In [1]:
```
from IPython.display import Image
```

In [2]:
```
Image(filename='geo.png')
```

Out[2]:

by [Abraão Nascimento (https://bit.ly/3rtpN4A)](https://bit.ly/3rtpN4A)

Why Geopandas?

[Geopandas (https://geopandas.org/)](https://geopandas.org/) is one of the most powerful ways to work with geospatial data currently. Also, it is an open-source project to make working with geospatial data in python easier. It combines the capabilities of two powerful python libraries: pandas and shapely, providing geospatial operations in pandas and a high-level interface to multiple geometries to shapely.

Agenda:

1. Set up environment
2. Read and visualize geospatial data
3. Plot, manipulate and execute geospatial operations
4. Save your geospatial data
5. Mention

1. Set up environment

Setting up your environment with github, docker and binder:
[https://mybinder.org/v2/gh/abraaonascimento/1_Hour_Geopandas_Training/HEAD (https://mybinder.org/v2/gh/abraaonascimento/1_Hour_Geopandas_Training/HEAD)](https://mybinder.org/v2/gh/abraaonascimento/1_Hour_Geopandas_Training/HEAD)

In [ ]:
```
pip install rtree
```

In [ ]:
```
pip install pygeos
```

In [ ]:
```
pip install geopandas
```

In [ ]:
```
pip install matplotlib
```

In [3]:
```
import geopandas
```

2. Read and visualize geospatial data

Geospatial data is available from specific GIS file formats like ESRI shapefiles, Geodatabase, GeoJSON files, geopackage files, PostgreSQL/PostGIS, etc.

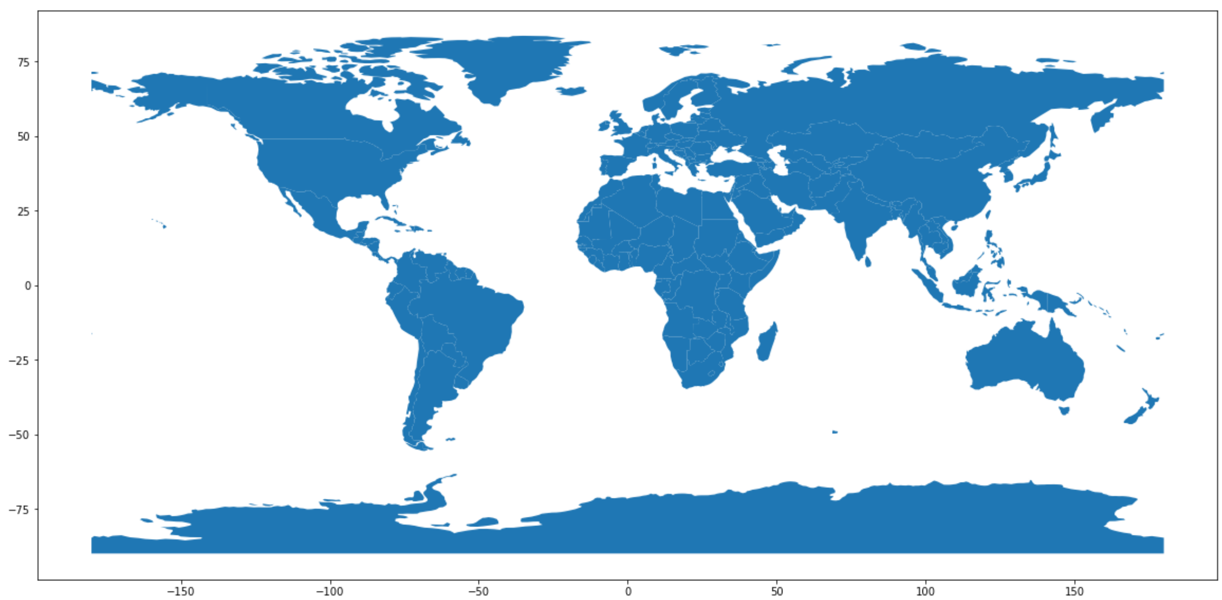
It is possible to use the GeoPandas library to read most all those GIS file formats since Geopandas uses GDAL/OGR in the background.

Let's start by reading a shapefile with all the countries of the world (adapted from naturalearthdata (http://www.naturalearthdata.com/downloads/110m-cultural-vectors/110m-admin-0-countries/), zip file is available in the /data directory), and inspect the data:

