefile layer to river_group
GeoJson(nile, name='River', color='navy').add_to(river_group)

# Add waste site markers to waste_sites_group
for idx, row in egypt_data.iterrows():
    folium.Marker([row['geometry'].y, row['geometry'].x],␣
↪popup=row['place_name']).add_to(waste_sites_group)

# Add buffer zones to their respective groups
for idx, row in intersections100m.iterrows():
    GeoJson(row['geometry'], color='red').add_to(buffer_100m_group)

for idx, row in intersections1000m.iterrows():
    GeoJson(row['geometry'], color='yellow').add_to(buffer_1000m_group)

for idx, row in intersections2000m.iterrows():
    GeoJson(row['geometry'], color='green').add_to(buffer_2000m_group)

# Add all FeatureGroups to the map
base_group.add_to(m)
river_group.add_to(m)
waste_sites_group.add_to(m)


buffer_2000m_group.add_to(m)
buffer_1000m_group.add_to(m)
buffer_100m_group.add_to(m)

# Add LayerControl to the map
folium.LayerControl().add_to(m)

# Display the Map
m
```

[57]:
```
<folium.folium.Map at 0x1e8c4f47f50>
```

[58]:
```python
# Save the map as an HTML file
m.save("Danger_zones.html")
```

[61]:
```python
intersections100m['Red']='Y'
intersections100m['Yellow']='N'
intersections100m['Green']='N'
```

[62]:
```python
intersections1000m['Red']='N'
intersections1000m['Yellow']='Y'
intersections1000m['Green']='N'
```

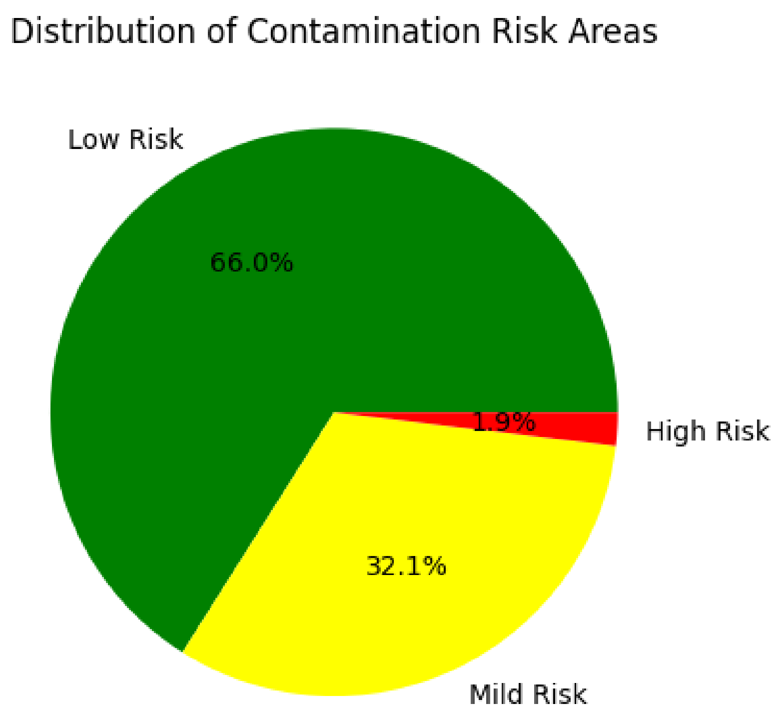
[63]:
```python
intersections2000m['Green']='Y'
intersections2000m['Yellow']='N'
intersections2000m['Red']='N'
```

[69]:
```python
all_intersections = pd.concat([intersections100m, intersections1000m,␣
↪intersections2000m], ignore_index=True)
```

[77]:
```python
import matplotlib.pyplot as plt
# Count the number of 'Y' and 'N' values in each color column
color_counts = pd.DataFrame({
    'Green': pd.Series(all_intersections['Green']).value_counts(),
    'Yellow': pd.Series(all_intersections['Yellow']).value_counts(),
    'Red': pd.Series(all_intersections['Red']).value_counts()
})

# Plotting the pie chart
colors = ['green', 'yellow', 'red']
labels = ['Low Risk', 'Mild Risk', 'High Risk']
plt.pie(color_counts.loc['Y'], labels=labels, colors=colors, autopct='%1.1f%%')
plt.title('Distribution of Contamination Risk Areas')

# Show the pie chart
plt.show()
```



[102]:
```python
import pandas as pd

# Assuming you have already created all_intersections GeoDataFrame

# Filter rows where risk level is 'Y' for each color
green_sites = all_intersections[all_intersections['Green'] ==␣
↪'Y']['place_name'].unique()
yellow_sites = all_intersections[all_intersections['Yellow'] ==␣
↪'Y']['place_name'].unique()
red_sites = all_intersections[all_intersections['Red'] == 'Y']['place_name'].
↪unique()

# Create a DataFrame with unique site names and risk levels
data = {
    'Site Name': all_intersections['place_name'].unique(),
    'Green': ['Y' if name in green_sites else 'N' for name in␣
↪all_intersections['place_name'].unique()],
    'Yellow': ['Y' if name in yellow_sites else 'N' for name in␣
↪all_intersections['place_name'].unique()],
    'Red': ['Y' if name in red_sites else 'N' for name in␣
↪all_intersections['place_name'].unique()]
}
table_df = pd.DataFrame(data)

table_df.to_csv('risk_levels.csv', index=False, encoding='utf-8-sig')
```