In [4]:
```
countries = geopandas.read_file("zip://./data/countries.zip")
```

In [5]:
```
countries.plot(figsize=(20,20))
#countries.plot(figsize=(20,20))
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x197be5cf430>



In [6]:
```
countries.head(20)
```

Out[6]:

	iso_a3	name	continent	pop_est	gdp_md_est	geometry
0	AFG	Afghanistan	Asia	34124811.0	64080.0	POLYGON ((61.211 35.650, 62.231 35.271, 62.985...
1	AGO	Angola	Africa	29310273.0	189000.0	MULTIPOLYGON (((23.904 -11.722, 24.080 -12.191...
2	ALB	Albania	Europe	3047987.0	33900.0	POLYGON ((21.020 40.843, 21.000 40.580, 20.675...
3	ARE	United Arab Emirates	Asia	6072475.0	667200.0	POLYGON ((51.580 24.245, 51.757 24.294, 51.794...
4	ARG	Argentina	South America	44293293.0	879400.0	MULTIPOLYGON (((-66.960 -54.897, -67.562 -54.8...
5	ARM	Armenia	Asia	3045191.0	26300.0	POLYGON ((43.583 41.092, 44.972 41.248, 45.179...
6	ATA	Antarctica	Antarctica	4050.0	810.0	MULTIPOLYGON (((-59.572 -80.040, -59.866 -80.5...
7	ATF	Fr. S. Antarctic Lands	Seven seas (open ocean)	140.0	16.0	POLYGON ((68.935 -48.625, 69.580 -48.940, 70.5...
8	AUS	Australia	Oceania	23232413.0	1189000.0	MULTIPOLYGON (((145.398 -40.793, 146.364 -41.1...
9	AUT	Austria	Europe	8754413.0	416600.0	POLYGON ((16.980 48.123, 16.904 47.715, 16.341...
10	AZE	Azerbaijan	Asia	9961396.0	167900.0	MULTIPOLYGON (((46.506 38.771, 46.483 39.464, ...
11	BDI	Burundi	Africa	11466756.0	7892.0	POLYGON ((29.340 -4.500, 29.276 -3.294, 29.025...
12	BEL	Belgium	Europe	11491346.0	508600.0	POLYGON ((4.047 51.267, 4.974 51.475, 5.607 51...
13	BEN	Benin	Africa	11038805.0	24310.0	POLYGON ((2.692 6.259, 1.865 6.142, 1.619 6.83...
14	BFA	Burkina Faso	Africa	20107509.0	32990.0	POLYGON ((2.154 11.940, 1.936 11.641, 1.447 11...
15	BGD	Bangladesh	Asia	157826578.0	628400.0	POLYGON ((92.673 22.041, 92.652 21.324, 92.303...
16	BGR	Bulgaria	Europe	7101510.0	143100.0	POLYGON ((22.657 44.235, 22.945 43.824, 23.332...
17	BHS	Bahamas	North America	329988.0	9066.0	MULTIPOLYGON (((-77.535 23.760, -77.780 23.710...
18	BIH	Bosnia and Herz.	Europe	3856181.0	42530.0	POLYGON ((19.368 44.863, 19.118 44.423, 19.600...
19	BLR	Belarus	Europe	9549747.0	165400.0	POLYGON ((23.484 53.912, 24.451 53.906, 25.536...

As was presented above:

- By using .plot() method we can quickly get a basic visualization of the data
- By using .head() method we can see the first rows of the dataset, just like in excel or QGIS.
- There is a 'geometry' column that keeps the geoinformation of the countries represented as polygons.

3. Plot, manipulate and execute geospatial operations

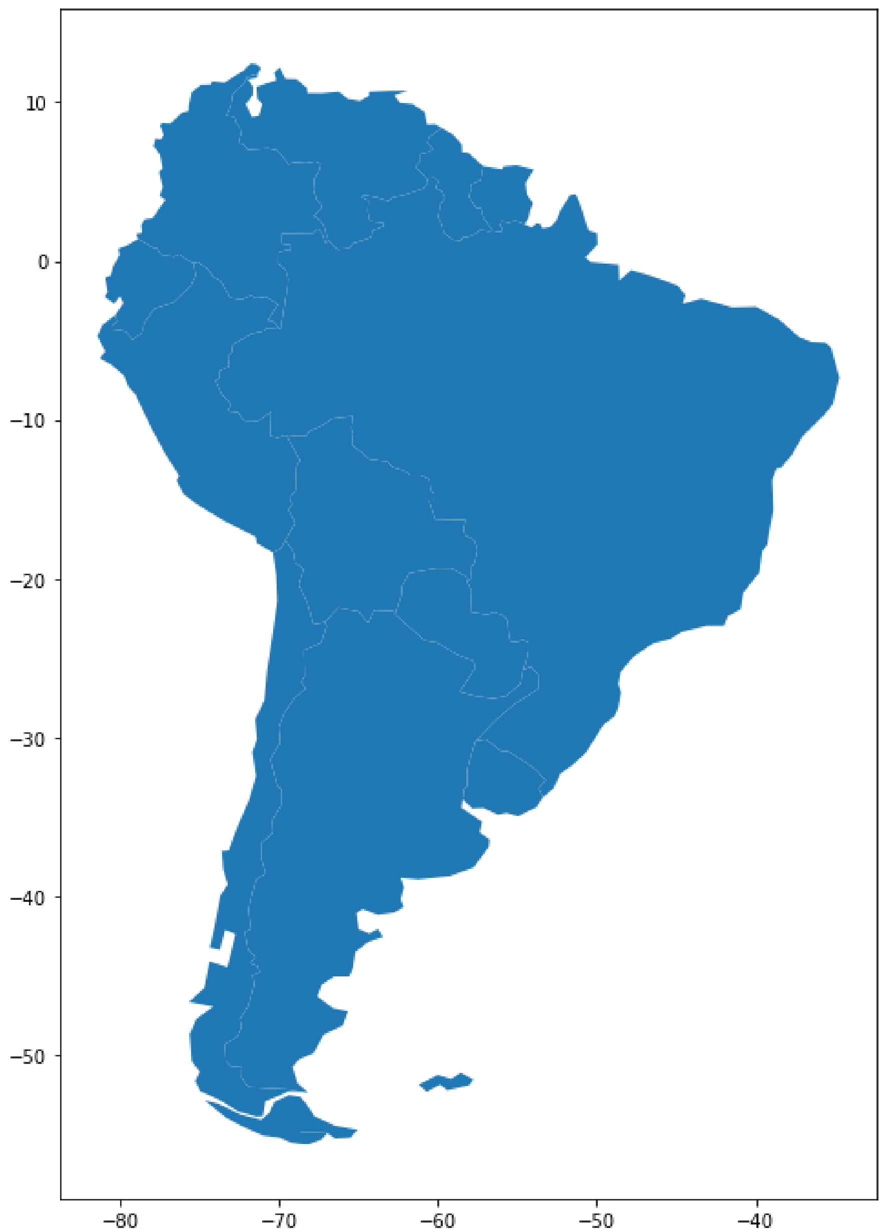
Such as ESRI shapefiles Geopandas dataframe format contains **attributes** (fields and records) and **geometry** (coordinates). Such as in QGIS we can use function and filter to manipulate the geodata

Filtering

In [7]:
```
lam = countries[countries['continent'] == 'South America']
```

In [8]:
```
lam.plot(figsize=(12,12))
```

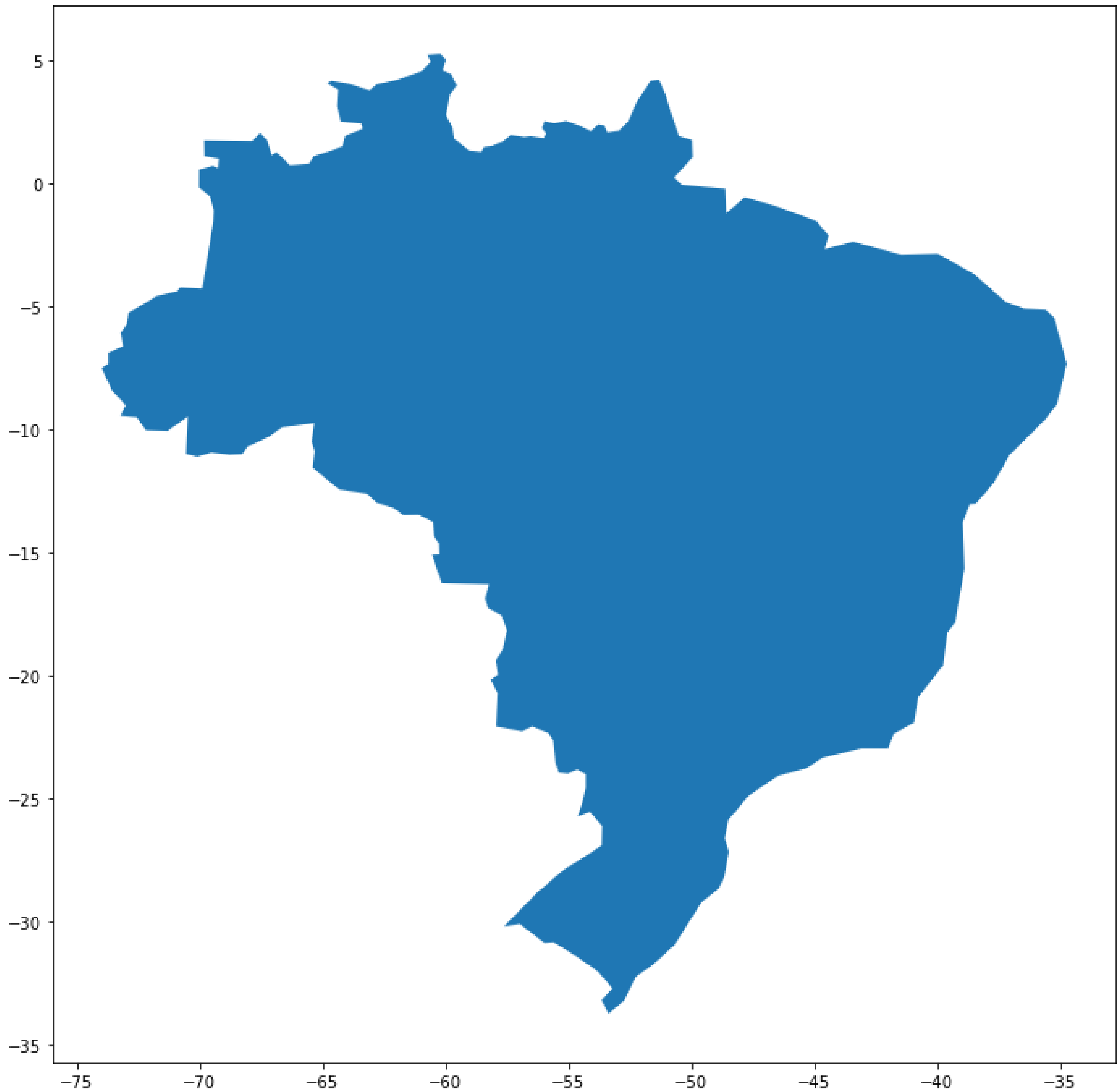
Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x197c06f1520>
```



In [9]:
```
bra = countries[countries['name'] == 'Brazil']
```

In [10]:
```
bra.plot(figsize=(12,12))
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x197c072c1f0>



Population

In [11]:
```
bra['pop_est']
```

Out[11]:
```
22    207353391.0
Name: pop_est, dtype: float64
```

In [12]:
```
"Population: " + "{:,}".format(int(bra['pop_est'].values[0]))
```

Out[12]: 'Population: 207,353,391'

In [13]:
```
#"{:,}".format(bra.area.values[0])
```

Coordinate reference systems

In [14]:
```
bra.crs
```

Out[14]:
```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

In [15]:
```
bra_utm = bra.to_crs(epsg=3395)
```

In [16]:
```
bra_utm.crs
```

Out[16]:
```
<Projected CRS: EPSG:3395>
Name: WGS 84 / World Mercator
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: World - between 80°S and 84°N
- bounds: (-180.0, -80.0, 180.0, 84.0)
Coordinate Operation:
- name: World Mercator
- method: Mercator (variant A)
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

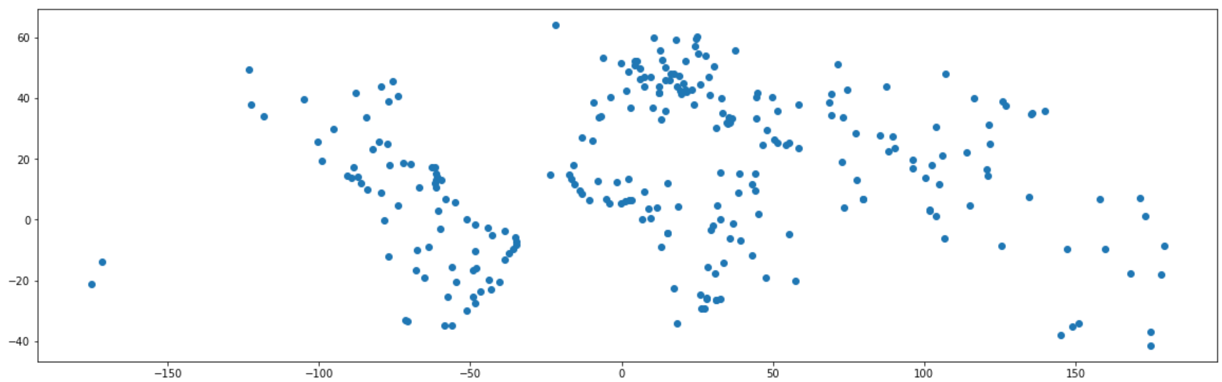
Geometries types: points, lines and polygons

Until now, we worked only with Polygons. Let's import points that represent cities and lines that represent road-network. Here we will use data from DNIT (http://servicos.dnit.gov.br/vgeo/) (National Department of Transport Infrastructure) and Natural Earth Data (http://www.naturalearthdata.com/downloads/110m-cultural-vectors/110m-populated-places/)

In [17]:
```
cities = geopandas.read_file("zip://./data/cities.zip")
roads  = geopandas.read_file("zip://./data/roads.zip")
```

In [18]:
```
cities.plot(figsize=(20,20))
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x197c07a03a0>



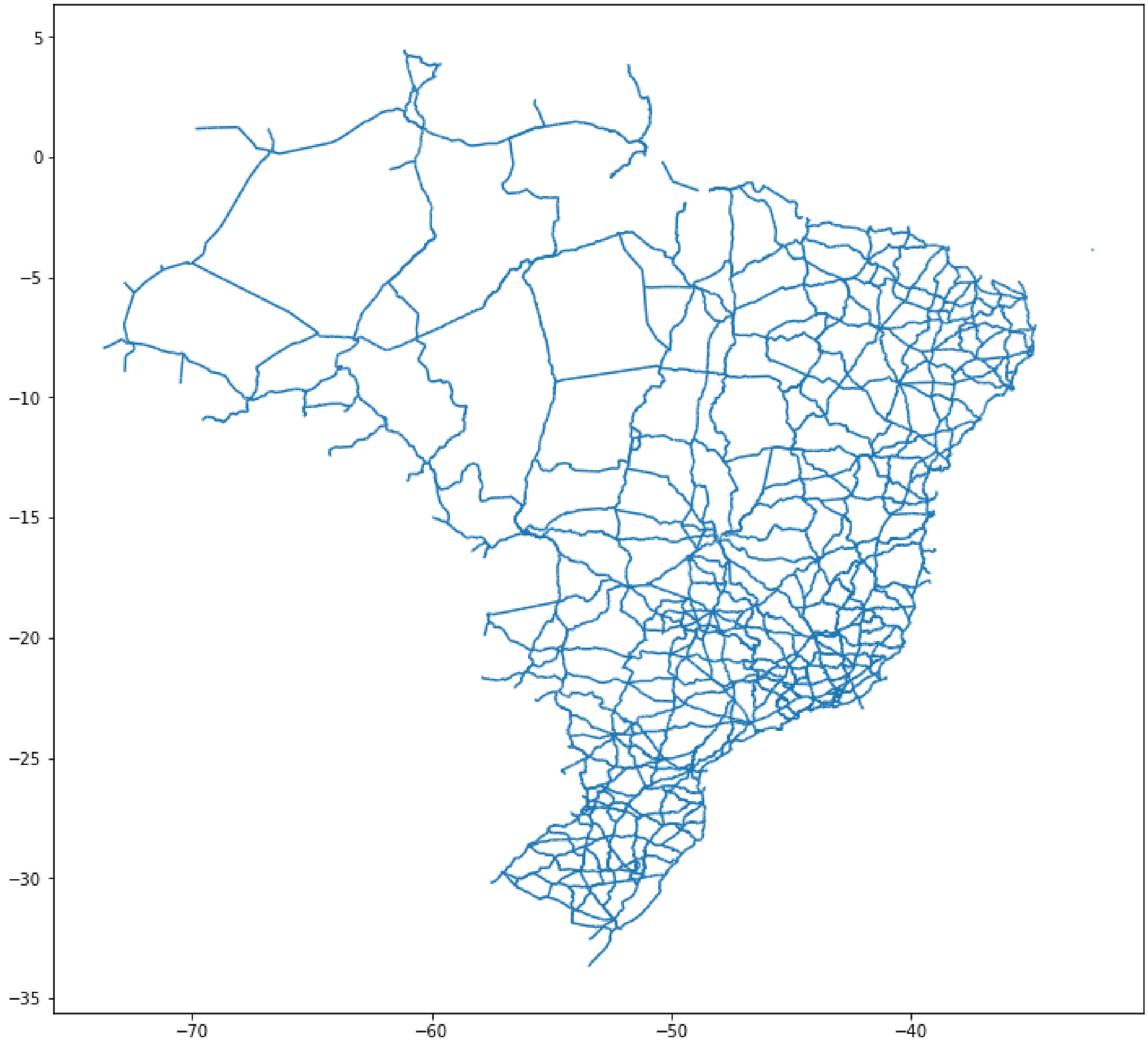
In [25]:
```
cities.head(10)
```

Out[25]:

	name	geometry
0	Aracaju/SE	POINT (-37.07232 -10.91151)
1	Belém/PA	POINT (-48.50440 -1.45644)
2	Belo Horizonte/MG	POINT (-43.95639 -19.81754)
3	Boa Vista/RR	POINT (-60.67347 2.81958)
4	Brasília/DF	POINT (-47.93041 -15.78052)
5	Campo Grande/MS	POINT (-54.64647 -20.44352)
6	Cuiabá/MT	POINT (-56.09746 -15.59650)
7	Curitiba/PR	POINT (-49.27345 -25.42855)
8	Florianópolis/SC	POINT (-48.54945 -27.59756)
9	Fortaleza/CE	POINT (-38.54333 -3.71746)

In [20]:
```
roads.plot(figsize=(12,12))
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x197c21a0760>

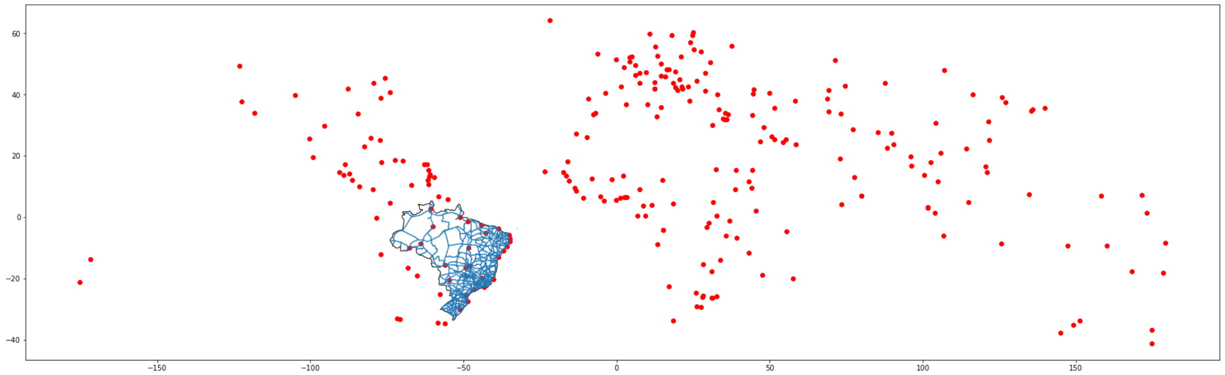


In [29]:
```
#roads.head(3)
```

Visualize different layers together

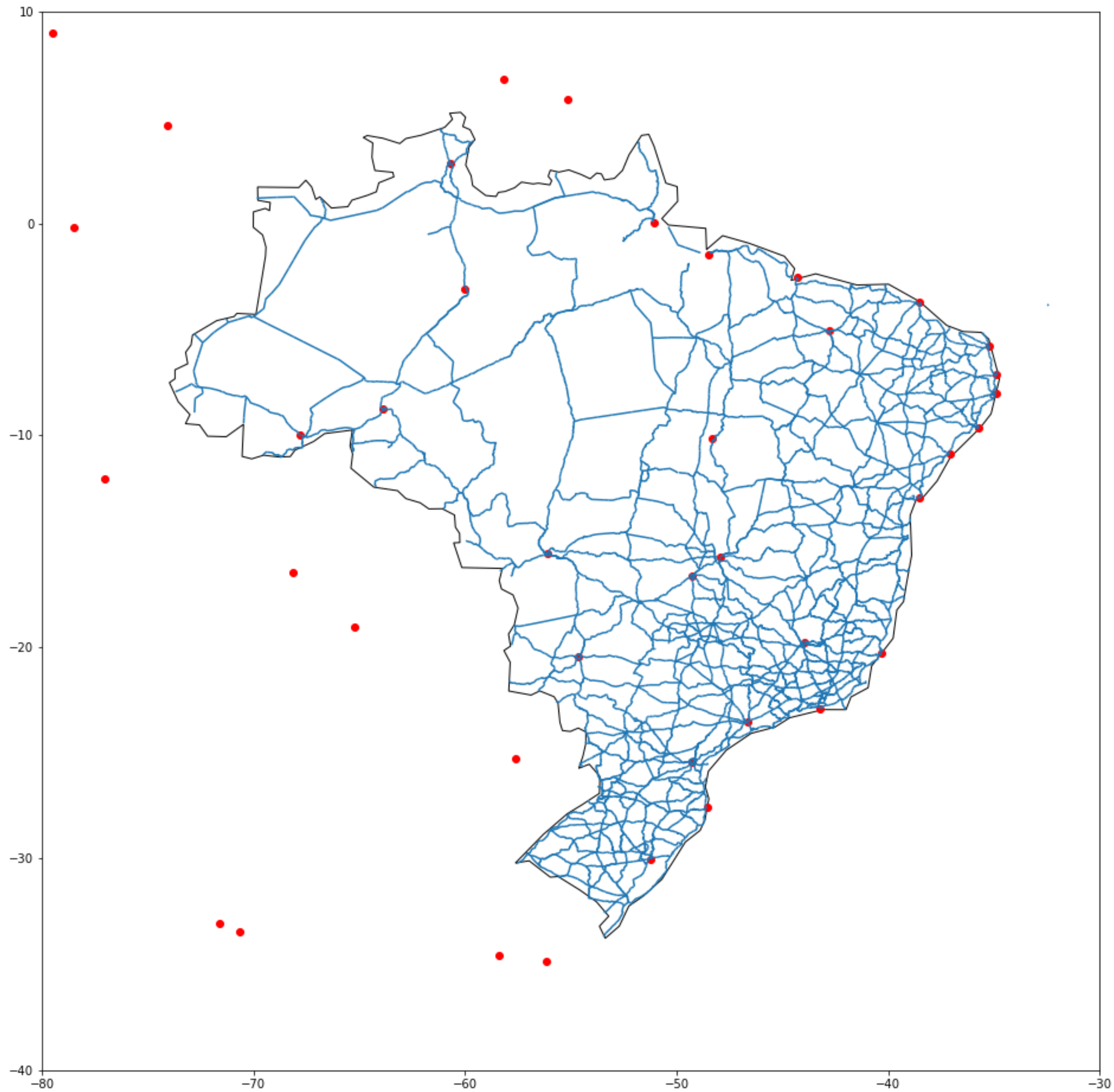
In [22]:
```
ax = bra.plot(edgecolor='k', facecolor='none', figsize=(30, 30))
roads.plot(ax=ax)
cities.plot(ax=ax, color='red')
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x197c35a38b0>



In [23]:
```
ax = bra.plot(edgecolor='k', facecolor='none', figsize=(16, 20))
roads.plot(ax=ax)
cities.plot(ax=ax, color='red')
ax.set(xlim=(-80, -30), ylim=(-40, 10))
```

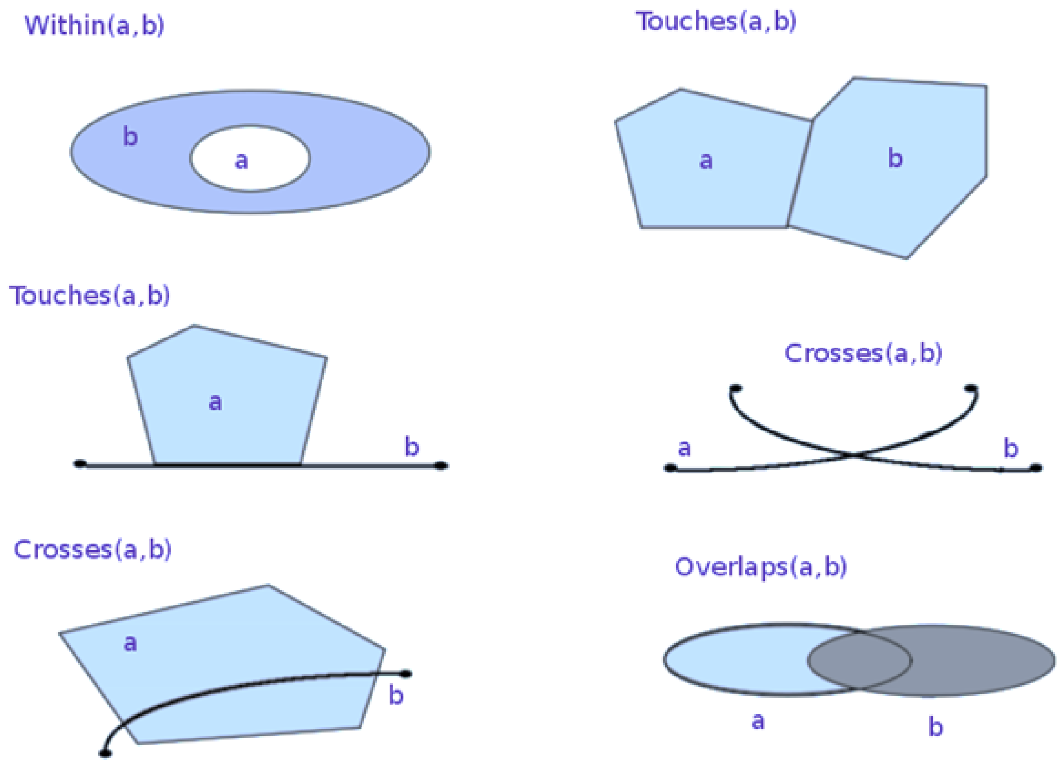
Out[23]:
```
[(-40.0, 10.0), (-80.0, -30.0)]
```



Geospatial operations

In [24]:
```
Image(filename='spatial_operations.png')
```

Out[24]:



Overview of functions to check spatial relationships:

- equals
- crosses
- overlaps
- touches
- `intersects`
- `within`

To check more manipulation and analysis of geometric objects please check the [Shapely User Manual (https://shapely.readthedocs.io/en/stable/manual.html#predicates-and-relationships)](https://shapely.readthedocs.io/en/stable/manual.html#predicates-and-relationships).

EXERCISE

- Which cities pass within Brazil?
- Which roads intersect the urban areas in Brazil?

We already have the geometry of Brazil selected in the variable 'bra'. Let's use it to find the cities within Brazil's polygon

In [30]:
```
bra.geometry.squeeze()
```

Out[30]:

In [31]:
```
bra_geometry = bra.geometry.squeeze()
```

In [32]:
```
bra_cities = cities[cities.within(bra_geometry)]
```

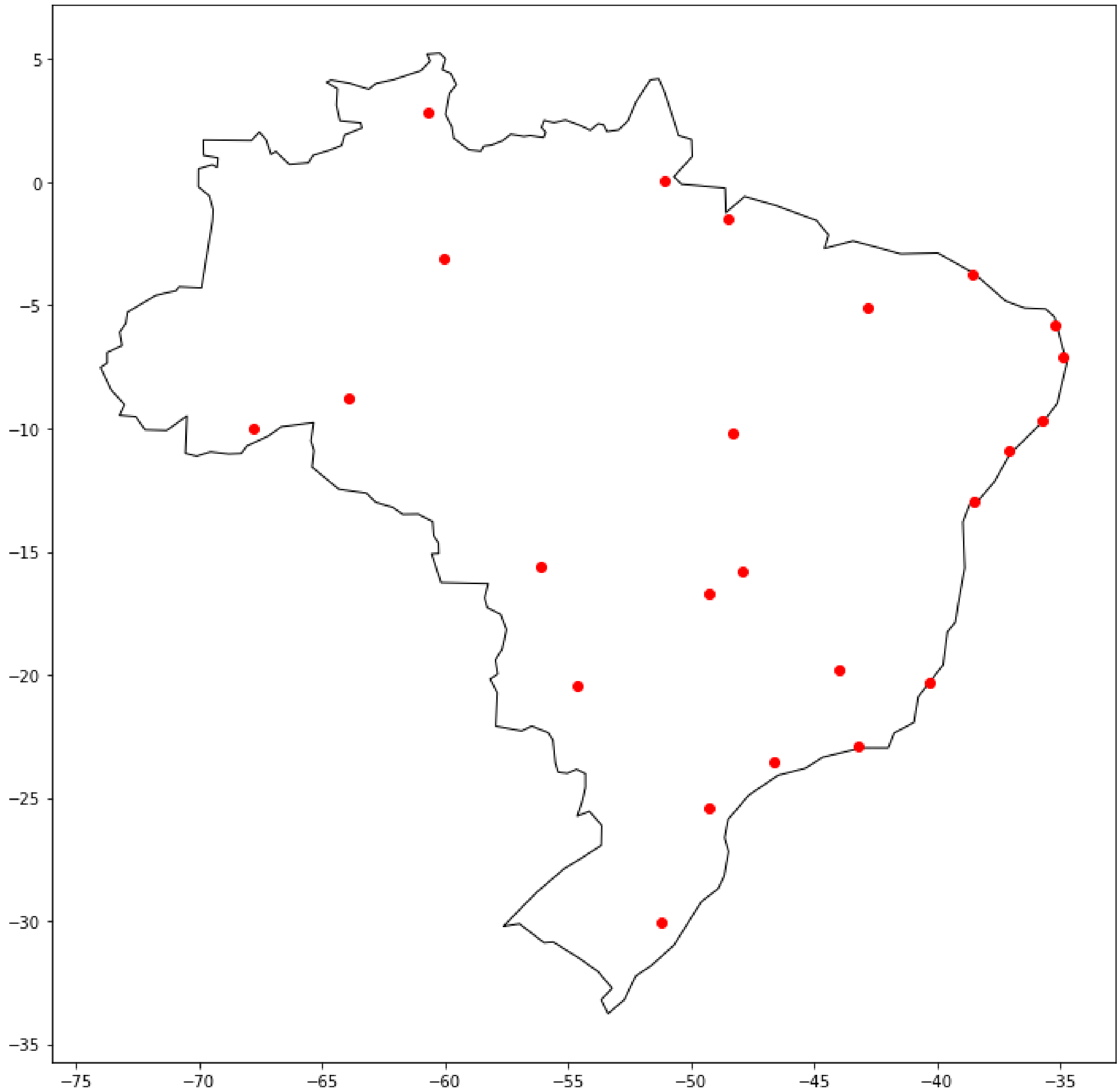
In [33]:
```
bra_cities.head(10)
```

Out[33]:

	name	geometry
0	Aracaju/SE	POINT (-37.07232 -10.91151)
1	Belém/PA	POINT (-48.50440 -1.45644)
2	Belo Horizonte/MG	POINT (-43.95639 -19.81754)
3	Boa Vista/RR	POINT (-60.67347 2.81958)
4	Brasília/DF	POINT (-47.93041 -15.78052)
5	Campo Grande/MS	POINT (-54.64647 -20.44352)
6	Cuiabá/MT	POINT (-56.09746 -15.59650)
7	Curitiba/PR	POINT (-49.27345 -25.42855)
9	Fortaleza/CE	POINT (-38.54333 -3.71746)
10	Goiânia/GO	POINT (-49.25443 -16.67952)

In [34]:
```
ax = bra.plot(edgecolor='k', facecolor='none', figsize=(12, 20))
bra_cities.plot(ax=ax, color='red')
```

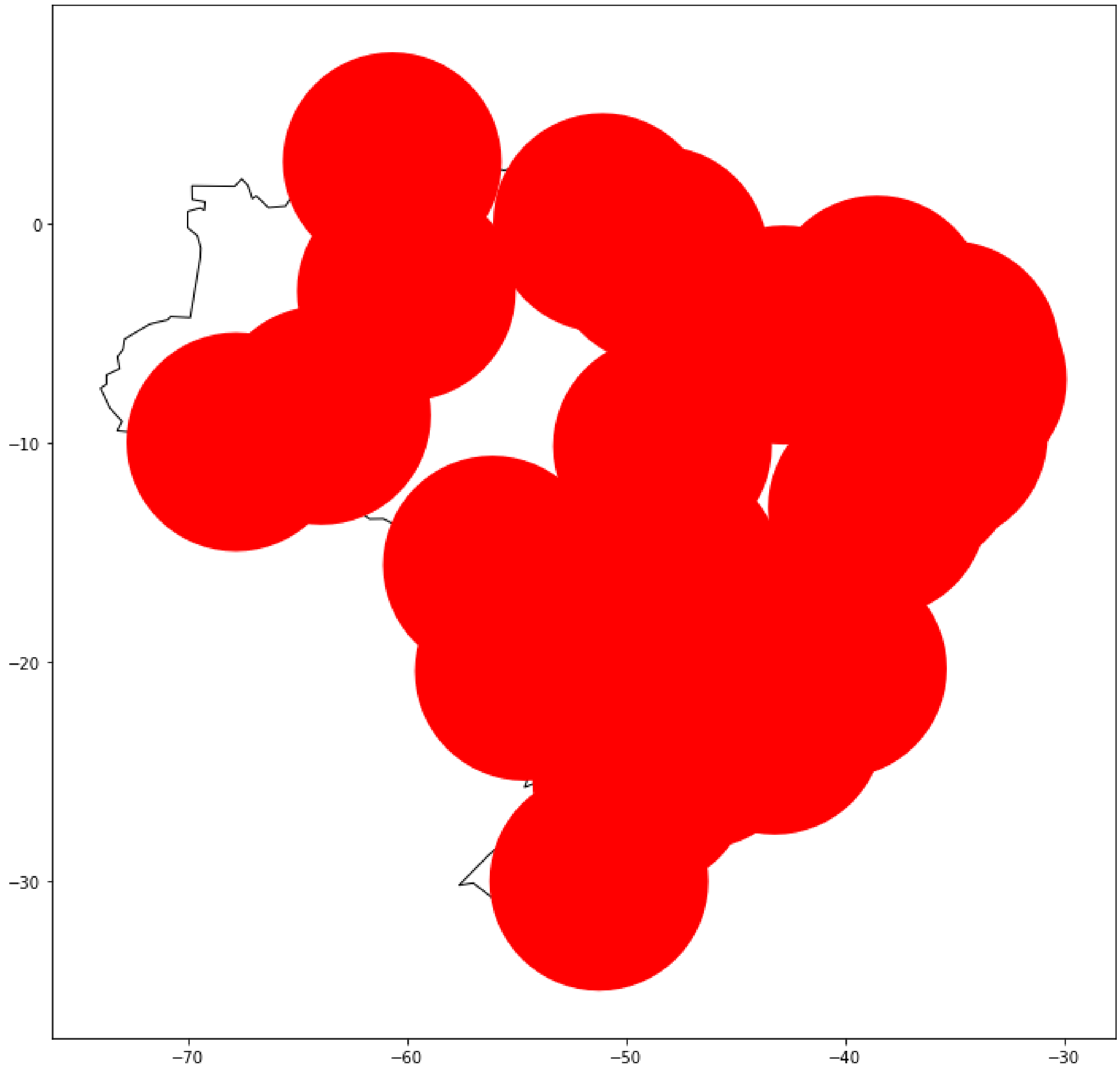
Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x197c5d02fa0>



To simulate urban areas we can create buffers using the POINT objects that represents the filtered cities within Brazil

In [35]:
```
ax = bra.plot(edgecolor='k', facecolor='none', figsize=(12, 20))
bra_cities.buffer(5).plot(ax=ax,color='red')
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x197c60a50a0>



NOTE

- Within the buffer function, it was used the number 5 '.buffer(5)'. In this case, 5 means 5 degrees. This is the reason why we got large buffers.
- To convert degrees to meters, we can transform the geographic coordinate system to a '2D' cartesian plan. That is also the possibility to manually convert degrees to meters, but it is not practical
- Approximate metric equivalents for degrees: 0.00001° = 1.11 meters

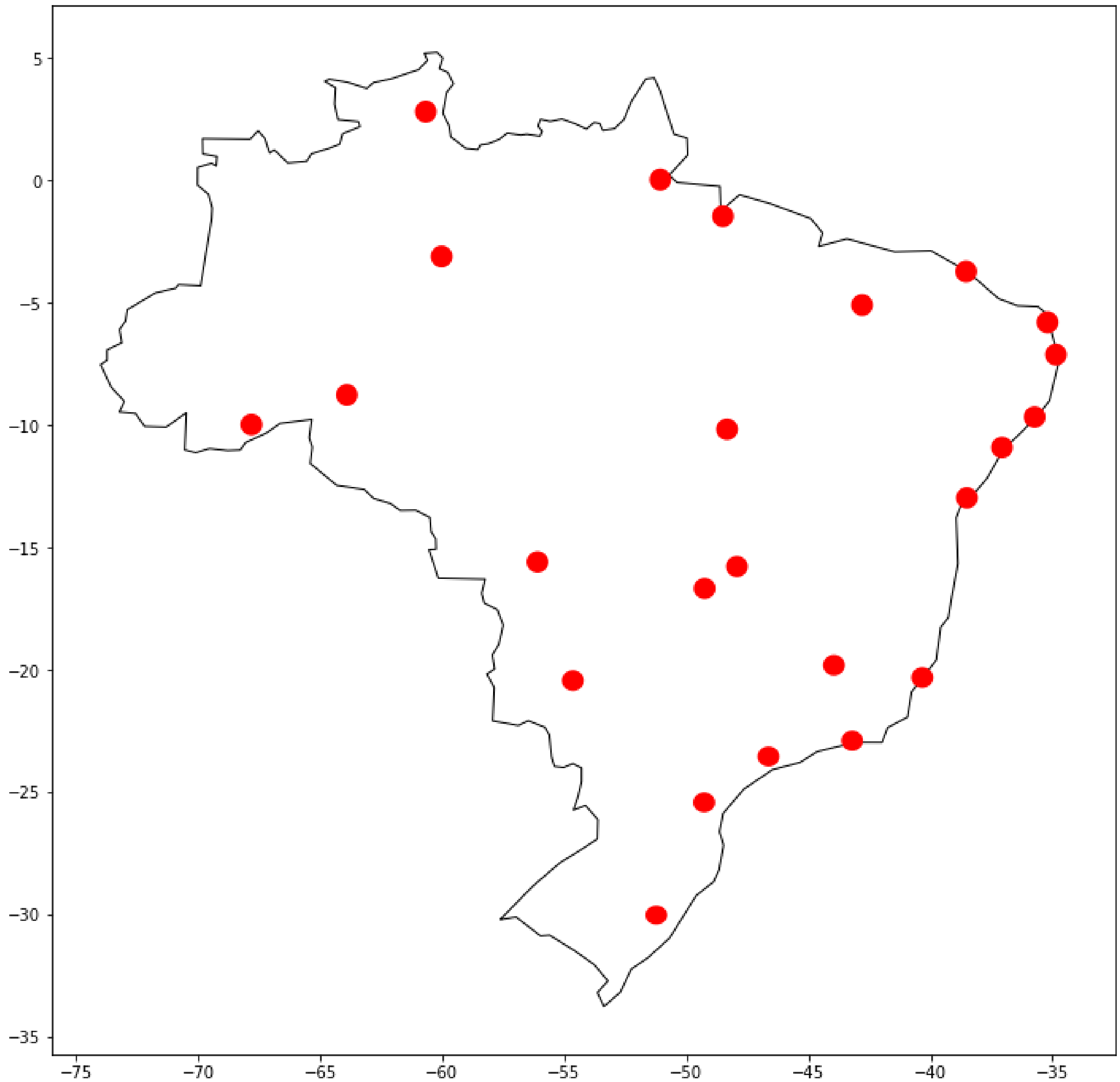
In [36]:
```
cities = bra_cities.to_crs(epsg=3395)  # 1. Converting geographic to a metric coordinat
e system
```

In [37]:
```
urban_areas = cities.buffer(50000)     # 2. Creating 50km buffer
```

In [38]:
```
urban_areas = urban_areas.to_crs(epsg=4326) # 3. Converting back: metric to a geographi
c coordinate system
```

In [39]:
```
ax = bra.plot(edgecolor='k', facecolor='none', figsize=(12, 20))
urban_areas.plot(ax=ax,color='red')
```

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x197c0797b80>



NOTE

- The buffer function gives back just the geometry. This operation does not keet the attributes
- To get the attributes back we can replace the cities 'points' (city centers) with cities 'polygons' (urban areas)

In [40]:
```
urban_areas
```

Out[40]:
```
0     POLYGON ((-36.62316 -10.91151, -36.62532 -10.9...
1     POLYGON ((-48.05524 -1.45644, -48.05740 -1.500...
2     POLYGON ((-43.50723 -19.81754, -43.50940 -19.8...
3     POLYGON ((-60.22431 2.81958, -60.22647 2.77531...
4     POLYGON ((-47.48125 -15.78052, -47.48341 -15.8...
5     POLYGON ((-54.19731 -20.44352, -54.19947 -20.4...
6     POLYGON ((-55.64830 -15.59650, -55.65047 -15.6...
7     POLYGON ((-48.82429 -25.42855, -48.82645 -25.4...
9     POLYGON ((-38.09417 -3.71746, -38.09633 -3.761...
10    POLYGON ((-48.80527 -16.67952, -48.80743 -16.7...
11    POLYGON ((-34.41414 -7.11549, -34.41631 -7.159...
12    POLYGON ((-50.61726 0.03857, -50.61942 -0.0057...
13    POLYGON ((-35.28615 -9.66650, -35.28831 -9.710...
14    POLYGON ((-59.57631 -3.10245, -59.57847 -3.146...
15    POLYGON ((-34.76015 -5.79548, -34.76231 -5.839...
16    POLYGON ((-47.88425 -10.16749, -47.88641 -10.2...
17    POLYGON ((-50.78132 -30.03355, -50.78349 -30.0...
18    POLYGON ((-63.45533 -8.76247, -63.45749 -8.806...
20    POLYGON ((-67.36135 -9.97546, -67.36351 -10.01...
21    POLYGON ((-42.75924 -22.90356, -42.76140 -22.9...
22    POLYGON ((-38.06218 -12.97152, -38.06434 -13.0...
24    POLYGON ((-46.18726 -23.54855, -46.18943 -23.5...
25    POLYGON ((-42.35320 -5.08947, -42.35537 -5.133...
26    POLYGON ((-39.88921 -20.31955, -39.89137 -20.3...
dtype: geometry
```

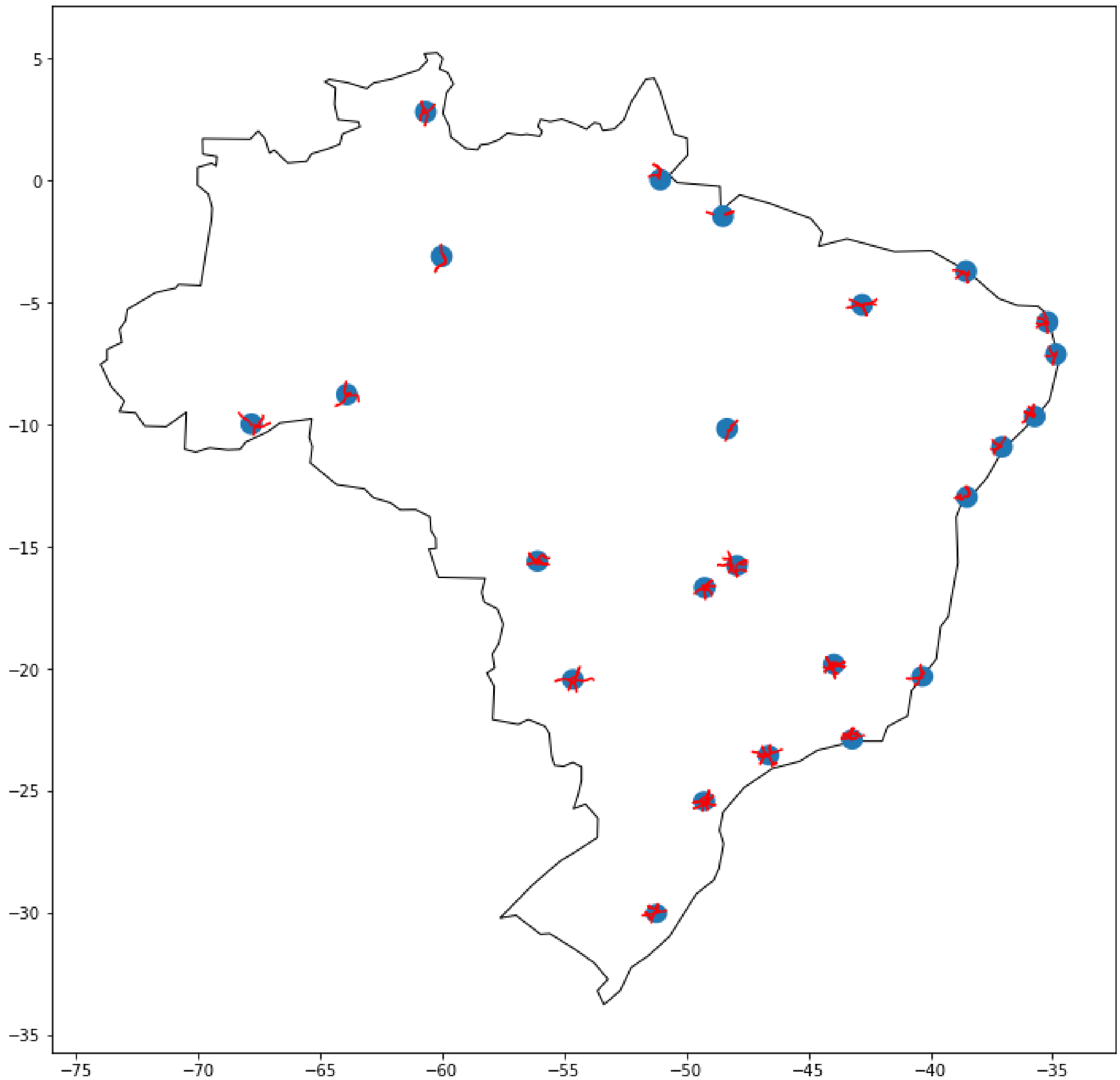
In [41]:
```
bra_cities['geometry'] = urban_areas  # 4. Getting back the attributes
```

With the 50km buffer created it is possible to get all the roads that intersects our simulated urban areas

In [42]:
```
urban_roads = geopandas.sjoin(roads, bra_cities, op='intersects')  # 5. Find which road
s intersect the urban capitals of Brazil by using spatial join.
```

In [43]:
```
ax = bra.plot(edgecolor='k', facecolor='none', figsize=(12, 20))
#roads.plot(ax=ax)
bra_cities.plot(ax=ax)
urban_roads.plot(ax=ax,color='red')
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x197c6c75040>



4. Save your geospatial data

To end let's save the data. In Geopandas we can save files in many formats: ESRI shapefiles, Geodatabase, GeoJSON files, geopackage files. You can check all the options in [Geopandas User Guide (https://geopandas.org/docs/user_guide/io.html)](https://geopandas.org/docs/user_guide/io.html). In this exercise, we will be saving the data in the ESRI Shapefile format.

In [ ]:
```
bra.to_file("country_bra.shp")
```

In [ ]:
```
bra_cities.to_file("cities_bra.shp")
```

In [ ]:
```
urban_roads.to_file("urban_roads_bra.shp")
```

5. Mention

This training was created based on the open resources:

- Geopandas - An Introduction (https://medium.com/thoughtful-data-science/geopandas-an-introduction-c544a352c662)
- Introduction to Geospatial Data Analysis with Python (https://github.com/geopandas/scipy2018-geospatial-data)
- Geopandas - User Guide (https://geopandas.org/index.html)
- Shapely - User Guide (https://shapely.readthedocs.io/en/stable/manual.html)

Thank you!